SDS 901172A
$29.50

TECHNICAL MANUAL

SIGMA 5 COMPUTER

September 1968

Xerox Data Systems 701 South Aviation Blvd., El Segundo, California 90245 (213) 772-4511, 679-4511

LIST OF EFFECTIVE PAGES

Total number of pages is 860, as follows:

Page No.	Issue	Page No.	Issue
Title	Original		
A	Original		
i thru xiv	Original		
1-1 thru 1-12	Original		
2-1 thru 2-20	Original		
3-1 thru 3-758	Original		
4-1 thru 4-54	Original		

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont.)

TABLE OF CONTENTS (Cont.)

TABLE OF CONTENTS (Cont.)

TABLE OF CONTENTS (Cont.)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (Cont.)

LIST OF ILLUSTRATIONS (Cont.)

LIST OF ILLUSTRATIONS (Cont.)

LIST OF ILLUSTRATIONS (Cont.)

LIST OF TABLES

LIST OF TABLES (Cont.)

LIST OF TABLES (Cont.)

LIST OF TABLES (Cont.)

RELATED PUBLICATIONS

The following publications contain information not included in this manual, but necessary for a complete understanding of the Sigma 5 computer.

Publication Title	Publication No.
Sigma 5/7 Memory (≥8K) Test (Medic 75) Diagnostic Program Manual	900825
Sigma Fortran IV Reference Manual for Sigma 5/7 Computers	900956
Fortran IV-H Reference Manual for SDS Sigma 5/7 Computers	900966
Sigma 5/7 Relocatable Diagnostic Program Loader Diagnostic Program Manual	900972
Sigma Computer Systems Interface Design Manual	900973
Sigma 5/7 Memory Interleaving Test (MIT) Diagnostic Program Manual	901071
Sigma 5/7 Systems Test Monitor Diagnostic Program Manual	901076
Sigma 5/7 Interrupt Test Diagnostic Program Manual	901134
Sigma 5/7 Power Fail-Safe Test Diagnostic Program Manual	901135
Sigma 5/7 Real-Time Clock Test Diagnostic Program Manual	901136
Fortran IV-H Library/Run Time Technical Manual	901138
Fortran IV-H Operations Manual for Sigma 5/7 Computers	901144
Sigma 5/7 Selector IOP Channel Test Diagnostic Program Manual	901158
Sigma 5 Integral IOP Channel Test Diagnostic Program Manual	901161
Sigma 5/7 Computer Numerical Subroutine Package Technical Manual	901505
Sigma 5/7 CPU Diagnostic Program (Memory Protect) Diagnostic Program Manual	901516
Sigma 5 CPU (Suffix) Diagnostic Program Manual	901519
Sigma 5 CPU Program Test (AUTO) Diagnostic Program Manual	901523
Sigma 5 Computer Reference Manual	900959

SECTION I
GENERAL DESCRIPTION

1-1 INTRODUCTION

This manual contains information necessary to operate and maintain the Sigma 5 computer, manufactured by Scientific Data Systems, Santa Monica, California. Following the general physical and functional description in this section, the material presented includes a section on operation and programming, basic and detailed principles of operation, maintenance instructions, performance testing, and a tabular list of replaceable parts.

Technical manuals describing equipment associated with the Sigma 5 computer and programming manuals are referred to in the List of Related Publications in the front matter of this manual.

The Sigma 5 computer is a high-speed, multipurpose, digital computer for use in business, scientific, process control, hybrid, and systems applications. The computer, which is organized around one or more high-speed central processing units with an integral input/output processor and fast magnetic core memory, functions efficiently in real-time, time-sharing, and multiusage computing environments. Figure 1-1 shows a typical Sigma 5 computer configuration.

1-2 PHYSICAL DESCRIPTION

1-3 BASIC COMPUTER

The basic Sigma 5 computer contains a central processing unit (CPU) and integral input/output processor contained in a single cabinet, an expandable memory contained in one to four cabinets, and at least one I/O device controller and a processor control panel, contained in an accessory cabinet. The CPU and memory are composed of printed circuit modules inserted into slots in chassis. Each chassis may contain up to 32 modules. Pins at the rear of the modules are plugged into sockets mounted on a rear wiring board that contains all wire connections. Module sizes are all identical except for the core diode modules in the memory units. These modules occupy the vertical space normally filled by two standard modules, and therefore require a double-sized chassis.

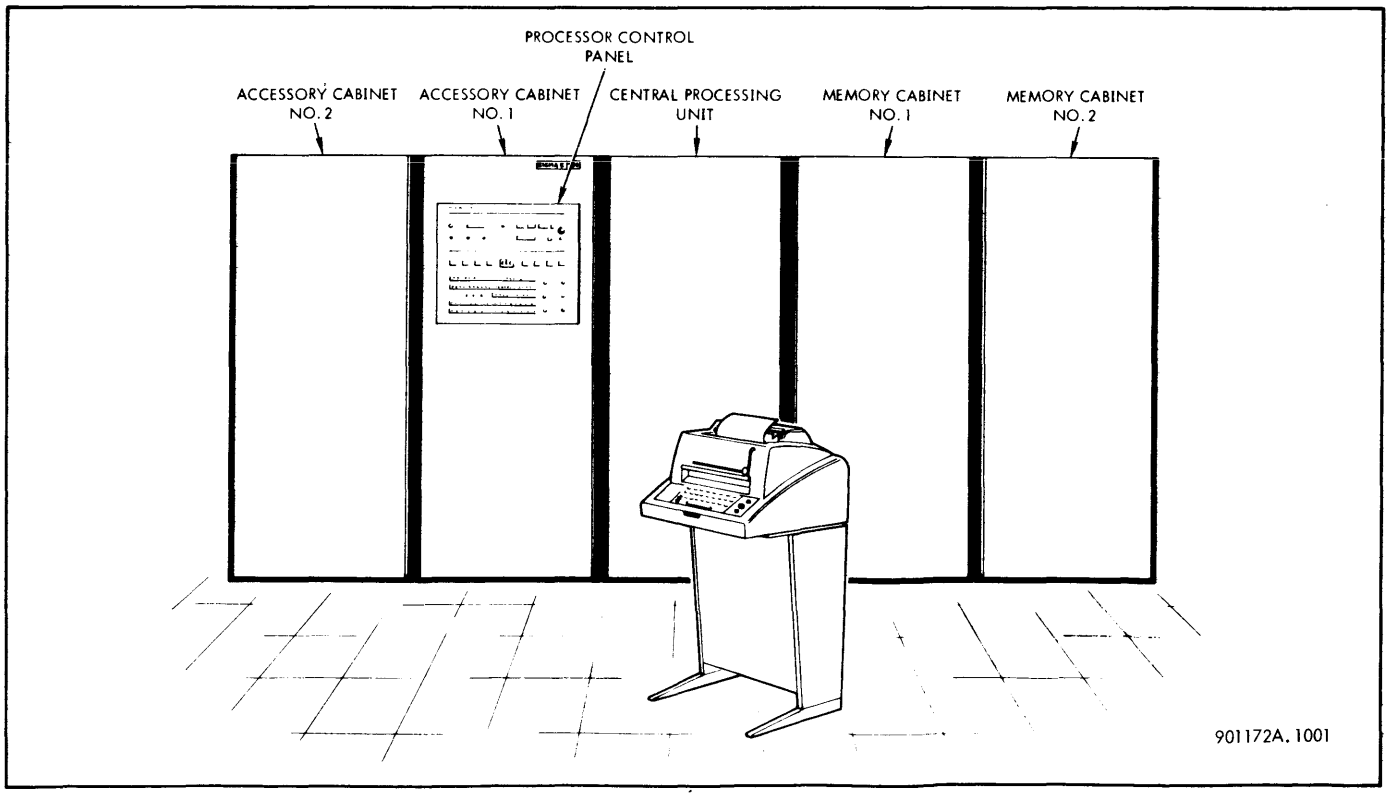


Figure 1-1. Sigma 5 Computer (Typical Configuration)

1-4 EQUIPMENT BREAKDOWN

Within each Sigma 5 system cabinet are two or three racks, either hinged or stationary, identified as frames. One frame may contain a maximum of nine module chassis. A module is a printed circuit board that fits into a slot in a chassis. Figure 1-2 shows the location of a module, chassis, and frame in a computer cabinet.

Table 1-1 lists the models in a basic Sigma 5 system, but does not include any of the several available input/output controllers which are normally housed in accessory cabinets.

1-5 COMPUTER CONFIGURATION

The Sigma 5 computer consists of one CPU cabinet, one to four memory cabinets, and at least one accessory cabinet.

The CPU cabinet contains two swinging frames that hold the active circuit boards and logic wiring and one stationary frame that mounts the PT14 and PT15 ac/dc and dc/ac power converters and the power distribution box. The logic power supply, PT16, is mounted on the sides of the swinging frames. See figure 1-3.

Each memory cabinet contains one or two swinging frames that hold from 4K to 16K of memory each. One stationary frame holds one or two memory port expanders if this option is present. For example, a memory cabinet containing only 8K of memory and no port expander will contain only a single frame. The PT17 memory power supplies are mounted on the sides of the memory frame. See figure 1-4.

The first accessory cabinet (accessory cabinet No. 1) contains the processor control panel (PCP) and at least one I/O device controller. This cabinet may also contain the optional floating point feature and an external multiplexing I/O processor. See figure 1-5. The frame-mounted PT18 power supply is required for input/output device controllers in the accessory cabinets. Power supplies PT14 and PT15 may be mounted in frame 3 of the accessory cabinets to meet power-loading requirements.

Additional accessory cabinets may be required to house additional priority interrupts and I/O equipment such as magnetic tape and disc file controllers, A/D and D/A converters, and so forth.

1-6 OPTIONAL FEATURES

Optional features that may be added to the basic computer are listed in table 1-2. Many of these features are made up of additional modules to be plugged into the CPU; others are added to accessory cabinets or to memory cabinets.

The following optional equipment is added by plugging additional modules into the chassis in the CPU cabinet: power fail-safe, floating point arithmetic, two additional real-time clocks (two real-time clocks are part of the basic computer), and memory protection. Three private memory register blocks, in addition to the one block contained in the basic computer, may be included in the CPU logic modules in the CPU cabinet. The remaining additional register blocks are obtained by adding separate chassis to accessory cabinets. Each register extension chassis may contain four private memory register blocks. The first three external interrupt chassis have a specific location in the CPU cabinet; others are added to the accessory cabinets. Each external interrupt chassis provides control and mounting space for up to eight interrupt modules, with two interrupt levels per module. In the memory cabinets, the first memory is always in frame 2, and the second is always in frame 1. Port expansion logic for both memories is located in frame 3. The first multiplexing IOP is always placed in accessory cabinet No. 1, frame 1; additional external IOP's are located in other accessory cabinets.

1-7 FUNCTIONAL DESCRIPTION

1-8 BASIC COMPUTER DESCRIPTION

For purposes of description, a minimum system is defined as one comprising a CPU, a 4K memory, a device controller, and a device, as shown in figures 1-6 and 1-7. The computer may comprise a CPU with an integral IOP (Model 8201) or a CPU without an integral IOP (Model 8202) and a 4K memory.

Although the CPU consists, physically, of rows of modules, certain basic functional elements can be identified. These are two real-time clocks, a watchdog timer, seven internal interrupt levels, arithmetic and control logic and associated register, a clock generator, a 1.024-mhz clock oscillator, and a 16-register block of private memory. The functions of the CPU are to address core memory, fetch and store information, perform arithmetic and logical operations, sequence and control instruction execution, and control the exchange of information between core memory and other elements of the system.

The memory contains magnetic core storage, addressing logic, port priority logic, control logic, a timing signal generator; also drive, predrive, inhibit, and sensing circuits. All memory is directly addressable by both the CPU and the IOP. Partial words may be stored in the form of 8-bit bytes and 16-bit halfwords.

The integral IOP contains input and output data storage registers and buffers, fast-access memory register for command manipulation, a timing signal generator, and control logic. The function of the integral IOP is to control and sequence input and output operations for eight (expandable to 32) peripheral devices simultaneously, allowing the CPU to concentrate on program execution. The active devices time-share the hardware in the integral IOP. For each device connected to the integral IOP, a storage unit called a subchannel is included in the IOP. All input/output events that require CPU intervention are brought to the attention of the CPU by means of the interrupt system. The device controllers and devices are described in other technical manuals.

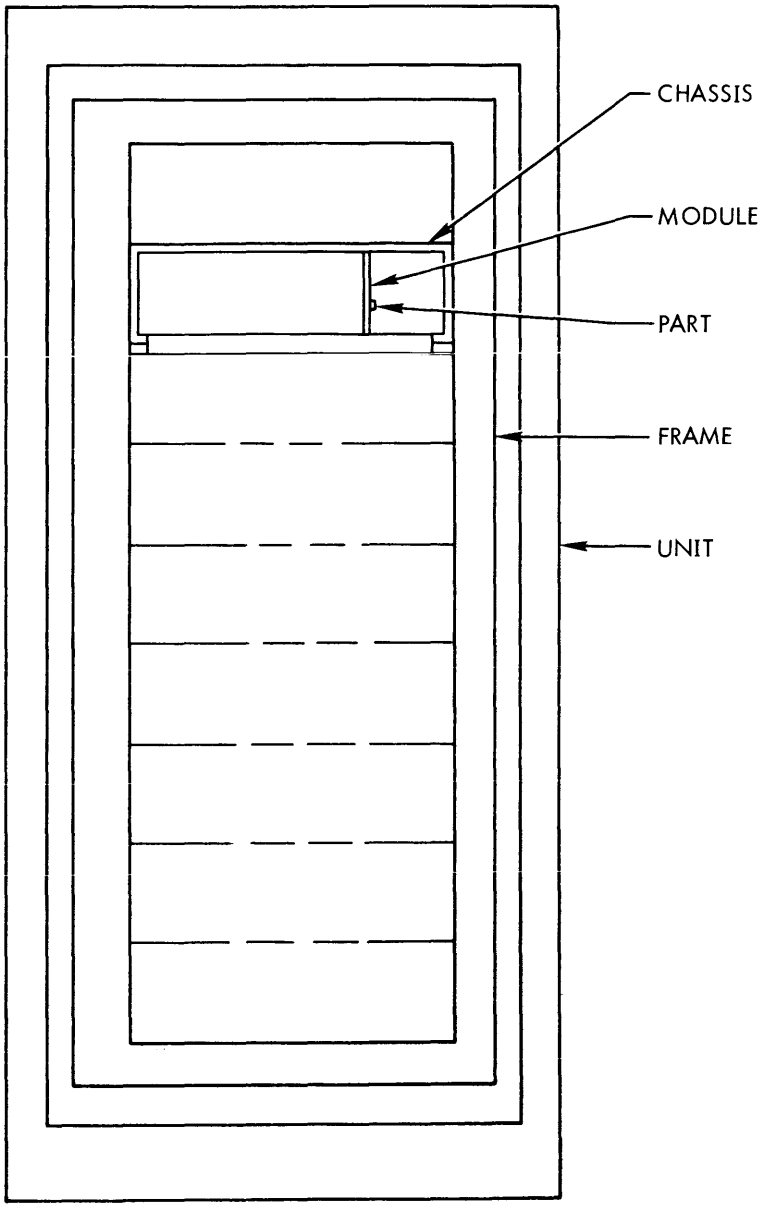


Figure 1-2. Equipment Breakdown

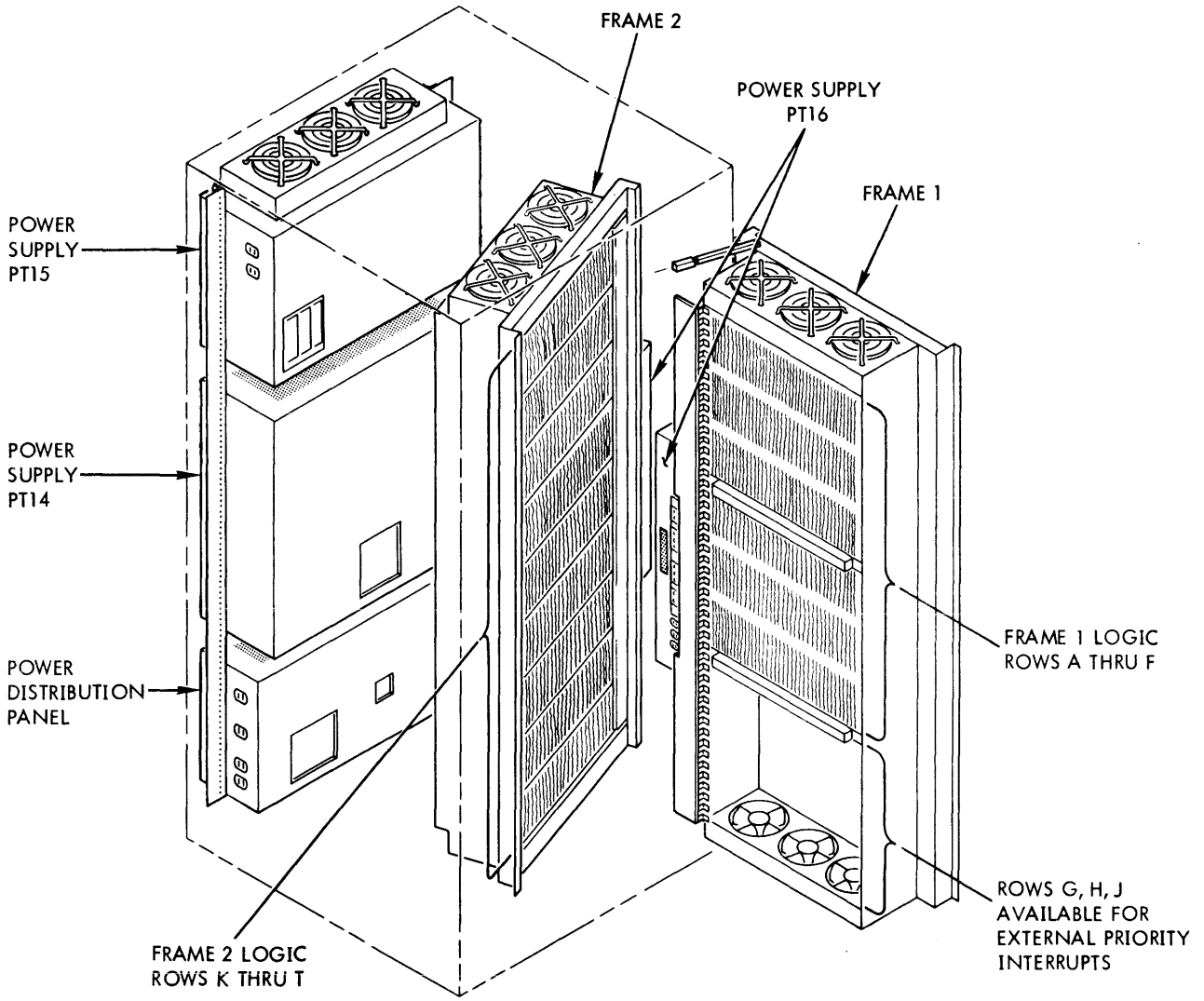


Figure 1-3. CPU Cabinet

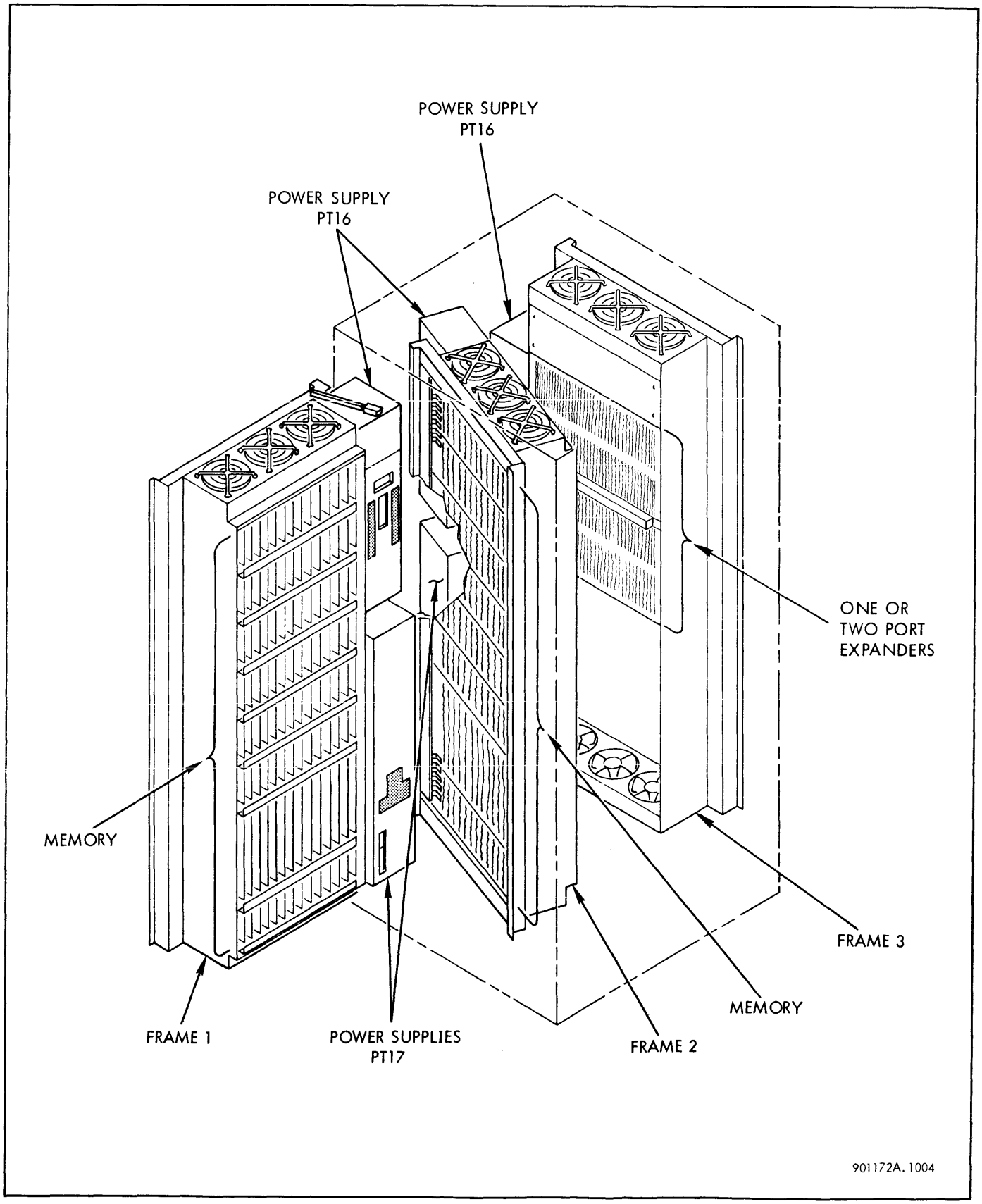


Figure 1-4. Memory Cabinet (Typical)

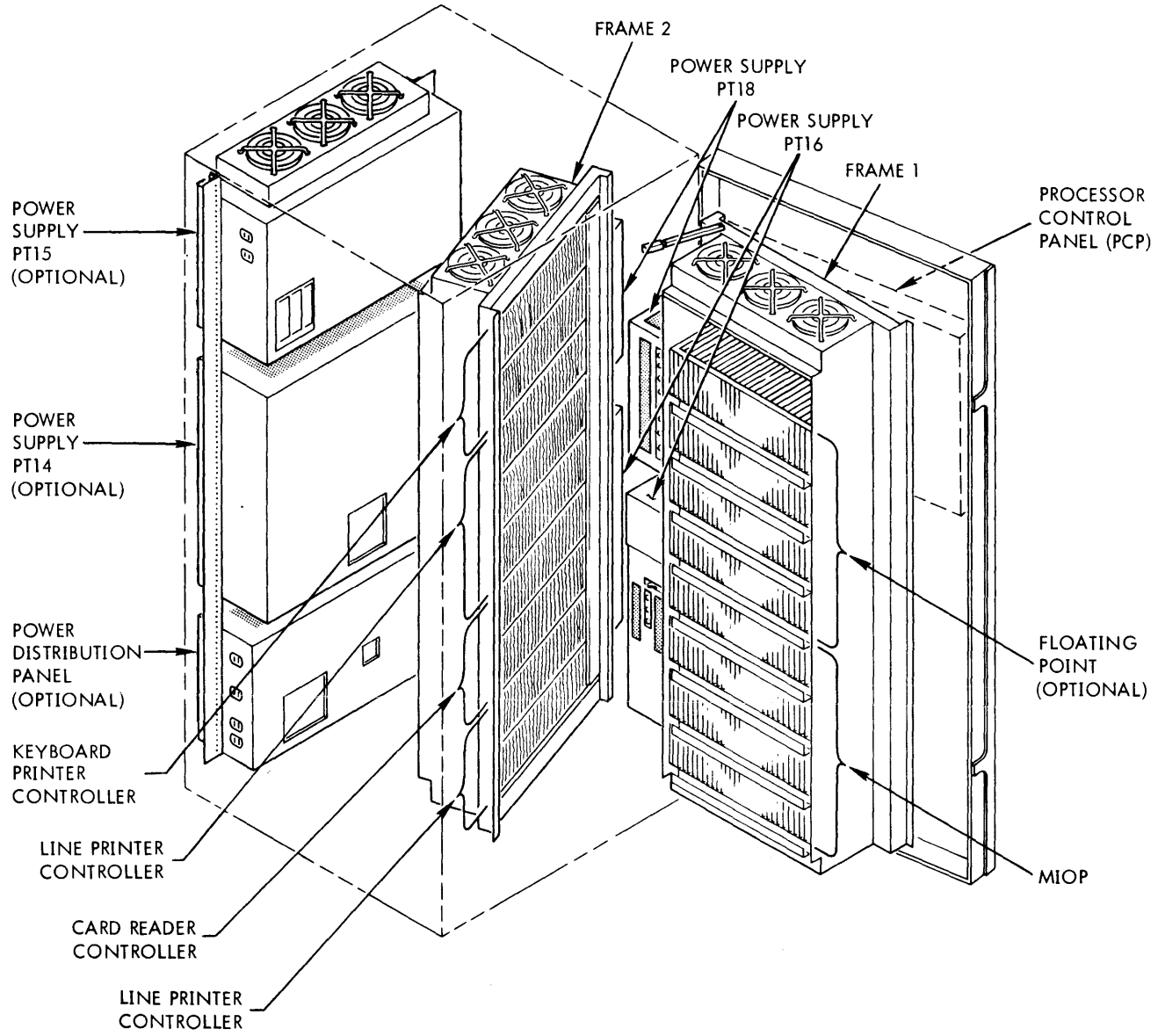


Figure 1-5. Accessory Cabinet No. 1 (Typical)

Table 1-1. Main Units

Model No.	Nameplate Nomenclature or Assembly Drawing Title	Common Name	Assembly Drawing No.	Location
8201	SDS Sigma 5	Central processing unit (CPU) with integral IOP	117282	CPU cabinet and accessory cabinet No. 1
8202	SDS Sigma 5	Central processing unit (CPU) without integral IOP		CPU cabinet and accessory cabinet No. 1
8203	Integral IOP	Integral IOP	137086	CPU cabinet
8251	Basic 4K x 33 bit	4K memory	132546	Memory cabinet

Table 1-2. Optional Features

Model No.	Nameplate Nomenclature or Assembly Drawing Title	Common Name	Assembly Drawing No.	Location
8211	Real-time clock	Two additional real-time clocks	117616	CPU cabinet
8213	Power fail-safe	Power fail-safe	117612	CPU cabinet
8214	Memory protection feature	Memory protection	117617	CPU cabinet
8216	Additional register block	Private memory		
8218	High speed register page		117621	CPU or Accessory cabinet
8221	Register extension unit		130071	Accessory cabinet
8222	REU Interface		132208	Accessory cabinet
8218	Floating point feature	Floating point	134099	Accessory cabinet
8221	Priority interrupt	External interrupt chassis	117330	CPU cabinet or accessory cabinet
8222	Interrupt 2 level	Interrupt, two levels	132206	CPU cabinet or accessory cabinet
8252	Memory expansion kit 4K to 8K	Memory expansion to 8K	117638	Memory cabinets 1, 2, 3, or 4
	Memory expansion kit 8K to 12K	Memory expansion to 12K	117639	Memory cabinets 1, 2, 3, or 4
	Memory expansion kit 12K to 16K	Memory expansion to 16K	117640	Memory cabinets 1, 2, 3, or 4
8255	Two-way access	One- to two-port expander	129463	Memory cabinets 1, 2, 3, or 4
8256	Three-way access	Two- to three-port expander	128125	Memory cabinets 1, 2, 3, or 4
8257	Memory port expander F	Three- to six-port expander (first)	130625 (one memory)	Memory cabinets 1, 2, 3, or 4
	Memory port expander S	Three- to six-port expander (second)	130626 (two memories)	
8270	External interface feature	External interface	137086	Accessory cabinet
8271	Input/output processor	Multiplexing input/output	117610	Accessory cabinet
8272	IOP/DC expansion	Additional eight subchannels	117618	Accessory cabinet
8281	Selector I/O processor A	Selector IOP	117620	Accessory cabinet
8284	Selector I/O processor B	Selector IOP chassis mod kit	117620	Accessory cabinet

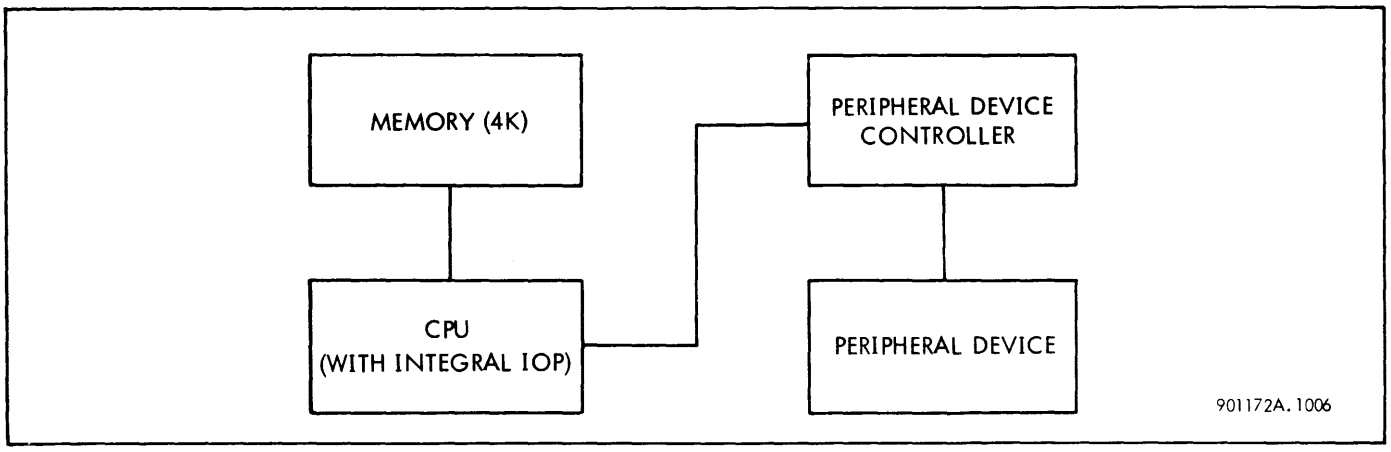


Figure 1-6. Sigma 5 Minimum System With Integral IOP

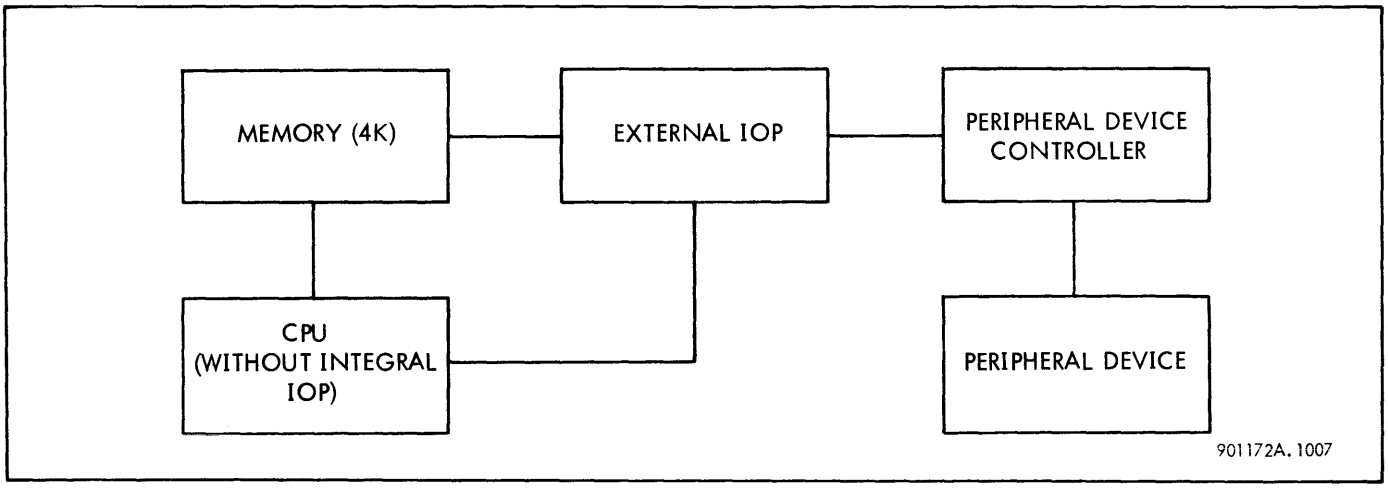


Figure 1-7. Sigma 5 Minimum System Without Integral IOP

1-9 COMPUTER OPTIONAL FEATURES

1-10 Two Additional Real-Time Clocks

This feature adds interrupt capability for two additional real-time clocks in addition to the two already in the CPU. With this feature installed, the CPU has four independent real-time clocks, each separately controlled by programming. The clocks can be used either as elapsed time counters or as real-pulse accumulators.

1-11 Power Fail-Safe Feature

The power fail-safe feature detects an imminent failure of primary power and, with the help of programming, brings the system to an orderly halt while power is still at a sufficient level to permit reliable operation. After shutdown, this feature automatically senses that power has returned to a normal level, and causes the machine to resume computation under program control at the point of prior interruption. The contents of all volatile registers are saved in nonvolatile magnetic core memory before shutdown occurs. The register contents are restored as part of the startup routine.

1-12 Memory Protection

The memory protection feature allows both real-time (foreground) programs and background programs to be run concurrently. A foreground program is protected against destruction by an unchecked background program. The memory protection feature allows protected areas of memory to be written into only under specified conditions.

1-13 Private Memory Register Extension

The private memory register extension provides additional private memory registers in blocks of 16 registers each. Up to 15 additional private memory register blocks may be added, making a total of 16 blocks in the computer.

1-14 Floating Point

The floating point feature enables floating point arithmetic to be performed, using both 32- and 64-bit precision.

Normalized or unnormalized modes of addition and subtraction may be selected by the program.

1-15 External Interrupts

The maximum external interrupt system provides 224 interrupt levels in addition to those already existing internally in the CPU. Each level can be individually armed or enabled under program control. External interrupts are added to the computer in groups of 16, and priorities are established at the time of installation.

1-16 Memory Expansion

Memory size can be expanded in increments of 4096 words up to a maximum of 131,072 words.

1-17 Port Expansion

Each memory block may have from one to six entry ports, each of which may be connected to a memory bus containing data and address lines and control signal lines. Each memory bus provides access to memory for one CPU or IOP. The basic computer includes one port for each memory block; an optional second or third port may be added for two- or three-way access. An optional expander to four ports may be added to either the second or third ports to provide six-way memory access. Since each CPU or IOP has its own bus to any memory block, a computer with more than one memory block can have more than one memory access occurring simultaneously.

1-18 Multiplexing Input/Output Processor

The multiplexing input/output processor controls and sequences input/output operations for eight to thirty-two peripheral devices simultaneously to provide input/output capabilities in addition to those provided by the optional Sigma 5 integral IOP. The MIOP incorporates up to 32 input/output channels in eight channel increments. The device controllers attached to the first eight channels of the MIOP can handle up to 16 devices each; the remaining channels can handle one device each.

1-19 Additional Eight Subchannels (IOP)

To increase the number of devices connected to one IOP, additional subchannels may be added in increments of eight up to a maximum of 32 subchannels for 32 devices.

1-20 Selector Input/Output Processor

The selector IOP provides control, sequencing, and data transmission for up to 32 high-speed peripheral devices operating one at a time. These devices may have data rates that would exceed the bandwidth of the multiplexing IOP or would use up so large a percentage of that bandwidth as to make it impractical to run any other device concurrently. In case a second high-speed data path is required, an optional additional selector channel may be added, identical to the first selector IOP. This optional additional selector channel may share the same memory bus as the first.

1-21 Six Internal Interrupt Levels

In addition to the seven internal interrupt levels included in the standard computer, six more internal interrupts are optional.

1-22 MAXIMUM COMPUTER SYSTEM

A maximum Sigma 5 computer system may consist of up to eight 16K memories, eight three- to six-port expansion units, three register extension units, and 14 external interrupt chassis, in addition to the standard features in the CPU. For maximum I/O capabilities, input/output processors may be connected in any of the combinations listed in table 1-3.

Table 1-3. Maximum Computer System IOP Combinations

Multiplexing IOP's*	Selector IOP's	Additional Selector Channels	Total IOP's
5	0	0	5
4	2	2	8
3	3	2	8
2	3	3	8
1	4	3	8
0	4	4	8

*Includes integral IOP

A block diagram of a typical maximum computer system is shown in figure 1-8.

1-23 SPECIFICATIONS AND LEADING PARTICULARS

The general specifications for the Sigma 5 are given in table 1-4.

The input power specifications for the power supplies used in the computer are given in table 1-5. Power supply PT14 receives 60-hz power from the main power source and supplies 60 vdc to the PT15 power supply. The 120-vac, 2000-hz output of the PT15 power supply is used as an input to the PT16, PT17, and PT18 power supplies. Since the PT14 and PT15 power supplies are always in series, the input and output power specifications are given as if the two were one power supply. The power output from the PT16, PT17, and PT18 power supplies, in watts, is determined by the power requirements of the computer as indicated in table 1-6. This table represents an arbitrary computer containing all possible optional features in the CPU. The total power requirements from table 1-6 may be used to calculate the necessary power supply input in table 1-5. Power requirements for peripheral devices are given in the technical manuals for those devices.

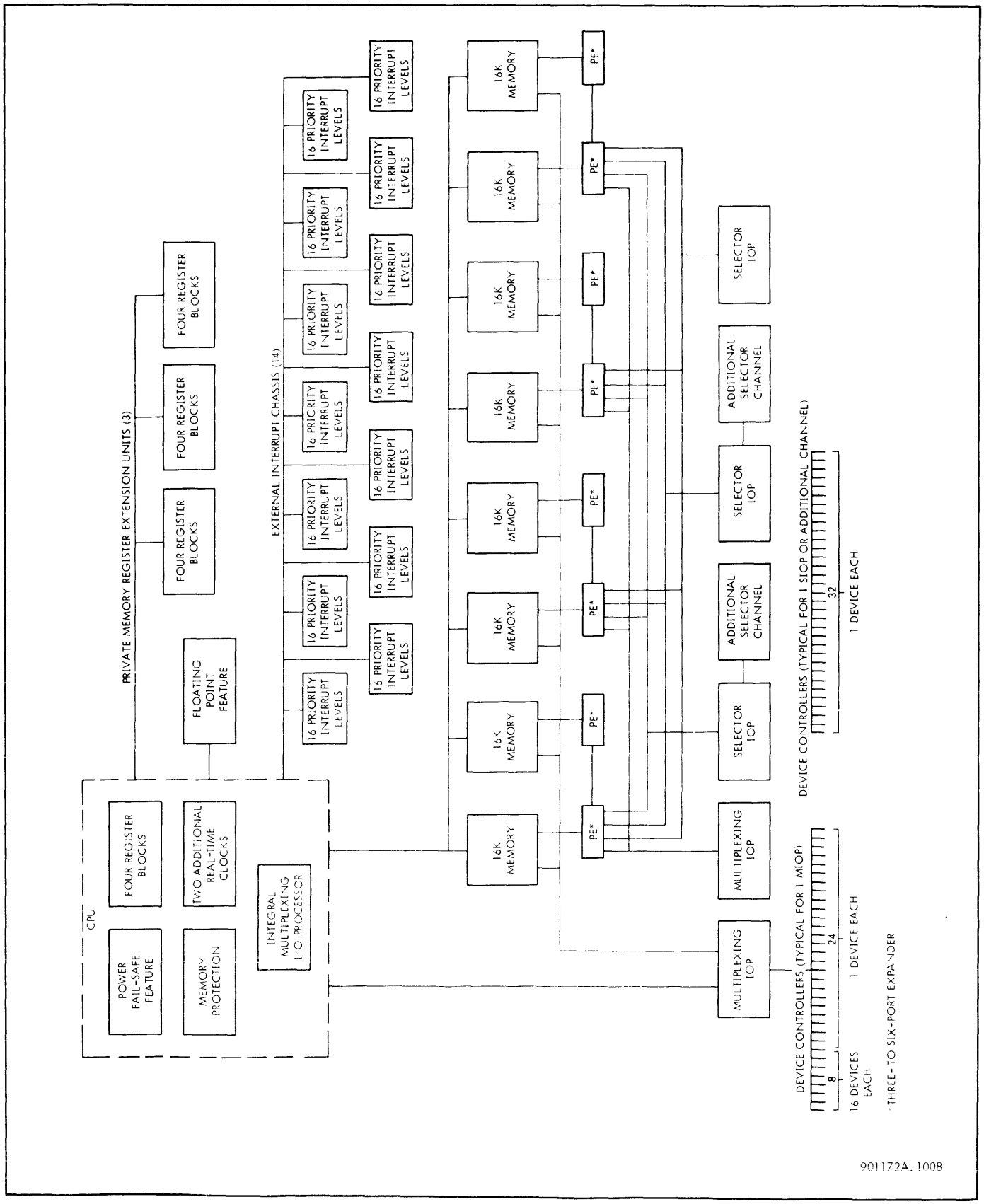


Figure 1-8. Sigma 5 Maximum System (Typical)

Table 1-4. General Specifications

Characteristic	Specification
Temperature (electronics)	
Nonoperating	-40°C to +60°C (-40°F to +140°F)
Operating	5°C to 50°C (41°F to 122°F)
Relative Humidity (operating)	10% to 95%
Altitude	
Nonoperating	20,000 feet maximum
Operating	10,000 feet maximum
Memory cycle	
Without interleaving	800 nanoseconds
With interleaving	635 nanoseconds, effective
Logic signal levels	ONE: +4v; ZERO: 0v
Word length	32-bits plus parity bit
Data format	8-bit byte, 16-bit halfword, fixed point and floating point word, fixed point and floating point doubleword
Coding	Binary, Hexadecimal, EBCDIC

Table 1-5. Power Supply Input Power Specifications

Power Supply	Power Input (volt-amperes)
PT14, PT15	1.66 times volt-amperes output (2000 Hz)
PT16 (2400-Hz input)	150 + 1.36 times dc output in watts (table 1-6)
PT17 (2400-Hz input)	150 + 1.36 times dc output in watts (table 1-6)
PT18 (2400-Hz input)	150 + 1.36 times dc output in watts (table 1-6)

Table 1-6. Computer Power Requirements

UNIT	POWER REQUIREMENTS OF PT16 (AMPS)			POWER REQUIREMENTS OF PT17 (AMPS)		TOTAL DC POWER (WATTS)
	+8v	-8v	+4v	+24v	Drive Voltage (25v max)	
CPU, frame 1	18.5	2.6	19.0			245
CPU, frame 2	20.0	1.3	35.0			310
16K memory	11.0	5.1	14.0	2.0	20.0	484

Table 1-6. Computer Power Requirements (Cont.)

UNIT	POWER REQUIREMENTS OF PT16 (AMPS)			POWER REQUIREMENTS OF PT17 (AMPS)		TOTAL DC POWER (WATTS)
	+8v	-8v	+4v	+24v	Drive Voltage (25v max)	
Port expander (1)	1.0	0.4	4.0			27
Multiplexing IOP	9.0	2.4	20.0			171
Register extension unit, including 4 register blocks	4.0	0.4	13.0			87
Processor control panel	3.5	2.5	0			48

SECTION II
OPERATION AND PROGRAMMING

2-1 GENERAL

This section is divided into two main categories: operating instructions and programming description. Sigma 5 operating instructions describe the purpose and function of the processor control panel (PCP), its control switches and displays. The paragraphs describing programming include instruction and data formats, memory addressing, indexing, and indirect addressing. Descriptions of individual instructions can be found in the Sigma 5 Computer Reference Manual (SDS 900959), or in the operation code descriptions in section III of this manual.

2-2 OPERATION

The following paragraphs describe the operating procedures required during maintenance of the computer. All operations are carried out from the PCP.

2-3 CONTROLS AND INDICATORS

The PCP is divided into two parts: a maintenance section on the upper half of the panel, and an operator's or programmer's section on the lower half. The various control switches, indicators, and displays on the PCP are shown in figure 2-1. Table 2-1 lists the switches and indicators found on the programmer's section of the PCP with their reference designators and a brief description of their functions. A similar list for the switches and indicators on the maintenance section of the PCP is given in table 2-2.

2-4 OPERATING PROCEDURES

The following paragraphs describe step by step manual procedures for the various operations from the processor control panel.

2-5 Applying Power

When the POWER switch is pressed both ac and dc power are applied to the CPU and to all units connected to it. When ac power is applied to the system, the POWER switch is lit. Application of power sets the CPU to initial conditions as described in table 2-1.

2-6 Displaying Contents of Memory Location

To display the contents of any memory location or the contents of any current private memory register perform the following steps:

a. Set the COMPUTE switch to IDLE. The PHASE display will indicate that the CPU is in phase PCP2.

b. Place the address of the memory location (or of the register of the current register block) into the SELECT ADDRESS switches by moving the DISPLAY switch to the SELECT ADDR position.

c. Move the DISPLAY switch to the momentary SELECT ADDR position and return to center.

d. Observe the binary contents of the selected address in the DISPLAY indicators. Memory protection, if included, is inhibited in this PCP operation.

To observe the contents of a private memory register in a register block other than the one currently displayed by the POINTER field of PSW2, the contents of the POINTER field must first be changed to point to the desired register block. This operation is described in paragraph 2-9.

The contents of the memory location pointed to by the instruction address indicators (address currently in the P-register) may be displayed by performing the following steps:

a. Move the DISPLAY switch to the momentary INSTR ADDR position and return to center.

b. The display indicators will now contain the contents of the location pointed to by the instruction address indicators.

If successive memory locations are to be displayed, move the INSTR ADDR switch to INCREMENT position momentarily and repeat steps a and b.

2-7 Storing Into Memory

Storing data or instructions into memory locations, either in core memory or in private memory, is accomplished by the following steps:

a. Set the COMPUTE switch to IDLE. The PHASE display will indicate that the CPU is in phase PCP2.

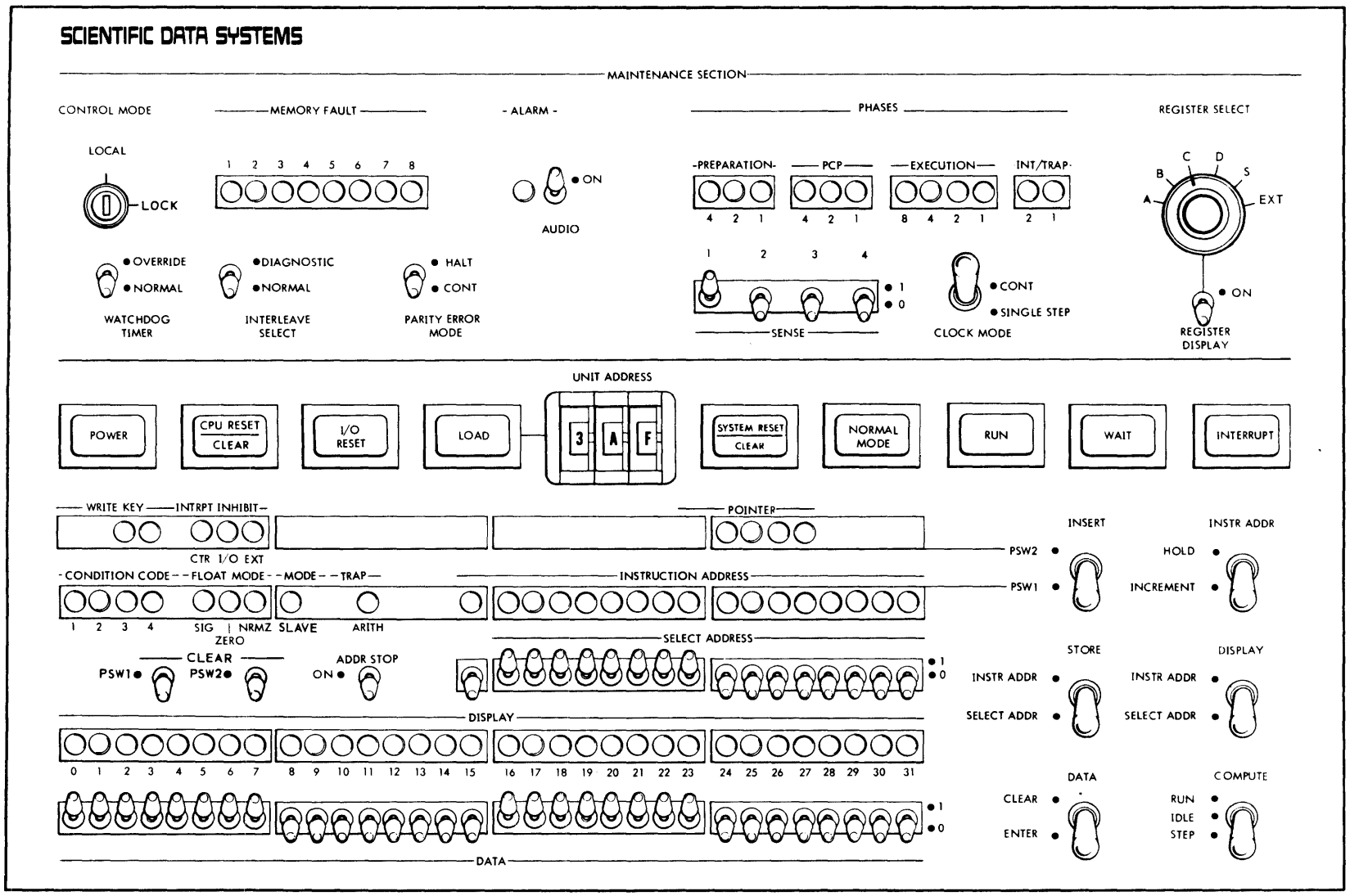
b. Place the address of the memory location into which the data is to be stored into the SELECT ADDRESS switches.

c. Set the single DATA switch to CLEAR. This resets the DISPLAY indicators (D-register).

d. Place the binary information to be stored into the DATA switches. In the DATA switches binary ones are indicated when a switch is in the upper position. The center position of a DATA switch cannot change the current state of the corresponding bit in the D-register.

e. Set the single DATA switch to ENTER and then release. The DISPLAY indicators (contents of D-register) will now assume the same information as the DATA switches.

f. Set the STORE switch to the SELECT ADDR position momentarily and release. The data will be stored in the selected memory location.



901172A.201

Figure 2-1. Sigma 5 Processor Control Panel (PCP)

Table 2-1. Controls and Indicators, PCP Programmer's Section

Control or Indicator	Reference Designator	Function
POWER	Switch S19 Indicator DS28	Push-on, push-off switch that supplies ac power to CPU and all units under its control. When power is first applied, indicator DS28 lights, and a signal is generated in memory power supply to initialize system. All reset functions normally performed by CPU RESET and SYSTEM RESET switches are performed by the POWER switch. When power is applied to the system (indicator DS28 lit), pressing the POWER switch will remove power from the system.
CPU RESET/CLEAR	Switch S18 Indicator DS100	Pressing switch establishes following initial conditions within CPU: a. All interrupts are disarmed and disabled b. WRITE KEY, INTRPT INHIBIT, POINTER, CONDITION CODE, FLOAT MODE, MODE and TRAP indicators are reset c. INSTRUCTION ADDRESS indicators are set to X'25' d. DISPLAY indicators are set to X'02000000', which is load conditions and floating control immediate instruction with R field of zero, to produce "no operation" instruction e. Resets MEMORY FAULT indicators Setting the CPU to initial conditions by pressing CPU RESET switch does not affect any current input-output operation that may be in progress
I/O RESET	Switch S17 Indicator DS99	This switch is used to initialize the standard input-output system by halting all peripheral devices under control of the CPU and resetting all status and control indicators in the input-output system. The I/O RESET switch does not affect any current CPU operation
LOAD	Switch S16 Indicator DS27	Pressing the LOAD switch sets memory to initial conditions to accept an input operation using peripheral input device selected by UNIT ADDRESS switch
UNIT ADDRESS	Switches S15A, S15B, S15C	The three UNIT ADDRESS switches select the peripheral unit to be used in loading process. Unit addresses are hexadecimally notated and provide for up to 2048 different addressing combinations. Addresses of peripheral devices may vary from system to system
SYSTEM RESET/ CLEAR	Switch S18 Indicator DS100	Pressing SYSTEM RESET/CLEAR causes all controls and indicators in the Sigma 5 system to reset. Pressing this switch initializes the memory control logic, resets the MEMORY FAULT indicators, and causes the CPU to perform all the operations described for both the CPU RESET/CLEAR and I/O RESET/CLEAR switches. The SYSTEM RESET/CLEAR and the CPU RESET/CLEAR switches are interlocked so that pressing both switches simultaneously clears core memory to zeros
NORMAL MODE	Indicator DS25	Indicator lights when all the following conditions are satisfied: a. WATCHDOG TIMER switch is set to NORMAL b. INTERLEAVE SELECT switch is set to NORMAL c. PARITY ERROR MODE switch is set to CONT (continue) d. CLOCK MODE switch is set to CONT (continuous) e. All voltage margins are normal

(Continued)

Table 2-1. Controls and Indicators, PCP Programmer's Section (Cont.)

Control or Indicator	Reference Designator	Function
RUN	Indicator DS24	Indicator lights when COMPUTE switch is set to RUN, and no halt condition exists
WAIT	Indicator DS23	Indicator lights when any of following conditions exist: a. CPU is executing wait instruction b. Program is stopped because of ADDR STOP switch c. CPU has attempted to execute instruction in interrupt location other than load or exchange program status doubleword or modify and test instruction
INTERRUPT	Switch S13 Indicator DS22	Switch is used by operator to activate control panel interrupt. If PCP interrupt level is armed, a single pulse is transmitted to interrupt level, advancing it to waiting state. INTERRUPT switch lights when this interrupt level is in waiting state and remains lit until interrupt level advances to active state
WRITE KEY	Indicators DS37-DS38	Two indicators, part of program status word 2 (PSW2), used to control write access in areas of memory when memory protection option is used
INTRPT INHIBIT	Indicators DS34, DS35, DS36	Three indicators, part of PSW2, used to designate which groups of interrupts are allowed or inhibited
POINTER	Indicators DS29-DS32	Four indicators, part of PSW2, used to represent current status of register pointer in CPU
CONDITION CODE	Indicators DS63-DS66	Four condition code indicators, part of program status word 1 (PSW1), used to indicate nature of results of instruction after instruction has been executed
FLOAT MODE SIG ZERO NRMZ	Indicators DS60, DS61, DS62	These three indicators, part of PSW1, represent current control modes for floating point operations: significance, zero, and normalize
MODE SLAVE	Indicator DS59	This indicator, part of PSW1, represents current mode of operation of CPU. Indicator lights when CPU is in slave mode
TRAP ARITH	Indicator DS56	Indicator ARITH, when lit, designates that trap conditions can occur with certain fixed point arithmetic operations. This indicator is part of PSW1
INSTRUCTION ADDRESS	Indicators DS39-DS55	These indicators normally represent the current contents of the P-register in the CPU and are part of PSW1. Address displayed in this field is address of next instruction when REGISTER SELECT switch is at EXT, the indicators normally displaying bits 16 through 25 of the P-register display the I/O address and the indicator normally displaying bit 26 of the P-register displays an internal I/O fast memory signal
CLEAR PSW1	Switch S77	This switch is used to clear the contents of the first program status word to zeros. Resets the condition code bits, the floating arithmetic code bits, the master mode flip-flop, and resets the contents of the P-register to zeros

(Continued)

Table 2-1. Controls and Indicators, PCP Programmer's Section (Cont.)

Control or Indicator	Reference Designator	Function
PSW2	Switch S76	This switch is used to clear the contents of the second program status word to zeros. Resets the write key code bits, the interrupt inhibit flip-flops, and resets the register pointer to zeros
ADDR STOP	Switch S41	ADDR STOP (address stop) switch causes CPU to halt whenever value of INSTRUCTION ADDRESS indicators and value set in SELECT ADDRESS switches or the value of the operand address are equal. When halt occurs, WAIT indicator lights, and instruction in location displayed by INSTRUCTION ADDRESS indicators appears in DISPLAY indicators. Instruction displayed is one that would have been executed next had halt not occurred. Address stop halt is reset when COMPUTE switch is moved from RUN to IDLE. If COMPUTE switch is then moved back to RUN (or to STEP), instruction shown in DISPLAY indicators is next instruction executed. ADDR STOP switch is not effective when selected address is that of private memory registers 00 through 0F
SELECT ADDRESS	Switches S24-S40	Used with ADDR STOP switch to select virtual address at which program is to be halted. They are used to select virtual address of location to be altered when used with STORE switch, and are used to select virtual address of word to be displayed when used with DISPLAY switch
DISPLAY	Indicators DS67-DS98	Indicators display contents of memory word when used with INSTR ADDR, STORE, DISPLAY, and DATA switches. DISPLAY indicators show current contents of internal CPU sum bus and represent the next instruction to be executed when the CPU is placed in the RUN mode
DATA	Switches S44-S75	Thirty-two DATA switches are used to change contents of program status doubleword when used with INSERT switch and to alter value of DISPLAY indicators when used with single DATA CLEAR/ENTER switch. Each DATA switch is inactive in center position and is latching in center and upper (1) positions. In center position, DATA switch represents no change. In upper position each switch represents 1
INSERT	Switch S21	Used to make changes in program status doubleword by manual manipulation. Switch is inactive in center position and is momentary in upper (PSW2) and lower (PSW1) positions. When switch is moved to either PSW1 or PSW2, corresponding portion of program status doubleword is altered according to current state of DATA switches
STORE	Switch S23	Used to change contents of either general register or memory location. Switch is inactive in center position and is momentary in INSTR ADDR and SELECT ADDR positions. When switch is moved to INSTR ADDR, current value of DISPLAY indicators is stored in location shown by INSTRUCTION ADDRESS indicators. When switch is moved to SELECT ADDR, current value of DISPLAY indicators is stored in location shown by SELECT ADDRESS switches
DATA	Switch S43	Single DATA switch is used to change state of DISPLAY indicators. Switch is not active in center position and is momentary in CLEAR and ENTER positions. When switch is moved to CLEAR, all DISPLAY indicators are reset (turned off). When switch is moved to ENTER, display indicators are altered according to state of 32 DATA switches

(Continued)

Table 2-1. Controls and Indicators, PCP Programmer's Section (Cont.)

Control or Indicator	Reference Designator	Function
INSTR ADDR	Switch S20	INSTR ADDR (instruction address) switch is inactive in center position. Upper position (HOLD) is latching, and lower position (INCREMENT) is momentary. When switch is placed in HOLD, the normal process of modifying instruction address portion of program doubleword with each instruction is inhibited. If COMPUTE switch is placed in RUN while INSTR ADDR switch is at HOLD, instruction in location displayed by INSTRUCTION ADDRESS indicators remaining unchanged unless the instruction contains a branch or is a load or exchange doubleword instruction. If COMPUTE switch is moved to STEP while INSTR ADDR switch is at HOLD, instruction is executed once each time COMPUTE switch is moved to STEP, and INSTRUCTION ADDRESS indicators remain unchanged. Each time INSTR ADDR switch is moved from center position to INCREMENT, the following operations are performed: a. Current value of INSTRUCTION ADDRESS indicators is counted up by one b. Contents of virtual address displayed by INSTRUCTION ADDRESS indicators are shown in DISPLAY indicators
DISPLAY	Switch S22	Displays contents of either general register or memory location. Switch is inactive in center position and is momentary in both INSTR ADDR and SELECT ADDR positions. When switch is moved to INSTR ADDR or SELECT ADDR, contents of location shown by indicators or switches, respectively, appear in DISPLAY indicators
COMPUTE	Switch S42	Controls execution of instructions. Center position (IDLE) and upper position (RUN) are both latching. Lower position (STEP) is momentary. When COMPUTE switch is at IDLE, all other control panel switches are operative. When COMPUTE switch is moved from IDLE to RUN, RUN indicator lights and CPU begins to execute instructions as follows: a. Current setting of DISPLAY indicators is taken as next instruction to be executed regardless of contents of location shown by current value of INSTRUCTION ADDRESS indicators b. Value in INSTRUCTION ADDRESS indicators is counted up by one c. Instruction execution continues with instruction in location shown by new value of INSTRUCTION ADDRESS indicators d. Steps b and c are repeated unless program branches out of sequence When COMPUTE switch is in RUN, the only switches operative are POWER, INTERRUPT, ADDR STOP, INSTR ADDR (in HOLD position), and switches in the maintenance section of control panel. Each time COMPUTE is moved from IDLE to STEP, the following operations occur: a. Current setting of DISPLAY indicators is taken as an instruction, and instruction is executed.

(Continued)

Table 2-1. Controls and Indicators, PCP Programmer's Section (Cont.)

Control or Indicator	Reference Designator	Function
COMPUTE (Cont.)		b. Current value of INSTRUCTION ADDRESS indicators is counted up by one. If stepped instruction was a branch instruction and branch should occur, INSTRUCTION ADDRESS indicators are set to the value of the effective address of branch instruction c. Instruction in location shown by new value of INSTRUCTION ADDRESS indicators is displayed in DISPLAY indicators If instruction is being stepped (executed by moving COMPUTE switch from IDLE to STEP), all controllable interrupt levels are temporarily inhibited while instruction is being executed; however, traps can occur. In this case, the XPSD instruction in the appropriate trap location is executed as if the COMPUTE switch were in RUN. Thus, if trap occurs during stepped instruction, program status doubleword display (PSW1 and PSW2) automatically reflects effects of XPSD instruction, and DISPLAY indicators then contain first instruction of trap routine

Table 2-2. Controls and Indicators, PCP Maintenance Section

Control or Indicator	Reference Designator	Function
CONTROL MODE	Switch S3	CONTROL MODE switch is a two-position key lock. When switch is in LOCAL, all controls and indicators on the PCP are operative. In LOCK, the following switches on the PCP are operative: POWER, INTERRUPT, all SENSE switches, and AUDIO. When the CONTROL MODE switch is in LOCK the following switches are interlocked to the following states regardless of their actual settings a. COMPUTE switch to RUN b. WATCHDOG TIMER switch to NORMAL c. INTERLEAVE SELECT switch to NORMAL d. PARITY ERROR MODE switch to CONT e. CLOCK MODE switch to CONT
MEMORY FAULT	Indicators DS14-DS21	Since the system is limited to no more than eight memory blocks, each MEMORY FAULT indicator corresponds to a specific memory block. Whenever a memory parity error occurs or an overtemperature condition exists in a memory block, the appropriate indicator lights and remains lit until indicator is reset. The MEMORY FAULT indicators can be reset by pressing CPU RESET or SYSTEM RESET switch or by read direct instruction coded to read MEMORY FAULT indicators. If MEMORY FAULT indicator is lit because corresponding memory block is beyond its maximum temperature range, and condition still exists when indicator is reset, it will immediately be turned on again
ALARM AUDIO	Indicator DS13 Switch S2	Indicator is used to attract operator's attention to some urgent operating condition, and is turned on and off under program control by execution of properly coded write direct instruction. When ALARM

(Continued)

Table 2-2. Controls and Indicators, PCP Maintenance Section (Cont.)

Control or Indicator	Reference Designator	Function
ALARM (Cont.)		indicator is lighted and AUDIO switch is ON, a 1-kHz signal is set to PCP speaker. ALARM indicator is reset by CPU RESET or SYS RESET switch
PREPARATION PHASES	Indicators DS10, DS11, DS12	Display one of four CPU phases during instruction preparation, or IOP subphases with REGISTER SELECT switch at EXT
PCP PHASES	Indicators DS7, DS8, DS9	Display CPU phases (PCP1 through PCP7 in binary notation) during PCP operation, or I/O service call in indicator 4 (DS9) when REGISTER SELECT switch is at EXT
EXECUTION PHASES	Indicators DS3-DS6	Display CPU phases (PH1 through PH10 in binary notation) during instruction execution, or internal IOP execution phases with REGISTER SELECT switch at EXT
INT/TRAP PHASES	Indicators DS1, DS2	Indicators are lighted when either an interrupt or a trap condition occurs to display the interrupt/trap phases of operation
REGISTER SELECT	Switch S1	Used to display contents of selected internal registers. With COMPUTE switch at IDLE, register selected by REGISTER SELECT switch may be shown in DISPLAY indicators by moving REGISTER DISPLAY switch to ON. When REGISTER DISPLAY switch is returned to inactive position, DISPLAY indicators display contents of sum bus. With REGISTER SELECT switch at EXT and CLOCK MODE switch in center position: a. I/O phases are displayed in EXECUTION PHASE indicators DS3-DS6, I/O subphases are displayed in PREPARATION PHASE indicators DS10-DS12, I/O service call is displayed in PCP phase indicator 4 (DS9), I/O address is displayed in the INSTRUCTION ADDRESS indicators that normally display bits 16 through 25 of the P-register (DS45-DS54), and internal I/O fast memory signal is displayed in the INSTRUCTION ADDRESS indicator that normally displays P-register bit 26 (DS44) b. Floating point data (if option present) is displayed in DISPLAY indicators DS67-DS98 according to switch settings on ST14 module in location 06A of floating point unit. Switches permit display of contents of floating point sum bus and A-, B-, and D-registers (upper and lower), as well as miscellaneous floating point control signals
WATCHDOG TIMER	Switch S12	CPU can be interrupted at end of each instruction and at certain points during execution of some instructions. An interval of not more than 40 μsec may occur between any two interruptible points. Watchdog timer is reset at each interruptible point and counts at a rate of 1-mhz between interruptible points. If count in watchdog timer reaches 40, CPU traps to location X'46'. When WATCHDOG TIMER switch is in OVERRIDE, watchdog timer is inoperative. When switch is in NORMAL, watchdog timer is operative

(Continued)

Table 2-2. Controls and Indicators, PCP Maintenance Section (Cont.)

Control or Indicator	Reference Designator	Function
INTERLEAVE SELECT	Switch S11	With this switch in NORMAL, interleaving between memory blocks is in effect. When switch is at DIAGNOSTIC, memory addresses are not interleaved between memory blocks
PARITY ERROR MODE	Switch S10	Controls action of CPU when a memory error occurs. If switch is at CONT (continue) when parity error occurs, appropriate MEMORY FAULT indicator lights, and an interrupt signal is transmitted to memory parity interrupt level. If switch is at HALT when a parity error occurs, appropriate MEMORY FAULT indicator lights, and CPU halts operation. Memory block in which error has occurred will not be available until its MEMORY FAULT indicator is reset
SENSE	Switches S6-S9	Switches are used under program control to set condition code portion of program status doubleword. When write direct instruction is executed in internal control mode, condition code is set according to state of the four SENSE switches, which are always operative. Normally, SENSE switches are used in this manner during diagnostic or other test routines
CLOCK MODE	Switch S5	Controls internal CPU clock. When switch is at CONT (continuous), clock operates at normal speed. When switch is in inactive (center) position, however, CPU clock pulses are inhibited. Under these circumstances a single clock will be generated each time CLOCK MODE switch is moved to SINGLE STEP position. As clock is stepped manually in this manner, PHASE indicators reflect CPU phase during each pulse of the clock
REGISTER DISPLAY	Switch S4	When switch is at ON, contents of register selected by REGISTER SELECT switch will be displayed in DISPLAY indicators. Switch is active only when CLOCK MODE switch is in center position

Memory protection, if included, is inhibited in this PCP operation. To store data into a private memory register block other than the one currently displayed by the POINTER field of PSW2, the contents of the POINTER field must be changed to point to the desired register block. This operation is described in the next paragraph.

Storing data into the memory location pointed to by the instruction address register (current address in the P-register) can also be accomplished by performing steps a, c, and d, and substituting step f1, following, for step f.

f1. Set the STORE switch to the INSTR ADDR position momentarily and release. The data will be stored in the memory location addressed by the instruction address indicators (current address in the P-register).

2-8 Clearing the Program Status Words

The contents of PSW1 may be reset to zeros by moving the CLEAR PSW1 switch to the momentary PSW1 position. The contents of PSW2 may be reset to zeros by moving the CLEAR PSW2 switch to the momentary PSW2 position.

2-9 Altering the Current Program Status Doubleword

Changing any of the data in the current PSD requires that PSW1 and PSW2 be treated separately. Changing any field of the PSD is accomplished by the following steps:

a. Set the COMPUTE switch to idle.

b. Enter the desired information into the 32 DATA switches only in those bit positions of PSW1 or PSW2 to be changed. In those bit positions in the fields where no change is to be made, the corresponding DATA switches must be in the center (no change) position. If any bit positions are to be changed from ONES to ZEROS, the PSW1 or PSW2 must be cleared with the CLEAR PSW1 or PSW2 switch.

c. Set the INSERT switch to PSW1 if the change is to be made in that portion of the PSD, or to PSW2 if the change is to be made in that portion of the PSD.

d. Release the INSERT switch. The new information will be entered into the program status doubleword.

2-10 Branching From the PCP

To cause the CPU to branch to any instruction in memory, regardless of what instruction is currently being executed, the following steps should be carried out:

a. Set the COMPUTE switch to IDLE.

b. Enter the address of the instruction to which it is desired to branch in the 17 least significant bits of the INSTRUCTION ADDRESS field of PSW1. (See paragraph 2-9.)

c. Move the DISPLAY switch momentarily to INSTR ADDR.

d. The instruction has been read from memory and will be the next instruction performed by the CPU.

e. Set the COMPUTE switch to either RUN or STEP.

2-11 Stepping Through a Program

It is often necessary when debugging programs or when maintaining the equipment to sequence slowly through the program one instruction at a time, observing the results of each instruction after it has been executed. This is accomplished by performing the following steps:

a. Set the COMPUTE switch to IDLE, and branch to that part of the program from where it is desired to step. See paragraph 2-10.

b. Set the COMPUTE switch to STEP. In the DISPLAY indicators the contents of the next instruction will be displayed.

c. The results of the instruction just executed can be seen by displaying the contents of the memory location or private memory register affected by the instruction. See paragraph 2-6, steps b, c, d, and e.

d. Repeat steps b, c, and d above to continue the program sequence step by step.

2-12 Single Clocking an Instruction

During maintenance operations it is often necessary to sequence through individual instructions from one clock period to the next, observing the results of the CPU internal registers after each clock pulse. To single clock instructions in this manner, the following steps are performed:

a. Branch to the malfunctioning instruction (see paragraph 2-10), or enter an identical instruction into the display (see paragraph 2-7).

b. Set the CLOCK MODE switch to its center position. This inhibits all clock pulses. The COMPUTE switch may be set to RUN at this point.

c. Set the CLOCK MODE switch to SINGLE STEP. This causes the instruction to sequence to its next phase.

d. Observe the contents of the affected internal registers by setting the REGISTER SELECT switch to the proper register position and by setting the REGISTER DISPLAY switch to ON.

e. After all affected internal registers have been observed and if no malfunction is seen, repeat steps c, d, and e.

In most single clock operations as just described, the INSTR ADDR switch can be placed in the HOLD position if it is desired to repeat the single clock operation through the instruction more than once.

2-13 Single Instruction Repetition

Single clocking a malfunctioning instruction as described in paragraph 2-12 may pinpoint the area of the malfunction without actually allowing the observer to determine what is causing the faulty condition. In some cases, an error may consistently occur while the CLOCK MODE switch is in the CONT (continuous) position, but may never occur when the switch is in the SINGLE STEP position. This could be caused by a slow gate or active circuit element. In such case, the operator should run the single malfunctioning instruction repeatedly using the oscilloscope to observe all signals that could be the cause of the error condition.

To run a single instruction repeatedly, the following steps should be followed:

a. Branch to the malfunctioning instruction. (See paragraph 2-10.)

b. Set the INSTR ADDR switch to HOLD. This prevents the instruction address field of PSW1 from changing after each execution of the instruction.

c. Set the COMPUTE switch to RUN, and observe all pertinent signals on the oscilloscope as the instruction is executed repeatedly.

Certain instructions (those, for example, in which an operand is changed each time the instruction is executed) cannot be repeated in this manner without destroying data meaningful to the observer. The multiply and divide instructions are examples of this. For this type of instruction it may be necessary to enter a small four- or five-word instruction program loop to establish initial conditions each time the instruction is observed.

2-14 Loading a Program

After the input device has been loaded with the program tape or cards and has been properly prepared to read,

the following steps should be followed to load the program into memory:

a. Set the COMPUTE switch to IDLE.

b. Press the SYSTEM RESET switch.

c. Set the UNIT ADDRESS switches to the address of the desired input peripheral device.

d. Press the LOAD switch.

e. Set the COMPUTE switch to RUN. The CPU will now read the program from the input device and store it in memory.

2-15 PROGRAMMING

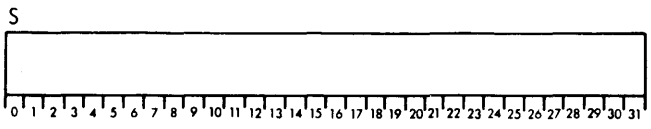
The following discussion of programming is intended to clarify some of the functions and requirements of the Sigma 5 computer. It includes data and instruction formats, addressing requirements, modes of operation, and the instruction repertoire in tabular form. For more detailed operation of individual instructions, see the Sigma 5 Reference Manual (SDS 900959), or refer to section III of this manual.

2-16 WORD FORMATS

2-17 Data Word Formats

Data words consist of 32 binary digits or bits. The CPU is capable of addressing words, doublewords, halfwords, or bytes (quarterwords) for many of its operations.

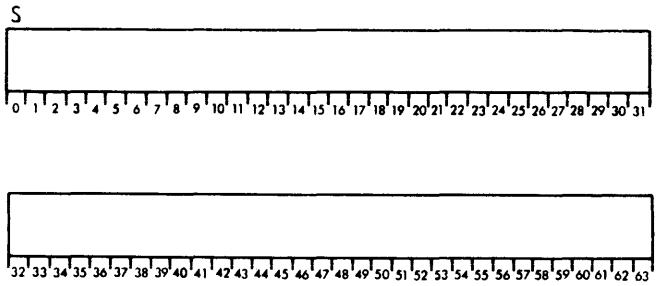
Word. A single word contains 32 bits numbered 0 through 31, from the most significant bit to the least significant bit.



901060A.201

If the binary configuration of ones and zeros in the 32 bit positions of a word represent a numeric value, the binary content of bit 0 is the sign of the value, and the binary configuration in bits 1 through 31 represents the magnitude of the value. Negative numbers in the computer are always held in two's complement form. If the sign bit is a zero, the magnitude of the number is positive; if the sign bit is a one, the magnitude of the number is negative and is represented as the two's complement of its positive form. For example, the decimal number +29 would appear in its hexadecimal form in a word as 0000001D, and the decimal number -29 would appear in its hexadecimal form in a word as FFFFFFE3.

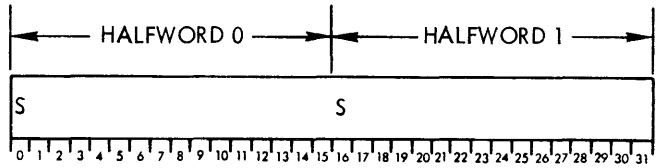
Doubleword. A doubleword in the computer consists of two consecutive 32-bit words, and contains 64 bits numbered 0 through 63.



901060A.202

In doublewords which represent a numeric value, bit 0 represents the sign of the magnitude, and bits 1 through 63 represent the magnitude of the value. A doubleword always consists of two consecutive single words whose addresses are n and n + 1, where n is an even-numbered address.

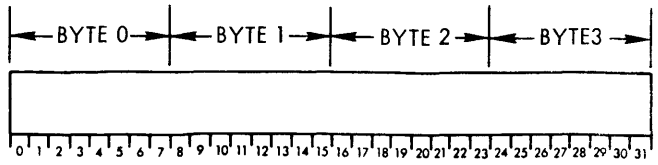
Halfword. Sigma 5 is capable of addressing halfwords. Two halfwords are contained in one single word where halfword HW0 consists of bits 0 through 15, and halfword HW1 consists of bits 16 through 31.



901060A.203

Each halfword is treated by the CPU as though it contains a signed value. Bit 0 of halfword HW0 is the sign of the magnitude contained in bits 1 through 15; bit 16 of halfword HW1 is the sign of the magnitude contained in bits 17 through 31. During halfword operations the integrity of the number contained within the addressed halfword is maintained by extending the sign of the halfword magnitude 16 bit positions to the left. For example, if a halfword is loaded into one of the private memory registers, it will consist of 32 bits with its sign bit extended from bit 16 of the register to bit 0. Halfwords used in all arithmetic operations have their signs extended in the CPU internal registers in this same manner.

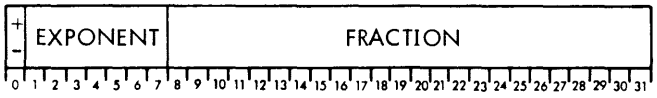
Byte. Four bytes of eight bits each can be contained in one single word where byte 0 consists of bits 0 through 7, byte 1 consists of bits 8 through 15, byte 2 consists of bits 16 through 23, and byte 3 consists of bits 24 through 31.



901060A.204

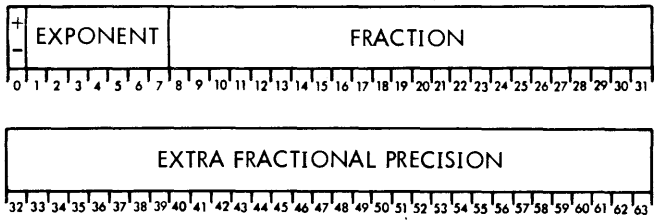
Bytes are addressable singly. Bytes normally contain absolute magnitudes in binary-coded decimal (BCD) form, extended binary-coded decimal interchange code (EBCDIC) characters, or similar types of data.

Floating Point Formats. The computer provides two formats for representing floating point numbers: a short format of 32 bits, and for extra precision, a longer format of 64 bits. The short floating point format consists of a 24-bit fractional magnitude, a 1-bit sign that establishes whether the fraction is positive or negative, and a 7-bit biased exponent. The short format for floating point numbers is shown below.



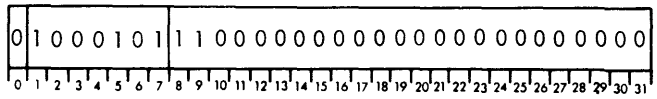
901060A.207

The long floating point format is similar to the short format except that the fraction field is increased from 24 to 56 bits.



901060A.206

Each incremental value of the exponent multiplies the binary value of the fraction by a power of 16; thus, floating point numbers are hexadecimally oriented. For example:



901060A.205

In this illustration the magnitude of the floating point number is the magnitude of the fraction (3/4) multiplied by 16^5, or 0.75 x 64,536 = 46,402.

The floating point fraction is determined by the placement of its binary point, which is fixed at the left of the fraction between bit positions 7 and 8.

The fractional values of any floating point number, n, can be either positive or negative and its exponent can be either positive or negative. Thus, the four different combinations can be grouped in the following manner: $+(16^e)$ n; $+(16^{-e})$ n; $-(16^e)$ n; and $-(16^{-e})$ n. Since the most significant bit of the exponent is the complement of its state, the exponent is always biased by a value of 64. For positive fractional values the positive exponents are not two's complemented; for positive fractional values the negative exponents are two's complemented. The following two positive fractions, one with a positive exponent of 16^4 and the other with a negative exponent of 16^{-4}, illustrate this rule.

$+(16^4)$ n = 0 1 0 0 0 1 0 0 . . . fraction n . . .

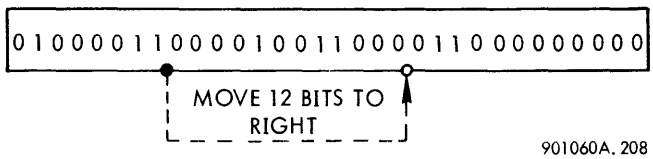
$+(16^{-4})$ n = 0 0 1 1 1 1 0 0 . . . fraction n . . .

For negative fractional values, positive and negative exponents are the one's complements of the corresponding exponents of the positive fractional values.

$-(16^4)$ = 1 0 1 1 1 0 1 1 . . . fraction n . . .

$-(16^{-4})$ = 1 1 0 0 0 0 1 1 . . . fraction n . . .

A simple method of determining the actual value of any floating point number, whether an integer, a fraction, or a mixed number, is to move the fixed binary point to the right or to the left the number of bit positions equal to four times the value in the exponent field. For example, to determine the value of the following floating point number, move the fixed point from its position between bit positions 7 and 8 to the right a number of bit positions determined by multiplying the exponent value by 4.



901060A.208

The value of this mixed number (integer and fraction) is the decimal equivalent 152.375. This method of determining the actual value of a floating point number may be simpler than the method of determining the fractional value and then multiplying this value by the third power of 16; for example, 1219/32, 768 x 4096 = 152.375.

The following examples of floating point numbers are shown in hexadecimal notation with their corresponding decimal values.

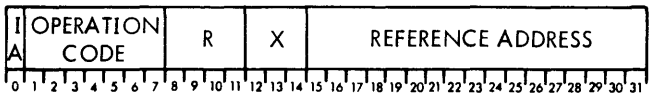
Hexadecimal	Decimal
435F5000	+1525
425F5000	+95.625
415F5000	+5.95703125
4105F500	+0.372314453
405F5000	+0.372314453
BDA0B000	-1525
BEA0B000	-95.625
BFA0B000	-5.95703125

A normalized floating point number is one in which the fractional value is equal to or greater than 1/16. For example, the floating point number X'43100000' is normalized, but the floating point number X'4401000' is not, although both numbers are equal.

2-18 Instruction Formats

Instructions in the CPU fall into two general classes: those that require a reference address field and those that contain an operand within the instruction word.

Reference Address Instructions. The normal reference address instruction has the following format:



901060A.211

The basic operation code of the instruction is contained in bits 1 through 7 of the instruction word.

The R-field, bits 8 through 11, addresses one of 16 private memory registers (R0 through RF). The reference address field, bits 15 through 31, represents the address of a location in memory from which the operand is to be taken or into which data is to be stored.

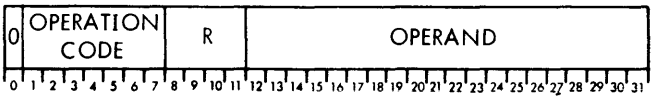
The X-field, bits 12 through 14, addresses one of seven private memory registers (R1 through R7), which indexes the address contained in the reference address field. If the X-field contains all zeros, the instruction is not indexed; if the X-field does not contain all zeros, then the address contained within the reference address field will be modified by the addition of the contents of the register specified in the X-field.

Bit 0 of the instruction (IA) is an indirect addressing bit. If this bit is a zero, the reference address is the address of the operand. If bit 0 is a one, the reference address is the virtual address of a word in memory which, in turn, contains the virtual address of the operand. Indirect addressing is limited to a single level.

Operation codes are described by two hexadecimal characters and include bits 0 through 7. The most significant of the two hexadecimal digits of a normally addressed instruction will always be a number less than X'8'. Any operation code with its most significant hexadecimal digit 8 or greater means that the instruction is indirectly addressed. For example, a normal add word instruction has the normal operation code X'30'. If the add word instruction is indirectly addressed, the operation code would be X'B0'.

Some instructions with reference address fields do not address memory. In these instructions the contents of the reference address field contain types of information other than memory addresses — usually control or conditional information relating to the operation of the instruction. Instructions that fall into this category are shifts, input-output, read direct, and write direct.

Immediate Operand Instructions. The format for immediate operand instructions follows.



901060A.212

The operation code of an immediate operand instruction specifies that the operand is contained within the instruction itself, that no access to memory is necessary, and that indexing is not possible. Immediate operand instructions cannot be indirectly addressed. If bit 0 of any immediate operand instruction contains a one, the instruction is aborted, and the CPU traps to location X'40'.

In an immediate operand instruction the contents of the R-field specify one of the private memory registers in the CPU. The number contained within the operand field is made up of a sign (bit 12) and a magnitude (bits 13 through 31). During the execution of an immediate operand instruction, the integrity of the value in the operand field is maintained by extending the sign bit 12 places to the left. Thus, the 20-bit immediate operand in bits 12 through 31 may be X'FFF2E' (-210 decimal), but in the course of executing the instruction the value becomes X'FFFFFF2E' (-210 decimal).

Throughout the following paragraphs, several examples of instructions are given. The instructions in these examples use the format indicated in example 1 which, in this case, is an indexed load word (LW) instruction.

Example 1. Instruction Format for Instruction Examples

I/A	Opcode		R	X=3	Reference Address			
0	011 0010		1011	011	0 0000	0000	1010	0011
0	- LW	-	B -	3	-	A3		
	3	2	B	6	0	0	A	3

In all examples wherever the instruction format is shown, its hexadecimal equivalent will also be indicated; for example:

Instruction 0-LW-B-3-A3 X'32B600A3'

Leading hexadecimal zeros of the reference address are omitted.

2-19 MEMORY ADDRESSING

Reference address instructions that require access to memory contain an address location in the reference address field. This reference address is subject to modification by indirect addressing or by indexing, and is referred to as the virtual address.

2-20 Reference Address

The address contained in the reference address field is the reference address. A reference address may or may not be the address of the memory location from which the operand is finally taken since this address is subject to change. If the reference address is not modified in any way during the execution of the instruction, the reference address is also the effective address.

2-21 Effective Address

The effective address is the final address seen by memory and is the address location from which the effective word (or actual operand) is taken or into which it is stored. A reference address may undergo one or two transformations before the address of the effective word is finally defined.

2-22 Indirect Addressing

The address in the reference address field of an indirectly addressed instruction (bit 0 = 1) does not refer to the location of the effective word or actual operand. Rather, it points to a location in memory where the effective operand is to be found. The memory access operation of an indirectly addressed load word (LW) instruction is shown in example 2.

Example 2. Indirect Addressing

Instruction 1-LW-B-0-103A X'B2B0103A'

The instruction addresses the operand indirectly through the contents of location X'103A'.

103A	00000B42	The address in X'103A' is the effective address of the operand.
B42	00000113	The effective operand in X'B42' is X'113'. This number is loaded into private memory register B.

The reference address (X'103A') of the instruction is indirectly addressed (bit 0 = 1); therefore, the contents of this location (X'103A') contain the actual address (X'B42') of the operand or effective word. The operand finally loaded into register B is X'113'.

2-23 Indexed Addressing

If the X-field (bits 12 through 14) of an instruction does not contain all zeros, the instruction is indexed. The contents of the X-field determine which index register (R1 through R7) is to be used in the indexing operation.

When an indexed instruction is executed, the contents of the register specified by the X-field are added to the virtual address of the instruction and the resultant sum becomes the effective address of the operand or storage location. The following example of an indexed subtract word instruction illustrates the operation of an indexed instruction.

Example 3. Indexed Addressing

Instruction 0-SW-C-5-407 X'38CA0407'

R5	000044E6	
RC	0000ABCD	(before execution)
RC	00009ABC	(after execution)
407	XXXXXXXX	(contents undefined)
48ED	00001111	

In this example memory location X'407' is not actually addressed and its contents are not affected in any way. The contents of register R5 added to the virtual address of the instruction result in an effective address of X'48ED'. The contents of memory location X'48ED' are subtracted from the contents of register RC and the difference is stored in register RC.

2-24 Indirect Indexed Addressing

An instruction may be both indexed and indirectly addressed. When this is the case, indexing occurs after indirect addressing takes place rather than before. This is called post-indexing. The following example of a store word (STW) instruction that is both indexed and indirectly addressed shows the addressing relationships. The operand in this instance is located in register R9. The location into which the operand is to be stored is the location resulting from the indirect and indexed addressing.

Example 4. Indirect and Indexed Addressing

Instruction 1-STW-9-3-5B6 'B59605B6'

R3	00000213	
R9	005B0000	
5B6	00000AAB	
CBE	XXXXXXXX	(before execution)
CBE	0005B000	(after execution)

The virtual address X'5B6' in the instruction word is translated into a second virtual address X'AAB'. The contents of register R3 are added to this second virtual address and the sum (CBE) becomes the effective address of the memory location into which the operand in register R9 is stored.

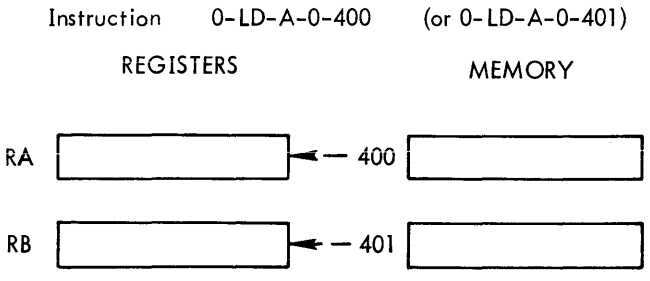
2-25 Doubleword Addressing

A doubleword consists of one even-numbered word and the next consecutive odd-numbered word. This convention applies to doublewords that exist either in core memory or in private memory. An attempt to address an odd-even doubleword combination will result in the CPU forcing an even-odd doubleword address where the first even numbered word is the addressed odd word minus one. For example, the load doubleword instruction 0-LD-2-0-537 will address the memory doubleword located in addresses X'536' and X'537', and not X'537' and X'538'.

The doubleword location in the private memory registers is addressed by the R-field of the instruction. To address the register doubleword, the address in the R-field must be an even-numbered address. Unlike the doubleword address for memory, however, an odd address in the R-field addresses only the odd word of the register doubleword.

The following examples of a load doubleword instruction (LD) illustrate the effect of the instruction when both even- and odd-numbered doubleword addresses are used.

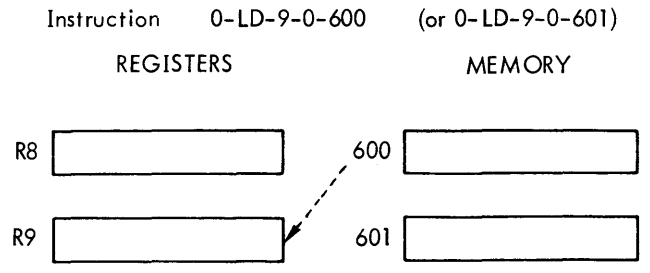
Example 5. Even Doubleword Addresses



901060A.213

Where even doubleword addresses are specified, the data transfer is from memory even word to register even word, and memory odd word to register odd word.

Example 6. Odd Doubleword Addresses



901060A.214

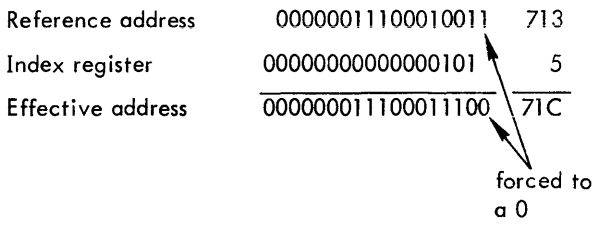
When an odd-numbered register address is placed in the R-field, both words of the effective doubleword are loaded into the same private memory register. As the most significant word of the doubleword is the last to be loaded, private memory register R contains the most significant word at the end of the instruction.

Using an odd-numbered register address in a doubleword instruction is a legitimate programming strategem and is not forbidden.

2-26 Indexed Doubleword Instructions

The least significant binary digit of a memory doubleword address in an instruction is always considered by the CPU to be a zero even though it may actually be a one. Thus, doubleword address boundaries start with even-numbered word locations. For example, a doubleword could consist of word X'406' and X'407', but not of words X'407' and X'408'. If the programmer were to address a memory doubleword as X'407', the CPU would address the doubleword contained in memory locations X'406' and X'407'.

When a doubleword address instruction is indexed, the index register is shifted to the left one bit position before the addition takes place, and therefore, any number in the index register is, in effect, twice its normal value when used for indexing. For example, an instruction addressing the doubleword X'713' will address words X'712' and X'713'. If the contents of the index register are equal to 5, the actual doubleword addressed in memory will be the doubleword located in X'71C' and X'71D'.



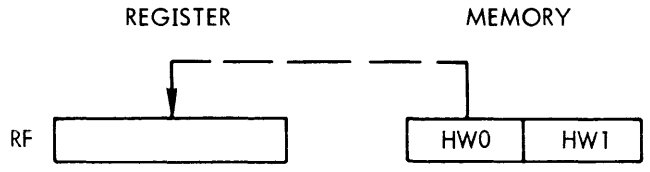
2-27 Halfword Addressing

Two halfwords, HW0 and HW1, can be placed within one 32-bit register or memory location. Halfword HW0 consists of bits 0 through 15, and halfword HW1 consists of bits 16 through 31. (See paragraph 2-17.)

A halfword instruction addressing the left-hand halfword HW0 uses the same address as though it were addressing the full word. The halfword instruction addressing the right-hand halfword HW1 also uses the full word address, but the instruction X-field must refer to one of the index registers in private memory, and this index register must contain a one in its low order bit. The next two examples show the operation and addressing scheme for loading halfwords HW0 and HW1 into register RF.

Example 7. Load Halfword HW0

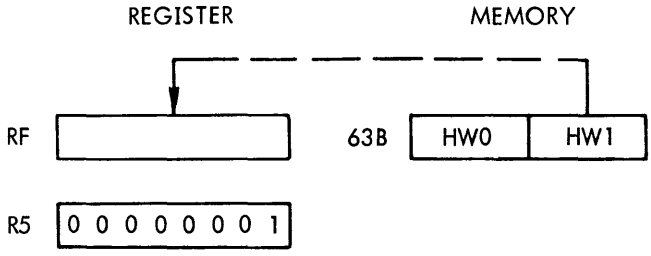
Instruction 0-LH-F-0-63B X'52F0063B'



901060A.215

Example 8. Load Halfword HW1

Instruction 0-LH-F-5-63B X'52FA063B'



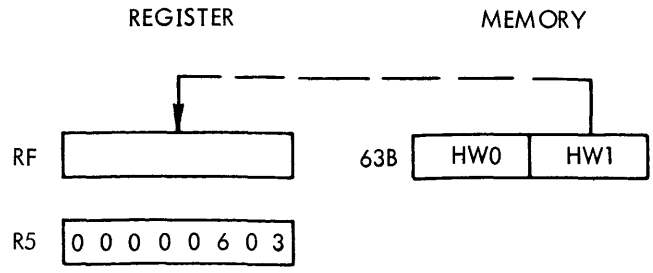
901060A.216

In each of the load halfword instructions shown in examples 7 and 8, the sign of the halfword is extended 16 places to the left before it is loaded into register RF. Thus, if the contents of instruction HW0 in example 7 were X'FF0A', register RF would be loaded with X'FFFFFF0A'; if the contents of instruction HW1 in example 8 were X'0004', register RF would be loaded with X'00000004'.

Use of the index register in example 8 to designate that HW1 was addressed does not imply that the instruction was an indexed instruction or that the contents of the reference address was modified in any manner. Neither should it be inferred that halfword instructions cannot be indexed. Example 9 shows how the halfword instruction in example 8 could have been indexed.

Example 9. Indexing Halfword Instructions

Instruction 0-LH-F-5-33A X'52FA033A'



901060A.217

In halfword instructions that are indexed, the index register is shifted to the right by one bit position so that bit 30 of the index register is aligned with bit 31 of the reference address. Bit 31 of the index register does not modify the actual operand address, but is used by the internal logic of the CPU to distinguish which halfword is addressed. The binary addition of index register R5 to the reference address of the instruction in halfword operations is shown below.

Reference address	0	0000	0011	0011	1010
+ Index register		0000	0011	0000	0001 1
Sum (actual address)	0	0000	0110	0011	1011

If no indexing is desired when addressing halfword HW1, the referenced index register must contain a one in bit position 31 and zeros in bit positions 14 through 30. (See example 8.)

2-28 Byte Addressing

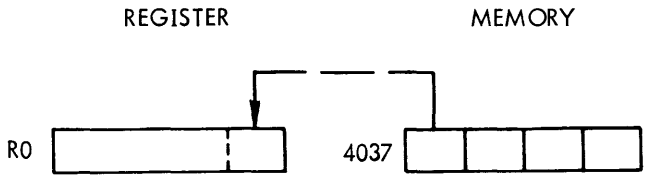
Four 8-bit bytes can be contained within one 32-bit register or memory location. Byte 0 consists of bits 0 through 7, byte 1 consists of bits 8 through 15, byte 2 consists of bits 16 through 23, and byte 3 consists of bits 24 through 31.

An instruction that addresses bytes operates in a manner similar to one addressing halfwords in that no indexing is required for the left-hand byte, but the index register must be specified and contain the proper information for the other bytes. The index register is displaced by two bits (instead of one as for halfword addressing) for byte operations affecting bytes 1, 2, and 3. The two least significant bits of the index register (bits 30 and 31) determine which of the three right-hand bytes is addressed.

The following four examples show how each of the four bytes are addressed in a load byte (LB) instruction. In each of the examples of the LB instructions that follow, the addressed byte is loaded into bit positions 24 through 31 of the addressed register, and bits 0 through 23 are cleared to zeros.

Example 10. Load Byte 0

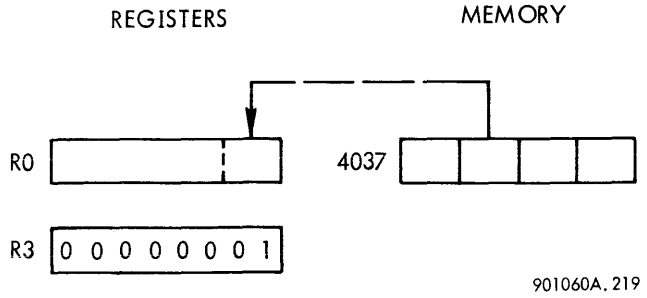
Instruction 0-LB-0-0-4037 X'72004037'



901060A.218

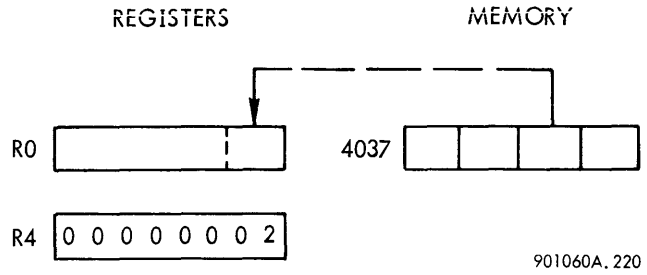
Example 11. Load Byte 1

Instruction 0-LB-0-3-4037 X'72064037'



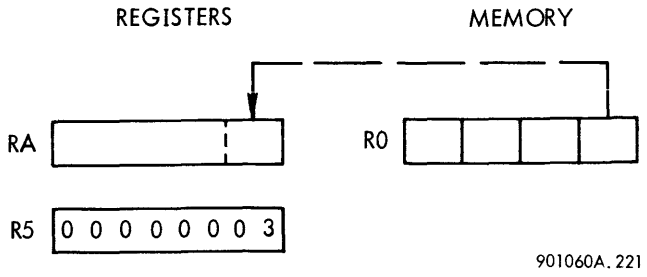
Example 12. Load Byte 2

Instruction 0-LB-0-4-4037 X'72084037'



Example 13. Load Byte 3

Instruction 0-LB-A-5-0 X'72A40000'



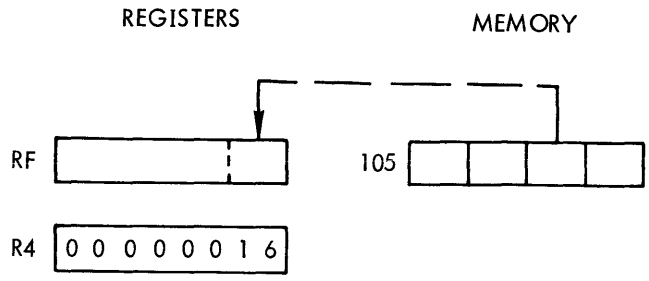
Core memory is not involved during the execution of the instruction in example 13 since reference address 0 refers to a private memory register rather than to a core address.

None of the operand addresses in the load byte instructions in examples 11, 12, and 13 are indexed since all of the indexed registers contain zeros in bit positions 0 through 29. During the execution of these load byte instructions the index register is shifted right two places in respect to the reference address. Thus, only bits 13 through 29 can be added to the virtual address. If these index bits are all zeros, the virtual address remains unchanged.

The following example shows how the load byte instruction may be indexed.

Example 14. Indexing a Byte Address Instruction

Instruction 0-LB-F-4-100 X'72F80100'



The binary addition of the contents of index register R4 to the virtual operand address of the instruction is performed in the following manner:

Virtual address	00000000100000000
+ Index register	00000000000010110
Sum (actual address)	00000000100000101

The two least significant bits of the index register are used to designate which byte (byte 2 in this instance) is to be loaded. These bits are not added to the virtual address. Bits 13 and 14 of the index register are added to bits 15 and 16 of the virtual address.

2-29 BASIC INSTRUCTIONS

Table 2-3 lists all the basic operation codes, including those instructions that are optional or privileged. For detailed operation of each instruction see Sigma 5 Reference Manual (SDS 900959), or refer to the operation code descriptions in section III of this technical manual.

Table 2-3. Basic Instructions

Mnemonic	Code	Instruction Name
Load-Store		
LI	22	Load Immediate
LB	72	Load Byte
LH	52	Load Halfword
LW	32	Load Word
LD	12	Load Doubleword
LCH	5A	Load Complement Halfword
LAH	5B	Load Absolute Halfword

(Continued)

Table 2-3. Basic Instructions (Cont.)

Mnemonic	Code	Instruction Name
Load-Store (Cont.)		
LCW	3A	Load Complement Word
LAW	3B	Load Absolute Word
LCD	1A	Load Complement Doubleword
LAD	1B	Load Absolute Doubleword
LS	4A	Load Selective
LM	2A	Load Multiple
LCFI	02	Load Conditions and Floating Control Immediate
LCF	70	Load Conditions and Floating Control
XW	46	Exchange Word
STB	75	Store Byte
STH	55	Store Halfword
STW	35	Store Word
STD	15	Store Doubleword
STS	47	Store Selective
STM	2B	Store Multiple
STCF	74	Store Conditions and Floating Control
Analyze-Interpret		
ANLZ	44	Analyze
INT	6B	Interpret
Logical		
OR	49	OR Word
EOR	48	Exclusive OR Word
AND	4B	AND Word
Floating Point Arithmetic (Optional Instructions)		
FAS	3D	Floating Add Short
FAL	1D	Floating Add Long
FSS	3C	Floating Subtract Short
FSL	1C	Floating Subtract Long
FMS	3F	Floating Multiply Short
FML	1F	Floating Multiply Long
FDS	3E	Floating Divide Short
FDL	1E	Floating Divide Long

(Continued)

Table 2-3. Basic Instructions (Cont.)

Mnemonic	Code	Instruction Name
Fixed Point Arithmetic		
AI	20	Add Immediate
AH	50	Add Halfword
AW	30	Add Word
AD	10	Add Doubleword
SH	58	Subtract Halfword
SW	38	Subtract Word
SD	18	Subtract Doubleword
MI	23	Multiply Immediate
MH	57	Multiply Halfword
MW	37	Multiply Word
DH	56	Divide Halfword
DW	36	Divide Word
AWM	66	Add Word to Memory
MTB	73	Modify and Test Byte
MTH	53	Modify and Test Halfword
MTW	33	Modify and Test Word
Comparison		
CI	21	Compare Immediate
CB	71	Compare Byte
CH	51	Compare Halfword
CW	31	Compare Word
CD	11	Compare Doubleword
CS	45	Compare Selective
CLR	39	Compare With Limits in Register
CLM	19	Compare With Limits in Memory
Shift		
S	25	Shift
SF	24	Shift Floating
Push-Down		
PSW	09	Push Word
PLW	60	Pull Word
PSM	0B	Push Multiple
PLM	0A	Pull Multiple
MSP	13	Modify Stack Pointer

(Continued)

Table 2-3. Basic Instructions (Cont.)

Mnemonic	Code	Instruction Name
Execute-Branch		
EXU	67	Execute
BCS	69	Branch on Conditions Set
BCR	68	Branch on Conditions Reset
BIR	65	Branch on Incrementing Register
BDR	64	Branch on Decrementing Register
BAL	6A	Branch and Link
Call		
CAL1	04	Call 1
CAL2	05	Call 2
CAL3	06	Call 3
CAL4	07	Call 4

(Continued)

Table 2-3. Basic Instructions (Cont.)

Mnemonic	Code	Instruction Name
Control (Privileged Instructions)		
LPSD	0E	Load Program Status Doubleword
XPSD	0F	Exchange Program Status Doubleword
LRP	2F	Load Register Pointer
MMC	6F	Move to Memory Control
WAIT	2E	Wait
RD	6C	Read Direct
WD	6D	Write Direct
Input-Output (Privileged Instructions)		
SIO	4C	Start Input-Output
HIO	4F	Halt Input-Output
TIO	4D	Test Input-Output
TDV	4E	Test Device
AIO	6E	Acknowledge Input-Output

SECTION III
PRINCIPLES OF OPERATION

3-1 INTRODUCTION

This section provides general and detailed principles of operation of the Sigma 5 computer. The general principles are presented on a block diagram level and stress the overall functions of the equipment. The detailed principles are presented on a logic and circuit diagram level and emphasize the operation of logical functions within the major elements of the equipment.

3-2 GENERAL PRINCIPLES OF OPERATION

The Sigma 5 is organized around one or more central processor units (CPU), magnetic core memories, input-output processors (IOP), device controllers, and peripheral devices. One of each major element is shown in figure 3-1. These elements operate asynchronously in relation to each other. The IOP shown in the figure may be a multiplexing type or a selector type. A multiplexing IOP allows up to 32 devices to operate simultaneously. A selector IOP allows only one device to operate at a time, but at a high transfer rate. The CPU may also be equipped internally with an integral IOP which allows the CPU to perform input-output operations with no external IOP. In that case, some CPU registers and control circuits are combined with IOP registers to perform input-output operations. The peripheral device in figure 3-1 is shown with a dashed block to indicate that it is not strictly a part of the basic computer, but nevertheless is a major element, its use being implied by the device controller.

3-3 CENTRAL PROCESSOR UNIT

The CPU sequences and controls program execution. In executing operations, the CPU performs arithmetic and logic functions, addresses private memory and core memory, fetches and stores instructions and data, controls information transfer between core memory and other elements connected to the CPU, and performs other subfunctions. The CPU also controls internal and external interrupts and provides manual program control through the processor control panel (PCP). A functional block diagram of the CPU is shown in figure 3-2.

3-4 Arithmetic, Control, and Address Functions

Arithmetic, control, and address functions are performed by the adder, sum bus, CPU registers, and associated control logic (see figure 3-3). In general, registers A and D combined with the adder and sum bus perform the arithmetic operations and other control functions. Register C is used for CPU input; register O holds the opcodes; and registers R, Rp, and P are used for addressing. Register B is used for temporary storage of the program address and as an extended accumulator with the A-register. Registers IOFR, IODA, and IOFM are components of the integral IOP, and the DIO registers are used for read direct and write direct operations. Register MC (macro-counter) is used for iteration counting.

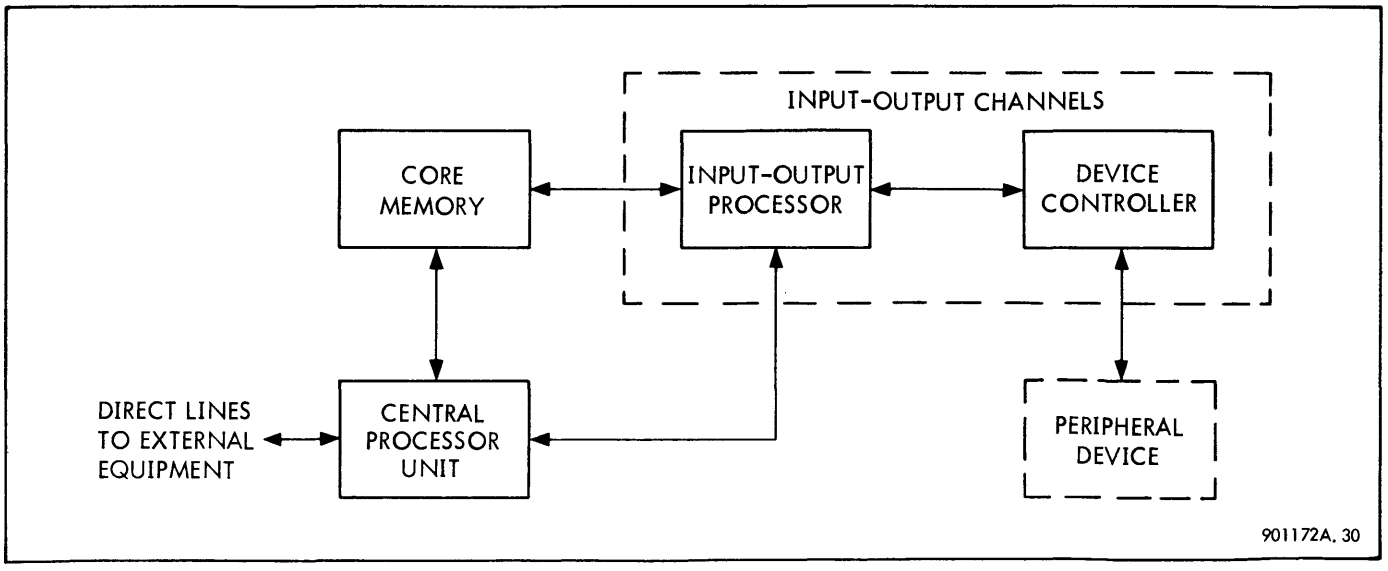


Figure 3-1. Sigma 5 Major Elements

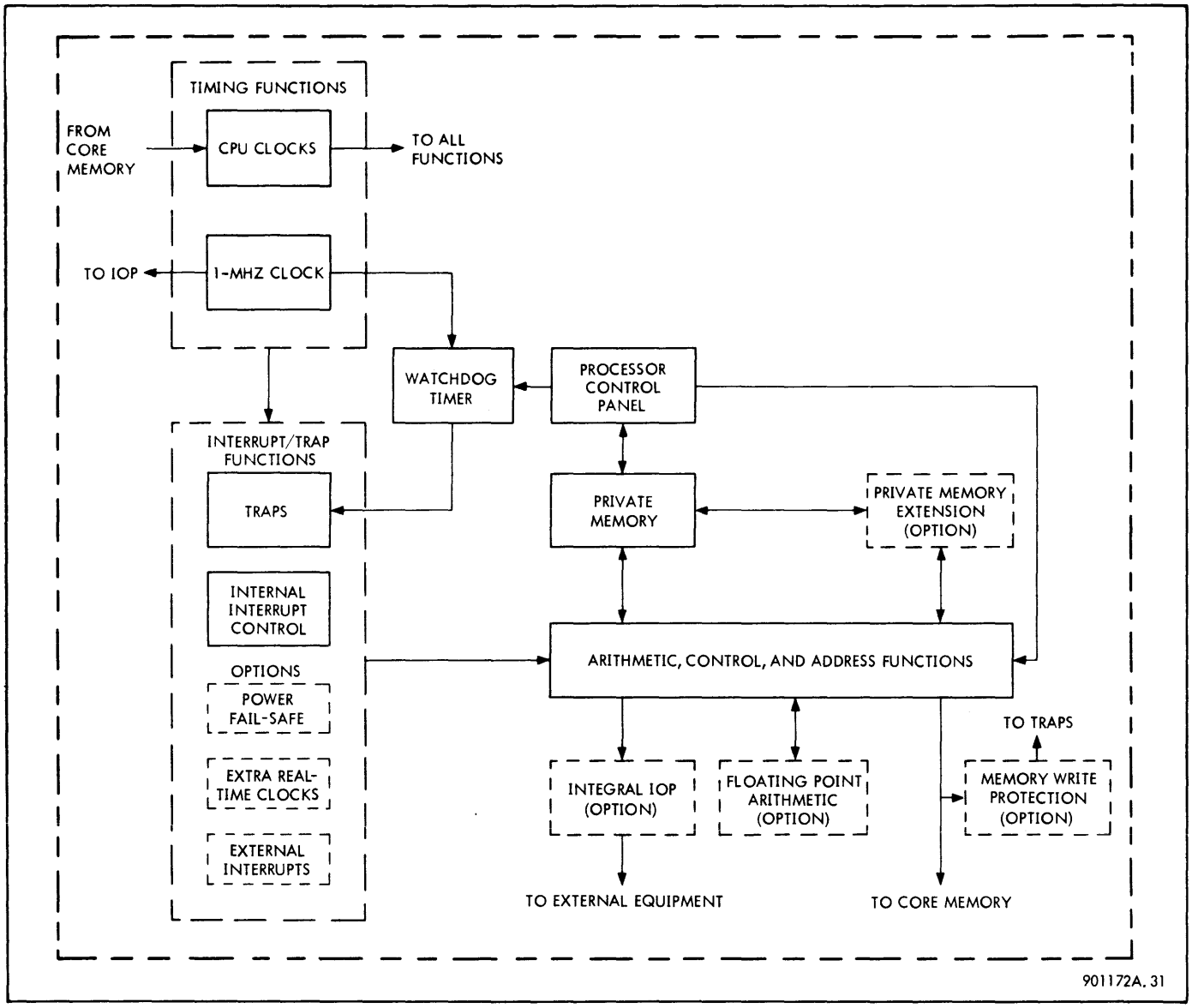


Figure 3-2. Central Processing Unit, Functional Block Diagram

Instructions or data from core memory enter the CPU through the C-register. From the C-register, operation codes are transferred to the O-register, private memory addresses are transferred to the R-register, and the entire word, including reference and index addresses, is transferred to the D-register. Operation codes in the O-register are decoded and activate the signal families peculiar to the operation. Instructions and data from private memory and from I/O fast memory enter the CPU through the C-register or the A-register. The A-register is used during many operations, examples of which include arithmetic functions, left and right shifts, and indexing.

PRIVATE MEMORY ADDRESSING. The address in the R-register is placed on the private memory address lines to address private memory. If private memory is extended to more than one block (page) of 16 registers by a private memory extension unit, the block in which the addressed register is located (current register block) is specified by the contents of the Rp-register. This register is part of the program status doubleword and is loaded by program control. Registers 1 through 7 of the current register block in private memory may be used as index registers. The index registers are addressed by the X field in the instruction at the time the instruction is in the D-register.

Private memory may also be addressed by the P-register. If an instruction produces an effective address in the range of 'X'0 through 'X'F, the four low order bits of the reference address are used to address the register in the current register block of private memory which corresponds to the address. The private memory register may be used as the

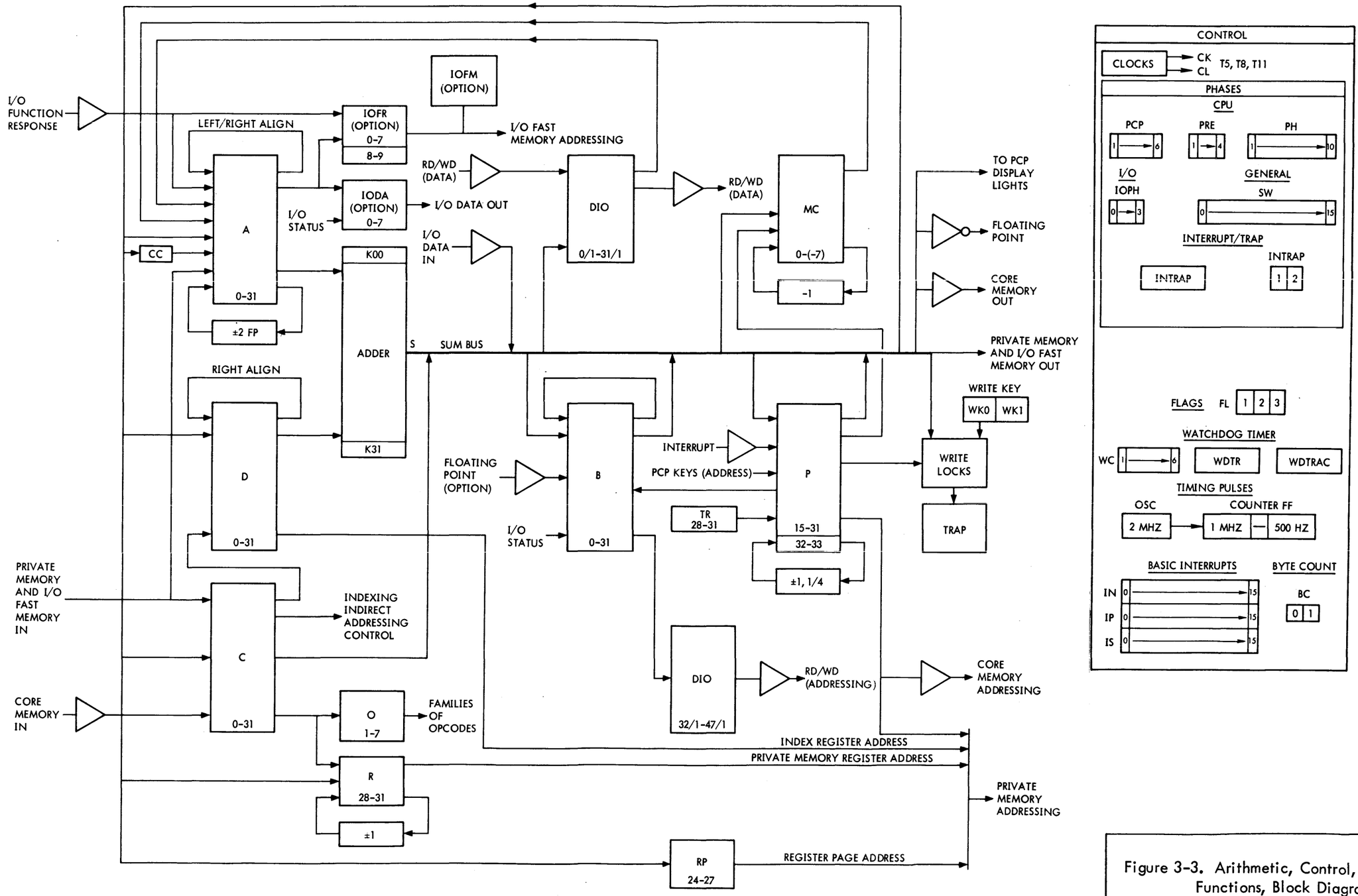


Figure 3-3. Arithmetic, Control, and Address Functions, Block Diagram

901172A.32

source of an operand, the location of a direct address, or the destination of a result. In this case core memory is not affected.

CORE MEMORY ADDRESSING. Core memory is addressed by the effective address in the P-register. The effective address is the final address produced for an instruction. With direct addressing the reference address of the instruction is the effective address. With indirect addressing, the initial reference address in the instruction corresponds to a location in core memory or private memory which contains an address value. This address value is accessed and transferred to the P-register where it becomes the effective address. When this occurs, the initial reference address is not lost but is temporarily stored in the B-register to be later updated and used in addressing the next instruction in the program sequence.

When an instruction specifies indexing with direct addressing, the effective address is produced by adding the contents of the index register to the reference address in the instruction. This function is performed by the adder, the A-register (containing the index value), and the D-register (containing the reference address). Index alignment is performed for byte, halfword, word, doubleword, and shift operations, and is a function which varies the effective length of the P-register to change the effective address displacement value. Index alignment is described in the Sigma 5 Reference Manual under Address Modification.

An instruction may specify both indexing and indirect addressing. In this case, the effective address is produced by adding the contents of the index register to the contents of the memory location corresponding to the initial reference address. Indexing occurs after the indirect location is accessed. Therefore, the initial reference address is not modified.

3-5 CPU Timing

Basic CPU timing for instruction execution is controlled by ac clock pulses having variable time intervals. The clock pulses are generated by the CPU clock generator. Phase control flip-flops toggled by the variable clock pulses determine the phase of the instruction being performed. Only one phase control flip-flop is set at any time. There are four preparation phases (PRE1 through PRE4) and ten execution phases (PH1 through PH10). In the preparation phases the functions common to most instructions are performed. In the execution phases the functions to complete the instruction are performed. In general, phases progress in numerical sequence, but a phase can be repeated or skipped depending on the instruction requirements. Most instructions require only a few phases. All instructions require at least two preparation phases (PRE1, and PRE3 or PRE4) and two execution phases (PH1 and PH10).

In a typical instruction, conditions are set during phase PH10 of the present instruction to read the next instruction into the C-register, and from there to transfer the instruction to the D-register, and the operation code and R field to the O- and R-registers, respectively, and to update the program address in the P-register. These functions are executed at the trailing edge of the next clock when phase PH10 ends and PRE1 begins. The phase which follows PRE1 may be any one of the other preparation phases depending on the instruction format. Typically, during preparation phase PRE1 the operation code is decoded and conditions are set to transfer the reference address from the D-register to the P-register. Phase PRE2 is used to compute the effective address and may require two clock times. Phase PRE3 is used to fetch the operand from core memory or private memory, and PRE4 is used for halfword or byte alignment and sign extension. PRE4 may require four clock times. At the end of the last preparation phase (PRE3 or PRE4) on the trailing edge of the clock, execution phase PH1 begins. When the instruction is nearly completed the phase sequence branches to PH10 to accomplish the final operations of the instruction and the normal end functions common to most instructions. The conditions are set to read the next instruction into the C-register. The process is then repeated for that instruction.

For operations other than those involved in preparation and execution of instructions, the CPU also has six phase control flip-flops (PCP1 through PCP6) for operating in the processor control panel mode, two flip-flops (INTRAP1 and 2) for interrupt trap mode, and four flip-flops (IOPH1 through IOPH4) for the input-output mode. The IOPH flip-flops operate in conjunction with sixteen general-purpose switch phase flip-flops (SW0 through SW15). Phases PCP, INTRAP, and IOPH operate in their respective modes in a way similar to the PRE and PH phases.

CPU CLOCK GENERATOR. Three tapped delay lines make up the CPU clock generator (see figure 3-4). The generator produces ac clocks for triggering ac flip-flops in the CPU and the floating-point option, ac clocks for triggering private memory, and dc clocks to trigger the C-register buffer flip-flops. A clock pulse is initiated each time delay line 1 is enabled. The pulse is tapped off at fixed intervals to form the required clock pulses. One of the tapped pulses is applied to the ac clock gates which generate an ac clock unless inhibited by a disabling function. The disable function shown in the simplified block diagram can be either a high disabling signal or the lack of a high enabling signal. In general, the ac clock gates provide a means of inhibiting a clock pulse until a certain time. An example is when a memory request has been generated but the data has not been released from core memory. Until the data is released, all clocks are inhibited. When data is released from memory, a data release function again enables the clock.

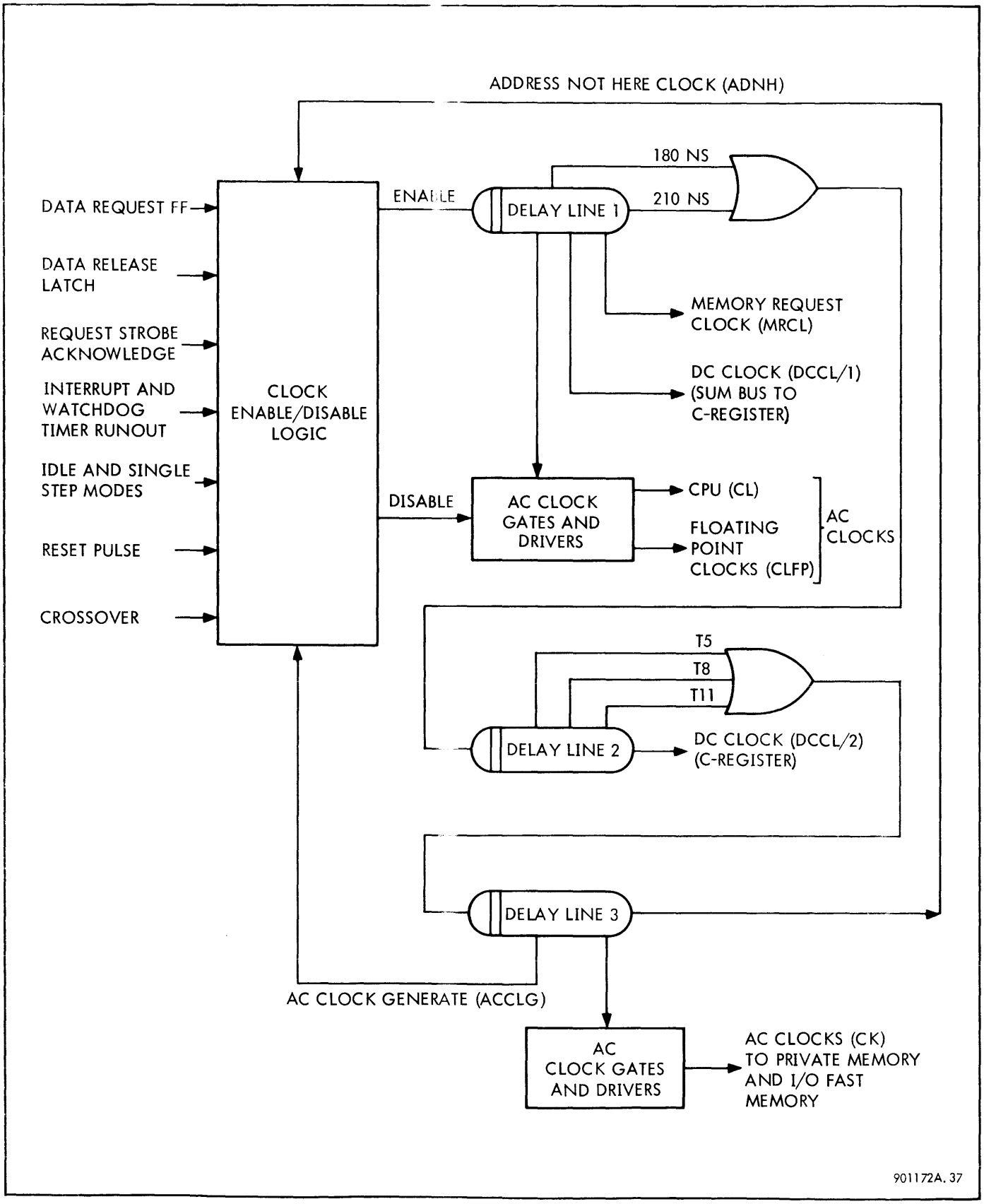


Figure 3-4. CPU Clock Generator, Simplified Block Diagram

Clock pulses CL and CK are outputs of the AC clock drivers. These clocks are 40 nsec and 50 nsec pulses, respectively, and recur at variable time intervals. The time intervals, designated T5, T8, and T11, are established by enabling or disabling delay line sensors associated with delay line 2. Nominal intervals for T5, T8, and T11 are 280, 380, and 500 nsec, respectively. Another variable interval occurs with the clock that initiates delay line 2. This clock is selected from either the 180 nsec or the 210 nsec tap of delay line 1. The tap selected depends on the status of the data request flip-flop. The 180 nsec tap is selected if the flip-flop is reset and the 210 nsec tap is selected with the flip-flop set.

If not inhibited by the clock enable/disable logic, delay line 1 is reinitiated by clock ACCLG (AC Clock Gate) from delay line 3 each time the clock reaches that point in the cycle. One of the conditions which inhibits an ac clock to the CPU is crossover. Crossover exists when private memory is addressed by the P-register, that is, when the effective address is in the range of 'X'0 through 'X'F. When this occurs, the private memory clock is generated as usual but the CPU ac clock is inhibited. The other functions which affect the clock enable/disable logic shown in figure 3-4 are described in the detailed principles of operation. Also shown in figure 3-4 is the address-not-here clock (ADNH) taken from delay line 3. This clock ensures that delay line 1 is enabled again in case a nonexistent location is addressed in core memory. If such a location is addressed, the memory request inhibits delay line 1 and the lack of a data release keeps the delay line inhibited. In that case, the address-not-here clock enables delay line 1 and sets the trap condition.

OSCILLATOR CLOCK GENERATOR. The oscillator clock generator consists of a 2-MHz sine wave oscillator followed by a frequency divider with seven flip-flops (see figure 3-5). Clocks of 1 MHz and 16 kHz are taken from the frequency divider. The 1 MHz clock steps the watchdog timer; is supplied to the input-output processors where it is routed to the device controllers; is fed to the CLOCK MODE switch on the PCP for single step operations; and is the source for the interrupt gate clocks which trigger the interrupt control flip-flops. The 16-kHz clock from the frequency divider supplies the time base selector which produces clock pulses for the real-time counter interrupts.

WATCHDOG TIMER. The watchdog timer ensures that the program periodically reaches interruptible points during instruction execution. The timer is a 6-bit counter triggered by the 1-MHz clock from the oscillator clock generator (see figure 3-5). The counter starts at the end of every instruction and at interruptible points in long instructions. The watchdog timer initiates the trap circuits if the count reaches 42 ms before another interruptible point or the end of an instruction occurs in the program sequence.

REAL-TIME CLOCK. The real-time clock, of which there are two standard levels and two optional levels, consists of a fixed interrupt routine preset to trigger at a frequency determined by the time base selector. Frequencies of 8 kHz, 4kHz, 2kHz, and 500 Hz are available from the time base selector. External frequencies and a 60-Hz line frequency may also be connected to control a real-time clock. In a typical application, when a real-time clock interrupt level is triggered, a fixed location in memory is accessed and the value contained in the location is decremented and restored to the fixed location. When the value becomes zero, the corresponding counter-equals-zero interrupt level is triggered. The counter-equals-zero interrupt level is associated with another interrupt routine at the discretion of the programmer.

3-6 Interrupt/Trap Functions

Interrupts and traps cause the normal program sequence to be interrupted. In general, interrupts allow the current instruction to be completed before entering the interrupt sequence and provide for returning to the interrupted point in the program to resume normal program operation after the interrupt is cleared. Traps cause the immediate execution of an instruction in a unique location in memory without necessarily allowing the current instruction to be completed. Traps are usually caused by program errors. A summary of the interrupts and traps is described in the SDS Sigma 5 Computer Reference Manual under Interrupt System and Trap System, respectively.

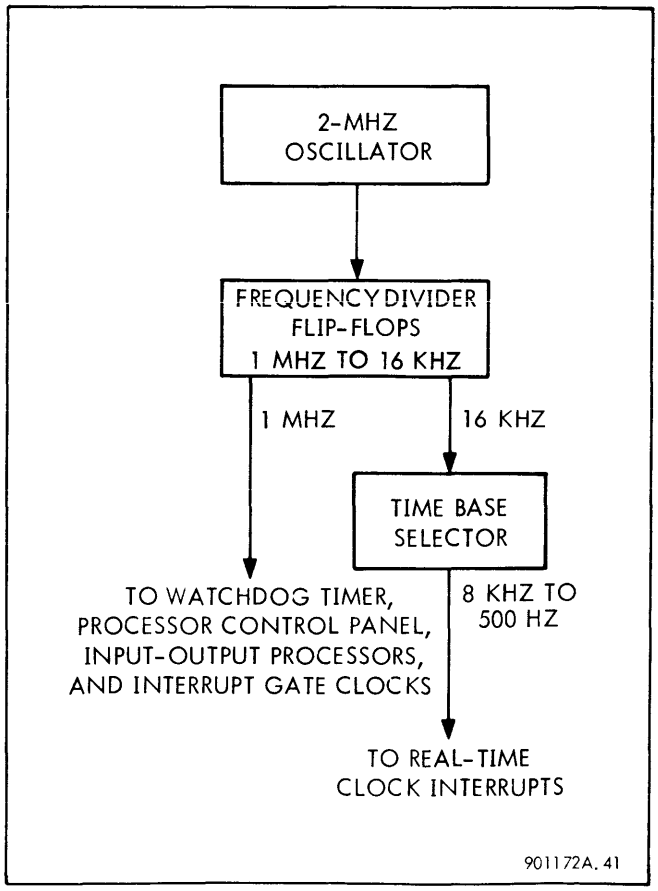


Figure 3-5. Oscillator Clock Generator, Simplified Block Diagram

INTERRUPTS. Each interrupt has an assigned priority determined by its position in a priority chain. In general, external interrupts have lower priority than internal levels. A level may be in one of six states depending on the condition of three control flip-flops assigned to each level. These states include armed, enabled, disarmed, disabled, waiting or active. When a level advances to the active state, the program branches to a memory address assigned to the interrupt and the instruction in that address is executed. The interrupt location may contain a single instruction (as in the real-time clocks) or the instruction may take the program to an interrupt subroutine. Interrupt operations are controlled by phase flip-flops INTRAP1 and INTRAP2. The phase flip-flops are clocked by the CPU ac clocks.

TRAPS. A trap is indicated by such conditions as nonexistent instructions, addressing a nonexistent memory location, watchdog timer runout, or an instruction calling for operation of an option when the option is not included in the equipment. As in the interrupts, each trap is associated with an instruction stored in a location assigned to the trap. When a trap condition is detected, the trap state is set, causing phases INTRAP1 and INTRAP2 to be entered. The current instruction may or may not be carried to completion, but in either case the instruction is terminated by the trap sequence. During the trap sequence, the instruction address of the current program status doubleword (which had already been incremented) is decremented and the instruction in the location associated with the trap is executed. The instruction in the trap location is an exchange program status doubleword (XPSD).

POWER FAIL-SAFE. The power fail-safe option includes a power monitor and two levels of interrupts. The power-on level (00) and the power-off level (01) have the highest priority in the interrupt chain. They are always armed and enabled while power is operating in the normal range. If the power monitor detects a power loss below a preset threshold, the monitor generates a power-off request signal which activates the power-off interrupt. The interrupt waits until the current instruction is completed. If the power-off request occurs during a service call and the service call had interrupted an instruction, then both the service call and the interrupted instruction are completed before the CPU services the power-off routine. The CPU has approximately 5 milliseconds after the power-off request goes true to complete the current operations, to store all the volatile information into core memory, and to shut down the computer. When power is restored to a level above the threshold, the CPU is initiated and a recovery subroutine associated with the power-on interrupt is executed. The CPU is returned to the state it was in before power failure.

3-7 Private Memory Organization

The standard private memory (CPU fast memory) in the Sigma 5 contains one block of 16 general registers. Each register has 32 bits. The term private implies that the registers may only be accessed by the CPU and by no other equipment. Optional register extension units may be added to the standard block to enlarge private memory. Each register extension unit contains a block of 16 registers. A total of 16 blocks, including the standard block, may be contained in a Sigma 5.

Registers in any block are addressed X'0' through X'F'. The block of registers currently available to a program is called the current register block. Register 0 in the current register block is used for special applications by the CPU. For example, during input-output operations the address of the first command doubleword in a sequence is obtained from register X'0'. Registers X'1' through X'7' are used in indexing operations and all the registers in a block may be used as accumulators (fixed point and floating point) and to hold control information.

3-8 Processor Control Panel

The PCP displays the states of selected registers in the central processor and provides switch-controlled signals for manual computer operation. The upper section of the panel is reserved for maintenance personnel, the lower section for the computer operator.

Most switches on the PCP are inhibited while in the run mode. When any control switch is operated while in the idle mode a phase sequence (PCP phases) similar to the CPU phases is entered. The PCP phases are controlled by six flip-flops, PCP1 through PCP6. The phases have uniform length. Placing the COMPUTE switch to IDLE places the PCP logic in phase PCP2. Placing the COMPUTE switch to RUN or STEP takes the PCP from the idle phase to PCP3, from which the CPU branches to PH10 of the current instruction. The preparation phases follow PH10 to execute the instruction.

3-9 Floating Point Unit

The floating point optional unit provides the CPU with floating point arithmetic capability. The unit is controlled by the floating point clocks generated by the CPU delay line clock generator. During floating point operations the unit is loaded from the CPU sum bus and the operation is performed by the registers and adder in the unit. The registers are expanded to accommodate both long and short number formats. After the operation is completed, the number is returned through the CPU B-register. The internal functions of the floating point unit are described in the detailed principles of operation.

3-10 Memory Protection

The memory protection option in the CPU consists of one 2-bit write-lock register for each 512-word block of core memory and one 2-bit write key. The write key is contained in bits 34 and 35 of the program status doubleword. The write locks and write keys allow access to core memory locations to be program controlled. The write lock codes

are first written into memory as a lock control image, 16 codes to a memory word. The lock control image is transferred to the write lock registers by a move-to-memory-control instruction. During memory access, the write lock codes are compared with the two write key bits in the program status doubleword to determine if the addressed block of memory can be accessed. Access control bit configurations are described in section II of this manual.

3-11 CORE MEMORY

The maximum core memory storage is 128K, comprising eight memory blocks, each containing 16K. A memory block may contain 4K, 8K, 12K, or 16K by adding optional 4K memory expansion kits. A minimum 4K block is standard with each computer. Each memory block is organized in stacks, core diode modules, bytes, and bit planes (see figure 3-6).

A 4K memory is called a stack; it comprises four core diode modules. Each stack has a capacity of 4096 words of 32 bits plus a parity bit. One byte in each of the 4096 words is held in a core diode module, hence, each word embraces all four modules in the stack. The cores on a module are arranged in matrices, 32 by 128 cores, called memory bit

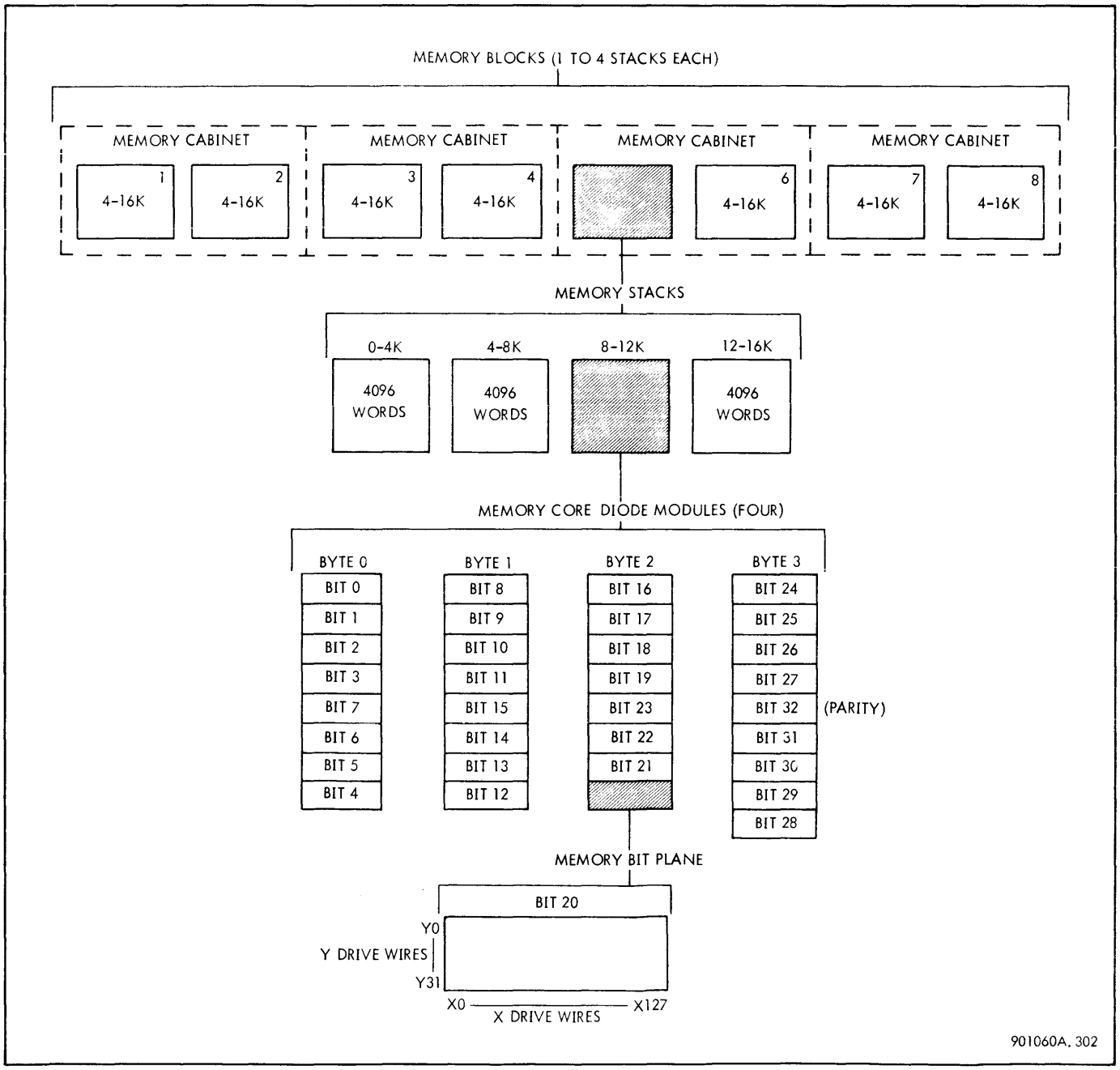


Figure 3-6. Core Memory Organization

planes. Each core in a bit plane corresponds to one bit in each of the 4096 words in the stack. Modules for bytes 0, 1, and 2 contain eight bit planes. The module for byte 3 contains nine bit planes to include the parity bit.

3-12 Port Expansion

Memory blocks are connected in parallel to the CPU (see figure 3-7). Each memory block has a standard port designated as port C.

A port is a section of memory logic that controls entry priority during memory access. Port C is always connected to the controlling CPU and is sufficient in systems where the only input-output processor is an integral IOP in the CPU. For each external IOP or CPU connected to a memory block an optional port is added. The first additional port is port B. It is commonly called a one-to-two port expander and provides a second access path. The next port added is port A and is commonly called a two-to-three port expander. Port A provides a third access path and has the highest priority. Port C has the lowest priority. For maximum port expansion on any memory block, a three-to-six port expander may be added to either port A or port B. The three-to-six port expander has four additional ports providing a total of six access paths when connected. The additional ports are numbered 1 through 4. Port 1 of the expander has the highest priority and port 4 the lowest.

3-13 Three-Wire Core Selection

The Sigma 5 combines the three-dimensional coincident current core selection method with the two-dimensional linear core selection method. This combination is commonly known as the 2-1/2 D system. The 2-1/2 D system has a coincident current read cycle and a linear select write cycle. Three wires are threaded through each core: an X wire (word wire), a Y wire, and a sense wire. No inhibit winding is present.

On each bit plane there are 128 X wires. Each X wire also threads all other bit planes on a core module. A bit plane contains 16 Y wires which are separate for each bit plane. Each Y wire doubles back through a second row of cores to provide 32 Y wires in all. Typical X and Y wiring for two cores in each of two bit planes on a memory module is shown in figure 3-8. A sense wire threads through all of the cores in one bit plane. Since each Y wire passes through two rows of cores there are two core intersections for each combination of X and Y wires. For a given direction, currents add in the core at one intersection and cancel in the other. Hence, core selection is determined by current direction as well as wire location.

To write ones, half current is passed through one word wire; half current is also passed through the selected Y wire to affect one core out of 4,096. The two half currents add at

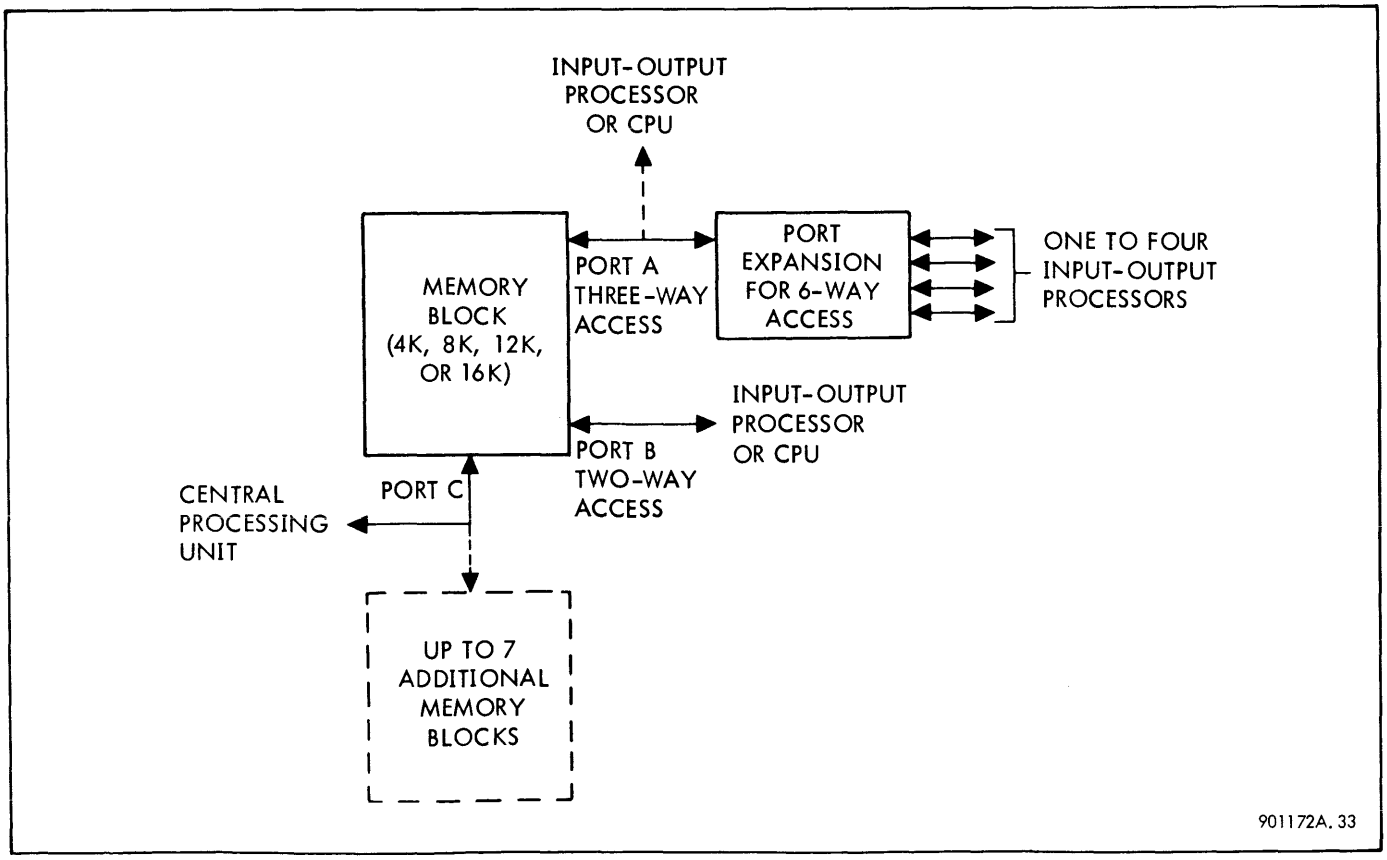


Figure 3-7. Memory Connections and Port Expansion

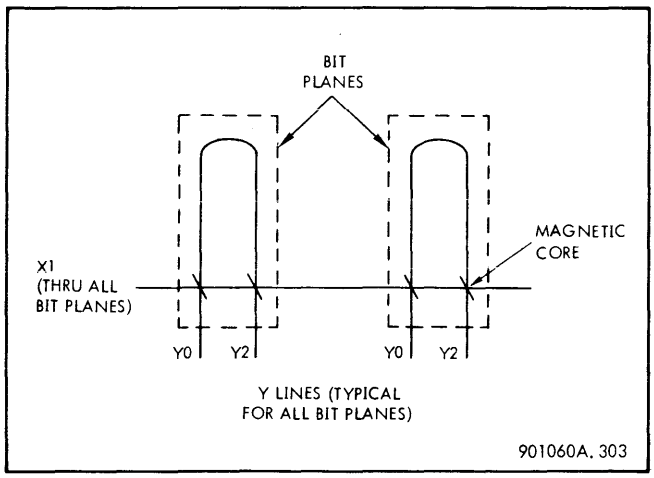


Figure 3-8. Typical X and Y Core Wiring

the intersection and force the core to the one state. To write zeros, the Y current is inhibited on the bit planes where zero bits are to be written. This method is similar to the linear select method in that the digit current is added to, rather than subtracted from, the word current.

To read, half current is passed through the appropriate X wire, half current is also passed through the same Y wire on all bit planes. All cores in the selected location are forced to zero, and the sense wires detect current from the bit planes that contained ones in the selected location.

3-14 Memory Input-Output

Data is interchanged between core memory and the CPU or IOP on a 32-bit bidirectional memory bus. Each memory block contains control logic, port priority logic, and core selection logic to control information flow within the block. Two latch registers are provided: one to hold location addresses (L-register) and the other to handle data entering and leaving memory (M-register). Data entering memory is gated from the CPU sum bus or IOP memory bus onto the core memory bus and loaded into the M-register. Data leaving memory is loaded into the M-register from sense amplifiers and is transmitted on the memory bus to the CPU C-register or IOP M-register. Addresses entering a core memory block may be modified by interleave logic before loading the L-register to address the cores.

MEMORY TIMING. Two delay lines in each memory block control timing: one controls the read cycles, the other controls the write cycles. The delay lines provide pulses at 20 nsec intervals. Memory access occurs in three modes: read-restore, full clear write, and partial clear write. Regardless of the mode, a read and a write cycle are required for each memory access. Every read cycle must be followed by a write cycle to replace the information in the same memory location. A write operation must be preceded by a read cycle to clear the location for storage.

In the read-restore mode a memory request signal sends a pulse down the read delay line. Outputs from the delay line taps provide timing signals to energize the X and Y drive lines, enable the register latches and strobe data into the M-register. Parity is checked in this mode. In the full clear write mode, a read cycle is executed to clear the location, but the read data is not gated into the M-register and is lost. During the partial clear write mode, the data from the read cycle is gated into the M-register and parity is checked. One, two, or three new bytes are inserted into the word and new parity is generated before the word is written into memory.

To execute a write cycle for the read-restore and full clear write modes, an output from the read delay line starts a pulse down the write delay line. Outputs from the write delay line energize the X and Y drive lines in the opposite direction from that in the read cycle, and inhibit the Y lines in bit planes where zeros are to be stored. Zeros are present in all bit positions of a word following a destructive read operation and remain in bit positions where writing a one is inhibited. Odd parity is checked in the full clear write mode, setting the parity error flip-flop if the M-register contains an even amount of ones. Timing for the write cycle in the partial clear write mode is the same as that for the read-restore and full clear write modes except that energizing the drive lines is delayed long enough to set byte indicators and route the information into the addressed byte locations.

INTERLEAVING. Memory access speed can be increased by overlapping the second cycle of one access with the first cycle of the next access. An example of interleave timing in a read-restore mode is shown in figure 3-9. The interleave method requires that successive words be stored in different memory blocks because in addressing the same memory block successively both the read and write cycles must be completed before another access is started. As an example of interleaving, consider two 4K memory blocks and a program that calls for storing data in sequential memory locations. The first word is stored in one of the blocks, the second word is stored in the other block in the same numbered location as the first, and the third word is again stored in the first block. In larger memories and different clock sizes, interleaving becomes more complex, but two successive words are never stored in the same block. Interleaving is performed by transforming certain bits in the address before entering the recognition logic of the port. Four switches on switch modules, and starting-address switches on the ports, are provided for interleave setup.

3-15 INPUT-OUTPUT CHANNEL

An input-output channel consists of an input-output processor (IOP) connected to one or more device controllers, each controlling one or more peripheral devices. The IOP controls data exchange between core memory and the device controllers. This discussion describes the three types of IOP's which a Sigma 5 system may contain: multiplexing (MIOP), selector (SIOP), and internal (integral) IOP. Device controllers and devices are not included in this discussion since their arrangements are unique to each system.

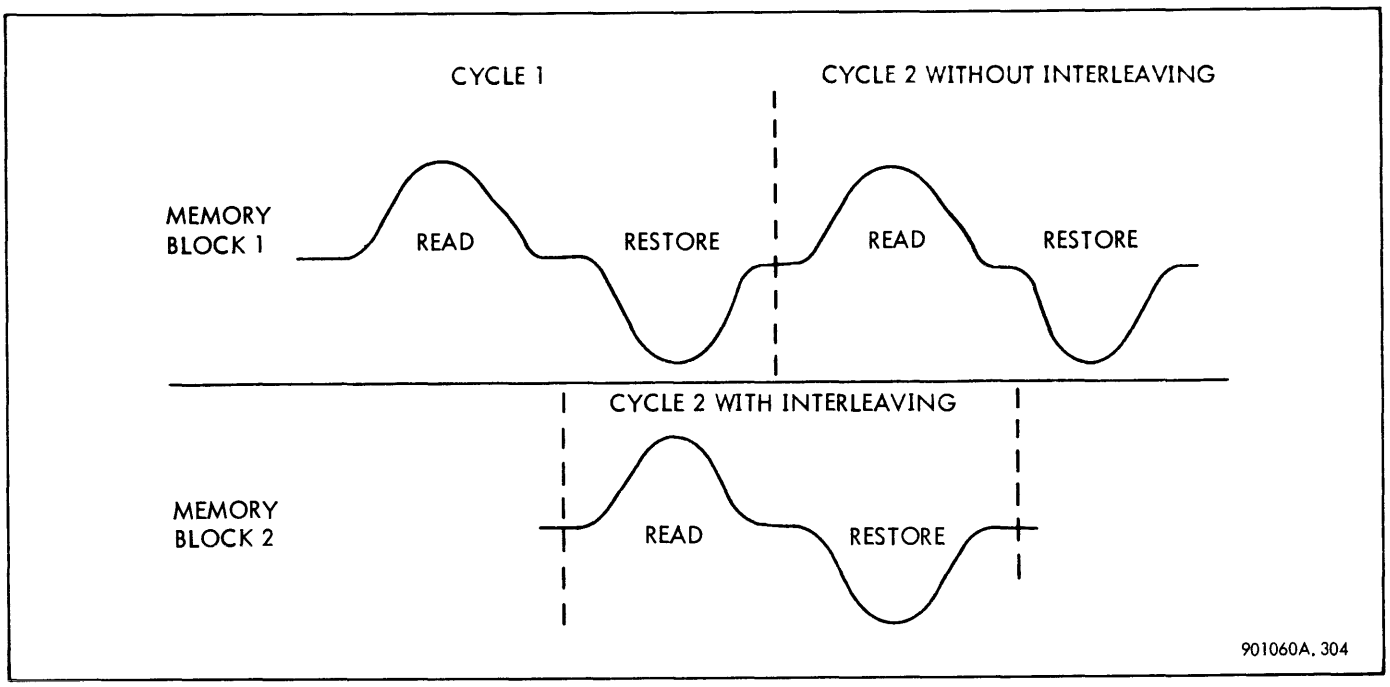


Figure 3-9. Example of Interleaving in Read-Restore Mode

Multiplexing and selector IOP's are external to the CPU and each is connected to one port in core memory by a single memory bus. This allows the I/O channels to communicate with core memory simultaneously with the CPU. The integral IOP is internal to the CPU and shares the CPU memory bus. Therefore, either the CPU or the integral IOP, but not both, may communicate with core memory at any time.

Once started by the CPU, the external IOP's operate independently in transferring data between device controllers and core memory. Data is transferred in words (four bytes at a time) between the IOP and core memory. Between MIOP's and device controllers, transfers are made a byte at a time up to four bytes per service cycle. Then a new order is executed. Between SIOP's and the device controllers transfers are made in bytes, halfwords, and words continuously until the specified number of bytes has been transferred without disconnecting and reconnecting the device for each byte or word.

Command doublewords stored in memory by the CPU before an I/O operation are used as instructions by the IOP. The doublewords contain an IOP order, byte address, flags, and byte count. An IOP order designates the operation to be performed such as read, write, and read backward; the byte address is the address of the next byte location in core memory where data is to be read or stored; the flags designate how the operation is to be handled (e.g., data chaining, command chaining); and the byte count is the number of bytes remaining to be transferred. The IOP's have four operating states: order out, data out, data in, and order in. These are defined as follows:

Order Out. During order out, the IOP accesses a command doubleword from memory, stores the doubleword in fast access memory except for the order, sends the order to the device controller, and terminates the operation.

Data Out. During data out, the IOP accesses the memory location determined by the current byte address and transmits the data from that location to the device controller. The IOP decrements the byte count to reflect the number of bytes remaining to be transferred and adjusts the byte address to access the next byte location. When the byte count is reduced to zero the IOP accesses another command doubleword and, if data chaining or command chaining is specified by either chaining flag, continues to transfer data. Otherwise, the data transfer is terminated.

Data In. During data in, the IOP transmits data from the device controller to core memory by accessing the memory locations where the data is to be stored. The byte count and byte address are decremented with each byte. When the byte count is reduced to zero, the IOP accesses the next command doubleword only if data chaining or command chaining is specified by either of the chaining flags. Otherwise the data transfer is terminated.

Order In. During order in, the device controller transmits the operational status of the device to the IOP and then terminates the operation. An order in is always followed by a terminal order. Terminal orders are sent from the IOP to the device controller to transfer control information when any one of four conditions occur: count done, command chaining, IOP halt, and interrupt-on-channel-end.

3-16 Multiplexing IOP

The principal elements contained in the MIOP include a data register, address registers, fast access memory, adder, input and output registers, timing delay lines, and a function register (see figure 3-10). Timing and some control functions are not shown. The CPU communicates with the MIOP on three IOP address lines, three function code lines, and two condition code lines. The IOP address code designates one of eight possible MIOP's, the function code designates the operation to be performed (SIO, HIO, TIO, TDV, or AIO), and the condition code informs the CPU whether the IOP address or interrupt has been recognized. All other communication between the CPU and the MIOP is through locations X'20' and X'21' in core memory. For example, during an SIO instruction the CPU supplies the IOP with the address of the first command doubleword, the address of the device controller, and the device number through locations X'20' and X'21'. These locations are also used to transmit response information and device status to the CPU.

The fast access memory in the MIOP contains 32 subchannels, one for each possible device controller. Stored in each subchannel is the device controller number to which the subchannel is assigned. Each subchannel has an 80-bit capacity contained in six registers. Multiplexing occurs on a subchannel level and therefore on a device controller level. Devices connected to the same device controller are not multiplexed. A new start instruction is required to access two devices consecutively on the same device controller.

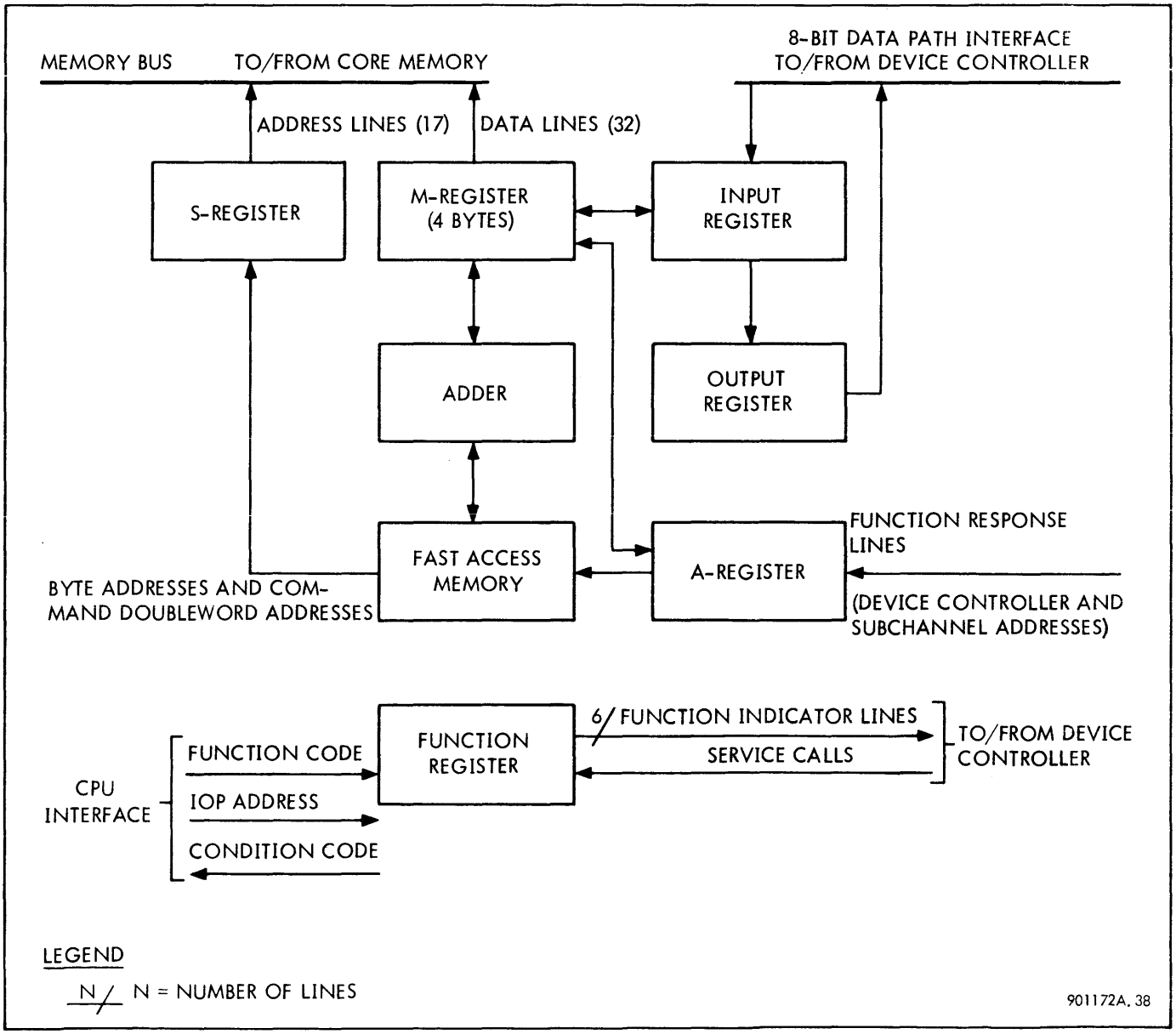


Figure 3-10. Multiplexing IOP, Simplified Block Diagram

The CPU starts an input-output operation by executing an SIO instruction. In a typical operation during an SIO, the address of the first command doubleword in core memory, the device controller address, and the device number are sent to locations X'20' and X'21' where they are accessed by the addressed MIOP. At the same time, the MIOP is addressed by the CPU and the function code is sent to the device controller. The condition codes respond to the CPU and indicate whether the device controller address is recognized, busy, or not recognized. If recognized, the device controller responds with device status on the function response lines. The status is stored in either location X'20' or X'21' or both so that it is available to the CPU. If ready, the device controller directs a service call to the MIOP and if no higher priority service call is pending, an order out service cycle is entered.

During an order out, the MIOP accesses the first command doubleword which is loaded into the M-register. The order is sent to the device controller while the remaining portion, containing the byte address, byte count, and flags, is loaded into the assigned subchannel in fast access memory. The order out is followed by either a data out or data in service cycle as specified by the order. From one to four bytes are transferred during each succeeding service cycle depending on the capabilities of the device and the conditions specified by the command. The byte count and byte address are decremented by the adder for each byte transferred. A service call is generated for each service cycle (after a maximum of four bytes are transferred). This allows a higher priority device controller to interrupt for service. Logically, the device controller is disconnected at the end of each service cycle and is reconnected after the MIOP acknowledges the new service call.

When the byte count has reached zero, the operation is terminated by an order in service cycle and a terminal order if neither command chaining nor data chaining flags specify chaining. If chaining is specified, the MIOP accesses the next command doubleword in sequence and continues the operation. When all data has been transferred, the I/O operation is ended with the order in and terminal order.

3-17 Selector IOP

The principal elements contained in the SIOP include a data register, memory address register (S), data buffer, register for counters and flags, input-output register, function register, and timing delay lines (see figure 3-11). The timing delay lines and some control functions are not shown. Since the SIOP is designed for high speed input-output devices such as RAD files and high speed tape stations, it only services one channel at a time and continues the data transfer without connecting and disconnecting the device controller as in multiplexing operations. The equivalent of one fast access memory subchannel is provided to store the byte count, byte address, and flags. The data buffer allows for memory port interference, provides delays in the IOP data path, assembles and disassembles data, decrements the byte count and byte address counter, and receives function response and status from the device controller.

CPU interface and core memory interface to the SIOP are the same as those for the MIOP. The SIOP is similarly addressed by the CPU, and communication between the CPU, core memory locations X'20' and X'21', and the SIOP are also similar. The SIOP may be equipped with an optional bus-sharing feature which allows the SIOP to time-share a core memory bus with another SIOP equipped with a similar bus-sharing feature.

Interface between the device controller and the SIOP may consist of 8, 16, or 32 bit data paths to transfer bytes, halfwords, or words, respectively. The SIOP responds to device controller service calls and performs order out, data out, data in, and order in functions. Once started, a data exchange continues until the entire record is transmitted, as indicated by a zero byte count or until the exchange is terminated by the device controller.

During the order out operation, the IOP accesses the command doubleword from core memory, sends the order to the device controller, stores the byte address, byte count and flags, and then terminates the order out. During the data out operation, the SIOP accesses core memory as determined by the byte address and loads the data into the data buffer. In response to device controller request strobes, the SIOP accesses the data buffer, aligns the data as required by the state of the byte address and byte count registers, generates odd parity for a one byte data path, and transmits the data to the device controller. When the byte count is reduced to zero, data chaining is performed if specified by the data chaining flag; otherwise the order out is terminated.

During a data in operation the SIOP responds to device controller requests and loads the data buffer. One byte odd parity checks are made if specified. The data buffer aligns the data according to the state of the byte count and byte address registers, accesses the core memory location designated by the current byte address, and controls partial or full write operations to core memory. The byte address is incremented if it is a forward operation and decremented if a backward operation. The byte count is decremented each time core memory is accessed. When the byte count is reduced to zero, the SIOP performs data chaining if specified by the data chaining flag; otherwise the order is terminated.

During order in, the SIOP accepts the operational status byte from the device controller in which any of the following conditions are reported: transmission error, incorrect length, chaining modifier, channel end, or unusual end. The SIOP responds to the conditions reported and then terminates the operation. The service sequence is terminated with a terminal order sent to the device controller. The terminal order may report any of the following: interrupt, count done, command chain, or IOP halt.

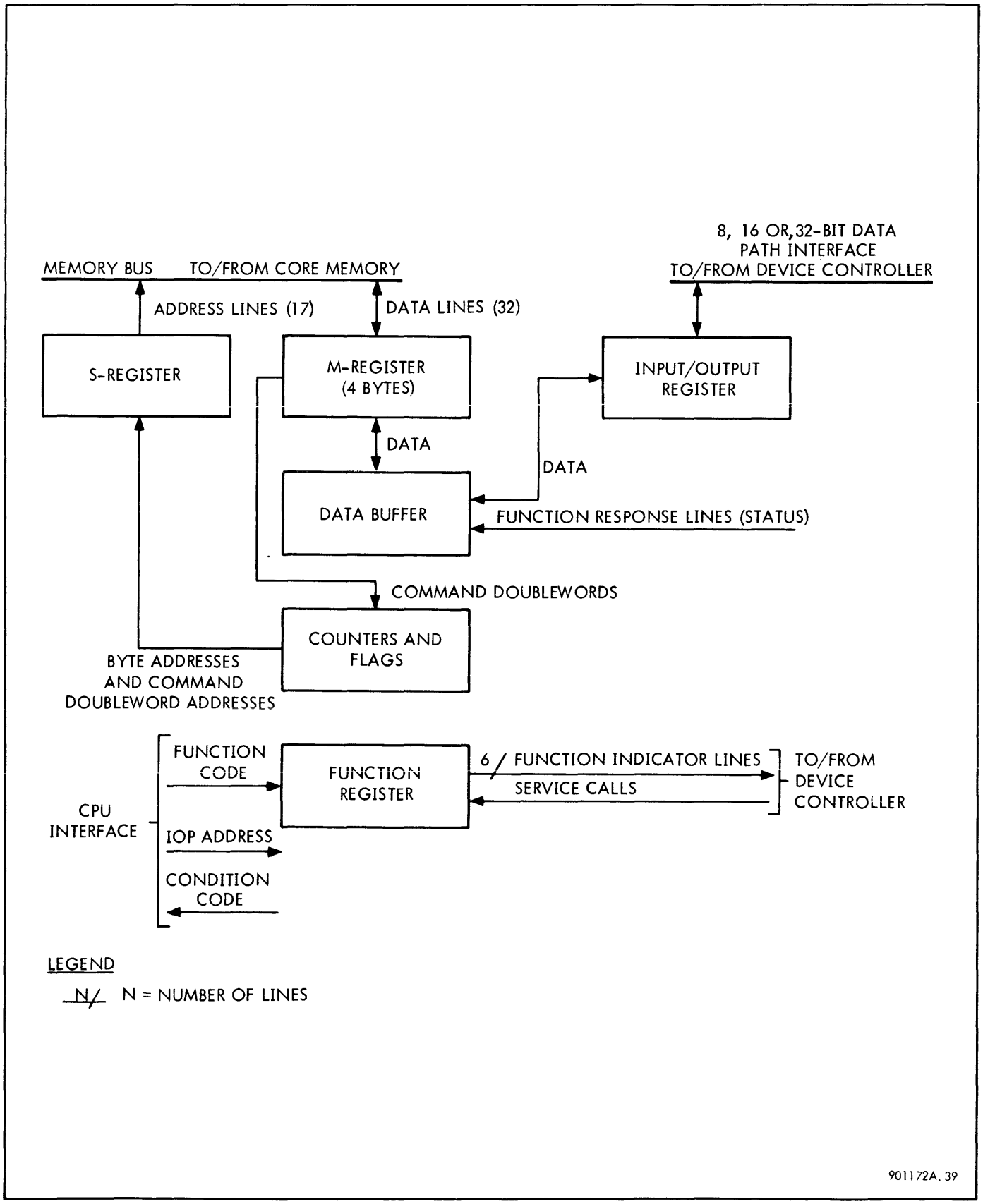


Figure 3-11. Selector IOP, Simplified Block Diagram

3-18 Integral IOP

The integral IOP is a multiplexing IOP which uses most of the CPU registers to perform I/O operations. A CPU equipped with an optional integral IOP contains additional registers IODA, IOFR, and IOFM, and a 32-channel fast access memory. The registers and their functions are shown on the CPU arithmetic, control, and address functions block diagram, figure 3-3. The fast access memory is not shown. The integral IOP responds to service calls from the device controllers and performs order out, data out, data in and order in operations in a manner similar to the MIOP. Timing is accomplished by the CPU clocks which control four input-output phase flip-flops and sixteen switch phase flip-flops. Data chaining and command chaining may also be performed.

3-19 Chaining

Chaining permits an IOP to execute two or more commands from memory for a single start instruction executed by the CPU. Command chaining is specified by setting the command chain flag in the command doubleword. Instead of terminating service when a command has been executed, the next command doubleword in sequence is read by the IOP. If the command chaining flag is also set in the new command doubleword, another command doubleword is read after the present one has been executed. Finally, when a command doubleword is accessed in which the command chaining flag is not set, the operation is terminated at the end of the current command doubleword.

Data chaining is specified by a data chaining flag in the command doubleword. Data chaining permits scatter reading and gather writing. Scatter reading is placing information from one physical record in a device into one or more noncontiguous memory locations. Gather writing takes information from one or more noncontiguous memory locations and writes it into one physical record in a device. When a data chain flag is detected, the IOP needs a command doubleword from the next successive memory location as in command chaining. but the order bits in the doubleword are not transmitted to the device controller. Thus, the operation called for in the previous order is continued without starting a new record. Data chaining stops when a zero is detected in the data chaining flag bit of the current command doubleword.

3-20 IOP Priority

IOP priority for external IOP's is established in relation to the CPU and in relation to core memory (see figure 3-12). In relation to the CPU, IOP's are connected in trunktail fashion. The IOP closest to the CPU has the highest priority; the one farthest from the CPU has the lowest. All of the IOP's share a single interrupt request line to the CPU. In relation to core memory, priority is determined by the memory port to which the IOP is connected. Port A has a higher priority than port B, and of the four port expander outputs, port 1 has the highest and port 4 the lowest priority.

3-21 DETAILED PRINCIPLES OF OPERATION

The detailed principles of operation describe the logical and nonlogical functions performed by each major equipment element. Detailed logical and circuit diagrams are used to develop the explanations of the logical functions. When a detail needs further clarification, a simplified diagram is included. Basic logic symbols used in the equipment documentation are defined in figure 3-13.

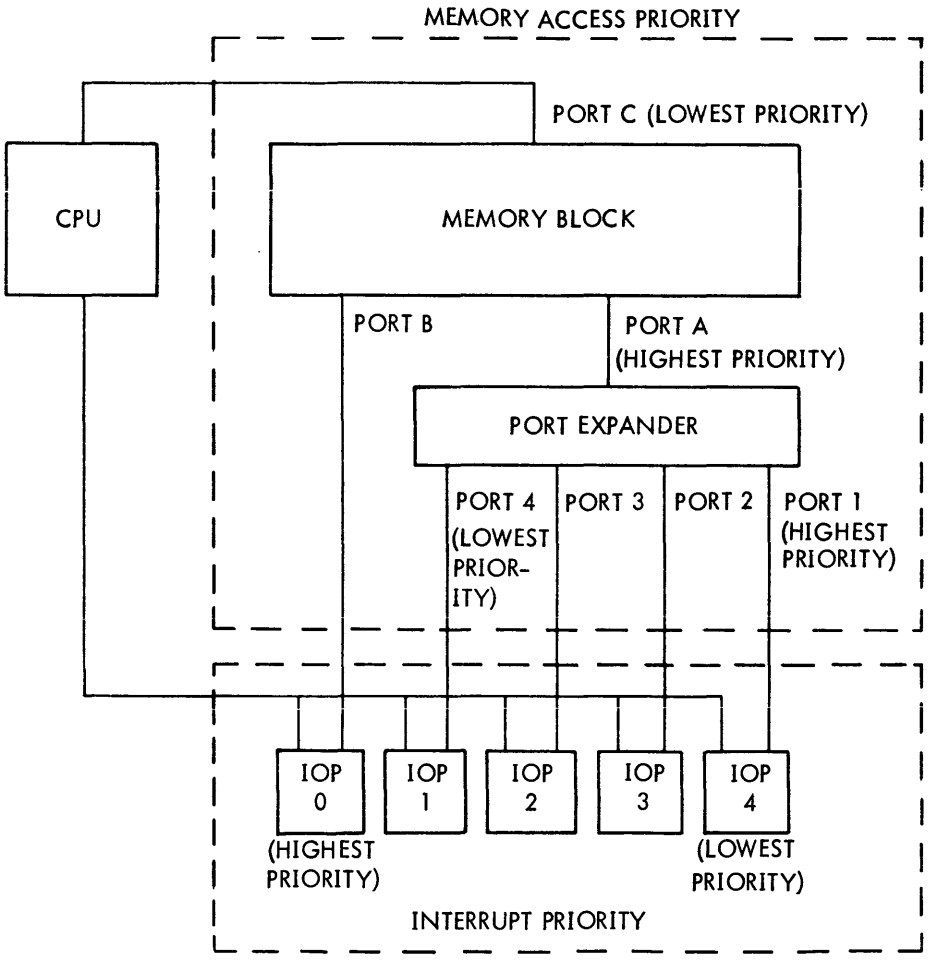
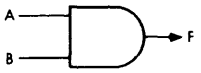
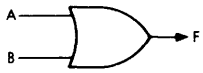
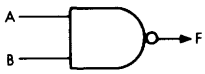
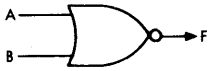
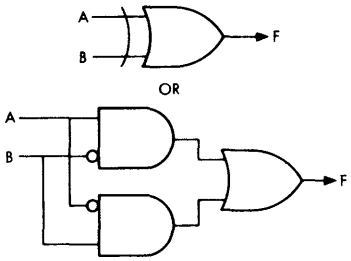
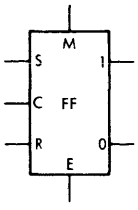
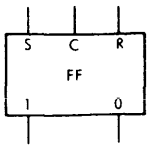
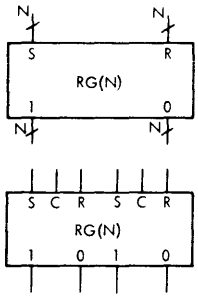
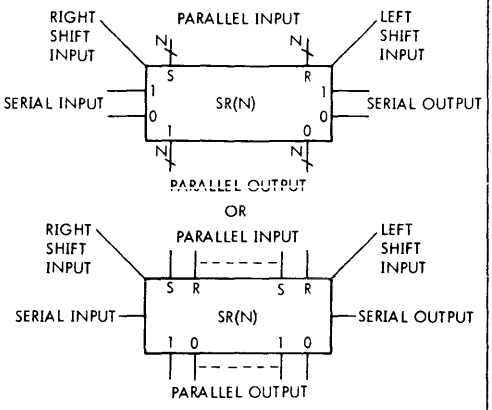
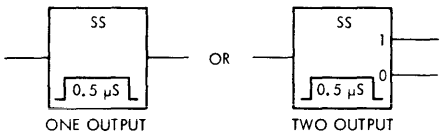
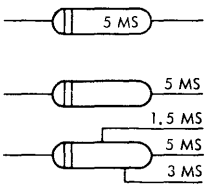
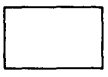
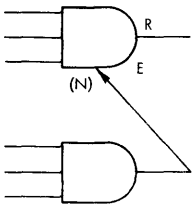


Figure 3-12. Typical IOP Priority Arrangement

LOGIC FUNCTION	SYMBOL	DESCRIPTION
AND		INPUT A B / OUTPUT F L L / L L H / L H L / L H H / H
OR		INPUT A B / OUTPUT F L L / L L H / H H L / H H H / H
STATE INDICATOR		INPUT CONDITION: A SMALL CIRCLE AT AN INPUT TO ANY ELEMENT (LOGICAL OR NONLOGICAL) INDICATES THAT THE RELATIVELY LOW (L) SIGNAL ACTIVATES THE FUNCTION. CONVERSELY, THE ABSENCE OF A SMALL CIRCLE INDICATES THAT THE RELATIVELY HIGH (H) SIGNAL ACTIVATES THE FUNCTION. OUTPUT CONDITION: A SMALL CIRCLE AT THE SYMBOL OUTPUT INDICATES THAT THE OUTPUT TERMINAL IS RELATIVELY LOW WHEN THE FUNCTION IS ACTIVATED.
NAND		INPUT A B / OUTPUT F L L / H L H / H H L / H H H / L OUTPUT LOW IF BOTH INPUTS HIGH
NOR		INPUT A B / OUTPUT F L L / H L H / L H L / L H H / L OUTPUT LOW IF ONE OR MORE INPUTS ARE HIGH
EXCLUSIVE OR		INPUT A B / OUTPUT F L L / L L H / H H L / H H H / L
GATED FLIP-FLOP		THE FLIP-FLOP ASSUMES THE 1 STATE WITH: A. INPUT S HIGH AND INPUT C CLOCKED OR B. INPUT M HIGH THE FLIP-FLOP ASSUMES THE 0 STATE WITH: A. INPUT R HIGH AND INPUT C CLOCKED OR B. INPUT E HIGH THE FLIP-FLOP ASSUMES THE 1 STATE IF BOTH INPUTS S AND R ARE HIGH AND INPUT C IS CLOCKED. THE FLIP-FLOP TOGGLES AT TRAILING EDGE OF CLOCK PULSE AND LEADING EDGE OF PULSE AT DIRECT INPUTS M AND E.
REPEATER FLIP-FLOP		THIS FLIP-FLOP ASSUMES THE 0 STATE WHEN THE S INPUT IS LOW AND THE C INPUT IS CLOCKED. THE FLIP-FLOP TOGGLES AT THE TRAILING EDGE OF THE CLOCK PULSE.

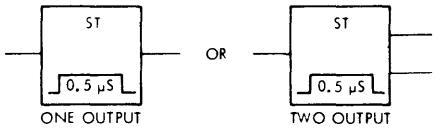
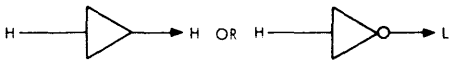
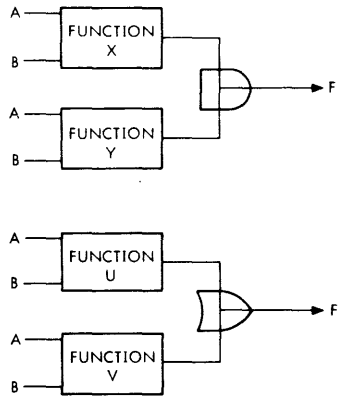
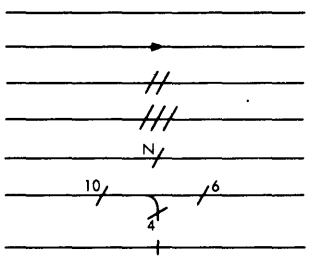
901172A.34/1

Figure 3-13. Basic Logic Symbols Chart (Sheet 1 of 3)

LOGIC FUNCTION	SYMBOL	DESCRIPTION
BINARY REGISTER		THE BINARY REGISTER SYMBOL REPRESENTS A GROUP OF FLIP-FLOPS USED AS A SINGLE REGISTER. THE LETTER N INDICATES THE NUMBER OF FLIP-FLOPS OR BITS IN THE REGISTER. IN SOME APPLICATIONS THE SYMBOL MAY BE SHOWN AS IN THE LOWER CONFIGURATION BUT IS USUALLY INDICATED AS IN THE UPPER DRAWING.
SHIFT REGISTER		THE SHIFT REGISTER SYMBOL REPRESENTS A BINARY REGISTER WITH PROVISIONS FOR DISPLACING OR SHIFTING THE CONTENTS OF THE REGISTER ONE STAGE AT A TIME TO THE RIGHT OR LEFT BY THE SHIFT INPUT. THE LETTER N INDICATES THE NUMBER OF FLIP-FLOPS OR BITS IN THE REGISTER. IN THE LOWER SYMBOL AT THE LEFT, THE NUMBER OF INPUTS AND OUTPUTS AGREE WITH THE NUMBER OF BITS (N). UNUSED INPUTS AND OUTPUTS ARE NOT SHOWN.
SINGLE SHOT		THE UNACTUATED STATE OF THE SINGLE SHOT IS EITHER ZERO OR ONE. WHEN ACTUATED, IT CHANGES TO THE OPPOSITE STATE AND REMAINS IN THAT STATE FOR THE DURATION OF THE ACTIVE TIME OF THE DEVICE. THE DURATION, AMPLITUDE, POLARITY AND SHAPE OF THE OUTPUT SIGNAL ARE DETERMINED BY THE CHARACTERISTICS OF THE SS AND NOT BY THE INPUT SIGNAL. A STYLIZED WAVEFORM MAY BE SHOWN INSIDE OR OUTSIDE OF THE SYMBOL TO INDICATE OUTPUT CHARACTERISTICS.
TIME DELAY		THE TIME DELAY DURATION IS SHOWN INSIDE OR OUTSIDE THE SYMBOL ADJACENT TO THE OUTPUT. TWO VERTICAL LINES IN THE SYMBOL INDICATE THE INPUT SIDE. IF THE DELAY DEVICE IS TAPPED, THE DELAY TIME RELATIVE TO THE INPUT IS SHOWN ADJACENT TO THE TAP OUTPUT.
GENERAL LOGIC FUNCTION		THE SYMBOL APPLIES TO FUNCTIONS NOT SPECIFIED ELSEWHERE. IT IS ADEQUATELY LABELED TO IDENTIFY THE FUNCTION PERFORMED
EXTENDED INPUTS		WHERE A CIRCUIT IS USED TO ADD INPUTS TO ANOTHER AND OR ANOTHER OR CIRCUIT, AND THE CONNECTION FROM THE SECOND CIRCUIT TO THE FIRST IS MADE AT OTHER THAN A NORMAL INPUT OR OUTPUT OF THE FIRST CIRCUIT, THE CONNECTION IS INDICATED AS SHOWN IN THE SYMBOL. THE LETTER E INDICATES EXTENSION AND THE LETTER R, WHEN SHOWN ADJACENT TO THE SYMBOL, INDICATES THAT THE OUTPUT REGISTER IS ADJACENT TO, OR IN THE VICINITY OF, THE HARDWARE PHYSICAL LOCATION AS DESCRIBED BY THE INTERNAL LABEL OF THE SYMBOL. LETTER N INDICATES PIN NUMBER OF EXTENSION POINT.

901172A.34/2

Figure 3-13. Basic Logic Symbols Chart (Sheet 2 of 3)

LOGIC FUNCTION	SYMBOL	DESCRIPTION
SCHMITT TRIGGER		THE SCHMITT TRIGGER ACTUATES WHEN THE INPUT SIGNAL EXCEEDS A THRESHOLD VOLTAGE. THE UNACTUATED STATE OF ST IS EITHER ZERO OR ONE. WHEN ACTUATED, IT CHANGES TO THE OPPOSITE STATE AND REMAINS IN THAT STATE UNTIL THE INPUT SIGNAL NO LONGER EXCEEDS THE THRESHOLD VALUE. THE OUTPUT SIGNAL AMPLITUDE AND POLARITY ARE DETERMINED BY THE DEVICE CHARACTERISTICS AND NOT BY THE INPUT SIGNAL. A STYLIZED WAVEFORM MAY BE SHOWN INSIDE OR OUTSIDE THE SYMBOL TO INDICATE AMPLITUDE, POLARITY, THRESHOLD VOLTAGE AND DURATION.
AMPLIFIER		THE SYMBOL REPRESENTS A LINEAR OR NONLINEAR CURRENT OR VOLTAGE AMPLIFIER. THE AMPLIFIER MAY HAVE ONE OR MORE STAGES AND MAY OR MAY NOT PRODUCE GAIN OR INVERSION. LEVEL CHANGERS AND INVERTERS, CABLE DRIVERS AND RECEIVERS, EMITTER FOLLOWERS, RELAY DRIVERS, LAMP DRIVERS AND SENSE AMPLIFIERS ARE EXAMPLES OF DEVICES FOR WHICH THIS SYMBOL APPLIES. THE AMPLIFIER FUNCTION IS IDENTIFIED BY A LETTER DESIGNATION INSIDE THE SYMBOL. LETTER DESIGNATIONS FOR THE LOGIC SYMBOLS ARE LISTED AT THE END OF THIS CHART.
DOT AND DOT OR		WHERE FUNCTIONS HAVE THE CAPABILITY OF BEING COMBINED ACCORDING TO THE AND OR THE OR FUNCTION SIMPLY BY CONNECTING THE OUTPUTS THAT CAPABILITY IS SHOWN BY ENVELOPING THE BRANCHED CONNECTION WITH AN AND OR AN OR SYMBOL OF SMALLER SIZE
SIGNAL PATHS		SINGLE CHANNEL SIGNAL FLOW 2 CHANNEL 3 CHANNEL — MULTIPLE CHANNEL N = NUMBER OF CHANNELS MULTIPLE CHANNEL WITH TAKEOFF SIGNAL PATHS CROSSING WITH NO CONNECTION (NOT NECESSARILY PERPENDICULAR)
SYMBOL DESIGNATIONS	B C CD CR E EF FF LD LS M (N) R RD RG(N) S SA SR SS ST	BUFFER AMPLIFIER CLOCK CABLE DRIVER CABLE RECEIVER ERASE (DIRECT RESET INPUT) EMITTER FOLLOWER FLIP-FLOP LAMP DRIVER LEVEL SETTER MARK (DIRECT SET INPUT) NUMBER OF STAGES RESET RELAY DRIVER REGISTER, N STAGES SET SENSE AMPLIFIER SHIFT REGISTER SINGLE SHOT SCHMITT TRIGGER

901172A.34/3

Figure 3-13. Basic Logic Symbols Chart (Sheet 3 of 3)

3-22 CENTRAL PROCESSING UNIT

The following is the detailed theory of the logic circuits contained in the central processing unit. The arithmetic and control circuits are discussed in terms of registers and control signals. The generation of clock pulses and the use of these clock pulses to establish variable time intervals, or phases, during instruction execution are also described. The operation of the real-time clock, the watchdog timer, and the power fail-safe option are discussed individually, and the interrupts or traps caused by outputs from these circuits are described under interrupt and trap operation. The logic theory of the processor control panel is included.

3-23 Arithmetic and Control Circuits

The arithmetic and control circuits in the CPU consist of registers, an adder, control flip-flops, and 32 multifunction lines called the sum bus. The registers are designated A, B, C, CC, D, DIO, MC, O, P, R, RP, IODA, and IOFR. The last two registers are part of the integral IOP, and are described in that section of the manual. A block diagram of the arithmetic and control circuits is shown in figure 3-14.

C-REGISTER (C0-C31). The C-register serves as an instruction register and is used in arithmetic calculations with the A- and D-registers. All core memory information enters the CPU by means of the C-register, and this register is one of two entrance paths for private memory information. During some calculation processes, the C-register receives sum bus outputs for shifting operands and is also used as a temporary storage register for numerical values to be later transferred to the D-register. Data may be transferred to other registers or stored in private memory from the C-register by means of the sum bus. A diagram of C-register inputs and their respective enabling signals is shown in figure 3-15.

The C-register is unique among the CPU registers in that its storage circuits are made up of buffered latches instead of flip-flops. In the logic equations, these buffered latches are referred to as buffer flip-flops, identified by the symbol FB.

The operation of a buffered latch is shown in figure 3-16, using bit 1 of the C-register as an example. When the C-register is to be loaded from private memory, core memory, or the sum bus, one of the three lower inputs to the OR gate goes true, and buffer output C1 is driven true. The C1 output is fed back to the input of an AND gate containing holding term HOLDC. As long as HOLDC is true, C1 will contain a logical one, even after the qualifying signal has dropped. A zero is placed in C1 when either early data release signal EDR from memory, DCCL/1, or DCCL/2 goes true, causing HOLDC to drop. Signals DCCL/1 and DCCL/2 are timing outputs from the CPU delay lines.

When an instruction is in the C-register, outputs are taken to control flip-flops for indexing and indirect addressing, to the R-register for private memory addressing, and to the O-register for opcode decoding.

A-REGISTER (A0-A31). The A-register is one of two inputs to the adder and is one of two entrance paths to the CPU from private memory. This register is used for arithmetic calculations, alignment, shifting, checking arithmetic results, masking certain bits during comparison operations, and as an intermediate register for transfer of information through the adder to other registers and to core and private memory.

The arithmetic function of the A-register is used for indexing, incrementing and decrementing count figures, modifying numerical values, and for addition, subtraction, multiplication, and division. The alignment function (left and right shifting from the A-register or the sum bus) is used for aligning such information as bytes, halfwords, count values, I/O addresses, and I/O status. When comparison operations are taking place, the A-register contains one of the numbers to be compared in the adder.

When arithmetic results are to be checked, the information is gated into the adder from the A-register, and the adder output on the sum bus is tested.

The inputs to the A-register and their enabling signals are shown in figure 3-17.

O-REGISTER (O1-O7). The O-register, or opcode register, receives the 7-bit operation code from the C-register. The O-register outputs are decoded to provide logic signals appropriate to the instruction being executed.

The inputs to the O-register and their enabling signals are shown in figure 3-18.

RP-REGISTER (RP24-RP27). The RP-register, or register block pointer, provides the address of one 16-register block out of 16 blocks in private memory. The private memory block selected by the register block pointer is referred to as the current register block. This register is part of the program status doubleword, occupying bits 56 through 59 of PSW2. The register block number is placed in the RP-register by way of the sum bus during a load register pointer, load program status doubleword, or exchange program status doubleword instruction. The RP-register outputs are used to set the four most significant bits of the private memory address lines, LR24 through LR27.

The inputs to the RP-register and their enabling signals are shown in figure 3-19.

R-REGISTER (R28-R31). The R-register holds the four-bit private memory address which specifies one of a block of 16 fast memory registers. The number is taken from the instruction word in the C-register, and the outputs of the R-register are used to set the four least significant bits of the private memory address lines, LR28 through LR31.

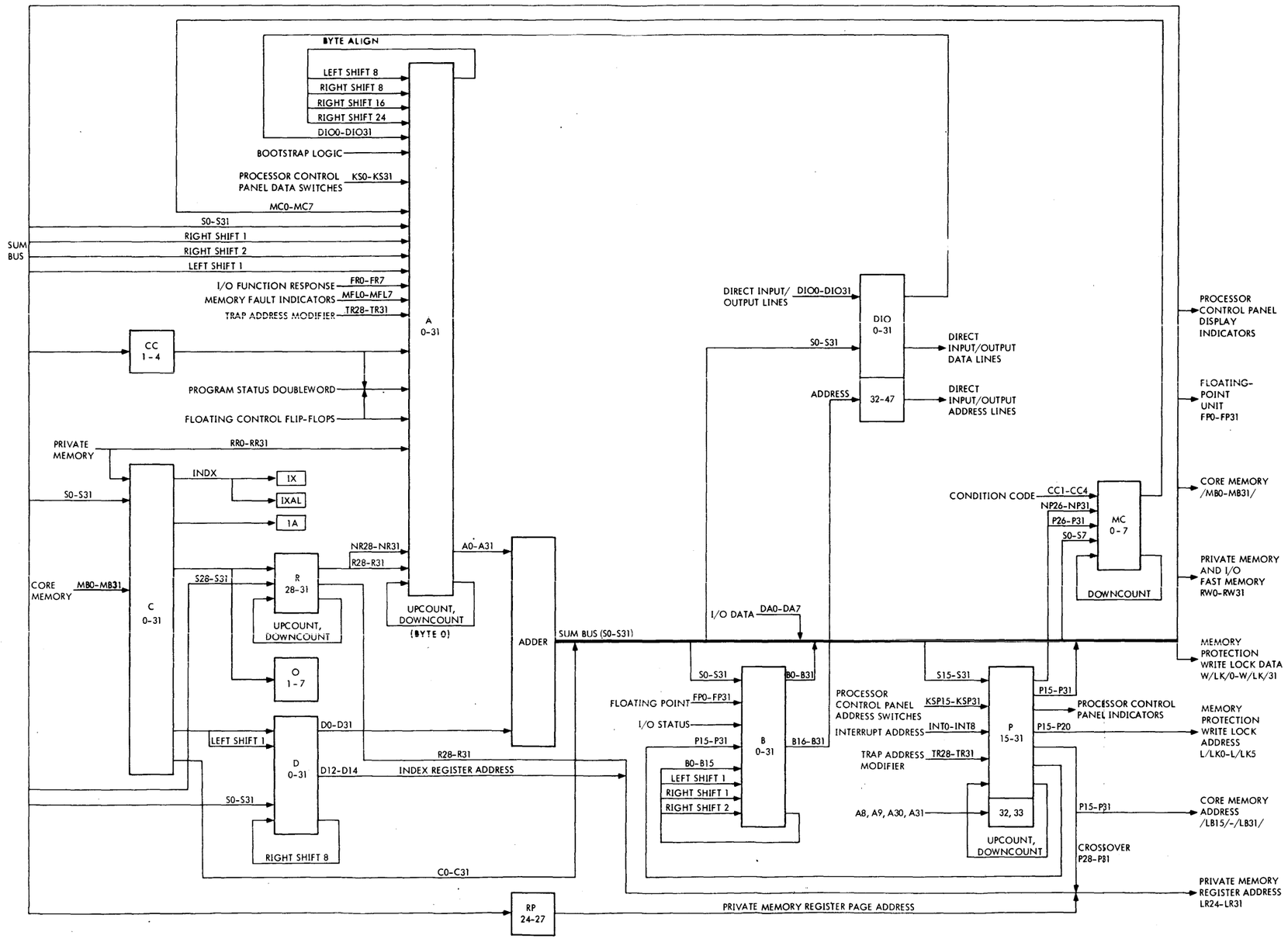


Figure 3-14. Arithmetic and Control Circuits

901172A.60

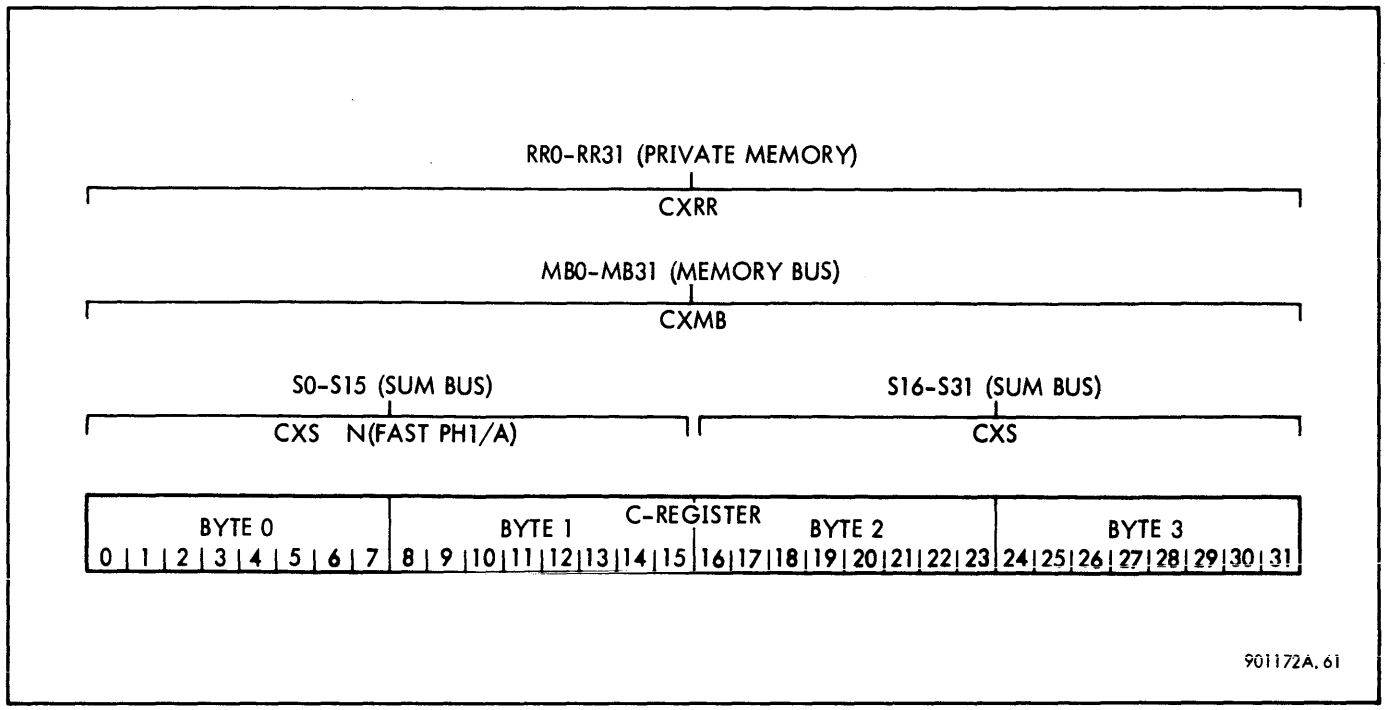


Figure 3-15. C-Register Inputs and Enabling Signals

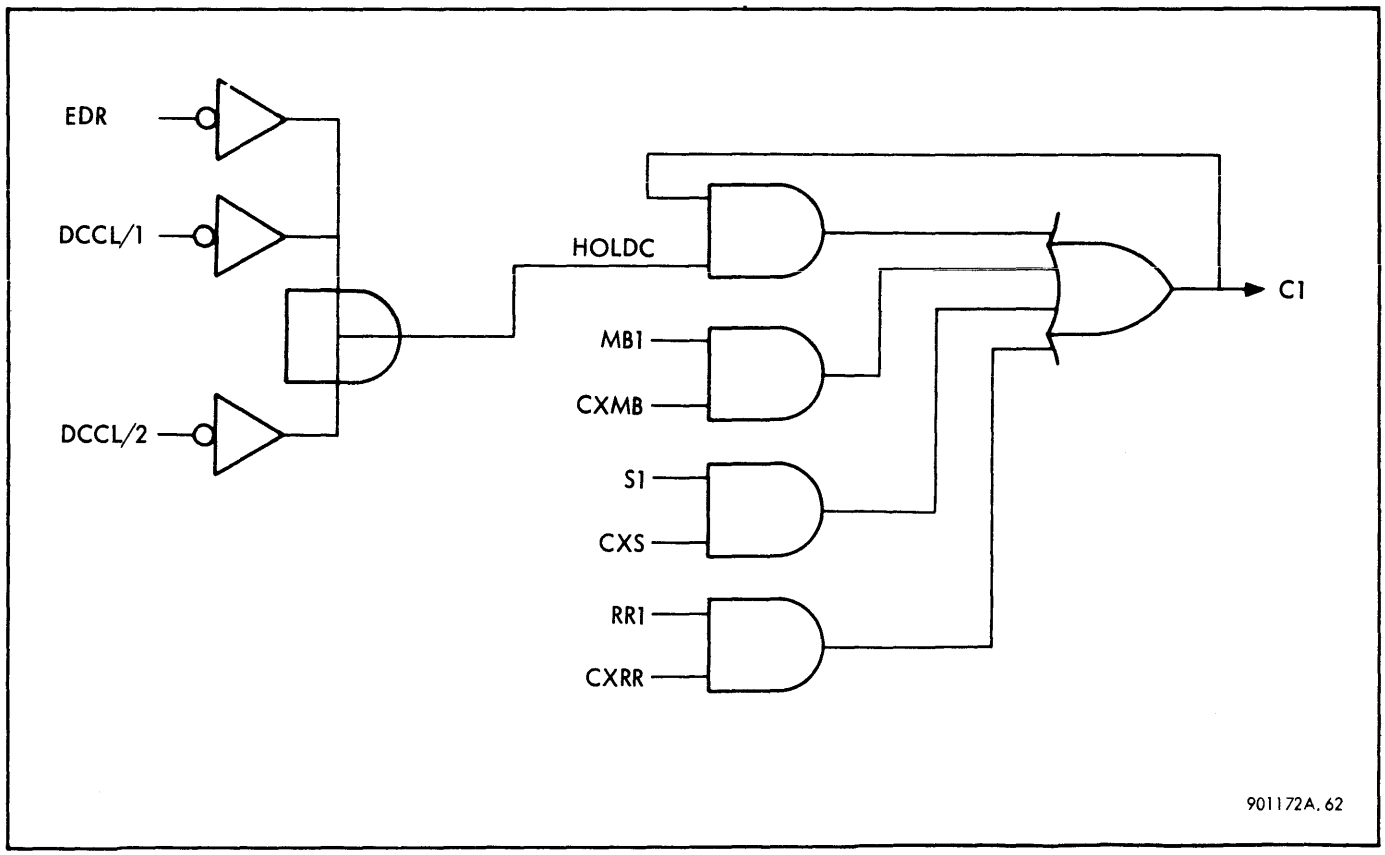


Figure 3-16. C-Register Bit 1 Logic Diagram

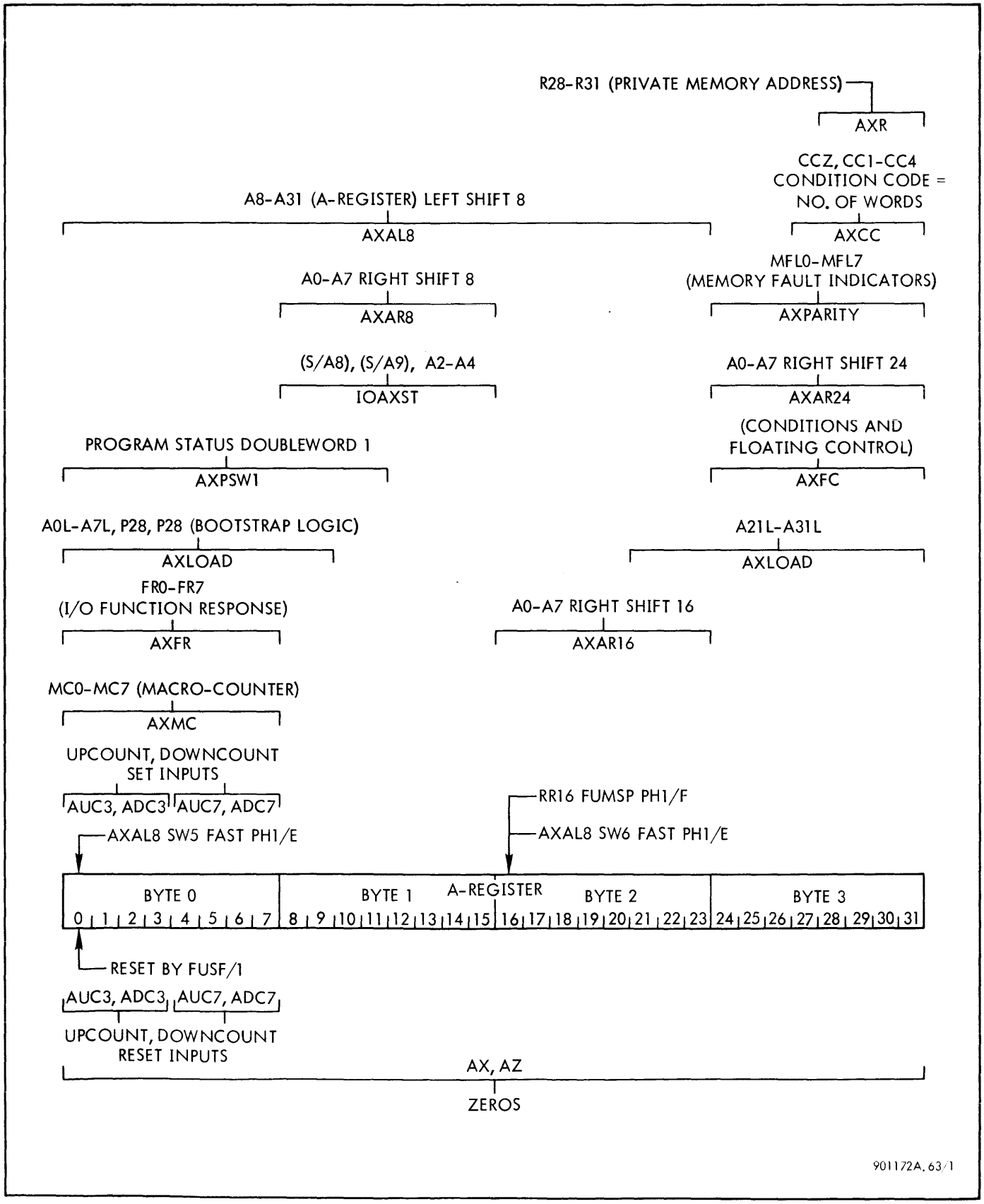


Figure 3-17. A-Register Inputs and Enabling Signals (Sheet 1 of 2)

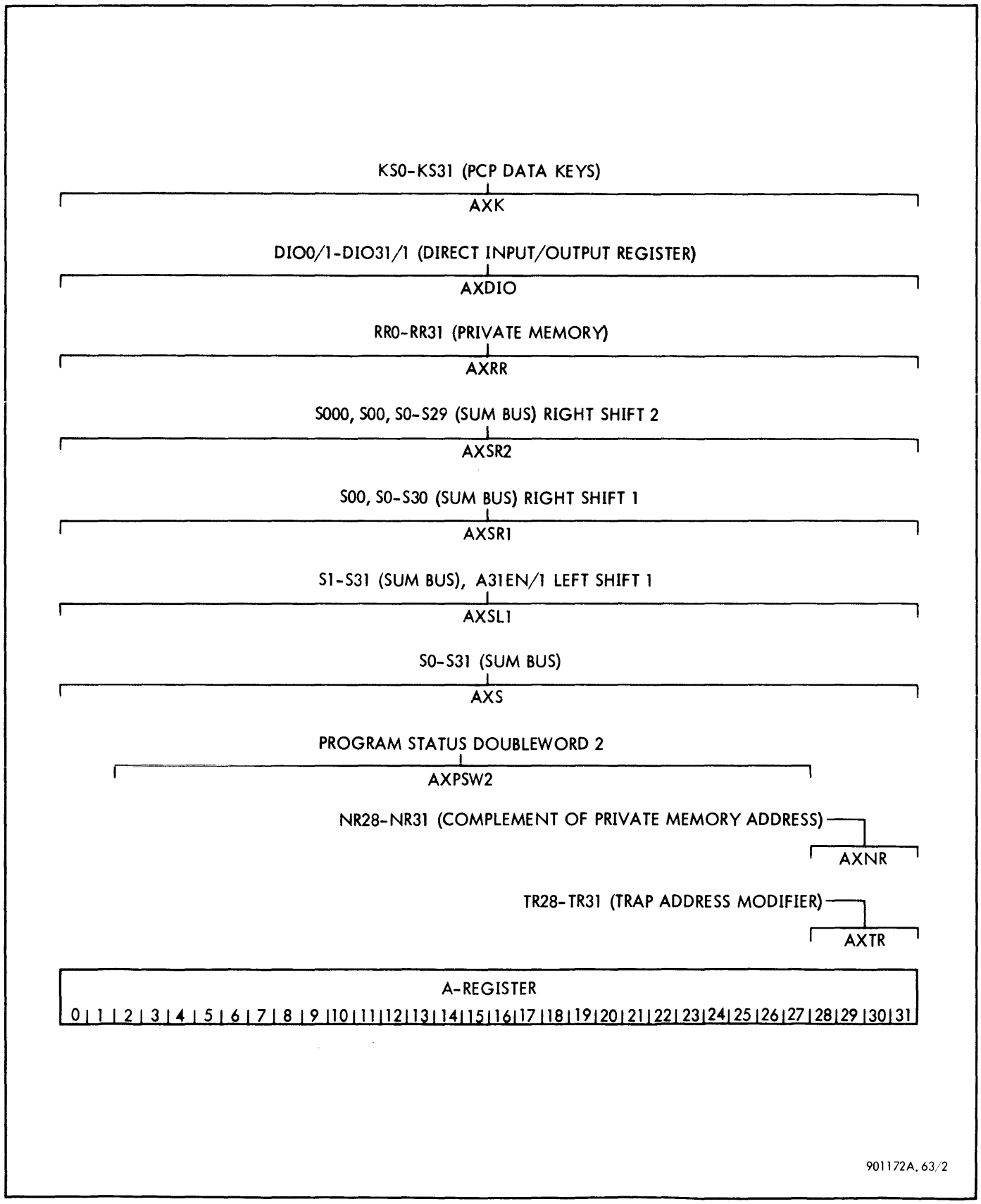


Figure 3-17. A-Register Inputs and Enabling Signals (Sheet 2 of 2)

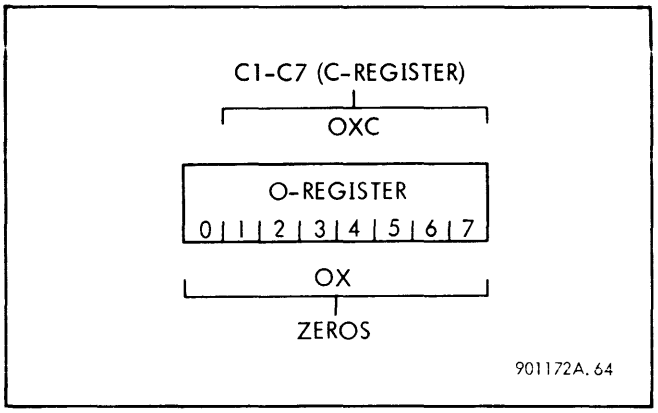


Figure 3-18. O-Register Inputs and Enabling Signals

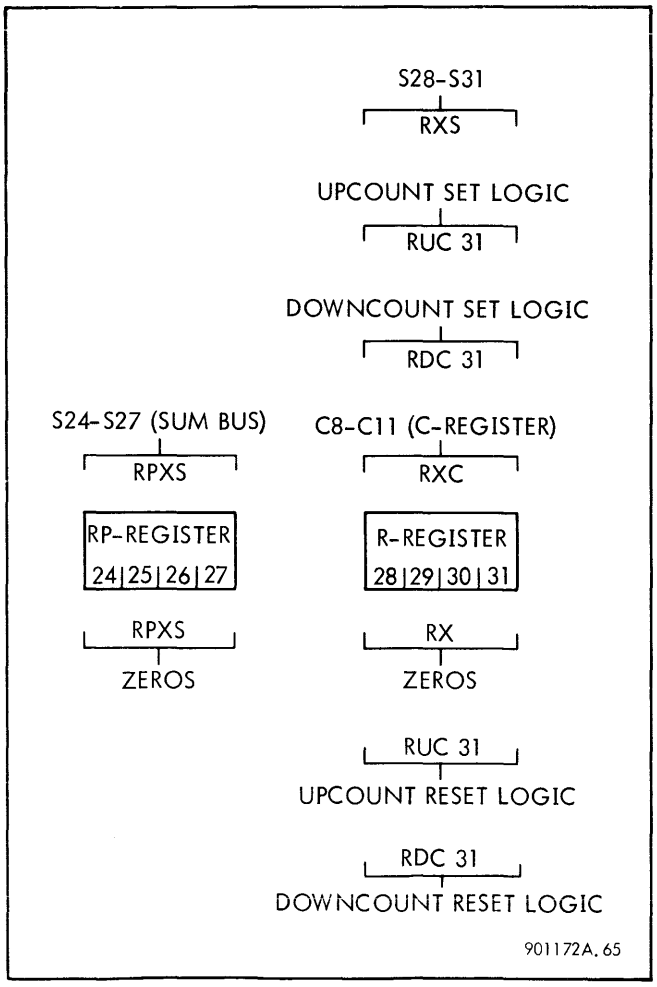


Figure 3-19. RP-Register and R-Register Inputs and Enabling Signals

The R-register contents may be increased or decreased by one to obtain the most or least significant half of a doubleword.

The inputs to the R-register and their enabling signals are shown in figure 3-19.

D-REGISTER (D0-D31). The D-register is one of two inputs to the adder. This register is used for arithmetic calculations and logic operations, sign extension, alignment, comparison, storing in core and private memory, and holding flags and status information for I/O operation.

The arithmetic functions of the D-register are used for indexing, incrementing and decrementing count figures, modifying, and for addition, subtraction, multiplication, and division.

The shift logic into the D-register from its own outputs and from the sum bus is used for alignment of bytes and halfwords before arithmetic operations and of addresses for I/O operation.

A portion of the D-register output is used to develop a private memory index register address from the index field of an instruction received from the C-register.

The inputs to the D-register and their enabling signals are shown in figure 3-20.

B-REGISTER (B0-B31). The B-register is used for temporary storage of the program address while the P-register is being used for other functions. An address in the B-register may be loaded into private memory by means of the sum bus.

During arithmetic calculations, the B-register is used to hold the multiplier, the partial product, the numerator, or the quotient. This register is also used for shifting these values when required. During direct input and output, the B-register holds the DIO effective address. During floating point operation, the B-register is used to transfer information from the floating point unit to private memory. The B-register also holds status information during I/O operation.

The B-register inputs and their enabling signals are shown in figure 3-21.

P-REGISTER (P15-P33). The P-register is primarily an address register and is used to develop core memory, private memory, and memory protection write lock addresses. The register may be incremented or decremented to obtain the next instruction in sequence, to return to an instruction after an interrupt, or to obtain the upper or lower address of a doubleword. Processor control panel addresses are transferred directly to the P-register from the PCP switches, and PCP address indicators are connected to the P-register outputs. Bits 15 through 31 of the P-register are used for addressing, and bits 32 and 33 are used for control during byte and halfword operation and for some I/O control functions.

The P-register inputs and their enabling signals are shown in figure 3-22.

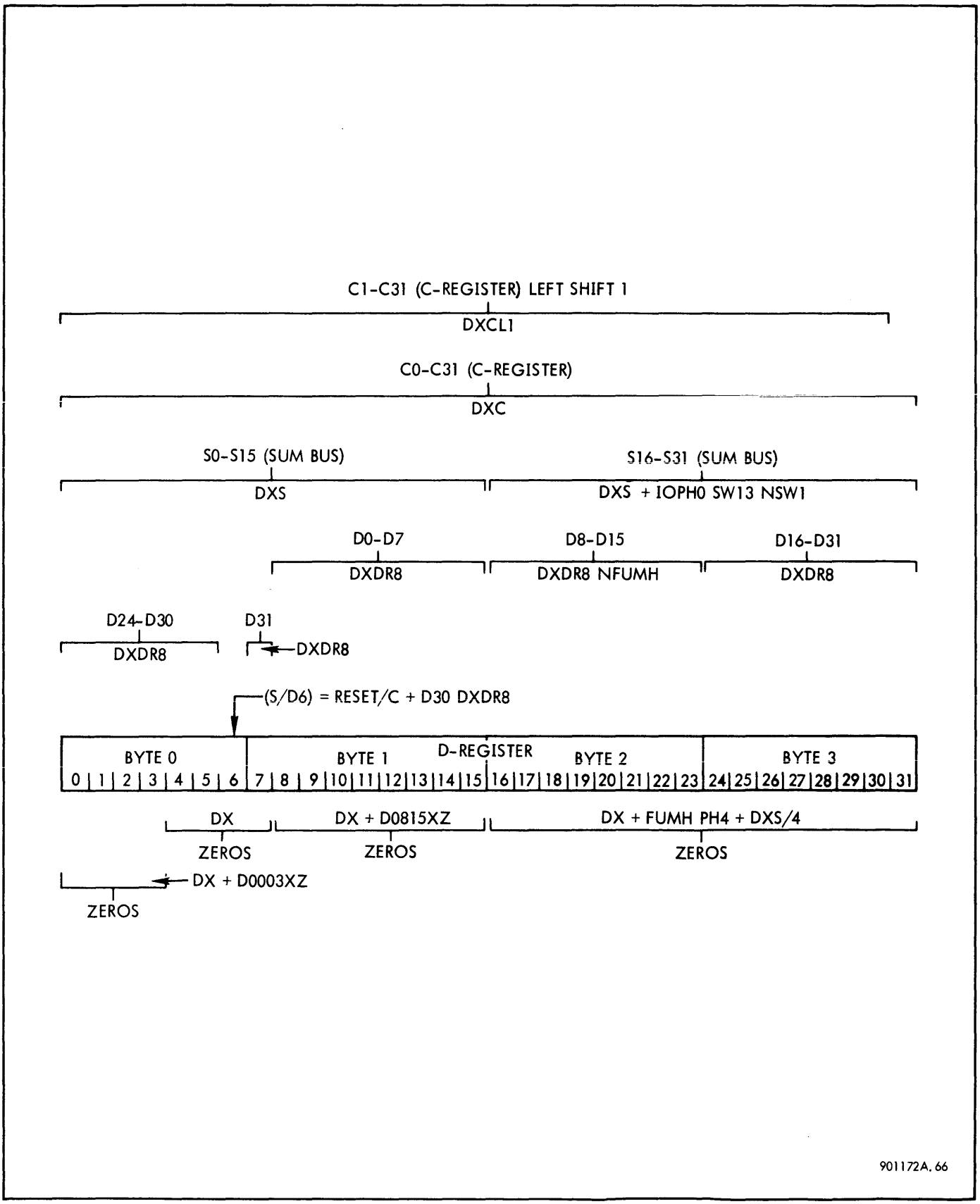


Figure 3-20. D-Register Inputs and Enabling Signals

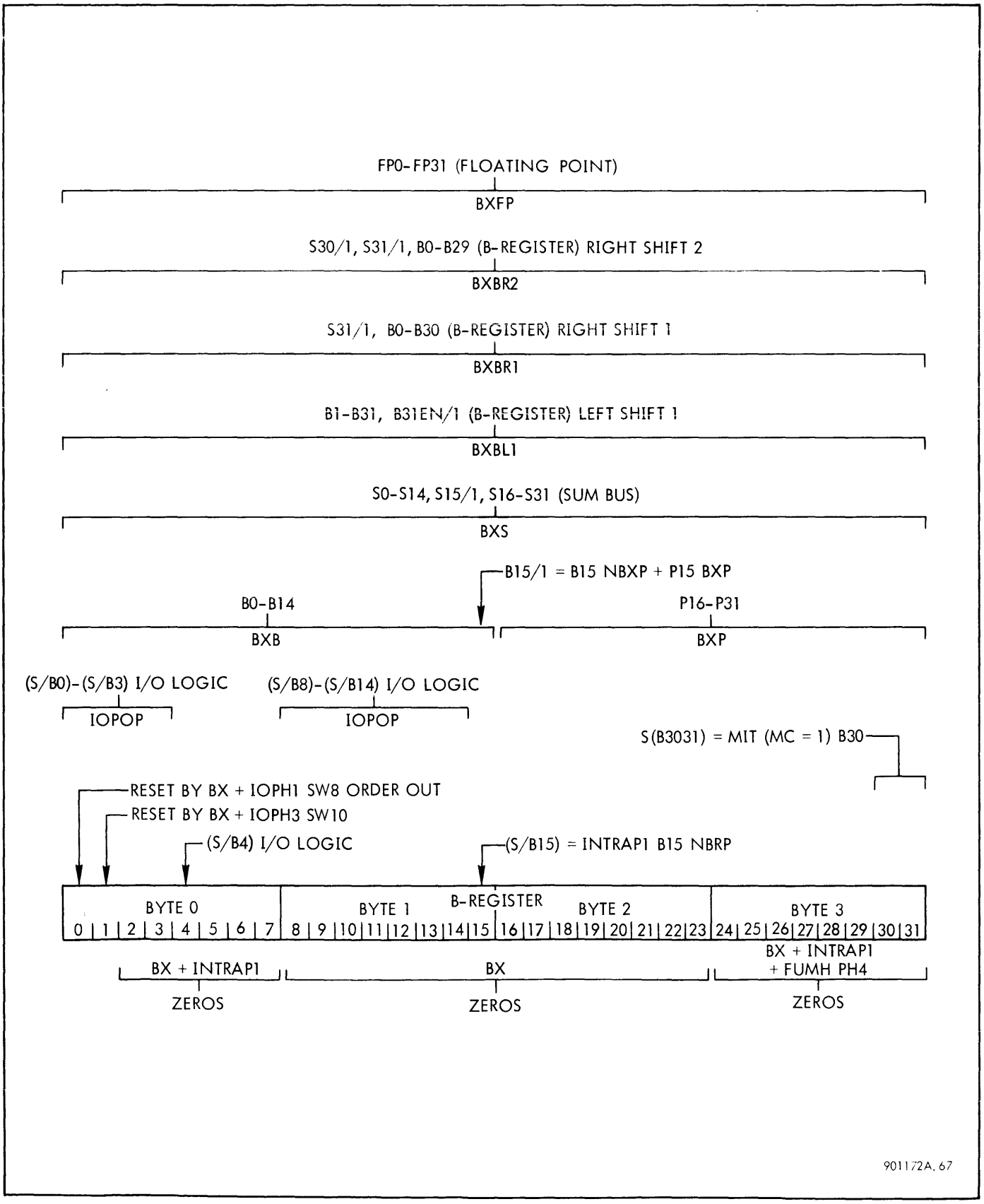


Figure 3-21. B-Register Inputs and Enabling Signals

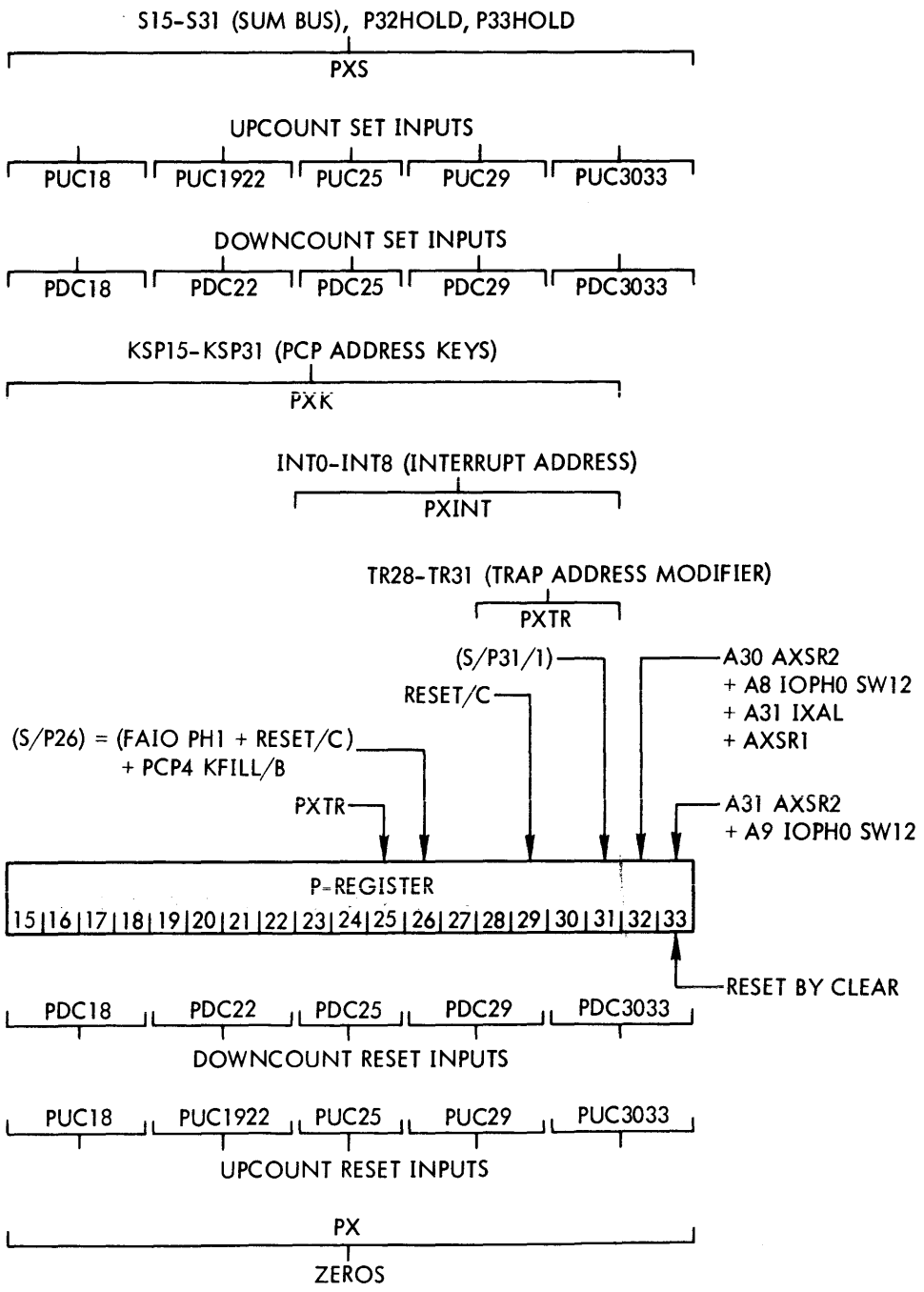


Figure 3-22. P-Register Inputs and Enabling Signals

DIO-REGISTER (DIO0/1-DIO47/1). The DIO-register holds direct input or output data and addresses during read direct and write direct instruction execution. Flip-flops DIO0/1 through DIO31/1 contain the data, and are loaded from the direct input/output lines during read direct operation and from the sum bus during write direct operation. Flip-flops DIO32/1 through DIO47/1 contain the address and are loaded from the B-register. The data outputs of the DIO-register are gated onto the sum bus during read direct operation and onto the direct input/output lines during write direct operation.

The inputs to the DIO-register and their enabling signals are shown in figure 3-23.

MC-REGISTER (MC0-MC7). The MC-register, or macro-counter, is used to keep track of the number of words for multiple-word instructions, the number of shifts for shift instructions, and the number of iterations for multiplication and division instructions. The counter is decremented by one each time the count is to be changed.

The macro-counter is loaded from the P-register during shift instructions, from the condition code register during stack and multiple instructions, and from the sum bus during the move to memory control instruction. The outputs of the counter are transferred to the A-register or are applied directly in control equations.

The inputs to the macro-counter and their enabling signals are shown in figure 3-24.

CONDITION CODE FLIP-FLOPS (CC1-CC4). The condition code flip-flops are part of the program status doubleword, occupying bit positions 0 through 3 of PSW1. These flip-flops are used as a 4-bit register in some operations. In other operations the flip-flops store bits representing the results of certain calculations. Only the register function will be discussed in this section.

During read and write direct internal mode operation, the condition code flip-flops are used to store the states of the four processor control panel sense switches, KSS1 through KSS4. When a trap occurs during program status doubleword operation, the CC flip-flops store the contents of the trap accumulator register, TRACC1 through TRACC4. During interpret and program status doubleword operation, bits 0 through 3 of the sum bus are loaded into the condition code flip-flops, and during the load conditions and floating control instruction, S24 through S27 are loaded into CC1 through CC4.

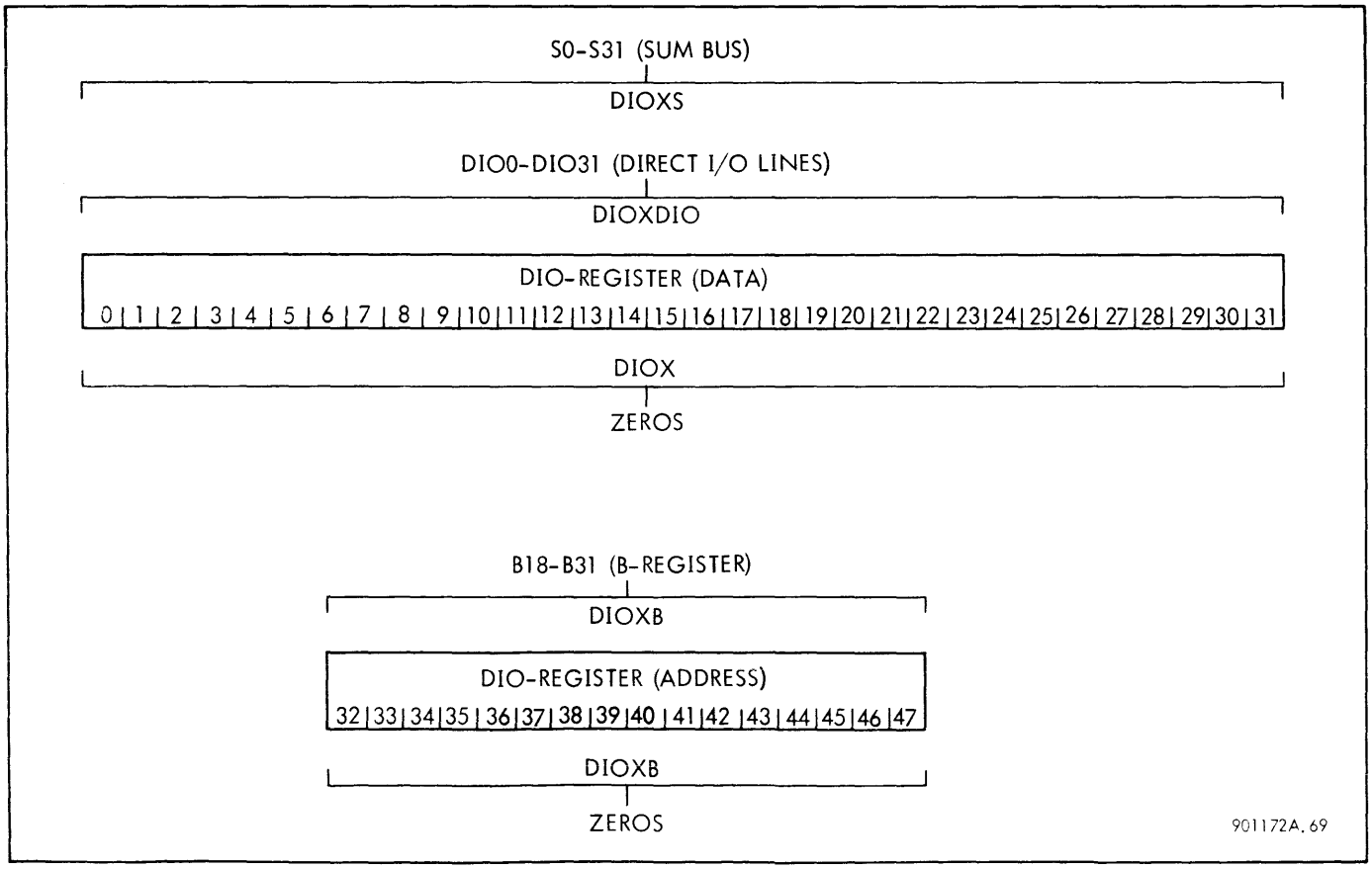


Figure 3-23. DIO-Register Inputs and Enabling Signals

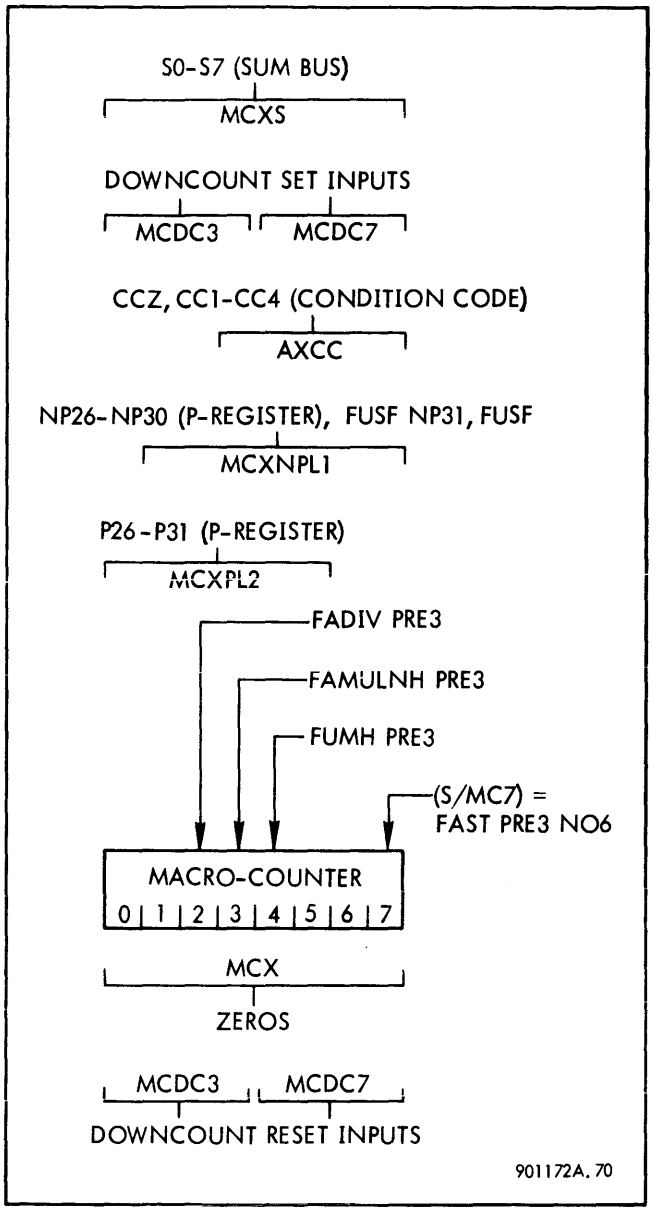


Figure 3-24. Macro-Counter Inputs and Enabling Signals

The inputs to the condition code flip-flops when used as a register are shown with their enabling signals in figure 3-25.

ADDER. The adder performs the basic arithmetic and logic operations of the computer. All adder inputs are taken from the A- and D-registers, and the sum bus, S0 through S31, is the common output for all of the results obtained in the adder.

The operations performed in the adder are listed in table 3-1. The gating terms at the top of the table are used to develop the generate and propagate signals used for parallel addition and subtraction. The enabling signals are the results of instruction decoding and are used to form the gating terms.

In parallel addition, all the bits of both arguments enter the adder at once, and all the bits of the sum or difference are formed at once. Typical addition logic is shown in figure 3-26, using bits 27 through 31 as an example. The generate terms, G_n, the propagate terms, PR_n, and the sum bits, S_n, are formed as follows:

$$G_n = A_n D_n$$

$$PR_n = A_n \oplus D_n$$

$$S_n = K_n \oplus PR_n$$

The outputs to the sum bus are gated by enabling term SXADD. The carry terms, K_n, are generated as shown in the figure.

The arrowheads pointing to each K term block represent an OR gate whose output is the appropriate carry term. Each continuous line, touching the K and PR term blocks, represents an AND gate containing the terms touched by the line and with its output at the arrowhead. From each group of four adder stages a higher order carry, represented in the figure by K27, is developed, and this term is used as the carry into the next group of four stages. The truth table for the A plus D operation is shown in table 3-2.

In the A minus D operation, the generate and propagate terms are developed as follows:

$$G_n = A \text{ ND}$$

$$PR_n = A \text{ D} + \text{NA ND}$$

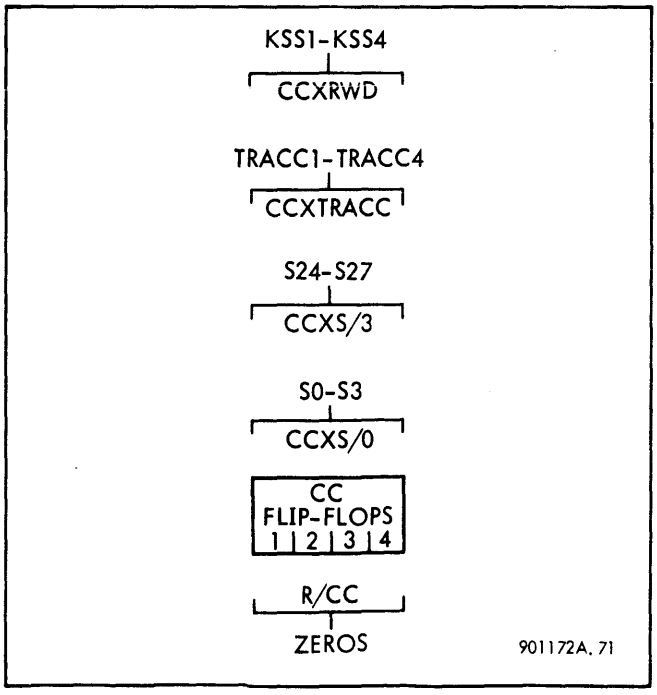


Figure 3-25. Condition Code Flip-Flop Register Inputs and Enabling Signals

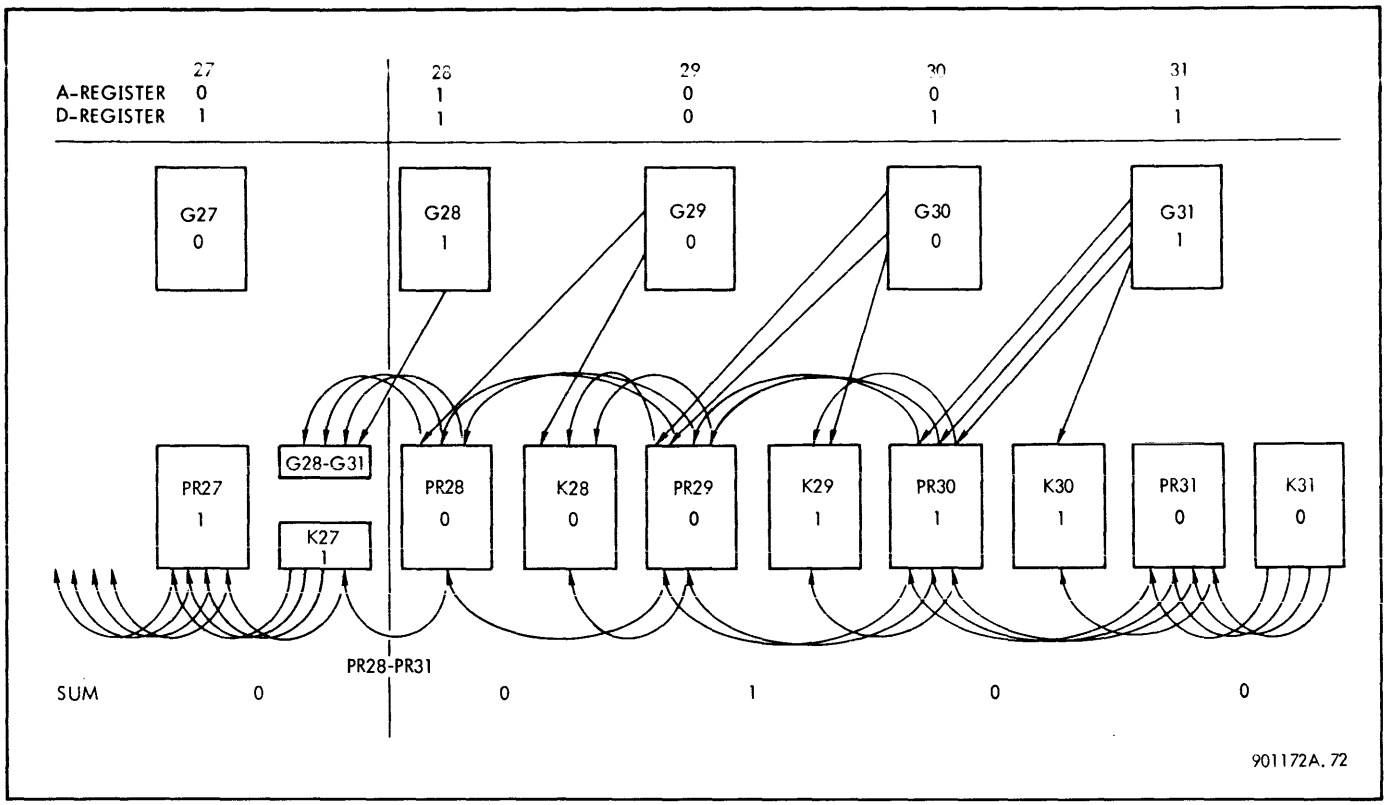


Figure 3-26. A Plus D Adder Logic

Table 3-1. Adder Operations

OPERATION	ENABLING SIGNAL	GATING TERM							
		PRXAD AD	PRXAND A ND	PRXNAD NA D	PRXNAND NA ND	GXAD AD	GXAND A ND	GXNAD NA D	K31
A+D	S/SXAPD		X	X		X			
A+D+1	*		X	X		X			X
A-D	S/SXAMD	X			X		X		X
A-D-1	†	X			X		X		
D-A	S/SXDMA	X			X			X	X
D-A-1	S/SXDMAM1	X			X			X	
A ∩ D	S/PRXAD	X							
NA ∩ D	S/PRXNAD			X					
A ∩ ND	S/PRXAND		X						
A ∩ D	S/SXAORD	X	X	X					
A ⊕ D	S/SXAEORD		X	X					
A	S/SXA	X	X						
D	S/SXD	X		X					
NA	S/SXNA			X	X				

(Continued)

Table 3-1. Adder Operations (Cont.)

OPERATION	ENABLING SIGNAL	GATING TERMS							
		PRXAD AD	PRXAND A ND	PRXNAD NA D	PRXNAND NA ND	GXAD AD	GXAND A ND	GXNAD NA D	K31
ND	S/SXND		X		X				
-A	S/SXMA			X	X				X
-D	S/SXMD		X		X				X
A+1	S/SXAP1	X	X						X
D+1	S/SXDP1	X		X					X
A-1	S/SXAM1			X	X	X	X		
D-1	S/SXDM1		X		X	X		X	

*Uses S/SXAPD with K31 set

†Uses S/SXAMD with S/K31 inhibited by raising N(S/K31)

Table 3-2. A Plus D Truth Table

A_n	D_n	G_n	PR_n	K_n	S_n
No Carry					
0	0	0	0	0	0
0	1	0	1	0	1
1	0	0	1	0	1
1	1	1	0	0	0
Carry					
0	0	0	0	1	1
0	1	0	1	1	0
1	0	0	1	1	0
1	1	1	0	1	1

The carry and sum bits are generated in the same manner as in the A plus D operation. The truth table for A minus D is shown in table 3-16.

Table 3-3. A Minus D Truth Table

A_n	D_n	G_n	PR_n	K_n	S_n
No Carry					
0	0	0	1	0	1
0	1	0	0	0	0
1	0	1	0	0	0
1	1	0	1	0	1
Carry					
0	0	0	1	1	0
0	1	0	0	1	1
1	0	1	0	1	1
1	1	0	1	1	0

In the D minus A operation, the generate and propagate terms are developed as follows:

$$G_n = NA\ D$$

$$PR_n = A\ D + NA\ ND$$

The carry and sum bits are generated in the same manner as in the A plus D operation, with flip-flop K31 initially set. The truth table for the D minus A operation is shown in table 3-17.

Table 3-4. D Minus A Truth Table

A_n	D_n	G_n	PR_n	K_n	S_n
No Carry					
0	0	0	1	0	1
0	1	1	0	0	0
1	0	0	0	0	0
1	1	0	1	0	1
Carry					
0	0	0	1	1	0
0	1	1	0	1	1
1	0	0	0	1	1
1	1	0	1	1	0

The logic for the D minus A minus 1 operation is the same as for D minus A except that flip-flop K31 is not set.

In the AND, OR, and exclusive OR logic operations, no generates or carries are formed, and the logic is developed in the PR term. This PR term becomes the result, since the sum is the exclusive OR of the PR term and a nonexistent carry. The following equation for the A AND D operation is typical of the logic operation in the adder:

$$PR_n = A_n\ D_n\ PRXAD$$

where PRXAD is the gating term for A AND D

The adder is used to gate the outputs of the A- or D-register, or the one's complement of these outputs, onto the sum bus. The enabling signals are S/SXA, S/SXD, S/SXNA, and S/SXND. In these cases, no generates or carries are formed, and the logic is developed in the PR term as in the logic operations. The following equation for A⟶S, enabled by signal S/SXA, is typical of this operation:

$$PR_n = A_n\ D_n\ PRXAD + A_n\ ND_n\ PRXAND$$

where PRXAD and PRXAND are the gating terms for A⟶S

The two's complement of the A- or D-register output is placed on the sum bus by enabling signal S/SXMA or S/SXMD. The gating signals used for S/SXNA and S/SXND are used in these cases, and flip-flop K31 is set. This is equivalent to adding one to the one's complement of the number in the register. Carries are generated in the same manner as in the A plus D operation. The same propagates are generated as in S/SXNA and S/SXND.

The A plus 1 and D plus 1 operations, enabled by S/SXAP1 and S/SXDP1, are performed by using the same gating terms as S/SXA and S/SXD and setting K31. This is equivalent to adding one to the number in the register. Carries are generated in the same manner as in the A plus D operation. The same propagates are generated as in S/SXA and S/SXD.

The A minus 1 and D minus 1 operations are performed by using the same gating terms as S/SXNA and S/SXND and developing generate terms. In the A minus 1 operation, the generate and propagate terms are as follows:

$$G_n = A_n D_n + A_n ND_n$$

$$PR_n = NA_n D_n + NA_n ND_n$$

Since the D-register is not used in this operation, the terms containing D_n are insignificant; therefore, a generate term is developed when the A-register bit is true, and a PR term is developed when the A-register bit is false.

Carries are generated as in the A plus D operation. The truth table for A minus 1 is shown in table 3-18.

Table 3-5. A Minus 1 Truth Table

A_n	G_n	PR_n	K_n	S_n
No Carry				
0	0	1	0	1
1	1	0	0	0
Carry				
0	0	1	1	0
1	1	0	1	1

In the D minus 1 operation, the generate and propagate terms are as follows:

$$G_n = A_n D_n + NA_n D_n$$

$$PR_n = A_n ND_n + NA_n ND_n$$

Since the A-register is not used in this operation, the terms containing A_n are insignificant; therefore, a generate term is developed when the D-register bit is true, and a PR term is developed when the D-register is false. Carries are generated as in the A plus D operation. The truth table for D minus 1 is shown in table 3-6.

SUM BUS. The sum bus is made up of 32 lines, S0 through S31. These lines receive inputs from several sources and have several destinations. The use of the sum bus in the arithmetic and control circuits is shown in the block diagram in figure 3-14.

All the adder outputs are carried by the sum bus. Other sources that feed the sum bus are the B-register, the P-register, the C-register, and the I/O data lines. Certain individual bits of the sum bus are set by single setting terms. All of the sum bus inputs and their enabling signals are shown in figure 3-27.

CONTROL SIGNALS. Control signals used in the CPU fall into three categories: timing signals, enabling or gating terms, and control flip-flop outputs.

Timing signals are generated by oscillators of various frequencies and from three CPU delay lines. The timing signals are discussed in the section on CPU timing.

Enabling signals are generated from instruction decoding and phase flip-flop outputs and are used to control the basic adder operation. The enabling signals are described in the adder discussion. Gating terms are derived from enabling signals, instruction decoding, and phase logic, and are used to transfer groups of information bits in parallel from one register or set of lines to another. The primary gating terms in the CPU are shown in the diagrams of the registers, the adder, and the sum bus.

Table 3-6. D Minus 1 Truth Table

D_n	G_n	PR_n	K_n	S_n
No Carry				
0	0	1	0	1
1	1	0	0	0
Carry				
0	0	1	1	0
1	1	0	1	1

Control flip-flops are used extensively in the CPU control circuits. Phase flip-flops and interrupt and trap flip-flops are discussed elsewhere in the detailed theory. Other important control flip-flops are described below. The detailed logic of the control functions of these flip-flops is described in the sequence charts for the instructions in which they are used.

Flip-Flop AM. Flip-flop AM is the arithmetic mask flip-flop in the program status doubleword. The fixed point arithmetic overflow trap is in effect when this flip-flop is set (bit 11 of the current PSW1 is a one). The trap is not in effect when the flip-flop is reset (bit 11 of the current PSW1 is a zero). Flip-flop AM is set by bit 11 of PSW1 during a load or exchange program status doubleword instruction (XPSD) or by inserting a one into bit 11 of PSW1 from the PCP with the equation:

S/AM = S11 PSW1XS

R/AM = PSW1XS

PSW1XS = FAPSD PH4 + PCP5 KPSW1/B

Flip-Flops BC0, BC1. Flip-flops BC0 and BC1 are the byte counter. The four states of the counter, 00, 01, 10, and 11, are used in the alignment of bytes or halfwords in the A-register or D-register before loading or storing takes place. The counter states represent byte numbers as follows:

	BC0	BC1	
Byte 0	1	1	(I/O, 0-0)
Byte 1	1	0	(I/O, 0-1)
Byte 2	0	1	(I/O, 1-0)
Byte 3	0	0	(I/O, 1-1)
Halfword 0	1	0	
Halfword 1	0	0	

During input/output operation, the counter is set in the reverse order from the CPU setting.

The byte counter contents are decreased by increments of one to control shifting of the appropriate register right or left eight bits at a time until the addressed byte or halfword is aligned in the right or left end of the register.

The byte counter is set during halfword addressing instructions according to the state of flip-flop P32, which contains the halfword number after index alignment has taken place. The counter is set during byte addressing instructions according to the states of P32 and P33, which contain the byte number in binary form after index alignment. In I/O operation, P32 and P33 also reflect the byte count, but receive the count from the byte count field in a private memory register rather than from the effective address.

The counter is set in IOPH2 during I/O instruction and during PH2 during modify and test instructions. The byte

count is decreased by one with control signals BCDC1 and BCDC0. The equations for the byte counter are as follows:

S/BC0 = PRE3 NP32 O1 NPRE/34
+ FAMT PH2 NRZ O1 NP32
+ IOPH2 SW13 P32 NPRE/34
+ NBC0 NBC1 BCDC1

R/BC0 = N(NBCX NCLEAR NBCDC0)

NBCDC0 = N(BCDC1 NBC1 BC0)

BCDC1 = FUMMC PH6
+ IOPH2 SW15
+ NBCZ PRE4
+ IOPH2 SW14

S/BC1 = PRE3 OU7 NPRE/34 NP33
+ FAMT PH2 NRZ OU7 NP33
+ IOPH2 SW13 P33 NPRE/34
+ NBC1 BCDC1

R/BC1 = N(NBCX NCLEAR NBCDC1)

Flip-Flops CC1 through CC4. Flip-flops CC1 through CC4 are condition code flip-flops which occupy bits 0 through 3 of the program status doubleword. They are set during load or exchange program status doubleword instructions or during load conditions and floating control instructions, or from the PCP. During certain other instructions they are set to indicate the nature of the results of the instruction. Loading the condition code flip-flops in parallel as a register is described in the paragraphs on registers.

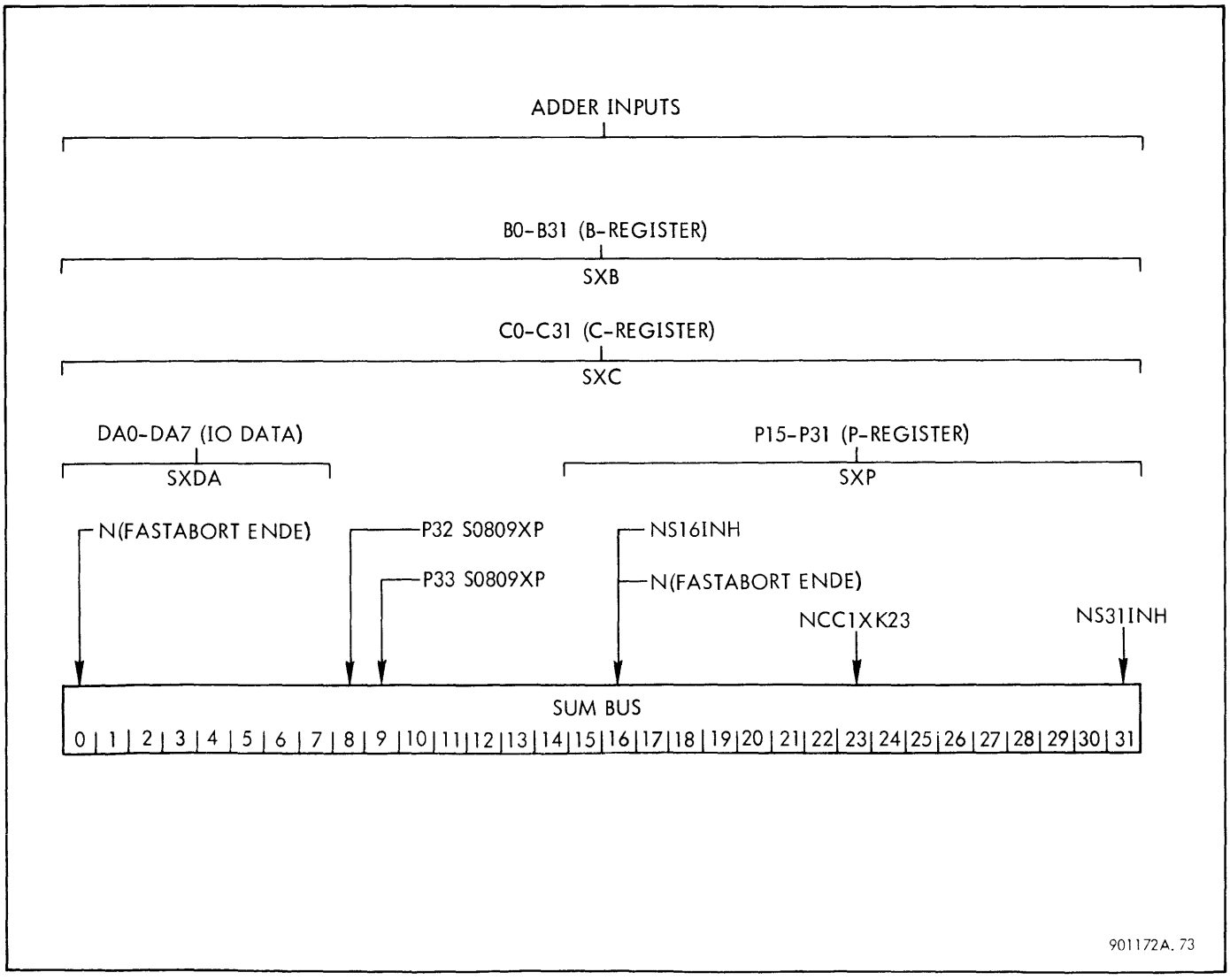


Figure 3-27. Sum Bus Inputs and Enabling Signals

Flip-Flop DM. Flip-flop DM is the decimal mask flip-flop, which occupies bit position 10 in the program status doubleword. The flip-flop is set by bit 10 of the sum bus during a program status doubleword instruction or from the PCP as follows:

S/DM = PSW1XS S10

R/DM = PSW1XS

The decimal mask bit does not affect the operation of the Sigma 5 computer. The bit position is used only to preserve the status of the Sigma 7 decimal arithmetic fault trap mask when a Sigma 7 program is being executed.

Flag Flip-Flops FL1 Through FL3. These flip-flops are used to store conditions during multiply, floating shift, divide, and modify and test instructions.

Flip-flop FL1 stores the sign in floating shift and divide instructions and serves with FL2 as part of a 2-bit extension of the A-register during multiply instructions in case of I/O intervention. The equations for FL1 are as follows:

S/FL1 = RR0 PRE3 + S30 MIT

R/FL1 = MIT + CLEAR

Flip-flop FL2 serves, with FL1, as part of a 2-bit extension of the A-register during multiply instructions in case of I/O intervention. This flip-flop also stores the sign of the shift count in a shift instruction. If FL2 is set, a right shift is indicated. The equations for FL2 are as follows:

S/FL2 = P25 PRE3 + S31 MIT

R/FL2 = MIT + CLEAR

Flag flip-flop FL3 has four functions and is set and reset according to the following equations:

S/FL3 = P31 PRE3 + FUSF/1 S0815Z S1631Z
B0031Z + K00 K00HOLD + CC1XK23

R/FL3 = N[(FUSF PH8) + (FUS PH5)]

The four functions are:

a. Stores the state of P31, which contains the shift count in a shift instruction. If FL3 is set, indicating an odd shift count, a fixed point shift instruction starts out with a 1-bit right shift, then shifts right by twos.

b. Indicates that the mantissa equals zero in a floating shift instruction (FUSF/1). Setting FL3 in this instruction causes CC1 to be set (fraction normalized) and CC2 to be reset (no characteristic underflow). Flip-flop FL3 also generates FSHEX in a floating shift instruction so that the instruction will exit from the shift operation when the mantissa equals zero.

c. During doubleword arithmetic instructions and load absolute instructions, stores a carry bit until the next phase of the instruction. Flip-flop K00 contains the carry, and K00HOLD is driven true by FADW/1 PH1 or FALOAD/A PH2.

d. Indicates a byte instruction in the family of modify and test instructions. The equation for CC1XK23, the setting term in this case, is as follows:

CC1XK23 = FAMT PH2 NINTRAP OU7

where OU7 defines byte addressing in modify and test instructions.

Floating Mode Control Flip-Flops FS, FZ, and FNF. Flip-flops FS, FZ, and FNF are floating significance, floating zero, and floating normalize flip-flops and occupy bits 5 through 7, respectively, in the program status doubleword. The outputs of these flip-flops are transmitted to the floating point unit to be used for control purposes. The flip-flops are set and reset by program status doubleword or from the PCP (PSW1XS) and load conditions and floating control instructions (FCXS).

Flip-flop FS controls the floating point unit with respect to floating point significance checking. The flip-flop is set and reset as follows:

S/FS = S5 PSW1XS + S29 FCXS

R/FS = FCXS + PSW1XS

Flip-flop FZ controls the floating point unit with respect to the generation of zero results. The flip-flop is set and reset as follows:

S/FZ = S6 PSW1XS + S30 FCXS

R/FZ = FCXS + PSW1XS

Flip-flop FNF controls the floating point unit with respect to the normalization of the results of floating point additions and subtractions. The flip-flop is set and reset as follows:

S/FNF = S7 PSW1XS + S31 FCXS

R/FNF = FCXS + PSW1XS

Flip-Flop IA. Flip-flop IA is the indirect address flip-flop, used to control indirect addressing during instruction preparation phases. The flip-flop is set during phase PRE1 if bit position zero of the instruction word contains a one as follows:

S/IA = C0 PRE1

R/IA = . . .

If flip-flop IA is set during an immediate instruction, signal FAILL is generated from IA and FAIM (immediate family) to start a trap sequence for the nonexistent instruction category. In this case, the trap routine is entered because an immediate instruction may not be indirectly addressed.

The IA outputs are used to control memory access during the preparation phases to read the indirect address from core memory. Signal BRPRE2, which causes preparation phase PRE2 to repeat, is qualified by a one in IA. Signal IA also helps to control the timing of the adder during addition of the contents of the A- and D-registers for indexing if the instruction is indirectly addressed. During an analyze instruction, signal IA is used to set condition code flip-flop CC3 to indicate indirect addressing.

Interrupt Group Inhibit Flip-Flops CI, II, and EI. Flip-flops CI, II, and EI are interrupt inhibit flip-flops and occupy bit positions 37 through 39 in the program status doubleword. If any of these flip-flops contain a one, the associated interrupt is inhibited. Zeros in these flip-flops permit the associated interrupts to occur.

The flip-flops are set with the PCP switches or a load or exchange program status doubleword instruction (PSW2XS) or a write direct instruction (INHXWD). A write direct instruction sets the interrupt inhibits in the internal mode when bit positions 26, 27, and 29 through 31 contain ones. A one in bit 29 sets flip-flop CIF; a one in bit 30 sets flip-flop II, and a one in bit 31 sets flip-flop EI. A write direct internal mode instruction resets the interrupt inhibit flip-flops with a zero in bit 27, a one in bit 26, and ones in the desired interrupt bit positions (29 through 31).

Flip-flop CIF, bit position 37 in the program status doubleword, is the counter interrupt group inhibit flip-flop and allows or prevents the four counter-equals-zero groups of interrupts. The equations are as follows:

$$S/CIF = S5\ PSW2XS + INHXWD\ B27\ B29$$

$$INHXWD = CCXRWD\ B26 + OLD$$

$$CCXRWD = FARWD\ B1619Z\ PH1$$

$$R/CIF = INHXWD\ B29 + PSW2XS$$

where FARWD is the read/write direct family and B1619Z indicates that bit positions 16 through 19 of the instruction word contain zeros (internal mode).

The false output of the CIF flip-flop is used to keep signal ENCNTR from enabling the counter-equals-zero interrupt levels as follows:

$$ENCNTR = NCIF\ NHRQBZC\ (R89 + R1011)$$

Flip-flop II, bit position 38 in the program status doubleword, is the input/output group inhibit flip-flop and allows or prevents the input/output and the control panel interrupts. The equations are as follows:

$$S/II = S6\ PSW2XS + INHXWD\ B27\ B30$$

$$R/II = B30\ INHXWD + PSW2XS$$

The false output of flip-flop II is used to keep signal ENIO from enabling the input/output and control panel interrupts as follows:

$$ENIO = NII\ NHRQBZI\ (R1213 + R1415)$$

Flip-flop EI, bit position 29 of the program status doubleword, is the external interrupt group inhibit flip-flop and allows or prevents the 14 groups of external interrupts. The equations are as follows:

$$S/EI = S7\ PSW2XS + INHXWD\ B27\ B31$$

$$R/EI = B31\ INHXWD + PSW2XS$$

The false output of flip-flop EI is used to keep signal DAT29 from enabling the external interrupt levels as follows:

$$DAT29 = NEI\ NEWDM + \ldots$$

Flip-Flop IX. Index flip-flop IX is used to control indexing in the instruction preparation phases. The flip-flop is set in phase PRE1 if the index field of the instruction word is nonzero. The equations are as follows:

$$S/IX = INDX\ PRE1$$

$$INDX = (C12 + C13 + C14)\ (C3 + C4 + C5)$$

$$R/IX = PRE/12 + CLEAR$$

The second AND gate on the INDX term is used to prevent indexing in instructions that may not be indexed.

The IX outputs are used to control the adder logic during the preparation phases when the contents of the A- and D-registers are being added.

Flip-Flop IXAL. Index alignment flip-flop IXAL is used to control register alignment according to byte, halfword, and doubleword addressing during indexing operation. The flip-flop is always set in PRE1 when the instruction is indexed unless word addressing is being used. The equations are as follows:

$$S/IXAL = PRE1\ INDX\ (FAHW + FABYTE + FADW)\ NCLEAR$$

$$R/IXAL = \ldots$$

The IXAL outputs are used to control the right or left shifting of the A-register in preparation phase PRE2 so that the index displacement value is correctly lined up with the word in the instruction register. (See the indexing discussion in the section on preparation phases.) Signal IXAL is also used to enable setting P32 in PRE2 during halfword operation.

Flip-Flop NMASTER. Master/slave mode control flip-flop NMASTER occupies bit position 8 in the program status

doubleword. The computer is in the master mode when this bit contains a zero and in the slave mode when the bit contains a one.

The flip-flop is set either from the PCP or with a load or exchange program status doubleword instruction as follows:

S/NMASTER = S8 PSW1XS

R/NMASTER = PSW1XS

The outputs of the NMASTER flip-flop are used with signal FAPRIV (family of privileged instructions) to set trap flip-flop TRAP and trap accumulator flip-flop TRACC3. Setting TRAP causes the program to trap to location X'40' because of a nonallowed operation. Signal TRACC3 causes condition code flip-flop CC3 to be set when an exchange program status doubleword instruction is executed as the result of the nonallowed operation trap.

Switch Flip-Flops SW0 Through SW15. Switch flip-flops SW0 through SW15 are used to define certain states in the CPU and to define subphases during CPU instruction execution and integral IOP service. The functions of switches SW0 through SW7 are listed below:

a. SW0

1. Indicates that S is not equal to zero in floating point, load absolute, and some doubleword instructions.

2. Indicates that proceed signal PR was not received in I/O instruction execution.

3. Indicates zero byte count in the integral IOP service operation.

b. SW1

1. Indicates space count overflow or underflow in stack instructions.

2. Indicates order in or order out during integral IOP service operation.

c. SW2

1. Indicates nonzero value in the R-register during modify and test instructions.

2. Indicates space count equals zero in stack instructions.

3. Sustains PH4 until control strobe signal is received during I/O instruction execution.

4. Indicates order out or data out during integral IOP service operation.

d. SW3

1. Indicates word count overflow or underflow in stack instructions.

2. Stores the state of P23 for IOP address purposes in I/O instruction execution.

3. Stores terminal order condition during integral IOP service operation.

e. SW4

1. Indicates word count equals zero in stack instructions.

2. Indicates data chaining during integral IOP service operation.

f. SW5

1. Stores the trap-on-space inhibit bit in stack pointer doubleword during stack instructions.

2. Indicates indirect addressing in analyze instruction.

3. Stores the state of P21 for IOP address purposes in I/O instruction execution.

4. Indicates transfer in channel condition during integral IOP service operation.

g. SW6

1. Stores the trap-on-word inhibit bit in the stack pointer doubleword during stack instructions.

2. Stores the state of P22 for IOP address purposes in I/O instruction execution.

3. Controls interface end data signal ED during integral IOP service operation.

h. SW7

1. Distinguishes between modify stack pointer and other instructions in stack family.

2. Indicates positive sign in load absolute word or doubleword instructions.

3. Stores function strobe leading acknowledge signal FSL or available output priority signal AVO during I/O instruction execution.

4. Controls end service signal ES during integral IOP service operation.

Switch flip-flops SW8 through SW15 define subphases in instruction execution and integral IOP service operation. The flip-flops are set sequentially by step signal STEP815. They may also be set as specified in the individual instructions or I/O operations by branch signals such as BRSW8. The following is a typical equation for these flip-flops:

S/SW11 = NRESET BRSW11 + SW10 STEP815

Flip-Flops WK0, WK1. Write key flip-flops WK0 and WK1 occupy bit positions 34 and 35 in the program status doubleword. These flip-flops contain the 2-bit write key used with the 2-bit write locks stored in the memory protection registers for each page of memory addresses. In order to read from the addressed memory location, the write key in the program status doubleword must match the write lock stored for the page containing the addressed memory location. The write key flip-flops are set from the PCP or by a program status doubleword instruction as follows:

S/WK0 = S2 PSW2XS

R/WK0 = PSW2XS

S/WK1 = S3 PSW2XS

R/WK1 = PSW2XS

The write key outputs are compared with the memory protection register outputs LCK0 and LCK1, and if a mismatch is detected where both are non-zero, a trap sequence is entered. The detailed logic of the write keys and write locks is given in the paragraphs on memory protection.

PRIVATE MEMORY REGISTERS. The private memory registers are located on a set of FT25 fast access memory modules. The registers are installed in blocks of 16, numbered register 0 through F in hexadecimal notation, as shown in figure 3-28. Each register contains a 32-bit word.

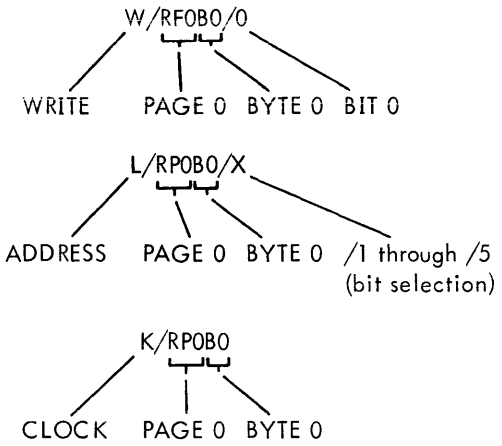
A maximum of 16 private memory blocks may be installed in the computer. Each block is assigned a page number, 0 through 16, and is addressed by the RP-register with codes from 0000 to 1111. Page 0 is included in the standard computer; pages 1 through 15 are optional.

Each private memory block consists of four FT25 fast access memory modules. The distribution of the words among the four modules is shown in block diagram form in figure 3-29. Each module contains one byte of any given word.

One FT25 module contains 16 SDS 304 8-bit integrated circuit memory elements. A simplified diagram of a single memory element is shown in figure 3-30. Although not shown in the diagram, the control, address, and V_{CC} inputs are applied to all flip-flops in the element.

Each bit of the element is addressed individually by the address lines on pins 2, 3, and 4. The 3-bit address code selects one of the eight flip-flops. The control line, also, contains address information. When the control line is false, the states of all bits in the memory element remain unchanged, regardless of the state of the read-write clock. When the control line is true, bits of the element may change state if the read-write clock is true. When a control line is false, the data output lines from all flip-flops under the control of that line are high. All the data output lines in one memory element are connected in parallel. The output lines from the two memory elements representing one bit on a module are also connected in parallel. Using this arrangement, it is possible to combine address and control lines to select the memory element and the flip-flop within the memory element that controls any one data output line.

The arrangement of bits in the memory elements on one FT25 fast access memory module is shown in figure 3-31. The module shown contains byte 0 of the standard private memory block, designated page 0. Each of the 16 memory elements contains eight corresponding bits in eight registers. The data, address, and write clock signals are interpreted as follows:



Input and output data signals for the four modules in a memory block are shown in figure 3-32. Address lines in the four modules are identical.

Individual bits in each memory element on the FT25 modules are selected by address lines LR28 through LR31. As indicated in figure 3-31, a memory element contains a corresponding bit for each of eight registers. Dividing the memory elements into two sets of eight, as shown in figure 3-31, the three least significant bits (LR29, LR30, and LR31) select one of eight registers in each set (see figure 3-33). Address line NLR28 gates information into the memory elements containing registers 0 through 7, and LR28 gates information into the elements containing registers 8 through F. Signal NIOFM, indicating that I/O fast memory is not being addressed, is connected to both control line gates. The gating signal is connected to the control line input of every memory element. The total effect of the LR28 through LR31 address lines is to select one of 16 registers for input or output of data. Gating signals RP24 through RP27, which designate the private memory page, are taken from the RP-register. Since each memory block is equivalent to one page, all four FT25 modules in one block receive the same code from the RP-register and are enabled at one time. The RP lines for the module in figure 3-31 contain NRP24 through NRP27, which select page 0000.

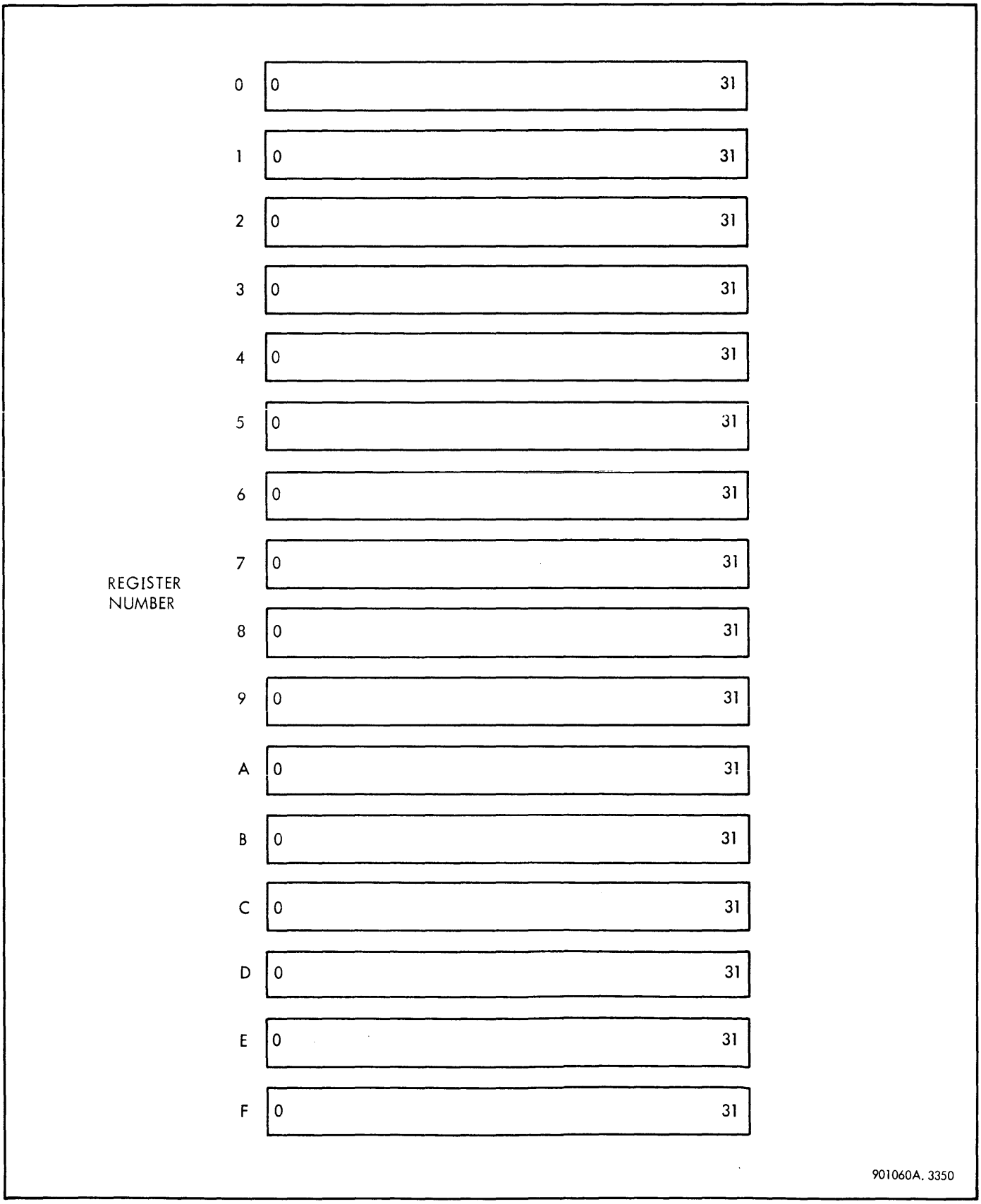


Figure 3-28. Private Memory Register Block

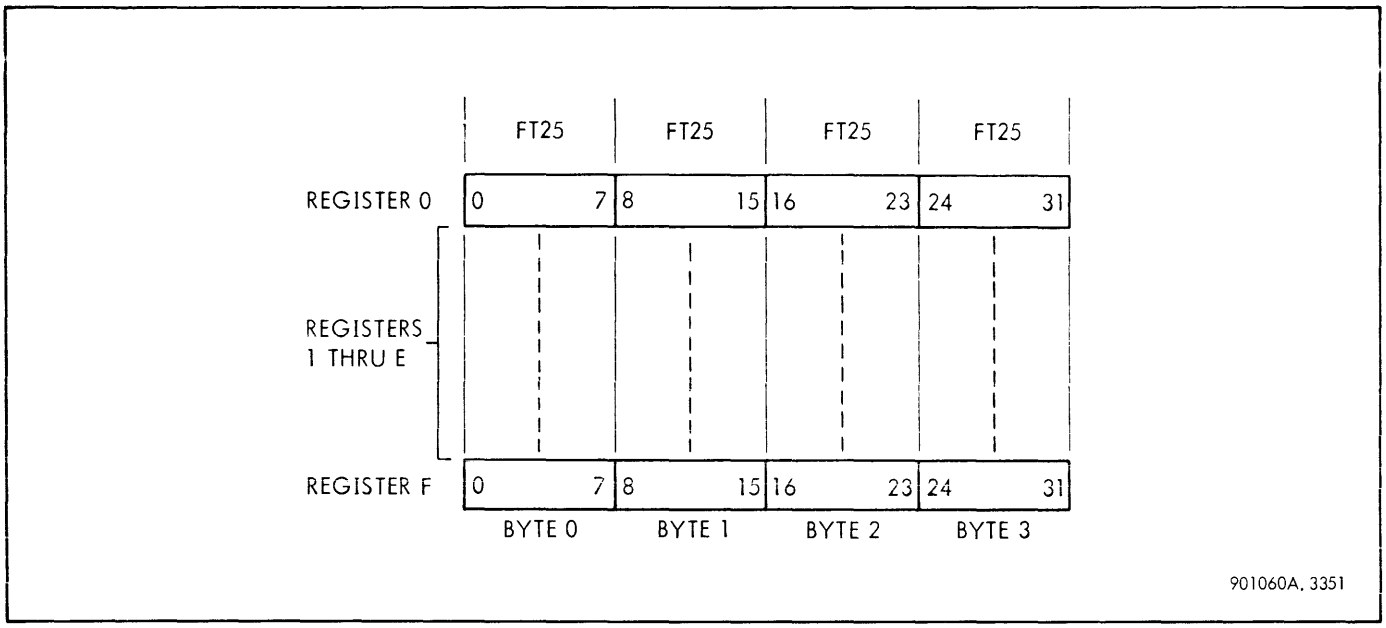


Figure 3-29. Word Distribution in Private Memory Block

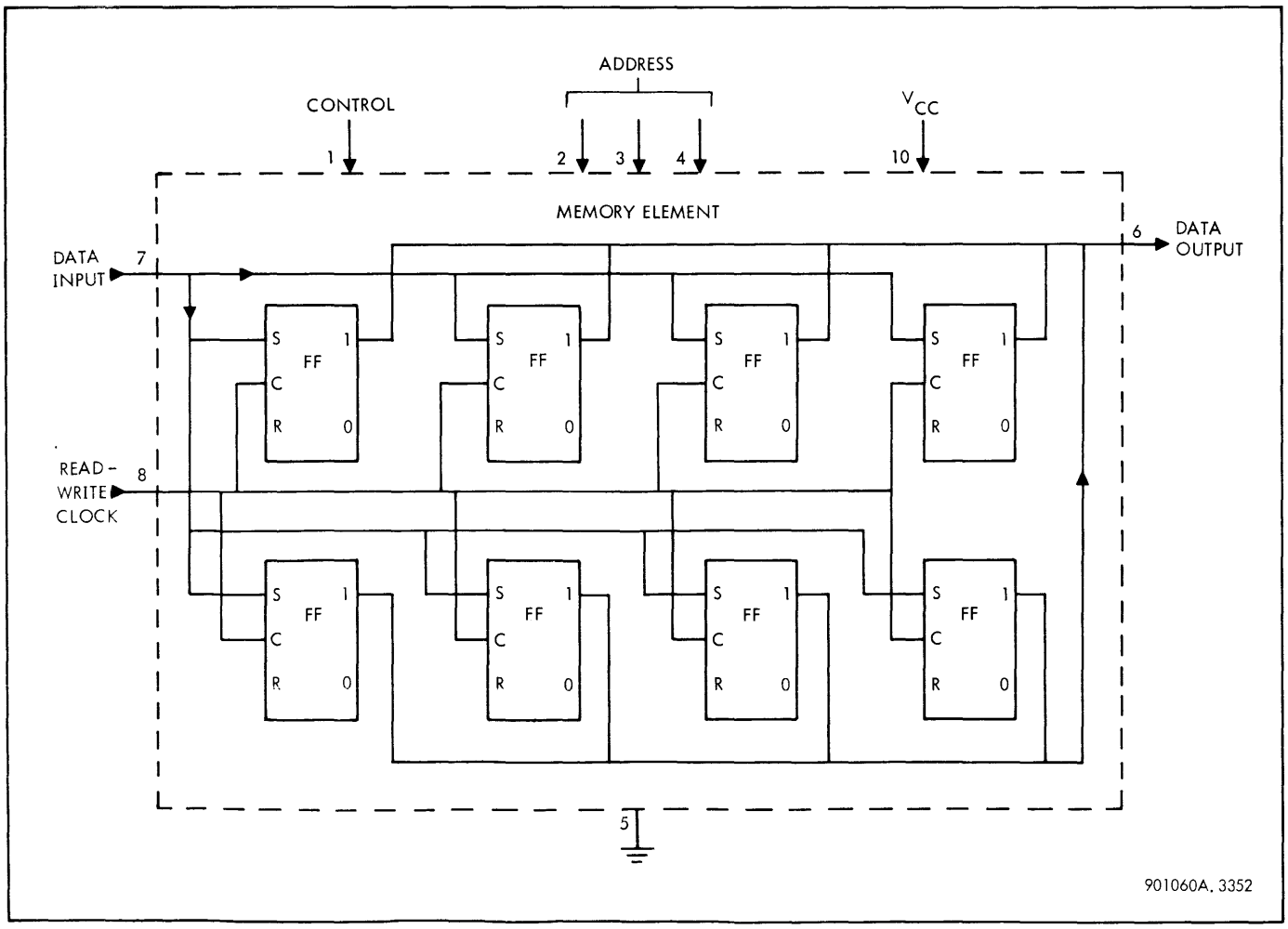


Figure 3-30. SDS 304 Memory Element, Simplified Diagram

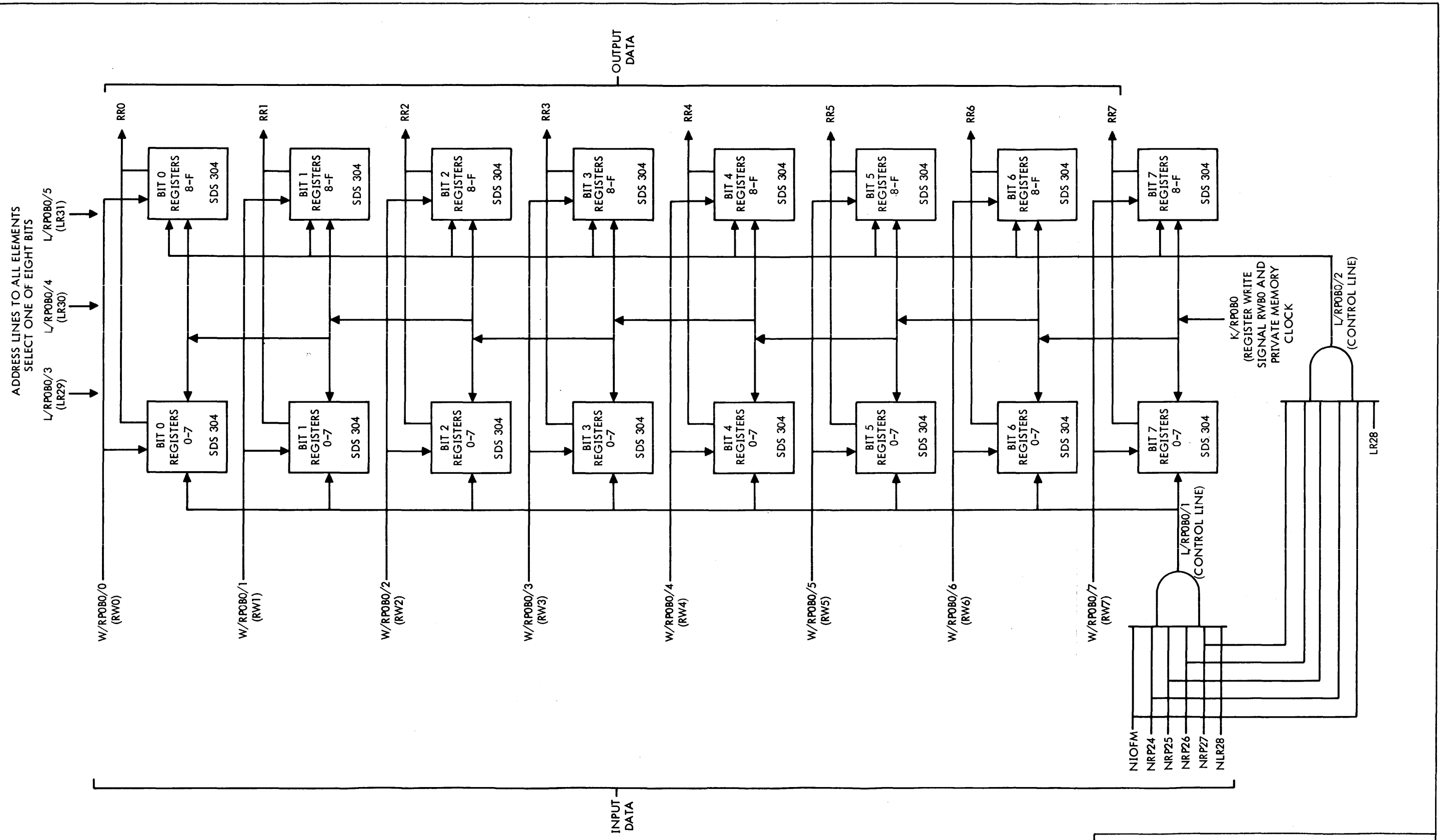


Figure 3-31. FT25 Module, Page 0, Byte 0, Simplified Diagram

901172A.74

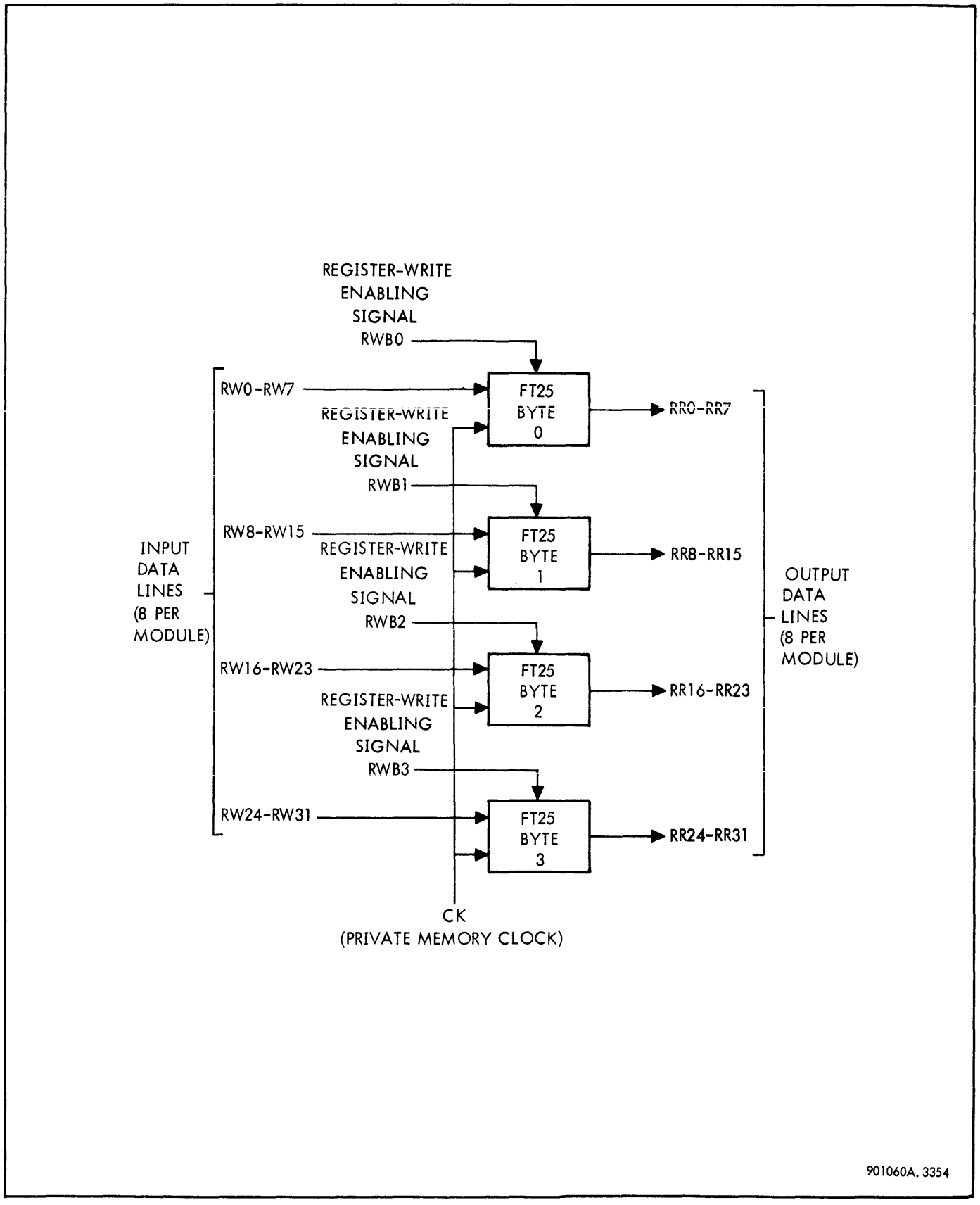


Figure 3-32. Private Memory Data Organization

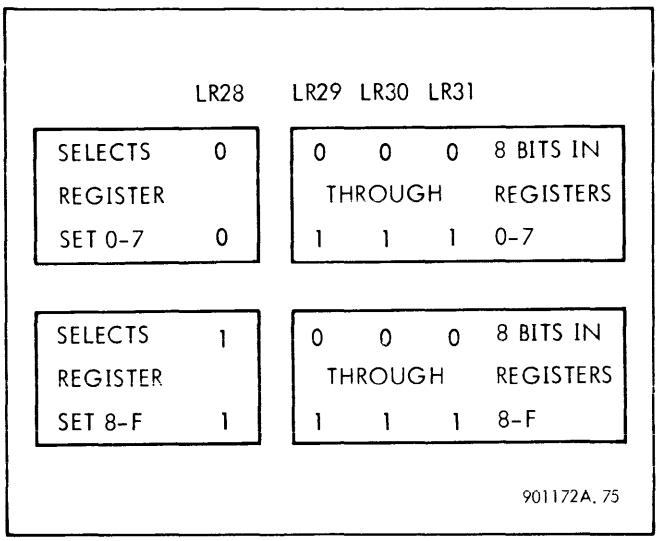


Figure 3-33. Bit Addressing on FT25 Module

Address signals LR28 through LR31 are generated, in general, from either the R-register or the D-register. The R-register contains the number of the private memory register to be addressed and is used whenever no crossover occurs. The equation is as follows:

$$LR28\text{-}LR31 = R28\text{-}R31\ (LRXR) + \ldots$$

Crossover takes place when a core memory location with an address of less than 16_{10} is addressed. During crossover, LR28 through LR30 are taken from the P-register with the equation

$$LR28\text{-}LR31 = (P28\text{-}P31)\ CROSSEN + \ldots$$

In cases of doubleword or multiple word operation, LR31 is generated from other sources either to select an odd-numbered register or to implement the function Ru1, as explained in the discussions on individual instructions. During the indexing operation, the private memory address is taken from bits 12 through 14 (the index field) of the D-register:

$$LR29\text{-}LR31 = D12\text{-}D14\ LRXD + \ldots$$

The equation for the register-write byte signal is as follows:

$$RWB0\text{-}RWB3 = RW + (MB0CRO\text{-}MB3CRO)$$

where RW is a register-write enabling signal and MB0CRO-MB3CRO are signals indicating crossover from core memory. The CK clock signal, gated with the write byte signals, is the private memory clock which comes true later than the ac clock signal.

Data signals are gated from the sum bus into the private memory as follows:

$$RW0\text{-}RW31 = S0\text{-}S31\ RWXS$$

Register Extension Chassis (REU). A register extension chassis may contain up to 16 FT25 fast access memory modules and adds from one to four blocks of additional private memory to the central processor. Since one block of private memory requires four FT25's, these modules in the register extension chassis must be added in multiples of four. Up to three register extension chassis may be added to the computer, making a maximum of 16 private memory blocks, including the four blocks in the CPU.

Additional modules in the REU provide cable drivers and receivers, terminators, chassis-selection switches and switch comparators, and logic circuits for selection and conversion of addresses and data signals. A simplified logic diagram is shown in figure 3-34.

Address, data, and control signals are transmitted from the CPU on cables and applied to cable receivers in the register extension unit. The data cables, being bidirectional, also have cable driver inputs from the REU. Clock signals are taken from the private memory clock circuit, CK/6, in the CPU. The nomenclature, functions, and decoding of the interface signals between the CPU and the REU are given in table 3-7.

Each register extension chassis is assigned an address from 01 through 11 by manually setting switches S3-1 and S2-2 on the LT26 switch comparator module in the desired configuration, with S2-2 as the least significant bit. The outputs of the switches are designated SWI1 for S3-1 and SWI2 for S2-2. A MATCH signal is generated in the selected REU by comparing the switch signals with the chassis-selection bits in the address as follows:

$$MATCH = N(SWI1\ NREU1 + NSWI1\ REU1 + SWI2\ NREU2 + NSWI2\ REU2)$$

This MATCH signal is applied to an AND gate containing another input, NREUZ, indicating that page 0 is not being addressed, and the AND gate output, REUSEL (register extension unit select) is connected to all of the FT25 modules in the selected register extension unit.

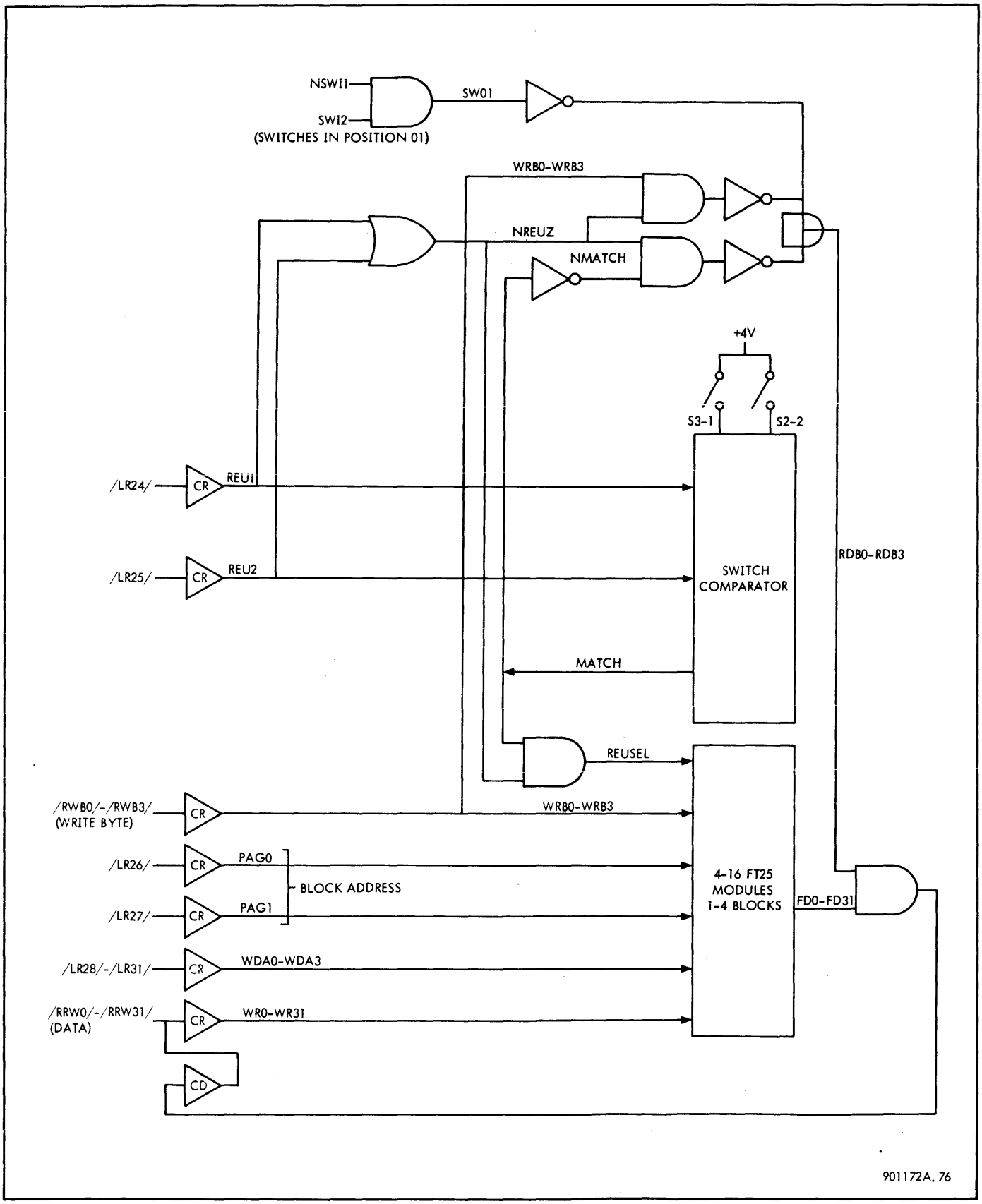
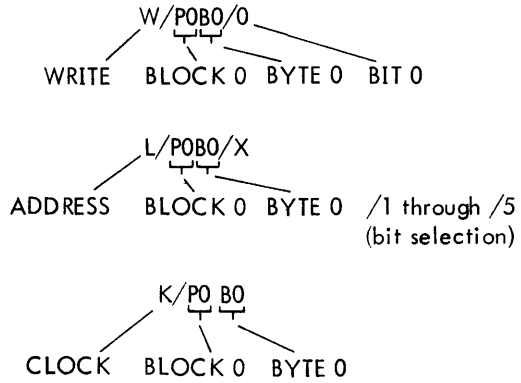


Figure 3-34. Register Extension Chassis, Simplified Logic Diagram

Table 3-7. REU Interface Signals

Input Cable	Function	Cable Receiver Output	Address Decoding
/LR24/	Address	REU1	Three chassis (in addition to one 4-module set in CPU)
/LR25/	Address	REU2	
/LR26/	Address	PAG0	Four 16-register blocks in a chassis
/LR27/	Address	PAG1	
/LR28/	Address	WDA0 or WDB0	Two sets of 8 memory elements on an FT25
/LR29/	Address	WDA1 or WDB1	Eight flip-flops in a memory element
/LR30/	Address	WDA2 or WDB2	
/LR31/	Address	WDA3 or WDB3	
/RRW0-RRW31/	Data	Cable receiver output: WR0-WR31 Cable driver input: FD0-FD31 RDB0-RDB3	
/RWB0-RWB3/	Write byte	WRB0-WRB3	

Address lines LR26 and LR27, designated PAG0 and PAG1 in the REU, are decoded to select one of four blocks in the selected REU. A simplified logic diagram of page 0 of the selected REU, a typical connection, is shown in figure 3-35. The data, address, and clock signals are interpreted as follows:



Within each register extension unit, the blocks are individually numbered 0 through 3.

Read byte signals RDB0 through RDB3 are generated when the switch settings match the chassis-selecting address lines and when the write byte signals are low, as shown in figure 3-35. The SW01 term is added to save power by turning off circuits in the unselected REU's.

Data is gated from the sum bus into the private memories in the CPU and in the REU as follows:

RW0-RW31 = S0-S31 RWXS/0 through RWXS/3

(CPU private memory)

/RRW0/-/RRW31/ = S0-S31 RRWXS/0 through RRWXS/3

(cables to REU)

3-24 Clock Logic

CLOCK GENERATOR. The clock generator in the CPU consists of three delay lines with associated gates and sense amplifiers to tap off pulses at the desired time intervals. Four basic types of clock signals are produced: a 40 nsec ac clock signal for trailing edge triggering of flip-flops throughout the CPU, a 40 nsec ac clock signal for use in the floating point unit, a 50 nsec private memory clock signal to gate information into the private memory registers, and a 50 nsec dc holding signal to clock data into the C-register buffer flip-flops. The C-register flip-flops are latching circuits and do not use an ac clock signal.

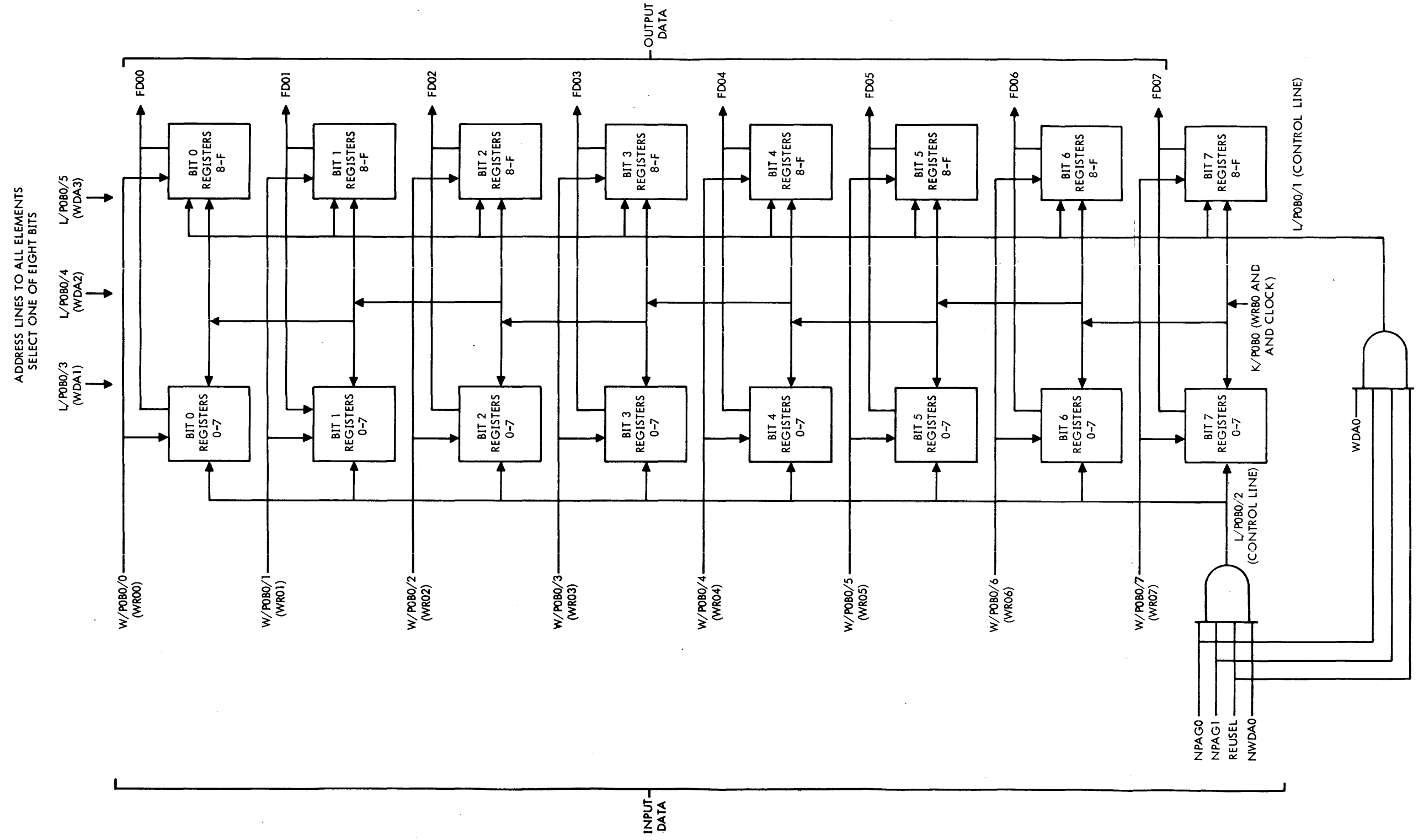


Figure 3-35. Register Extension FT25 Module, Page 0, Byte 0, Simplified Diagram

901172A.89

A simplified block diagram of the clock generator is shown in figure 3-36. Only the basic timing functions are shown. Gating, latching, pulse shaping, enabling, and other timing control functions are shown in detail later in this section. All clock signals except the first one originate with a recirculated clock input to delay line 1. A 40 nsec clock pulse is tapped off at the zero point on the delay line and is gated to clock drivers, from which the clock signals are distributed to the CPU and floating point unit flip-flops. At the 160 nsec tap, a signal is sensed and applied to the gate that produces a dc holding clock for the C-register during transfer of the sum outputs to the C-register.

From delay line 1, outputs are taken from the 180 nsec or the 210 nsec tap, depending on the state of data request flip-flop DRQ, to the input of delay line 2. Taps are taken from this delay line according to the time interval needed between one clock pulse and the next. These taps are indirectly controlled by flip-flops T8L, T11L, and signal T5EN, which is true when T8L and T11L are false. The control signal used is selected according to the number of logic operations to be performed before another clock signal is needed.

The T5, T8, and T11 outputs from delay line 2 are fed to delay line 3, from which the private memory clock signals are tapped at the zero point. A pulse tapped at 40 nsec is fed back to the input to delay line 1, and the clock cycle is started again if an enable signal is true. If the enable signal is false, the pulse is held in a latching circuit until the clock enable signal rises.

The ultimate clock intervals are affected by circuit and cable delays following the delay line taps. Each clock signal is transmitted through an 18-foot, 14-conductor coaxial cable and a 3-foot single-conductor cable. These cables introduce delays in addition to those encountered in the clock logic circuits.

Timing signals other than clock signals taken from the delay lines are applied to gates in the CPU and are discussed in the following detailed descriptions of each delay line.

Delay Line 1. A detailed logic diagram of the circuits associated with delay line 1 is shown in figure 3-37.

The first pulse is started down the delay line by force clock signal FORCL if force clock enable signal FORCLEN is true. Signal FORCL goes true as a result of pressing the CPU RESET button on the processor control panel. Enable signal FORCLEN is true until the pulse reaches the 60 nsec point. The equation for buffer flip-flop FORCLEN is as follows:

$$\text{NFORCLEN} = \text{FORCL NFORCLEN} + \text{DL1/060S FORCL}$$

Subsequent inputs to delay line 1 are provided by recirculation feedback signal ACCLG/1, from delay line 3, or by ACCLG, the output of a latching circuit set by ACCLG/1. Clock enable signal CLEN must be true for either input to start the delay line. Since clock pulse ACCLG/1 is lost when CLEN is false, ACCLG reserves the signal for use when CLEN goes true.

The CPU and floating point clock tap is ACCL/1, which rises as the delay line is triggered and is cut off by an inverting delay line sensor at 40 nsec. This shapes the ac clock pulse to a 40 nsec width. The CPU clock signals, CL/1 through CL/12, and floating point clock signals, CLFP/1 through CLFP/12, are gated by NCROSCL, which indicates that crossover from core to private memory is not taking place. The floating point clock signals are also gated by floating point clock enable signals FPCLEN/1 and FPCLEN/2, whose equations are:

$$\text{FPCLEN/1} = \text{NIOEN NIOIN} + \text{NFPRR}$$

$$\text{FPCLEN/2} = \text{NT5EN}$$

At the 50 nsec point on delay line 1, a signal is tapped and inverted to form R/ACCLG, which when low resets clock-storing latch ACCLG. Signal DL1/060S, taken from the 60 nsec tap, is used to set the force clock buffer latch. The inverted tap at 80 nsec shapes the delay line pulse to an 80 nsec width. The 160 nsec tap provides dc holding signal DCCL/2, an input to the HOLDC gate which controls the latching of information into the C-register buffer flip-flops. Signal DCCL/2 is true only when CXS is true, indicating that sum bus information is being transferred to the C-register. This signal is shaped to a 50 nsec width by the inverted 210 nsec tap. Signal R/DPL, at the 170 nsec tap, is used to reset dead pulse latches T5DP, NT5DP/1, and T8DP, explained later under delay line 2. Signal CROSSDCL, also from the 170 nsec tap, is part of the setting logic for the CROSSD latching circuit, which disables the ac clock during memory crossover operation.

A signal from the 180 nsec point and one from the 210 nsec point are applied to the input of delay line 2 to provide a 30 nsec variation in clock time, depending on whether data request flip-flop DRQ is set. Signal MRCL, at the 240 nsec point, is used to gate the memory request signal to core memory.

Delay Line 2. A detailed logic diagram of delay line 2 is shown in figure 3-38. As explained above, DL2 is set at 180 or 210 nsec according to the state of flip-flop DRQ. Timing signal T5 is tapped off at the zero point if T5 enable signal T5EN is true and crossover is not taking place as the result of a memory request. Signal T5 is fed to delay line 3 so that the clock interval during an instruction phase when T5EN is true is nominally 280 nsec. The T5 enable signal is true when flip-flops T8L and T11L are reset and instruction logic indicates that this clock interval is needed. Dead pulse latch T5DP prevents the T8, T11, and DCCL/1 outputs from the delay line from going true after a T5 pulse has been tapped.

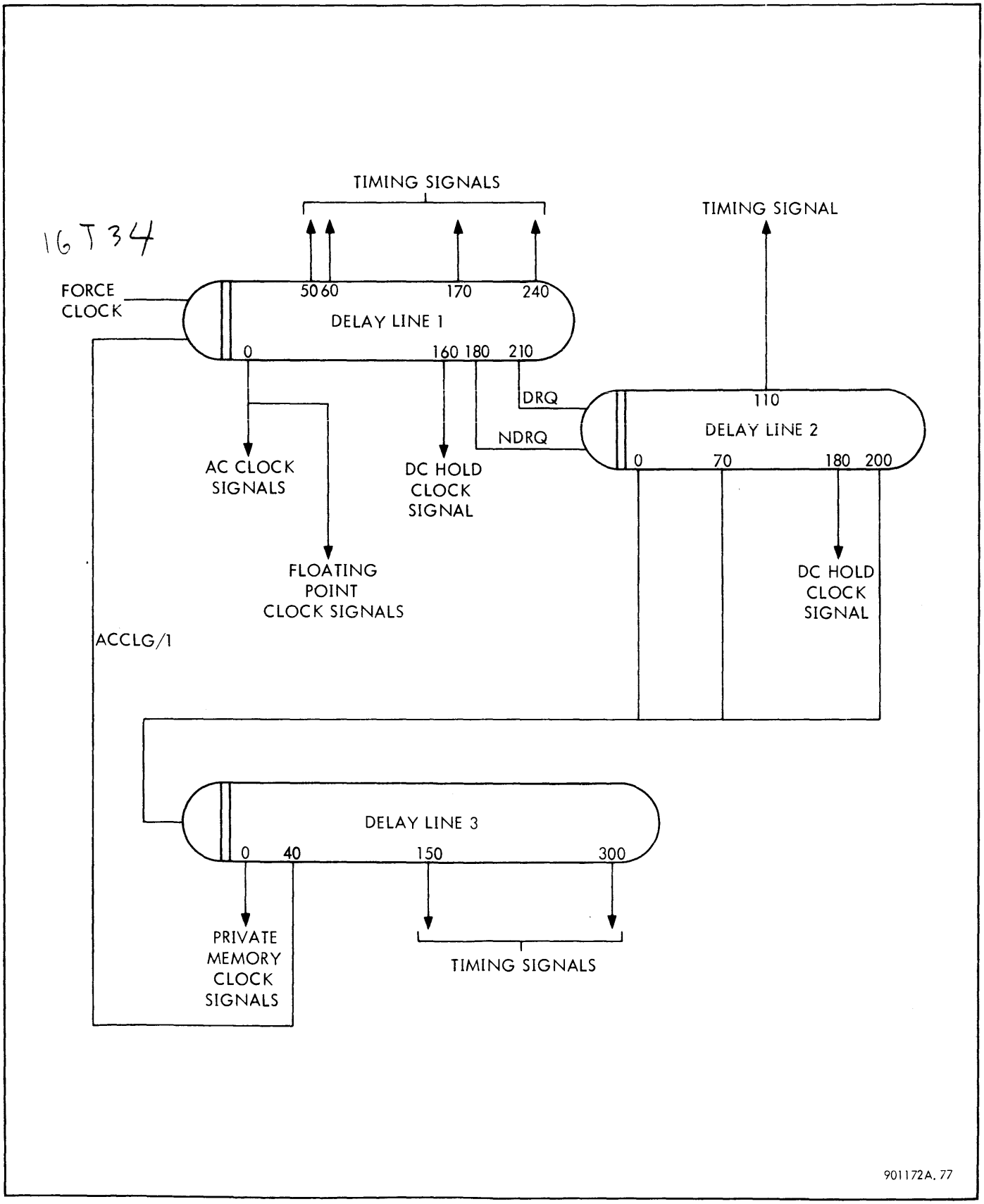


Figure 3-36. Clock Generator, Simplified Block Diagram

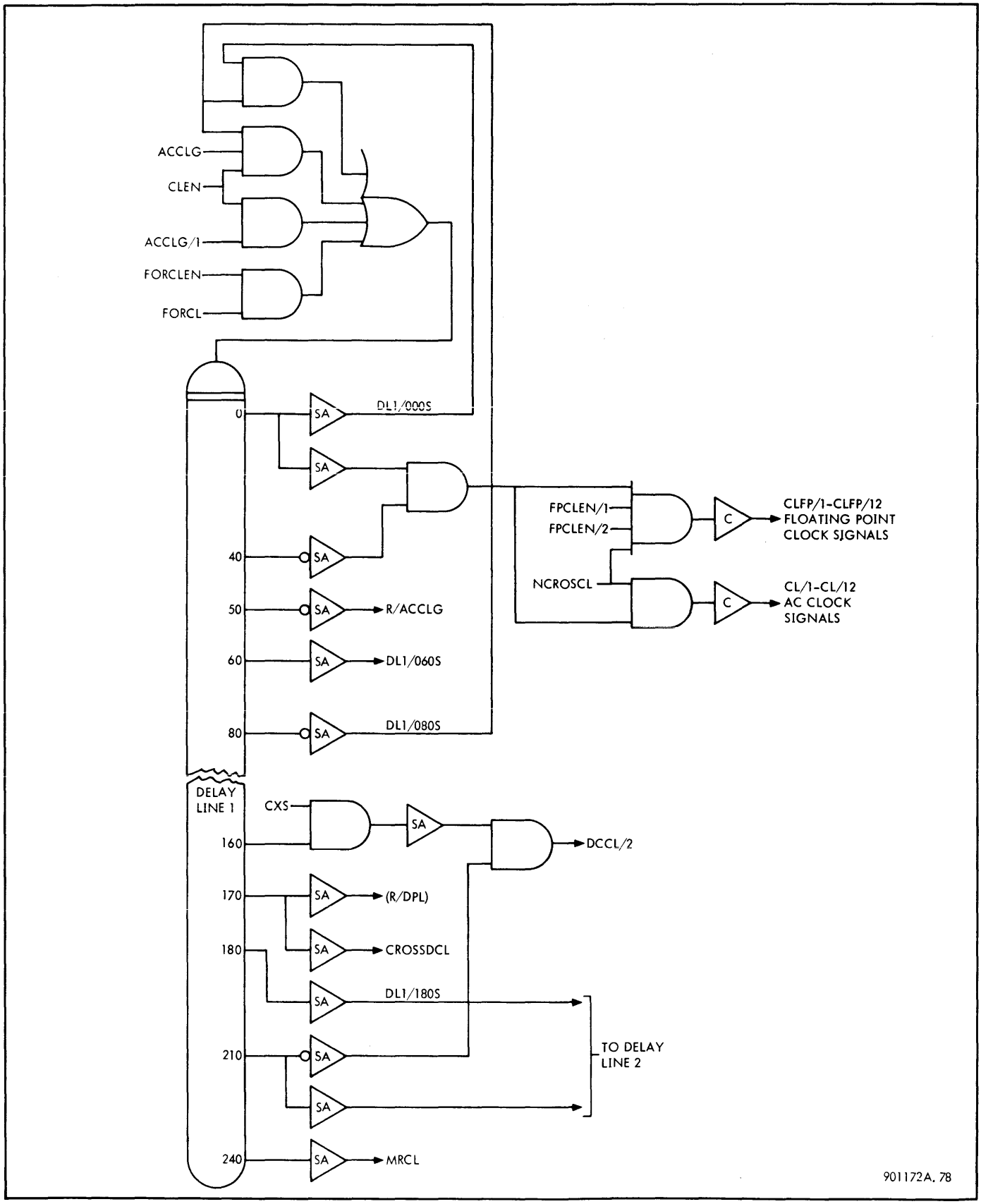


Figure 3-37. Delay Line 1, Logic Diagram

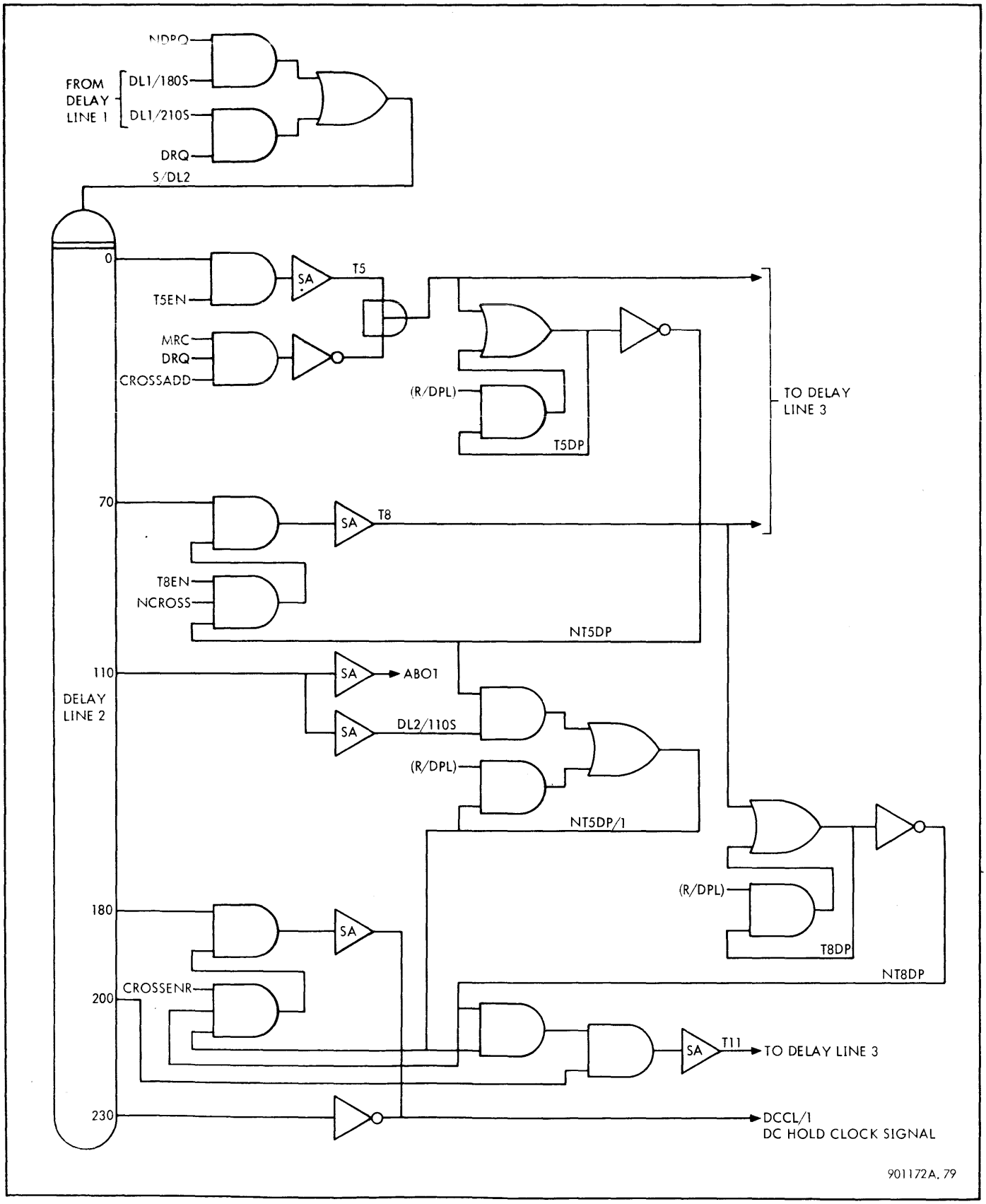


Figure 3-38. Delay Line 2, Logic Diagram

A T8 timing signal is tapped from delay line 2 at 70 nsec and fed to delay line 3 to provide a nominal 380 nsec interval between ac clock signals when crossover is not taking place. When flip-flop T8L is set at the end of an instruction phase, T5EN is driven false, and this drives T8EN true if T11L is reset and other instruction logic requests this particular clock interval. Dead pulse latch T8DP prevents the T11 and DCCL/1 outputs from the delay line from going true after a T8 pulse has been tapped. An ABOT signal from the 110 nsec tap clocks buffer flip-flop ABO/1, used when a memory access has been aborted.

At 180 nsec, dc hold signal DCCL/1 goes true and is shaped to 50 nsec by the inverting delay line sensor at the 230 nsec tap. This clock signal gates information into the C-register during crossover operation, when crossover read signal CROSSENR is true.

A T11 timing pulse is tapped from delay line 2 at 200 nsec and fed to delay line 3 to provide a nominal 500 nsec interval between ac clock signals. This pulse is allowed if no T5 or T8 pulse has been enabled. Flip-flop T11L, set during instruction phases when the following phase should be nominally 500 nsec long, is used to disable the T5 and T8 enable signals.

Delay Line 3. A detailed logic diagram of delay line 3 is shown in figure 3-39. A T5, T8, or T11 timing pulse starts a pulse down delay line 3. This pulse is shaped to 80 nsec by an inverting delay line sensor at 80 nsec. At the zero point on the delay line, FMCL/1 is tapped off and is shaped to 50 nsec by an inverting delay line sensor at 50 nsec. Signal FMCL/1 is applied to clock drivers to produce private memory clock signals CK/1 through CK/12, used to clock information into the private memory registers. At 40 nsec, clock generation signal ACCLG/1 is tapped off and fed back to the input of delay line 1 to start another clock pulse cycle if clock enable signal CLEN is true. The pulse from delay line 3 is shaped to 40 nsec by an inverting delay line sensor at the 80 nsec tap. Signal ACCLG/1 is applied to a buffer latch, where the pulse is stored as ACCLG in case CLEN is false. Signal ACCLG is gated into delay line 1 by signal CLEN. The ACCLG latch is reset by signal (R/ACCLG) from delay line 1. A signal identified as (NAH AHCL) is tapped at 150 nsec if there is no address here signal from core memory. Memory address not here clock ADNHC is tapped at 300 nsec from delay line 3.

Clock Enable Signal. As explained above, recirculated clock pulse ACCLG/1 sets delay line 1 only if clock enable signal CLEN is true. This signal is the output of a six-input AND gate, and all of the inputs must be true in order to generate signal CLEN.

To illustrate the functions of the clock-enabling gates, the equation for CLEN is divided into sections in figure 3-40. The function of each section is described separately.

Gate 1 disables the ac clock signal if none of the following conditions exist:

a. Data request flip-flop DRQ reset

b. DRQ set and data release signal DR received

c. DRQ set and data release latch DR/1 set

d. DRQ set, no memory request sent to memory, and CPU RESET switch not pressed

An example of the ac clock inhibiting logic during a memory cycle is shown in figure 3-41, using a full write store operation as an example. On the trailing edge of the clock signal, flip-flops MBXS, MRQ, MRC, and DRQ are set. As soon as DRQ is set, clock enable signal CLEN is driven low by gate 1. Memory request clock signal MRCL is tapped from delay line 1 240 nsec later, and /MQC/ is sent to memory. The ac clock generate latch, ACCLG, is set at 50 nsec on delay line 3 to store the clock pulse while CLEN is low. Data release signal DR is received from memory after /MQC/ is sent. This re-enables CLEN at gate 1, and CLEN plus ACCLG at the input to delay line 1 starts another pulse down the delay line.

A logic diagram of data release latch DR/1 is shown in figure 3-42. The latch is set by one of the following conditions:

a. Address here signal not received from core memory at 150 nsec on delay line 3

b. Effective address less than 16 (crossover), memory request and data request made, and T11 clock signal

c. Memory request made and data release signal received from memory

The latch is held by feedback to the AND gate containing DRQAC and NRESET/F. Signal NRESET/F is dropped by pressing the CPU RESET switch, thereby resetting the data release latch. The latch is also reset at the first clock signal following the setting of data request flip-flop DRQ.

Outputs from the DR/1 latch are also used in the circuits that enable or disable the ac clock and floating point clock outputs from delay line 1 during crossover operation.

Gates 2 and 3 disable the CPU clock signal during parts of the I/O operation. If no input or output is taking place, request service clock enable signal RSCLEN is false, holding the output of gate 2 true. During an I/O instruction, RSCLEN goes true, disabling the NRSCLEN gate. In this case, since request strobe acknowledge signal RSA has not been generated, (RSCLEN NRSA) is true and the ac clock signal is disabled until request strobe signal RS is received from the device controller. When RS is received, clock enable signal CLEN is again generated, and the resulting clock signal resets flip-flop RSCLEN.

At the end of an I/O operation, when the last byte is being transferred, flip-flop RSACLEN is set as the result of a

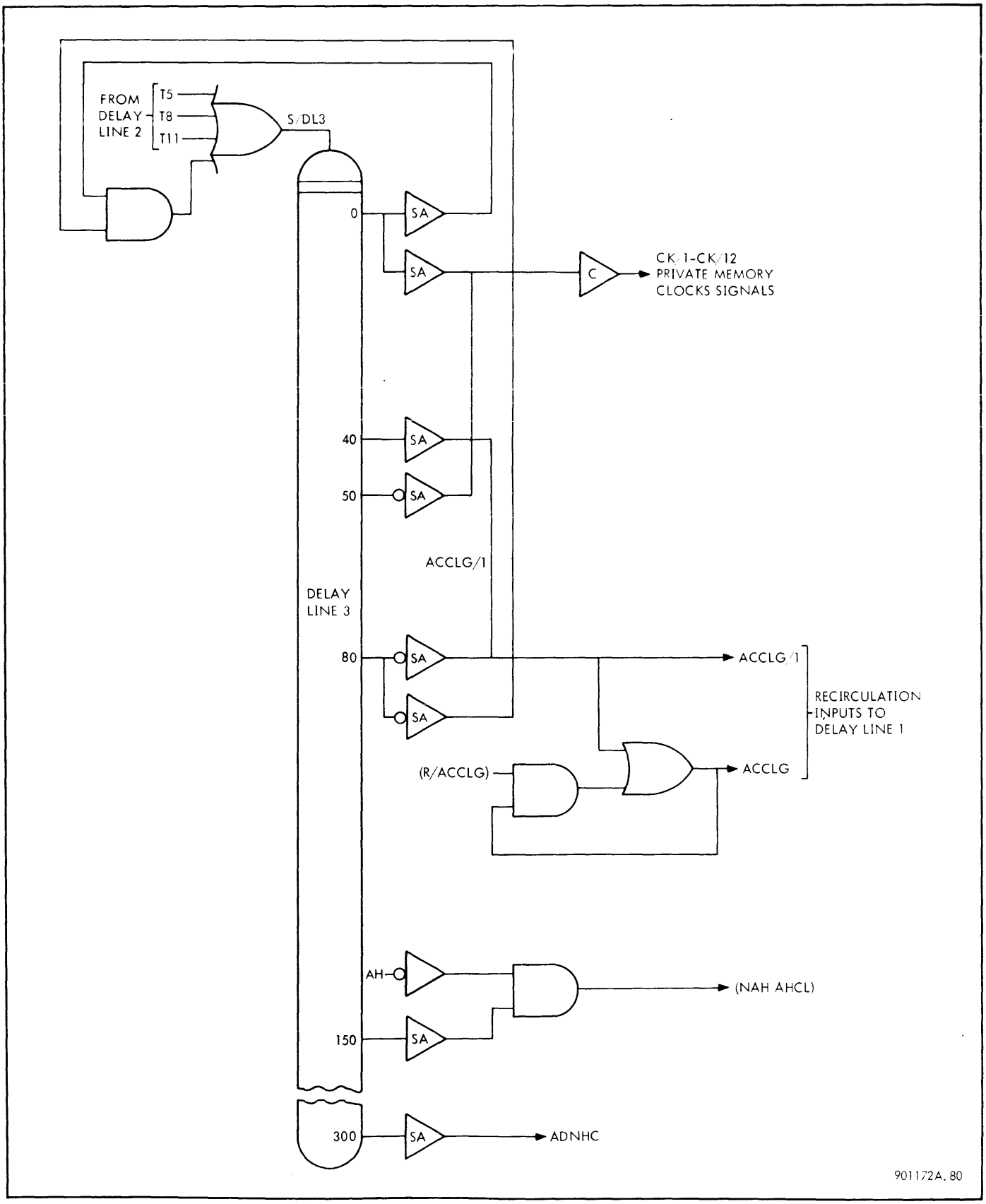


Figure 3-39. Delay Line 3, Logic Diagram

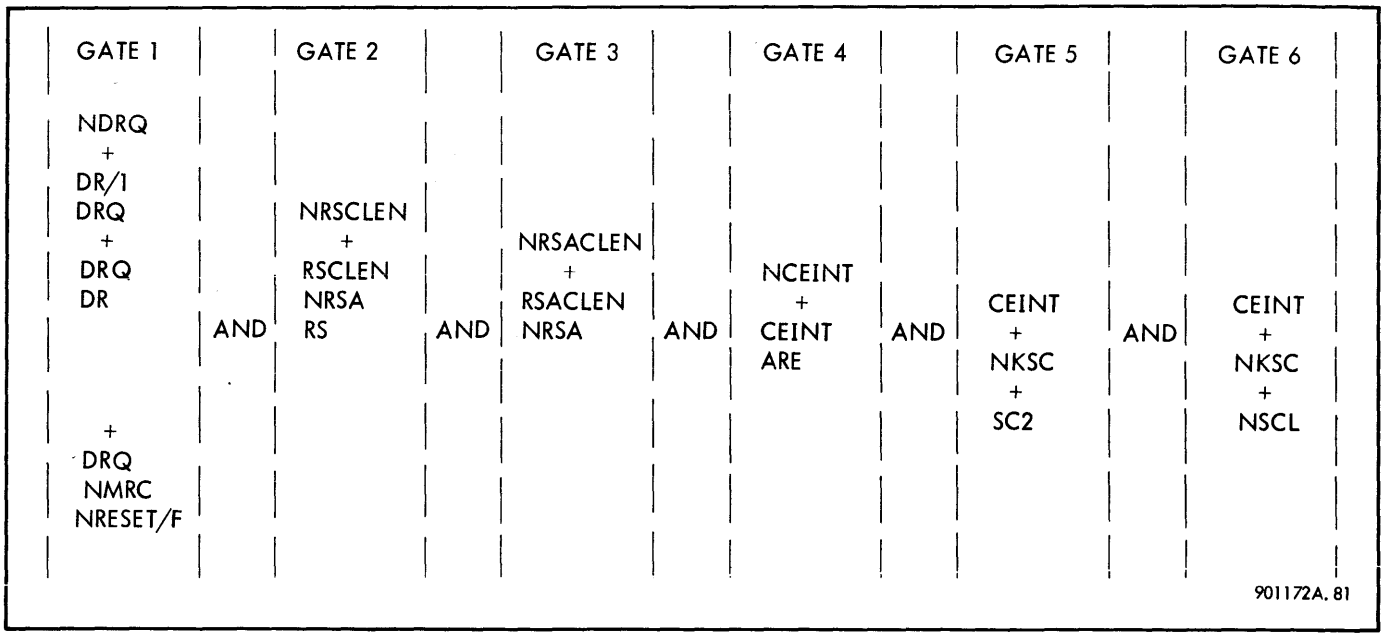


Figure 3-40. Clock Enabling Gates

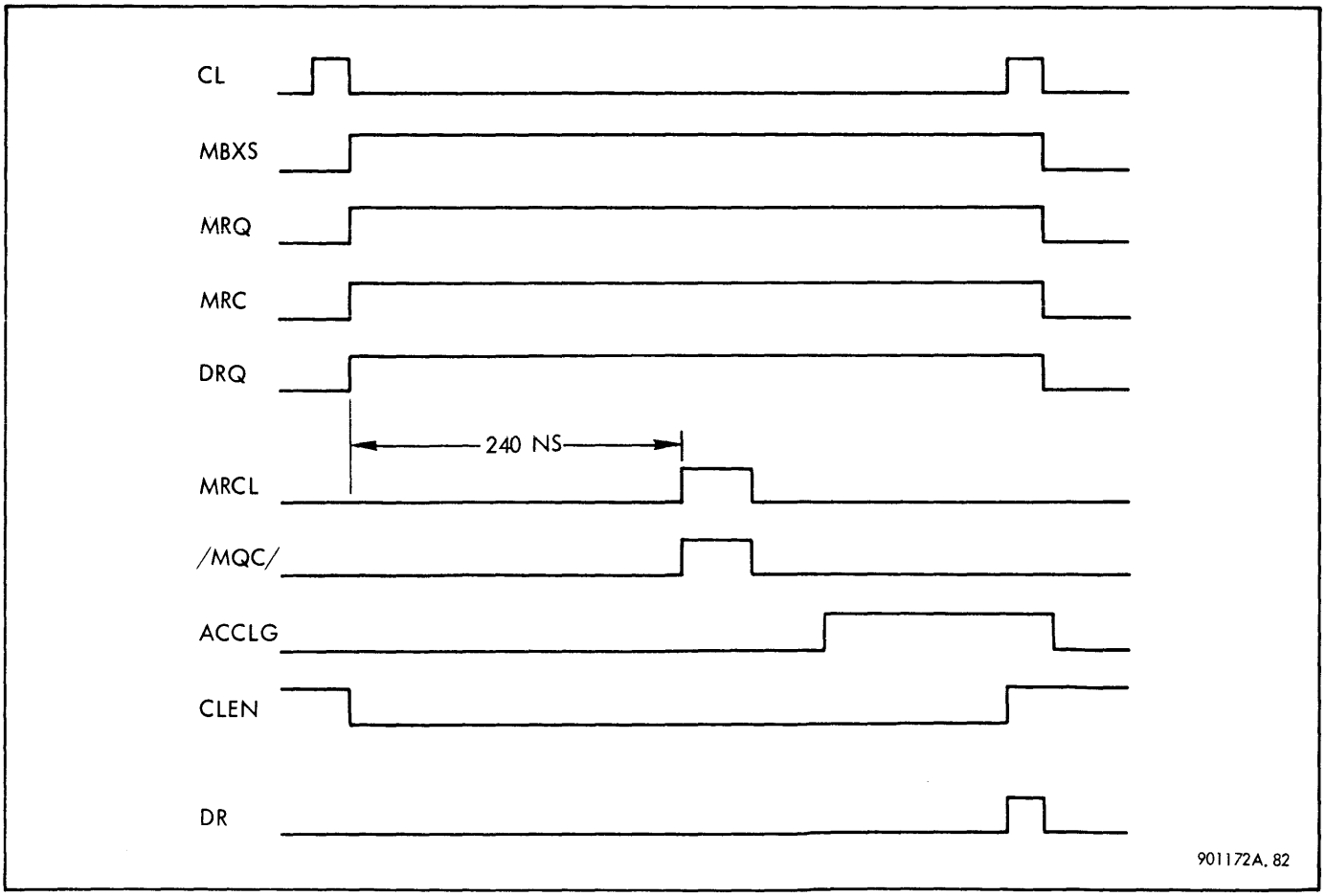


Figure 3-41. Store Operation Timing Diagram

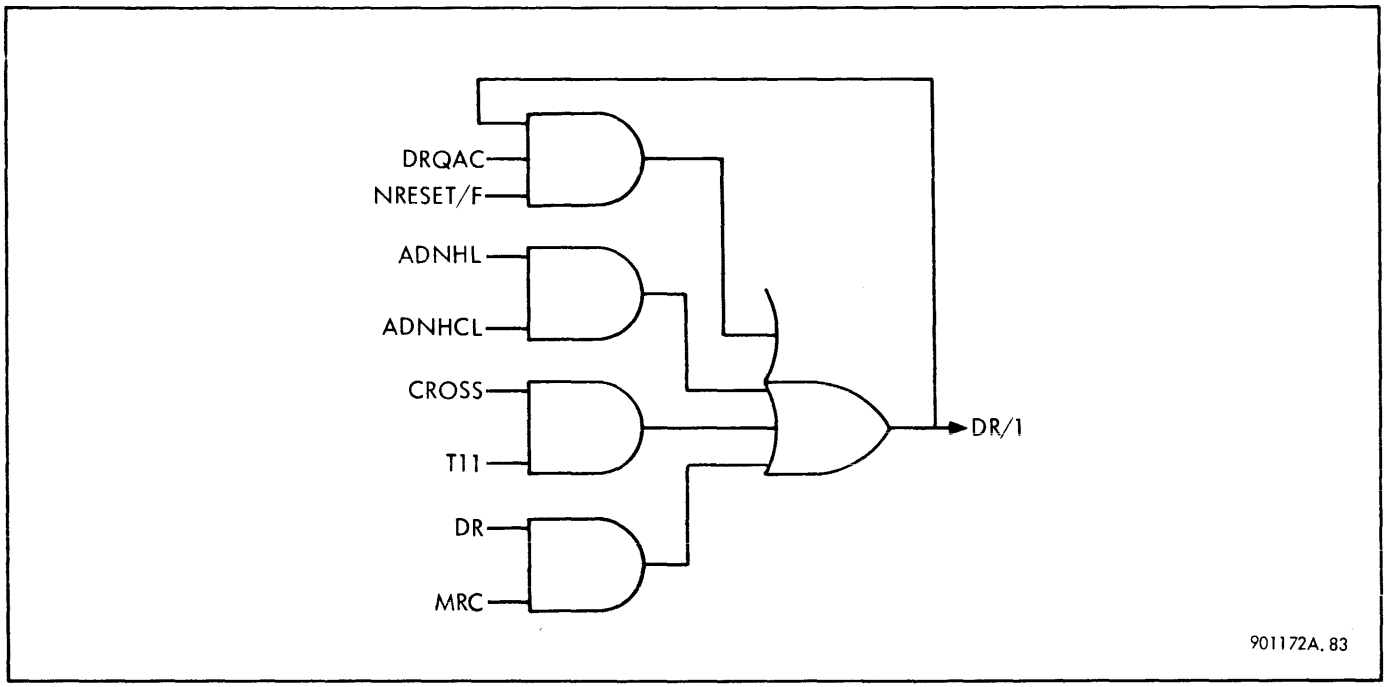


Figure 3-42. Data Release Latch, Logic Diagram

terminal order. This disables the NRSACLEN gate, and the clock signal is inhibited. Since RSA is set at this time, the clock signal is inhibited until the fall of signal RS dc resets flip-flop RSACLEN. At this time, CLEN is developed, and a clock is generated.

Gate 4 inhibits the clock signal during interrupts and watchdog timer runout. During watchdog timer runout, CEINT ensures that a clock has not just been sent down the delay line. During interrupt processing, CEINT inhibits the clock signal until action response signal ARE is received from the interrupt logic.

Gates 5 and 6 are used to disable the clock signal when the CPU CLOCK MODE switch goes into the center position. They also provide a temporary clock enabling signal when the CLOCK MODE switch is set to SINGLE STEP. A timing diagram of this single clock type of operation is shown in figure 3-43.

When the CLOCK MODE switch is set to the center position, signal KSC goes true and the output of gate 5 drops, driving clock enable signal CLEN low. Setting the CLOCK MODE switch to SINGLE STEP drives KC true and KSC remains true. Because flip-flop SC1 is clocked by the 1-MHz clock signal rather than by the continuous delay line clock signal enabled by CLEN, SC1 is set on the trailing edge of the following clock signal from the 1-MHz oscillator. The equations for flip-flop SC1 are as follows:

$$S/SC1 = KSC\ KC$$

$$R/SC1 = NKC/B\ SCL + NKSC$$

where NKC/B is true when the CLOCK MODE switch is in the CONT or center position and false in the single step mode. Signal SCL is the output of a buffer latch used to disable CLEN after a single step clock signal has occurred.

Flip-flop SC2 sets on the trailing edge of the next 1-MHz clock signal, and the output of gate 5 goes true, generating an ac clock signal. The equations for SC2 are as follows:

$$S/SC2 = SC1$$

$$R/SC2 = NSC1$$

Setting SC2 sets single clock buffer latch SCL when the clock enabled by CLEN is generated, according to the equation:

$$SCL = SCL\ SC2 + (SC2\ NCEINT)\ CL32P14$$

where CL32P14 is the result of a clock output from the delay lines. Signal SCL is latched by feedback as long as the switch is kept in the SINGLE STEP position, and signal CLEN is disabled by gate 6. Releasing the switch resets SC1 with NKC/B and SCL, and SC2 is reset on the following 1-MHz clock signal. Signal CLEN is now disabled by gate 5 until the switch is set in SINGLE STEP again. Resetting SC2 drops the latch holding SCL true.

Crossover Clocks. When an effective address less than 16 generates a CROSSADD signal, the ac clock signal from delay line 1 is disabled by gating ac clock output ACCL/1 with a crossover signal, NCROSCL. When NCROSCL is low, ACCL/1 does not produce any clock signals, and the

fast memory clock signals from delay line 3 are used to clock the private memory registers. The equation for NCROSCL is as follows:

NCROSCL = N(DR/1 CROSSEN)

CROSSEN = CROSSADD MRC DRQ NCROSSD

CROSSD = CROSSD (R/ACCLG)
+ CROSS CROSSDCL DR/1

OSCILLATOR CLOCK GENERATOR. A 1-MHz signal used to clock flip-flops in certain CPU circuits such as the interrupt circuits, the watchdog timer, and the single clock generator, is taken from a CT16 medium frequency oscillator module as shown in figure 3-44. A sine wave from a 2-MHz oscillator goes through a frequency divider consisting of seven flip-flops, each of which divides the frequency by two. Outputs from the 1-MHz flip-flop are distributed to the points where this frequency is needed, and the output of the 16-KHz flip-flop is connected to the ST29 time base selector to be used in the generation of real-time clock signals.

3-25 CPU Phases and Timing

PHASES. The CPU phases are variable time intervals separated by ac clock pulses from the CPU delay lines. The phases are identified as preparation phases 1 through 4 and execution phases 1 through 10. Each phase is entered by setting one of the phase flip-flops; PRE1 through PRE4 and PH1 through PH10. The setting logic for the phase flip-flops is determined by the type of instruction being executed, the previous phase in the phase sequence, and certain conditions peculiar to the individual instruction.

The length of a phase is determined by the time that a clock signal is sensed and gated from one of the CPU delay lines. During each phase, the length of the following phase is established by resetting flip-flops NT8L or NT11L, or by allowing both of these flip-flops to remain set. If neither NT8L nor NT11L is reset, the clock interval is set by T5EN, an enable signal that can be true only when NT8L and NT11L are true. Another time element is introduced by the presence of a memory request with data request flip-flop DRQ set. In this case, the clock signal is delayed until a data release signal is received from core memory.

Ac clock signal generation is described in the section on clock logic. The phase flip-flop setting logic for each phase is explained in the phase sequence charts included with the instruction descriptions.

3-26 Real-Time Clock

The real-time clock signals are generated in a frequency divider circuit on an ST29 time base selector module. Outputs from the frequency divider are switched on a ST14 toggle switch module to four clock pulse flip-flops on the time base selector. Outputs from these flip-flops are applied to the interrupt circuits on the appropriate LT16

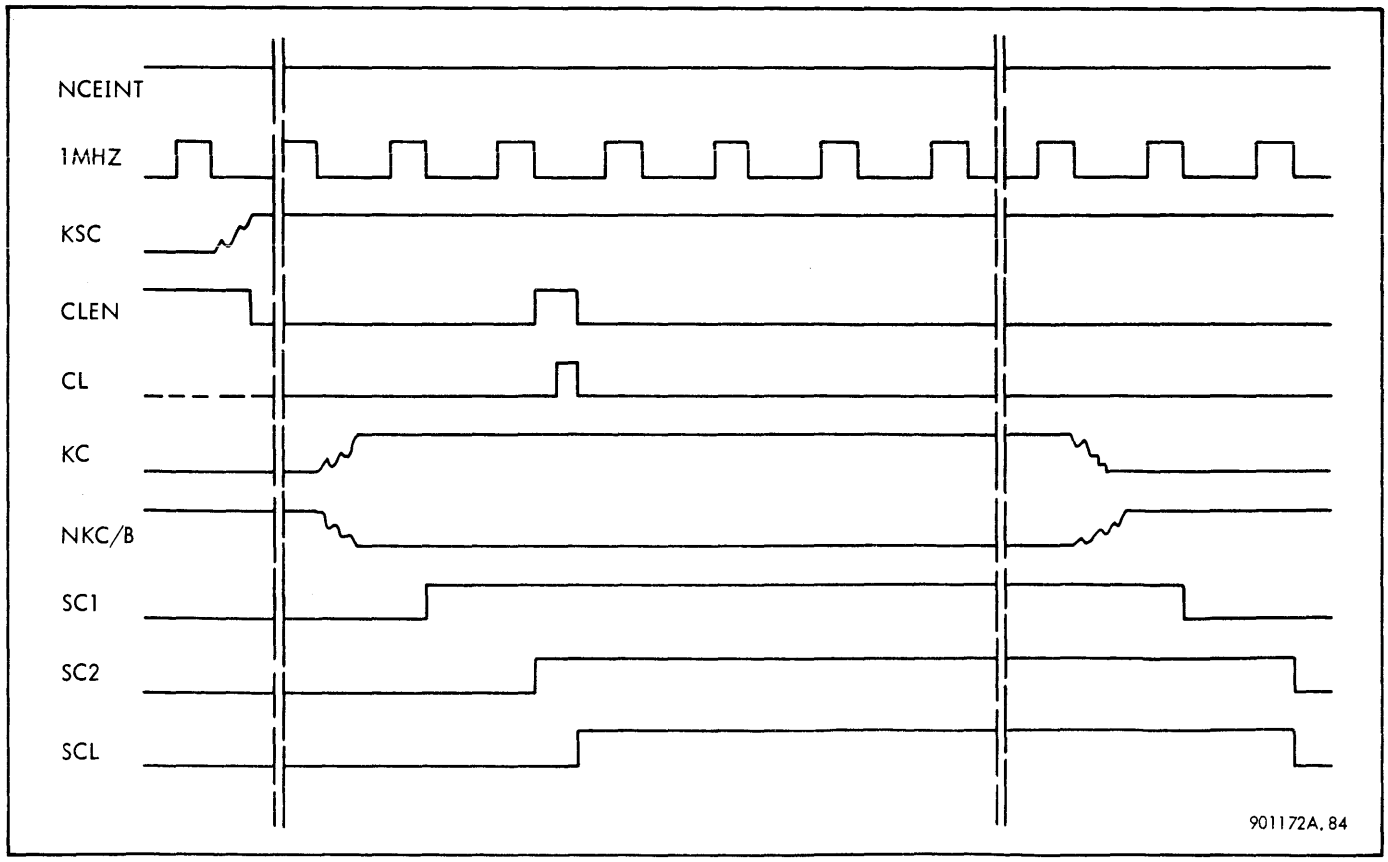


Figure 3-43. Single Clock Generation

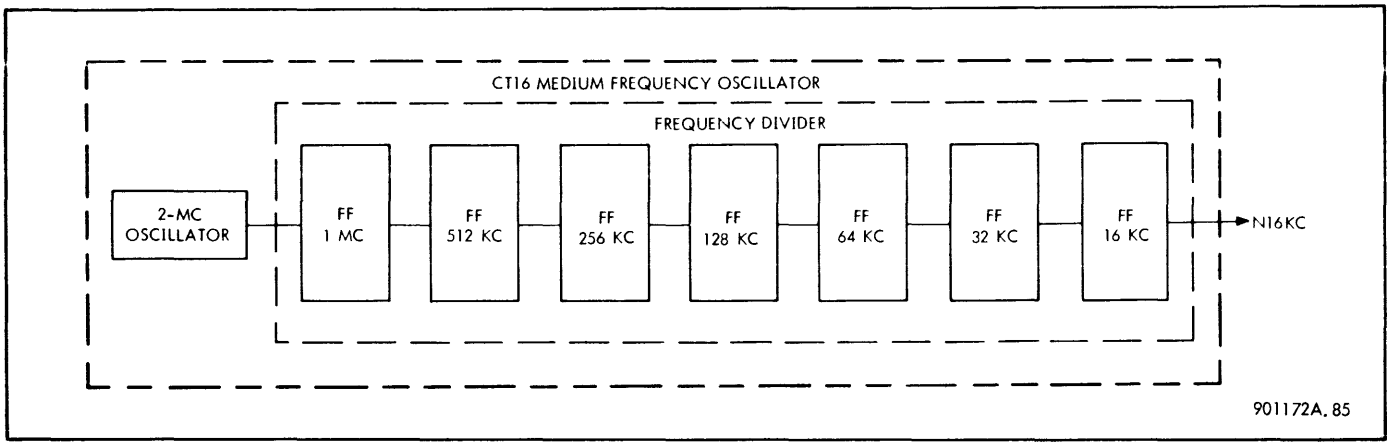


Figure 3-44. Oscillator Clock Generator, Block Diagram

priority interrupt modules. A simplified diagram of the real-time clock circuits is shown in figure 3-45.

A 16-kHz signal from the CT16 medium frequency oscillator described under Oscillator Clock Generator is applied to the input of the first of five frequency divider flip-flops on the time base selector. The frequency is divided by two each time it goes through a frequency divider flip-flop, so that the five frequencies are 8 kHz, 4 kHz, 2 kHz, 1 kHz, and 500 Hz. The outputs of the 8 kHz, 2 kHz, and 500 Hz flip-flops are connected through switches to flip-flops CPUL1, CPUL2, and CPUL3, respectively, as shown in the figure. Flip-flop CPUL4 is clocked by the 500 Hz signal, unswitched.

Each clock pulse flip-flop is connected to a group of four switches in series. These switch groups are switches 15-14-13-12 for flip-flop CPUL1, switches 10-9-8-7 for flip-flop CPUL2, and switches 5-4-3-2 for flip-flop CPUL3. When one switch in a group is in the up position and the other three switches are in the down position, the frequency connected to the up switch clocks its corresponding clock pulse flip-flop. The inputs designated RTC are optional frequencies from sources external to the CPU. The line frequency of 50 or 60 Hz may be used at any of these inputs.

The outputs of flip-flops CPUL3 and CPUL4 are taken to the priority interrupt modules used to process the standard counter 3 count pulse and counter 4 count pulse interrupts. The outputs of flip-flops CPUL1 and CPUL2 are used only if the optional counter 1 count pulse and counter 2 count pulse interrupt levels are included in the CPU. The events that occur after a count pulse signal enters the interrupt circuits are described in the section on interrupts.

3-27 Watchdog Timer

The watchdog timer is a 6-bit flip-flop binary counter clocked by the 1-MHz clock signal. The counter is set at interruptible points in the program and counts up by ones to 42, thereby allowing 42 μsec before runout. If the timer runs out before another interruptible point is reached, a trap sequence is entered.

A logic diagram of the watchdog timer control circuits is shown in figure 3-46. The counter is started by loading it with ones at one of the following interruptible points:

a. At the first ac clock pulse after interrupt enable signal IEN goes true.

b. When flip-flop PH10 is set to start the final execution phase of an instruction.

c. At the start of phase 8 (PH8) of a move to memory control instruction if the last control image word has not been loaded.

d. During I/O phase 1 (PH1) of an I/O operation if switch 13 is set.

Any one of these conditions sets flip-flop WDTRAC, and WDTRAC sets all of the watchdog timer flip-flops, WCT1 through WCT6.

When the counter has counted from 111111 to 000000 and then to 42 without being restarted, timer runout has occurred, indicating that 42 microseconds have elapsed since the last interruptible point was reached. A timing diagram of watchdog timer runout is shown in figure 3-47. Flip-flop WDTA is set by WCT1, NWCT2, WCT3, and WCT5. A one at the set output of WDTA causes flip-flop WDTRAC to be set, restarting the counter. At the same time, clock inhibit flip-flop CEINT is dc set, and signal STRAP is generated from CEINT, WDTA, and WDTRAC. Signal (S/TRAP) sets flip-flop INTRAP to start a trap sequence, which takes the CPU to location X'46'.

When flip-flop CEINT is set, clock enable signal CLEN is driven low and the CPU clock is disabled. Since CEINT must be reset by a clock signal, the CPU clock must be started by another signal. In this case the signal is force clock signal FORCL:

FORCL = STRAP 1MC 2MC

Signal FORCL is one of the setting inputs to delay line 1; therefore, an ac clock signal is immediately generated. Flip-flop CEINT is reset by this clock.

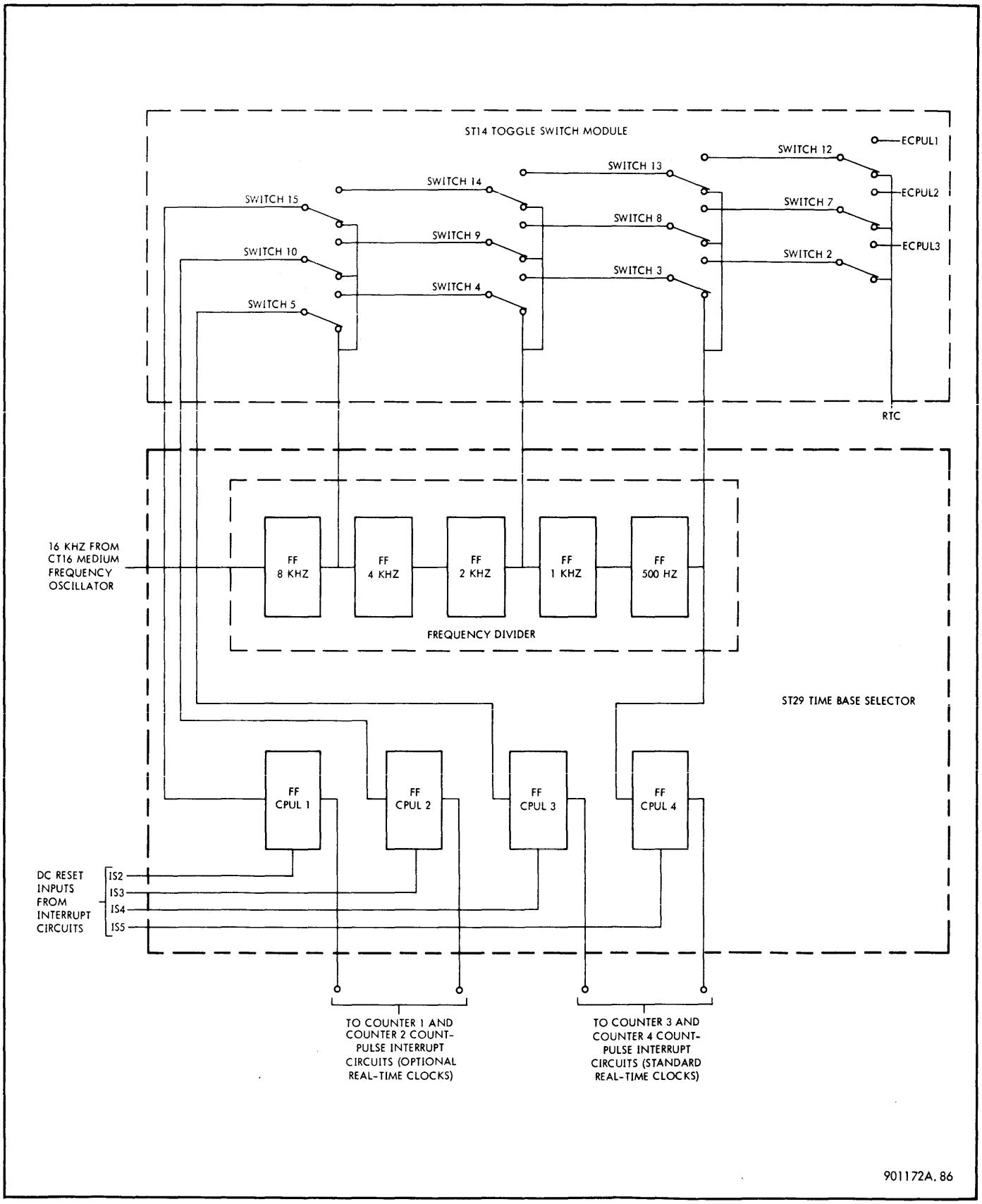


Figure 3-45. Real-Time Clock, Simplified Diagram

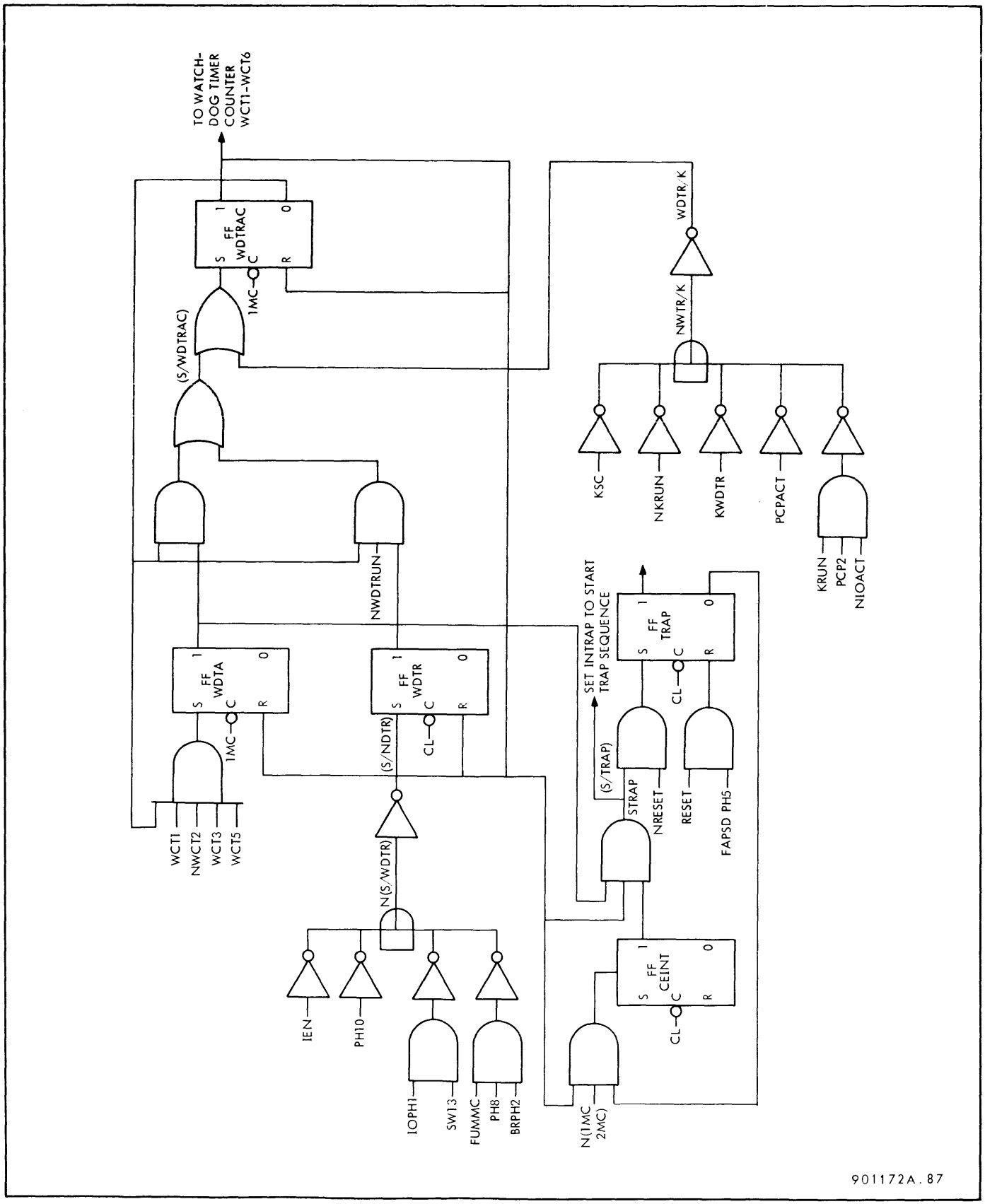


Figure 3-46. Watchdog Timer Control Circuits, Logic Diagram

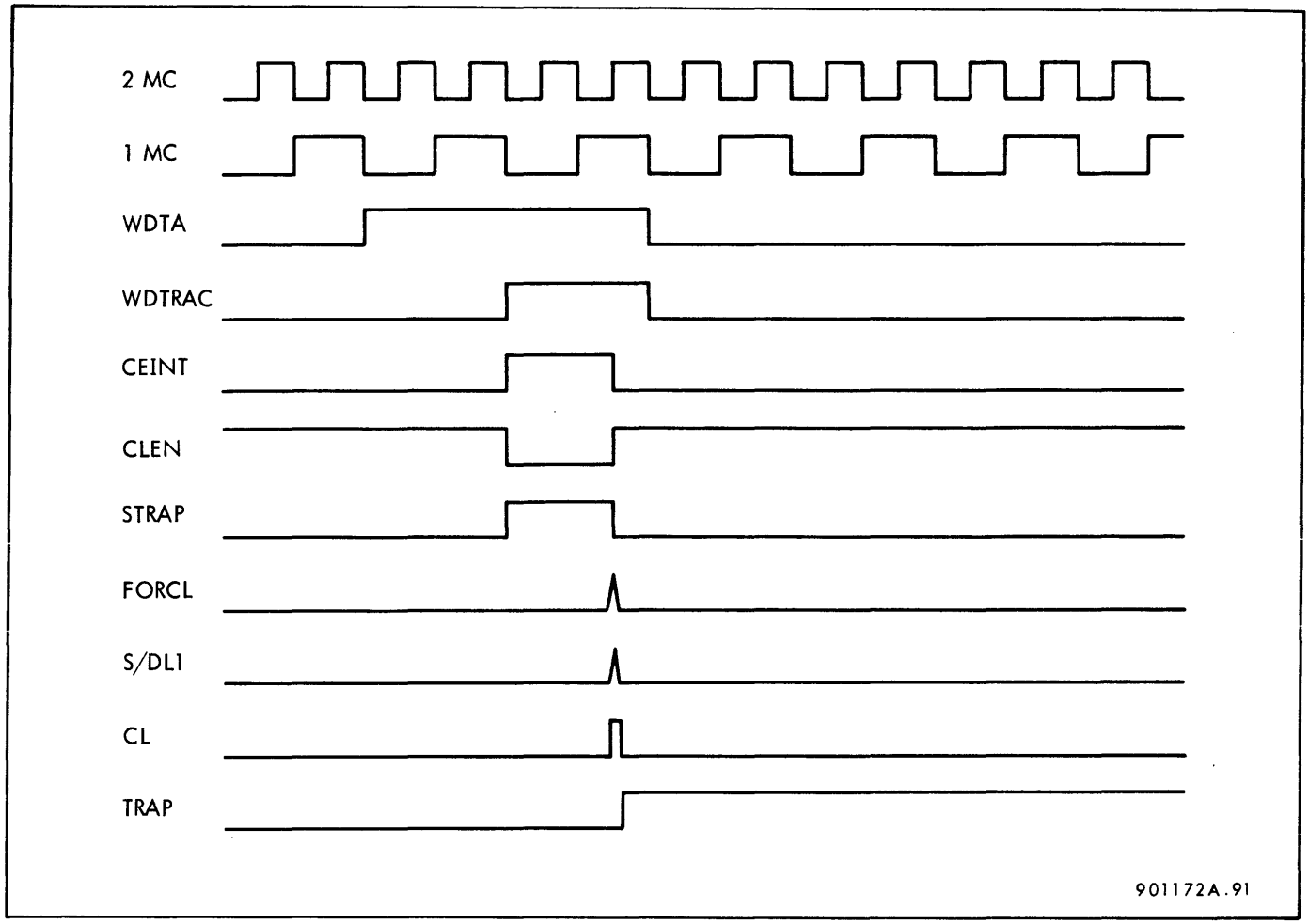


Figure 3-47. Watchdog Timer Runout, Timing Diagram

The watchdog timer is inhibited under the following conditions: (See figure 3-47.)

a. The continuous clock is disabled by placing the CLOCK MODE switch in the center position (KSC true).

b. The COMPUTE switch is set to IDLE (NKRUN true).

c. The WATCHDOG TIMER switch is placed in the OVERRIDE position (KWDTR true).

d. The CPU is in PCP phase PCP1, PCP3, PCP4, PCP5, or PCP6 (PCPACT true).

e. The COMPUTE switch is placed in RUN in PCP2 (KRUN and PCP2 true).

3-28 Memory Protection

The memory protection feature prevents alteration of specified areas of address in memory. The program is subjected to memory protection in both master and slave modes. Memory locations are protected in groups of addresses referred to as pages. Each page contains 512 memory locations.

A 2-bit write lock code for each page of core memory is stored on a set of four FT25 fast access memory modules. There are 256 of these write locks. The association of the write locks with the pages of memory addresses may be represented as shown in figure 3-48. During memory access, two flip-flops designated the write key, bits 3 and 4 of program status doubleword 2, are compared with the write lock bits for the memory page being addressed. The write locks and write key are interpreted as shown in table 3-8.

Table 3-8. Memory Protection Functions

Write Lock	Write Key	Protection
0 0	X X	Write access permitted independent of key value
X X	0 0	Write access permitted independent of lock value
0 1 through 1 1	0 1 through 1 1	Write access permitted only if lock value matches key value

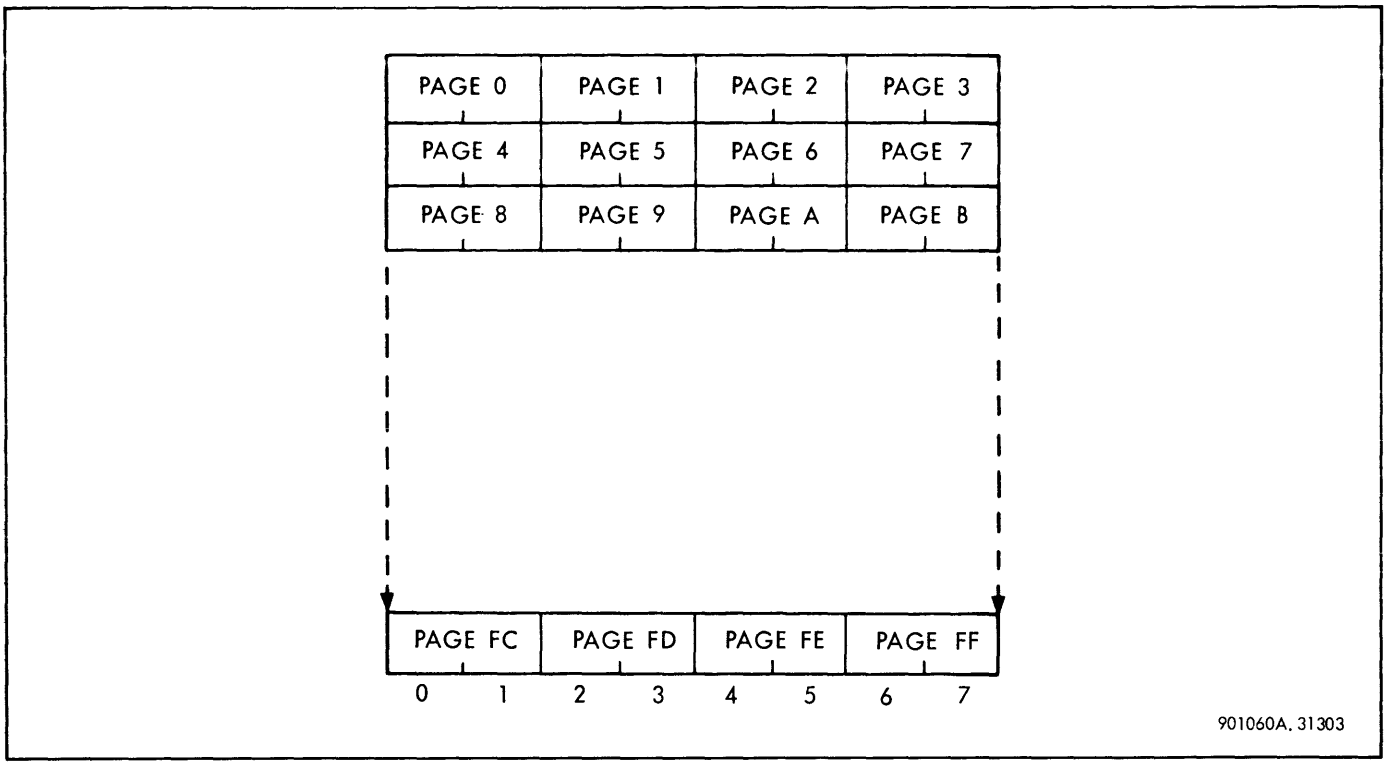


Figure 3-48. Write Lock Registers

If an instruction attempts to write into a protected memory page, the trap flip-flop is set and a trap sequence is entered.

The actual organization of the write lock bits in the SDS 304 integrated circuit memory elements on an FT25 module is shown in figure 3-49, using the first FT25 as an example. The memory elements on the left half of the diagram contain the write locks for pages 0 through 1F; those on the right half contain the write locks for pages 20 through 3F, for a total of 128 bits, or 64 write locks, on one module. Only the first, second, and last bits in each memory element are shown in the diagram.

LOADING THE WRITE LOCKS. A move to memory control instruction with a one in bit 14 transfers the write locks in core memory, referred to in the reference manual as the memory lock control image, from core memory to the memory protection registers. The instruction sequence is described in the move to memory control opcode description. The data is first placed in the C-register, then transferred to the A-register. From the A-register the memory lock control image is transferred a byte at a time by means of the sum bus onto the write lock data lines. The equations are:

$$W/LK0/1\text{-}W/LK0/7 = S0\text{-}S7$$
$$\vdots \qquad \vdots$$
$$W/LK3/1\text{-}W/LK3/7 = S0\text{-}S7$$

The data is shifted left in the A-register eight bits at a time to align it with sum bus bits 0 through 7.

The individual bits in the memory elements are selected by address signals generated from the P-register with the equations

$$L/LK0/3 = P18$$

$$L/LK0/4 = P19$$

$$L/LK0/5 = P20$$

for the first FT25 module. The modules are selected by decoding P-register bits P15 and P16, inputs to the AND gates in figure 3-49. The two halves of the module are selected by P-register bit 17 as shown in the figure.

The write lock clock signal is generated from a write lock signal LOCKW and a private memory clock signal with the equation

$$K/LK0\text{-}K/LK3 = LOCKW\ CK$$

$$S/LOCKW = FUMMC\ [PH3 + PH6\ N(BC = 1)]$$

where FUMMC is the move to memory control function.

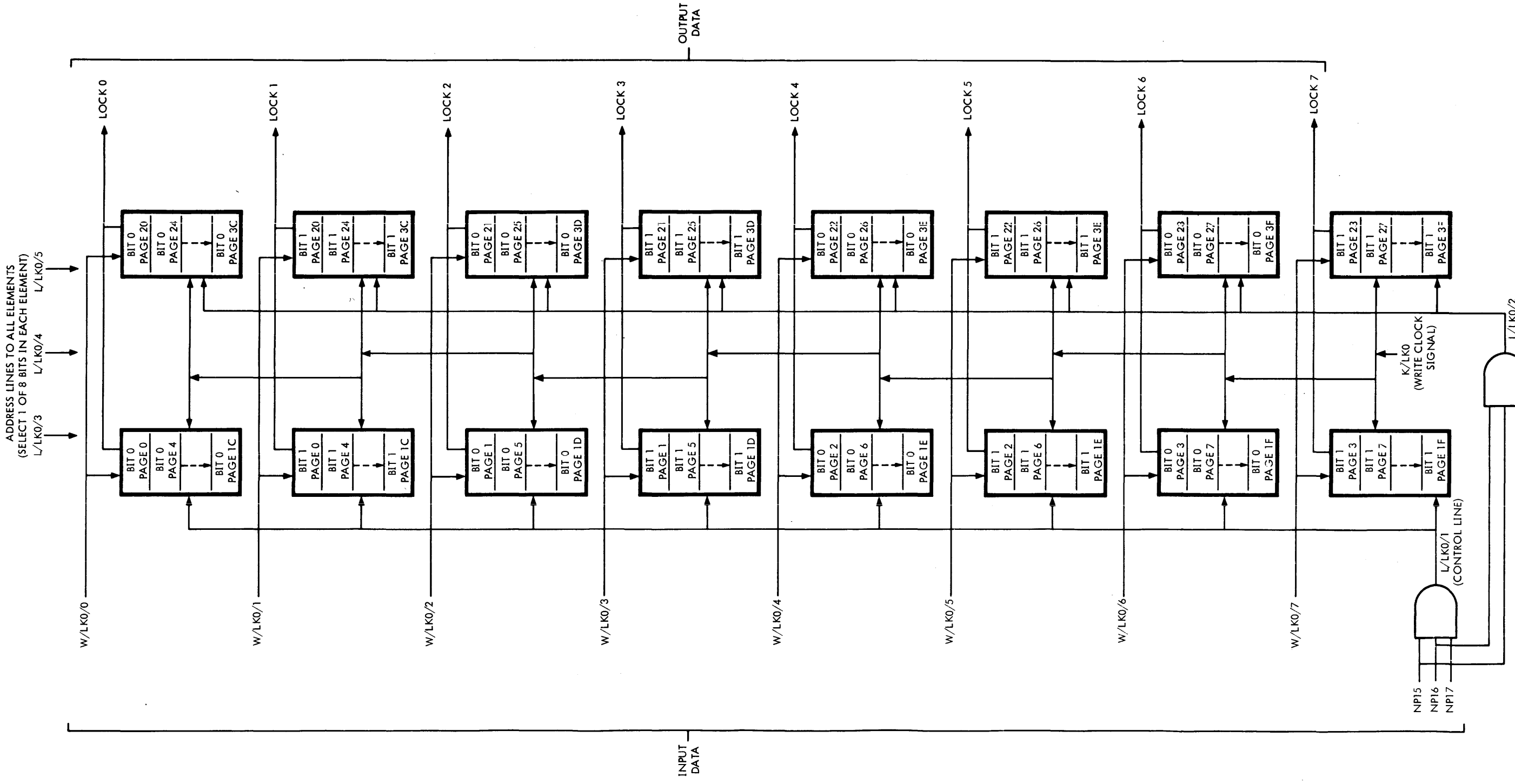


Figure 3-49. Organization of Write Lock Bits on SDS 304 Integrated Circuit

901172A.88

USING THE WRITE LOCKS. To read the memory protection register outputs, a module of write lock registers, a control line for half the module, and one bit in each of the eight memory elements are addressed by P-register outputs P15 through P20, as shown in figure 3-49. A one on the control line allows outputs LOCK0 through LOCK7 to be sensed. P-register bits P21 and P22 are decoded to select one of four 2-bit codes in the half module of write locks, as shown in figure 3-50. The outputs of the decoding circuit, LCK0 and LCK1, contain the write lock stored with the selected page address.

A mismatch of a write lock and the write key generates an ABO signal, as shown below:

Write Lock	Write Key
01	1X
10	X1
1X	0X
X1	X0

The logic equation for the ABO signal is as follows:

$$ABO = ABO/1 \; ABO/2$$

$$ABO/2 = (LCK0 \; WK1 + LCK1 \; WK0) \; N(LCK0 \; LCK1 \; WK0 \; WK1)$$

$$ABO/1 = [\, ABO/1 \; (R/DPL) + ABOT \; MBXS \; NPCPACT \; NIOACT \; NINTRAP \; N(FAIO \; PH3) \,] \; NCROSSADD$$

Signal ABO sets flip-flop TRAP, and a trap sequence is entered to take the program to trap location X'40'. An /ABOC/ signal is sent to core memory to prevent writing into the addressed location.

INHIBITING MEMORY PROTECTION. Memory protection is inhibited by disabling the signal ABO under the following conditions:

a. The CPU is in phase PCP1, PCP3, PCP4, PCP5, or PCP6 (PCPACT true).

b. Integral I/O operation in process (IOACT true).

c. Trap or interrupt in effect (INTRAP true).

d. Writing into memory location 20 during I/O operation (FAIO PH3 true).

e. Crossover in effect (CROSSADD true).

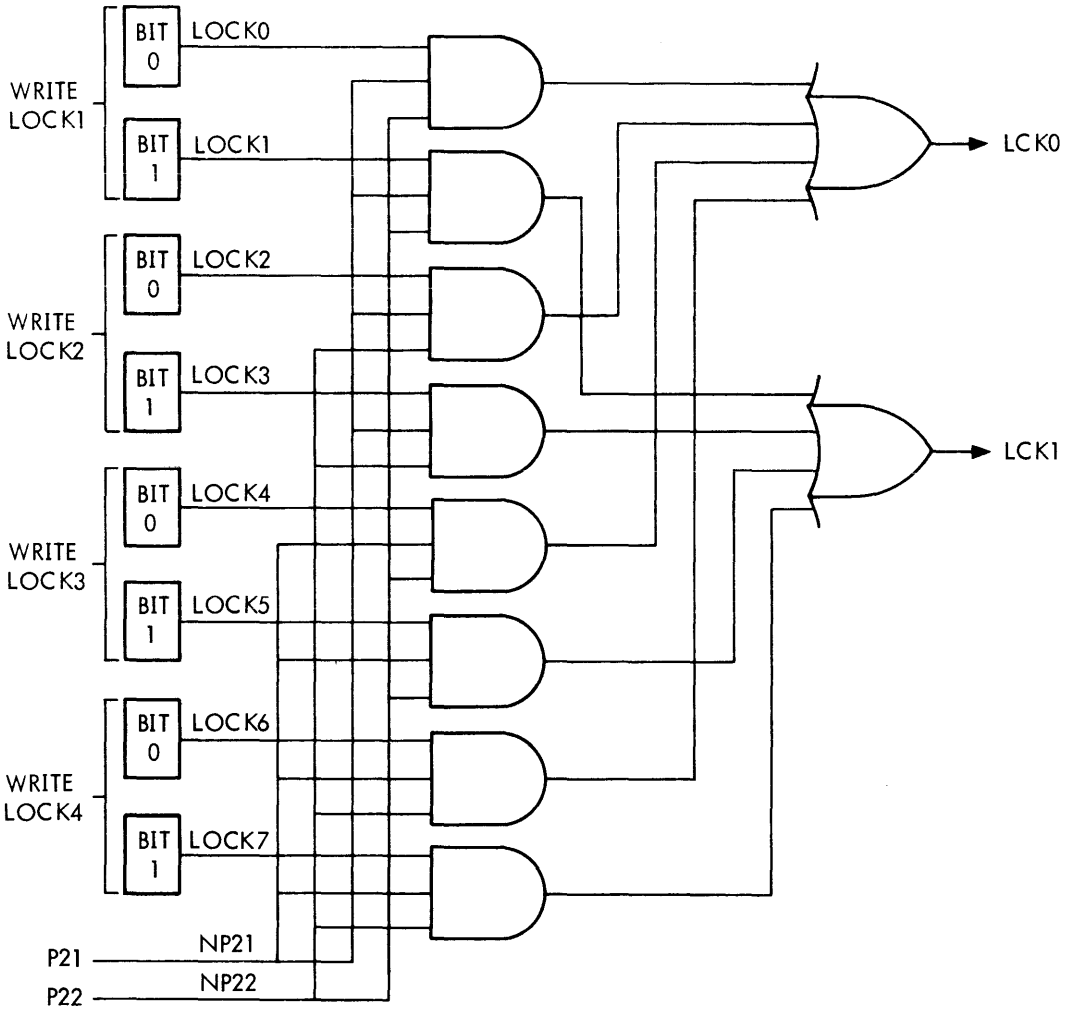


Figure 3-50. Write Lock Addressing

3-29 Traps

GENERAL. The primary use of the trap system is detection of program errors. A trap indication results from a condition such as nonexistent instruction, addressing a nonexistent memory location, watchdog timer runout, or an instruction calling for floating point operation when the option is not included in the system. Unless a power-on or power-off interrupt has been detected, a trap operation has priority over any interrupt. The detection of a trap condition causes the execution of a trap instruction in a specified location in memory. The trap instruction is executed in place of the next instruction in normal sequence.

Trap operations are also used to simulate instructions not included in the system logic. In such cases, a call instruction (CAL1, CAL2, CAL3, or CAL4) causes the program to trap to a specified location, from which a branch is made to a subroutine to carry out the desired operation.

Trap operations are controlled by interrupt/trap flip-flops INTRAP, INTRAP1, and INTRAP2, and are distinguished from interrupt operations by flip-flop TRAP, which is set during trap operations. A trap sequence may be entered in the first clock cycle following the end phase of the instruction in process, but most frequently takes place before the instruction is completed. During the trap sequence, the address of the next instruction in sequence is stored, and the trap address associated with the trap signal received is presented to memory for access. The program then branches to the memory address stored in the trap location. The conditions that result in trap operations are listed in the Sigma 5 Computer Reference Manual, along with the corresponding assigned trap locations in core memory, the time of occurrence, and special actions. An outline of the trap sequence is presented in table 3-9.

Table 3-9. Trap Sequence

Phase	Function Performed	Signals Involved			Comments
Preliminary	Set four flip-flops (TRAP, INTRAP, INTRAP1, and INTRAP2) to establish trap condition	S/TRAP	=	(S/TRAP) NRESET	True (S/TRAP) signal distinguishes trap condition from interrupt condition and is generated during preparation phases or execution phases of instructions
		(S/TRAP)	=	Indication of trap condition	
		R/TRAP	=	(R/TRAP)	
		(R/TRAP)	=	RESET + FAPSD PH5	
		S/INTRAP	=	(S/INTRAP) NRESET	
		(S/INTRAP)	=	(S/TRAP) NINTRAP ... + ...	
		R/INTRAP	=	(R/INTRAP) = (R/TRAP) + ...	
		S/INTRAP1	=	(S/INTRAP) NRESET	
		R/INTRAP1	=	(R/INTRAP1)	
		(R/INTRAP1)	=	RESET + NINTRAP2	
		S/INTRAP2	=	(S/INTRAP2) NRESET	
		(S/INTRAP2)	=	(S/INTRAP) + INTRAP1 NINTRAP2	
		R/INTRAP2	=	...	
	Inhibit reset of flip-flop NPRE1	S/NPRE1	=	N(SPRE1)	Entry into PREP phase of subsequent instruction inhibited by true (S/TRAP) signal
		(S/PRE1)	=	PRE1EN PH10 + ...	
		NPRE1EN	=	(S/TRAP) + ...	
		R/NPRE1	=	...	
	Enable CLEAR signal	CLEAR	=	(S/INTRAP) + ...	Clear selected flip-flops
	If watchdog timer runout trap, direct set CEINT	F/CEINT	=	(F/CEINT) = WDTA N(1MC 2MC) NTRAP	CPU clock inhibited while CEINT set
		R/CEINT	=	...	

(Continued)

Table 3-9. Trap Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
INTRAP1	Clock enable for reset of CEINT (required only for watchdog timer runout trap)	S/DL1 = FORCL FORCLEN + ... FORCL = STRAP 1MC 2MC + ... STRAP = WDTA WDTRAC CEINT + ... NFORCLEN = NFORCLEN FORCL + DL1/060S FORCL R/CEINT = ...	Clock pulse to delay line generated by STRAP signal and timing signals
INTRAP2	(P15-P31)⟶(B15-B31)	Bn = Pn BXP + ... = BXP/1 + ... BXP/1 = INTRAP BRP + ...	Address of next instruction in normal sequence of program
	Reset flip-flop BRP	R/BRP = INTRAP1 + ...	Indicates that address of next instruction in sequence is in the B-register
	Reset flip-flop INTRAP2	R/INTRAP2 = ...	
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = (S/DRQ/2) + ... (S/DRQ/2) = INTRAP1 TRAP + ... R/DRQ = ...	Enable complete cycle of data release if MRQ set before trap
INTRAP1	Enable clock if NMRC	CLEN = DRQ NMRC NRESET/F + ...	MRC set if MRQ set
	Sustain B15	S/B15 = (S/B15) + ... (S/B15) = B15 NBRP INTRAP1 + ... R/B15 = INTRAP1 + ...	Prevent reset of B15 if set
	Set P25	S/P25 = PXTR + ...	Store address of trap instruction (40 through 44, 46, 48 through 4B)
	Transfer (TR28-TR31)⟶(P28-P31)	PXTR = INTRAP1 NINTRAP2 TRAP R/P25 = PX + ... = INTRAP1 NINTRAP2 + ...	
	Set flip-flop MRQ	S/MRQ = (S/MRQ/2) + ... (S/MRQ/2) = INTRAP1 NINTRAP2 + ... R/MRQ = ...	Request for core memory cycle
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = (S/MRQ/2) + ... R/DRQ = ...	Inhibit transmission of CPU clock until data release from core memory
	Reset flip-flop INTRAP1	R/INTRAP1 = NINTRAP2	
	Set flip-flop INTRAP2	S/INTRAP2 = INTRAP1 NINTRAP2 + ...	

(Continued)

Table 3-9. Trap Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
INTRAP2	(MB0-MB31)⟶(C0-C31)	CXMB = /DG/	Extract addressed word and store for execution of instruction (XPSD)
	(C0-C31)⟶(D0-D31)	(S/SXD) = NINTRAP1 INTRAP2 + ...	
	(C0-C31)⟶(O0-O7)	OXC = NINTRAP1 INTRAP2 + ...	
	(C8-C11)⟶(R28-R31)	RXC = NINTRAP1 INTRAP2 + ...	
	Reset flip-flop NPRE1	S/NPRE1 = N(S/PRE1) + ...	Enable entry into PREP phase
		(S/PRE1) = NINTRAP1 INTRAP2	
		R/NPRE1 = ...	
	Reset flip-flop INTRAP2	R/INTRAP2 = ...	
	Reset flip-flop TRAP	R/TRAP = (R/TRAP)	Exit from TRAP at PH5 of XPSD instruction
		(R/TRAP) = FAPSD PH5 + ...	
	Reset flip-flop INTRAP	R/INTRAP = (R/INTRAP)	
		(R/INTRAP) = (R/TRAP) + ...	
	Trap sequence ended		

TRAP SEQUENCE. The trap sequence is illustrated in figure 3-51. When signal (S/TRAP) is true, flip-flops TRAP, INTRAP, INTRAP1, and INTRAP2 are set. At the same time, one of the codes listed in table 3-10 is stored in the trap accumulator register (TRACC1 through TRACC4), and the least significant hexadecimal digit of the trap address is stored in the trap address register (TR28 through TR31). The controlling equations are listed in the paragraphs describing trap conditions.

After the address of the next instruction in normal sequence is transferred from the P-register to the B-register, and BRP is reset, the trap circuits enter the INTRAP1 phase. The least significant hexadecimal digit of the trap address is transferred from the trap address register to the least significant flip-flops of the P-register.

S/P28 = TR28 PXTR + ...

PXTR = INTRAP1 NINTRAP2 TRAP + ...

(R/P28-R/P31) = PX + ... = INTRAP1 NINTRAP2 + ...

S/P29 = TR29 PXTR + ...

S/P30 = TR30 PXTR + ...

S/P31 = TR31 PXTR + ...

Table 3-10. Trap Codes and Address Digits

Cause of Trap	Trap Accumulator Register (TRACC1-TRACC4)	Trap Address* Register (TR28-TR31)
Abort	0001	0000
Watchdog timer runout	0000	0110
Floating point fault	0000	0100
Fixed point overflow	0000	0011
Privileged instruction	0010	0000
Nonexistent memory address	0100	0000
Not implemented	0000	0001
Illegal	1000	0000
Stack fault	0000	0010
CAL1 instruction	(R28-R31)†	1000
CAL2 instruction		1001
CAL3 instruction		1010
CAL4 instruction		1011

*Stores least significant hexadecimal digit of trap location

†Contents of (R28-R31) transferred to (TRACC1-TRACC4)

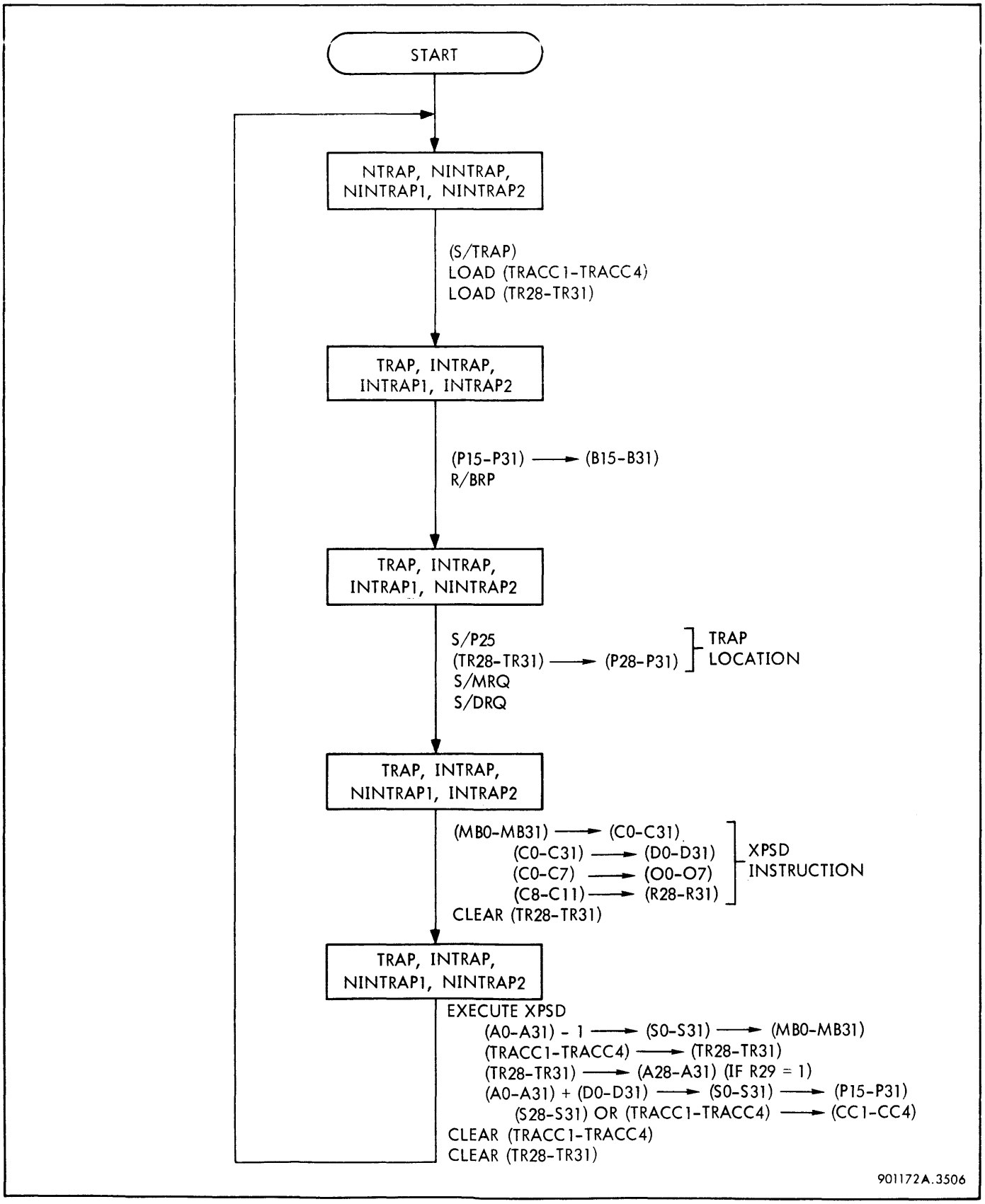


Figure 3-51. Trap Sequence, Flow Diagram

The most significant hexadecimal digit (4) is established by setting P25. Flip-flops MRQ and DRQ are set to permit access to memory during the INTRAP2 phase. The XPSD instruction stored in memory is placed in the D-, O-, and R-registers, and the trap address register is cleared.

(R/TR28-R/TR31) = (R/TR) = NINTRAP1 INTRAP2 + ...

During execution of the XPSD instruction, the 17-bit instruction address is decremented by one before storage in memory. This operation is required to reduce the address, which was previously incremented by one. The code stored in the trap accumulator register is transferred to the trap address register.

S/TR28 = NSTRAP (S/TR28) + ...
(S/TR28) = TRACC1 FAPSD PH1
S/TR29 = NSTRAP (S/TR29) + ...
(S/TR29) = TRACC2 FAPSD PH1
S/TR30 = NSTRAP (S/TR30)
(S/TR30) = TRACC3 FAPSD PH1 + ...
S/TR31 = NSTRAP (S/TR31)
(S/TR31) = TRACC4 FAPSD PH1 + ...

Signal NSTRAP is true unless a watchdog timer runout has occurred.

If bit 9 of the XPSD instruction (now in R29) is a one, the code is transferred from the trap address register to the least significant bits of the A-register.

S/A28 = TR28 AXTR + ...
AXTR = FAPSD 07 PH3 TRAP R29
S/A29 = TR29 AXTR + ...
R/A30 = TR30 AXTR + ...
S/A31 = TR31 AXTR + ...

This number is added to the contents of the D-register and stored in the P-register during PH4. The number is also retained in the trap accumulator register so that during PH4 it is merged with the existing condition code (S0 through S3) to form a new condition code.

S/CC1 = (S/CC1/3) + S0 CCXS/0 + ...
(S/CC1/3) = (S/CC1/1) + = CCXTRACC TRACC1 + ...
CCXTRACC = FAPSD 07 PH4 TRAP
CCXS/0 = PSW1XS + ... = FAPSD PH4 + ...
S/CC2 = CCXTRACC TRACC2 + S1 CCXS/0 + ...
S/CC3 = CCXTRACC TRACC3 + S1 CCXS/0 + ...
S/CC4 = CCXTRACC TRACC4 + S3 CCXS/0 + ...
(R/CC1-R/CC4) = CCXS/0 + ...

During phase 5 of the XPSD instruction, the trap accumulator register and the trap address register are cleared.

(R/TRACC1-R/TRACC4) = (R/TRACC) = FAPSD PH5 + ...
(R/TR28-R/TR31) = (R/TR) = (R/TRACC/1) + ... = FAPSD PH5

The trap sequence is terminated at the same time.

TRAP CONDITIONS. The conditions for entering the trap sequence are represented by the inputs to signal (S/TRAP).

(S/TRAP) = ABO (Abort)
+ STRAP (Watchdog timer runout)
+ FAFL NRW ENDE NINTRAP (Floating point)
+ FACAL PH1 (Call)
+ ENDE AM CC2 OVERIND (Fixed point overflow)
+ FAPRIV NMASTER PRETR NINTRAP (Privileged)
+ ADNH NIOACT (Address not here)
+ FANIMP PRETR (Not implemented)
+ FAILL PRETR (Illegal)
+ FAST PH2 SW1 NSW5 (Stack)
+ FAST PH2 SW3 NSW6 (Stack)

Figure 3-52 indicates all opcodes that might generate a true (S/TRAP) signal. Opcodes for which no operation is defined unconditionally generate a true (S/TRAP) signal. Some opcodes generate a true (S/TRAP) signal only for selected conditions. For example, an immediate instruction (FAIM) with an indirect address bit (IA) equal to one will cause a trap sequence. Conditions such as watchdog timer runout or addressing nonexistent locations are not associated with particular opcodes.

Call Instructions. The four call instructions (CAL1, CAL2, CAL3, and CAL4) cause the computer to trap to location X'48', X'49', X'4A', and X'4B', respectively.

For the CAL1 instruction (opcode 04), only TR28 is set.

S/TR28 = FACAL PH1 NTRAP NSTRAP + ...
(R/TR28-R/TR31) = (R/TR) = (S/TRAP) + ...

For the CAL2 instruction (opcode 05), TR28 and TR31 are set.

S/TR31 = NSTRAP (S/TR31)
(S/TR31) = FACAL PH1 NTRAP NSTRAP O7 + ...

For the CAL3 instruction (opcode 06), TR28 and TR30 are set.

S/TR30 = NSTRAP (S/TR30) + ...
(S/TR30) = FACAL PH1 NTRAP NSTRAP O6

For the CAL4 instruction (opcode 07), TR28, TR30, and TR31 are set, since inputs to all three flip-flops are true.

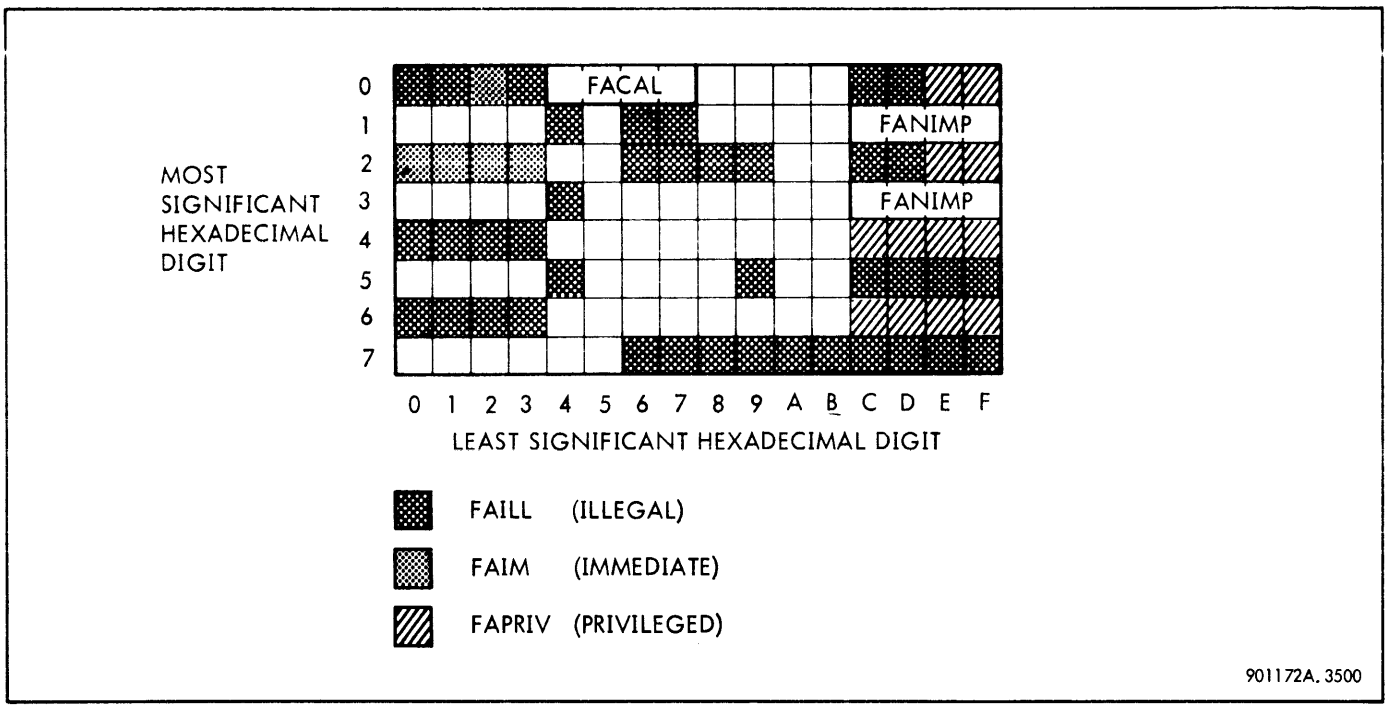


Figure 3-52. Operation Codes Resulting in Trap

The contents of R28 through R31 are stored in the trap accumulator register as the address code is stored in the trap address register.

S/TRACC1 = FACAL PH1 NTRAP NSTRAP R28 + ...

(R/TRACC1-R/TRACC4) = (R/TRACC) = (S/TRAP) + ...

S/TRACC2 = FACAL PH1 NTRAP NSTRAP R29 + ...

S/TRACC3 = FACAL PH1 NTRAP NSTRAP R30 + ...

S/TRACC4 = FACAL PH1 NTRAP NSTRAP R31 + ...

This code is transferred to the trap address register during the XPSD instruction. If bit 9 of the XPSD instruction is a one, it is added to the contents of the D-register and merged with the contents of the trap accumulator register to set condition code flip-flops CC1 through CC4.

Push-Down Stack Limit Instructions. During the execution of any stack-manipulating instruction, words are either added to or removed from the stack. In either case, the space count fields of the stack pointer doubleword are tested before moving any words. If the execution of the instruction would cause the space count to become less than zero or greater than $(2^{15}-1)$, the instruction is aborted with memory and register unchanged; then, if bit 32 (TS) of the stack pointer doubleword is zero, the CPU traps to location X'42'.

(S/TRAP) = FAST PH2 SW1 NSW5 + ...

If execution of the instruction would cause the word count to become less than zero or greater than $(2^{15}-1)$, the instruction is aborted with memory and registers unchanged; then, if bit 48 (TW) of the stack pointer doubleword is a zero, the CPU traps to location X'42'.

(S/TRAP) = FAST PH2 SW3 NSW6 + ...

FAST = FAST/A + FUMSP

FAST/A = OU0 O4 NO5 (PLW, PSW, PLM, PSM)

FUMSP = OU1 OL3 (MSP)

When a trap is caused by a stack fault, the trap accumulator register is cleared.

(R/TRACC1-R/TRACC4) = (R/TRACC) = (S/TRAP) + ...

The least significant hexadecimal digit is set to 2 by setting TR30.

S/TR30 = NSTRAP (S/TR30) + ...

(S/TR30) = FAST PH2 SW1 NSW5 + FAST PH2 SW3 NSW6 + ...

(R/TR28-R/TR31) = (R/TR) = (S/TRAP) + ...

Therefore, a stack fault causes a trap to location X'42' with a code of 0000.

Floating Point Fault. A floating point fault is detected after the operation called for by the instruction code is performed, but before any results are actually loaded in to the general registers. If no error is detected, signal (S/RW/FP) from logic in the floating-point box sets flip-flop RW.

S/RW = (S/RW/1)

(S/RW/1) = (S/RW/FP) + ...

R/RW = ...

This signal may be generated during floating point operations, as described elsewhere in this manual:

Opcodes	Phase	Reference
FAS, FSS, FAL, FSL	CPU PH7, box PH9	Table 3-65
FAS, FSS, FAL, FSL	CPU PH8, box PH10	Table 3-65
FMS, FML	CPU PH7, box PH9	Table 3-66
FMS, FML	CPU PH8, box PH10	Table 3-66
FDS, FDL	CPU PH7, box PH9	Table 3-67
FDS, FDL	CPU PH8, box PH10	Table 3-67

If RW is not set during floating point operations, a trap occurs during the end phase of the CPU, and the trap accumulator register is cleared:

(S/TRAP) = FAFL NRW ENDE NINTRAP + ...

FAFL = NO1 O3 O4 O5

(R/TRACC1-R/TRACC4) = (R/TRACC) = (S/TRAP) + ...

The least significant hexadecimal digit is set to 4 by setting TR29 and resetting TR28, TR30, and TR31.

S/TR29 = FAFL NRW PH10 + ...

(R/TR28-R/TR31) = (R/TR) = (S/TRAP) + ...

Therefore, a floating point fault causes a trap to location X'44' with a code of 0000.

When a trap is caused by a floating point fault, the trap accumulator register is cleared.

(R/TRACC1-R/TRACC4) = (R/TRACC) = (S/TRAP) + ...

The least significant hexadecimal digit is set to 4 by setting P29.

S/TR29 = FAFL NRW PH10

Therefore, a floating point fault causes a trap to location X'44' with a code of 0000.

(R/TR28-R/TR31) = (R/TR) = (S/TRAP) + ...

Nonexistent Memory Address. Any attempt to access a nonexistent memory address causes a trap to location X'40' at the time of the request for memory service.

(S/TRAP) = ADNH NIOACT + ...

F/ADNH = ADNHL (Direct set)

ADNHL = ADNHL NACCL/1 + DRQ/1 (NAH AHCL)

R/ADNH = (R/ADNH)

(R/ADNH) = NIOACT + IOPH1 SW11

Flip-flop ADNH is direct set after the memory has had sufficient time to recognize the address. If the internal I/O is not active, (NIOACT) the trap sequence is initiated.

When a trap is caused by addressing a nonexistent memory location, the trap accumulator register is set to 0100 by setting TRACC2.

S/TRACC2 = NTRAP NSTRAP ADNH TRACC2 INH + ...

TRACC2 INH= N(FAILL PRETR) N(FANIMP PRETR)

(R/TRACC1-R/TRACC4) = (R/TRACC) = (S/TRAP) + ...

The least significant hexadecimal digit is set to 0.

(R/TR28-R/TR31) = (R/TR) = (S/TRAP) + ...

Therefore, a nonexistent memory trap causes a trap to location X'40' with a code of 0100.

Nonexistent Instructions. Any instruction on Sigma 5 that is neither standard nor optional is defined as nonexistent. This classification includes immediate addressing instructions that are indirectly addressed. If the execution of a nonexistent instruction is attempted, the CPU traps to location X'40' at the time the instruction is decoded.

(S/TRAP) = FAILL PRETR + ...

S/PRETR = NANLZ PRE1

R/PRETR = ...

FAILL = IA FAIM (Immediate instruction with indirect hit)

+ FUMMC N(ND12 ND13 D14) (Move to memory control with invalid X code)

+ OU2 O4 NO5 NO6 (28, 29)

+ OU1 NO4 O5 O6 (16, 17)

+ OU7 NO4 O5 O6 (76, 77)

+ OU2 NO4 O5 O6 (26, 27)

+ OU7 O4 (78 through 7F)

+ O1 NO3 NO4 NO5 (40 through 43, 60 through 63)

+ OU0 NO4 NO5 NFALCF (00, 01, 03)

+ OU5 OL9 (59)

+ OL4 O3 NFABYTE (14, 34, 54)

+ FAILL/1

FAILL/1 = O4 O5 O1 O3 (5C through 5F, 7C through 7F)

+ O4 O5 NO3 NO6 NO1 (0C, 0D, 2C, 2D)

When a trap is caused by attempted execution of an illegal instruction, the trap accumulator register is set to 1000 by setting TRACC1.

S/TRACC1 = FAILL PRETR NTRAP NSTRAP + ...

(R/TRACC1-R/TRACC4) = (R/TR) = (S/TRAP) + ...

The least significant hexadecimal digit is set to 0.

(R/TR28-R/TR31) = (R/TRACC) = (S/TRAP) + ...

Therefore, attempted execution of an illegal instruction causes a trap to location X'40' with a code of 1000.

Privileged Instructions. Privileged instructions can be implemented only by a CPU operating in the master mode, as indicated by flip-flop NMASTER. If this flip-flop, which is part of the program status doubleword (PSD), is set, privileged instructions cannot be implemented, but cause a trap to location X'40' at the time of instruction decoding.

(S/TRAP) = FAPRIV NMASTER PRETR NINTRAP + ...

FAPRIV = O4 O5 NO3

S/PRETR = NANLZ PRE1

R/PRETR = ...

S/NMASTER = S8 PSW1XS (Load bit 8 of PSD)

R/NMASTER = PSW1XS

The privileged instructions are LPSD, XPSD, WAIT, LRP, SIO, TIO, TDV, HIO, RD, WD, AIO, and MMC.

When a trap is caused by attempted execution of a privileged instruction by a CPU operating in the slave mode, the trap accumulator register is set to 0100 by setting TRACC3.

S/TRACC3 = FAPRIV NMASTER PRETR NTRAP NSTRAP + ...

(R/TRACC1-R/TRACC4) = (R/TRACC) = (S/TRAP) + ...

The least significant hexadecimal digit is set to 0.

(R/TR28-R/TR31) = (R/TR) = (S/TRAP) + ...

Therefore, a privileged instruction trap causes a trap to location X'40' with a code of 0100.

Unimplemented Instructions. Unimplemented instructions consist of all floating point instructions. If the floating point option is not included in the CPU, any floating point opcode generates an (S/TRAP) signal and causes a trap to location X'41' at the time of instruction decode.

(S/TRAP) = FANIMP PRETR

FANIMP = NO1 O3 O4 O5 NFPOPTION

NFPOPTION = Floating-point option not installed

The floating point opcodes are FSL, FAL, FDL, FML, FSS, FAS, FDS, and FMS.

When a trap is caused by an unimplemented instruction, the trap accumulator register is cleared.

(R/TRACC1-R/TRACC4) = (R/TRACC) = (S/TRAP) + ...

The least significant hexadecimal digit is set to 0001 by setting TR31.

S/TR31 = NSTRAP (S/TR31)

(S/TR31) = FANIMP + ...

(R/TR28-R/TR31)= (R/TR) = (S/TRAP) + ...

Therefore, an unimplemented instruction trap causes a trap to location X'41' with a code of 0000.

Fixed Point Overflow Instructions. Fixed point overflow can occur for the LCW, LAW, LCD, LAD, AI, AH, AW, AD, SH, SW, SD, DH, DW, AWM, MTH, and MTW instructions. Except for the DH and DW instructions, execution is allowed to proceed to completion. For DH and DW, the instruction execution is aborted without changing any register. If the trap mask (AM) is a one, the CPU traps to location X'43' instead of executing the next instruction in sequence.

(S/TRAP) = ENDE AM CC2 OVERIND + ...

S/AM = S11 PSW1XS (Set when PSD stored)

R/AM = PSW1XS

S/CC2 = (S15 + S16) PROBOVER/H
+ (S00 + S0) PROBOVER
+ FADIV PH4 + ...

R/CC2 = (After exit from trap)

OVERIND = FADIV + OVERIND/1

FADIV = FUDW NR31 + FADIVH (DW, DH)

S/OVERIND/1 = PROBOVER + PROBOVER/H

R/OVERIND/1 = CLEAR

PROBOVER/H = FAMT PH2 NINTRAP OU5 (MTW, MTH)

PROBOVER = FUAWM (PH1 + PH3) (AWM)
+ FALOAD/C (PH1 + PH3) NO1 (LCD, LCW)
+ FALOAD/A PH4 (LAD, LAW)
+ FALOAD/A PH2 NO1
+ FAARITH (PH1 + PH3) (AD, AI, AW, AH, SD, SW, SH)
+ FAMT PH2 NINTRAP (MTW, MTH, MTB)

An overflow resulting from a division instruction is detected before the instruction is executed; therefore, the divide instruction which would cause an overflow is aborted. An addition with the addend and augend having like signs, or a subtraction with a minuend and subtrahend having unlike signs, can cause an overflow.

When a trap is caused by a fixed point overflow fault, the trap accumulator register is cleared.

(R/TRACC1-R/TRACC4) = (R/TRACC) = (S/TRAP) + ...

The least significant hexadecimal digit is set to 3 by setting TR30 and TR31.

S/TR30 = NSTRAP (S/TR30) + ...

(S/TR30) = OVERIND PH10 AM CC2 + ...

S/TR31 = NSTRAP (S/TR31) + ...

(S/TR31) = OVERIND PH10 AM CC2 + ...

(R/TR28-R/TR31) = (R/TR) = (S/TRAP) + ...

Therefore, a fixed point overflow fault causes a trap to location X'43' with a code of 0000.

Memory Write-Protection Violation. A memory protection violation occurs when any instruction attempts to alter write-protected memory and the correct write key is non-zero and does not match the write lock for the memory page. When a memory protection violation occurs, the CPU aborts execution of the current instruction (without changing protected memory) and traps to location X'40'. The trap occurs before memory access.

(S/TRAP) = ABO + ...

ABO = ABO/1 ABO/2

ABO/1 = [(S/ABO/1) ABOT + ...] NCROSSADD

(S/ABO/1) = MBXS NPCPACT NIOACT NINTRAP N(FAIO PH3)

ABOT = DL2/110

ABO/2 = (LCK0 WK1 + LCK1 WK0) N(LCK0 LCK1 WK0 WK1)

When a trap is caused by a memory write-protection violation, the trap accumulator register is set to 0001 by setting TRACC4.

S/TRACC4 = NTRAP NSTRAP ABO + ...

(R/TRACC1-R/TRACC4) = (R/TRACC) = (S/TRAP) + ...

The least significant hexadecimal digit is set to 0.

(R/TR28-R/TR31) = (R/TR) = (S/TRAP) + ...

Therefore, a memory write-protection violation causes a trap to location X'40' with a code of 0001.

WATCHDOG TIMER. The watchdog timer (WDT) ensures that the CPU must periodically reach interruptible points of operation in the execution of instructions. An interruptible point is a time during the execution of a program when an interrupt request (if present) would be acknowledged. Interruptible points occur at the end of every instruction and during the execution of some instructions. The WDT measures elapsed time from the last interruptible point. If the maximum allowable time has been reached before the next time that an interrupt could be recognized, the current instruction is aborted and the WDT runout trap is activated. Except for a nonexistent address used with RD or WD, programs trapped by the WDT cannot (in general) be continued. After a WDT runout, the CPU traps to location X'46'.

WDT signal STRAP is controlled by a binary counter and control flip-flops.

STRAP = WDTA WDTRAC CEINT + ...

In the 6-bit binary counter, WCT1 represents the most significant bit, and WCT6 represents the least significant bit. The counter is advanced by the 1-MHz clock signal (1MC).

When a trap is caused by watchdog timer runout, all flip-flops of the trap accumulator register are reset.

(R/TRACC1-R/TRACC4) = (R/TRACC) = (S/TRAP) + ...

The least significant hexadecimal digit is set to 6 by setting TR29 and TR30.

S/TR29 = STRAP + ...

S/TR30 = STRAP + ...

(R/TR28-R/TR31) = (S/TRAP) + ...

Therefore, a watchdog timer runout trap causes a trap to location X'46' with a code of 0000.

3-30 Interrupts

GENERAL. The interrupt system provides for a maximum of 237 interrupt levels, of which 13 are internal and 224 are external. The 13 internal interrupt levels include seven standard features (two count-pulse interrupts, a memory parity interrupt, two counter-equals-zero interrupts, an input/output interrupt, and a control panel interrupt) and six optional features (the power-on interrupt, the power-off interrupt, two additional count-pulse interrupts, and two additional counter-equals-zero interrupts). The 224 external interrupts are divided into 14 groups of 16 interrupt levels each. Chassis writing in the CPU divides the internal interrupts into the override group, the counter-equals-zero group, and the input/output group. The override group has priority over all interrupt groups. The priority sequence of all other groups is optional, as described in the Sigma 5 Computer Reference Manual under the Interrupt System heading.

Interrupt Control. Interrupt operations are controlled by logic and by programming. Each of the 237 interrupt levels is assigned a unique memory location to which the CPU branches when the interrupt level is acknowledged. The contents of the memory location are transferred to the CPU. The interrupt location must contain one of the following instructions: modify and test byte (MTB), modify and test halfword (MTH), modify and test word (MTW), or exchange program status doubleword (XPSD). The MTB, MTH, and MTW instructions are single instruction interrupts. The

XPSD instruction transfers control of the CPU to a service routine stored in memory. The service routine must end with a load program status doubleword instruction (LPSD).

Operation of groups of interrupt levels is controlled by the program status doubleword. If bit 37 is a one, CIF is set, and the count-equals-zero interrupts are inhibited. If bit 38 is a one, II is set, and the input/output interrupts are inhibited. If bit 39 is a one, EI is set, and all external interrupt groups are inhibited. The power-on and power-off interrupts, if installed, are always enabled and armed, and cannot be inhibited. The override interrupts also cannot be inhibited.

The address of the memory location associated with each interrupt level is controlled by signals which indicate that an interrupt level is waiting, enabled, and has priority over other interrupt levels.

Interrupt Levels. Each of the 237 interrupt levels includes an interrupt circuit consisting of three flip-flops. The state of the interrupt circuit indicates the status of the interrupt level. A circuit which is disarmed is effectively removed from the interrupt system. A circuit which is armed is transferred to the waiting state when an event or condition associated with the circuit is detected. (The event or condition may be a power failure, a programmed count sequence, or a control panel operation, as typical examples.) If a circuit in the waiting state is enabled, it causes an interrupt operation to begin when that interrupt level has priority. Priority is established by a combination of signals generated by interrupt circuits and by system cabling. Priority is also controlled by bits 37, 38, and 39 of a program status doubleword (PSD), which in turn are controlled by write direct instructions. An interrupt circuit may be enabled only by a write direct instruction, or by an XPSD or LPSD instruction. When an enabled circuit in the waiting state is acknowledged, it is transferred to the active state. Any number of interrupt circuits may be in the waiting and enabled state, but only one may be in the active state at any one time.

Interrupt Sequence. The interrupt system permits the interruption. Interrupt operations are controlled by interrupt/trap phase flip-flops INTRAP, INTRAP1, and INTRAP2, as summarized in table 3-11 and illustrated in figure 3-53.

Table 3-11. Interrupt Sequence

Phase	Function Performed	Signals Involved			Comments
Preliminary	Set flip-flop INT	S/INT	=	INT9	Flip-flop INT set when an interrupt circuit is waiting, enabled, and has priority
		R/INT	=	...	
	Enable signal IEN	IEN	=	KRUN PH10 NIOSC NDCSTOP	IEN can be true only at end of execution (PH10)
	Set flip-flops INTRAP, INTRAP1, and INTRAP2 to establish interrupt condition	S/INTRAP	=	(S/INTRAP) NRESET	True (S/INTRAP) signal sets three flip-flops when interrupt enabled (INT IEN). Flip-flop TRAP remains in reset state to distinguish interrupt and trap
		(S/INTRAP)	=	INT IEN NINTRAP + ...	
		R/INTRAP	=	(R/TRAP) + FAMT PH9	
		(R/TRAP)	=	FAPSD PH5 + RESET	
		S/INTRAP1	=	(S/INTRAP) NRESET	
		R/INTRAP1	=	RESET + NINTRAP2	
		S/INTRAP2	=	(S/INTRAP2) NRESET	
		(S/INTRAP2)	=	(S/INTRAP) + INTRAP1 NINTRAP2	
		R/INTRAP2	=	...	
	Inhibit reset of flip-flop NPRE1	S/NPRE1	=	N(S/PRE1)	Entry into PREP phase of subsequent instruction inhibited by true (S/INTRAP) signal
		(S/PRE1)	=	PRE1EN PH10 + ...	
		NPRE1EN	=	(S/INTRAP) + ...	
		R/NPRE1	=	...	

(Continued)

Table 3-11. Interrupt Sequence (Cont.)

Phase	Function Performed	Signals Involved			Comments
Preliminary (Cont.)	Enable CLEAR signal	CLEAR	=	(S/INTRAP) + ...	Clear selected flip-flops
INTRAP1	Set flip-flop CEINT	S/CEINT	=	INTRAP1 INTRAP2 NTRAP + ...	Inhibits CPU clock during next phase, until action response from interrupt circuits
		R/CEINT	=	...	
	(P15-P31)⟶(B15-B31)	Bn	=	Pn BXP + ...	Next instruction in program sequence
		BXP	=	BXP1 + ...	
		BXP/1	=	INTRAP1 BRP + ...	
INTRAP2	Reset flip-flop BRP	R/BRP	=	INTRAP1 + ...	Indicate next instruction in sequence is in B-register
	Reset flip-flop INTRAP2	R/INTRAP2	=	...	
INTRAP1	Sustain B15	S/B15	=	(S/B15) + ...	Prevent reset of B15 if set
		(S/B15)	=	B15 NBRP INTRAP1 + ...	
		R/B15	=	INTRAP1 + ...	
	Inhibit CPU clock until ARE	CLEN	=	NCEINT + CEINT ARE + ...	ARE controlled by interrupt sequence
		ARE	=	AIE1 1MC + ...	
	(INT0-INT8)⟶(P23-P31)	PXINT	=	INTRAP1 NINTRAP2 NTRAP + ...	Clear P-register and store interrupt address
		PX	=	INTRAP1 NINTRAP2 + ...	
	Set flip-flop MRQ	S/MRQ	=	(S/MRQ/2) + ...	Request for core memory cycle
		(S/MRQ/2)	=	INTRAP1 NINTRAP2 + ...	
		R/MRQ	=	...	
	Set flip-flop DRQ	S/DRQ	=	(S/DRQ) NCLEAR	Data request, inhibiting transmission of another clock until data release
		(S/DRQ)	=	(S/MRQ/2) + ...	
		R/DRQ	=	...	
	Reset flip-flop CEINT	R/CEINT	=	...	Enable CPU clock
	Reset flip-flop INTRAP1	R/INTRAP1	=	NINTRAP2	
	Set flip-flop INTRAP2	S/INTRAP2	=	INTRAP1 NINTRAP2 + ...	
INTRAP2	(MB0-MB31)⟶(C0-C31)	CXMB	=	DG	Extract addressed word and store for execution of instruction (MTB, MTH, MTW, or XPSD)
	(C0-C31)⟶(D0-D31)	(S/SXD)	=	NINTRAP1 INTRAP2	
	(C0-C7)⟶(O0-O7)	DXC	=	INTRAP2 + ...	
	C10⟶R30	OXC	=	NINTRAP1 INTRAP2 + ...	
	C11⟶R31	RXC	=	NINTRAP1 INTRAP2 + ...	

(Continued)

Table 3-11. Interrupt Sequence (Cont.)

Phase	Function Performed	Signals Involved			Comments
INTRAP 2 (Cont.)	Reset flip-flop NPRE1	S/NPRE1	=	N(S/PRE1) + ...	Enable entry into PREP phase
		(S/PRE1)	=	NINTRAP1 INTRAP2 + ...	
		R/NPRE1	=	...	
	Reset flip-flop INTRAP2	R/INTRAP2	=	...	
	If instruction is modify and test, reset INTRAP during phase 9	R/INTRAP	=	FAMT PH9 + ...	
	If count reduced to zero, CNTZREQ during phase 2	CNTZREQ	=	S0031Z FAMT PH2 INTRAP	Enable interrupt level
	If instruction is XPSD, execute service routine, terminate with LPSD unless service routine interrupted	R/INTRAP	=	FAPSD PH5 + ...	
	Trap sequence ended				

The interrupt/trap phase flip-flops are clocked by CPU ac signals; however, actual control of the phases is in the interrupt circuits, which are clocked by a 1-MHz clock from the CPU. Synchronization of the clocks is done by disabling the CPU clock until a 1-MHz clock is received from the interrupt chassis. During the interrupt phases, the next instruction address is stored and the interrupt address associated with the interrupt in progress is received by the CPU for memory access. The general sequence of operations for an interrupt is illustrated in figure 3-54.

The program is executed in normal sequence until an interrupt is detected. A signal generated by the interrupt circuits is sampled at the end of each instruction, during iterated sequences, and during execution of a move to memory control instruction. If any interrupt circuit is waiting and enabled, is not inhibited, and has priority an interrupt sequence begins.

The contents of the P-register, which contains the address of the next program instruction in normal sequence, are stored in the B-register. The code from interrupt signals INT0 through INT8, which is the address of the memory location associated with the interrupt, are stored in the P-register. The CPU then accesses that location in memory, transfers the contents to the C-register, and stores the data in the D-, O-, and R-registers. If the contents of the memory location are not an XPSD, MTB, MTH, or MTW instruction, a program error has occurred, and subsequent operations are meaningless.

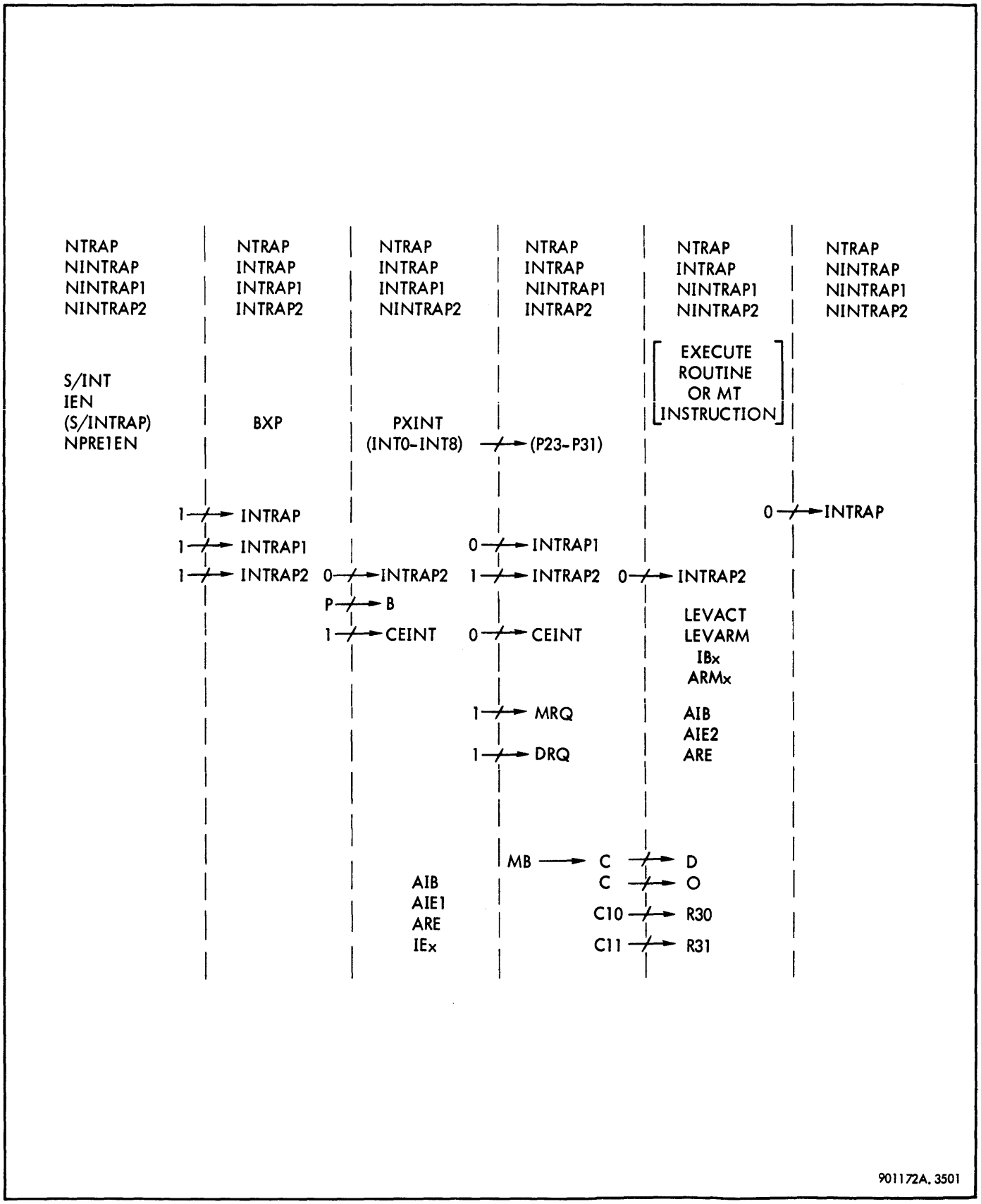


Figure 3-53. Interrupt Phases

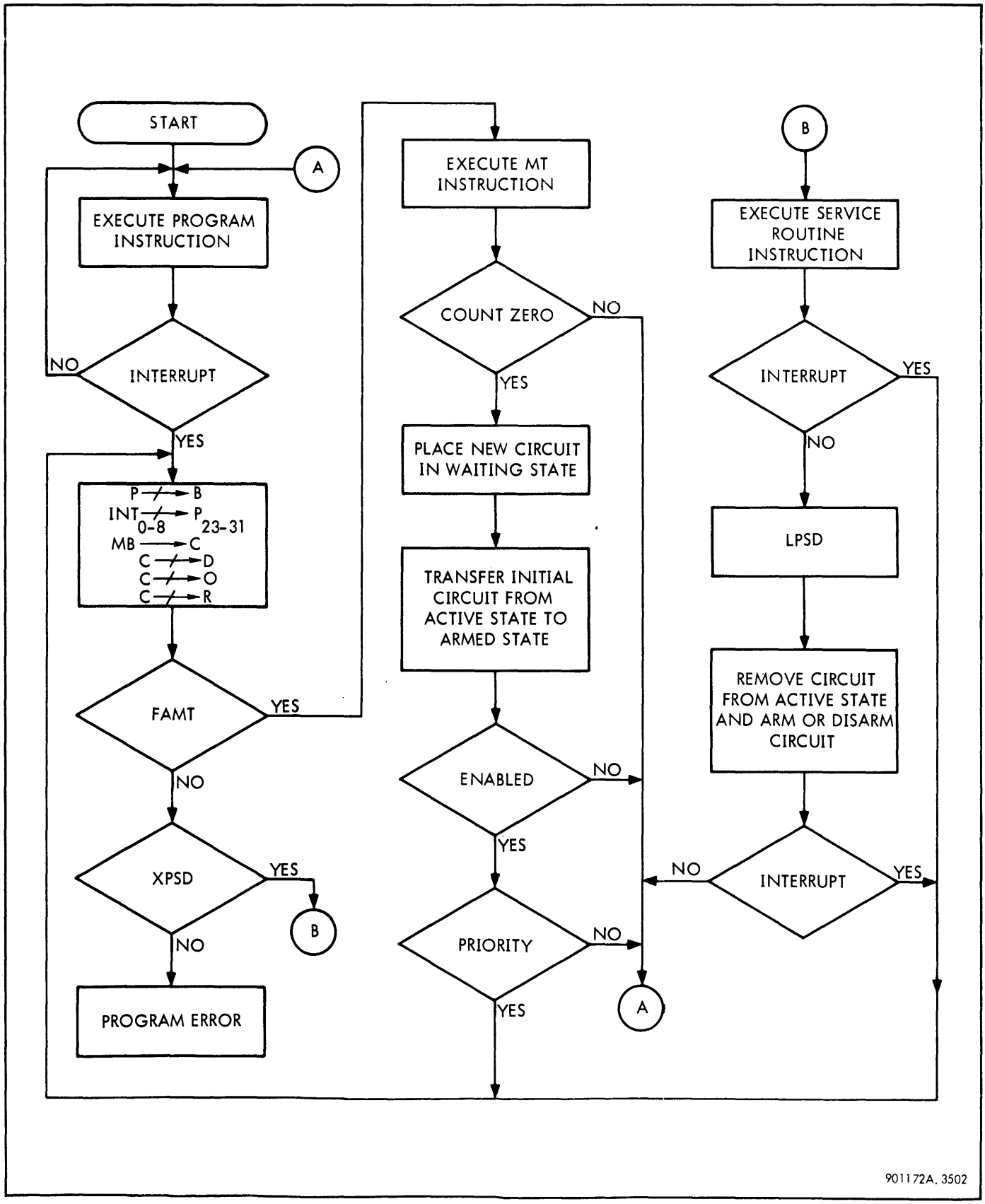
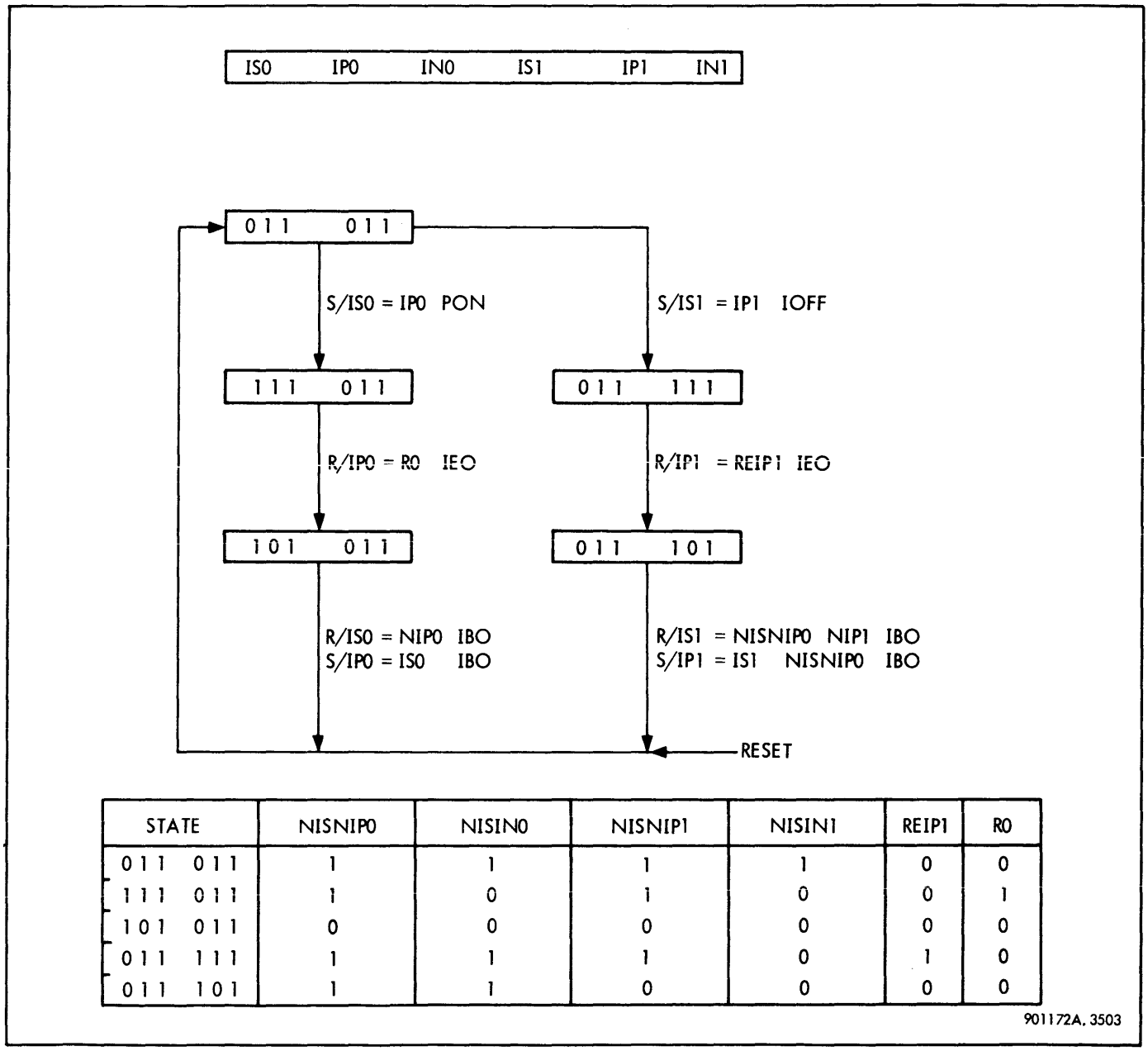


Figure 3-54. Interrupt Sequence, Flow Diagram



STATE	NISNIP0	NISIN0	NISNIP1	NISIN1	REIP1	R0
0 1 1 0 1 1	1	1	1	1	0	0
1 1 1 0 1 1	1	0	1	0	0	1
1 0 1 0 1 1	0	0	0	0	0	0
0 1 1 1 1 1	1	1	1	0	1	0
0 1 1 1 0 1	1	1	0	0	0	0

901172A.3503

Figure 3-55. Power-On and Power-Off Interrupt Circuits, Cycle of Operation

After the XPSD instruction in the interrupt location has been executed, the associated service routine is followed, and the interrupt circuit returns to the armed state.

R/IS1 = NISNIP0 NIP1 IBO

IBO = AIB LEVACT

LEVACT = FAPSD PH5 NO7 R30

S/IP1 = IS1 NISNIP0 IBO

Signal AIB is a timing signal generated during the interrupt sequence. Signal LEVACT is generated by an LPSD signal in the service routine.

When power is applied, signal PON is true, and interrupt circuit 0 goes through a similar sequence of operations.

S/IP0 = IS0 IBO + RESET

R/IP0 = R0 IEO

R0 = IN0 IP0 IS0

S/IS0 = IP0 PON

R/IS0 = NIP0 IBO + RESET

If a modify and test instruction is accessed, the instruction is executed, and the modified contents of the addressed location are sampled. If the count is not zero, the CPU returns to the program. If the count is zero, an additional interrupt circuit may be placed in the waiting state (not the circuit which caused the interrupt sequence to start). If the circuit now placed in the waiting state is enabled and has priority, the CPU will begin a new interrupt cycle. If the interrupt circuit is not enabled or does not have priority, the CPU returns to the program.

If an XPSD instruction is accessed, the instruction is executed and the CPU is controlled by a service routine stored in memory. This service routine may itself be interrupted at the end of any instruction in the routine. If the service routine is not interrupted, it is terminated by an LPSD instruction. When the LPSD instruction is executed, the interrupt circuit which caused the interrupt sequence to start is transferred from the active state to either the armed or the disarmed state. If a new interrupt circuit is waiting and enabled and has priority, a new interrupt sequence is begun; otherwise, the CPU returns to the program.

INTERRUPT CIRCUITS. Each interrupt level is controlled by an interrupt circuit which establishes the state of interrupt level and generates signals which control priority of each level. Each interrupt circuit consists of three flip-flops – ISn, IPn, INn (n = 0, 1, 2, ... 15). The five significant states of an interrupt circuit and the conditions established for the level are summarized in table 3-12.

Power Fail-Safe Interrupts. Interrupt levels 0 and 1 are the power-on and power-off interrupts, which are controlled by optional equipment. These circuits have the highest priority level, and are always enabled. The cycle of operation for the power fail-safe interrupts is illustrated in figure 3-55.

These interrupt levels are always enabled because the inputs to INn flip-flops are hardwired.

S/IN0 = ...
R/IN0 = GND
S/IN1 = ...
R/IN1 = GND

They are placed in the armed state by a reset signal, and are normally in the armed and enabled state.

S/IP0 = RESET + ...
R/IS0 = RESET + ...
S/IP1 = RESET + ...
R/IS1 = RESET + ...

When power fails, signal IOFF is true, and interrupt level 1 is placed in the waiting and enabled state.

S/IS1 = IP1 IOFF

Table 3-12. Significant States of Interrupt Circuit

FLIP-FLOPS ISn (n=0, 1, 2, ... 15)	FLIP-FLOPS IPn	FLIP-FLOPS INn*	STATE
0	0	X	Disarmed. Circuit effectively removed from interrupt system. Interrupt signal neither accepted nor remembered. Change of state only by program intervention
0	1	X	Armed. Can accept and remember interrupt signal. Advances to waiting state when interrupt signal is recognized
1	1	0	Waiting and disabled. Cannot advance to active state. Requires program intervention to be enabled
1	1	1	Waiting and enabled. Can advance to active state if interrupt circuit has highest priority
1	0	1	Active or waiting. The highest priority circuit in this state will become active when accepted as an interrupt by the CPU. Other circuits in this state wait for acceptance in priority sequence

*Power fail-safe interrupt circuits (0 and 1) are always enabled, so that state of flip-flops is XX1 at all times (INn set).

An interrupt sequence takes place, during which the interrupt level is placed in the active state.

R/IP1 = REIP1 IEO

REIP1 = IN1 IP1 IS1 NISIN0 NISNIP0

NISIN0 = NIS0 + NIN0

NISNIP0 = N(IS0 NIP0) = NIS0 + IP0

IEO = AIE1 ENOVRD

ENOVRD = R01 + ...

R01 = REIP1 + ...

Signals NISIN0 and NISNIP0 are priority signals that prevent a change of state if the higher priority 0 level is waiting and enabled, or active. Signal ENOVRD initiates the interrupt sequence. Signal AIE1 is a timing signal generated during an interrupt sequence. All interrupt circuits are clocked by the 1-MHz signal (1MCS).

Because interrupt level 0 has the highest priority of all interrupts, no priority signals such as NISNIP0 or NISIN0 are required. Signal ENOVRD is enabled by signal R0.

ENOVRD = R01 + ...

R01 = R0 + ...

Count-Pulse Interrupts. Interrupt levels 2 through 5 are count-pulse interrupts, two of which are standard and two of which are optional. The feature of an interrupt circuit that distinguishes its function is the input that enables the change of state from armed to waiting (01X to 11X). For the count-pulse interrupts, the signals are CPUL1 through CPUL4.

S/IS2 = IP2 CPUL1 + ...

S/IS3 = IP3 CPUL2 + ...

S/IS4 = IP4 CPUL3 + ...

S/IS5 = IP5 CPUL4 + ...

These signals are generated by flip-flops of the real-time counters, which are direct reset after the transfer from the armed state (01X) to the waiting state (11X).

E/CPUL1 = IS2

E/CPUL2 = IS3

E/CPUL3 = IS4

E/CPUL4 = IS5

The inputs to interrupt circuit 2 are typical of interrupt circuits 2 through 15.

S/IN2 = DAT16 AEENLE

R/IN2 = DAT16 ADBDB + REN

S/IP2 = IS2 NISNIP1 ARMOVD
+ DAT16 AEADB

NISNIP1 = NIS1 NISNIP0 + IP1 NISNIP0

ARMOVD = IBO LEVARM

LEVARM = LEVACT N(FAPSD PH5 NR31)

R/IP2 = R2 IEO + DAT16 DARM

R2 = IS2 IP2 IN2 NISIN1

NISIN1 = NIS1 NISIN0 NISNIP0
+ NIN1 NISIN0 NISNIP0

S/IS2 = IP2 CPUL1 + DAT16 IP2 TRIG

R/IS2 = NIP2 NISNIP1 IBO + DAT16 DARM

Signals associated with DAT16 are controlled by a write direct instruction. Therefore, the circuit can be enabled (placed in state XX1) only during a WD instruction. The circuit is also initially placed in the armed state (010 or 011) by a WD instruction.

Signals NISNIP1, NISNIP0, NISIN0, and NISIN1 are priority signals that prevent changes of state when higher priority circuits are active, or waiting and enabled. Signals such as this are generated at all levels, making it possible for only one interrupt circuit to transfer to the active state (101) at any time. More than one interrupt circuit can be in the active state, if a higher priority interrupt circuit goes active during a subroutine for a lower priority interrupt circuit. Several interrupt circuits may be in the waiting and enabled state (111) at one time. However, only the circuit having the highest priority can be transferred to the active state (111 to 101).

Signals IEO, IBO, and ARMOVD are generated during the interrupt sequence for the override interrupts. Corresponding signals for the counter-equals-zero interrupts are IEC, IBC, and ARMCNTR. Corresponding signals for the input/output interrupts are IEI, IBI, and ARMIO.

The normal sequence of operations for an interrupt circuit begins when the circuit is armed (placed in state 01X). When the triggering signal is true, the circuit is placed in the waiting state (11X).

S/IS2 = IP2 CPUL1 + ...

When the circuit is enabled and waiting, and has highest priority, it initiates an interrupt sequence and is transferred to the active state (101).

R/IP2 = R2 IEO + ...

While the interrupt circuit is in the active state, the instruction stored in the associated memory location is executed. After all operations associated with the interrupt have been completed, the interrupt circuit leaves the active state (101 to 011 or 001). (The circuit cannot be disabled by an interrupt sequence.)

S/IP2 = IS2 NISNIP1 ARMOVD + ...

R/IS2 = NIP2 NISNIP1 IBO + ...

ARMOVD = IBO LEVARM

Signal IBO will always be true at the end of the interrupt sequence, causing a transfer from state 101 to state 0X1. If signal LEVARM is also true at the end of the active state, the transfer is from state 101 to 011; if LEVARM is false, the transfer is from state 101 to 001.

Memory Parity Interrupt. Interrupt circuit 6 is transferred from the armed state to the waiting state (01X to 11X) if flip-flop PEINT is set, indicating parity error.

S/IS6 = IP6 PEINT + ...

Flip-flop PEINT is reset after the interrupt circuit exits from the active state.

R/PEINT = (R/PEINT/2) + ...

(R/PEINT/2) = IN6 NIS6

Counter-Equals-Zero Interrupts. Interrupt circuits 8, 9, 10, and 11 are controlled by interrupt circuits 2, 3, 4, and 5, respectively.

```
S/IS8     = IP8 SR8
  SR8     = CNTZREQ ISNIP2
  CNTZREQ = FAMT PH2 INTRAP S0031Z
  ISNIP2  = IS2 NIP2
S/IS9     = IP9 SR9
  SR9     = CNTZREQ IS3 NIP3
S/IS10    = IP10 SR10
  SR10    = CNTZREQ ISNIP4
  ISNIP4  = IS4 NIP4
S/IS11    = IP11 SR11
  SR11    = CNTZREQ IS5 NIP5
```

Whenever one of the count-pulse interrupt circuits is in the active state (101) and the count has been reduced to zero (S0031Z) one of the counter-equals-zero interrupt circuits is placed in the waiting state (01X to 11X).

Input-Output Interrupt. Interrupt circuit 12 is placed in the waiting state (11X) by an IOP interrupt request signal.

```
S/IS12 = IP12 IR
```

Control Panel Interrupt. Interrupt circuit 13 is placed in the waiting state (11X) by a control panel switch interlocked with a flip-flop.

```
S/IS13  = IP13 SR13
  SR13  = KINTRP NCNLK
S/CNLK  = IS13
R/CNLK  = NKINTRP/B
C/CNLK  = N1MCS
```

PRIORITY SIGNALS. Signals generated by interrupt circuits are interconnected in order to control priority of interrupt levels. Interrupt levels 0 through 7 have the highest priority. Counter-equals-zero interrupts, input/output interrupts, and external interrupts in groups of 16 may be connected in any priority sequence at the option of the user. Signals external to the CPU control all priority assignments after interrupt level 7.

The priority signals permit only one interrupt circuit in the waiting and enabled state (111) to be transferred to the active or waiting state (101) on a particular interrupt clock. More than one interrupt circuit may be in the active or waiting state at a given time. For example, if a high-priority interrupt circuit is transferred to the active or waiting state while a low-priority interrupt circuit is active, the high-priority circuit will override the low-priority circuit. While the high-priority circuit is active, the low-priority circuit previously active will be dormant until the high-priority circuit has completed its operation. The low-priority circuit, having remained in the active or waiting state (101), then resumes operation.

Interrupt Circuit Priority Signals. Associated with each interrupt circuit are priority signals. Typical signals for even-numbered circuits are:

```
R10     = IN10 IP10 IS10 NISIN9
NISIN10 = N(IS10 IN10)
NISNIP10 = N(IS10 NIP10)
```

Thus, R10 can be true only if interrupt circuit 10 is waiting and enabled and if no high priority interrupt in that group is waiting or active. Signal NISIN10 is true only if circuit 10 is not active, and not waiting and enabled, and NISNIP10 is true only if circuit 10 is not active.

Typical signals for odd-numbered circuits are:

```
REIP11   = IN11 IP11 IS11 NISIN10 NISNIP10
           NISIN9
NISIN11  = NIS11 NISIN10 NISNIP10 NISIN9
           + NIN11 NISIN10 NISNIP10 NISIN9
NISNIP11 = NIS11 NISNIP10 NISNIP9
           + IP11 NISNIP10 NISNIP9
```

Thus, REIP11 can be true only if interrupt circuit 11 is waiting and enabled and no higher priority interrupt in that group is waiting or active. Signal NISIN11 can be true only if interrupt circuit 11 is not active, and not waiting and enabled and no higher priority interrupt in that group is waiting or active. In addition, signals NISIN10 NISNIP10 prevent REIP11 or NISIN11 from being true unless interrupt circuit 10 is not active, and not waiting and enabled. Signal NISNIP11 can be true only if interrupt circuit 11 is not active and interrupt circuit 10 is not active.

Signals NISIN9 and NISNIP9 are controlled by all interrupt circuits from 0 through 9. Similar signals are generated at all levels of interrupts.

Signals generated by odd-numbered circuits and even-numbered circuits are combined into such signals as:

```
R1011 = R10 + REIP11
```

Signal R1011 will be true if either circuit 10 or circuit 11 is waiting and enabled, and no higher priority circuit is waiting and enabled.

Priority Chain Signals. An interrupt sequence is initiated after signal INT9 is true, enabling flip-flop INT to be set. Signal INT9 is controlled by inputs from all interrupt circuits, including external interrupts.

```
INT9 = ENOVRD + ENCNTR + ENIO + ...
```

Signal ENOVRD is true if any of the first eight interrupt circuits is waiting and enabled.

```
ENOVRD = R01 + R23 + R45 + R67
  R01  = R0 + REIP1 (typical)
```

More than one interrupt circuit may be waiting and enabled; however, only one can generate a true Rx signal or REIPy signal.

Signal ENCNTR is true if any of interrupt circuits 8 through 11 is waiting and enabled (R89 + R1011), provided the group is not inhibited by the program status doubleword (NCIF), and no higher priority group is waiting and enabled.

ENCNTR = NCIF NHRQBZC (R89 + R1011)

HRQBZC = /HRQBZC/ = RQBZO

NRQBZO = NENOVRD NISNIP7 NIEO

Signal HRQBZC goes outside the CPU to provide for the option of external interrupts with higher priority than the counter-equals-zero interrupt group or the input/output interrupt group. Signal NISNIP7 is generated by interrupt circuit 7. Signal NIEO is generated during an operation sequence.

Signal ENIO is true if any of interrupt circuits 12 through 15 is waiting and enabled (R1213 + R1415), provided the group is not inhibited by the program status doubleword (NII), and no higher priority group is waiting and enabled.

ENIO = NII NHRQBZI (R1213 + R1415)

HRQBZI = /HRQBZI/ = RQBZC

NRQBZC = NENCNTR NHRQBZC NISNIP11 NIEC

Signal LINREQ is generated in external equipment when an external interrupt is waiting and enabled, and starts an interrupt sequence if no write direct instruction in the interrupt mode (0001) is active. The NEWDM term is required because /DATm/ lines are shared between trigger arm, enable data on output during a write direct instruction, and memory address data on input during interrupt operations.

INT9 = LINREQ NEWDM + ...

LINREQ = /DAT25/

EWDM = NB16 NB17 NB18 B19 DIOWD

Group Control. Priority signals generated by the interrupt circuits also control signals which cause changes of state in the interrupt circuits during an interrupt sequence. These signals permit only one group of interrupts to be controlled at any time. The family of IEx signals cause a change of state from waiting and enabled (111) to active (101).

IEO = AIE1 ENOVRD

IEC = AIE1 ENCNTR

IEI = AIE1 ENIO

The family of IBx signals remove an interrupt circuit from the active state.

IBO = AIB LEVACT

IBC = AIB LEVACT NHBZC

HBZC = /HBZC/ = BZO

BZO = ISNIP7

IBI = AIB LEVACT NHBZI

HBZI = /HBZI/ = BZC

BZC = N(NHBZC NISNIP11)

BZI = N(NHBZI NISNIP15)

The family of ARMx signals cause a change of state from active to armed (011).

ARMOVD = IBO LEVARM

ARMCTR = IBC LEVARM

ARMIO = IBI LEVARM

Signals AIE1, AIB, LEVACT, and LEVARM are generated during an interrupt sequence. If the LEVARM signal is false, the interrupt circuit will be left in the disarmed state (001) after transfer from the active state. Signal BZI enables external interrupts to be controlled.

Memory Address Control. Signals INT0 through INT8 retain a code addressing the memory location associated with an interrupt level. This code, which is transferred to the P-register during the interrupt sequence, is established by priority signals generated in the interrupt circuits.

Signals INT0, INT1, and INT3 are false for any internal interrupt level. Signals INT2 and INT4 are true for any internal interrupt level.

INT2 = ENOVRD + ENCNTR + ENIO + ...

INT4 = ENOVRD + ENCNTR + ENIO + ...

Thus signals INT0 through INT8 hold the code 0 0101 XXXX for any internal interrupt level.

Signals INT5 through INT8 hold a code dependent upon the internal interrupt level enabled.

INT5 = ENCNTR + ENIO + ...

INT6 = OVLN6 ENOVRD + ENIO + ...

OVLN6 = R45 + R67

INT7 = OVLN7 ENOVRD + CNLN7 ENCNTR + IOLN7 ENIO + ...

OVLN7 = R23 + R67

CNLN7 = R1011

IOLN7 = R1415

INT8 = OVLN8 ENOVRD + CNLN8 ENCNTR + IOLN8 ENIO + ...

OVLN8 = REIP1 + REIP3 + REIP5 + REIP7

CNLN8 = R911

IOLN8 = REIP13 + REIP15

Although more than one circuit may be waiting and enabled at a time, only one of the Rx or REIPy signals associated with the internal interrupt circuits can be true at any time. The last four bits of the code held by signals INT0 through INT8 will be any of 0000 through 1111, depending upon the interrupt circuit which controls the interrupt. Therefore, the address code for an internal interrupt will be any value between 0 0101 0000 and 0 0101 1111 (hexadecimal 050 through 05F).

SERVICE ROUTINE SEQUENCE. A timing diagram for an interrupt operation which transfers control to a stored service routine is illustrated in figure 3-56.

When an interrupt circuit is waiting and enabled, INT is set.

S/INT = INT9 = ENOVRD + ENCNTR + ENIO + LINREQ NEWDM

The three interrupt flip-flops are then set at the end of phase 10 of a program instruction.

S/INTRAP = (S/INTRAP) NRESET
(S/INTRAP) = INT IEN NINTRAP
IEN = KRUN PH10 NIOSC NDCSTOP
S/INTRAP1 = (S/INTRAP) NRESET
S/INTRAP2 = (S/INTRAP2) NRESET
(S/INTRAP2) = (S/INTRAP) + ...

At the following CPU clock, CEINT is set and INTRAP2 is reset.

S/CEINT = INTRAP1 INTRAP2 NTRAP + ...
R/INTRAP2 = ...

The CPU clock is inhibited until signal ARE is true, and signal PXINT is true to enable transfer of address data to the P-register.

CLEN = NCEINT + CEINT ARE + ...
ARE = AIE1 IMC
PXINT = INTRAP1 NINTRAP2 NTRAP

Signal ARE is controlled by the 1-MHz clock and prevents changes of state in the interrupt circuit by inhibiting gated clock signal GCLK.

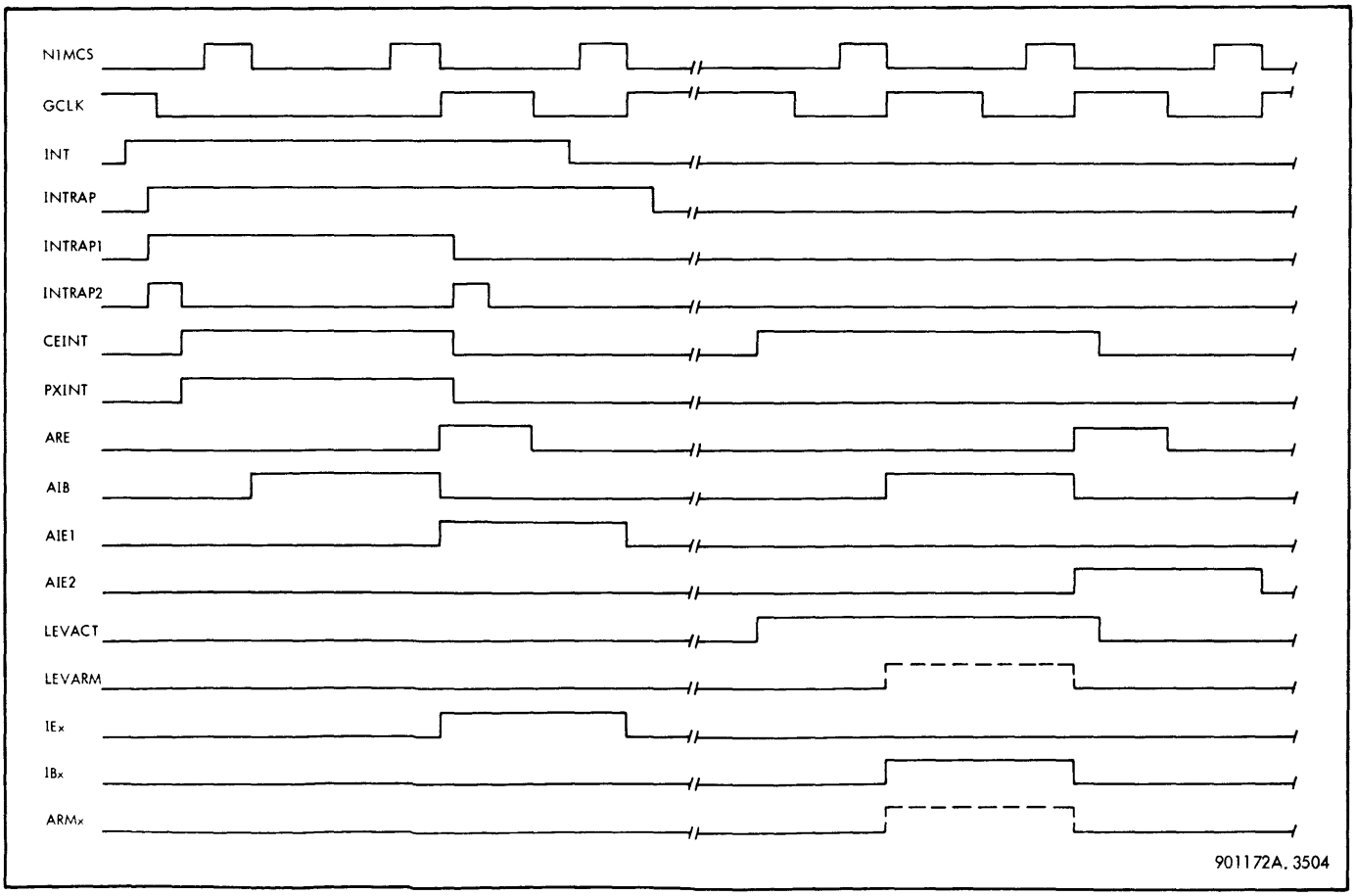


Figure 3-56. Service Routine, Timing Diagram

S/AIB = (S/AIB) NAIB
(S/AIB) = PXINT NAIE1 + ...
R/AIB = ...
C/AIB = N1MCS
S/AIE1 = AIB PXINT
R/AIE1 = ...
C/AIE1 = N1MCS
GCLK = 1MC (NAIB + NPXINT)

While signal AIE1 is true, a true IEO, IEC, or IEI signal causes the interrupt circuit to transfer from waiting and enabled (111) to active (101).

R/IPn = Rn IEx + ... (typical)
IEO = AIE1 ENOVRD
IEC = AIE1 ENCNTR
IEI = AIE1 ENIO

If the interrupt logic is servicing an external interrupt, signal DAT26 is generated to enable the change of state in the external circuit.

DAT26 = AIE1 + ...

After signal ARE is true, a CPU clock is generated, resetting INTRAP1, setting INTRAP2, and resetting CEINT.

R/INTRAP1 = NINTRAP2
S/INTRAP2 = INTRAP1 NINTRAP2 + ...
R/CEINT = ...

At this time, the XPSD instruction in the assigned memory location is extracted from memory and executed. The service routine addressed by the XPSD instruction may itself be interrupted. A service routine which is not interrupted is terminated by an LPSD instruction which sets CEINT at exit from phase 4 and causes LEVACT to be true during phase 5.

S/CEINT = FAPSD PH4 NO7 R30 + ...
LEVACT = FAPSD PH5 NO7 R30 + ...

The CPU clock is inhibited until signal ARE is true.

CLEN = NCEINT + CEINT ARE + ...
ARE = AIE2 1MC + ...
S/AIB = (S/AIB) NAIB
(S/AIB) = LEVACT NAIE2 + ...
R/AIB = ...
C/AIB = N1MCS
S/AIE2 = AIB LEVACT
R/AIE2 = ...
C/AIE2 = N1MCS

As the interrupt service routine ends, the interrupt circuit which initiated the operation is transferred from the active state (101) to either the disarmed state (001) or the armed state (011). For any change of state, bit position 10 of the LPSD instruction must contain a one, causing R30 to be set, and enabling LEVACT to be true.

R/ISn = NIPn NISNIPy IBO + ... (typical)
IBO = AIB LEVACT

If bit position 11 of the LPSD instruction contains a one, R31 will be set, LEVARM will be true, and the change of state will be from active to armed (101 to 011).

S/LPn = ISn NISNIPy ARMOVD + ... (typical)
ARMOVD = IBO LEVARM
LEVARM = LEVACT N(FAPSD PH5 NR31)

If bit position 11 of the LPSD instruction contains a zero, R31 will not be set, and the change of state will be from active to disarmed (101 to 001).

If the interrupt logic is servicing an external interrupt, signals DAT27 and DAT28 are generated to enable changes of state in the external circuit.

DAT27 = AIB LEVACT
DAT28 = LEVARM

After signal ARE is true, CEINT is reset to return all signals to the state existing before start of the interrupt operation.

R/CEINT = ...

MODIFY AND TEST SEQUENCE. The sequence of operations for an interrupt that transfers control to a modify and test instruction is similar to the sequence that transfers control to an XPSD instruction and the associated service routine. When the interrupt circuit is waiting and enabled and has priority, INT is set, and the interrupt operations follow phase 10 of the program instruction. After INTRAP, INTRAP1, and INTRAP2 are set, the CPU clock is inhibited, the interrupt circuit is placed in the active state, and the contents of the memory location are extracted and executed, as described for the service routine sequence.

During a modify and test sequence of a counter interrupt, count-equals-zero signal S0031Z is sampled, and a count-equals-zero interrupt circuit may be placed in the waiting state (11X) if the register contains all zeros.

S/IS8 = IP8 SR8 (typical)
SR8 = CNTZREQ ISNIP2
CNTZREQ = FAMT PH2 INTRAP S0031Z
ISNIP2 = IS2 NIP2

The interrupt circuit is always transferred from the active state (101) to the armed state (011), because LEVARM is always true.

```
R/ISn    = NIPn NISNIPy IBO + ... (typical)
 IBO     = AIB LEVACT
 LEVACT  = FAMT PH2 INTRAP + ...
S/IPn    = ISn NISNIPy ARMOVD + ...
 ARMOVD  = IBO LEVARM
 LEVARM  = LEVACT N(FAPSD PH5 NR31)
```

After the modify and test instruction has been extracted from memory, it inhibits the CPU clock by setting CEINT,

```
S/CEINT = FAMT PH1 INTRAP + ...
```

enables the CPU clock by controlling AIB and AIE2,

```
(S/AIB)  = LEVACT NAIE2 + ...
 S/AIE2  = AIB LEVACT
 LEVACT  = FAMT PH2 INTRAP
```

and terminates the sequence by resetting INTRAP.

```
R/INTRAP = FAMT PH9
```

The modify and test sequence is controlled by the modify and test word instruction described in paragraph 3-69. The address of the next instruction in sequence is stored during PREP phases. Therefore, a modify and test sequence is a single-instruction interrupt.

EXTERNAL INTERRUPTS. External interrupts also control an interrupt circuit containing three flip-flops. The priority of an external interrupt depends upon cable connections with the CPU and the position of the external interrupt in the set of 16.

When an external interrupt is waiting and enabled and has priority, it will generate a true INT9 signal, and cause INT to be set, as for an internal interrupt. The true INT9 signal is generated by a DAT25 signal when no WD instruction in the interrupt mode is active.

```
INT9   = LINREQ NEWDM + ...
 LINREQ = /DAT25/
 EWDM   = NB16 NB17 NB18 B19 DIOWD
```

The address of the interrupt is transmitted over lines DAT16 through DAT24.

```
INT0   = LIN00 NEWDM + ...
  :        :
INT8   = LIN08 NEWDM
 LIN00 = /DAT16/
  :        :
 LIN08 = /DAT24/
```

Change of state of the external interrupt circuit from waiting and enabled (111) to active (101) is controlled by line DAT26.

```
DAT26 = AIE1 + ...
```

Change of state from active to armed (011) or disarmed (001) is controlled by lines DAT27 and DAT28.

```
DAT27 = AIB LEVACT + ...
DAT28 = LEVARM + ...
```

All external interrupts are inhibited if EI of the program status doubleword is set.

```
DAT29 = NEI NEWDM + ...
```

WRITE DIRECTION OPERATION. A write direction (WD) instruction can control the interrupt operation in two ways. When operating in the internal mode, it may control the states of flip-flops CIF, EI, and II, which may inhibit the priority chain signals. When operating in the interrupt mode, it enables signals DAT16 through DAT31, which are inputs to interrupt circuits 2 through 15. These inputs have the following general form, in which y = x + 14 for x = 2,3,...15.

```
S/INx  = DATy AEENLE
 DATy  = Sy EWDM
R/INx  = DATy ADBDB + REN
S/IPx  = DATy AEADB + ...
R/IPx  = DATy DARM + ...
S/ISx  = DATy IPx TRIG + ...
R/ISx  = DATy DARM + ...
```

These signals change the state of interrupt circuits when a WD instruction in the interrupt control mode (bits 16 through 19) presents a code (bits 21 through 23) addressed to any group (bits 28 through 31). The details of this operation are explained in the following paragraphs.

When a WD instruction in the interrupt control mode is executed, NDIOWD is reset and signal EWDM is true.

```
R/NDIOWD = FARWD PH1 OLD (WD instruction)
EWDM     = NB16 NB17 NB18 B19 DIOWD
           (Interrupt control mode 0001)
```

These signals initiate the sequence of operations illustrated in the timing diagram of figure 3-57.

During phase 3 of the WD instruction, NDIOFS is reset and CNA is set.

```
R/NDIOFS = FARWD PH3

S/CNA    = DIOFS EWDM NCNB
```

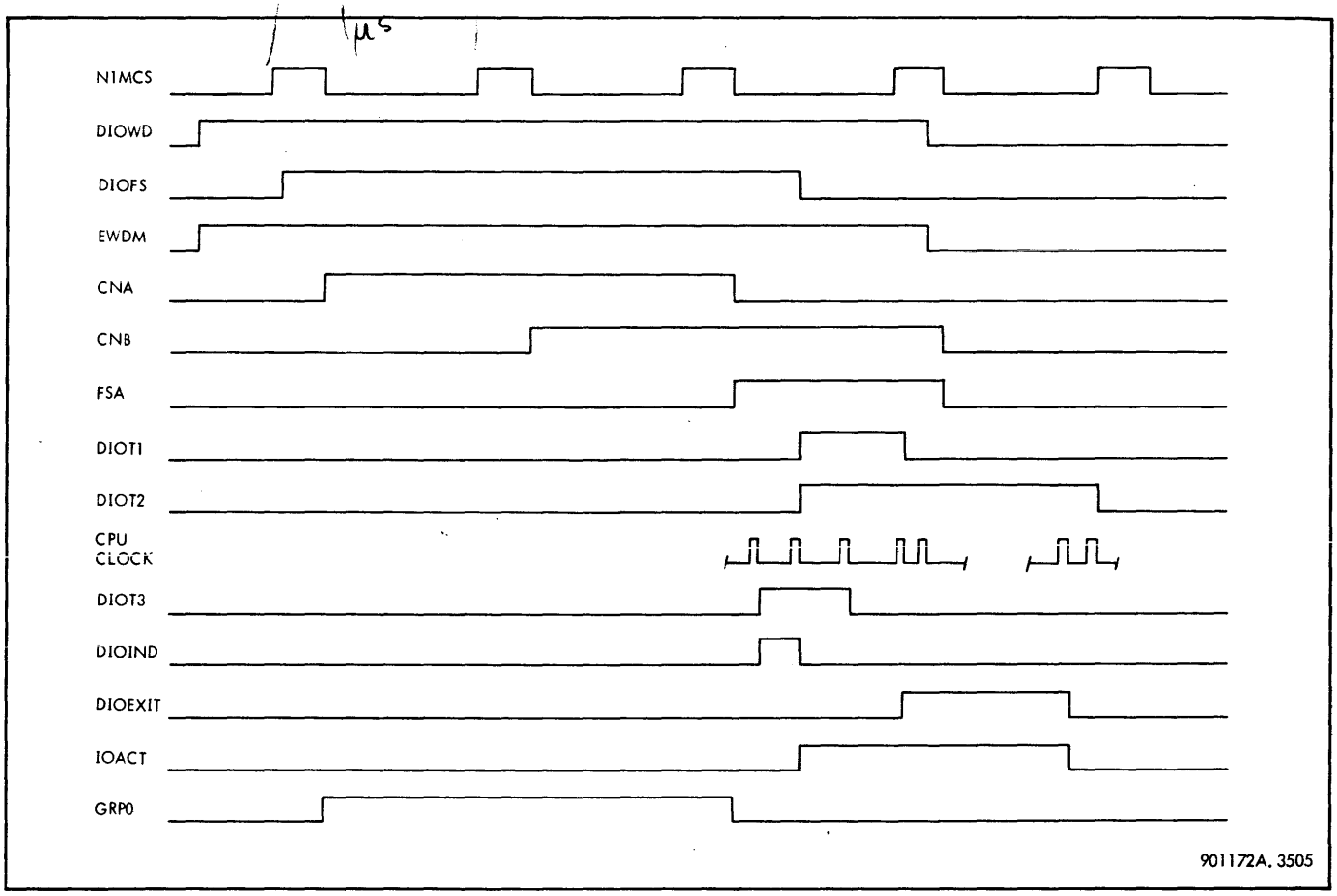


Figure 3-57. Write Direct Sequence, Timing Diagram

If the WD instruction is addressed to group 0, signals DAT16 through DAT29 control the interrupt circuits while CNA remains in the set state.

GRP0 = NB28 NB29 NB30 NB31 CNA (Group 0000)

AEENLE = NB23 GRP0 (Code XX0)

ADBDB = B21 NB22 GRP0 + NB21 B22 GRP0 (Code 10X + 01X)

AEADB = NB21 B22 GRP0 (Code 01X)

DARM = NB21 GRP0 (Code 0XX)

REN = B21 B22 NB23 GRP0 + RESET (Code 110)

TRIG = B21 B22 B23 GRP0 (Code 111)

The code stored in bits B21, B22, and B23 cause changes of state in the interrupt circuits as summarized in tables 3-13 and 3-14.

While signals DIOFS and EWDM are true, flip-flops CNA and CNB cycle through states (00, 10, 11, 01) at the 1-MHz clock rate. Return to state 00 cannot occur until signal NDIOFS is true.

S/CNA = DIOFS EWDM NCNB
R/CNA = ...
C/CNA = N1MCS
S/CNB = CNA
R/CNB = NDIOFS
C/CNB = N1MCS

Table 3-13. Function of Codes for WD Interrupt Control Mode

CODE			OPERATION
B21	B22	B23	
0	0	0	Undefined
0	0	1	Disarm all levels selected by a 1; all levels selected by a 0 are not affected

(Continued)

Table 3-13. Function of Codes for WD Interrupt Control Mode (Cont.)

CODE			OPERATION
B21	B22	B23	
0	1	0	Arm and enable all levels selected by a 1; all levels selected by a 0 are not affected
0	1	1	Arm and disable all levels selected by a 1; all levels selected by a 0 are not affected
1	0	0	Enable all levels selected by a 1; all levels selected by a 0 are not affected
1	0	1	Disable all levels selected by a 1; all levels selected by a 0 are not affected
1	1	0	Enable all levels selected by a 1 and disable all levels selected by a 0
1	1	1	Trigger all levels selected by a 1. All such levels that are currently armed advance to the waiting state. Those levels currently disarmed are not altered, and all levels selected by a 0 are not affected

Table 3-14. Signals Enabled by Codes for WD Interrupt Control Mode, and Resulting Changes of State

CODE	TRUE CONTROL SIGNALS	CHANGE OF STATE (ISn, IPn, INn)	
		DATy = 1	DATy = 0
001	DARM	XXX → 00X	No change
010	AEENLE, ADBDB, AEADB, DARM	XXX → 011	No change
011	ADBDB, AEADB, DARM	XXX → 010	No change
100	AEENLE, ADBDB	XXX → XX1	No change
101	ADBDB	XXX → XX0	No change

Table 3-14. Signals Enabled by Codes for WD Interrupt Control Mode, and Resulting Changes of State (Cont.)

CODE	TRUE CONTROL SIGNALS	CHANGE OF STATE (ISn, IPn, INn)	
		DATy = 1	DATy = 0
110	AEENLE, REN	XXX → XX1	XXX → XX0
111	TRIG (if IPn)	X1X → 11X	No change
	(if NIPn)	X0X → X0X	

When CNA and CNB reach the 01 state, they generate a true FSA signal (function strobe acknowledge) which sets DIOT3.

FSA = NCNA CNB + ...

S/DIOT3 = FSA + ...

Flip-flops DIOT1, DIOT2, and DIOT3 cycle through a sequence (000, 001, 111, 110, 010) and wait for a false IOACT (input/output active) signal.

S/DIOT1 = DIOIND

DIOIND = NDIOT2 DIOT3

S/NDIOFS = DIOIND + ...

R/DIOT1 = NDIOT3

S/DIOT2 = DIOT1 + DIOIND

R/DIOT2 = NIOACT

S/DIOT3 = DIOIND + ...

R/DIOT3 = ...

C/DIOT1 = C/DIOT2 = C/DIOT3 = CL (CPU clock rate)

When these flip-flops reach state 010, they generate a true DIOEXIT signal and set NDIOWD.

DIOEXIT = NDIOT1 DIOT2

S/NDIOWD = DIOEXIT + ...

3-31 MEMORY

3-32 Introduction

The Sigma 5 memory has a maximum storage capability of 131,072 33-bit words. Physically, a memory of this size occupies eight separate frames mounted in four memory cabinets. The total memory size of any Sigma 5 computer can range from 4K to 128K words in increments of 4K. The abbreviations for memory sizes (4K, 8K, 16K, 32K, 64K, etc.) are used for convenience throughout this manual. The factor K is equal to 1024; thus, for example, a 128K memory contains 131,072 words.

The various standard and optional units that make up the total Sigma 5 memory are listed in table 3-15.

Figure 3-58 shows the interconnection for eight memory banks and three ports that make up the total Sigma 5 memory system consisting of one CPU and two input/output processors.

3-33 Memory Bank

Figure 3-59 shows a functional block diagram of a memory bank. A sigma 5 computer system can have up to eight of these memory banks, each bank containing from 4K to 16K words in increments of 4K. This diagram also shows the ports (A, B, and C) through which data, address, and control signals flow.

The magnetics section contains the following:

a. Ferrite cores

b. Decoding logic

c. Current and voltage switches

d. Current and voltage predrivers

e. Sense amplifiers

Data is stored in the ferrite cores. The decoding logic, electronic switches, drivers, and sense amplifiers are used to put data into the cores and to read the data out of the cores.

MEMORY PORTS. The memory ports provide a means of accessing memory from different sources. The standard Sigma 5 computer is provided with one port (port C) through which the CPU (and the integral IOP) accesses memory. A second port (port B) and a third port (port A) may be added as options to provide memory access by input/output device controllers and input/output devices via multiplexing or selector IOP's.

The L-register holds address information fed through the port address paths. Addresses are fed through the ports as follows:

a. LA15 through LA31 are fed through the port A address path to the L-register.

b. LB15 through LB31 are fed through the port B address path to the L-register.

c. LC15 through LC31 are fed through the port C address path to the L-register.

Table 3-15. Sigma 5 Memory Models and Options

Model	Description	Prerequisite	Maximum Number Required
8251	4K Memory, Single Access (Port C)	8201	8
8252	4K-8K Memory Expansion	8251	8
8252	8K-12K Memory Expansion	8252	8
8252	12K-16K Memory Expansion	8252	8
8255	Two-Way Access (Port B)	8251	8
8256	Three-Way Access (Port A)	8255	8
8257	Port Expander F (First) (Six-Way Access, One Memory)	8256	4
8257	Port Expander S (Second) (One Memory-Two Memory Six-Way Access Expander)	8257 (F)	4

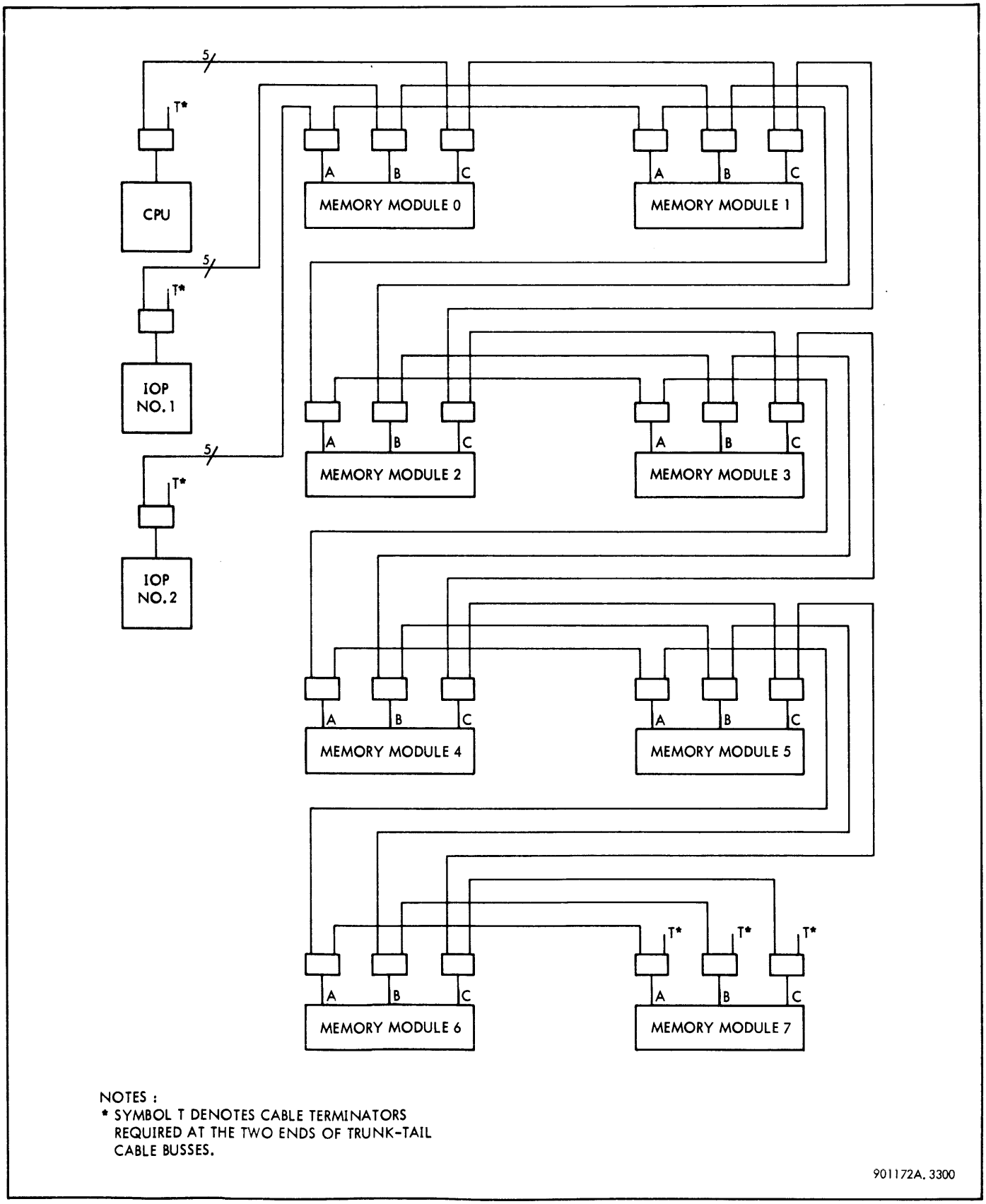


Figure 3-58. Memory System Interconnection for Eight Memory Modules, One CPU, and Three IOP's

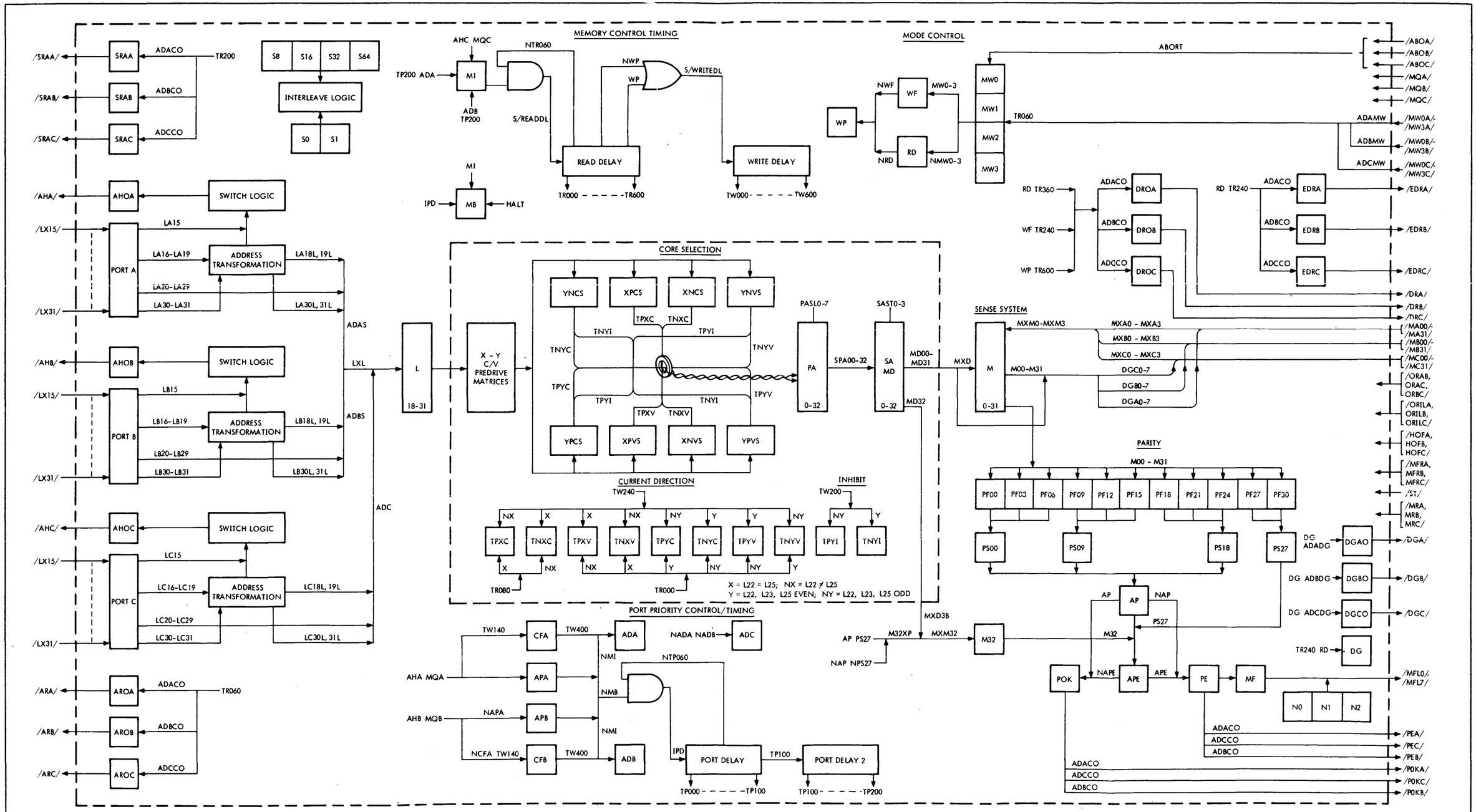


Figure 3-59. Sigma 5 Memory Bank, Functional Diagram

901172A.3301

Before data is processed and stored in the magnetics section, the data is transferred to the M-register. For example, for data to be stored, the sequence occurs as follows:

a. MA00 through MA31 are fed through the port A data path to the M-register. The data then goes from the M-register to the magnetics section.

b. MB00 through MB31 are fed through the port B data path to the M-register. The data then goes from the M-register to the magnetics section.

c. MC00 through MC31 are fed through the port C data path to the M-register. The data then goes from the M-register to the magnetics section.

Similarly, data read out of the magnetics section goes through the ports via the M-register. Parity is generated in the section labeled Parity.

Control logic is contained in the central control section and individual port controls (port A control, port B control, and port C control). The logic controls port priority. In case of memory access conflict, port A has the highest priority, port B second highest, and port C the lowest.

Signals fed into the memory via the ports are shown in the functional block diagram, figure 3-59. Note that the last letter of a signal in the block diagram usually indicates in which port the signal originates. For example, signal AHA comes from port A logic, AHB comes from port B logic, and AHC comes from port C logic. Exceptions are signals ORBC, ORAC, and ORAB, which are port override signals. Signal ORBC, for example, is fed to port A to override ports B and C.

PORT EXPANDERS. A port expander unit accepts up to four input buses and connects to a memory port to expand that port from one to four inputs specified as 0, 1, 2, and 3. A port expander can be connected to either port A or port B, but not both, in a Sigma 5 memory. The port expander must provide address modification for each of its four inputs so that the memory may be assigned independent addresses for each input bus. The four inputs to the expander have a fixed priority relationship for the resolution of access request conflicts in decreasing numerical order.

Figure 3-60 shows a port expander connected to port A of memory banks 0 and 1. Port expander F is connected to bank 0, and port expander S is connected to bank 1.

3-34 Interleaving

Address interleaving between any two or more memory banks in a Sigma 5 system exists whenever the INTERLEAVE SELECT switch on the PCP is in NORMAL position and certain addressing constraints have been met. The objective of interleaving is to obtain a faster average access time for a sequence of addresses. With interleaving in effect, no two consecutive addresses will reside in the same memory bank. Since each memory bank is independent of the others, memory access to two or more modules simultaneously is possible. This simultaneous access to memory is common between the CPU and the I/O channels. Whenever addressing conflict occurs, as when two separate sources attempt to access the same bank at the same time, access is granted on a port priority basis with port A having the highest priority, port B the next highest priority, and port C the lowest priority.

Each memory bank is assigned a set of addresses to which it responds. As viewed from any of the memory ports, each memory bank may have a different set of addresses. In general, however, each bank is assigned the same addresses for each port to which it is attached.

The basic interleaving constraints are:

a. The starting address of a memory bank must be a multiple of the bank size.

b. The total interleaved memory must be on its own boundary.

c. The starting addresses for each bank must be assigned so that no gaps or overlaps exist in the address field for the noninterleaved mode.

d. The total interleaved memory size must be 8K, 16K, 32K, or 64K. Interleaving cannot extend from the first 64K memory into the second 64K memory.

e. No more than four banks can be interleaved.

f. 12K banks cannot be interleaved, and their starting addresses must begin on an integral boundary of 16K.

Any combination of memory banks that satisfies the above constraints can be interleaved. The interleaved address field will cover the same range as the noninterleaved field. All interleaving capabilities are nullified when the INTERLEAVE SELECT switch on the PCP is placed in the DIAGNOSTIC position.

3-35 Memory Elements

The elements making up the total memory are defined as follows:

MEMORY. A memory consists of the total number of memory words in a Sigma 5 system. The minimum memory consists of 4K words, the maximum memory consists of 128K words.

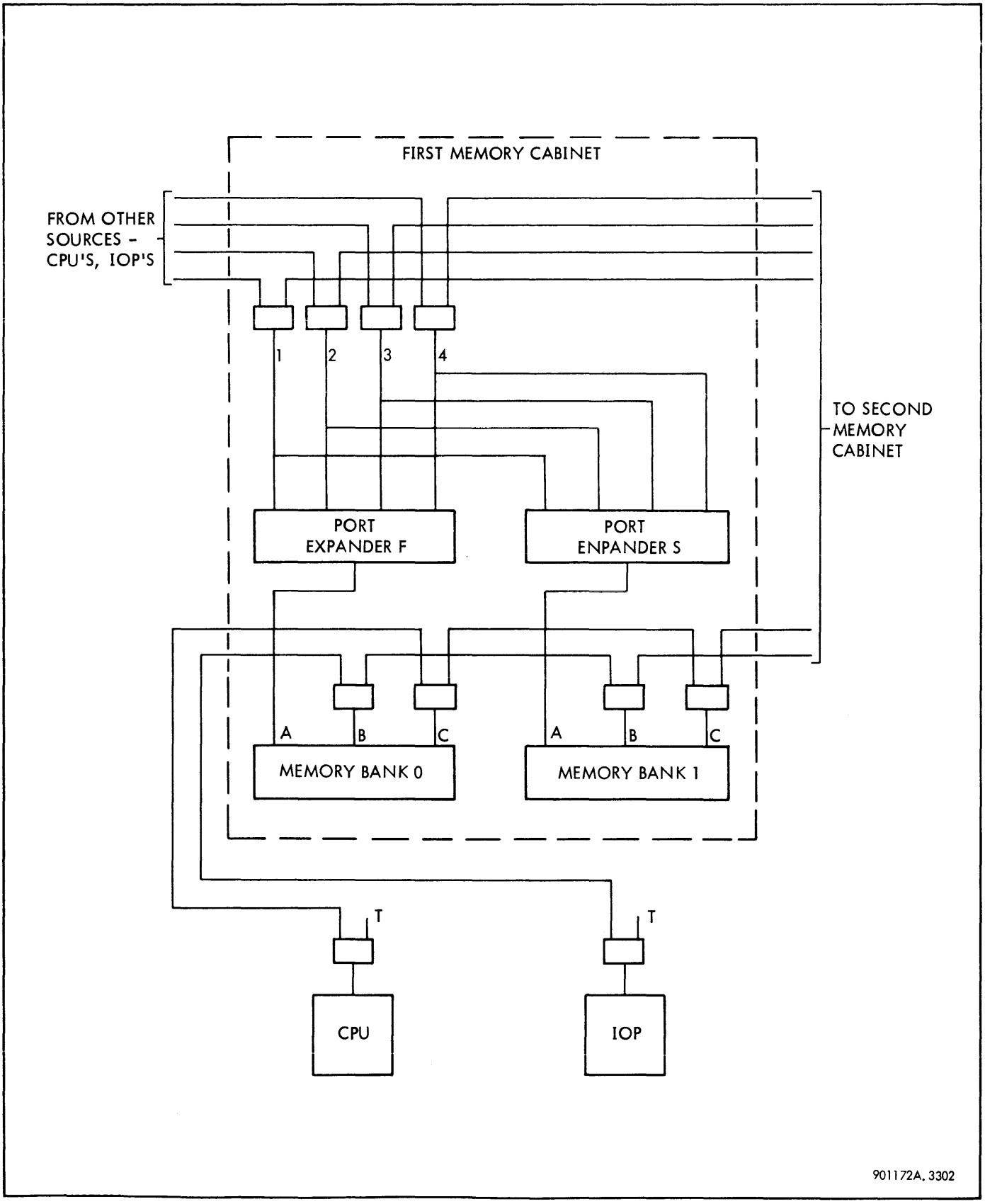


Figure 3-60. Port Expanders F and S (First and Second)

BANK. A memory bank is a complete and independent memory unit and consists of from one to four memory stacks located in a single memory frame. The memory bank is made up of 4 to 16 core diode modules plus other control and timing electronics. A memory bank is mounted on four wired backboards together with a PT16 logic supply and a PT17 memory supply side-mounted to the frame. All memory banks are wired in exactly the same way and differ only in the complement of core diode modules (or stacks) which are mounted on the frame.

STACK. A memory stack is the smallest memory increment. It consists of 4096 (4K) words of core memory mounted on four core diode modules.

3-36 Memory Switches

Several toggle switches are associated with each memory bank. These switches, mounted on ST14 switch modules, are set to designate the bank number, the total interleave memory size, the memory bank size, and ports A, B, and C starting addresses for each memory bank. (See figure 3-61.)

BANK NUMBER SWITCHES. Three bank number switches, N0, N1, and N2, are provided on each frame. These switches are set to a binary configuration representing the number assigned to that bank and are associated with the number of the memory fault light appearing on the PCP. A maximum of eight memory banks can be incorporated in a Sigma system. These banks are assigned numbers 0 through 7, representing all the combinations of the three toggle switches. In general, memory banks in the left-most memory cabinet are assigned numbers 0 and 1; the next memory cabinet to the right contains banks 2 and 3, and so on until all banks have been assigned numbers.

BANK SIZE SWITCHES. Bank sizes are available in 4K, 8K, 12K, and 16K words. Two toggle switches, S0 and S1, are provided on the switch module in each bank for identifying memory size. These switches must be set to the number corresponding to the bank size. The binary configuration 00 represents 4K, 01 represents 8K, 10 represents 12K, and 11 represents 16K.

STARTING ADDRESS SWITCHES. Five toggle switches, S15 through S19, are provided in each memory bank for each port that the bank contains. These five switches are set to represent the five most significant bits of the starting address contained in that bank. The five most significant bits of the bank address are address lines L15 through L19.

INTERLEAVE SIZE SWITCHES. Each memory bank has four toggle switches to designate the total interleaved memory size. These switches, S64, S32, S16, S8, are used to indicate the total size of the memory to be interleaved. Only one of these switches can be true at the same time since only 8K, 16K, 32K, or 64K size memories can be interleaved.

PORT EXPANDER SWITCHES. Each bank has a port expander switch for port A and another port expander switch for port B. If a port expander is connected to either one of these ports, its port expander switch must be set to a one; otherwise, the port expander switches must be set to zero.

3-37 Memory Configuration

Many Sigma memory configurations are possible. Figures 3-62 through 3-64 show these examples of several possible combinations of an interleaved memory, their physical placements in frames and cabinets, their interleaving capabilities, and their corresponding switch settings. Note that there is no fixed and arbitrary relationship between the memory addresses and their physical placement. However, it is general practice to designate the banks in the left-most cabinet (cabinet 1) as banks 0 and 1; the banks in the cabinet to the right (cabinet 2) has modules 2 and 3, and so on until all memory banks have been assigned numbers. Note that the example shown in figure 3-64 does not follow this convention. It would be preferable to locate the two 8K banks in cabinet 1 and the 12K bank in cabinet 2. It is not possible, in this example, to assign the 12K bank any number other than bank 2.

3-38 Interleave Transformation

With interleaving of memory addresses in effect, the port address lines are transformed by exchanging two of address bits 16, 17, 18, and 19 with address bits 30 and 31, depending upon the configurations of the bank size switches, the interleave size switches, and the bank number switches. (See figure 3-65.) The discussion of interleaving in the following paragraphs is limited to port C, although all statements apply to ports A and B as well.

Interleaving occurs only when the INTERLEAVE SELECT switch on the PCP is in the NORMAL position. When this switch is in DIAGNOSTIC, interleaving is inhibited. The override interleave signal, ORIL, is derived from the INTERLEAVE SELECT switch, and is true with the switch in the DIAGNOSTIC position.

Figure 3-66 shows a simplified diagram of how the address lines of port C, LC15 through LC31, are transformed to the interleaved address that selects the memory bank and the X and Y predrive selection circuits of core memory. The address bit exchanges that perform the interleave address transformation are indicated in table 3-16. Detailed interleave transformation logic for port C is shown in figure 3-67.

Table 3-16. Interleaving Address Bit Exchange

Memory Interleave Size	Bank Size	Address Bit Exchange
8K	4K (NS0 NS1)	19↔31
16K	4K (NS0 NS1)	19↔30, 18↔31
	8K (NS0 S1)	, 18↔31
32K	4K (NS0 NS1)	17↔31, 18↔30
	8K (NS0 S1)	17↔31, 18↔30
64K	16K (S0 S1)	17↔31
	16K (S0 S1)	17↔30, 16↔31

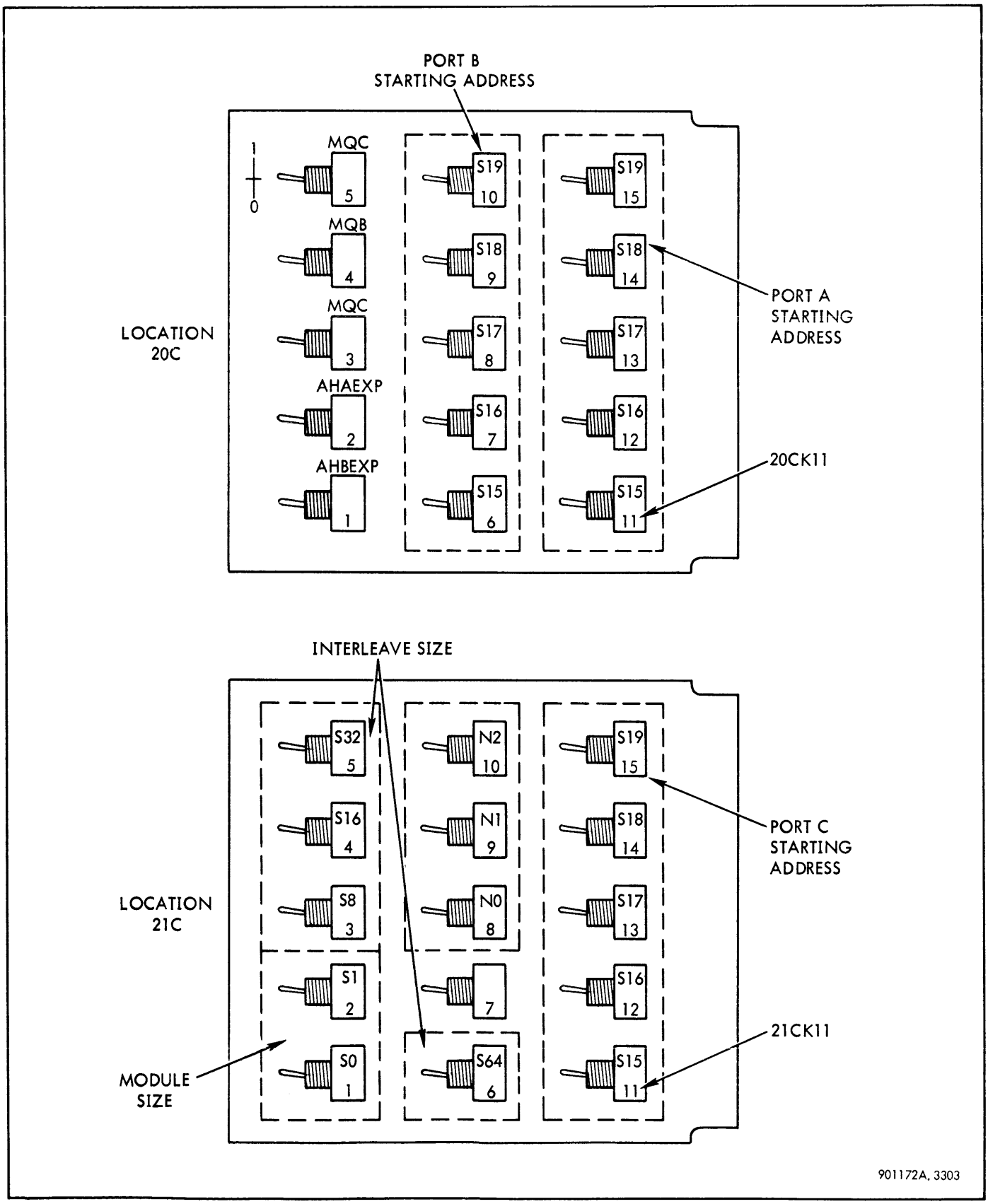
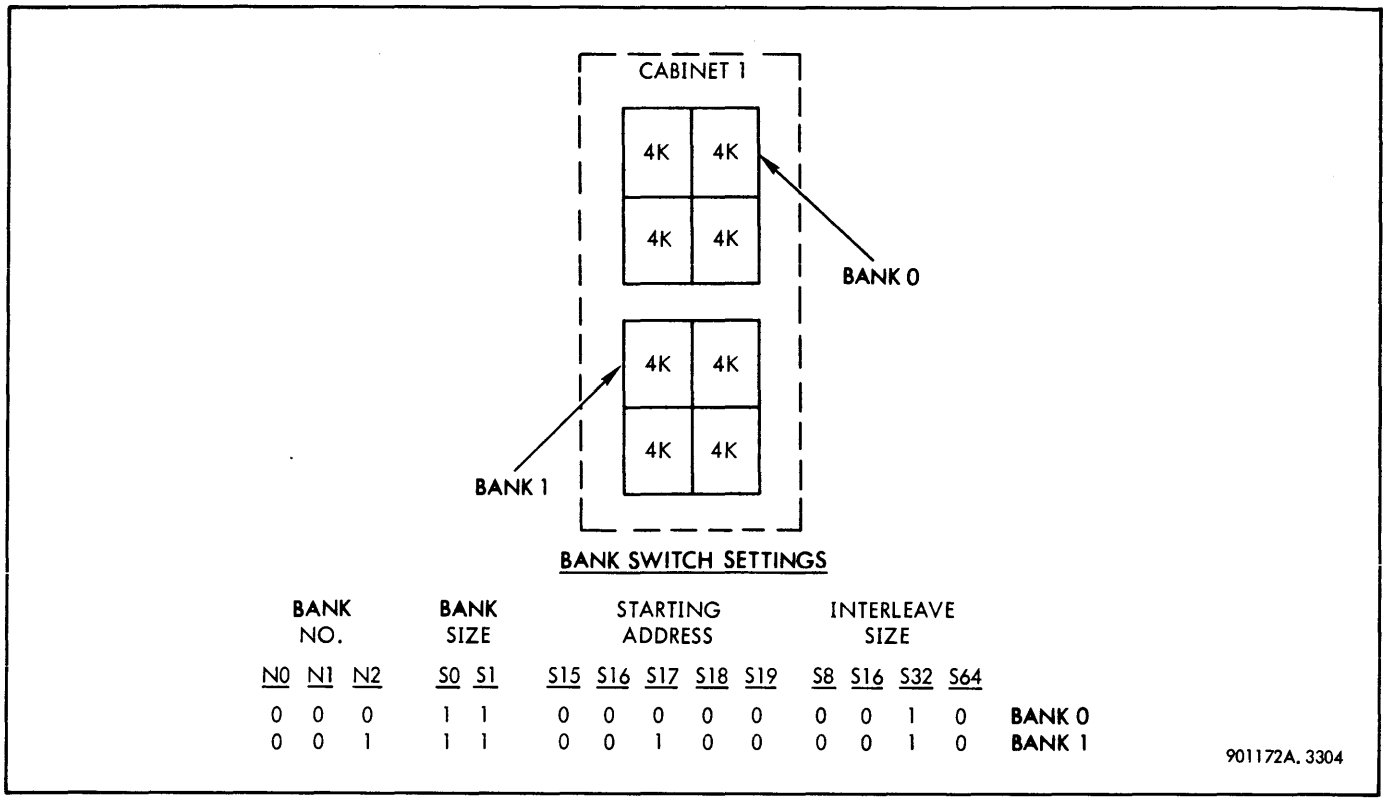


Figure 3-61. Toggle Switch Modules (ST14)

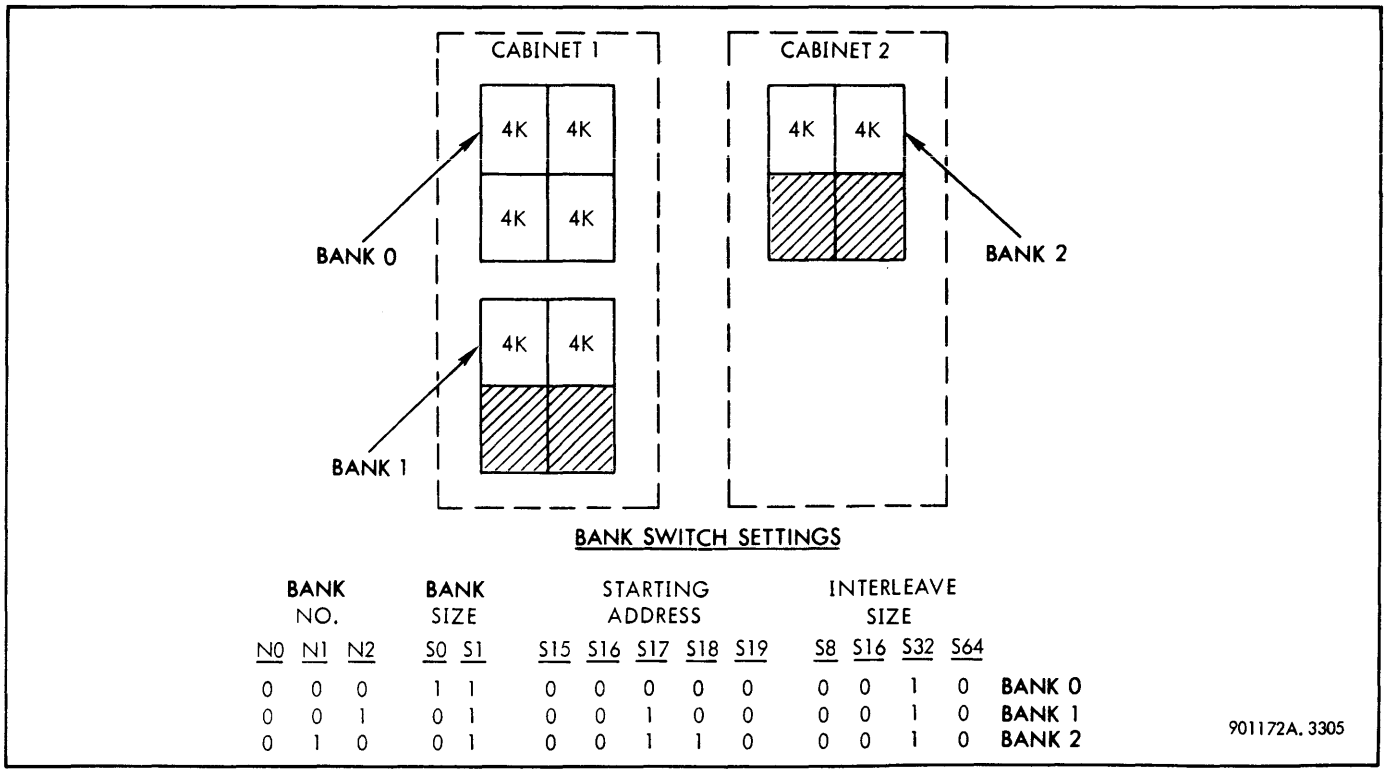


BANK SWITCH SETTINGS

BANK NO.			BANK SIZE		STARTING ADDRESS					INTERLEAVE SIZE				
N0	N1	N2	S0	S1	S15	S16	S17	S18	S19	S8	S16	S32	S64	
0	0	0	1	1	0	0	0	0	0	0	0	1	0	BANK 0
0	0	1	1	1	0	0	1	0	0	0	0	1	0	BANK 1

901172A. 3304

Figure 3-62. 32K Interleaved Memory, Example 1

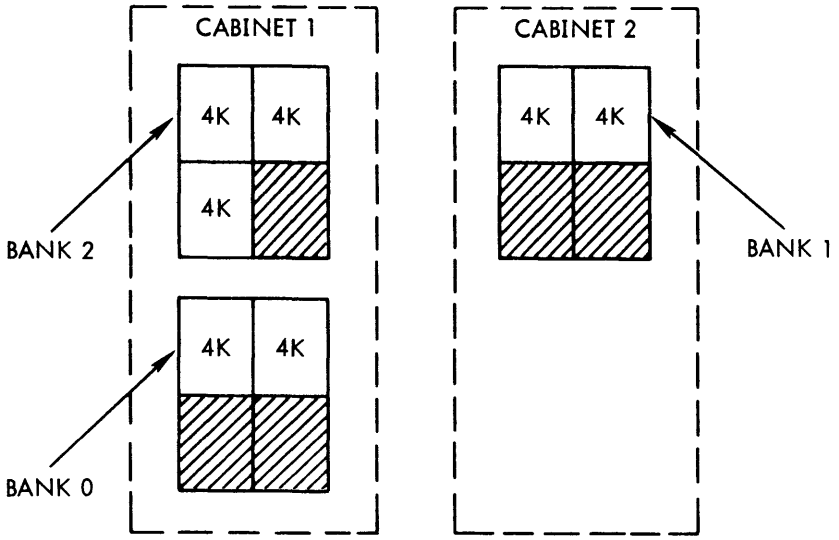


BANK SWITCH SETTINGS

BANK NO.			BANK SIZE		STARTING ADDRESS					INTERLEAVE SIZE				
N0	N1	N2	S0	S1	S15	S16	S17	S18	S19	S8	S16	S32	S64	
0	0	0	1	1	0	0	0	0	0	0	0	1	0	BANK 0
0	0	1	0	1	0	0	1	0	0	0	0	1	0	BANK 1
0	1	0	0	1	0	0	1	1	0	0	0	1	0	BANK 2

901172A. 3305

Figure 3-63. 32K Interleaved Memory, Example 2



BANK SWITCH SETTINGS

BANK NO.			BANK SIZE		STARTING ADDRESS					INTERLEAVE SIZE				
N0	N1	N2	S0	S1	S15	S16	S17	S18	S19	S8	S16	S32	S64	
0	0	0	0	1	0	0	0	0	0	0	1	0	0	BANK 0
0	0	1	0	1	0	0	0	1	0	0	1	0	0	BANK 1
0	1	0	1	0	0	0	1	0	0	0	0	0	0	BANK 2

901172A.3306

Figure 3-64. 32K Interleaved Memory, Example 3

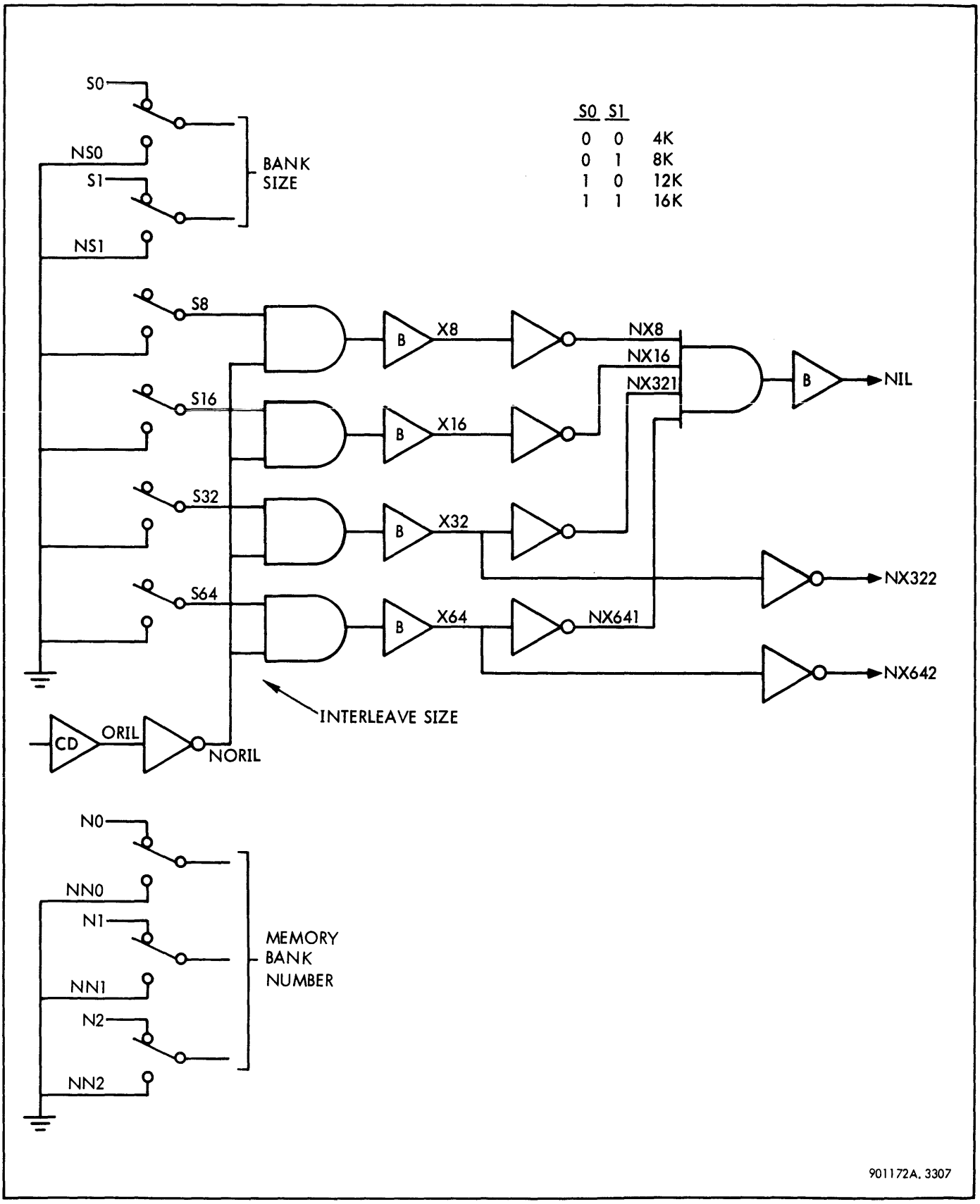


Figure 3-65. Bank Size, Interleave Size, and Bank Number Switches

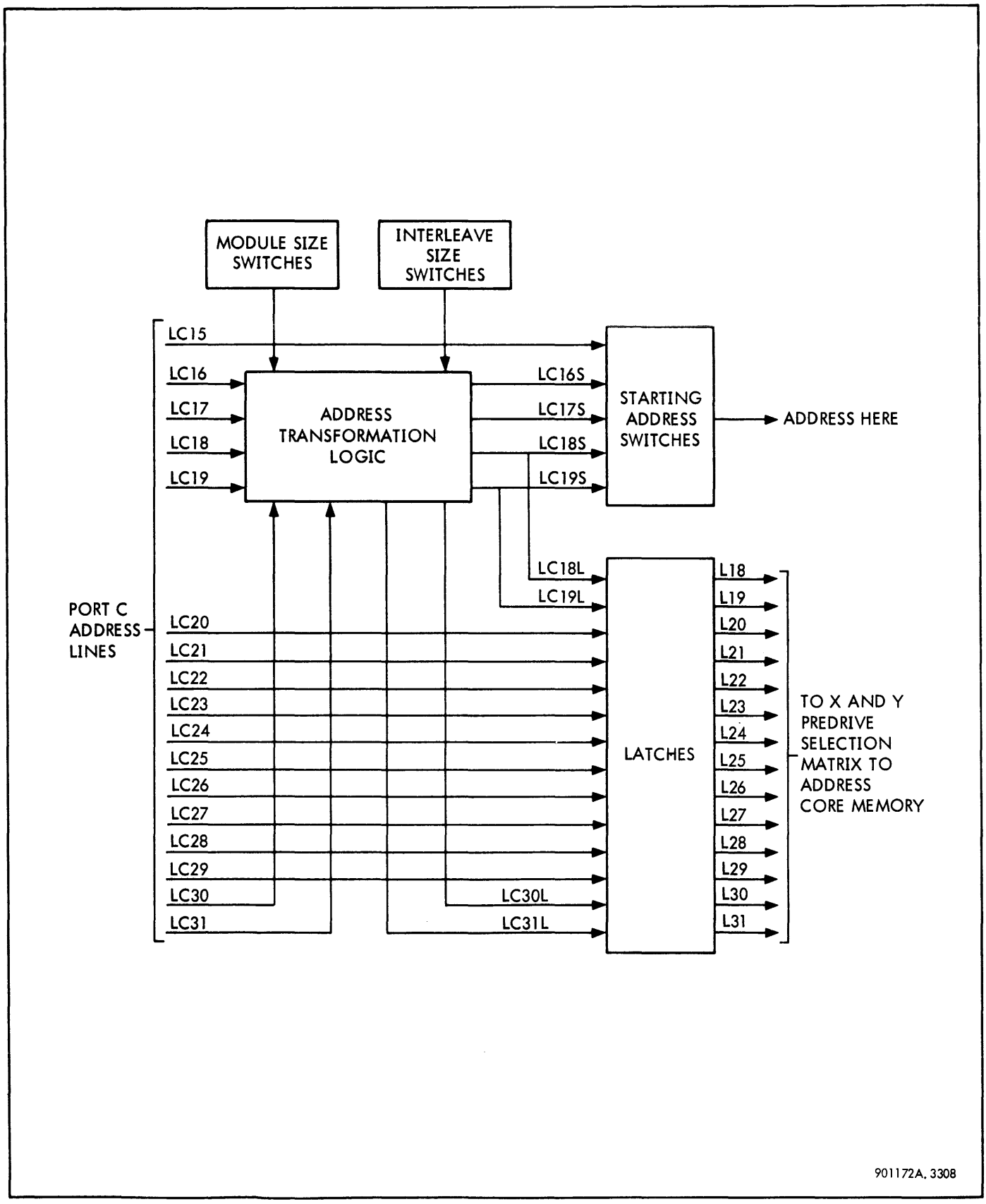


Figure 3-66. Address Transformation for Interleaving (Port C), Simplified Diagram

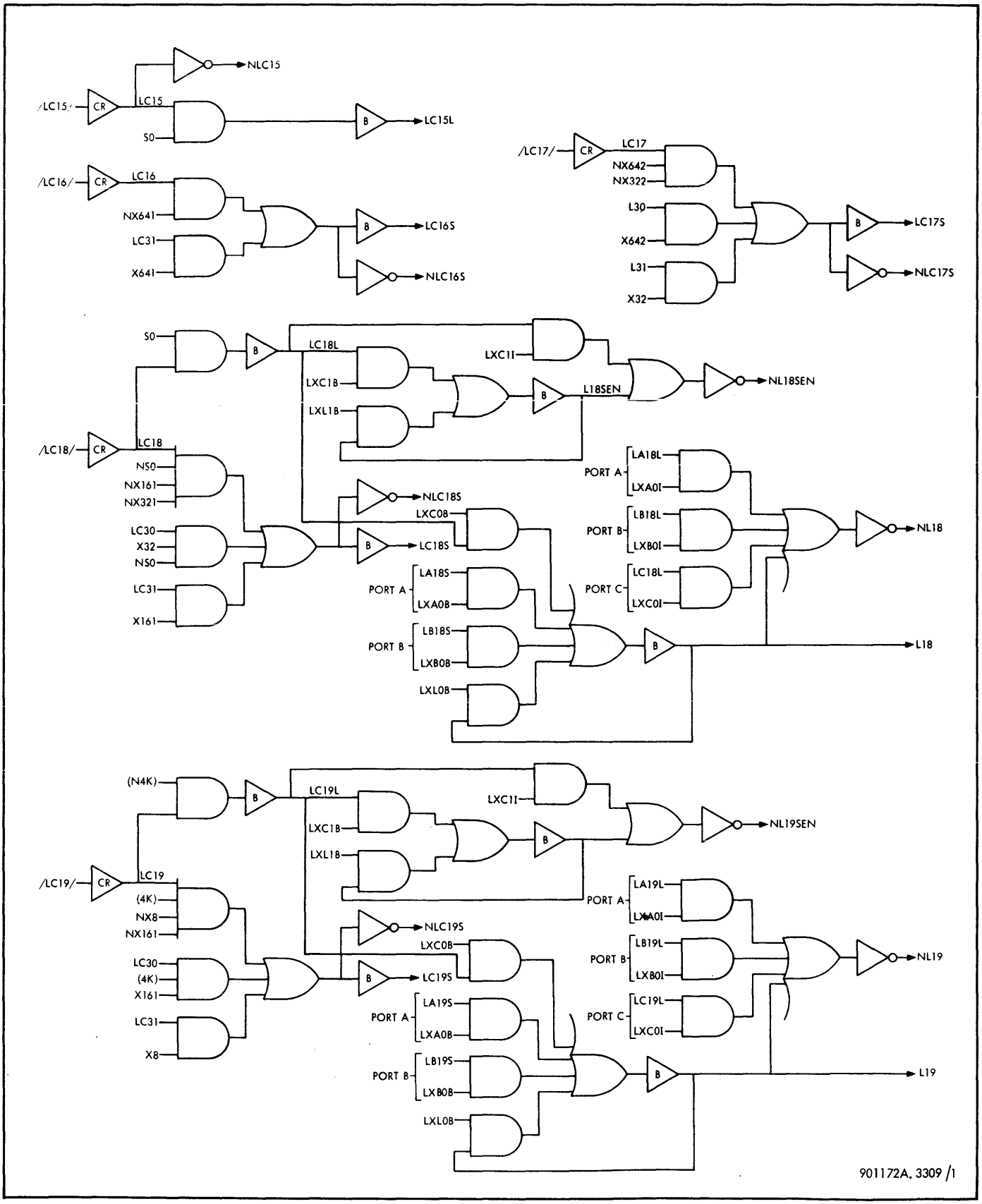


Figure 3-67. Memory Address Register and Interleave Transformation Logic (Sheet 1 of 2)

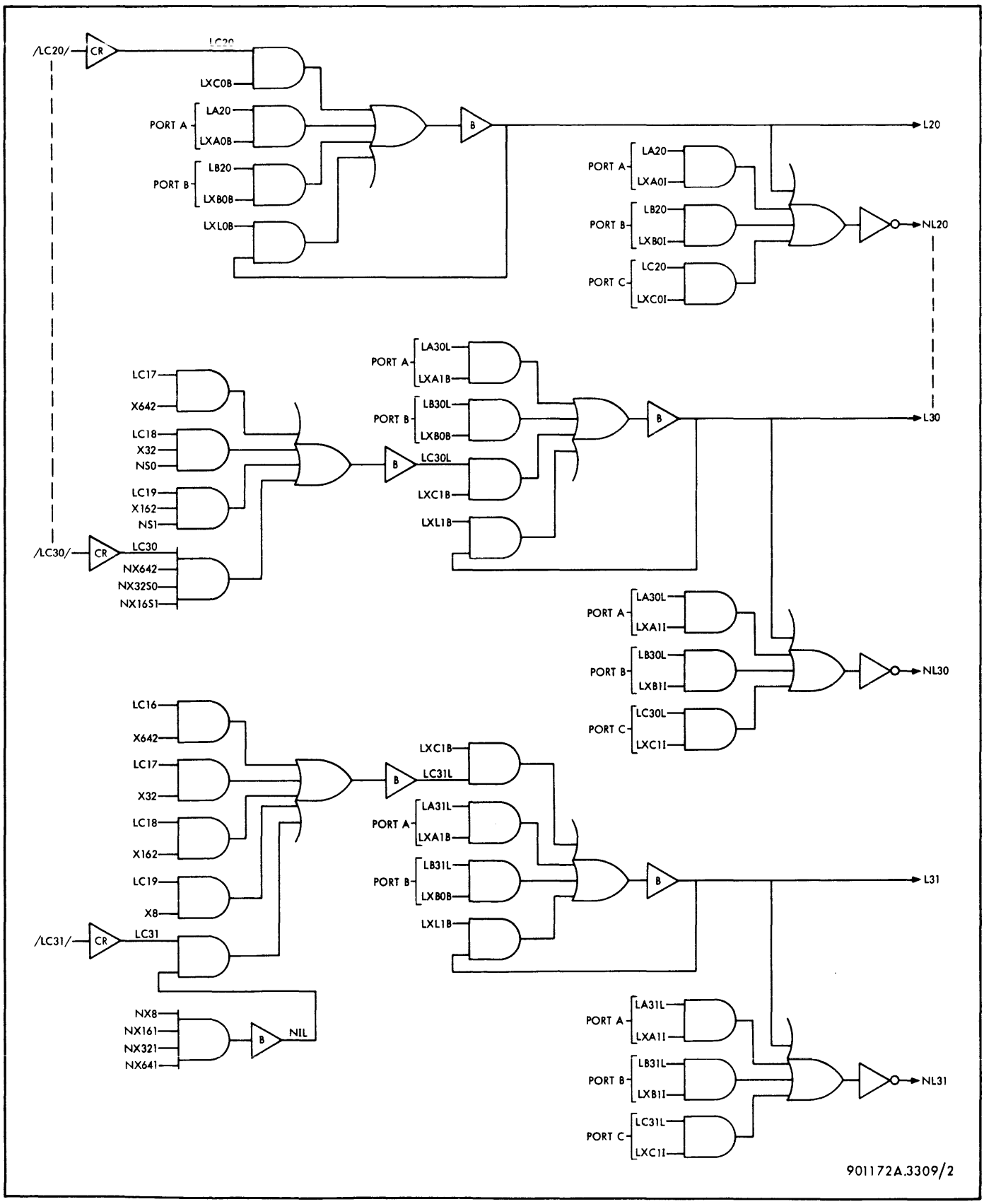


Figure 3-67. Memory Address Register and Interleave Transformation Logic (Sheet 2 of 2)

ADDRESS HERE. After address transformation takes place, the five most significant address bits are compared with the configuration set into the starting address switches for each port. These five address bits make a valid comparison with the starting address switches in only one bank. (See figure 3-68.)

Signal AHC is returned to the source requesting memory access to indicate that the address exists. Note that if address lines L18 and L19 are true in a 12K memory bank, the address here signal will be inhibited.

3-39 Memory Access Request

The memory request interface signals, /MQA/, /MQB/, and /MQC/ are initiated by the source through either ports A, B, or C. After these signals are received by the memory bank, they are subjected to port override logic gating. (See figure 3-69.) The port override signals are available for special system uses to allow any designated port access to memory at the exclusion of the other two ports.

The port override signal, ORAB, when true, allows a request from port C to be initiated, but denies all access to ports A and B. Signal ORAC, when true, allows a request from port B to be initiated, but denies all access to ports A and C. Signal ORBC, when true, allows a request for port A to be initiated, but denies all access to ports B and C.

3-40 Port Priority

The three ports – A, B, and C – are assigned priority in alphabetic sequence. Port A has the highest priority, port B has the next highest, and port C has the lowest. Ports A and B are called the slow ports because of delays involved in assigning priority before the memory cycle. Port C is called the fast port because the logic is designed to favor it. In the absence of requests from either port A or port B, the logic is already set up to handle a request from port C; that is, signal ADC is normally true when no requests are present from ports A or B.

Ports A and B have two separate logic paths by which priority is assigned. One logic path is used when the memory is idle or not busy (NMB). The other logic path is used when requests arrive while the memory is busy processing a previous request (MB).

Port priority logic is shown in figure 3-70. Signals ADA, ADB, and ADC establish priority for ports A, B, and C, respectively. If neither port A nor port B is requesting access to memory, signals ADA and ADB are false, forcing signal ADC true. (See figure 3-70.)

ADC = NADA NADB

The two separate logic paths previously mentioned by which priority between ports A and B is established are shown in figure 3-70 A and B. The first path (logic for APA and APB) is used when a request from port A or port B is made and the memory is not busy processing a previous request (NMB).

APA = AHAEXP MQAT NMB NCFA NCFB

APB = AHBEXP MQBT NMB NCFA NCFB NAPA

Note that if APA is true, APB is forced false, thus establishing port A priority over port B.

The second path (logic for CFA and CFB) is used when a request from port A or port B is made while the memory is busy processing a previous request (MB).

CFA = AHAEXP MQAT TW320 NTW360

CFB = AHBEXP MQBT TW320 NTW360 NCFA

Note that if CFA is true, CFB is forced false, thus establishing port A priority over port B.

3-41 Address Release

After a memory request has been made and port priority established, the memory cycle is initiated.

MI = AHC MQC NMB NAB
+ TP200 (ADA + ADB)
+ NMB (CFA + CFB)

At this point, the source making a request has not been informed whether its request has been accepted or not. Sixty nanoseconds after the memory cycle has been initiated, the address release signal for the active port is raised. (See figure 3-70.)

/ARA/ = AROA = ADACO TR060

/ARB/ = AROB = ADBCO TR060

/ARC/ = AROC = ADCCO TR060

It is this signal that informs the source requesting memory access that its request has been honored and that it may now release its address.

3-42 Memory Cycles

Information is transmitted to or accepted from memory in the form of words accompanied by byte presence indicators. Parity checking and generation is provided for all memory operations on a word basis. Thus, the Sigma 5 has three modes of operation:

a. Read-restore

b. Full clear-write

c. Partial clear-write

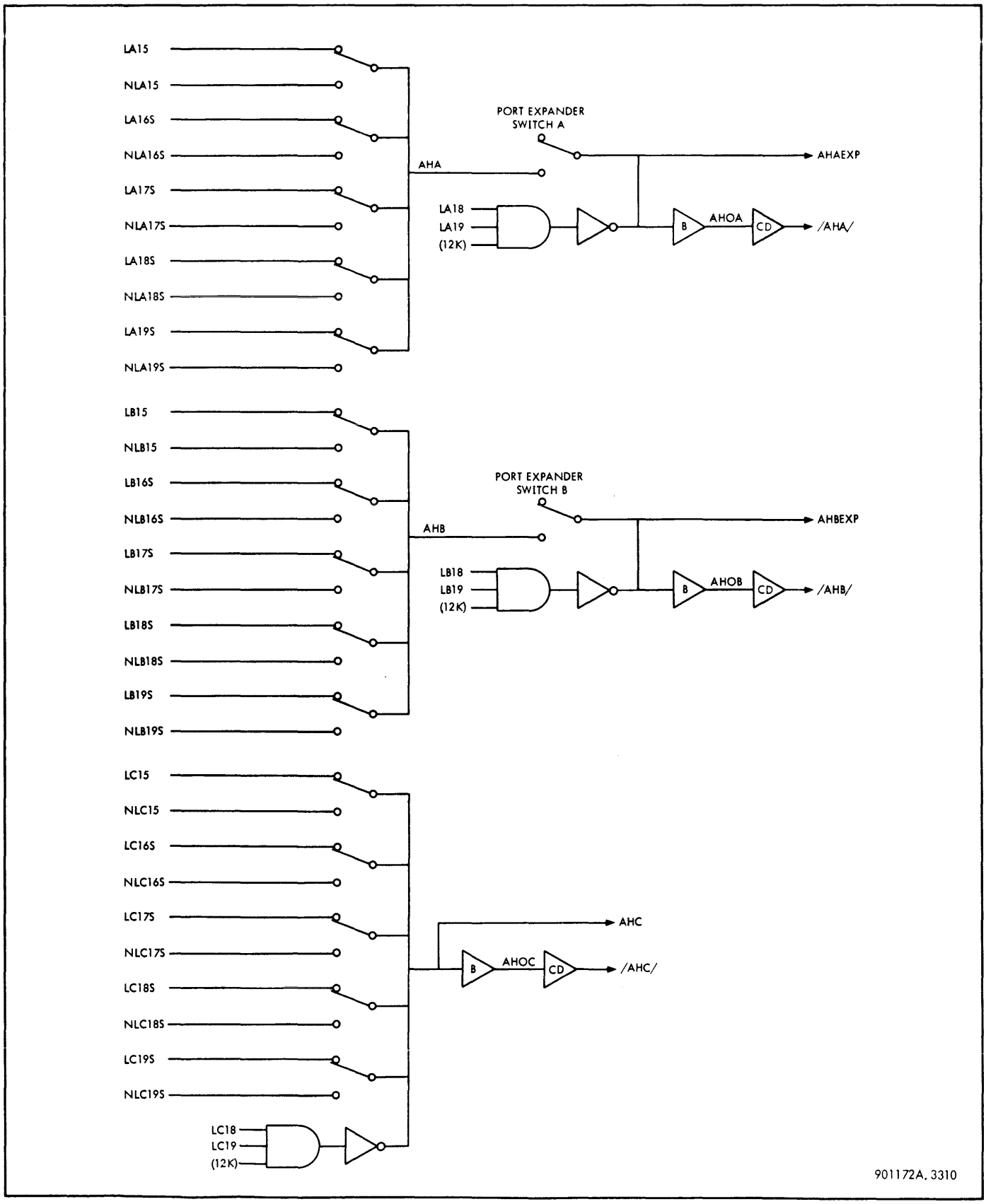


Figure 3-68. Address Here Logic, Ports A, B, and C

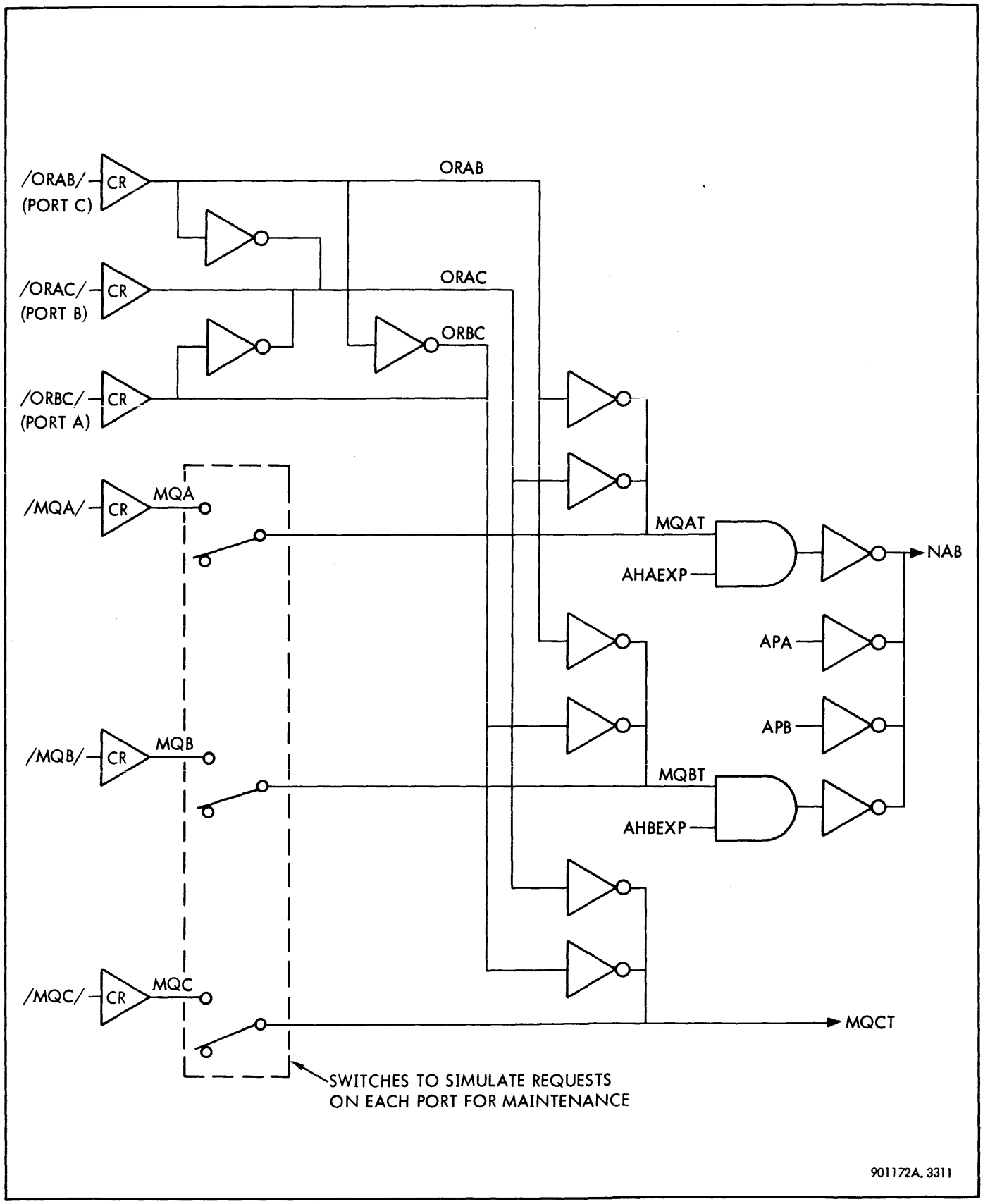


Figure 3-69. Memory Request and Port Override Logic

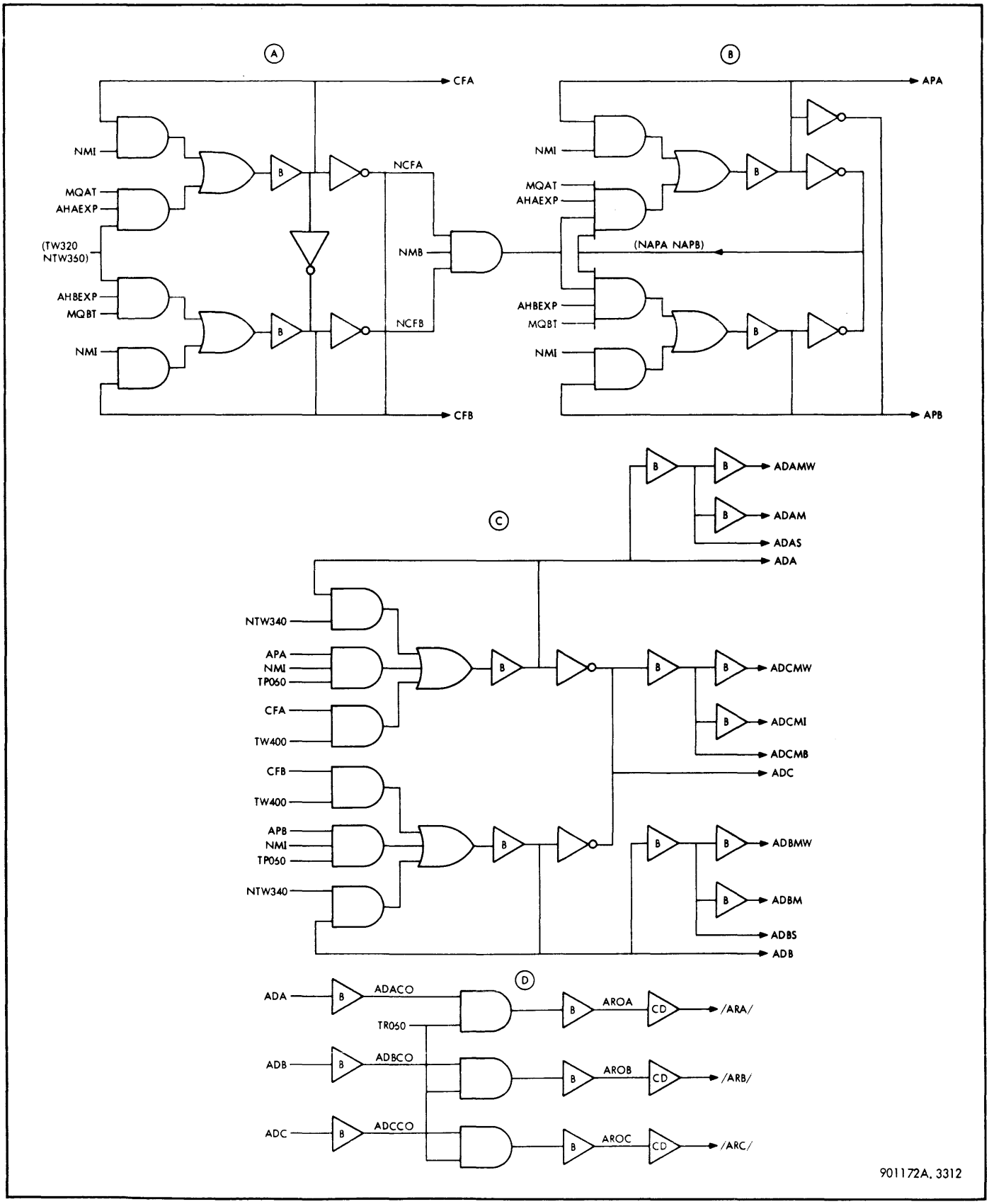


Figure 3-70. Port Priority and Address Release Logic

READ-RESTORE. The read-restore operation consists of:

a. Reading a word from a specified address in memory.

b. Gating the word into the M-register.

c. Checking parity.

d. Writing (restoring) the word that has just been read and gated into the M-register back into memory. The word is written back into the same address from which it was extracted.

FULL CLEAR-WRITE. A full clear-write operation consists of:

a. Clearing the memory location. This is done by reading the word but not gating it into the M-register.

b. Placing a new word into the M-register.

c. Writing the new word into memory.

PARTIAL CLEAR-WRITE. A partial clear-write operation consists of:

a. Reading a word from a specified address in memory.

b. Gating the word into the M-register.

c. Checking parity.

d. Inserting the new byte or bytes into the M-register under control of the byte presence indicators without disturbing the remaining bytes.

e. Generating new parity.

f. Writing the contents of the M-register into memory.

The mode of memory operation is determined by either the CPU or the IOP by setting the byte presence indicators, MW0, MW1, MW2, and MW3 in the memory via ports A, B, or C. The basic logic for read-restore, full write, and partial write is:

RD = NMW0 NMW1 NMW2 NMW3

WF = MW0 MW1 MW2 MW3

WP = NRD NWF

Figure 3-71 shows detailed logic for the memory mode determination.

3-43 Memory Delay Lines

Memory timing is controlled by two 600 nsec delay lines. Each delay line has taps at every 20 nsec interval. Buffer or inverter delay sensors pick off the delay line pulse at strategic intervals. The outputs of these buffers and inverters are distributed to the memory control logic to provide the basic memory timing.

One memory delay line is associated with the first half-cycle of a memory operation and is initiated by signal S/READDL. The other delay line is associated with the second half-cycle of a memory operation and is initiated by the signal S/WRITEDL. Two separate delay lines are required for the partial write mode in order to split the first and second half-cycles because of the time involved in checking parity after the read half-cycle, and regenerating a new parity before the write half-cycle is initiated. For this mode of operation the two half-cycles must be separated by more than the normal amount of time.

Figure 3-72 shows all the major input logic and output timing signals associated with these two delay lines.

Communication between the memory and units connected to memory through the ports is asynchronous. For this reason the timing of many signals is referenced to an interval time designated as t0. Time t0 corresponds to the actual start of a cycle for any given port.

The time interval between the receipt of a memory request at the port and the occurrence of t0 is called the selection interval. The time interval between t0 and the end of the memory cycle (when the memory is no longer busy) is called the active interval and is dependent upon the mode of operation. The active interval satisfies the following requirements:

Mode	Minimum Nanoseconds	Maximum Nanoseconds
Read-restore	755	830
Full clear-write	755	830
Partial clear-write	1155	1230

The active interval and the basic cycle time are not the same. The basic cycle time is the inverse of the maximum cycle rate in nanoseconds and can vary as follows:

Mode	Minimum Nanoseconds	Maximum Nanoseconds
Read-restore	770	870
Full clear-write	770	870
Partial clear-write	1170	1270

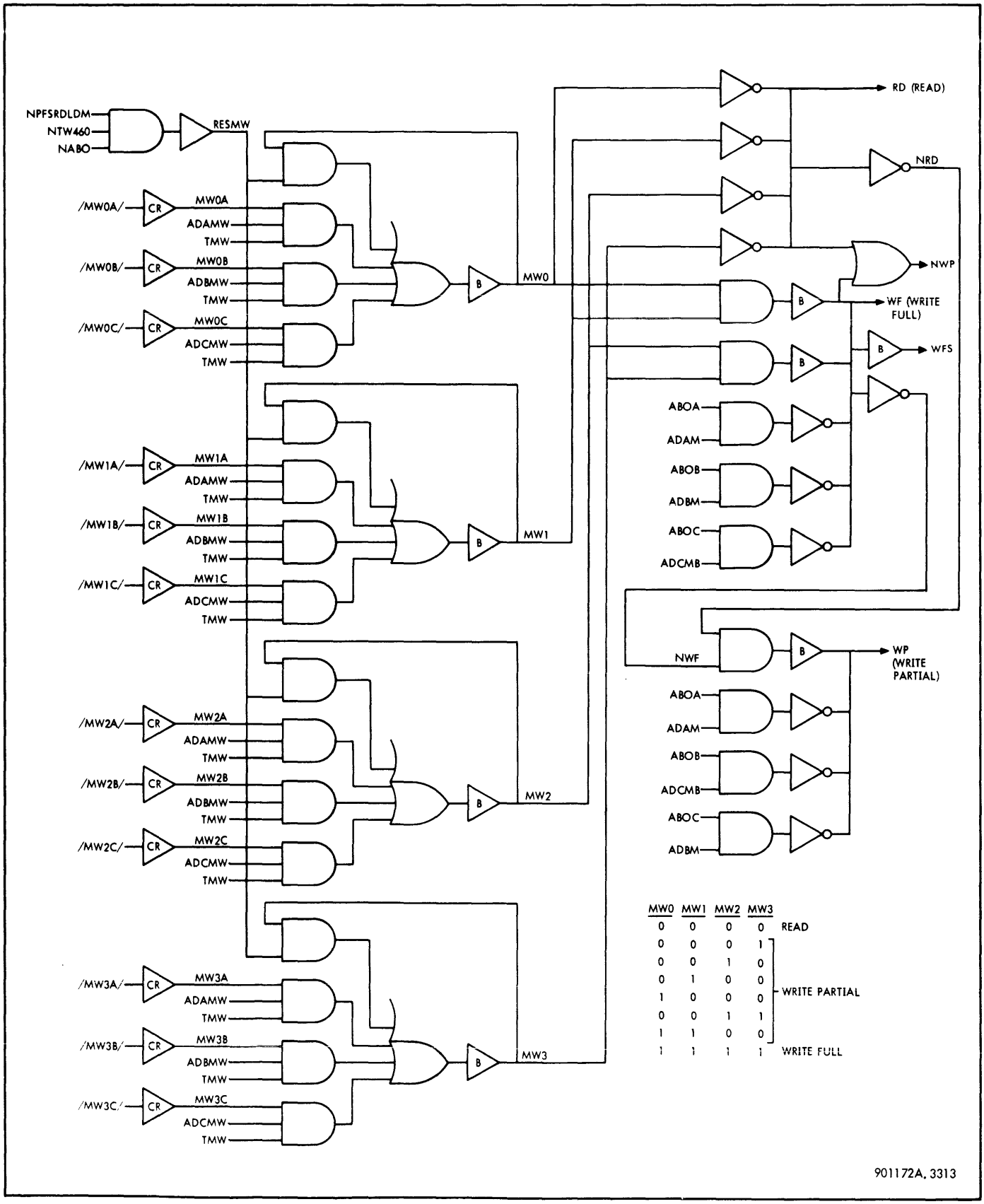


Figure 3-71. Read, Full Write, and Partial Write Logic Diagram

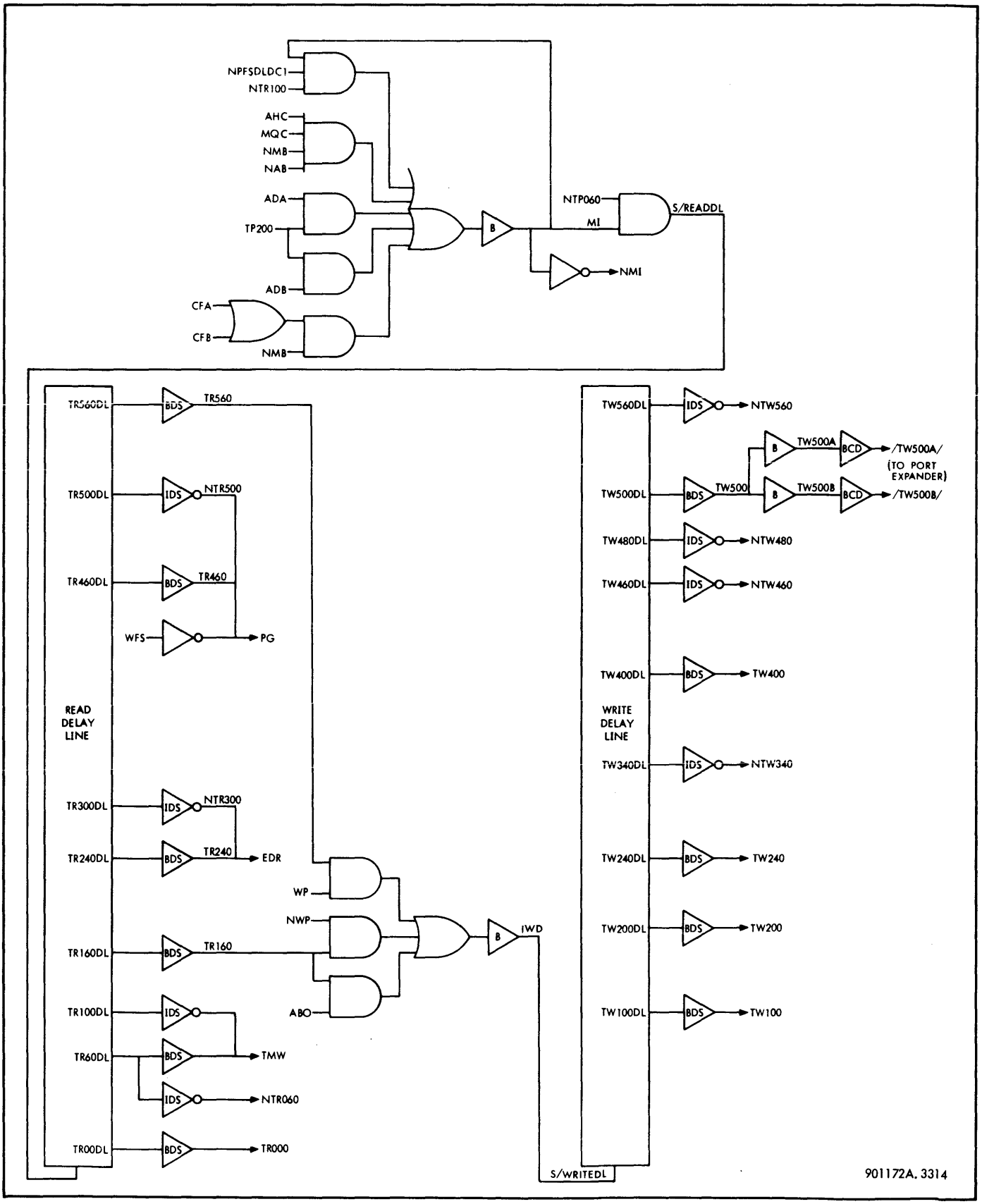


Figure 3-72. Read and Write Delay Lines

In contrast to the precision of the active interval, the selection interval is widely variable. The selection interval is dependent upon the following:

a. The port requesting service.

b. The current state of the memory.

c. The mode of operation.

d. Current and subsequent requests presented to other ports.

With respect to the start-up condition (that is, with memory not initially busy and no request on other ports), the following conditions exist:

Port	Minimum Nanoseconds	Maximum Nanoseconds
A	235	330
B	235	330
C	25	80

The start-up condition corresponds to the minimum selection interval that would be observed for a given port of a given memory. The sum of an access interval and a selection interval is not necessarily related to the cycle time of the memory. This is so because the selection interval of a request may be overlapped with the active interval of the previous request.

An example of basic memory cycle timing for both a read-restore and a full-write operation for a memory request from port C is shown in figure 3-73. An example of a memory cycle timing for a partial-write operation for a request from port C is shown in figure 3-74. In this latter case, the clear half-cycle and the write half-cycle are separated in time by 260 nsec to allow for parity checking and regeneration.

The selection interval time required when either port A or port B initiates a memory request is shown in figure 3-75.

With the memory not busy (NMB) the selection interval is provided by the port delay line. (See figure 3-76.) The port delay line is initiated only when a request is made from port A or port B and the memory is not busy.

S/PORTDL = IPD

IPD = NMB (APA + APB)

APA = MQAT AHAEXP NMB NCFA NAPA NAPB + APA NMI

APB = MQBT AHBEXP NMB NCFB NCFB NCFC NAPA + APB NMI

After 200 nanoseconds, the memory cycle is initiated by setting a timing pulse into the read delay line.

S/READL = MI

MI = TP200 (ADA + ADB)

ADA = APA TP060 NMI + ADA NTW340 + . . .

ADB = APB TP060 MI + ADB NTW340 + . . .

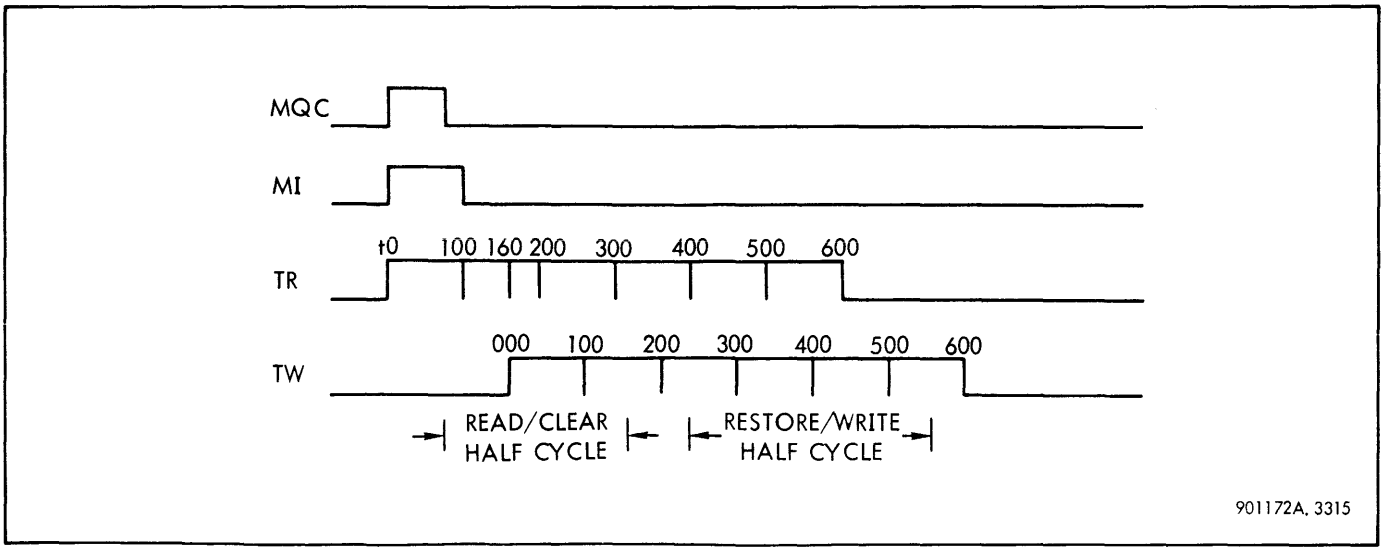


Figure 3-73. Read-Restore and Full Write Delay Line Timing for Port C

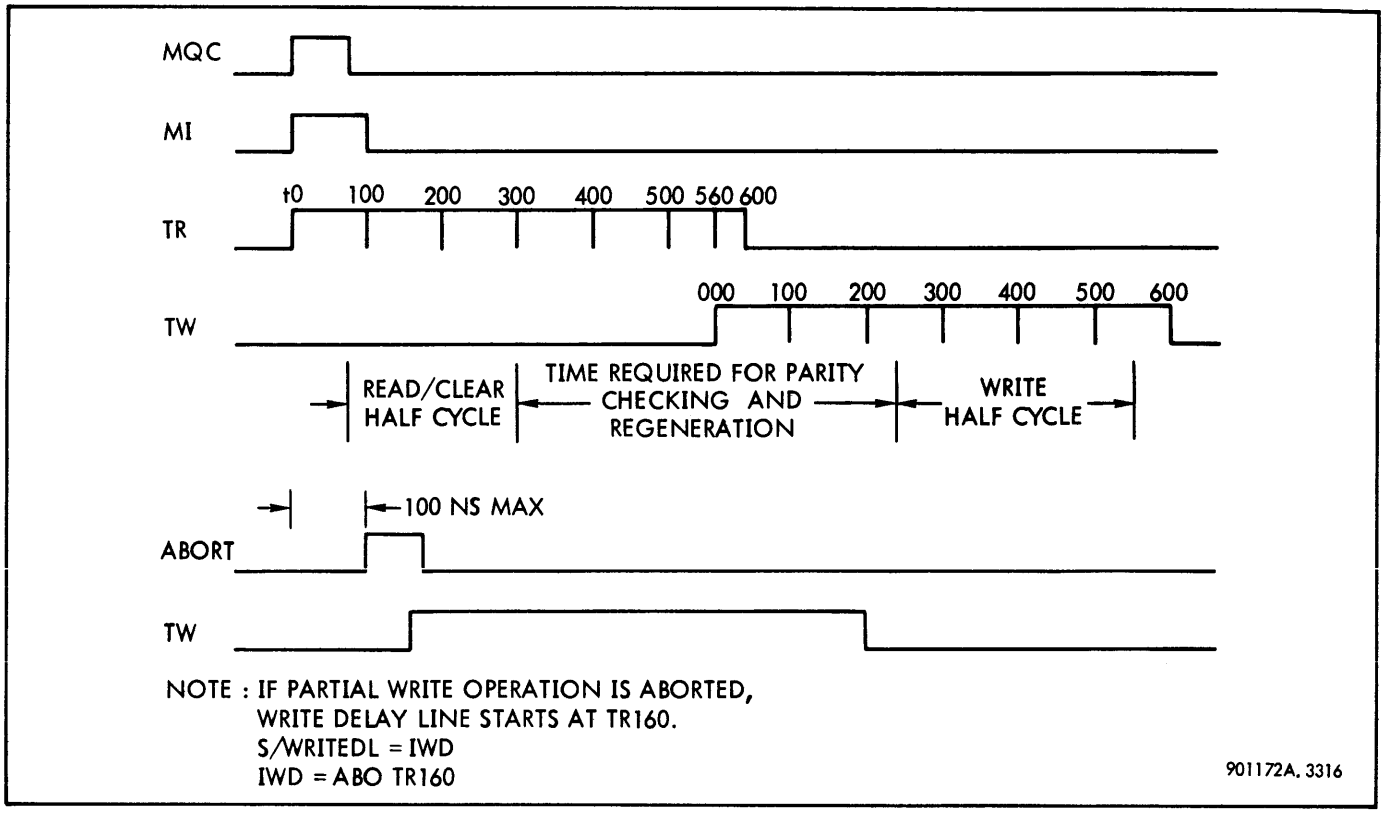


Figure 3-74. Partial Write Delay Line Timing for Port C

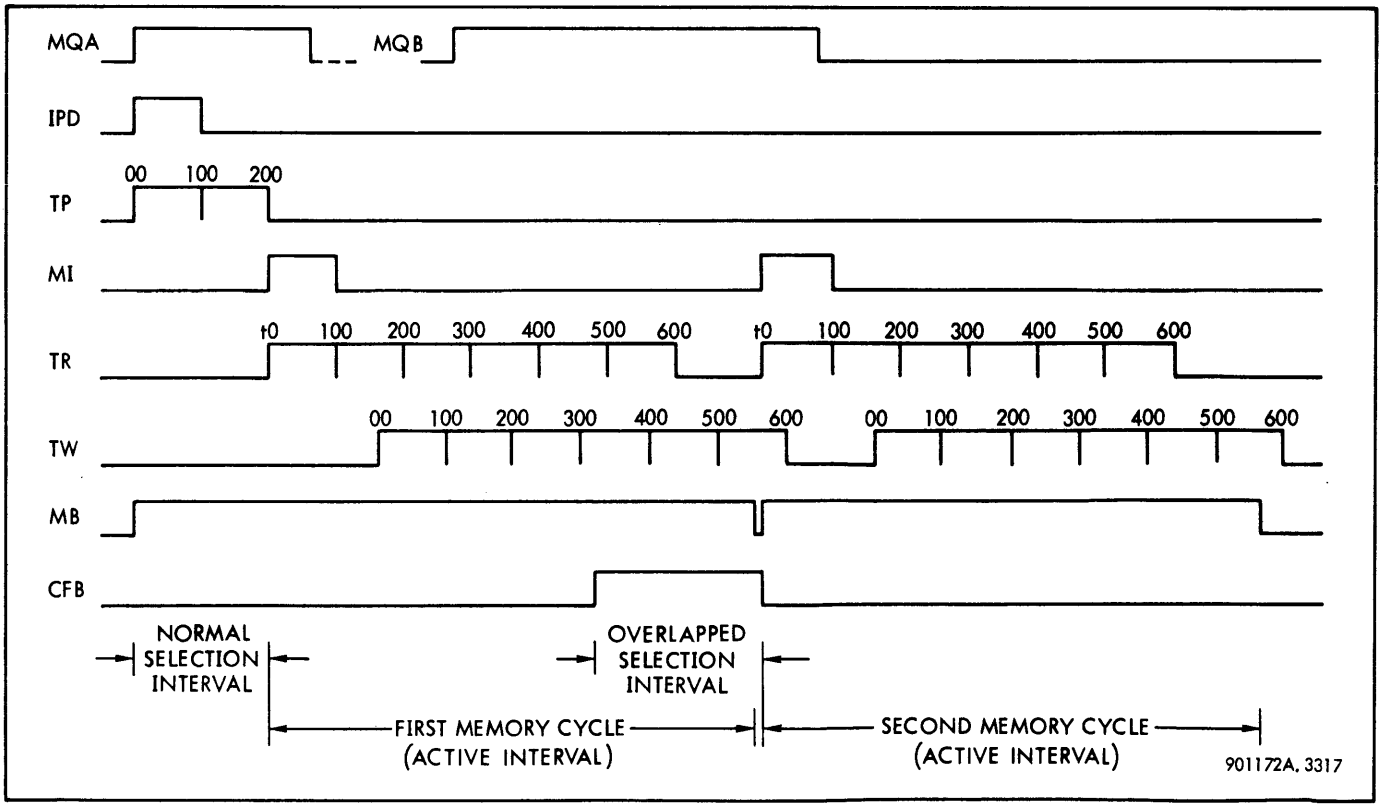


Figure 3-75. Read-Restore Delay Line Timing for Ports A or B

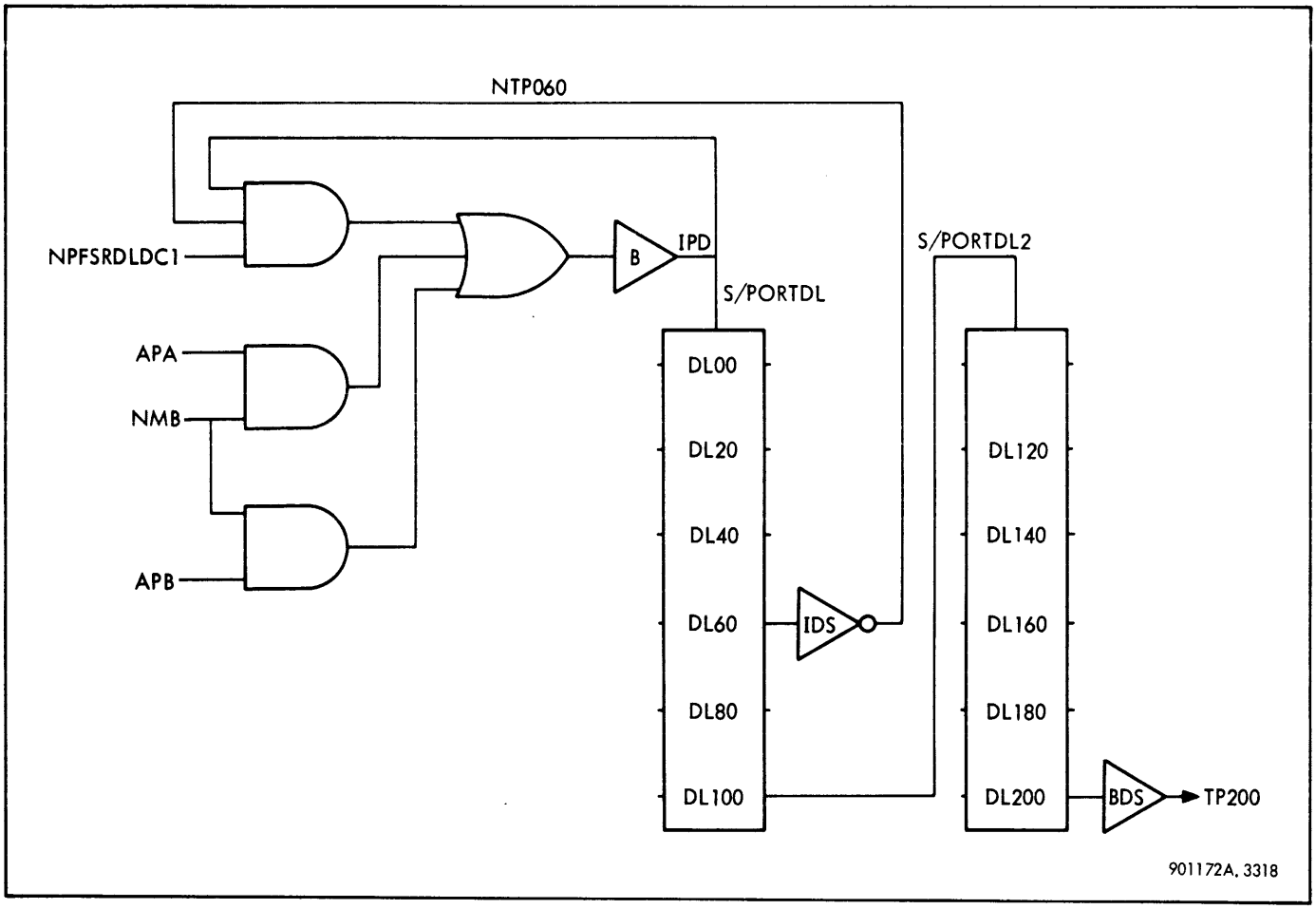


Figure 3-76. Ports A and B Delay Line

If a memory request is made from either port A or port B while the memory is busy with a previous request, but has not yet reached TW320 time of the current memory cycle, signals CFA or CFB anticipate the required selection interval delay by latching at TW320 time. Thus, at the end of the current memory cycle, the read delay line will be initiated. This condition is also shown in figure 3-75.

MI = NMB (CFA + CFB)

CFA = MQAT AHAEXP TW320 NTW360
+ CFA NMI

CFB = MQBT AHBEXP TW320 NTW360
+ CFB NMI

ADA = CFA TW400 + ADA NTW340 + ...

ADB = CFB TW400 + ADB NTW340

S/READDL = MI

The memory busy signal MB, when true, indicates that the memory is engaged in a read or write operation. This signal is also held true during a memory halt condition to prevent any new memory requests from being honored. (See logic diagram 3-77.)

MB = MI + IPD + HALT + NTW560 NMBDLD

HALT = MR Memory reset
+ HOF MF Halt on memory fault

MF = PE Parity error
+ MF NMFR

3-44 Abort

If an instruction attempts to write into a protected area of memory, the CPU will raise the abort signal ABOC. This signal must be seen in the memory within 100 nsec after the read delay line has been initiated. With ABOC true, the byte presence indicators, MW0 through MW3, will unlatch and be set to zeros.

MW0/3 = NABO NTW460 + ... Latch

With the byte presence indicators set to zeros, the memory cycle is changed from write mode to read mode, thus preserving the integrity of the protected memory location.

If a partial write operation is aborted, a special gate exists to initiate the write delay line earlier than it would have been if the instruction had not been aborted.

S/WRITEDL = IWD

IWD = ABO TR160

This timing is shown in figure 3-74.

3-45 Memory Reset

Memory is reset when power is applied to the Sigma 5 computer (ST) or when the SYSTEM RESET button on the PCP is pushed (MR). When MR goes true, signal HALT goes high and inhibits any more memory cycles from starting. After a 1 to 3 μs delay to allow the current cycle to be completed, MRD goes true. Signal PFSRDLD, which is normally false during memory operation, goes true and sets signal NTSSTB to inhibit the memory strobe. Signal PFSRDLD also causes the memory fault indicators to drop. Signals NPFSRDLDT1, NPFSDLDT2, NPFSRDLDM and NPFSRDLDC1 go false to drop other latches. In this way the system is returned to its initial state. The data latches are not reset.

3-46 Memory Fault

Each memory module contains eight memory fault gates. (See figure 3-78.) Only one gate in each module can go true, however, when a parity error occurs, depending upon the setting of the module number switches N0, N1, and N2, which is different in each module.

When a parity error occurs, the memory fault lamp associated with the memory module in which the error occurred will light. Memory fault lights will be turned off when either the I/O RESET or the SYSTEM RESET button is pressed, when power is first applied to the system, or when the proper read direct instruction is executed in the internal control mode.

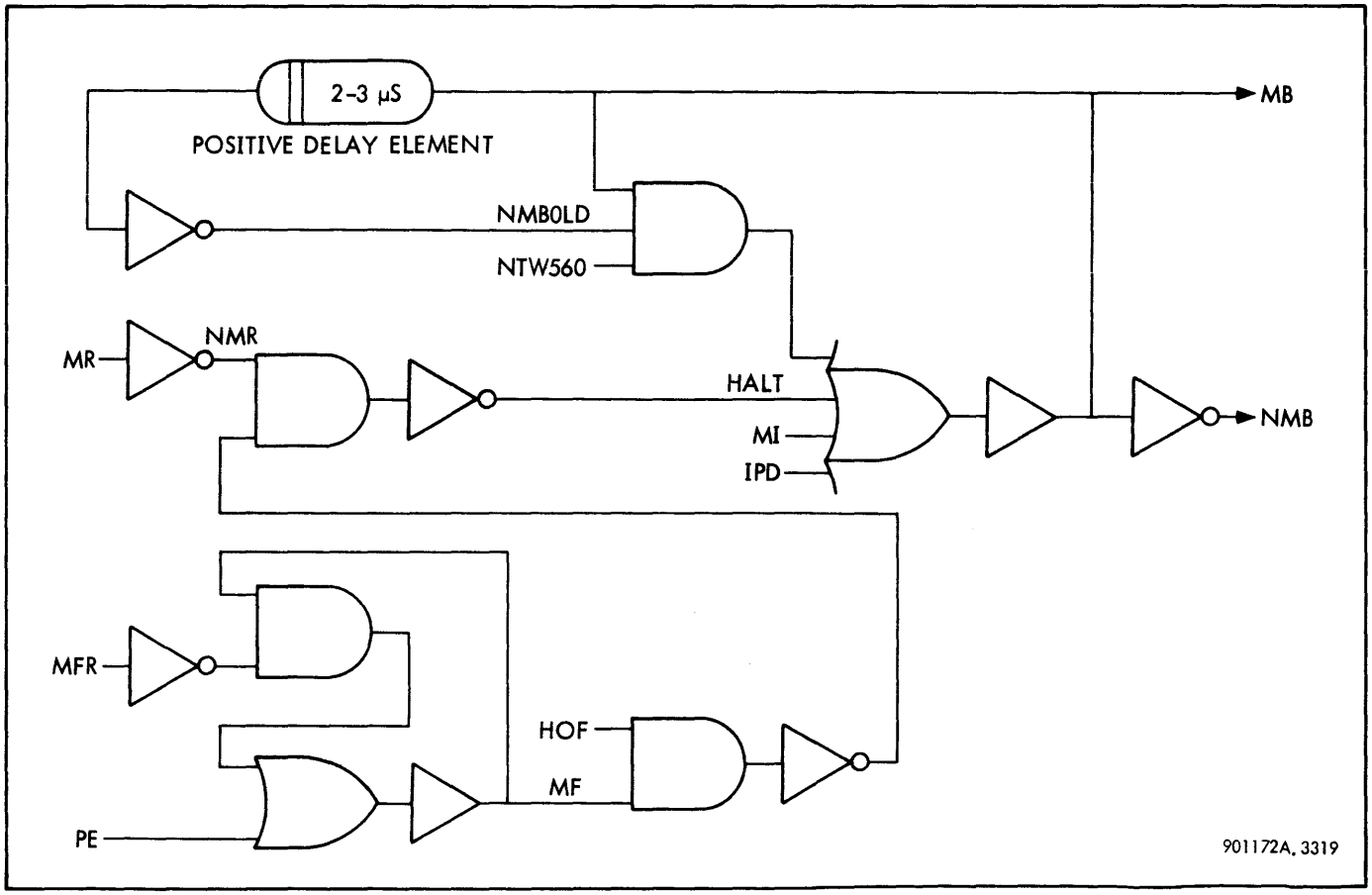


Figure 3-77. Memory Busy (MB), Logic Diagram

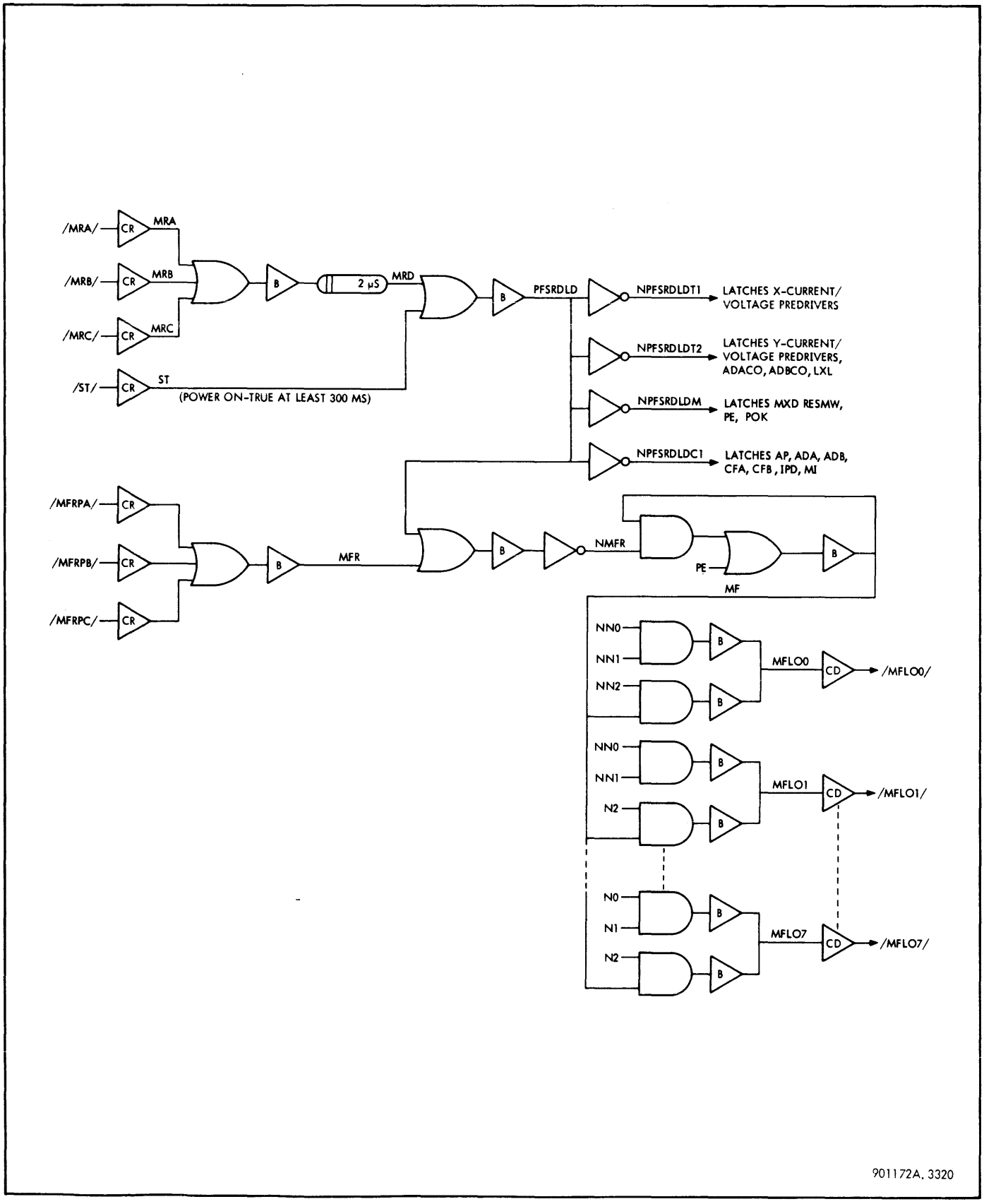


Figure 3-78. Power Fail-Safe, Reset, and Memory Fault, Logic Diagram

3-47 Data Register

The M-register, which is made up of the 32 buffer latches, M00 through M31, accepts data from the memory core sense amplifiers (MD00-MD31) during read and partial write operations. During full write and partial write operations the M-register accepts the port data and holds this data until it is written into the core memory by means of Y inhibit circuits.

Figure 3-79 shows all input and output gating for the most significant bit of the M-register, M00. Gating for M00 is typical of all M-register bits, M00 through M31. The parity bit, M32, is discussed separately in paragraph 3-51.

3-48 Read Timing and Data Flow

Figure 3-80 describes the basic timing requirements of memory read operations. During the read operation the M-register is cleared at TR020 time, and the core data is gated into the register at TR220 time, latched by signal MXM0/3.

M00 = MD00 MXD0B + M00 MXM0
⋮ ⋮
M31 = MD31 MXD3B + M31 MXM3

MXD0B/3B = MWF TR220 + MXD0B/3B NTR420

MXM0/3 = NTR020 (NMW0/3 + NTR380 + NWP)

Actually, the core data MD00-MD31 is placed onto the data bus lines, MC00-MC31, before the M-register can latch, by the following speedup gates.

MC00 = DGC0 MD00 + DGC0 M00
⋮ ⋮
MC31 = DGC3 MD31 + DGC3 M31

The core data is also gated to the inverse M-register bits, NM00-NM31, by speedup gates.

NM00 = N(MD00 MXD0I)
⋮ ⋮
NM31 = N(MD31 MXD3I

MXD0I/3I = TR220 NWF + MXD0I/3I NTR420

The data remains on the memory bus only as long as the data gate signal, DGC, is true – that is, from TR240 to TR420 time.

DGC0/3 = DG

DG = RD TR240 + DG NTR420

The data in the M-register, however, remains latched until TR020 time of the next memory operation.

3-49 Full Write Timing and Data Flow

During a full write operation, the M-register is cleared at TR020 time. (See figure 3-81.) The data is read from the addressed core location as in the read operation; however, the core data is not gated to the M-register since the transfer terms MXD0B/3B cannot come true. At TR160 time the data to be written into memory is gated into the M-register by signals MXC0B/3B.

M00 = MXC0B MC00 + M00 MXM0
⋮ ⋮
M31 = MXC3B MC31 + M31 MXM3

MXM0/3 = NTR020 (NMW0 + NTR480 + NWP)

3-50 Partial Write Timing and Data Flow

A partial write operation is distinguished by the configuration of the byte presence indicators, MW0 through MW3. (See figure 3-81.) If these indicators are neither all zeros (read) nor all ones (full write), the operation to be performed is a partial write.

RD = (NMW0 NMW1 NMW2 NMW3)

WF = (MW0 MW1 MW2 MW3)

WP = N(RD + WF)

A partial write operation reads a word from the addressed memory location, retains those bytes of the word for which the corresponding byte presence indicator is false, inserts into the word new data into those bytes for which the corresponding byte presence indicator is true, and writes the result back into the same memory location. Parity is checked on the contents of the original memory word, and new parity is generated before the modified word is written back into memory.

Timing and data flow for this operation is indicated in figure 3-82.

The M-register is cleared of its previous contents at TR020 time, and the memory word is read into the M-register at TR220, gated by the signals MXD0B-MXD3B.

MXD0B/3B = NWF TR220

The contents of the M-register are then checked for parity. At TR480 time those bytes of the original memory word now in the M-register are cleared to zeros. This is accomplished by dropping the latches of those bytes for which the corresponding byte presence indicator is true.

M00 = M00 MXM0 + . . .
⋮ ⋮
M31 = M31 MXM3 + . . .

MXM0 = N(MW0 TR480 WP) NTR020
⋮ ⋮
MXM3 = N(MW3 TR480 WP) NTR020

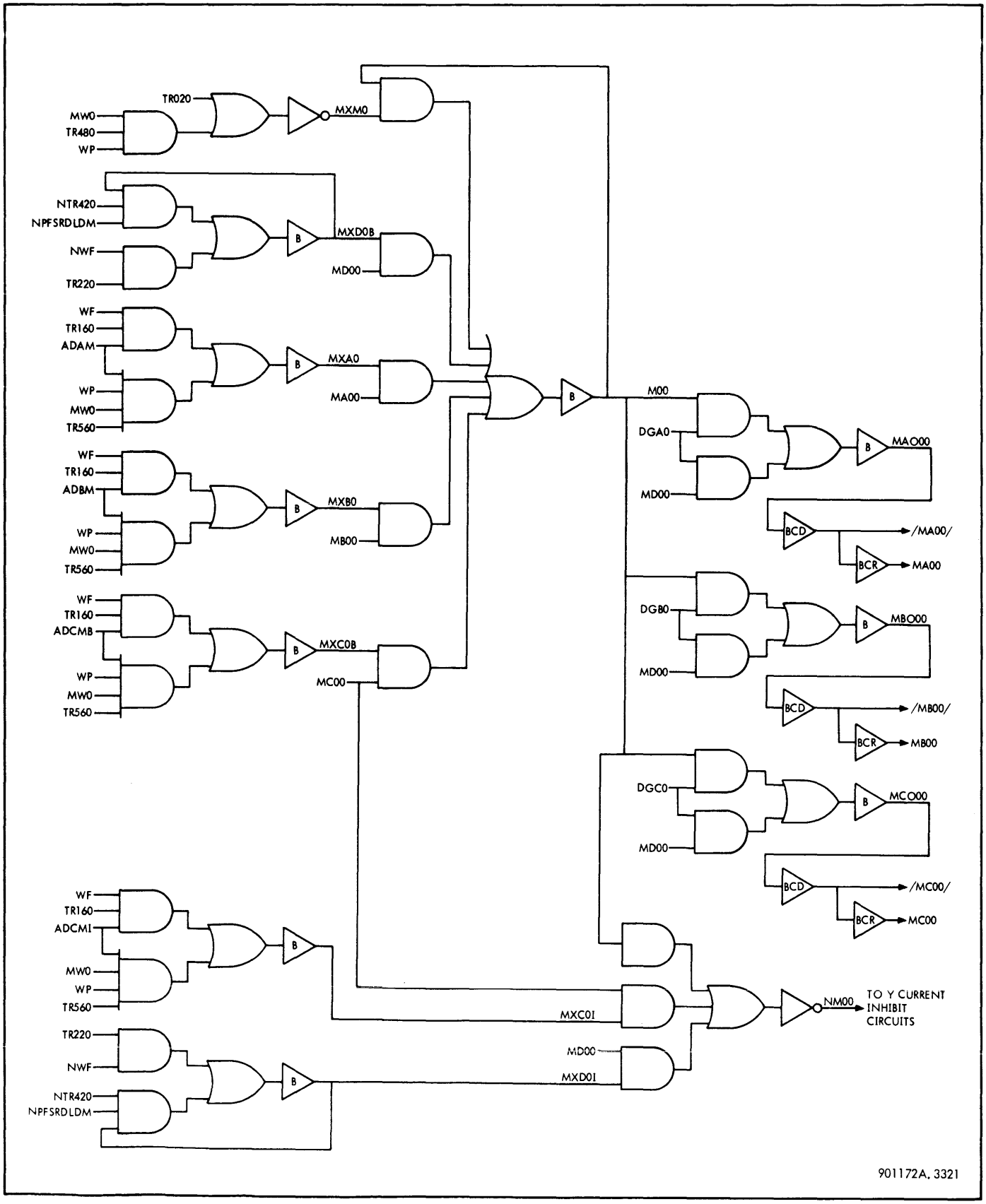


Figure 3-79. M-Register (M00, Typical of M00-M31)

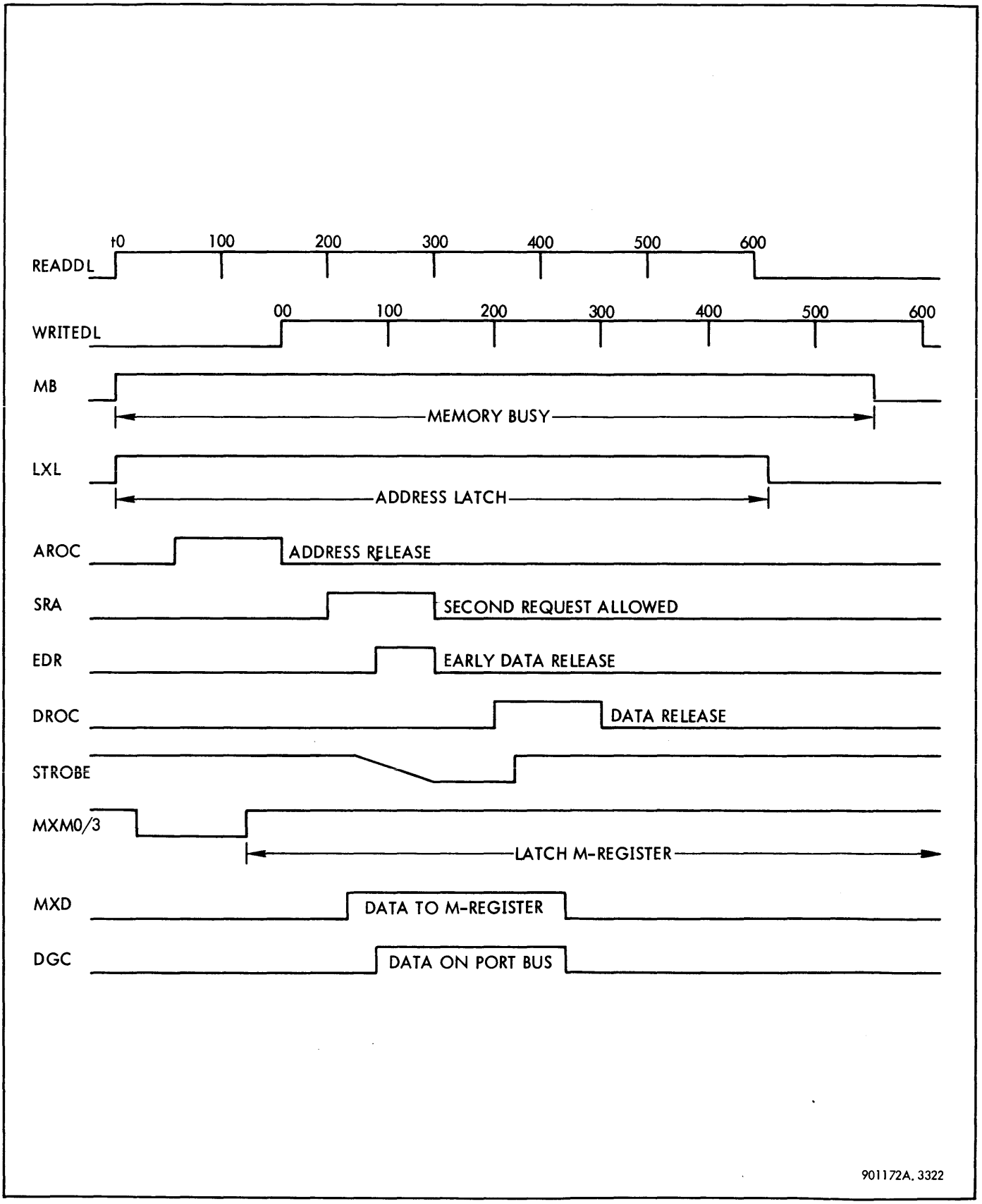


Figure 3-80. Read Timing Diagram

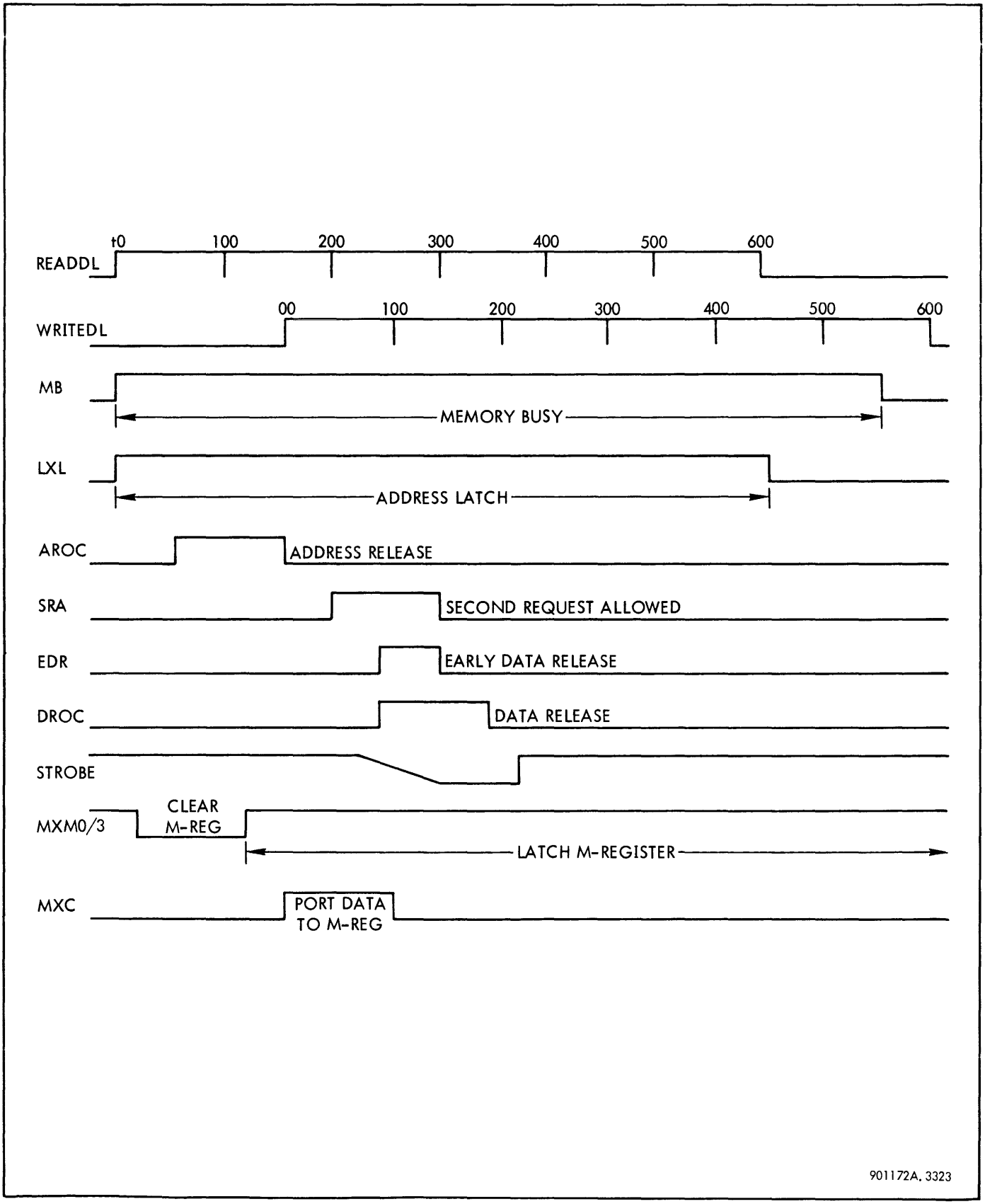


Figure 3-81. Full Write Timing Diagram

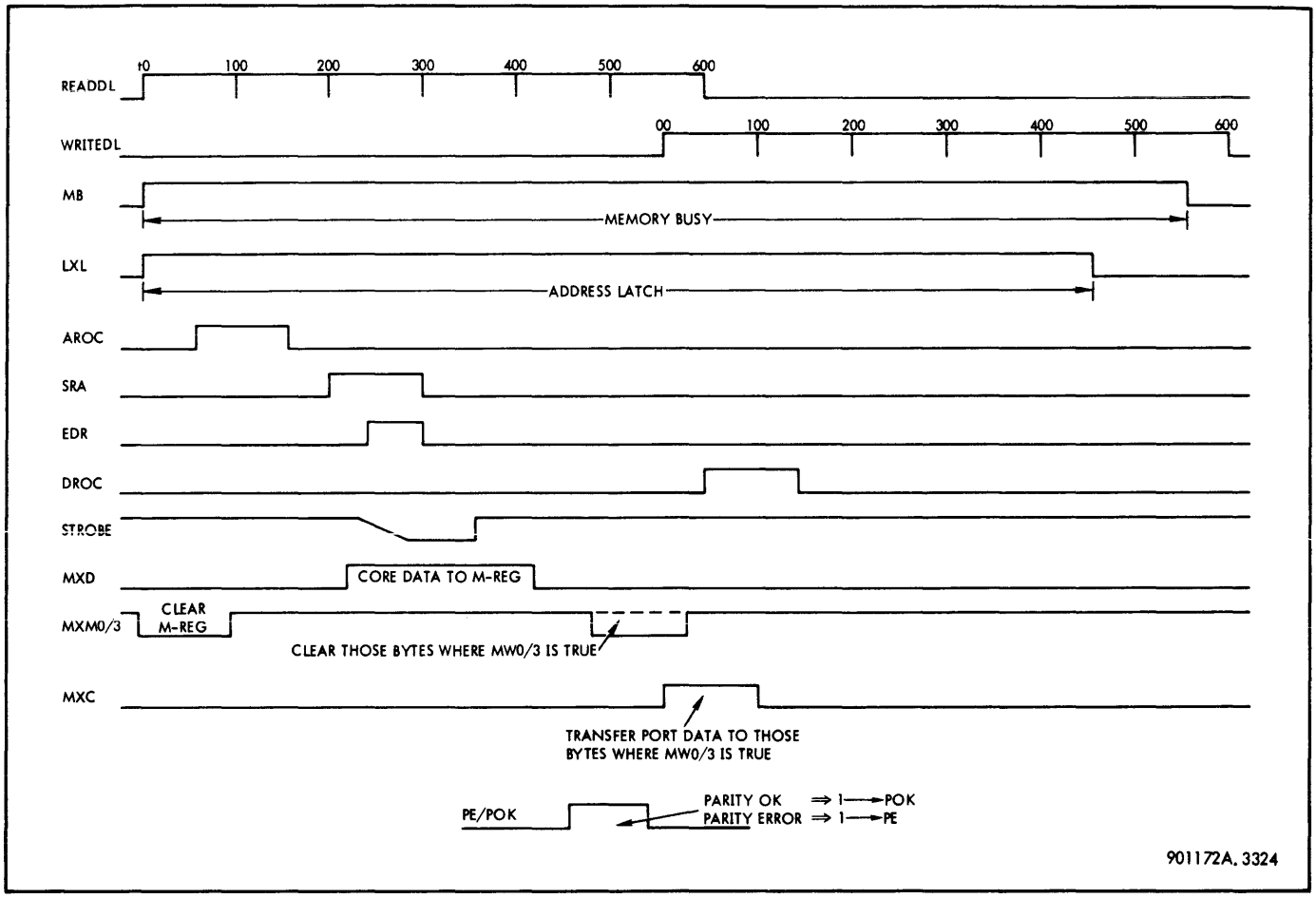


Figure 3-82. Partial Write Timing Diagram

At TR560 time the new data bytes on the port bus are inserted into the previously cleared byte positions of the M-register by signals MXC0B-MXC3B.

$$\begin{aligned} \text{MXC0B} &= \text{MW0 WP TR560} \\ &\vdots \\ \text{MXC3B} &= \text{MW3 WP TR560} \end{aligned}$$

and the write delay line is initiated.

$$\begin{aligned} \text{S/WRITEDL} &= \text{IWD} \\ \text{IWD} &= \text{WP TR560} \end{aligned}$$

3-51 Parity Checking and Parity Generation

The Sigma 5 memory employs odd parity; that is, if any memory word contains an even number of one-bits, its accompanying parity bit will contain a one, or if any memory word contains an odd number of one-bits, its accompanying parity bit will contain a zero. Thus, each word in memory is made up of 33 bits, 32 data bits plus one parity bit, and the total number of one-bits in each 33-bit word must always be an odd number.

Whenever a word is read from memory, its parity bit is also read into the buffer latch, M32.

$$\begin{aligned} \text{M32} &= \text{MD32 MXD3B} + \text{M32 MXM32} \\ \text{MXM32} &= \text{NTR020 (NWP + NTR560)} \quad \text{Latch} \end{aligned}$$

Parity determination for both reading (parity checking) and for writing (parity generation) is similar and shares much of the same logic. Parity checking consists of checking bits M00 through M32 for an odd number of one-bits. If these 33 bits contain an odd number of ones, signal POK is raised. If these 33 bits contain an even number of ones, the parity error signal, PE, is raised. Parity generation consists of checking bits M00 through M31 for an odd number of one-bits. If these 32 bits contain an odd number of ones, M32 is allowed to remain in its reset state. If these 32 bits contain an even number of ones, M32 is set to a one.

Figure 3-83 shows the scheme for determining the odd/even one-bit contents of the M-register. Four logic levels consisting of parity generator circuits are required for parity generation and parity checking. The first level

consisting of PF00 through PF27 determines odd/even configurations of the M-register in groups of three bits. PF30 performs the same function except on only the two bits, M30 and M31 and is true only if these two bits do not contain an odd number of ones. The following logic is typical of the first level.

PF00 = NM00 NM01 NM02
+ NM00 M01 NM02
+ M00 NM01 NM02
+ M00 M01 M02

Logic for PF30, however, is

PF30 = NM30 NM31
+ M30 M31

The second level parity determination gates, PS00, PS09, PS18, and PS27, use first level parity determination for their inputs. The logic for PS00 is typical of PS09 and PS18. PS27 compares first level terms PF27 and PF30 only.

PS00 = NPF00 NPF03 PF06
+ NPF00 PF03 NPF06
+ PF00 NPF03 NPF06
+ PF00 PF03 PF06

PS27 = PF27 PF30
+ NPF27 NPF30

Third level parity determination signal AP uses the second level signals PS00, PS09, and PS18 as inputs while the fourth level priority determination signal APE uses the third level term AP together with PS27 and M32 as its inputs.

AP = NPS00 NPS09 PS18
+ NPS00 PS09 NPS18
+ PS00 NPS09 NPS18
+ PS00 PS09 PS18

APE = NAP PS27 M32
+ AP NPS27 M32
+ AP PS27 NM32
+ NAP NPS27 NM32

PARITY CHECKING. Checking for parity occurs as soon as a word has been read from the memory cores and is transferred to the M-register. If all 33 bits of the word (including the parity bit) contain an odd number of one-bits, signal POK will go true at TR460 time.

POK = AP PS27 M32 PG
+ NAPE PG
+ POK TR460

PG = NWFS TR460 NTR500

If all 33 bits of the word (including the parity bit) contain an even number of ones, signal PE will go true at TR460 time.

PE = NAP NPS27 NM32 PG + APE PG
+ PE TR460

PG = NWFS TR460 NTR500

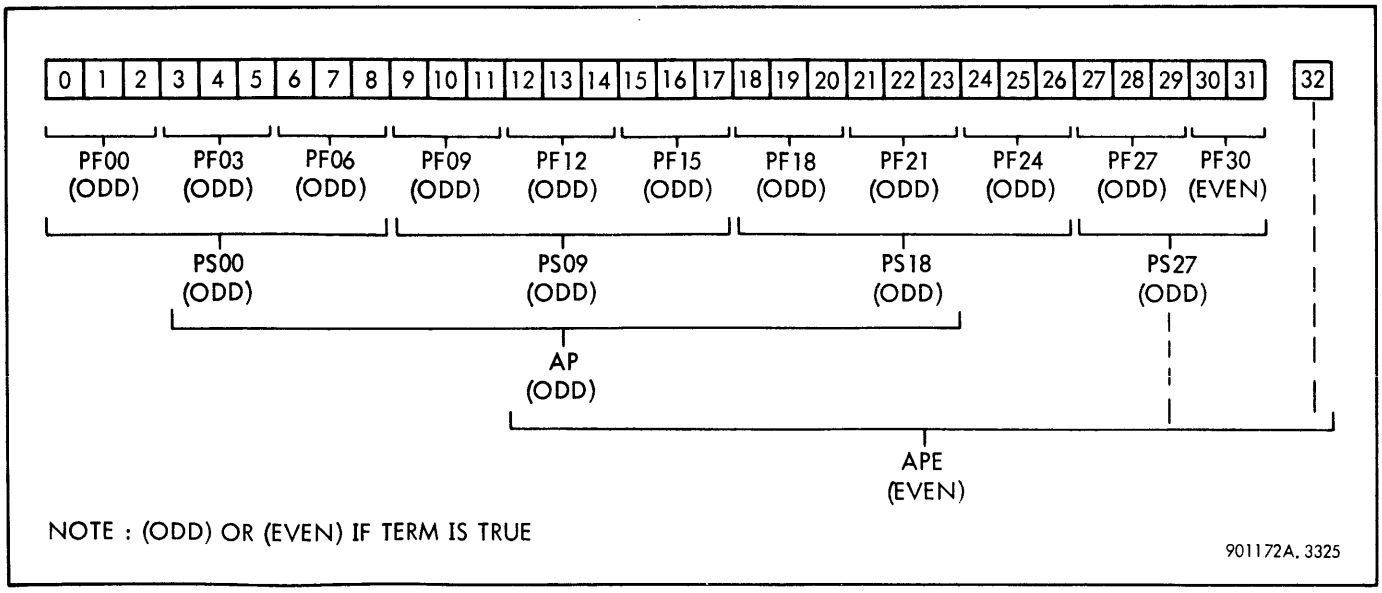


Figure 3-83. Parity Determination Logic Scheme

PARITY GENERATION. Parity generation is required whenever new data in the M-register is to be stored into the memory cores. This is accomplished by setting the parity bit of the M-register, M32, to correspond to the 32 data bits of the M-register so that all 33 bits contain an odd number of ones.

M32 = AP PS27 M32XP
+ NAP NPS27 M32XP
+ M32 MXM32

MX32P = WFS TR300 + WP TW100

3-52 Sigma 5 Core Selection

Current technical literature explaining the basics of magnetic core operation is readily available; therefore, the following discussion on the operation of Sigma 5 memory omits these fundamentals. It is assumed that the reader is familiar with such subjects pertaining to core switching as magnetomotive force, hysteresis effect, flux density, permeability, retentivity, etc. Emphasis is placed on core selection, control, and timing as they apply to the Sigma 5 memory.

3-53 Core Characteristics

Table 3-17 gives the characteristics of cores used in Sigma 5 memory.

3-54 Basic Core Switching

Sigma 5 memory employs a three-wire memory system, sometimes referred to as the common Y-digit or 2-1/2 D system in which three wires are strung through each core. These wires are:

a. X wire

b. Y wire

c. Sense wire

Table 3-17. Core Characteristics

Outer diameter of core	0.022 in.
Switching time	240 nsec approx
Nominal full drive current at 25°C	700 ma
Current compensation for temperature	4 ma/°C
Core output	-25 mv approx

This system does not use an inhibit winding. The X and Y wires that are activated to address a specific memory core are selected by an X-Y matrix composed of positive and negative X current and voltage switches in one direction, and positive and negative Y current and voltage switches in the other direction. The current through both the X and Y windings is approximately 350 ma, or half the total current required to switch the core from one state to the other. When the two half-currents through the X and Y windings at the junction of any core are in such a direction as to be additive, the core senses sufficient current to cause it to switch its state. When the two half-currents through the X and Y windings at the junction of any core are in such a direction as to be subtractive, the two currents cancel, and the core senses no switching current. In this case, the state of the core is not affected.

Each core senses one of four different current conditions. These conditions are:

a. X and Y half-currents are additive in a direction to cause the core to switch from a one to a zero.

b. X and Y half-currents are additive in a direction to cause the core to switch from a zero to a one.

c. X and Y half-currents flow in a direction so as to be subtractive – each half-current canceling the effects of the other. In this case the core is not affected.

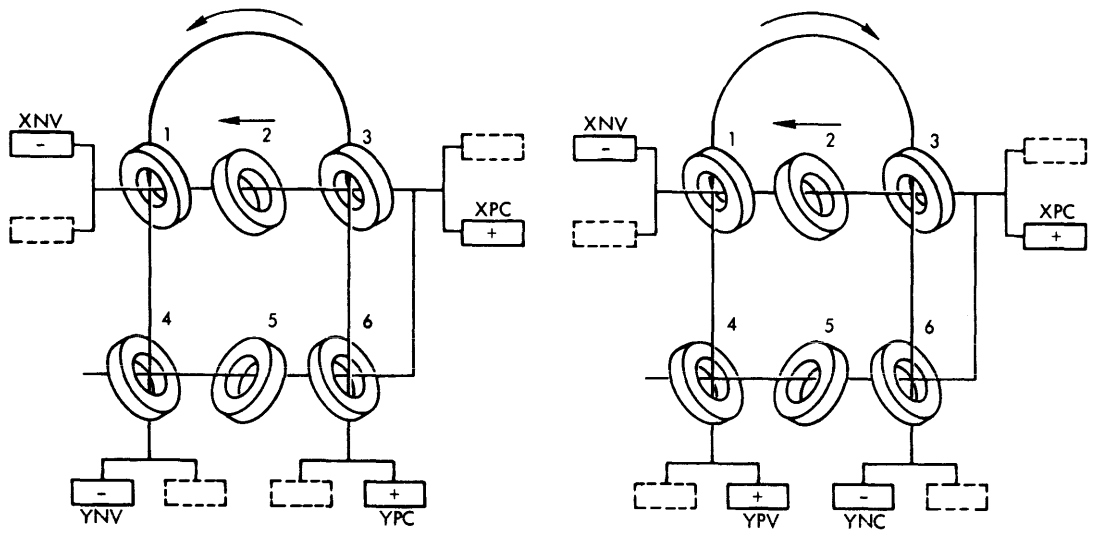
d. Current does not flow in one or the other, or neither, of the X and Y windings. In this case the core is not affected.

Figure 3-84 is a sequence of drawings that show the principles of core switching in Sigma 5 memory. Note that the Y winding is folded back in such a manner as to form a junction with the X winding at two cores – core 1 and core 3. The X and Y half-currents flow either from the positive current switch to the negative voltage switch or from the positive voltage switch to the negative current switch. The direction of the X and Y current flow depends ultimately on the anticoincident bits of the core address in the L-register – bits L22, L23, and L25.

3-55 Reading From Memory

Figure 3-85 shows a memory of 16 three-bit words, including an address register (L-register), a data register (M-register), X and Y address selection circuits, and bit plane sense amplifiers.

Assume that the data word stored in location X'7' is equal to 101_2. Note that each bit plane has one sense wire. The sense wire in each bit plane is strung through each core in its plane and is returned to a sense amplifier. To read the data word from memory, the X address selection circuits feed a positive half-current on line X3. The Y selection circuits feed a positive half-current on line Y1.

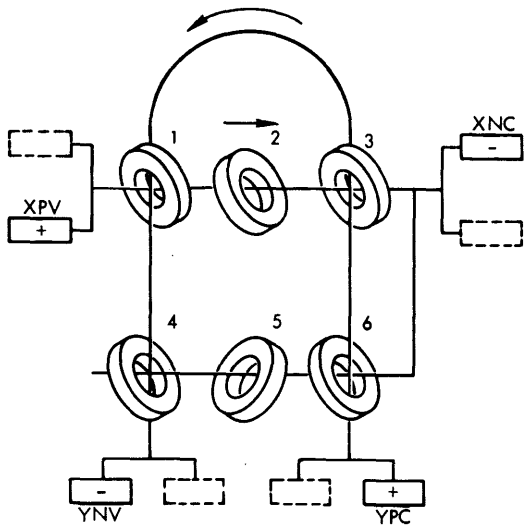


(A)

X AND Y CURRENTS ARE ADDITIVE IN CORE 1 - CANCEL IN CORE 3. CORE 1 SWITCHES FROM ONE TO ZERO

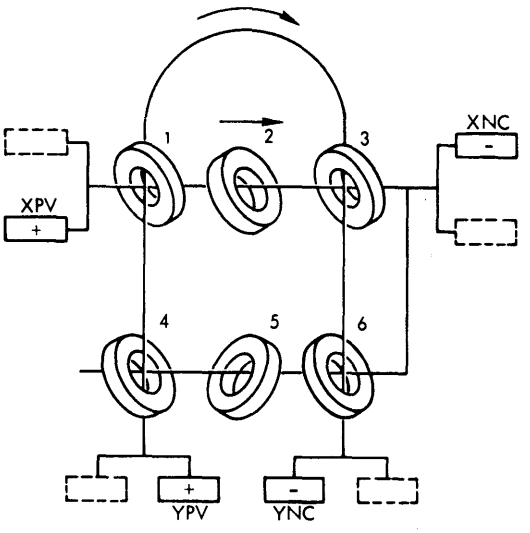
(B)

X AND Y CURRENTS ARE ADDITIVE IN CORE 3 - CANCEL IN CORE 1. CORE 3 SWITCHES FROM ONE TO ZERO



(C)

X AND Y CURRENTS ARE ADDITIVE IN CORE 3 - CANCEL IN CORE 1. CORE 3 SWITCHES FROM ZERO TO ONE



(D)

X AND Y CURRENTS ARE ADDITIVE IN CORE 1 - CANCEL IN CORE 3. CORE 1 SWITCHES FROM ZERO TO ONE

NOTE : IN ALL EXAMPLES HALF CURRENTS IN CORES 2, 4, AND 6, AND NO CURRENT IN CORE 5 HAVE NO SWITCHING EFFECT

901172A. 3326

Figure 3-84. Basic Core Switching

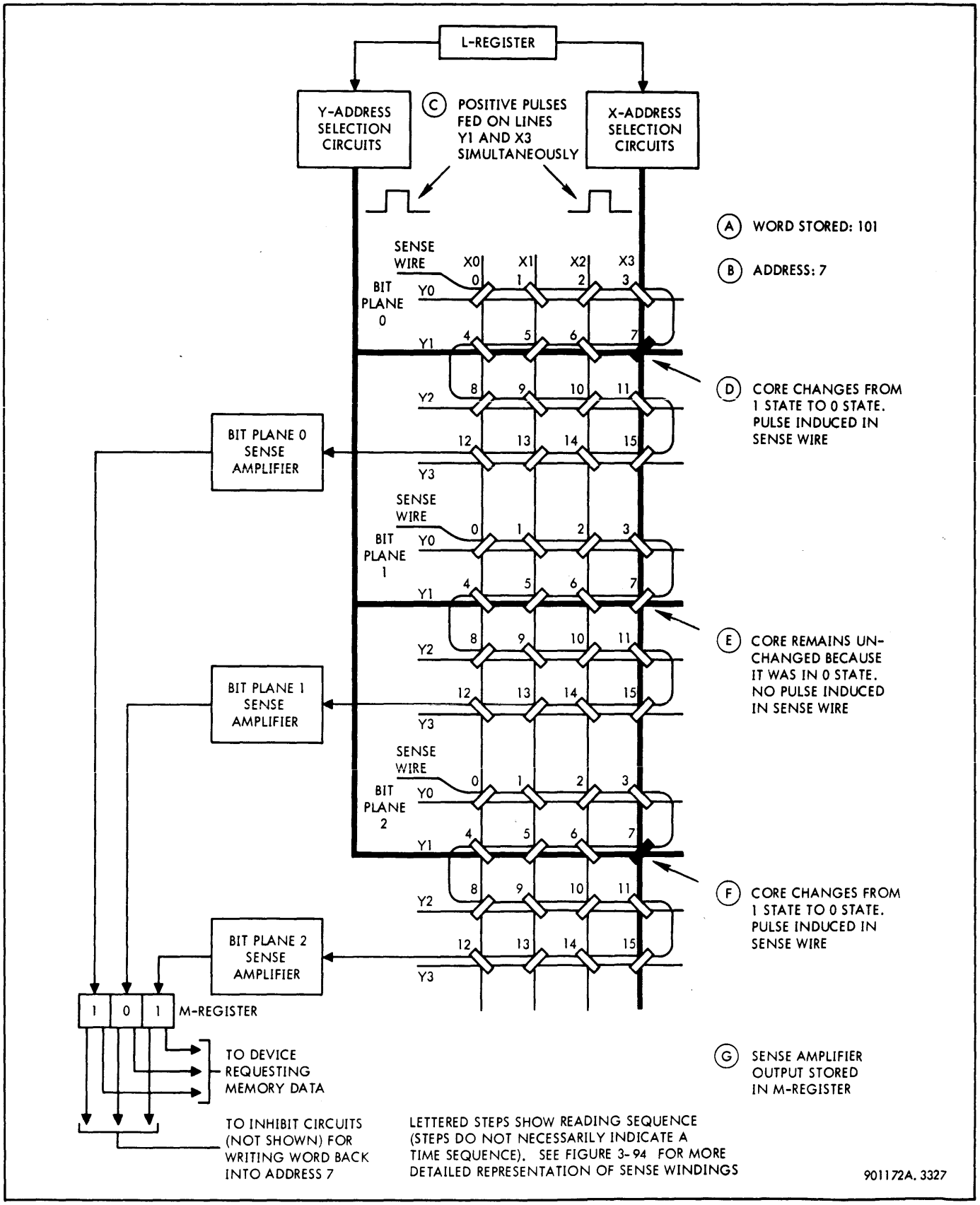


Figure 3-85. Simplified Memory, Read-Restore Operation

When additive coincident half-currents are simultaneously fed through a core, a core in the one state will be switched to the zero state. This causes a pulse to be induced in the sense wire. At the output of the sense amplifier this pulse is interpreted as a one. If the addressed core is already in the zero state, it remains a zero, and no pulse is induced in the sense wire. The absence of a pulse is interpreted as a zero at the output of the sense amplifier. Thus, with respect to a word stored in a particular memory location, either ones or zeros are read out of each bit plane. The outputs of the sense amplifiers are gated into the M-register.

A memory read operation consists of two half-cycles – a read half-cycle and a write half-cycle. Because the cores that stored ones are switched to zeros during the read half-cycle, the readout is called destructive. For this reason, the information that was read out must be restored by writing the data word back into memory again in the second half-cycle. Therefore, during the restore phase of the read-restore cycle, the word that was read out is taken from the M-register and written back into memory.

3-56 Writing Into Memory

Assume that a data word containing the number 101_2 is to be written into memory location (address) X'7'. (See figure 3-86.) First, the memory location is cleared by reading out the data in location X'7'. The data, however, is not gated to the M-register as it is during read operations. Because the cores must be cleared to zeros before a new word can be written into the cores, the write operation consists of a clear half-cycle followed by a write half-cycle. During the write half-cycle the new data to be written is in the M-register.

One of four X lines is selected by the X address selection circuits. In the example shown in figure 3-86, a negative pulse is fed through the X3 wire causing that half-current to travel through cores 3, 7, 11, and 15 in each bit plane.

During write half-cycles, an inhibit circuit controls each Y current. An inhibit circuit will either allow current to pass through or will block (inhibit) it. If the inhibit circuit receives a one from the M-register, the inhibit circuit allows current to pass through. If the inhibit circuit receives a zero from the M-register, the inhibit circuit blocks the flow of current through the Y wire.

In the example shown in figure 3-86, only the Y1 line of bit plane 1 is inhibited. Current does not flow through Y1 of bit plane 1. A negative half-current is fed through the noninhibited Y wires on which cores 4, 5, 6, and 7 are strung. This occurs in bit planes 0 and 2. The cores that receive coincident additive current switch to the one state. This occurs in bit planes 0 and 2. Therefore, the following occurs:

a. A one is written into bit plane 0.

b. A zero remains in bit plane 1.

c. A one is written into bit plane 2.

3-57 Core Diode Module

The core diode module contains 4096 bytes of either eight or nine bits. Figure 3-87 shows a nine-bit module with each bit place labeled. The ninth bit (used for parity) is designated bit 8A, and is shared by both halves of the core diode module. Figure 3-88 shows a photograph of a core diode module lying open to expose the bit planes. Also shown in the open view is the printed circuit wiring for the diodes. The diodes are mounted on the reverse side of the board shown in the photographs. The individual cores, which form the bit planes, are too small to be seen in the photograph.

The core diode module consists of two halves that are hinged together. In the photograph, figure 3-89, the X wires can be seen jumpered across the hinge. When the core diode module is put into use by being inserted into its socket, both halves are folded together and look like the one shown in figure 3-89.

The core diode module is completely symmetrical, both physically and electrically. This means that the module will operate whichever way it is inserted into the chassis.

In addition to the diodes required in the decode system, two extra diodes used in a temperature sensing network are also included. This network controls the output of the memory power supply for drive current compensation to automatically raise or lower drive current to the core diode modules inversely as the temperature varies. Because less current is required to switch a core at higher temperatures, it is necessary that the drive current tracks inversely with temperature. The temperature compensation network reduces core current at higher temperatures by lowering the supply voltage. The reverse occurs if core diode module temperature is reduced.

Figures 3-90 through 3-94 are relatively detailed core diode module drawings. Certain symbols used in the drawings are defined as follows:

a. Symbol 8YC3- means bit 8, Y current bus number 3, negative.

b. Symbol XV15 means X voltage bus number 15.

c. 5S0+ means bit 5, sense wire 0, positive side.

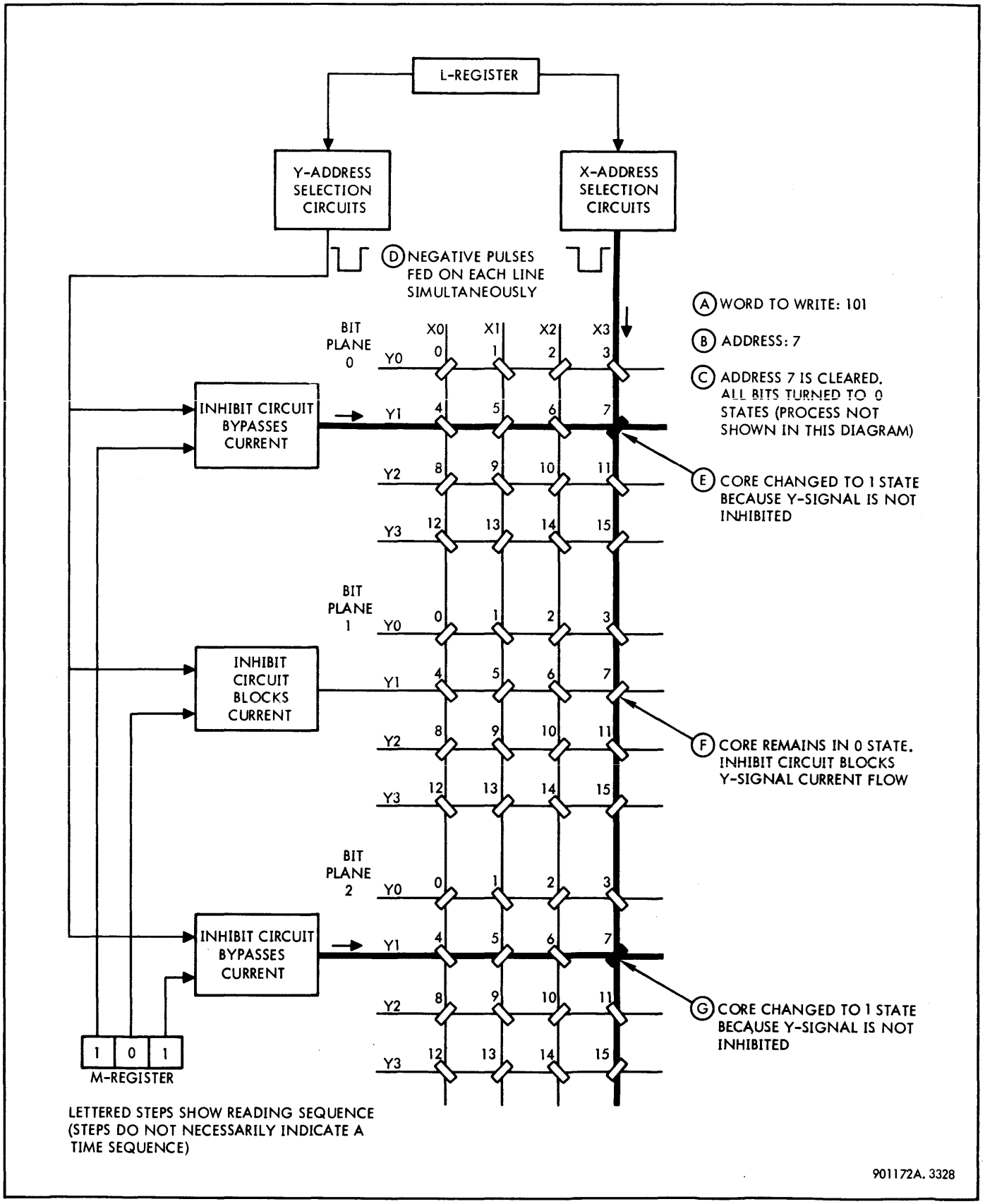


Figure 3-86. Simplified Memory, Clear Write Operation

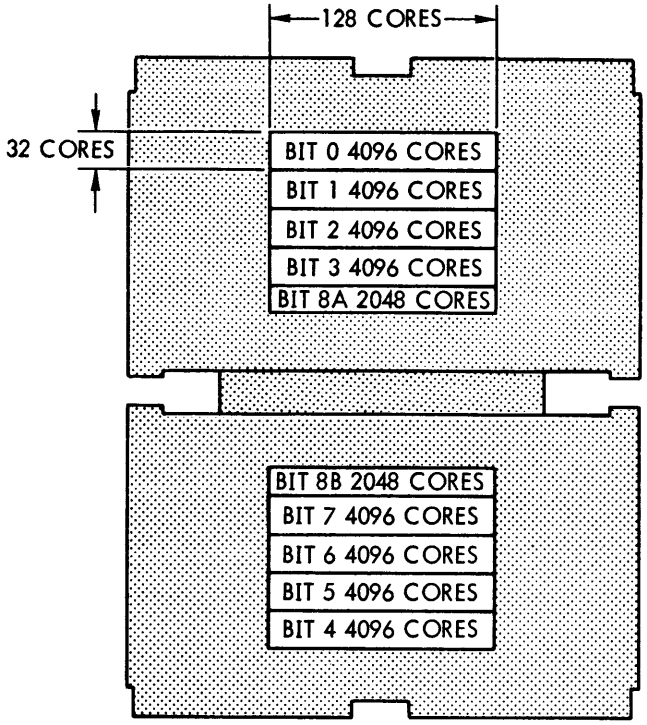


Figure 3-87. Bit Plane Layout in a Core Diode Module

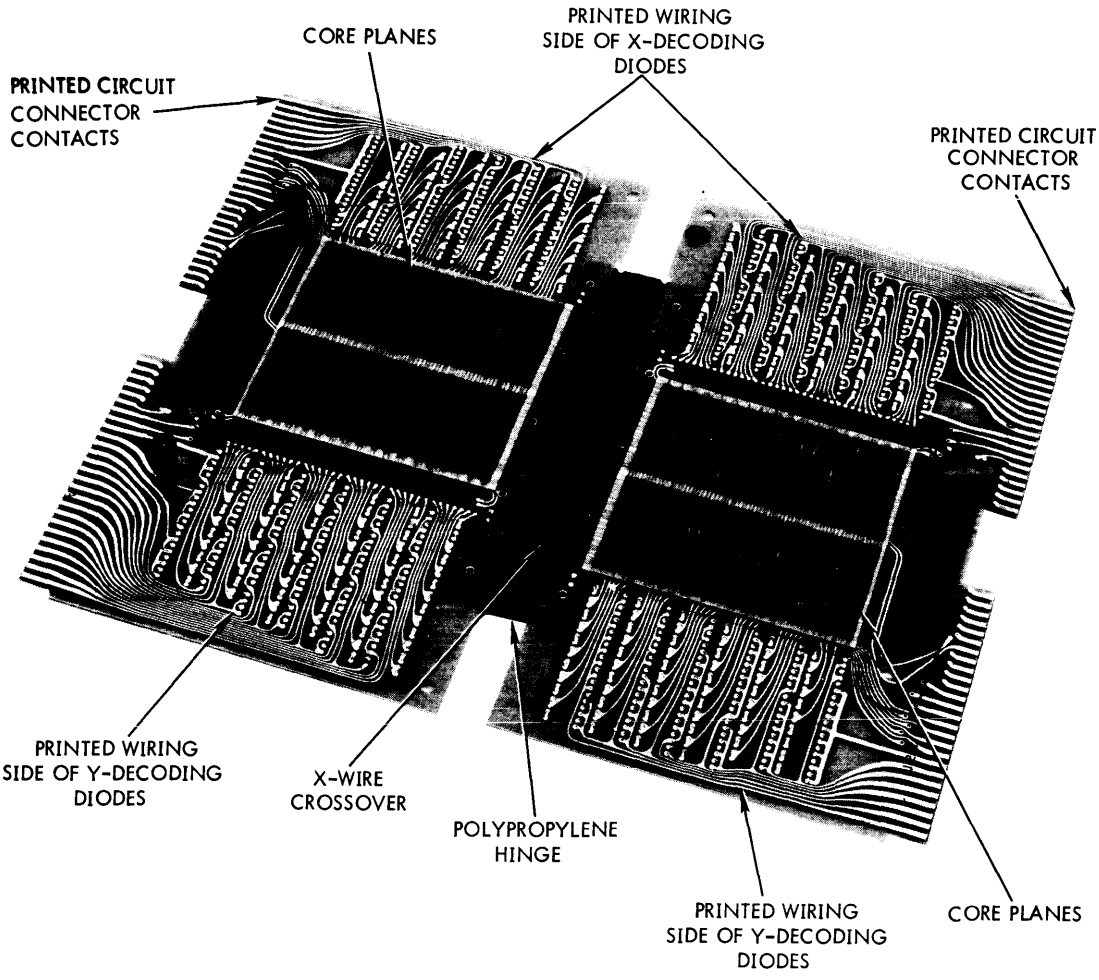


Figure 3-88. Core Diode Module, Open to Expose Bit Planes

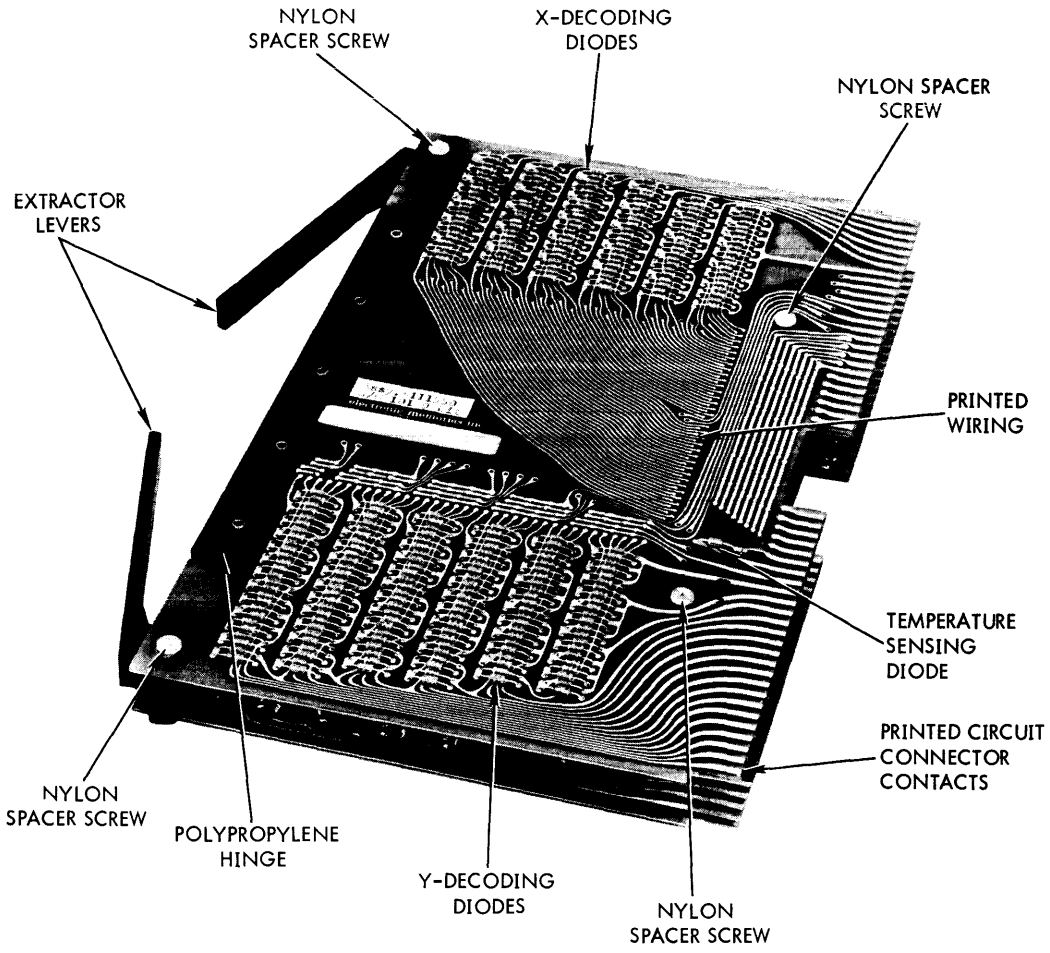
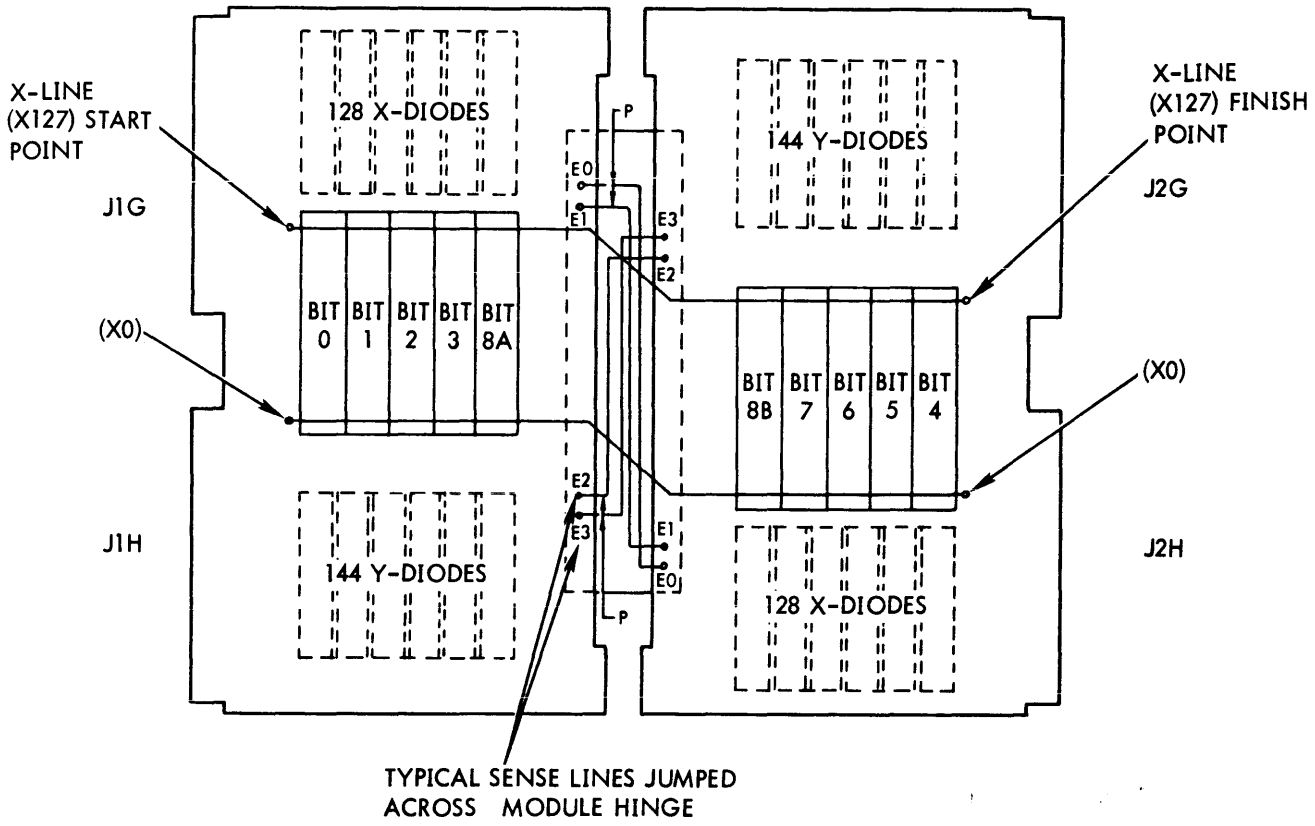


Figure 3-89. Core Diode Module, Closed, as Inserted



NOTE: THIS DRAWING WAS PREPARED FROM THE FOLLOWING ENGINEERING DRAWING: 111526-1C

901172A.3332

Figure 3-90. Core Diode Module, Bit Planes, X Wire Crossover

J1G

PIN NO.	SIGNAL	PIN NO.	SIGNAL
0	XC0-	1	XC0+
2	XC1-	3	XC1+
4	XC2-	5	XC2+
6	XC3-	7	XC3+
8	XC4-	9	XC4+
10	XC5-	11	XC5+
12	XC6-	13	XC6+
14	XC7-	15	XC7+
16	8YV1	17	8YV0
18	3YV1	19	3YV0
20	3YV3	21	3YV2
22	2YV1	23	2YV0
24	2YV3	25	2YV2
26	1YV1	27	1YV0
28	1YV3	29	1YV2
30	0YV1	31	XV15
32	XV13	33	XV11
34	XV9	35	XV7
36	XV5	37	XV3
38	XV1	39	8YV2
40	8YV3	41	SPARE
42	8AS1-	43	8AS1+
44	3S1-	45	3S1+
46	2S1-	47	2S1+
48	1S1-	49	1S1+
50	0S1-	51	0S1+

J2G

PIN NO.	SIGNAL	PIN NO.	SIGNAL
0	8YC3-	1	8YC3+
2	8YC2-	3	8YC2+
4	7YC3-	5	7YC3+
6	7YC2-	7	7YC2+
8	7YC1-	9	7YC1+
10	7YC0-	11	7YC0+
12	6YC3-	13	6YC3+
14	6YC2-	15	6YC2+
16	6YC1-	17	6YC1+
18	6YC0-	19	6YC0+
20	5YC3-	21	5YC3+
22	5YC2-	23	5YC2+
24	5YC1-	25	5YC1+
26	5YC0-	27	5YC0+
29	4YC3-	29	4YC3+
30	4YC0-	31	4YC0+
32	4YC1-	33	4YC1+
34	4YC2-	35	4YC2+
36	4YV2	37	4YV3
39	4YV0	39	TEMP SENSE DIODE -
40	TEMP SENSE DIODE +	41	TEMP SENSE DIODE +
42	8BS0-	43	8BS0+
44	7S0-	45	7S0+
46	6S0-	47	6S0+
49	5S0-	49	5S0+
50	4S0-	51	4S0+

J1H

PIN NO.	SIGNAL	PIN NO.	SIGNAL
0	0S0-	1	0S0+
2	1S0-	3	1S0+
4	2S0-	5	2S0+
6	3S0-	7	3S0+
8	8AS0-	9	8AS0+
10	TEMP SENSE DIODE +	11	TEMP SENSE DIODE +
12	TEMP SENSE DIODE -	13	0YV0
14	0YV3	15	0YV2
16	0YC2+	17	0YC2-
18	0YC1+	19	0YC1-
20	0YC0+	21	0YC0-
22	0YC3+	23	0YC3-
24	1YC0+	25	1YC0-
26	1YC1+	27	1YC1-
28	1YC2+	29	1YC2
30	1YC3+	31	1YC3-
32	2YC0+	33	2YC0-
34	2YC1+	35	2YC1-
36	2YC2+	37	2YC2-
38	2YC3+	39	2YC3-
40	3YC0+	41	3YC0-
42	3YC1+	43	3YC1-
44	3YC2+	45	3YC2-
46	3YC3+	47	3YC3-
48	8YC0+	49	8YC0-
50	8YC1+	51	8YC1-

J2H

PIN NO.	SIGNAL	PIN NO.	SIGNAL
0	4S1-	1	4S1+
2	5S1-	3	5S1+
4	6S1-	5	6S1+
6	7S1-	7	7S1+
8	8BS1-	9	8BS1+
10	SPARE	11	8YX3
12	8YV2	13	XV14
14	XV12	15	XV10
16	XV8	17	XV6
18	XV4	19	XV2
20	XV0	21	4YV1
22	5YV2	23	5YV3
24	5YV0	25	5YV1
26	6YV2	27	6YV3
28	6YV0	29	6YV1
30	7YV2	31	7YV3
32	7YV0	33	7YV1
34	8YV0	35	8YV1
36	XC0+	37	XC0-
38	XC1+	39	XC1-
40	XC2+	41	XC2-
42	XC3+	43	XC3-
44	XC4+	45	XC4-
46	XC5+	47	XC5-
48	XC6+	49	XC6-
50	XC7+	51	XC7-

901060A. 3151

Figure 3-91. Core Diode Module, Jack Pins and Signals

CHART 1 (BIT 1)

SIGNAL	PIN	CONNECTS TO Y-LINE
1YC0+	J1H-24	0, 3, 4, 7
1YC0-	J1H-25	
1YC1+	J1H-26	8, 11, 12, 15
1YC1-	J1H-27	
1YC2+	J1H-28	16, 19, 20, 23
1YC2-	J1H-29	
1YC3+	J1H-30	24, 27, 28, 31
1YC3-	J1H-31	
1YV0	J1G-27	2, 10, 18, 26
1YV1	J1G-26	1, 9, 17, 25
1YV2	J1G-29	6, 14, 22, 30
1YV3	J1G-28	5, 13, 21, 29

CHART 2 (BIT 2)

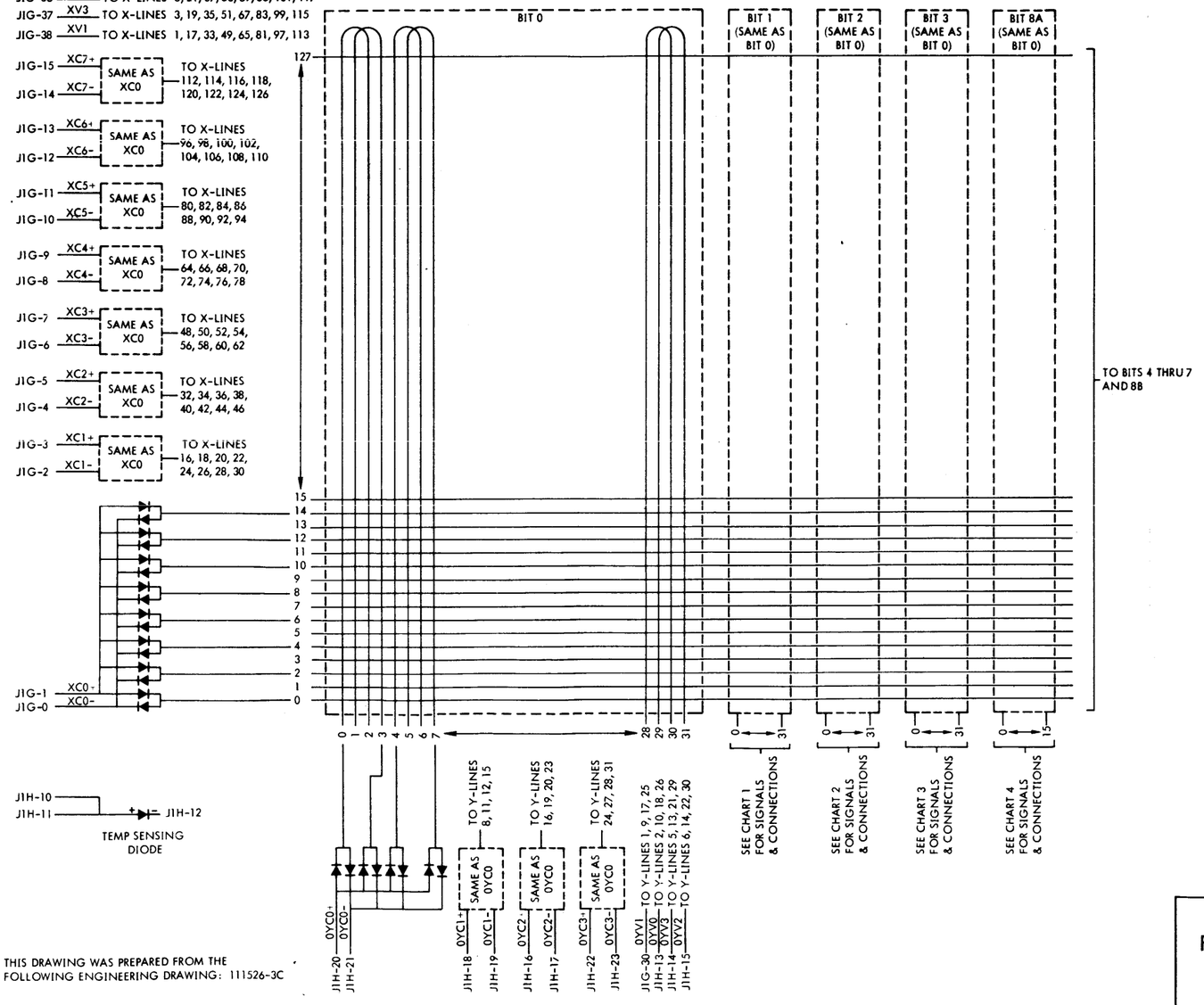
SIGNAL	PIN	CONNECTS TO Y-LINE
2YC0+	J1H-32	0, 3, 4, 7
2YC0-	J1H-33	
2YC1+	J1H-34	8, 11, 12, 15
2YC1-	J1H-35	
2YC2+	J1H-36	16, 19, 20, 23
2YC2-	J1H-37	
2YC3+	J1H-38	24, 27, 28, 31
2YC3-	J1H-39	
2YV0	J1G-23	2, 10, 18, 26
2YV1	J1G-22	1, 9, 17, 25
2YV2	J1G-25	6, 14, 22, 30
2YV3	J1G-24	5, 13, 21, 29

CHART 3 (BIT 3)

SIGNAL	PIN	CONNECTS TO Y-LINE
3YC0+	J1H-40	0, 3, 4, 7
3YC0-	J1H-41	
3YC1+	J1H-42	8, 11, 12, 15
3YC1-	J1H-43	
3YC2+	J1H-44	16, 19, 20, 23
3YC2-	J1H-45	
3YC3+	J1H-46	24, 27, 28, 31
3YC3-	J1H-47	
3YV0	J1G-19	2, 10, 18, 26
3YV1	J1G-18	1, 9, 17, 25
3YV2	J1G-21	6, 14, 22, 30
3YV3	J1G-20	5, 13, 21, 29

CHART 4 (BIT 8A)

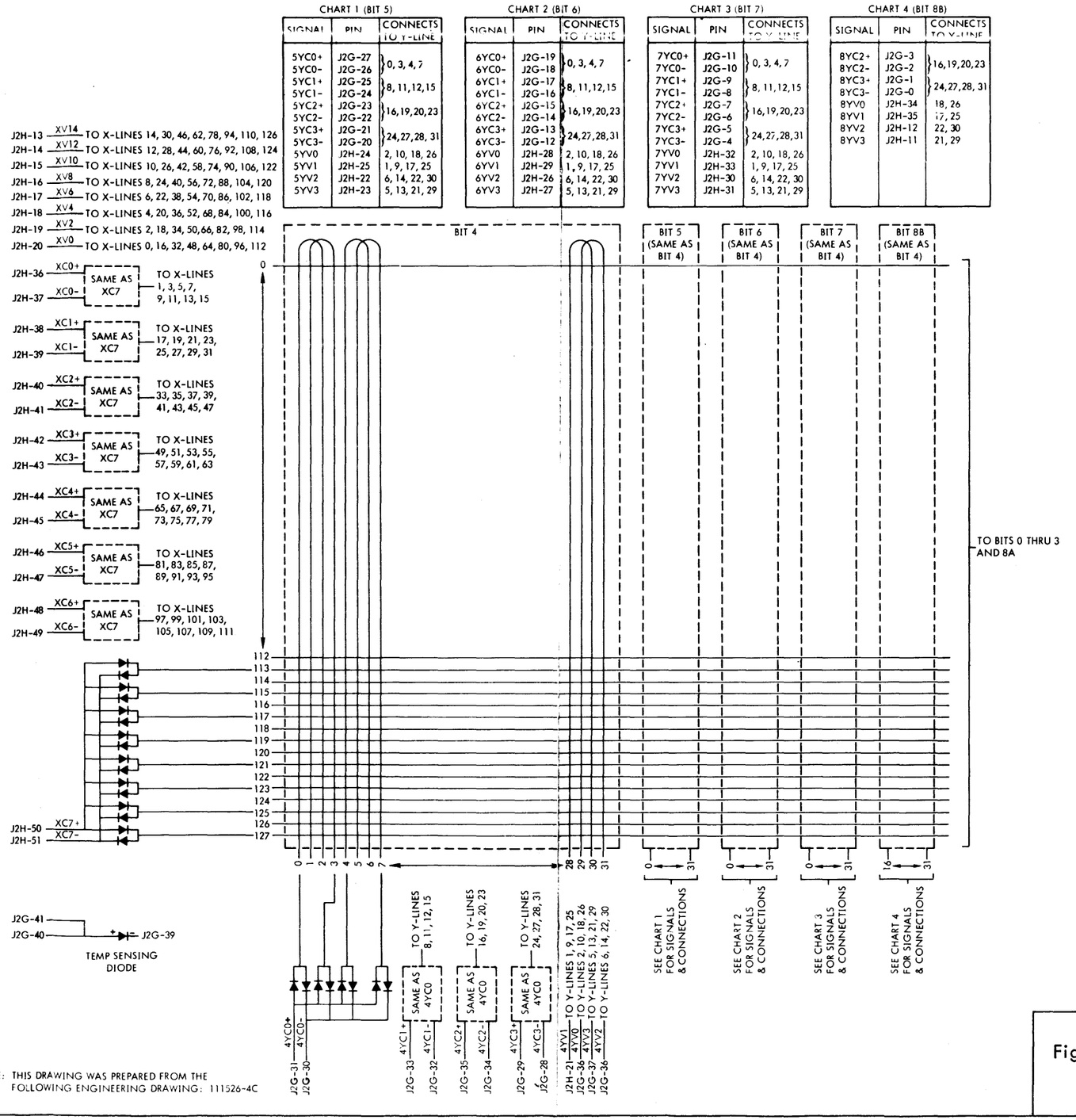
SIGNAL	PIN	CONNECTS TO Y-LINE
8YC0+	J1H-48	0, 3, 4, 7
8YC0-	J1H-49	
8YC1+	J1H-50	8, 11, 12, 15
8YC1-	J1H-51	
8YV0	J1G-17	2, 10
8YV1	J1G-16	1, 9
8YV2	J1G-39	6, 14,
8YV3	J1G-40	5, 13



NOTE: THIS DRAWING WAS PREPARED FROM THE FOLLOWING ENGINEERING DRAWING: 111526-3C

Figure 3-92. Core Diode Module, Left Half Wiring Details

901060A. 3152



NOTE: THIS DRAWING WAS PREPARED FROM THE FOLLOWING ENGINEERING DRAWING: 111526-4C

Figure 3-93. Core Diode Module, Right Half Wiring Details

901060A.3153

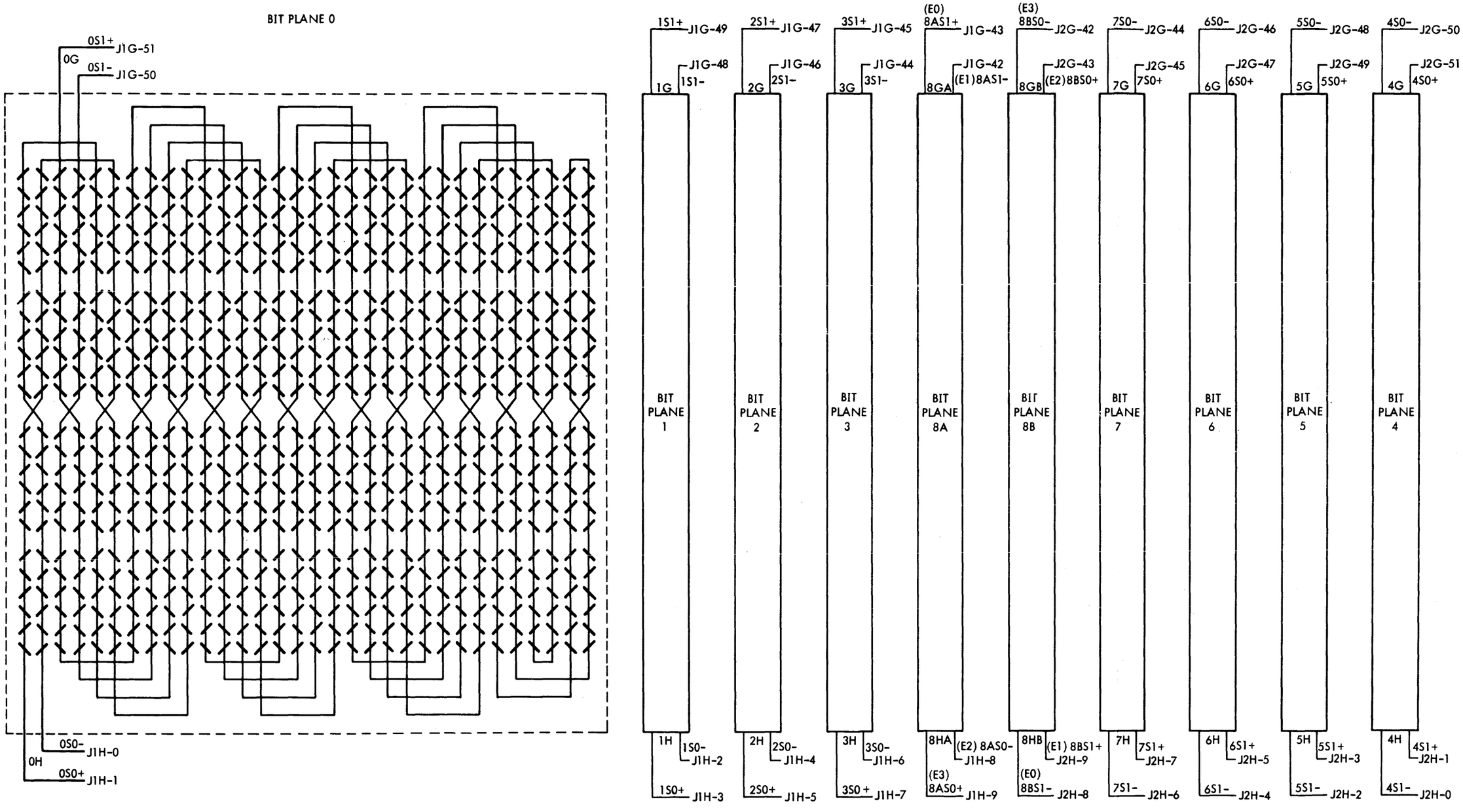


Figure 3-94. Sense Line Wiring in a 4K Core Diode Module

901060A.3110

Figure 3-90 shows a nine-bit core diode module, lying open, as seen from the core side. The presence of diodes on the reverse side of the core diode module is indicated by the dotted lines. The X lines, which are connected across the hinge, are indicated by the two X lines shown in the diagram. In an actual core diode module, there are a total of 128 X wires jumpered across the hinge connecting the left half of the core diode module with the right half.

The designations J1G, J1H, J2G, and J2H indicate the jacks that receive one core diode module. Jacks J1G, J1H, J2G, and J2H are shown as typical. The signals to be found on the pins on the wiring side of the jacks are shown in figure 3-91.

Considerable detail concerning the wiring of one half of a nine-bit core diode module is shown in figure 3-87. Not all wires are shown, however. Instead, their presence is implied. For example, there are 128 X wires. Figure 3-92 shows X wires 0 through 15. The gap indicates that X wires continue from 16 to 127. The diagram also contains the implication that the X wires continue through bit planes 0, 1, 2, 3, 8A and to the second half of the core diode module shown in figure 3-93. In the latter drawing, the bit planes are shown in reverse order if the core side of the module is being viewed. In both drawings (of both halves of a nine-bit core diode module), a portion of the diode decode matrix is shown, with pin and signal numbers given. In studying both drawings, note the way the Y wire is folded back. The foldback is concerned with the anticoincidence principle discussed in the following paragraphs. The temperature sensing diode, explained in paragraph 3-57, is shown in both diagrams, with pin numbers included.

Figure 3-94 shows a nine-bit core diode module with emphasis placed on the sense windings. The bit planes are shown as they appear on both halves of an open core diode module. Details are shown for bit plane 0. The remaining bit planes are shown as blocks with pin and signal numbers shown at sense winding terminations. In figure 3-94, although considerable detail is shown for bit plane 0, all detail is not shown because of the repetitive and greatly detailed nature of this type of unit. A complete bit plane has 4,096 cores. A lesser number of cores is shown in the diagram, with the remainder implied.

Each core, shown schematically in figure 3-94 as a straight line, represents a core standing on end, as the rim of the core is seen when viewed from above. The sense wire goes through the hole in the core. Sense wires terminate with the pin and signal number shown.

DRIVE SYSTEM MODULES ST10 AND ST11. Figure 3-95 shows a simplified diagram of the Sigma 5 memory drive system. In this discussion of the drive system, references to letters refer to lettered points shown in the diagram. The term "switch" means "electronic switch."

In the Sigma 5 memory drive system diagram, only one drive wire is shown with its pair of decode diodes. The number of drive wires used varies with the size of the memory. For example, with a 16K memory, the X drive system would have 32 additional diodes connected to point A and 32 additional diodes connected to point B. Each diode connects, through a drive line, to one of the 32 voltage switches in the X drive matrix. In the same 16K memory (X drive system), there would be a total of 16 drive wires connected to point C. Each drive wire is connected to one of the 16 current switches in the X drive matrix.

The mode of operation is as follows: To pass a positive current through the drive wire, the positive current switch and negative voltage switch are turned on. The flow of current takes the following path: From the $+V_D$ supply (point D in figure 3-95), through the 53-ohm resistor, through the positive current switch, through the drive wire (and cores), through the negative voltage switch, to ground.

To pass a negative current, the positive voltage switch and negative current switch are turned on. The flow of current takes the following path: From the $+V_D$ supply (point E in figure 3-95), through the positive voltage switch, through the drive wire (and cores), through the negative current switch, through the 53-ohm resistors, to ground.

The supply Vm is not externally generated in the power supply system. Instead, Vm is the product of 37 (4X and 33Y) 53-ohm divider chains passing current continuously through the Vm clamp diodes. On each switch module there are decoupling capacitors for Vm, as shown in figure 3-95.

The 1K resistors connected to the voltage switches bias all drive wires quiescently to Vm. The 1K resistors connected to the current switches reverse-bias all the diodes, so that drive current is not lost into other lines as charging current. The Vm diodes prevent the voltage at the current switches (developed across the inductance of the drive line during the rise of the current) from exceeding Vm. Thus, forward-biasing the decode diodes is prevented.

The current and voltage switches are all SDS 226 transistors. Their bases are driven by transformers whose primaries consist of one turn and secondaries consist of four turns. The magnetizing current built up in the transformer during the time the transistor is on serves to turn the transistor off when base drive is removed.

The drive circuit described is used for all four X drive matrices and all 33 Y drive matrices. The 53-ohm resistors are located in the uppermost chassis underneath the fans to provide heat dissipation. Connection is made to the resistors by means of twisted pairs to minimize the inductance of the drive loop.

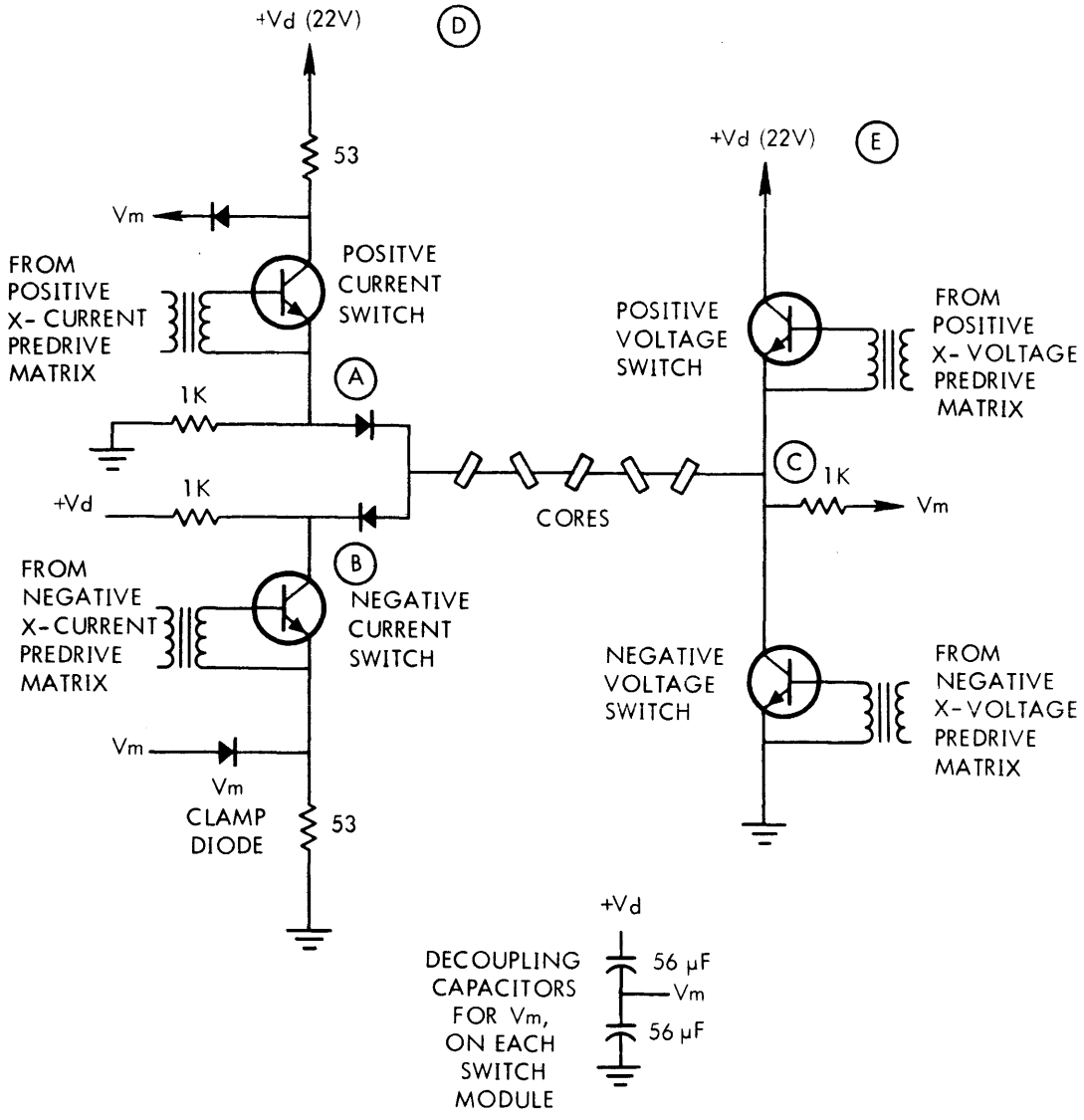


Figure 3-95. Memory Core Drive System, Simplified Schematic

X Core Matrix. The X matrix for each 4K core memory increment consists of 32 positive and negative current switches and 16 positive and negative voltage switches. As the size of the core memory is increased, current and voltage switches must be added. Current and voltage switches for each 4K increment are shared in matrix form, as shown in figure 3-96, so that a 16K memory requires 64 positive and negative X current switches and 32 positive and negative voltage switches. Note that the same number of X current and voltage switches is required for a 12K memory as for a 16K memory.

The relationship of positive and negative X current switches to positive and negative voltage switches is shown in figure 3-97. Each X current switch connects to 16X buses. The corresponding X bus wires (first, second ... through sixteenth) of each current switch are connected and tied to a corresponding X voltage switch.

Y Core Matrix. The Y matrices, which are bit oriented, consist of four positive and negative current switches and four positive and negative voltage switches for each bit. Current and voltage switches for each 4K increment of core memory are shared in a matrix arrangement as indicated in figure 3-98. One set of current switches selects the 0 through 4K and 8K through 12K memory stacks, and another set selects the 4K through 8K and 12K through 16K memory stacks. One set of voltage switches selects the 0 through 4K and 4K through 8K stacks, and another set selects the 8K through 12K and 12K through 16K stacks.

The Y current and voltage switches for a single bit are shown in figure 3-99. This matrix arrangement is typical of all bits. Note that each of the 16 Y wires is folded back on itself through the cores and that in the bit plane the X and Y wires intersect at two cores. For given directions of current flow, the currents add in the core at one intersection and cancel at the other. A reversal of one of the currents enables the other core to be accessed. This technique is called anticoincidence. Current direction flow in the Y windings is dependent on the status of the address bits L22, L23, and L25.

Predrive System, Model ST22. In this discussion of the predrive system, the term "switch" means "electronic switch." As indicated in the previous discussion of the drive system, to read a word of memory it is necessary to do the following: Turn on, simultaneously, four X positive current switches, four X negative voltage switches, 33 Y positive current switches, and 33 Y negative voltage switches, or a similar combination. To restore the word on the second half-cycle, the complementary (interchange positive and negative) set of switches is operated.

To operate 33 Y positive current switches, the primaries of the transformers belonging to this group of switches are connected in series. Other groups of switches are operated similarly. Because the primaries consist of one turn, switches are arranged in groups of four on ST10 and ST11 modules. One wire passes through each set of four transformers. In the case of the Y switches, nine of the transformer groups are in series. Therefore, 36 electronic switches (three of which are not used) are operated by one predrive current. Because of the 4:1 step-down in current through the transformer, the predrive current is approximately 300 ma. Therefore, a power transistor is used in the predrive circuit.

Figure 3-100 shows a typical Y predrive circuit. The three address lines and a timing signal, TPYC (time for positive Y current), are ANDed into an integrated inverter. The output of the AND gate drives the primary of a 6:4 transformer on the base of the power transistor (SDS 226). The output of the SDS 226 transistor drives approximately 300 ma into the string of 36 (of which 33 are used) voltage or current switch primaries.

The Y decode for each bit has the following:

a. Eight positive current switches

b. Eight negative current switches

c. Eight positive voltage switches

d. Eight negative voltage switches

Because of the number of electronic switches mentioned above, there are eight circuits like the one shown in figure 3-100. Therefore, there are a total of 32 such Y predrive circuits.

The X predrive system uses predrive circuits identical to the Y predrive circuits (ST22), but the outputs of the circuits are arranged in the form of a matrix. This is done because a larger number of electronic switches are used in the X drive switch matrices. For example, 16 x 32 electronic switches are used in the X drive system. The number of electronic switches used in the Y drive system is 8 x 8, as explained earlier.

The X positive current predrive matrix is shown in figure 3-101. Note that there are only four transformer primaries in series because there are only four X drive switch matrices. Matrices of the type shown in the diagram are used for the following:

a. Positive X current

b. Negative X current

c. Positive X voltage

d. Negative X voltage

The relationship of the X and Y matrix predrive circuits to the transformed address bits in the L-register is shown in figure 3-102.

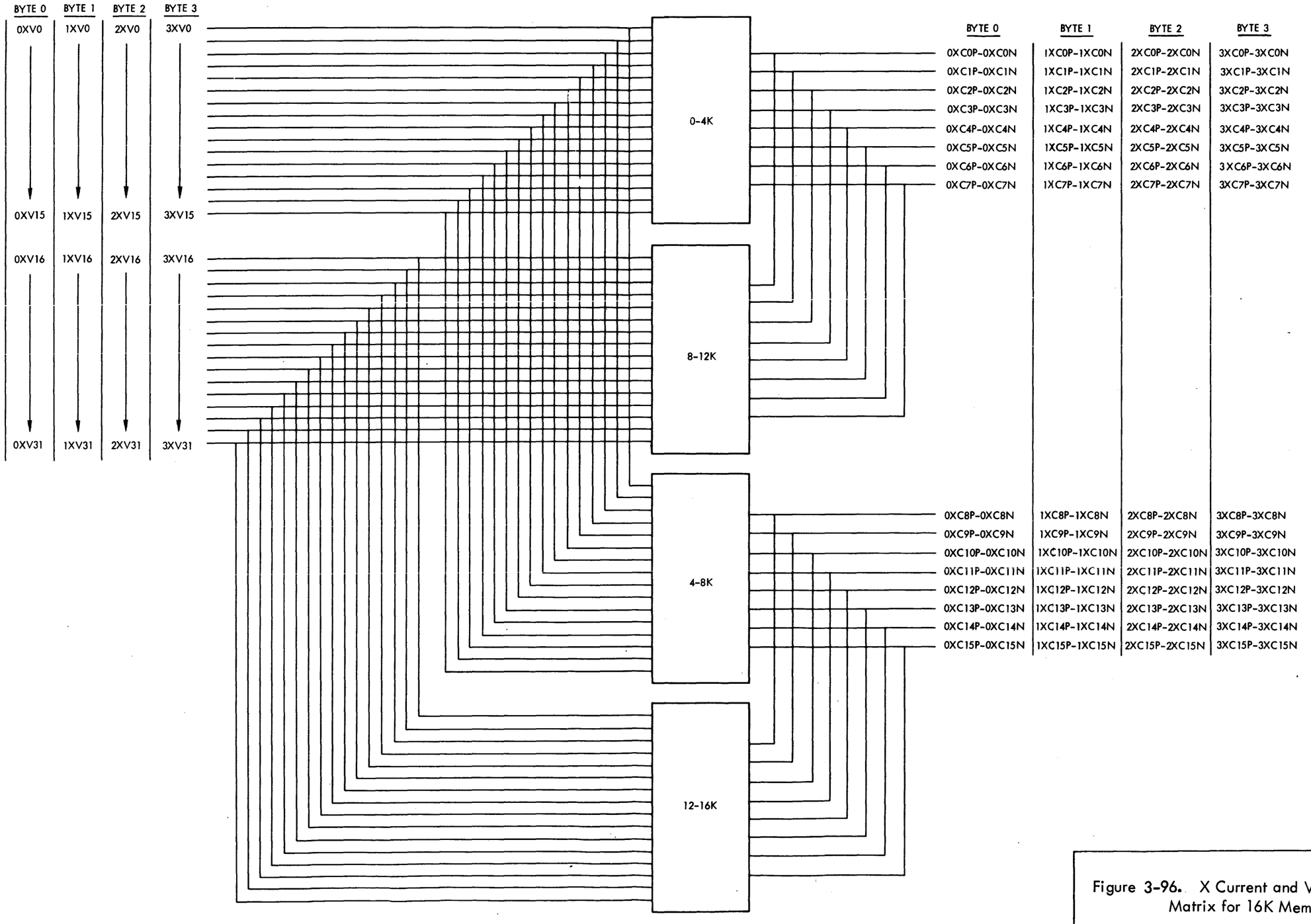


Figure 3-96. X Current and Voltage Switch Matrix for 16K Memory

901172A.3338

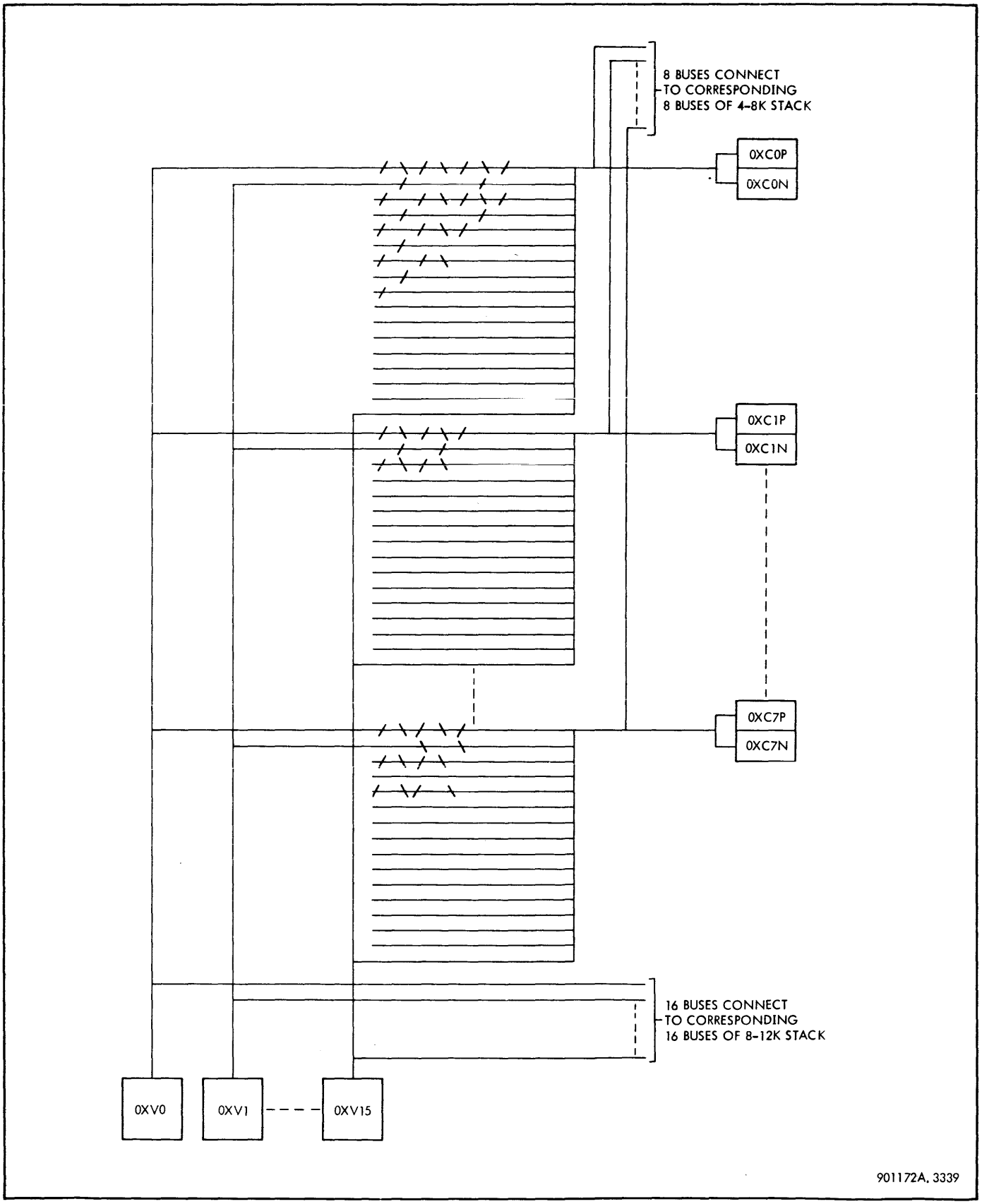


Figure 3-97. X Current and Voltage Switch Matrix, Byte 0, 4K Stack

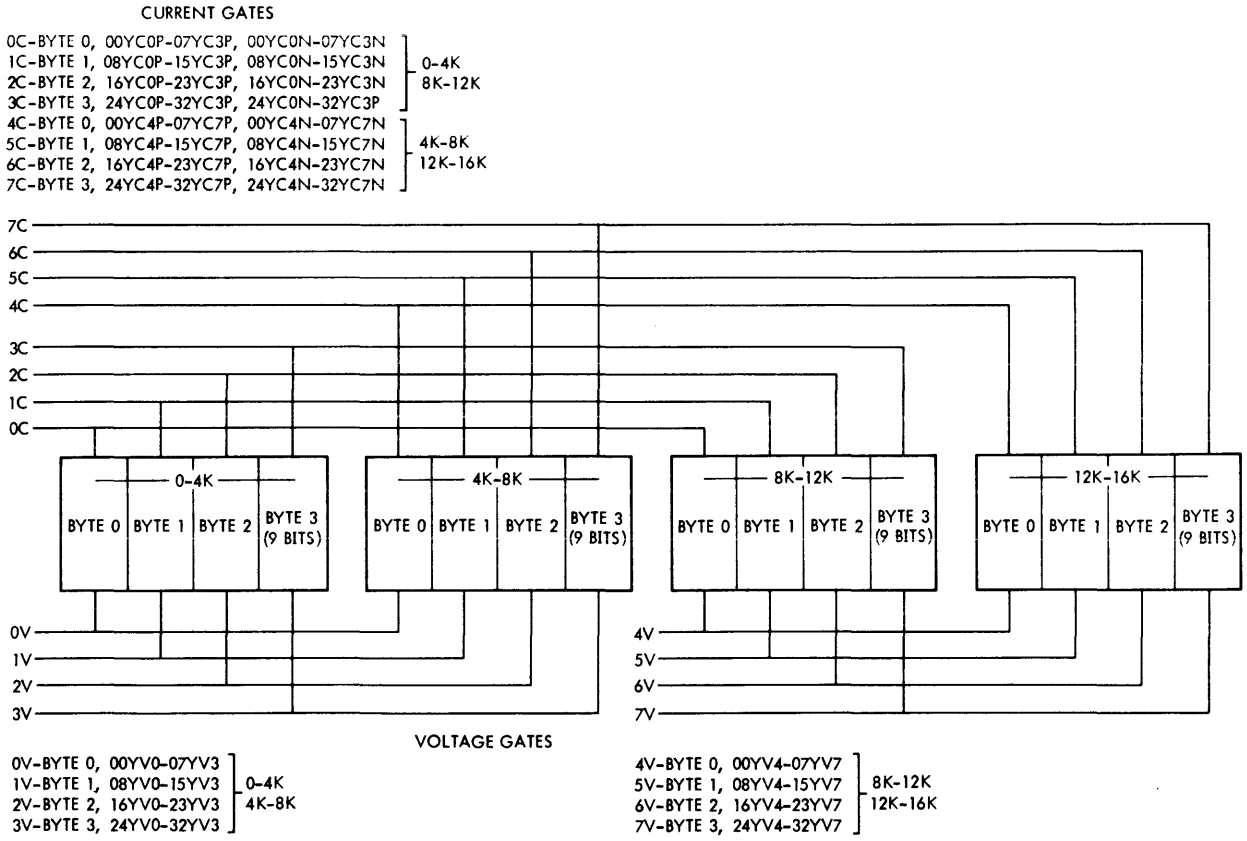


Figure 3-98. Y Current and Voltage Switch Matrix for 16K Memory

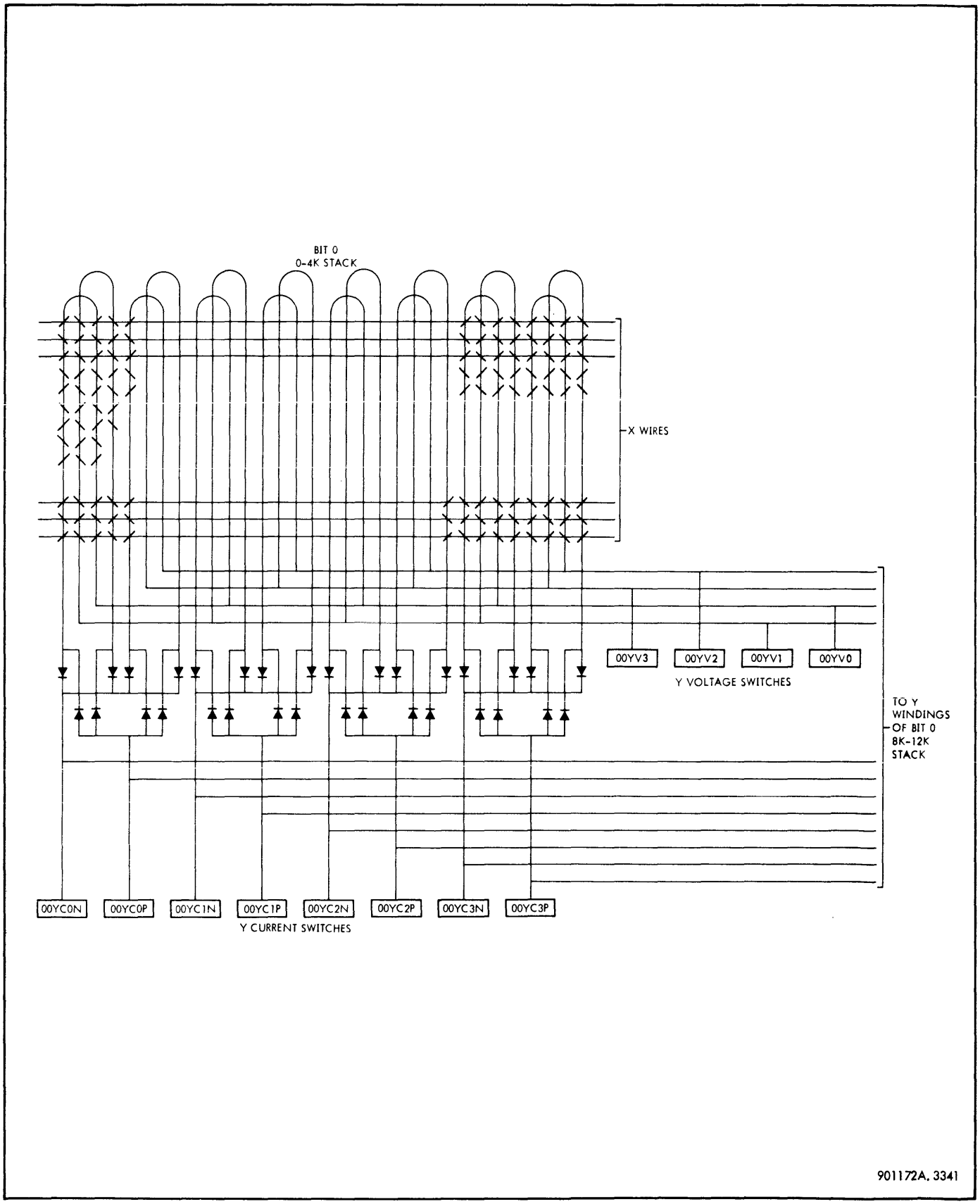


Figure 3-99. Y Current and Voltage Switch Matrix for Bit 0

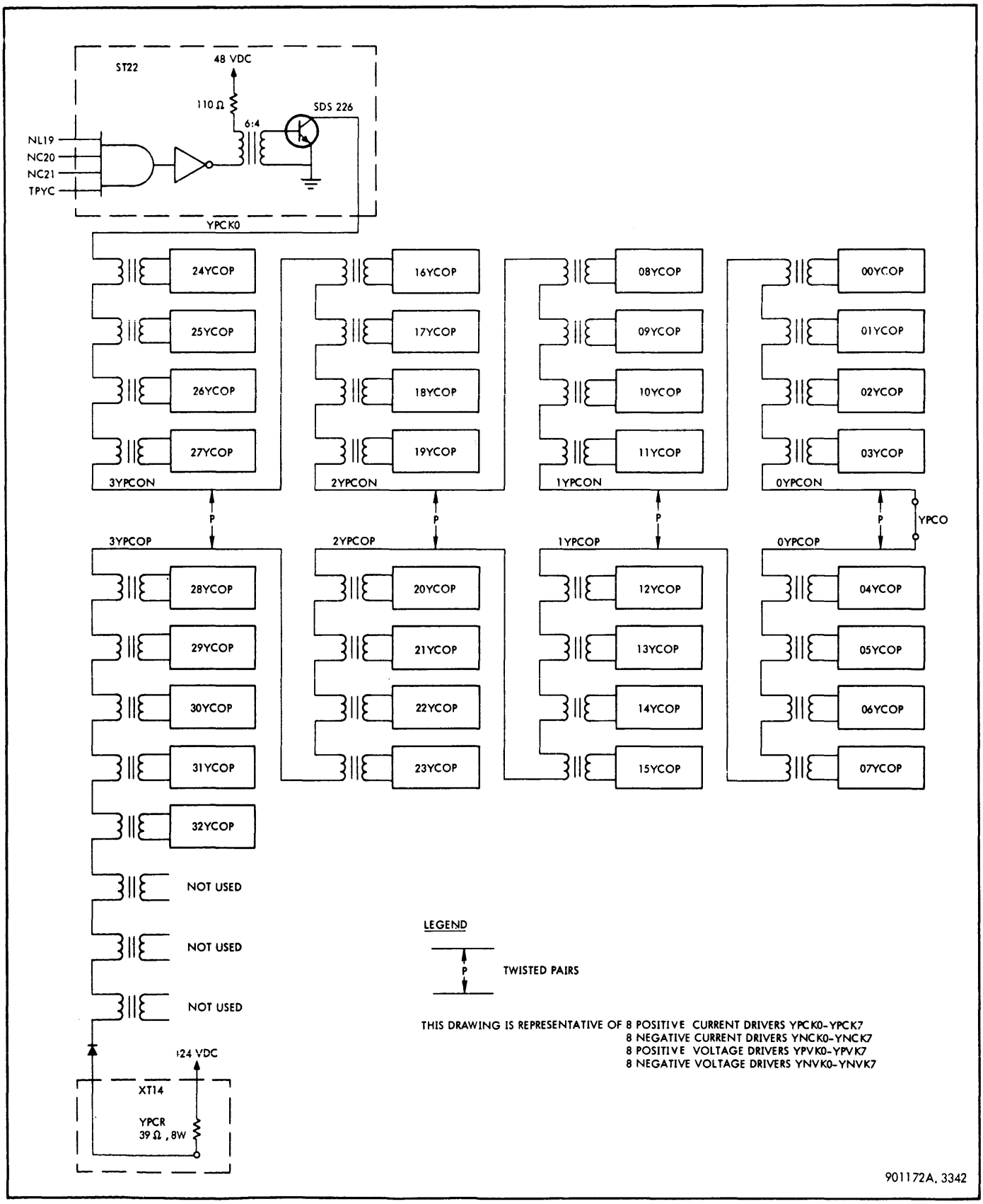


Figure 3-100. Y Positive Current Predrive/Drive Coupling, Simplified Schematic

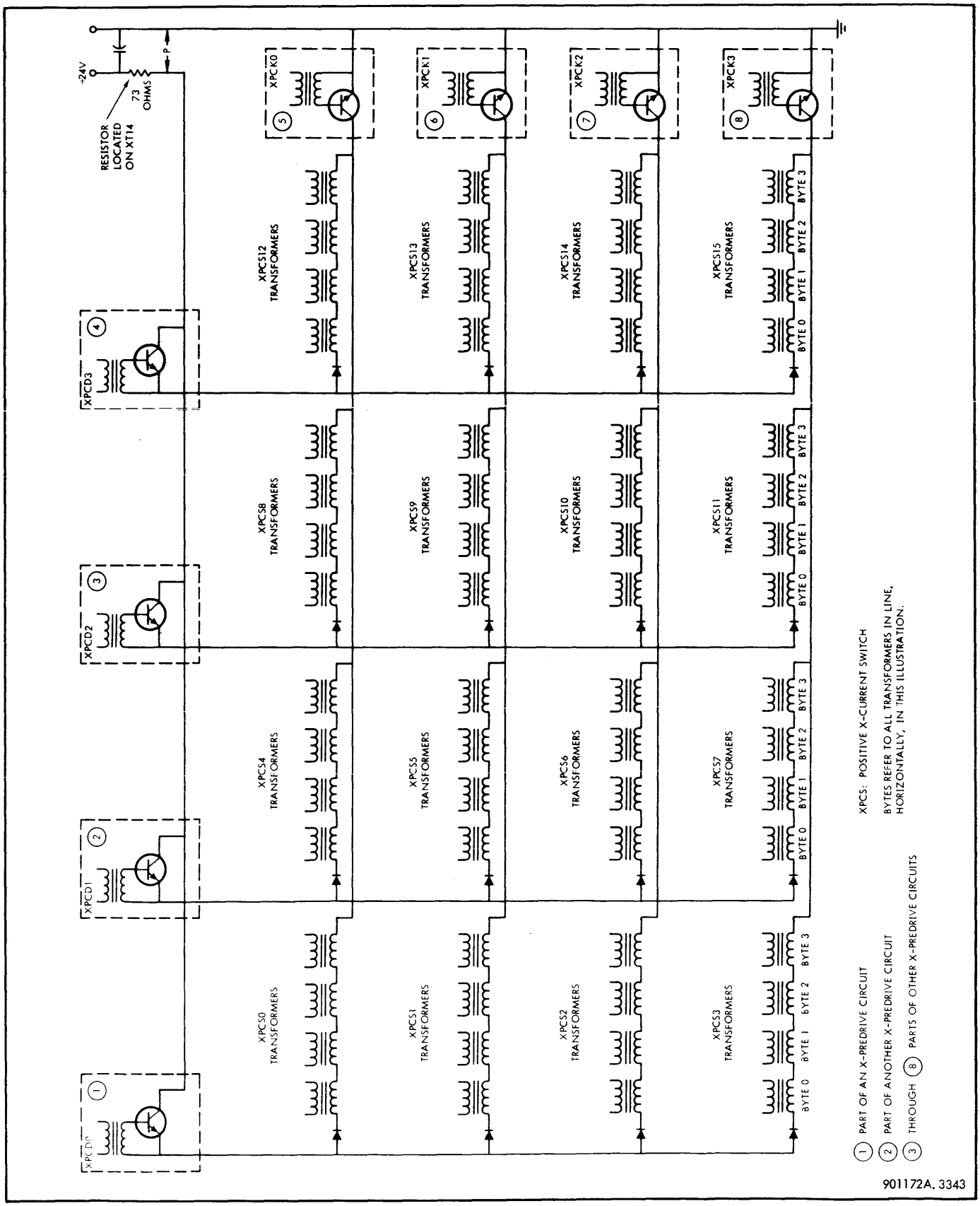


Figure 3-101. X Positive Current Predrive Matrix, Simplified Schematic

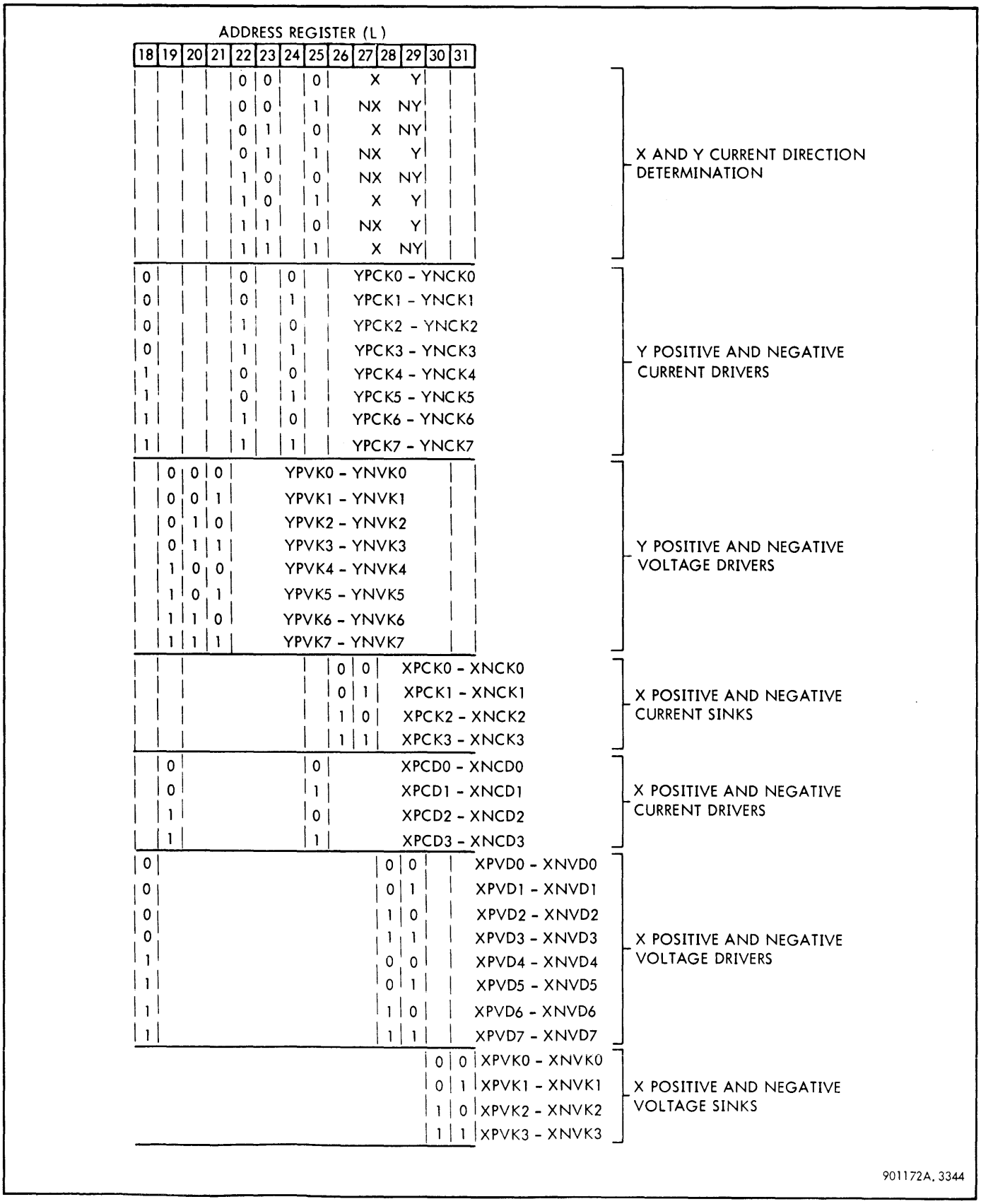


Figure 3-102. X and Y Predrive Selection Relative to Memory Address

Figures 3-103 through 3-106 show the X predrive matrices for positive and negative X current and X voltage switches, module locations, and output pin numbers.

Figures 3-107 through 3-109 show the Y predrive matrices for positive and negative Y current and Y voltage switches, module locations, and output pin numbers.

Current Direction Control. The effects of X and Y half-current direction through the memory cores was shown in figure 3-84. Current polarity is determined ultimately by the transformed address bits L22, L23, and L25. X current polarity is determined by the status of the address bits L22 and L25. If these bits are equal to each other, signal X will be true. If these bits are not equal to each other, signal NX will be true.

$$X = NL22\ NL25 + L22\ L25$$

$$NX = NL22\ L25 + L22\ NL25$$

Y current polarity is determined by the status of the address bits L22, L23, and L25. If these three bits contain an even number of ones (or all zeros), signal Y will be true. If these bits contain an odd number of ones, signal NX will be true.

$$Y = NL22\ NL23\ NL25 + L22\ L23\ NL25 + L22\ NL23\ L25 + NL22\ L23\ L25$$

$$NY = L22\ L23\ L25 + NL22\ NL23\ L25 + NL22\ L23\ NL25 + L22\ NL23\ NL25$$

The input logic to the positive or negative X and Y positive or negative current and voltage drivers includes not only the polarity determination logic (X or NX and Y or NY), but the proper timing signals as well. With the system of current reversals used in the Sigma 5 memory, a timing signal can occur either in the read or the write half-cycle. Timing signal TPXC (time for positive X current) is an example of a signal that can occur either in the read or the write half-cycle, depending upon the address selected.

The timing diagram, figure 3-110, shows the principal memory timing signals relating to the memory cores. Note that during the read half-cycle, the X-current lags the Y-current in time by 80 nsec. This is done purposely during the read half-cycle to minimize the effects of delta noise. Delta noise is the result of the nonsquareness of the BH curve, and causes a small flux change to be generated in the read winding when a half-current is passed through either the X or Y winding of a nonselected core.

The following buffer latch logic describes how the positive and negative current and voltage predrivers are controlled according to the read and write half-cycle timing and the status of the X, NX, Y, and NY signals.

$$TPXC = TPXC\ NTR320\ NTW480 + X\ TR080 + NX\ TW240$$

$$TNXC = TNXC\ NTR320\ NTW480 + NX\ TR080 + X\ TW240$$

$$TNXV = TNXV\ NTR320\ NTW480 + X\ TR000 + NX\ TW240$$

$$TPXV = TPXV\ NTR320\ NTW480 + NX\ TR000 + X\ TW240$$

$$TPYC = TPYC\ NTR320\ NTW480 + Y\ TR000 + NY\ TW240$$

$$TNYC = TNYC\ NTR320\ NTW480 + NY\ TR000 + Y\ TW240$$

$$TPYV = TPYV\ NTR320\ NTW480 + NY\ TR000 + Y\ TW240$$

$$TNYV = TNYV\ NTR320\ NTW480 + Y\ TR000 + NY\ TW240$$

Sense Preamplifier, Module HT26. The sense preamplifier is a differential pair transistor, Q1, shown in figure 3-111. Input to the sense preamplifier is buffered from severe common mode excursions by transformer T1, called the common mode transformer or balun. The gain of the differential pair is controlled by the internal emitter resistance of each transistor. The emitter resistance serves as a feedback resistor. The emitter current is derived from a current source, Q2, and the precision 1,000-ohm resistor. Voltage Ve controls the gain of the preamplifier by changing the emitter current and thus the emitter resistance of Q1. Module ST17 supplies the voltage Ve, which is temperature controlled. This is done to provide gain compensation of preamplifier with temperature.

Transistors Q1 and Q2 are physically located in one housing and are pairs with matched Vbe characteristics. Using matched pairs eliminates Vbe offset error. The preamplifier is made operative or inoperative by switching the emitter current of Q1 on or off. This is done by means of the selector circuit, module ST15, which ANDs address and timing signals and produces an output. The output voltage varies between ground and -8v to activate the preamplifier. The outputs of two preamplifiers are connected on a module. Four module outputs – one for each 4K stack – are connected to a sense amplifier, making a total of eight preamplifiers feeding one sense amplifier. This arrangement is shown in figure 3-112 for bit 0 of four 4K memory stacks, which is typical for all bits 0 through 32. Figure 3-113 lists the sense lines, preamplifiers, preamplifier select circuits, and sense amplifiers for a maximum of 16K memory.

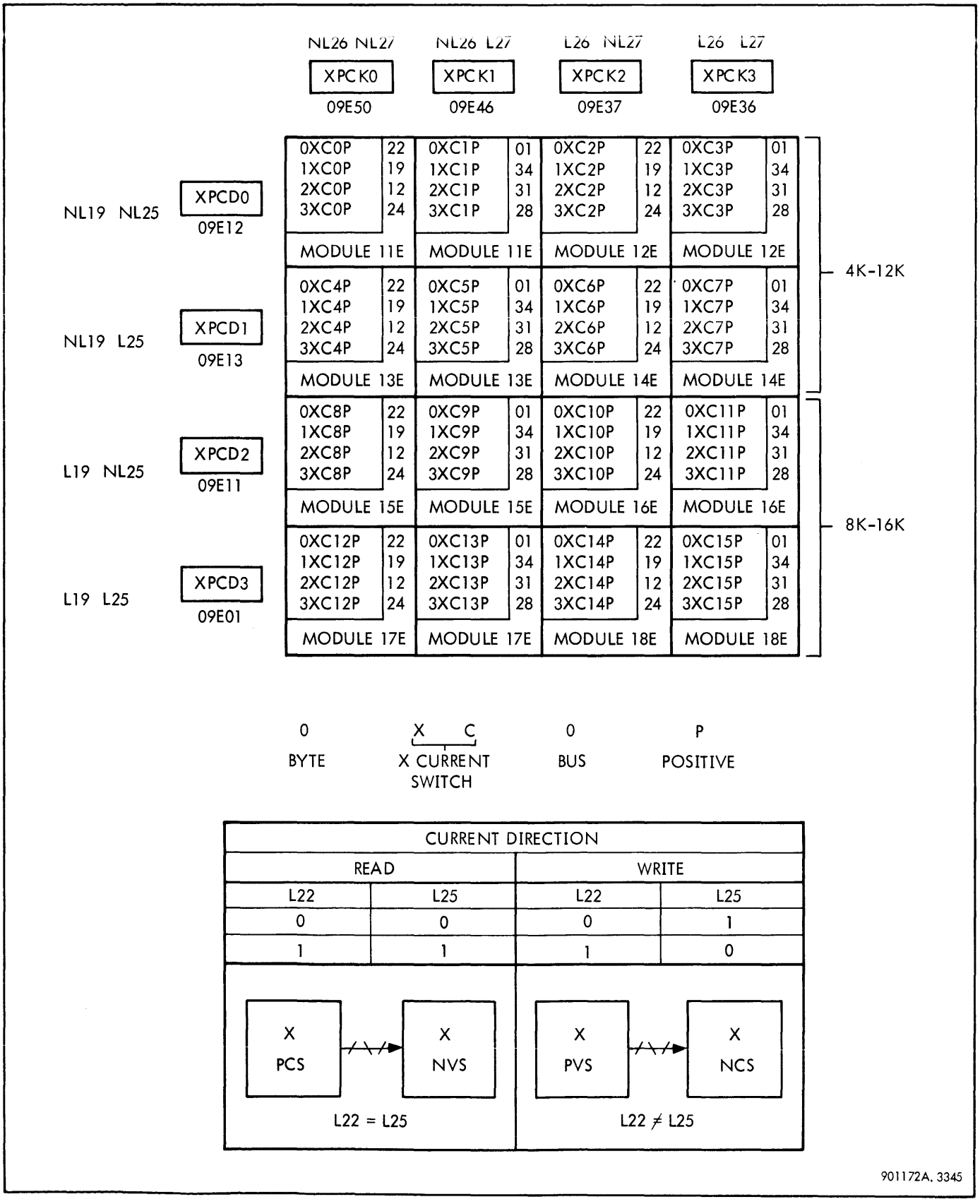


Figure 3-103. X Positive Current Predrive Matrix

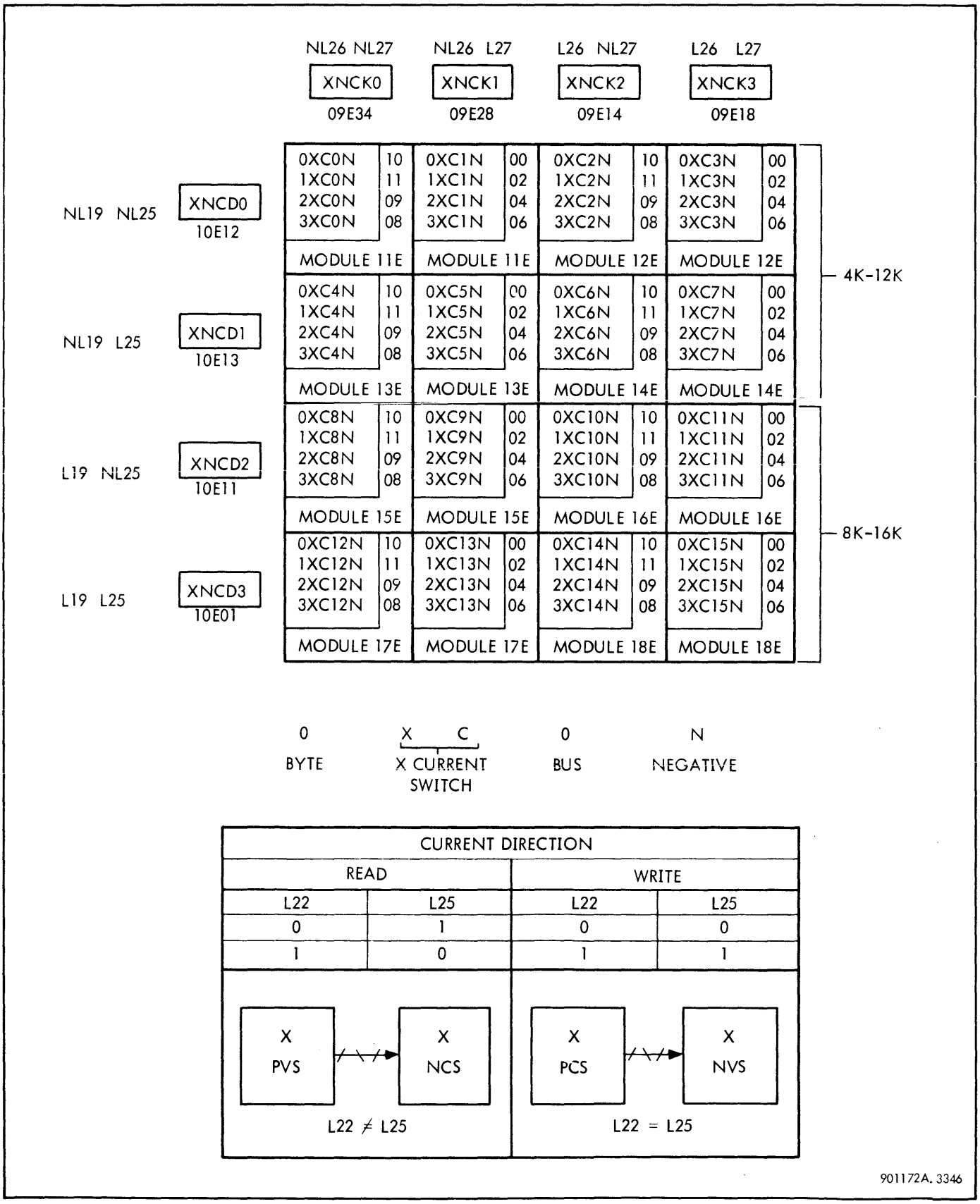


Figure 3-104. X Negative Current Predrive Matrix

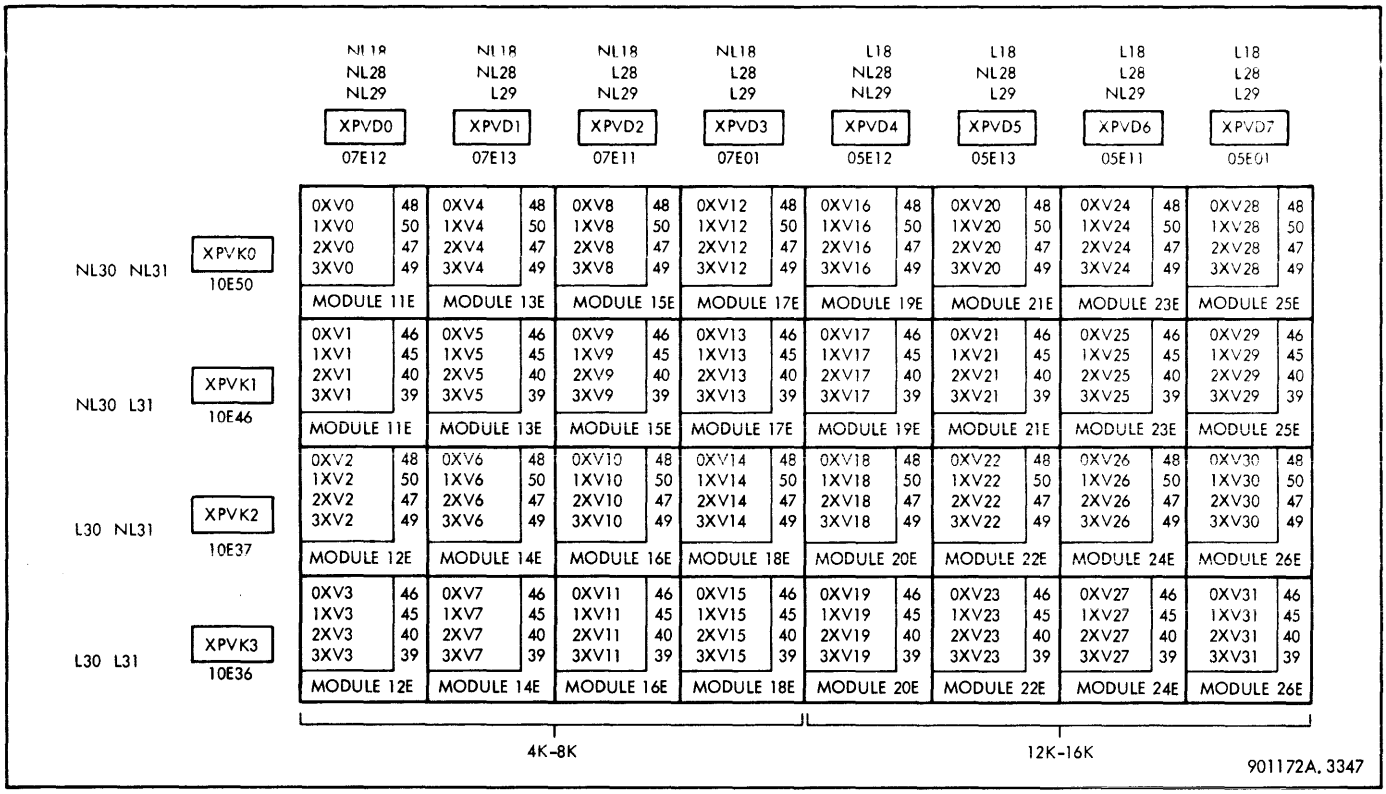


Figure 3-105. X Positive Voltage Predrive Matrix

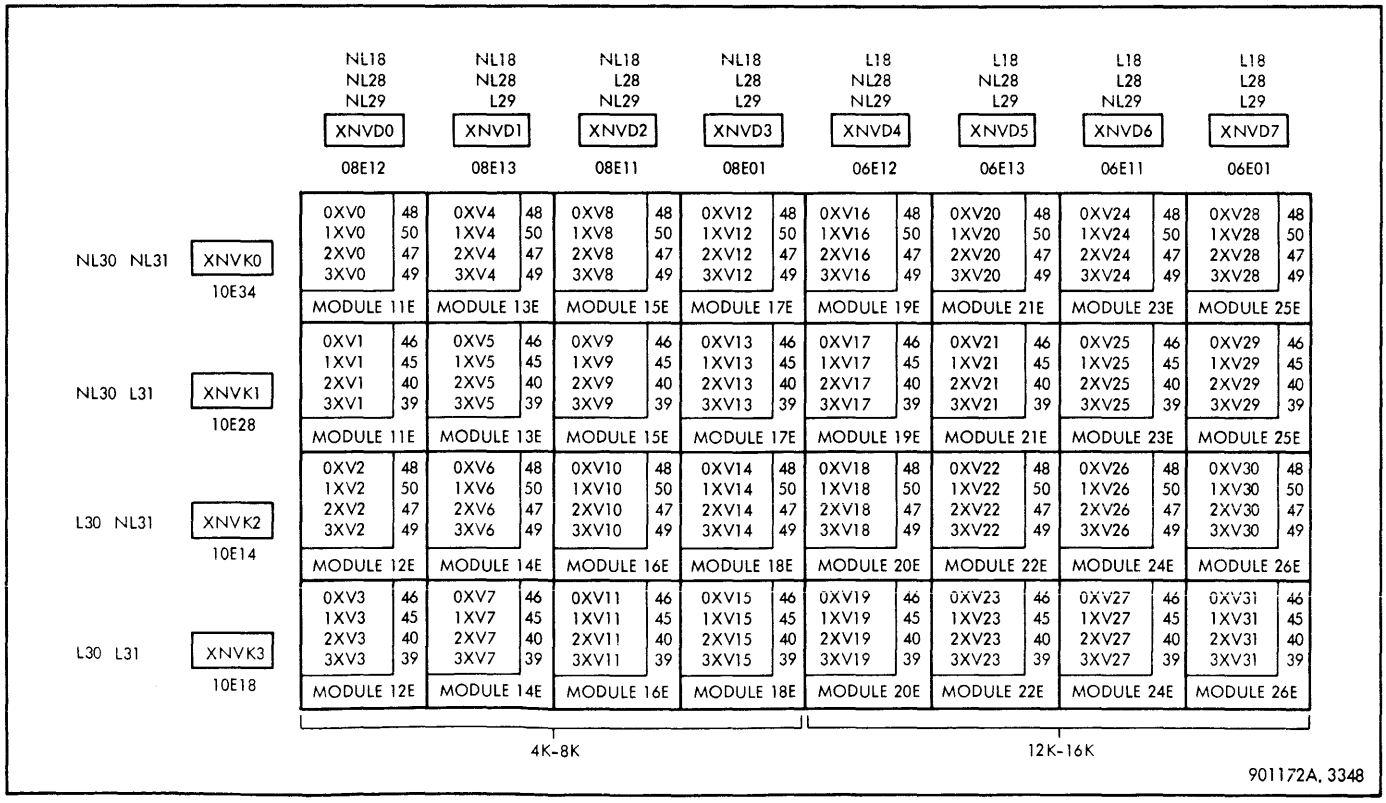


Figure 3-106. X Negative Voltage Predrive Matrix

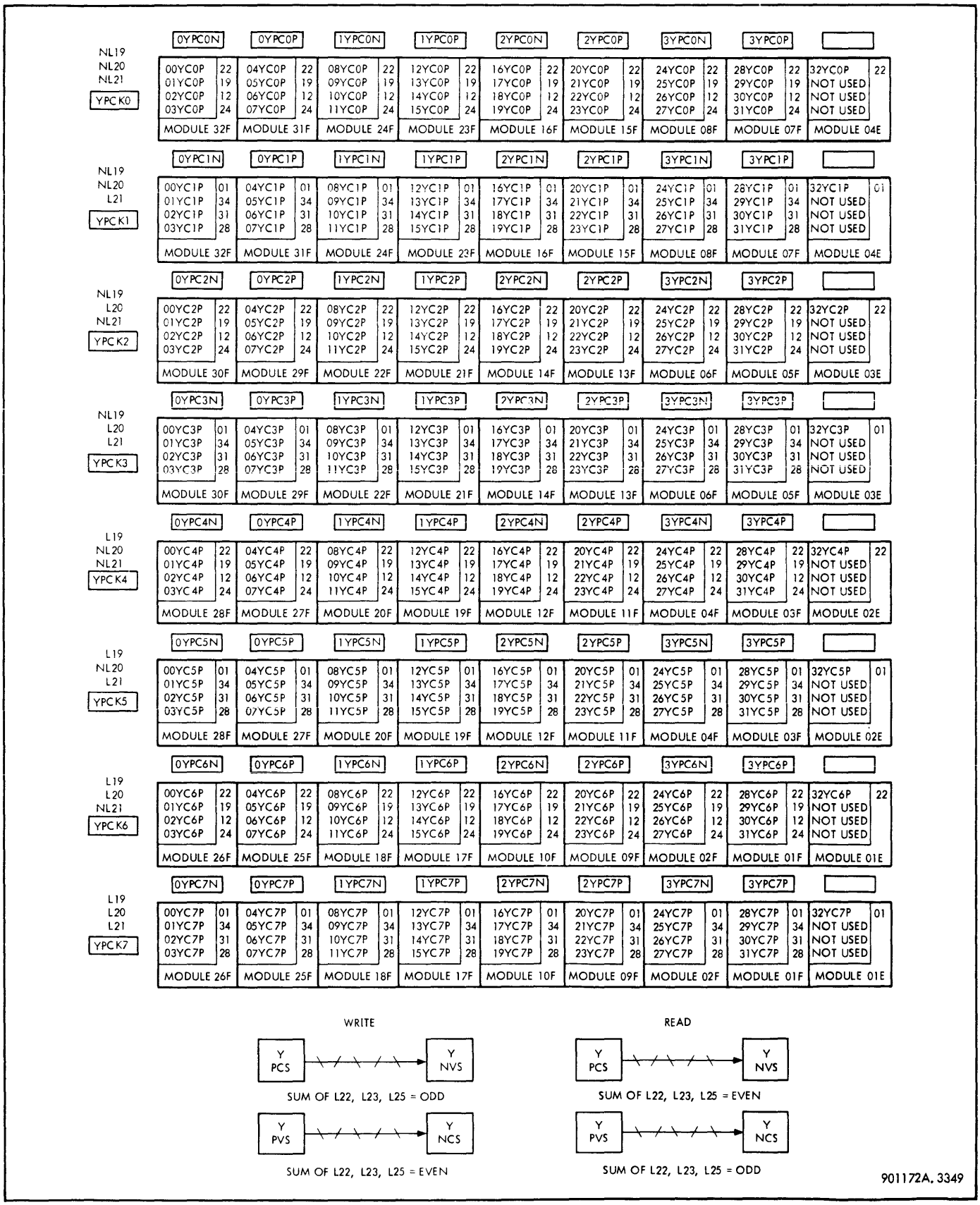


Figure 3-107. Y Positive Current Predrive/Drive Coupling System

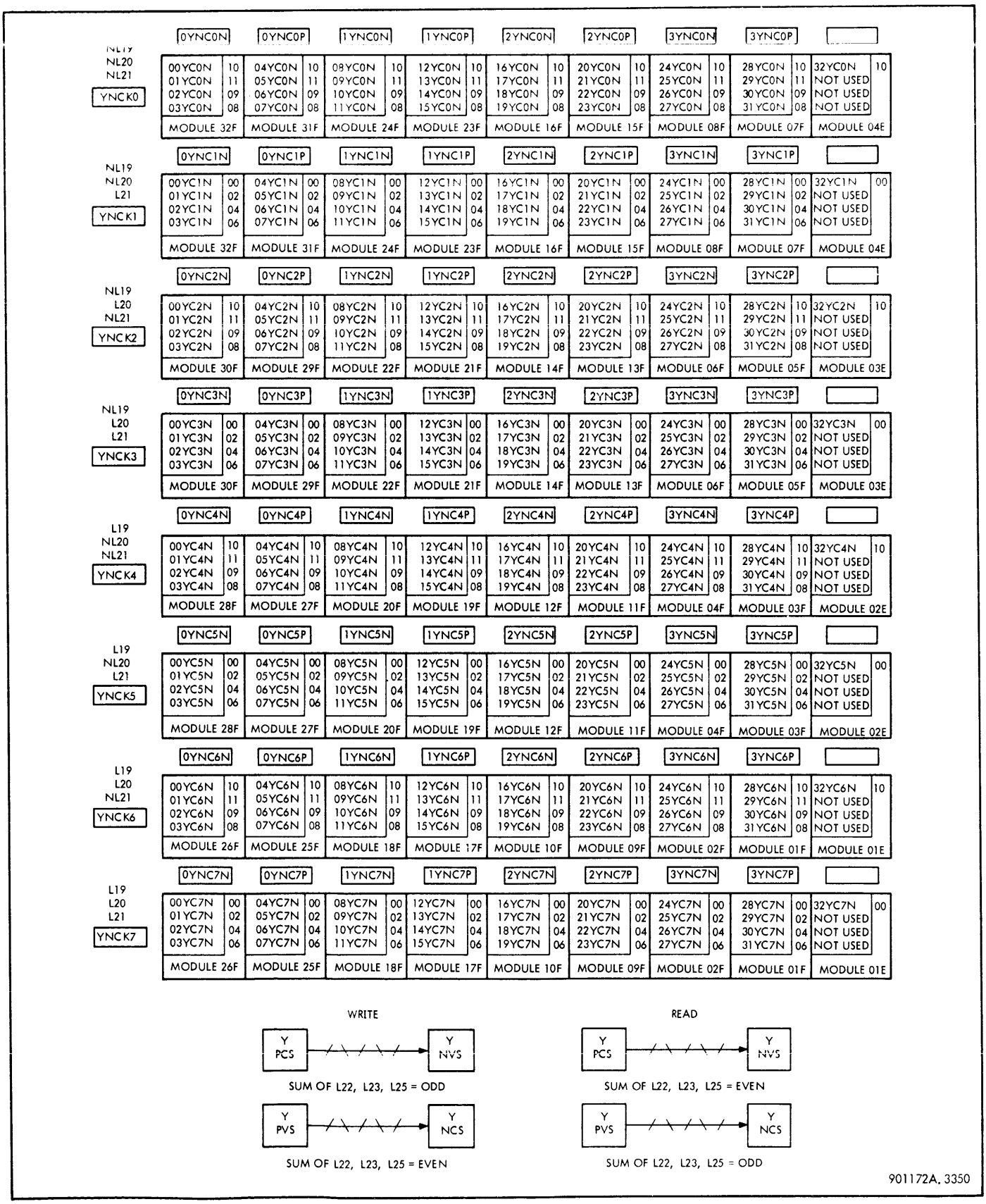


Figure 3-108. Y Negative Current Predrive/Drive Coupling System

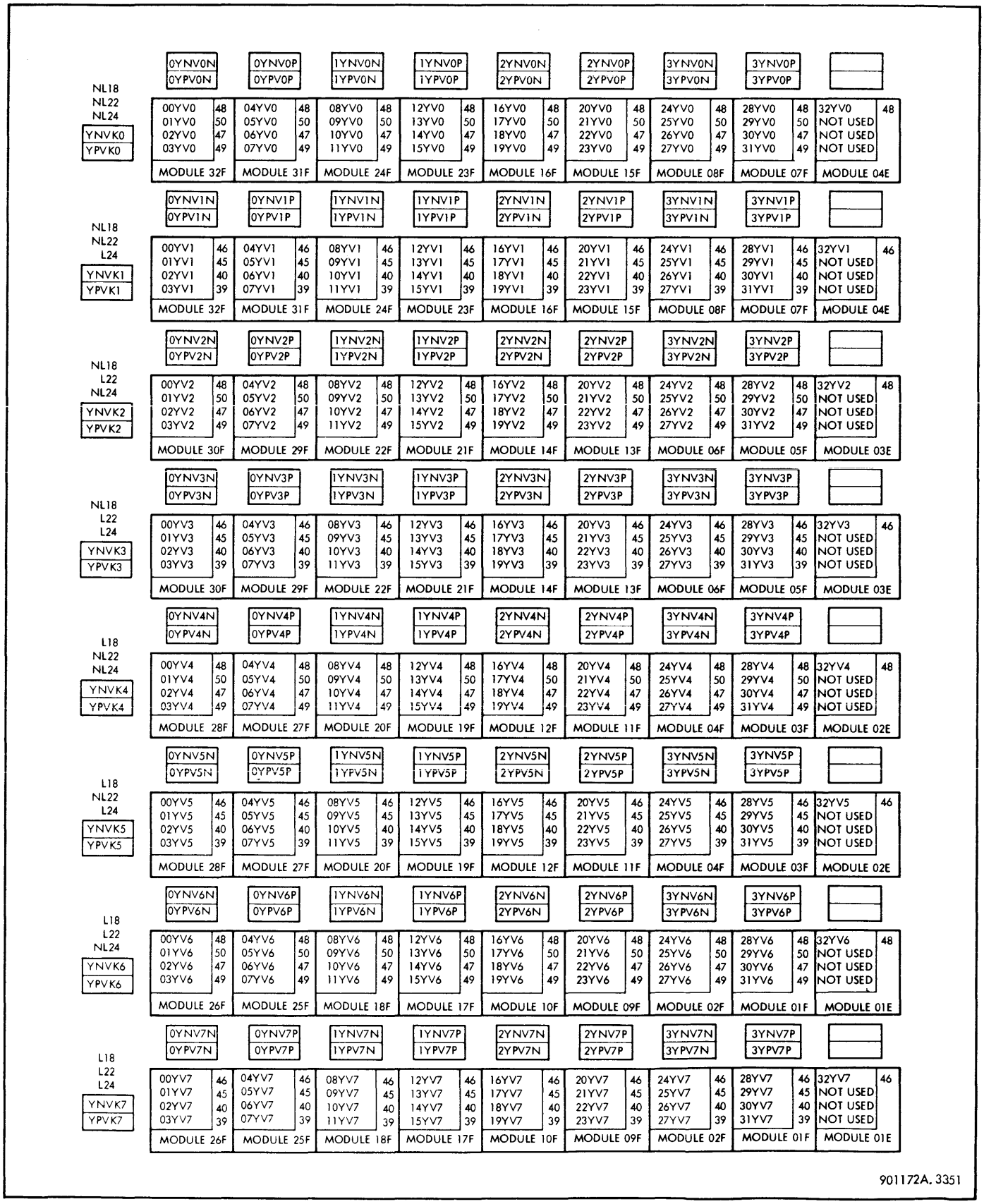
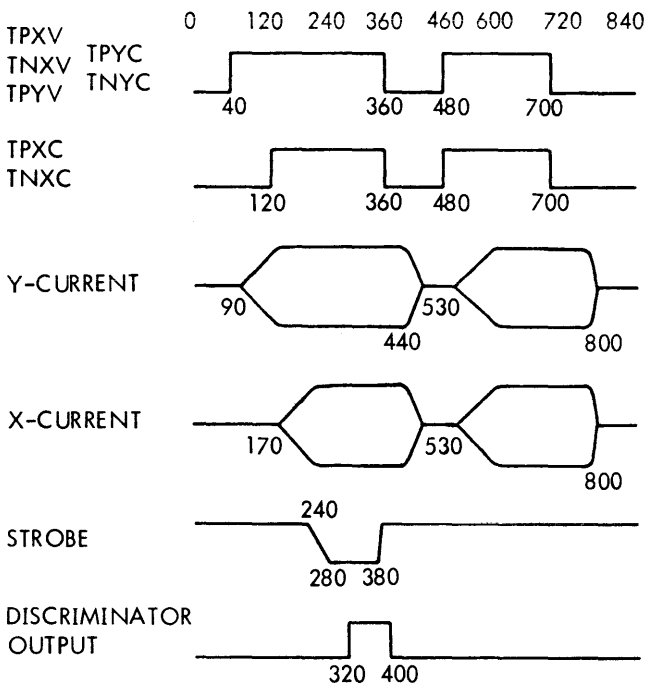


Figure 3-109. Y Positive/Negative Predrive/Drive Coupling System



NOTE: ALL TIMES ARE IN NANOSECONDS AND ARE WITH REFERENCE TO TAP 0 ON THE THE READ DELAY LINE

901172A. 3352

Figure 3-110. Magnetics Timing Diagram

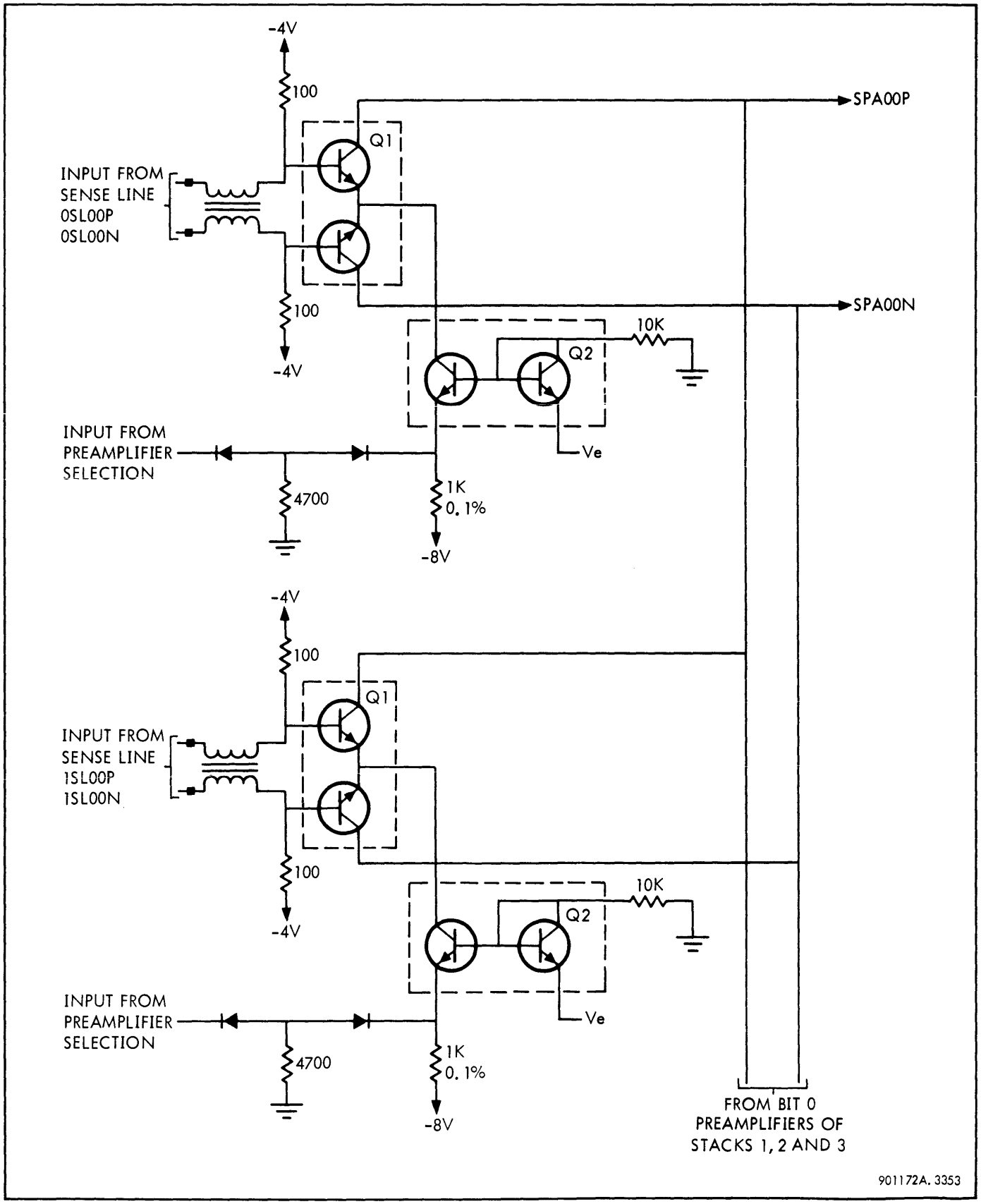


Figure 3-111. Sense Preamplifier (HT26) Simplified Schematic, Bit 0, Stack 0

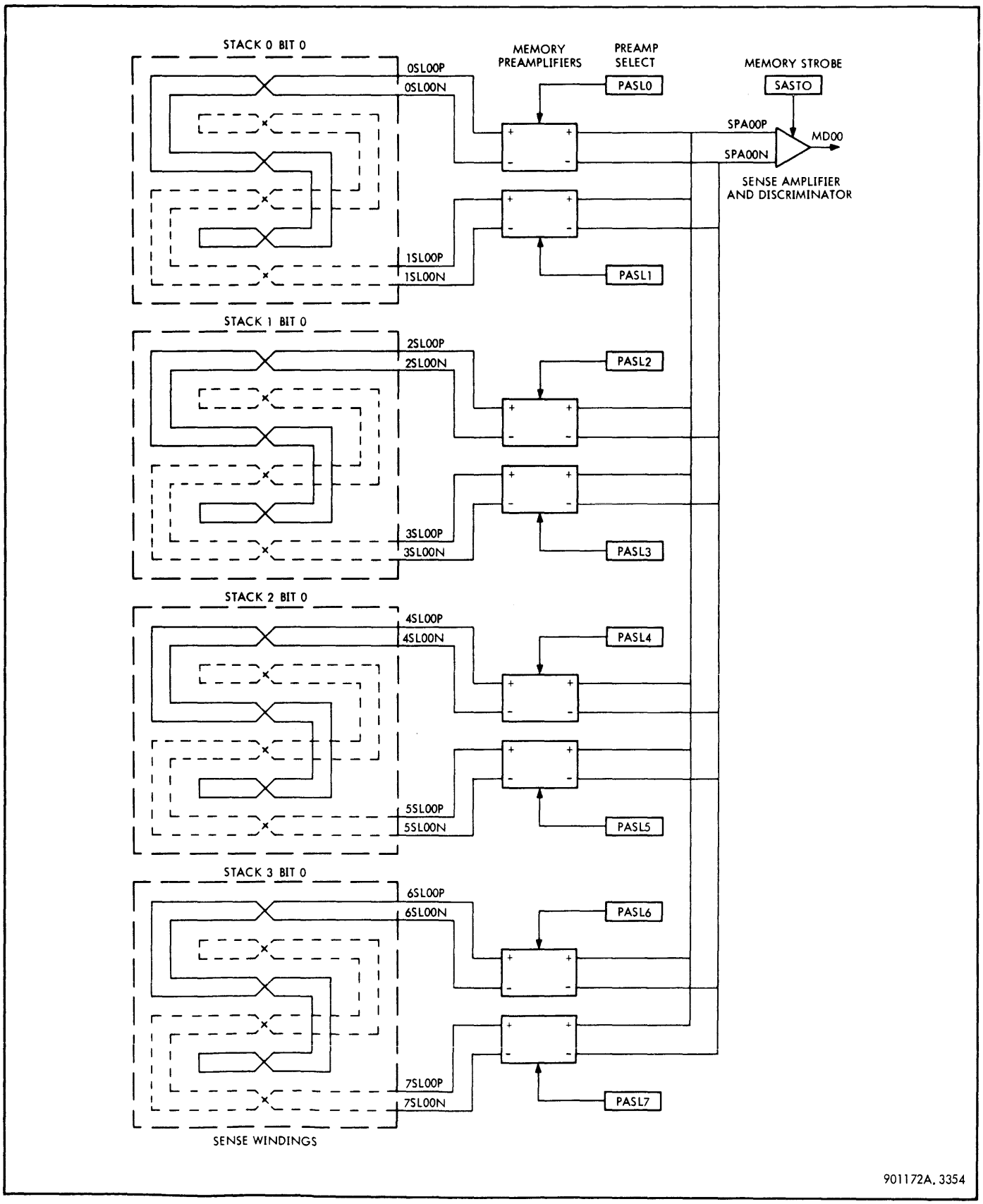


Figure 3-112. Sensing System for Bit 0 (Typical)

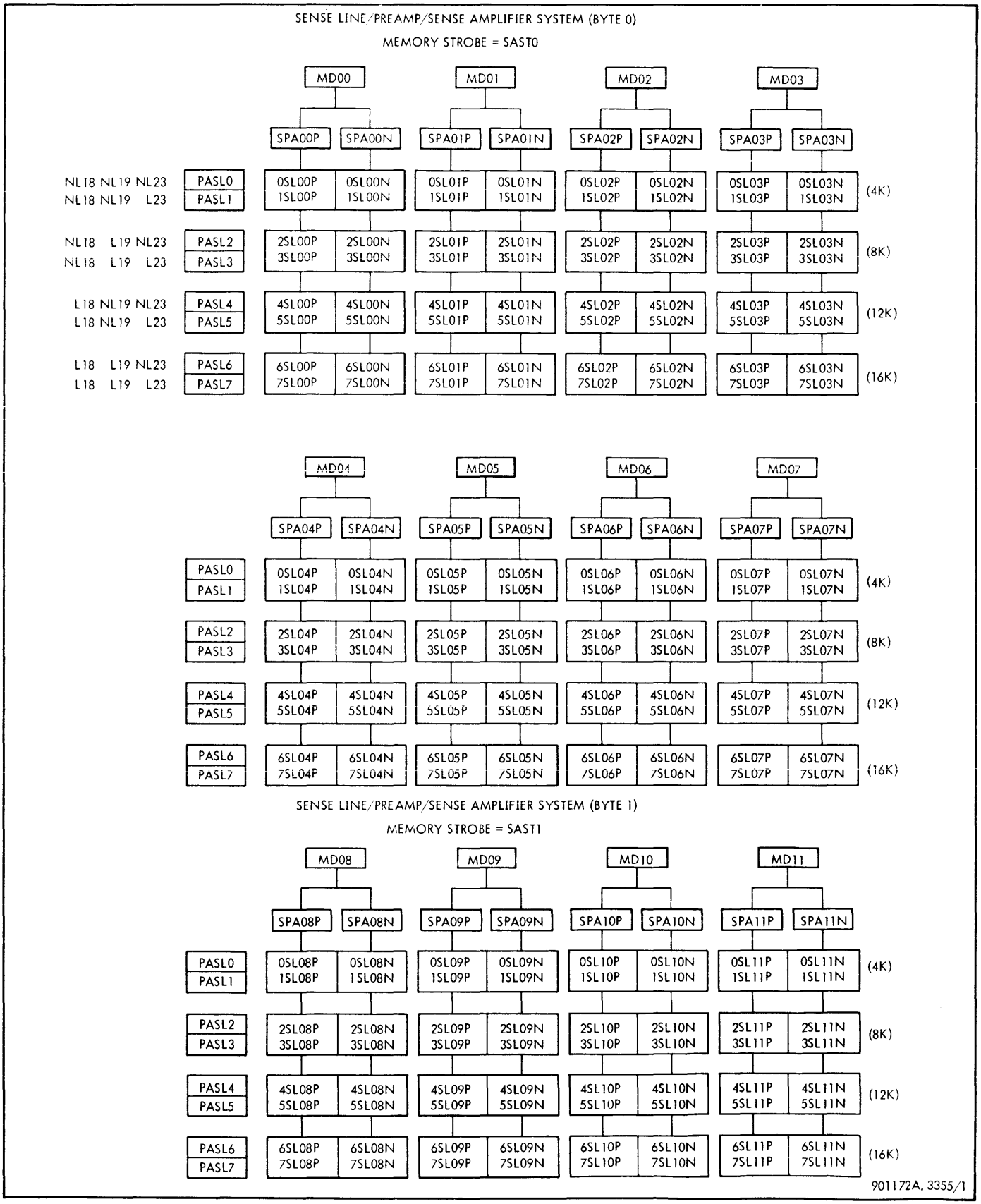


Figure 3-113. Sense Line/Preamp/Sense Amplifier System (Bytes 0 and 1) (Sheet 1 of 3)

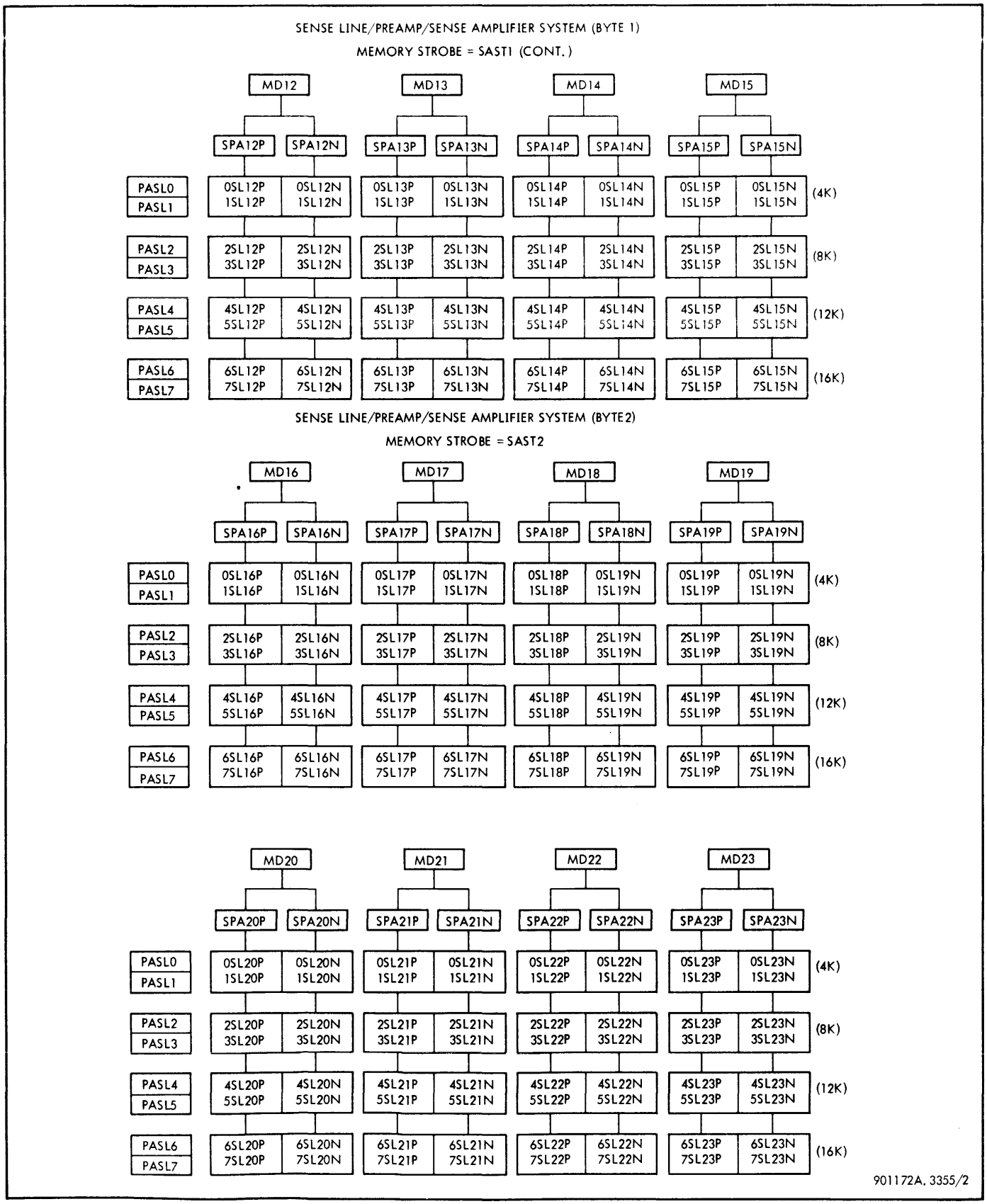


Figure 3-113. Sense Line/Preamp/Sense Amplifier System (Bytes 1 and 2) (Sheet 2 of 3)

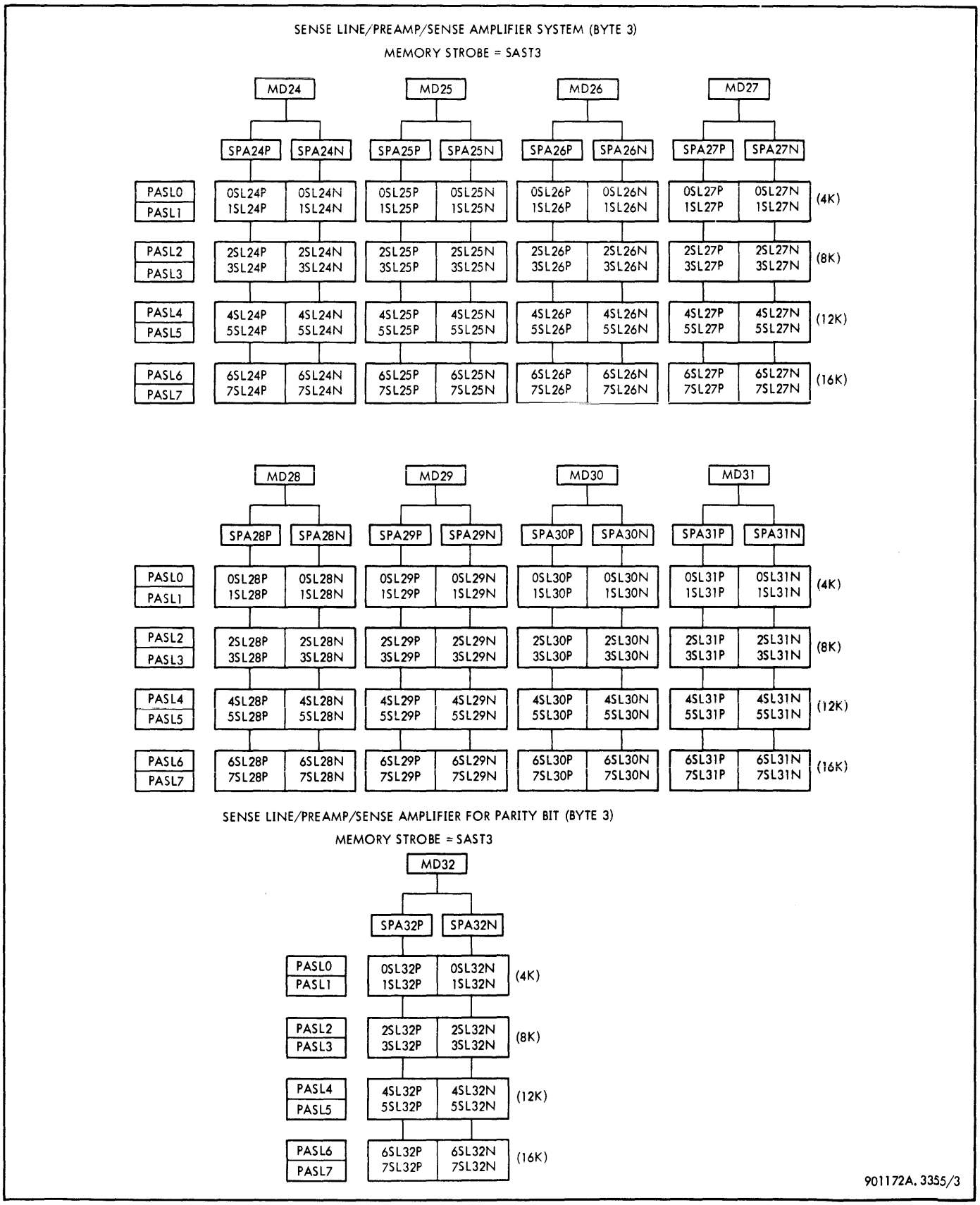


Figure 3-113. Sense Line/Preamp/Sense Amplifier System (Bytes 2 and 3) (Sheet 3 of 3)

The preamplifier select terms PASL0 through PASL7 drop at TR000 time and remain false for 60 nsec during the critical delta noise time. Timing for these select circuits is controlled by signal SDECEN. Two preamplifier select buffers are required for each 4K memory stack. (See figure 3-129.)

```
PASL0  =  NL18 NL19 NL23 SDECEN
PASL1  =  NL18 NL19 L23 SDECEN
  :            :
PASL7  =  L18 L19 L23 SDECEN
```

Sense Amplifier, Module HT11. The sense amplifier can be understood by regarding it as a differential amplifier with feedback connections, as shown in figure 3-114. Idealized waveforms are shown in figure 3-115. Assuming that there is no differential input present, the circuit acts as a unity gain amplifier to the strobe signal. The strobe signal swings over a range of 3v to approximately 0.7v.

In operation, the core output, of about 26 mV, causes the output to go negative by about 1v from its quiescent 3v level. The application of the strobe causes the output to swing down through 0 to about -0.5v. The discriminator discriminates about ground, and therefore responds to such a signal. In the absence of either a strobe signal or a core output signal, the output of the sense amplifier does not fall to ground and therefore no discriminator output is produced.

Figure 3-116 shows a schematic diagram of the sense amplifier. Transistors Q1 and Q2 are grounded base buffer amplifiers and provide a low impedance into which the preamplifier outputs are fed. The outputs of Q1 and Q2 provide a high impedance to drive the sense amplifier in a differential fashion. Transistors Q3, Q4, and Q5 form the sense amplifier, while transistors Q6, Q7, and Q8 make up the discriminator. The strobe signal is generated on module ST34, which also has the Vs and Vt regulators which set the voltage levels between which the strobe output varies.

The application of the strobe signals to the memory amplifiers is byte-oriented, as indicated in figure 3-113.

```
SAST0   =  NTSSTB
SAST1   =  NTSSTB
SAST2   =  NTSSTB
SAST3   =  NTSSTB
NTSSTB  =  NTSSTB NTR140 + TR360 + ...
```

Inhibit System. The operation of addressing the cores during the read (or clear) half-cycle sets all the selected cores to the zero state. Therefore, during write or restore half-cycle operations, ones are written only into those cores that correspond to M-register bits containing ones. It is not necessary to write zeros into those cores that already contain zeros.

To avoid writing a one in a particular location, it is necessary to inhibit the Y current for that bit. It is not possible to inhibit Y current by turning on Y switches in bit planes where a zero is to be written because all 33 Y switches share the same primary wire. The method used for writing zeros (or rather, for not writing ones) is shown in figure 3-117. The drive switch matrix is short-circuited by the inhibit drivers. The circuit used, ST21, is internally identical to the predrive circuit, ST22. The data register signal output is inverted and this output is ANDed with a timing signal TPYI (time for positive Y inhibit) or TNYI (time for negative Y inhibit).

```
00YPIP  =  NM00 TPYI
00YNIN  =  NM00 TNYI
01YPIP  =  NM01 TPYI
01YNIN  =  NM01 TNYI
  :           :
31YPIP  =  NM31 TPYI
31YNIN  =  NM31 TNYI
32YPIP  =  NM32 TPYI
32YNIN  =  NM32 TNYI
TPYI    =  TPYI NTW560 + NY TW200
TNYI    =  TNYI NTW560
           + Y TW200
```

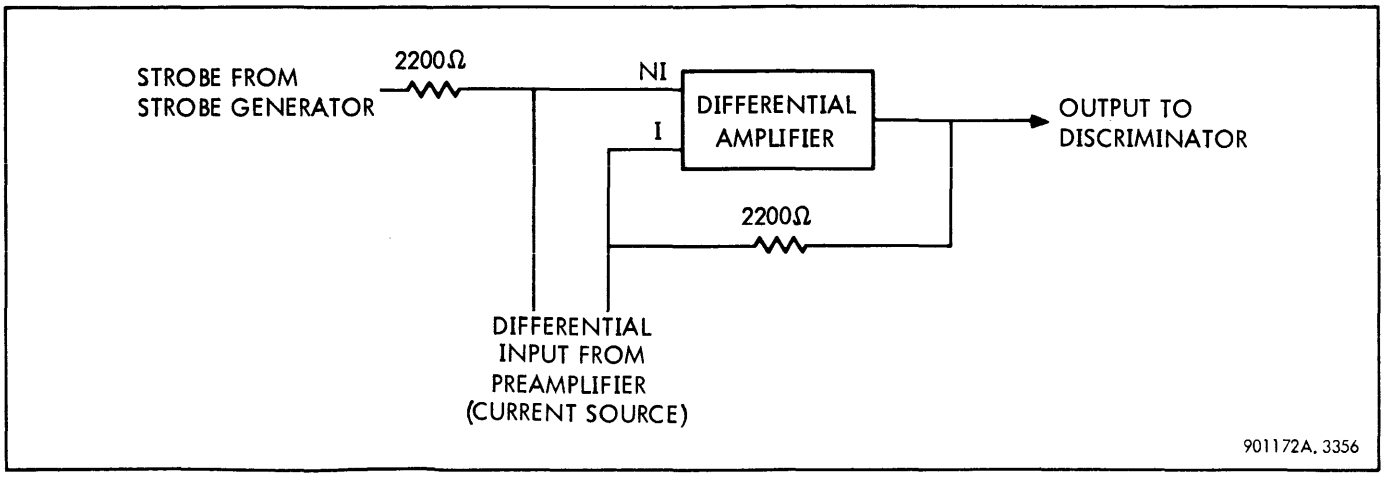


Figure 3-114. Basic Sense Amplifier, Logic Diagram

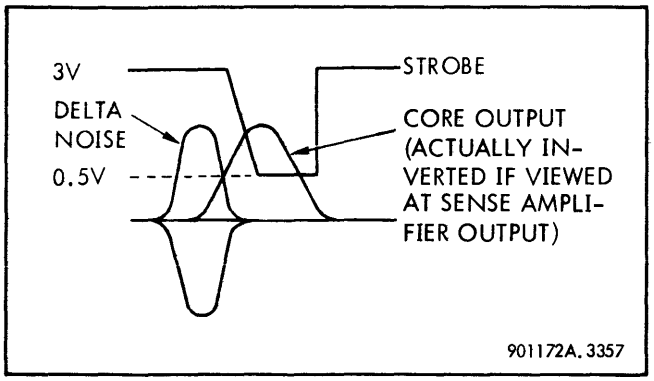


Figure 3-115. Sense Waveforms

Distribution of the Y inhibit circuit outputs for bit 0 to the positive and negative current switches of each 4K memory stack is shown in figure 3-118. This drawing is typical of the distribution of Y inhibits for all bits, 0 through 32.

Timing for the three modes of memory operations – read-restore, clear-write, and partial-write – is shown in figures 3-119 through 3-121.

Figure 3-122 is a module location chart showing the location of all modules required for a 4K, 8K, 12K, or 16K memory.

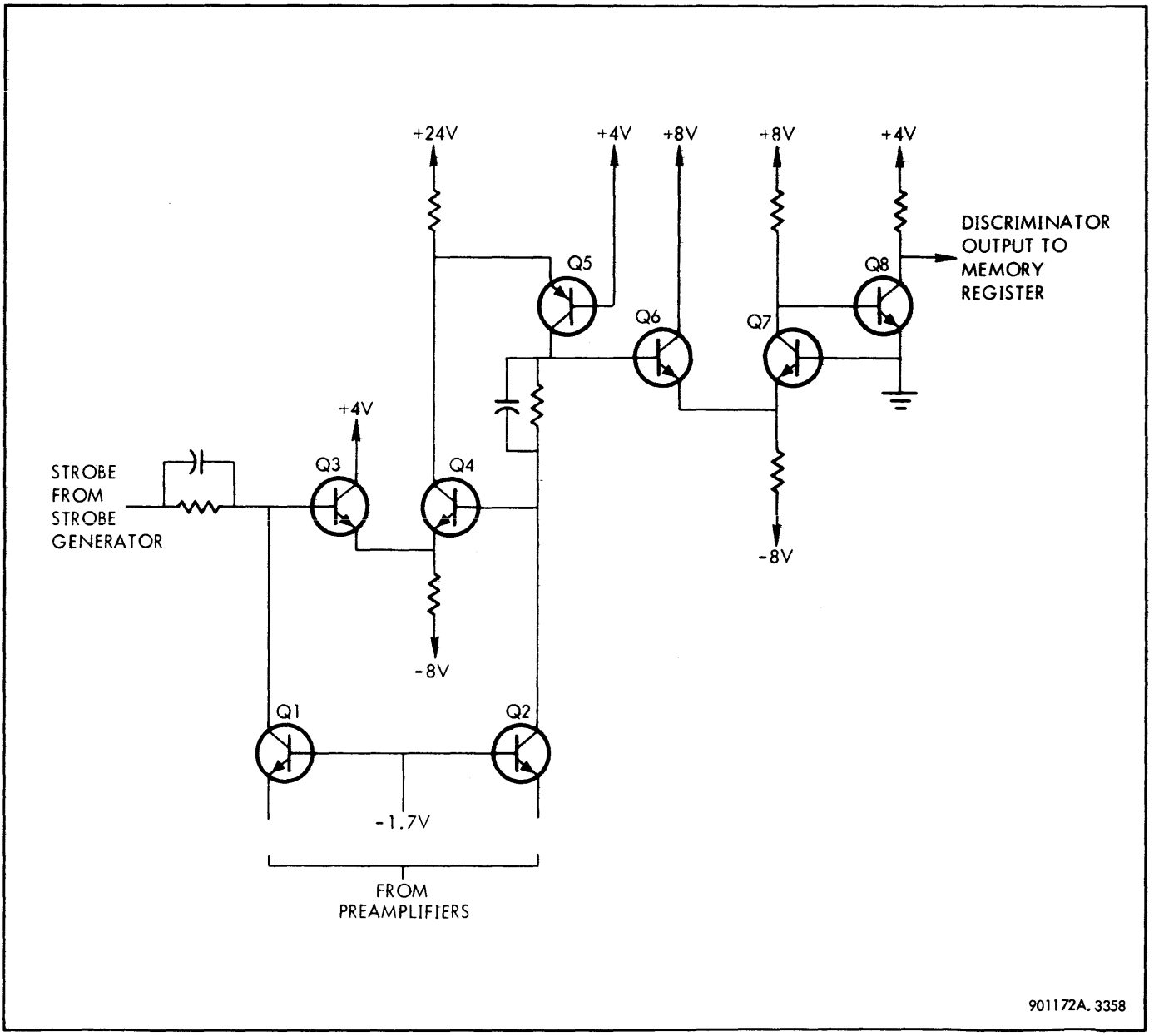


Figure 3-116. Sense Amplifier, Simplified Schematic

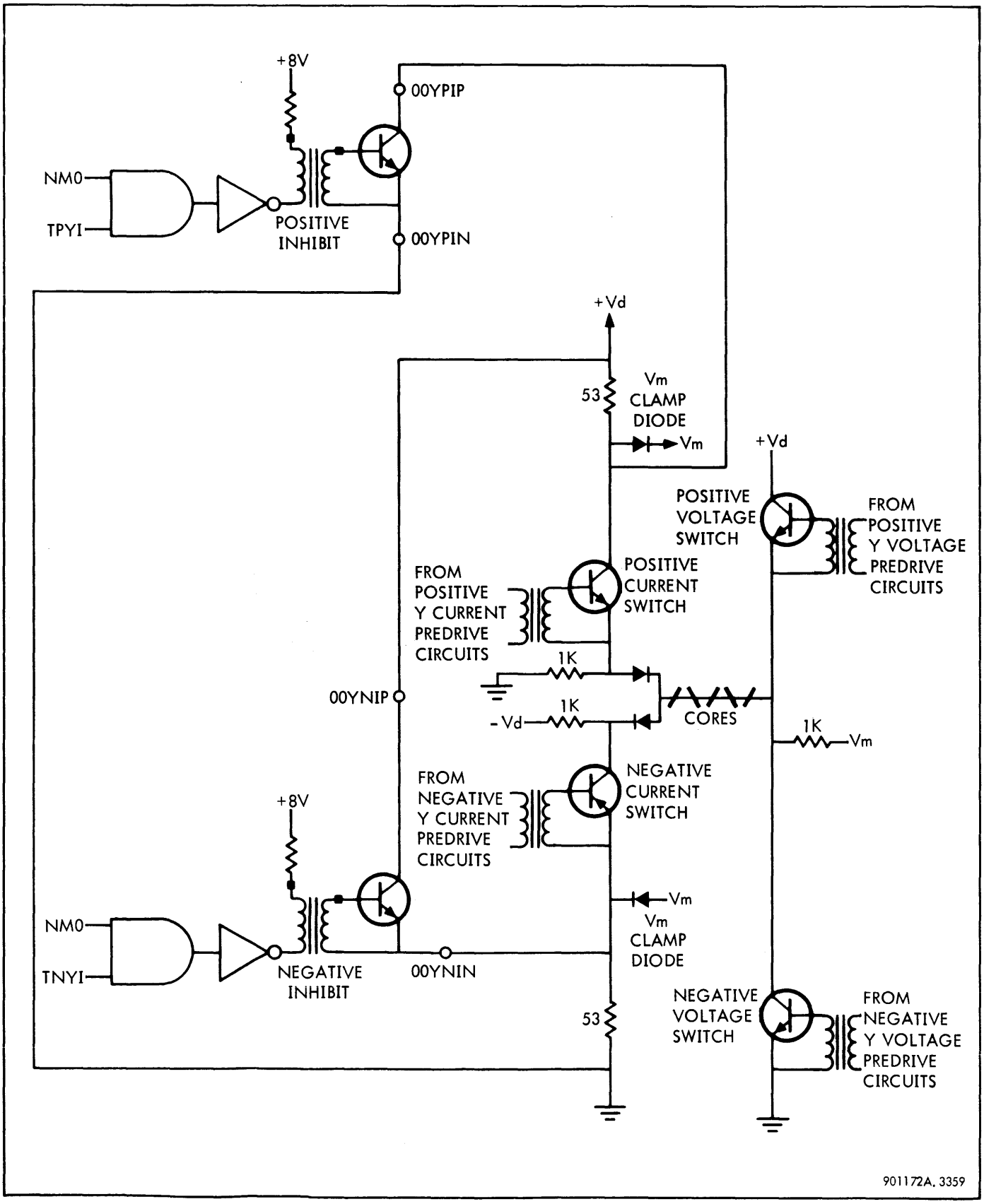


Figure 3-117. Y Current Inhibit Circuits, Simplified Diagram

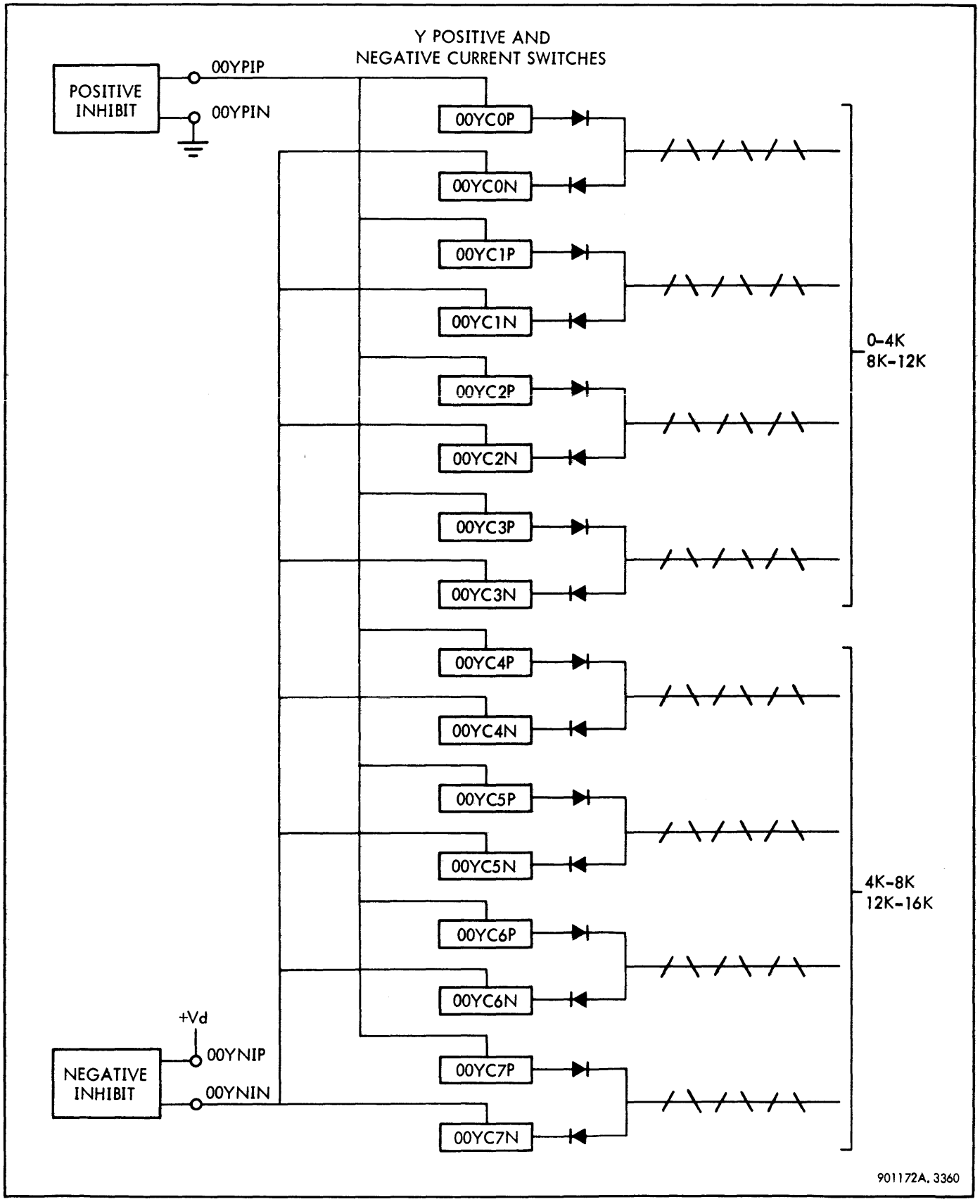


Figure 3-118. Positive and Negative Y Current Inhibit, Bit 0

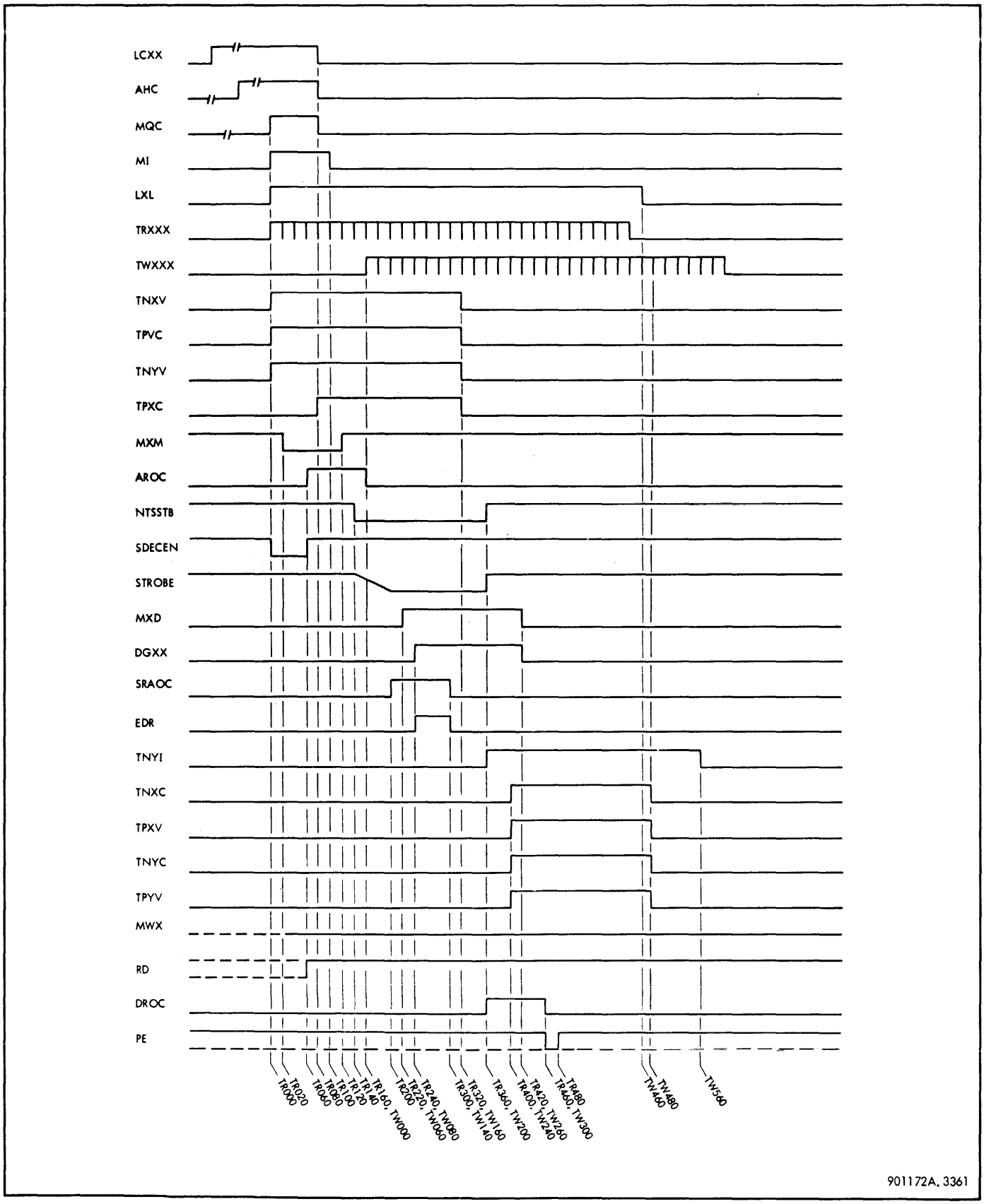


Figure 3-119. Read-Restore, Timing Diagram

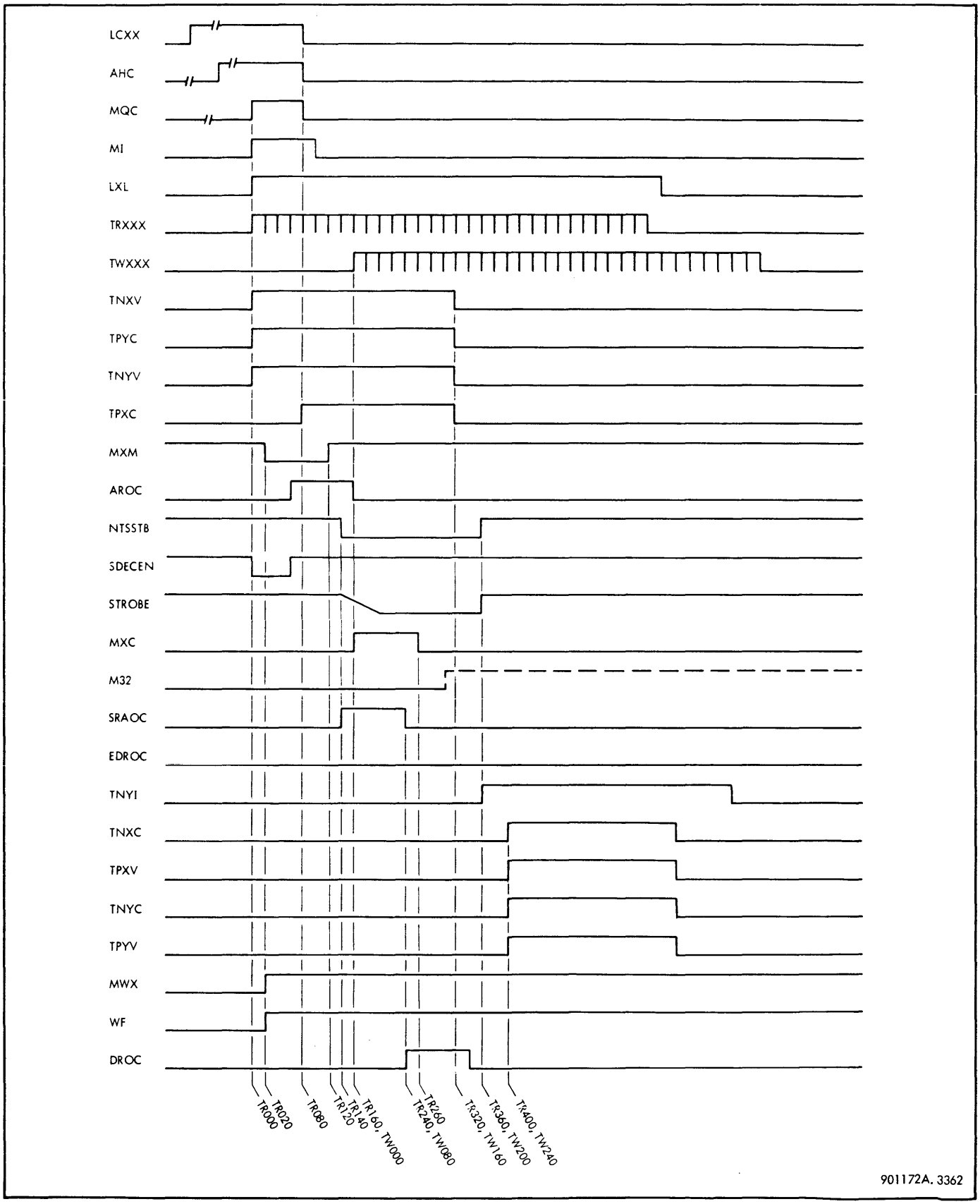


Figure 3-120. Full Clear Write, Timing Diagram

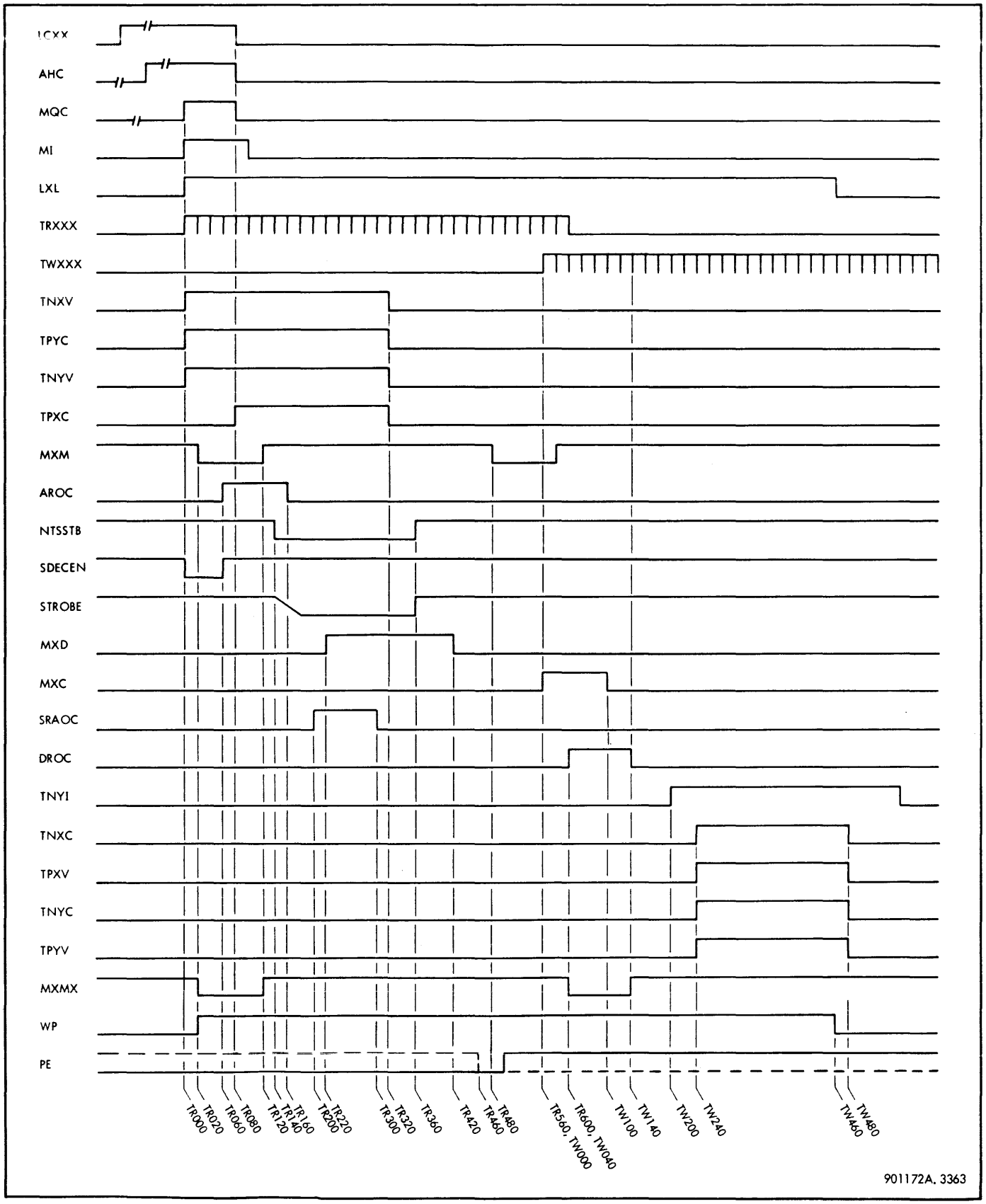


Figure 3-121. Partial Write, Timing Diagram

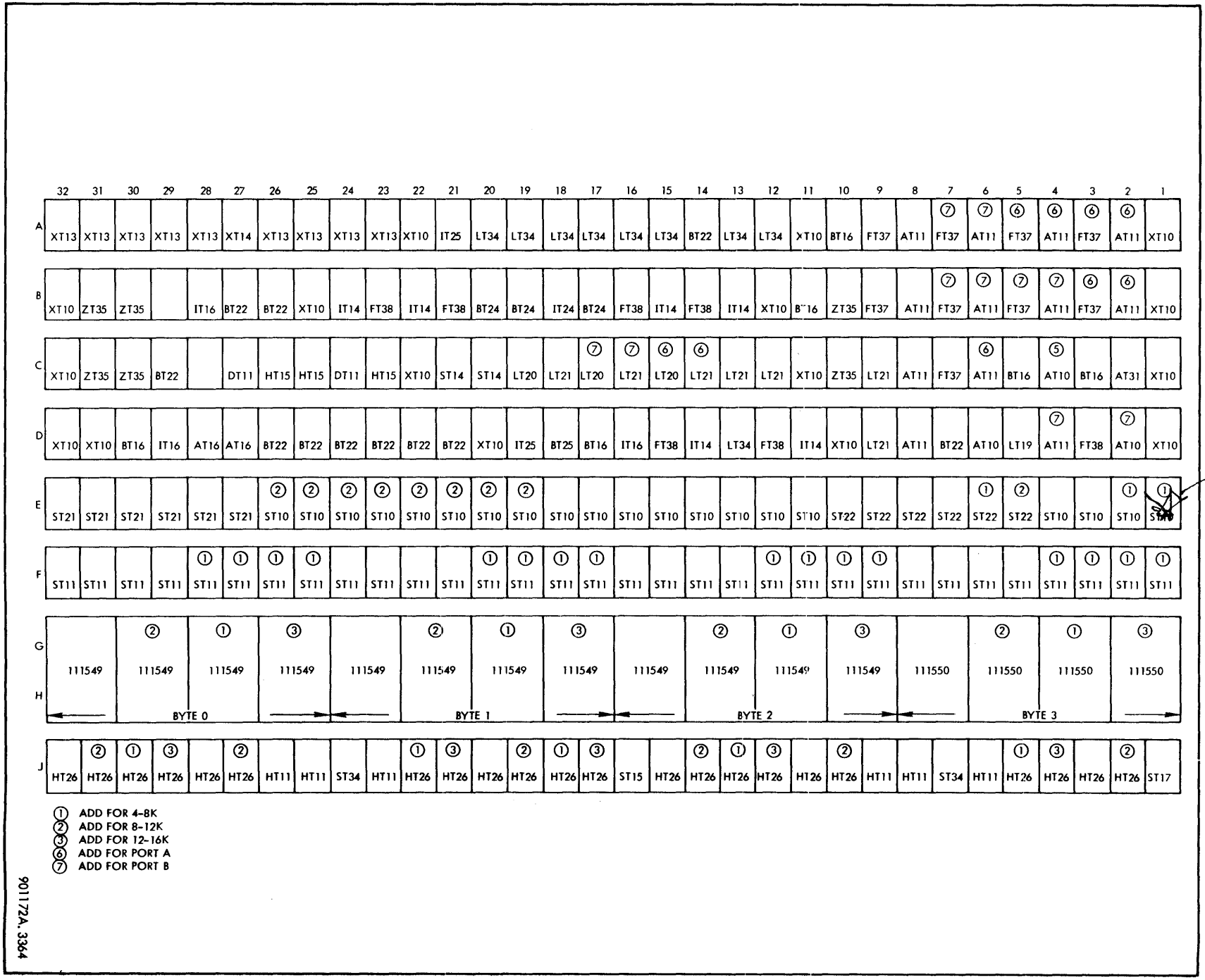


Figure 3-122. Memory Module Location Chart

3-58 OPERATION CODE IMPLEMENTATION

3-59 Preparation Phases

Every instruction performs certain preliminary operations that are common to many other instructions. The clock phases during which these operations are performed are identified as preparation phases PRE1 through PRE4. Each instruction goes through two or more of these preparation phases before entering its individual execution phases.

Preparation for instruction execution is actually started during the last phase of the previous instruction. This phase is identified as PH10, and signal ENDE is true. In phase 10 the next instruction is read from core memory and placed in the C-register and the D-register. The operation code is stored in the O-register and the private memory address in the R field is transferred to the R-register. The contents of the P-register are increased by one to update the current instruction address. If indexing is specified, preset signals are generated to load the index displacement value into the A-register in preparation for phase PRE1. A sequence chart of the operations performed in phase 10 is shown in table 3-18.

Every instruction enters preparation phase PRE1. At the end of this phase the program moves to PRE2, PRE3, or PRE4, depending on whether the instruction is indexed, indirectly addressed, or is an immediate instruction.

A block diagram of the general functions of the preparation phases and their sequence is shown in figure 3-123.

During PRE1 or PRE2, signal PRE/12 is true if the phase is the last one in which effective address computation takes place. This computation includes reading the indirect address from memory if required and, if indexing is specified, adding the index displacement value to the reference address or the indirect address. Indirect addressing always precedes indexing. If the instruction is indirectly addressed or if it is an indexed byte, halfword, or doubleword instruction, phase PRE2 is repeated and PRE/12 is true during the second pass.

During PRE3 or PRE4, signal PRE/34 is true if the phase is the last preparation phase. Signal PRE/34 is true during PRE3 except during byte, halfword (except multiply halfword), or absolute instructions, when the program goes to PRE4. During PRE4, when a zero value in the byte counter indicates that byte or halfword alignment in the D-register is complete, PRE/34 is true and the program goes to PH1.

A sequence chart of all the preparation phases is shown in table 3-19. A flow chart of the preparation phase sequence for immediate instructions is shown in figure 3-124. A detailed flow chart of each phase for nonimmediate instructions is shown in figures 3-125 through 3-129.

⇒ implies
⟶ transfer to
⇢ clock transfer

Table 3-18. Phase 10 (ENDE) Sequence

Phase	Function Performed	Signals Involved	Comments
PH10	One clock long		
DR	Enable signal ENDE	ENDE = PH10 EXC S/EXC = PRE1 NCLEAR	Signal ENDE signifies end of instruction execution. Flip-flop EXC was set at previous PRE1
	(MB0-MB31)⟶(C0-C31)	CXMB = DG = /DG/	Next instruction⟶C-register if MRQ set at previous clock. P-register was incremented during previous ENDE
	(C0-C31)⇢(D0-D31)	DXC = PH10 + ...	Next instruction⇢D-register
	(C1-C7)⇢(O1-O7)	OXC = PH10 + ...	Opcode of next instruction⇢O-register
	(C8-C11)⇢(R28-R31)	RXC = PH10 + ...	R field of next instruction⇢R-register
			Mnemonic: ENDE

(Continued)

Table 3-18. Phase 10 (ENDE) Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH10 DR (Cont.)	Change address in P-register as follows:		
	Increment program address unless one of the following conditions is present: EXU instruction I/O service call Interrupt Halt condition PCP INSTR ADDR switch in HOLD position	PUC31 = N(FUEXU ENDE) PH10 NHALT NIOSC N(INT KR0N) NKAHOLD + ...	Point to new instruction in sequence. P-register holds address of instruction just executed +1 before incrementing
	Decrement program address if instruction is to be repeated	PDC31 = FUEXU (INT + IOSC) ENDE + FUMMC PH10 NMCZ ENDE (INT + IOSC) + ...	Repeat instruction if EXU or MMC was just executed and interrupt or I/O service call is present
	Enable signal (S/SXD)	(S/SXD) = PH10 + ...	Preset adder for D ⟶ S in PRE1
	Set flip-flop LRXD	S/LRXD = OXC + ... R/LRXD = ...	For index operations in PREP phases
	Reset flip-flop NAXRR	S/NAXRR = N(S/AXRR) (S/AXRR) = PH10 + ... R/NAXRR = ...	For index operations in PREP phases
	Branch to PRE1 unless one of the following conditions is present:	PRE1EN = N(S/TRAP) N(S/INTRAP) NIOSC NHALT S/NPRE1 = N(S/PRE1) (S/PRE1 = PRE1EN PH10 + ... R/NPRE1 = ...	
	Trap. If trap, branch to INTRAP phases	(S/INTRAP) = (S/TRAP) + ... (S/TRAP) = FAFL NRW ENDE NTRAP + ENDE AM CC2 OVERIND + ...	Floating-point trap condition Fixed-point arithmetic overflow
			Mnemonic: ENDE

(Continued)

Table 3-18. Phase 10 (ENDE) Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH10 (Cont.)	Interrupt. If interrupt pending, branch to INTRAP phases	(S/INTRAP) = INT IEN + ...	
		IEN = KRUN PH10 NIOSC	Interruptible point in instruction specified by signal IEN
	I/O service call. If I/O service call pending, branch to I/O phases	(S/IOSC) = SC NSCINH	(S/IOSC) indicates that I/O service call is pending. (S/IOEN) enables branch to I/O phases
		(S/IOEN) = IOSC PH10 NIOINH + ...	
	Halt condition. If HALT flip-flop is set, branch to PCP phases	S/HALT = (S/HALT)	
		(S/HALT) = FUWAIT PH1 + INTRAP PRETR N(FAMT + XPSD) + NKRUN PRE1 NFUEXU	HALT flip-flop is set previous to PH10
		BRPCP1 = NFUEXU PH10 NIOSC HALT	
	Clear and reset functions	CLEAR = PH10 + ...	General reset terms for various CPU flip-flops
		RESET/A = CLEAR + ...	
			Mnemonic: ENDE

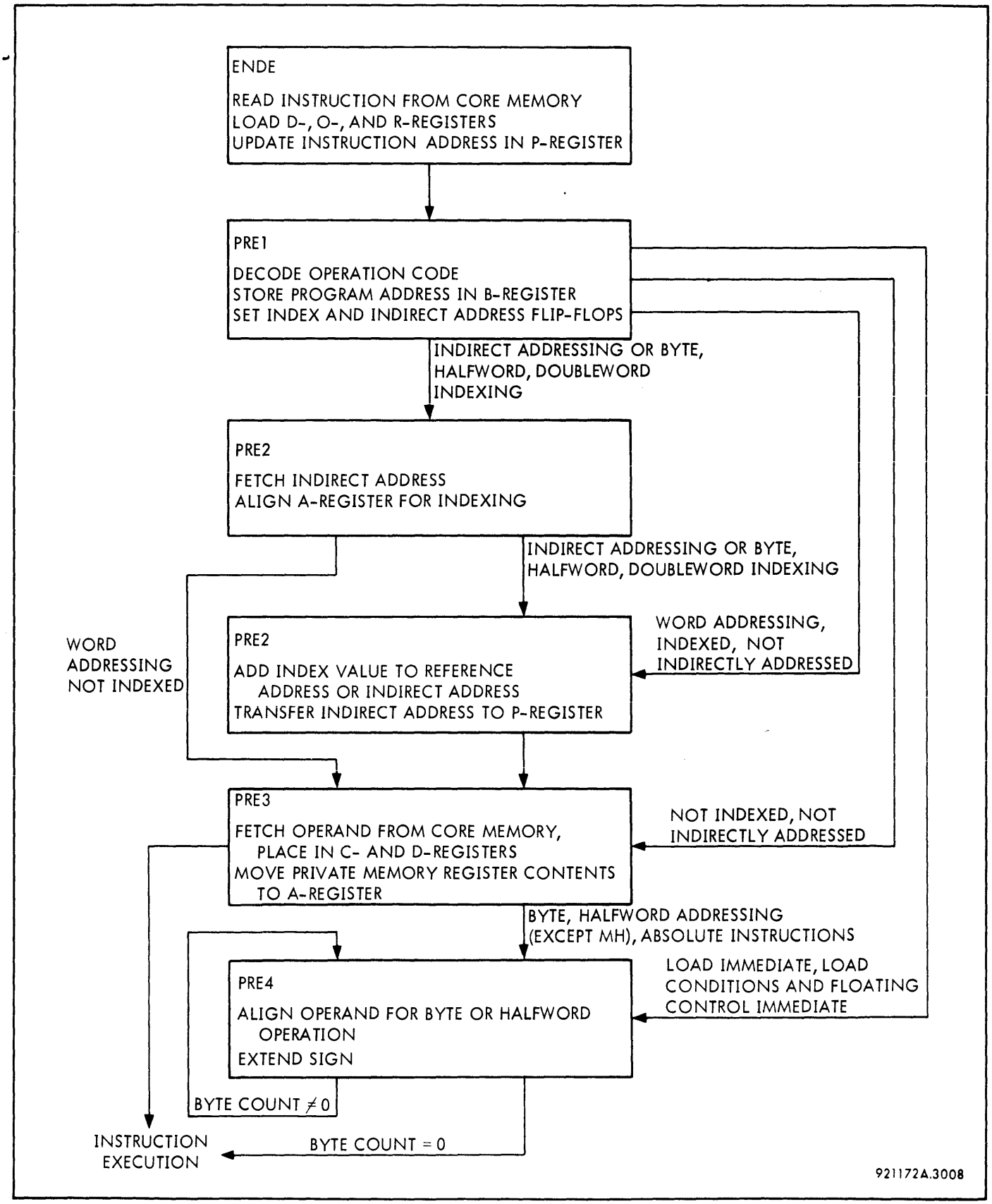


Figure 3-123. Preparation Phases General Functions, Block Diagram

Table 3-19. Preparation Phases Sequence

Phase	Function Performed	Signals Involved	Comments
PRE1	One clock long		
T5L	NANLZ ⟹ (P15-P31)⟶ (B15-B31)	BXP/1 = BRP PRE1 + ...	Move program address from P- to B-register. BRP set during previous instruction execution when B-register contents transferred to P-register
	(D0-D31) ⟶ (S0-S31)	SXD preset during PH10 of previous instruction	D-register contains current instruction
	NFAIM ⟹ (S15-S31)⟶ (P15-P31)	PXS = NFAIM PRE1 + ...	Move operand reference address to P-register if not immediate instruction
	Indexed instruction ⟹ gating signal INDX	INDX = (C3 + C4 + C5) (C12 + C13 + C14)	C-register contains instruction word. Indexing specified by instruction bits 12 through 14
	(D12-D14) ⟶ /LR29/-/LR31/	(LR29-LR31) = D12-D14 LRXD + ... S/LRXD = OXC R/LRXD = ...	If indexing specified, place index register address on private memory address lines
	(RR0-RR31)⟶ (A0-A31)	NAXRR reset during PH10 of previous instruction	Move displacement value in index register to A-register
	INDX ⟹ set flip-flop IX	S/IX = INDX PRE1 R/IX = (R/IX)	
	INDX NFAW ⟹ set index alignment flip-flop IXAL	(R/IX) = PRE/12 + CLEAR S/IXAL = (S/IXAL) NCLEAR (S/IXAL) = INDX PRE1 NFAW + ...	Index value must be aligned if byte, halfword, or doubleword operation
	C0 ⟹ set flip-flop IA	S/IA = C0 PRE1 R/IA = ...	Bit 0 of instruction word specifies indirect addressing
	C0 ⟹ set flip-flops MRQ and DRQ	S/MRQ = (S/MRQ/2) + ... (S/MRQ/2) = C0 PRE1 NFAIM + ... R/MRQ = ... S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = (S/MRQ/2) + ... R/DRQ = ...	Request for core memory cycle Inhibits transmission of another clock until data release is received from memory
			Mnemonic: PREP

(Continued)

Table 3-19. Preparation Phases Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PRE1 T5L (Cont.)	Word operation, indexed, without indirect addressing ⟹ enable signal (S/SXAPD)	(S/SXAPD) = FAW PRE1 NC0 INDX + ...	Preset adder for A plus D ⟶ S in PRE/12 if unaligned indexing only
	Indexing with byte, halfword, or doubleword operation ⟹ enable signal (S/SXA)	(S/SXA) = (S/IXAL) + ...	Preset adder for A ⟶ S in PRE2
	LI, LCFI ⟹ enable signal (S/SXD), sign extension, and branch to PRE4	(S/SXD) = (FULI + FULCFI) PRE1 + ...	Preset adder for D ⟶ S in PRE/12
		SPIM = (FULI + FULCFI) PRE1 + ...	Sign-pad immediate signal
		S/SPW = SPIM D12 R/SPW = ...	Sign-pad ones if D12 in LI instruction is 1. Bit 1 in load immediate is sign bit
		S/SPZ = SPIM ND12 R/SPZ = ...	Sign-pad zeros if D12 in load immediate instruction is 0
		BRPRE4 = (FULI + FULCFI) PRE1 NC0 NANLZ + ...	PRE2 and PRE3 not entered if no indexing, indirect addressing, or memory access
	Indexing or indirect addressing ⟹ branch to PRE2	BRPRE2 = INDX PRE1 + C0 PRE1 + ...	Go to PRE2 to fetch indirect address or align word for indexing
	Enable trap	S/PRETR = PRE1 NANLZ R/PRETR = ...	Permit trap operation in case of nonexistent operation code, unimplemented instruction, or privileged instruction in slave mode
	Set HALT flip-flop if compute switch is moved out of RUN position	S/HALT = (S/HALT) + ... (S/HALT) = NKRUN PRE1 NFUEXU + ... R/HALT = (R/HALT) (R/HALT) = INTRAP1 + NKAS/B PCP2	Halts preparation phases
PRE2 1st Pass DR or T5L	One clock long or sustained until data release IXAL ⟹ (A0-A31) ⟶ (S0-S31)	Adder preset in PRE1	Place index register value in A-register on sum bus
			Mnemonic: PREP

(Continued)

Table 3-19. Preparation Phases Sequence (Cont.)

Phase	Function Performed	Signals Involved		Comments
PRE2 1st Pass DR or T5L (Cont.)	Byte alignment ⟹ 1/4S→A	AXSR2	= IXAL PRE2 FABYTE + ...	Move index displacement value two bit positions right if byte addressing
	Halfword alignment ⟹ 1/2S→A	AXSR1	= IXAL PRE2 FAHW + ...	Move index displacement value one bit position right if halfword addressing
	Doubleword alignment ⟹ 2S→A	AXSL1	= IXAL PRE2 FADW + ...	Move index displacement value one bit position left if doubleword addressing
	Set flip-flop P32 according to byte count in two least significant bits or halfword count in least significant bit of index displacement value in A-register	S/P32	= A30 AXSR2 + A31 IXAL AXSR1 + ...	A31 contains least significant bit of byte count; A30 contains most significant bit. A30 contains halfword count
	Set flip-flop P33 according to byte count	S/P33	= A31 AXSR2	
	(MB0-MB31)→(C0-C31)	CXMB	= DG	Read indirect address from memory into C-register
	(C0-C31)→(D0-D31)	DXC	= IA PRE2 + ...	Move indirect address from C-register into D-register
	IA and not IX ⟹ enable signal (S/SXD)	(S/SXD)	= IA NIX PRE2	Preset adder for D→S in second PRE2
	IA and IX or reference address and IX with alignment ⟹ enable signal (S/SXAPD)	(S/SXAPD)	= IA IX PRE2 + IXAL PRE2 + ...	Preset adder for A plus D→S in second PRE2
	Sustain PRE2 if indirect addressing or index alignment	BRPRE2	= IXAL PRE2 + IA PRE2 + ...	
				Mnemonic: PREP

(Continued)

Table 3-19. Preparation Phases Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PRE2	One clock long		
2nd Pass	IA ⇒ (D0-D31) ⟶ (S0-S31)	Adder logic preset in first PRE2	Place indirect address on sum bus
T5L	IA and IX or index alignment ⇒ (A0-A31) + (D0-D31) ⟶ (S0-S31)	Adder logic preset in first PRE2	Place indirect address plus index displacement or reference address plus index displacement on sum bus
	(S15-S31) ⟶/⟶ (P15-P31)	PXS = PRE2 + ...	Transfer effective address from sum bus into P-register
PRE/12 (PRE1 or PRE2)	PRE/12 is not a phase flip-flop; it is the output of a gate whose simplified equation is as follows:		
	PRE/12 = PRE1 NINDX NC0 + PRE2 NIA NIXAL		
	The signal PRE/12 is therefore true during either PRE1 or PRE2 when the phase represents the end of address modification. When either PRE1 or PRE2 is equivalent to PRE/12, the next phase entered is PRE3		
	The following operations take place during either PRE1 or PRE2 if PRE/12 is true during the phase:		
	Reset flip-flop IX	R/IX = (R/IX) (R/IX) = PRE/12 + CLEAR	Reset index flip-flop
	Set flip-flop MRQ	S/MRQ = (S/MRQ/2) + (S/MRQ/1) + ... (S/MRQ/2) = PREOPER PRE/12 + ...	Request for core memory cycle. PREOPER is true for instructions that require reading contents of effective address
		(S/MRQ/1) = FABRANCH PRE/12 NANLZ R/MRQ = ...	No data release requested by this memory request during a branch instruction
	Set flip-flop DRQ	S/DRQ = (S/DRQ) (S/DRQ) = (S/MRQ/2) + (S/DRQ/2) + ... (S/DRQ/2) = OU6 OL7 PRE/12 + ... R/DRQ = ...	Inhibits transmission of another clock until data release is received from memory. Execute instruction is in FABRANCH and therefore requires special setting equation for data request. DRQ is set in PH1 after (S/MRQ (S/MRQ/1)
			Mnemonic: PREP

(Continued)

Table 3-19. Preparation Phases Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PRE/ 12 (PRE1 or PRE2) (Cont.)	Reset flip-flop NAXRR	S/NAXRR = N(S/AXRR) (S/AXRR) = PRE/12 + ... R/NAXRR = ...	Preset for transfer of private memory R contents into A-register in PRE3
	Merge one ⟶ LR lines	S/NLR31F = N(S/LR31) (S/LR31) = PRE/12 (S/LR31/12) + ... (S/LR31/12) = (S/LR31/1) + FAMULNH + FASEL + ... (S/LR31/1) = OU1 OL0 + OU1 OL1 + OU1 OL5 + OU1 OL8 + OU3 OL9 + OU2 OL5 R/NLR31F = ...	Set odd-numbered address in private memory address lines
	FADW and not LAD or floating-point ⟹ merge one into LB lines	/LR31/ = P31 (S/P31/1) = FADW PRE/12 N(O4 O5) (NOU1 OLB) + ...	Set odd-numbered address in core memory address lines
	No indexing ⟹ (D0-D31) ⟶ (S0-S31)	(S/SXD) preset in PRE2 first pass or PH10 of previous instruction	Place indirect address or reference address on sum bus
	Indexing ⟹ (A0-A31) + (D0-D31) ⟶ (S0-S31)	(S/SXAPD) preset in PRE1 or first PRE2	Place reference address plus index displacement on sum bus
	(S0-S31) ⟶/ (P0-P31)	PXS = NFAIM PRE1 + PRE2 + ...	Place effective address in P-register if PRE2 or if PRE1 and not immediate instruction
	FAIO ⟹ (S0-S31) ⟶/ (D0-D31)	DXS = FAIO PRE/12 + ...	Effective I/O address ⟶ D-register
	FADW ⟹ force zero into S31	S31INH = FADW PRE/12	Inhibit least significant bit of address to place even-numbered doubleword address in P-register. This addresses most significant half of doubleword
	Reset flip-flop NT8L	S/NT8L = N(S/T8L) (S/T8L) = PRE/12 + ... R/NT8L = + ...	Set T8L clock for PRE3
			Mnemonic: PREP

(Continued)

Table 3-19. Preparation Phases Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PRE/ 12 (PRE1 or PRE2) (Cont.)	Enable signal (S/SXB) if not immediate instruction Reset flip-flop NPRE3	(S/SXB) = PRE/12 NFAIM + ... S/NPRE3 = N(S/PRE3) (S/PRE3) = PRE/12 NBR R/NPRE3 = ...	Preset adder for B⟶S in PRE3 Go to preparation phase PRE3
PRE3 T8L or DR	(B0-B31)⟶(S0-S31) (S15-S31)⟶/⟶(P15-P31)	(S/SXB) preset in PRE/12 PXS = FAS10 NFAIM PRE3 NANLZ + FUWAIT PRE3 NANLZ + FASEL PRE3 NOL7 NANLZ + FUEXU PRE3 NANLZ + FARWD PRE3 NANLZ + FAMDS NFAIM PRE3 NANLZ + ...	Return program address to P-register from B-register
	Not word operation ⟹ load byte counter	S/BC0 = NP32 PRE3 NPRE/34 O1 + ... S/BC1 = NP33 PRE3 NPRE/34 OU7 + ...	Byte counter BC0 and BC1 are interpreted as follows: BC0 BC1 Byte 0 0 0 Byte 1 0 1 Byte 2 1 0 Byte 3 1 1 Halfword 0 0 0 Halfword 1 1 0
			Mnemonic: PREP

(Continued)

Table 3-19. Preparation Phases Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PRE3 T8L or DR (Cont.)	Not word operation ⟹ set sign extention		
	Set flip-flop SPW	S/SPW = SPIM D12 + FAHW C16 P32 PRE3 SPIM + FAIM PRE3 + ... R/SPW = ...	Sign pad ones if sign is 1. In an immediate instruction (SPIM) bit 12 contains the sign. In a halfword instruction (FAHW) bit 16 contains the sign. P32 indicates halfword 1
	Set flip-flop SPZ	S/SPZ = SPIM ND12 + FAHW NC16 P32 PRE3 + FABYTE P32 P33 PRE3 + ... R/SPZ = ...	Sign-pad zeros if sign is 0. In byte operation this refers only to byte 3 (P32 and P33), when bits 0 through 23 of the contents of register R are to be cleared
	Enable signal (S/SXD)	(S/SXD) = PRE3 BRPRE4 + ...	Preset adder for D⟶S in PRE4
	Set flip-flop PRE4	S/PRE4 = BRPRE4 NCLEAR R/PRE4 = ... BRPRE4 = PRE3 NPRE/34 NANLZ	Branch to PRE4. PRE/34 is true if multiply halfword or word or doubleword operation
	FUMI and FADIVH ⟹ reset flip-flop NCXS	S/NCXS = N(S/CXS) (S/CXS) = FUMI PRE3 + FADIVH BREPRE4 R/NCSX = ...	Preset for transfer of S⟶C in PRE4
	PREOPER ⟹ (MB0-MB31) ⟶ (C0-C31)	CXMB = DG = /DG/	Read contents of effective address, or effective address u 1 if doubleword operation, from core memory into D-register via C-register
	(C0-C31) ⟶ (D0-D31)	DXC = PREOPER PRE3 + ...	
	(RR0-RR31) ⟶ (A0-A31)	AXRR set at PRE/12 clock S/A_n = RR_n AXRR NAXRRINH + ...	Read contents of private memory register R into A-register. Inhibit AXRR for given instructions
			Mnemonic: PREP

(Continued)

Table 3-19. Preparation Phases Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PRE3 T8L or DR (Cont.)		AXRRINH = FASTORE PRE3 OL4 + FAPSD PRE3 + FAST/M PRE3 + FARWD OLC PRE3 + ...	
	STCF ⟹ CF⟶(A24-A31)	AXFC = FASTORE PRE3 OL4	Move contents of condition code and floating control flip-flops to A-register if store conditions and floating control instruction
	XPSD ⟹ PSW1⟶(A0-A31)	AXPSW1 = FAPSD PRE3 + ...	Move contents of program status doubleword 1 into A-register for exchange program status doubleword instruction
	PSM, PLM, LM, STM⟹ CC⟶(A28-A31) CC⟶(MC4-MC7)	AXCC = FAST/M O6 (PRE3 + ...)	AXCC places condition code in A-register and macro-counter during specified instructions. Condition code contains number of words
	PSW, PLW ⟹ 1⟶MC	S/MC7 = FAST/M PRE3 NO6	Set one into macro-counter for push word and pull word instructions
	FAIO ⟹ 0⟶LR lines	LRXZ = FUSIO PRE3	Address private memory register 0 to get command doubleword address
	P-1⟶P	PDC31 = FADW/1 PRE3 + FAST PRE3 + FACOMP PRE3 OU1	Increment P-register to get address of most significant word of doubleword
	FULAD ⟹ set flip-flop MRQ	S/MRQ = (S/MRQ/2) + ... (S/MRQ/2) = FALOAD/A OU1 PRE3 + ... R/MRQ = ...	Core memory request if load absolute doubleword instruction
	Set flip-flop DRQ	(S/DRQ) = (S/MRQ/2) + ... R/DRQ = ...	Inhibits transmission of another clock until data release is received from memory
			Mnemonic: PREP

(Continued)

Table 3-19. Preparation Phases Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PRE3 T8L or DR (Cont.)	P + 1 → P Branch to execution phase if PRE/34 Go to PRE4 if not PRE/34	PUC31 = FULAD PRE3 + ... See PRE/34 BRPRE4 = PRE3 NPRE/34 NANLZ	Increment P-register to get address of least significant word
PRE4 T5L	One clock long Set sign extension Enable signal (S/SXD) Set flip-flop SPW Set flip-flop SPZ (D0-D31) → (S0-S31) (S0-S31) → (D0-D31) (S0-S31) → (C0-C31) FULAD ⟹ (MB0-MB31) → (C0-C31)	(S/SXD) = PRE4 (BC=1) + ... S/SPW = (S/SPW) + ... (S/SPW) = FAHW D8 (BC=1) PRE4 + ... R/SPW = ... S/SPZ = (S/SPZ) + ... (S/SPZ) = FAHW ND8 (BC = 1) PRE4 + FABYTE (BC = 1) PRE4 + ... R/SPZ = ... (S/SXD) set in PRE3 or previous PRE4 DXS = BCZ PRE4 NFULAD + ... NCXS reset in PRE3 CXMB = DG	Preset adder for D → S in next PRE4 Sign-pad ones if sign is 1. Flip-flop D8 contains sign when halfword 0 has been moved right eight positions (BC = 1) Sign-pad zeros if sign is 0 for halfword operation. Clears bits 0 through 23 for load byte instruction Propagates are inhibited (SPZ) or enabled (SPZ) or enabled (SPW) upward from sign position. Sign extended value placed in C-register as well as D-register for multiply immediate and divide halfword, and for divide word with even R field Read least significant word of doubleword from memory into C-register
			Mnemonic: PREP

(Continued)

Table 3-19. Preparation Phases Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PRE4	(C0-C31)—/→(D0-D31)	DXC = FULAD PRE4 + ...	Load least significant word into D-register
T5L (Cont.)	P-1—/→P	PDC31 = FULAD PRE4 + ...	Decrement P-register to get address of most significant word
	Load ⟹ down align D-register	DXDR8 = NBCZ PRE4 + ...	For load operation, align byte or halfword to right end of D-register by shifting D-register right one byte at a time until byte counter = 0
	Store ⟹ left align A-register	AXAL8 = FASTORE NBCZ PRE4 + FAMT SW2 NBCZ PRE4 + ...	For store operation or for modify and test instructions when R field is nonzero, align A-register left one byte at a time until information is in proper position for effective location (byte count = 0)
	BC ≠ 0 ⟹ decrement byte counter	BCDC1 = NBCZ PRE4 + ...	Subtract one from byte count during each PRE4 until BC=0, NBCZ false
	Compare byte ⟹ 0—/→ (A0-A23)	AXZ/012 = FACOMP/1 OU7 PRE4	Clear lower bit positions of aligned effective byte for comparison with contents of register R
	BC ≠ 0 ⟹ sustain PRE4	BRPRE4 = PRE4 NBCZ + ...	Repeat PRE4 until byte count = 0 indicates that alignment is complete
			Mnemonic: PREP

(Continued)

Table 3-19. Preparation Phases Sequence (Cont.)

Phase	Function Performed	Signals Involved		Comments
PRE/34	PRE/34 is not a phase flip-flop; it is the output of a gate whose simplified equation is as follows: PRE/34 = PRE3 FAWORDW NFALOAD/A + PRE3 FUMH + PRE4 NBC1 NBC0 NANLZ The signal PRE/34 is true during the last preparation phase when register alignment is taking place. Signal PRE/34 is true during PRE3 when a branch to an execution phase is to be made, and during PRE4 when the byte count is zero The following operations take place during either PRE3 or PRE4 if PRE/34 is true during the phase:			
	Enable signal (S/AXAPD)	(S/SXAPD)	= FAADD (PRE/34 + ...)	Preset adder for A plus D⟶S to add two operands
	Enable signal (S/SXAMD)	(S/SXAMD)	= FASUB (PRE/34 + ...)	Preset adder for A minus D⟶S to subtract one operand from another
	Enable signal (SXDMA)	(S/SXDMA)	= FUPLM (PRE3 + ...)	Preset adder for D minus A⟶S to check for word count underflow in stack pointer doubleword during pull multiple instruction
	Enable signal (S/SXA)	(S/SXA)	= FASTORE PRE/34 + FAMT (PRE/34 + ...) + FUMMC PRE3 + FUS PRE3 + FAMUL (PRE/34 + ...) + FARWD NRZ (PRE/34 + ...) + ...	Preset adder for A⟶S in first execution phase
				Mnemonic: PREP

(Continued)

Table 3-19. Preparation Phases Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PRE/ 34 (Cont.)	Enable signal (S/SXD)	(S/SXD) = FUINT PRE3 + FALCFP PRE/34 + FALOAD (PRE/34 +...) + FUXW PRE3 + ...	Preset adder for D ⟶ S in first execution phase
	Enable signal (S/SXDM1)	(S/SXDM1) = FUPLW (PRE3 + ...)	Preset adder for D minus 1⟶S to check for word count underflow in stack pointer doubleword during pull word instruction
	Enable signal (S/SXDP1)	(S/SXDP1) = FUPSW (PRE3 + ...)	Preset adder for D plus 1⟶S to check for word count overflow during push word instruction
	Enable signal (S/SXAP1)	(S/SXAP1) = FUBIR PRE3 + ...	Preset adder for A plus 1⟶S to increment contents of register during branch on incrementing register instruction
	Enable signal (S/SXAM1)	(S/SXAM1) = FUBDR PRE3 + ...	Preset adder for A minus 1⟶S to decrement register contents during branch on decrementing register instruction
	Enable signal (S/SXMD)	(S/SXMD) = FALOAD/C (PRE/34 + ...)	Preset adder for minus D⟶S to load two's complement of effective word into private memory during load complement instructions
	Set flip-flop NPRXAD	S/NPRXAD = N(S/PRXAD) N(S/SXAMD/1) (S/PRXAD) = FASEL PRE3 NOL7 + OU4 OLB PRE3 + ... R/NPRXAD = ...	Preset for A AND D ⟶S during AND word and compare and load selective instructions
			Mnemonic: PREP

(Continued)

Table 3-19. Preparation Phases Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PRE/ 34 (Cont.)	Reset flip-flop NPRXNAD	S/NPRXNAD = N(S/PRXNAD) N(S/SXAPD/1) (S/PRXNAD) = FASEL PRE3 OL7 + ... R/NPRXNAD = ...	Preset adder for NA AND D⟶S to AND complemented mask with effective word during store selective instruction
	Enable signal (S/SXAEORD)	(S/SXAEORD) = OU4 OL8 PRE3	Preset adder for A ⊕ D for exclusive OR operation in exclusive OR word instruction
	Enable signal (S/SXAORD)	(S/SXAORD) = OU4 OL9 PRE3 + ...	Preset adder for A OR D⟶S for OR operation during OR word instruction
	Set flip-flop MRQ	S/MRQ = (S/MRQ/1) + (S/MRQ/2) + (S/MRQ/3) + ...	Core memory request, inhibited in PRE3 if analyze instruction
	Set memory request	(S/MRQ/1) = FUINT PRE3 + FUWAIT PRE3 + FAS10 PRE/34 + ...	No data release requested. Fetches next instruction during short instructions that do not require use of C-register
	Set data release memory request	(S/MRQ/2) = FAPSD (PRE/34 + ...) + FAST/M PRE3 NOU0 OLA R/MRQ = ...	Fetches next PSW1 in program status doubleword instructions and first word to be loaded in load multiple instruction
	Set flip-flop DRQ	S/DRQ = (S/DRQ) + NCLEAR (S/DRQ) = (S/MRQ/2) + ... R/DRQ = ...	Inhibits transmission of another clock until data release is received from memory
	Set delayed data release memory request	(S/MRQ/3) = [(FADW/1 (PRE/34 + ...) + FACOMP/L OU1 PRE3)] NANLZ + ...	Fetches effective word for some doubleword instructions (FADW/1) and compare with limits in memory instruction
			Mnemonic: PREP

(Continued)

Table 3-19. Preparation Phases Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PRE/34 (Cont.)	Reset flip-flop NMRQP1	S/NMRQP1 = N(S/MRQ/3) R/NMRQP1 = ...	Delays setting data request flip-flop until next phase
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = MRQP1 + ... R/DRQ = ...	Data request
	Set flip-flop MBXS	S/MBXS = (S/MBXS) (S/MBXS) = FASTORE PRE/34 + FAMT SW2 PRE/34 + FAPSD PRE3 O7 + ... R/MBXS = ...	Presets for S ⟶ MB in first execution phase to place on memory bus information to be stored in core memory
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = (S/MBXS) + ...	Data request
	Enable signal (S/SXB)	(S/SXB) = FUBAL PRE3 + FALOAD/A (PRE/34 + ...) + ...	Preset adder for B⟶S in first execution phase to place program address in private memory during branch and link instruction and to place program address in P-register during load absolute halfword and word instructions
	Reset flip-flop NCXS	S/NCXS = N(S/CXS) (S/CXS) = FAST PRE3 + FASEL PRE3 + ... R/NCXS = ...	Preset adder for S⟶C in first execution phase
	Reset flip-flop NAXRR	S/NAXRR = N(S/AXRR) (S/AXRR) = PRE3 (FUDW NR31) + ... + FASTORE PRE/34 NO2 + (FAST/M PRE3 NOU0) NOLA + FUMMC PRE3 + ... R/NAXRR = ...	Preset logic for transfer of contents of private memory register contents into A-register
	Merge 1⟶LR31	(S/LR31) = FADW/1 PRE3 NANLZ + FUMMC PRE3 NANLZ	Address odd-numbered private memory register
			Mnemonic: PREP

(Continued)

Table 3-19. Preparation Phases Sequence (Cont.)

Phase	Function Performed	Signals Involved		Comments
PRE/ 34 (Cont.)	Set flip-flop RW	S/RW	= FUXW PRE3 NANLZ + FAS11 NOL1 (PRE/34 + ...) + FUBAL PRE3 NANLZ + FUBDR PRE3 NANLZ + FUBIR PRE3 NANLZ + ...	Prepare to write into private memory
	FAST ⟹ set flip-flop SW8	S/SW8	= BRSW8 NRESET/A + ...	SW8 identifies PH1 as PH1/A in stack and multiple instructions
		BRSW8	= FAST PRE3 + ...	
	FUMSP ⟹ set flip-flop SW7	S/SW7	= (S/SW7)	SW7 indicates first pass through phases in stack and multiple instructions and sign in load absolute instruction
		(S/SW7)	= FAST PRE3 NO4 + FULAWORDW ND0 PRE/34 + FALOAD/A OU5 ND16 PRE/34 + ...	
	STM ⟹ P - 1 → P	PDC31	= (FAST/M PRE3 NOU0) NOLA + ...	Obtain address of first core memory location in sequential set during store multiple instruction
	LM ⟹ R - 1 → R	RDC31	= (FAST/M PRE3 NOU0) OLA + ...	Obtain address of first private memory register in sequential set for load multiple instruction
	Reset flip-flop NT11L	S/NT11L	= N(S/T11L)	Set clock T11L for PH1
		(S/T11L)	= FACOMP/1 (PRE/34 + ...) + FAST PRE3 + (ANLZ PRE3) + FACOMP/1 (PRE/34 + ...)	
				Mnemonic: PREP

(Continued)

Table 3-19. Preparation Phases Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PRE/ 34 (Cont.)	Set flip-flop PH3	S/PH3 = BRPH3 NCLEAR + ...	Branch to phase 3 for multiply and divide instructions
		BRPH3 = FAMDS PRE/34 NBRPH5 NANLZ + FAPSD PRE3 NANLZ NANLZ NO7	
		R/PH3 = ...	
	Set flip-flop PH5	S/PH5 = BRPH5 NCLEAR + ...	Branch to phase 5 for analyze and shift instructions
		BRPH5 = ANLZ PRE3 + FUS PRE3 + FUSF PRE3 ND23 + ...	
		R/PH5 = ...	
	Set flip-flop PH6	S/PH6 = (BRPH6 NIOEN) NCLEAR	Branch to phase 6 for if store multiple or modify stack counter instruction
		BRPH6 = FAST/M PRE3 NOU0 NANLZ	
		R/PH6 = ...	
	Set flip-flop PH8	S/PH8 = BRPH8 NCLEAR + ...	Branch to phase 8 if modify and test instruction
		BRPH8 = FAMT SW2 PRE/34 + ...	
		R/PH8 = ...	
	Set flip-flop PH10	S/PH10 = BRPH10 NCLEAR + ...	Branch to phase 10 if execute instruction
		BRPH10 = FUEXU PRE3 NANLZ + ...	
		R/PH10 = ...	
			Mnemonic: PREP

INDIRECT ADDRESSING. When bit position 0 of the instruction word contains a one, indirect addressing is to be performed during the preparation phases. Indirect address flip-flop IA is set during PRE1, the indirect address is read from the contents of the reference location in the first PRE2 and placed in the C- and D-registers. If no indexing is required, this address is the effective address and is transferred to the P-register in the second pass through PRE2. If indexing is specified, the index displacement value is added to the indirect address in the second PRE2, and the resulting effective address is transferred to the P-register. The effective address in the P-register is used to read the operand from memory in PRE3.

The logic used to implement the indirect addressing feature is shown in table 3-19.

INDEXING. Indexing is done in the Sigma 5 computer by adding a displacement value in one of seven index registers to the reference address in the instruction word, or to the indirect address from core memory if indirect addressing is specified. Registers 1 through 7 in the first private memory block are used as index registers. Register 0 is not used for indexing. A nonzero value in the X field causes the contents of the specified index register to be transferred to the A-register for addition to the reference or indirect address in the D-register.

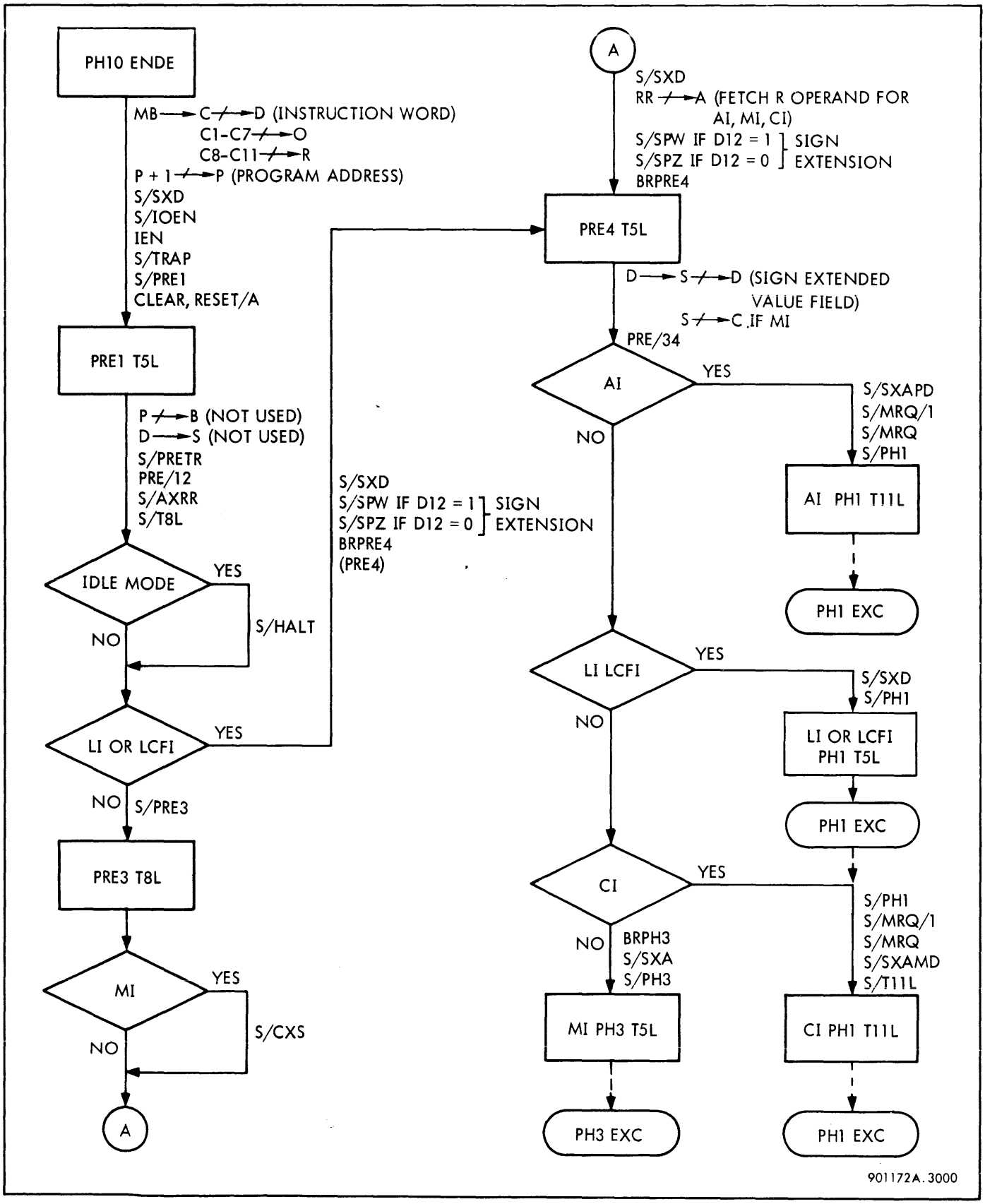


Figure 3-124. Immediate Instruction Preparation Phases, Flow Diagram

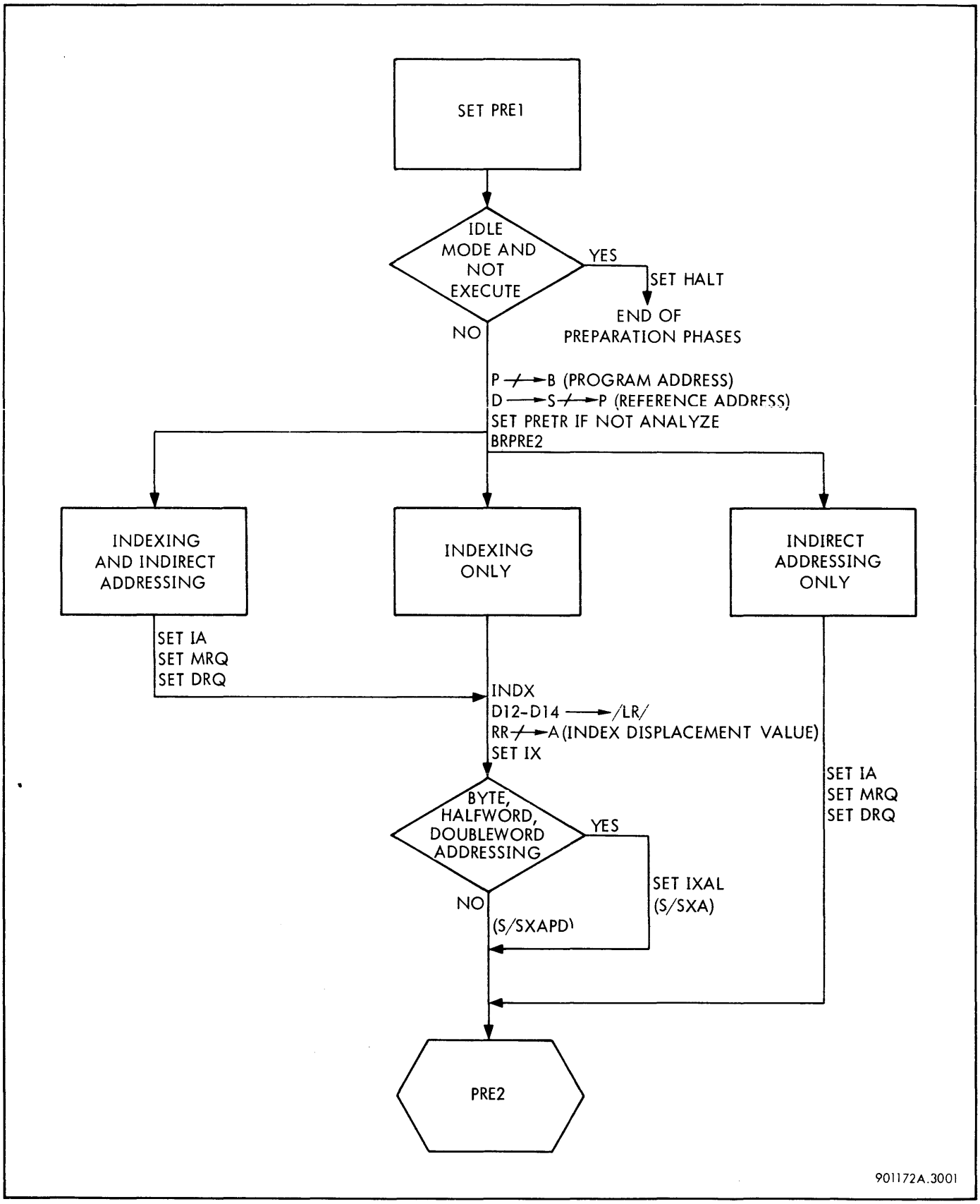


Figure 3-125. Preparation Phase PRE1, Flow Diagram

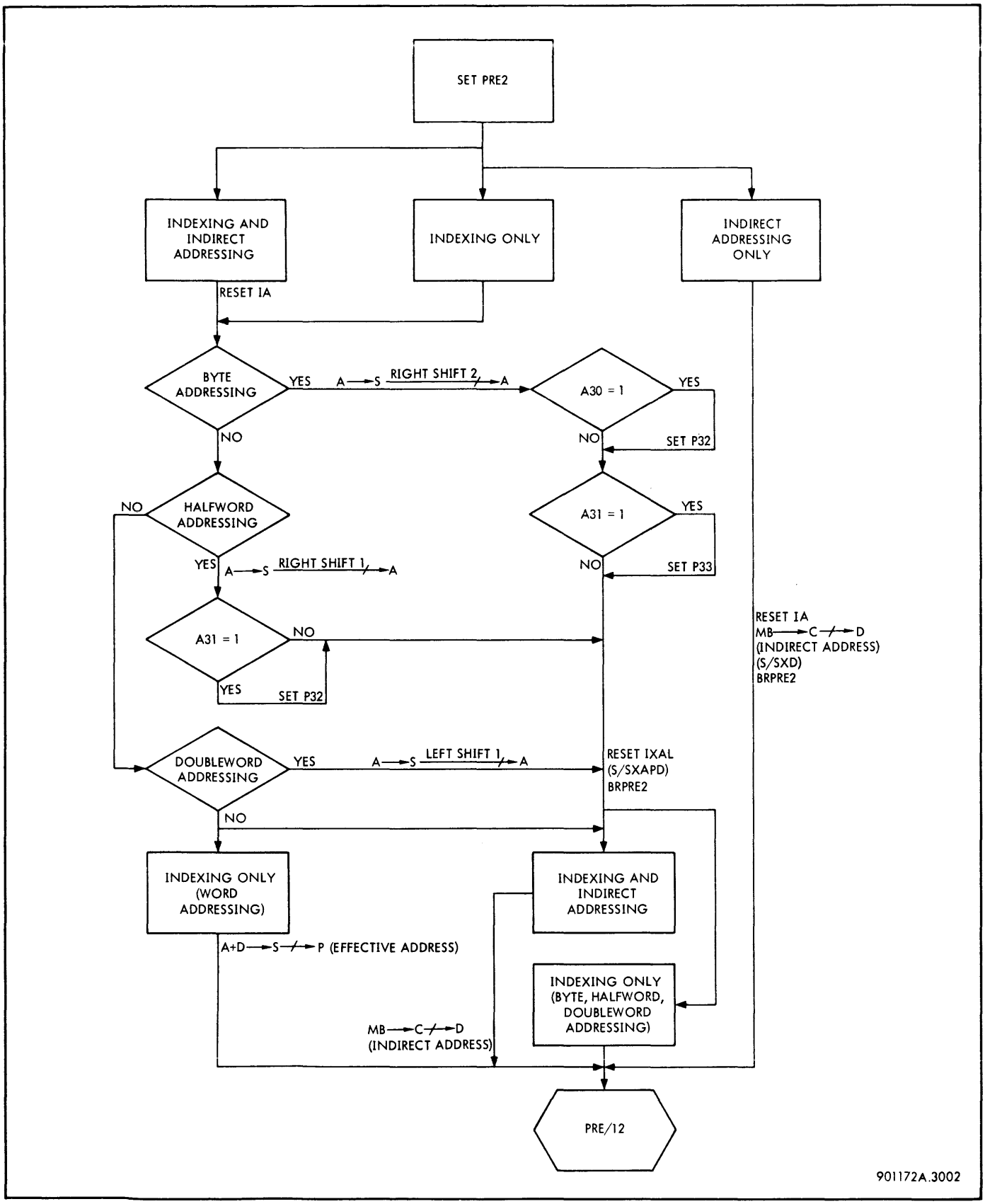


Figure 3-126. Preparation Phase PRE2 (Not PRE/12), Flow Diagram

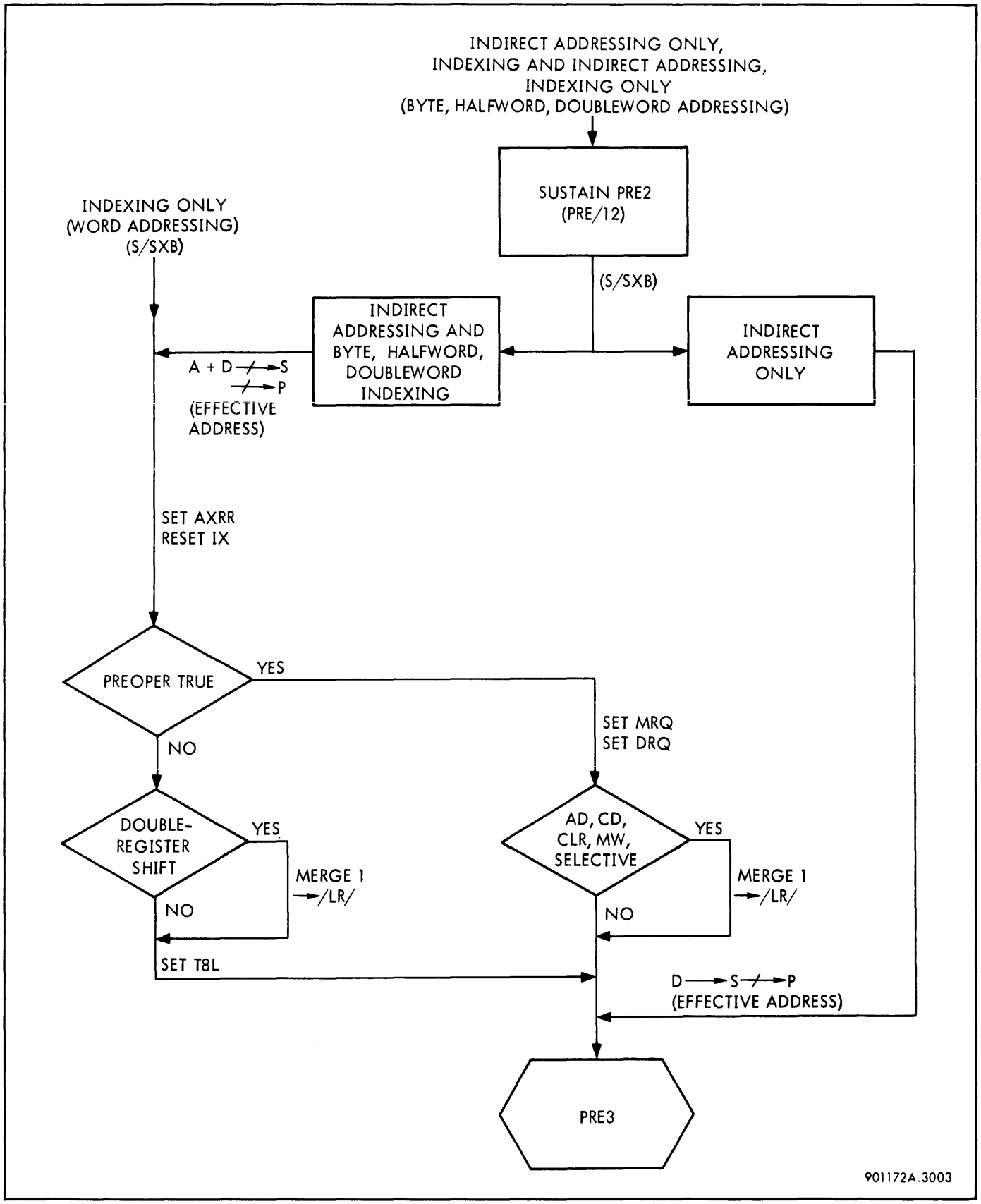


Figure 3-127. Preparation Phase PRE2 (PRE/12 Time), Flow Diagram

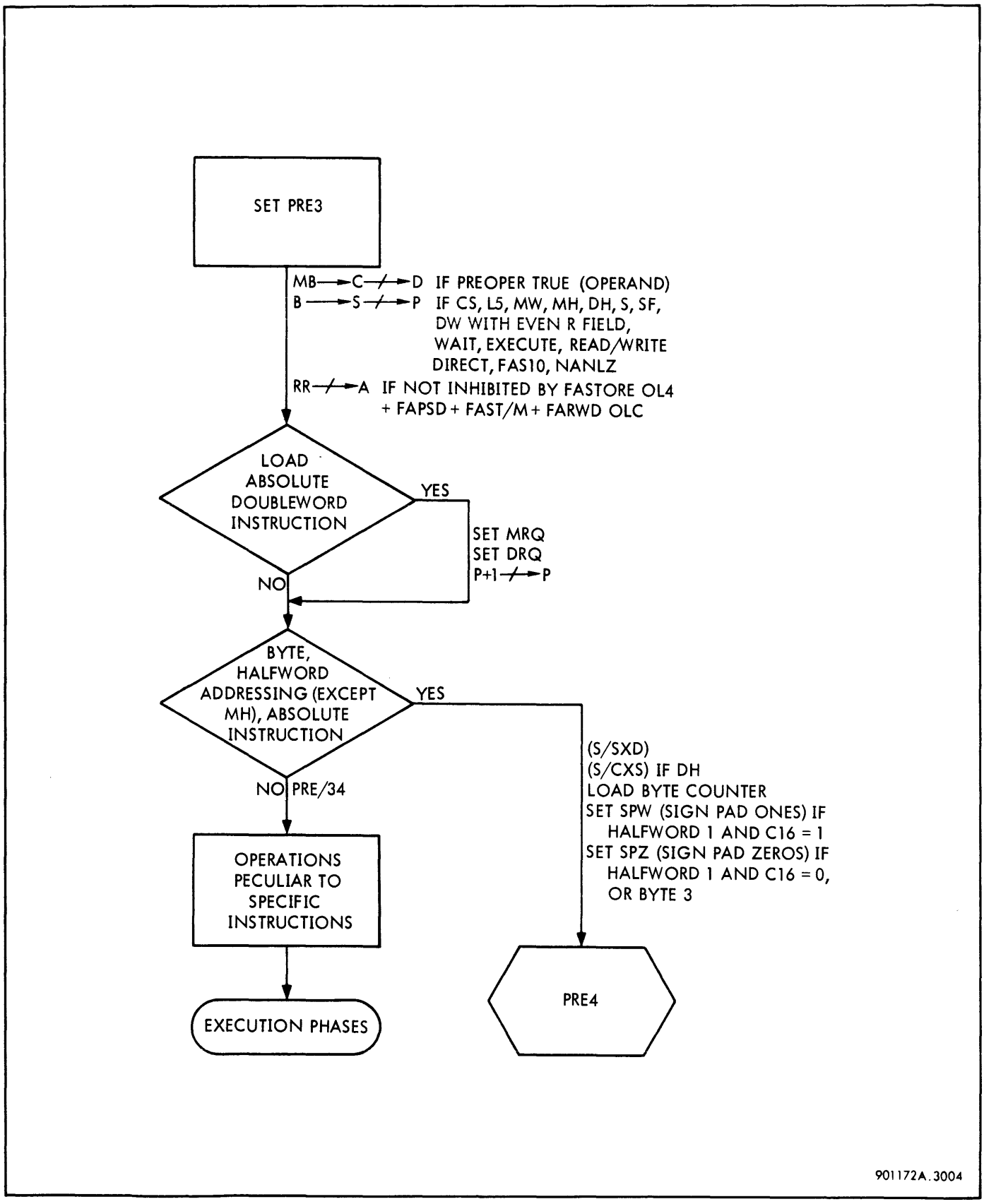


Figure 3-128. Preparation Phase PRE3, Flow Diagram

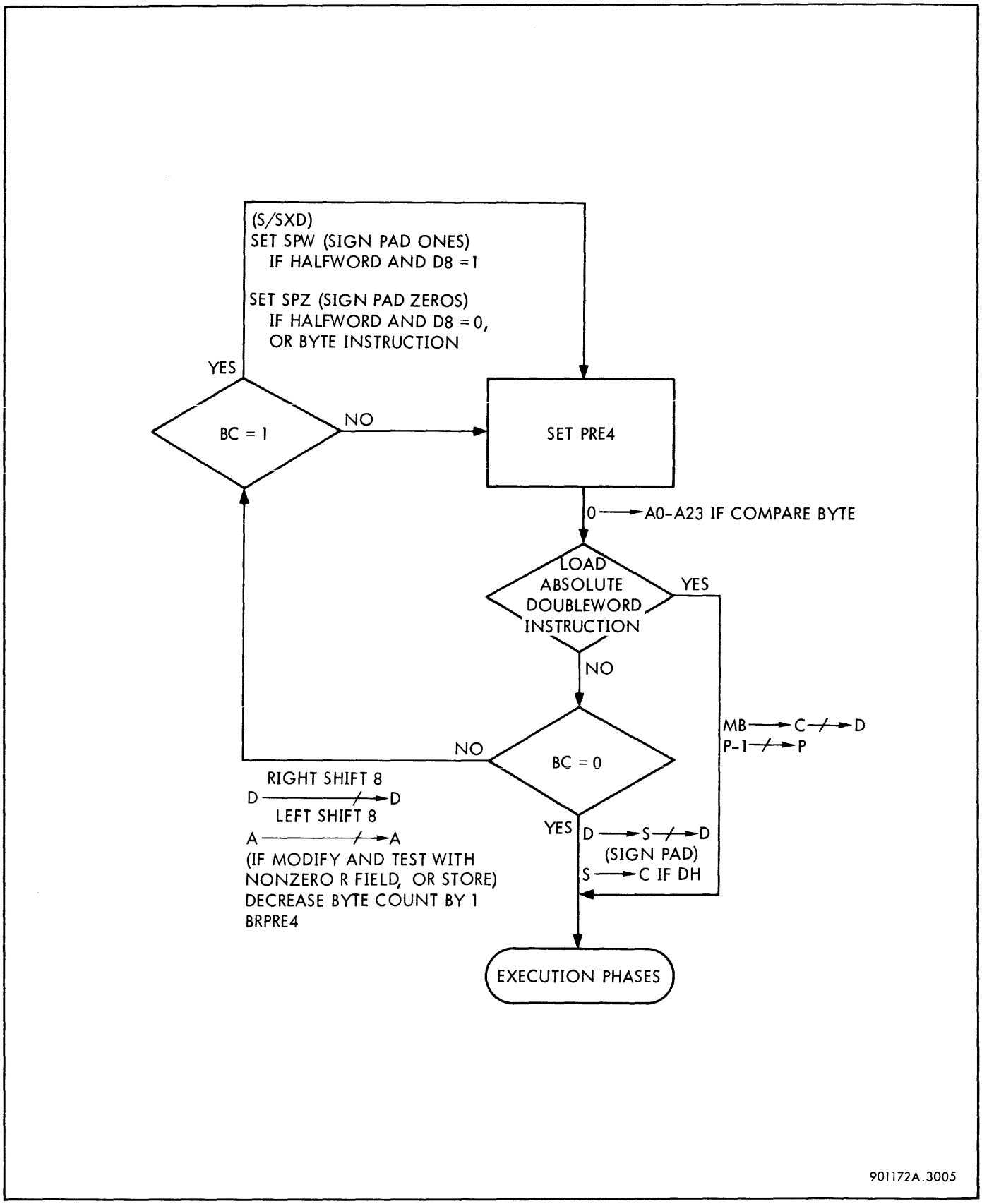


Figure 3-129. Preparation Phase PRE4, Flow Diagram

The index registers are used for byte and halfword addressing as well as for memory address displacement. Byte 0 or halfword 0 of any word may be selected simply by using a byte or halfword instruction. If byte 1, 2, or 3 or halfword 1 is desired, indexing must be performed. To address halfword 1 of any word, the X field of the instruction must designate a register that contains a one in its low-order bit position. To address bytes 1, 2, or 3 of a word, the X field of the instruction must designate a register that contains 01, 10, or 11, respectively, in its two low-order bit positions. The significance of the index register contents for various types of addressing is shown in figure 3-130.

Before the contents of the A-register and D-register are added to obtain the effective address, the A-register must be aligned so that the memory address displacement bits match the effective address bits in the D-register. This alignment operation is shown in figure 3-131. In the case of word operation, the index value is properly placed in the A-register as the value comes from private memory.

For byte operation, bit positions 30 and 31 of the index value contain the byte number and should not be added to the core memory address. In PRE2 the A-register contents are shifted right two bit positions so that the least significant bits of the reference address and the address displacement value are aligned. At the time the shift is made, the byte number is transferred to flip-flops P32 and P33. The outputs of these flip-flops are used to set byte counter BC0 and BC1 in PRE3, and the outputs of the counter flip-flops are used in PRE4 to shift the operand in the D-register or the A-register until the specified byte is in the proper position for instruction execution. The logic used to perform byte indexing is describedin table 3-19.

For halfword operation, bit position 31 of the index value contains a one if halfword 1 is to be addressed. In PRE2, the A-register contents are shifted right one bit position to align the address displacement value with the reference address or indirect address. At the time the shift is made, if halfword 1 is designated the one in A31 is transferred to flip-flop P32. The output of this flip-flop is used to set the byte counter in PRE3, and the counter outputs are used in PRE4 to shift the operand right in the D-register for load operation, or left in the A-register for store operation, until the specified halfword is in the proper position for instruction execution. The logic used to perform halfword indexing is described in table 3-19.

In doubleword operation, the memory address displacement value is treated as an even number by shifting the A-register left one bit position and clearing A31. Bit 31 of the instruction reference address is ignored. The instruction plus the index value, therefore, always addresses the even-numbered location of the specified doubleword. The odd-numbered location is addressed when required by setting flip-flop P31 during PRE/12. Bit 31 of the effective address is inhibited by S31INH when the effective address is placed on the sum bus in PRE/12. At the same time flip-flop P31 is set for all doubleword instructions except floating-point and load absolute doubleword.

In shift operation, the index value alters the shift count and direction and has nothing to do with addressing.

The detailed logic used to implement the indexing feature is explained in table 3-19.

BYTE COUNTER. The byte counter, BC0 and BC1, is used in preparation phase PRE4 to control the shifting of bytes and halfwords in the A- and D-registers before instruction execution.

For load operation, the effective byte is read into the D-register and moved to the least significant end of the register in PRE4. Byte 0 is shifted eight bit positions to the right three times, byte 1 is shifted right twice, and byte 2 is shifted right once. If the effective byte is byte 0, the counter is loaded with 11; if the effective byte is byte 1, the counter is loaded with 10; if the effective byte is byte 2, the counter is loaded with 01. The counter contents are decreased by one with each pass through PRE4 so that shifting is complete. Byte 3 requires no shifting; therefore, the counter remains in the zero state when byte 3 is addressed.

For halfword load operation, the byte counter remains clear for halfword 1 and is set to 10 for halfword 0. This causes the halfword to be shifted right twice in PRE4 to place it in the least significant half of the register.

For store operation or a modify and test instruction, the effective byte or halfword or the byte or halfword from private memory loaded into the A-register must be shifted left for computation or for storage in the effective memory byte or halfword location. The byte counter is set in the same manner as for load operation. If the effective byte is 0, the byte in A24 through A31 must be shifted left three times; therefore, the byte counter is set to 11. The byte must be shifted twice to reach effective byte location 1 and once to reach byte location 2; therefore, the counter is set to 10 and 01, respectively. Similarly, the byte counter is set to 10 for halfword 0 store or modify and test operation, because two 8-bit shifts are required to move the halfword from the least significant end of the register to the halfword 0 location. During a modify and test instruction, the phase sequence returns to PRE4 for register shifting after some of the execution phases have taken place.

SIGN EXTENSION. Sign extension is required for most immediate and halfword instructions, and the load byte instruction requires clearing the most significant bits of the effective memory location. Preparation for sign extension is done in PRE3 or PRE4 by setting flip-flop SPW for sign-padding ones and flip-flop SPZ for sign-padding zeros. The equations for setting these flip-flops are given in table 3-19 under phases PRE3 and PRE4. The outputs of flip-flops SPW and SPZ are used in the adder propagate logic to generate ones or zeros where needed.

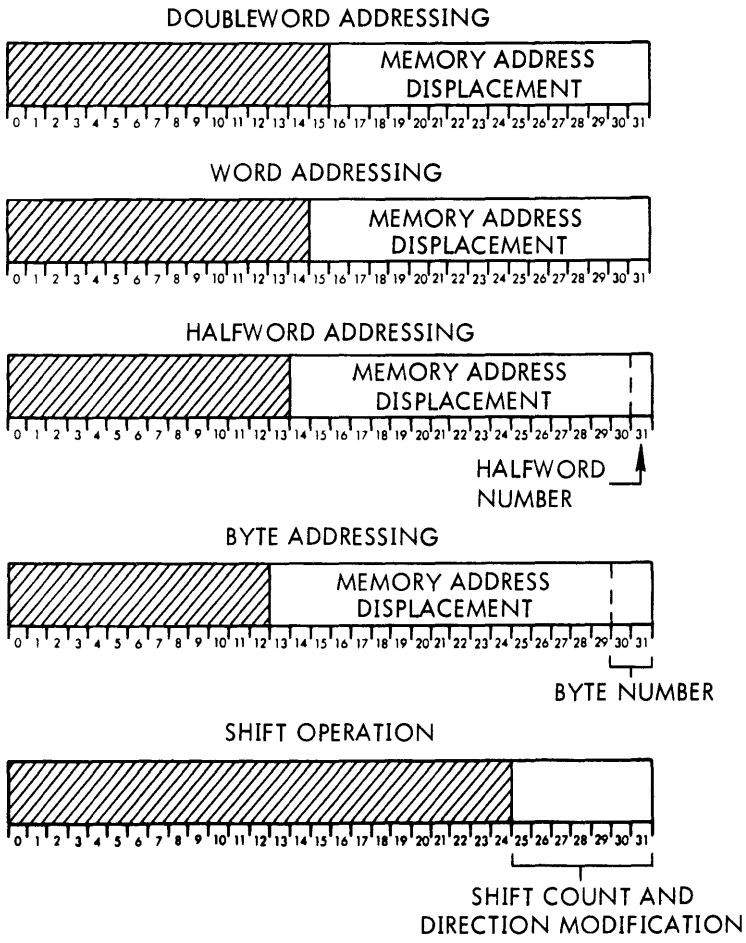


Figure 3-130. Index Register Contents for Byte, Halfword, Word, Doubleword, and Shift Operations

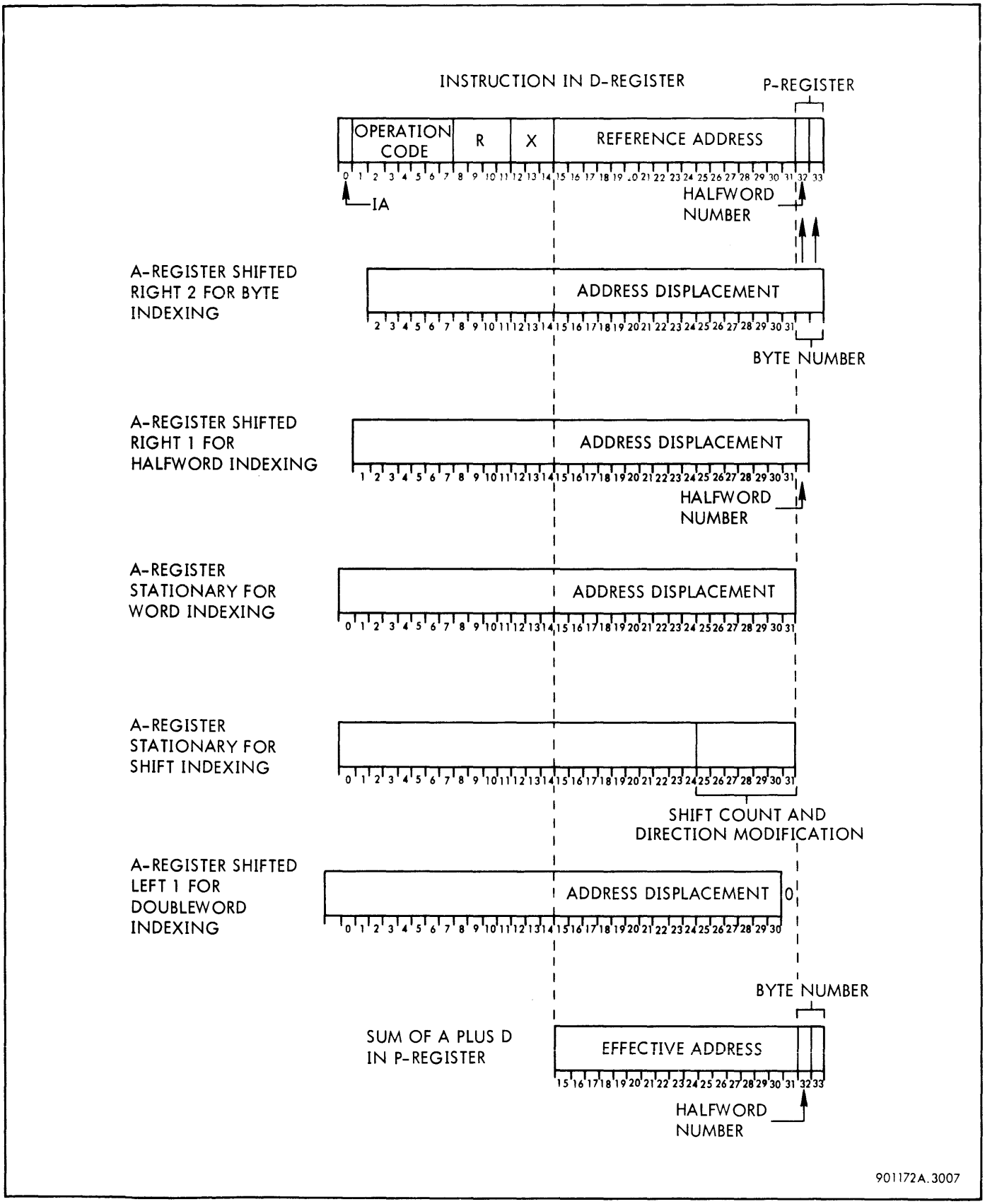


Figure 3-131. Index Register Alignment for Effective Address Computation

The following equations illustrate the generation of propagate terms in the adder and their use in sign extension. Bits 0 through 7 of the sum bus are used as an example.

S0-S7 = PR0-PR7

PR0-PR7 = (A0 D0)-(A7 D7) PRXAD/0
+ (A0 ND0)-(A7 ND7) PRXAND/0
+ (NA0 D0)-(NA7 D7) PRXNAD/0
+ (NA0 ND0)-(NA7 ND7) PRXNAND/0

PRXAD/0 = PRXAD/1B N(SPZ NDIS)

PRXAD/1B = N(NFAIM (PZ NDIS)

PRXAND/0 = PRXAND/1A

PRXAND/1A = N(NPRXAND NDIS NSPW)

PRXNAD/0 = PRXNAD/1B N(SPZ NDIS)

PRXNAD/1B = N(NFAIM SPZ NDIS)

PRXNAND/0 = (PRXNAND + SPW) NDIS

Therefore:

SPZ $\Rightarrow$ NPRXAD/0 NPRXAND/0 NPRXNAD/0 NPRXNAND/0

SPW $\Rightarrow$ PRXAD/0 PRXAND/0 PRXNAD/0 PRXNAND/0

If SPZ is true, the propagates for bits 0 through 7 of the sum bus are disabled regardless of the states of A- and D-register flip-flops 0 through 7, and the propagates are unconditionally enabled if SPW is true. This causes zeros to be placed on sum bus bits 0 through 7 if SPZ is true and ones to be placed in the same bits if SPW is true. This type of logic operates for bits 8 through 15, except that the logic differs for immediate instructions, since only bits 8 through 11 are affected.

3-60 Family of Load Instructions (FALOAD)

LOAD IMMEDIATE (LI; 22). The LI instruction extends the sign (bit 12) of the value field of the instruction word (bits 12 through 31) 12 bit positions to the left and loads the 32-bit result into private memory register R.

General. This instruction is of the immediate addressing type. Therefore, the value field in the instruction word contains an operand which is used as part of the instruction execution. Sign extension is executed in the preparation phases to produce a 32-bit effective word.

Condition Codes. If the effective word is positive and not zero, condition code flip-flop CC3 is set. If the effective word is negative, condition code flip-flop CC4 is set. Both flip-flops CC3 and CC4 are reset if the effective word is zero.

Load Immediate Phase Sequences. Preparation phases for the LI instruction are the same as the general PREP phases for immediate instructions, paragraph 3-59. Table 3-20 lists the detailed logic sequence during the LI execution phases.

Table 3-20. Load Immediate Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(C) : Value field$_{SE}$		Sign-extended value field of instruction word
	(D) : Value field$_{SE}$		
	(P) : Program address		Address of next instruction in sequence
	Enable signal (S/SXD)	(S/SXD) = FALOAD (PRE/34 + PH2) + ...	Preset adder for D⟶S in PH1
		FALOAD = NOU0 OL2	
	Set flip-flop MRQ	S/MRQ = (S/MRQ/1) + ...	Core memory request for next instruction in sequence
		(S/MRQ/1) = FAS10 PRE/34 + ...	
		FAS10 = FAS11/1 NOU1	
		FAS11/1 = FALOAD + ...	
		R/MRQ = ...	
	Set flip-flop RW	S/RW = (S/RW/1) + ...	Preset to write value field into private memory register R in PH1
		(S/RW/1) = FAS11 (PRE/34 + PH2) NOL1 + ...	
		R/RW = ...	
PH1	One clock long		
T8EN (OR T11L)	(D0-D31)⟶(S0-S31) ⟶(RR0-RR31)	Adder logic set at last PREP clock	
		RWXS/0-RWXS/3 = RW + ...	Transfer value field into private memory register R
		RW = Set at last PREP clock	
	Set flip-flop CC3 if (S0-S31) is positive and nonzero; otherwise reset CC3	S/CC3 = SGTZ TESTS + ...	Set condition codes if applicable
		SGTZ = (S0 + S1 + ... + S31) NS0	
		TESTS = FAS11 (PH1 + PH3) + ...	
		R/CC3 = TESTS + ...	
			Mnemonic: LI (22)

(Continued)

Table 3-20. Load Immediate Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH1 T8EN (OR T11L) (Cont.)	Set flip-flop CC4 if (S0-S31) is negative; otherwise reset CC4	S/CC4 = S0 TESTS + ... R/CC4 = TESTS + ...	
	Enable clock T8 if R is in register blocks 0-3; disable clock T8, allowing T11L, if R is in register extension unit, blocks 4-15	T8EN = NT5EN NT11L N(SXADD/1 RW) N(RW REU) N(REU AXRR)	T5EN is disabled by signal RW T11L is enabled if T8EN is disabled by REU and RW
	Branch to PH10	BRPH10 = FAS10 PH1 + ... S/PH10 = BRPH10 NCLEAR-1 + ... R/PH10 = ...	
		S/DRQ = (S/DRQ) NCLEAR-2 (S/DRQ) = BRPH10 + ... R/DRQ = ...	Inhibits transmission of another clock until data release received from core memory
PH10 DR	ENDE functions	See table 3-18	
			Mnemonic: LI (22)

LOAD BYTE (LB; 72, F2). The LB instruction loads the effective byte into bit positions 24 through 31 of private memory register R and clears bit positions 0 through 23 of the register.

General. The effective byte is transferred to the D-register during the load byte PREP phases. If the effective byte is not located in bits 24 through 31 (byte position 3) of the word, the byte is shifted one, two, or three bytes to the right to place the byte in byte position 3. Zeros are then placed in bit positions 0 through 23. The 32-bit result is transferred to private memory register R during execution phase PH1.

Condition Codes. If the result in the R-register is zero, the condition codes are set to XX00. If the result is nonzero, the condition codes are set to XX10.

Load Byte Phase Sequences. Preparation phases for the LB instruction are the same as the general PREP phases for byte instructions, paragraph 3-59. Table 3-21 lists the detailed logic sequence during the LB execution phases.

Table 3-21. Load Byte Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(C) : EB		Effective byte
	(D) : EB		
	(P) : Program address		Address of next instruction in sequence
	Enable signal (S/SXD)	(S/SXD) = FALOAD (PRE34/ + PH2) + ...	Preset to place EB on sum bus
		FALOAD = NOU0 OL2	
	Set flip-flop MRQ	S/MRQ = (S/MRQ/1) + ...	Core memory request for next instruction in sequence
		(S/MRQ/1) = FAS10 PRE/34 + ...	
		FAS10 = FAS11/1 + ...	
		FAS11/1 = FALOAD + ...	
		R/MRQ = ...	
	Set flip-flop RW	S/RW = (S/RW/1) + ...	Prepare to write EB into private memory register R
		(S/RW/1) = FAS11 (PRE/34 + PH2) NOL1 + ...	
		R/RW = ...	
PH1 T8EN (OR T11L)	One clock long (D0-D31) ⟶ (S0-S31) ⟶ (RR0-RR31)	Adder logic set at last PREP clock	Transfer effective word to private memory register R
		RWXS/0-RWXS/3 = RW + ...	
		RW = Set at last PREP clock	
	Set flip-flop CC3 if at least one bit in EB is a one; otherwise reset CC3	S/CC3 = SGTZ TESTS + ...	
		SGTZ = (S0 + S1 + ... + S31) NS0	
		TESTS = FAS11 (PH1 + PH3) + ...	
		R/CC3 = TESTS + ...	
			Mnemonic: LB (72, F2)

(Continued)

Table 3-21. Load Byte Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH1 T8EN (OR T11L) (Cont.)	Reset flip-flop CC4	S/CC4 = S0 TESTS + ... R/CC4 = TESTS + ...	CC4 does not set because S0 = 0
	Enable clock T8 if R is in register blocks 0-3; disable clock T8, allowing T11L, if R is in register extension unit, blocks 4-15	T8EN = NT5EN NT11L N(SXADD/1 RW) N(RW REU) N(REU AXRR)	T5EN is disabled by signal RW T11L is enabled if T8EN is disabled by REU and RW
	Branch to PH10	BRPH10 = FAS10 PH1 + ... S/PH10 = BRPH10 NCLEAR-1 + ... R/PH10 = ...	
		S/DRQ = (S/DRQ) NCLEAR-2 (S/DRQ) = BRPH10 + ... R/DRQ = ...	Inhibits transmission of another clock until data release received from core memory
PH10 DR	ENDE functions	See table 3-18	
			Mnemonic: LB (72, F2)

LOAD HALFWORD (LH; 52, D2). The LH instruction loads the sign-extended effective halfword into private memory register R.

General. The effective halfword is transferred to bit positions 16 through 31 of the D-register during the LH PREP phases. The sign of the effective halfword is extended to occupy bit positions 0 through 15 of the D-register. The 32-bit result is transferred to private memory register R during execution phase PH1.

Condition Codes. If the result in the R-register is zero, the condition codes are set to XX00. If the result is nonzero and positive, the condition codes are set to XX10. A negative result produces condition code settings of XX01.

Load Halfword Phase Sequences. Preparation phases for the LH instruction are the same as the general PREP phases for halfword instructions, paragraph 3-59. Table 3-22 lists the detailed logic sequence during all LH execution phases.

LOAD WORD (LW; 32, B2). The LW instruction loads the effective word into private memory register R. Condition codes are set as in the LH instruction.

Load Word Phase Sequences. Preparation phases for the LW instruction are the same as the general PREP phases for word instructions, paragraph 3-59. Table 3-22 lists the detailed logic sequence during all LW execution phases.

Table 3-22. Load Word and Load Halfword Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(C) : EW		Effective word (in halfword instructions, D contains sign-extended effective halfword)
	(D) : EW		
	(P) : Program address		Address of next instruction in sequence
	Enable signal (S/SXD)	(S/SXD) = FALOAD (PRE/34 + PH2) + ...	Preset adder for D ⟶ S in PH1
		FALOAD = NOU0 OL2	
	Set flip-flop MRQ	S/MRQ = (S/MRQ/1) + ...	Core memory request for next instruction in sequence
		(S/MRQ/1) = FAS10 PRE/34 + ...	
		FAS10 = FAS11/1 + ...	
		FAS11/1 = FALOAD + ...	
		R/MRQ = ...	
	Set flip-flop RW	S/RW = (S/RW/1) + ...	Prepare to write EW into private memory register R
		(S/RW/1) = FAS11 (PRE/34 + PH2) NOL1 + ...	
		R/RW = ...	
PH1 T8EN (OR T11L)	One clock long	Adder logic set at last PREP clock	
	(D0-D31) ⟶ (S0-S31) ⟶ (RR0-RR31)	RWXS/0-RWXS/3 = RW + ...	Transfer effective word to private memory register R
		RW = Set at last PREP clock	
	Set flip-flop CC3 if (S0-S31) is positive and nonzero; otherwise reset CC3	S/CC3 = SGTZ TESTS + ...	Set condition codes if applicable
		SGTZ = (S0 + S1 + ... + S31) NS0	
		TESTS = FAS11 PH1 + ...	
		R/CC3 = TESTS + ...	
			Mnemonic: LW (32, B2) LH (52, D2)

(Continued)

Table 3-22. Load Word and Load Halfword Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH1 T8EN (OR T11L) (Cont.)	Set flip-flop CC4 if (S0-S31) is negative; otherwise reset CC4	S/CC4 = S0 TESTS + ... R/CC4 = TESTS + ...	
	Enable clock T8 if R is in register blocks 0-3; disable clock T8, allowing T11L, if R is in register extension unit, blocks 4-15.	T8EN = NT5EN NT11L N(SXADD/1 RW) N(RW REU) N(REU AXRR)	T5EN is disabled by signal RW T11L is enabled if T8EN is disabled by REU and RW
	Branch to PH10	BRPH10 = FAS10 PH1 + ... S/PH10 = BRPH10 NCLEAR-1 + ... R/PH10 = ...	
		S/DRQ = (S/DRQ) NCLEAR-2 (S/DRQ) = BRPH10 + ... R/DRQ = ...	Inhibits transmission of another clock until data release received from core memory
PH10 DR	ENDE functions	See table 3-18	
			Mnemonic: LW (32, B2) LH (52, D2)

LOAD DOUBLEWORD (LD; 12, 92). The LD instruction loads the least significant word (bits 32 through 63) of the effective doubleword into private memory register Ru1 and the most significant word (bits 0 through 31) of the effective doubleword is loaded into private memory register R.

If the R field is odd, both words of the effective doubleword are loaded into the same private memory register. At the end of the instruction, private memory register R contains the most significant word of the doubleword (since it is the last to be loaded).

Condition Codes. If the effective doubleword is zero, the condition codes are set to XX00. If the result is nonzero and positive, the condition codes are set to XX10. A negative result produces condition code settings of XX01.

Load Doubleword Phase Sequences. Preparation phases for the LD instruction are the same as the general PREP phases for doubleword instructions, described in paragraph 3-59. Table 3-23 lists the detailed logic sequence during all LD execution phases.

Table 3-23. Load Doubleword Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(D) : ED_{LSW}		Least significant word of effective doubleword
	(C) : ED_{LSW}		
	(P) : ED_{MSW} address		Address of most significant word of effective doubleword
	(B) : Program address		Address of next instruction in sequence
	Enable signal (S/SXD)	(S/SXD) = FALOAD (PRE/34 + PH2) + ... FALOAD = NOU0 OL2	Preset adder for D⟶S in PH1
	Set flip-flop MRQ	S/MRQ = (S/MRQ/3) + ... (S/MRQ/3) = FADW/1 PRE/34 + ... FADW/1 = OU1 FAS11 FAS11 = FAS11/1 + ... FAS11/1 = FALOAD + ... R/MRQ = ...	Preset to fetch most significant word of doubleword from memory
	Reset flip-flop NMRQP1	S/NMRQP1 = N(S/MRQ/3) R/NMRQP1 = ...	Used to delay setting flip-flop DRQ
	Set flip-flop RW	S/RW = (S/RW/1) + ... (S/RW/1) = FAS11 PRE/34 NOL1 + ... R/RW = ...	Prepare to write least significant word of effective doubleword into private memory register Ru1
	Reset flip-flop NLR31F	S/NLR31F = N(S/LR31) (S/LR31) = FADW/1 (NANLZ PRE3) + ... R/NLR31F = ...	Force a one on private memory address line LR31 during PH1 to select private memory register Ru1
PH1 T8EN (OR T11L)	One clock long (D0-D31)⟶(S0-S31) ⟶(RW0-RW31)	Adder logic set at last PREP clock RWXS/0-RWXS/3 = RW + ... RW = Set at last PREP clock	Transfer least significant word of effective doubleword to private memory register Ru1
			Mnemonic: LD (12, 92)

(Continued)

Table 3-23. Load Doubleword Sequence (Cont.)

Phase	Function Performed	Signals Involved			Comments
PH1 T8EN (OR T11L) (Cont.)	Set condition codes	S/CC3	=	SGTZ TESTS + ... (See PH3)	Condition codes set at this time but have no significance. They are set again during PH3
		SGTZ	=	(S0 + S1 + ... + S31) NS0	
		TESTS	=	FAS11 PH1 + ...	
		R/CC3	=	TESTS	
		S/CC4	=	S0 TESTS + ...	
		R/CC4	=	TESTS + ...	
	Reset flip-flop NSXBF	S/NSXBF	=	N(S/SXB)	Preset logic for B⟶S in PH2
		(S/SXB)	=	FADW/1 PH1 + ...	
		R/NSXBF	=	...	
	Reset flip-flop NAXRR	S/NAXRR	=	N(S/AXRR)	No significance during LD
		(S/AXRR)	=	FADW/1 PH1 + ...	
		R/NAXRR=	=	...	
	Enable clock T8 if R and Ru1 are in register blocks 0-3; disable clock T8, allowing T11L, if R and Ru1 are in register extension unit, blocks 8-15	T8EN	=	NT5EN NT11 N(SXADD/1 RW) N(RW REU) N(REU AXRR)	T5EN is disabled by signal RW. T11L is enabled by REU and RW
	Set flip-flop DRQ	S/DRQ	=	(S/DRQ) NCLEAR-2	Inhibits transmission of another clock until data release received from core memory
		(S/DRQ)	=	MRQP1 + ...	
		R/DRQ	=	...	
PH2	One clock long				
DR	(MB0-MB31)⟶(C0-C31) ⟶(D0-D31)	CXMB	=	DG = /DG/	Read most significant word of doubleword into C-register and transfer to D-register
		DXC	=	FADW/1 PH2 + ...	
	(B0-B31)⟶(S0-S31)	SXB	=	NDIS SXBF + ...	Transfer program address to P-register
	⟶(P15-P31)	SXBF	=	Set at PH1 clock	
		PXS	=	FADW/1 PH2 + ...	
	Enable signal (S/SXD)	(S/SXD)	=	FALOAD PH2 + ...	Preset adder for D⟶S in PH3
		S/RW	=	(S/RW/1) + ...	Prepare to write most significant word of effective doubleword into register R
		(S/RW/1)	=	FAS11 PH2 NOL1	
		R/RW	=	...	
	Set flip-flop MRQ	S/MRQ	=	(S/MRQ/3) + ...	Core memory request for next instruction
		(S/MRQ/3)	=	FADW/1 PH2	
		R/MRQ	=	...	
					Mnemonic: LD (12, 92)

(Continued)

Table 3-23. Load Doubleword Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH2 DR (Cont.)	Reset flip-flop NMRQP1	S/NMRQP1 = N(S/MRQ/3) R/NMRQP1 = ...	Prepare to set DRQ at end of PH3
PH3 T8EN (OR T11L)	One clock long (D0-D31)⟶(S0-S31) ⟶(RW0-RW31)	SXD = Set at PH2 clock Adder logic set at PH2 clock RWXS/0-RWXS/3 = RW + ... RW = Set at PH2 clock	Write most significant word of effective doubleword into private memory register R
	Set flip-flop CC3 if (S0-S31) is positive and nonzero; otherwise reset CC3	S/CC3 = SGTZ TESTS + ... SGTZ = (S0 + S1 + ... + S31) NS0 TESTS = FAS11 PH3 + ... R/CC3 = TESTS + ...	Set condition codes if applicable
	Set flip-flop CC4 if (S0-S31) is negative; otherwise reset CC4	S/CC4 = S0 TESTS + ... R/CC4 = TESTS + ...	
	Enable clock T8 if R and Ru1 are in register blocks 0-3; disable clock T8, allowing T11L, if R and Ru1 are in register extension unit, blocks 4-15	T8EN = NT5EN NT11 N(SXADD/1 RW) N(RW REU) N(REU AXRR)	T5EN is disabled by signal RW. T11L is enabled if T8EN is disabled by REU and RW
	Branch to PH10	BRPH10 = FADW/1 PH3 + ... S/PH10 = BRPH10 NCLEAR-1 R/PH10 = ...	
		S/DRQ = (S/DRQ) NCLEAR-2 (S/DRQ) = BRPH10 + MRQP1 + ... R/DRQ = ...	Inhibits transmission of another clock until data release received from core memory
PH10 DR	ENDE functions	See table 3-18	
			Mnemonic: LD (12, 92)

LOAD COMPLEMENT HALFWORD (LCH; 5A, DA). The LCH instruction loads the sign-extended effective halfword into private memory register R.

General. The effective halfword is transferred to bit positions 16 through 31 of the D-register during the LCH PREP phases. The sign of the effective halfword is extended to occupy bit positions 0 through 15 of the D-register. The two's complement of the 32-bit result is transferred to private memory register R during execution phase PH1.

Condition Codes. If the result in the R-register is zero, the condition codes are set to XX00. If the result is nonzero and positive, the condition codes are set to XX10. A negative result produces condition code settings of XX01.

Load Complement Halfword Phase Sequences. Preparation phases for the LCH instruction are the same as the general PREP phases for halfword instructions, paragraph 3-59. Table 3-24 lists the detailed logic sequence during all LCH execution phases.

Table 3-24. Load Complement Halfword Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(C) : EH_{SE}		Sign-extended effective halfword
	(D) : EH_{SE}		
	(P) : Program address		Address of next instruction in sequence
	Enable signal (S/SXMD)	(S/SXMD) = FALOAD/C PRE/34 + ... FALOAD/C = OLA O3	Preset adder for -D⟶S in PH1
	Set flip-flop MRQ	S/MRQ = (S/MRQ/1) + ... (S/MRQ/1) = FAS10 PRE/34 + ... FAS10 = FAS11/1 + ... FAS11/1 = FALOAD/C + ... R/MRQ = ...	Prepare to read next instruction
	Set flip-flop RW	S/RW = (S/RW/1) + ... (S/RW/1) = FAS11 (PRE/34 + PH2) NOL1 + ... FAS11 = FAS11/1 + ... R/RW = ...	Preset to transfer two's complemented effective halfword into private memory register R during PH1
PH1 T8EN (OR T11L)	One clock long -(D0-D31)⟶(S0-S31)⟶ (RW0-RW31)	Adder logic set at last PREP clock RWXS/0-RWXS/3 = RW + ... RW = Set at last PREP clock	Transfer two's complemented effective halfword into private memory register R
	Set flip-flop CC3 if (S0-S31) is positive and nonzero; otherwise reset CC3	S/CC3 = SGTZ TESTS + ... SGTZ = (S0 + S1 + ... + S31) NS0 TESTS = FAS11 (PH1 + PH3) + ... R/CC3 = TESTS + ...	Set condition codes if applicable
			Mnemonic: LCH (5A, DA)

(Continued)

Table 3-24. Load Complement Halfword Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH1 T8EN (OR T11L) (Cont.)	Set flip-flop CC4 if (S0-S31) is negative; otherwise reset CC4	S/CC4 = TESTS NFACOMP S0 + ... R/CC4 = TESTS	
	Enable clock T8 if R is in register blocks 0-3; disable clock T8, allowing T11L, if R is in register extension unit, blocks 4-15	T8EN = NT5EN NT11 N(SXADD/1 RW) N(RW REU) N(REU AXRR)	T5EN is disabled by signal RW T11L is enabled if T8EN is disabled by REU and RW
	Branch to PH10	BRPH10 = FAS10 PH1 + ... S/PH10 = BRPH10 NCLEAR-1 + ... R/PH10 = ...	
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR-2 (S/DRQ) = BRPH10 + ... R/DRQ = ...	Inhibits transmission of another clock until data release received from core memory
PH10 DR	ENDE functions	See table 3-18	
			Mnemonic: LCH (5A, DA)

LOAD COMPLEMENT WORD (LCW; 3A, BA). The LCW instruction loads the two's complemented effective word into private memory register R.

Condition Codes. If the result in the R-register is zero, the condition codes are set to X000. If the result is negative, the condition code flip-flops are set to XX01. A positive result produces condition code flip-flop settings of X010. Overflow can only occur if the effective word is -2^{31} (X'80000000'). Overflow is indicated by setting flip-flop CC1 to produce condition code settings of X101.

Trap Conditions. A trap to memory location X'43' occurs if there is arithmetic overflow and the fixed-point arithmetic mask bit is a one. The result in private memory register R remains unchanged. If overflow occurs and the mask bit is a zero, the next instruction in sequence is executed.

Load Complement Word Phase Sequences. Preparation phases for the PCW instruction are the same as the general PREP phases for word instructions, paragraph 3-59. Table 3-38 lists the detailed logic sequence during all LCW execution phases.

Table 3-25. Load Complement Word Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(D) : EW		Effective word
	(P) : Program address		Next instruction in sequence
	Enable signal (S/SXMD)	(S/SXMD) = FALOAD/C PRE/34 + ...	Preset adder for -D⟶S in PH1
		FALOAD/C = OLA O3	
	Set flip-flop MRQ	S/MRQ = (S/MRQ/1) + ...	Prepare to read next instruction
		(S/MRQ/1) = FAS10 PRE/34 + ...	
		FAS10 = FAS11/1 + ...	
		FAS11/1 = FALOAD/C + ...	
		R/MRQ = ...	
	Set flip-flop RW	S/RW = (S/RW/1) + ...	Preset to transfer two's complemented effective word into private memory register R during PH1
		(S/RW/1) = FAS11 (PRE/34 + PH2) NOL1 + ...	
		FAS11 = FAS11/1 + ...	
		R/RW = ...	
PH1	One clock long		
T8EN (OR T11L)	-(D0-D31)⟶(S0-S31)	Adder logic set at last PREP clock	
	⟶(RW0-RW31)	RWXS/0-RWXS/3 = RW + ...	Transfer two's complemented effective word into private memory register R
		RW = Set at last PREP clock	
	Set flip-flop CC2 if overflow occurs; otherwise reset CC2	S/CC2 = (S00 ⊕ S0) PROBOVER + ...	Set condition codes if applicable. Fixed-point overflow only occurs in this operation when the effective word to be complemented is -2^{31} (X'80000000')
		PROBOVER = FALOAD/C PH1 NO1-1 + ...	
		R/CC2 = PROBOVER + ...	
			Mnemonic: LCW (3A, BA)

(Continued)

Table 3-25. Load Complement Word Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH1 T8EN (OR T11L) (Cont.)	Set flip-flop CC3 if (S0-S31) is positive and nonzero; otherwise reset CC3	S/CC3 = SGTZ TESTS + ... SGTZ = (S0 + S1 + ... + S31) NS0 TESTS = FAS11 PH1 + ... R/CC3 = TESTS + ...	
	Set flip-flop CC4 if (S0-S31) is negative; otherwise reset CC4	S/CC4 = S0 TESTS NFACOMP + ... R/CC4 = ...	
	Enable clock T8 if R is in register blocks 0-3; disable clock T8, allowing T11L, if R is in register extension unit, blocks 4-15	T8EN = NT5EN NT11 N(SXADD/1 RW) N(RW REU) N(REU AXRR)	T5EN is disabled by signal RW T11L is enabled if T8EN is disabled by REU and RW
	Branch to PH10	BRPH10 = FAS10 PH1 + ... S/PH10 = BRPH10 NCLEAR-1 + ... R/PH10 = ...	
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR-2 (S/DRQ) = BRPH10 + ... R/DRQ = ...	Inhibits transmission of another clock until data release received from core memory
PH10 DR	ENDE functions	See table 3-18	
			Mnemonic: LCW (3A, BA)

LOAD COMPLEMENT DOUBLEWORD (LCD; 1A, 9A). The LCD instruction loads the two's complement of the effective doubleword into private memory registers R and Ru1. If the R field is odd, both words of the effective doubleword are loaded into the same private memory register. At the end of the instruction, private memory register R contains the most significant word of the doubleword (since it is the last to be loaded).

Condition Codes. If the two's complemented result is zero, the condition code flip-flops are set to X000. If the result is nonzero and positive, the condition code flip-flops are set to X010. A negative result produces condition code flip-flop settings of XX01. Overflow can only occur if the effective doubleword is -2^{63} (X'8000000000000000'). Overflow is indicated by setting flip-flop CC1, to produce condition code settings of X101.

Trap Conditions. A trap to memory location X'43' occurs if there is arithmetic overflow and the fixed-point arithmetic mask bit is a one. The result in private memory register remains unchanged. If overflow occurs and the mask bit is a zero, the next instruction in sequence is executed.

Load Complement Doubleword Phase Sequences. Preparation phases for the LCD instruction are the same as the general PREP phases for doubleword instructions, paragraph 3-59. Table 3-26 lists the detailed logic sequence during all LCD execution phases.

Table 3-26. Load Complement Doubleword Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(C) : ED_{LSW}		Least significant word of effective doubleword
	(D) : ED_{LSW}		Least significant word of effective doubleword
	(P) : ED_{MSW} address		Address of most significant word of effective doubleword
	(B) : Program address		Address of next instruction in sequence
	Enable signal (S/SXMD)	(S/SXMD) = FALOAD/C PRE/34 + ... FALOAD/C = OLA O3	Preset adder for -D⟶S in PH1
	Set flip-flop MRQ	S/MRQ = (S/MRQ) + ... (S/MRQ) = (S/MRQ/3) (S/MRQ/3) = FADW1 (PRE/34 + PH2) + ... FADW1 = OU1 FAS11 FAS11 = FAS11/1 + ... FAS11/1 = FALOAD + ...	Memory request for most significant word of effective doubleword
	Reset flip-flop NMRQP1	S/NMRQP1 = N(S/MRQ/3) R/NMRQP1 = ...	Used to delay setting flip-flop DRQ
	Set flip-flop RW	S/RW = (S/RW/1) + ... (S/RW/1) = FAS11 PH2 NOL1 + ... R/RW = ...	Prepare to write least significant word of two's complemented doubleword into private memory register Ru1
	Reset flip-flop NLR31F	S/NLR31F = N(S/LR31) (S/LR31) = FADW/1 (NANLZ PRE3) + ... R/NLR31F = ...	Force a one on private memory address line LR31 during PH1 to select private memory register Ru1
PH1 T8 (OR T11L)	One clock long -(D0-D31)⟶(S0-S31)⟶ (RW0-RW31)	Adder logic set at last PREP clock RWXS/0-RWXS/3 = RW + ... RW = Set at last PREP clock	Transfer two's complemented least significant word of effective doubleword to private memory register Ru1
	Reset flip-flop NSXBF	S/NSXBF = N(S/SXB) (S/SXB) = FADW/1 PH1 + ... R/NSXBF = ...	Preset logic for B⟶S in PH2
			Mnemonic: LCD (1A, 9A)

(Continued)

Table 3-26. Load Complement Doubleword Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH1 T8 (OR T11L) (Cont.)	Set flip-flop SW0 if result is not equal to zero	S/SW0 = NS0031Z (S/SW0/NZ) + ... NS0031Z = (S0 + S1 + ... + S31) (S/SW0/NZ) = K00HOLD + ... R/SW0 = RESET/A + ...	SW0 is used in PH3 to set CC3. CC2 through CC4 may be set in this phase, but action is meaningless since they are also set in PH3
	Set flip-flop FL3 if end carry	S/FL3 = K00 K00HOLD R/FL3 = N(FUSF PH8 + FUS PH5)	K00 is end carry from complementing the least significant word of the effective doubleword. Flip-flop NK31 will be reset in PH2 if end carry exists
	Enable clock T8 if R and Ru1 are in register blocks 0-3; disable clock T8, allowing T11L, if R and Ru1 are in register extension unit, blocks 4-15	T8EN = NT5EN NT11 N(SXADD/1 RW) N(RW REU) N(REU AXRR)	T5EN is disabled by signal RW. T11L is enabled if T8EN is disabled by REU and RW
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR-2 (S/DRQ) = MRQP1 + ... R/DRQ = ...	Inhibits transmission of another clock until data release received from core memory
PH2 DR	One clock long		
	(MB0-MB31)⟶(C0-C31) ⟶/⟶(D0-D31)	CXMB = DG = /DG/ DXC = FADW/1 PH2 + ...	Read most significant word of doubleword into C-register and clock into D-register
	(B0-B31)⟶(S0-S31) (S15-S31)⟶/⟶(P15-P31)	SXB = SXBF NDIS + ... SXBF = Set at PH1 clock PXS = FADW/1 PH2 + ...	Transfer program address to P-register
	Set flip-flop BRP	S/BRP = FADW/1 PH2 + ... R/BRP = PRE1 NFAIM + ...	Signifies that program address is in P-register
	Enable signal (SXMD)	(S/SXMD) = FALOAD/C PH2 + ...	Preset adder for -D⟶S in PH3
	Reset flip-flop NK31 if there was end carry in PH1; if no end carry, set flip-flop NK31 with N(S/K31/1)	S/NK31 = N(S/K31) N(S/SXAMD/1) + N(S/K31/1) (S/K31) = FADW/1 PH2 + ... (S/K31/1) = K00 (S/K31/3) + ... (S/K31/3) = N(FADW/1 PH2 NFL3) + ...	Provides carry to complementing of most significant word of effective doubleword
	Set flip-flop RW	S/RW = (S/RW/1) + ... (S/RW/1) = FAS11 PH2 NOL1 + ... R/RW = ...	Prepare to write two's complemented most significant word of effective doubleword into private memory register R
			Mnemonic: LCD (1A, 9A)

(Continued)

Table 3-26. Load Complement Doubleword Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH2 DR (Cont.)	Set flip-flop MRQ	S/MRQ = (S/MRQ/3) + ... (S/MRQ/3) = FADW/1 (PRE/34 + PH2) R/MRQ = ...	Core memory request for next instruction in sequence
	Reset flip-flop NMRQP1	S/NMRQP1 = N(S/MRQ/3) R/NMRQP1 = ...	Used to delay setting DRQ
PH3 T8EN (OR T11L)	One clock long		
	-(D0-D31)⟶(S0-S31) ⟶(RW0-RW31)	RWXS/0-RWXS/3 = RW + ... RW = Set at PH2 clock	Write two's complemented most significant word of doubleword into private memory register R
	Set flip-flop CC2 if overflow occurs; otherwise reset CC2. Fixed point overflow occurs if the effective word to be complemented is -2^{63} ('800 ... 00')	S/CC2 = (S00 ⊕ S0) PROBOVER + ... PROBOVER = FALOAD/C PH3 NO1-1 R/CC2 = PROBOVER + ...	Set condition codes if applicable
	Set flip-flop CC3 if (S0-S31) is positive and nonzero; otherwise reset CC3	S/CC3 = SGTZ TESTS + ... SGTZ = (S0 + S1 + ... + S31 + SW0 + ...) NS0 TESTS = FAS11 PH3 + ... R/CC3 = TESTS + ...	SW0 was set in PH1 if two's complement of least significant word of effective doubleword was nonzero
	Set flip-flop CC4 if (S0-S31) is negative; otherwise reset CC4	S/CC4 = S0 TESTS + ... R/CC4 = TESTS + ...	
	Enable clock T8 if R and Ru1 are in register blocks 0-3; disable clock T8, allowing T11L, if R and Ru1 are in register extension unit, blocks 4-15	T8EN = NT5EN NT11 N(SXADD/1) N(RW REU) N(REU AXRR)	T5EN is disabled by signal RW. T11L is enabled if T8EN is disabled by REU and RW
	Branch to PH10	BRPH10 = FAS10 PH1 + ... S/PH10 = BRPH10 NCLEAR-1 R/PH10 = ...	
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR-2 (S/DRQ) = BRPH10 + MRQP1 + ... R/DRQ = ...	Inhibits transmission of another clock until data release received from core memory
PH10 DR	ENDE functions	See table 3-18	
			Mnemonic: LCD (1A, 9A)

LOAD CONDITIONS AND FLOATING CONTROL (LCF; 70, F0). If bit position 10 of the instruction word is a one, LCF loads bit positions 0 through 3 of the effective byte into condition code flip-flops CC1 through CC4. If bit position 11 of the instruction word is a one, LCF loads bits 5 through 7 of the effective byte into floating-point mode control flip-flops FS, FZ, and FNF. If bit position 10 or 11 is a zero, the corresponding transfer is not made.

Load Conditions and Floating Control Phase Sequences. Preparation phases for the LCF instruction are the same as the general PREP phases for byte instructions, paragraph 3-59. Table 3-27 lists the detailed logic sequence during all LCF execution phases.

LOAD CONDITIONS AND FLOATING CONTROL IMMEDIATE (LCFI, 02). If bit position 10 of the instruction word is a one, LCFI loads bit positions 24 through 27 of the instruction word into condition code flip-flops CC1 through CC4. If bit position 11 of the instruction word is a one, LCFI loads bit positions 29 through 31 of the instruction word into floating-point mode control flip-flops FS, FZ, and FNF. If bit position 10 or 11 is a zero, the corresponding transfer is not made.

Load Conditions and Floating Control Immediate Phase Sequences. Preparation phases for the LCFI instruction are the same as the general PREP phases for immediate instructions, paragraph 3-59. Table 3-28 lists the detailed logic sequence during all LCFI execution phases.

Table 3-27. Load Conditions and Floating Control Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(C) : EB		Effective byte (C24-C31)
	(D) : EB		Effective byte (D24-D31)
	(R) : R field of instruction word		The R field of the instruction word contains the two control bits, 10 and 11
	(P) : Program address		Address of next instruction in sequence
	Enable signal (S/SXD)	(S/SXD) = FALCFP PRE/34 + ... FALCFP = FALCF + ... FALCF = OU7 OL0 + ...	Preset adder for D⟶S in PH1
	Set flip-flop MRQ	S/MRQ = (S/MRQ/1) + ... (S/MRQ/1) = FAS10 PRE/34 + ... FAS10 = FAS11/1 NOU1 FAS11/1 = FALCF + ... R/MRQ = ...	Core memory request for next instruction in sequence
PH1 T5L	One clock long (D0-D31)⟶(S0-S31) (S24-S27)⟶(CC1-CC4)	Adder logic set at last PREP clock S/CC1 = S24 CCXS/3 + ... ⋮ ⋮ S/CC4 = S27 CCXS/3 + ... R/CC1-R/CC4 = (R/CC) + ... (R/CC) = CCXS/3 + ... CCXS/3 = FALCF PH1 R30	Load condition code bits from effective byte into CC1 through CC4, providing bit 10 is a one. (R30 holds bit 10 of instruction word)
			Mnemonic: LCF (70, F0)

(Continued)

Table 3-27. Load Conditions and Floating Control Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH1 T5L (Cont.)	S29→FS	S/FS = S29 FCXS + ... R/FS = FCXS + ... FCXS = FALCF PH1 R31	Load floating-point mode control bits from effective byte into FS, FZ, and FNF, providing bit 11 is a one. (R31 holds bit 11 of instruction word)
	S30→FZ	S/FZ = S30 FCXS + ... R/FZ = FCXS + ...	
	S31→FNF	S/FNF = S31 FCXS + ... R/FNF = FCXS + ...	
	Branch to PH10	BRPH10 = FAS10 PH1 + ... S/PH10 = BRPH10 NCLEAR-1 + ... R/PH10 = ...	
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR-2 (S/DRQ) = BRPH10 + ... R/DRQ = ...	Inhibits transmission of another clock until data release received from core memory
PH10 DR	ENDE functions	See table 3-18	
			Mnemonic: LCF (70, F0)

Table 3-28. Load Conditions and Floating Control Immediate Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(C) : Value field$_{SE}$		Sign-extended value field of instruction word
	(D) : Value field$_{SE}$		Sign-extended value field of instruction word
	(R) : R field of instruction		The R field of instruction word contains the two control bits, bits 10 and 11
	(P) : Program address		Address of next instruction in sequence
			Mnemonic: LCFI (02)

(Continued)

Table 3-28. Load Conditions and Floating Control Immediate Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PREP (Cont.)	Enable signal (S/SXD)	(S/SXD) = FALCFP PRE/34 + ... FALCFP = FALCF + ... FALCF = FULCFI + ...	Preset adder for D⟶S in PH1
	Set flip-flop MRQ	S/MRQ = (S/MRQ/1) + ... (S/MRQ/1) = FAS10 PRE/34 + ... FAS10 = FAS11/1 NOU1 FAS11/1 = FALCF + ... R/MRQ = ...	Core memory request for next instruction in sequence
PH1 T5L	One clock long (D0-D31)⟶(S0-S31)	Adder logic set at last PREP clock	
	(S24-S27)⟶(CC1-CC4)	S/CC1 = S24 CCXS/3 + ... ⋮ ⋮ S/CC4 = S27 CCXS/3 + ... R/CC1-R/CC4 = (R/CC) + ... (R/CC) = CCXS/3 + ... CCXS/3 = FALCF PH1 R30	Load condition code bits from value field into CC1 through CC4, providing bit 10 is a one. (R30 holds bit 10 of instruction word)
	S29 ⟶/⟶ FS	S/FS = S29 FCXS + ... R/FS = FCXS + ... FCXS = FALCF PH1 R31	Load floating-point mode control bits from value field into FS, FZ, and FNF, providing bit 11 is a one. (R31 holds bit 11 of instruction word)
	S30 ⟶/⟶ FZ	S/FZ = S30 FCXS + ... R/FZ = FCXS + ...	
	S31 ⟶/⟶ FNF	S/FNF = S31 FCXS + ... R/FNF = FCXS + ...	
	Branch to PH10	BRPH10 = FAS10 PH1 + ... S/PH10 = BRPH10 NCLEAR-1 + ... R/PH10 = ...	
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR-2 (S/DRQ) = BRPH10 + ... R/DRQ = ...	Inhibits transmission of another clock until data release received from core memory
PH10 DR	ENDE functions	See table 3-18	
			Mnemonic: LCFI (02)

LOAD REGISTER POINTER (LRP; 2F, AF). The LRP instruction loads bits 24 through 27 of the effective word into flip-flops RP24 through RP27, respectively. These flip-flops correspond to bits 56 through 59 of the program status doubleword. If the computer contains less than the maximum number of 16 blocks of general registers, it is possible to load the pointer with a value that points to a nonexistent register block. If the pointer is loaded with such a value, all ones are generated when a register of the nonexistent block is addressed by the R field of a subsequent instruction.

Load Register Pointer Phase Sequences. Preparation phases for the LRP instruction are the same as the general PREP phases for word instructions, paragraph 3-59. Table 3-29 lists the detailed logic sequence during all LRP execution phases.

Table 3-29. Load Register Pointer Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(C) : EW		Effective word
	(D) : EW		Effective word (bit positions 24 through 27 contain the number of the current register block to be loaded into register RP)
	(P) : Program address		Address of next instruction in sequence
	Enable signal (S/SXD)	(S/SXD) = FALCFP PRE/34-A + ... FALCFP = FULRP + ...	Preset adder for D⟶S in PH1
	Set flip-flop MRQ	S/MRQ = (S/MRQ/1) + ... (S/MRQ/1) = FAS10 PRE/34 FAS10 = FAS11/1 NOU1 FAS11/1 = FULRP + ... FULRP = OU2 OLF R/MRQ = ...	Core memory request for next instruction in sequence
PH1	One clock long		
T5L	(D0-D31)⟶(S0-S31)	Adder logic set at last PREP clock	
	(S24-S27)⟶(RP24-RP27)	S/RP24 = S24 RPXS + ... ⋮ S/RP27 = S27 RPXS + ... R/RP24-R/RP27 = RPXS RPXS = FULRP PH1-F + ...	Transfer bits 24 through 27 of effective word to register RP
	Branch to PH10	BRPH10 = FAS10 PH1 + ... S/PH10 = BRPH10 NCLEAR-1 R/PH10 = ...	
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR-2 (S/DRQ) = BRPH10 + ... R/DRQ = ...	Inhibits transmission of another clock until data release received from core memory
PH10 DR	ENDE functions	See table 3-18	
			Mnemonic: LRP (2F, AF)

3-61 Family of Load Absolute Instructions (FALOAD/A)

LOAD ABSOLUTE HALFWORD (LAH, 5B, DB). The LAH instruction extends the sign of the effective halfword 16 bit positions to the left and takes the absolute value of the resulting 32-bit number. The absolute value equals the number when the sign is positive or the two's complement when the sign is negative. The absolute value is then loaded into private memory register R. Examples of an LAH are:

EH	1111111111101110 (-18_{10})
Sign-extended EH	11111111111111111111111111101110 (-18_{10})
Absolute value	00000000000000000000000000010010 (18_{10})
R	00000000000000000000000000010010 (18_{10})

Condition Codes. If the result in the R-register is zero, the condition codes are set to XX00. If the result is nonzero, the condition codes are set to XX10. A nonzero result is always positive.

LAH Phase Sequence. LAH preparation phases are the same as the general PREP phases for halfword instructions as described in paragraph 3-59. Figure 3-132 shows the simplified phase sequence for the instruction during execution and table 3-31 lists the detailed logic sequence during the LAH execution phases.

LOAD ABSOLUTE WORD (LAW, 3B, BB). The LAW instruction loads the absolute value of the effective word into private memory register R. The absolute value equals the effective word if the sign of the effective word is positive. If the effective word is negative, the absolute value equals the two's complement of the effective word.

Overflow. Fixed-point arithmetic overflow occurs if the effective word is -2^{31} (1000.....000) since recomplementing produces a positive number too large to be held in a 32-bit register. Overflow causes a trap to memory location X'43' after execution of LAW if the arithmetic mask is a one. If the arithmetic mask is a zero, the next instruction in sequence is executed.

Condition Codes. If the R-register result is zero, the condition codes are set to XX00. If the result is nonzero, the condition codes are set to XX10. Flip-flop CC2 of the condition codes is set if fixed-point arithmetic overflow occurs.

Load Absolute Word Phase Sequence. LAW preparation phases are the same as the general PREP phases for word instructions as described in paragraph 3-59. Figure 3-133 shows the simplified phase sequence for the instruction during execution. Table 3-31 lists the detailed logic sequence during all LAW execution phases.

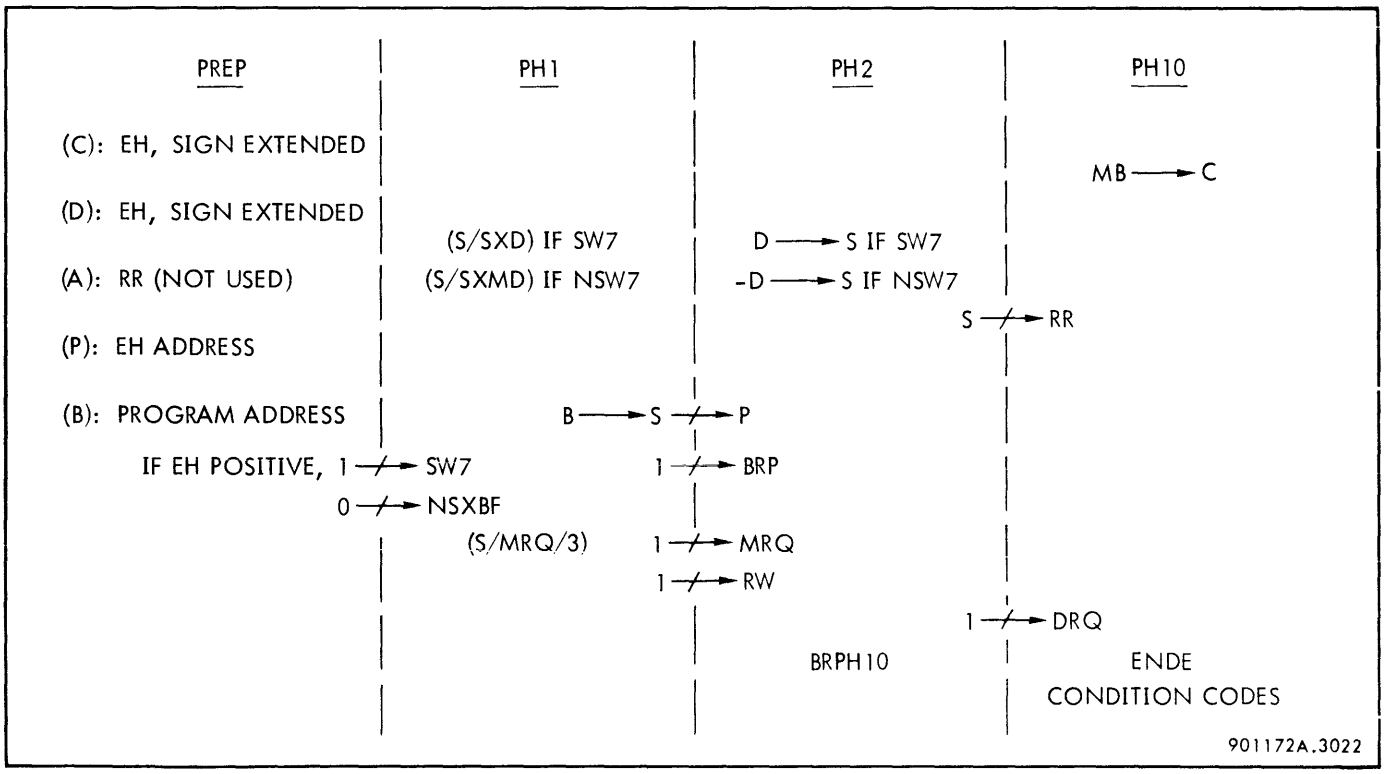


Figure 3-132. Load Absolute Halfword Phases

Table 3-30. Load Absolute Halfword Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(C) : EH, sign-extended		Effective halfword, with sign-extended 16 bit positions to the left. In two's complement form if negative
	(D) : EH, sign-extended		
	(A) : RR (not used)		Contents of private memory register R. Not used during this instruction
	(P) : EH address		Effective halfword address
	(B) : Program address		Address of next instruction in sequence
	Set flip-flop SW7 if EH positive	S/SW7 = FALOAD/A ND16 OU5 PRE/34 + ...	Flip-flop SW7 stores polarity of EH for computing absolute value in PH2
		R/SW7 = RESET/A + ...	
	Reset flip-flop NSXBF	S/NSXBF = N(S/SXB)	Preset logic for B⟶S in PH1
		(S/SXB) = FALOAD/A PRE/34 + ...	
		R/NSXBF = ...	
PH1	One clock long		
T5L	(B0-B31)⟶(S0-S31)	SXB = NDIS SXBF + ...	Transfer program address to P-register
		SXBF = Set at last PREP clock	
	(S15-S31)⟶(P15-P31)	PXS = FALOAD/A PH1 NOU1 + ...	
	If sign-extended EH positive, enable signal (S/SXD)	(S/SXD) = FALOAD/A PH1 SW7 + ...	Preset adder for D⟶S in PH2. Sign-extended effective halfword equals absolute value
	If sign-extended EH negative, enable signal (S/SXMD)	(S/SXMD) = FALOAD/A PH1 NSW7 + ...	Preset adder for -D⟶S in PH2. Sign-extended effective halfword two's complemented to find absolute value
	Set flip-flop MRQ	S/MRQ = (S/MRQ/3) + ...	Core memory request for next instruction in sequence
		(S/MRQ/3) = FALOAD/A PH1 + ...	
		R/MRQ = ...	
	Set flip-flop RW	S/RW = FALOAD/A PH1 + ...	Prepare to write result into private memory
		R/RW = ...	
	Set flip-flop BRP	S/BRP = FALOAD/A PH1 NOU1 + ...	Signifies that program address is in P-register
		R/BRP = PRE1 NFAIM + INTRAP1 + ...	
			Mnemonic: LAH (5B, DB)

(Continued)

Table 3-30. Load Absolute Halfword Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH2	One clock long		
T8L	If sign-extended EH positive	Adder logic set at PH1 clock	Transfer absolute value to private memory register R
	(D0-D31)⟶(S0-S31)⟶ (RW0-RW31)	RWXS/0-RWXS/3 = RW + ... RW = Set at PH1 clock	
	If sign-extended EH negative	Adder logic set at PH1 clock	Transfer absolute value to private memory register R
	-(D0-D31)⟶(S0-S31)⟶ (RW0-RW31)	RWXS/0-RWXS/3 = RW + ...	
	Set flip-flop CC3 if (S0-S31) is nonzero; otherwise reset CC3	S/CC3 = SGTZ TESTS + ... SGTZ = (S0 + S1 + ... + S31) = N(S0 NFACOMP) + ... TESTS FALOAD/A PH2 + ... R/CC3 = TESTS + ...	Absolute value nonzero S0 must be a zero; absolute value never negative
	Reset flip-flop CC4	R/CC4 = ...	CC4 is always zero
	Enable clock T8L	T8EN = NT5EN NT11L N(SXADD/1 RW) N(RW REU) N(REU AXRR) NT5EN = RW + ...	T5EN disabled by signal RW
	Branch to PH10	BRPH10 = FALOAD/A PH2 NOU1 + ... S/PH10 = BRPH10 NCLEAR + ... R/PH10 = ...	
	Set flip-flop DRQ	S/DRQ = BRPH10 NCLEAR + ... R/DRQ = ...	Inhibits transmission of another clock until data release signal from core memory
PH10 DR	ENDE functions	See table 3-18	
			Mnemonic: LAH (5B, DB)

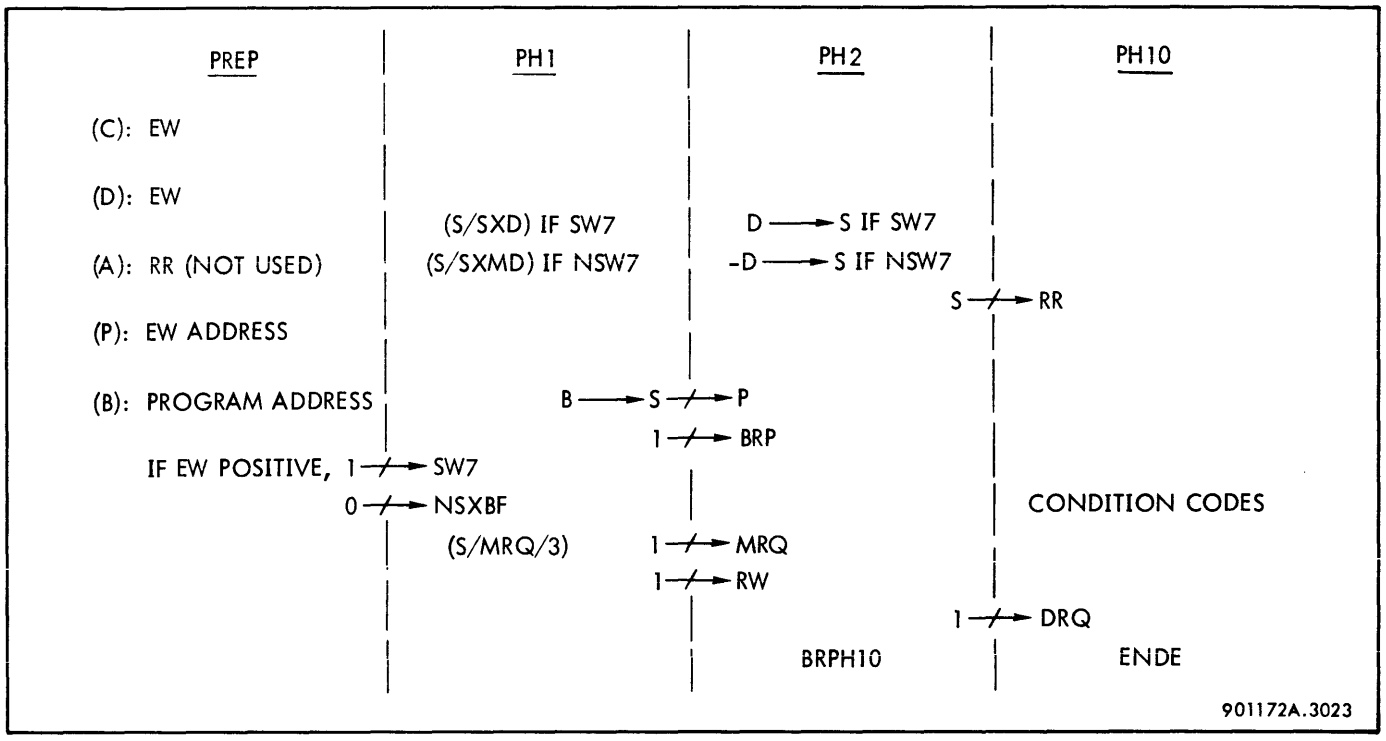


Figure 3-133. Load Absolute Word Phases

Table 3-31. Load Absolute Word Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(C): EW		Effective word. May be positive or negative
	(D): EW		Effective word
	(A): RR (not used)		Contents of private memory register R. Not used during this instruction
	(P): EW address		Effective word address
	(B): Program address		Address of next instruction in sequence
	Set flip-flop SW7 if EW positive	S/SW7 = FULAWORDW ND0 PRE/34 + ... FULAWORDW = FALOAD/A NO1 R/SW7 = RESET/A + ...	Flip-flop SW7 stores polarity of effective word for computing absolute value in PH2
	Reset flip-flop NSXBF	S/NXSBF = N(S/SXB) (S/SXB) = FALOAD/A PRE/34 + ... R/NSXBF = ...	Preset logic for B → S in PH1
			Mnemonic: LAW (3B, BB)

(Continued)

Table 3-31. Load Absolute Word Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH1	One clock long		
T5L	(B0-B31)⟶(S0-S31)	SXB = NDIS SXBF + ... SXBF = Set at last PREP clock	Transfer program address to P-register
	(S15-S31)⟶(P15-P31)	PXS = FALOAD/A PH1 NOU1 + ...	
	Set flip-flop BRP	S/BRP = FALOAD/A PH1 NOU1 + ... R/BRP = PRE1 NFAIM + INTRAP1 + ...	Signifies that program address is in P-register
	If EW positive, enable signal (S/SXD)	(S/SXD) = FALOAD/A PH1 SW7 + ...	Preset adder for D⟶S in PH2. Effective word equals absolute value
	If EW negative, enable signal (S/SXMD)	(S/SXMD) = FALOAD/A PH1 NSW7 + ...	Preset adder for -D⟶S in PH2. Effective word two's complemented to find absolute value
	Set flip-flop MRQ	S/MRQ = (S/MRQ/3) + ... (S/MRQ/3) = FALOAD/A PH1 + ... R/MRQ = ...	Core memory request for next instruction in sequence
	Set flip-flop RW	S/RW = FALOAD/A PH1 + ... R/RW = ...	Prepare to write result in private memory
PH2	One clock long		
T8L	If EW positive (D0-D31)⟶(S0-S31)⟶(RW0-RW31)	Adder logic set at PH1 clock RWXS/0-RWXS/3 = RW + ... RW = Set at PH1 clock	Transfer absolute value to private memory register R
	If EW negative -(D0-D31)⟶(S0-S31)⟶(RW0-RW31)	Adder logic set at PH1 clock RWXS/0-RWXS/3 = RW + ...	Transfer absolute value to private memory register R
	Set flip-flop CC2 if arithmetic overflow; otherwise reset CC2	S/CC2 = (S00 ⊕ S0) PROBOVER + ... PROBOVER = FALOAD/A PH2 NO1 + ... R/CC2 = PROBOVER + ...	Arithmetic overflow occurs during LAW only for effective word 100 ... 00 (-2^{31}). Two's complementing produces $+2^{31}$ and overflow into sign bit position. TRAP flip-flop is set during ENDE if overflow exists and arithmetic mask is a one
	Set flip-flop CC3 if (S0-S31) is nonzero; otherwise reset CC3	S/CC3 = SGTZ TESTS + ... SGTZ = (S0 + S1 + ... + S31) = N(S0 NFACOMP) + ... TESTS = FALOAD/A PH2 + ... R/CC3 = TESTS + ...	CC3 indicates absolute value is nonzero
	Reset flip-flop CC4	R/CC4 = ...	CC4 is always zero
	Enable clock T8L	T8EN = NT5EN NT11L N(SXADD/1 RW) N(RW REU) N(REU AXRR) NT5EN = RW + ...	T5EN is disabled by signal RW
			Mnemonic: LAW (3B, BB)

(Continued)

Table 3-31. Load Absolute Word Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH2 T8L (Cont.)	Branch to PH10	BRPH10 = FALOAD/A PH2 NOU1 +... S/PH10 = BRPH10 NCLEAR +... R/PH10 = ...	
	Set flip-flop DRQ	S/DRQ = BRPH10 NCLEAR +... R/DRQ = ...	Inhibits transmission of another clock until data release signal from core memory
PH10 DR	ENDE functions	See table 3-18	
			Mnemonic: LAW (3B, BB)

LOAD ABSOLUTE DOUBLEWORD (LAD, 1B, 9B). The LAD instruction loads the absolute value of the effective doubleword into private memory. The absolute value equals the effective doubleword if the sign of the effective doubleword is positive. If the effective doubleword is negative, the absolute value equals the two's complement of the effective doubleword. If the R field of the instruction is even, the most significant half of the absolute value is transferred to private memory register R and the least significant half to private memory register Ru1. If the R field of the instruction word is odd, only the most significant half of the absolute value is transferred to private memory register R. Examples of an LAD with both an even and odd R field are:

Even R Field

ED	1011011...1011	1000110...1100	Before execution
	0100100...0100	0111001...0100	After execution
	Register R	Register Ru1	

Odd R Field

ED	0101111...1000	0101010...1011	Before execution
	0101111...1000		After execution
	Register R		

Overflow. Fixed-point arithmetic overflow occurs if the effective doubleword is -2^{63} (100000...000) since recomplementing produces a positive number too large to be held in two 32-bit registers. Overflow causes a trap to memory location X'43' after execution of LAD if the arithmetic mask is a one. If the arithmetic mask is a zero, the next instruction sequence is executed.

Condition Codes. LAD condition code settings are:

CC1	CC2	CC3	CC4	Absolute Value of ED
X	0	0	0	Zero – no overflow
X	0	1	0	Nonzero – no overflow
X	1	0	0	Overflow

LAD Phase Sequence. LAD preparation phases are the same as the general PREP phases for doubleword instructions as described in paragraph 3-59. Figure 3-134 shows the simplified phase sequence for the instruction during execution and table 3-32 lists the detailed logic sequence during all LAD execution phases.

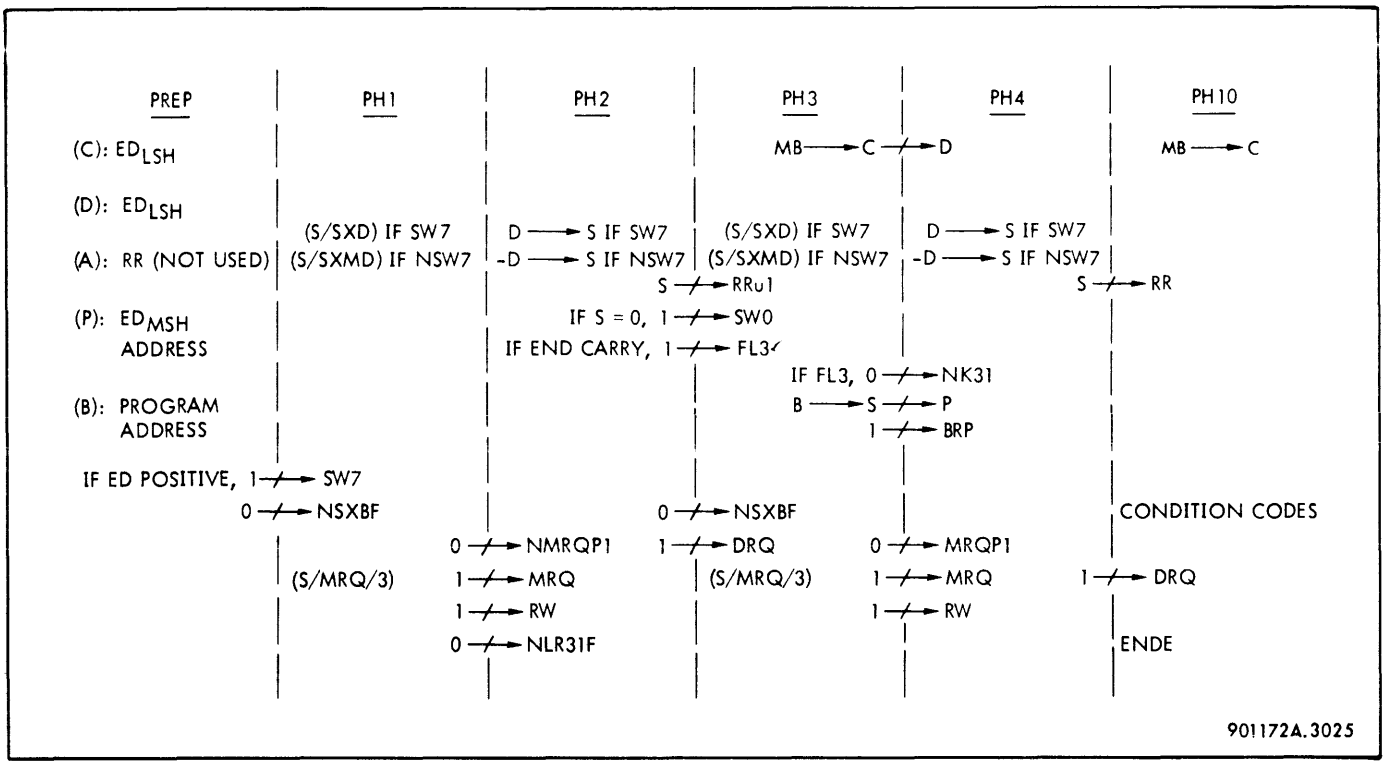


Figure 3-134. Load Absolute Doubleword Phases

Table 3-32. Load Absolute Doubleword Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(C): ED_{LSH}		Least significant half of effective doubleword
	(D): ED_{LSH}		Least significant half of effective doubleword
	(A): RR (not used)		Contents of private memory register R. Not used during this instruction
	(P): ED_{MSH} address		Address of most significant half of effective doubleword
	(B): Program address		Address of next instruction in sequence
	Set flip-flop SW7 if ED positive	S/SW7 = FULAWORDW ND0 PRE/34 + ... FULAWORDW = FALOAD/A NO1 R/SW7 = RESET/A + ...	Flip-flop SW7 stores sign of effective doubleword for computing absolute value in PH2. When SW7 is set, D0 is sign bit of most significant half of doubleword
			Mnemonic: LAD (1B, 9B)

(Continued)

Table 3-32. Load Absolute Doubleword Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PREP (Cont.)	Reset flip-flop NSXBF	S/NXSBF = N(S/SXB) (S/SXB) = FALOAD/A PRE/34 + ... R/NSXBF = ...	Preset logic for B⟶S in PH1
PH1	One clock long		
T5L	(B0-B31)⟶(S0-S31)	SXB = NDIS SXBF + ... SXBF = Set at last clock	Meaningless for this instruction
	If ED is positive, enable signal (S/SXD)	(S/SXD) = FALOAD/A PH1 SW7 + ...	Preset adder for transferring least significant half of effective doubleword to sum bus in PH2. Effective doubleword equals absolute value
	If ED is negative, enable signal (S/SXMD)	(S/SXMD) = FALOAD/A PH1 NSW7 + ...	Preset adder for -D⟶S in PH2. Effective doubleword two's complemented for absolute value
	Set flip-flop MRQ	S/MRQ = (S/MRQ/3) + ... (S/MRQ/3) = FALOAD/A PH1 + ... R/MRQ = ...	Core memory request for most significant half of effective doubleword. Flip-flop DRQ set on next clock
	Reset flip-flop NMRQP1	S/NMRQP1 = N(S/MRQ/3) R/NMRQP1 = ...	Delays setting flip-flop DRQ
	Set flip-flop RW	S/RW = FALOAD/A PH1 + ... R/RW = ...	Prepare to write least significant half of result in private memory register Ru1
	Reset flip-flop NLR31F	S/NLR31F = N(S/LR31) (S/LR31) = FULAD PH1 + ... R/NLR31F = ...	Force a one on private memory address line LR31 during PH2 to select private memory register Ru1
PH2	One clock long		
T8L	If ED is positive, (D0-D31)⟶(S0-S31)⟶(RW0-RW31)	Adder logic set at PH1 clock RWXS/0-RWXS/3 = RW + ... RW = Set at PH1 clock	Transfer absolute value of least significant half of doubleword to private memory register Ru1
	If ED is negative, -(D0-D31) (S0-S31)⟶(RW0-RW31)	Adder logic set at PH1 clock RWXS/0-RWXS/3 = RW + ...	Transfer absolute value of least significant half of doubleword to private memory register Ru1
	Reset flip-flop NSXBF	S/NSXBF = N(S/SXB) (S/SXB) = FALOAD/A PH2 + ... R/NSXBF = ...	Preset logic for B⟶S in PH3
			Mnemonic: LAD (1B, 9B)

(Continued)

Table 3-32. Load Absolute Doubleword Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH2 T8L (Cont.)	Set flip-flop SW0 if (S0-S31) nonzero	S/SW0 = NS0031Z (S/SW0/NZ) + ... NS0031Z = (S0 + S1 + ... + S31) (S/SW0/NZ) = K00HOLD + ... K00HOLD = FALOAD/A PH2 + ... R/SW0 = ...	Sets CC3 in PH4. CC2, CC3, CC4 if set are meaningless
	Set flip-flop FL3 if end carry	S/FL3 = K00 K00HOLD + ... R/FL3 = ...	K00 is end carry; results when effective doubleword is negative and least significant half is 000...000
	Enable clock T8L	T8EN = NT5EN NT11L N(SXADD/1 RW) N(RW REU) N(REU AXRR) NT5EN = RW + ...	T5 is disabled by signal RW
	Set flip-flop DRQ	S/DRQ = MRQP1 + ... R/DRQ = ...	MRQP1 set on previous clock. DRQ inhibits transmission of another clock until data release signal received from core memory
PH3 DR	One clock long (B0-B31) ⟶ (S0-S31) (S15-S31) ⟶/ (P15-P31)	SXB = NDIS SXBF + ... SXBF = Set at PH2 clock PXS = FALOAD/A PH3 + ...	Transfer program address to P-register
	Set flip-flop BRP	S/BRP = FADW/1 PH2 + ... R/BRP = PRE1 NFAIM + INTRAP1 + ...	Signifies that program address is in P-register
	(MB0-MB31) ⟶ (C0-C31) ⟶/ (D0-D31)	CXMB = DG = /DG/ DXC = FALOAD/A PH3 + ...	Transfer most significant half of effective doubleword to D-register
	Set flip-flop MRQ	S/MRQ = (S/MRQ/3) + ... (S/MRQ/3) = FALOAD/A PH3 + ... R/MRQ = ...	Core memory request for next instruction in sequence
	Reset flip-flop NMRQP1	S/MRQP1 = N(S/MRQ/3) R/MRQP1 = ...	Delays setting flip-flop DRQ. DRQ set on next clock
	Set flip-flop RW	S/RW = FALOAD/A PH3 + ... R/RW = ...	Prepare to write most significant half of result into private memory register R
			Mnemonic: LAD (1B, 9B)

(Continued)

Table 3-32. Load Absolute Doubleword Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH3 DR (Cont.)	If ED positive, enable signal (S/SXD)	(S/SXD) = FALOAD/A PH3 SW7 +...	Preset adder for transferring most significant half of effective doubleword to sum bus in PH4. Effective doubleword equals absolute value
	If ED negative, enable signal (S/SXMD)	(S/SXMD) = FALOAD/A PH3 NSW7 +...	Preset adder for -D⟶S in PH4. Most significant half of effective doubleword two's complemented to find absolute value
	Reset flip-flop NK31 if end carry occurred in PH2; if no end carry, set flip-flop NK31 with N(S/K31/1)	S/NK31 = N(S/K31) N(S/SXAMD/1) + N(S/K31/1) (S/K31/1) = K00 (S/K31/3) +... (S/K31/3) = N(FALOAD/A PH3 NFL3) + ... R/NK31 = ...	Occurs if effective doubleword negative. Setting K31 provides a carry to most significant half of effective doubleword complemented in PH4
PH4	One clock long		
T8L	If ED positive (D0-D31)⟶	Adder logic set at PH3 clock	Transfer absolute value of most significant half of doubleword to private memory register R
	(S0-S31)⟶(RW0-RW31)	RWXS/0-RWXS/3 = RW + ... RW = Set at PH3 clock	
	If ED negative -(D0-D31)⟶	Adder logic set at PH1 clock	Transfer absolute value of most significant half of doubleword to private memory register R
	(S0-S31) ⟶(RW0-RW31)	RWXS/0-RWXS/3 = RW + ...	
	Set flip-flop CC2 if arithmetic overflow; otherwise reset CC2	S/CC2 = (S00 ⊕ S0) PROBOVER + ... PROBOVER = FALOAD/A PH2 NO1 +... R/CC2 = PROBOVER + ...	Arithmetic overflow during LAD when effective doubleword 100...00 (-2^{63}). Two's complementing produces $+2^{63}$ and overflow into sign bit position. TRAP flip-flop is set during ENDE if overflow exists and arithmetic mask is a one
	Set flip-flop CC3 if (S0-S31) nonzero; otherwise reset CC3	S/CC3 = SGTZ TESTS + ... SGTZ = (NS3263Z + S0 + S1 +... + S31) NS0 +... TESTS = FALOAD/A PH4 +... NS3263Z= SW0 +... R/CC3 = TESTS +...	CC3 indicates absolute value is nonzero
	Reset flip-flop CC4	R/CC4 = ...	CC4 always zero for LAD
			Mnemonic: LAD (1B, 9B)

(Continued)

Table 3-32. Load Absolute Doubleword Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH4 T8L (Cont.)	Enable clock T8L	T8EN = NT5EN NT11L N(SXADD/1 RW) N(RW REU) N(REU AXRR) NT5EN = RW + ...	T5EN is disabled by signal RW
	Branch to PH10	BRPH10 = FALOAD/A PH2 NOU1 + ... S/PH10 = BRPH10 NCLEAR + ... R/PH10 = ...	
	Set flip-flop DRQ	S/DRQ = BRPH10 NCLEAR + MRQP1 + ... R/DRQ = ...	Inhibits transmission of another clock until data release signal received from core memory
PH10 DR	ENDE functions	See table 3-18	
			Mnemonic: LAD (1B, 9B)

3-62. Family of Store Instructions (FASTORE)

STORE BYTE (STB; 75, F5). The STB instruction stores the least significant byte (bit positions 24 through 31) of private memory register R into the effective byte location.

Store Byte Phase Sequences. Preparation phases for the STB instruction are the same as the general PREP phases for byte instructions, paragraph 3-59. Table 3-33 lists the detailed logic sequence during all STB execution phases.

Table 3-33. Store Byte Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(A) : RR, byte aligned		Contents of private memory register R, with least significant byte shifted to byte position of the effective byte
	(B) : Program address		Address of next instruction in sequence
	(P) : Effective byte address		
	Set flip-flop MRQ and flip-flop MBXS	S/MRQ = (S/MBXS) + ... (S/MBXS) = FASTORE PRE/34 + ... FASTORE = FASTORE/3 + FUXW/1 FASTORE/3 = NO6 O5 NO4 O3 R/MRQ = ...	Prepare to store byte in effective byte location
		S/MBXS = (S/MBXS) R/MBXS = ...	Prepare to gate byte from sum bus to memory bus
	Enable signal (S/SXA)	(S/SXA) = FASTORE PRE/34 + ...	Preset adder for A ⟶ S in PH1
	Set flip-flop DRQ	S/DRQ = (S/MBXS) + ... R/DRQ = ...	Inhibits transmission of another clock until data release signal received from core memory
PH1 DR	Sustained until data release		
	(A0-A31) ⟶ (S0-S31)	Adder logic set at last PREP clock	
	(S0-S7) ⟶ (MB0-MB7) OR	S/MBXS/0 = NP32 NP33 FABYTE EXC + ... R/MBXS/0 = DRQ FABYTE = O1 O2 O3	Byte from bit positions 24 through 31 of private memory register R transferred to effective byte location
	(S8-S15) ⟶ (MB8-MB15) OR	S/MBXS/1 = NP32 P33 FABYTE EXC + ... R/MBXS/1 = DRQ	
	(S16-S23) ⟶ (MB16-MB23) OR	S/MBXS/2 = P32 NP33 FABYTE EXC + ... R/MBXS/2 = DRQ	
	(S24-S31) ⟶ (MB24-MB31)	S/MBXS/3 = P32 P33 FABYTE EXC + ... R/MBXS/3 = DRQ	
			Mnemonic: STB (75, F5)

(Continued)

Table 3-33. Store Byte Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH1 DR (Cont.)	Branch to PH9	BRPH9 = FASTORE NFASTORE/1 PH1 + ... FASTORE/1 = FUSTD + FUXW/1 S/PH9 = BRPH9 NCLEAR + ... R/PH9 = ...	
PH9 T5L	One clock long		
	(B0-B31)⟶(S0-S31)	SXB = PXSXB NDIS + ...	Transfer program address to P-register
	(S15-S31)⟶/(P15-P31)	PXSXB = PH9 NFAFL NFAMDS	
		PXS = PXSXB + ...	
	Set flip-flop BRP	S/BRP = PXSXB + ... R/BRP = PRE1 NFAIM + ...	Signifies that program address is in the P-register
	Set flip-flop MRQ	S/MRQ = (S/MRQ/3) + ... (S/MRQ/3) = NINTRAP2 PXSXB + ... R/MRQ = ...	Core memory request for next instruction in sequence
	Set flip-flop DRQ	S/DRQ = BRPH10 + ... R/DRQ = ...	Inhibits transmission of another clock until data release signal received from core memory
	Enable signal (S/SXA)	(S/SXA) = FASTORE PH9 + ...	Preset adder for A⟶S in PH10
PH10 DR	Sustained until data release (A0-A31)⟶(S0-S31) ENDE functions	Adder logic set at PH9 clock	Not used for STB
			Mnemonic: STB (75, F5)

STORE HALFWORD (STH; 55, D5). The STH instruction stores the contents of bit positions 16 through 31 of the private memory register specified in the R field of the instruction in the effective halfword location. If the information in register R exceeds halfword data limits, condition code flip-flop CC2 is set to one; otherwise, CC2 is reset to zero.

Store Halfword Phase Sequences. Preparation phases for the STH instruction are the same as the general PREP phases for halfword instructions, paragraph 3-59. Table 3-34 lists the detailed logic sequence during all STH execution phases.

STORE WORD (STW; 35, B5). The STW instruction stores the contents of the private memory register specified in the R field of the instruction into the effective word location.

Store Word Phase Sequences. Preparation phases for the STW instruction are the same as the general PREP phases for word instructions, paragraph 3-59. Table 3-35 lists the detailed logic sequence during all STW execution phases.

Table 3-34. Store Halfword Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(A) : RR, halfword aligned		Contents of private memory register R, with bits 16 thru 31 shifted to halfword position of the effective halfword
	(B) : Program address		Address of next instruction in sequence
	(P) : Effective halfword address		
	Set flip-flop MRQ	S/MRQ = (S/MBXS) + ...	Prepare to store halfword in effective halfword location
	Enable signal (S/MBXS)	(S/MBXS) = FASTORE PRE/34 + ... FASTORE = NO6 O5 NO4 O3 R/MRQ = ... S/MBXS = (S/MBXS) R/MBXS = ...	Prepare to gate halfword from sum bus to memory bus
	Set flip-flop DRQ	S/DRQ = (S/MBXS) + ... R/DRQ = ...	Data request, inhibiting transmission of another clock until data release received from memory
	Reset flip-flop NAXRR	S/NAXRR = N(S/AXRR) (S/AXRR) = FASTORE PRE/34 NO2 + ... R/NAXRR = ...	Prepare to read from private memory register R
	Enable signal (S/SXA)	(S/SXA) = FASTORE PRE/34 + ...	Preset adder for A⟶S in PH1
PH1	Sustained until data release		
DR	(A0-A31)⟶(S0-S31)	Adder preset at last PREP clock	Store halfword in effective halfword location in core memory
	(S0-S31)⟶(MB0-MB31)	MBXS set at last PREP clock	
	(RR0-RR31)⟶(A0-A31)	AXRR set at last PREP clock	Read private memory register R into A-register
	Branch to PH9	BRPH9 = FASTORE NFASTORE/1 PH1 + ... S/PH9 = BRPH9 NCLEAR + ... R/PH9 = ...	
			Mnemonic: STH (55, D5)

(Continued)

Table 3-34. Store Halfword Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH9	One clock long		
T5L	(B0-B31)⟶(S0-S31)	SXB = PXSXB NDIS + ... PXSXB = PH9 NFAFL NFAMDS	Transfer program address from B-register to P-register for access of next instruction
	(S15-S31)⟶(P15-P31)	PXS = PXSXB + ...	
	Set flip-flop BRP	S/BRP = PXSXB + ... R/BRP = PRE1 NFAIM + ...	Signifies that program address is in P-register
	Set flip-flop MRQ	S/MRQ = (S/MRQ/2) + ... (S/MRQ/2) = NINTRAP2 PXSXB R/MRQ = ...	Request for next instruction in sequence
	Set flip-flop DRQ	S/DRQ = (S/MRQ/2) + ... R/DRQ = ...	Data request, inhibiting transmission of another clock until data release received from core memory
	Enable signal (S/SXA)	(S/SXA) = FASTORE PH9 + ...	Preset adder for A⟶S in PH10
PH10	Sustained until data release		
DR	(A0-A31)⟶(S0-S31)	Adder preset in PH9	Place contents of private memory register R on sum bus for data limit check
	Set condition code flip-flop CC2 if S0-S16 ≠ 0 or all 1's	S/CC2 = N(S0016Z + S0016W) (FUSTH ENDE) + ...	If most significant halfword in private memory word does not contain all zeros or all ones, the halfword data limits are exceeded and CC2 must be set. All zeros or all ones represent the sign extension of number in bit positions 16 through 31
	Reset flip-flop CC2 if set conditions are not met	R/CC2 = (R/CC2/1) + ... (R/CC2/1) = FUSTH ENDE + ...	Reset CC2 if data limits are not exceeded
	ENDE functions		
			Mnemonic: STH (55, D5)

Table 3-35. Store Word Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(A) : RR		Contents of private memory register R
	(B) : Program address		Address of next instruction in sequence
	(P) : Effective address		
	Set flip-flop MRQ	S/MRQ = (S/MBXS) + ...	Prepare to store word in effective word location
	Enable signal (S/MBXS)	(S/MBXS) = FASTORE PRE/34 + ...	
		FASTORE = NO6 O5 NO4 O3	
		R/MRQ = ...	
	Set flip-flop DRQ	S/DRQ = (S/MBXS) + ...	Data request, inhibiting transmission of another clock until data release received from core memory
		R/DRQ = ...	
	Enable signal (S/SXA)	(S/SXA) = FASTORE PRE/34 + ...	Preset adder for A⟶S in PH1
PH1	Sustained until data release		
DR	(A0-A31)⟶(S0-S31)	Adder logic set at last PREP clock	Store word in core memory at effective location
	(S0-S31)⟶(MB0-MB31)	MBXS = Set at last PREP clock	
	Branch to PH9	BRPH9 = FASTORE NFASTORE/1 PH1 + ...	
		S/PH9 = BRPH9 NCLEAR + ...	
		R/PH9 = ...	
PH9	One clock long		
T5L	(B0-B31)⟶(S0-S31)	SXB = PXSXB + ...	Transfer program address to P-register for access of next instruction
		PXSXB = PH9 NFAFL NFAMDS	
	(S15-S31)⟶(P15-P31)	PXS = PXSXB + ...	
	Set flip-flop BRP	S/BRP = PXSXB + ...	Signifies that program address is in P-register
		R/BRP = PRE1 NFAIM + ...	
	Set flip-flop MRQ	S/MRQ = (S/MRQ/2) + ...	Request for next instruction in sequence
		(S/MRQ/2) = NINTRAP2 PXSXB + ...	
		R/MRQ = ...	
	Set flip-flop DRQ	S/DRQ = (S/DRQ) + ...	Data request, inhibiting transmission of another clock until data release received from core memory
		(S/DRQ) = (S/MRQ/2) + ...	
		R/DRQ = ...	
			Mnemonic: STW (35, B5)

(Continued)

Table 3-35. Store Word Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH10 DR	Sustained until data release ENDE functions	See table 3-18	
			Mnemonic: STW (35, B5)

STORE DOUBLEWORD (STD; 15, 95). The STD instruction stores the contents of private memory register R into the 32 high-order bit positions of the effective doubleword location. The contents of private memory register Ru1 are stored in the 32 low-order bit positions of the effective doubleword location.

Store Doubleword Phase Sequences. Preparation phases for STD are the same as the general PREP phases for doubleword instructions, paragraph 3-59. Figure 3-135 shows the simplified phase sequence for the STD instruction during execution. Table 3-36 lists the detailed logic sequence during all STD execution phases.

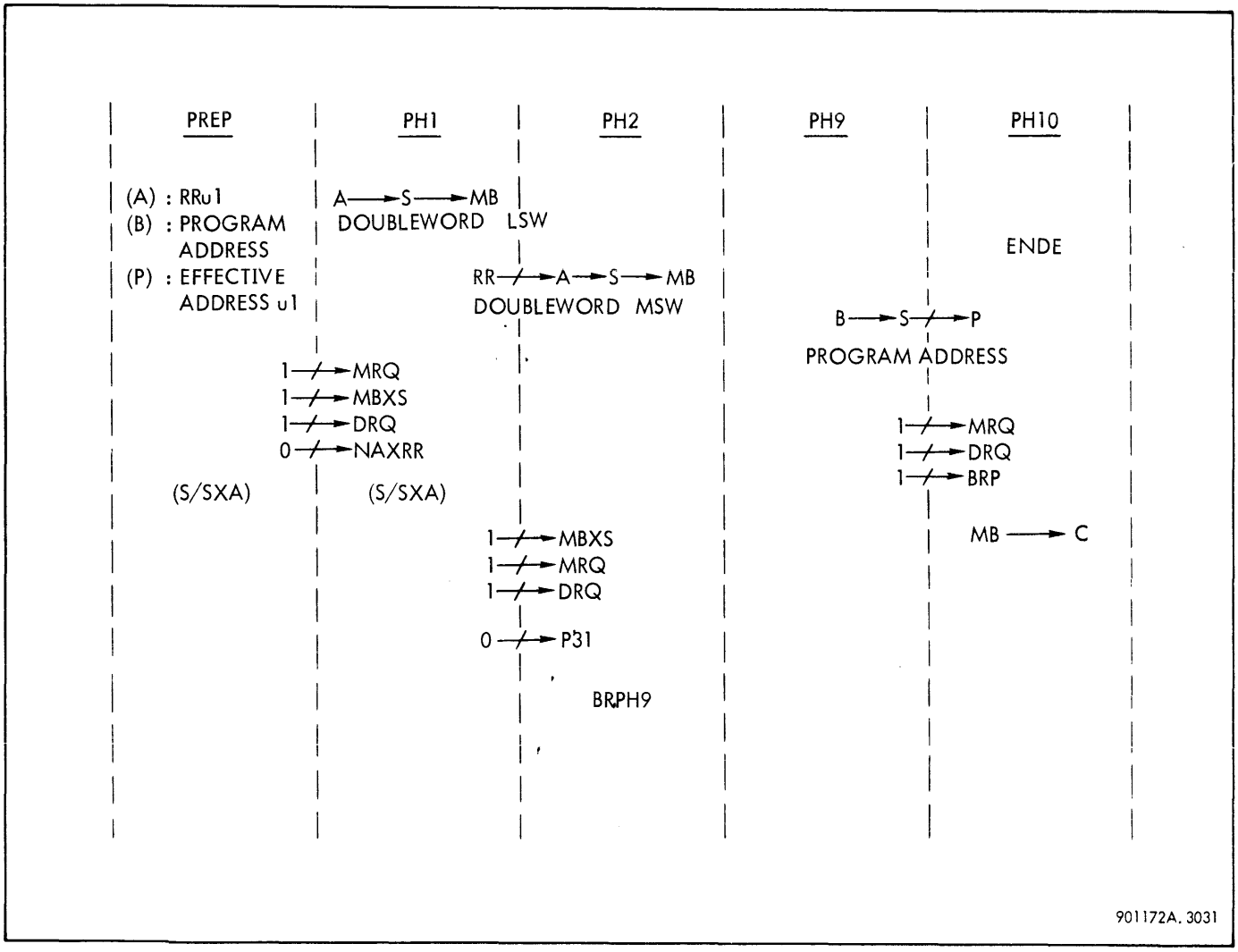


Figure 3-135. Store Doubleword Phases

Table 3-36. Store Doubleword Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(A) : RRu1		Contents of private memory register Ru1
	(B) : Program address		Address of next instruction in sequence
	(P) : Effective address		Least significant word location of effective doubleword location
	Set flip-flop MRQ Enable signal (S/MBXS)	S/MRQ = (S/MBXS) + ... (S/MBXS) = FASTORE PRE/34 + ... FASTORE = FASTORE/3 + FUXW/1 FASTORE/1 = FUSTD + FUXW/1 FUSTD = OU1 FASTORE/3 FASTORE/3 = NO6 O5 NO4 O3 R/MRQ = ...	Prepare to store least significant word in least significant word location
	Set flip-flop DRQ	S/DRQ = (S/MBXS) + ... R/DRQ = ...	Inhibits transmission of another clock until data release received from core memory
	Enable signal (S/SXA)	(S/SXA) = FASTORE PRE/34 + ...	Preset adder for A ⟶ S in PH1
	Reset flip-flop NAXRR	S/NAXRR = N(S/AXRR) (S/AXRR) = FASTORE PRE/34 NO2 + ... R/NAXRR = ...	Preset for private memory register R ⟶/⟶ A-register in PH1
PH1	Sustained until data release		
DR	(A0-A31) ⟶ (S0-S31)	Adder logic set at last PREP clock	Store contents of private memory register Ru1 in 32 low-order bits of effective doubleword location
	(S0-S31) ⟶/⟶ (MB0-MB31)	MBXS = Set at last PREP clock	
	(RR0-RR31) ⟶/⟶ (A0-A31)	AXRR = Set at last PREP clock	Transfer contents of private memory register R to A-register
	Enable signal (S/SXA)	(S/SXA) = FASTORE/1 PH1 + ... FASTORE/1 = FUSTD + ... FUSTD = OU1 FASTORE/3 FASTORE/3 = O3 NO4 O5 NO6	Preset adder for A ⟶ S in PH2
	Set flip-flop MRQ	S/MRQ = (S/MBXS) + ...	
	Enable signal (S/MBXS)	(S/MBXS) = FASTORE/1 PH1 + ... R/MRQ = ...	Request for core memory cycle
	Set flip-flop DRQ	S/DRQ = (S/MBXS) R/DRQ = ...	Data request, inhibiting transmission of another clock until data release received from memory
			Mnemonic: STD (15, 95)

(Continued)

Table 3-36. Store Doubleword Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH1 DR (Cont.)	Reset flip-flop P31	PDC31 = FASTORE PH1 OU1	Decrement P-register by 1 to obtain most significant word location
PH2 DR	Sustained until data release (A0-A31)⟶(S0-S31) (S0-S31)⟶(MB0-MB31) Branch to PH9	Adder logic set at PH1 clock MBXS = Set at PH1 clock BRPH9 = FASTORE PH2 + ...	Store contents of private memory register R in 32 high-order bits of effective doubleword location
PH9 T5L	One clock long (B0-B31)⟶(S0-S31) (S15-S31)⟶(P15-P31) Set flip-flop BRP Set flip-flop MRQ Set flip-flop DRQ	SXB = PXSXB + ... PXSXB = PH9 NFAFL NFAMDS PXS = PXSXB + ... S/BRP = PXSXB + ... R/BRP = PRE1 NFAIM + ... S/MRQ = (S/MRQ/2) + ... (S/MRQ/2) = NINTRAP2 PXSXB R/MRQ = ... S/DRQ = (S/MRQ/2) + ... R/DRQ = ...	Transfer program address to P-register for access of next instruction Signifies that program address is in P-register Request for next instruction in sequence Data request, inhibiting transmission of another clock until data release received from memory
PH10 DR	Sustained until data release ENDE functions	See table 3-18	
			Mnemonic: STD (15, 95)

STORE CONDITIONS AND FLOATING CONTROL (STCF; 74, F4). The STCF instruction stores the current condition code and the current values of the floating significance (FS), floating zero (FZ), and floating normalize (FN) bits of the program status doubleword in the effective byte location. CC1 through CC4 are stored in bit positions 0 through 3 of the effective byte location. FS, FZ, and FN are stored in bit positions 5, 6, and 7, respectively. Bit position 4 is a zero.

Store Conditions and Floating Control Phase Sequences. Preparation phases for the STCF instruction are the same as the general PREP phases for byte instructions, paragraph 3-59. Table 3-37 lists the detailed logic sequence during all execution phases of the instruction.

ADD WORD TO MEMORY (AWM; 66, E6). The AWM instruction adds the contents of register R to the effective word and stores the sum in the effective word location.

Condition Codes. If the result in the effective word location is zero, the condition codes are set to XX00. If the result is nonzero and positive, the condition codes are set to XX10. A negative result produces condition code settings of XX01. Flip-flop CC2 is set if fixed-point overflow occurs during the addition. Flip-flop CC1 is set if there is a carry from bit position 0.

Trap Conditions. A trap to memory location X'43' occurs if there is fixed-point overflow and the fixed-point arithmetic mask bit is a one. The result in the effective memory location remains unchanged. If overflow occurs and the mask bit is a zero, the next instruction in sequence is executed.

Add Word to Memory Phase Sequences. Preparation phases for the AWM instruction are the same as the general PREP phases for word instructions, paragraph 3-59. Table 3-38 lists the detailed logic sequence during all AWM execution phases.

Table 3-37. Store Conditions and Floating Control

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(A) : Byte-aligned CC, FS, FZ, and FN bits of program status doubleword		
	(B) : Program address		Address of next instruction in sequence
	(P): Effective byte address		
	Disable signal AXRRINH	AXRRINH = FASTORE PRE3 OL4 + ... FASTORE = NO6 O5 NO4 O3	Transfer of R→A is an automatic function in PRE3
	Enable signal AXFC	AXFC = FASTORE PRE4 NBCZ	Enable transfer of condition code and floating control bits to A-register
	(CC1-CC4)→(A24-A27)	S/A24 = CC1 AXFC + ... S/A25 = CC2 AXFC + ... S/A26 = CC3 AXFC + ... S/A27 = CC4 AXFC + ...	Transfer CC and FC
	FS→A29 FZ→A30 FNF→A31	S/A29 = FS AXFC + ... S/A30 = FZ AXFC + ... S/A31 = FNF AXFC + ...	
	Left align A-register	AXAL8 = FASTORE PRE4 NBCZ + ...	Move condition code and floating control bits left the number of bytes specified by index register. Byte count in BC0 and BC1 is decremented with each shift
	Set flip-flop MRQ	S/MRQ = (S/MBXS) + ...	
	Enable signal (S/MBXS)	(S/MBXS) = FASTORE PRE/34 + ... R/MRQ = ...	Request for core memory cycle
	Set flip-flop DRQ	S/DRQ = (S/MBXS) + ... R/DRQ = ...	Data request, inhibiting transmission of another clock until data release received from core memory
	Enable signal (S/SXA)	(S/SXA) = FASTORE PRE/34 + ...	Preset adder for A→S in PH1
PH1	Sustained until data release		
DR	(A0-A31)→(S0-S31)	Adder logic set at last PREP clock	Store condition code and FS, FZ, and FN in core memory at effective byte location
	(S0-S31)→(MB0-MB31)	MBXS = Set at last PREP clock	
	Branch to PH9	BRPH9 = FASTORE NFASTORE/1 PH1 + ... S/PH9 = BRPH9 NCLEAR + ... R/PH9 = ...	
			Mnemonic: STCF (74, F4)

(Continued)

Table 3-37. Store Conditions and Floating Control (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH9 T5L	One clock long (B0-B31)⟶(S0-S31)	SXB = PXSXB + ... PXSXB = PH9 NFAFL NFAMDS	Transfer program address from B-register to P-register for access of next instruction
	(S15-S31)⟶(P15-P31)	PXS = PXSXB + ...	
	Set flip-flop BRP	S/BRP = PXSXB + ... R/BRP = PRE1 NFAIM + ...	Signifies that program address is in P-register
	Set flip-flop MRQ	S/MRQ = (S/MRQ/2) + ... (S/MRQ/2) = NINTRAP2 PXSXB R/MRQ = ...	Request for next instruction in sequence
	Set flip-flop DRQ	S/DRQ = (S/MRQ/2) + ... R/DRQ = ...	Data request, inhibiting transmission of another clock until data release received from core memory
PH10 DR	Sustained until data release ENDE functions	See table 3-18	
			Mnemonic: STCF (74, F4)

Table 3-38. Add Word to Memory Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(A) : RR		Contents of private memory register R
	(C) : EW		Effective word
	(D) : EW		Effective word
	(B) : Program address		Address of next instruction in sequence
	(P): Effective address		Address of effective word
	Enable signal (S/SXAPD)	(S/SXAPD) = FAADD PRE/34 + ... FAADD = FUAWM PRE3 + ... FUAWM = OU6 OL6	Preset adder for A + D ⟶S in PH1
PH1 T8L	One clock long (A0-A31) + (D0-D31)⟶(S0-S31)	Adder logic set at last PREP clock	Add the contents of private memory register R and effective word and transfer result to the A-register
	⟶(A0-A31)	AXS = FUAWM PH1 + ...	
			Mnemonic: AWM (66, E6)

(Continued)

Table 3-38. Add Word to Memory Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH1 T8L (Cont.)	Enable signal (S/SXA)	(S/SXA) = FASTORE/1 PH1 + ... FASTORE/1 = FUXW/1 + ... FUXW/1 = NPREP FUAWM + ...	Preset adder for A⟶S in PH2
	Set flip-flop MRQ	S/MRQ = (S/MBXS) + ... (S/MBXS) = FASTORE/1 PH1 + ... R/MRQ = ...	Prepare to write result into effective memory location
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR + ... (S/DRQ) = (S/MBXS) + ... R/DRQ = ...	Inhibits transmission of another clock until data release signal received from core memory
	Set flip-flop CC3 if result in effective memory location will be positive and nonzero; otherwise reset CC3	S/CC3 = SGTZ TESTS + ... SGTZ = (S0 + S1 + ... + S31) NS0 NFACOMP + ... TESTS = FUAWM PH1 + ... R/CC3 = TESTS + ...	
	Set flip-flop CC4 if result in effective memory location will be negative; otherwise reset CC4	S/CC4 = (S/CC4/2) TESTS + ... (S/CC4/2) = S0 NFACOMP + ... R/CC4 = TESTS + ...	
	Set flip-flop CC2 if overflow resulted from the addition; otherwise reset CC2	S/CC2 = (S00 ⊕ S0) PROBOVER + ... PROBOVER = FUAWM PH1 + ... R/CC2 = PROBOVER + ...	Arithmetic overflow occurs when two numbers of like signs are added and their sum cannot be held in 32 bits
	Set flip-flop OVERIND/1	S/OVERIND/1 = PROBOVER + ... R/OVERIND/1 = CLEAR	Setting OVERIND/1 enables trap if overflow occurs and mask bit is equal to a one. Trap is set during ENDE
	Set flip-flop CC1 if end carry from result; otherwise reset CC1	S/CC1 = K00 CC1XK00 + ... CC1XK00 = FUAWM PH1 + ... R/CC1 = CC1XK00 + ...	K00 is end carry from the addition
	Enable clock T8	T8EN = NT5EN NT11EN N(SXADD/1 RW) N(RW REU) N(REU AXRR) NT5EN = RW + ...	
PH2 DR	Sustained until data release (A0-A31)⟶(S0-S31) ⟶(MB0-MB31)	Adder logic set at PH1 clock MBXS = Set at PH1 clock	Write results of addition into effective memory location
			Mnemonic: AWM (66, E6)

(Continued)

Table 3-38. Add Word to Memory Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH2 DR (Cont.)	Branch to PH9	BRPH9 = FASTORE PH2 + ... FASTORE = FUXW/1 + ... FUXW/1 = NPREP FUAWM + ... S/PH9 = BRPH9 NCLEAR + ... R/PH9 = ...	
PH9 T5L	One clock long (B0-B31)⟶(S0-S31) (S15-S31)⟶(P15-P31)	SXB = PXSXB NDIS + ... PXSXB = PH9 NFAFL NFAMDS PXS = PXSXB + ...	Transfer program address to P-register
	Set flip-flop BRP	S/BRP = PXSXB + ... R/BRP = PRE1 NFAIM + ...	Signifies that program address is in the P-register
	Enable signal (S/SXA)	(S/SXA) = FASTORE PH9 + ...	Preset adder for A⟶S in PH10. (Not used for AWM instruction.)
	Set flip-flop MRQ	S/MRQ = (S/MRQ/3) + ... (S/MRQ/3) = NINTRAP2 PXSXB + ... R/MRQ = ...	Core memory request for next instruction in sequence
	Set flip-flop DRQ	S/DRQ = BRPH10 + ... R/DRQ = ...	Inhibits transmission of another clock until data release signal received from core memory
PH10 DR	Sustained until data release ENDE functions	See table 3-18	
			Mnemonic: AWM (66, E6)

EXCHANGE WORD (XW; 46, C6). The XW instruction exchanges the contents of private memory register R with the contents of the effective word location.

Condition Codes. If the result in private memory register R is zero, the condition codes are set to XX00. If the result is nonzero and positive, the condition codes are set to XX10. A negative result produces condition code settings of XX01.

Exchange Word Phase Sequences. Preparation phases for the XW instruction are the same as the general PREP phases for word instructions, paragraph 3-59. Table 3-39 lists the detailed logic sequence during all XW execution phases.

Table 3-39. Exchange Word Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(A) : RR		Contents of private memory register R
	(C) : EW		Effective word
	(D) : EW		Effective word
	(B) : Program address		Address of next instruction in sequence
	(P) : Effective address		Address of effective word
	Enable signal (S/SXD)	(S/SXD) = FUXW PRE3 + ...	Preset adder for D⟶S in PH1
		FUXW = OU4 OL6	
	Set flip-flop RW	S/RW = FUXW NANLZ PRE3 + ...	Prepare to write effective word into private memory register R
		R/RW = ...	
PH1	One clock long		
T8L	(D0-D31)⟶(S0-S31)⟶	Adder logic set at last PREP clock	Write effective word into private memory register R
	(RW0-RW31)	RWXS/0-RWXS/3 = RW + ...	
		RW = Set at last PREP clock	
	Enable signal (S/SXA)	(S/SXA) = FASTORE/1 PH1 + ...	Preset adder for A⟶S in PH2
		FASTORE/1 = FUSTD FUXW/1	
		FUXW/1 = NPREP FUXW + ...	
	Set flip-flop MRQ	S/MRQ = (S/MBXS) + ...	Prepare to write contents of private memory register R into effective memory location
		(S/MBXS) = FASTORE/1 PH1 + ...	
		R/MRQ = ...	
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR + ...	Inhibits transmission of another clock until data release signal received from core memory
		(S/DRQ) = (S/MBXS) + ...	
		R/DRQ = ...	
	Set flip-flop CC3 if result in private memory register R will be positive and nonzero; otherwise reset CC3	S/CC3 = SGTZ TESTS + ...	
		SGTZ = (S0 + S1 + ... + S31) NS0 NFACOMP + ...	
		TESTS = FUXW PH1 + ...	
		R/CC3 = TESTS + ...	
	Set flip-flop CC4 if result in private memory register R will be negative; otherwise reset CC4	S/CC4 = (S/CC4/2) TESTS + ...	
		(S/CC4/2) = S0 NFACOMP + ...	
		R/CC4 = TESTS + ...	
	Enable clock T8	T8EN = NT5EN NT11EN N(SXADD/1 RW) N(RW REU) N(REU AXRR)	
		NT5EN = RW + ...	
			Mnemonic: XW (46, C6)

(Continued)

Table 3-39. Exchange Word Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH2 DR	Sustained until data release (A0-A31)⟶(S0-S31)⟶ (MB0-MB31) Branch to PH9	Adder logic set at PH1 clock MBXS = Set at PH1 clock BRPH9 = FASTORE PH2 + ... FASTORE = FUXW/1 + ... FUXW/1 = NPREP FUXW + ... S/PH9 = BRPH9 NCLEAR + ... R/PH9 = ...	Write contents of private memory register R into effective memory location
PH9 T5L	One clock long (B0-B31)⟶(S0-S31) (S15-S31)⟶/⟶(P15-P31) Set flip-flop BRP Enable signal (S/SXA) Set flip-flop MRQ Set flip-flop DRQ	SXB = PXSXB NDIS + ... PXSXB = PH9 NFAFL NFAMDS PXS = PXSXB + ... S/BRP = PXSXB + ... R/BRP = PRE1 NFAIM + ... (S/SXA) = FASTORE PH9 + ... S/MRQ = (S/MRQ/3) + ... (S/MRQ/3) = PH9 NFAFL NFAMDS R/MRQ = ... S/DRQ = BRPH10 + ... R/DRQ = ...	Transfer program address to P-register Signifies that program address is in the P-register Preset adder for A⟶S in PH10. (Not used for XW instruction.) Core memory request for next instruction in sequence Inhibits transmission of another clock until data release signal received from core memory
PH10 DR	Sustained until data release ENDE functions	See table 3-18	
			Mnemonic: XW (46, C6)

3-63 Family of Selective Instructions (FASEL)

LOAD SELECTIVE (LS; 4A, CA). The LS instruction loads the effective word into private memory register R using private memory register Ru1 as a mask.

General. If the R field of the instruction word is even, the instruction operates as follows: If a bit in private memory register Ru1 is a one, the corresponding bit in the effective word is loaded into the same bit position in private memory register R. If the bit is a zero, the corresponding bit in R remains unchanged. Logically, the operation is as follows, where n is any bit position:

$$R_n = EW_n \; Ru1_n + R_n \; NRu1_n$$

- R_n: Result in bit position n of R register
- $EW_n \; Ru1_n$: Mask bit = 1
- $R_n \; NRu1_n$: Mask bit = 0

If the R field of the instruction word is odd, the instruction AND's the effective word and the contents of private memory register R and loads the result back into R. Logically, for every n bit position:

$$R_n = EW_n \; R_n$$

- R_n: Result in bit position n of R-register

Examples. Examples of LS with both an even and odd R field are:

Even R Field

```
EW   00001111XXXXXXXXXXXXXXXXXXXXXXXX
Ru1  00110011XXXXXXXXXXXXXXXXXXXXXXXX   Before
R    01010101XXXXXXXXXXXXXXXXXXXXXXXX   execution
-------------------------------------------------
R    01000111XXXXXXXXXXXXXXXXXXXXXXXX   After
                                        execution
```

Odd R Field

```
EW   00110101XXXXXXXXXXXXXXXXXXXXXXXX
R    01010110XXXXXXXXXXXXXXXXXXXXXXXX   Before
                                        execution
-------------------------------------------------
R    00010100XXXXXXXXXXXXXXXXXXXXXXXX   After
                                        execution
```

Load Selective Examples

Condition Codes. If the result in the R-register is zero, the condition codes are set to XX00. If the result is negative, the condition codes are set to XX01. A positive result produces condition code settings of XX10.

Load Selective Phase Sequence. Preparation phases for the LS instruction are the same as the general PREP phases for word instructions, paragraph 3-59. Figure 3-136 shows the simplified phase sequence for the LS instruction during execution. Table 3-40 lists the detailed logic sequence during all LS execution phases.

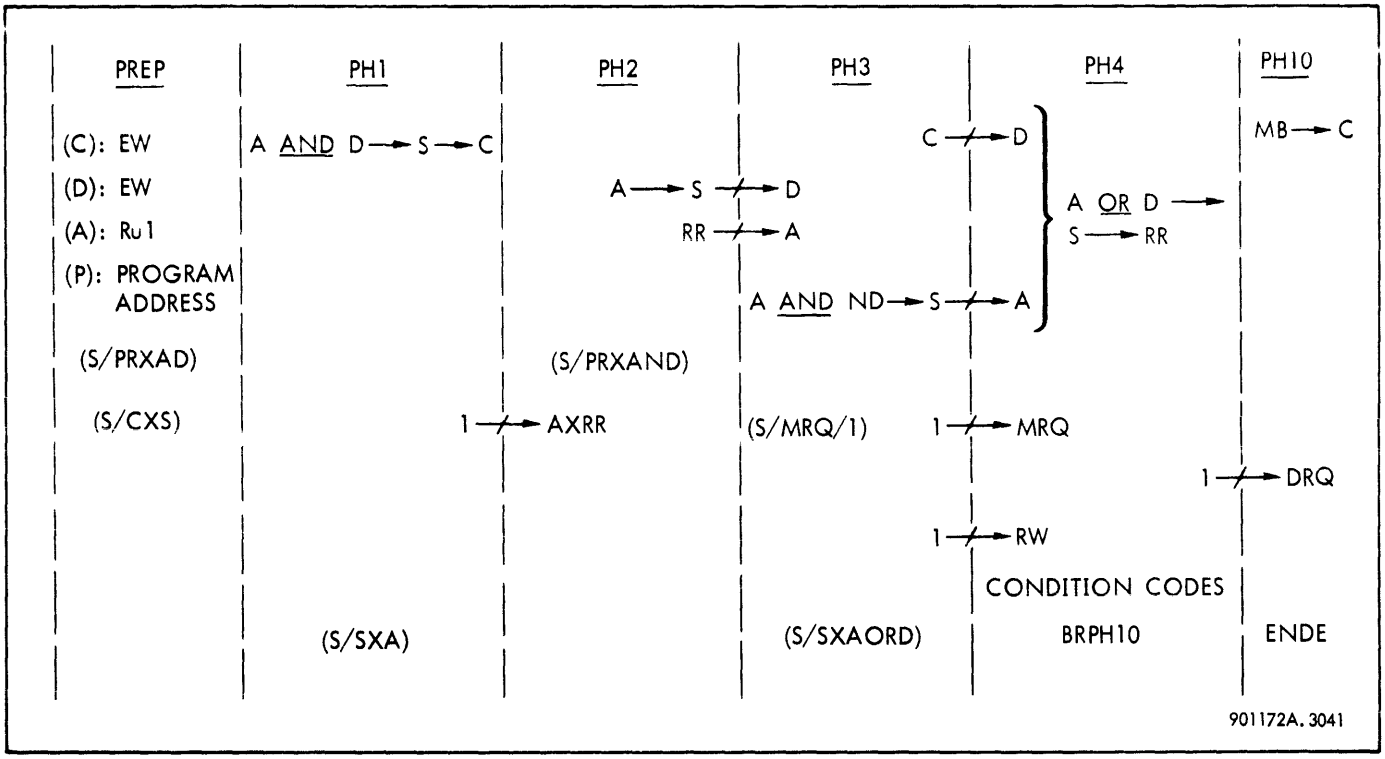


Figure 3-136. Load Selective Phases

Table 3-40. Load Selective Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(C) : EW		Effective word
	(D) : EW		Effective word
	(A) : RRu1		Private memory register Ru1 holds mask
	(P) : Program address		Next instruction in sequence
	Enable signal (S/PRXAD)	(S/PRXAD) = FASEL PRE3 NOL7 + ...	Preset adder for A AND D → S in PH1
	Enable signal (S/CXS)	(S/CXS) = FASEL PRE3 + ...	Preset adder for S → C in PH1
			Mnemonic: LS (4A, CA)

(Continued)

Table 3-40. Load Selective Sequence (Cont.)

Phase	Function Performed	Signal Involved	Comments
PH1	One clock long		
T5L	(A0-A31) AND (D0-D31) ⟶ (S0-S31) ⟶ (C0-C31)	Adder logic set at last PREP clock CXS = Set at last PREP clock	Effective word ANDed with mask and temporarily stored in C-register
	Enable signal (S/SXA)	(S/SXA) = FASEL PH1 + ...	Preset adder logic for A⟶S in PH2
	Set flip-flop AXRR	S/AXRR = FASEL PH1 + ... R/AXRR = ...	Preset for transfer of private memory register R⟶/⟶A in PH2
PH2	One clock long		
T5L	(A0-A31) ⟶ (S0-S31) ⟶/⟶ (D0-D31)	Adder logic set at PH1 clock DXS = FASEL PH2 + ...	Store private memory register Ru1 contents in D-register
	(RR0-RR31) ⟶/⟶ (A0-A31)	AXRR = Set at PH1 clock	Store private memory register R contents in A-register
	Enable signal (S/PRXAND)	(S/PRXAND) = FASEL PH2 OLA + ...	Preset for A AND ND⟶S in PH3
PH3	One clock long		
T5L	(A0-A31) AND (ND0-ND31) ⟶ (S0-S31) ⟶/⟶ (A0-A31)	Adder logic set at PH2 clock AXS = FASEL PH3 + ...	AND contents of private memory register R and one's complement of private memory register Ru1. If R = Ru1, A ND = 0 and $R_n = EW_n\ R_n$. Otherwise, $R_n = EW_n\ Ru1_n + R_n\ NRu1_n$
	(C0-C31) ⟶/⟶ (D0-D31)	DXC = FASEL PH3 + ...	EW Ru1⟶/⟶D-register in preparation for PH4
	Enable signal (S/SXAORD)	(S/SXAORD) = FASEL PH3 NOL5 + ...	Preset adder for OR operation in PH4
			Mnemonic: LS (4A, CA)

(Continued)

Table 3-40. Load Selective Sequence (Cont.)

Phase	Function Performed	Signal Involved	Comments
PH3 T5L (Cont)	Set flip-flop MRQ	S/MRQ = (S/MRQ/1) + ... (S/MRQ/1) = FASEL PH3 NOL7 + ... R/MRQ = ...	Core memory request for next instruction in sequence
	Set flip-flop RW	S/RW = FASEL PH3 OLA R/RW = ...	Prepare to write result into private memory
PH4	One clock long		
T8L	(A0-A31) OR (D0-D31) ⟶ (S0-S31) ⟶ RR0-RR31	Adder logic set at PH3 clock RWXS/0-RWXS/3 = RW + ... RW = Set at PH3 clock	$R_n = EW_n\ Ru1_n + R_n\ NRu1_n$
	Set flip-flop CC3 if S0-S31 is nonzero and positive	S/CC3 = SGTZ TESTS + ... TESTS = FASEL PH4 NOL7 + ... SGTZ = (S0 + S1 + ... + S31) N(S0 NFACOMP) + ... R/CC3 = TESTS + ...	Result is positive and nonzero
	Set flip-flop CC4 if S0-S31 is negative	S/CC4 = NFACOMP S0 TESTS R/CC4 = TESTS + ...	Result is negative
	Branch to PH10	BRPH10 = FASEL PH4 NOL7 + ... S/PH10 = BRPH10 NCLEAR + ... R/PH10 = ...	
	Set flip-flop DRQ	S/DRQ = BRPH10 NCLEAR + ... R/DRQ = ...	Inhibits transmission of another clock until data release from core memory
	Enable clock T8	T8EN = NT5EN NT11L N(SXADD/1 RW) N(RW REU) N(REU AXRR) NT5EN = RW + ...	T5EN is disabled by signal RW
PH10 DR	ENDE functions	See table 3-18	
			Mnemonic: LS (4A, CA)

STORE SELECTIVE (STS; 47, C7). The STS instruction stores the contents of private memory register R into the effective word location, using private memory register Ru1 as a mask.

General. If the R field of the instruction word is even, the instruction operates as follows: If a bit in private memory register Ru1 is a one, the corresponding bit in private memory register R is loaded into the same bit position in the effective word location. If the bit is a zero, the corresponding bit in the effective word location remains unchanged. Logically, the operation is as follows, where n is any bit position:

$$EWL_n = R_n \; Ru1_n + EW_n \; NRu1_n$$

Result in bit position n of effective word location; Mask bit = 1; Mask bit = 0

If the R field of the instruction word is odd, the instruction ORs the contents of private memory register R and the effective word location and stores the result back into the effective word location. Logically, for every n bit position:

$$EWL_n = EW_n + R_n$$

Result in bit position n of effective word location

Examples. Examples of STS with both an even and odd R field are:

Even R Field

```
R    00001111XXXXXXXXXXXXXXXXXXXXXXXX
Ru1  00110011XXXXXXXXXXXXXXXXXXXXXXXX
EW   01010101XXXXXXXXXXXXXXXXXXXXXXXX  Before
                                        execution
-------------------------------------------------
EWL  01000111XXXXXXXXXXXXXXXXXXXXXXXX  After
                                        execution
```

Odd R Field

```
R    00110101 1XXXXXXXXXXXXXXXXXXXXXXX
EW   01010110 1XXXXXXXXXXXXXXXXXXXXXXX  Before
                                        execution
-------------------------------------------------
EWL  01110111 1XXXXXXXXXXXXXXXXXXXXXXX
```

Store Selective Examples

Store Selective Phase Sequence. Preparation phases for the STS instruction are the same as the general PREP phases for word instructions, paragraph 3-59. Figure 3-137 shows the simplified phase sequence for the STS instruction during execution. Table 3-41 lists the detailed logic sequence during all STS execution phases.

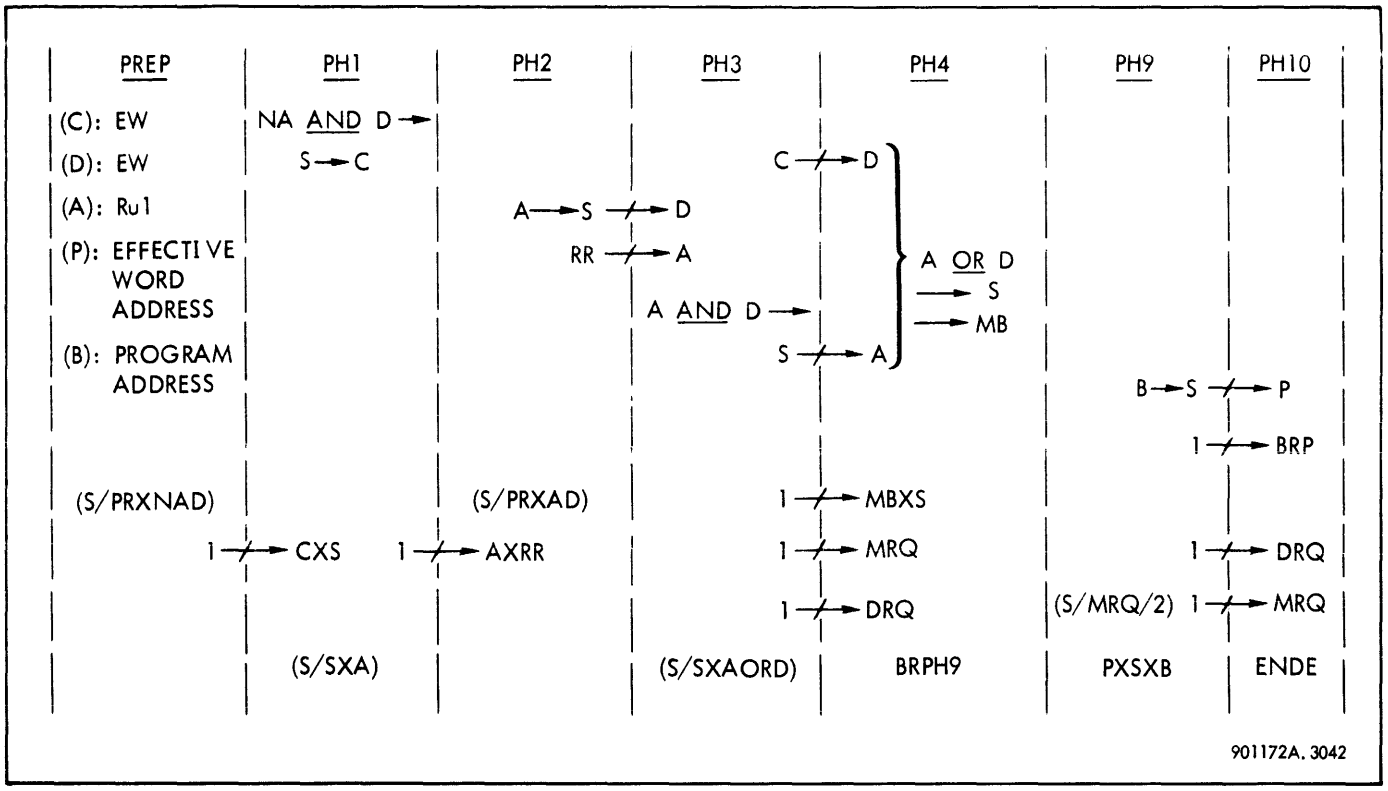


Figure 3-137. Store Selective Phases

Table 3-41. Store Selective Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(C) : EW		Effective word
	(D) : EW		Effective word
	(A) : RRu1		Private memory register Ru1 holds mask program address → in PH9
	(P) : Effective word address		
	(B) : Program address		Temporary storage
	Enable signal (S/PRXNAD)	(S/PRXNAD) = FASEL PRE3 OL7 + ...	Preset adder for NA AND D → S in PH1
	Reset flip-flop NCXS	S/NCXS = N(S/CXS) (S/CXS) = FASEL PRE3 + ... R/NCXS = ...	Preset for S → C in PH1
			Mnemonic: STS (47, C7)

(Continued)

Table 3-41. Store Selective Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH1	One clock long		
T5L	(NA0-NA31) AND (D0-D31) ⟶ (S0-S31) ⟶ (C0-C31)	Adder logic set at last PREP clock CXS = Set in PREP	Complemented mask and effective word ANDed and temporarily stored in C-register
	Enable signal (S/SXA)	(S/SXA) = FASEL PH1 + ...	Preset adder logic for A ⟶ S in PH2
	Set flip-flop AXRR	S/AXRR = FASEL PH1 + ... R/AXRR = ...	Preset for transfer of private memory register R ⟶/⟶ A in PH2
PH2	One clock long		
T5L	(A0-A31) ⟶ (S0-S31) ⟶/⟶ (D0-D31)	Adder logic set at PH1 clock DXS = FASEL PH2 + ...	Store private memory register Ru1 contents in D-register
	(RR0-RR31) ⟶/⟶ (A0-A31)	AXRR = Set in PH1	Preset adder for A AND D ⟶/⟶ S in PH3
	Enable signal (S/PRXAD)	(S/PRXAD) = FASEL PH2 NOLA	
PH3	One clock long		
T5L	(A0-A31) AND (D0-D31) ⟶ (S0-S31) ⟶/⟶ (A0-A31)	Adder logic set at PH2 clock	AND contents of private memory register R and contents of private memory register Ru1. This is significant only when R field is even
	(C0-C31) ⟶/⟶ (D0-D31)	DXC = FASEL PH3 + ...	NRu1 EW ⟶/⟶ D-register in preparation for PH4
	Enable signal (S/SXAORD)	(S/SXAORD) = FASEL PH3 NOL5 + ...	Preset adder for OR operation in PH4
	Set flip-flop MBXS	S/MBXS = FASEL PH3 OL7 + ... R/MBXS = ...	Preset for transfer of result to core memory in PH4
	Set flip-flop MRQ	S/MRQ = (S/MBXS) + ... R/MRQ = ...	Memory request for transferring result
			Mnemonic: STS (47, C7)

(Continued)

Table 3-41. Store Selective Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH3 T5L (Cont.)	Set flip-flop DRQ	S/DRQ = (S/MBXS) + ... R/DRQ = ...	Inhibits transmission of another clock until data release received from core memory
PH4	Sustained until DR		
DR	(A0-A31) OR (D0-D31) ⟶ (S0-S31) ⟶ (MB0-MB31)	Adder logic set at PH3 clock MBXS = Set at PH3 clock	$EWL_n = R_n\ Ru1_n + EW_n\ NRu1_n$
	Branch to PH9	BRPH9 = FASEL PH4 OL7 + ... S/PH9 = BRPH9 NCLEAR + ... R/PH9 = ...	
PH9	One clock long		
T5L	(B0-B31) ⟶ (S0-S31) (S15-S31) ⟶/ (P15-P31)	SXB = PXSXB NDIS + ... PXSXB = NFAFL NFAMDS PH9 PXS = PXSXB + ...	Transfer program address to the P-register
	Set flip-flop BRP	S/BRP = PXSXB + ... R/BRP = PRE1 NFAIM + INTRAP1 + ...	Signifies that program address is in P-register
	Set flip-flop MRQ	S/MRQ = (S/MRQ/2) + ... (S/MRQ/2) = PXSXB NINTRAP2 +... R/MRQ = ...	Core memory request for next instruction in sequence
	Set flip-flop DRQ	S/DRQ = (S/MRQ/2) NCLEAR + ... R/DRQ = ...	Inhibits transmission of another clock until data release received from core memory
PH10	ENDE functions	See table 3-18	
DR			
			Mnemonic: STS (47, C7)

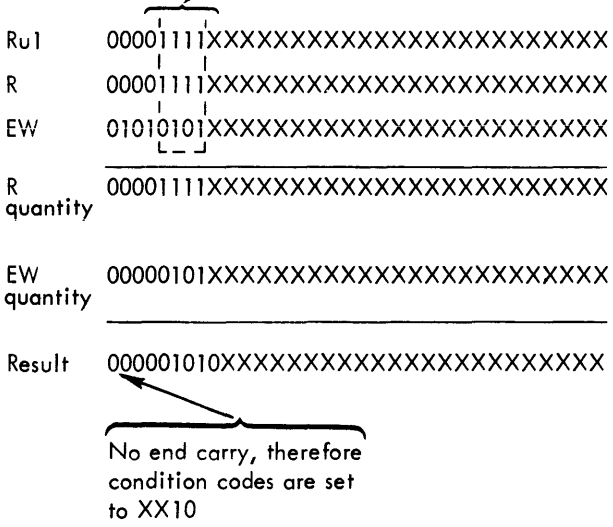
COMPARE SELECTIVE (CS; 45, C5). The CS instruction compares the contents of private memory register R with the contents of the effective word. Only those bit positions of the two operands are compared which are selected by a one in corresponding bit positions of private memory register Ru1. The selected portions of the operands are treated as positive integer quantities.

General. Bit positions containing a one are selected by ANDing the contents of register Ru1 with the contents of register R and with the effective word. If the R field of the instruction word is odd, registers R and Ru1 are identical. Therefore, ANDing the contents of register R and Ru1 is insignificant. The effective word is subtracted from register R to compare the two operands and conditions codes 3 and 4 are set according to the result.

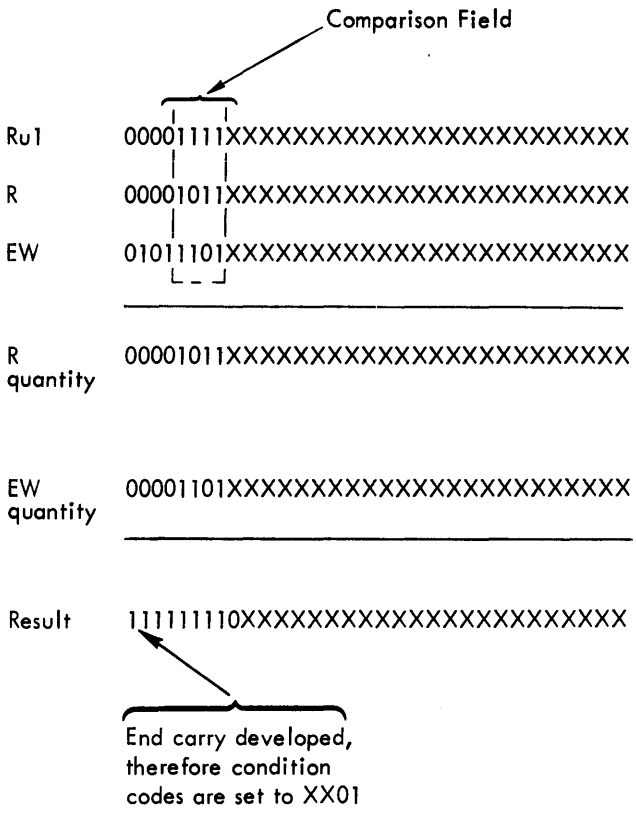
Condition Codes. If the result of the subtraction is zero, the quantities are equal and the condition codes are set to XX00. If the result is negative, the effective word quantity is larger than R-register quantity, and the condition codes are set to XX01. If the result is nonzero and positive, the R-register quantity is larger than the effective word quantity, and the condition codes are set to XX10.

Examples. Examples of CS with R greater than EW and EW greater than R are:

R Greater than EW



EW Greater than R



Compare Selective Phase Sequence. Preparation phases for the CS instruction are the same as the general PREP phases for word instructions, paragraph 3-59. Figure 3-138 shows the simplified phase sequence for the CS instruction, and table 3-42 lists the detailed logic sequence during all execution phases of the instruction.

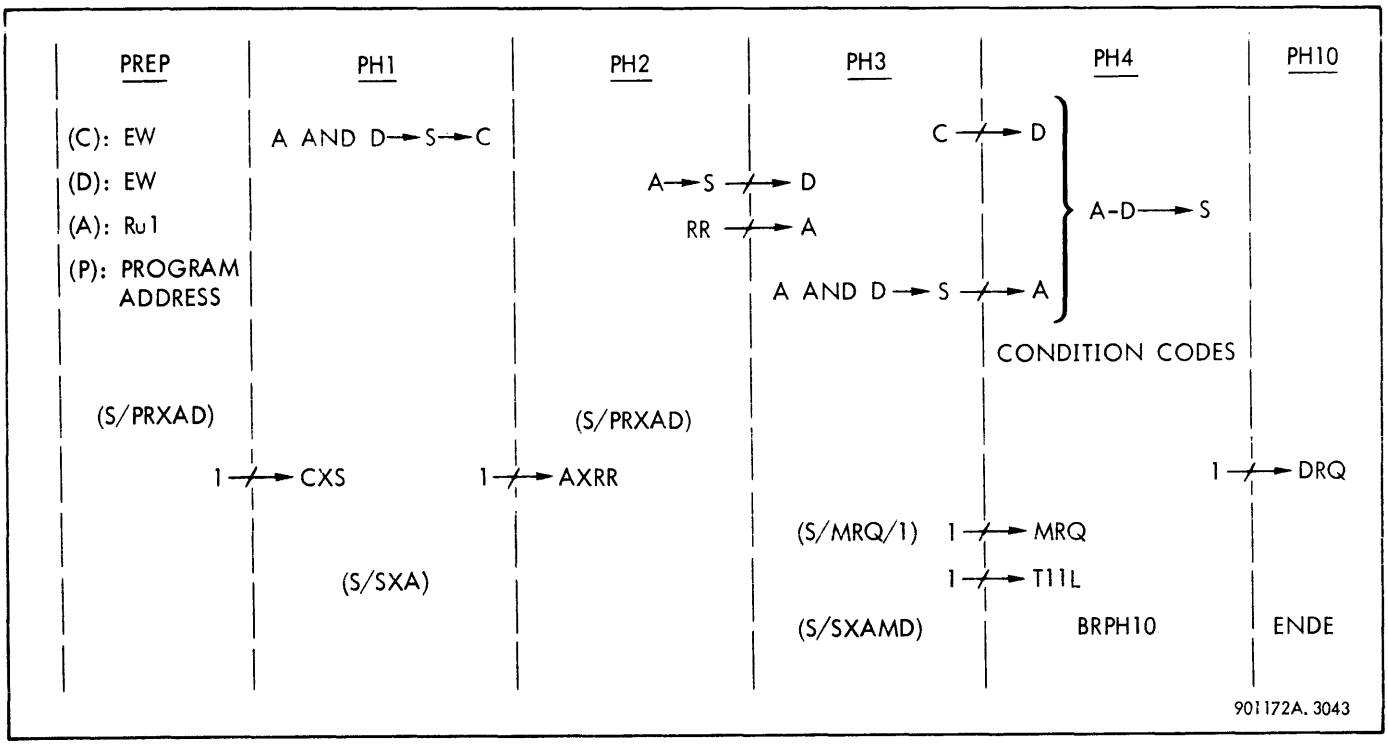


Figure 3-138. Compare Selective Phases

Table 3-42. Compare Selective Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(C) : EW		Effective word
	(D) : EW		Effective word
	(A) : Ru1		Private memory register Ru1 holds mask
	(P) : Program address		Next instruction in sequence
	Enable signal (S/PRXAD)	(S/PRXAD) = FASEL PRE3 NOL7 + ... FASEL = FUCS + ... FACOMP = FUCS + ... FUCS = OU4 OL5	Preset adder for A AND D → S in PH1
	Reset flip-flop NCXS	S/NCXS = N(S/CXS) (S/CXS) = FASEL PRE3 + ... R/NCXS = ...	Preset for S → C in PH1
			Mnemonic: CS (45, C5)

(Continued)

Table 3-42. Compare Selective Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH1	One clock long		
T5L	(A0-A31) AND (D0-D31) ⟶ (S0-S31) ⟶ (C0-C31)	Adder logic set at last PREP clock CXS = Set at last PREP clock	Effective word ANDed with mask and temporarily stored in C-register
	Enable signal (S/SXA)	(S/SXA) = FASEL PH1 + ...	Preset adder logic for A⟶S in PH2
	Set flip-flop AXRR	S/AXRR = FASEL PH1 + ... R/AXRR = ...	Preset for transfer of private memory register R⟶/⟶A in PH2
PH2	One clock long		
T5L	(A0-A31) ⟶ (S0-S31) ⟶/⟶ (D0-D31)	Adder logic set at PH1 clock DXS = FASEL PH2 + ...	Store private memory register Ru1 contents in D-register
	(RR0-RR31) ⟶/⟶ (A0-A31)	AXRR = Set at PH1 clock	Store private memory register R contents in A-register
	Enable signal (PRXAD)	(S/PRXAD) = FASEL PH2 NOLA	Preset adder for A AND D in PH3
PH3	One clock long		
T5L	(A0-A31) AND (D0-D31) ⟶ (S0-S31) ⟶/⟶ (A0-A31)	Adder logic set at PH2 clock	AND contents of private memory register R and contents of private memory register Ru1. This is significant only when R field is even
	(C0-C31) ⟶/⟶ (D0-D31)	DXC = FASEL PH3 + ...	(EW Ru1) ⟶/⟶ D-register in preparation for PH4
	Enable signal (S/SXAMD)	(S/SXAMD) = FASEL PH3 OL5 + ...	Preset adder for (A-D) ⟶ S in PH4
	Set flip-flop MRQ	S/MRQ = (S/MRQ/1) + ... (S/MRQ/1) = FASEL PH3 NOL7 + ... R/MRQ = ...	Core memory request for next instruction in sequence
			Mnemonic: CS (45, C5)

(Continued)

Table 3-42. Compare Selective Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH3 T5L (Cont.)	Reset flip-flop NT11L	S/NT11L = N(S/T11L) + ... S/T11L = FASEL PH3 OL5 + ... R/NT11L = ...	Set clock T11L for PH4
PH4 T11L	One clock long (A0-A31) - (D0-D31) ⟶ (S0-S31)	Adder logic set at PH3 clock	(Ru1 R) - (Ru1 EW) ⟶ S For every bit position n on the sum bus, $S_n = Ru1_n (R_n - EW_n)$. For odd R field, $S_n = R_n (1 - EW_n)$ Result on sum bus is an unsigned quantity
	Hold A00 false Hold flip-flop D00 disabled	A00 = NFUCS + ... D00 = C0 (DXC + DXCL1) + ... (DXC + DXCL1) = NFUCS + ...	A00 and D00 prevented from affecting possible end carry in S00
	Set flip-flop CC3 if result is positive and nonzero; otherwise reset CC3	S/CC3 = SGTZ TESTS + ... SGTZ = (S0 + S1 + ... + S31) N(S00 FACOMP) + ... TESTS = FASEL PH4 NOL7 + ... R/CC3 = TESTS + ...	R-register quantity is larger than EW quantity S00 is developed from end carry and indicates that EW quantity in D-register was larger than R quantity in A-register
	Set flip-flop CC4 if result is negative; otherwise reset CC4 Branch to PH10	S/CC4 = FACOMP S00 + ... R/CC4 = TESTS + ... BRPH10 = FASEL PH4 NOL7 + ... S/PH10 = BRPH20 NCLEAR + ... R/PH10 = ...	
	Set flip-flop DRQ	S/DRQ = BRPH10 NCLEAR + ... R/DRQ = ...	Inhibits transmission of another clock until data release received from core memory
	Enable clock T11	NT5EN = T11L + ... NT8EN = T11L + ...	Clock T11 is enabled by disabling clocks T5 and T8
PH10 DR	ENDE functions	See table 3-18	
			Mnemonic: CS (45, C5)

3-64 Family of Analyze Instructions

ANALYZE (ANLZ; 44, C4). The ANLZ instruction treats the effective word as a Sigma 5 instruction and determines its addressing type (immediate, byte, halfword, word, doubleword). If the instruction to be analyzed is not an immediate address instruction, the ANLZ instruction calculates the effective address that would be produced by the instruction to be analyzed and loads this effective address into private memory register R. The condition codes are set to indicate the addressing type. If the instruction analyzed is an Immediate Address instruction, the condition code is set to indicate the addressing type, and the original contents of private memory register R are not changed.

Trap Conditions. During preparation phases of an ANLZ instruction, the contents of the location pointed to by the effective address of the instruction are obtained. The nonexistent memory address trap can occur as a result of this memory access. The nonexistent instruction trap, the privileged instruction trap, and the unimplemented instruction trap conditions can never occur during execution of an ANLZ instruction. However, the nonexistent memory address trap can occur as a result of any memory access initiated by ANLZ. If this trap condition occurs, the instruction address stored by the XPSD in trap location X'40' is the address of the ANLZ instruction.

Condition Codes. The following condition codes may be stored during execution of an ANLZ instruction:

CC1	CC2	CC3	CC4	Instruction Addressing Type
0	0	0	0	Direct byte addressing
0	0	0	1	Immediate byte addressing
0	0	1	0	Indirect byte addressing
0	1	0	0	Direct halfword addressing
0	1	1	0	Indirect halfword addressing
1	0	0	0	Direct word addressing
1	0	1	0	Indirect word addressing
1	0	0	1	Immediate addressing
1	1	0	0	Direct doubleword addressing
1	1	1	0	Indirect doubleword addressing

Analyze Execution Sequence. If the operation code portion of the effective word specifies a nonimmediate addressing type, the condition code is set to indicate the addressing type of the analyzed instruction. The effective address of the analyzed instruction is computed, using all the normal address computation rules. If bit 0 of the effective word is a one, the contents of the memory location specified by bits 15 through 31 of the effective word are obtained and used as a direct address. If bits 12 through 14 of the analyzed instruction are nonzero, indexing is performed, using the index register in the current register block. (The R field of the instruction in the effective word location is ignored.)

The effective address of the analyzed instruction is aligned as an integer displacement value and loaded into private memory register R according to the instruction addressing type, as follows:

Addressing Type	Location of Address
Byte	Zeros in bits 0 through 12, 19-bit byte displacement in bits 13 through 31
Halfword	Zeros in bits 0 through 13, 18-bit halfword displacement in bits 14 through 31
Word	Zeros in bits 0 through 14, 17-bit word displacement in bits 15 through 31
Doubleword	Zeros in bits 0 through 15, 16-bit doubleword displacement in bits 16 through 31

Analyze Phase Sequence. Preparation phases for the ANLZ instruction are the same as the general PREP phases for word instructions, paragraph 3-59. Figure 3-139 shows the simplified phase sequence for the Analyze execution phases, and table 3-43 lists the detailed logic sequence during the ANLZ instruction execution phases. The second cycle of preparation phases entered to analyze the instruction is illustrated in figure 3-140.

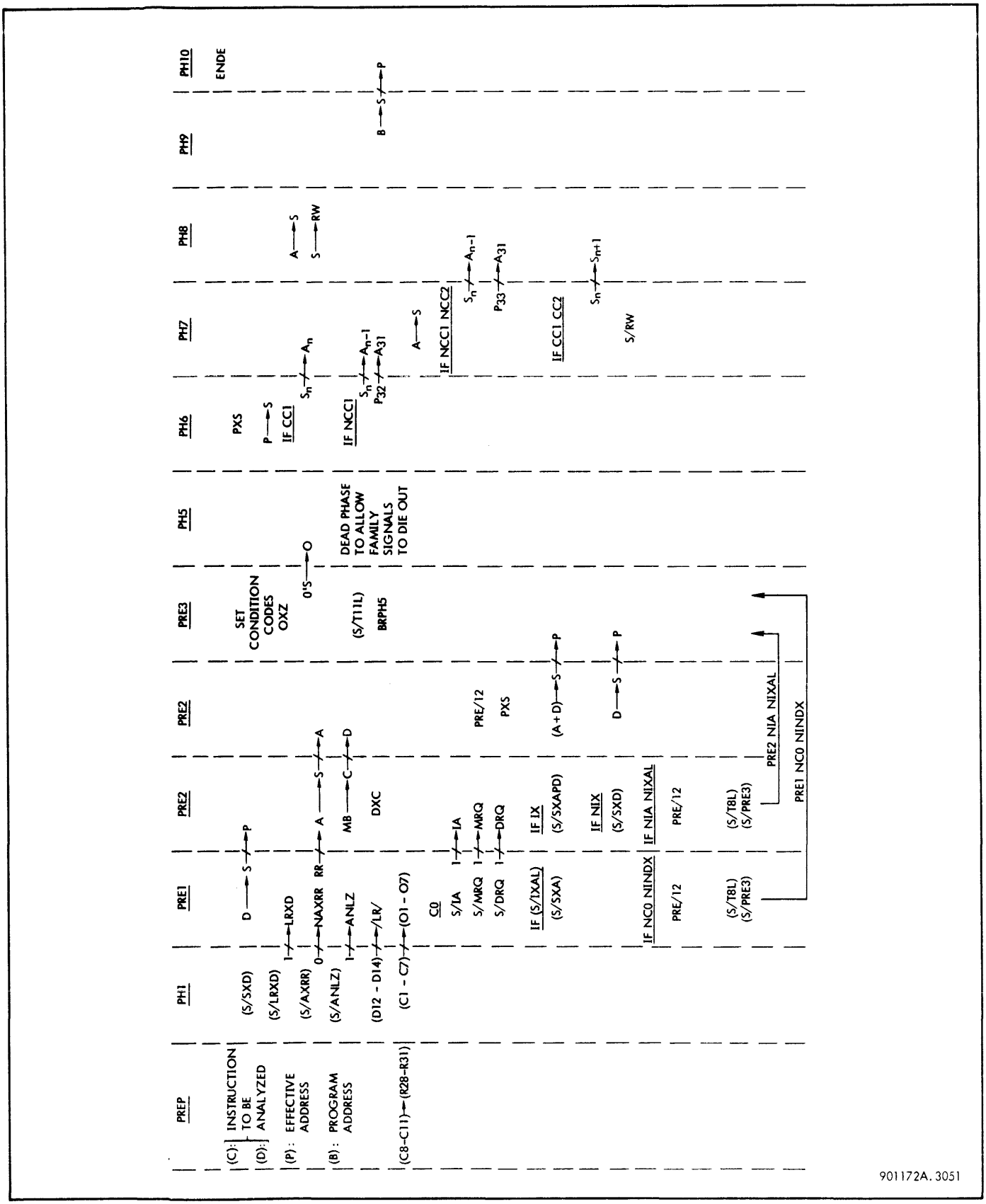


Figure 3-139. Analyze Instruction, Phase Sequence Diagram

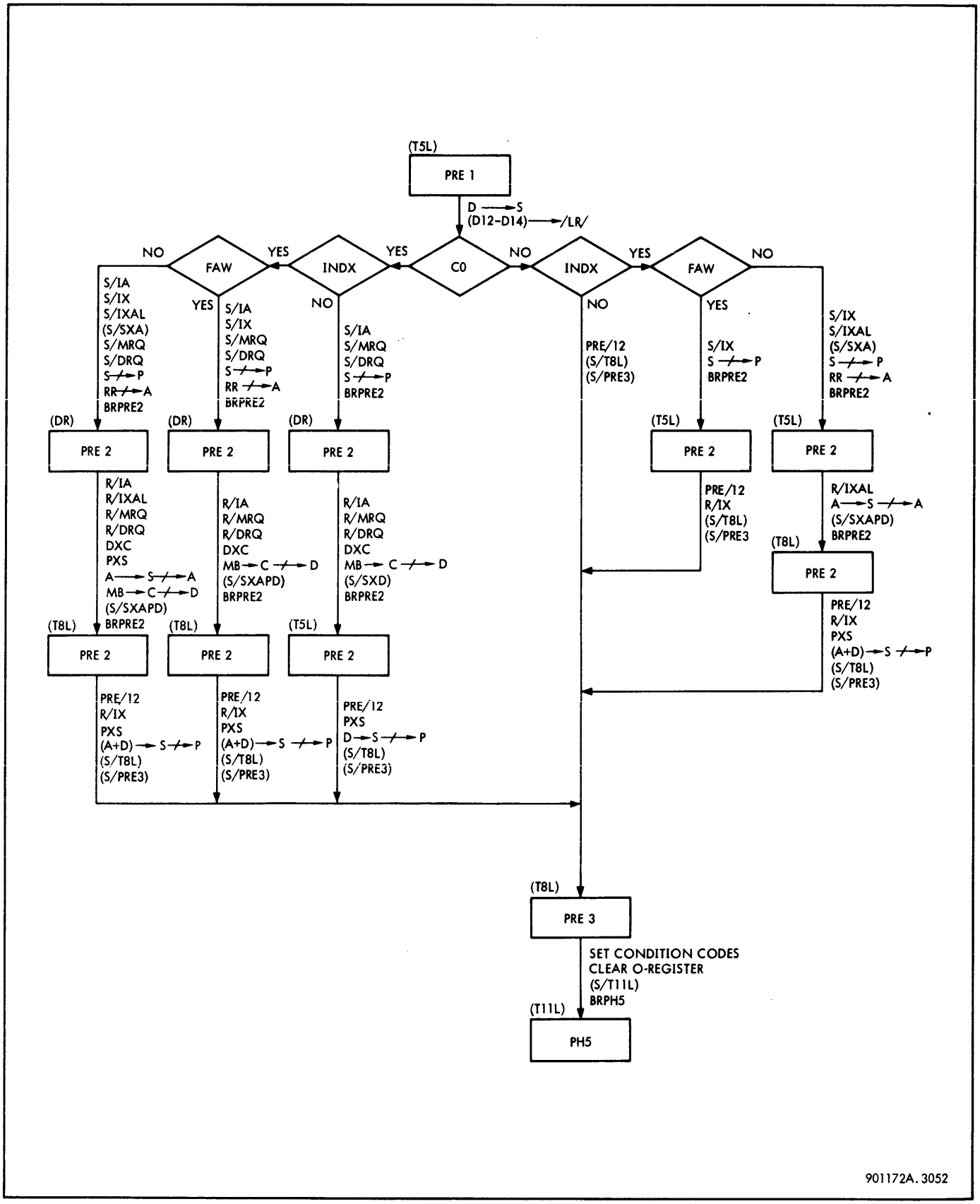


Figure 3-140. Analyze Instruction, Preparation Phases Flow Diagram

Table 3-43. Analyze Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(C) : Instruction to be analyzed		
	(D) : Instruction to be analyzed		
	(P) : Effective address of ANLZ instruction		
	(B) : Program address		
	(C8-C11)—/→(R28-R31)	RXC = PH10 + ...	R field of ANLZ instruction stored for future use
PH1	One clock long		
T5L	(C1-C7)—/→(O1-O7)	OXC = FUANLZ PH1 FUANLZ = OU4 OL4	Operation code of instruction to be analyzed
	Enable signal (S/SXD)	(S/SXD) = FUANLZ PH1 + ...	Preset for D——→S transfer in PRE1
	Set flip-flop LRXD	S/LRXD = (S/LRXD) (S/LRXD) = OXC R/LRXD = ...	Preset for (D12-D14) ——→/LR/ in PRE1
	Set flip-flop ANLZ	S/ANLZ = FUANLZ PH1 R/ANLZ = CLEAR = PH10 + ...	Maintains ANLZ sequence until PH10
	Reset flip-flop NAXRR	S/NAXRR = N(S/AXRR) (S/AXRR) = FUANLZ PH1 + ... R/NAXRR = ...	Preset for RR —/→A in PRE1
	Reset flip-flop NPRE1	S/NPRE1 = N(S/PRE1) (S/PRE1) = NCLEAR FUANLZ PH1 + ... R/NPRE1 = ...	Branch to phase PRE1
PRE1	One clock long		
T5L	(D0-D31)——→(S0-S31)	Adder logic set at PH1 clock	Instruction to be analyzed
	(S15-S31)—/→(P15-P31)	PXS = NFAIM PRE1 FAIM = NO3 NO4 NO5	Address to program register. Significant except when NC0 NINDX (immediate)
	(RR0-RR31)—/→(A0-A31)	AXRR = Set at PH1 clock	Significant only for instructions in which R value is not zero
			Mnemonic: ANLZ (44, C4)

(Continued)

Table 3-43. Analyze Sequence (Cont.)

Phase	Function Performed	Signals Involved			Comments
PRE1	(D12-D14)⟶(/LR29/-/LR31/)	LRXD	=	Set at PH1 clock	R value stored
T5L (Cont.)	If C0 or INDX:				
	Set flip-flop IA	S/IA	=	PRE1 C0	Indirect addressing of instruction to be analyzed
		R/IA	=	...	
	Set flip-flop IX	S/IX	=	PRE1 INDX	
		INDX	=	(C3 + C4 + C5) (C12 + C13 + C14)	Not immediate addressing or R value not zero. Remains in set state until last cycle of phase PRE2
		R/IX	=	PRE/12 + ...	
		PRE/12	=	NIA NIXAL PRE2	
	Set flip-flop IXAL	S/IXAL	=	(S/IXAL) NCLEAR	Not word addressing
		(S/IXAL)	=	PRE1 INDX (FAHW + FABYTE + FADW)	
		FAHW	=	O1 NO2 O3	Halfword addressing
		FABYTE	=	O1 O2 O3	Byte addressing
		FADW	=	NO1 NO2 O4 + NO1 NO2 O3	Doubleword addressing
	Set flip-flop MRQ	S/MRQ	=	(S/MRQ/2) + ...	Core memory request for address
		(S/MRQ/2)	=	PRE1 C0 NFAIM + ...	
		R/MRQ	=	...	
	Set flip-flop DRQ	S/DRQ	=	(S/DRQ) NCLEAR	Inhibits transmission of another clock until data release signal received from core memory
		(S/DRQ)	=	(S/MRQ/2) + ...	
		R/DRQ	=	...	
	Enable signal (S/SXA)	(S/SXA)	=	(S/IXAL) + ...	Preset for A⟶S in PRE2
	If NC0 NINDX:				
	End phase PRE1	PRE/12	=	PRE1 NC0 NINDX	Immediate addressing
	Branch to phase PRE3	S/NRPE3	=	N(S/PRE3)	NBR true because no branch signal true
		(S/PRE3)	=	PRE/12 NBR	
		R/NRPE3	=	...	
	Reset flip-flop NT8L	S/NT8L	=	N(S/T8L)	Set clock T8L for PRE3
		(S/T8L)	=	PRE/12 + ...	
		R/NT8L	=	...	
					Mnemonic: ANLZ (44, C4)

(Continued)

Table 3-43. Analyze Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PRE2	Timing:		
	If IA IX, DR followed by T8L If IA NIX, DR followed by T5L If NIA NIX, T5L followed by T8L If NIA IX, T5L only		PRE2 phase is maintained if BRPRE2 is true at clock time
	If IXAL, An ⟶ Sn ⟶/ Sm, with shifts	AXSR2 = IXAL PRE2 FABYTE	Right shift 2 if byte addressing
		AXSR1 = IXAL PRE2 FAHW	Right shift 1 if halfword addressing
		AXSL1 = IXAL PRE2 FADW	Left shift 1 if double-word addressing
	If IA, (MB0-MB31) ⟶ (C0-C31)	CXMB = DG = /DG/	Transfer address to C-register
	(C0-C31) ⟶/ (D0-D31)	DXC = IA PRE2 + ...	Address to D-register
	Set flip-flop SW5 if IA	S/SW5 = ANLZ IA + ... R/SW5 = (R/SW5)	Store information that IA set for PRE3 operation
	Enable signal (S/SXAPD)	(S/SXAPD) = PRE2 IA IX + PRE2 IXAL + ...	Preset for (A + D) ⟶ S in next PRE2 phase
	Enable signal (S/SXD)	(S/SXD) = PRE2 IA NIX + ...	Preset for D ⟶ S in next PRE2 phase
	Branch to PRE2	BRPRE2 = IA PRE2 + IXAL PRE2 + ...	Remain in phase PRE2 until IA and IXAL reset
	Enable clock T8 for PRE2 if addition is to be performed	T8EN = NT5EN NT11L N(SXADD/1 RW) N(RW REU) N(REU AXRR) NT5EN = SXADD/1 + ...	T5EN is disabled by signal SXADD/1 when (S/SXAPD) is true
	(A0-A31) + (D0-D31) ⟶ (S0-S31)	Adder logic set at PRE2 clock by (S/SXAPD)	When IA IX or IX IXAL
	(D0-D31) ⟶ (S0-S31)	Adder logic set at PRE2 clock by (S/SXD)	When IA NIX
	(S15-S31) ⟶/ (P15-P31)	PXS = PRE2	
	Terminate PRE2 phase	PRE/12 = NIA NIXAL PRE2 + ...	Remain in PRE2 phase until NIA NIXAL
	Reset flip-flop NPRE3	S/NPRE3 = N(S/PRE3) NBR (S/PRE3) = PRE/12 R/NPRE3 = ...	Branch to PRE3 phase
			Mnemonic: ANLZ (44, C4)

(Continued)

Table 3-43. Analyze Sequence (Cont.)

Phase	Function Performed	Signals Involved			Comments
PRE2 (Cont.)	Reset flip-flop NT8L	S/NT8L (S/T8L) R/NT8L	= = =	N(S/T8L) PRE/12 + ...	Set clock T8L for phase PRE3
PRE3	One clock long				
T8L	Set flip-flop CC1 if instruction to be analyzed is word or doubleword addressing	S/CC1 (S/CC1/3) (S/CC1/1) R/CC1 (R/CC)	= = = = =	(S/CC1/3) + ... (S/CC1/1) + ... ANLZ PRE3 NFAIM NO3 + ANLZ PRE3 NO1 + ... (R/CC1) = (R/CC) + ... ANLZ PRE3 + ...	Word or doubleword addressing identified by signals NFAIM NO3 or NO1
	Set flip-flop CC2 if instruction to be analyzed is halfword or doubleword addressing	S/CC2 (S/CC2/3) (S/CC2/1) R/CC2	= = = =	(S/CC2/3) + ... (S/CC2/1) + ... FADW ANLZ PRE3 + FAHW ANLZ PRE3 (R/CC) + ...	
	Set flip-flop CC3 if instruction to be analyzed is indirect addressing (C0 = 1)	S/CC3 (S/CC3/1) R/CC3	= = =	(S/CC3/1) + ... ANLZ PRE3 SW5 + ... (R/CC) + ...	Flip-flop SW5 set during PRE2 phase if IA set at PRE1 clock
	Set flip-flop CC4 if instruction to be analyzed is immediate addressing	S/CC4 (S/CC4/1) R/CC4	= = =	(S/CC4/1) + ... ANLZ PRE3 FAIM + ... (R/CC) + ...	
	Reset flip-flop NT11L	S/NT11L (S/T11L) R/NT11L	= = =	N(S/T11L) + ... PRE3 ANLZ + ...	Set clock T11L for PH5
	Clear O-register	OXZ	=	ANLZ PRE3 + ...	Store zeros in O-register
	Branch to PH5	BRPH5	=	ANLZ PRE3 + ...	
PH5 T11L	One clock long Dead phase to allow family signals to die out				
PH6 T5L	One clock long Enable signal (S/SXA)	(S/SXA)	=	ANLZ PH6 + ...	Preset for A ⟶ S in PH7
					Mnemonic: ANLZ (44, C4)

(Continued)

Table 3-43. Analyze Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH6 T5L (Cont.)	(P15-P31)——►(S15-S31)	SXP = ANLZ PH6 NDIS + ...	
	If CC1 set in PRE3:		
	(S15-S31)—/—►(A15-A31)	AXS = ANLZ PH6 CC1 + ...	Word or doubleword addressing
	If CC1 reset in PRE3:		
	(S15-S31)—/—►(A14-A30)	AXSL1 = ANLZ PH6 NCC1 + ...	Byte or halfword addressing
	P32—/—►A31	A31XP32 = ANLZ PH6 NCC1	
PH7 T5L	One clock long (A0-A31)——►(S0-S31)	Adder logic set at PH6 clock	
	If NCC1 NCC2 in PRE3:		
	(S15-S31)—/—►(A14-A30)	AXSL1 = NCC1 NCC2 ANLZ PH7 +...	Byte addressing
	P32—/—►A31	A31XP33 = NCC1 NCC2 ANLZ PH7 + ...	
	If CC1 CC2 in PRE3:		
	(S14-S30)—/—►(A15-A31)	AXSR1 = CC1 CC2 ANLZ PH7 + ...	Doubleword addressing. If CC1 NCC2 or NCC1 CC2, no additional transfers
	Set flip-flop RW	S/RW = (S/RW/1) (S/RW/1) = (S/RW) + ... (S/RW) = PH7 ANLZ NOL1 NCC4 + ... R/RW = ...	Prepare to write result in private memory if analyzed instruction was not immediate addressing
	Enable signal (S/SXA)	(S/SXA) = ANLZ PH7 + ...	Preset for A——►S in PH8
PH8 T8	One clock long Enable clock T8 for PRE2 if addition is to be performed	T8EN = NT5EN NT11L N(RW REU) N(SXADD/1 RW) N(REU AXRR) NT5EN = RW + ...	T5EN is disabled by signal RW
	(A0-A31)——►(S0-S31)	Adder logic set at PH7 clock	
			Mnemonic: ANLZ (44, C4)

(Continued)

Table 3-43. Analyze Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH8 T8 (Cont.)	If NFAIM: (S0-S31) ⟶ (RW0-RW31)	RWXS/0-RWXS/3 = RW + ... RW = Set at PH7 clock if NFAIM	Transfer address to private memory register R if not immediate address
PH9 T5L	One clock long (B0-B31) ⟶ (S0-S31)	SXB = PXSXB NDIS + ... PXSXB = NFAFL NFAMDS PH9	Transfer program address to P-register from temporary storage in B-register
	(S15-S31) ⟶ (P15-P31)	PXS = PXSXB + ...	
	Set flip-flop BRP	S/BRP = PXSXB + ... R/BRP = PRE1 NFAIM + ...	Indicates that program address is in P-register
	Set flip-flop MRQ	S/MRQ = (S/MRQ/2) + ... (S/MRQ/2) = PXSXB NINTRAP2 R/MRQ = ...	Core memory request for next instruction in sequence
	Set flip-flop DRQ	S/DRQ = (S/MRQ/2) + ... R/DRQ = ...	Inhibits transmission of another clock until data release from core memory
PH10 DR	Sustained until data release ENDE functions	See table 3-18	
			Mnemonic: ANLZ (44, C4)

3-65 Interpret (INT; 6B, EB)

GENERAL. The INT instruction operates with an even R field in the instruction word as follows:

a. Bits 0 through 3 of the effective word are loaded into condition code flip-flops CC1 through CC4.

b. Bits 4 through 15 of the effective word are loaded into bit positions 20 through 31 of private memory register R. The remainder of the register is cleared.

c. Bits 16 through 31 of the effective word are loaded into bit positions 16 through 31 of private memory register Ru1. The remainder of the register is cleared.

If the R field of the instruction word is odd:

a. Bits 0 through 3 of the effective word are loaded into the condition code flip-flops.

b. Bits 16 through 31 of the effective word are loaded into bit positions 16 through 31 of private memory register R. The remainder of the R-register is cleared.

c. Bits 4 through 15 of the effective word are ignored.

Examples. Examples of INT with both an even and odd R field are shown in figure 3-141.

Interpret Phase Sequence. Preparation phases for the INT instruction are the same as the general PREP phases for word instructions, paragraph 3-59. Figure 3-142 shows the simplified phase sequence for INT execution phases, and table 3-44 lists the detailed logic sequence during the instruction execution phases.

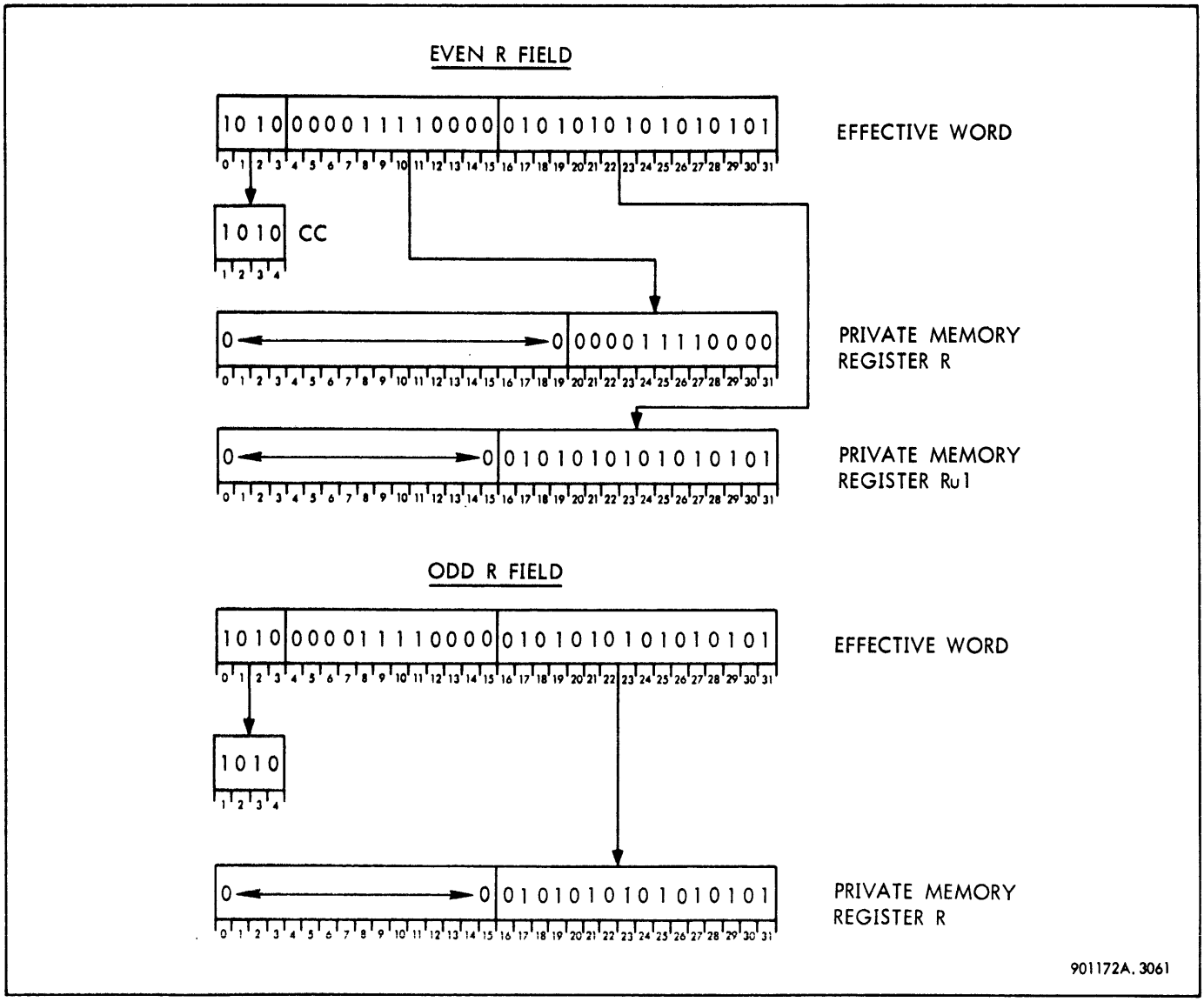


Figure 3-141. Interpret Examples

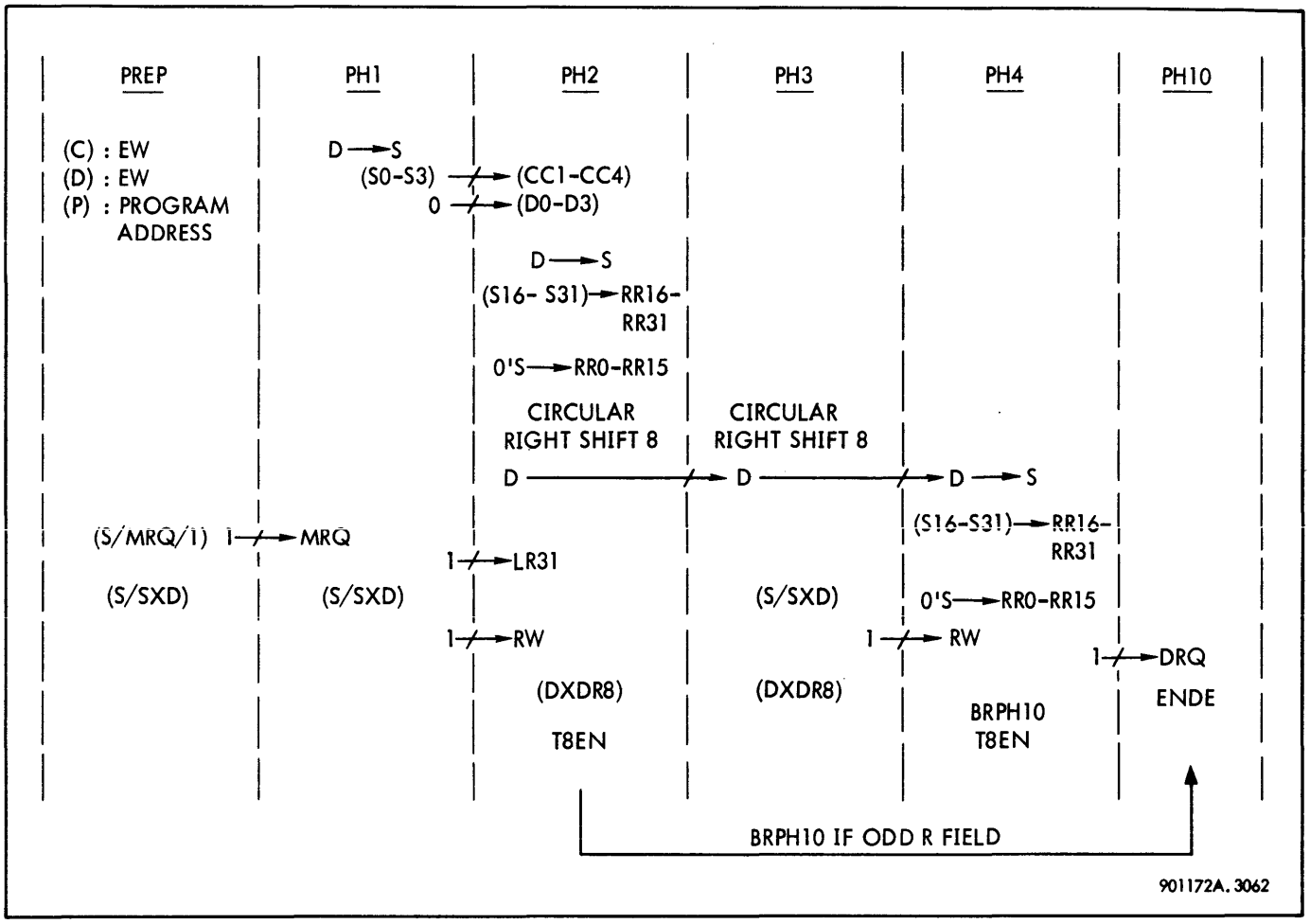


Figure 3-142. Interpret Phases

Table 3-44. Interpret Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(C) : EW		Effective word. C-register not used during INT
	(D) : EW		Effective word
	(P) : Program address		Next instruction in sequence
	Set flip-flop MRQ	S/MRQ = (S/MRQ/1) + ... (S/MRQ/1) = FUINT PRE3 + ... R/MRQ = ...	Core memory request for next instruction in sequence
	Enable signal (S/SXD)	(S/SXD) = FUINT PRE3 + ...	Preset adder logic for D → S in PH1
			Mnemonic: INT (6B, EB)

(Continued)

Table 3-44. Interpret Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH1	One clock long		
T5L	(D0-D31) ⟶ (S0-S31)	Adder logic set at last PREP clock	Effective word ⟶ S
	(S0-S3) ⟶/⟶ (CC1-CC4)	S/CC1 = S0 CCXS/0 + ... ⋮ ⋮ S/CC4 = S3 CCXS/0 + ...	First four bits of effective word ⟶/⟶ condition code flip-flops
		CCXS/0 = FUINT PH1 + ...	
		R/CC1-R/CC4 = CCXS/0 + ...	
	Clear (D0-D3) at clock	DX/0A = D0003XZ + ...	
		D0003XZ = FUINT PH1 + ...	
	Force a one onto LR31 address line	(S/LR31) = FUINT PH1 + ...	Selects private memory register Ru1 for transfer of bits 16 through 31 of effective word in PH2
	Enable signal (S/SXD)	(S/SXD) = FUINT PH1 + ...	Preset adder logic for D ⟶ S in PH2
	Set flip-flop RW	S/RW = FUINT PH1 + ...	Prepare to write bits 16 through 31 into register Ru1
		R/RW = ...	
PH2	One clock long		
T8L	(D0-D31) ⟶ (S0-S31)	Adder logic set at PH1 clock	
	(S16-S31) ⟶ (RW16-RW31)	RWXS/2-RWXS/3 = RW	Write bits 16 through 31 of effective word into private memory register Ru1
	0's ⟶ (RW0-RW15)	RWXS/0-RWXS/1 = RW NRWXZ/01	No gating term enabled
		RWXZ/01 = FUINT PH2 + ...	Effectively clears least significant half of Ru1 register
		RW = Set at PH1 clock	
	Circular right shift D-register eight bit positions	DXDR8 = FUINT PH2 + ...	Bring bits 4 through 15 of effective word into position. One more shift is done in PH3
	Enable clock T8	T8EN = NT5EN NT11L N(SXADD/1 RW) N(RW REU) N(REU AXRR)	T5EN is disabled by signal RW
		NT5EN = RW + ...	
	Branch to PH10 if R field of instruction word is odd	BRPH10 = FUINT PH2 R31 + ...	Bits 4 through 15 of effective word not transferred if R field is odd
		S/PH10 = BRPH10 NCLEAR + ...	
		R/PH10 = ...	
			Mnemonic: INT (6B, EB)

(Continued)

Table 3-44. Interpret Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH3	One clock long		
T5L	Circular right shift D-register eight bit positions	DXDR8 = FUINT PH3 + ...	Bits 0 through 15 of effective word are now in bit positions 16 through 31 of D-register (bits 0 through 3 are zeros)
	Enable signal (S/SXD)	(S/SXD) = FUINT PH3 + ...	Preset adder logic for D ⟶ S in PH4
	Set flip-flop RW	S/RW = FUINT PH3 + ... R/RW = ...	Prepare to write bits 4 through 15 of effective word into private memory register R
PH4	One clock long		
T8L	(D0-D31) ⟶ (S0-S31)	Adder logic set at PH3 clock	
	(S16-S31) ⟶ (RW16-RW31)	RWXS/2-RWXS/3 = RW	Write bits 4 through 15 of effective word into private memory register R (S0 through S3 are zeros)
	0's ⟶ (RW0-RW15)	RWXS/0-RWXS/1 = RW NRWXZ/01	No gating term enabled
		RWXZ/01 = FUINT PH2 + ...	Effectively clears least significant half of R-register
		RW = Set at PH3 clock	
	Enable clock T8	T8EN = NT5EN NT11L N(SXADD/1 RW) N(RW REU) N(REU AXRR)	T5EN is disabled by signal RW
		NT5EN = RW + ...	
	Branch to PH10	BRPH10 = FUINT PH4 + ...	
		S/PH10 = BRPH10 NCLEAR + ...	
		R/PH10 = ...	
	Set flip-flop DRQ	S/DRQ = BRPH10 NCLEAR + ...	Inhibits transmission of another clock until data release received from core memory
		R/DRQ = ...	
PH10 DR	Entered from PH2 if R field is odd or from PH4 if R field is even ENDE functions	See table 3-18	
			Mnemonic: INT (6B, EB)

3-66 Family of Arithmetic Instructions (FAARITH)

ADD IMMEDIATE (AI; 20, A0). The AI instruction adds the sign-extended value field of the instruction word to the contents of private memory register R and loads the sum back into private memory register R.

General. The sign of the value field, bit position 12, is extended 12 bit positions to the left during the PREP phases for this instruction. The actual addition of the 32-bit sign-extended value is performed during AI execution phases.

Condition Codes. If the result in the R-register is zero, the condition codes are set to XX00. If the result is non-zero and positive, the condition codes are set to XX10. A negative result produces condition code settings of XX01. Flip-flop CC2 is set if fixed-point overflow occurs during the addition. Flip-flop CC1 is set if there is a carry from bit position zero.

Trap Conditions. A trap to memory location X'43' occurs if there is fixed-point overflow and the fixed-point arithmetic mask bit is a one. The result in private memory register R remains unchanged. If overflow occurs and the mask bit is a zero, the next instruction in sequence is executed.

Add Immediate Phase Sequence. Preparation phases for the AI instruction are the same as the general PREP phases for immediate instructions described in paragraph 3-59. Table 3-45 lists the detailed logic sequence during all AI execution phases.

Table 3-45. Add Immediate Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(C) : Value Field$_{SE}$		Sign-extended value field of instruction word
	(D) : Value Field$_{SE}$		Sign-extended value field of instruction word
	(A) : RR		Contents of private memory register R. Value field will be added to this quantity
	(P) : Program address		Next instruction in sequence
	Enable signal (S/SXAPD)	(S/SXAPD) = FAADD PRE/34 + ...	Preset adder for A + D ⟶ S in PH1
	Set flip-flop MRQ	S/MRQ = (S/MRQ/1) + ... (S/MRQ/1) = FAS10 PRE/34 + ... FAS10 = FAARITH + ... R/MRQ = ...	Core memory request for next instruction in sequence
	Set flip-flop RW	S/RW = FAS11 PRE/34 NOL1 + ... FAS11 = FAARITH + ... R/RW = ...	Prepare to write result into private memory register R
PH1 T5L	One clock long (A0-A31) + (D0-D31) ⟶ (S0-S31) ⟶ (RW0-RW31)	Adder logic set at last PREP clock RWXS/0-RWXS/3 = RW + ... RW = Set at last PREP clock	Store result (sum) in private memory register R
			Mnemonic: AI (20, A0)

(Continued)

Table 3-45. Add Immediate Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH1 T5L (Cont.)	Set flip-flop CC1 if end carry from result	S/CC1 = K00 CC1XK00 + ... CC1XK00 = FAARITH PH1 + ... R/CC1 = CC1XK00 + ...	K00 is end carry from the addition
	Set flip-flop CC2 if arithmetic overflow	S/CC2 = (S00 ⊕ S0) PROBOVER + ... PROBOVER = FAARITH PH1 + ... R/CC2 = PROBOVER + ...	Arithmetic overflow occurs when two numbers of like sign are added and their sum cannot be held in 32 bits
	Set flip-flop OVERIND	S/OVERIND = PROBOVER + ... R/OVERIND = CLEAR	Setting OVERIND enables trap if overflow, and mask bit is equal to a one. Trap is set during ENDE
	Set flip-flop CC3 if result of addition is positive and nonzero; otherwise reset CC3	S/CC3 = SGTZ TESTS + ... SGTZ = (S0 + S1 + ... + S31) NS0 + ... TESTS = FAS11 PH1 + ... R/CC3 = TESTS + ...	
	Set flip-flop CC4 if result of addition is negative; otherwise reset CC4	S/CC4 = S0 TESTS + ... R/CC4 = TESTS + ...	
	Enable clock T11	NT5EN = RW + ... RW = Set at last PREP clock NT8EN = SXADD/1 RW + ... SXADD/1 = GXAD + ... GXAD = Set at last PREP clock	Clock T11 is enabled by disabling clocks T5 and T8 Flip-flop GXAD is part of the adder logic
	Branch to PH10	BRPH10 = FAS10 PH1 + ...	
	Set flip-flop DRQ	S/DRQ = BRPH10 + ... R/DRQ = ...	Inhibits transmission of another clock until data release signal is received from core memory
PH10 DR	ENDE functions	See table 3-18	
			Mnemonic: AI (20, A0)

ADD HALFWORD (AH; 50, D0) AND SUBTRACT HALFWORD (SH; 58, D8). The AH and SH instructions add or subtract the sign-extended effective halfword from the contents of private memory register R and load the result back into private memory register R.

General. The sign of the effective halfword is extended 16 bit positions to the left to produce a 32-bit quantity. Sign extension occurs during the PREP phases. The actual addition or subtraction of the sign-extended halfword is performed during AH or SH execution phases. Implementation

of the two instructions is identical except for the arithmetic operation involved.

Condition Codes. If the result in the R-register is zero, the condition codes are set to XX00. If the result is nonzero and positive, the condition codes are set to XX10. A negative result produces condition code settings of XX01. Flip-flop CC2 is set if fixed-point overflow occurs during addition or subtraction. Flip-flop CC1 is set if there is a carry from bit position zero.

Trap Conditions. A trap to memory location X'43' occurs if there is fixed-point overflow and the fixed-point arithmetic mask bit is a one. The result in private memory register R remains unchanged. If overflow occurs and the mask bit is a zero, the next instruction in sequence is executed.

Add Halfword and Subtract Halfword Phase Sequences. Preparation phases for the two instructions are the same as the general PREP phases for halfword instructions, paragraph 3-59. Table 3-46 lists the detailed logic sequence during all AH and SH execution phases.

Table 3-46. AH and SH Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(C) : EH, sign-extended		Effective halfword with sign-extended 16 bit positions to the left
	(D) : EH, sign-extended		
	(A) : RR		Contents of private memory register R. Sign-extended effective halfword will be added to this quantity
	(P) : Program address		Next instruction in sequence
	If AH, enable signal (S/SXAPD)	(S/SXAPD) = FAADD PRE/34 + ...	Preset adder for A + D ⟶S in PH1
	If SH, enable signal (S/SXAMD)	(S/SXAMD) = FASUB PRE/34 + ...	Preset adder for A - D ⟶S in PH1
	Set flip-flop MRQ	S/MRQ = (S/MRQ/1) + ... (S/MRQ/1) = FAS10 PRE/34 + ... FAS10 = FAARITH + ... R/MRQ = ...	Core memory request for next instruction in sequence
	Set flip-flop RW	S/RW = FAS11 PRE/34 NOL1 + ... FAS11 = FAARITH + ... R/RW = ...	Prepare to write result into private memory register R
PH1	One clock long		
	If AH, (A0-A31) + (D0-D31) ⟶(S0-S31)	Adder logic set at last PREP clock	Add sign-extended effective halfword and contents of register R and gate result to sum bus
	If SH, (A0-A31) - (D0-D31) ⟶(S0-S31)	Adder logic set at last PREP clock	Subtract sign-extended effective halfword from contents of register R and gate result to sum bus
			Mnemonic: AH (50, D0) SH (58, D8)

(Continued)

Table 3-46. AH and SH Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH1 (Cont.)	(S0-S31) ⟶ (RW0-RW31)	RWXS/0-RWXS/3 = RW + ... RW = Set at last PREP clock	Store result in private memory register R
	Set flip-flop CC1 if end carry from result	S/CC1 = K00 CC1XK00 + ... CC1XK00 = FAARITH PH1 + ... R/CC1 = CC1XK00 + ...	K00 is end carry from addition or end borrow from subtraction
	Set flip-flop CC2 if arithmetic overflow	S/CC2 = (S00 ⊕ S0) PROBOVER + ... PROBOVER = FAARITH PH1 + ... R/CC2 = PROBOVER + ...	Arithmetic overflow occurs when two numbers of like signs are added and their sum cannot be held in 32 bits
	Set flip-flop OVERIND	S/OVERIND/1 = PROBOVER + ... R/OVERIND/1 = CLEAR	Setting OVERIND/1 enables trap if overflow, and mask bit is equal to a one. Trap is set during ENDE
	Set flip-flop CC3 if result is positive and nonzero; otherwise reset CC3	S/CC3 = SGTZ TESTS + ... SGTZ = (S0 + S1 + ... + S31) NS0 + ... TESTS = FAS11 PH1 + ... R/CC3 = TESTS + ...	
	Set flip-flop CC4 if result is negative; otherwise reset CC4	S/CC4 = S0 TESTS + ... R/CC4 = TESTS + ...	
	Enable clock T11	NT5EN = RW + ... RW = Set at last PREP clock NT8EN = SXADD/1 RW + ... SXADD/1 = GXAD + GXNAD + ... GXAD = Set at last PREP clock if AH GXAND = Set at last PREP clock if SH	Clock T11 is enabled by disabling clocks T5 and T8 Flip-flop GXAD is part of adder logic Flip-flop GXNAD is part of adder logic
	Branch to PH10	BRPH10 = FAS10 PH1 + ...	
	Set flip-flop DRQ	S/DRQ = BRPH10 + ... R/DRQ = ...	Inhibits transmission of another clock until data release signal received from core memory
PH10 DR	ENDE functions	See table 3-18	
			Mnemonic: AH (50, D0) SH (58, D8)

ADD WORD (AW; 30, B0) AND SUBTRACT WORD (SW; 38, B8). The AW and SW instructions add or subtract the effective word from the contents of private memory register R and load the result back into private memory register R.

General. The implementation of AW and SW is identical except for the arithmetic operation involved.

Condition Codes. If the result in the R-register is zero, the condition codes are set to XX00. If the result is non-zero and positive, the condition codes are set to XX10. A negative result produces condition code settings of XX01. Flip-flop CC2 is set if fixed-point overflow occurs during addition or subtraction. Flip-flop CC1 is set if there is a carry from bit position zero.

Trap Conditions. A trap to memory location X'43' occurs if there is fixed-point overflow and the fixed-point arithmetic mask bit is a one. The result in private memory register R remains unchanged. If overflow occurs and the mask bit is a zero, the next instruction in sequence is executed.

Add Word and Subtract Word Phase Sequences. Preparation phases for the two instructions are the same as the general PREP phases for word instructions, paragraph 3-59. Table 3-47 lists the detailed logic sequence during all AW and SW execution phases.

Table 3-47. AW and SW Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(C) : EW		Effective word
	(D) : EW		Effective word
	(A) : RR		Contents of private memory register R. Effective word will be added to this quantity
	(P) : Program address		Next instruction in sequence
	If AW, enable signal (S/SXAPD)	(S/SXAPD) = FAADD PRE/34 + ...	Preset adder for A + D ⟶ S in PH1
	If SW, enable signal (S/SXAMD)	(S/SXAMD) = FASUB PRE/34 + ...	Preset adder for A - D ⟶ S in PH1
	Set flip-flop MRQ	S/MRQ = (S/MRQ/1) + ... (S/MRQ/1) = FAS10 PRE/34 + ... FAS10 = FAARITH + ... R/MRQ = ...	Core memory request for next instruction in sequence
	Set flip-flop RW	S/RW = FAS11 PRE/34 NOL1 + ... FAS11 = FAARITH + ... R/RW = ...	Prepare to write result into private memory register R
PH1	One clock long		
	If AW, (A0-A31) + (D0-D31) ⟶ (S0-S31)	Adder logic set at last PREP clock	Add effective word to contents of register R and gate results to sum bus
			Mnemonic: AW (30, B0) SW (38, B8)

(Continued)

Table 3-47. AW and SW Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH1 (Cont.)	If SW, (A0-A31) - (D0-D31) ⟶ (S0-S31)	Adder logic set at last PREP clock	Subtract effective word from contents of register R and gate results to sum bus
	(S0-S31) ⟶ (RW0-RW31)	RWXS/0-RWXS/3 = RW + ...	Store result in private memory register R
	Set flip-flop CC1 if end carry from result	S/CC1 = K00 CC1XK00 + ... CC1XK00 = FAARITH PH1 + ... R/CC1 = CC1XK00 + ...	K00 is end carry from addition or end borrow from subtraction
	Set flip-flop CC2 if arithmetic overflow	S/CC2 = (S00 ⊕ S0) PROBOVER + ... PROBOVER = FAARITH PH1 + ... R/CC2 = PROBOVER + ...	Arithmetic overflow occurs when two numbers of like signs are added and their sum cannot be held in 32 bits
	Set flip-flop OVERIND	S/OVERIND/1 = PROBOVER + ... R/OVERIND/1 = CLEAR	Setting OVERIND/1 enables trap if overflow, and mask bit is equal to a one. Trap is set during ENDE
	Set flip-flop CC3 if result is positive and nonzero; otherwise reset CC3	S/CC3 = SGTZ TESTS + ... SGTZ = (S0 + S1 + ... + S31) NS0 + ... TESTS = FAS11 PH1 + ... R/CC3 = TESTS + ...	
	Set flip-flop CC4 if result is negative; otherwise reset CC4	S/CC4 = S0 TESTS + ... R/CC4 = TESTS + ...	
	Enable clock T11	NT5EN = RW + ... RW = Set at last PREP clock NT8EN = SXADD/1 RW + ... SXADD/1 = GXAD + GXNAD + ... GXAD = Set at last PREP clock if AW GXNAD = Set at last PREP clock if SW	Clock T11 is enabled by disabling clocks T5 and T8 Flip-flop GXAD is part of adder logic Flip-flop GXNAD is part of adder logic
	Branch to PH10	BRPH10 = FAS10 PH1 + ...	
	Set flip-flop DRQ	S/DRQ = BRPH10 + ... R/DRQ = ...	Inhibits transmission of another clock until data release signal received from core memory
PH10 DR	ENDE functions	See table 3-18	

Mnemonic: AW (30, B0)
SW (38, B8)

ADD DOUBLEWORD (AD; 10, 90) AND SUBTRACT DOUBLEWORD (SD; 18, 98). The AD and SD instructions add or subtract the effective doubleword from the contents of private memory registers R and Ru1, treated as a doubleword value, and load the result back into private memory registers R and Ru1. The R field of the instruction word must specify an even private memory register for a correct result.

General. The implementation of AD and SD is identical except for the arithmetic operation involved.

Condition Codes. If the result in the private memory registers is zero, the condition codes are set to XX00. If the result is nonzero and positive, the condition codes are set to XX10. A negative result produces condition code settings of XX01. Flip-flop CC2 is set if there is fixed-point overflow during the addition or subtraction. Flip-flop CC1 is set if there is a carry from bit position zero.

Trap Conditions. A trap to memory location X'43' occurs if there is a fixed-point overflow and the fixed-point arithmetic mask bit is a one. The result in private memory remains unchanged. If overflow occurs and the mask bit is a zero, the next instruction in sequence is executed.

Add Doubleword and Subtract Doubleword Phase Sequences. Preparation phases for the two instructions are the same as the general PREP phases for doubleword instructions, paragraph 3-59. Figure 3-143 shows the simplified phase sequence for the two instructions during execution, and table 3-48 lists the detailed logic sequence during all AD and SD execution phases.

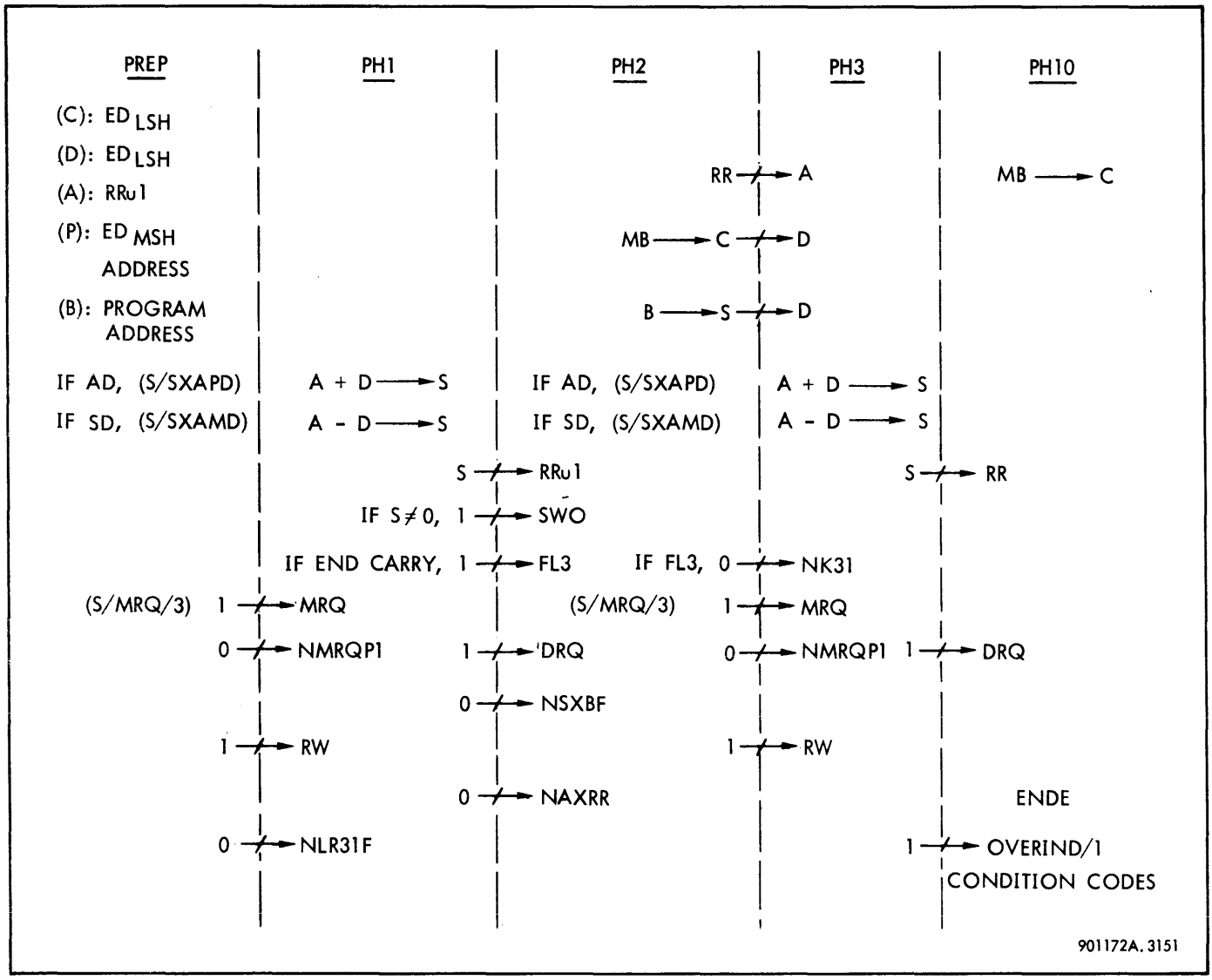


Figure 3-143. Add Doubleword and Subtract Doubleword Instruction, Phase Diagram

Table 3-48. Add Doubleword and Subtract Doubleword Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(C) : ED_{LSH}		Least significant half of effective doubleword
	(D) : ED_{LSH}		Least significant half of effective doubleword
	(A) : RRu1		Contents of private memory register Ru1. This is the least significant word of operand stored in private memory
	(P) : ED_{MSH} address		Address of most significant word of effective doubleword
	(B) : Program address		Address of next instruction in sequence
	If AD, enable signal (S/SXAPD)	(S/SXAPD) = FAADD PRE/34 + ...	Preset adder for A + D ⟶ S in PH1
	If SD, enable signal (S/SXAMD)	(S/SXAMD) = FASUB PRE/34 + ...	Preset adder for A - D ⟶ S in PH1
	Set flip-flop MRQ	S/MRQ = (S/MRQ/3) + ... (S/MRQ/3) = FADW/1 PRE/34 + ... FADW/1 = OU1 FAS11 FAS11 = FAARITH + ... R/MRQ = ...	Core memory request for most significant word of doubleword. Flip-flop DRQ set on next clock
	Reset flip-flop NMRQP1	S/NMRQP1 = N(S/MRQ/3) + ... R/NMRQP1 = ...	Used to delay setting flip-flop DRQ
	Set flip-flop RW	S/RW = FAS11 PRE/34 NOL1 + ... R/RW = ...	Prepare to write least significant word of result into private memory register Ru1
	Reset flip-flop NLR31F	S/NLR31F = N(S/LR31) (S/LR31) = FADW/1 NANLZ PRE3 + ... R/NLR31F = ...	Force a one on private memory address line LR31 during PH1 to select private memory register Ru1
			Mnemonic: AD (10, 90) SD (18, 98)

(Continued)

Table 3-48. Add Doubleword and Subtract Doubleword Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH1	One clock long		
T11L	If AD, (A0-A31) + (D0-D31) ⟶ (S0-S31)	Adder logic set at last PREP clock	Add least significant word of effective doubleword to least significant word of private memory doubleword, and gate result to sum bus
	If SD, (A0-A31) - (D0-D31) ⟶ (S0-S31)	Adder logic set at last PREP clock	Subtract least significant word of effective doubleword from least significant word of private memory doubleword, and gate result to sum bus
	(S0-S31) ⟶ (RW0-RW31)	RWXS/0-RWXS/3 = RW + ... RW = Set at last PREP clock	Store least significant word of result in private memory register Ru1
	Reset flip-flop NSXBF	S/NSXBF = N(S/SXB) (S/SXB) = FADW/1 PH1 + ... R/NSXBF = ...	Preset logic for B ⟶ S in PH2
	Reset flip-flop NAXRR	S/NAXRR = N(S/AXRR) (S/AXRR) = FADW/1 PH1 + ... R/NAXRR = ...	Preset logic for transferring most significant word of private memory doubleword to A-register in PH2
	Set flip-flop SW0 if (S0-S31) is nonzero	S/SW0 = NS0031Z (S/SW0/NZ) + ... NS0031Z = (S0 + S1 + ... + S31) (S/SW0/NZ) = K00HOLD + ... K00HOLD = FADW/1 PH1 + ... R/SW0 = ...	Used in setting CC3 in PH4. CC1 through CC4 may be set in this phase, but action is meaningless since they are also set in PH4
	Set flip-flop FL3 if end carry	S/FL3 = K00 K00HOLD + ... R/FL3 = ...	K00 is end carry or end borrow from adding or subtracting the least significant words of the two operands. Flip-flop NK31 will be reset in PH2 if end carry exists
	Enable clock T11	NT5EN = RW + ... RW = Set at last PREP clock	Clock T11 is enabled by disabling clocks T5 and T8
			Mnemonic: AD (10, 90) SD (18, 98)

(Continued)

Table 3-48. Add Doubleword and Subtract Doubleword Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH1 T11L (Cont.)		NT8EN = SXADD/1 RW + ... SXADD/1 = GXAD + GXNAD + ... GXAD = Set at last PREP clock if AD GXNAD = Set at last PREP clock if SD	Flip-flop GXAD is part of adder logic Flip-flop GXNAD is part of adder logic
	Set flip-flop DRQ	S/DRQ = MRQP1 + ... R/DRQ = ...	MRQP1 was set on previous clock. DRQ inhibits transmission of another clock until data release signal received from core memory
PH2 DR	One clock long (B0-B31) ⟶ (S0-S31) (S15-S31) ⟶/ (P15-P31) (MB0-MB31) ⟶ (C0-C31) ⟶/ (D0-D31) (RR0-RR31) ⟶/ (A0-A31)	SXB = NDIS SXBF + ... SXBF = Set at PH1 clock CXMB = DG = /DG/ DXC = FADW/1 PH2 + ... AXRR = Set at PH1 clock	Transfer program address to P-register Transfer most significant word of effective doubleword to D-register Transfer most significant word of private memory doubleword to A-register
	If AD, enable signal (S/SXAPD)	(S/SXAPD) = FAADD PH2 + ...	Preset adder for A + D ⟶ S in PH3
	If SD, enable signal (S/SXAMD)	(S/SXAMD) = FASUB PH2 + ...	Preset adder for A - D ⟶ S in PH3
	Set flip-flop MRQ	S/MRQ = (S/MRQ/3) + ... (S/MRQ/3) = FADW/1 PH2 + ... R/MRQ = ...	Core memory request for next instruction in sequence
	Reset flip-flop NMRQP1	S/NMRQP1 = N(S/MRQ/3) + ... R/NMRQP1 = ...	Used to delay setting flip-flop DRQ
	Set flip-flop RW	S/RW = FAS11 PH2 NOL1 + ... R/RW = ...	Prepare to write most significant word of result into private memory register R
	Reset flip-flop NK31 if there was end carry in PH2	S/NK31 = N(S/K31) N(S/SXAMD/1) + N(S/K31/1)	Provides carry to most significant word addition in PH3
			Mnemonic: AD (10, 90) SD (18, 98)

(Continued)

Table 3-48. Add Doubleword and Subtract Doubleword Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH2 DR (Cont.)		(S/K31) = FADW/1 PH2 + ... (S/K31/1) = K00 (S/K31/3) + ... + ... (S/K31/3) = N(FADW/1 PH2 NFL3) R/NK31 = ...	
PH3 T11L	One clock long If AD, (A0-A31) + (D0-D31) ⟶ (S0-S31)	Adder logic set at PH1 clock	Add most significant word of effective doubleword to most significant word of private memory doubleword, and gate result to sum bus. Carry to least significant bit is present if flip-flop NK31 was reset in PH2
	If SD, (A0-A31) - (D0-D31) ⟶ (S0-S31)	Adder logic set at PH1 clock	Subtract most significant word of effective doubleword from most significant word of private memory doubleword, and gate result to sum bus. Borrow from least significant bit is present if flip-flop NK31 was reset in PH2
	(S0-S31) ⟶ (RW0-RW31)	RWXS/0-RWXS/3 = RW + ... RW = Set at PH2 clock	Store most significant word of result in private memory register R
	Set flip-flop CC1 if end carry from result	S/CC1 = K00 CC1XK00 + ... CC1XK00 = FAARITH PH3 + ... R/CC1 = CC1XK00 + ...	K00 is end carry from addition
	Set flip-flop CC2 if arithmetic overflow	S/CC2 = (S00 + S0) PROBOVER + ... PROBOVER = FAARITH PH3 + ... R/CC2 = PROBOVER + ...	Arithmetic overflow occurs when two numbers of like signs are added and their sum cannot be held in 32 bits
	Set flip-flop OVERIND/1	S/OVERIND/1 = PROBOVER + ... R/OVERIND/1 = CLEAR	Setting OVERIND/1 enables trap if overflow, and mask bit is equal to a one. Trap is set during ENDE
			Mnemonic: AD (10, 90) SD (18, 98)

(Continued)

Table 3-48. Add Doubleword and Subtract Doubleword Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH3 T11L (Cont.)	Set flip-flop CC3 if result is positive and nonzero; otherwise reset CC3	S/CC3 = SGTZ TESTS + ...	
		SGTZ = (S0 + S1 + ... + S31) NS0 + ...	
		TESTS = FAS11 PH3 + ...	
		R/CC3 = TESTS + ...	
	Set flip-flop CC4 is result is negative; otherwise reset CC4	S/CC4 = S0 TESTS + ...	
		R/CC4 = TESTS + ...	
	Enable clock T11	NT5EN = RW + ...	Clock T11 is enabled by disabling clocks T5 and T8
		RW = Set at PH2 clock	
		NT8EN = SXADD/1 RW + ...	
		SXADD/1 = GXAD + GXNAD + ...	
		GXAD = Set at PH2 clock if AD	Flip-flop GXAD is part of adder logic
		GXNAD = Set at PH2 clock if SD	Flip-flop GXNAD is part of adder logic
	Branch to PH10	BRPH10 = FADW/1 PH3 + ...	
	Set flip-flop DRQ	S/DRQ = BRPH10 + MRQP1 + ...	Inhibits transmission of another clock until data release signal received from core memory
		R/DRQ = ...	
PH10 DR	ENDE functions	See table 3-18	

Mnemonic: AD (10, 90)
SD (18, 98)

3-67 Family of Multiply Instructions (FAMUL)

GENERAL. Multiplication in the Sigma 5 computer is done with the Multiply Immediate, Multiply Halfword, and Multiply Word instructions. The multiplicand is located either in a specified memory location or, in the case of the Multiply Immediate instruction, in bits 12 through 31 of the instruction word. The multiplier is located in a specified private memory register. The product is stored in private memory.

The Sigma 5 computer uses the bit-pair method of multiplication, in which the multiplier is examined two bits at a time and one addition or one subtraction is performed for that bit pair. With each addition or subtraction the partial product is shifted two bit positions to the right. An example of bit-pair multiplication is shown in figure 3-144.

There are four possible states for one bit pair: 00, 01, 10, and 11. Multiplying by the first three types of bit pairs is done by normal shift and add operations. Multiplying by bit pair 00 is done by adding zeros to the partial product; multiplying by bit pairs 01 and 10 is done by adding the multiplicand or two times the multiplicand, respectively, to the partial product. Multiplying by bit pair 11 is a special case. Multiplication by 3 cannot be represented by a multiple of 2; therefore, simply shifting the multiplicand and adding is not possible in this case. To multiply by this bit pair, 1 times the multiplicand is subtracted from the partial product during one iteration, and 4 times the multiplicand is added to the partial product during the next iteration. Adding 4 times the multiplicand is accomplished by adding 1 to the next higher bit pair at the time that bit pair is under examination. The next bit pair becomes 01 if it was 00, 10 if it was 01, 11 if it was 10, or 00 with a 1 to be added to the next bit pair if it was 11. The two multiplier bits to be examined are in bits 30 and 31 of the B-register, and a 1, or carry, to the next higher bit pair is saved in flip-flop BC31 until that bit pair comes under examination.

Table 3-49 shows all possible combinations of bit pairs and carries, the weight of each bit pair with its carry, and the manner in which each weight is implemented. During the multiplication iterations, the multiplicand is in the C- and D-registers, and the partial product is in the A- and B-registers, with the most significant half in the A-register. When zeros are to be added to the partial product, the contents of the A-register are simply placed on the sum bus and shifted right two bit positions into the A- and B-registers. When 1 times the multiplicand is to be added to the partial product, the contents of the A- and D-registers are added together. When 2 times the multiplicand is to be added, the multiplicand is shifted left one bit position in the C-register and placed in the D-register before adding to the partial product. When -1 times the multiplicand is to be added, the two's complement of the multiplicand in the D-register is added to the partial product in the A-register. The shift and add operations take place 16 times in the case of immediate and word operation (32-bit multiplier) and 8 times in the case of halfword operation (16-bit multiplier). These iterations are brought about by 16 or 8 repetitions of phase 6 of the instruction. The number of iterations is controlled by counting the macro-counter down from 16 or 8 to zero.

A multiply instruction may be interrupted for input/output operation during any one of the iteration phases up to the last four iterations. If such an interrupt takes place, the adder output is shifted right two bit positions into the A-register, but the B-register remains stationary. In order to save the two least significant bits from the adder, which would normally be shifted into the B-register, these bits are clocked into flip-flops FL1 and FL2. When the I/O operation is complete, the outputs of FL1 and FL2 are clocked into B0 and B1, where these bits would have been if the interrupt had not occurred.

MULTIPLY IMMEDIATE (MI; 23) AND MULTIPLY WORD (MW; 37, 77). The MI and MW instructions are identical except for the preparation phases, and will therefore be discussed as one instruction. The MI instruction uses as the multiplicand the value in bits 12 through 31 of the instruction word, treated as a 20-bit integer with the sign extended left to bit 0. The MW instruction takes the multiplicand from the core memory location specified by the reference address field of the instruction word. In both cases the multiplier is taken from the private memory register specified by the R field of the instruction word plus 1 if the R field is even, or from the register specified by the R field if the R field contains an odd number. If the R field contains an even number, the 32 high-order bits of the product are loaded into register R and the 32 low-order bits are loaded into register R plus 1. If the R field contains an odd number, the 32 low-order bits of the product are loaded into register R, and a 64-bit product cannot be generated. Condition code bit 4 is set if the product is negative, CC3 is set if the product is positive, and CC2 is set if the product is not correctly representable in register Ru1 alone.

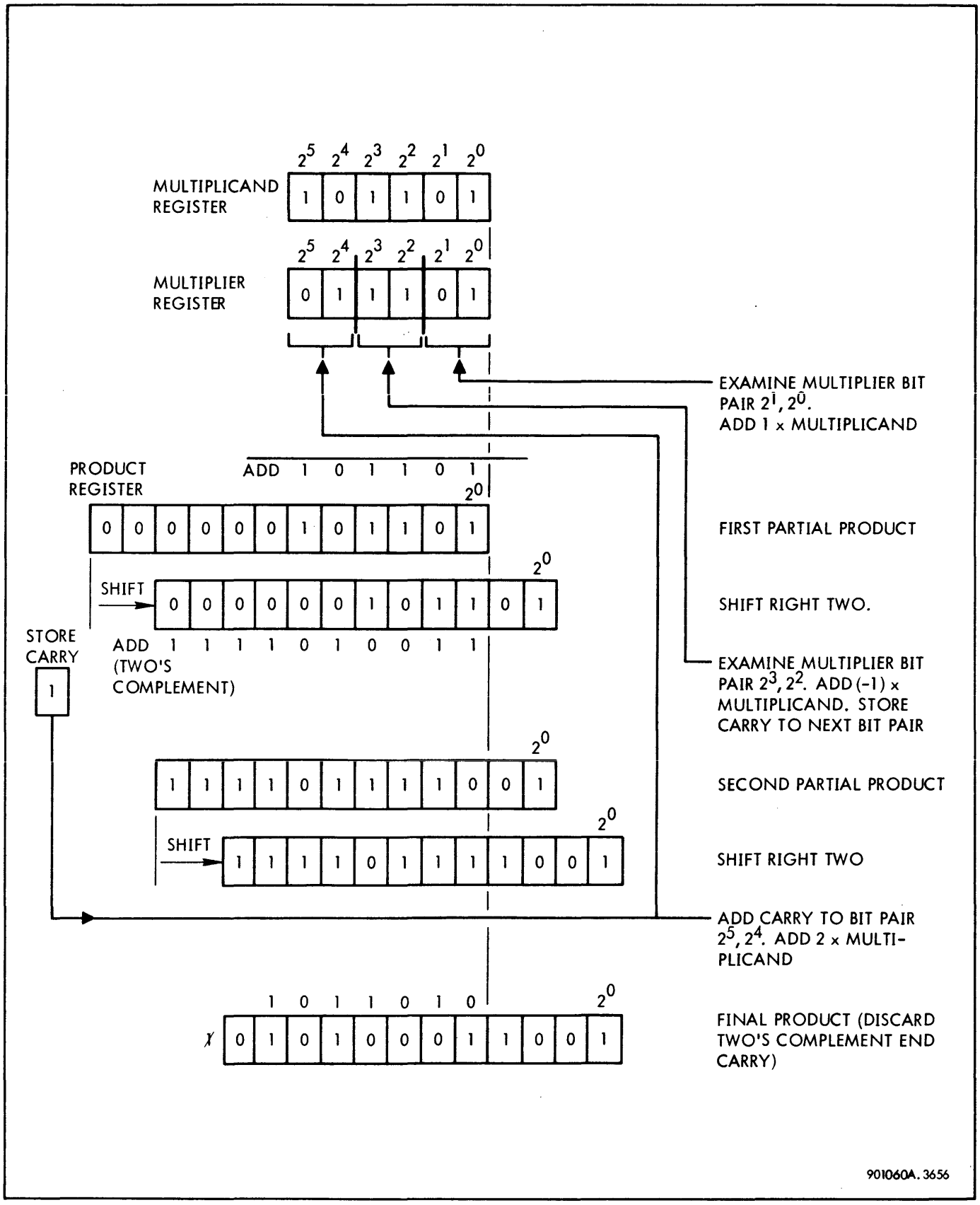


Figure 3-144. Bit-Pair Multiplication

Table 3-49. Bit Weights and Operations for Bit-Pair Multiplication

B30	B31	BC31	Weight	Implementation of Weight	Operation						
0	0	0	0	0		2C→D			S/A→S		BX1/4→B
0	0	1	1	1	C→D		S/A+D→S				BX1/4→B
0	1	0	1	1	C→D		S/A+D→S				BX1/4→B
0	1	1	2	2		2C→D	S/A+D→S				BX1/4→B
1	0	0	2	2		2C→D	S/A+D→S				BX1/4→B
1	0	1	3	-1 +4	C→D			S/A-D→S		1→BC31	BX1/4→B
1	1	0	3	-1 +4	C→D			S/A-D→S		1→BC31	BX1/4→B
1	1	1	4	0 +4		2C→D			S/A→S	1→BC31	BX1/4→B

Preparation phases for the MI instruction are the same as the general PREP phases for immediate instructions, paragraph 3-59; preparation phases for the MW instruction are the same as the general PREP phases for word instructions. Figure 3-145 shows the simplified phase sequence for the MI and MW instructions. Table 3-50 lists the logic sequence during all the execution phases of these instructions.

MULTIPLY HALFWORD (MH; 57, D7). The MH instruction multiplies the effective halfword by bits 16 through 31 of the contents of the private memory register specified in the R field of the instruction. The product is stored in the private memory register specified by the R field plus 1 if the R field is even or in the register specified by the R field if the R field is odd. Condition code flip-flop CC4 is set if the result is negative; flip-flop CC3 is set if the result is positive.

Preparation phases for the MH instruction are the same as the general PREP phases for halfword instructions, paragraph 3-59. Figure 3-146 shows the simplified phase sequence for the MH instructions. Table 3-51 lists the logic sequence during all the execution phases of the instruction.

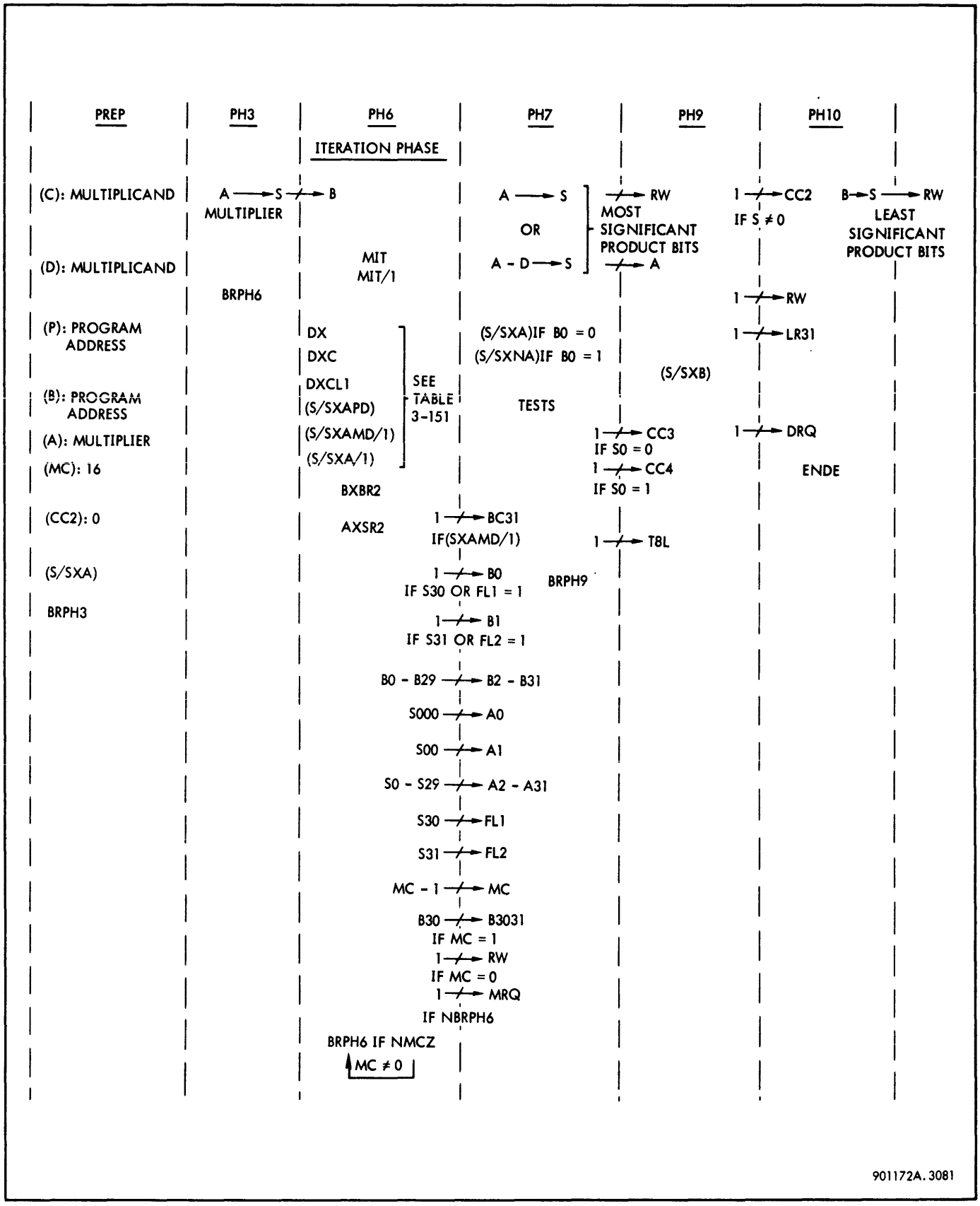


Figure 3-145. Multiply Immediate and Multiply Word Instructions, Phase Sequence Diagram

Table 3-50. Multiply Immediate and Multiply Word Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(C): Multiplicand (sign padded if MI)		
	(D): Multiplicand (sign padded if MI)		
	(P): Program address		
	(B): Program address		
	(A): Multiplier (RRu1)		
	(MC): 16		
	(CC2): 0		
	Enable signal (S/SXA)	(S/SXA) = FAMUL PRE/34 + ...	
		FAMUL = OU3 OL7 + OU2 OL3	
	Branch to PH3	BRPH3 = FAMDS NBRPH5 NANLZ PRE/34 + ...	
		FAMDS = FAMULNH + ...	
		FAMULNH = OU3 OL7 + OU2 OL3	
PH3	One clock long		
T5L	(A0-A31) ⟶ (S0-S31)	Adder preset at last PREP clock	Transfer multiplier to B-register
	(S0-S31) ⟶/ (B0-B31)	BXS = FAMUL PH3 + ...	
	Branch to PH6	BRPH6 = FAMULNH PH3 + ...	
PH6	One clock long		
T5L	Iteration phase – repeated until MC = 0		
	Enable signal MIT	MIT = FAMUL PH6	Control signal to handle direct logic
	Enable signal MIT/1	MIT/1 = FAMUL NIOEN (PH6 + S/PH6/IO)	Control signal to handle preset logic. S/PH6/IO is true when returning from I/O operation
			Mnemonic: MI(23), MW (37, 77)

(Continued)

Table 3-50. Multiply Immediate and Multiply Word Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH6 T5L (Cont.)	Preset logic for register control:	See table 3-150	
	Reset input for D-register flip-flops	DX = MIT/1 + DXC + ...	To place zeros in D-register when transferring data into register and previous bit content was 1
	(C0-C31) $\not\longrightarrow$ (D0-D31)	DXC = MIT/1 (B31 ⊕ BC31) + ...	Multiplicand into D-register
	2(C0-C31) $\not\longrightarrow$ (D0-D31)	DXCL1 = MIT/1 NDXC	Prepare to add 2 times the multiplicand to partial product (insignificant when (S/SXA/1) is true)
	(A0-A31) + (D0-D31) $\longrightarrow$ (S0-S31)	(S/SXAPD) = MIT/1 N(S/AXAMD/1) N(S/SXA/1) + ...	Add adjusted multiplicand to partial product
	(A0-A31) - (D0-D31) $\longrightarrow$ (S0-S31)	(S/SXAMD/1) = MIT/1 B30 (B31 ⊕ BC31) + ...	Add two's complement of multiplicand to partial product
	(A0-A31) $\longrightarrow$ (S0-S31)	(S/SXA/1) = MIT/1 (NB30 NB31 NBC31 + B30 B31 BC31)	Add zeros to partial product
	Set flip-flop BC31	(S/BC31) = (S/SXAMD/1)	Carry 1 to next higher bit pair
		(R/BC31) = MIT/1 NB30 + CLEAR	
	Set flip-flop B0	S/B0 = BXBR2 S30/1 + ...	Shift partial product right two bit positions into B-register. Bits are in FL1 and FL2 if returning from I/O operation
		S30/1 = B0001EN/1 S30 + (S/PH6/IO) FL1	
	Set flip-flop B1	S/B1 = BXBR2 S31/1 + ...	
		S31/1 = B0001EN/1 S31 + (S/PH6/IO) FL2	
	(B00-B29) $\not\longrightarrow$ (B02-B31)	BXBR2 = MIT/1 + ...	
	Direct logic for register control:		
	S000 $\not\longrightarrow$ A0 S00 $\not\longrightarrow$ A1 (S00-S29) $\not\longrightarrow$ (A02-A31)	AXSR2 = MIT + ...	Shift partial product right two bit positions into A-register
	S30 $\not\longrightarrow$ FL1	S/FL1 = S30 MIT + ...	Two-bit extension of A-register for I/O operation
		R/FL1 = MIT + CLEAR	
			Mnemonic: MI(23), MW(37, 77)

(Continued)

Table 3-50. Multiply Immediate and Multiply Word Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH6 T5L (Cont.)	S31—/→FL2	S/FL2 = S31 MIT + ...	
		R/FL2 = MIT + CLEAR	
	General control functions:		
	MC - 1—/→MC	MCDC7 = MIT/1 + ...	Decrement macro-counter 16 times to provide required number of iterations
	Sustain PH6 until MC = 0	BRPH6 = FAMDS PH6 NMCZ NBRPH10 NFSHEX + ...	
	On the next to last clock:		
	B30—/→B3031 if MC = 1	S/B3031 = B30 MIT (MC = 1)	Extend multiplier sign bit two bit positions to the left on next to final clock. Bit 28 and 29 are 0 at this time and will not interfere
	Final clock:		
	Enable signal (S/SXAMD/1) if negative multiplier and BC31 = 0	(S/SXAMD/1) = MIT/1 B30 (B31 ⊕ BC31) + ...	Sets up sign iteration logic. If positive multiplier, BC31 cannot be 1 because there can be no carry from the previous bit pair, containing the sign bit, if the sign bit is 0
	Enable signal (S/SXA/1) if positive multiplier and BC31 = 0 or negative multiplier and BC31 = 1	(S/SXA/1) = MIT/1 (NB30 NB31 NBC31 + B30 B31 BC31)	
	Set flip-flop RW	S/RW = (S/RW) (S/RW) = MIT MCZ + ... R/RW = ...	Prepare to write into private memory
	Set flip-flop MRQ	S/MRQ = (S/MRQ/1) + ... (S/MRQ/1) = FAMDS PH6 NBRPH6 NIOEN R/MRQ = ...	Request for core memory cycle to read next instruction
	I/O service call:		
	Enable signal IOEN6	IOEN6 = FAMDS PH6 NFPRR NFSHEX IOEN6/1 + ... IOEN6/1 = MC0005Z + ...	Enable I/O service call if MC ≥ 0
	Inhibit PH6	S/PH6 = BRPH6 NCLEAR NIOEN R/PH6 = ...	Proceed to I/O phases
			Mnemonic: MI(23), MW(37, 77)

(Continued)

Table 3-50. Multiply Immediate and Multiply Word Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH6 T5L (Cont.)	Inhibit I/O from performing C → D so that MUL instruction can perform C → D or 2C → D	DXC = IOPH1 SW13 NBXBR2 + ...	This input to DXC is low at this time
	FL1 → B0	S/B0 = BXBR2 S30/1 + ...	Last two partial product bits stored in FL1 and FL2 during I/O operation
	FL2 → B1	S30/1 = (S/PH6/IO) FL1 + ...	
		S/B1 = BXBR2 S31/1 + ...	
		S31/1 = (S/PH6/IO) FL2 + ...	
PH7	One clock long		
T5L	(A0-A31) → (S0-S31) or (A0-A31) - (D0-D31) → (S0-S31)	Logic preset in PH6	Store 32 high-order bits of product in private memory register R
	(S0-S31) → (RW0-RW31)	RWXS = RW	
	(S0-S31) → (A0-A31)	AXS = FAMULNH PH7 + ...	Place final product in A-register for magnitude test
	Enable signal (S/SXA) if B0 = 0	(S/SXA) = FAMULNH PH7 NB0 + ...	Preset for A-register contents on sum bus for magnitude test in PH9
	Enable signal (S/SXNA) if B0 = 1	(S/SXNA) = FAMULNH PH7 B0 + ...	Preset for one's complement of A-register contents on sum bus for magnitude test in PH9
	Enable signal TESTS	TESTS = FAMULNH PH7	Enable S-register test to set condition code
	Set flip-flop CC3 if S0 = 0	S/CC3 = SGTZ TESTS + ...	S0 = 0 indicates positive product
		SGTZ = N(S0 NFACOMP) (NS0007Z + NS0815Z + NS1631Z + NS3263Z)	
	Set flip-flop CC4 if S0 = 0	S/CC4 = (S/CC4/2) TESTS	
		(S/CC4/2) = S0 NFACOMP	S0 = 1 indicates negative product
			Mnemonic: MI(23), MW(37, 77)

(Continued)

Table 3-50. Multiply Immediate and Multiply Word Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH7 T5L (Cont.)	Reset flip-flop NT8L	S/NT8L = N(S/T8L) (S/T8L) = FAMULNH PH7 R/NT8L = ...	Set clock T8L for PH9
	Branch to PH9	BRPH9 = FAMULNH PH7 + ...	
PH9	One clock long		
T8L	Set flip-flop CC2 if S ≠ 0	S/CC2 = NS0031Z (S/CC2/NZ) + ... S/CC2/NZ = FAMULNH PH9	S = 0 indicates top 33 product bits are not the same, therefore result is not correctly representable in register Ru1 alone
	Set flip-flop RW	S/RW = (S/RW) (S/RW) = FAMDS PH9 + ... R/RW = ...	Prepare to write into private memory
	Set flip-flop LR31	(S/LR31) = FAMULNH PH9 + ...	Set least significant bit of private memory address lines to access register Ru1
	Enable signal (S/SXB)	(S/SXB) = FAMULNH PH9 + ...	Preset for B ⟶ S in PH10
	Set flip-flop DRQ	S/DRQ = (S/DRQ) (S/DRQ/2) = PH9 + ... R/DRQ = ...	Data request, inhibiting transmission of another clock until data release received from memory
PH10	One clock long		
T5L	(B0-B31) ⟶ (S0-S31)	Logic preset in PH9	Place least significant 32 product bits on sum bus
	(S0-S31) ⟶/⟶ (RW0-RW31)	RWXS = RW	Write least significant 32 product bits in register Ru1
	ENDE functions		

Mnemonic: MI(23), MW(37, 77)

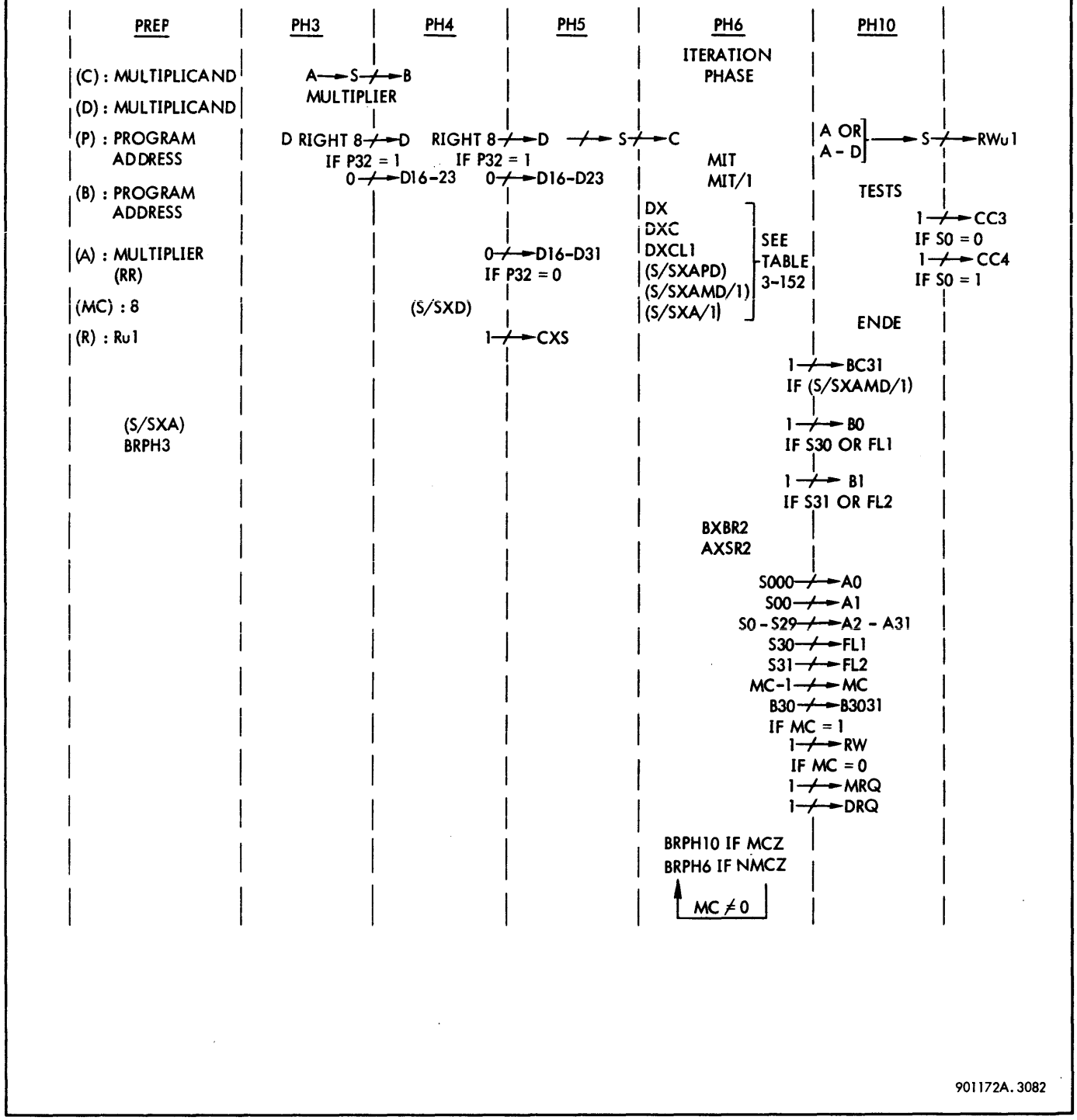


Figure 3-146. Multiply Halfword Instruction, Phase Sequence Diagram

Table 3-51. Multiply Halfword Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(C) : Multiplicand		
	(D) : Multiplicand		
	(P) : Program address		
	(B) : Program address		
	(A) : Multiplier (RR)		
	(MC) : 8		
	(R) : Ru1		
	Enable signal (S/SXA)	(S/SXA) = FAMUL PRE/34 + ...	
	Branch to PH3	BRPH3 = FAMDS NBRPH5 NANLZ PRE/34 + ...	
PH3	One clock long		
T5L	(A0-A31) ⟶ (S0-S31)	Adder preset at last PREP clock	Transfer multiplier to B-register
	(S0-S31) ⟶/⟶ (B0-B31)	BXS = FAMUL PH3 + ...	
	Right cycle D-register 1 byte if P32 = 1	DXDR8 = FUMH PH3 P32 + ...	First half of up alignment of halfword to bits 0 through 15. P32 = 1 indicates that the least significant halfword will be used as the multiplicand
		FUMH = OU5 OL7	
	0 ⟶/⟶ (D16-D23)	DXDR8/2 = DXDR8 NFUMH (10W)	
PH4	One clock long		
T5L	Right cycle D-register 1 byte if P32 = 1	DXDR8 = FUMH P32 (PH4 + ...) + ...	Multiplicand is now in most significant 32 bits of D-register
	0 ⟶/⟶ (D16-D23)	DXDR8/2 = DXDR8 NFUMH	NFUMH is false at this time. DXDR8/2 shifts data into bits 16 through 23 of D-register
			Mnemonic: MH(57, D7)

(Continued)

Table 3-51. Multiply Halfword Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH4 T5L (Cont.)	0 → (D16-D31) if P32 = 0	DX/2 = FUMH PH4 + ... DX/3 = FUMH PH4 + ...	DX/2 clears bits 16 through 23 of D-register. DX/3 clears bits 17 through 31 of D-register. If P32 = 0, halfword to be multiplied is in most significant half of original multiplicand
	0 → (B8-B15)	BX/1 = BX/4 + ... BX/4 = FUMH PH4 + ...	BX/1 clears bits 8 through 15 of B-register so that when MC = 1 in PH6, B28 and B29 will be clear for sign extension
	Enable signal (S/SXD)	(S/SXD) = FUMH PH4 + ...	Preset adder for D → S in PH5
	Reset flip-flop NCXS	S/NCXS = N(S/CXS) (S/CXS) = FUMH PH4 + ... R/NCXS = ...	Preset for S → C in PH5
PH5	One clock long		
T5L	(D0-D31) → (S0-S31)	Logic preset in PH4	Transfer aligned multiplicand to C-register
	(S0-S31) → (C0-C31)	Logic preset in PH4	
PH6	One clock long		
T5L	Iteration phase – repeated until MC = 0		
	Enable signal MIT	MIT = FAMUL PH6	Control signal to handle direct logic
	Enable signal MIT/1	MIT/1 = FAMUL NIOEN (PH6 + S/PH6/IO)	Control signal to handle preset logic. S/PH6/IO is true when returning from I/O operation
	Preset logic for register control:	See table 3-49	
	Reset input for D-register flip-flops	DX = MIT/1 + DXC + ...	To place zeros in D-register when transferring data into register and previous bit content was 1
			Mnemonic: MH(57, D7)

(Continued)

Table 3-51. Multiply Halfword Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH6 T5L (Cont.)	(C0-C31) $\not\rightarrow$ (D0-D31)	DXC = MIT/1 (D31 ⊕ BC31) + ...	Multiplicand into D-register
	2(C0-C31) $\not\rightarrow$ (D0-D31)	DXCL1 = MIT/1 NDXC	Prepare to add 2 times the multiplicand to partial product (insignificant when (S/SXA/1) is true)
	(A0-A31) + (D0-D31) $\rightarrow$ (S0-S31)	(S/SXAPD) = MIT/1 N(S/AXAMD/1) N(S/SXA/1) + ...	Add adjusted multiplicand to partial product
	(A0-A31) - (D0-D31) $\rightarrow$ (S0-S31	(S/SXAMD/1) = MIT/1 B30 (B31 ⊕ BC31) + ...	Add two's complement of multiplicand to partial product
	(A0-A31) $\rightarrow$ (S0-S31)	(S/SXA/1) = MIT/1 (NB30 NB31 NBC31 + B30 B31 BC31)	Add zeros to partial product
	Set flip-flop BC31	S/BC31 = (S/BC31) (S/BC31) = (S/SXAMD/1) (R/BC31) = MIT/1 NB30 + CLEAR	Carry 1 to next higher bit pair
	Set flip-flop B0	S/B0 = BXBR2 S30/1 + ... S30/1 = B0001EN/1 S30 + (S/PH6/IO) FL1	Shift partial product right two bit positions into B-register. Bits are in FL1 and FL2 if returning from I/O operation
	Set flip-flop B1	S/B1 = BXBR2 S31/1 + ... S31/1 = B0001EN/1 S31 + (S/PH6/IO) FL2	
	(B00-B29) $\not\rightarrow$ (B02-B31)	BXBR2 = MIT/1 + ...	
	Register control – direct logic:		
	S000 $\not\rightarrow$ A0 S00 $\not\rightarrow$ A1 (S00-S29) $\not\rightarrow$ (A02-A31)	AXSR2 = MIT + ...	Shift partial product right two bit positions into A-register
	S30 $\not\rightarrow$ FL1	S/FL1 = S30 MIT + ... R/FL1 = MIT + CLEAR	Two-bit extension of A-register for I/O operation
	S31 $\not\rightarrow$ FL2	S/FL2 = S31 MIT + ... R/FL2 = MIT + CLEAR	
			Mnemonic: MH(57, D7)

(Continued)

Table 3-51. Multiply Halfword Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH6 T5L (Cont.)	General control functions:		
	MC - 1 → MC	MCDC7 = MIT/1 + ...	Decrement macro-counter 8 times to provide required number of iterations
	Sustain PH6 until MC = 0	BRPH6 = FAMDS PH6 NMCZ NBRPH10 NFSHEX + ...	
	On the next to last clock:		
	B30 → B3031 if MC = 1	S/B3031 = B30 MIT (MC = 1)	Extend multiplier sign bit two bit positions to the left on next to final clock. Bit 28 and 29 are are 0 at this time and will not interfere
	Final clock:		
	Enable signal (S/SXAMD/1) if negative multiplier and BC31 = 0	(S/SXAMD/1) = MIT/1 B30 (B31 ⊕ BC31) + ...	Sets up sign iteration logic. If positive multiplier, BC31 cannot be 1 because there can be no carry from the previous bit pair, containing the sign bit, if the sign bit is 0
	Enable signal (S/SXA/1) if positive multiplier and BC31 = 0 or negative multiplier and BC31 = 1	(S/SXA/1) = MIT/1 (NB30 NB31 NBC31 + B30 B31 BC31)	
	Set flip-flop RW	S/RW = (S/RW) (S/RW) = MIT MCZ + ... R/RW = ...	Prepare to write into private memory
	Set flip-flop MRQ	S/MRQ = (S/MRQ/1) + ... (S/MRQ/1) = FAMDS PH6 NBRPH6 NIOEN R/MRQ = ...	Request for core memory cycle to read next instruction
	Set flip-flop DRQ	S/DRQ = (S/DRQ) (S/DRQ) = BRPH10 + ... R/DRQ = ...	Data request inhibiting transmission of another clock until data release received from memory
	Branch to PH10	BRPH10 = FUMH PH6 MCZ + ...	
			Mnemonic: MH(57, D7)

(Continued)

Table 3-51. Multiply Halfword Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH6 T5L (Cont.)	I/O service call:		
	Enable signal IOEN6	IOEN6 = FAMDS PH6 NFPRR NFSHEX IOEN6/1 + ...	Enable I/O service call if MC $\geq$ 0
		IOEN6/1 = MC0005Z + ...	
	Inhibit PH6	S/PH6 = BRPH6 NCLEAR NIOEN	Proceed to I/O phases
		R/PH6 = ...	
	Inhibit I/O from performing C $\not\rightarrow$ D so that MUL instruction can perform C $\not\rightarrow$ D or 2C $\not\rightarrow$ D	DXC = IOPH1 SW13 NBXBR2 + ...	This input to DXC is low at this time
	FL1 $\not\rightarrow$ B0	S/B0 = BXBR2 S30/1 + ...	Last two partial product bits stored in FL1 and FL2 during I/O operation
		S30/1 = (S/PH6/IO) FL1 + ...	
	FL2 $\not\rightarrow$ B1	S/B1 = BXBR2 S31/1 + ...	
		S31/1 = (S/PH6/IO) FL2 + ...	
PH10	Sustained until data release		
DR	(A0-A31) $\rightarrow$ (S0-S31) or (A0-A31) - (D0-D31) $\rightarrow$ (S0-S31)	(S/SXA) or (S/SXAMD) preset in PH6	Transfer product to private memory register Ru1. R31 set in PRE3. Product bits shifted into B-register are insignificant
	(S0-S31) $\rightarrow$ (RW0-RW31)	RWXS = RW	
	Enable signal TESTS	TESTS = FUMH ENDE + ...	Enable S-register test to set condition code
	Set flip-flop CC3 if S0 = 0	S/CC3 = SGTZ TESTS + ...	S0 = 0 indicates positive product
		SGTZ = N(S0 NFACOMP) (NS0007Z + NS0815Z + NS1631Z + NS3263Z)	
	Set flip-flop CC4 if S0 = 1	S/CC4 = (S/CC4/2) TESTS	S0 = 1 indicates negative product
		(S/CC4/2) = S0 NFACOMP	
	ENDE functions		
			Mnemonic: MH(57, D7)

3-68 Family of Divide Instructions (FADIV)

GENERAL. Two division instructions are available for the Sigma 5 computer: Divide Halfword (DH) and Divide Word (DW). In both cases the numerator is in private memory and the denominator is in core memory. The quotient is stored in private memory.

Since the logic sequence for the DW instruction with an odd number in the R field is identical to the DH instruction, these two operations are discussed together. The Divide Word instruction with an even R field is discussed separately.

Nonrestoring Division. The Sigma 5 computer uses the nonrestoring division method for the DH and DW instructions. Multiples of the denominator are repeatedly subtracted from the numerator. The method differs from restoring division in that each subtraction is allowed regardless of whether the residue is positive or negative. If the residue is negative, a zero is placed in the quotient for that order and the next multiple of the denominator is added to, rather than subtracted from, the residue. Every time the residue is positive, a one is added to the appropriate order of the quotient and the next denominator multiple is subtracted. The result is the same as in restoring division. The total of all multiples subtracted minus the total of all multiples added plus the remainder equals the numerator. The zero point (residue = 0) is approached from both sides.

An example of the additions and subtractions used in nonrestoring division is shown in figure 3-147. A graphic representation of the process, showing the movement of the residue on both sides of zero, is shown in figure 3-148.

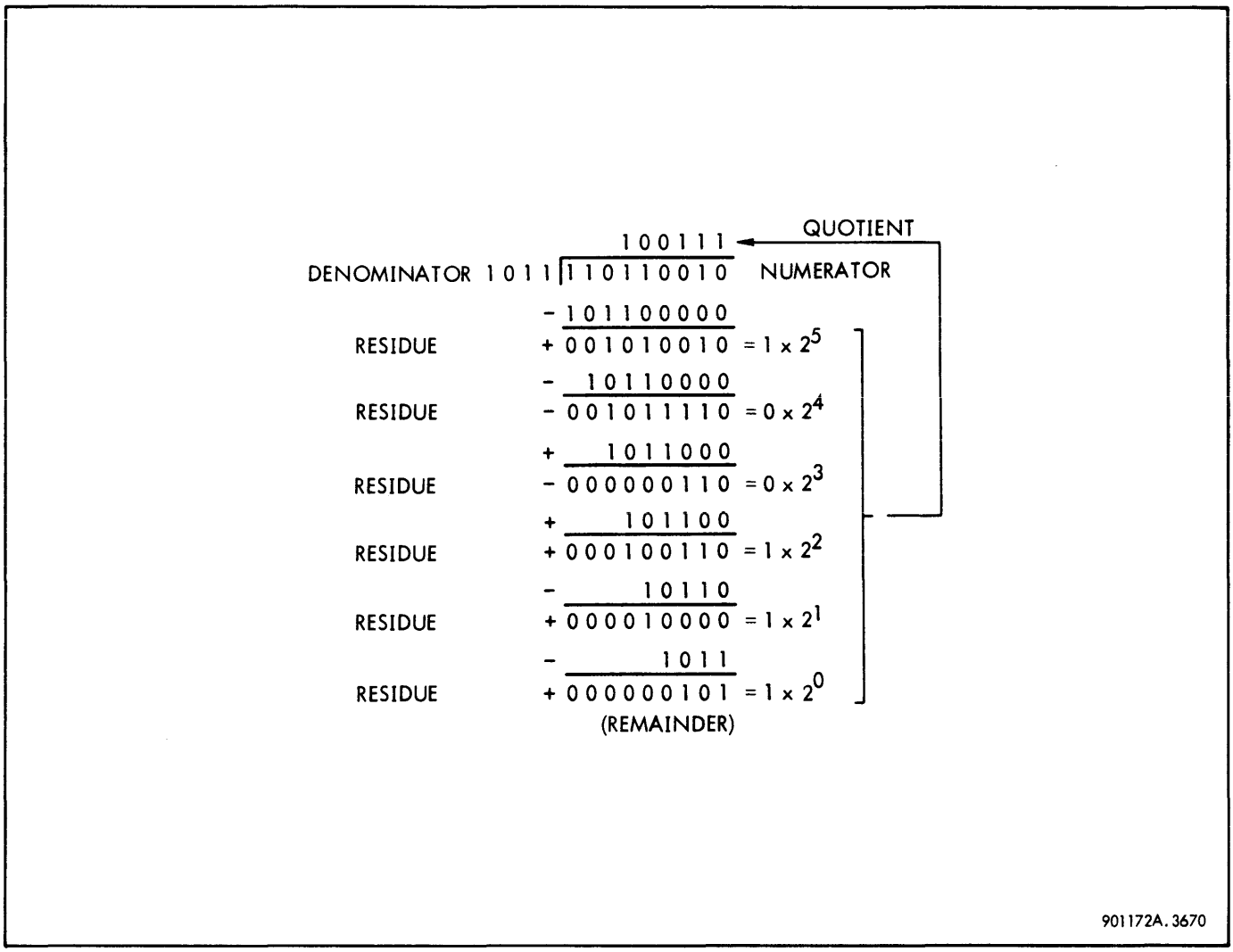


Figure 3-147. Nonrestoring Division

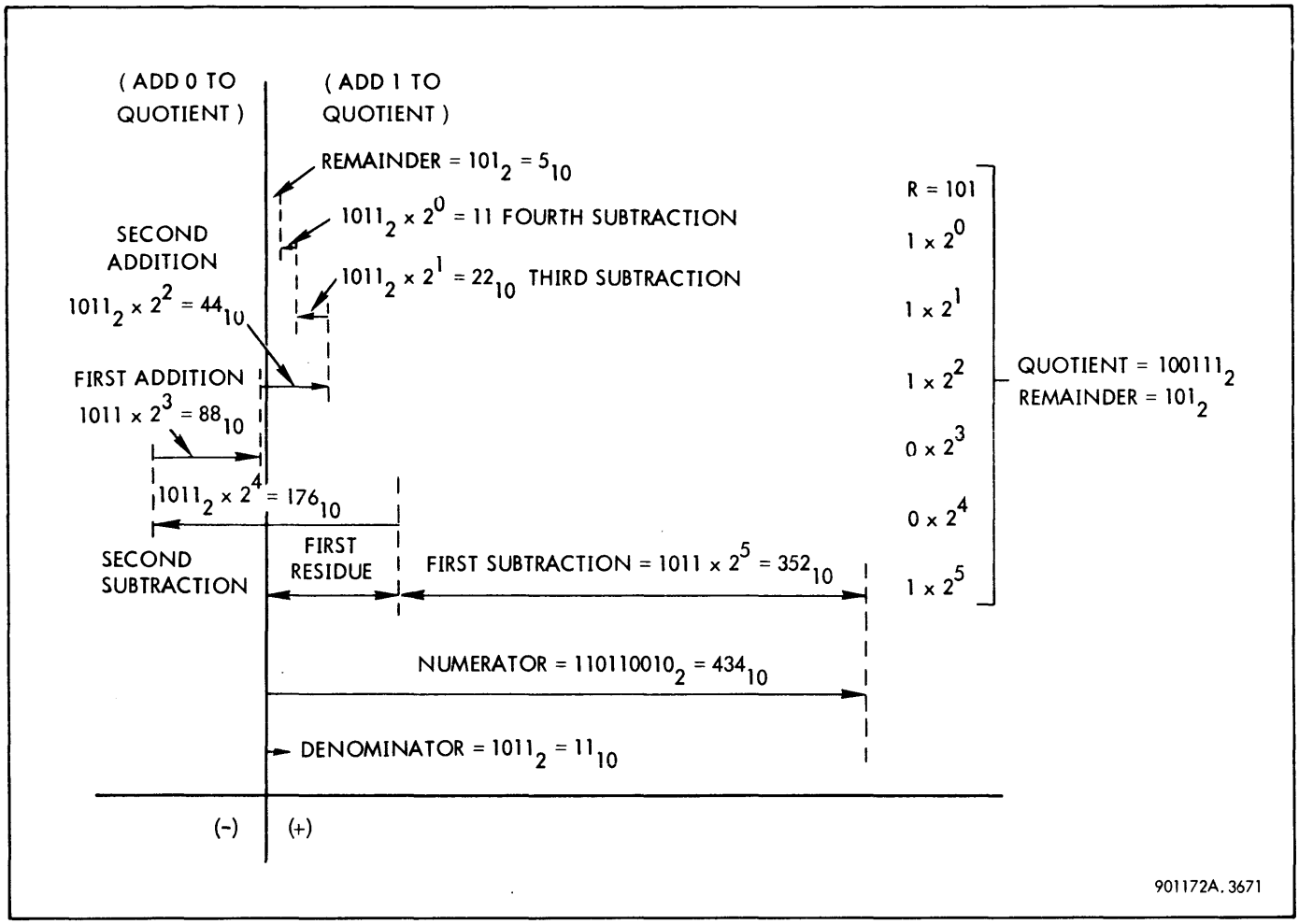


Figure 3-148. Nonrestoring Division, Graphic Representation

When the denominator is to be subtracted from the numerator, the two's complement of the denominator is added to the residue. This technique, with sign bits, is shown in figure 3-149. Each time a one appears in the sign bit of the residue, a zero is added to the appropriate order of the quotient and the next denominator multiple is added (remaining in the uncomplemented form). Each time a zero appears in the sign bit of the residue, a one is added to the appropriate order of the quotient and the next denominator multiple is subtracted (two's complement form is added). Each time a positive residue is reached, the end carry bit is a one. Normally, in two's complement additions, this end carry is discarded. In Sigma 5 division, the end carry, designated K00, is used to signify that a positive partial dividend has been obtained.

As the numerator is transferred to the B-register, in the case of DH, and the A- and B-registers, in the case of DW, the absolute value of the numerator is obtained. This is done by looking at the sign of the numerator, in flip-flop FL1, and taking the two's complement if the numerator is negative.

During each iteration, the residue from the addition is shifted left one bit position in the A- and B-registers so that the next addition will produce the quotient bit for the next lower order. Each quotient bit is transferred to the least significant bit of the B-register. The sign of the denominator is tested during every iteration and compared with the carry bit to determine whether the denominator is to be added or subtracted at the next iteration. At the time the quotient is transferred to private memory, the final sign adjustment is made by taking the two's complement of the quotient if the numerator and denominator signs are unlike.

DIVIDE HALFWORD (DH; 56, D6) AND DIVIDE WORD WITH ODD R FIELD. The DH instruction divides the contents of the private memory register specified by the R field of the instruction (treated as a 32-bit fixed-point integer) by the halfword specified in the reference address field and the contents of the private memory register specified by the X field. The quotient is loaded into the private memory register specified by the R field. If the absolute value of the quotient cannot be correctly represented in 32 bits,

fixed-point overflow occurs, CC2 is set to one, and the contents of register R, CC1, CC3, and CC4 are unchanged. If overflow does not occur, CC4 is set to indicate a negative quotient, and CC3 is set to indicate a positive quotient.

If CC2 is set to one and the fixed-point arithmetic trap mask, flip-flop AM in the program status doubleword is set to one, the program traps to location X'43' with the contents of register R, CC1, CC3, and CC4 unchanged; otherwise the computer executes the next instruction in sequence.

In the case of the divide word instruction with an odd R field, the numerator is in register R as in the divide halfword, and the quotient is loaded into register R. The remainder is lost.

Preparation phases for Divide Halfword are the same as the general PREP phases for halfword instructions, described in paragraph 3-59. Figure 3-150 shows the simplified phase sequence for the DH instruction. Table 3-52 lists the logic sequence during all the execution phases of the instruction.

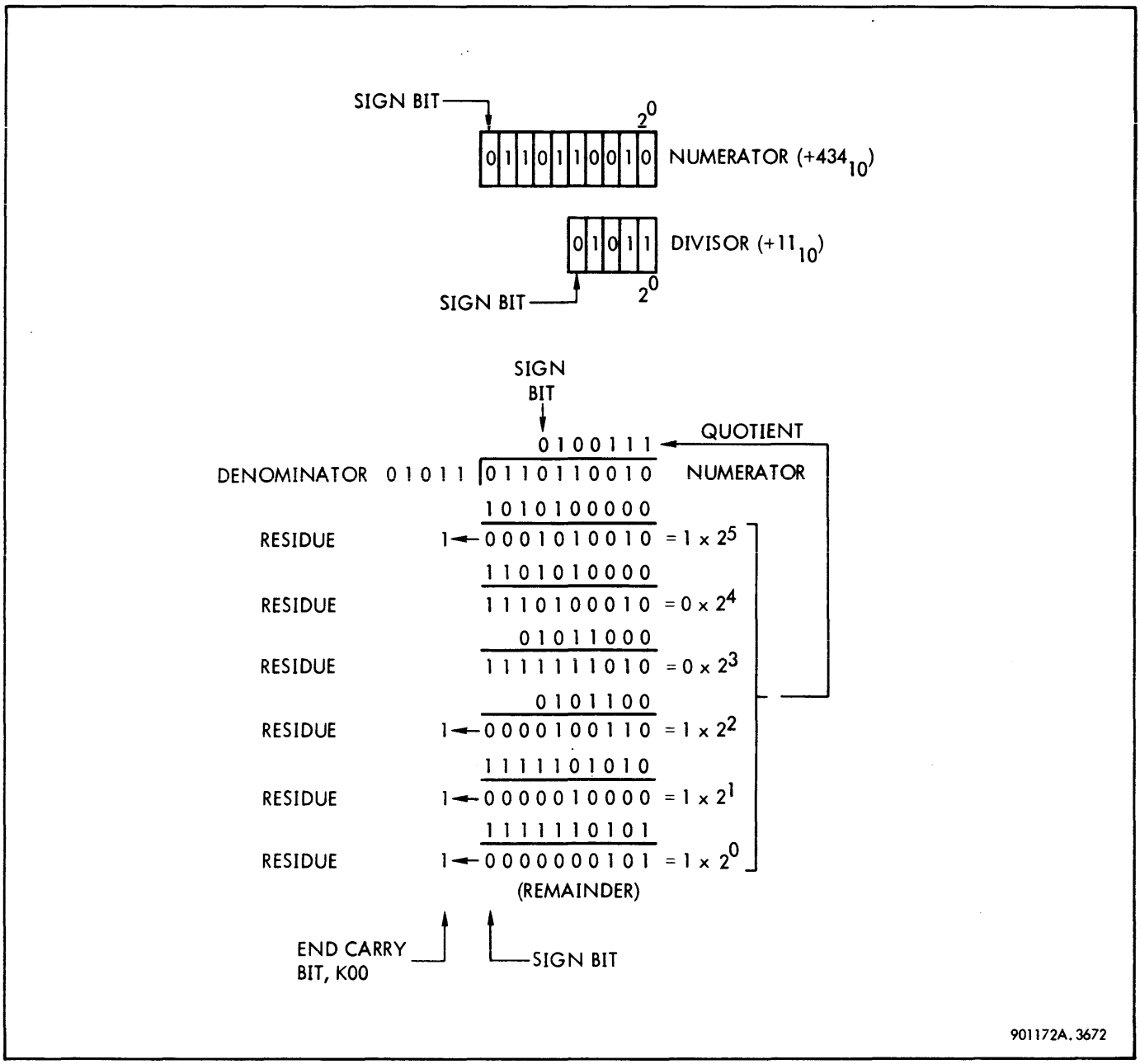


Figure 3-149. Nonrestoring Division With Two's Complement Addition

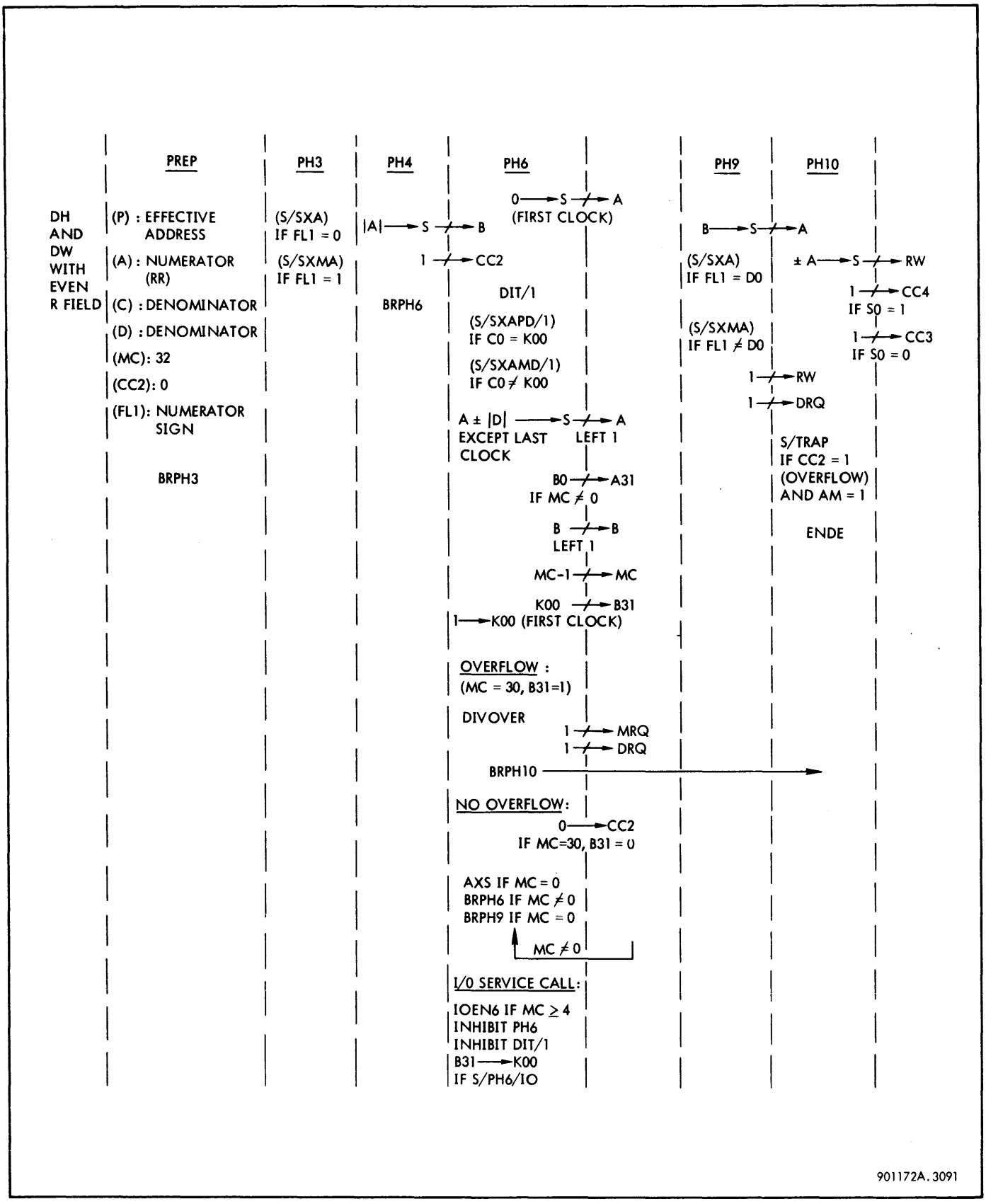


Figure 3-150. Divide Halfword Instruction, Phase Sequence Diagram

Table 3-52. Divide Halfword and Divide Word With Odd R Field Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(P) : Program address		
	(A) : Numerator (RR)		
	(C) : Denominator (sign padded, and down aligned if halfword 0 of DH)		
	(D) : Denominator (sign padded, and down aligned if halfword 0 of DH)		
	(MC) : 32	S/MC2 = FADIV PRE3	
		FADIV = OU5 OL6	
	(CC2) : 0	R/CC2/1) = FAMDS NFUMNH PRE3 + ...	
	(FL1) : Numerator sign	S/FL1 = PRE3 RR0 + ...	
	Branch to PH3	BRPH3 = FAMDS NBRPH5 NANLZ PRE/34 + ...	
		FAMDS = OU5 OL6 + ...	
PH3	One clock long		
T5L	Enable signal (S/SXA) if FL1 = 0	(S/SXA) = NFL1 (S/SX/FL1) + ...	Preset adder for A⟶S in PH4 if positive numerator
		(S/SX/FL1) = FADIV PH3 + ...	
	Enable signal (S/SXMA) if FL1 = 1	(S/SXMA) = FL1 (S/SX/FL1) + ...	Preset adder for two's complement of A⟶S if negative numerator
PH4	One clock long		
T5L	|A| ⟶S	Logic preset in PH3	Absolute value of numerator into B-register via sum bus
	(S0-S31)⟶(B0-B31)	BXS = FADIV PH4 + ...	
	Set flip-flop CC2	S/CC2 = (S/CC2/1) + ...	For overflow test
		(S/CC2/1) = FADIV PH4 + ...	
	Branch to PH6	BRPH6 = FADIV PH4 + ...	
PH6	33 clocks long		
	Iteration phase – repeated until MC = 0		
	Enable signal DIT/1	DIT/1 = FADIV NIOEN (PH6 + S/PH6/IO) N(FADIVH MCZ)	Iteration phase enable signal
			Mnemonic: DH (56, D6)

(Continued)

Table 3-52. Divide Halfword and Divide Word With Odd R Field Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH6 (Cont.)	Preset logic for register control:		
	Enable signal (S/SXAPD/1) if C0 = K00	(S/SXAPD/1) = DIT/1 N(C0 ⊕ K00) + ...	Subtract denominator in D-register from numerator in A-register if the sign of the residue (K00) equals the sign of the denominator (C0). Add denominator to numerator if sign of residue does not equal sign of denominator. K00 = 1 means positive residue
	Enable signal (S/SXAMD/1) if C0 ≠ K00	(S/SXAMD/1)= DIT/1 (C0 ⊕ K00)	
	Direct logic for register control:		
	(S1-S31)—/→(A0-A30) except on final clock	AXSL1 = FADIV PH6 NMCZ	Shift adder output left one bit position with each iteration (equivalent to shifting denominator right)
	B0—/→A31 except on final clock	S/A31 = AXSL1 A31EN/1 + ... A31EN/1 = B0 FAMDS PH6 + ...	Shift numerator and quotient in B-register left into A-register with each iteration
	(B1-B31)—/→(B0-B30)	BXBL1 = FADIV PH6 + ...	Shift numerator one bit position left in B-register. Equivalent to shifting denominator right
	K00—/→B31	S/B31 = BXBL1 B31EN/1 + ... B31EN/1 = K00 FADIV + ...	Shift quotient bits into B-register via B31
	General control functions:		
	MC -1—/→MC	MCDC7 = DIT/1 + ...	Decrement macro-counter 32 times to provide required number of iterations
	Sustain PH6 until MC = 0 or overflow detected	BRPH6 = FAMDS PH6 NFSHEX NMCZ NBRPH10 + ... BRPH10 = DIVOVER + ...	
	On the first clock:		
	1——→K00	K00 = G0003 + ... G0003 = FADIV K00/1 + ... K00/1 = CC2 MC2 + ...	Forces A - |D| ——→S to subtract denominator from numerator on first iteration
	0—/→S—/→A		On first clock nothing is preset in the adder; therefore, the A-register is cleared on AXSL1
			Mnemonic: DH (56, D6)

(Continued)

Table 3-52. Divide Halfword and Divide Word With Odd R Field Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH6 (Cont.)	1 → B31		Because K00 is forced high. The 1 in B31 is insignificant
	On the second clock:		
	A - |D| → S unconditionally		Subtraction of denominator from numerator forced by K00 = 1 on first iteration
	On the third clock:		
	Raise overflow indicator DIVOVER if B31 = 1	DIVOVER = B31 (DIT MC = 30) (DIT MC = 30) = FADIV CC2 MC6 NMC7	B31 contains 2^{31} quotient bit. A 1 indicates overflow
	Set flip-flop PH10 if B31 = 1	S/PH10 = BRPH10 NCLEAR + ... BRPH10 = DIVOVER + ...	Branch to PH10 to set condition code and terminate instruction execution
	Set flip-flop MRQ if B31 = 1	S/MRQ = (S/MRQ/1) + ... (S/MRQ/1) = FAMDS PH6 NIOEN NBRPH6 + ...	Request for core memory cycle for next instruction
	Set flip-flop DRQ if B31 = 1	R/MRQ = ...	
	Set flip-flop DRQ if B31 = 1	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = BRPH10 + ... R/DRQ = ...	Data request, inhibiting transmission of another clock until data release received from core memory
	Reset flip-flop CC2 if B31 = 0	R/CC2 = (R/CC2) (R/CC2/2) = NB31 (DIT MC = 30) + ...	Condition code bit 2 = 0 at end of instruction means no overflow
	On the last clock (MC = 0):		
	Inhibit AXSL1	AXSL1 = FADIV PH6 NMC0 + ...	Residue into A-register
	Enable signal AXS	AXS = FADIV PH6 MCZ + ...	
	Inhibit A ± D → S	(S/SXAPD/1) and (S/SXAMD/1) are qualified by DIT/1, which is qualified by N(FADIVH MCZ)	Discard remainder
	B-register shifts left as before	BXBL1 = FADIV PH6 + ...	Places 2^{0} quotient bit in B0 and 2^{31} quotient bit in B31 (K00)
	Branch to PH9	BRPH9 = FADIVH MCZ + ...	
	I/O service call:		
	Enable signal IOEN6 if MC ≥ 4	IOEN6 = IOEN6/1 PH6 NFPRR NFSHEX + ... IOEN6/1 = N(MC0005Z + ...) + ...	NMC0005Z indicates that MC is greater than 4
			Mnemonic: DH (56, D6)

(Continued)

Table 3-52. Divide Halfword and Divide Word With Odd R Field Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH6 (Cont.)	Inhibit PH6	S/PH6 = BRPH6 NCLEAR NIOEN + ... S/IOEN = IOSC IOEN6 NIOINH R/PH6 = ...	IOEN is set when an I/O service call is received
	Inhibit DIT/1	DIT/1 is qualified by NIOEN on exit and enabled by (S/PH6/IO) on reentry	DIT/1 is used for preset logic, and is one clock ahead of PH6 when interrupt occurs
	B31 ⟶ K00 if (S/PH6/IO)	K00 = G0003 + ... G0003 = FADIV K00/1 + ... K00/1 = B31 (S/PH6/IO) + ...	Quotient bit returned to K00 after I/O interrupt to enable A ± D ⟶ S
PH9	One clock long		
T5L	(B0-B31) ⟶ (S0-S31)	SXB = (FADIV PH9) NDIS + ...	Quotient from B-register into A-register
	(S0-S31) ⟶/ (A0-A31)	AXS = (FADIV PH9) + ...	
	Enable signal (S/SXA) if FL1 = D0	(S/SXA) = FADIV PH9 N(FL1 ⊕ D0) + ...	Preset for A ⟶ S if numerator and denominator have like signs (FL1 contains numerator sign; D0 contains denominator sign)
	Enable signal (S/SXMA) if F1 ≠ D0	(S/SXMA) = FADIV PH9 N(S/SXA) + ...	Preset for two's complement of A ⟶ S if numerator and denominator have unlike signs
	Set flip-flop RW	S/RW = (S/RW) (S/RW) = FAMDS PH9 + ... R/RW = ...	Prepare to write into private memory
	Set flip-flop DRQ	S/DRQ = (S/DRQ/2) + ... (S/DRQ/2) = PH9 + ... R/DRQ = ...	Data request, inhibiting transmission of another clock until data release received from core memory
PH10	Sustained until data release		
DR	No overflow (CC2 = 0):		
	±(A0-A31) ⟶ (S0-S31)	Adder logic preset in PH9	Quotient loaded into private memory register R. (Remainder lost if divide word with even R field)
	(S0-S31) ⟶ (RW0-RW31)	RWXS = RW	
	Set flip-flop CC4 if S0 = 1	S/CC4 = (S/CC4/2) TESTS (S/CC4/2) = NFACOMP S0 + ... TESTS = FADIV ENDE NCC2 + ...	S0 ⟹ negative quotient
			Mnemonic: DH (56, D6)

(Continued)

Table 3-52. Divide Halfword and Divide Word With Odd R Field Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH10 DR (Cont.)	Set flip-flop CC3 if S0 = 0	S/CC3 = SGTZ TESTS SGTZ = N(S0 NFACOMP) (NS0007Z + NS0815Z + NS1631Z + NS3263Z)	NS0 and nonzero quotient ⇒ positive quotient
	Overflow (CC2 = 1):		
	Trap to X'43' if AM = 1	(S/TRAP) = ENDE AM CC2 OVERIND OVERIND = FADIV + . . .	AM is fixed-point arithmetic trap mask bit in program status doubleword
	ENDE functions		
			Mnemonic: DH (56, D6)

DIVIDE WORD (DW; 36, B6) WITH EVEN R FIELD. The DW instruction divides the contents of the private memory registers specified by the R field and Ru1 (treated as a 64-bit fixed-point integer) by the contents of the core memory location specified in the reference address field. The remainder is loaded into register R and the quotient into register Ru1. If a nonzero remainder occurs, the remainder has the same sign as the dividend. Fixed-point overflow occurs if the absolute value of the quotient cannot be correctly represented in 32 bits. In this case, flip-flop CC2 is set to one, and the contents of registers R and Ru1, flip-flops CC1, CC3, and CC4 remain unchanged; otherwise flip-flop CC2 is set to zero, flip-flop CC3 is set to reflect a positive quotient, and flip-flop CC4 is set to reflect a negative quotient. Flip-flop CC1 is unchanged.

If flip-flop CC2 is set to one and the fixed-point arithmetic trap mask, flip-flop AM in the program status doubleword, contains a one, the computer traps to location X'43' with the original contents of registers R and Ru1, and flip-flops CC1, CC3, and CC4 unchanged; otherwise the computer executes the next instruction in sequence.

Preparation phases for Divide Word are the same as the general PREP phases for word instructions, described in paragraph 3-59. Figure 3-151 shows the simplified phase sequence for the DW instruction. Table 3-53 lists the logic sequence during all the execution phases of the instruction.

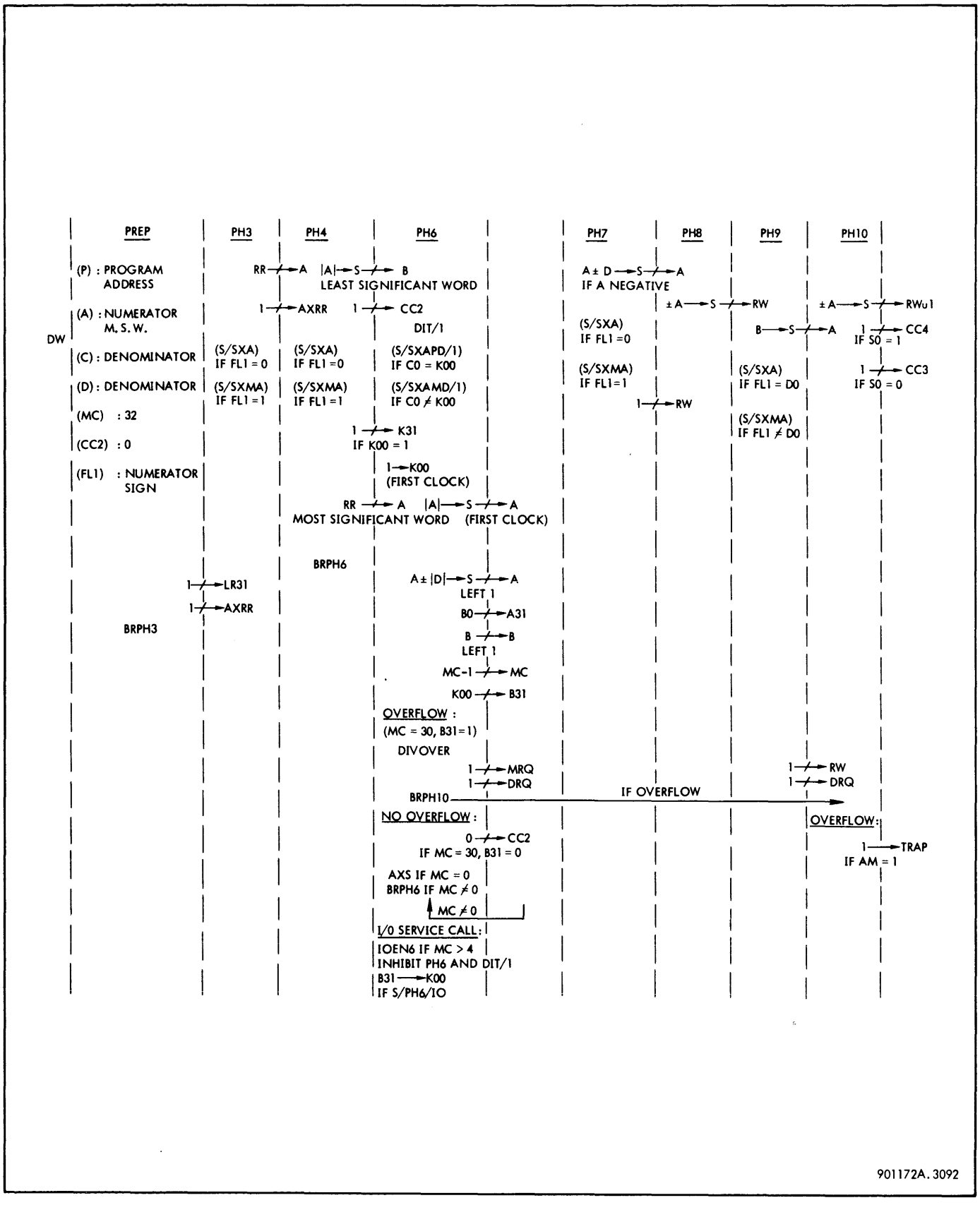


Figure 3-151. Divide Word Instruction, Phase Sequence Diagram

Table 3-53. Divide Word Sequence (Even R Field)

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(P) : Program address		
	(A) : Most significant word of numerator		
	(C) : Denominator		
	(D) : Denominator		
	(MC) : 32	S/MC2 = FADIV PRE3	
		FADIV = OU3 OL6 R31 + FUDW NR31	
	(CC2) : 0	(R/CC2/1) = FAMDS NFUMH PRE3 + ...	
		FAMDS = (FUDW NR31) + ...	
	(FL1) : Numerator sign	S/FL1 = PRE3 RR0 + ...	
	Reset flip-flop NLR31F	S/NLR31F = N(S/LR31)	Place address of odd-numbered private memory register on address lines by setting least significant bit of address
		(S/LR31) = (FUDW NR31) PRE3	
		(FUDW NR31) = OU3 OL6 NR31	
		R/NLR31F = ...	
	Reset flip-flop NAXRR if R is even	S/NAXRR = N(S/AXRR)	Preset for transfer of contents of private memory register Ru1 to A-register in PH3
		(S/AXRR) = PRE3 (FUDW NR31 + ...)+...	
		R/NAXRR = ...	
	Branch to PH3	BRPH3 = FAMDS NBRPH5 NANLZ PRE/34 + ...	
PH3	One clock long		
T5L	Enable signal (S/SXA) is FL1 = 0	(S/SXA) = NFL1 (S/SX/FL1) + ...	Preset adder for A⟶S in PH4 if positive numerator
		(S/SX/FL1) = FADIV PH3 + ...	
	Enable signal (S/SXMA) if FL1 = 1	(S/SXMA) = FL1 (S/SX/FL1) + ...	Preset adder for two's complement of A⟶S if negative numerator
	(RR1-RR31)⟶(A0-A31)	Logic preset in PRE3	Transfer least significant word of numerator to A-register
	Reset flip-flop NAXRR	See PREP phase	Preset for transfer of contents of private memory register R to A-register in PH4
			Mnemonic: DW (36, B6)

(Continued)

Table 3-53. Divide Word Sequence (Even R Field) (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH4	One clock long		
T5L	|A| ⟶ S	Logic set in PH3	Absolute value of least significant word of numerator into B-register via sum bus
	(S0-S31) ⟶ (B0-B31)	BXS = FADIV PH4 + ...	
	Reset flip-flop NK31 if K00 = 1	S/NK31 = N(S/K31/1) + ...	Since (S/K31/2) is low, resetting of carry flip-flop NK31 is determined by state of K00. Flip-flop K31 propagates carry from least significant bit to most significant bit of numerator when forming two's complement
		(S/K31/1) = [K00 (S/K31/3) + (S/K31/2) (S/K31/3)]	
		(S/K31/2) = N(FAMDS PH4) + ...	
	Set flip-flop CC2	S/CC2 = (S/CC2/1) + ...	For overflow test
		(S/CC2/1) = FADIV PH4 + ...	
	Enable signal (S/SXA) if FL1 = 0	(S/SXA) = NFL1 (S/SX/FL1) + ...	Preset adder for A ⟶ S in PH6 if positive numberator
		(S/SX/FL1) = (FUDW NR31 + ...) (PH4 + ...) + ...	
	Enable signal (S/SXMA) if FL1 = 1	(S/SXMA) = FL1 S/SX/FL1	Preset adder for two's complement of A plus carry ⟶ S if negative numerator
	(RR0-RR31) ⟶ (A0-A31)	A_n = RR_n AXRR (preset in PH3)	Most significant word of numerator into A-register
	Branch to PH6	BRPH6 = FADIV PH4 + ...	
PH6	33 clocks long		
	Iteration phase – repeated until MC = 0		
	Enable signal DIT/1	DIT/1 = FADIV NIOEN (PH6 + S/PH6/IO) N(FADIVH MCZ)	Iteration phase enable signal
	Preset logic for register control:		
	Enable signal (S/SXAPD/1) if C0 = K00	(S/SXAPD/1) = DIT/1 N(C0 ⊕ K00) + ...	Subtract denominator in D-register from numerator in A-register if the sign of the residue (K00) equals the sign of the denominator (C0). Add denominator to numerator if sign of residue does not equal sign of denominator. K00 = 1 means positive residue
	Enable signal (S/SXAMD/1) if C0 ≠ K00	(S/SXAMD/1)= DIT/1 (C0 ⊕ K00)	
			Mnemonic: DW (36, B6)

(Continued)

Table 3-53. Divide Word Sequence (Even R Field) (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH6 (Cont.)	Direct logic for register control:		
	(S1-S31)—/→(A0-A30) except on final clock	AXSL1 = FADIV PH6 NMCZ	Shift adder output left one bit position with each iteration (equivalent to shifting denominator right)
	B0—/→A31 except on final clock	S/A31 = AXSL1 A31EN/1 + ... A31EN/1 = B0 FAMDS PH6 + ...	Shift numerator and quotient in B-register left into A-register with each iteration
	(B1-B31)—/→(B0-B30)	BXBL1 = FADIV PH6 + ...	Shift numerator one bit position left in B-register. Equivalent to shifting denominator right
	K00—/→B31	S/B31 = BXBL1 B31EN/1 + ... B31EN/1 = K00 FADIV + ...	Shift quotient bits into B-register via B31
	General control functions:		
	MC -1—/→MC	MCDC7 = DIT/1 + ...	Decrement macro-counter 32 times to provide required number of iterations
	Sustain PH6 until MC = 0 or overflow detected	BRPH6 = FAMDS PH6 NFSHEX NMCZ BRPH10 + ... BRPH10 = DIVOVER + ...	
	On the first clock:		
	1——→K00	K00 = G0003 + ... G0003 = FADIV K00/1 + ... K00/1 = CC2 MC2 + ...	Forces A - |D|——→S to subtract denominator from numerator on first iteration
	(NA0-NA31) plus carry——→(S0-S31) if FL1 = 1 (A0-A31)——→(S0-S31) if FL1 = 0	Adder preset in PH4. Carry set in K31 in PH4	Absolute value of most significant word of numerator into A-register
	1—/→B31		Because K00 is forced high. The 1 in B31 is insignificant
	On the second clock:		
	A - |D|——→S unconditionally		Subtraction of denominator from numerator forced by K00 = 1 on first iteration
			Mnemonic: DW (36, B6)

(Continued)

Table 3-53. Divide Word Sequence (Even R Field) (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH6 (Cont.)	On the third clock:		
	Raise overflow indicator DIVOVER if B31 = 1	DIVOVER = B31 (DIT MC = 30) (DIT MC = 30) = FADIV CC2 MC6 NMC7	B31 contains 2^{31} quotient bit. A 1 indicates overflow
	Set flip-flop PH10 if B31 = 1	S/PH10 = BRPH10 NCLEAR + ... BRPH10 = DIVOVER + ...	Branch to PH10 to set condition code and terminate instruction execution
	Set flip-flop MRQ if B31 = 1	S/MRQ = (S/MRQ/1) + ... (S/MRQ/1) = FAMDS PH6 NIOEN NBRPH6 + ... R/MRQ = ...	Request for core memory cycle for next instruction
	Set flip-flop DRQ if B31 = 1	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = BRPH10 + ... R/DRQ = ...	Data request, inhibiting transmission of another clock until data release received from core memory
	Reset flip-flop CC2 if B31 = 0	R/CC2 = (R/CC2/2) + ... (R/CC2/2) = NB31 (DIT MC = 30) + ...	Condition code bit 2 = 0 at end of instruction means no overflow
	On the last clock (MC = 0):		
	Inhibit AXSL1	AXSL1 = FADIV PH6 NMC0 + ...	Residue into A-register
	Enable signal AXS	AXS = FADIV PH6 MCZ + ...	
	Enable signal A ± D ⟶ S	(S/SXAPD/1) and (S/SXAMD/1) are qualified by DIT/1, which is qualified by N(FADIVH MCZ)	Save remainder in A-register
	B-register shifts left as before	BXBL1 = FADIV PH6 + ...	Places 2^0 quotient bit in B0 and 2^{31} quotient bit in B31 (K00)
	I/O service call:		
	Enable signal IOEN6 if MC ≥ 4	IOEN6 = IOEN6/1 PH6 NFPRR NFSHEX + ... IOEN6/1 = N(MC0005Z + ...) + ...	NMC0005Z indicates that MC is greater than 4
	Inhibit PH6	S/PH6 = BRPH6 NCLEAR NIOEN + ... S/IOEN = IOSC IOEN6 NIOINH	IOEN is set when an I/O service call is received
	Inhibit DIT/1	DIT/1 is qualified by NIOEN on exit and enabled by (S/PH6/IO) on reentry	DIT/1 is used for preset logic, and is one clock ahead of PH6 when interrupt occurs
			Mnemonic: DW (36, B6)

(Continued)

Table 3-53. Divide Word Sequence (Even R Field) (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH6 (Cont.)	B31 ⟶ K00 if (S/PH6/IO)	K00 = G0003 + ... G0003 = FADIV K00/1 + ... K00/1 = B31 (S/PH6/IO) + ...	Quotient bit returned to K00 after I/O interrupt to enable A ± D ⟶ S
PH7	One clock long		
T5L	(A0-A31) + (D0-D31) ⟶ (S0-S31) or (A0-A31) - (D0-D31) ⟶ (S0-S31)	Adder preset at last PH6 clock	Restore residue to positive state if negative to provide positive remainder
	(S0-S31) ⟶/⟶ (A0-A31) if A is negative	(AXS) = (FUDW NR31) PH7 NB31	
	Enable signal (S/SXA) if FL1 = 0	(S/SXA) = NFL1 (S/SX/FL1) + ... (S/SX/FL1) = FADIV PH7 + ...	Preset adder for A ⟶ S if positive numerator
	Enable signal (S/SXMA) if FL1 = 1	(S/SXMA) = FL1 (S/SX/FL1) + ...	Preset adder for two's complement of A ⟶ S if negative numerator
	Set flip-flop RW	S/RW = (S/RW) (S/RW) = (FUDW NR31) PH7 + ...	Prepare to write into private memory
PH8	One clock long		
T5L	(A0-A31) ⟶ (S0-S31) or N(A0-A31) ⟶ (S0-S31)	Adder preset at PH7 clock	
	(S0-S31) ⟶ (RW0-RW31)	RWXS = RW	Transfer remainder into private memory register R. Take two's complement if numerator is negative
PH9	One clock long		
T5L	(B0-B31) ⟶ (S0-S31)	SXB = (FADIV PH9) NDIS + ...	Quotient from B-register into A-register
	(S0-S31) ⟶/⟶ (A0-A31)	AXS = (FADIV PH9) + ...	
	Enable signal (S/SXA) if FL1 = D0	(S/SXA) = FADIV PH9 N(FL1 ⊕ D0) + ...	Preset for A ⟶/⟶ S if numerator and denominator have like signs. (FL1 contains numerator sign; D0 contains denominator sign)
	Enable signal (S/SXMA) if FL1 ≠ D0	(S/SXMA) = FADIV PH9 N(S/SXA) + ...	Preset for two's complement of A ⟶ S if numerator and denominator have unlike signs
			Mnemonic: DW (36, B6)

(Continued)

Table 3-53. Divide Word Sequence (Even R Field) (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH9 T5L (Cont.)	Reset flip-flop NLR31F	S/NLR31F = N(S/LR31) (S/LR31) = (FUDW NR31) PH9 + ... R/NLR31F = ...	Place address of odd-numbered private memory register on address lines setting least significant bit of address
	Set flip-flop RW	(S/RW) = FAMDS PH9 + ...	Prepare to write into private memory
	Set flip-flop DRQ	(S/DRQ/2) = PH9 + ...	Data request, inhibiting transmission of another clock until data release received from core memory
PH10 DR	Sustained until data release No overflow (CC2 = 0):		
	± (A0-A31) ⟶ (S0-S31)	Adder logic preset in PH9	Quotient loaded into private memory register Ru1
	(S0-S31) ⟶ (RW0-RW31)	RWXS = RW	
	Set flip-flop CC4 if S0 = 1	S/CC4 = (S/CC4/2) TESTS (S/CC4/2) = NFACOMP S0 + ... TESTS = FADIV ENDE NCC2 + ...	S0 ⟹ negative quotient
	Set flip-flop CC3 if S0 = 0	S/CC3 = SGTZ TESTS SGTZ = N(S0 NFACOMP) (NS0007Z + NS0815Z + NS1631Z + NS3263Z)	NS0 and nonzero quotient ⟹ positive quotient
	Overflow (CC2 = 1):		
	Trap to X'43' if AM = 1	(S/TRAP) = ENDE AM CC2 OVERIND OVERIND = FADIV + ...	AM is fixed-point arithmetic trap mask bit in program status doubleword
	ENDE functions		
			Mnemonic: DW (36, D6)

3-69 Family of Modify and Test Instructions

MODIFY AND TEST BYTE (MTB; 73, F3). The MTB instruction performs one of two operations, depending upon the value in the R field of the instruction word (bits 8 through 11). If the value is zero, the effective byte is tested to determine if it is zero or nonzero, and the condition code flip-flops are set accordingly. If the value is not zero, the four bits are treated as a signal quantity of (-8 to +7), the sign (bit 8) is extended to form a byte, and this byte is effectively added to the effective byte. The resulting value is loaded into the effective byte location, and the condition codes are set according to the result. The effective byte is thereby modified by a value of -8 to +7 and tested. If the MTB instruction is executed in an interrupt location, the condition code is not affected.

Condition Codes. Condition codes for the MTB instruction are:

CC1	CC2	CC3	CC4	Result in EW Location
-	0	0	0	Zero
-	0	1	0	Nonzero
0	-	-	-	No carry from byte
1	-	-	-	Carry from byte

Examples. Examples of the MTB instruction are:

Instruction

0011 0011 1011 XXXX XXXX XXXX XXXX XXXX

Effective byte 0 0 1 1 1 0 0 1

(EB + R) 0 0 1 1 0 1 0 0

Condition code: 0010

Instruction

0011 0011 0110 XXXX XXXX XXXX XXXX XXXX

Effective byte 0 0 0 0 0 1 1 0

(EB + R) 0 0 0 0 1 1 0 0

Condition code: 0010

Modify and Test Byte Phase Sequence. Preparation phases for the Modify and Test Byte instruction are the same as the general PREP phases for byte instructions, paragraph 3-59. Figure 3-152 shows the simplified phase sequence for the MTB instruction. Table 3-54 lists the detailed logic sequence during all MTB execution phases.

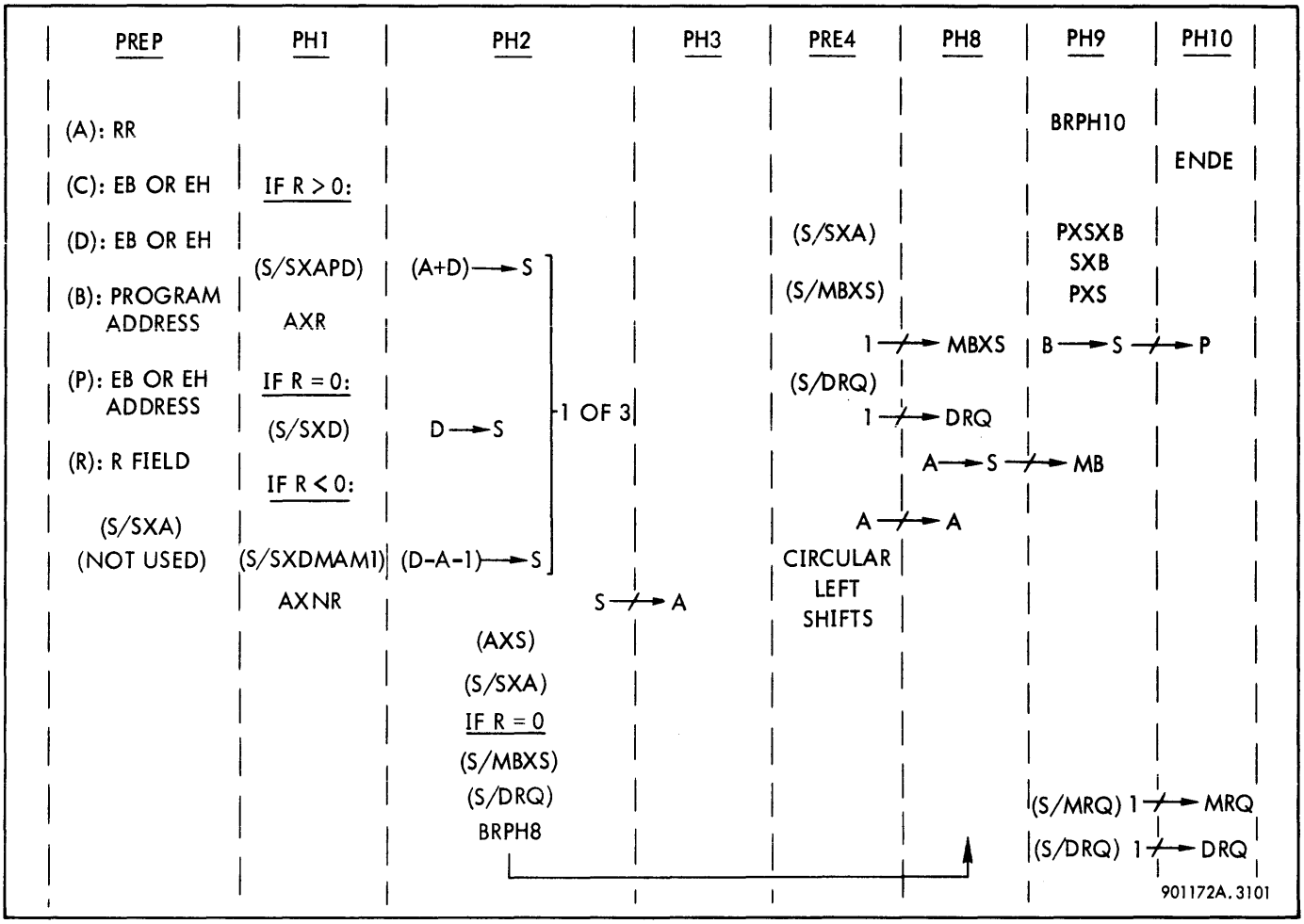


Figure 3-152. Modify and Test Byte and Modify and Test Halfword Instructions, Phase Sequence Diagram

Table 3-54. Modify and Test Byte Sequence

Phase	Function Performed	Signals Involved			Comments
PREP	At end of PREP:				
	(C): EB				Effective byte
	(D): EB				Effective byte
	(B): Program address				Temporary storage for address of next instruction
	(P): EW address				Address of effective word
	(R): R field of instruction word				Bits 28-31 of instruction word
	Enable signal (S/SXA)	(S/SXA)	=	FAMT (PRE/34 + PH2)	Not used
		FAMT	=	NOU1 OL3 O3	
PH1	One clock long				
T5L	If R is equal to zero:				
	Enable signal (S/SXD)	(S/SXD)	=	FAMT PH1 RZ + ...	Preset adder for D ⟶ S transfer in PH2
		RZ	=	NR28 NR29 NR30 NR31	R field is zero
	If R is less than zero:				
	Set flip-flop SW2	S/SW2	=	FAMT PH1 NRZ + ...	Control alignment in PRE4
		R/SW2	=	RESET/A + ...	
	(R28-R31) ⟶ (A28-A31) zeros ⟶ (A0-A27)	AXNR	=	FAMT PH1 R28	N(R field) ⟶/ A-register
	Enable signal (S/SXDMAM1)	(S/SXDMAM1)	=	FAMT PH1 R28	Preset adder for (D - A - 1) ⟶ S transfer in PH2
	If R is greater than zero:				
	Set flip-flop SW2	S/SW2	=	FAMT PH1 NRZ + ...	Control alignment in PRE4
		R/SW2	=	RESET/A + ...	
	(R28-R31) ⟶/ (A28-A31) zeros ⟶/ (A0-A27)	AXR	=	FAMT PH1 NR28 + ...	R field ⟶/ A-register
	Enable signal (S/SXAPD)	(S/SXAPD)	=	FAMT PH1 NR28 NRZ	Preset adder for (A+D) ⟶ S transfer in PH2
	If INTRAP, set flip-flop CEINT	S/CEINT	=	FAMT PH1 INTRAP	Enable interrupt clock
		R/CEINT	=	...	
					Mnemonic: MTB (73, F3)

(Continued)

Table 3-54. Modify and Test Byte Sequence (Cont.)

Phase	Function Performed	Signals Involved			Comments
PH2 T5L or T8L	One clock long				
	(D0-D31) ⟶ (S0-S31)	Adder logic set at PH1 clock			If R equals zero
	(D0-D31) - (A0-A31) -1 ⟶ (S0-S31)	Adder logic set at PH1 clock			If R less than zero
	(A0-A31) + (D0-D31) ⟶ (S0-S31)	Adder logic set at PH1 clock			If R greater than zero
	(S0-S31) ⟶/ (A0-A31)	AXS	=	FAMT PH2	Transfer result of operation to A-register
	Enable clock T8 if arithmetic operation required	T8EN	=	NT5EN NT11L N(SXADD/1 RW) N(RW REU) N(REU AXRR)	T5EN is disabled by SXADD/1 if addition or subtraction required
	If NINTRAP				
	Set CC3 if result is nonzero	S/CC3	=	SGTZ TESTS + ...	S0 will always be a zero for MTB
		SGTZ	=	(S0+S1 + ... + S31) NS0 NFACOMP	
		TESTS	=	FAMT PH2 NINTRAP + ...	
		R/CC3	=	TESTS + ...	
	Reset CC4	S/CC4	=	(S/CC4/2) TESTS + ...	
		(S/CC4/2)	=	NFACOMP S0 + ...	
		R/CC4	=	TESTS + ...	
	Set CC1 if end carry from byte	S/CC1	=	K23 CC1XK23 + ...	
		CC1XK23	=	FAMT PH2 NINTRAP OU7	
		R/CC1	=	(R/CC1/1)	
		(R/CC1/1)	=	CC1XK23 + ...	
	Reset CC2	S/CC2	=	(S00 ⊕ S0) PROBOVER + ...	No overflow is possible for MTB
		PROBOVER	=	FAMT PH2 NINTRAP + ...	
		R/CC2	=	CC1XK00 + ...	
		CC1XK00	=	FAMT PH2 NINTRAP	
		S/FL3	=	CC1XK23 + ...	Set flag for PH3
		R/FL3	=	...	
		NS23	=	CC1XK23 + ...	Inhibit set of S23
	If INTRAP				
	Activate highest priority	LEVACT	=	FAMT PH2 INTRAP	
	Arm highest priority	LEVARM	=	FAMT PH2 INTRAP	
					Mnemonic: MTB (73, F3)

(Continued)

Table 3-54. Modify and Test Byte Sequence (Cont.)

Phase	Function Performed	Signals Involved			Comments
PH2 T5L or T8L (Cont.)	Trigger count zero	CNTZERO	=	FAMT PH2 INTRAP S0031Z	S-register contains zero
	INTRAP or NINTRAP				
	Enable signal (S/SXA)	(S/SXA)	=	FAMT (PRE/34 + PH2)	Preset adder for A ⟶ S transfer in PH3 or PH8
	If R equals zero:				
	Branch to PH8	BRPH8	=	FAMT PH2 (RZ + NO1)	If R equals zero, no shifting required
		S/PH8	=	BRPH8 NCLEAR	
		R/PH8	=	...	
	If R not zero:				
	Load byte counter	S/BC0	=	FAMT PH2 NRZ NP32 O1 + ...	Stores number of left shifts required in PRE4
		R/BC0	=	BCX + ...	
		S/BC1	=	FAMT PH2 NRZ NP33 OU7 + ...	
		R/BC1	=	BCX + ...	
		BCX	=	FAMT PH2 NRZ O1	
PH3 T5L	Adjust sign	FUMTSIGN	=	FAMT PH3 NINTRAP (CC2 + NOU5)	Always enabled by NOU5
	Test for byte equal to zero	S/CC3	=	FUMTSIGN FL3 NS1631Z + ...	Set CC3 if S-register does not contain all zeros
		R/CC3	=	FUMTSIGN + ...	
	Branch to PRE4	BRPRE4	=	FAMT PH3	
PRE4 T5L	Sustained until byte counter zero (BCZ)				
	Circular left shift of A-register one byte for each clock	AXAL8	=	FAMT PRE4 SW2 NBCZ	Repeated while BCZ false
		BCZ	=	NBC0 NBC1	
	Branch to PRE4	BRPRE4	=	PRE4 NBCZ	Remain in PRE4 while BCZ false
	Enable signal PRE/34	PRE/34	=	PRE4 NBC1 NBC0 NANLZ	Terminate PRE4 after BCZ true
	Enable signal (S/SXA)	(S/SXA)	=	FAMT (PRE/34 + PH2)	Preset adder for A ⟶ S transfer in PH8
	Enable signal (S/MBXS)	(S/MBXS)	=	FAMT PRE/34 SW2	Preset for S ⟶/⟶ MB transfer in PH8
					Mnemonic: MTB (73, F3)

(Continued)

Table 3-54. Modify and Test Byte Sequence (Cont.)

Phase	Function Performed	Signals Involved			Comments
PRE4 T5L (Cont.)	Branch to PH8 Set flip-flop DRQ	BRPH8 S/DRQ R/DRQ	= = =	FAMT PRE/34 SW2 (S/MBXS) + ...	Inhibits transmission of another clock until data release from core memory
PH8 DR	Sustained until DR (A0-A31) ⟶ (S0-S31) (S0-S31) ⟶/⟶ (MB0-MB31)	Adder logic set at PRE4 clock MBXS	= =	Set at PRE4 clock	Modified byte transferred to effective byte location
PH9 T5L	One clock long (B0-B31) ⟶ (S0-S31) (S15-S31) ⟶/⟶ (P15-P31)	SXB PXS PXSXB S/MRQ (S/MRQ/2) R/MRQ S/DRQ R/DRQ R/INTRAP	= = = = = = = = =	$\overline{PXSXB}$ + ... PXSXB + ... NFAFL NFAMDS PH9 (S/MRQ/2) + ... PXSXB NINTRAP + ... (S/MRQ/2) NCLEAR ... FAMT PH9 + ...	Program address Program address bits only Core memory request for effective word Inhibits transmission of another clock until data release from core memory Reset if INTRAP
PH10 DR	ENDE functions				Mnemonic: MTB (73, F3)

MODIFY AND TEST HALFWORD (MTH, 53, D3). The MTH instruction performs one of two operations, depending upon the value in the R field of the instruction word (bits 8 through 11). If the value is zero, the effective halfword is tested to determine if it is zero, negative, or positive, and the condition code flip-flops are set accordingly. If the value is not zero, the four bits are treated as a signal quantity of -8 to +7, the sign bit (bit 8) is extended to form a halfword, and this halfword is effectively added to the effective halfword. The resulting value is loaded into the effective byte location, and the condition codes are set according to the result. The effective halfword is thereby modified by a value of -8 to +7 and tested.

If fixed-point overflow occurs, flip-flop CC2 is set to 1, and the computer traps to location X'43' if the fixed-point arithmetic mask (AM) is 1. The trap occurs after the result is stored in the effective halfword location. If the MTH instruction is executed in an interrupt location, the condition code is not affected, and no fixed-point overflow trap can occur.

Condition Codes. Condition codes for the MTH instruction are:

CC1	CC2	CC3	CC4	Result in EW Location
-	-	1	0	Positive
-	-	0	0	Zero
-	-	0	1	Negative
-	0	-	-	No fixed-point overflow
-	1	-	-	Fixed-point overflow
0	-	-	-	No carry from word
1	-	-	-	Carry from word

Examples. Examples of the MTH instruction are:

```
Instruction
    0011 0011 1011 XXXX XXXX XXXX XXXX XXXX
Effective halfword      0000 0001 0011 1001
(EHW + R)               0000 0001 0011 0100
```

Condition code: 0010

```
Instruction
    0011 0011 0110 XXXX XXXX XXXX XXXX XXXX
Effective halfword      1111 1111 1111 1010
(EHW + R)               0000 0000 0000 0000
```

Condition code: 0000

Modify and Test Halfword Phase Sequence. Preparation phases for the MTH instruction are the same as the general PREP phases for halfword instructions, paragraph 3-59. Figure 3-152 shows the simplified phase sequence for the MTH halfword instruction. Table 3-55 lists the detailed logic sequence during all MTH execution phases.

Table 3-55. Modify and Test Halfword Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(A): RR		Private memory register R (not used)
	(D): EW		Effective word
	(B): Program address		Temporary storage for address of next instruction
	(P): EW address		Address of effective word
	(R): R field of instruction		Bits 28-31
	Enable signal (S/SXA)	(S/SXA) = FAMT (PRE/34 + PH2) FAMT = NO41 OL3 O3	Not used
PH1 T5L	One clock long		
	If R is equal to 0:		
	Enable signal (S/SXD)	(S/SXD) = FAMT PH1 RZ	Preset adder for D ⟶ S transfer in PH2
		RZ = NR28 NR29 NR30 NR31	R field is zero
	If R is less than 0:		
	Set flip-flop SW2	S/SW2 = FAMT PH1 NRZ + ...	Control alignment in PRE4
		R/SW2 = RESET/A + ...	
			Mnemonic: MTH (53, D3)

(Continued)

Table 3-55. Modify and Test Halfword Sequence (Cont.)

Phase	Function Performed	Signals Involved			Comments
PH1 T5L (Cont.)	Enable signal AXNR	AXNR	=	FAMT PH1 R28	N(R field) ⟶ A-register
	Enable signal (S/SXDMAM1)	(S/SXDMAM1)	=	FAMT PH1 R28	Preset adder for (D - A - 1) ⟶ S transfer in PH2
	If R is greater than 0:				
	Set flip-flop SW2	S/SW2	=	FAMT PH1 NRZ + ...	Control alignment in PRE4
		R/SW2	=	RESET/A + ...	
	Enable signal AXR	AXR	=	FAMT PH1 NR28 + ...	R field ⟶ A-register
	Enable signal (S/SXAPD)	(S/SXAPD)	=	FAMT PH1 NR28 NRZ + ...	Preset adder for (A + D) ⟶ S transfer in PH2
	If INTRAP, set flip-flop CEINT	S/CEINT	=	FAMT PH1 INTRAP + ...	Enable interrupt clock
		R/CEINT	=	...	
	(If NINTRAP, T5L or T8L)				
PH2 T5L or T8L	One clock long				
	(D0 - D31) ⟶ (S0 - S31)	Adder logic set at PH1 clock			If R equals zero
	(D0 - D31) - (A0 - A31) - 1 ⟶ (S0 - S31)	Adder logic set at PH1 clock			If R less than zero
	(D0 - D31) + (A0 - A31) ⟶ (S0 - S31)	Adder logic set at PH1 clock			If R greater than zero
	(S0 - S31) ⟶ (A0 - A31)	AXS	=	FAMT PH2	Transfer result of operation to A-register
	Enable clock T8 if arithmetic operation required	T8EN	=	NT5EN NT11L N(SXADD/1 RW) N(RW REU) N(REU AXRR)	T5EN is disabled by SXADD/1 if addition or subtraction is performed
		SXADD/1	=	true when addition or subtraction is performed	
	If NINTRAP				
	Set condition code flip-flops	S/CC3	=	SGTZ TESTS + ...	
		SGTZ	=	(S0 + S1 + ... + S31) S0 NFACOMP	
		TESTS	=	FAMT PH2 NINTRAP + ...	
		R/CC3	=	TESTS + ...	State of CC3 and CC4 indicates polarity of data in A-register after operation
		S/CC4	=	(S/CC4/2) TESTS + ...	
		(S/CC4/2)	=	NFACOMP S0 + ...	
		R/CC4	=	TESTS + ...	
					Mnemonic: MTH (53, D3)

(Continued)

Table 3-55. Modify and Test Halfword Sequence (Cont.)

Phase	Function Performed	Signals Involved			Comments
PH2 T5L or T8L (Cont.)		S/CC1	=	K00 CC1XK00 + ...	Set CC1 if end carry
		CC1XK00	=	FAMT PH2 NINTRAP	
		R/CC1	=	CC1XK00 + ...	
		S/CC2	=	(S00 ⊕ S0) PROBOVER + (S15 ⊕ S16) PROBOVER/H + ...	Set CC2 if overflow
		PROBOVER	=	FAMT PH2 NINTRAP + ...	
		PROBOVER/H	=	FAMT PH2 NINTRAP OU5 + ...	
		R/CC2	=	CC1XK00	
	If INTRAP				
	Activate highest priority	LEVACT	=	FAMT PH2 INTRAP	
	Arm highest priority	LEVARM	=	FAMT PH2 INTRAP	
	Trigger count zero	CNTZERO	=	FAMT PH2 INTRAP S0031Z	S-register contains zero
	INTRAP or NINTRAP				
	Enable signal (S/SXA)	(S/SXA)	=	FAMT (PRE/34 + PH2)	Preset adder for A ⟶ S transfer in PH3 or PH8
	If R equals zero:				
	Branch to PH8	BRPH8	=	FAMT PH2 (RZ + NO1)	If R equals zero, no shifting required
	If R not zero:				
	Load byte counter	S/BC0	=	FAMT PH2 NRZ NP32 O1 + ...	Stores number of left shifts required in PRE4
		R/BC0	=	BCX + ...	
		BCX	=	FAMT PH2 NRZ O1	
PH3 T5L	One clock long				
	Adjust sign if overflow	FUMTSIGN	=	FAMT PH3 NINTRAP (CC2 + NOU5) + ...	CC2 set during PH2 if overflow detected
		FUMTOVER	=	FUMTSIGN NFL3	Flip-flop FL3 not set during MTH sequence
	Exchange CC3 and CC4	S/CC3	=	FUMTOVER CC4 + ...	
		R/CC3	=	FUMTSIGN + ...	
		S/CC4	=	FUMTOVER CC3 + ...	
		R/CC4	=	FUMTSIGN + ...	
	Branch to PRE4	BRPRE4	=	FAMT PH3	
					Mnemonic: MTH (53, D3)

(Continued)

Table 3-55. Modify and Test Halfword Sequence (Cont.)

Phase	Function Performed	Signals Involved			Comments
PRE4 T5L	Sustained until byte counter zero				
	Circular left shift of A-register one byte for each clock	AXAL8	=	FAMT PRE4 SW2 NBCZ	Repeated while BCZ false
		BCZ	=	NBC0 NBC1	
	Branch to PRE4	BRPRE4	=	PRE4 NBCZ	Remain in PRE4 while BCZ false
	Enable signal PRE/34	PRE/34	=	PRE4 NBC1 NBC0 NANLZ	Terminate PRE4 after BCZ true
	Enable signal (S/SXA)	(S/SXA)	=	FAMT (PRE/34 + PH2)	Preset adder for A ⟶ S transfer in PH8
	Enable signal (S/MBXS)	(S/MBXS)	=	FAMT PRE/34 SW2	Preset for S ⟶/⟶ MB transfer in PH8
	Set flip-flop DRQ	S/DRQ	=	(S/MBXS) + ...	Inhibits transmission of another clock until data release from core memory
		R/DRQ	=	...	
	Branch to PH8	BRPH8	=	FAMT PRE/34 SW2	
PH8 DR	Sustained until DR				
	(A0 - A31) ⟶ (S0 - S31)	Adder logic set at PRE4 clock			
	(S0 - S31) ⟶/⟶ (MB0 - MB31)	MBXS	=	Set at PRE4 clock	
PH9 T5L	One clock long				
	(B0 - B31) ⟶ (S0 - S31)	Sn	=	Bn SXB	Program address
	(S15 - S31) ⟶/⟶ (P15-P31)	S/Pn	=	Sn PXS	Program address bits only
		SXB	=	PXSXB	
		PXS	=	PXSXB	
		PXSXB	=	NFAFL NFAMDS PH9	
		R/PM	=	PX + ...	
	Set flip-flop MRQ	S/MRQ	=	(S/MRQ/2) + ...	Core memory request for effective word
		(S/MRQ/2)	=	PXSXB NINTRAP + ...	
		R/MRQ	=	...	
	Set flip-flop DRQ	S/DRQ	=	(S/MRQ/2) NCLEAR	Inhibits transmission of another clock until data release from core memory
		R/DRQ	=	...	
		R/INTRAP	=	FAMT PH4 + ...	Reset if INTRAP
PH10 DR	ENDE functions				

Mnemonic: MTH (53, D3)

MODIFY AND TEST WORD (MTW; 33, B3). The MTW instruction performs one of two operations, depending upon the value in the R field of the instruction word (bits 8 through 11). If the value is zero, the effective word is tested to determine if it is zero, negative, or positive, and the condition code flip-flops are set accordingly. If the value is not zero, the four bits are treated as a signal quantity, the sign (bit 8) is extended to form a word, and this word is effectively added to the effective word. The resulting value is loaded into the effective byte location, and the condition codes are set according to the result. The effective word is thereby modified by a value of -8 to +7 and tested.

If fixed-point overflow occurs, CC2 is set to 1, and the computer traps to location X'43' if the fixed-point arithmetic mask (AM) is 1. The trap occurs after the result is stored in the effective word location. If the MTW instruction is executed in an interrupt location, the condition code is not affected, and no fixed-point overflow trap can occur.

Condition Codes. Condition codes for the MTW instruction are:

CC1	CC2	CC3	CC4	Result in EW Location
-	-	1	0	Positive
-	-	0	0	Zero
-	-	0	1	Negative
-	0	-	-	No fixed-point overflow
-	1	-	-	Fixed-point overflow
0	-	-	-	No carry from word
1	-	-	-	Carry from word

Examples. Examples of the MTW instruction are:

Instruction
0011 0011 1011 XXXX XXXX XXXX XXXX XXXX

Effective word
0000 0000 0000 0000 0000 0001 0011 1001

(EW + R)
0000 0000 0000 0000 0000 0001 0011 0100

Condition code: 0010

Instruction
0011 0011 0110 XXXX XXXX XXXX XXXX XXXX

Effective word
1111 1111 1111 1111 1111 1111 1111 1010

(EW + R)
0000 0000 0000 0000 0000 0000 0000 0000

Condition code: 0000

Modify and Test Word Phase Sequence. Preparation phases for the Modify and Test Word instruction are the same as the general PREP phases for word instructions, paragraph 3-59. Figure 3-153 shows the simplified phase sequence for the MTW instruction. Table 3-56 lists the detailed logic sequence during all MTW execution phases.

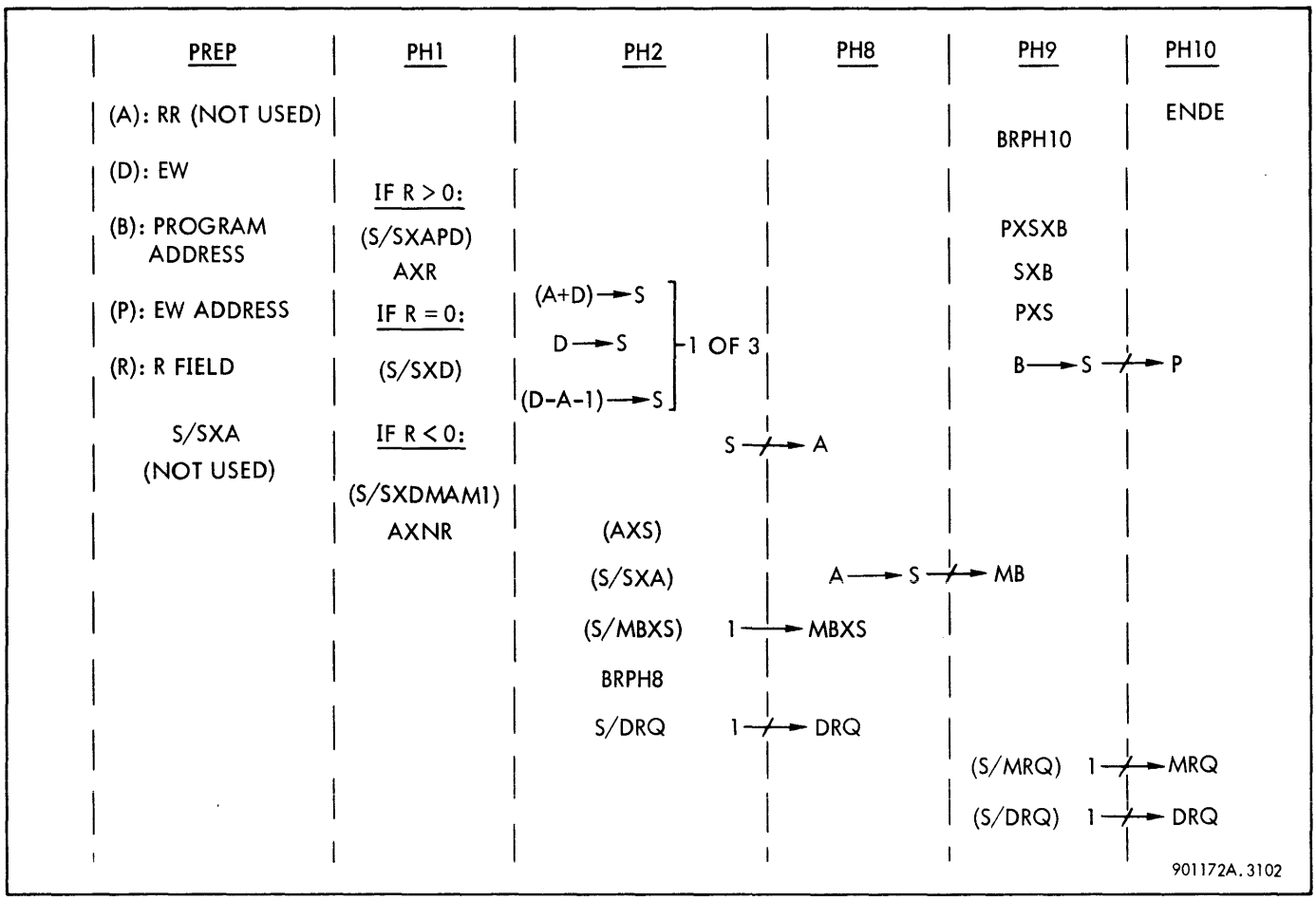


Figure 3-153. Modify and Test Word Instruction, Phase Sequence Diagram

Table 3-65. Modify and Test Word Sequence

Phase	Function Performed	Signals Involved			Comments
PREP	At end of PREP:				
	(A): RR				Private memory register R (not used)
	(D): EW				Effective word
	(B): Program address				Temporary storage for address of next instruction
	(P): EW address				Address of effective word
	(R): R field of instruction				Bits 28 - 31
	Enable signal (S/SXA)	(S/SXA)	=	FAMT (PRE/34 + PH2)	Not used
		FAMT	=	NOU1 OL3 O3	
PH1 T5L	One clock long				
	If R is equal to 0:				
	Enable signal (S/SXD)	(S/SXD)	=	FAMT PH1 RZ	Preset adder for D ⟶ S in PH2
		RZ	=	NR28 NR29 NR30 NR31	R field zero
	If R is less than 0:				
	Set flip-flop SW2	S/SW2	=	FAMT PH1 NRZ + ...	Not used for MTW
		R/SW2	=	RESET/A + ...	
	Enable signal AXNR	AXNR	=	FAMT PH1 R28	N(R field) ⟶/ A-register
	Enable signal (S/SXDMAM1)	(S/SXDMAM1)	=	FAMT PH1 R28	Preset adder for (D - A - 1) ⟶ S in PH2
	If R is greater than 0:				
	Set flip-flop SW2	S/SW2	=	FAMT PH1 NRZ + ...	
		R/SW2	=	RESET/A + ...	Not used for MTW
	Enable signal AXR	AXR	=	FAMT PH1 NR28	R field ⟶/ A-register
	Enable signal (S/SXAPD)	(S/SXAPD)	=	FAMT PH1 NR28 NRZ + ...	Preset adder for (A + D) ⟶ S in PH2
	If INTRAP, set flip-flop CEINT	S/CEINT	=	FAMT PH1 INTRAP + ...	Enable interrupt clock
		R/CEINT	=	...	
	(If NINTRAP, T5L or T8L)				
PH2 T5L or T8L	One clock long				
	(D0 - D31) ⟶ (S0 - S31)	Adder logic set at PH1 clock			If R = 0
	(D0 - D31) - (A0 - A31) - 1 ⟶ (S0 - S31)	Adder logic set at PH1 clock			If R less than 0
					Mnemonic: MTW (33, B3)

(Continued)

Table 3-56. Modify and Test Word Sequence (Cont.)

Phase	Function Performed	Signals Involved			Comments
PH2 T5L or T8L	(A0 - A31) + (D0 - D31) ⟶ (S0 - S31)	Adder logic set at PH1 clock			If R greater than 0
	(S0 - S31) ⟶/⟶ (A0 - A31)	AXS	=	FAMT PH2	Transfer result of operation to A-register
	Enable clock T8 if arithmetic operation required	T8EN	=	NT5EN NT11L N(SXADD/1 RW) N(RW REU) N(REU AXRR)	T5EN is disabled by SXADD/1 if addition or subtraction required
	If NINTRAP				
	Set condition code flip-flops	S/CC3	=	SGTZ TESTS + ...	
		TESTS	=	FAMT PH2 NINTRAP + ...	
		SGTZ	=	(S0 + S1 + ... + S31) S0 NFACOMP	
		R/CC3	=	TESTS + ...	State of CC3 and CC4 indicates polarity of data in A-register after arithmetic operation
		S/CC4	=	(S/CC4/2) TESTS + ...	
		(S/CC4/2)	=	NFACOMP S0 + ...	
		R/CC4	=	TESTS + ...	
		S/CC1	=	K00 CC1XK00 + ...	Set CC1 if end carry
		CC1XK00	=	FAMT PH2 NINTRAP	
		R/CC1	=	(R/CC) (R/CC1/1) (R/CC1/2) CC1XK00	
		S/CC2	=	(S00 ⊕ S0) PROBOVER	Set CC2 if overflow
		PROBOVER	=	FAMT PH2 NINTRAP	
		R/CC2	=	(R/CC) (R/CC2/1) (R/CC2/2) CC1XK00	
	If INTRAP				
	Activate highest priority	LEVACT	=	FAMT PH2 INTRAP	
	Arm highest priority	LEVARM	=	FAMT PH2 INTRAP	
	Trigger count zero	CNTZERO	=	FAMT PH2 INTRAP S0031Z	
	INTRAP or NINTRAP				
	Enable signal (S/SXA)	(S/SXA)	=	FAMT (PRE/34 + PH2)	Preset adder for A ⟶ S in PH8
	Enable signal (S/MBXS) if R ≠ 0	(S/MBXS)	=	FAMT PH2 NO1 NRZ	Preset for S ⟶/⟶ MB transfer in PH8
	Branch to PH8	BRPH8	=	FAMT PH2 (NO1 + RZ)	Inhibits transmission of another clock until data release from core memory
	Set flip-flop DRQ	S/DRQ	=	(S/MBXS) + ...	
		R/DRQ	=	...	
					Mnemonic: MTW (33, B3)

(Continued)

Table 3-56. Modify and Test Word Sequence (Cont.)

Phase	Function Performed	Signals Involved			Comments
PH8 DR	Sustained until DR (A0 - A31) ⟶ (S0 - S31) (S0 - S31) ⟶/ (MB0 - MB31)	Adder logic set at PH2 clock MBXS	=	Set at PH2 clock	
PH9 T5L	One clock long (B0 - B31) ⟶ (S0 - S31) (S15 - S31) ⟶/ (P15 - P31)	SXB PXS PXSXB S/MRQ (S/MRQ/2) R/MRQ S/DRQ R/DRQ R/INTRAP	= = = = = = = = =	PXSXB + ... PXSXB + ... NFAFL NFAMDS PH9 (S/MRQ/2) + ... PXSXB NINTRAP + ... (S/MRQ/2) NCLEAR ... FAMT PH9	Program address Program address bits only Core memory request for effective word Inhibits transmission of another clock until data release from core memory Reset if INTRAP
PH10 DR	ENDE functions				
					Mnemonic: MTW (33, B3)

3-70 Family of Compare Instructions

COMPARE IMMEDIATE (CI; 21). The Compare Immediate instruction compares the contents of private memory register R with the sign-extended value field of the instruction word, and sets the condition code according to the results of the comparison.

General. Both operands are treated as signed fixed point quantities. The value field (sign extended) is subtracted from the contents of register R, and condition code flip-flops CC3 and CC4 are set to indicate the results of the comparison. An AND operation is performed on the two operands. Flip-flop CC2 is set if the result of the AND operation is nonzero and reset if the result is zero.

Condition Codes. Condition codes for the CI instruction are:

CC2	CC3	CC4	Result of Comparison
-	0	0	Operands are equal
-	0	1	Register word less than immediate value
-	1	0	Register word greater than immediate value
0	-	-	Logical product (AND) of operands is zero
1	-	-	Logical product of operands is nonzero

Examples. Examples of the CI instruction are:

RR	1111 1111 1111 1011 0010 0101 1100 0011
EW	1111 1111 1111 1001 1001 1011 0001 0110
RR - EW	0000 0000 0000 0001 1000 1010 1010 1101
EW AND RR	1111 1111 1111 1001 0000 0001 0000 0010

Condition code: X101

RR	1111 1111 1111 1111 1101 0110 0010 0101
EW	1111 1111 1111 1111 1111 1001 0100 1001
RR - EW	1111 1111 1111 1111 1101 1100 1101 1100
EW AND RR	1111 1111 1111 1111 1101 0000 0000 0001

Condition code: X110

Compare Immediate Phase Sequence. Preparation phases for the Compare Immediate instruction are the same as the general PREP phases for immediate instructions, paragraph 3-59. Figure 3-154 shows the simplified phase sequence for the Compare Immediate instruction. Table 3-57 lists the detailed logic sequence during all Compare Immediate execution phases.

COMPARE BYTE (CB; 71, F1). The Compare Byte instruction compares the contents of bit positions 24 through 31 of private memory register R with the effective byte and sets the condition code according to the results of the comparison.

General. Both bytes are treated as positive integer magnitudes. The effective byte is subtracted from the contents of register R, and condition code flip-flops CC3 and CC4 are set to indicate the results of the operation. An AND operation is performed on the two bytes. Flip-flop CC2 is set if the result of the AND operation is nonzero and reset if the result is zero.

Condition Codes. Condition codes for the CB instruction are:

CC2	CC3	CC4	Result of Comparison
-	0	0	Operands are equal
-	0	1	Register byte less than effective byte
-	1	0	Register byte greater than effective byte
0	-	-	Logical product (AND) of operands is zero
1	-	-	Logical product of operands is nonzero

Examples. Examples of the CB instruction are:

RR	0000 0000 0000 0000 0000 0000 0100 1001
EW	0000 0000 0000 0000 0000 0000 0101 0101
(RR - EW)	1111 1111 1111 1111 1111 1111 1111 0100
RR AND EW	0000 0000 0000 0000 0000 0000 0100 0001

Condition code: X101

RR	0000 0000 0000 0000 0000 0000 0010 0110
EW	0000 0000 0000 0000 0000 0000 0010 0110
(RR - EW)	0000 0000 0000 0000 0000 0000 0000 0000
RR AND EW	0000 0000 0000 0000 0000 0000 0010 0110

Condition code: X100

Compare Byte Phase Sequence. Preparation phases for the Compare Byte instruction are the same as the general PREP phases for byte instructions, paragraph 3-59. Figure 3-154 shows the simplified phase sequence for the Compare Byte instruction. Table 3-57 lists the detailed logic sequence during all Compare Byte execution phases.

COMPARE HALFWORD (CH; 51, D1). The Compare Halfword instruction compares the contents of private memory

register R with the effective halfword and sets the condition code according to the results of the comparison.

General. Both operands are treated as signed fixed-point quantities. The effective halfword (sign extended) is subtracted from the contents of register R, and condition code flip-flops CC3 and CC4 are set to indicate the results of the comparison. An AND operation is performed on the two operands. Flip-flop CC2 is set if the result of the AND operation is nonzero and reset if the result is zero.

Condition Codes. Condition codes for the CH instruction are:

CC2	CC3	CC4	Result of Comparison
-	0	0	Operands are equal
-	0	1	Register word less than effective halfword
-	1	0	Register word greater than effective halfword
0	-	-	Logical product (AND) of operands is zero
1	-	-	Logical product of operands is nonzero

Examples. Examples of the CH instruction are:

RR	0000 0000 0000 0000 0010 1001 0101 0100
EW	0000 0000 0000 0000 0110 1011 1000 0011
(RR - EW)	1111 1111 1111 1111 1011 1101 1101 0001
RR AND EW	0000 0000 0000 0000 0010 1001 0000 0000

Condition code: X101

RR	1111 1111 1111 1111 1100 1001 0110 1101
EW	1111 1111 1111 1111 1000 0111 1101 0010
(RR - EW)	0000 0000 0000 0000 0100 0001 1001 1011
RR AND EW	1111 1111 1111 1111 0000 0001 0100 0000

Condition code: X110

Compare Halfword Phase Sequence. Preparation phases for the Compare Halfword instruction are the same as the general PREP phases for halfword instructions, paragraph 3-59. Figure 3-154 shows the simplified phase sequence for the Compare Halfword instruction. Table 3-57 lists the detailed logic sequence during all compare halfword execution phases.

COMPARE WORD (CW; 31, B1). The Compare Word instruction compares the contents of private memory register R with the effective word and sets the condition code according to the results of the comparison.

General. Both operands are treated as signed fixed-point quantities. The contents of register R are subtracted from the effective word, and condition code flip-flops CC3 and CC4 are set to indicate the results of the comparison. An AND operation is performed on the two operands. Flip-flop CC2 is set if the result of the AND operation is nonzero and reset if the result is zero.

Condition Codes. Condition codes for the CW instruction are:

CC2	CC3	CC4	Result of Comparison
-	0	0	Operands are equal
-	0	1	Register word less than effective word
-	1	0	Register word greater than effective word
0	-	-	Logical product (AND) of operands is zero
1	-	-	Logical product of operands is nonzero

Examples. Examples of the CW instruction are:

RR	0000 0000 0010 1011 0110 0101 1101 1001
EW	0000 0000 1000 0100 0001 1000 0000 0010
(RR - EW)	1111 1111 1010 0111 0100 1101 1101 0111
RR AND EW	0000 0000 0000 0000 0000 0000 0000 0000

Condition code: X001

RR	1111 1111 1001 0010 0111 1011 0100 1100
EW	1111 1111 1100 0111 0100 1110 1110 0110
(RR - EW)	0000 0000 0011 0100 1101 0011 1001 1010
EW AND RR	1111 1111 1000 0010 0100 1010 0100 0100

Condition code: X110

Compare Word Phase Sequence. Preparation phases for the Compare Word instruction are the same as the general PREP phases for word instructions, paragraph 3-59. Figure 3-154 shows the simplified phase sequence for the Compare Word instruction. Table 3-57 lists the detailed logic sequence during all Compare Word execution phases.

Table 3-57. Compare Sequence (CI, CB, CH, CW)

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		Sign-extended value field or contents of effective address (word, halfword, or byte) properly aligned
	(C): EW or value field		
	(D): EW or value field		
	(A): RR		Contents of private memory register R
	(P): Program address		Address of next instruction in sequence
	For all instructions:		
	Enable signal (S/SXAMD)	(S/SXAMD) = FASUB PRE/34 + ...	Preset adder for (A - D)
		FASUB = OL1 + ...	⟶S in PH1
	Set flip-flop MRQ	S/MRQ = (S/MRQ/1) + ...	Core memory request for next instruction in sequence
		(S/MRQ/1) = FAS10 PRE/34 + ...	
		FAS10 = FAS11/1 NOU1 + ...	
		FAS11/1 = OL1 + ...	
		R/MRQ = ...	
	Reset flip-flop S/NT11L	S/NT11L = N(S/T11L)	Signals T5EN and T8EN disabled when NT11L reset, permitting T11 clock
		(S/T11L) = FACOMP/1 (PRE/34 + PH2) + ...	
		FACOMP/1 = OL1	
		R/NT11L = ...	
	For CB only:		
	Enable signal AXZ/012	AXZ/012 = FACOMP/1 OU7 PRE4	Stores zeros in A0 through A23
PH1 T11L	One clock long		
	(A0-A31) -(D0-D31)⟶(S0-S31)	Adder logic set at PH1 clock	(RR-EW) on sum bus output
	Set condition code flip-flop CC3 or CC4	S/CC3 = SGTZ TESTS + ...	Set if sum bus output greater than zero
		R/CC3 = TESTS + ...	
			Mnemonic: CI(21), CB (71, F1), CH(51, D1), CW (31, B1)

(Continued)

Table 3-57. Compare Sequence (CI, CB, CH, CW) (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH1 T11L (Cont.)		S/CC4 = (S/CC4/2) TESTS + ... (S/CC4/2) = FACOMP S00 + ... R/CC4 = TESTS + ... TESTS = FAS11 PH1 + PH3) + ... FAS11 = FAS11/1 NFALCFP	Set if sum bus output less than zero
		SGTZ = (S0 + S1 + ... + S31) N(S00 FACOMP)	(S0-S31) greater than zero
	Enable signal (S/PRXAD)	(S/PRXAD) = FACOMP/1 PH1 NOU1	Preset adder for A AND D ⟶ S in PH10
	Set flip-flop DRQ	S/DRQ = BRPH10 NCLEAR + ... R/DRQ = ...	Inhibit transmission of another clock until data release from memory
		BRPH10 = FAS10 PH10 + ...	
	Branch to PH10	S/PH10 = NCLEAR BRPH10 + ... R/PH10 = ...	
PH10	(A0-A31) AND (D0-D31) ⟶ (S0-S31)	Adder logic set at PH1 clock	A AND D ⟶ sum bus
	Set flip-flop CC2 if A AND D = 0	S/CC2 = NS0031Z (S/CC2/NZ) + ...	
		(S/CC2/NZ) = (FACOMP ENDE) NOU1	
		R/CC2 = (R/CC2/1) + ...	
	ENDE functions	(R/CC2/1) = (S/CC2/NZ) + ...	
			Mnemonic: CI(21), CB (71, F1), CH(51, D1), CW (31, B1)

COMPARE DOUBLEWORD (CD; 11, 91). The Compare Doubleword instruction compares the contents of private memory registers R and Ru1 with the effective doubleword and sets the condition code according to the results of the comparison.

General. Both doublewords are treated as signed fixed-point quantities. The least significant word of the effective doubleword is subtracted from the contents of register Ru1; the most significant word of the effective doubleword is subtracted from the contents of register R. If the R field of the CD instruction is an odd value, the CD instruction forms a 64-bit register operand by duplicating the contents of register R for both the 32 high-order bits and the 32 low-order bits. Condition code flip-flops CC3 and CC4 are set to indicate the results of the 64-bit comparison.

Condition Codes. Condition codes for the CD instruction are:

CC3	CC4	Result of Comparison
0	0	Operands are equal
0	1	Register doubleword less than effective doubleword
1	0	Register doubleword greater than effective doubleword

Examples. Examples of the CD instruction are:

Even R Field

Ru1	0000 0101 1101 1011 0010 0110 1000 1100
ED_{LSW}	0000 0011 1000 0101 1001 0111 1110 0000
	0000 0010 0101 0110 1000 1110 1010 1100
R	1101 0110 1001 1011 0100 1000 0101 1010
ED_{MSW}	1101 0110 1001 1011 0100 1000 0101 1010
	0000 0000 0000 0000 0000 0000 0000 0000

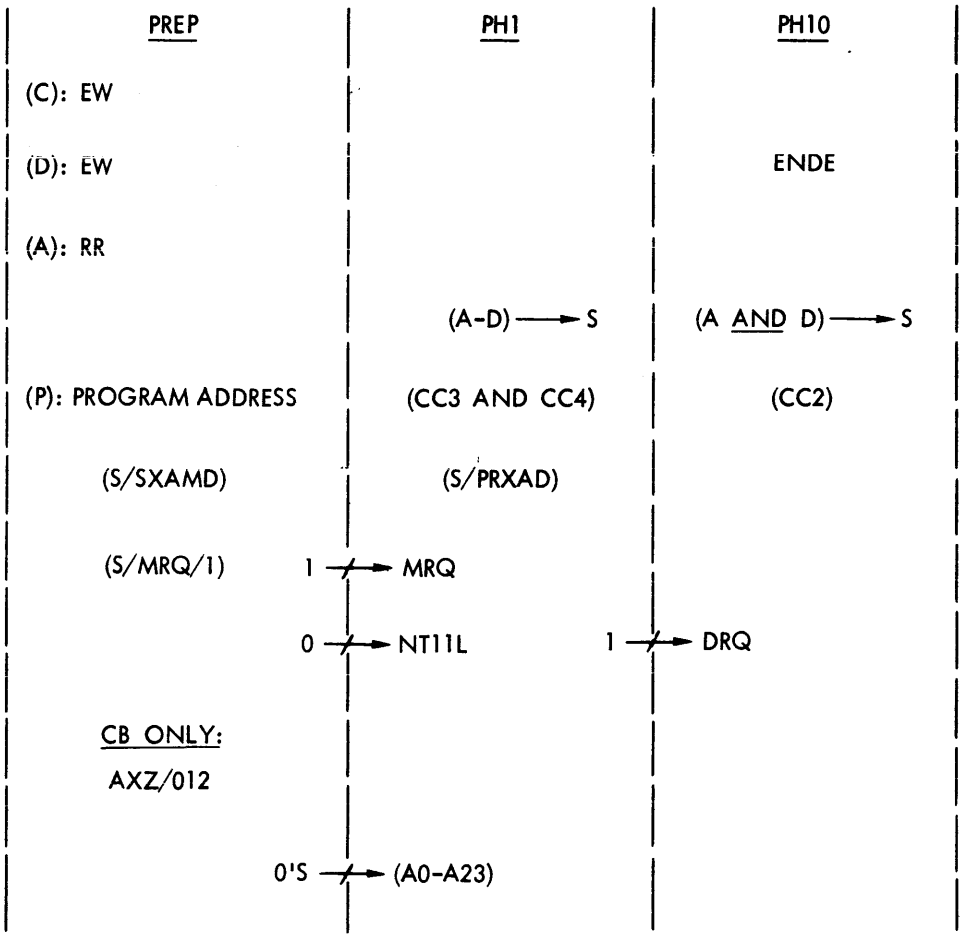
Condition code: XX10

Odd R Field

Ru1	1101 0110 1010 0110 0101 1110 0000 0011
ED_{LSW}	0000 0011 1000 0101 1001 0111 1110 0000
	1101 0011 0010 0000 1100 0110 0010 0011
Ru1	1101 0110 1010 0110 0101 1110 0000 0011
ED_{MSW}	1101 0110 1001 1011 0100 1000 0101 1010
	0000 0000 0000 1011 0001 0101 1010 1001

Condition code: XX10

Compare Doubleword Phase Sequence. Preparation phases for the CD instruction are the same as the general PREP phases for doubleword instructions. Figure 3-155 shows the simplified phase sequence for the CD instruction. Table 3-58 lists the detailed logic sequence during all CD execution phases.



901172A.3111

Figure 3-154. Compare Immediate, Compare Byte, Compare Halfword, and Compare Word Instructions, Phase Diagram

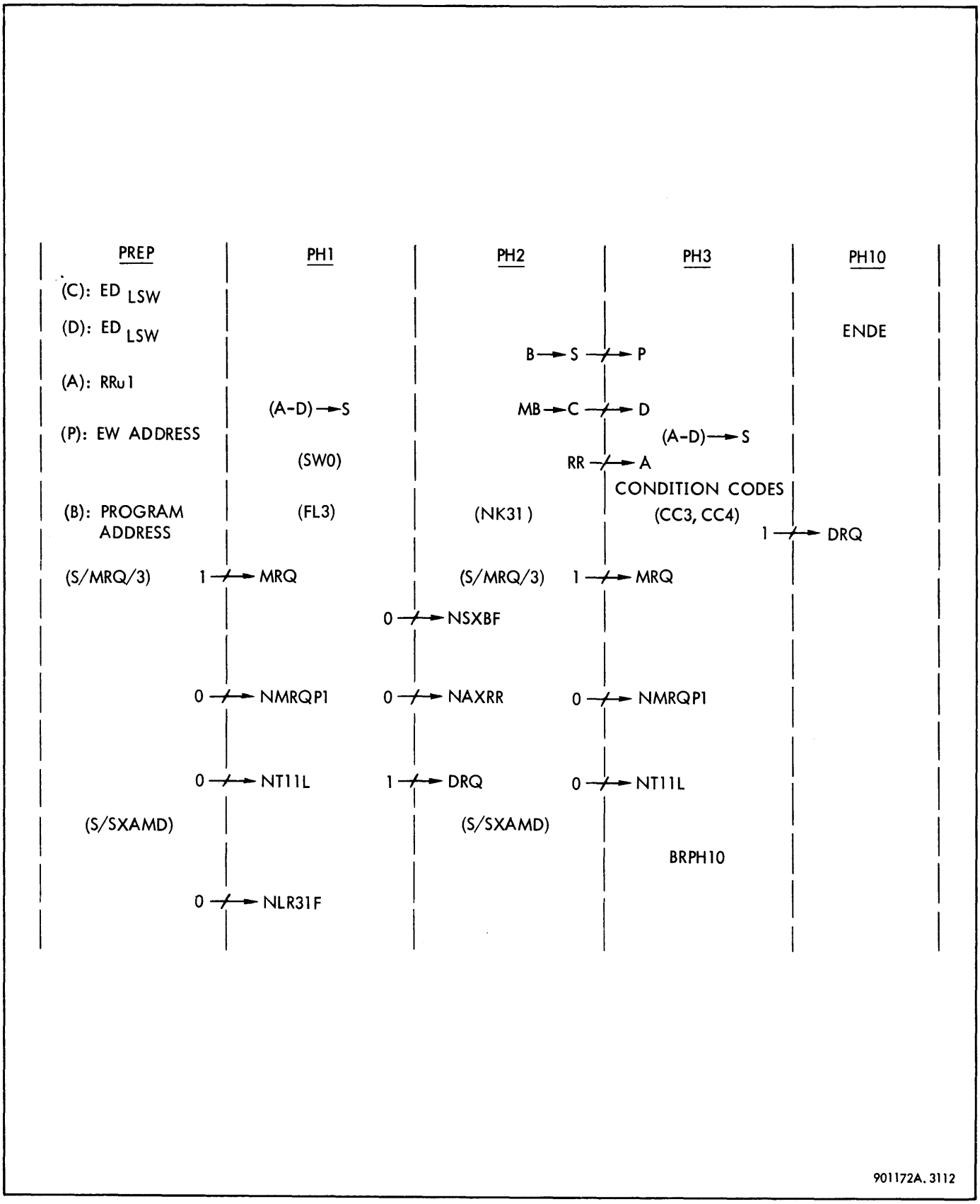


Figure 3-155. Compare Doubleword Instruction, Phase Diagram

Table 3-58. Compare Doubleword Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	**At end of PREP:**		
	(C): ED_{LSW}		Least significant word of effective doubleword
	(D): ED_{LSW}		
	(A): RRu1		Contents of private memory register Ru1
	(P): EW_{MSW} address		Address of most significant word of effective doubleword
	(B): Program address		Temporary storage for address of next instruction
	Set flip-flop MRQ	S/MRQ = (S/MRQ/3) +... (S/MRQ/3) = FADW/1 PRE/34 +... FADW/1 = OU1 FAS11 +... FAS11 = FAS11/1 NFALCFP FAS11/1 = OL1 +... R/MRQ = ...	Core memory request for most significant word of effective doubleword
	Reset flip-flop NMRQP1	S/NMRQP1 = (S/MRQ/3) R/NMRQP1 = ...	MRQP1 sets flip-flop DRQ at PH1 clock
	Reset flip-flop NT11L	S/NT11L = N(S/T11L) +... (S/T11L) = FACOMP/1 PRE/34 +... FACOMP/1 = OL1 +... R/NT11L = ...	Set clock T11L for PH1
	Enable signal (S/SXAMD)	(S/SXAMD) = FASUB PRE/34 +... FASUB = OL1 +...	Preset adder for (A-D) ⟶ S in PH1
	Reset flip-flop NLR31F	S/NLR31F = N(S/LR31) (S/LR31F) = FADW/1 NANLZ PRE3 +... R/NLR31F = ...	Force a one on private memory address line LR31 during PH1 to select private memory register Ru1
PH1 T11L	One clock long (A0-A31) -(D0-D31)⟶(S0-S31)	Adder logic set at last PREP clock	Adder output is ($RRu1-ED_{LSW}$)
			Mnemonic: CD (11, 91)

(Continued)

Table 3-58. Compare Doubleword Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH1 T11L (Cont.)	Reset flip-flop NSXBF	S/NSXBF = N(S/SXB) (S/SXB) = FADW/1 PH1 + ... R/NSXBF = ...	Preset logic for B ⟶ S in PH2
	Reset flip-flop NAXRR	S/NAXRR = N(S/AXRR) (S/AXRR) = FADW/1 PH1 + ... R/NAXRR = ...	Preset for R ⟶/⟶ A transfer in PH2
	Set flip-flop SW0 if (S0-S31) not zero	S/SW0 = NS0031Z (S/SW0/NZ) + ... (S/SW0/NZ) = K00HOLD + ... K00HOLD = FADW/1 PH1 + ... R/SW0 = RESET/A = CLEAR = PH10	Retain information that S not zero. Condition codes CC3 and CC4 may also be set during this phase, but action is meaningless since they are again set in PH3
	Set flip-flop FL3 if end carry	S/FL3 = K00 K00HOLD + ... R/FL3 = ...	Retain end carry
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = MRQP1 + ... R/DRQ = ...	Inhibits transmission of another clock until data release received from core memory
PH2 DR	Sustained until DR (B0-B31) ⟶ (S0-S31)	SXB = NDIS SXBF SXBF = set at PH1 clock	Transfer program address to P-register
	(S15-S31) ⟶/⟶ (P15-P31)	PXS = FADW/1 PH2 + ...	
	(MB0-MB31) ⟶ (C0-C31)	CXMB = DG = /DG/	Transfer most significant word of effective doubleword to C- and D-registers
	(C0-C31) ⟶/⟶ (D0-D31)	DXC = FADW/1 PH2 + ...	
	(RR0-RR31) ⟶/⟶ (A0-A31)	AXRR = set at PH1 clock	Private memory register Ru1 ⟶/⟶ A-register
	Enable signal (S/SXAMD)	(S/SXAMD) = FASUB PH2 + ... FASUB = OL1 + ...	Preset adder for (A-D) ⟶ S in PH3
	Set flip-flop MRQ	S/MRQ = (S/MRQ/3) + ... (S/MRQ/3) = FADW/1 PH2 + ... R/MRQ = ...	Core memory request for next instruction in sequence
	Reset flip-flop NMRQP1	S/NMRQP1 = (S/MRQ/3) R/NMRQP1 = ...	MRQP1 sets flip-flop DRQ at PH3 clock
			Mnemonic: CD (11, 91)

(Continued)

Table 3-58. Compare Doubleword Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH2 DR (Cont.)	Reset flip-flop NK31 if there was an end carry in PH1	S/NK31 = N(S/K31) N(S/SXAMD/2) + N(S/K31/1) (S/K31/1) = K00 (S/K31/3) + ... (S/K31/3) = FADW/1 PH2 FL3 + ... (S/K31) = FADW/1 PH2 + ... R/NK31 = ...	Setting K31 effectively provides a carry (or borrow) to the most significant half of the effective doubleword
	Reset flip-flop NT11L	S/NT11L = N(S/T11L) + ... (S/T11L) = FACOMP/1 PH2 + ... R/NT11L = ...	Set clock T11L for PH3
PH3 T11L	One clock long		
	(A0-A31) -(D0-D31) ⟶ (S0-S31)	Adder logic set at PH2 clock	Adder output is ($RRu1\text{-}ED_{MSW}$)
	Set flip-flop CC3 if (S0-S63) nonzero and positive	S/CC3 = SGTZ TESTS + ... SGTZ = (S0 + S1 + ... + S31) N(S00 FACOMP) S3263Z S3263Z = NSW0 NTESTS/1 + ... TESTS/1 = FUSF ENDE + FAMULNH PH7 TESTS = FAS11 (PH1 + PH3) R/CC3 = TESTS + ...	Result is nonzero and positive TESTS/1 signal enables SW0 to control SGTZ signal
	Set flip-flop CC4 if (S0-S63) negative	S/CC4 = (S/CC4/2) TESTS + ... (S/CC4/2) = FACOMP S00 R/CC4 = TESTS + ...	Result is negative
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = MRQP1 + ... R/DRQ = ...	Inhibits transmission of another clock until data release received from core memory
	Branch to PH10	BRPH10 = FADW/1 PH3 + ... S/PH10 = BRPH10 NCLEAR + ... R/PH10 = ...	
PH10 DR	ENDE functions		

Mnemonic: CD (11, 91)

3-71 Family of Compare With Limits Instructions (FACOMP/L)

COMPARE WITH LIMITS IN REGISTER (CLR; 39, B9). The Compare With Limits in Register instruction simultaneously compares the effective word with the contents of private memory register R and with the contents of private memory register Ru1 and sets the condition codes according to the results. For these comparisons, all three words are treated as signed fixed point numbers.

General. Condition code flip-flops CC3 and CC4 indicate whether the contents of R are greater than (10), equal to (00), or less than (01), the effective word. Condition code flip-flops CC1 and CC2 indicate whether the contents of Ru1 are greater than, less than, or equal to the effective word. If the R field of the instruction word contains an odd value, both pairs of flip-flops will be in identical states.

Examples. Examples of the Compare with Limits in Register operation are:

EW 0000 0100 1100 0101 1010 0110 1111 1101
R 0000 0001 1101 1111 0101 1010 0110 0011
Ru1 0000 0100 1100 0101 1010 0110 1111 1101

Condition code: 0001

EW 0000 0100 1100 0101 1010 0110 1111 1101
R 0000 1001 0101 0110 0011 0111 1011 0100
Ru1 0000 1001 0101 0110 0011 0111 1011 0100

Condition code: 1010

Condition Codes. Condition code settings for the Compare With Limits in Register operation are:

CC1	CC2	CC3	CC4	Results of Comparison
-	-	0	0	Contents of R equal to effective word
-	-	0	1	Contents of R less than effective word
-	-	1	0	Contents of R greater than effective word
0	0	-	-	Contents of Ru1 equal to effective word
0	1	-	-	Contents of Ru1 less than effective word
1	0	-	-	Contents of Ru1 greater than effective word

CLR Phase Sequence. The preparation phases for the CLR instruction are the same as the general PREP phases for word instructions, paragraph 3-59. Figure 3-156 shows the simplified phase sequence for the execution of CLR instruction. Table 3-59 lists the detailed logic sequence during all CLR execution phases.

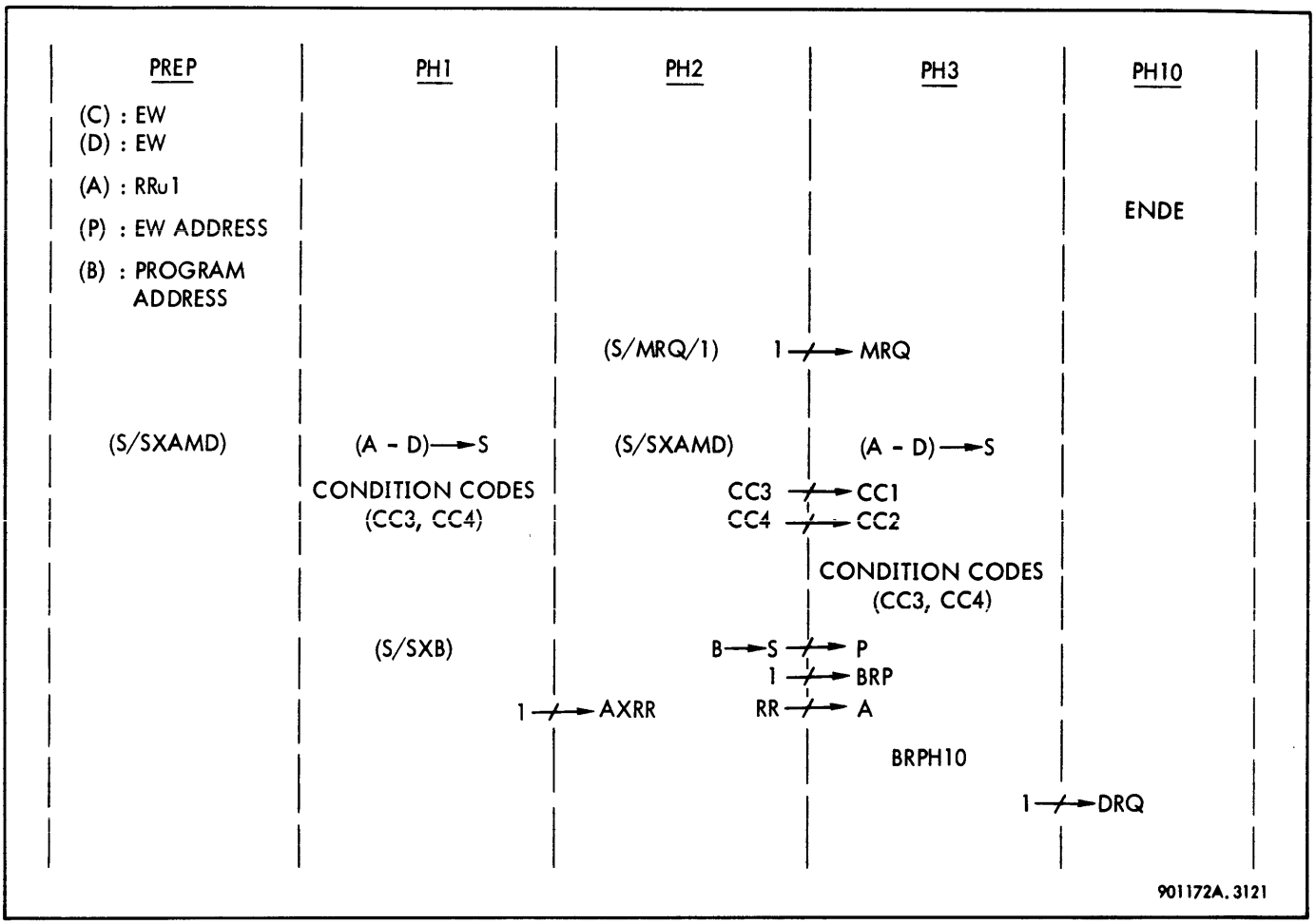


Figure 3-156. Compare With Limits in Register, Phase Diagram

Table 3-59. Compare With Limits in Register Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(C) : EW		Effective word
	(D) : EW		Effective word
	(A) : RRu1		Contents of private memory register R
	(P) : EW address		Address of effective word
	(B) : Program address		Temporary storage for address of next instruction
		Mnemonic: CLR (39, B9)	

(Continued)

Table 3-59. Compare With Limits in Register Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PREP	Reset flip-flop NT11L	S/NT11L = N(S/T11L) + ... (S/T11L) = FACOMP/L (PRE/34 + PH2) + ... FACOMP/L = OL9 NO1 O3 R/NT11L = ...	Set clock T11L for PH1
	Enable signal (S/SXAMD)	(S/SXAMD) = FASUB (PRE/34 + PH2) + ... FASUB = FACOMP/L + ...	Preset adder for (A-D) ⟶ S in PH1
PH1	One clock long		
T11L	(A0-A31) - (D0-D31) ⟶ (S0-S31)	Adder logic set at last PREP clock	Adder output is (RRu1-EW)
	Set flip-flop CC3 if (S0-S31) is nonzero and positive	S/CC3 = SGTZ TESTS + ... SGTZ = (S0 + S1 + ... + S31) N(S00 FACOMP) + ... TESTS = FACOMP/L PH1 + ... R/CC3 = TESTS + ...	Result is nonzero and positive
	Set flip-flop CC4 if (S0-S31) is negative	S/CC4 = (S/CC4/2) TESTS + ... (S/CC4/2) = FACOMP S00 + ... R/CC4 = TESTS + ...	Result is negative
	Reset flip-flop NSXBF	S/NXBF = N(S/SXB) (S/SXB) = FACOMP/L PH1 + ... R/NSXBF = ...	Preset for B ⟶ S transfer in PH2
	Reset flip-flop NAXRR	S/NAXRR = N(S/AXRR) (S/AXRR) = FACOMP/L PH1 OU3 + ... R/NAXRR = ...	Preset for R ⟶/⟶ A transfer in PH2

Mnemonic: CLR (39, B9)

(Continued)

Table 3-59. Compare With Limits in Register Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH2	One clock long		
T5L	(RR0-RR31)—/→(A0-A31)	AXRR = Set at PH1 clock	Private memory register R—/→A-register
	(B0-B31)——→(S0-S31)	SXB = NDIS SXBF + ...	Transfer program address to P-register
		SXBF = Set at PH1 clock	
	(S15-S31)—/→(P15-P31)	PXS = FACOMP/L PH2 + ...	
	Set flip-flop BRP	S/BRP = FACOMP/L PH2 + ...	Signifies that program address is in P-register
		R/BRP = PRE1 NFAIM + INTRAP1 + ...	
	Transfer CC3—/→CC1	S/CC1 = CC3 FACOMP/L PH2 + ...	Prepare for new code bits in PH3
		R/CC1 = (R/CC) + ...	
		(R/CC) = FACOMP/L PH2 + ...	
	Transfer CC4—/→CC2	S/CC2 = CC4 FACOMP/L PH2 + ...	
		R/CC2 = (R/CC) + ...	
	Enable signal (S/SXAMD)	(S/SXAMD) = FASUB (PRE/34 + PH2) + ...	Preset adder for (A-D) ——→S in PH3
		FASUB = FACOMP/L + ...	
	Set flip-flop MRQ	S/MRQ = (S/MRQ/1) + ...	Core memory request for next instruction in sequence
		(S/MRQ/1) = FACOMP/L PH2 + ...	
		R/MRQ = ...	
	Reset flip-flop NT11L	S/NT11L = N(S/T11L) + ...	Set clock T11L for PH3
		(S/T11L) = FACOMP/L (PRE/34 + PH2) + ...	
		R/NT11L = ...	
PH3 T11L	One clock long		
	(A0-A31) - (D0-D31) ——→(S0-S31)	Adder logic set at PH2 clock	Adder output is (RR-EW)
	Set condition codes as described in PH1 with new adder output on sum bus	Same as PH1	Same as PH1

Mnemonic: CLR (39, B9)

(Continued)

Table 3-59. Compare With Limits in Register Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH3 T11L (Cont.)	Set flip-flop DRQ	S/DRQ = BRPH10 NCLEAR + ... BRPH10 = FACOMP/L PH3 + ... R/DRQ = ...	Inhibits transmission of another clock until data release from memory
	Branch to PH10	S/PH10 = BRPH10 NCLEAR + ... R/PH10 = ...	
PH10 DR	ENDE functions		

Mnemonic: CLR (39, B9)

COMPARE WITH LIMITS IN MEMORY (CLM; 19, 99). The Compare With Limits in Memory instruction simultaneously compares the contents of private memory register R with the 32 high-order bits of the effective doubleword and with the 32 low-order bits of the effective doubleword and sets the condition codes according to the results. For these comparisons all 32-bit words are treated as signed, fixed point numbers.

General. The state of flip-flops CC1 and CC2 indicates whether the contents of R are greater than (10), equal to (00), or less than (01), the least significant word (bits 32 through 63). Similarly, the state of CC3 and CC4 indicates the relation between the contents of R and the most significant word (bits 0 through 31).

Examples. Examples of the Compare With Limits in Memory instruction are:

ED_{0-31} 0101 1000 0110 1101 1000 0001 0111 1011

ED_{32-63} 0101 1100 0011 0100 0010 1100 1001 0101

R 0101 1000 0111 1000 1101 1000 0010 0110

Condition code: 0110

ED_{0-31} 0000 0110 1101 0111 1011 0101 1111 0011

ED_{32-63} 0000 0101 0110 1110 0010 0100 1001 0110

R 0000 0101 0110 1110 0010 0100 1001 0110

Condition code: 0100

Condition Codes. Condition code settings for the Compare With Limits in Memory operation are:

CC1	CC2	CC3	CC4	Results of Comparison
-	-	0	0	Contents of R equal to most significant word (bits 0 through 31 of doubleword)
-	-	0	1	Contents of R less than most significant word (bits 0 through 31 of doubleword)
-	-	1	0	Contents of R greater than most significant word (bits 0 through 31 of doubleword)
0	0	-	-	Contents of R equal to least significant word (bits 32 through 63 of doubleword)
0	1	-	-	Contents of R less than least significant word (bits 32 through 63 of doubleword)
1	0	-	-	Contents of R greater than least significant word (bits 32 through 63 of doubleword)

CLM Phase Sequence. The preparation phases for the CLM instruction are the same as the general PREP phases for word instructions, paragraph 3-59. Figure 3-157 shows the simplified phase sequence for the CLM instruction execution. Table 3-60 lists the detailed logic sequence during all CLM execution phases.

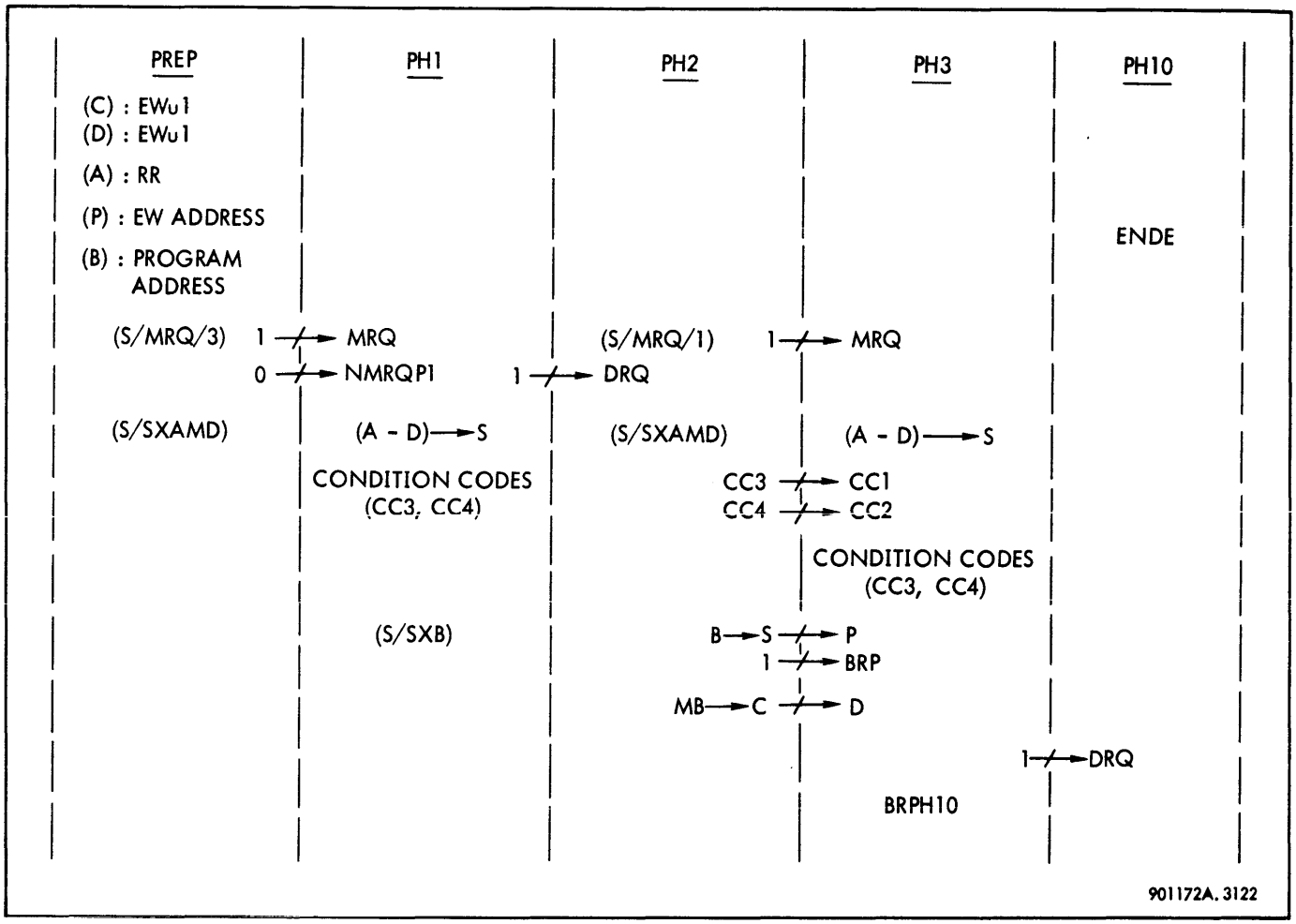


Figure 3-157. Compare With Limits in Memory, Phase Diagram

Table 3-60. Compare With Limits in Memory Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(C) : ED_{LSW}		Least significant word of effective doubleword
	(D) : ED_{LSW}		Least significant word of effective doubleword
	(A) : RR		Contents of private memory register R
		Mnemonic: CLM (19, 99)	

(Continued)

Table 3-60. Compare With Limits In Memory, Phase Diagram (Cont.)

Phase	Function Performed	Signals Involved	Comments
PREP (Cont.)	(P) : EW address		Address of most significant word of effective doubleword
	(B) : Program address		Temporary storage for address of next instruction
	Set flip-flop MRQ	S/MRQ = (S/MRQ/3) + ... (S/MRQ/3) = FACOMP/L PRE3 OU1 NANLZ + ... FACOMP/L = OL9 NO1 O3 R/MRQ = ...	Core memory request for most significant word of effective doubleword
	Reset flip-flop NMRQP1	S/NMRQP1 = (S/MRQ/3) R/NMRQP1 = ...	MRQP1 sets flip-flop DRQ at PH1 clock
	Reset flip-flop NT11L	S/NT11L = N(S/T11L) + ... (S/T11L) = FACOMP/L (PRE/34 + PH2) + ... R/NT11L = ...	Set clock T11L for PH1
	Enable signal (S/SXAMD)	(S/SXAMD) = FASUB (PRE/34 + PH2) + ... FASUB = FACOMP/L + ...	Preset adder for (A-D) ⟶S in PH1
PH1 T11L	One clock long		
	(A0-A31) - (D0-D31) ⟶(S0-S31)	Adder logic set at last PREP clock	Adder output is (RR-EWu1)
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = MRQP1 + ... R/DRQ = ...	Inhibits transmission of another clock until data release received from core memory

Mnemonic: CLM (19, 99)

(Continued)

Table 3-60. Compare With Limits in Memory Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH1 T11L	Set flip-flop CC3 If (S0-S31) is nonzero and positive	S/CC3 = SGTZ TESTS + ... TESTS = FACOMP/L PH1 + ... SGTZ = (S0 + S1 + ... + S31) N(S00 FACOMP) R/CC3 = TESTS + ...	Result is nonzero and positive
	Set flip-flop CC4 is (S0-S31) is negative	S/CC4 = (S/CC4/2) TESTS + ... (S/CC4/2) = FACOMP S00 + ... R/CC4 = TESTS + ...	Result is negative
	Reset flip-flop NSXBF	S/NXBF = N(S/SXB) (S/SXB) = FACOMP/L PH1 + ... R/NXBF = ...	Preset for B⟶S transfer in PH2
PH2 DR	Sustained until DR		
	(MB0-MB31)⟶(C0-C31)	CXMB = DG = /DG/	Transfer most significant word of effective doubleword to C- and D-registers
	(C0-C31)⟶(D0-D31)	DXC = FACOMP/L PH2 OU1	
	(B0-B31)⟶(S0-S31)	SXB = NDIS SXBF + ... SXBF = Set at PH1 clock	Transfer program address to P-register
	(S15-S31)⟶(P15-P31)	PXS = FACOMP/L PH2 + ...	
	Set flip-flop BRP	S/BRP = FACOMP/L PH2 + ... R/BRP = PRE1 NFAIM + INTRAP1 + ...	Signifies that program address is in P-register
	Transfer CC3⟶CC1	S/CC1 = CC3 FACOMP/L PH2 + ... R/CC1 = (R/CC) + ... (R/CC) = FACOMP/L PH2 + ...	Prepare for new code bits in PH3
	Transfer CC4⟶CC2	S/CC2 = CC4 FACOMP/L PH2 + ... R/CC2 = (R/CC) + ...	

Mnemonic: CLM (19, 99)

(Continued)

Table 3-60. Compare With Limits in Memory Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH2 DR (Cont.)	Enable signal (S/SXAMD)	(S/SXAMD) = FASUB (PRE/34 + PH2) + ... FASUB = FACOMP/L + ...	Preset adder for (A-D) ⟶S in PH3
	Set flip-flop MRQ	S/MRQ = (S/MRQ/1) + ... (S/MRQ/1) = FACOMP/L PH2 + ... R/MRQ = ...	Core memory request for next instruction in sequence
	Reset flip-flop NT11L	S/NT11L = N(S/T11L) + ... (S/T11L) = FACOMP/L (PRE/34 + PH2) + ... R/NT11L = ...	Set T11L clock for PH3
PH3 T11L	One clock long		
	(A0-A31) - (D0-D31)⟶ (S0-S31)	Adder logic set at last PH2 clock	Adder output is (RR-EW)
	Set condition code flip-flops CC1 and CC2 as described in PH1, with new output on sum bus	Same as PH1	Same as PH1
	Branch to PH10	BRPH10 = FACOMP/L PH3 + ... S/PH10 = BRPH10 NCLEAR + ... R/PH10 = ...	
PH10 DR	ENDE functions		

Mnemonic: CLM (19, 99)

3-72 Family of Logical Instructions (FALOGIC)

OR WORD (OR; 49, C9). The OR word instruction performs a logical OR operation on the contents of the effective word and private memory register R, and stores the result in register R.

General. If the corresponding bits of private memory register R and the effective word are both zero, a zero remains in R; otherwise, a one is placed in the corresponding bit position of register R. No change is made in the effective word. The operation is defined by the following equation, in which n denotes any bit position:

$$R_n = R_n + EW_n$$

Examples. Examples of the logical OR operation are:

EW	0000	1111	0101	1101	0110	0010	1010	1001
R	0011	0011	0110	1001	0000	1111	0101	0100

Before Execution

R	0011	1111	0111	1101	0110	1111	1111	1101

After Execution

Condition Codes. If the result in private memory register R is zero, the condition codes are XX00. If bit 0 of register R is a one, the condition codes are set to XX01. If bit 0 is a zero and bits 1 through 31 contain at least one 1, the condition codes are set to XX10.

OR Word Phase Sequence. Preparation phases for the OR instruction are the same as the general PREP phases for word instructions, paragraph 3-59. Figure 3-157 shows the simplified phase sequence for the OR word instruction. Table 3-61 lists the detailed logic sequence during OR word execution phases.

EOR WORD (EOR; 48, C8). The EOR word instruction performs a logical exclusive OR operation on the contents of the effective word and private memory register R and stores the result in register R.

General. If corresponding bits of register R and the effective word are different, a one is placed in the corresponding bit position of R. No change is made in the effective word. The operation is defined by the following equation, in which n denotes any bit position:

$$R_n = R_n NEW_n + NR_n EW_n$$

Examples: Examples of the exclusive OR operation are:

EW	0000	1111	0101	1101	0110	0010	1010	1001
R	0011	0011	1001	1001	0000	1111	0101	0100

Before Execution

R	0011	1100	1100	0100	0110	1101	1111	1101

After Execution

Condition Codes. If the result in private memory register R is zero, the condition codes are XX00. If bit 0 of register R is a one, the condition codes are set to XX01. If bit 0 is a zero, and bits 1 through 31 contain at least one 1, the condition codes are set to XX10.

EOR Word Phase Sequence. Preparation phases for the EOR instruction are the same as the general PREP phases for word instructions, paragraph 3-59. Figure 3-158 shows the simplified phase sequence for the EOR word instruction. Table 3-61 lists the detailed logic sequence during all EOR word execution phases.

AND WORD (AND; 4B, CB). The AND word instruction performs a logical AND operation on the contents of the effective word and private memory register R and stores the result in register R.

General. If the corresponding bits of register R and the effective word are both one, a one remains in R; otherwise, a zero is placed in the corresponding bit position of R. No change is made in the effective word. The operation is defined by the following equation, in which n denotes any bit position:

$$R_n = R_n EW_n$$

Examples: Examples of the logical AND operation are:

EW	0000	1111	0101	1101	0110	0010	1010	1001
R	0011	0011	0110	1001	0000	1111	0101	0100

Before Execution

R	0000	0011	0100	1001	0000	0010	0000	0000

After Execution

Condition Codes. If the result in register R is zero, the condition codes are set to XX00. If bit 0 of register R is a one, the condition codes are set to XX01. If bit 0 is a

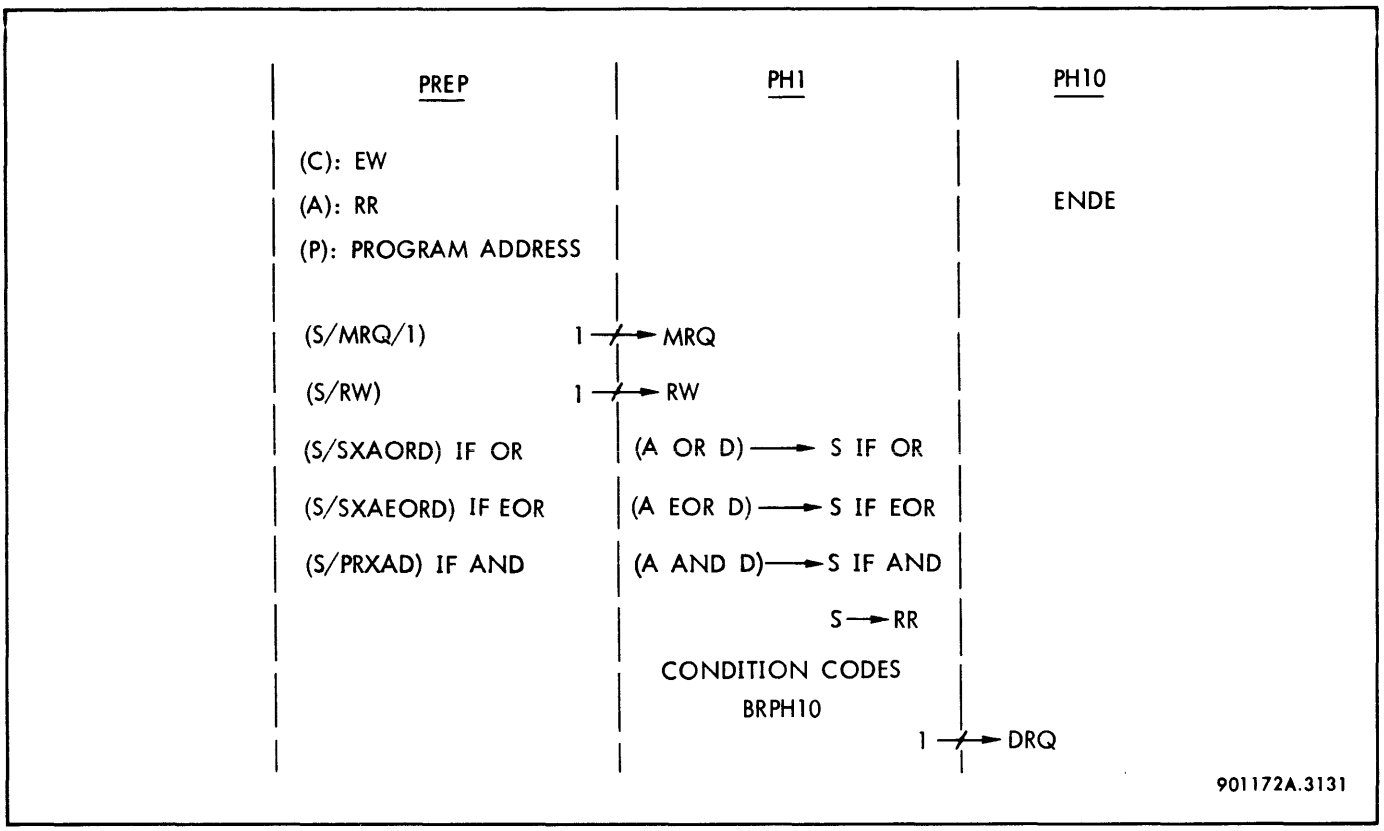


Figure 3-158. AND Instruction Phase Sequence

zero, and bits 1 through 31 contain at least one 1, the condition codes are set to XX10.

AND Word Phase Sequence. Preparation phases for the AND instruction are the same as the general PREP phases for word instructions, paragraph 3-59. Figure 3-158 shows the simplified phase sequence for the AND word instruction. Table 3-61 lists the detailed logic sequence during all AND word execution phases.

Table 3-61. OR, EOR, AND Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At the end of PREP:		
	(C) : EW		Effective word
	(A) : RR		Contents of private memory register R
	(P) : Program address		Address of next instruction in sequence

Mnemonic: AND (4B, CB) OR (49, C9) EOR (48, C8)

(Continued)

Table 3-61. OR, EOR, AND Sequence (Cont.)

Phase	Function Performed	Signals Involved			Comments
PREP (Cont.)	Set flip-flop MRQ	S/MRQ	=	(S/MRQ/1) + ...	Core memory request for next instruction in sequence
		(S/MRQ/1)	=	FAS10 PRE/34	
		FAS10	=	FAS11/1 + ...	
		FAS11/1	=	FALOGIC + ...	
		R/MRQ	=	...	
	Set flip-flop RW	S/RW	=	FAS11 (PRE/34 + PH2) NOL1	Prepare to write result in private memory
		FAS11	=	FAS11/1 + ...	
		R/RW	=	...	
	Enable signal (S/SXAORD) if OR	(S/SXAORD)	=	OU4 OL9 PRE3 + ...	Preset adder for (A OR D) ⟶ S in PH1
	Enable signal (S/SXAEORD) if EOR	(S/SXAEORD)	=	OU4 OL8 PRE3 + ...	Preset adder for (A EOR D) ⟶ S in PH1
	Enable signal (S/PRXAD) if AND	(S/PRXAD)	=	OU4 OLB PRE3 + ...	Preset adder for (A AND D) ⟶ S in PH1
PH1	One clock long				
T8L	(A0-A31) AND (D0-D31) ⟶ (S0-S31) if AND (A0-A31) OR (D0-D31) ⟶ (A0-S31) if OR (A0-A31) EOR (D0-D31) ⟶ (S0-S31) if EOR	Adder logic set at last PREP clock			Transfer result to private memory register R
	(S0-S31) ⟶ (RW0-RW31)	RWXS/0	=	RWXS/3 = RW + ...	
		RW	=	Set at last PREP clock	
	Enable clock T8	T8EN	=	NT5EN NT11L N (SXADD/1 RW) N(RW REU) N(REU AXRR)	T5EN is disabled by signal RW
		NT5EN	=	RW + ...	
	Set flip-flop CC3 if result is positive quantity and nonzero Set flip-flop CC4 if result is negative quantity	S/CC3	=	SGTZ TESTS + ...	State of flip-flops CC3 and CC4 indicates polarity of data in private memory register R after operation
		TESTS	=	FAS11 (PH1 + PH3) + ...	
		SGTZ	=	(S0 + S1 + ... + 31) S0 NFACOMP	
		R/CC3	=	TESTS + ...	
		S/CC4	=	(S/CC4/2) TESTS + ...	
		(S/CC4/2)	=	NFACOMP S0 + ...	
		R/CC4	=	TESTS + ...	

Mnemonic: AND (4B, CB) OR (49, C9) EOR (48, C8)

(Continued)

Table 3-61. OR, EOR, AND Sequence (Cont.)

Phase	Function Performed	Signals Involved			Comments
PH1 T8L (Cont.)	Branch to PH10	BRPH10	=	FAS10 PH1 + ...	
		S/PH10	=	BRPH10 NCLEAR + ...	
		R/PH10	=	...	
	Set flip-flop DRQ	S/DRQ	=	BRPH10 NCLEAR + ...	Inhibits transmission of another clock until data release from core memory
		R/DRQ	=	...	
PH10	ENDE functions				

Mnemonic: AND (4B, CB) OR (49, C9) EOR (48, C8)

3-73 Family of Shift Instructions (FASH)

SHIFT (S; 25, A5). The S instruction shifts the contents of private memory register R or the contents of private memory registers R and Ru1 (treated as a single 64-bit register) in a specified manner, amount, and direction.

Types of Shifts. The S instruction may be a logical shift, in which bits shifted off the end bit positions are lost; a circular shift, in which bits shifted off the end enter the opposite end in circular fashion; or an arithmetic shift, in which the sign of the number is extended to the right to fill the vacated bit positions as the right shift is performed. When an arithmetic shift is performed to the left, it is identical to the left logical shift. All three types of shifts may be performed on either the contents of private memory register R (single register shift) or the contents of private memory register R and private memory register Ru1 (double register shift). If a double register shift is performed, the R field of the instruction word must be an even quantity for correct results.

The type of shift to be performed is specified by three bits in the instruction word or indirectly addressed word, bit positions 21 through 23. (Performing an indexing operation does not change these bits.) A bit configuration of 00X in bit positions 21 through 23 specifies a logical shift; a configuration of 01X specifies a circular shift; 10X specifies an arithmetic shift. A one in bit position 23 denotes a single-register shift, while a zero denotes a double register shift.

Amount and Direction of Shift. The amount and direction of the shift are determined by bit positions 25 through 31 of the effective address. These bits are regarded as a signed quantity, with bit position 25 the sign bit position. If bit position 25 is a zero, bits 25 through 31 are a positive quantity, and a left shift is required. If bit position 25 is a one, bits 25 through 31 are a negative quantity (in two's complement form), and a right shift is required. The amount of the shift is determined by the absolute value of the shift count and may range from zero through 64.

Examples. Examples of the three types of shifts are shown in figure 3-159.

Condition Codes. At the completion of a logical right, circular right, or arithmetic right shift, the condition codes are set to 00XX. At the completion of a logical left, circular left, or arithmetic left shift, condition code flip-flop CC1 is set if there have been an odd number of one bits shifted off the left end of the register, or reset if there have been an even number shifted off; condition code flip-flop CC2 is set if there has been overflow into the sign bit position.

Implementation of Shift Instructions. Implementation of the various shifts is dependent primarily on the direction of the shift.

Figure 3-160 shows the basic implementation of a left shift. The shift count is transferred from the P-register to bit positions 0 through 5 of the macro-counter. If a single register shift is to be performed, the contents of private memory register R are transferred to the A-register, and zeros are transferred to the B-register. If a double register shift is to be performed, both private memory registers are transferred to the A- and B-register combination. The A- and B-registers act as a single 64-bit register during shifting operations. The A- and B-registers are shifted one bit at a time to the left, and the shift count is decremented by one with each shift. The most significant bit of the A-register is either discarded, in the case of a logical or arithmetic shift; routed to A31, in the case of a single register circular shift; or routed to B31, in the case of a double register circular shift. When the count is reduced to zero, shifting stops and the result is transferred back to the private memory registers. Flip-flop CC1 indicates whether an odd or even number of one bits have been shifted out of the A-register. Flip-flop CC2 is set if overflow has occurred.

Figure 3-161 shows the basic implementation of a right shift. The private memory registers are transferred to the A- and B-registers as before. The shift count is in the P-register in two's complement form. The shift count is transferred to the macro-counter and flip-flop FL3 as follows: bits 26 through 30 are inverted and transferred to MC1 through MC5. If P31 is a one, flip-flop FL3 is set. If the shift count is even, MC1 through MC6 now hold (shift count -2), and flip-flop FL3 is reset; if the shift count is odd, MC1 through MC6 hold (shift count -1), and FL3 is set.

If an odd shift is being performed (FL3 set), the A- and B-registers are shifted right one bit position for the first shift and right two bit positions for every other shift. The count in MC1 through MC6 is decremented by two for each shift. If an even shift is being performed, the A- and B-registers are shifted right two bit positions for each shift, and the count in MC1 through MC6 is decremented by two for each shift. The least significant bit position during the shift (A31 for a single register shift, B31 for a double register shift) is either discarded, in the case of a logical shift or arithmetic shift, or routed to the most significant end of the register, in the case of a circular shift. AO is extended to vacant bit positions for the arithmetic shift. Shifting continues until the shift count equals zero. The condition codes are set to 00XX.

Shift Phase Sequence. Preparation phases for the S instruction are the same as the general PREP phases for word instructions, described in paragraph 3-59. Table 3-62 lists the detailed logic sequence during all shift execution phases.

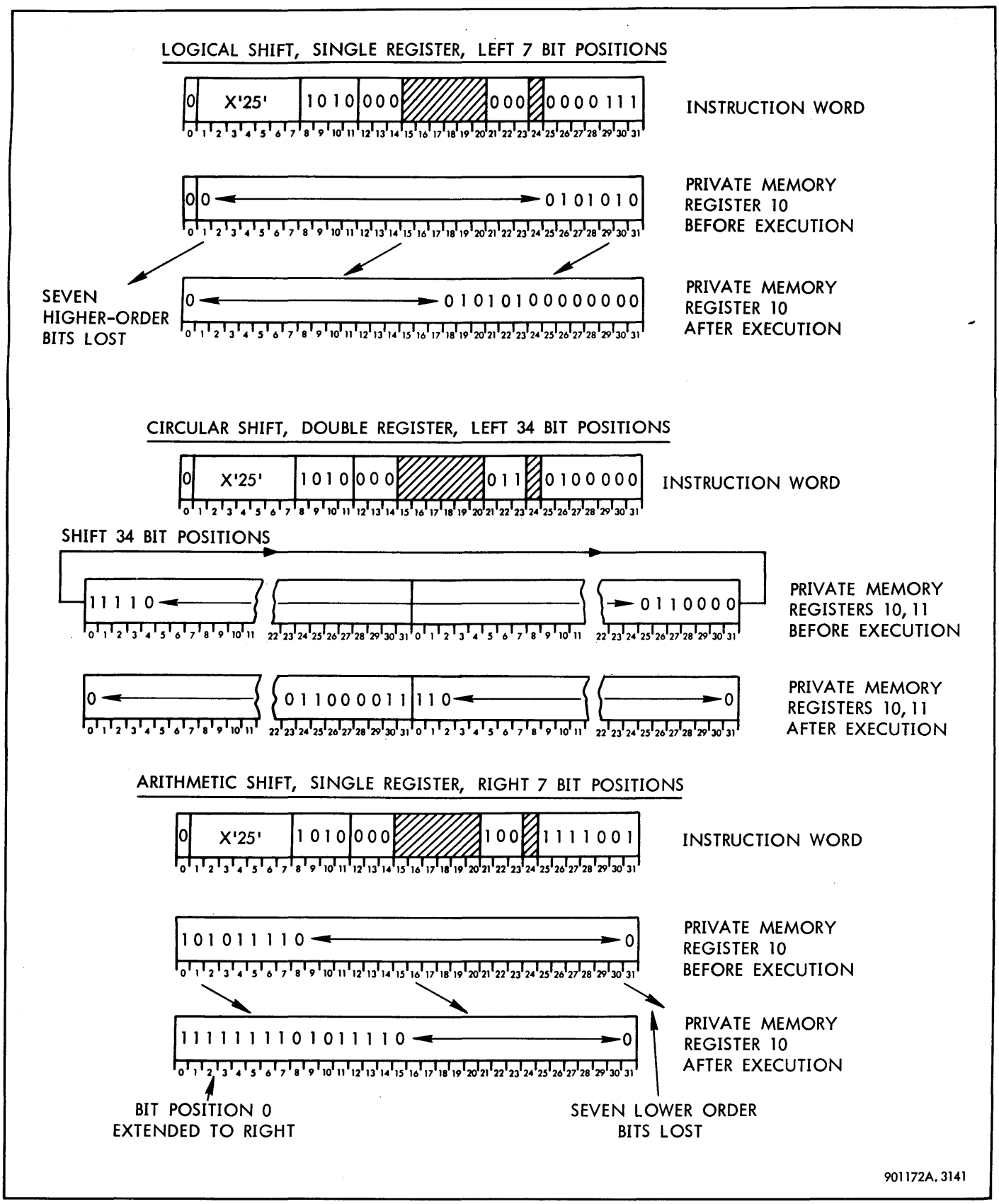


Figure 3-159. Shift Examples

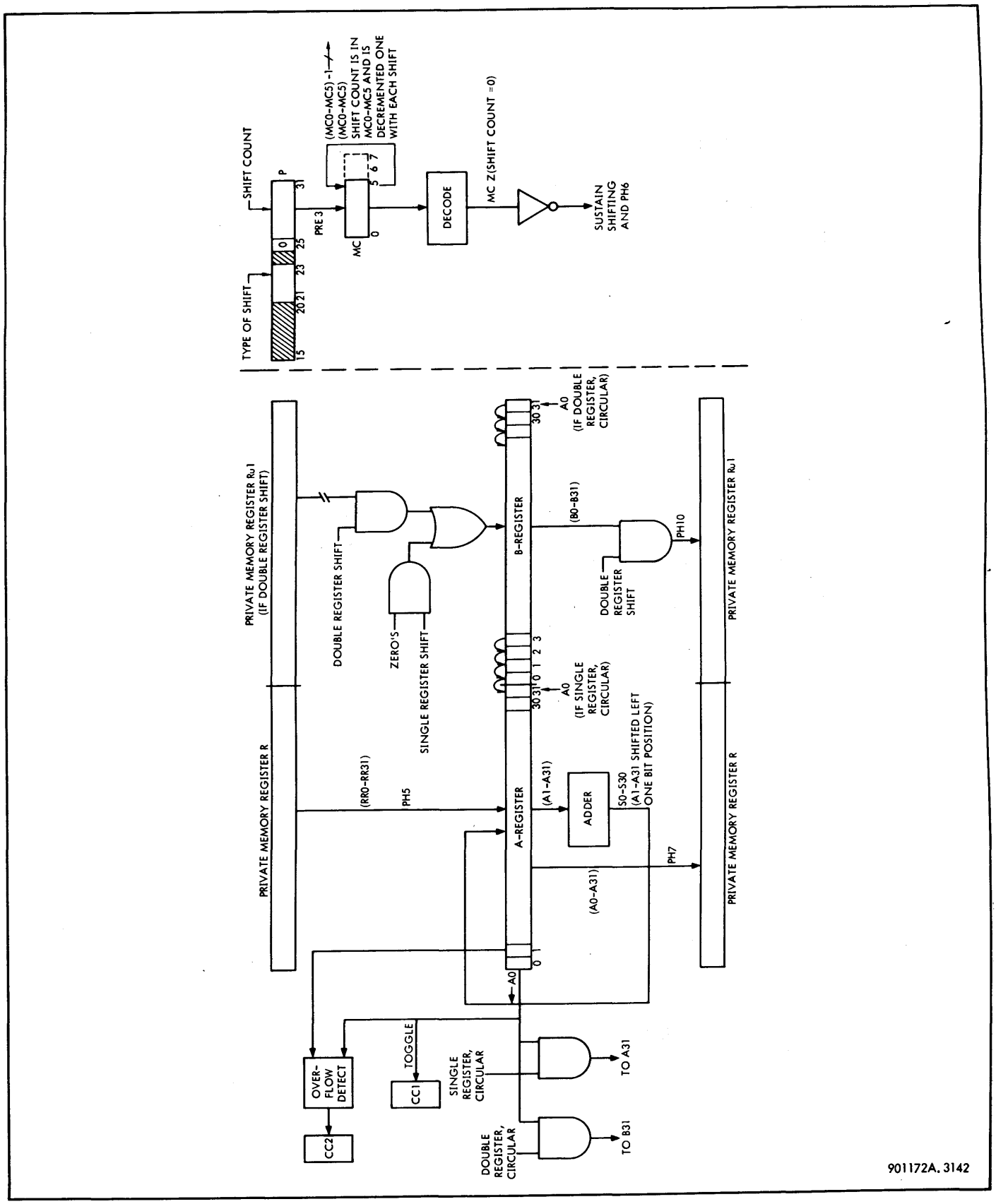
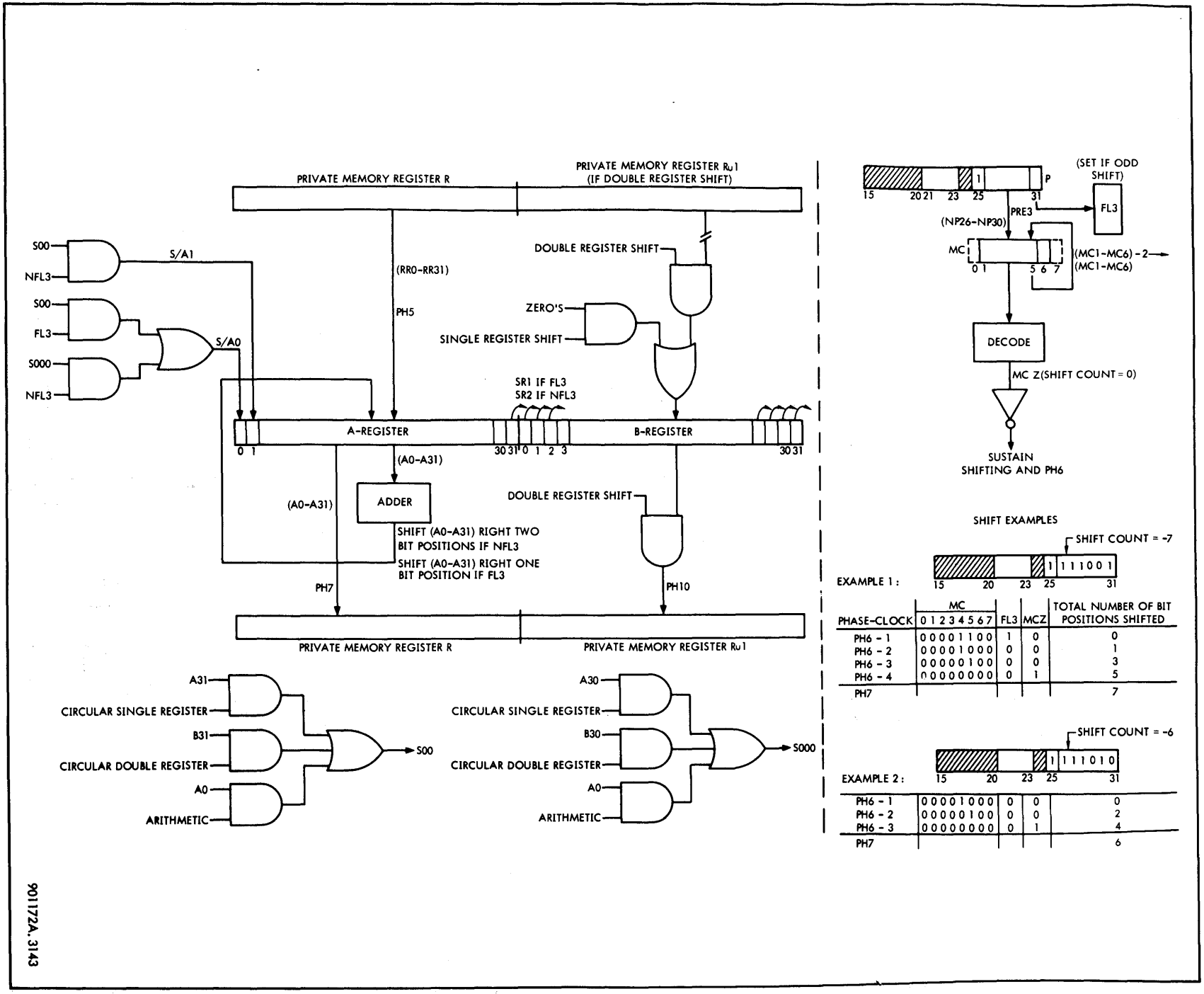


Figure 3-160. Implementation of Left Shift



PHASE-CLOCK	MC 0 1 2 3 4 5 6 7	FL3	MCZ	TOTAL NUMBER OF BIT POSITIONS SHIFTED
PH6 - 1	0 0 0 0 1 1 0 0	1	0	0
PH6 - 2	0 0 0 0 1 0 0 0	0	0	1
PH6 - 3	0 0 0 0 0 1 0 0	0	0	3
PH6 - 4	0 0 0 0 0 0 0 0	0	1	5
PH7				7

PHASE-CLOCK	MC 0 1 2 3 4 5 6 7	FL3	MCZ	TOTAL NUMBER OF BIT POSITIONS SHIFTED
PH6 - 1	0 0 0 0 1 0 0 0	0	0	0
PH6 - 2	0 0 0 0 0 1 0 0	0	0	2
PH6 - 3	0 0 0 0 0 0 0 0	0	1	4
PH7				6

Figure 3-161. Implementation of Right Shift

901172A.3143

901172

Table 3-62. . Shift Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(A) : RRu1		Contents of private memory register Ru1. If double register shift is being performed, this is least significant half of number to be shifted. If single register shift is being performed, this number will be destroyed
	(C) : Effective address (shift type and shift count)		
	(D) : Effective address (shift type and shift count)		
	(P) : Program address		P-register hold reference address during PRE3, but program address is clocked into P-register at PRE3 clock
	If right shift is specified (shift count is negative) perform the following functions:		
	Set flip-flop FL2	S/FL2 = P25 PRE3 + ... R/FL2 = CLEAR + ...	Flip-flop FL2 signifies that right shift is being performed. P25 is a one if shift count is negative
	(NP26-NP30)→(MC1-MC5) P31→FL3	MCXNPL1 = P25 PRE3 FASH FASH = OU2 OL5 + ... S/FL3 = PRE3 P31 + ... R/FL3 = N(FUS PH5) N(FUSF PH8)	If shift count is even, MC1-MC6 hold (SC-2), and FL3 is reset; if shift count is odd, MC1-MC6 hold (SC-1),and FL3 is set
	If left shift is specified perform following function:		
	(P26-P31)→(MC0-MC5)	MCXPL2 = FASH PRE3 NP25	MC0-MC5 now hold shift count
	Reset flip-flops CC1 and CC2	R/CC1 = FASH NFUMH PRE3 + ... R/CC2 = FAMDS NFUMH PRE3 + ...	Reset for PH6 use. CC1 and CC2 remain reset for right shift
	Clear B-register	BX = FAMDS PRE3 + ... FAMDS = OU2 OL5 + ...	Zeros are clocked into B-register to clear the register in the event of a single register left shift. B0 is transferred to A31 in this case, and the B-register must therefore contain zeros
			Mnemonic: S (25, A5)

(Continued)

Table 3-62. Shift Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PREP (Cont.)	Enable signal (S/SXA)	(S/SXA) = FUS PRE3 + ... FUS = PU2 OL5 + ...	Preset adder for A ⟶ S in PH5
	Reset flip-flop NAXRR	S/NAXRR = N(S/AXRR) (S/AXRR) = FUS PRE3 + ... R/NAXRR = ...	Preset for transfer of private memory register R to A-register in PH5
	Branch to PH5	BRPH5 = FUS PRE3 + ... S/PH5 = BRPH5 NCLEAR + ... R/PH5 = ...	
PH5 T5L	One clock long (A0-A31) ⟶ (S0-S31)	Adder logic set at last PREP clock	If a double register shift is being performed, the contents of private memory registers R and Rul are in the A- and B-registers, respectively. If a single register shift is being performed, the contents of private memory register R are now in the A-register, and the B-register contains all zeros.
	If double register shift is being performed, (S0-S31) ⟶/⟶ (B0-B31) (RR0-RR31) ⟶/⟶ (A0-A31)	BXS = FUS PH5 D23 + ... AXRR = set at last PREP clock	
	Sustain contents of flip-flop FL3	R/FL3 = N(FUS PH5)	Flip-flop FL3 is set if the shift count is odd. This data must be saved until PH6
	Enable signal (S/SXA)	S/SXA) = FASH PH5 + ...	Preset adder logic for A ⟶ S in PH6
	If left shift is in effect, perform the following functions: Decrement shift count in MC0-MC5 by one	MCDC7 = FASH PH5 NFL2 + ...	MC6 and MC7 are held to zero and inhibited from counting down by signal FUS. Down-counting starts with MC5. At the PH5 clock MC0-MC5 hold (SC-1)
	If shift count equals zero, branch to PH7, and request next instruction	BRPH7 = FASH PH5 NFL2 MCZ MCZ = NMC0 NMC1 ... NMC7	Shifting is not called for by instruction
			Mnemonic: S (25, A5)

(Continued)

Table 3-62. Shift Sequence (Cont.)

Phase	Function Performed	Signals Involved			Comments
PH5 T5L (Cont.)		S/PH7	=	BRPH7 NCLEAR + ...	
		R/PH7	=	...	
		S/MRQ	=	(S/MRQ/1) + ...	Core memory request for next instruction in sequence
		(S/MRQ/1)	=	FASH PH5 NFL2 MCZ + ...	
		R/MRQ	=	...	
PH6	PH6 lasts from one to 63 clocks				
T5L	If a left shift is being performed, the following functions occur during each clock period:				
	Enable signal SFT	SFT	=	FASH NIOEN PH6	Signal SFT signifies that shifting iterations are occurring
	Enable signal (S/SXA)	(S/SXA)	=	SFT + FASH (S/PH6/IO) + ...	Preset adder for A ⟶ S during next clock period. Signal (S/PH6/IO) is a return to shift iterations from I/O service
		(S/PH6/IO)	=	IOPH1 SW13 NIPH10. NPCP2 (NIOEN + IOB0)	
	(A0-A31) ⟶ (S0-S31)	Adder logic set at previous clock by (S/SXA)			
	(S1-S31) ⟶ (A0-A30)	AXSL1	=	SFT NFL2 NFSHEX + ...	Output of adder shifted left one bit position
	If a single register circular shift is being performed, A0 ⟶ A31; otherwise, B0 ⟶ A31	S/A31	=	AXSL1 A31EN/1 + ...	B-register contains least significant half of number to be shifted if double register shift or zeros if single register shift
		A31EN/1	=	AO FUS D22 ND23 PH6 + BO FAMDS PH6 + ...	
		R/A31	=	AXSL1 + ...	
	(B1-B31) ⟶ (B0-B30)	BXBL1	=	SFT NFL2 NFSHEX + ...	B-register shifted left one bit position
	If a double register circular shift is being performed A0 ⟶ B31	S/B31	=	BXBL1 B31EN/1 + ...	Most significant bit of A-register brought around in circular fashion
		B31EN/1	=	AO FUS D22 D23 + ...	
		FUS	=	OU2 OL5	
		R/B31	=	BXBL1 + ...	
	Decrement the shift count in MC0-MC5 by one	MCDC7	=	FASH PH5 NFL2 + ...	MC6 and MC7 are held to zero and inhibited from counting down by signal FUS. Down-counting starts with MC5
					Mnemonic: S (25, A5)

(Continued)

Table 3-62. Shift Sequence (Cont.)

Phase	Function Performed	Signals Involved			Comments
PH6 T5L (Cont.)	Toggle flip-flop CC1 as one bits shift through A0	S/CC1	=	A0 NCC1 FUS/1 + ...	CC1 provides an indication of whether an odd or even number of one bits have been shifted out of the A-register
		FUS/1	=	FUS SFT NFL2	
		R/CC1	=	A0 FUS/1 + ...	
	Set flip-flop CC2 if overflow into sign bit position	S/CC2	=	FUS/1 (A0 + A1) + ...	
		R/CC2	=	FAMDS NFUMH PRE3 + ...	
	If an even right shift is being performed, following functions occur during each clock period:				
	Enable signal SFT	SFT	=	FASH NIOEN PH6	Shifting iterations
	Enable signal (S/SXA)	(S/SXA)	=	SFT + FASH (S/PH6/IO) + ...	Preset adder for A ⟶ S during next clock period
	(A0-A31) ⟶ (S0-S31)	Adder logic set at previous clock by (S/SXA)			
	(S0-S29) ⟶ (A2-A31)	AXSR2	=	SFT FL2 NFL3 NFSHEX + ...	Output of adder shifted right two bit positions
	Set A0 and A1 according to type and length of shift	S/A0	=	S000 AXSR2 + ...	
		R/A0	=	AXSR2 + ...	
		S/A1	=	S00 AXSR2 + ...	
		R/A1	=	AXSR2 + ...	
		S000	=	A30 FUS D22 ND23 + ...	Single register, circular shift A30, A31 ⟶/⟶ A0, A1, respectively
		S00	=	A31 FUS D22 ND23 + ...	
		S000	=	B30 FUS D22 D23 + ...	Double register circular shift B30, B31 ⟶/⟶ A0, A1, respectively
		S00	=	B31 FUS D22 D23 + ...	
		S000	=	A00 + ...	Arithmetic shift. original contents of A0 extended to right as shift is made
		A00	=	A0 D21 + ...	
		S00	=	A00 + ...	
	If a double register shift is being performed, S30 and S31 are clocked into B0 and B1, respectively	S/B0	=	BXBR2 S30/1 + ...	
		BXBR2	=	FL2 NFL3 NFSHEX SFT + ...	
		S/30/1	=	B0001EN/1 FL1 + ...	
		B0001EN/1	=	FASH D23 + ...	
					Mnemonic: S (25, A5)

(Continued)

Table 3-62. Shift Sequence (Cont.)

Phase	Function Performed	Signals Involved			Comments
PH6 T5L (Cont.)		R/B0	=	BXBR2 + ...	
		S/B1	=	S31/1 BXBR2 + ...	
		S31/1	=	B0001EN/1 S31 + ...	
		R/B1	=	BXBR2 + ...	
	(B0-B29)—/→(B2-B31)	BXBR2	=	FL2 NFL3 NFSHEX SFT+...	
	Decrement shift count in MC1-MC6 by two	MCDC7	=	SFT BRP + ...	MC6 and MC7 are held to zero and inhibited from counting down by signal FUS. Down-counting starts with MC5, thereby decrementing shift count by two
	If an odd right shift is being performed, the following functions occur during the first clock period:				
	Enable signal SFT	SFT	=	FASH NIOEN PH6	Shifting iterations
	Enable signal (S/SXA)	(S/SXA)	=	SFT + FASH (S/PH6/I0) + ...	Preset adder for A→S during next clock period
	(A0-A31)→(S0-S31)	Adder logic set at previous clock by (S/SXA)			
	(S0-S30)—/→(A1-A31)	AXSR1	=	SFT FL2 FL3 + ...	Output of adder shifted right one bit position
	Set A0 according to type and length of shift	S/A0	=	S00 AXSR1 + ...	
		S00	=	A31 FUS D22 ND23 + ...	Single register circular shift. A31—/→A0
		S00	=	B31 FUS D22 D23 + ...	Double register circular shift. B31—/→A0
		S00	=	A00 + ...	Arithmetic shift. A0 extended to right
		A00	=	A0 D21 + ...	
		R/A0	=	AXSR1 + ...	
	If a double register shift is being performed, S31 is clocked into B0	S/B0	=	S31/1 BXBR1 + ...	
		S31/1	=	B0001EN/1 S31 + ...	
		B0001EN/1	=	FASH D23 + ...	
		BXBR1	=	FL2 FL3 SFT	
		R/B0	=	BXBR1 + ...	
	(B0-B30)—/→(B1-B31)	BXBR1	=	FL2 FL3 SFT	
	Decrement the shift count in MC1-MC6 by two	MCDC7	=	SFT BRP + ...	MC6 and MC7 are held to zero and inhibited from counting down by signal FUS. Down-counting starts with MC5, thereby decrementing shift count by two
					Mnemonic: S (25, A5)

(Continued)

Table 3-62. Shift Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH6 T5L (Cont.)	Reset flip-flop FL3	R/FL3 = N(FUS PH5 + FUSF PH8)	Resetting FL3 indicates that the one odd shift has been performed. Shifting henceforth will be by two's
	During each succeeding clock period, the functions described under the even right shift are performed, with the shift count decremented by two and the number shifted right two bit positions for each clock period		
	Sustain PH6 if the shift count does not equal zero, and an I/O service call is not pending	BRPH6 = FAMDS NFSHEX PH6 NBRPH10 NMCZ +...	
		MCZ = (MC0 MC1 ... MC7)	Signal MCZ is true when shift count reaches zero
	If an I/O service call is in effect and the shift count is four or above (at least two more shifts to be performed), inhibit setting PH6, complete the I/O operation, and set PH6 to complete the shifting	S/PH6 = BRPH6 NIOEN NCLEAR +...	
		IOEN = IOSC IOEN6 NIOINH+ ...	I/O enable
		S/IOSC = SC NSCINH IOPOP	Set at previous clock
		IOEN6 = FAMDS NFPRR NFSHEX IOEN6/1 PH6 NDIOEXIT	I/O enable, phase 6
		NMC0005Z = NMC0005Z NFADIV CC2 NEWDM	
		= MC0 + MC1 + ... + MC5	
		NIOINH = N(ADNH PH6 + AB0 PH6 + INTRAP NPCP2)	Not I/O inhibit
		R/PH6 = ...	
	If the shift count is zero, perform the following functions:		
	Disable signal BRPH6	NBRPH6 = MCZ +...	
	Branch to PH7	S/PH7 = PH6 NBR NIOEN +...	
		NBR = NBRPH6 ...	
		R/PH7 = ...	
	Set flip-flop MRQ	S/MRQ = (S/MRQ/1) + ...	Memory request for next instruction in sequence
		(S/MRQ/1) = FAMDS PH6 NBRPH6 NIOEN +...	
		R/MRQ = ...	
	Set flip-flop RW	S/RW = SFT FUS MCZ +...	Set to transfer the result of the shift to private memory register R
		R/RW = ...	
			Mnemonic: S (25, A5)

(Continued)

Table 3-62. Shift Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH7	One clock long		
T8L	(A0-A31)⟶(S0-S31)⟶ (RW0-RW31)	Adder logic set at last PH6 clock RWXS/0-RWXS/3 = RW + ... RW = set at last PH6 clock	Transfer most significant word of result to private memory register R
	Reset flip-flop NLR31F	S/NLR31F = N(S/LR31) (S/LR31) = FUS PH7 + ... R/NLR31F = ...	Force a one on private memory address line LR31 during PH10 to select private memory register Ru1
	Reset flip-flop NSXBF	S/NSXBF = N(S/SXB) (S/SXB) = FASH PH7 + ... R/NSXBF = ...	Preset logic for B⟶S in PH10
	If double register shift is being performed, set flip-flop RW	S/RW = FUS PH7 D23 NR31 + ... R/RW = ...	If the R field of the instruction word is odd, R31 inhibits storing the least significant word of the result in private memory register R. The R-register contains the most significant word of the result at the end of the instruction in this case
	Set flip-flop DRQ	S/DRQ = BRPH10 NCLEAR + ... R/DRQ = ...	Inhibits transmission of another clock until data release signal is received from core memory
	Branch to PH10	BRPH10 = FUS PH7 + ... S/PH10 = BRPH10 NCLEAR + ... R/PH10 = ...	
	Enable clock T8L	T8EN = NT5EN NT11L N(SXADD/1 RW) N(RW REU) N(REU AXRR) NT5EN = RW + ...	T5 is disabled by signal RW
PH10 DR	(B0-B31)⟶(S0-S31)⟶ (RW0-RW31)	SXB = NDIS SXBF + ... SXBF = Set at PH7 clock RWXS/0-RWXS/3 = RW + ... RW = Set at PH7 clock	Transfer least significant word of result to private memory register Ru1. This transfer is not made if the R field of the instruction word was odd or if single register shift is in effect
	ENDE functions		
			Mnemonic: S (25, A5)

SHIFT FLOATING (SF; 24, A4). The SF instruction performs a right or left shift operation on a short-format floating point number in private memory register R, or on a long-format floating point number in private memory registers R and Ru1. The shifted result is loaded back into private memory. Both formats of floating point numbers are described in detail in paragraph 3-74.

Right Shift Operations. For a right shift of either a long- or short-format floating point number the fraction of the floating point number is shifted one hexadecimal place to the right and the exponent of the number incremented by one. Shifting continues until the number of shifts specified by the instruction have been performed or until the exponent field of the number overflows. The shifted result is loaded back into private memory registers R and Ru1; the exponent and fraction of the result is set to all zeros ("true" zero) if the fraction of the floating point number was zero or became zero.

The condition codes are set to 00XX if the number of hexadecimal shifts specified by the instruction have been performed. If exponent overflow has occurred, the condition codes are set to 01XX. Flip-flops CC3 and CC4 are set to 00 if the result is zero, 01 if the result is negative, and 10 if the result is positive.

Left Shift Operations. For a left shift of either a long- or short-format floating point number the fraction of the floating point number is shifted one hexadecimal place to the left, and the exponent of the number is decremented by one. Shifting continues until the number of shifts specified by the instruction have been performed, until the number is normalized (significance in the most significant hexadecimal digit of the fraction), or until the exponent field underflows. The shifted result is then loaded back into private memory. The result is set equal to true zero if the fraction of the floating point number was zero.

The condition codes are set to 00XX if the number of hexadecimal shifts specified by the instruction have been completed. If the fraction is normalized, the condition code settings are 1XXX. Exponent underflow produces settings of X1XX. Exponent underflow and normalization can appear simultaneously. Flip-flops CC3 and CC4 are set to 00 if the result is zero, 01 if the result is negative, or 10 if the result is positive.

Short and Long Formats. If bit position 23 in the instruction word or indirectly addressed word is a zero, the contents of private memory register R are treated as a short-format floating point number. If bit position 23 is a one, the contents of private memory registers R and Ru1 are treated as a long-format floating point number.

Amount and Direction of Shift. The amount and direction of shift are determined by bit positions 25 through 31 of the effective address. These bits are regarded as a signed quantity, with bit position 25 the sign position. If bit position 25 is a zero, bits 25 through 31 are a positive quantity, and a left shift is required. If bit position 25 is a one, bits 25 through 31 are a negative quantity (in two's complement form), and a right shift is required. The amount of shift is determined by the absolute value of the shift count. The shift count specifies the number of hexadecimal shifts of the fraction to be performed.

Examples. Examples of a short-format right shift and a short-format left shift are shown in figure 3-162.

Implementation of the Shift Floating Instruction. Figure 3-163 shows the basic implementation of a left shift. The shift count is transferred from the P-register to bit positions 0 through 5 of the macro-counter. MC0 through MC7 now hold four times the shift count. If a short format shift is to be performed, the absolute value of the short-format floating point number in the R-register is transferred to the A-register and zeros are transferred to the B-register. If a long-format floating point shift is to be performed, the absolute value of both the R and Ru1 registers is transferred to the A- and B-registers. The A- and B-registers are treated as a single register during shifting operations.

The fraction of the floating point number is contained in A8 through A31 for a short-format shift, or A8 through B31 for a long-format shift. The exponent of the number is in A1 through A7; these seven bits are isolated from the remainder of the A- and B-registers. A0 contains a zero. The fraction in the A- and B-registers is shifted left one bit at a time, and the macro-counter is decremented one count with each shift (effectively decrementing the shift count by one-quarter with each shift; the total number of shifts performed must be a multiple of four). The exponent field in A1 through A7 is decremented by one count for every four shifts, since the fraction has been increased by a factor of 16 and the exponent must be reduced to restore the original magnitude of the floating point number.

When the shift count has been reduced to zero, when the fraction of the floating point number has been normalized (a one bit in A8 through A11), or when the exponent field underflows (ones in A0 through A7), shifting stops. The shifted result is corrected to its original sign and transferred back into private memory. If the fraction of the floating point number was originally zero, the floating point number is changed to true zero before the transfer is made.

Figure 3-164 shows the basic implementation of a right shift. The shift count is in the P-register in two's complement form. The shift count is transferred to the macro-counter as follows: bits 26 through 30 are inverted and transferred to MC1 through MC6. A one is forced into MC7. The macro-counter now holds twice the shift count minus one. The floating point number is transferred to the A- and B-registers as in the left shift. The fraction in the A- and B-registers is shifted right two bits at a time with each shift (effectively decrementing the shift count by one-half with each shift; the total number of shifts performed must be a multiple of two). The exponent is incremented by one count with every two shifts to compensate for shifting the fraction. The

fraction of the number is reduced by a factor of 16 with each two shifts effected. The two least significant bits of the fraction are lost with each shift.

When the shift count is reduced to zero, when the exponent field overflows (a one in A0), or when the fraction of the number goes to zero, shifting stops. The shifted result is corrected to its original sign and transferred back into private memory. If the fraction of the floating point number was originally zero or became zero, the floating point number is changed to true zero before the transfer is made.

Shift Floating Phase Sequences. Preparation phases for the SF instruction are the same as the general PREP phases for word instructions, paragraph 3-59. Table 3-63 lists the detailed logic sequence during all SF execution phases.

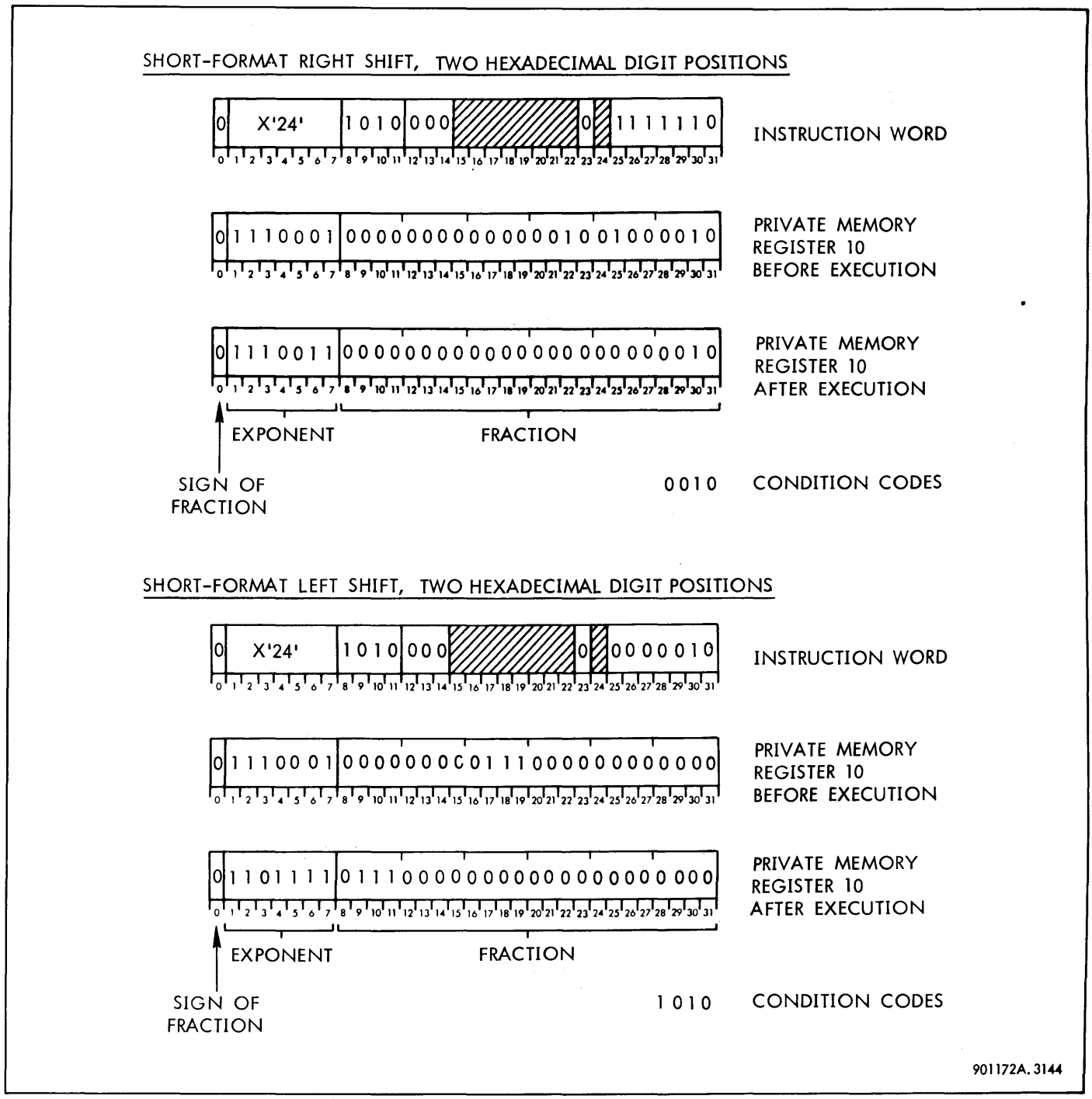


Figure 3-162. Shift Floating Examples

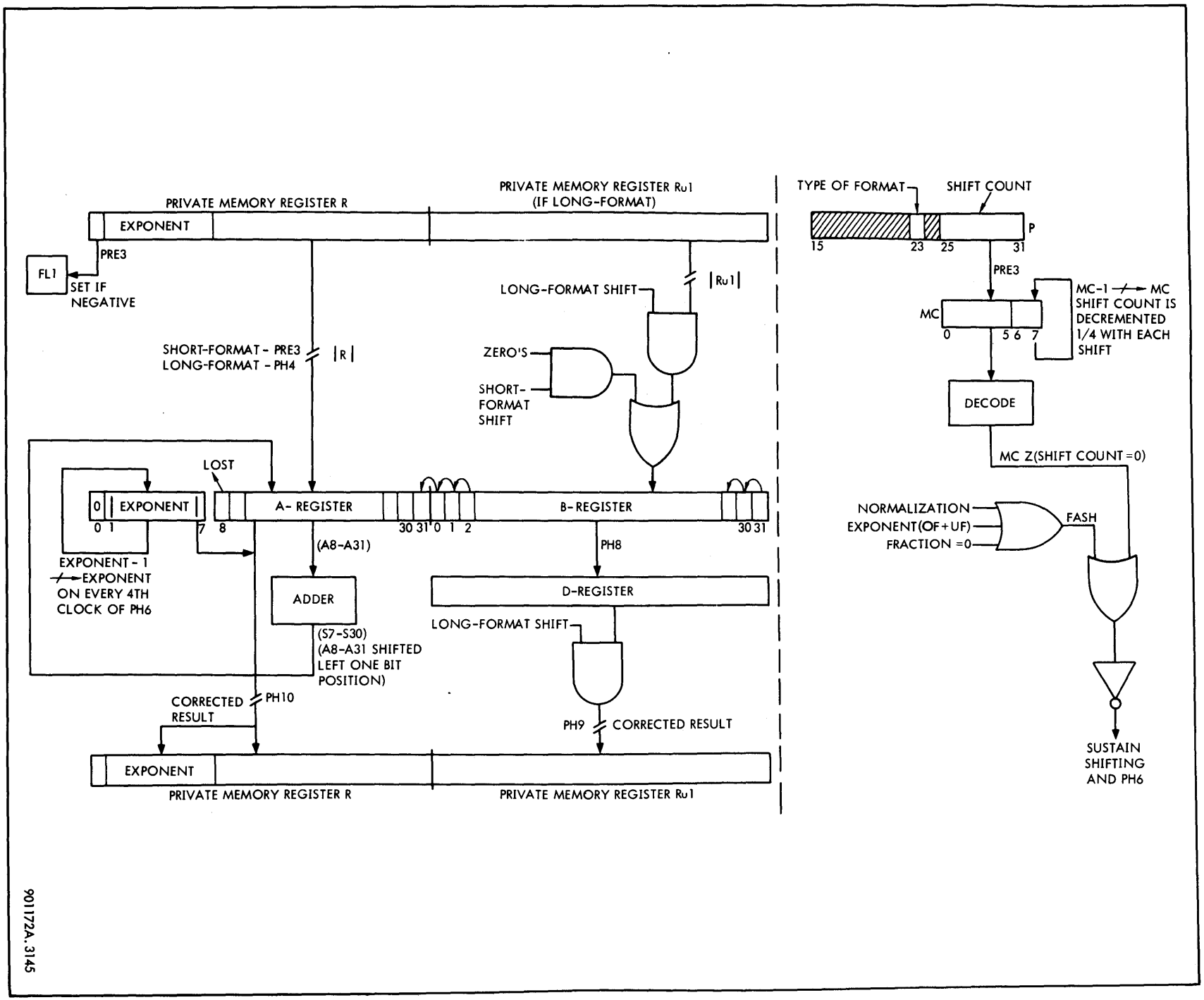


Figure 3-163. Implementation of Left Shift Floating

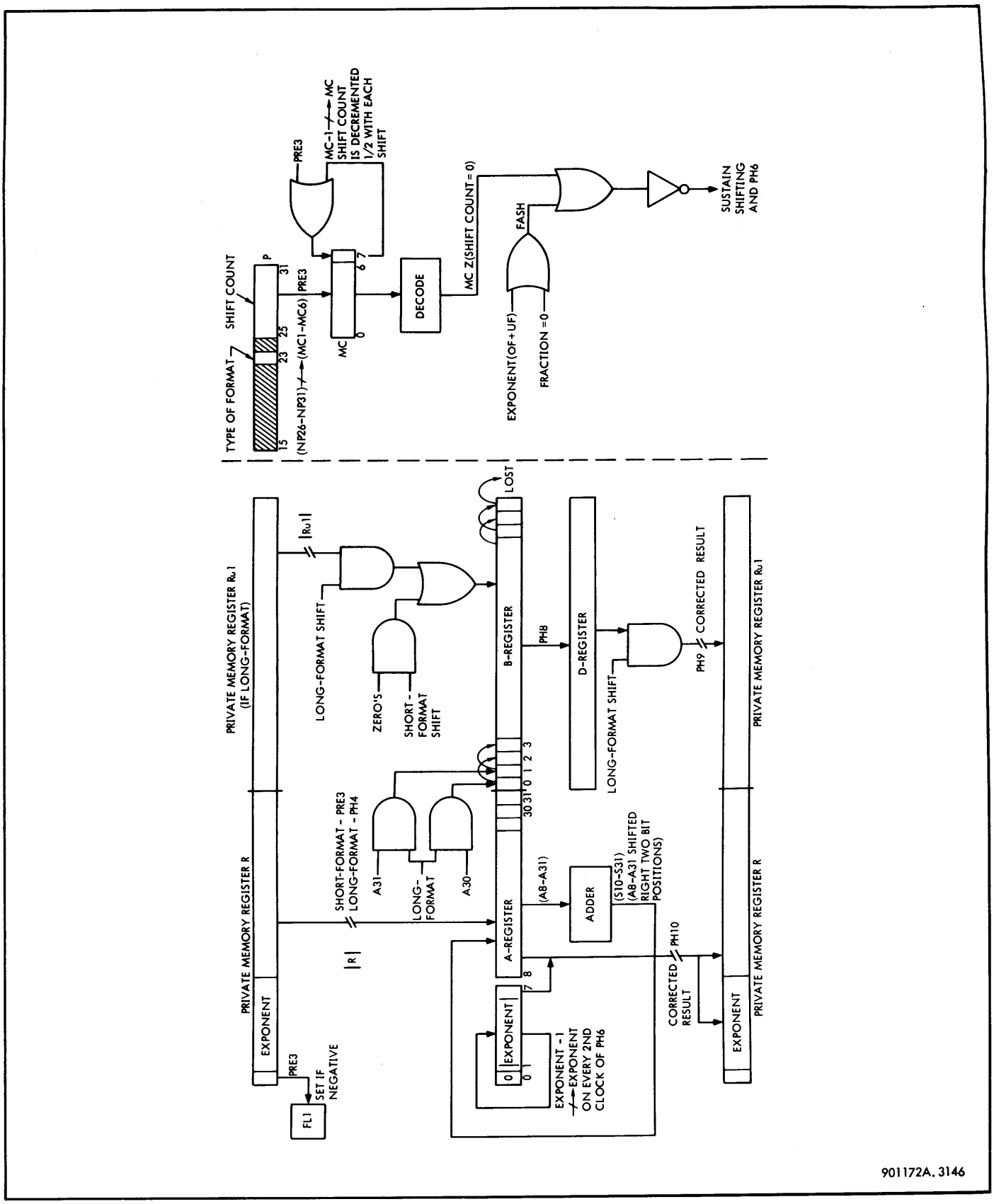


Figure 3-164. Implementation of Right Shift Floating

Table 3-63. Shift Floating Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(A) : RR		Contents of private memory register R. If long-format shift is being performed, this number will be destroyed in PH3. If short-format shift, this is entire floating point number
	(C) : Effective address (shift type and shift count)		
	(D) : Effective address (shift type and shift count)		
	(P) : Program address		P-register holds reference address during PRE3, but program address is clocked into P-register at PRE3 clock
	If right shift is specified (shift count is negative), perform the following functions:		
	Set flip-flop FL2	S/FL2 = P25 PRE3 + ...	Flip-flop FL2 specifies that right shift is being performed. P25 is a one if shift count is negative
		R/FL2 = CLEAR + ...	
	(NP26-NP31)→(MC1-MC6)	MCXNPL1 = P25 PRE3 FASH	MC1-MC7 now contain (2 shift count - 1). Shifting in PH6 will be two bit positions at a time
	One→MC7	S/MC7 = FUSF MCXNPL1 + ...	
		R/MC7 = MCDC7 NFUS + ...	
	If left shift is specified, perform the following functions:		
	(P26-P31)→(MC0-MC5)	MCXPL2 = FASH PRE3 NP25	MC0-MC7 now contain (4 shift count)
	Set flip-flop FL1 if floating point number to be shifted is negative	S/FL1 = PRE3 RR0 + ...	Store sign of floating point number to be shifted
		R/FL1 = CLEAR + ...	
	Clear B-register	BX = FAMDS PRE3 + ...	Zeros are clocked into B-register to clear the register in the event of a short-format left shift. B0 is transferred to A31 in this case, and the B-register must therefore contain zeros
	Reset flip-flops CC1 and CC2	R/CC1 = FASH NFUMH PRE3 + ...	Reset for PH6 use
		R/CC2 = FAMDS NFUMH PRE3 + ...	
			Mnemonic: SF (24, A4)

(Continued)

Table 3-63. Shift Floating Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PREP (Cont.)	If long-format floating point shift is being performed, perform the following functions:		
	Reset flip-flop NAXRR	S/NAXRR = N(S/AXRR) (S/AXRR) = PRE3 FUSF ND23 + ... R/NAXRR = ...	Prepare to read least significant half of floating point number
	Reset flip-flop NLR31F	S/NLR31F = N(S/LR31) (S/LR31) = PRE3 FUSF ND23 + ... R/NLR31F = ...	Force a one on private memory address line LR31 during next phase
	Branch to PH3	BRPH3 = FAMDS PRE/34 NBRPH5 NANLZ + ... S/PH3 = BRPH3 NCLEAR + ... R/PH3 = ...	
	If short-format floating point shift is being performed, perform the following functions:		
	Enable signal (S/SXMA)	(S/SXMA) = FUSF PRE3 ND23 + ...	Preset adder for -A⟶S during next phase. Used if floating point number is negative
	Branch to PH5	BRPH5 = FUSF PRE3 ND23 + ... S/PH5 = BRPH5 NCLEAR + ... R/PH5 = ...	PH3 and PH4 operations involve obtaining the most significant word of the long-format floating point number
PH3 T5L	One clock long. Entered only for long-format shift (RR0-RR31)⟶(A0-A31)	AXRR = Set at last PREP clock	Transfer least significant word of long-format number to A-register
	If long-format floating point number is even, enable signal (S/SXA); if odd, enable signal (S/SXMA)	(S/SXA) = NFL1 (S/SX/FL1) + ... (S/SX/FL1) = FUSF D23 PH3 + ... (S/SXMA) = FL1 (S/SX/FL1) + ...	Preset adder to gate absolute value of least significant word of floating point number to sum bus
	Reset flip-flop NAXRR	S/NAXRR = N(S/AXRR) + ... (S/AXRR) = FUSF D23 PH3 + ... R/NAXRR = ...	Prepare to read most significant half of floating point number from private memory register R
			Mnemonic: SF (24, A4)

(Continued)

Table 3-63. Shift Floating Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH4 T5L	One clock long. Entered only for long-format shift		
	If floating point number is positive, (A0-A31)⟶(S0-S31)	Adder logic set at PH3 clock	Transfer absolute value of least significant half of floating point number to sum bus
	If floating point number is negative, -(A0-A31)⟶(S0-S31)	Adder logic set at PH3 clock	Transfer absolute value of least significant half of floating point number to sum bus
	(S0-S31)⟶(B0-B31)	BXS = FUSF D23 PH4 + ...	Transfer absolute value of least significant half of floating point number to B-register
	If long-format floating point number is even, enable signal (S/SXA); if odd, enable signal (S/SXMA), and reset flip-flop NK31 if end carry	(S/SXA) = NFL1 (S/SX/FL1) + ... (S/SX/FL1) = FUSF D23 PH4 + ... (S/SXMA) = FL1 (S/SX/FL1) + ...	Preset adder to gate absolute value of most significant word of floating point number to sum bus
		S/NK31 = N(S/K31) N(S/SXAMD/1) + N(S/K31/1) (S/K31) = (S/SXMA) + ... (S/K31/1) = K00 (S/K31/3) + ... (S/K31/3) = High during SF R/NK31 = ...	Setting K31 effectively provides a carry to most significant word of the floating point number as it is complemented in PH5. If no end carry exists, K31 will be reset for PH5
	(RR0-RR31)⟶(A0-A31)	AXRR = Set at PH3 clock	Transfer most significant half of floating point number to A-register
PH5 T5L	One clock long If short-format shift is being implemented, perform the following functions:		
	-(A0-A31)⟶(S0-S31)	Adder logic set at last PREP clock	If the short-format number is positive, the negated result will not be transferred back to A-register. If the short-format number is negative, the negated result replaces original number. A-register now contains absolute value of original floating point number
	If floating point number to be shifted is negative, (S0-S31)⟶(A0-A31)	AXS = FUSF PH5 FL1 + ...	
			Mnemonic: SF (24, A4)

(Continued)

Table 3-63. Shift Floating Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH5 T5L (Cont.)	If long-format shift is being implemented, perform the following functions:		
	If long-format floating point number is positive, (A0-A31) ⟶ (S0-S31)	Adder logic set at PH4 clock	Insignificant action. Original floating point number in A and B remains unchanged
	If long-format floating point number is negative, (NA0-NA31) + K31 ⟶ (S0-S31) ⟶ (A0-A31)	Adder logic set at PH4 clock AXS = FUSF PH5 FL1 + ...	Absolute value of original long-format floating point number is now in A- and B-registers
	Enable signal (S/SXA)	(S/SXA) = FASH PH5 + ...	Preset adder for A⟶S in PH6
	If left shift is being implemented, decrement count in macro-counter by one	MCDC7 = FASH PH5 NFL2 + ...	Precount shift count towards zero. MC6 and MC7 hold 11 at end of PH5 if left shift
	If shift count equals zero, branch to PH7 and request next instruction	BRPH7 = FASH PH5 NFL2 MCZ S/PH7 = BRPH7 NCLEAR + ... R/PH7 = ...	Shifting is not called for by instruction
		S/MRQ = (S/MRQ/1) + ... (S/MRQ/1) = FASH PH5 NFL2 MCZ + ... R/MRQ = ...	Core memory request for next instruction in sequence
PH6 T5L	Length of PH6 determined by shift count, exponent value, normalization, etc. If left shift is being performed, the following functions occur during each clock period:		
	Enable signals FUSF/1 and SFT	FUSF/1 = FUSF NIOEN PH6 + ... SFT = FASH NIOEN PH6	Signal SFT signifies that shifting iterations are occurring
	Enable signal (S/SXA)	(S/SXA) = SFT + FASH (S/PH6/IO) + ... S/PH6/IO) = IOPH1 SW13 NIPH10 NPCP2 (NIOEN + IOB0)	Preset adder for A⟶S during next clock period. Signal (S/PH6/IO) is a return to shift iterations from I/O service
	Zeros ⟶ (S0-S7)	NPRXAD/0 = FUSF/1 NDIS + ... NPRXAND/0 = FUSF/1 NDIS + ...	NPRXAD and NPRXAND effectively disable adder logic so that S0-S7 are always zeros
			Mnemonic: SF (24, A4)

(Continued)

Table 3-63. Shift Floating Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH6	(A8-A31)⟶(S8-S31)	Adder logic set at previous clock by (S/SXA)	
T5L (Cont.)	(S8-S31)⟶(A7-A30)	AXSL1 = NFSHEX NFL2 SFT + ...	Output of adder shifted left one bit position. S8 will always be zero because shifting will stop with normalization of number; A7 is consequently never set as a result of shift
	B0⟶A31	S/A31 = AXSL1 A31EN/1 + ... A31EN/1 = B0 FAMDS PH6 + ... R/A31 = AXSL1 + ...	B-register contains the least significant word of long-format floating point number, or zeros if a short-format shift is in effect
	(B1-B31)⟶(B0-B30) Zero⟶B31	BXBL1 = SFT NFL2 NFSHEX + ...	B-register shifted left one bit position
	Decrement exponent by one on fourth clock of PH6 and every fourth clock thereafter	ADC7 = FUSF/1 NFL2 NMC6 NMC7	The exponent is decremented after every four shifts to compensate for shifting (the size of the number has effectively been increased by a factor of 16 due to shifting)
	Decrement shift count in MC0-MC7 by one	MCDC7 = SFT BRP + ...	The original shift count (specifying the number of hexadecimal digit positions to be shifted) is decremented by one-quarter at each clock
	If a right shift is being performed, the following functions occur during each clock period:		
	Enable signals FUSF/1 and SFT	FUSF/1 = FUSF NIOEN PH6 + ... SFT = FASH NIOEN PH6	Signal SFT signifies that shifting iterations are occurring
	Enable signal (S/SXA)	(S/SXA) = SFT + FASH (S/PH6/IO) + ... (S/PH6/IO) = IOPH1 SW13 NIPH10 NPCP2 (NIOEN + IOB0)	Preset adder for A⟶S during next clock period. Signal (S/PH6/IO) is a return to shift iterations from I/O service
			Mnemonic: SF (24, A4)

(Continued)

Table 3-63. . Shift Floating Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH6 T5L (Cont.)	Zeros ⟶ (S0-S7)	NPRXAD/0 = FUSF/1 NDIS + ... NPRXAND/0 = FUSF/1 NDIS + ...	NPRXAD and NPRXAND effectively disable adder logic so that S0-S7 are always zeros
	(A8-A31) ⟶ (S8-S31)	Adder logic set at previous clock by (S/SXA)	
	(S8-S31) ⟶/⟶ (A10-A31)	AXSR2 = NFSHEX FL2 NFL3 SFT + ...	Output of adder shifted right two bit positions. A8 and A9 will always be reset, since zeros are present on S6 and S7
	If a long-format shift is being performed, S30, S31 ⟶/⟶ B0, B1 Otherwise, zeros ⟶/⟶ B0, B1	S/B0 = S31/1 BXBR2 + ... S30/1 = S30 B0001EN/1 + ... B0001EN/1 = FASH D23 + ... R/B0 = BXBR2 + ... S/B1 = S31/1 BXBR2 + ... S31/1 = S31 B0001EN/1 + ... R/B1 = BXBR2 + ...	
	(B0-B29) ⟶/⟶ (B2-B31)	BXBR2 = FL2 NFL3 NFSHEX SFT + ...	NFL3 applies to fixed point shift only
	Increment exponent by one on second clock of PH6 and every second clock thereafter	AUC7 = NCC2 FL2 FUSF/1 NMC7	Exponent incremented after every two shifts to compensate for shifting (size of number has been effectively decreased by a factor of 16 due to shifting)
	Decrement shift count in MC1-MC7 by one	MCDC7 = SFT BRP + ...	The original shift count (specifying number of hexadecimal digit positions to be shifted) is decremented by one-half at each clock
	Sustain PH6 until shift count reaches zero or until normalization occurs on left shift, providing none of following conditions exist: a. Exponent overflow or underflow b. Fraction equal to zero c. I/O service call pending	BRPH6 = FAMDS PH6 NMCZ NFSHEX NBRPH10 MCZ = NMC0 NMC1 ... NMC7 FSHEX = FUSF (FNORM + A0 + FL3) S/PH6 = BRPH6 NIOEN NCLEAR + ... R/PH6 = ...	
			Mnemonic: SF (24, A4)

(Continued)

Table 3-63. Shift Floating Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH6 T5L (Cont.)	If an I/O service call is in effect and more shifts are to be performed (macro-counter holds a count of four or above, and signal FSHEX is false), inhibit setting PH6, complete I/O operation, and then set PH6 to complete shifting	S/PH6 = BRPH6 NIOEN NCLEAR + ...	
		IOEN = IOSC IOEN6 NIOINH + ...	I/O enable
		S/IOSC = SC NSCINH IOPOP	Set at previous clock
		IOEN6 = FAMDS NFPRR NFSHEX IOEN6/1 PH6 NDIOEXIT	I/O enable, phase 6
		IOEN6/1 = NMC0005Z N(FADIV CC2) NEWDM	
		NMC0005Z = MC0 + MC1 + ... + MC5	
		NIONH = N(ADNH PH6 + ABO PH6 + INTRAP NPCP2)	Not I/O inhibit
		R/PH6 = ...	
	If shift count equals zero or signal FSHEX is true, and no I/O service call is pending, perform the following operations:		
	Set flip-flop MRQ	S/MRQ = (S/MRQ/1) + ...	Core memory request for next instruction in sequence
		(S/MRQ/1) = FAMDS PH6 NIOEN NBRPH6 + ...	
		R/MRQ = ...	
	Set flip-flop CC1 if fraction is normalized on left shift	S/CC1 = FUSF/1 FNORM + ...	
		FNORM = (A8 + A9 + A10 + A11) MC6 MC7 NFL2	
		R/CC1 = FASH NFUMH PRE3 + ...	
	Set flip-flop FL3 if fraction equals zero	S/FL3 = FUSF/1 N(S0 + S1 + ... + S31) N(B0 + B1 + ... + B31)	
		R/FL3 = N(FUSF PH8 + FUS PH5)	
	Set flip-flop CC2 if exponent underflow or overflow has occurred	S/CC2 = FUSF/1 A0 + ...	Exponent overflow occurs if value in exponent field is incremented over 127. Exponent underflow occurs if value is decremented below zero
		R/CC2 = FUSF PH8 FL3 NFUMH + ...	
	Disable signal BRPH6 and branch to PH7	NBRPH6 = MCZ + FSHEX + ...	
		S/PH7 = PH6 NBR NIOEN + ...	Signal NBR is true when BRPH6 is disabled
		R/PH7 = ...	
			Mnemonic: SF (24, A4)

(Continued)

Table 3-63. Shift Floating Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH7	One clock long		
T5L	Reset flip-flop NSXBF	S/NSXBF = N(S/SXB) (S/SXB) = FASH PH7 + ... R/NSXBF = ...	Preset logic for B ⟶ S in PH7
	Reset A0 to clear possible exponent overflow or underflow	R/A0 = FUSF/1 + ... FUSF/1 = FUSF NIOEN PH7 + ...	
	Set flip-flop FL3 if fraction equals zero	S/FL3 = FUSF/1 N(S0 + S1 + ... + S31) N(B0 + B1 + ... + B31) R/FL3 = (R/FL3) (R/FL3) = FUSF PH8 + ...	Fraction may have gone to zero on last shift of PH6
	Set flip-flop CC1 if fraction is normalized on left shift	S/CC1 = FUSF/1 FNORM + ... FNORM = (A8 + A9 + A10 + A11) MC6 MC7 NFL2 R/CC1 = FASH NFUMH PRE3 + ...	Fraction may have been normalized on last shift of PH6
	Set flip-flop CC2 if exponent overflow or underflow has occurred	S/CC2 = FUSF/1 A0 + ... R/CC2 = FUSF PH8 FL3 NFUMH + ...	Overflow or underflow may have occurred on last shift of PH6
PH8	One clock long		
T5L	(B0-B31) ⟶ (S0-S31) ⟶/ (D0-D31)	SXB = NDIS SXBF + ... SXBF = Set at PH7 clock DXS = FUSF PH8 + ...	Least significant word of shifted result transferred to D-register. This quantity is all zeros if a short-format shift is in effect
	If original floating point number was positive, enable signal (S/SXD). If original floating point number was negative, enable signal (S/SXMD), and reset flip-flop NK31	(S/SXD) = FUSF PH8 NFL1 + ... (S/SXMD) = FUSF PH8 FL1 + ... S/NK31 = N(S/K31) N(S/SXAMD/1) + N(S/K31/1) (S/K31) = (S/SXMD) + ... (S/K31/1) = (S/K31/2) (S/K31/3) + ... (S/K31/2) = High during Sf (S/K31/3) = High during SF R/NK31 = ...	Take correct value of the shifted result in PH9 and PH10
	Reset flip-flop NLR31F	S/NLR31F = N(S/LR31) (S/LR31) = FUSF PH8 + ... R/NLR31F = ...	Force a one on private memory address line LR31 during PH9 to select private memory register Ru1
			Mnemonic: SF (24, A4)

(Continued)

Table 3-63. Shift Floating Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH8 T5L (Cont.)	If long-format shift is being performed, set flip-flop RW	S/RW = FUSF PH8 D23 + ... R/RW = ...	Prepare to write least significant word of shifted result into private memory register Ru1
	If fraction is zero, perform the following operations:		
	Set flip-flop CC1 if left shift	S/CC1 = FUSF PH8 NFL2 FL3 + ... R/CC1 = FASH NFUMH PRE3 + ...	A zero fraction is considered to be normalized
	Reset flip-flop CC2	R/CC2 = FUSF PH8 FL3 + ...	Cancel a possible exponent underflow or overflow indication. Result is true zero
	Sustain the state of flip-flop FL3	R/FL3 = N(FUSF PH8) + ...	FL3 indicates a fraction of zero. This data is needed in PH9
PH9	One clock long		
T8L	If original floating point number was positive, (D0-D31) ⟶ (S0-S31)	Adder logic set at PH8 clock by (S/SXD)	
	If original floating point number was negative, (ND0-ND31) + K31 ⟶ (S0-S31)	Adder logic set at PH8 clock by (S/SXMD)	
	If long-format shift, (S0-S31) ⟶ (RW0-RW31)	RWXS/0-RWXS/3 = RW + ... RW = Set at PH8 clock	Transfer corrected shifted result to private memory register Ru1
	Set K31 if end carry resulted from (S/SXMD) operation	S/NK31 = N(S/K31) N(S/SXAMD/1) + N(S/K31/1) (S/K31) = (S/SXMD) + ... (S/K31/1) = K00 (S/K31/3) + ... (S/K31/3) = High during SF R/NK31 = ...	Provide carry to most significant word of result in PH10
	Set flip-flop RW	S/RW = FAMDS PH9 + ... R/RW = ...	Prepare to write most significant word of result into private memory register R (complete result if short-format shift is being performed)
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = PH9 + ... R/DRQ = ...	Inhibits transmission of another clock until data release signal is received from core memory
			Mnemonic: SF (24, A4)

(Continued)

Table 3-63. Shift Floating Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH9 T8L (Cont.)	Enable clock T8 if short-format shift	T8EN = NT5EN NT11L N(SXADD/1 RW) N(RW REU) N(REU AXRR)	T5 is disabled by signal SXADD/1
		NT5EN = RW + ...	
	Enable clock T11 if long-format shift	NT8EN = SXADD/1 RW + ...	
	If fraction is not zero, perform the following operations:		
	If original floating point number was positive, enable signal (S/SXA). If original floating point number was negative, enable signal (S/SXMD) and do not set K31 unless end carry is present	(S/SXA) = NFL1 (S/SX/FL1) + ...	Take correct value of most significant word of shifted result in PH10. If FL3 is set, zeros will be gated to sum bus in PH10
		(S/SX/FL1) = FUSF PH9 NFL3 + ...	
		(S/SXMD) = FL1 (S/SX/FL1) + ...	
PH10	Sustained until data release		
DR	If original floating point number was positive (A0-A31)⟶(S0-S31)	Adder logic set at PH9 clock by (S/SXA)	
	If original floating point number was negative (NA0-NA31) + K31 ⟶(S0-S31)	Adder logic set at PH9 clock by (S/SXMD)	
	(S0-S31)⟶(RW0-RW31)	RWXS/0-RWXS/3 = RW + ...	Transfer corrected shifted result to private memory register R. Zeros are transferred if fraction is zero (floating point number has been changed to true zero)
		RW = Set at PH9 clock	
	Set flip-flop CC3 if the result of the shift was greater than zero. Otherwise, reset CC3	S/CC3 = SGTZ TESTS + ...	
		SGTZ = (S0 + S1 + ... + S31) + (B0 + B1 + ... + B31)	
		= (NS0 NFACOMP) + ...	B-register contains least significant word of shifted result
		TESTS FUSF ENDE + ...	
		R/CC3 = TESTS + ...	
	Set flip-flop CC4 if shift result was less than zero. Otherwise, reset CC4	S/CC4 = (S/CC4/2) TESTS + ...	
		(S/CC4/2) = S0 NFACOMP + ...	
		R/CC4 = TESTS + ...	
	ENDE functions		
			Mnemonic: SF (24, A4)

3-74 Family of Floating Point Instructions

GENERAL. Implemented floating point instructions can be used to add, subtract, multiply, and divide floating point numbers. Floating point instructions are implemented by the addition of the floating point option to the Sigma 5 computer system. If the floating point option is not present in the system, and execution of a floating point instruction is attempted, the computer will abort execution of the instruction. A trap to memory location X'41' (65_{10}), the unimplemented instruction trap location, will result.

The following instructions are included in the floating point option:

Instruction	Mnemonic	Opcode
Floating Add, Short	FAS	3D, BD
Floating Add, Long	FAL	1D, 9D
Floating Subtract, Short	FSS	3C, BC
Floating Subtract, Long	FSL	1C, 9C
Floating Multiply, Short	FMS	3F, BF
Floating Multiply, Long	FML	1F, 9F
Floating Divide, Short	FDS	3E, BE
Floating Divide, Long	FDL	1E, 9E

FLOATING POINT HARDWARE. The floating point option of the Sigma 5 computer consists of additional modules that supplement the CPU logic. The assembly is called the floating point box since it is physically separate from the main CPU logic. Registers and logic in the floating point box are very similar to the main CPU logic.

Note

Actions that take place in the floating point box are underscored in the sequence charts for the floating point instructions. Main CPU functions are not underscored.

FLOATING POINT FORMATS. There are two floating point number formats for the Sigma 5 computer, short and long. Both are shown in figure 3-165. The short format is made up of a sign bit, bit 0; an excess-64 biased exponent, bits 1 through 7; and a 24-bit mantissa, bits 8 through 31. The long format adds 32 bits of lower significance to the mantissa.

A floating point number in the Sigma 5 has the following form:

$$\text{Floating Point Number} = S(M \times 16^{E})$$

S represents the sign bit of the number, bit 0. If the sign bit is a zero, the number is positive and in true form. If the sign bit is a one, the entire number is in two's complement form. M is the mantissa of 24 or 56 bits. The mantissa is a fraction with the binary point immediately before bit position 8. E is the exponent, with the bias of 64 removed. The term "inverted" refers to the exponent in a negative

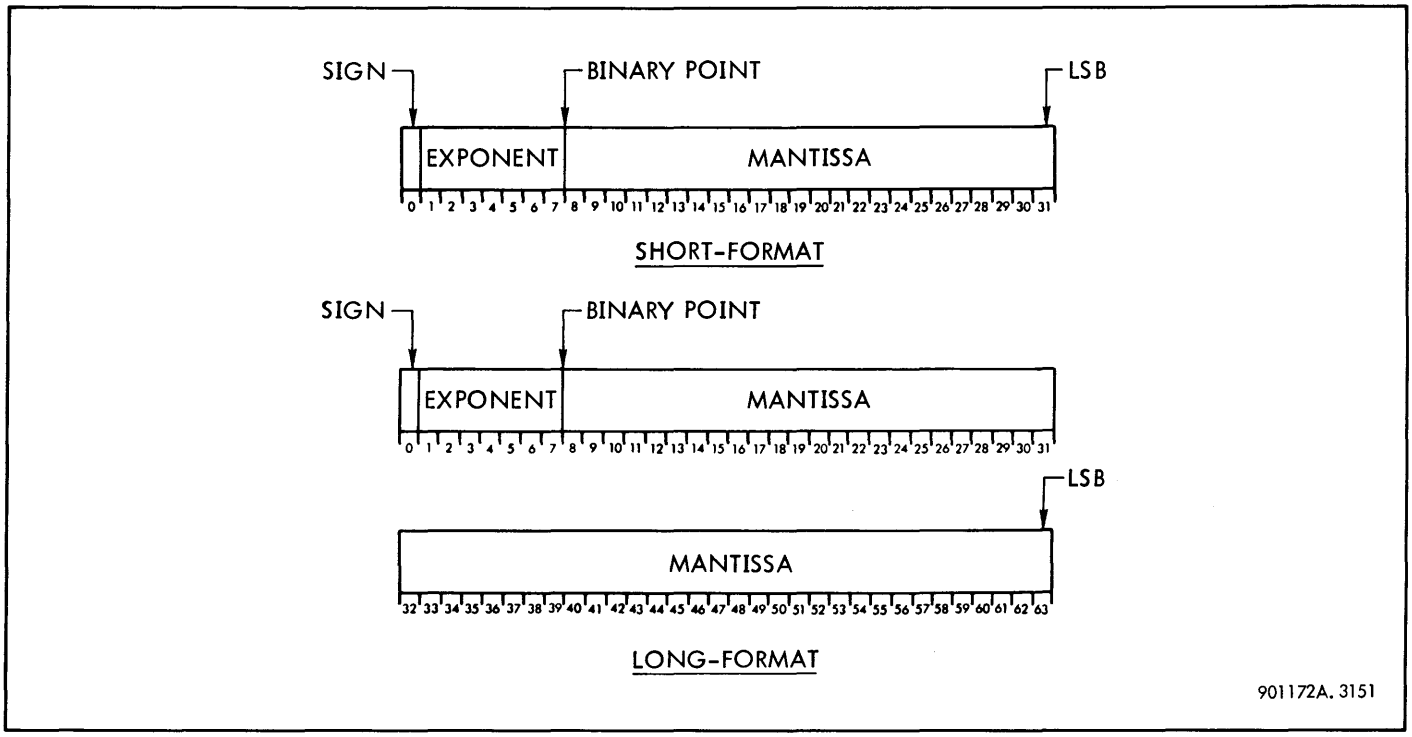


Figure 3-165. Floating Point Number Formats

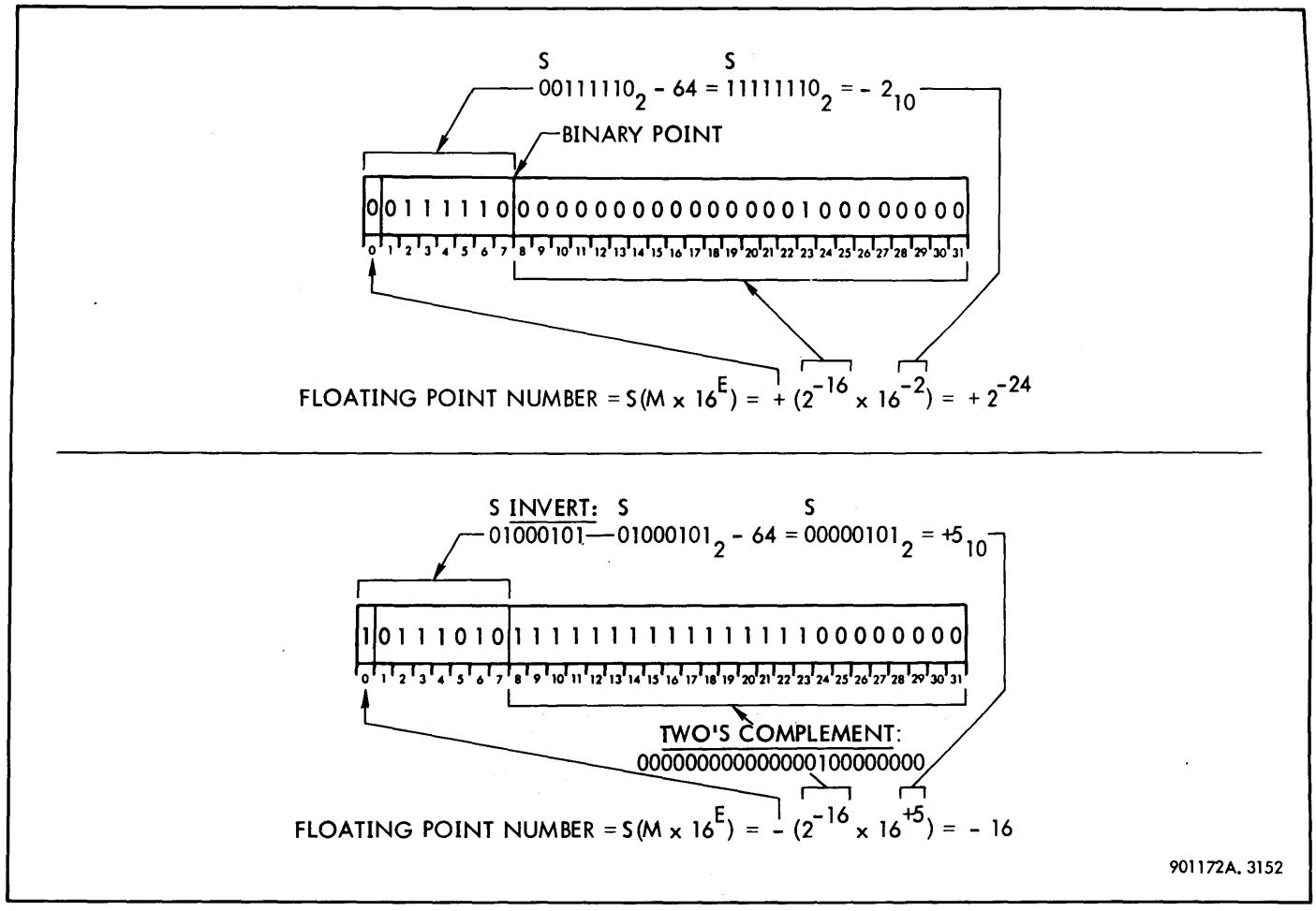


Figure 3-166. Floating Point Number Example

floating point number; the exponent must be uninverted before bias is removed.

The largest positive mantissa has all ones in the mantissa field of the floating point number, representing a quantity of $(1-2^{-24})$ or $(1-2^{-56})$. The smallest positive mantissa is all zeros, representing a quantity of zero. The largest mantissa in a negative floating point number is all zeros with a one in the least significant bit, $-(1-2^{-24})$ or $-(1-2^{-56})$; the smallest bit is a mantissa of all ones, $-(2^{-24})$ or $-(2^{-56})$.

The sign bit and the exponent field of the floating point number, taken together, represent an excess-64, signed, exponent quantity. For example, if a positive floating point number has an exponent field of 1111111_2, the excess-64, signed quantity represented is $\overset{S}{0}1111111_2$, or + 127. If the bias is removed by subtracting 64, the result is $\overset{S}{0}0111111_2$ or + 63. In this manner the exponent of a floating point number may be treated as a separate entity in arithmetic operations with other exponents. As another example, consider an exponent of 0111101_2 in a positive number. The exponent 0111101_2 in this number represents $\overset{S}{0}0111101_2 - \overset{S}{0}1000000_2 = \overset{S}{1}1111101_2$, or -3. Exponents in negative floating point numbers are inverted form. The exponent 1000010_2 in a negative number, when inverted, becomes $\overset{S}{0}0111101_2$. When bias is removed, the result is $\overset{S}{0}0111101_2 - \overset{S}{0}1000000_2 = \overset{S}{1}1111101_2$, or -3.

EXAMPLES OF FLOATING POINT NUMBERS. Examples of a positive and a negative floating point number with an explanation of the fields and conversion are shown in figure 3-166.

NORMALIZATION. A floating point number is said to be normalized if the absolute value of the mantissa is less than one but greater than 1/16. The mantissa of a positive floating point number must have a one somewhere in the four most significant bits (most significant hexadecimal digit must be nonzero) for the number to be normalized. A negative number is in two's complement form; a negative floating point number, therefore, must have a zero in the four most significant bits or have all ones in the four most significant bits and zeros in the remaining digits.

The negative floating point number 1XXXXXXX00←→0 is illegal, as the absolute value of this number is equal to one.

Normalization of a floating point number takes place as follows: the mantissa is shifted one hexadecimal place to the left. The exponent is decremented by one to compensate for the shift. Left shifting and decrementing of the fraction continues until the absolute value of the mantissa is greater than or equal to 1/16. Normalization is illustrated in figure 3-167.

A floating point number is said to be simple-normalized if it is the range of $+ 1/16 \leq N < 1$ or $-1 \leq N < -1/16$. Simple-normalized numbers are permissible only in the hardware while a floating point instruction is being implemented, and are not legal in memory.

FLOATING POINT MODE CONTROL BITS. Three bits in PSW1 of the program status doubleword control performance of floating point instructions.

Floating Point Addition and Subtraction. FN, floating normalize, is significant only during floating point additions and subtractions. If FN is a zero, the results of additions and subtractions are to be postnormalized. If exponent underflow occurs, if the result is zero, or if more than two hexadecimal shifts are required for normalization, floating mode bits FS and FZ determine the resultant action. If FN is a one, postnormalization is inhibited and FS and FZ are ignored.

FZ, floating zero, is significant during additions or subtractions if FN is a zero. If exponent underflow occurs during floating point addition or subtraction and the FZ bit is a zero, the result is set equal to all zeros, providing there is no trap by the FS bit. If exponent underflow occurs and the FZ bit is a one, the computer traps to location X'44' with the contents of private memory unchanged.

FS, floating significance, is significant during additions or subtractions if FN is a zero. If FS is a zero and the result is zero, the mantissa and exponent of the result are set

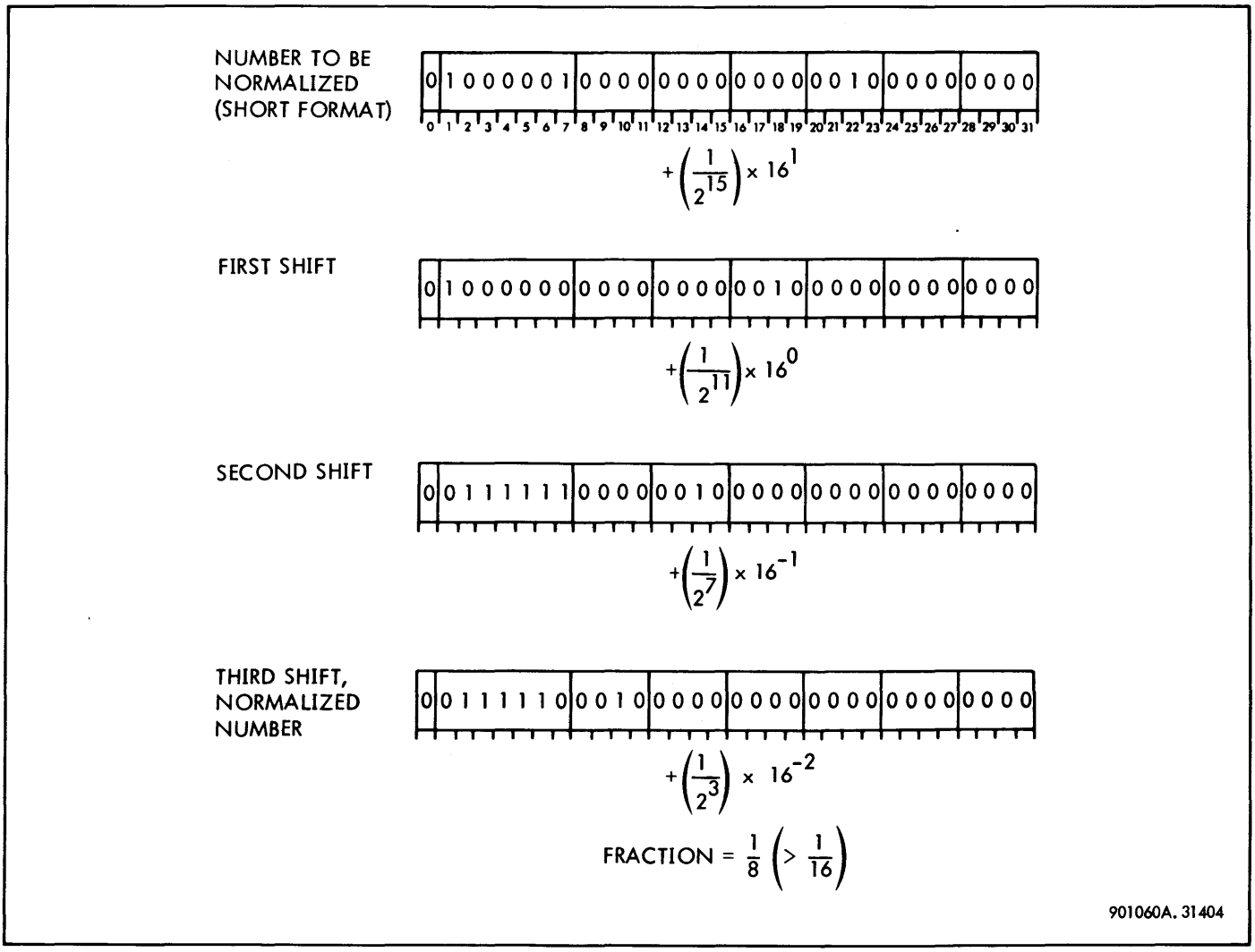


Figure 3-167. Normalization of Floating Point Numbers

equal to all zeros. If FS is a one and the result is zero, or more than two hexadecimal shifts are required for normalization of the result, the computer traps to location X'44' with the contents of private memory unchanged.

Exponent overflow unconditionally causes a trap to location X'44' with private memory unchanged.

Floating Point Multiplication and Division. If exponent overflow occurs during a floating multiply or divide or if division by zero is attempted, the computer unconditionally traps to location X'44'. Private memory remains unchanged.

If the FZ bit is a zero and the exponent of the result of a multiplication or division has been reduced below zero (underflow) or if the mantissa of the result is zero, the exponent and mantissa of the result are set equal to all zeros. If FZ is a one and one of these conditions occurs, the computer traps to location X'44'. Private memory remains unchanged.

Condition Code Settings. Condition code settings for the eight floating point instructions are shown in table 3-64.

FLOATING ADD, SHORT (FAS; 3D, BD). FAS adds the effective word and private memory register R. If no floating point arithmetic fault occurs, the sum is loaded into private memory register R.

FLOATING ADD, LONG (FAL; 1D, 9D). FAL adds the effective doubleword and private memory registers R and Rul. R must be an even value for correct results. If no floating point arithmetic fault occurs, the sum is loaded into private memory registers R and Rul

FLOATING SUBTRACT, SHORT (FSS; 3C, BC). FSS subtracts the effective word from the contents of private memory register R. If no floating point arithmetic fault occurs, the difference is loaded into private memory register R.

FLOATING SUBTRACT, LONG (FSL; 1C, 9C). FSL subtracts the effective doubleword from the contents of private memory registers R and Rul. R must be an even value for correct results. If no floating point arithmetic fault occurs, the sum is loaded into private memory registers R and Rul.

Table 3-64. Floating Point Condition Code Settings

CONDITION CODE (1 2 3 4)	MEANING IF NO TRAP TO LOCATION X'44' OCCURS	MEANING IF TRAP TO LOCATION X'44' OCCURS
0 0 0 0	A x 0, 0/A, or -A + A ① with FN=1 } Normal results	* ②
0 0 0 1	N < 0	*
0 0 1 0	N > 0	*
0 1 0 0	* ②	Divide by zero } Always trapped
0 1 0 1	*	Overflow, N < 0
0 1 1 0	*	Overflow, N > 0
③ 1 0 0 0	-A + A ①	-A + A
③ 1 0 0 1	N < 0 } > 2 Postnormalizing shifts } FS=0, FN=0, and no underflow	N < 0 } > 2 Postnormalizing shifts } FS=1, FN=0, and no underflow with FZ=1
③ 1 0 1 0	N > 0	N > 0
1 1 0 0	Underflow with FZ=0 and no trap by FS=1 ①	*
1 1 0 1	*	Underflow, N < 0 } FZ = 1
1 1 1 0	*	Underflow, N > 0

① Result set to true zero

② * indicates impossible configurations

③ Applies to add and subtract only where FN=0

FLOATING ADD AND SUBTRACT PHASE SEQUENCE. Preparation phases for FAS and FSS are the same as the general PREP phases for word instructions, paragraph 3-59. Preparation phases for FAL and FSL are the same as the general PREP phases for doubleword instruction, paragraph 3-59.

Figure 3-168 shows the general method of FAS, FAL, FSS, and FSL implementation. The example shown is one of a simplified floating addition. Table 3-65 lists the detailed logic for execution of floating add and floating subtract instructions.

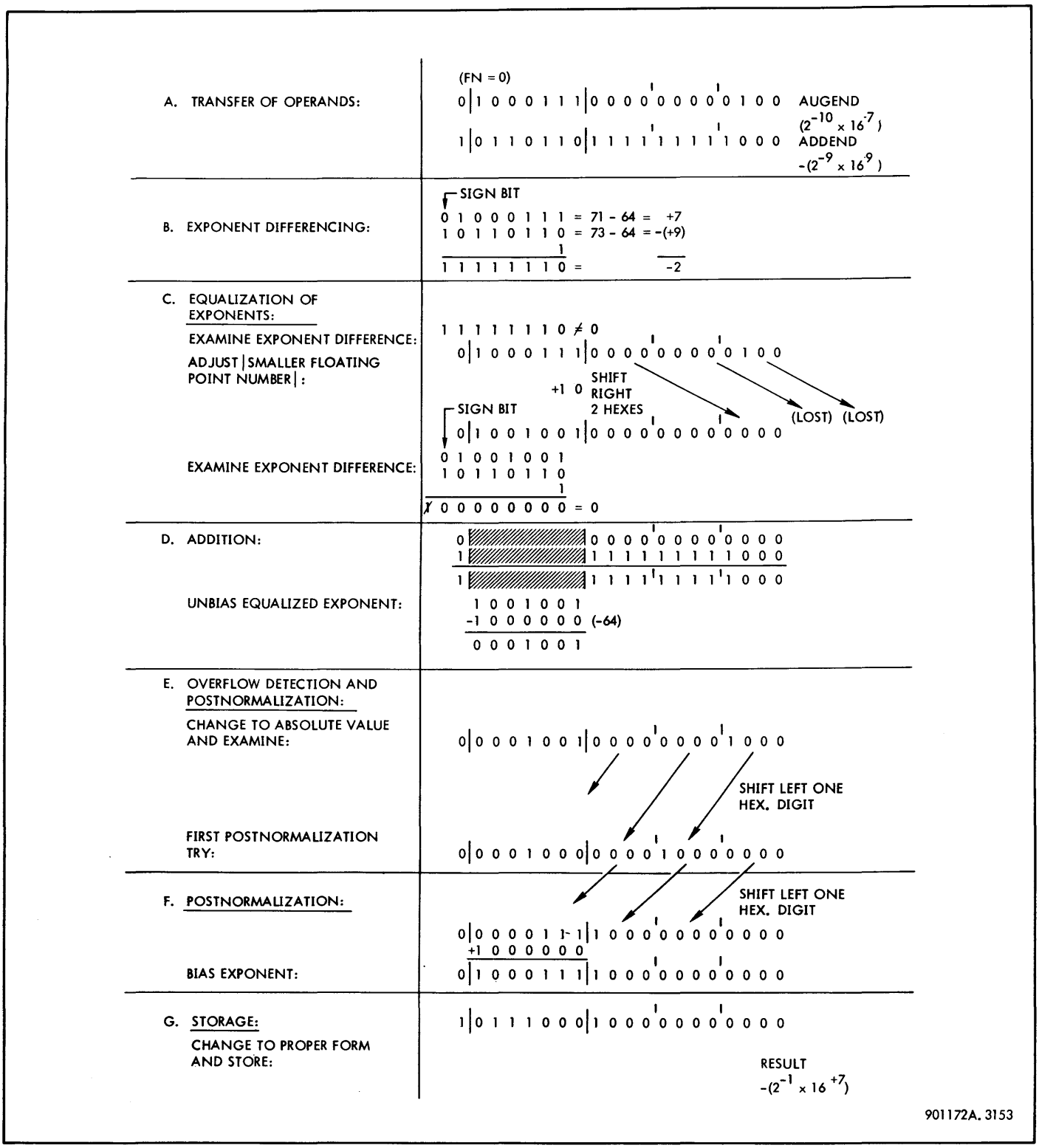


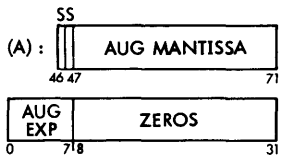
Figure 3-168. Floating Add and Subtract Implementation

Table 3-65. FAS, FSS, FAL, FSL Sequence

Phase	Function Performed	Signals Involved	Comments
		Note Actions that take place in the floating point box are underscored in the sequence charts for the floating point instructions. Main CPU functions are not underscored.	
PREP	At end of PREP:		
	(A): RR		Contents of private memory register R. Most significant word (MSW) of augend
	(C): Core memory operand MSW		MSW of addend
	(D): Core memory operand MSW		MSW of addend
	Enable signal (S/SXNA)	(S/SXNA) = FAFL PRE/34 + ...	Preset adder for -A→S in PH1
	If long format instruction is in effect, perform the following functions:		
	Force a one into P31	PUC31 = FAFL NO2 PRE3 NANLZ + ...	Prepare to obtain LSW of addend
	Set flip-flop MRQ	S/MRQ = (S/MRQ/1) + ... (S/MRQ/1) = FAFL NO2 NANLZ PRE3 + ... R/MRQ = ...	Memory request for LSW of addend. Inhibited if trap because floating point option not present
	Enable clock T8	S/NT8L = N(S/T8L) (S/T8L) = FAFL NIOACT NPH10 R/NT8L = ...	Clocks for remainder of floating point phases are T8 unless I/O service call is in effect (PH6)
	FPCON→floating point box	FPCON = FAFL PRE3 + ...	Start functions in floating point box
	Set flip-flop PH1	S/PH1 = FPCON NPH1 R/PH1 = ...	Sets Box PH1
CPU PH1; Box PH1; T8L	One clock long (NA0-NA31)→(S0-S31) (NS0-NS31)→(FP0-FP31)	Adder logic set at PH1 clock FPXS = NPH8 NDIS	Gate MSW of augend to FP lines
			Mnemonic: FAS (3D, BD) FSS (3C, BC) FAL (1D, 9D) FSL (1C, 9C)

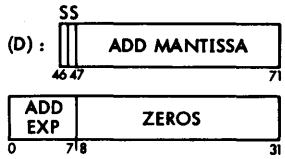
(Continued)

Table 3-65. FAS, FSS, FAL, FSL Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH1; Box PH1; T8L (Cont.)	FP0 ⟶ S46, S47	SXFP/U = S4607XFP	Sign of augend ⟶ S46, S47
		S4607XFP = PH1 NFPDIS + ...	
	(FP8-FP31) ⟶ (S48-S71)	SXFP/U = S4607XFP	Mantissa of augend ⟶ (S48-S71)
	(FP0-FP7) ⟶ (S0-S7)	SXFP/4 = S4607XFP + ...	Exponent of augend ⟶ (S0-S7)
	Zeros ⟶ (S8-S31)		No gating term enabled
	(S46-S71, S0-S31) ⟶ (A46-A71, A0-A31)	AXS = PH1 + ...	 901172A. 3154
	D0 ⟶ FPCON ⟶ floating point box	FPCON = FAFL PH1 D0 + ...	Transfer sign of addend to MWN in floating point box
	FPCON ⟶/⟶ MWN	S/MWN = FPCON PH1	
		R/MWN = PH1	
	Clear B-register	BX = PH1 + ...	
	Clear E-register	EX = PH1 + ...	
	Clear F-register	FX = PH1 + ...	
	If long format instruction is in effect, perform the following functions:		
	Force a one on private memory address line LR31	(S/LR31) = FAFL NO2 PH1 + ...	Prepare to obtain LSW of augend
	Reset flip-flop NAXRR	S/NAXRR = N(S/AXRR)	Preset logic for RRu1 ⟶/⟶ A in PH2
		(S/AXRR) = FAFL NO2 PH1 + ...	
		R/NAXRR = ...	
	Enable signal (S/SXND)	(S/SXND) = FAFL PH1 + ...	Preset adder for ND ⟶ S in PH2
	Set flip-flop PH2	S/PH2 = PH1	Box PH2
		R/PH2 = ...	
			Mnemonic: FAS (3D, BD) FSS (3C, BC) FAL (1D, 9D) FSL (1C, 9C)

(Continued)

Table 3-65. FAS, FSS, FAL, FSL Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH2; Box PH2; T8L	One clock long		
	(ND0-ND31) ⟶ (S0-S31)	Adder logic set at PH1 clock	MSW of addend ⟶ FP lines
	(NS0-NS31) ⟶ (FP0-FP31)	FPXS = NPH8 NDIS	
	FP0 ⟶ S46, S47	SXFP/U = S4607XFP	Sign of addend ⟶ S46, S47
		S4607XFP = PH2 NFPDIS + ...	
	(FP8-FP31) ⟶ (S48-S71)	SXFP/U = S4607XFP	Mantissa of addend ⟶ (S48-S71)
	(FP0-FP7) ⟶ (S0-S7)	SXFP/4 = S4607XFP	Exponent of addend ⟶ (S0-S7)
	Zeros ⟶ (S8-S31)		No gating term enabled
	(S46-S71, S0-S31) ⟶/⟶ (D46-D71, D0-D31)	DXS = PH2 + ...	 901172A.3155
	(NA0-NA7) ⟶/⟶ (F0-F7)	FXNA = PH2 NO6	Augend exponent ⟶/⟶ F-register
	If augend is negative	R/A0 = AX/L	Uninverted augend exponent ⟶/⟶ (A0-A7)
	(NA0-NA7) ⟶/⟶ (A0-A7)	AX/L = AXL + ...	
		AXL = PH2 A0	
		S/A1 = NA1 PH2 A0	
		⋮ ⋮	
		S/A7 = NA7 PH2 A0	
		R/A1-R/A7 = AX/L	
	Set flip-flop A8	S/A8 = PH2 NMUL + ...	For PH3 use
		R/A8 = AX/L	
	Set flip-flop D8 if addend is negative	S/D8 = PH2 MWN + ...	For PH3 use
		R/D8 = DX/L	
		DX/L = DX + ...	
		DX = PH2 + ...	

Mnemonic: FAS (3D, BD)
FSS (3C, BC)
FAL (1D, 9D)
FSL (1C, 9C)

(Continued)

Table 3-65. FAS, FSS, FAL, FSL Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH2; Box PH2; T8L (Cont.)	Enable signal (S/SXAPD) if addend is negative	(S/SXAPD) = (S/SXAPD/1) + ... (S/SXAPD/1) = PH2 NMUL MWN + ...	For exponent arithmetic in PH3
	Enable signal (S/SXAMD) if addend is positive	(S/SXAMD) = N(S/SXAPD) (S/SXAMD/2) + ... (S/SXAMD/2) = PH2 + ...	For exponent arithmetic in PH3
	If long format instruction is in effect, perform the following functions:		
	(RR0-RR31) ⟶ (A0-A31)	AXRR = Set at PH1 clock	LSW of augend ⟶ A-register
	Enable signal (S/SXNA)	(S/SXNA) = FAFL PH2 + ...	Preset adder for -A ⟶ S in PH3
	Set flip-flop DRQ	S/DRQ = (S/DRQ/2) + ... (S/DRQ/2) = FAFL NO2 PH2 + ... R/DRQ = ...	Inhibits transmission of another clock until data release received from core memory. (Memory request made during PREP)
	Set flip-flop PH3	S/PH3 = PH2 R/PH3 = ...	Box PH3
CPU PH3; Box PH3; T8L if short, DR if long	One clock long (A46-A71, A0-A31) ± (D46-D71, D0-D31) ⟶ (S46-S71, S0-S31)	Adder logic set at PH2 clock	(A0-A7) contains uninverted augend exponent. The adder is set to subtract the uninverted addend exponent from the uninverted augend exponent. D8, set in PH2 if addend negative, effectively adds a one for two's complement of addend exponent
	(S0-S7) ⟶ (E0-E7)	S/E0 = S0 PH3 + ... ⋮ S/E7 = S7 PH3 + ...	
		R/E0 = EX + ... ⋮ R/E7 = EX + ...	
	(NA0-NA31) ⟶ (S0-S31)	Adder logic set at PH2 clock	LSW of augend ⟶ FP lines if long format instruction. If short format, action is meaningless
	(NS0-NS31) ⟶ (FP0-FP31)	FPXS = NPH8 NDIS	

Mnemonic: FAS (3D, BD)
FSS (3C, BC)
FAL (1D, 9D)
FSL (1C, 9C)

(Continued)

Table 3-65. FAS, FSS, FAL, FSL Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH3; Box PH3; T8L if short, DR if long (Cont.)	If long format instruction (FP0-FP31)—/→(A0-A31)	AXFP = PH3 NO2	LSW of augend—/→ A-register
	If short format instruction Zeros—/→(A0-A31)		No gating term enabled
	(MB0-MB31)——→(C0-C31)—/→ (D0-D31) if long format instructions	DXC = FAFL NO2 PH3 + ...	LSW of addend——→C- and D-registers
	Enable signal (S/SXND)	(S/SXND) = FAFL PH3 + ...	Preset adder for -D——→S in PH4
	Set flip-flop PH4	S/PH4 = PH3 S/PH4 = ...	Box PH4
CPU PH4; Box PH4; T8L	One clock long (ND0-ND31)——→(S0-S31)	Adder logic set at PH3 clock	LSW of addend if long format instruction. Meaningless if short format
	(NS0-NS31)——→(FP0-FP31)	FPXS = NPH8 NDIS	
	If long format instruction (FP0-FP31)——→(S0-S31)	SXFP/4 = S0031XFP + ... SXFP/A = S0031XFP + ... S0031XFP = PH4 NO2 NFPDIS	
	If short format instruction Zeros——→(S0-S31)		No gating term enabled
	(S0-S31)—/→(D0-D31)	DXS/L = PH4 + ...	LSW of addend (if long format) or zeros (if short format)—/→D-register
	Clear condition code flip-flops	R/CC = FAFL PH4 + ...	
	Enable signal (S/SXB)	(S/SXB) = FAFL PH4 + ...	Preset logic for B——→S in PH5
	Branch to CPU PH5	S/PH5 = PH4 NBR R/PH5 = ...	CPU enters PH5. Floating point box may go to PH5 or PH6
	If exponent difference in E-register is equal to zero, perform the following functions:		
	Signal ASPP is enabled	ASPP = PH4 NO6 E0003Z E0407Z + ...	Add/subtract preparation. Addition or subtraction may be done, as exponents are equal
	Enable signal (S/SXAPD) if add instruction .	(S/SXAPD) = (S/SXAPD/1) + ... (S/SXAPD/1) = ASPP O7 NSW1 + ...	Preset adder for A + D——→S in PH5
			Mnemonic: FAS (3D, BD) FSS (3C, BC) FAL (1D, 9D) FSL (1C, 9C)

(Continued)

Table 3-65. FAS, FSS, FAL, FSL Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH4; Box PH4; T8L (Cont.)	Enable signal (S/SXAMD) if subtract instruction		Preset adder for A - D ⟶S in PH5
	(F0-F7) -64⟶(E0-E7) if exponent in F-register is uninverted	EXFM64 = ASPP NF0	The uninverted, unbiased augend exponent is transferred to F-register. This also is the exponent of the addend in this case
	(NF0-NF7) -64⟶(E0-E7) if exponent in F-register is inverted	EXNFM64 = ASPP F0	
	Branch to Box PH6	S/NPH6 = N(S/PH6) (S/PH6) = ASPP + ...	Go to add/subtract phase
		F/NPH6 = ...	
	If exponent difference in E-register is greater than zero, perform the following functions:		Augend exponent is greater than addend exponent
	Signal ALM is enabled	ALM = PH4 NO6 NE0 NASPP + ...	Right align addend
	Enable signal (S/SXD) if addend positive	(S/SXD) = (S/SXAVD) ND46 + ...	Prepare to take absolute value of addend in PH5. Preset adder logic
		(S/SXAVD) = PH4 ALM + ...	
	Enable signal (S/SXMD) if addend	(S/SXMD) = (S/SXAVD) D46 + ...	
	negative, and set flip-flop SW1	S/SW1 = (S/SW1/1) + ...	SW1 signifies that operand sign has been reversed
		(S/SW1/1) = ALM MWN + ...	
		R/SW1 = NPH9	
	Set flip-flop FPR if result of arithmetic operation in PH6 will be opposite to correct result	S/FPR = ALM N(MWN ⊕ O7) +...	Floating polarity reversed
		R/FPR = PH7 NO6 D46 + ...	
	(E0-E7) -1⟶(E0-E7)	EDC7 = ALM + ...	Downcount exponent difference toward zero. (Exponent difference is a positive number)
	Set flip-flop SW2	S/SW2 = PH4 ALM NPH7 + ...	Signifies that A⟶D transfer will be made in PH5
		R/SW2 = ...	
	Branch to Box PH5	S/NPH5 = N(S/PH5)	Go to alignment phase
		(S/PH5) = PH4 NO6 NASPP + ...	
		R/NPH5 = ...	

Mnemonic: FAS (3D, BD)
FSS (3C, BC)
FAL (1D, 9D)
FSL (1C, 9C)

(Continued)

Table 3-65. FAS, FSS, FAL, FSL Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH4; Box PH4; T8L (Cont.)	If exponent difference in E-register is less than zero, perform the following functions:		Addend exponent is greater than augend exponent
	Signal ALR is enabled	ALR = PH4 NO6 E0 NRTZ	Right align augend
	Enable signal (S/SXA) if augend is positive	(S/SXA) = (S/SXAVA) NA47 + ... (S/SXAVA) = PH4 ALR + ...	Prepare to take absolute value of augend in PH5
	Enable signal (S/SXMA) if augend	(S/SXMA) = (S/SXAVA) A47	Preset adder logic
	is negative, and set flip-flop SW1	S/SW1 = (S/SW1/1) + ... (S/SW1/1) = ALR A47 + ... R/SW1 = NPH9	SW1 signifies that operand sign has been reversed
	Set flip-flop FPR if result of arithmetic operation in PH6 will be opposite to correct result	S/FPR = ALR A47 + ... R/FPR = PH7 NO6 D46 + ...	Floating polarity reversed
	(E0-E7) -1 → (E0-E7)	EUC7 = ALR + ...	Upcount exponent difference toward zero. (Exponent difference is a negative number)
	(D0-D7) → (F0-F7)	FXD = PH4 ALR	Larger (addend) exponent transferred to F-register
	Branch to Box PH5	S/NPH5 = N(S/PH5) (S/PH5) = PH4 NO6 NASPP + ... R/NPH5 = ...	Go to alignment phase
CPU PH5 or PH6; Box PH5; T8L	This phase is entered only if the exponent difference in PH4 was nonzero Perform the following functions during the first clock period: If (E0-E7) > 0 in PH4: (A47-A71, A0-A31) → (D46-D71, D0-D31)	DXA = PH5 SW2 + ...	Larger (augend) operand → D-register
			Mnemonic: FAS (3D, BD) FSS (3C, BC) FAL (1D, 9D) FSL (1C, 9C)

(Continued)

Table 3-65. FAS, FSS, FAL, FSL Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH5 or PH6; Box T8L (Cont.)	|(D46-D71, D0-D31)| ⟶ (S46-S71, S0-S31)	Adder logic set at PH4 clock	Absolute value of smaller (addend) operand shifted right one hexadecimal and clocked into A-register
	(S46-S71, S0-S31) ⟶ 1/16 ⟶/ (A50-A71, A0-A31)	AXSR4 = AXSR4/1 AXSR4/1 = PH5 NO6 + ...	
	Zeros ⟶/ (A4-A7) if short format	S/A4 = S0 AXSR4 NO2 + ... ⋮ ⋮ S/A7 = S3 AXSR4 NO2 + ...	Guard digit is retained in A0-A3 if short format. Zeros are in A4-A31
		R/A4 = AX/L R/A7 = AX/L AX/L = AX + ... AX = AXSR4/1 AXSR4/1 = PH5 NO6 + ...	
	If (E0-E7) < 0 in PH4:		
	|(A46-A71, A0-A31)| ⟶ (S46-S71, S0-S31)	Adder logic set at PH4 clock	Absolute value of smaller (augend) operand shifted right one hexadecimal and clocked into A-register.
	(S46-S71, S0-S31) ⟶ 1/16 ⟶/ (A50-A71, A0-A31)	AXSR4 = AXSR4/1	
	Zeros ⟶/ (A4-A7) if short format		Guard digit is retained in A0-A3 if short format
	Count (E0-E7) towards zero	EUC7 = ALR + ... EDC7 = ALM + ...	E < 0 case E > 0 case
	Sustain the state of SW1	S/SW1 = (S/SW1/1) (S/SW1/1) = ALM MWN + ALR SW1 R/SW1 = NPH9	
	Enable signal (S/SXA)	(S/SXA) = PH5 NO6 NASPP + ...	Preset adder for A ⟶ S in next clock period
	Set flip-flop RTZ if sum bus is zero	S/RTZ = PH5 SZU SZL NSXADD NASPP SZU = N(S47 + S48 + ... + S71) SZL = N(S0 + S1 + ... + S31) NSXADD = NGXAD NPRXNAND R/RTZ = PH1 + ASPP	
	Sustain PH5 if exponents have not been equalized (E-register contents not zero) and mantissa is not zero	S/NPH5 = N(S/PH5) (S/PH5) = PH5 NO6 NASPP + ... NASPP = PH5 NO6 [NRTZ N(E0003Z E0407Z)]	PH5 is sustained as long as exponent difference has not been counted to zero or mantissa has not been shifted to zero

Mnemonic: FAS (3D, BD)
FSS (3C, BC)
FAL (1D, 9D)
FSL (1C, 9C)

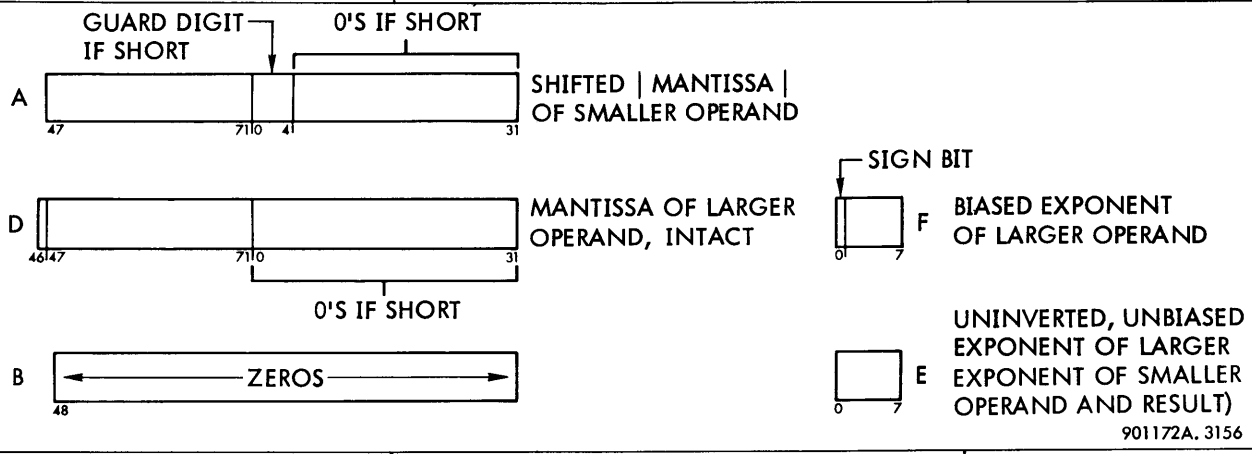
(Continued)

Table 3-65. FAS, FSS, FAL, FSL Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH5 or PH6; Box PH5; T8L (Cont.)		E0003Z = N(E0 + E1 + E2 + E3)	
		E0407Z = N(E4 + E5 + E6 + E7)	
		R/NPH5 = ...	
	Perform the following functions during all clocks but the last:		
	(A46-A71, A0-A31)⟶	Adder logic set at previous clock	Shift smaller operand
	(S46-S71, S0-S31)⟶1/16⟶ (A50-A71, A0-A31)	AXSR4 = AXSR4/1	right one hexadecimal
	Zeros⟶(A4-A7) if short format		Guard digit logic
	Count (E0-E7) toward zero	EUC7 = ALR + ...	E < 0 case
		EDC7 = ALM + ...	E > 0 case
	Sustain the state of SW1	S/SW1 = (S/SW1/1)	
		R/SW1 = NPH9	
	Enable signal (S/SXA)	(S/SXA) = PH5 NO6 NASP + ...	
	Set flip-flop RTZ if sum bus is zero	S/RTZ = PH5 SZU SZL NSXADD NASPP	
		R/RTZ = ASPP + ...	
	Sustain PH5 if E ≠ 0 and mantissa is not zero	(S/PH5) = PH5 NO6 NASPP + ...	
	Perform the following functions during the last clock period:		
	Signal ASPP is enabled	ASPP = PH5 NO6 (RTZ + E0003Z E0407Z)	
	(A46-A71, A0-A31)⟶	Adder logic set at previous clock	Last shift of one
	(S46-S71, S0-S31)⟶1/16⟶ (A50-A71, A0-A31)	AXSR4 = AXSR4/1	hexadecimal
	Zeros⟶(A4-A7) if short format		Guard digit logic
	Enable signal (S/SXAPD) if add and operand sign was not reversed or subtract and operand sign was reversed	(S/SXAPD) = (S/SXAPD/1) + ...	
		(S/SXAPD/1) = ASPP (O7 + SW1) + ...	
	Enable signal (S/SXAMD) if add and operand sign was reversed or subtract and operand sign was reversed	(S/SXAMD) = N(S/SXAPD) (S/SXAMD/2) + ...	
		(S/SXAMD/2) = ASPP + ...	
			Mnemonic: FAS (3D, BD) FSS (3C, BC) FAL (1D, 9D) FSL (1C, 9C)

(Continued)

Table 3-65. FAS, FSS, FAL, FSL Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH5 or PH6; Box PH5; T8L (Cont.)	(F0-F7) -64→(E0-E7) if NF0	EXFM64 = ASPP NF0	F0 holds sign of larger operand and (F1-F7) hold exponent of larger operand. (E0-E7) now hold the uninverted, unbiased exponent of the larger operand
	(NF0-NF7) -64→(E0-E7) if F0	EXNFM64 = ASPP F0	
	Reset flip-flop RTZ	R/RTZ = ASPP + ...	
	Branch to Box PH6	S/NPH6 = N(S/PH6) (S/PH6) = ASPP + ... R/NPH6 = ...	
CPU PH5 or PH6; Box PH6; T8L*	One clock long Resume of register contents:		
	 901172A.3156		
	(A46-A71, A0-A31) ± (D46-D71, D0-D31)→ (S46-S71, S0-S31)→ (D46-D71, D0-D31)	Adder logic set at PH5 clock DXS = PH6 NO6 + ...	Add or subtract the mantissas of the two floating point operands
	*If CPU accepts I/O service call, clocks to floating point box are rejected since they are T5L	FPCLEN/1 = NIOEN NIOIN + NFPRR FPCLEN/2 = NT5EN N(S/T8L) = FAFL (IOACT + PH10)	Floating point box continues operation after I/O service
			Mnemonic: FAS (3D, BD) FSS (3C, BC) FAL (1D, 9D) FSL (1C, 9C)

(Continued)

Table 3-65. FAS, FSS, FAL, FSL Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH5 or PH6; Box PH6; T8L (Cont.)	If intermediate result is positive, enable signal (S/SXAPD) If intermediate result is negative, enable signal (S/SXAMD) Clear A-register Branch to Box PH7	(S/SXAPD) = PH6 NO6 N(PR46⊕K46) + ... (S/SXAMD) = N(S/SXAPD) (S/SXAMD/2) + ... (S/SXAMD/2) = PH6 NO6 + ... AX = PH6 NO6 + ... S/NPH7 = N(S/PH7) (S/PH7) = PH6 NO6 + ... R/NPH7 = ...	Absolute value of intermediate result will be gated to sum bus in PH7
CPU PH6; Box PH7; T8L	One clock long (D46-D71, D0-D31) ⟶ (S46-S71, S0-S31) Reverse the state of flip-flop FPR if intermediate result is negative If D46 does not equal D47; perform the following functions: (S46-S71, S0-S31) x 1/16 ⟶ (A50-A71, A0-A31) Increment exponent of result by one	Adder logic set at PH6 clock S/FPR = PH7 NO6 D46 NFPR + ... R/FPR = PH7 NO6 D46 + ... AXSR4 = AXSR4/1 AXSR4/1 = PH7 NO6 (D46⊕D47) + ... EUC7 = AXSR4/1 + ...	Absolute value of intermediate result If FPR is now set, the quantity on the sum bus represents the reverse polarity of the actual result Shift result right one hexadecimal. Overflow has resulted from the addition or subtraction and mantissa must be shifted to correct. The exponent of the result in E-register is incremented by one to compensate for the shift
			Mnemonic: FAS (3D, BD) FSS (3C, BC) FAL (1D, 9D) FSL (1C, 9C)

(Continued)

Table 3-65. FAS, FSS, FAL, FSL Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH6; Box PH7; T8L (Cont.)	If no overflow has occurred, if the intermediate result is not simple-normalized, and if normalization is called for, perform the following functions:		
	(S46-S71, S0-S31) x 16 → (A47-A71, A0-A27)	AXSL4 = AXSL4/1	Shift result left one hexadecimal for normalization
		AXSL4/1 = PH7 NO6 NAXSR4/1 NDSN N(FNF NO6)	
		NDSN = D47 D48 D49 D50 D51 + ND47 D4851Z	No significance in most significant hexadecimal of mantissa
		S/FNF = S7 PSW1XS + ...	Floating normalize bit in PSW1
		R/FNF = PSW1XS + ...	
	Decrement exponent of result by one	EDC7 = AXSL4/1 N(PH5 DIV) + ...	The exponent of the result in E-register is decremented by one to compensate for the shift
	(B48-B71, B0-B31) x 2 → (B48-B71, B0-B30)	BXBL1 = AXSL4/1 NO6 + ...	For postnormalization counting
	Set flip-flop B67	S/B67 = NO6 BXBL1 + ...	
	If no overflow has occurred, if the intermediate result is simple-normalized, or if normalization is not called for, perform the following functions:		
	(S46-S71, S0-S31) → (A47-A71), A0-A31)	AXS = PH7 NO6 NAXSL4/1 NAXSR4/1 + ...	Absolute value of intermediate result → A-register
	Enable signal (S/SXA)	(S/SXA) = PH7 N(S/PH7)) + ...	Preset adder for A→S in PH8
		N(S/PH7) = N(PH7 DIV A47)N(MIT) ...	
	Branch to Box PH8	S/NPH8 = N(S/PH8)	
		(S/PH8) = PH7 N(S/PH7) + ...	
		R/NPH8 = ...	

Mnemonic: FAS (3D, BD)
FSS (3C, BC)
FAL (1D, 9D)
FSL (1C, 9C)

(Continued)

Table 3-65. FAS, FSS, FAL, FSL Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH6; Box PH8; T8L*	Number of clocks dependent upon normalization requirements		
	A-register contains the absolute value of the result in the range $0 \leq \text{result} \leq 1$		
	If the result is not simple-normalized and normalization is called for, perform the following functions:		If the |result| is not simple-normalized in this phase it is in the range $0 \leq \text{result} < 1/16$
	(A47-A71, A0-A31) ⟶ (S47-S71, S0-S31) x 16⟶ (A47-A71, A0-A27)	Adder logic set at PH7 clock	
		AXSL4 = AXSL4/1	
		AXSL4/1 = PH8 NDIV NASN + ...	
		NASN = A47 A48 A49 A50 A51 + NA47 A4851Z N(FNF NO6)	If FNF is a one, the result is not to be postnormalized
	Decrement exponent of result by one	EDC7 = AXSL4/1 N(PH5 DIV) + ...	The exponent of the result in the E-register is decremented by one to compensate for the shift
	(B48-B71, B0-B31) x 2⟶ (B48-B71, B0-B30)	BXBL1 = AXSL4/1 NO6 + ...	For postnormalization counting
	Set flip-flop B67	S/B67 = NO6 BXBL1	
	Set flip-flop RTZ if sum bus quantity is zero	S/RTZ = PH8 SZU SZL NASPP NSXADD + PH8 SZU O2 NO6 + ...	Short format case where significance exists in guard digit only
		SZU = N(S47 S48 ... S71)	
		SZL = N(S0 S1 ... S31)	
	*If CPU accepts I/O service call, clocks to floating point box are rejected since they are T5L	FPCLEN/1 = NIOEN NIOIN + NFPRR FPCLEN/2 = NT5EN N(S/T8L) = FAFL (IOACT + PH10)	Floating point box continues operation after I/O service
			Mnemonic: FAS (3D, BD) FSS (3C, BC) FAL (1D, 9D) FSL (1C, 9C)

(Continued)

Table 3-65. FAS, FSS, FAL, FSL Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH6; Box PH8; T8L (Cont.)	Enable signal (S/SXA) if result will not be ready at this clock	(S/SXA) = AXSL4/1 NFPRR + ... FPRR = PH8 NDIV (ASN + NA5255Z) + ...	Result will not be simple-normalized at this clock
	Sustain PH8 if result will not be ready at this clock	S/NPH8 = N(S/PH8) (S/PH8) = PH8 NFPRR + ... R/NPH8 = ...	More shifting must be done to normalize result. Repeat the same functions as above until result is ready
	If result will be normalized at next clock or if result is equal to zero, enable signal NFPRR and perform functions listed at end of this phase	FPRR = PH8 NDIV (ASN + NA5255Z) + RTZ + ...	Result is nearly ready to be sent back to CPU
	If the result is simple-normalized and A47 is a one, perform the following functions:		|Result| is equal to one in this case, and must be shifted right to represent a legal floating point number
	(A47-A71, A0-A31) ⟶ (S47-S71, S0-S31) x 1/16 ⟶ (A51-A71, A0-A31)	Adder logic set at PH7 clock AXSR4 = AXSR4/1 AXSR4/1 = PH8 NDIV A47 + ...	
	Increment exponent of result by one	EUC7 = NPH5 AXSR4/1 + ...	
	Enable signal FPRR and perform functions listed at end of this phase	FPRR = PH8 NDIV ASN + ...	Prepare to send result back to CPU
	If the result is simple-normalized and A47 is a zero, perform the following functions: (A47-A71, A0-A31) ⟶ (S47-S71, S0-S31)	Adder logic set at PH7 clock	|Result| is equal to 1/16 ≤ A < 1 in this case, or FNF is equal to a one
			Mnemonic: FAS (3D, BD) FSS (3C, BC) FAL (1D, 9D) FSL (1C, 9C)

(Continued)

Table 3-65. FAS, FSS, FAL, FSL Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH6; Box PH8; T8L (Cont.)	Set flip-flop RTZ if sum bus quantity is zero	S/RTZ = PH8 SZU SZL NASPP NSXADD + PH8 SZU O2 NO6 + ... R/RTZ = ASPP + ...	This is the case in which ASN is true because FNF is equal to a one
	Enable signal FPRR and perform functions listed at end of this phase	FPRR = PH8 NDIV ASN + ... ASN = NA47 A4851Z N(FNF NO6) + ...	
	FPRR functions:		
	Enable CPU PH7	S/PH7 = PH6 NBR NIOEN + ... NBR = NBRPH6 ... NBRPH6 = N(FAFL PH6 NFPRR) + ... R/PH7 = ...	
	Set flip-flop MRQ	S/MRQ = (S/MRQ/1) + ... (S/MRQ/1) = FAFL PH6 NIOEN NBRPH6 + ... R/MRQ = ...	Request for next instruction in sequence
	Enable signal (S/SXA) if NFPR Enable signal (S/SXMA) if FPR	(S/SXA) = FPRR NFPR + ... (S/SXMA) = FPRR FPR + ...	Preset adder logic to give result the proper polarity
	Set Box PH9	S/PH9 = FPRR R/PH9 = ...	
CPU PH7; Box PH9; T8L	One clock long (A47-A71, A0-A31)→ or -(A47-A71, A0-A31)→ (S47-S71, S0-S31)	Adder logic set at PH8 clock	Mantissa of result, in proper polarity, transferred to sum bus
	Transfer (S0-S31) to FP0-FP31 lines providing none of the following conditions are present:	FPXSL = NFPDIS PH9 FPRD NRTZ N(FEUF NFZ) + ...	LSW of floating point result
	Short format instruction in effect (FAS, FSS)	FPRD = NO2 + ...	

Mnemonic: FAS (3D, BD)
FSS (3C, BC)
FAL (1D, 9D)
FSL (1C, 9C)

(Continued)

Table 3-65. FAS, FSS, FAL, FSL Sequence (Cont.)

Phase	Function Performed	Signals Involved			Comments
CPU PH7; Box PH9; T8L (Cont.)	Result is equal to zero	RTZ			
	Exponent underflow with FZ equal to zero	FEUF	=	E0 NE1 NRTZ N(B65 NO6 FS NFZ)	Exponent was decremented below zero
	If one of the above conditions exists, transfer zeros to (FP0-FP31)				No gating term enabled
	(FP0-FP31)⟶(B0-B31)	BXFP	=	FAFL PH7 + ...	LSW of floating point result
	Reset flip-flop NSXBF	S/NSXBF	=	N(S/SXB)	Preset logic for B⟶S in PH8
		(S/SXB)	=	FAFL PH7 + ...	
		R/NSXBF	=	...	
	Force a one on private memory address line LR31	S/NLR31F	=	N(S/LR31)	Select odd-numbered private memory address during PH10
		(S/LR31)	=	FAFL PH7 + ...	
		R/NLR31F	=	...	
	Set flip-flop RW if long format instruction and TRAP signal is not true	S/RW	=	(S/RW/FP)	Prepare to send LSW of result to CPU
		(S/RW/FP)	=	PH9 NTRAP FPRD + ...	
		R/RW	=	...	
	Set flip-flop CC1 if exponent underflow or if FN = 0 and more than two postnormalizing shifts are required	S/CC1	=	(S/CC1/3) + ...	
		(S/CC1/3)	=	(S/CC1/FP) + ...	
		(S/CC1/FP)	=	PH9 (FEUF + B65 NO6)	
		R/CC1	=	(R/CC1)	
	Set flip-flop CC2 if exponent underflow	S/CC2	=	(S/CC2/3) + ...	
		(S/CC2/3)	=	(S/CC2/FP) + ...	
		(S/CC2/FP)	=	PH9 (FEUF + ...)	
		R/CC2	=	(R/CC2)	
	Enable TRAP signal if exponent underflow has occurred and FZ is a one, or if a trap on significance is called for (FS is a one)	TRAP	=	FEUF N(FEUF NFZ) + B65 NO6 FS + ...	TRAP prevents RW from being set in PH9 and PH10
	Enable signal (S/SXA) if NFPR is true	(S/SXA)	=	PH9 NFPR + ...	Preset adder logic to give result the proper polarity
	Enable signal (S/SXMA) if FPR is true	(S/SXMA)	=	PH9 NFPR + ...	
	If FPR is true, transfer (NE0-NE7)⟶(E0-E7)	EXNE	=	PH9 FPR NTRAP N(FEUF NFZ)	A negative result requires an inverted exponent
					Mnemonic: FAS (3D, BD) FSS (3C, BC) FAL (1D, 9D) FSL (1C, 9C)

(Continued)

Table 3-65. FAS, FSS, FAL, FSL Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH7; Box PH9; T8L (Cont.)	Branch to Box PH10	S/PH10 = PH9 R/PH10 = ...	
CPU PH8; Box PH10; T8L	One clock long (B0-B31)⟶(S0-S31)⟶ (RW0-RW31)	Logic set at PH7 clock RWXS/0-RWXS/3 = RW ... RW = Set at PH7 clock if no trap condition and long format	Transfer LSW of result to private memory register Ru1
	If LSW of result is not equal to zero, set flip-flop SW0	S/SW0 = NS0031Z (S/SW0/NZ) +... (S/SW0/NZ) = FAFL NO2 PH8 + ... R/SW0 = RESET/A + ...	Used in PH10 for condition code settings
	Reset flip-flop NSXBF	S/NSXBF = N(S/SXB) (S/SXB) = FAFL PH8 + ... R/NSXBF = ...	Preset logic for B⟶S in PH10
	Set flip-flop RW if TRAP signal is not true	S/RW = (S/RW/FP) (S/RW/FP) = PH10 NTRAP + ... R/RW = ...	Prepare to send MSW of result to CPU
	Set flip-flop DRQ	S/DRQ = BRPH10 + ... BRPH10 = FAFL PH8 + ... R/DRQ = ...	Inhibits transmission of another clock until data release received from core memory. Request for next instruction made in PH6
	(A47-A71, A0-A31)⟶ or -(A47-A71, A0-A31)⟶ (S47-S71, S0-S31)	Adder logic set at PH9 clock	MSW of mantissa⟶ sum bus
			Mnemonic: FAS (3D, BD) FSS (3C, BC) FAL (1D, 9D) FSL (1C, 9C)

(Continued)

Table 3-65. FAS, FSS, FAL, FSL Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH8; Box PH10; T8L (Cont.)	S47 ⟶ FP0	FP0 = S47 FPXSU + ...	MSW of result transferred to FP lines if result not equal to zero or if underflow with FZ = 0 does not exist
		FPXSU = NFPDIS PH10 NRTZ N(FEUF NFZ)	
	NE1 ⟶ FP1 (+64 bias)	FP1 = NE1 FPXSU + ...	
	(E2-E7) ⟶ (FP2-FP7)	FPXSU	
	(S48-S71) ⟶ (FP8-FP31)	FPXSU	
	(FP0-FP31) ⟶/⟶ (B0-B31)	BXFP = FAFL PH8 + ...	MSW of result ⟶/⟶ B-register
	Reset Box PH10	R/PH10 = ...	Floating point box actions are finished
	Branch to CPU PH10	S/PH10 = BRPH10 NCLEAR + ...	
		R/PH10 = ...	
CPU PH10; Box actions over; T8L	One clock long		
	(B0-B31) ⟶ (S0-S31) ⟶ (RW0-RW31)	Adder logic set at PH8 clock RWXS/0-RWXS/3 = RW ... RW = Set at PH8 clock if no trap condition	
	Set flip-flop CC3 if floating point result is positive	S/CC3 = SGTZ TESTS	
		SGTZ = (S0 + S1 + ... S31 + SW0) NS0 ...	SW0 is set when there is significance in LSW
		R/CC3 = TESTS + ...	
		TESTS = FAFL ENDE + ...	
	Set flip-flop CC4 if floating point result is negative	S/CC4 = (S/CC4/2) TESTS + ...	
		(S/CC4/2) = NFACOMP S0 + ...	
	ENDE functions	R/CC4 = TESTS + ...	
			Mnemonic: FAS (3D, BD) FSS (3C, BC) FAL (1D, 9D) FSL (1C, 9C)

FLOATING MULTIPLY, SHORT (FMS; 3F, BF). FMS multiplies the effective word by the contents of private memory register R. If no floating point arithmetic fault occurs, the product is loaded into private memory as follows: If R is an even value, the product is loaded into private memory registers R and Ru1 as a long format floating point number. If R is an odd value, the product is effectively truncated and loaded into private memory register R. The product is always normalized.

FLOATING MULTIPLY, LONG (FML; 1F, 9F). FML multiplies the effective doubleword by the contents of private memory registers R and Ru1. R must be an even value for correct results. If no floating point arithmetic fault occurs, the product is truncated and loaded into private memory registers R and Ru1 as a long format floating point number. The product is always normalized.

FLOATING MULTIPLY PHASE SEQUENCE. Preparation phases for FMS are the same as the general PREP phases for word instructions, paragraph 3-59. FML preparation phases are described in paragraph 3-59. Figure 3-169 shows the general method of FMS and FML execution. Bit-pair multiplication (described in paragraph 3-67) is used during the actual multiply iterations. Table 3-66 lists the detailed logic for execution of the floating multiply instructions.

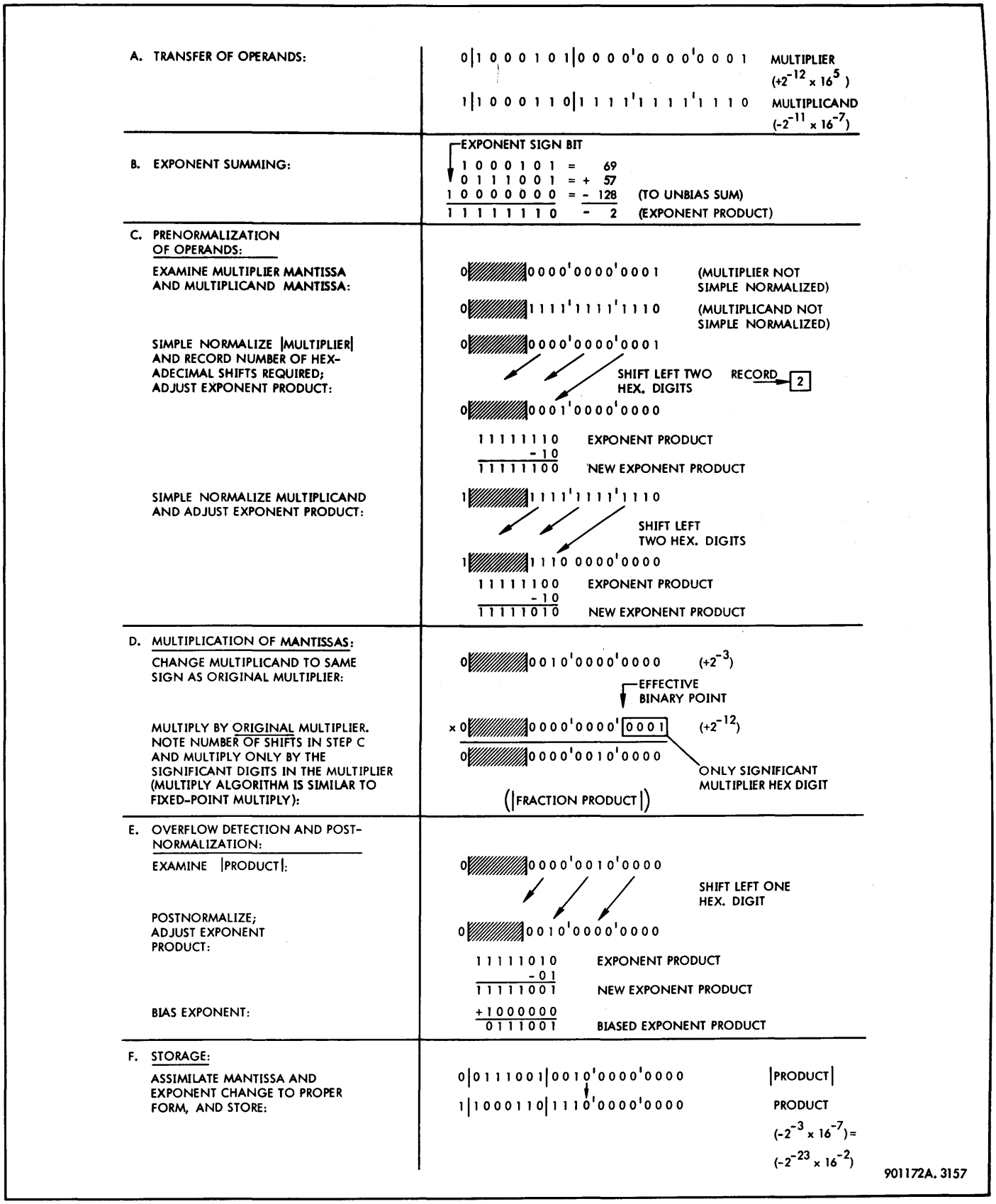


Figure 3-169. Floating Multiply Implementation

Table 3-66. FMS, FML Sequence

Phase	Function Performed	Signals Involved	Comments
		Note Actions that take place in the floating point box are underscored in the sequence charts for the floating point instructions. Main CPU functions are not underscored.	
PREP	At end of PREP:		
	(A): RR		Contents of private memory register R. MSW of multiplier
	(C): Core memory operand MSW		MSW of multiplicand
	(D): Core memory operand MSW		MSW of multiplicand
	Enable signal (S/SXNA)	(S/SXNA) = FAFL PRE/34 + ...	Preset adder for -A→S in PH1
	If long format instruction is in effect perform the following functions:		
	Force a one into P31	PUC31 = FAFL NO2 PRE3 NANLZ + ...	Prepare to obtain LSW of multiplicand
	Set flip-flop MRQ	S/MRQ = (S/MRQ/1) + ... (S/MRQ/1) = FAFL NO2 NANLZ PRE3 + ... R/MRQ = ...	Memory request for LSW of multiplicand. Inhibited if floating point option trap is present
	Enable clock T8	S/NT8L = N(S/T8L) (S/T8L) = FAFL NIOACT NPH10 R/NT8L = ...	Clocks for remainder of floating point phases are T8 unless I/O service call is in effect (PH6)
	FPCON→floating point box	FPCON = FAFL PRE3 + ...	Start functions in floating point box
	Set flip-flop PH1	S/PH1 = FPCON NPH1 R/PH1 = ...	Sets Box PH1
CPU PH1; Box PH1; T8L	One clock long		
	(NA0-NA31)→(S0-S31)	Adder logic set at PH1 clock	Gate MSW of multiplier to FP lines
	(NS0-NS31)→(FP0-FP31)	FPXS = NPH8 NDIS	
	FP0→S46, S47	SXFP/U = S4607XFP S4607XFP = PH1 NFPDIS + ...	Sign of multiplier→S46, S47
			Mnemonic: FMS (3F, BF) FML (1F, 9F)

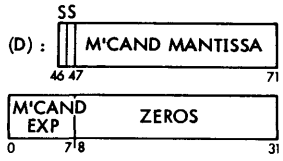
(Continued)

Table 3-66. FMS, FML Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH1; Box PH1; T8L (Cont.)	(FP8-FP31)⟶(S48-S71)	SXFP/U = S4607XFP	Mantissa of multiplier ⟶(S48-S71)
	(FP0-FP7)⟶(S0-S7)	SXFP/4 = S4607XFP	Exponent of multiplier ⟶(S0-S7)
	Zeros⟶(S8-S31)		No gating term enabled
	(S46-S71, S0-S31)⟶ (A46-A71, A0-A31)	AXS = PH1 + ...	(A): SS, MUL'IER MANTISSA (46 47 71); MUL EXP, ZEROS (0 7 8 31)
	(FP31-FP08)⟋⟶B4871	BXFP/U = PH1 MUL	MSW of multiplier mantissa⟋⟶B-register, backwards
	Zeros⟶(B0-B31)	MUL = O6 O7	(B): MUL'IER MANTISSA (BACKWARDS) (48 71); ZEROS (0 7 8 31)
			901172A.3158
	D0⟶FPCON⟶floating point box	FPCON = FAFL PH1 D0 + ...	Transfer sign of multiplier to MWN in floating point box
	FPCON⟋⟶MWN	S/MWN = FPCON PH1	
		R/MWN = PH1	
	Clear E-register	EX = PH1 + ...	
	Set F-register to 5_{10}	S/F5 = BXFP/U + ...	F-register and SW0 used as iteration counter
		S/F7 = BXFP/U + ...	
		R/FS, F/57 = EX + ...	
		FX = PH1 + ...	
	Reset flip-flop SW0	R/SW0 = BX	
	If FML is in effect, perform the following functions:		
	Force a one on private memory address line LR31	(S/LR31) = FAFL NO2 PH1 + ...	Prepare to obtain LSW of multiplier
	Reset flip-flop NAXRR	S/NAXRR = N(S/AXRR)	Preset logic for RRu1 ⟋⟶A in PH2
		(S/AXRR) = FAFL NO2 PH1 + ...	
		R/NAXRR = ...	
			Mnemonic: FMS (3F, BF) FML (1F, 9F)

(Continued)

Table 3-66. FMS, FML Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH1; Box PH1; T8L (Cont.)	Enable signal (S/SXND)	(S/SXND) = FAFL PH1 + ...	Preset adder for ND→S in PH2
	Set flip-flop PH2	S/PH2 = PH1 R/PH2 = ...	Box PH2
CPU PH2; Box PH2; T8L	One clock long		
	(ND0-ND31)→(S0-S31)	Adder logic set at PH1 clock	MSW of multiplicand →FP lines
	(NS0-NS31)→(FP0-FP31)	FPXS = NPH8 NDIS	
	FP0→S46, S47	SXFP/U = S4607XFP S4607XFP = PH2 NFPDIS + ...	Sign of multiplicand →S46, S47
	(FP8-FP31)→(S48-S71)	SXFP/U = S4607XFP	Mantissa of multiplicand →(S48-S71)
	(FP0-FP7)→(S0-S7)	SXFP/U = S4607XFP	Exponent of multiplicand →(S0-S7)
	Zeros→(S8-S31)		No gating term enabled
	(S46-S71, S0-S31)→/→(D46-D71, D0-D31)	DXS = PH2 + ...	 901172A.3159
	If multiplier is negative, (NA0-NA7)→/→(A0-A7). Merge one into A0 unconditionally	S/A0 = PH2 MUL + ... S/A1 = NA1 PH2 A0 + ... ⋮ ⋮ S/A7 = NA7 PH2 A0 + ... R/A0-R/A7 = AX/L AX/L = AXL + ... AXL = PH2 A0 + ...	Uninverted exponent →/→(A0-A7). The one in A0 effectively removes the bias of 128 which will result when the exponents of the multiplier and multiplicand are added in PH3
	Set flip-flop D8 if multiplicand is negative	S/D8 = PH2 MWN R/D8 = DX/L DX/L = DX + ... DX = PH2 + ...	For PH3 use
			Mnemonic: FMS (3F, BF) FML (1F, 9F)

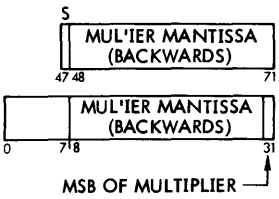
(Continued)

Table 3-66. FMS, FML Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH2; Box PH2; T8L (Cont.)	Enable signal (S/SXAPD) if multiplicand is positive	(S/SXAPD) = (S/SXAPD/1) + ... (S/SXAPD/1) = PH2 MUL NMWN + ...	For exponent arithmetic in PH3
	Enable signal (S/SXAMD) if multiplicand is negative	(S/SXAMD) = N(S/SXAPD) (S/SXAMD/2) + ... (SXAMD/2) = PH2 + ...	For exponent arithmetic in PH3
	If FML is in effect, perform the following functions:		
	(RR0-RR31) —/→ (A0-A31)	AXRR = Set at PH1 clock	LSW of multiplier —/→ A-register
	Enable signal (S/SXNA)	(S/SXNA) = FAFL PH2 + ...	Preset adder for -A——→ S in PH3
	Set flip-flop DRQ	S/DRQ = (S/DRQ/2) + ... (S/DRQ/2) = FAFL NO2 PH2 + ... R/DRQ = ...	Inhibits transmission of another clock until data release received from core memory. (Memory request made during PREP)
	Set flip-flop PH3	S/PH3 = PH2 R/PH3 = ...	Box PH3
CPU PH3; Box PH3; T8L if short; DR if long	One clock long		
	(A0-A7) ± (D0-D7) -128——→	Adder logic set at PH2 clock	Arithmetic operation is performed that adds the uninverted multiplicand exponent to the uninverted multiplier exponent. This results in a bias of 128, which is effectively removed by merging a one into A0 at the PH2 clock. The E-register now holds the unbiased sum of the exponents
	(S0-S7) —/→ (E0-E7)	S/E0 = S0 PH3 + ... ⋮ ⋮ S/E7 = S7 PH3 + ... R/E0-R/E7 = PH1 + ...	
	(NA0-NA31)——→(S0-S31)	Adder logic set at PH2 clock	LSW of multiplier ——→ FP lines if long format. If FMS, action is meaningless
	(NS0-NS31)——→(FP0-FP31)	FPXS = NPH8 NDIS	
	If FML is being performed:	AXFP = PH3 NO2	LSW of multiplier —/→ A-register
	(FP0-FP31) —/→ (A0-A31)		
	If FMS is being performed:		No gating term enabled
	Zeros —/→ (A0-A31)		
			Mnemonic: FMS (3F, BF) FML (1F, 9F)

(Continued)

Table 3-66. FMS, FML Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH3; Box PH3; T8L if short; DR if long (Cont.)	If FML is being implemented, perform the following functions:		Contents of B-register at the end of PH3 if FML:
	(B48-B71)—/→(B8-B31)	BXFP/L = PH3 MUL NO2	
	(FP7-FP0)—/→(B0-B7)	BXFP/L	 901172A.3160
	(FP31-FP8)—/→(B48-B71)	BXFP/U = PH3 MUL NO2	
	Set flip-flop F4 of F-register	S/F4 = BXFP/L + ...	Change the count in F-register from 5_{10} to 13_{10}. Count is now 5 if FMS or 13 if FML
		R/F4 = FX + ...	
	(MB0-MB31)—→(C0-C31)—/→ (D0-D31) if FML	DXC = FAFL NO2 PH3 + ...	LSW of multiplicand —→C- and D-registers
	Enable signal (S/SXND)	(S/SXND) = FAFL PH3 + ...	Preset adder for -D —→S in PH4
	Set flip-flop PH4	S/PH4 = PH3 (...)	Box PH4
		R/PH4 = ...	
CPU PH4; Box PH4; T8L	One clock long		
	(ND0-ND31)—→(S0-S31)	Adder logic set at PH3 clock	LSW of multiplicand if FML. Meaningless if FMS
	(NS0-NS31)—→(FP0-FP31)	FPXS = NPH8 NDIS	
	If FML:	SXFP/4 = S0031XFP + ...	
	(FP0-FP31)—→(S0-S31)	SXFP/A = S0031XFP + ...	
		S0031XFP = PH4 NO2 NFPDIS	
	If FMS:		No gating term enabled
	Zeros—→(S0-S31)		
	(S0-S31)—/→(D0-D31)	DXS/L = PH4 + ...	LSW of multiplicand (if FML) or zeros (if FMS)—/→D-register
	Clear condition code flip-flops	R/CC = FAFL PH4 + ...	
	Enable signal (S/SXB)	(S/SXB) = FAFL PH4 + ...	Preset logic for B—→S in PH5
	Branch to CPU PH5	S/PH5 = PH4 NBR	CPU enters PH5. Floating point box may go to PH5 or PH6
		R/PH5 = ...	
			Mnemonic: FMS (3F, BF) FML (1F, 9F)

(Continued)

Table 3-66. FMS, FML Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH4; Box PH4; T8L (Cont.)	If the multiplicand (D-register) and multiplier (A-register) are both simple-normalized, perform the following functions:	ASN = N(A47 A48 A49 A50 A51) N[NA47 A4851Z N(FNF NO6)] DSN = N(D47 D48 D49 D50 D51) N(ND47 ND4851Z)	
	Branch to Box PH7	S/NPH7 = N(S/PH7) (S/PH7) = PH4 O6 MUL ASN DSN + ... R/NPH7 = ...	
	Set SW2 if product will be negative	S/SW2 = (S/PH7) NPH7 MUL FPR + ... R/SW2 = ...	FPR set in PH2 if operand signs are not alike
	If the multiplicand is simple-normalized and multiplier is not, perform the following functions:		Prepare to normalize multiplier in PH6
	Enable signal (S/SXA) if multiplier positive	(S/SXA) = (S/SXAVA) NA47 + ... (S/SXAVA) = PH4 O6 DSN N(S/PH7) + ...	Preset adder to gate absolute value of multiplier to sum bus
	Enable signal (S/SXMA) if multiplier negative	(S/SXMA) = (S/SXAVA) A47 + ...	
	Branch to Box PH6	S/NPH6 = N(S/PH6) (S/PH6) = PH4 O6 DSN N(S/PH7) + ... R/NPH5 = ...	
	If the multiplicand is not simple-normalized, perform the following functions:		Prepare to normalize multiplicand in PH5
	Enable signal (S/SXD)	(S/SXD) = PH4 O6 NDSN + ...	Preset adder for D→S in PH5
	Set flip-flop SW2	S/SW2 = (S/SW2/1) + ... (S/SW2/1) = PH4 O6 NDSN + ... R/SW2 = ...	SW2 indicates that A↛D will be performed in PH5
	Branch to Box PH5	S/NPH5 = N(S/PH5) (S/PH5) = PH4 O6 NDSN + ... R/NPH5 = ...	
			Mnemonic: FMS (3F, BF) FML (1F, 9F)

(Continued)

Table 3-66. FMS, FML Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH5 or PH6; Box PH5; T8L	This phase is entered only if the multiplicand requires prenormalization. Phase is sustained until multiplicand is simple-normalized or found to be zero		
	Perform the following functions during the first clock period:		
	(A47-A71, A0-A31)—/→ (D47-D71, D0-D31)	DXA = PH5 SW2 + ...	Save multiplier (at clock) shift multiplicand left one hexadecimal for first normalization try
	(D47-D71, D0-D31)—→	Adder logic set at PH4 clock	
	(S47-S71, S0-S31) x 16—/→	AXSL4 = AXSL4/1	
	(A47-A71, A0-A27)	AXSL4/1 = PH5 O6 N(S/PH6) + ...	
	Decrement exponent of product in E-register by one	EDC7 = AXSL4/1 N(PH5 DIV) + ...	Exponent decremented to compensate for shift
	Set flip-flop RTZ if sum bus quantity is zero	S/RTZ = SZU SZL NSXADD NASPP PH5 + ...	If sum bus is zero, multiplicand is zero, and therefore product is zero. The multiplication in PH7 will be bypassed
		SZU = N(S47 + S48 + ... + S71)	
		SZL = N(S0 + S1 + ... + S31)	
		R/RTZ = PH1 + ASPP	
	Enable signal (S/SXA)	(S/SXA) = AXSL4/1 NFPRR + ...	Preset adder for A—→S in next clock period
		FPRR = PH5 O6 RTZ + ...	
	Perform the following functions during the second and following clock periods:		
	If multiplicand is zero (RTZ), enable signal FPRR and branch to PH9	FPRR = PH5 O6 RTZ + ...	
		S/PH9 = FPRR	
		R/PH9 = ...	
	If multiplicand is not zero and if it is not simple-normalized, shift the multiplicand toward normalization as follows:		
	(A47-A71, A0-A31)—→ (S47-S71, S0-S31) x 16—/→ (A47-A71, A0-A27)	Adder logic set at previous clock	Shift multiplicand left one hexadecimal for another attempt at normalization
		AXSL4 = (AXSL4/1)	
		(AXSL4/1) = PH5 O6 N(S/PH6) + ...	
		N(S/PH6) = PH5 O6 NASN + ...	

Mnemonic: FMS (3F, BF)
FML (1F, 9F)

(Continued)

Table 3-66. FMS, FML Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH5 or PH6; Box PH5; T8L (Cont.)	Decrement exponent of product in E-register by one	EDC7 = AXSL4/1 N(PH5 DIV) + ...	Exponent decremented to compensate for the shift
	Sustain PH5 until multiplicand is simple-normalized. When normalization occurs (second clock of PH5 or later) perform the following functions:	S/NPH5 = N(S/PH5) (S/PH5) = PH5 O6 N(S/PH6) NRTZ + ... N(S/PH6) = PH5 O6 NASN + ... R/NPH5 = ...	
	If multiplier (in D-register) is positive, enable signal (S/SXD)	(S/SXD) = (S/SXAVD) ND46 + ... (S/SXAVD) = PH5 O6 ASN NSW2 + ...	Preset adder to gate absolute value of multiplier to sum bus in PH6
	If multiplier is negative, enable signal (S/SXMD)	(S/SXMD) = (S/SXAVD) D46 + ...	
	Set flip-flop SW2	S/SW2 = (S/SW2/1) + ... (S/SW2/1) = PH5 O6 ASN NSW2 + ... R/SW2 = ...	SW2 indicates that A→D will be performed in PH6
	Branch to Box PH6	S/NPH6 = N(S/PH6) (S/PH6) = PH5 O6 ASN NSW2 + ... R/NPH6 = ...	
CPU PH5 or PH6; Box PH6; T8L	This phase is entered from PH5 (multiplicand was not originally normalized) or from PH4 (multiplicand normalized, multiplier not normalized). Phase is sustained until multiplier is simple-normalized or found to be zero Perform the following functions during the first clock period:		
	If entered from PH4, |(A47-A71, A0-A31)| → (S47-S71, S0-S31)	Adder logic set at PH4 clock	|Multiplier| → sum bus
	If entered from PH5, |(D47-D71, D0-D31)| → (S47-S71, S0-S31) and (A47-A71, A0-A31) → (D47-D71, D0-D31)	Adder logic set at PH5 clock DXA = PH6 SW2 + ...	|Multiplier| → sum bus Simple-normalized multiplicand→D-register
	(S47-S71, S0-S31) x 16→ (A47-A71, A0-A27)	AXSL4 = AXSL4/1 AXSL4/1 = PH6 O6 N(S/PH7) + ...	Shift |multiplier| left one hexadecimal for first try at normalization
			Mnemonic: FMS (3F, BF) FML (1F, 9F)

(Continued)

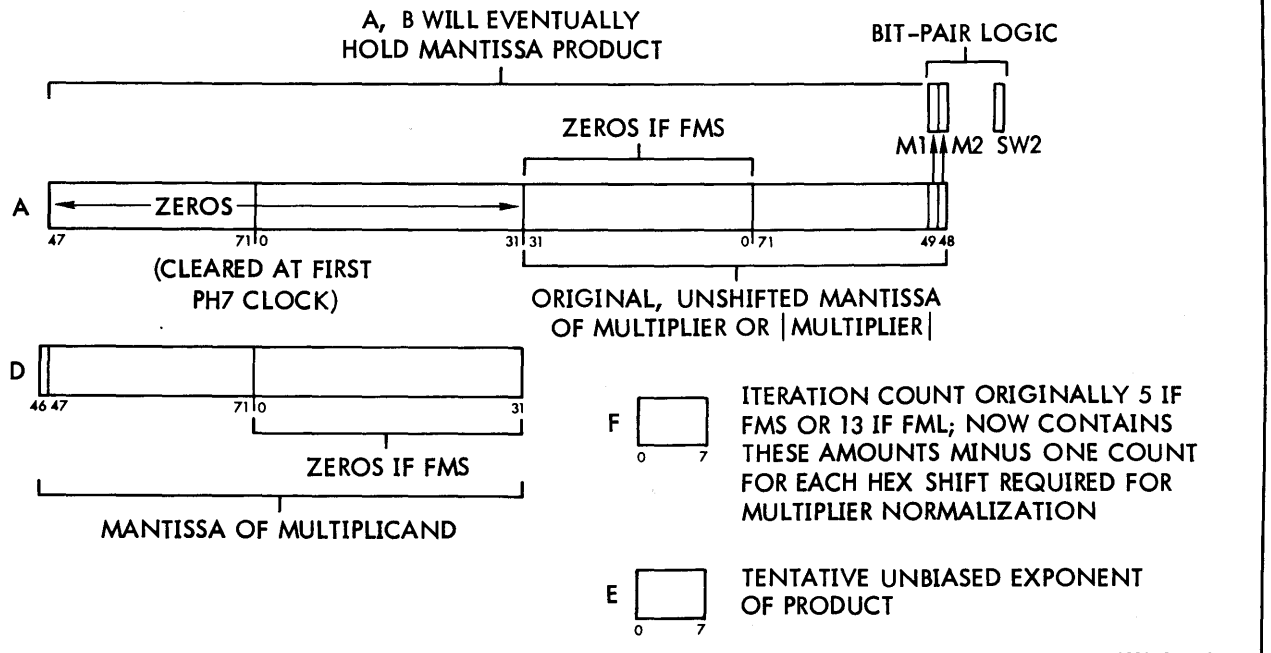
Table 3-66. FMS, FML Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH5 or PH6; Box PH6; T8L (Cont.)	Decrement exponent of product in E-register by one	EDC7 = AXSL4/1 N(PH5 DIV)	Exponent decremented to compensate for the shift
	Set flip-flop RTZ if sum bus quantity is zero	S/RTZ = SZU SZL NSXADD NASPP PH6 + ...	If sum bus is zero, multiplier is zero and therefore product is zero. The multiplication in PH7 will be bypassed
		SZU = N(S47 + S48 + ... + S71)	
		SZL = N(S0 + S1 + ... + S31)	
		R/RTZ = PH1 + ASPP	
	Enable signal (S/SXA)	(S/SXA) = AXSL4/1 NFPRR + ...	Preset adder for A⟶S in next clock period
		FPRR = PH6 NPH5 RTZ + ...	
	Decrement count in F-register by one	FDC7 = PH6 MUL N(S/PH7) + ...	F-register holds 5 (if FMS) or 13 (if FML). These counts represent iterations. Two iterations are deleted for every hexadecimal shift required to normalize multiplier
	Perform the following functions during the second and following clock periods:		
	If multiplier is zero (RTZ), enable signal FPRR, branch to PH9	FPRR = PH6 NPH5 RTZ + ...	
		S/PH9 = FPRR	
		R/PH9 = ...	
	If multiplier is not zero and if it is not simple-normalized, shift the multiplier toward normalization as follows:		
	Enable signal (S/SXA)	(S/SXA) = AXSL4/1 NFPRR + ...	Preset adder for A⟶S
	(A47-A71, A0-A31)⟶	Adder logic set at previous clock	Shift multiplier left one hexadecimal for another attempt at normalization
	(S47-S71, S0-S31) x 16⟶		
	(A47-A71, A0-A27)	AXSL4 = AXSL4/1	
		AXSL4/1 = PH6 O6 N(S/PH7) + ...	
		N(S/PH7) = PH6 O6 NASN + ...	
	Decrement exponent of product in E-register by one	EDC7 = AXSL4/1 N(PH5 DIV) + ...	Exponent decremented to compensate for the shift
	Decrement iteration count in F-register by one	FDC7 = PH6 MUL N(S/PH7) + ...	Delete number of iterations needed for multiplication by one
			Mnemonic: FMS (3F, BF) FML (1F, 9F)

(Continued)

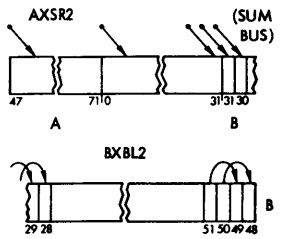
Table 3-66. FMS, FML Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH5 or PH6; Box PH6; T8L (Cont.)	Sustain PH6 until multiplier is simple-normalized. When simple-normalization occurs (second clock of PH6 or later), perform the following functions:	S/NPH6 = N(S/PH6) (S/PH6) = PH6 O6 N(S/PH7) NRTZ + ... N(S/PH7) = PH6 O6 NASN + ... R/NPH6 = ...	
	Set flip-flop SW2 if product will be negative	S/SW2 = (S/PH7) NPH7 MUL FPR + ... R/SW2 = ...	FPR set in PH2 if operand signs are not alike
	Branch to Box PH7	S/NPH7 = N(S/PH7) (S/PH7) = PH6 O6 ASN DSN + ... R/NPH7 = ...	Multiplication, iterations may proceed
CPU PH5 or PH6; Box PH7; T8L*	FMS: 14 clocks – two clocks for each multiplier normalization shift FML: 30 clocks – two clocks for each multiplier normalization shift Resume of register contents:		
	*If CPU accepts I/O service call, clocks to floating point box are rejected, as they are T5L	FPCLEN/1 = NIOEN NIOIN + NFPRR FPCLEN/2 = NT5EN N(S/T8L) = FAFL (IOACT + PH10) + ...	Floating point box continues operation after I/O service
			Mnemonic: FMS (3F, BF) FML (1F, 9F)



(Continued)

Table 3-66. FMS, FML Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH5 or PH6; Box PH7; T8L (Cont.)	Perform the following functions during the first clock period of PH7:		
	Enable signal MIT	MIT = PH7 MUL N(F0 SW0)	Sustain multiply iterations until final clock of PH7
	(S46-S71, S0-S31)→ (A47-A71, A0-A31), B31, B30	S/A47 = (S/A47/2) + ... (S/A47/2) = (G46 + PR46 NK46) BXBL2 + ... BXBL2 = MIT AXSR2 = MIT S/B30 = S31 BXBL2 + ... R/B30 = · BX S/B31 = S30 BXBL2 + ... R/B31 = BX	No gating term to sum bus enabled during first clock period of PH7, therefore A-register and B31, B30 are cleared
	(B31-B50)→ (B29-B48)	BXBL2 = MIT	 901172A.3962
	Enable M1 and M2 according to state of FPR flip-flop	M1 = B49 NFPR NF1 + NB49 FPR NF1 + ... M2 = B48 NFPR NF1 + NB48 FPR NF1 + ...	FPR is set if operand signs are unlike. M1 and M2 are enabled so that |product| is produced
	Set flip-flop SW0	S/SW0 = NSW0 BXBL2 + ... BXBL2 = MIT R/SW0 = MIT + ...	SW0 is true on all even-numbered clocks
	Preset adder according to bit pair in B49, B48, and state of SW2	(S/SXA) = MIT (NM1 NM2 NSW2 + M1 M2 SW2) (S/SXAMD) = (S/SXAMD/1) + ... (S/SXAMD/1)= MIT M1 (M2 ⊕ SW2) + ...	Preset adder for A→S in next clock period Preset adder for A - D →S in next clock period
			Mnemonic: FMS (3F, BF) FML (1F, 9F)

(Continued)

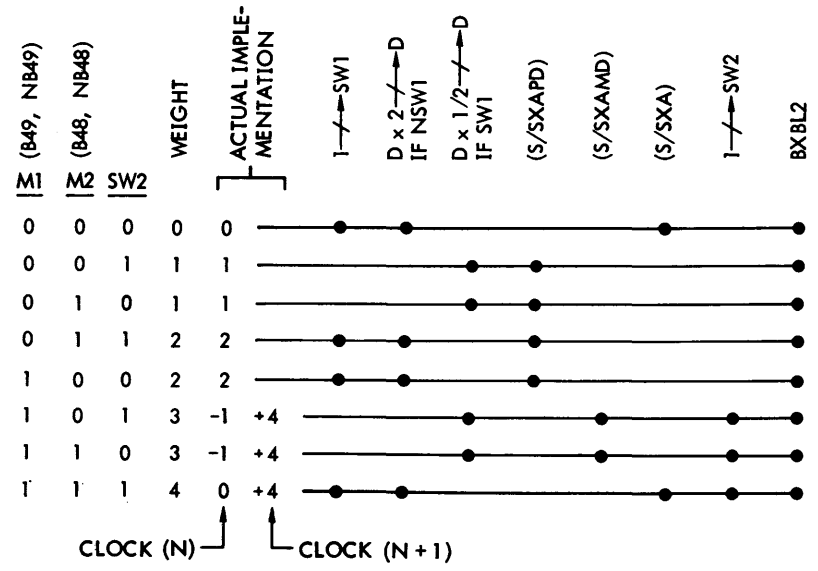
Table 3-66. FMS, FML Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH5 or PH6; Box PH7; T8L (Cont.)	Perform the following functions during the second and following clock periods of PH7 (except the last):		
	Enable signal MIT	MIT = PH7 MUL N(F0 SW0)	Sustain multiply iterations
	(A47-A71, A0-A31) ± (D46-D71, D0-D31) ⟶ (S47-S71, S0-S31)	Adder logic set at previous clock	Adder logic set by previous bit pair and SW2
	or		
	(A47-A71, A0-A31) ⟶ (S47-S71, S0-S31)		
	(S46-S71, S0-S31) ⟶ (A47-A71, A0-A31), B31, B30	Logic same as first clock period of PH7	Current partial sum shifted right two bit positions into A-register and B31, B30
	(B31-B50) ⟶ (B29-B48)		
	Enable M1 and M2		
	Toggle flip-flop SW0		
	Preset adder according to bit pair in B49, B48 and state of SW2		Preset adder for next partial sum
	Set flip-flop SW2 if bit-pair weight of 3 or 4		Add one to next bit pair
	If bit-pair logic calls for 2 x multiplicand or 1 x multiplicand, shift D-register accordingly		Preset partial product for addition to partial sum during next clock period
	Decrement iteration count in (F0-F7) by one at every even-numbered clock	FDC7 = PH6 MUL N(S/PH7) + ... (S/PH7) = MIT MIT = PH7 MUL N(F0 SW0)	
	Perform the following functions during the last clock period of PH7		
	Disable MIT and MIT functions	MIT = PH7 MUL N(F0 SW0)	F0 is a one because iteration count has been decremented below zero. SW0 is true during even clock periods
			Mnemonic: FMS (3F, BF) FML (1F, 9F)

(Continued)

Table 3-66. FMS, FML Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH5 or PH6; Box PH7; T8L (Cont.)		(S/SXAPD) = (S/SXAPD/1) + ... (S/SXAPD/1) = MIT N(SXAMD/1) N(S/SXA) + ...	Preset adder for A + D → S in next clock period
	Set flip-flop SW2 if bit-pair weight of 3 or 4	S/SW2 = MIT M1 N(S/SXAPD/1) MUL + ... R/SW2 = ...	Effectively adds one to next bit pair
	If bit-pair logic calls for 2 x multiplicand or 1 x multiplicand, shift D-register accordingly	DXDL1 = MIT N(M2 ⊕ SW2) NSW1 S/SW1 = MIT N(M2 ⊕ SW2) R/SW1 = NPH9 DXDR1 = MIT (M2 ⊕ SW2) SW1	Find 2 x multiplicand, providing 2 x multiplicand is not already in D-register (SW1 set) Restore original multiplicand in D-register, providing it is not already present (SW1 reset)
	Sustain PH7	S/NPH7 = N(S/PH7) (S/PH7) = MIT + ... R/NPH7 = ...	
	Summary of control signals during PH7:		



901172A.3963

Mnemonic: FMS (3F, BF)
FML (1F, 9F)

(Continued)

Table 3-66. FMS, FML Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH5 or PH6; Box PH7; T8L (Cont.)	(S46-S71, S0-S31) → / ← (A47-A71, A0-A31)	AXS = PH7 N(S/PH7) MUL + ... (S/PH7) = MIT	Last partial sum is on the sum bus. Adder logic preset in previous clock period
	Enable signal (S/SXA)	(S/SXA) = PH7 N(S/PH7) + ...	Preset adder for A→S in PH8
	Branch to Box PH8	S/NPH8 = N(S/PH8) (S/PH8) = PH7 N(S/PH7) + ... R/NPH8 = ...	
CPU PH6; Box PH8; T8L	One clock long A-register contains the absolute value of the result in the range $1/256 \leq \lvert result \rvert < 1/16$		
	If the result is not simple-normalized, perform the following functions: (A47-A71, A0-A31) (S47-S71, S0-S31) x 16 (A47-A71, A0-A27) (B31-B28) (A28-A31)	Adder logic set at PH7 clock AXSL4 = AXSL4/1 AXSL4/1 = PH8 NDIV NASN + ... S/A28 = B31 A2831XB + ... S/A31 = B28 A2831XB + ... A2831XB = PH8 MUL NASN R/A28-A31 = AX/L + ...	Shift A-register product to normalize it. Product in remainder of B-register will be lost
	Decrement exponent of product in E-register by one	EDC7 = AXSL4/1 N(PH5 DIV) + ...	Compensate for the shift
	Enable signal FPRR and perform functions listed at end of this phase. If the result is simple-normalized and A47 is a one, perform the following functions: (A47-A71, A0-A31) → (S47-S71, S0-S31) x 1/16 → / (A51-A71, A0-A31)	FPRR = PH8 NDIV NA5255Z + ... Adder logic set at PH7 clock AXSR4 = AXSR4/1 AXSR4/1 = PH8 NDIV A47 + ...	Prepare to send result to CPU |Result| equals 1 in this case, and must be shifted right to represent a legal floating point number
			Mnemonic: FMS (3F, BF) FML (1F, 9F)

(Continued)

Table 3-66. FMS, FML Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH6; Box PH8; T8L (Cont.)	Increment exponent of result by one	EUC7 = NPH5 AXSR4/1 + ...	Compensate for the shift
	Enable signal FPRR and perform functions listed at end of this phase	FPRR = PH8 NDIV ASN + ...	Prepare to send result to CPU
	If the result is simple-normalized and A47 is a zero, enable signal FPRR and perform functions listed at end of this phase	FPRR = PH8 NDIV ASN + ...	
	FPRR functions:		
	Enable (CPU) PH7	S/PH7 = PH6 NBR NIOEN + ... NBR = NBRPH6 ... NBRPH6 = N(FAFL PH6 NFPRR) + ... R/PH7 = ...	
	Set flip-flop MRQ	S/MRQ = (S/MRQ/1) + ... (S/MRQ/1) = FAFL PH6 NIOEN NBRPH6 + ... R/MRQ = ...	Request for next instruction in sequence
	Enable signal (S/SXA) if NFPR	(S/SXA) = FPRR NFPR + ...	Preset adder logic to give result the proper polarity
	Enable signal (S/SXMA) if FPR	(S/SXMA) = FPRR FPR + ...	
	Set Box PH9	S/PH9 = FPRR R/PH9 = ...	
CPU PH7; Box PH9; T8L	One clock long. Entered from PH5 if multiplicand is zero, from PH6 if multiplier is zero, or from PH8		
	(A47-A71, A0-A31)⟶ or -(A47-A71, A0-A31)⟶ (S47-S71, S0-S31)	Adder logic set at PH5, PH6, or PH8 clock	Mantissa of result, in proper polarity, transferred to sum bus
	Transfer (S0-S31) to (FP0-FP31) lines providing none of the following conditions are present:	FPXSL = NFPDIS PH9 FPRD NRTZ N(FEUF NFZ) + ...	LSW of floating point result
	FMS or FML with odd R field in effect	FPRD = NO2 + MUL NR31	
	Result is equal to zero	RTZ	
			Mnemonic: FMS (3F, BF) FML (1F, 9F)

(Continued)

Table 3-66. FMS, FML Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH7; Box PH9; T8L (Cont.)	Exponent underflow with FZ equal to zero	FEUF = E0 NE1 NRTZ N(B65 NO6 FS NFZ)	Exponent was decremented below zero
	If one of the above conditions exists, transfer zeros to (FP0-FP31)		No gating term enabled
	(FP0-FP31)→(B0-B31)	BXFP = FAFL PH7 + ...	LSW of floating point result
	Reset flip-flop NSXBF	S/NSXBF = N(S/SXB) (S/SXB) = FAFL PH7 + ... R/NSXBF = ...	Preset logic for B→S in PH8
	Force a one on private memory address line LR31	S/NLR31F = N(S/LR31) (S/LR31) = FAFL PH7 + ... R/NLR31F = ...	Select odd-numbered private memory address during PH10
	Set flip-flop RW if long format instruction and TRAP signal is not true	S/RW = (S/RW/FP) (S/RW/FP) = PH9 NTRAP FPRD + ... R/RW = ...	Prepare to send LSW of result to CPU
	Set flip-flop CC1 if exponent underflow has occurred and FZ is a one	S/CC1 = (S/CC1/3) + ... (S/CC1/3) = (S/CC1/FP) + ... (S/CC1/FP) = PH9 FEUF + ... R/CC1 = (F/CC1)	
	Set flip-flop CC2 if exponent underflow or overflow	S/CC2 = (S/CC2/3) + ... (S/CC2/3) = (S/CC2/FP) + ... (S/CC2/FP) = PH9 (FEUF + FEOF) + ... R/CC2 = (R/CC2) FEOF = NE0 E1 NRTZ	
	Enable TRAP signal if exponent underflow has occurred and FZ is a one, or if exponent overflow has occurred	TRAP = FEOF + FEUF FZ + ...	TRAP prevents RW from being set in PH9 and PH10
	Enable signal (S/SXA) if NFPR is true	(S/SXA) = PH9 NFPR + ...	Preset adder to give result the proper polarity
	Enable signal (S/SXMA) if FPR is true	(S/SXMA) = PH9 FPR + ...	
	If FPR is true, transfer (NE0-NE7) →(E0-E7)	EXNE = PH9 FPR NTRAP N(FEUF NFZ)	A negative result requires an inverted exponent
	Branch to Box PH10	S/PH10 = PH9 R/PH10 = ...	
			Mnemonic: FMS (3F, BF) FML (1F, 9F)

(Continued)

Table 3-66. FMS, FML Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH8; Box PH10; T8L	One clock long (B0-B31)⟶(S0-S31)⟶ (RW0-RW31)	Logic set at PH7 clock RWXS/0-RWXS/3 = RW ... RW = Set at PH7 clock, if no trap condition and long format	Transfer LSW of result to private memory register Ru1
	If LSW of result is not equal to zero, set flip-flop SW0	S/SW0 = NS0031Z (S/SW0/NZ) + ... (S/SW0/NZ) = FAFL NO2 PH8 R/SW0 = RESET/A + ...	Used in PH10 for condition code settings
	Reset flip-flop NSXBF	S/NSXBF = N(S/SXB) (S/SXB) = FAFL PH8 + ... R/NSXBF = ...	Preset logic for B⟶S in PH10
	Set flip-flop RW if TRAP signal is not true	S/RW = (S/RW/FP) (S/RW/FP) = PH10 NTRAP + ... R/RW = ...	Prepare to send MSW of result to CPU
	Set flip-flop DRQ	S/DRQ = BRPH10 + ... BRPH10 = FAFL PH8 + ... R/DRQ = ...	Inhibits transmission of another clock until data release received from core memory. Request for next instruction mode in PH6
	(A47-A71, A0-A31)⟶ or -(A47-A71, A0-A31)⟶ (S47-S71, S0-S31)	Adder logic set at PH9 clock	MSW of mantissa⟶ sum bus
	S47⟶FP0	FP0 = S47 FPXSU + ...	MSW of result transferred to FP lines if result not equal to zero or if underflow with FZ = 0 does not exist
		FPXSU = NFPDIS PH10 NRTZ N(FEUF NFZ)	
	NE1⟶FP1 (+64 bias)	FP1 = NE1 FPXSU + ...	
	(E2-E7)⟶(FP2-FP7)	FPXSU	
	(S48-S71)⟶(FP8-FP31)	FPXSU	
	(FP0-FP31)⟶(B0-B31)	BXFP = FAFL PH8 + ...	MSW of result⟶ B-register
	Reset Box PH10	R/PH10 = ...	Floating point box actions are finished
			Mnemonic: FMS (3F, BF) FML (1F, 9F)

(Continued)

Table 3-66. FMS, FML Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH8; Box PH10; T8L (Cont.)	Branch to CPU PH10	S/PH10 = BRPH10 NCLEAR + ... R/PH10 = ...	
CPU PH10; Box actions over; T8L	One clock long (B0-B31)⟶(S0-S31)⟶ (RW0-RW31)	Adder logic set at PH8 clock RWXS/0-RWXS/3 = RW ... RW = Set at PH8 clock if no trap condition	
	Set flip-flop CC3 if floating point result is positive	S/CC3 = SGTZ TESTS + ... SGTZ = (S0 + S1 + ... + S31 + SW0) NS0 TESTS = FAFL ENDE + ... R/CC3 = TESTS + ...	SW0 is set when there is significance in LSW of result
	Set flip-flop CC4 if floating point result is negative	S/CC4 = (S/CC4/2) TESTS + ... (S/CC4/2) = NFACOMP S0 + ...	
	ENDE functions	R/CC4 = TESTS + ...	
			Mnemonic: FMS (3F, BF) FML (1F, 9F)

FLOATING DIVIDE, SHORT (FDS; 3E, BE). FDS divides the contents of private memory register R by the effective word. If no floating point arithmetic occurs, the quotient is loaded into private memory register R.

FLOATING DIVIDE, LONG (FDL; 1E, 9E). FDL divides the contents of private memory registers R and Ru1 by the effective doubleword. R must be an even value for correct results. If no floating point arithmetic fault occurs, the quotient is loaded into private memory registers R and Ru1 as a long format floating point number.

FLOATING DIVIDE PHASE SEQUENCE. Preparation phases for FDS are the same as the general PREP phases for word instructions, paragraph 3-59. FDL preparation phases are described in paragraph 3-59. Figure 3-170 shows the general method of FDS and FDL execution. Nonrestoring division (described in paragraph 3-68) is used during the actual divide iterations. Table 3-67 lists the detailed logic for execution of the floating multiply instructions.

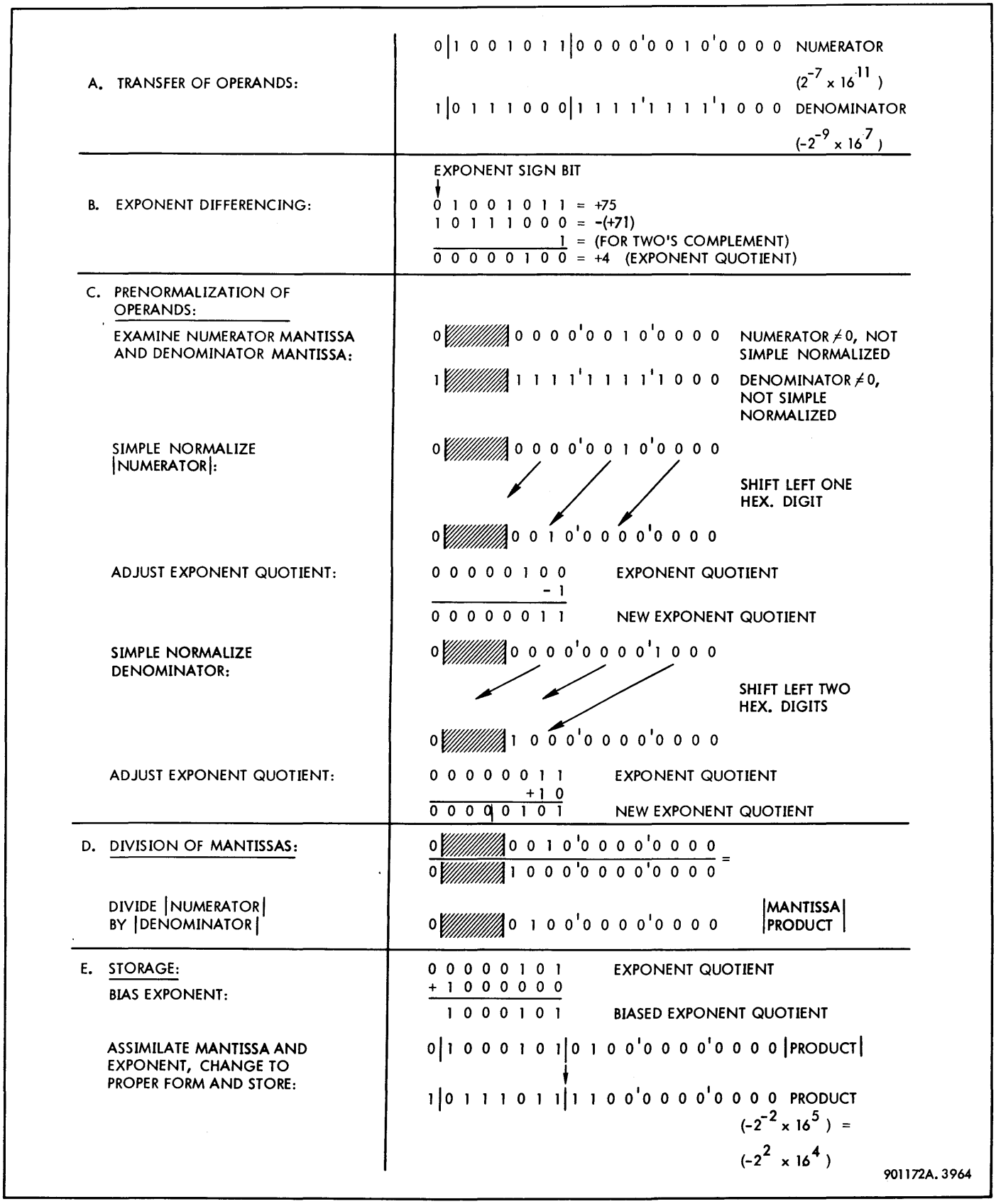


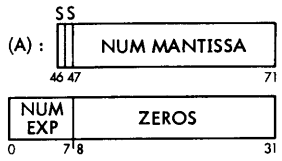
Figure 3-170. Floating Divide Implementation

Table 3-67. FDS, FDL Sequence

Phase	Function Performed	Signals Involved	Comments
		Note Actions that take place in the floating point box are underscored in the sequence charts for the floating point instructions. Main CPU functions are not underscored.	
PREP	At end of PREP:		
	(A) : RR		Contents of private memory register R. MSW of numerator
	(C) : Core memory operand MSW		MSW of denominator
	(D) : Core memory operand MSW		MSW of denominator
	Enable signal (S/SXNA)	(S/SXNA) = FAFL PRE/34 + ...	Preset adder for -A→S in PH1
	If long-format instruction is in effect perform the following functions:		
	Force a one into P31	PUC31 = FAFL NO2 PRE3 NANLZ + ...	Prepare to obtain LSW of denominator
	Set flip-flop MRQ	S/MRQ = (S/MRQ/1) + ... (S/MRQ/1) = FAFL NO2 NANLZ PRE3 + ... R/MRQ = ...	Memory request for LSW of denominator. Inhibited if floating point option trap exists
	Enable clock T8	S/NT8L = N(S/T8L) (S/T8L) = FAFL NIOACT NPH10 R/NT8L = ...	Clocks for remainder of floating point phases are T8 unless I/O service call is in effect (PH6)
	FPCON→floating point box	FPCON = FAFL PRE3 + ...	Start functions in floating point box
	Set flip-flop PH1	S/PH1 = FPCON NPH1 R/PH1 = ...	Sets Box PH1
CPU PH1; Box PH1; T8L	One clock long		
	(NA0-NA31)→(S0-S31)	Adder logic set at PH1 clock	Gate MSW of numerator to FP lines
	(NS0-NS31)→(FP0-FP31)	FPXS = NPH8 NDIS	
	FP0→S46, S47	SXFP/U = S4607XFP S4607XFP = PH1 NFPDIS + ...	Sign of numerator→S46, S47
	(FP8-FP31)→(S48-S71)	SXFP/U = S4607XFP	Mantissa of numerator→(S48-S71)
			Mnemonic: FDS (3E, BE) FDL (1E, 9E)

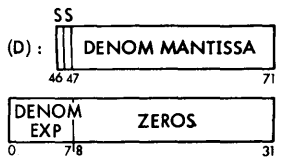
(Continued)

Table 3-67. FDS, FDL Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH1; Box PH1; T8L (Cont.)	(FP0-FP7)⟶(S0-S7)	SXFP/4 = S4607XFP	Exponent of numerator ⟶(S0-S7)
	Zeros⟶(S8-S31)⟶		No gating term enabled
	(S46-S71, S0-S31)⟶	AXS = PH1 + ...	
	(A46-A71, A0-A31)		
	Clear B-register	BX = PH1 + ...	901172A.3965
	Clear E-register	EX = PH1 + ...	
	Clear F-register	FX = PH1 + ...	
	D0⟶FPCON⟶floating-point box	FPCON = FAFL PH1 D0 + ...	Transfer sign of denominator to MWN in floating point box
	FPCON⟶/⟶MWN	S/MWN = FPCON PH1	
		R/MWN = PH1	
	If FDL is in effect, perform the following functions:		
	Force a one on private memory address line LR31	(S/LR31) = FAFL NO2 PH1 + ...	Prepare to obtain LSW of numerator
	Reset flip-flop NAXRR	S/NAXRR = N(S/AXRR)	Preset logic for RRu1 ⟶/⟶A in PH2
		(S/AXRR) = FAFL NO2 PH1 + ...	
		R/NAXRR = ...	
	Enable signal (S/SXND)	(S/SXND) = FAFL PH1 + ...	Preset adder for ND⟶S in PH2
	Set flip-flop PH2	S/PH2 = PH1	Box PH2
		R/PH2 = ...	
CPU PH2; Box PH2; T8L	One clock long		
	(ND0-ND31)⟶(S0-S31)	Adder logic set at PH1 clock	MSW of denominator ⟶FP lines
	(NS0-NS31)⟶(FP0-FP31)	FPXS = NPH8 NDIS	
	FP0⟶S46, S47	SXFP/U = S4607XFP	Sign of denominator ⟶S46, S47
		S4607XFP = PH2 NFPDIS + ...	
	(FP8-FP31)⟶(S48-S71)	SXFP/U = S4607XFP	Mantissa of denominator ⟶(S48-S71)
			Mnemonic: FDS (3E, BE) FDL (1E, 9E)

(Continued)

Table 3-67. FDS, FDL Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH2; Box PH2; T8L (Cont.)	(FP0-FP7)⟶(S0-S7)	SXFP/U = S4607XFP	Exponent of denominator ⟶(S0-S7)
	Zeros⟶(S8-S31)		No gating term enabled
	(S46-S71, S0-S31)⟶/⟶ (D46-D71, D0-D31)	DXS = PH2 + ...	 901172A.3966
	Set flip-flop FPR if operand signs are unlike	S/FPR = PH2 O6 (MWN⊕A47) + ... R/FPR = PH1 + ...	Signifies that intermediate result will be opposite in polarity to final result
	If numerator is negative, (NA0-NA7)⟶/⟶(A0-A7)	S/A0 = ... S/A1 = NA1 PH2 A0 + ... ⋮ ⋮ S/A7 = NA7 PH2 A0 + ... R/A0-R/A7 = AX/L AX/L = AXL + ... AXL = PH2 A0 + ...	Uninverted exponent ⟶(A0-A7)
	Set flip-flop A8	S/A8 = PH2 NMUL + ... R/A8 = AX/L	For PH3 use
	Set flip-flop D8 if denominator is negative	S/D8 = PH2 MWN + ... R/D8 = DX/L DX/L = DX + ... DX = PH2 + ...	For PH3 use (for K7)
	Enable signal (S/SXAPD) if denominator is negative	(S/SXAPD) = (S/SXAPD/1) + ... (S/SXAPD/1) = PH2 NMUL MWN + ...	For exponent arithmetic in PH3
	Enable signal (S/SXAMD) if denominator is positive	(S/SXAMD) = N(S/SXAPD) (S/SXAMD/2) + ... (S/SXAMD/2) = PH2 + . . .	For exponent arithmetic in PH3
	If FML is in effect, perform the following functions:		
	(RR0-RR31)⟶/⟶(A0-A31)	AXRR = Set at PH1 clock	LSW of numerator ⟶/⟶ A-register
			Mnemonic: FDS (3E, BE) FDL (1E, 9E)

(Continued)

Table 3-67. FDS, FDL Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH2; Box PH2; T8L (Cont.)	Enable signal (S/SXNA)	(S/SXNA) = FAFL PH2 + ...	Preset adder for -A ⟶ S in PH3
	Set flip-flop DRQ	S/DRQ = (S/DRQ/2) + ... (S/DRQ/2) = FAFL NO2 PH2 + ... R/DRQ = ...	Inhibits transmission of another clock until data release received from core memory (memory request made during PREP)
	Set flip-flop PH3	S/PH3 = PH2 R/PH3 = ...	Box PH3
CPU PH3; Box PH3; T8L if short, DR if long	One clock long		
	(A0-A7) ± (D0-D7) ⟶ (S0-S7) ⟶/⟶ (E0-E7)	Adder logic set at PH2 clock S/E0 = S0 PH3 + ... ⋮ ⋮ S/E7 = S7 PH3 + ... R/E0-R/E7 = PH1 + ...	Arithmetic operation is performed that subtracts the uninverted denominator exponent from the uninverted numerator exponent. The E-register now holds the unbiased difference of the exponents
	(NA0-NA31) ⟶ (S0-S31) (NS0-NS31) ⟶ (FP0-FP31)	Adder logic set at PH2 clock FPXS = NPH8 NDIS	LSW of numerator ⟶ FP lines if FDL. If FDS, action is meaningless
	If FDL is being performed: (FP0-FP31) ⟶/⟶ (A0-A31)	AXFP = PH3 NO2	LSW of numerator ⟶/⟶ A-register
	If FDS is being performed: Zeros ⟶/⟶ (A0-A31)		No gating term enabled
	(MB0-MB31) ⟶ (C0-C31) ⟶/⟶ (D0-D31) if FDL	DXC = FAFL NO2 PH3 + ...	LSW of denominator ⟶ C- and D-registers
	Enable signal (S/SXND)	(S/SXND) = FAFL PH3 + ...	Preset adder for -D ⟶ S in PH4
	Set flip-flop PH4	S/PH4 = PH3 R/PH4 = ...	Box PH4
CPU PH4; Box PH4; T8L	One clock long		
	(ND0-ND31) ⟶ (S0-S31)	Adder logic set at PH3 clock	LSW of denominator if FDL. Meaningless if FDS
	(NS0-NS31) ⟶ (FP0-FP31)	FPXS = NPH8 NDIS	
			Mnemonic: FDS (3E, BE) FDL (1E, 9E)

(Continued)

Table 3-67. FDS, FDL Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH4; Box PH4; T8L (Cont.)	If FDL:		
	(FP0-FP31) ⟶ (S0-S31)	SXFP/4 = S0031XFP + ...	
		SXFP/A = S0031XFP + ...	
		S0031XFP = PH4 NO2 NFPDIS	
	If FMS:		
	Zeros ⟶ (S0-S31)		No gating term enabled
	(S0-S31) ⟶/ (D0-D31)	DXS/L = PH4 + ...	LSW of denominator (if FDL) or zeros (if FDS) ⟶/ D-register
	Clear condition code flip-flops	R/CC = FAFL PH4 + ...	
	Enable signal (S/SXB)	(S/SXB) = FAFL PH4 + ...	Preset logic for B ⟶ S in PH5
	Branch to CPU PH5	S/PH5 = PH4 NBR	CPU enters PH5. Floating point box may go to PH5 or PH6
		R/PH5 = ...	
	If the denominator is simple-normalized, perform the following functions:	DSN = N(D47 D48 D49 D50 D51) N(ND47 ND4851Z)	
	Enable signal (S/SXA) if denominator positive	(S/SXA) = (S/SXAVA) NA47 + ...	Preset adder to gate absolute value of denominator to sum bus
		(S/SXAVA) = PH4 O6 DSN N(S/PH7) + ...	
	Enable signal (S/SXMA) if denominator negative	(S/SXMA) = (S/SXAVA) A47 + ...	
	Branch to Box PH6	S/NPH6 = N(S/PH6)	
		(S/PH6) = PH4 O6 DSN N(S/PH7) + ...	
		R/NPH6 = ...	
	If the denominator is not simple-normalized, perform the following functions:		Prepare to normalize denominator in PH5
	Enable signal (S/SXD)	(S/SXD) = PH4 O6 NDSN + ...	Preset adder for D ⟶ S in PH5
	Set flip-flop SW2	S/SW2 = (S/SW2/1) + ...	SW2 indicates that A ⟶/ D will be performed in PH5
		(S/SW2/1) = PH4 O6 NDSN + ...	
		R/SW2 = ...	
	Branch to Box PH5	S/NPH5 = N(S/PH5)	
		(S/PH5) = PH4 O6 NDSN + ...	
		R/NPH5 = ...	
			Mnemonic: FDS (3E, BE) FDL (1E, 9E)

(Continued)

Table 3-67. FDS, FDL Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH5 or PH6; Box PH5; T8L	This phase is entered only if the denominator requires prenormalization. Phase is sustained until denominator is simple-normalized or found to be zero		
	Perform the following functions during the first clock period:		
	(A47-A71, A0-A31) → / → (D47-D71, D0-D31)	DXA = PH5 SW2 + ...	Save numerator (at clock)
	(D47-D71, D0-D31) →	Adder logic set at PH4 clock	Shift denominator left one hexadecimal for first normalization try
	(S47-S71, S0-S31) x 16 → / → (A47-A71, A0-A27)	AXSL4 = AXSL4/1 AXSL4/1 = PH5 O6 N(S/PH6) + ...	
	Increment exponent of quotient in E-register by one	EUC7 = PH5 DIV N(S/PH6) + ...	Exponent incremented to compensate for shift
	Set flip-flop RTZ if sum bus quantity is zero	S/RTZ = SZU SZL NSXADD NASPP PH5 + ... SZU = N(S47 + S48 + ... + S71) SZL = N(S0 + S1 + ... + S31) R/RTZ = PH1 + ASPP	If sum bus is zero, denominator is zero, and division is not allowed. A trap will result
	Enable signal (S/SXA)	(S/SXA) = AXSL4/1 NFPRR + ... FPRR = PH5 O6 RTZ + ...	Preset adder for A → S in next clock period
	Perform the following functions during the second and following clock periods:		
	If denominator is zero (RTZ), enable signal FPRR, transfer B-register → sum bus → A-register, branch to PH9	FPRR = PH5 O6 RTZ + ... SXB = FPRR DIV NSDIS + ... AXS = FPRR DIV + ... S/PH9 = FPRR R/PH9 = ...	B-register (zeros) → / → A-register
	If denominator is not zero and if it is not simple-normalized, shift the multiplicand towards normalization as follows:		
	(A47-A71, A0-A31) → (S47-S71, S0-S31) x 16 → / → (A47-A71, A0-A27)	Adder logic set at previous clock AXSL4 = (AXSL4/1) (AXSL4/1) = PH5 O6 N(S/PH6) + ...	Shift denominator left one hexadecimal for another attempt at normalization
			Mnemonic: FDS (3E, BE) FDL (1E, 9E)

(Continued)

Table 3-67. FDS, FDL Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH5 or PH6; Box PH5; T8L (Cont.)		N(S/PH6) = PH5 O6 NASN + ...	
	Increment exponent of product in E-register by one	EUC7 = PH5 DIV N(S/PH6) + ...	Exponent incremented to compensate for shift
	Sustain PH5 until denominator is simple-normalized. When normalization occurs (second clock of PH5, or later), perform the following functions:	S/NPH5 = N(S/PH5) (S/PH5) = PH5 O6 N(S/PH6) NRTZ + ... N(S/PH6) = PH5 O6 NASN + ... R/NPH5 = ...	
	If numerator (in D-register) is positive, enable signal (S/SXD)	(S/SXD) = (S/SXAVD) ND46 + ... (S/SXAVD) = PH5 O6 ASN NSW2 + ...	Preset adder to gate absolute value of numerator to sum bus in PH6
	If numerator is negative, enable signal (S/SXMD)	(S/SXMD) = (S/SXAVD) D46 + ...	SW2 indicates that A ⟶ D will be performed in PH6
	Set flip-flop SW2	S/SW2 = (S/SW2/1) + ... (S/SW2/1) = PH5 O6 ASN NSW2 + ... R/SW2 = ...	
	Branch to Box PH6	S/NPH6 = N(S/PH6) (S/PH6) = PH5 O6 ASN NSW2 + ... R/NPH6 = ...	
CPU PH5 or PH6; Box PH6; T8L	This phase is entered from PH4 (denominator was simple-normalized) or from PH5 (denominator was not originally simple-normalized, but has been). Phase is sustained until numerator is simple-normalized or found to be zero Perform the following functions during the first clock period:		
	If entered from PH4, |(A47-A71, A0-A31)| ⟶ (S47-S71, S0-S31)	Adder logic set at PH4 clock	|Numerator| ⟶ sum bus
			Mnemonic: FDS (3E, BE) FDL (1E, 9E)

(Continued)

Table 3-67. FDS, FDL Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH5 or PH6; Box PH6; T8L (Cont.)	If entered from PH5, |(D47-D71, D0-D31)|⟶ (S47-S71, S0-S31) and (A47-A71, A0-A31) ⟋⟶ (D47-D71, D0-D31)	Adder logic set at PH5 clock DXA = PH6 SW2 + ...	|Numerator|⟶sum bus Simple-normalized denominator⟋⟶D-register
	If numerator is not simple-normalized: (S47-S71, S0-S31) x 16 ⟋⟶ (A47-A71, A0-A27)	AXSL4 = AXSL4/1 AXSL4/1 = PH6 O6 N(S/PH7) + ... N(S/PH7) = PH6 O6 ASN DSN + ...	
	Decrement exponent of quotient in E-register by one	EDC7 = AXSL4/1 N(PH5 DIV) + ...	Exponent decremented to compensate for the shift
	Set flip-flop RTZ if sum bus quantity is zero	S/RTZ = SZU SZL NSXADD NASPP PH6 + ... SZU = N(S47 + S48 + ... + S71) SZL = N(S0 + S1 + ... + S31) R/RTZ = PH1 + ASPP	If sum bus is zero, numerator is zero and therefore quotient is zero. The division in PH8 will be bypassed
	Enable signal (S/SXA)	(S/SXA) = AXSL4/1 NFPRR + ... FPRR = PH6 NPH5 RTZ	Preset adder for A⟶S in next clock period
	If numerator is simple-normalized: (S47-S71, S0-S31) ⟋⟶ (A47-A71, A0-A31)	AXS = PH6 O6 ASN DSN + ...	|Numerator|⟋⟶A-register
	Branch to Box PH7	S/NPH7 = N(S/PH7) (S/PH7) = PH6 O6 ASN DSN + ... R/NPH7 = ...	Both numerator and denominator have now been simple-normalized
	Perform the following functions during the second and following clock periods:		
	If numerator is zero (RTZ), enable signal FPRR, (B47-B71, B0-B31) ⟶(S47-S71, S0-S31)⟋⟶ (A47-A71, A0-A31), branch to PH9	FPRR = PH6 NPH5 RTZ + ... SXB = FPRR DIV NSDIS + ... AXS = FPRR DIV + ... S/PH9 = FPRR R/PH9 = ...	B-register (zeros) ⟋⟶ A-register
			Mnemonic: FDS(3E, BE) FDL (1E, 9E)

(Continued)

Table 3-67. FDS, FDL Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH5 or PH6; Box PH6; T8L (Cont.)	If numerator is not zero and not simple-normalized, shift the numerator towards normalization as follows:		
	Enable signal (S/SXA)	(S/SXA) = AXSL4/1 NFPRR + ...	Preset adder for A ⟶ S
	(A47-A71, A0-A31) ⟶	Adder logic set at previous clock	Shift numerator left one hexadecimal for another attempt at normalization
	(S47-S71, S0-S31) x 16 ⟶	AXSL4 = AXSL4/1	
	(A47-A71, A0-A27)	AXSL4/1 = PH6 O6 N(S/PH7) + ...	
		N(S/PH7) = PH6 O6 NASN + ...	
	Decrement exponent of quotient in E-register by one	EDC7 = AXSL4/1 N(PH5 DIV) + ...	Exponent decremented by one to compensate for the shift
	Sustain PH6 until numerator is simple-normalized. When simple-normalization occurs (first clock of PH6, or later), perform the following functions:	S/NPH6 = N(S/PH6)	
		(S/PH6) = PH6 O6 N(S/PH7) NRTZ + ...	
		N(S/PH7) = PH6 O6 NASN + ...	
		R/NPH6 = ...	
	(S47-S71, S0-S31) ⟶	AXS = PH6 O6 ASN DSN + ...	|Numerator| ⟶ A-register
	(A47-A71, A0-A31)		
	Branch to Box PH7	S/NPH7 = N(S/PH7)	Both numerator and denominator have now been simple-normalized
		(S/PH7) = PH6 O6 ASN DSN + ...	
		R/NPH7 = ...	
CPU PH6; Box PH7; T8L*	One or two clocks long		
	Perform the following functions if A47 (numerator sign bit) is a one:		Since A-register contains |numerator|, A47=1 means that right-shift must be made. This case can only occur if original numerator was -1 or -1/16. A-register now holds +1
	Set flip-flop A51	S/A51 = PH7 DIV A47 + ...	Effectively shifts |numerator| right one hexadecimal
	Clear A-register	R/A51 = AX	
		AX = PH7 DIV A47 + ...	
	*If CPU accepts I/O service call, clocks to floating point box are rejected, as they are T5L	FPCLEN/1 = NIOEN NIOIN + NFPRR	Floating point box continues operation after I/O service
		FPCLEN/2 = NT5EN	
		N(S/T8L) = FAFL (IOACT + PH10) + ...	
			Mnemonic: FDS (3E, BE) FDL (1E, 9E)

(Continued)

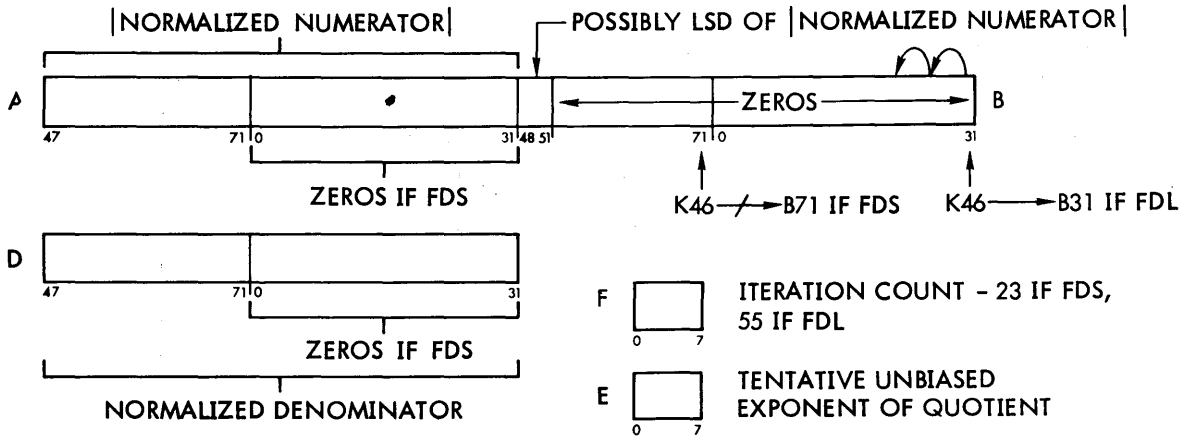
Table 3-67. FDS, FDL Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH6; Box PH7; T8L (Cont.)	Increment exponent of quotient in E-register by one	EUC7 = PH7 DIV A47 + ...	To compensate for right shift
	Sustain PH7 for one more clock	S/NPH7 = N(S/PH7) (S/PH7) = PH7 DIV A47 + ... R/NPH7 = ...	
	Preset adder for A - |D| ⟶ S in PH8	(S/SXAPD) = (S/SXAPD/1) NDIT N(PH6 NO6) + ... (S/SXAPD/1) = MWN DIV (S/PH7) + ...	If denominator is negative, preset adder for A + D ⟶ S in PH8
		(S/SXAMD) = (S/SXAMD/1) + ... (S/SXAMD/1)= NMWN DIV (S/PH7) + ...	If denominator is positive, preset adder for A - D ⟶ S in PH8
	Perform the following functions if A47 is a zero (first or second clock period of PH7):		If A47 is a zero, the numerator is less than +1
	Enable signal DPP	DPP = PH7 DIV NA47	Divide preparation signal
	If FDS is being performed, set F-register to 23_{10}	S/F3 = DPP + ... S/F5 = DPP + ... S/F6 = DPP + ... S/F7 = DPP + ...	For divide iteration counting
	If FDL is being performed, set F-register to 55_{10}	S/F2 = DPP NO2 + ... R/F2-F7 = FX + ...	
	Enable signal (S/SXA)	(S/SXA) = PH7 N(S/PH7) + ... N(S/PH7) = PH7 DIV NA47 + ...	
	Set flip-flop SW1 if |numerator| ≥ |denominator|	S/SW1 = K46 DPP DIV + ... R/SW1 = NPH9	K46 is true only if this condition exists
	Branch to Box PH8	S/PH8 = PH7 N(S/PH7) + ... R/PH8 = ...	
			Mnemonic: FDS (3E, BE) FDL (1E, 9E)

(Continued)

Table 3-67. FDS, FDL Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH6; Box PH8, SW1; T8L	PH8, SW1. This subphase is entered only if the absolute value of numerator is greater than or equal to the absolute value of the denominator. One clock long if entered		
	(A47-A71, A0-A31) ⟶	Adder logic set at last PH7 clock	Shift |numerator| so that it is smaller than |denominator| in preparation for division operation
	(S47-S71, S0-S31) x 1/16 ⟶	AXSR4 = AXSR4/1	
	(A51-S71, A0-A31), (B48-B51)	AXSR4/1 = PH8 SW1 DIV + ...	
		S/B48 = (S/B48/1) + ...	
		⋮ ⋮	
		S/B51 = (S/B51/1) + ...	
		(S/B48/1) = S28 PH8 DIV SW1 + ...	
		⋮ ⋮	
		(S/B51/1) = S31 PH8 DIV SW1 + ...	
		R/B48-B51 = BXFP + ...	
	Increment exponent of quotient in E-register by one	EUC7 = AXSR4/1 NPH5 + ...	To compensate for right shift
	Reset flip-flop SW1	R/SW1 = NPH9	
	Enable signal (S/SXA)	(S/SXA) = PH8 DIV SW1 + ...	Preset adder for A ⟶ S in PH8. NSW1
	Sustain Box PH8	S/NPH8 = N(S/PH8) (S/PH8) = PH8 NFPRR + ... R/NPH8 = ...	



901172A.3967

Mnemonic: FDS (3E, BE)
FDL (1E, 9E)

(Continued)

Table 3-67. FDS, FDL Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH6; Box PH8, SW1; T8L (Cont.)	Perform the following functions during the first clock period of PH8:		
	Enable signal DIT	DIT = PH8 DIV NSW1 NF0	High during all clocks except the final two
	Enable signal DIT/1	DIT/1 = PH8 DIV NSW1 NFPRR	High during all clocks except the final
	(A47-A71, A0-A31) ⟶ (S47-S71, S0-S31)	Adder logic set at previous clock	First iteration amounts to shifting numerator right one bit position so that first quotient bit produced is 2^{-1} bit. A 2^0 quotient bit is produced in this clock period but since the quotient is the absolute value of the actual quotient it is a zero and does not need to be clocked into B (B contains all zeros)
	(S48-S71, S0-S31) x 2 ⟶ (A47-A71, A0-A30)	AXSL1 = DIT/1	
	(B48 ⟶ A31)	S/A31 = AXSL1 B48 + ...	
	(B49-B31) ⟶ (B48-B30)	R/A31 = PH6 NO6 + ...	
		BXBL1 = DIT/1 + ...	
	Set flip-flop SW0 if FDL	S/SW0 = BXBL1 NO2 + ...	
		R/SW0 = DIT/1 + ...	
	If denominator is negative, preset adder for A + D ⟶ S in second clock period of PH6	(S/SXAPD) = (S/SXAPD/2) + ...	First subtraction is |numerator|–|denominator|
		(S/SXAPD/2) = DIT MWN ...	
	If denominator is positive, preset adder for A - D ⟶ S in second clock period of PH6	(S/SXAMD) = N(S/SXAPD) (S/SXAMD/2) + ...	
		(S/SXAMD/2) = DIT + ...	
	Decrement iteration count in F-register by one	FDC7 = DIT/1 + ...	
	Sustain PH8	S/NPH8 = N(S/PH8)	
		(S/PH8) = PH8 NFPRR + ...	
		R/NPH8 = ...	
			Mnemonic: FDS (3E, BE) FDL (1E, 9E)

(Continued)

Table 3-67. FDS, FDL Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH6; Box PH8, SW1; T8L (Cont.)	Perform the following functions during the second and following clock periods of PH8, except the last two		
	Enable signal DIT Enable signal DIT/1 (except second to last and last clock periods) (A47-A71, A0-A31)⟶ (S47-S71, S0-S31) (S47-S71, S0-S31) x 2⟶ (A47-A71, A0-A31) B48⟶A31 (B49-B31)⟶(B48-B30)	Logic same as PH8, first clock period	Residue of original numerator and quotient bits shift to the right as iterations progress
	Quotient bit⟶B31 if FDL or Quotient bit⟶B71 if FDS	S/B31 = BXBL1 K46 SW0 R/B31 = DIT/1 + ... S/B71 = BXBL1 K46 NSW0 R/B71 = DIT/1 + ...	K46 is a one if residue in A-register goes positive, signifying that divisor (denominator) could be successfully subtracted from residue (remainder)
	Set flip-flop SW0 if FDL	S/SW0 = BXBL1 NO2 + ... R/SW0 = DIT/1 + ...	
	If (K46 ⊕ SXADD ⊕ MWN), preset adder for A + D⟶S in next clock period If N(K46 ⊕ SXADD ⊕ MWN), preset adder for A - D⟶S in next clock period	(S/SXAPD) = (S/SXAPD/2) + ... (S/SXAPD/2) = DIT (MWN ⊕ SXADD ⊕ K46) (S/SXAMD) = N(S/SXAPD) (S/SXAMD/2) + ... (S/SXAMD/2) = DIT + ...	Next iteration will combine residue and divisor so that divisor has opposite polarity residue, bringing residue toward zero
	Decrement iteration count in F-register by one	FDC7 = DIT/1 + ...	
	Sustain PH8	S/NPH8 = N(S/PH8) (S/PH8) = PH8 NFPRR + ... R/NPH8 = ...	
			Mnemonic: FDS (3E, BE) FDL (1E, 9E)

(Continued)

Table 3-67. FDS, FDL Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH6; Box PH8, SW1; T8L (Cont.)	Perform the following functions during the second-to-last clock period of PH8:		
	All of the above functions execpt (S/SXAPD) or (S/SXAMD)	DIT = PH8 DIV NSW1 NF0	F0 is equal to a one at this time, and disables signal DIT
	Perform the following functions during the last clock period of PH8:		
	Enable signal FPRR	FPRR = PH8 DIV F0 NF7 + ...	Floating point result ready. B-register now contains |quotient| in range $1/16 \leq \|Q\| < 1$
	Enable (CPU) PH7	S/PH7 = PH6 NBR NIOEN + ... NBR = NBRPH6 ... NBRPH6 = N(FAFL PH6 NFPRR) + ... R/PH7 = ...	
	Set flip-flop MRQ	S/MRQ = (S/MRQ/1) + ... (S/MRQ/1) = FAFL PH6 NIOEN NBRPH6 + ... R/MRQ = ...	Request for next instruction in sequence
	(B47-B71, B0-B31) ⟶ (S47-S71, S0-S31) ⟶ (A47-A71, A0-A31)	SXB = FPRR DIV NSDIS + ... AXS = FPRR DIV + ...	|Quotient| ⟶ A-register
	Enable signal (S/SXA) if NFPR Enable signal (S/SXMA) if FPR	(S/SXA) = FPRR NFPR + ... (S/SXMA) = FPRR FPR + ...	Preset adder logic to give result the proper polarity
	Set Box PH9	S/PH9 = FPRR R/PH9 = ...	
CPU PH7; Box PH9; T8L	One clock long. Entered from PH5 if denominator is zero, from PH6 if numerator is zero, or from PH8		

Mnemonic: FDS (3E, BE)
FDL (1E, 9E)

(Continued)

Table 3-67. FDS, FDL Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH7; Box PH9; T8L (Cont.)	(A47-A71, A0-A31)⟶ or -(A47-A71, A0-A31)⟶ (S47-S71, S0-S31)	(S/SXA) = FPRR NFPR or (S/SXMA) = FPRR FPR } Set at previous clock	Mantissa of quotient, in proper polarity, transferred to sum bus
	Transfer (S0-S31) to (FP0-FP31) lines, providing none of the following conditions are present:	FPXSL = PH9 NFPDIS FPRD NRTZ N(FEUF NFZ) + ...	LSW of floating point result
	FDS in effect	NFPRD = O2 + ...	
	Numerator or denominator was equal to zero	RTZ (set in PH5 or PH6)	
	Exponent underflow with FZ equal to zero	FEUF = E0 NE1 NRTZ N(B65 NO6 FS NFZ)	Exponent was decremented below zero
	If one of the above conditions exists, transfer zeros to (FP0-FP31)		No gating term enabled
	(FP0-FP31) ⟶/⟶ (B0-B31)	BXFP = FAFL PH7 + ...	LSW of floating point result
	Reset flip-flop NSXBF	S/NSXBF = N(S/SXB) (S/SXB) = FAFL PH7 + ... R/NSXBF = ...	Preset logic for B⟶S in PH8
	Force a one on private memory address line LR31	S/NLR31F = N(S/LR31) (S/LR31) = FAFL PH7 + ... R/NLR31F = ...	Select private memory register Ru1 address during PH10
	Set flip-flop RW if FDL and TRAP signal is not true	S/RW = (S/RW/FP) (S/RW/FP) = PH9 NTRAP FPRD + ... R/RW = ...	Prepare to send LSW of result to CPU
	Set flip-flop CC1 if exponent underflow has occurred and FZ is a one	S/CC1 = (S/CC1/3) + ... (S/CC1/3) = (S/CC1/FP) + ... (S/CC1/FP) = PH9 FEUF + ... R/CC1 = (R/CC1)	
	Set flip-flop CC2 if exponent underflow or overflow or divide by zero attempted	S/CC2 = (S/CC2/3) + ... (S/CC2/3) = (S/CC2/FP) + ... (S/CC2/FP) = PH9 (FEUF + FEOF + SW1) + ... R/CC2 = (R/CC2) FEOF = NE0 E1 NRTZ	
			Mnemonic: FDS (3E, BE) FDL (1E, 9E)

(Continued)

Table 3-67. FDS, FDL Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH7; Box PH9; T8L (Cont.)	Enable TRAP signal if exponent underflow has occurred and FZ is a one, if exponent overflow has occurred, or if a divide by zero was attempted	TRAP = FEUF FZ + FEOF + SW1 + ...	TRAP prevents RW from being set in PH9 and PH10
	Enable signal (S/SXA) if NFPR is true	(S/SXA) = PH9 NFPR + ...	Preset adder to give result the proper polarity
	Enable signal (S/SXMA) if FPR is true	(S/SXMA) = PH9 FPR + ...	
	If FPR is true, transfer (NE0-NE7) → (E0-E7)	EXNE = PH9 FPR NTRAP N(FEUF NFZ)	A negative result requires an inverted exponent
	Branch to Box PH10	S/PH10 = PH9 R/PH10 = ...	
CPU PH8; Box PH10; T8L	One clock long (B0-B31) → (S0-S31) → (RW0-RW31)	Logic set at PH7 clock RWXS/0-RWXS/3 = RW ... RW = Set at PH7 clock if no trap condition and long format	Transfer LSW of result to private memory register Ru1
	If LSW of result is not equal to zero, set flip-flop SW0	S/SW0 = NS0031Z (S/SW0 NZ) (S/SW0/NZ) = FAFL NO2 PH8 R/SW0 = RESET/A + ...	Used in PH10 for condition code settings
	Reset flip-flop NSXBF	S/NSXBF = N(S/SXB) (S/SXB) = FAFL PH8 + ... R/NSXBF = ...	Preset logic for B → S in PH10
	Set flip-flop RW if TRAP signal is not true	S/RW = (S/RW/FP) (S/RW/FP) = PH10 NTRAP + ... R/RW = ...	Prepare to send MSW of result to CPU
	Set flip-flop DRQ	S/DRQ = BRPH10 + ... BRPH10 = FAFL PH8 + ... R/DRQ = ...	Inhibits transmission of another clock until data release received from core memory. Request for next instruction made in PH6
	(A47-A71, A0-A31) → or -(A47-A71, A0-A31) → (S47-S71, S0-S31)	Adder logic set at PH9 clock	MSW of mantissa → sum bus
			Mnemonic: FDS (3E, BE) FDL (1E, 9E)

(Continued)

Table 3-67. FDS, FDL Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
CPU PH8; Box PH10; T8L (Cont.)	S47 ⟶ FP0	FP0 = S47 FPXSU + ...	MSW of result transferred to FP lines if result not equal to zero or if underflow with FZ = 0 does not exist
		FPXSU = NFPDIS PH10 NRTZ N(FEUF NFZ)	
	NE1 ⟶ FP1 (+64 bias)	FP1 = NE1 FPXSU + ...	
	(E2-E7) ⟶ (FP2-FP7)	FPXSU	
	(S48-S71) ⟶ (FP8-FP31)	FPXSU	
	(FP0-FP31) ⟶/⟶ (B0-B31)	BXFP = FAFL PH8 + ...	MSW of result ⟶/⟶ B-register
	Reset Box PH10	R/PH10 = ...	Floating point box actions are finished
	Branch to CPU PH10	S/PH10 = BRPH10 NCLEAR + ...	
		R/PH10 = ...	
CPU PH10; Box actions over; T8L	One clock long		
	(B0-B31) ⟶ (S0-S31) ⟶	Adder logic set at PH8 clock	
	(RW0-RW31)	RWXS/0-RWXS/3 = RW + ...	
		RW = Set at PH8 clock if no trap condition	
	Set flip-flop CC3 if floating point result is positive	S/CC3 = SGTZ TESTS + ...	
		SGTZ = (S0 + S1 + ... + S31 + SW0) NS0 ...	SW0 is set when there is significance in LSW of result
		TESTS = FAFL ENDE + ...	
		R/CC3 = TESTS + ...	
	Set flip-flop CC4 if floating point result is negative	S/CC4 = (S/CC4/2) TESTS + ...	
		(S/CC4/2) = NFACOMP S0 + ...	
		R/CC4 = TESTS + ...	
	ENDE functions		
			Mnemonic: FDS (3E, BE) FDL (1E, 9E)

3-75 Family of Stack and Multiple Instructions (FAST)

GENERAL. Seven instructions are included in the family of stack and multiple instructions: Push Word (PSW), Pull Word (PLW), Push Multiple (PSM), Pull Multiple (PLM), Modify Stack Pointer (MSP), Load Multiple (LM), and Store Multiple (STM). The family is divided into six instruction categories determined by the logic used to implement these instructions. Each instruction is included in more than one category. The categories are as follows:

FAST – PSM, PSW, PLM, PLW, MSP

FAST/A – PSM, PSW, PLM, PLW

FAST/M – PSM, PLM, PSW, PLW, LM, STM

FAST/L – PLM, PLW, LM

FAST/S – PSM, PSW, STM

FAST/C – PSM, MSP

STACK POINTER DOUBLEWORD. All FAST instructions except LM and STM operate with a stack and a stack pointer doubleword. An area of consecutive memory locations reserved for a particular purpose is called a stack. Operands are stored, or pushed into the stack and loaded, or pulled from the stack on a last-in, first-out basis. The push instructions are PSW and PSM; the pull instructions are PLW and PLM. The location of each stack is defined by a stack pointer doubleword stored elsewhere in memory. The format of the stack pointer doubleword is shown below.

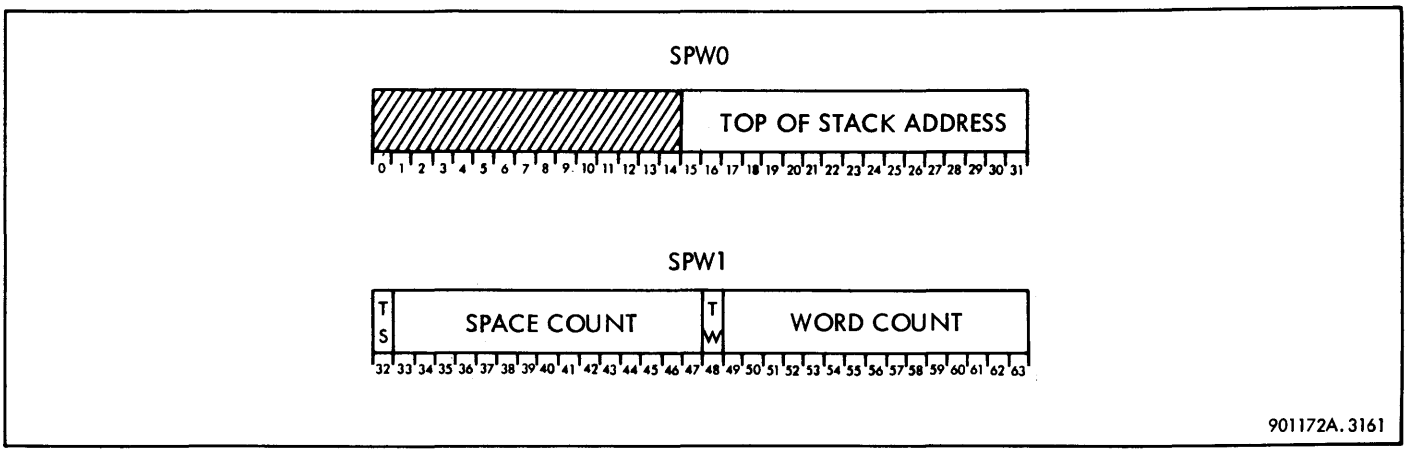
Bits 0 through 31 of the doubleword comprise stack pointer doubleword 0 (SPW0). Bits 0 through 15 of SPW0 are insignificant, and bits 15 through 31 indicate the address of the word currently at the top of the stack (TSA), that is, the highest numbered address in the stack as it exists at the time of the current instruction. In stack pointer doubleword 1 (SPW1), bit positions 33 through 47 contain the space count, that is, the number of word locations currently available in the region of memory allocated to the stack. Bit positions 49 through 63 contain the word count, that is, the number of words currently in the stack.

Bit 32 in SPW1 is the trap-on-space inhibit bit (TS), and is used to determine what the computer does if the current instruction would cause the space count to exceed $2^{15}-1$ or to be less than zero. If TS is zero and overflow or underflow occurs, the computer traps to location X'42' and the condition code remains unchanged. If TS is a one and overflow or underflow occurs, the computer sets condition code bit CC1 to a one and executes the next instruction in sequence.

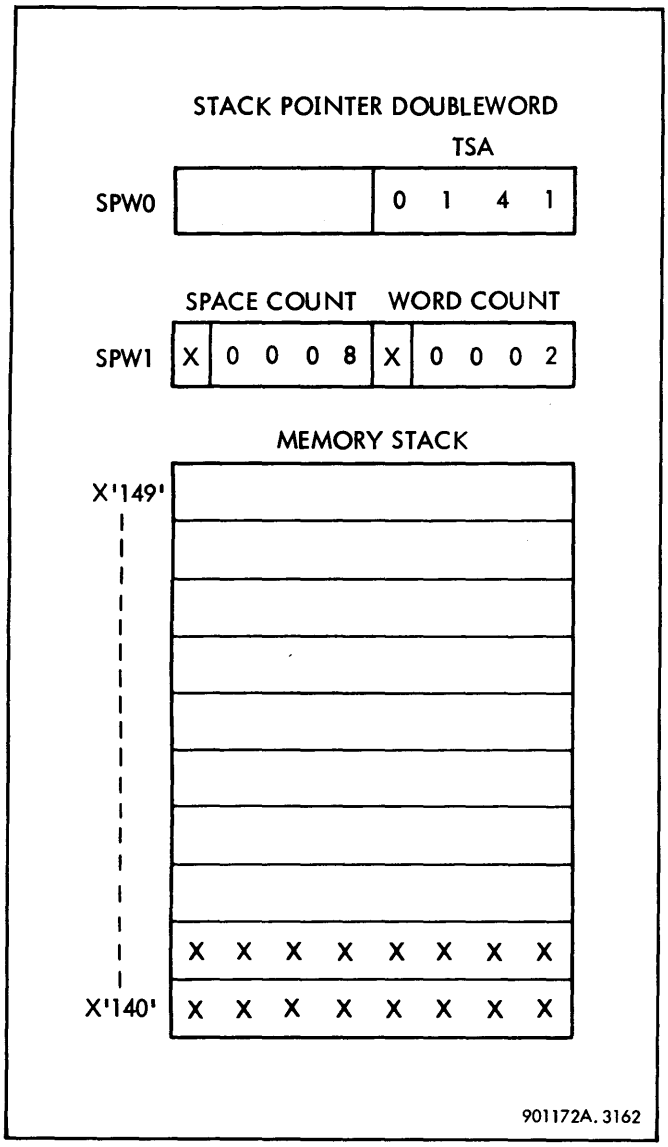
Bit 48 of SPW1 is the trap-on-word inhibit bit (TW), and is used to determine what the computer does if the current instruction would cause the word count to exceed $2^{15}-1$ or to be less than zero. If TW is zero and overflow or underflow occurs, the computer traps to X'42' and the condition code remains unchanged. If TW is a one and overflow or underflow occurs, the computer sets condition code bit CC3 to a one and executes the next instruction in sequence.

If the push or pull instruction is successfully executed, condition code bits CC1 and CC3 are reset and condition code bits CC2 and CC4 are set to indicate the current status of the space and word counts. These bits both remain zero if the space and word count are both greater than zero. If the word count is zero, indicating that the stack is now empty, condition code bit CC4 is set. If the space count is zero, indicating that the stack is now full, condition code bit CC2 is set.

If the instruction is aborted, condition code bits CC2 and CC4 are set if the space count or word count was zero before the instruction was started.



Example. An example of a memory stack with the corresponding stack pointer doubleword is shown below.



PUSH WORD (PSW; 09, 89). The PSW instruction stores the contents of the private memory register specified in the R field into the top of the core memory stack defined by the stack pointer doubleword. The stack pointer doubleword is located at the address specified in the reference address field of the PSW instruction. The location in which the word is stored is the next higher core memory address than that specified by the top of stack address in the stack pointer doubleword. The current top of stack address in the stack pointer doubleword is incremented by one to point to the new top of stack location. The space count in the stack pointer doubleword is decremented by one and the word count is incremented by one. The condition code is set as described under Stack Pointer Doubleword (page 3-438) to reflect the new status of the space count and word count.

If the space count or word count limits would be exceeded by the instruction, the instruction is aborted or a trap routine is entered if allowed by the TS or TW inhibit bits. The condition code is then set as described under Stack Pointer Doubleword (page 3-438).

PUSH WORD PHASE SEQUENCE. Preparation phases for the PSW instruction are the same as the general PREP phases for word instructions, paragraph 3-59. Figure 3-171. shows the simplified phase sequence for the PSW instruction. Table 3-68 lists the detailed logic sequence during all PSW execution phases. During the first pass through the phase 1 phases, word count overflow and space count underflow are checked in the adder and indicators are set, but the adder output is not used. The instruction branches from PH1/C to PH2, obtains the top of stack address, and stores the push word in core memory during two passes through PH6. From PH8 the instruction branches back to PH1/A to update and store the new top of stack address, word count, and space count. After PH1/G, PH9 is entered to obtain the address of the next instruction in sequence, and PH10 enables the ENDE operation to take place.

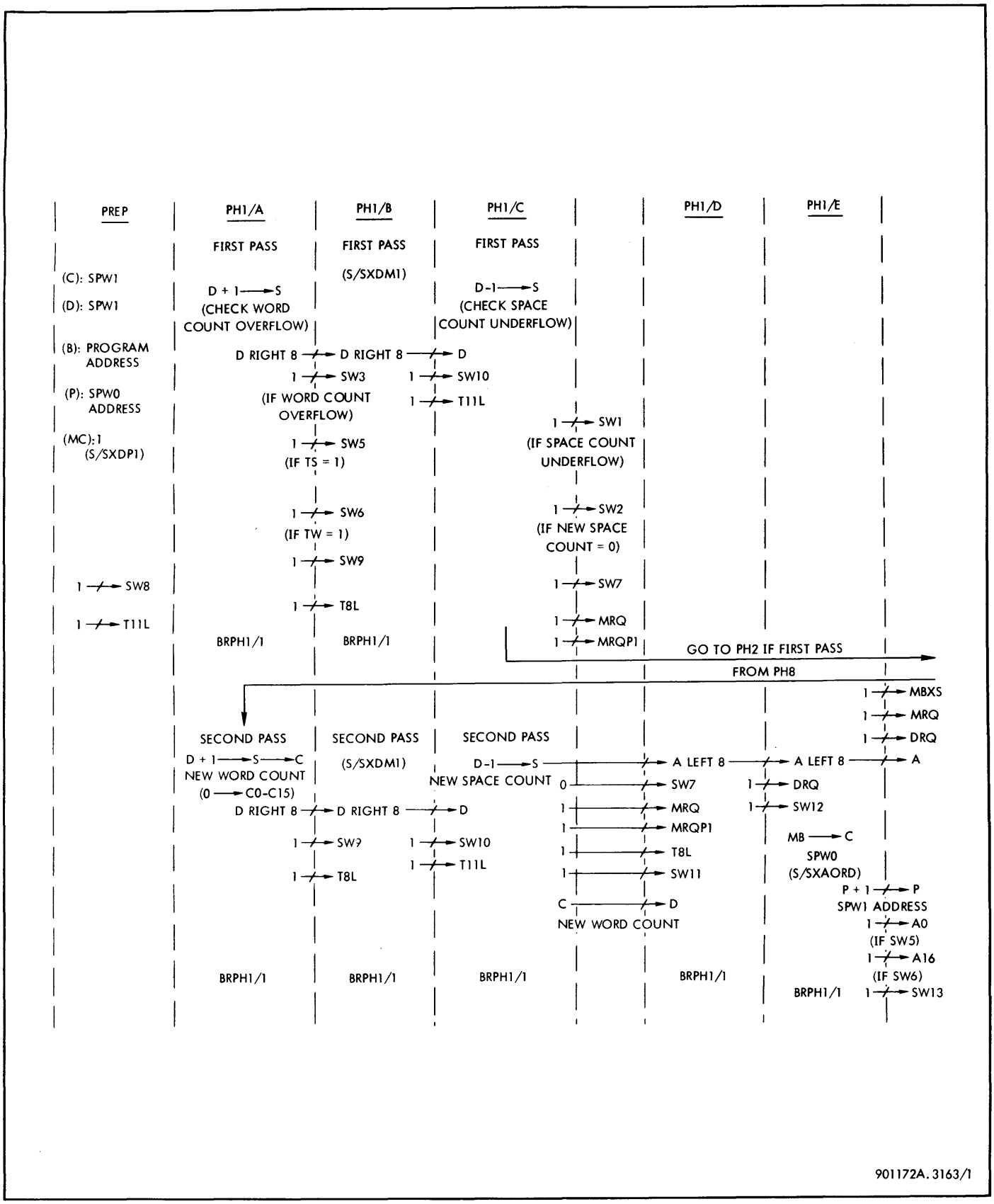


Figure 3-171. Push Word Instruction, Phase Sequence Diagram (Sheet 1 of 3)

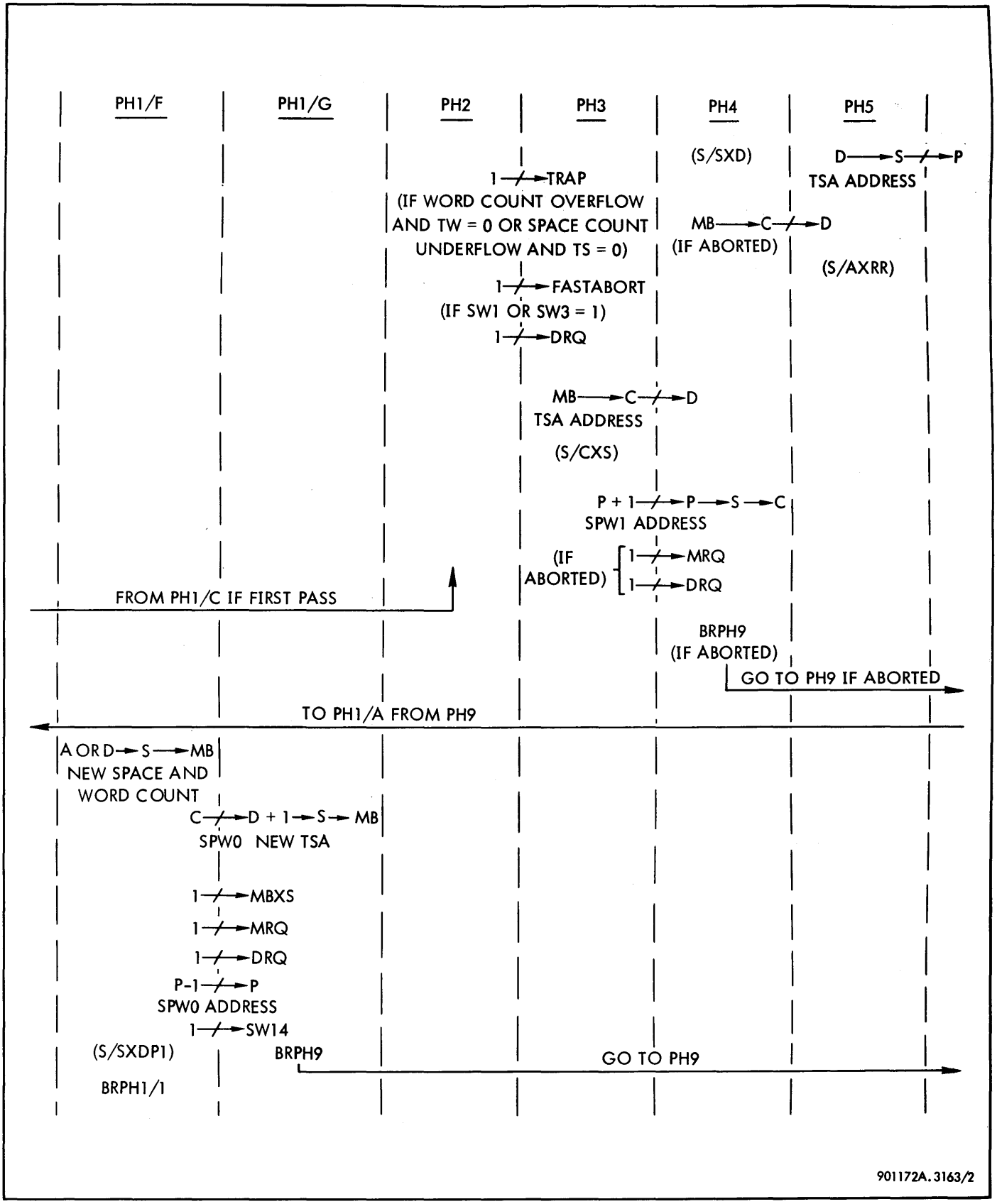


Figure 3-171. Push Word Instruction, Phase Sequence Diagram (Sheet 2 of 3)

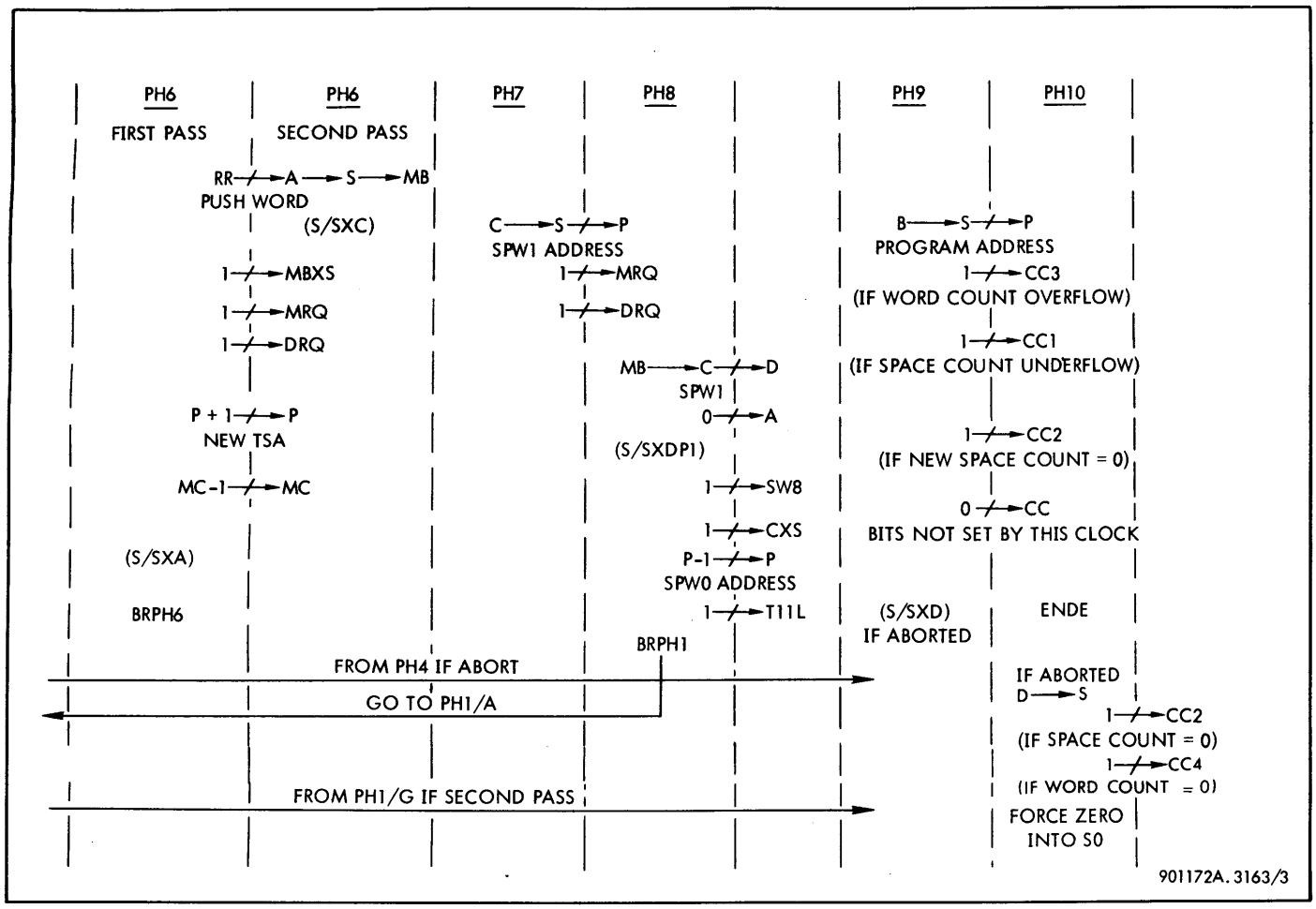


Figure 3-171. Push Word Instruction, Phase Sequence Diagram (Sheet 3 of 3)

Table 3-68. Push Word Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(C) : SPW1		Stack pointer double-word 1
	(D) : SPW1		Stack pointer double-word 1
	(B) : Program address		Address of next instruction in sequence
	(P) : SPW0 address		Location of bits 0 through 31 of stack pointer doubleword
	(MC) : 1		Macro-counter set to 1
			Mnemonic: PSW (09, 89)

(Continued)

Table 3-68. Push Word Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PREP (Cont.)	Preset conditions with PRE3		
	Enable signal (S/SXDP1)	(S/SXDP1) = FUPSW (PRE3 + ...) + ... FUPSW = OU0 (O4 NO5) OL9 + ...	Preset adder for D plus 1 in PH1/A
	Set flip-flop SW8	S/SW8 = BRSW8 NRESET/A BRSW8 = (FAST PRE3) + ... (FAST PRE3) = OU0 (O4 NO5) PRE3	
	Reset flip-flop NT11L	S/NT11L = N(S/T11L) (S/T11L) = (FAST PRE3) + ... R/NT11L = ...	Set clock T11L for PH1/A
PH1/A	One clock long	PH1/A = PH1 SW8	
T11L	D + 1 ⟶ S	Adder logic set at last PREP clock	Add 1 to word count in SPW1 to check for overflow
	Set SW3 if word count overflows	S/SW3 = (S/SW3) (S/SW3) = (A16 ⊕ K16) FAST PH1/A + ...	Word count overflows into adder bit 16
	Set SW5 if TS is 1	S/SW5 = (S/SW5) (S/SW5) = FAST PH1/A D0 + ...	Trap-on-space inhibit bit is in D0
	Set SW6 if TW is 1	S/SW6 = (S/SW6) (S/SW6) = FAST PH1/A D16 + ...	D16 contains trap-on-word inhibit bit TW
	Down align D-register	DXDR8 = FAST PH1/A + ...	Shift D-register 8 bits right as first half of 16-bit down alignment
	Set flip-flop SW9	S/SW9 = SW8 STEP815 STEP815 = NBRSW8 NBRSW10 NBRSW11 NBRSW12 NBRSW13 NBRSW15	
	Reset flip-flop NT8L	S/NT8L = N(S/T8L) (S/T8L) = FAST PH1 R/NT8L = ...	Set clock T8L for PH1/B
	Sustain PH1	BRPH1/1 = FAST PH1 N(NSW7 PH1/C) + ...	Hold PH1 for PH1/B
PH1/B	One clock long	PH1/B = PH1 SW9	
T8L	Down align D-register	DXDR8 = FAST PH1/B + ...	Shift D-register 8 bits right to complete 16-bit down alignment. Space count is now in D17 through D31
			Mnemonic: PSW (09, 89)

(Continued)

Table 3-68. Push Word Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH1/B T8L (Cont.)	Enable signal (S/SXDM1)	(S/SXDM1) = FUPSW PH1/B + ...	Preset adder for D minus 1
	Set flip-flop SW10	S/SW10 = SW9 STEP815	
	Reset flip-flop NT11L	S/NT11L = N(S/T11L) (S/T11L) = FAST PH1/B R/NT11L = ...	Set clock T11L for PH1/C
	Sustain PH1	BRPH1/1 = FAST PH1 N(...)	Hold PH1 for PH1/C
PH1/C T11L	One clock long	PH1/C = PH1 SW10	
	D - 1 ⟶ S	Adder logic set at PH1/B clock	Decrement space count in D17 through D31 for underflow check only
	Force a zero into S16	S16INH = FAST PH1/C	Inhibit TS
	Set SW1 if space count underflows	S/SW1 = (S/SW1) (S/SW1) = (A16 ⊕ K16) FAST PH1/C + ...	Space count underflows into adder bit 16
	Set SW2 if new space count = 0	S/SW2 = (S/SW2) (S/SW2) = N(A16 ⊕ K16) S1631Z FAST PH1/C + ...	New space count = 0 if bits 16 through 31 of S-register = 0
	Set flip-flop SW7	S/SW7 = (S/SW7) (S/SW7) = FAST PH1/C NSW7 + ...	
	Set flip-flop MRQ	S/MRQ = (S/MRQ/3) + ... (S/MRQ/3) = FAST PH1/C +... R/MRQ = ...	Request for core memory cycle
	Reset flip-flop NMRQP1	S/NMRQP1 = N(S/MRQ/3) R/NMRQP1 = ...	Delay flip-flop for data release signal Go to PH2 if abort or first pass
PH2 T5L	One clock long Trap conditions:		
	Set flip-flop TRAP if word count overflows and TW = 0 or if space count underflows and TS = 0	S/TRAP = (S/TRAP) NRESET (S/TRAP) = FAST PH2 SW3 NSW6 + FAST PH2 SW1 NSW5	SW3 is word count overflow, SW1 is space count underflow, NSW6 ⟹ TW = 0, NSW5 ⟹ TS = 0
	Abort if SW1 or SW3 is set	S/FASTABORT = FAST PH2 SW1 + FAST PH2 SW3 S/FASTF1 = SW3 + SW1	Instruction unconditionally aborted on overflow or underflow. Note that FASTABORT is built with two flip-flops, FASTF1 and FASTF2
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = MRQP1 + ... R/DRQ = ...	Data request, inhibits transmission of another clock until data release received from core memory
			Mnemonic: PSW (09, 89)

(Continued)

Table 3-68. Push Word Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH3	Sustained until data release		
DR	(MB0-MB31)⟶(C0-C31)	CXMB = DG (data gate)	Top of stack address (SPW0) from memory⟶C-register
	(C0-C31)⟶/⟶(D0-D31)	DXC = FAST/A PH3	Top of stack address ⟶/⟶D-register
	If not aborted, reset flip-flop NCXS	S/NCXS = N(S/CXS) (S/CXS) = FAST/A PH3 NFASTF1 + ... R/NCXS = ...	Preset for S⟶C in PH4
	P + 1⟶/⟶P	PUC3I = FAST/A PH3 + ...	Add to SPW0 address to obtain SPW1 address
	Set flip-flop MRQ if instruction aborted	S/MRQ = (S/MRQ/2) + ... (S/MRQ/2) = FASTABORT PH3 + ... R/MRQ = ...	Request for core memory cycle
	Set flip-flop DRQ if instruction aborted	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = (S/MRQ/2) + ... R/DRQ = ...	Data request, inhibits transmission of another clock until data release from core memory
PH4	One clock long		
T5L	(P0-P31)⟶(S0-S31)	SXP = FAST PH4 NDIS + ...	Store SPW1 address in C-register
(DR if abort)	(S0-S31)⟶(C0-C31)	CXS set at PH3 clock	
	If instruction not aborted, enable signal (S/SXD)	(S/SXD) = FAST PH4 NBRPH9 + ...	Preset adder logic for D⟶S in PH5
	Abort conditions:		
	If SW1 or SW3 set, branch to PH9	BRPH9 = FAST PH4 (SW1 + SW3)	Branch to PH9 to set condition code
	(MB0-MB31)⟶(C0-C31)	CXMB = DG	Load SPW1 from memory into C-register
	(C0-C31)⟶/⟶(D0-D31)	DXC = FASTABORT PH4	Return SPW1 to D-register
PH5	One clock long		
T5L	(D0-D31)⟶(S0-S31)	Adder logic set at PH4 clock	Top of stack address (SPW0)⟶/⟶P-register
	(S0-S31)⟶/⟶(P0-P31)	PXS = FAST/A PH5 + ...	
	Reset flip-flop NAXRR	S/NAXRR = N(S/AXRR) (S/AXRR) = FAST/S PH5 + ... R/NAXRR = ...	Preset for transfer of private memory R contents ⟶/⟶A-register in PH6
			Mnemonic: PSW (09, 89)

(Continued)

Table 3-68. Push Word Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH6	One clock long		
T5L	(RR0-RR31) —/→ (A0-A31)	AXRR set at PH5 clock	Store private memory register R contents in A-register
	Set flip-flop MBXS	S/MBXS = (S/MBXS) (S/MBXS) = FAST/S PH6 NMCZ	Preset for transfer of A-register contents to core memory in second PH6
	Set flip-flop MRQ	S/MRQ = (S/MRQ) (S/MRQ) = (S/MBXS) + ... R/MRQ = ...	Request for core memory cycle
	Set flip-flop DRQ	S/DRQ = (S/DRQ) (S/DRQ) = (S/MBXS) + ...	Data request, inhibits transmission of another clock until data release from core memory
	P + 1 —/→ P	PUC31 = FAST/S PH6 + ...	Upcount P-register to obtain new top of stack address
	MC - 1 —/→ MC	MCD7 = FAST/M PH6 NIOEN + ...	Decrement macro-counter by 1
	Enable signal (S/SXA)	(S/SXA) = FAST/S PH6 NMCZ	Preset adder logic for A ——→ S in second PH6
	Sustain PH6	BRPH6 = FAST/M PH6 NMCZ	Repeat PH6 to store contents of A-register in memory
PH6	Sustained until data release		
DR	(A0-A31) ——→ (S0-S31)	Adder logic set at first PH6 clock	Store A-register contents in memory at new top of stack address
	(S0-S31) ——→ (MB0-MB31)	MBXS set at first PH6 clock	
	Enable signal (S/SXC) if MC = 0	(S/SXC) = FAST/S PH6 OU0 MCZ + ...	Preset adder for C ——→ S in PH7
PH7	One clock long		
T5L	(C0-C31) ——→ (S0-S31)	Adder logic set at first PH6 clock	SPW1 address ——→ S
	(S0-S31) —/→ (P0-P31)	PXS = FAST/A PH7 + ...	SPW1 address —/→ P
	Set flip-flop MRQ	S/MRQ = (S/MRQ/2) + ... (S/MRQ/2) = FAST/A PH7 + ... R/MRQ = ...	Request for memory cycle
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = (S/MRQ/2) + ... R/DRQ = ...	Data request, inhibits transmission of another clock until data release from memory
PH8	Sustained until data release		
DR	(MB0-MB31) ——→ (C0-C31)	CXMB = DG	SPW1 from core memory ——→ C-register
			Mnemonic: PSW (09, 89)

(Continued)

Table 3-68. Push Word Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH8	(C0-C31)—/→(D0-D31)	DXC = FAST/A PH8 + ...	SPW1—/→D-register
DR (Cont.)	Zeros—/→(A0-A31)	AXZ = FAST (PH8 + ...)	Clear A-register for word count and space count
	Enable signal (S/SXDP1)	(S/SXDP1) = FUPSW (PH8 + ...)	Preset adder for D plus 1 in PH1/A
	Set flip-flop SW8	S/SW8 = NRESET BRSW8 BRSW8 = FAST/A PH8 + ...	
	Reset flip-flop NCXS	S/NCXS = N(S/CXS) (S/CXS) = FAST/A PH8 + ... R/NCXS = ...	Preset for S——→C in PH1/A
	P - 1—/→P	PDC31 = FAST/A PH8 + ...	Decrement P-register to obtain SPW0 address
	Reset flip-flop NT11L	S/NT11L = N(S/T11L) (S/T11L) = FAST PH8 + ... R/NT11L = ...	Set clock T11L for PH1/A
	Branch to PH1/A	BRPH1 = FAST/A PH8 + ... S/PH1 = BRPH1 NCLEAR	
PH1/A	One clock long	PH1/A = PH1 SW8 FAST	
T11L	D + 1——→S	Adder logic for D plus 1 set at PH8 clock	Update word count by adding 1 to SPW1 in D-register. Gate onto sum bus
	Force a zero into S16	S16 = (K16 ⊕ PR16) SXADD NS16INH S16INH = FAST PH1/A + ...	S16 (bit 48 of SPW1) is trap-on-word bit TW and not included in word count
	(S16-S31)——→(C16-C31)	CXS set at PH8 clock	New word count into C-register bits 16 through 31
	Zeros——→(C0-C15)	CXS/0 = CXS N(FAST PH1/A) CXS/1 = CXS N(FAST PH1/A)	S0-S15 not gated into C0-C15 because CXS/0 and CXS/1 are low
	Down align D-register	DXDR8 = FAST PH1/A + ...	Shift D-register 8 bits right as first half of 16-bit down alignment
	Set flip-flop SW9	S/SW9 = SW8 STEP815	
			Mnemonic: PSW (09, 89)

(Continued)

Table 3-68. Push Word Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH1/A T11L (Cont.)	Reset flip-flop NT8L	S/NT8L = N(S/T8L) (S/T8L) = FAST PH1 + ... R/NT8L = ...	Set clock T8L for PH1/B
	Sustain PH1	BRPH1/1 = FAST PH1 N(...)	
PH1/B T8L	One clock long	PH1/B = PH1 SW9	
	Down align D-register	DXDR8 = FAST PH1/B + ...	Shift D-register 8 bits right to complete 16-bit down alignment. Space count is now in D17-D31
	Enable signal (S/SXDM1)	(S/SXDM1) = FUPSW PH1/B + ...	Preset adder for D minus 1 in PH1/C
	Set flip-flop SW10	S/SW10 = SW9 STEP815	
	Reset flip-flop NT11L	S/NT11L = N(S/T11L) (S/T11L) = FAST PH1 + ... R/NT11L = ...	Set clock T11L for PH1/C
	Sustain PH1	BRPH1/1 = FAST PH1 N(...)	
PH1/C T11L	One clock long	PH1/C = PH1 SW10	
	D - 1 ⟶ S	Adder logic set for D minus 1 in PH1/B	Update space count by subtracting 1 from D17-D31
	Force a zero into S16	S16INH = FAST PH1/C	S16 is now trap-on-space inhibit bit TS and is not included in space count
	(S0-S31) ⟶ (A0-A31)	AXS = FAST PH1/C SW7	Store new word count in D-register
	(C0-C31) ⟶ (D0-D31)	DXC = FAST PH1/C + ...	Store new word count in D-register
	Reset flip-flop SW7	R/SW7 = (R/SW7) (R/SW7) = FAST PH1/C SW7	
	Set flip-flop MRQ	S/MRQ = (S/MRQ/3) + ... (S/MRQ/3) = FAST PH1/C R/MRQ = ...	Request for core memory cycle
	Reset flip-flop NMRQP1	S/NMRQP1 = N(S/MRQ/3) R/NMRQP1 = ...	Delay flip-flop for data release signal
	Reset flip-flop NT8L	S/NT8L = N(S/T8L) + ... (S/T8L) = FAST PH1 R/NT8L = ...	Set clock T8L for PH1/D
	Set flip-flop SW11	S/SW11 = SW10 STEP815	
	Sustain PH1	BRPH1/1 = FAST PH1 N [PH1C (NSW7 + SW3) + ...]	
			Mnemonic: PSW (09, 89)

(Continued)

Table 3-68. Push Word Sequence (Cont.)

Phase	Function Performed	Signals Involved			Comments
PH1/D	One clock long	PH1/D	=	PH1 SW11	
T8L	Up align A-register	AXAL8	=	FAST PH1/D + ...	Shift A-register 8 bits left as first half of 16-bit up alignment
	Set flip-flop DRQ	S/DRQ	=	MRQP1 + ...	Data request, inhibits transmission of another clock until data release received from core memory
	Set flip-flop SW12	S/SW12	=	SW11 STEP815	
PH1/E	Sustained until data release	PH1/E	=	PH1 SW12	
DR	(MB0-MB31) ⟶ (C0-C31)	CXMB	=	DG	SPW0 ⟶ C-register
	Up align A-register	AXAL8	=	FAST PH1/E + ...	Shift A-register 8 bits left as second half of 16-bit up alignment. New space count is now in A1 through A15
	Enable signal (S/SXAORD)	(S/SXAORD)	=	FAST PH1/E + ...	Preset adder for A OR D ⟶ S in PH1/F
	Set flip-flop A0 if TS is 1 (SW5)	S/A0	=	FAST PH1/E SW5 AXAL8 + ...	Set trap-on-space inhibit bit if set in original SPW1
	Set flip-flop A16 if TW is 1 (SW6)	S/A16	=	FAST PH1/E SW6 AXAL8 + ...	Set trap-on-word inhibit bit if set in original SPW1
	Set flip-flop MBXS	S/MBXS	=	(S/MBXS)	Preset for transfer of A OR D to core memory in PH1/F
		(S/MBXS)	=	FAST PH1/E + ...	
		R/MBXS	=	...	
	Set flip-flop MRQ	S/MRQ	=	(S/MRQ)	Request for core memory cycle
		(S/MRQ)	=	(S/MBXS) + ...	
		R/MRQ	=	...	
	Set flip-flop DRQ	S/DRQ	=	(S/DRQ NCLEAR)	Data request, inhibits transmission of another clock until data release from memory
		(S/DRQ)	=	(S/MBXS) + ...	
		R/DRQ	=	...	
	P + 1 ⟶/⟶ P	PUC31	=	FAST PH1/E	Increment P-register to obtain SPW1 address
	Set flip-flop SW13	S/SW13	=	SW12 STEP815	
PH1/F	Sustained until data release	PH1/F	=	PH1 SW13	
DR	A_V D ⟶ S	Adder logic set at PH1/E clock			New word count in D-register and new space count in A-register ⟶ S
					Mnemonic: PSW (09, 89)

(Continued)

Table 3-68. Push Word Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH1/F DR (Cont.)	(S0-S31) ⟶ (MB0-MB31)	MBXS set by PH1/E clock	Store new space count and word count in core memory at SPW1 location
	Set flip-flop MBXS	S/MBXS = (S/MBXS) (S/MBXS) = FAST PH1/F + ... R/MBXS = ...	Preset memory write
	Set flip-flop MRQ	S/MRQ = (S/MRQ) (S/MRQ) = (S/MBXS) + ... R/MRQ = ...	Request for core memory cycle
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = (S/MBXS) + ... R/DRQ = ...	Data request, inhibits another clock until data release received from core memory
	(C0-C31) ⟶/⟶ (D0-D31)	DXC = FAST PH1/F + ...	Top of stack address (SPW0) in C-register clocked into D-register
	Enable signal (S/SXDP1)	(S/SXDP1) = FUPSW (PH1/F + ...) + ...	Preset adder for D plus 1 in PH1/G
	P - 1 ⟶/⟶ P	PDC31 = FAST PH1/F + ...	Decrement P-register to obtain SPW0 address
	Set flip-flop SW14	S/SW14 = SW13 STEP815	
	Sustain PH1	BRPH1/1 = FAST PH1 N(NSW7 PH1/C) + ...	
PH1/G	Sustained until data release	PH1/G = SW14 PH1	
DR	D + 1 ⟶ S	Adder logic set at PH1/F clock	Add 1 to top of stack address in D-register to obtain new top of stack address
	(S0-S31) ⟶ (MB0-MB31)	MBXS set by PH1/F clock	Store new top of stack address in memory at SPW0 location
	Branch to PH9	BRPH9 = FAST PH1/G S/PH9 = BRPH9 NCLEAR + ... R/PH9 = ...	
PH9	One clock long		
T5L	(B0-B31) ⟶ (S0-S31)	SXB = PXSXB NDIS + ... PXSXB = NFAFL NFAMDS PH9	Program address ⟶/⟶ P-register via sum bus
			Mnemonic: PSW (09, 89)

(Continued)

Table 3-68. Push Word Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH9 T5L (Cont.)	(S0-S31)—/→(P0-P31)	PXS = PXSXB + ...	
	Set condition code:		
	Set CC3 if word count overflow and TW = 1 (SW6)	S/CC3 = (S/CC3/1) + ... (S/CC3/1) = FAST PH9 SW3 + ...	SW3 indicates word count overflow. If TW were 0, instruction would have trapped and not reached PH9
	Set CC1 if space count underflow and TS = 1 (SW5)	S/CC1 = (S/CC1/1) + ... (S/CC1/1) = FAST PH9 SW1 + ...	SW1 indicates space count underflow. If TS were 0, instruction would have trapped and not reached PH9
	Set CC2 if new space count = 0	S/CC2 = (S/CC2/1) + ... (S/CC2/1) = (FASTNABORT PH9) SW2 + ...	If instruction is successfully completed and stack is full, CC2 is set
		R/CC = FAST PH9 + ...	Inputs to reset sides of CC flip-flops to reset those not set by this instruction
	Enable signal (S/SXD) if instruction aborted	(S/SXD) = FASTABORT PH9	Preset adder for D—→S in PH10
PH10 DR	Sustained until data release		
	Normal ENDE		
	If instruction aborted:		
	Correct CC2	S/CC2 = (S/CC2/4) + ... (S/CC2/4) = S0007Z S0815Z (FASTABORT ENDE)	Set CC2 if original space count (in D-register) = 0
	Force zeros in SGTZ, S16, and S0	SGTZ = N(FASTABORT ENDE)	To prevent setting CC3
		S16 = N(FASTABORT ENDE)	S16 is TW inhibit bit. S0 is TS bit. Neither should be checked for zero
		S0 = N(FASTABORT ENDE)	
	Correct CC4	S/CC4 = (S/CC4/2) + ... (S/CC4/2) = (FASTABORT ENDE) S1631Z	Set CC4 if original word count (in D-register) = 0
			Mnemonic: PSW (09, 89)

PULL WORD (PLW; 08, 88). The PLW instruction loads the private memory register specified in the R field of the instruction with the word currently at the top of the core memory stack. The top of stack word is at the location specified in the top of stack address field in the stack pointer doubleword.

The current top of stack address in the stack pointer doubleword is decremented by one to point to the new top of stack location. The space count in the stack pointer doubleword is incremented by one and the word count is decremented by one. The condition code is set as described under Stack Pointer Doubleword (page 3-438)

to reflect the new status of the space count and word count.

If the space count or word count limits would be exceeded by the instruction, the instruction is aborted and a trap routine is entered if allowed by the TW or TS bit. The condition code is set as described under Stack Pointer Doubleword (page 3-438).

PULL WORD PHASE SEQUENCE. Preparation phases for the PLW instruction are the same as the general PREP phases for word instructions, paragraph 3-59. Figure 3-172 shows the simplified phase sequence for the PLW instruction. Table 3-69 lists the detailed logic sequence during all PLW execution phases. During the first pass through the phase 1 phases, word count underflow and space count overflow are checked in the adder and indicators are set, but the adder output is not used. The instruction branches from PH1/C to PH2, obtains the top of stack address, reads the pull word from core memory, and stores the word in private memory during two passes through PH6. From PH8 the instruction branches back to PH1/A to update and store the new top of stack address, word count, and space count. After PH1/G, PH9 is entered to obtain the address of the next instruction in sequence, and PH10 enables the ENDE operation to take place.

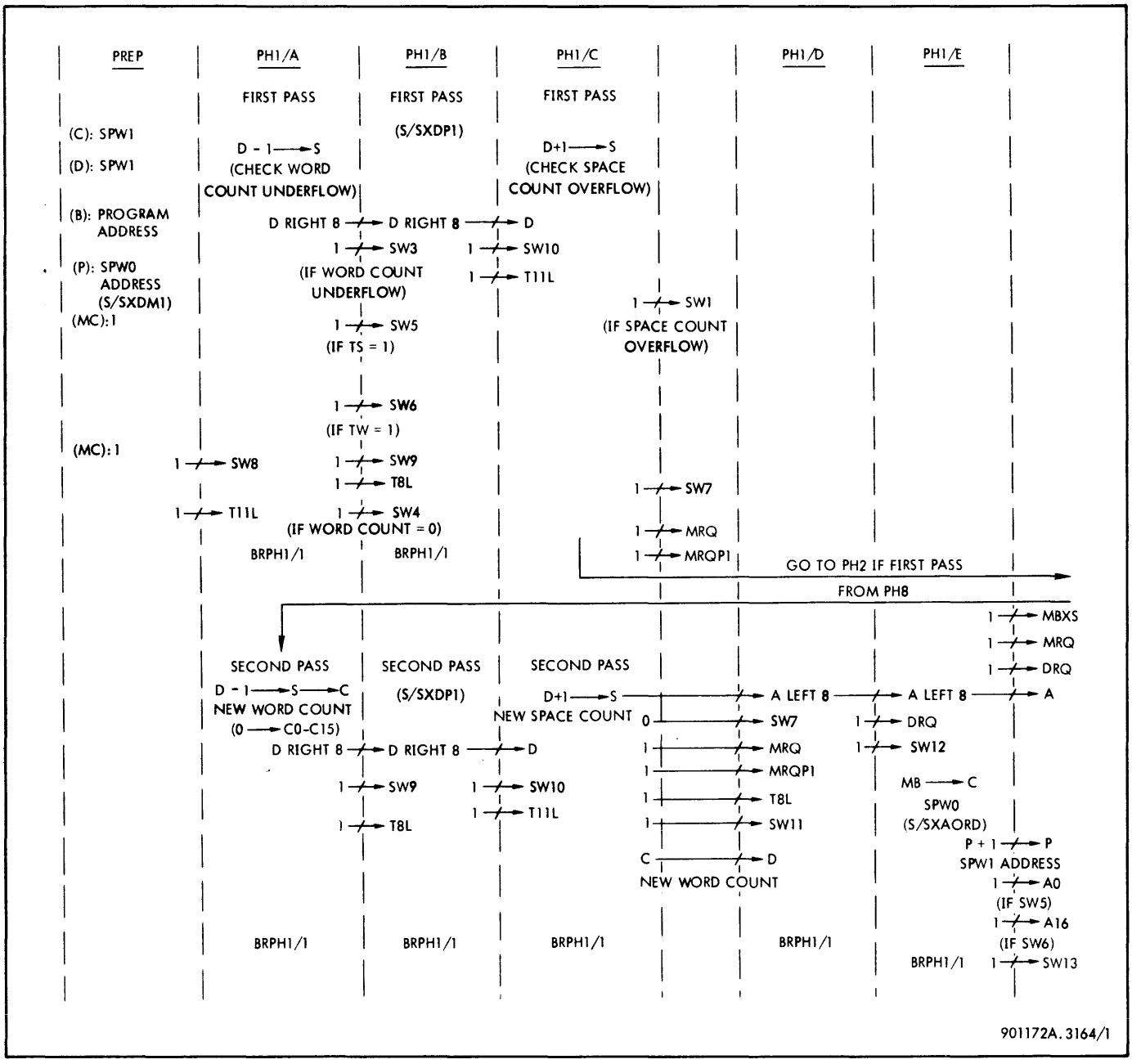


Figure 3-172. Pull Word Instruction, Phase Sequence Diagram (Sheet 1 of 3)

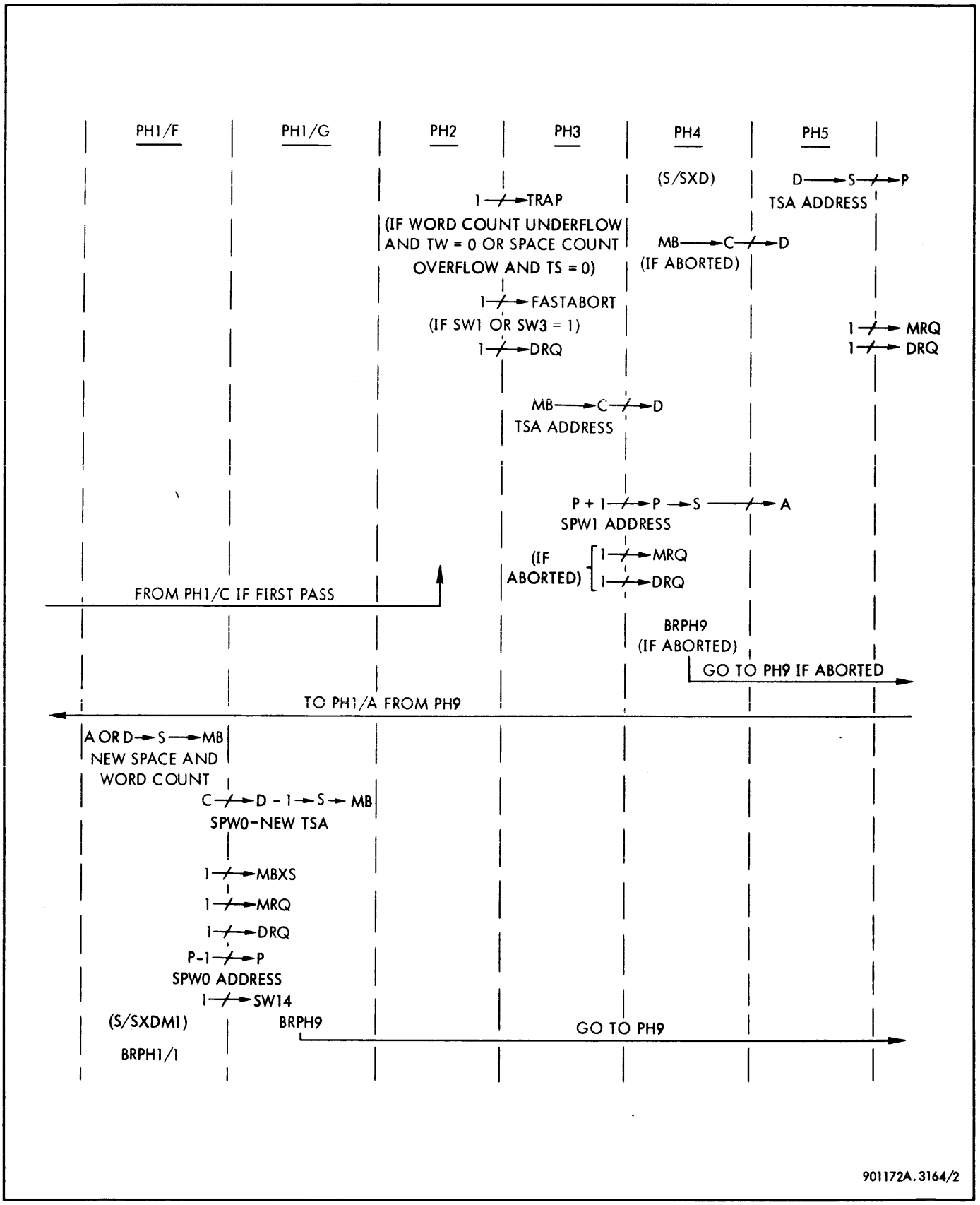


Figure 3-172. Pull Word Instruction, Phase Sequence Diagram (Sheet 2 of 3)

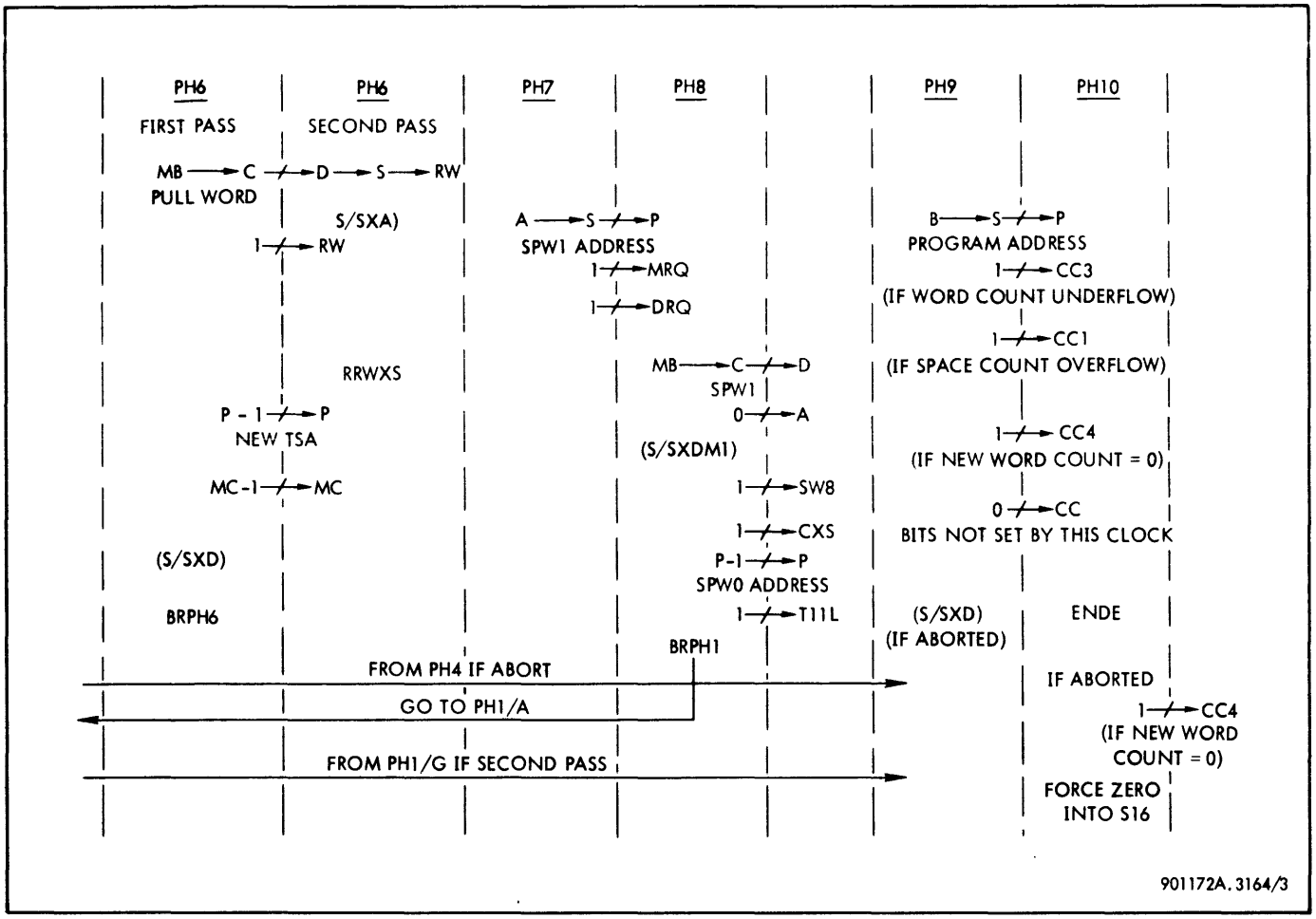


Figure 3-172. Pull Word Instruction, Phase Sequence Diagram (Sheet 3 of 3)

Table 3-69. Pull Word Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(C) : SPW1		Stack pointer doubleword 1
	(D) : SPW1		Stack pointer doubleword 1
	(B) : Program address		Address of next instruction in sequence
	(P) : SPW0 address		Location of bits 0 through 31 of stack pointer doubleword
	(MC) : 1		Macro-counter set to 1
			Mnemonic: PLW (08, 88)

(Continued)

Table 3-69. Pull Word Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PREP (Cont.)	Preset conditions with PRE3:		
	Enable signal (S/SXDM1)	(S/SXDM1) = FUPLW (PRE3 + ...) + ... FUPLW = OU0 (O4 NO5) OL8	Preset adder for D minus 1 in PH1/A
	Set flip-flop SW8	S/SW8 = BRSW8 NRESET/A BRSW8 = FAST PRE3 + ...	
	Reset flip-flop NT11L	S/NT11L = N(S/T11L) (S/T11L) = FAST PRE3 + ...	Set clock T11L for PH1/A
PH1/A	One clock long	PH1/A = PH1 SW8	
T11L	D - 1→S	Adder logic set at last PREP clock	Subtract 1 from word count in SPW1 to check for underflow
	Force a zero into S16	S16INH = FAST PH1/A	Inhibit TW
	Set SW3 if word count underflows	S/SW3 = (S/SW3) (S/SW3) = (A16 ⊕ K16) FAST PH1/A	Word count underflows into adder bit 16
	Set SW5 if TS is 1	S/SW5 = (S/SW5) (S/SW5) = FAST PH1/A D0 + ...	Trap-on-space inhibit bit is in D0
	Set SW6 if TW is 1	S/SW6 = (S/SW6) (S/SW6) = FAST PH1/A D16 + ...	D16 contains trap-on-word inhibit bit TW
	Down align D-register	DXDR8 = FAST PH1/A + ...	Shift D-register 8 bits right as first half of 16-bit down alignment
	Set SW4 if word count = 0	S/SW4 = (S/SW4) (S/SW4) = N(A16 ⊕ K16) S1631Z FAST PH1/A + ...	New word count = 0 if S16-S31 = 0
	Set flip-flop SW9	S/SW9 = SW8 STEP815 + ... STEP815 = NBRSW8 NBRSW10 NBRSW11 NBRSW12 NBRSW13 NBRSW15	
	Reset flip-flop NT8L	S/NT8L = N(S/T8L) (S/T8L) = FAST PH1 R/NT8L = ...	Set clock T8L for PH1/B
	Sustain PH1	BRPH1/1 = FAST PH1 N(NSW7 PH1/C) + ...	Hold PH1 for PH1/B
PH1/B	One clock long	PH1/B = PH1 SW9	
T8L	Down align D-register	DXDR8 = FAST PH1/B + ...	Shift D-register 8 bits right to complete 16-bit down alignment. Space count is now in D17 through D31
			Mnemonic: PLW (08, 88)

(Continued)

Table 3-69. Pull Word Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH1/B T8L (Cont.)	Enable signal (S/SXDP1)	(S/SXDP1) = FUPLW PH1/B + ...	Preset adder for D plus 1
	Set flip-flop SW10	S/SW10 = SW9 STEP815	
	Reset flip-flop NT11L	S/NT11L = N(S/T11L) (S/T11L) = FAST PH1/B R/NT11L = ...	Set clock T11L for PH1/C
	Sustain PH1	BRPH1/1 = FAST PH1 N(NSW7 PH1/C) + ...	Hold PH1 for PH1/C
PH1/C T11L	One clock long	PH1/C = PH1 SW10	
	D + 1 ⟶ S	Adder logic set at PH1/B clock	Increment space count in D17 through D31 for overflow check only
	Set SW1 if space count overflows	S/SW1 = (S/SW1) (S/SW1) = (A16 ⊕ K16) FAST PH1/C + ...	Space count overflows into adder bit 16
	Set flip-flop SW7	S/SW7 = (S/SW7) (S/SW7) = FAST PH1/C NSW7 + ...	
	Set flip-flop MRQ	S/MRQ = (S/MRQ/3) + ... (S/MRQ/3) = FAST PH1/C + ... R/MRQ = ...	Request for core memory cycle
	Reset flip-flop NMRQP1	S/NMRQP1 = N(S/MRQ/3) R/NMRQP1 = ...	Delay flip-flop for data release signal Go to PH2 if abort or first pass
PH2 T5L	One clock long Trap conditions:		
	Set flip-flop TRAP if word count underflows and TW = 0 or if space count overflows and TS = 0	S/TRAP = (S/TRAP) (S/TRAP) = FAST PH2 SW3 NSW6 + FAST PH2 SW1 NSW5	SW3 is word count underflow, SW1 is space count overflow, NSW6 ⟹ TW = 0, NSW5 ⟹ TS = 0
	Abort if SW1 or SW3 is set	S/FASTABORT = FAST PH2 SW1 + FAST PH2 SW3 S/FASTF1 = SW3 + SW1	Instruction unconditionally aborted on overflow or underflow. Note that FASTABORT is built with two flip-flops, FASTF1 and FASTF2
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = MRQP1 + ... R/DRQ = ...	Data request, inhibits transmission of another clock until data release received from core memory
PH3	Sustained until data release		
DR	(MB0-MB31)⟶(C0-C31)	CXMB = DG (data gate)	Top of stack address from memory⟶C-register
			Mnemonic: PLW (08, 88)

(Continued)

Table 3-69. Pull Word Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH3 DR (Cont.)	(C0-C31) —/→ (D0-D31)	DXC = FAST/A PH3	Top of stack address —/→ D-register
	P + 1 —/→ P	PUC31 = FAST/A PH3 + ...	Add 1 to SPW0 address to obtain SPW1 address
	Set flip-flop MRQ if instruction aborted	S/MRQ = (S/MRQ/2) + ... (S/MRQ/2) = FASTABORT PH3 + ... R/MRQ = ...	Request for core memory cycle
	Set flip-flop DRQ if instruction aborted	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = (S/MRQ/2) + ... R/DRQ = ...	Data request, inhibits transmission of another clock until data release from core memory
PH4	One clock long		
T5L (DR if abort)	(P0-P31) ——→ (S0-S31)	SXP = FAST PH4 NDIS	Store SPW1 address in A-register
	(S0-S31) ——→ (A0-A31)	AXS = FAST PH4	
	If instruction not aborted, enable signal (S/SXD)	(S/SXD) = FAST PH4 NBRPH9	Preset adder logic for D ——→ S in PH5
	Abort conditions:		
	If SW1 or SW3 set, branch to PH9	BRPH9 = FAST PH4 (SW1 + SW3)	Branch to PH9 to set condition code
	(MB0-MB31) ——→ (C0-C31)	CXMB = DG	Load SPW1 from memory into C-register
	(C0-C31) —/→ (D0-D31)	DXC = FASTABORT PH4	Return SPW1 to D-register
PH5	One clock long		
T5L	(D0-D31) ——→ (S0-S31)	Adder logic set at PH4 clock	Top of stack address (SPW0) —/→ P-register
	(S0-S31) —/→ (P0-P31)	PXS = FAST/A PH5 + ...	
	Set flip-flop MRQ	S/MRQ = (S/MRQ/2) + ... (S/MRQ/2) = FAST/L PH5 R/MRQ = ...	Request for core memory cycle
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = S/MRQ/2	Data request, inhibits transmission of another clock until data release from memory
PH6 DR 1st Pass	Sustained until data release (MB0-MB31) ——→ (C0-C31)	CXMB = DG	Load pull word from top of stack address in memory ——→ C-register
			Mnemonic: PLW (08, 88)

(Continued)

Table 3-69. Pull Word Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH6 DR	(C0-C31) —/→ (D0-D31)	DXC = FAST/L PH6 + ...	Place pull word in D-register for transfer to private memory
1st Pass (Cont.)	Set flip-flop RW	S/RW = (S/RW) (S/RW) = FAST/L PH6 NMCZ + ...	RW is private memory write flip-flop
	P - 1 —/→ P	PDC31 = FAST/L PH6 OU0 + ...	Decrement P-register to obtain new top of stack address
	MC - 1 —/→ MC	MDC7 = FAST/M PH6 NIOEN + ...	Decrement macro-counter by 1
	Enable signal (S/SXD)	(S/SXD) = FAST/L PH6 NMCZ + ...	Preset adder logic for D ——→ S in second PH6
	Sustain PH6	BRPH6 = FAST/M PH6 NMCZ + ...	Repeat PH6 to store contents of D-register in private memory
PH6 T5L 2nd Pass	Sustained until data release		
	(D0-D31) ——→ (S0-S31)	Adder logic set at first PH6 clock	Transfer D-register contents to private memory via S-register
	(S0-S31) ——→ (RW0-RW31)	RWXS = RW	
	Enable signal (S/SXA)	(S/SXA) = FAST/L PH6 MCZ OU0 + ...	Preset adder for A ——→ S in PH7
PH7 T5L	One clock long		
	(A0-A31) ——→ (S0-S31)	Adder logic set at first PH6 clock	SPW1 address ——→ S
	(S0-S31) —/→ (P0-P31)	PXS = FAST/A PH7 + ...	SPW1 address —/→ P
	Set flip-flop MRQ	S/MRQ = (S/MRQ/2) + ... (S/MRQ/2) = FAST/A PH7 + ... R/MRQ = ...	Request for memory cycle
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = (S/MRQ/2) + ... R/DRQ = ...	Data request, inhibits transmission of another clock until data release from memory
PH8 DR	Sustained until data release		
	(MB0-MB31) ——→ (C0-C31)	CXMB = DG	SPW1 from core memory ——→ C-register
	(C0-C31) —/→ (D0-D31)	DXC = FAST/A PH8 + ...	SPW1 —/→ D-register
	Zeros —/→ (A0-A31)	AXZ = FAST (PH8 + ...)	Clear A-register for word count and space count
	Enable signal (S/SXDM1)	(S/SXDM1) = FUPLW (PH8 + ...) + ...	Preset adder for D minus 1 in PH1/A
			Mnemonic: PLW (08, 88)

(Continued)

Table 3-69. Pull Word Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH8 DR (Cont.)	Set flip-flop SW8	S/SW8 = NRESET BRSW8 BRSW8 = FAST/A PH8 + ...	
	Reset flip-flop NCXS	S/NCXS = N(S/CXS) (S/CXS) = FAST/A PH8 + ... R/NCXS = ...	Preset for S⟶C in PH1/A
	P - 1⟶P	PDC31 = FAST/A PH8 + ...	Decrement P-register to obtain SPW0 address
	Reset flip-flop NT11L	S/NT11L = N(S/T11L) (S/T11L) = FAST PH8 + ... R/NT11L = ...	Set clock T11L for PH1/A
	Branch to PH1/A	BRPH1 = FAST/A PH8 + ... S/PH1 = BRPH1 NCLEAR	
PH1/A T11L	One clock long	PH1/A = PH1 SW8 FAST	
	D - 1⟶S	Adder logic for D minus 1 set at PH8 clock	Update word count by subtracting 1 from SPW1 in D-register. Gate onto sum bus
	Force a zero into S16	S16 = (K16 ⊕ PR16) SXADD NS16INH S16INH = FAST PH1/A + ...	S16 (bit 48 of SPW1) is trap-on-word inhibit bit TW, and is not included in word count
	(S16-S31)⟶(C16-C31)	CXS set at PH8 clock	New word count into C-register bits 17 through 31
	Zeros⟶(C0-C15)	CXS/0 = CXS N(FAST PH1/A) CXS/1 = CXS N(FAST PH1/A)	S0-S15 not gated into C0-C15 because CXS/0 and CXS/1 are low
	Down align D-register	DXDR8 = FAST PH1/A + ...	Shift D-register 8 bits right as first half of 16-bit down alignment
	Set flip-flop SW9	S/SW9 = SW8 STEP815	
	Reset flip-flop NT8L	S/NT8L = N(S/T8L) (S/T8L) = FAST PH1 + ... R/NT8L = ...	Set clock T8L for PH1/B
	Sustain PH1	BRPH1/1 = FAST PH1 N(NSW7 PH1/C) + ...	
			Mnemonic: PLW (08, 88)

(Continued)

Table 3-69. Pull Word Sequence (Cont.)

Phase	Function Performed	Signals Involved			Comments
PH1/B	One clock long	PH1/B	=	PH1 SW9	
T8L	Down align D-register	DXDR8	=	FAST PH1/B + ...	Shift D-register 8 bits right to complete 16-bit down alignment. Space count is now in D17-D31
	Enable signal (S/SXDP1)	(S/SXDP1)	=	FUPLW PH1/B + ...	Preset adder for D plus 1 in PH1/C
	Set flip-flop SW10	S/SW10	=	SW9 STEP815	
	Reset flip-flop NT11L	S/NT11L	=	N(S/T11L)	Set clock T11L for PH1/C
		(S/T11L)	=	FAST PH1 + ...	
		R/NT11L	=	...	
	Sustain PH1	BRPH1/1	=	FAST PH1 N(NSW1 PH1/C) + ...	
PH1/C	One clock long	PH1/C	=	PH1 SW10	
T11L	D + 1 ⟶ S	Adder logic set for D plus 1 in PH1/B			Update space count by adding 1 to D17-D31
	Force a zero into S16	S16	=	(K16 ⊕ PR16) PH1/A NS16INH	S16 is now trap-on-space inhibit bit TS, and is not included in space count
		S16INH	=	FAST PH1/C	
	(S0-S31) ⟶ (A0-A31)	AXS	=	FAST PH1/C SW7 + ...	Hold new space count in A-register
	(C0-C31) ⟶ (D0-D31)	DXC	=	FAST PH1/C + ...	Hold new word count in D-register
	Reset flip-flop SW7	R/SW7	=	(R/SW7)	
		(R/SW7)	=	FAST PH1/C SW7 + ...	
	Set flip-flop MRQ	S/MRQ	=	(S/MRQ/3) + ...	Request for core memory cycle
		(S/MRQ/3)	=	FAST PH1/C	
		R/MRQ	=	...	
	Reset flip-flop NMRQP1	S/NMRQP1	=	N(S/MRQ/3)	Delay flip-flop for data release signal
		R/NMRQP1	=	...	
	Reset flip-flop NT8L	S/NT8L	=	N(S/T8L) + ...	Set clock T8L for PH1/D
		(S/T8L)	=	FAST PH1	
		R/NT8L	=	...	
	Set flip-flop SW11	S/SW11	=	SW10 STEP815	
	Sustain PH1	BRPH1/1	=	FAST PH1 N(NSW7 PH1/C) + ...	
PH1/D	One clock long	PH1/D	=	PH1 SW11	
T8L	Up align A-register	AXAL8	=	FAST PH1/D + ...	Shift A-register 8 bits left as first half of 16-bit up alignment
					Mnemonic: PLW (08, 88)

(Continued)

Table 3-69. Pull Word Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH1/D T8L (Cont.)	Set flip flop DRQ	S/DRQ = (S/DRQ) NCLEAR S/DRQ = MRQP1 + ... R/DRQ = ...	Data request, inhibits transmission of another clock until data release received from core memory
	Set flip-flop SW12	S/SW12 = SW11 STEP815	
PH1/E DR	Sustained until data release	PH1/E = PH1 SW12	
	(MB0-MB31) ⟶ (C0-C31)	CXMB = DG	SPW0 (TSA) ⟶ C-register
	Up align A-register	AXAL8 = FAST PH1/E + ...	Shift A-register 8 bits left as second half of 16-bit up alignment. New space count is now in A1 through A15
	Enable signal (S/SXAORD)	(S/SXAORD) = FAST PH1/E + ...	Preset adder for A OR D ⟶ S in PH1/F
	Set flip-flop A0 if TS is 1 (SW5)	S/A0 = FAST PH1/E SW5 AXAL8 + ...	Set trap-on-space inhibit bit if set in original SPW1
	Set flip-flop A16 if TW is 1 (SW6)	S/A16 = FAST PH1/E SW6 AXAL8 + ...	Set trap-on-word inhibit bit if set in original SPW1
	Set flip-flop MBXS	S/MBXS = (S/MBXS) (S/MBXS) = FAST PH1/E + ...	Preset for transfer of A OR D to core memory in PH1/F
	Set flip-flop MRQ	S/MRQ = (S/MRQ) (S/MRQ) = (S/MBXS) + ... R/MRQ = ...	Request for core memory cycle
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = (S/MBXS) + ... R/DRQ = ...	Data request, inhibits transmission of another clock until data release from memory
	P + 1 ⟶/ P	PUC31 = FAST PH1/E + ...	Increment P-register to obtain SPW1 address
	Set flip-flop SW13	S/SW13 = SW12 STEP815	
PH1/F DR	Sustained until data release	PH1/F = PH1 SW13	
	A OR D ⟶ S	Adder logic set at PH1/E clock	New word count in D-register and new space count in A-register ⟶ S
	(S0-S31) ⟶ (MB0-MB31)	MBXS set by PH1/E clock	Store new space count and word count in core memory at SPW1 location
	Set flip-flop MBXS	S/MBXS = (S/MBXS) (S/MBXS) = FAST PH1/F + ... R/MBXS = ...	Preset for memory write
			Mnemonic: PLW (08, 88)

(Continued)

Table 3-69. Pull Word Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH1/F DR (Cont.)	Set flip-flop MRQ	S/MRQ = (S/MRQ) (S/MRQ) = (S/MBXS) + ... R/MRQ = ...	Request for core memory cycle
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = (S/MBXS) + ... R/DRQ = ...	Data request, inhibits another clock until data release received from core memory
	(C0-C31) → (D0-D31)	DXC = FAST PH1/F + ...	Top of stack address (SPW0) in C-register clocked into D-register
	Enable signal (S/SXDM1)	(S/SXDM1) = FUPLW (PH1/F + ...) + ...	Preset adder for D minus 1 in PH1/G
	P - 1 → P	PDC31 = FAST PH1/F + ...	Decrement P-register to obtain SPW0 address
	Set flip-flop SW14	S/SW14 = SW13 STEP815	
	Sustain PH1	BRPH1/1 = FAST PH1 N(NSW7 PH1/C) + ...	
PH1/G DR	Sustained until data release	PH1/G = SW14 PH1	
	D - 1 → S	Adder logic set at PH1/F clock	Subtract 1 from top of stack address in D-register to obtain new top of stack address
	(S0-S31) → (MB0-MB31)	MBXS set by PH1/F clock	Store new top of stack address in memory at SPW0 location
	Branch to PH9	S/PH9 = BRPH9 NCLEAR + ... BRPH9 = FAST PH1/G	
PH9 T5L	One clock long		
	(B0-B31) → (S0-S31)	SXB = PXSXB NDIS PXSXB = NFAFL NFAMDS PH9	Program address → P-register via sum bus
	(S0-S31) → (P0-P31)	PXS = PXSXB	
	Set condition code:		
	Set flip-flop CC3 if word count underflow and TW = 1 (SW6)	S/CC3 = (S/CC3/1) + ... (S/CC3/1) = FAST PH9 SW3 + ...	SW3 indicates word count underflow. If TW were 0, instruction would have trapped and not reached PH9
	Set flip-flop CC1 if space count overflow and TS = 1 (SW5)	S/CC1 = (S/CC1/1) + ... (S/CC1/1) = FAST PH9 SW1 + ...	SW1 indicates space count overflow. If TS were 0, instruction could have trapped and not reached PH9
			Mnemonic: PLW (08, 88)

(Continued)

Table 3-69. Pull Word Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH9 T5L (Cont.)	Set flip-flop CC4 if new word count = 0	S/CC4 = (S/CC4/1) + ... (S/CC4/1) = (FASTNABORT PH9) SW4 + ...	If instruction is successfully completed and stack is empty, flip-flop CC4 is set
		R/CC = FAST PH9 + ...	Reset inputs to CC flip-flops to reset those not set in this phase
	Enable signal (S/SXD) if instruction aborted	(S/SXD) = FASTABORT PH9	Preset adder for D→S in PH10
PH10 DR	Sustained until data release Normal ENDE		
	If instruction aborted: Correct CC2	S/CC2 = (S/CC2/4) + ... (S/CC2/4) = S0007Z S0815Z (FASTABORT ENDE)	Set flip-flop CC2 if original space count (in D-register) = 0
	Correct CC4	S/CC4 = (S/CC4/2) + ... (S/CC4/2) = (FASTABORT ENDE) S1631Z	Set flip-flop CC4 if original word count (in D-register) = 0
	Force zeros into SGTZ, S0, and S16	SGTZ = N(FASTABORT ENDE)	To prevent setting CC3
		S0 = N(FASTABORT ENDE)	S0 is TS inhibit bit. S16 is TW inhibit bit. Neither should be checked for zero
		S16 = N(FASTABORT ENDE)	
			Mnemonic: PLW (08, 88)

PUSH MULTIPLE (PSM; 0B, 8B). The PSM instruction stores the contents of a sequential set of private memory registers into the push-down stack defined by the stack pointer doubleword. The number of words to be pushed is indicated by the condition code. If the contents of all 16 private memory registers are to be pushed into the stack, the initial value of the condition code is 0000. The private memory registers are treated as a circular set, with register 0 following register 15. The first register to be pushed into the stack is the register specified in the R field of the instruction. The contents of the last register pushed become the contents of the new top of stack location.

The private memory register contents are stored in core memory in ascending order, beginning with the location plus 1 of the current top of stack address pointed to in the stack pointer doubleword and ending with the current top of stack address plus the condition code.

The current top of stack address in the stack pointer doubleword is incremented by the value of the condition code to point to the new top of stack location. The space count in the stack pointer doubleword is decremented by the value of the condition code, and the word count is incremented by the value of the condition code. The condition code is set as described under Stack Pointer Doubleword (page 3-438) to reflect the new status of the space count and word count. If the space count or word count limits would be exceeded by the instruction, the instruction is aborted and a trap routine is entered if allowed by the TS or TW inhibit bit. The condition code is set as described under Stack Pointer Doubleword (page 3-438).

PUSH MULTIPLE PHASE SEQUENCE. Preparation phases for the PSM instruction are the same as the general PREP phases for word instructions, paragraph 3-59. Figure 3-173 shows the simplified phase sequence for the PSM instruction. Table 3-70 lists the detailed logic sequence during all PSM execution phases. During the first pass through the phase 1 phases, word count overflow and space count underflow are checked in the adder and indicators are set, but the adder output is not used. The instruction branches from PH1/C to PH2 and obtains

the top of stack address before PH6. The instruction loops through PH6, storing words from private memory into core memory, the number of loops depending on the number of words to be pushed. When a zero value in the macro-counter indicates that the last word has been stored, the instruction proceeds to PH7, and from PH8 branches back to PH1/A. From PH1/A to PH1/G, the new top of stack address, new space count, and new word count are calculated and stored in core memory in the stack pointer doubleword. After PH1/G, PH9 is entered to obtain the address of the next instruction, and PH10 enables the ENDE operation to take place.

If the condition code at the beginning of the instruction contains 0000, indicating that all 16 private memory registers are involved in the push operation, bit 3 of the macro-counter is set at the time the condition code is transferred to the A-register, thereby establishing 10000 as the number of words.

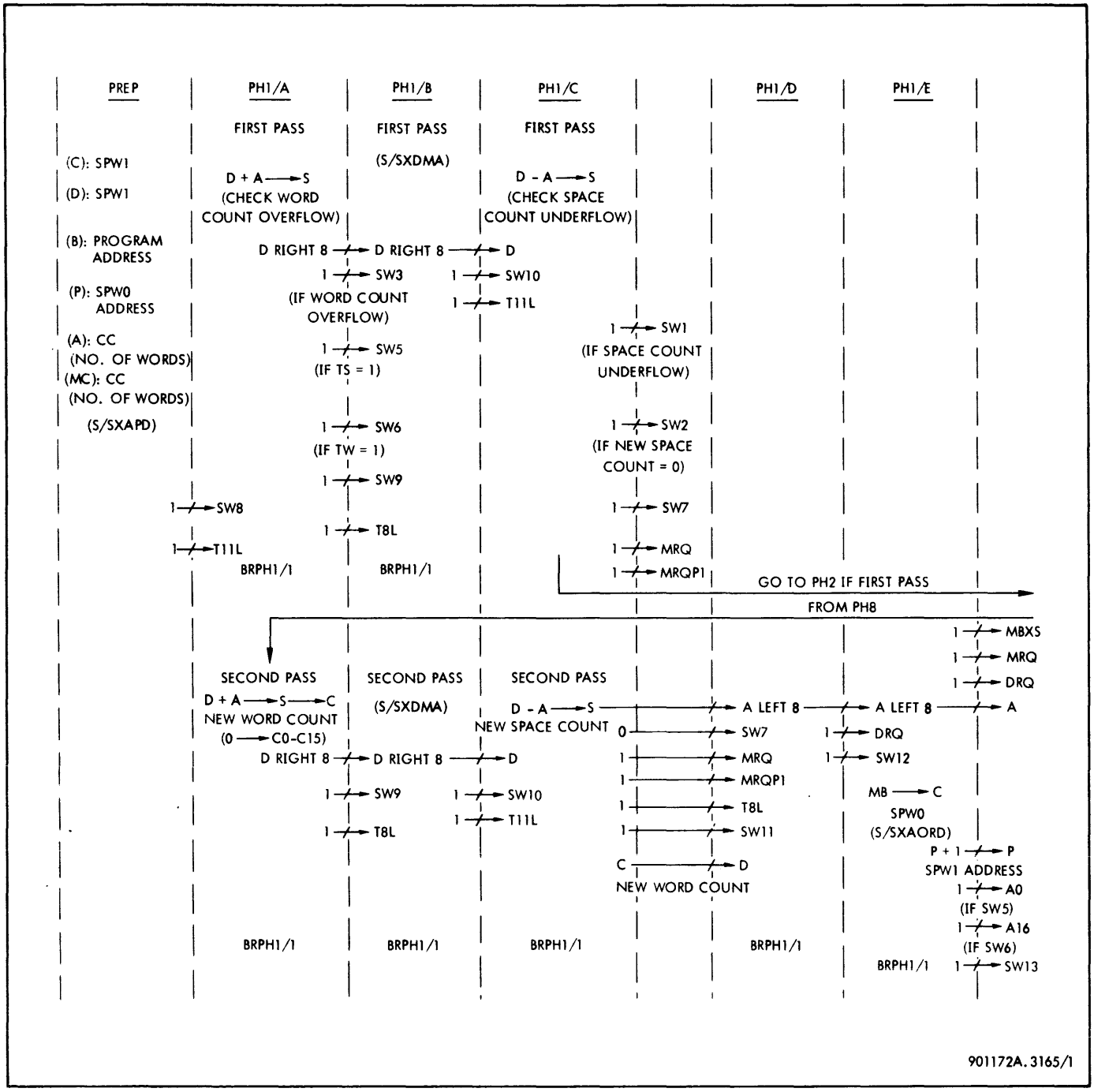


Figure 3-173. Push Multiple Instruction, Phase Sequence Diagram (Sheet 1 of 3)

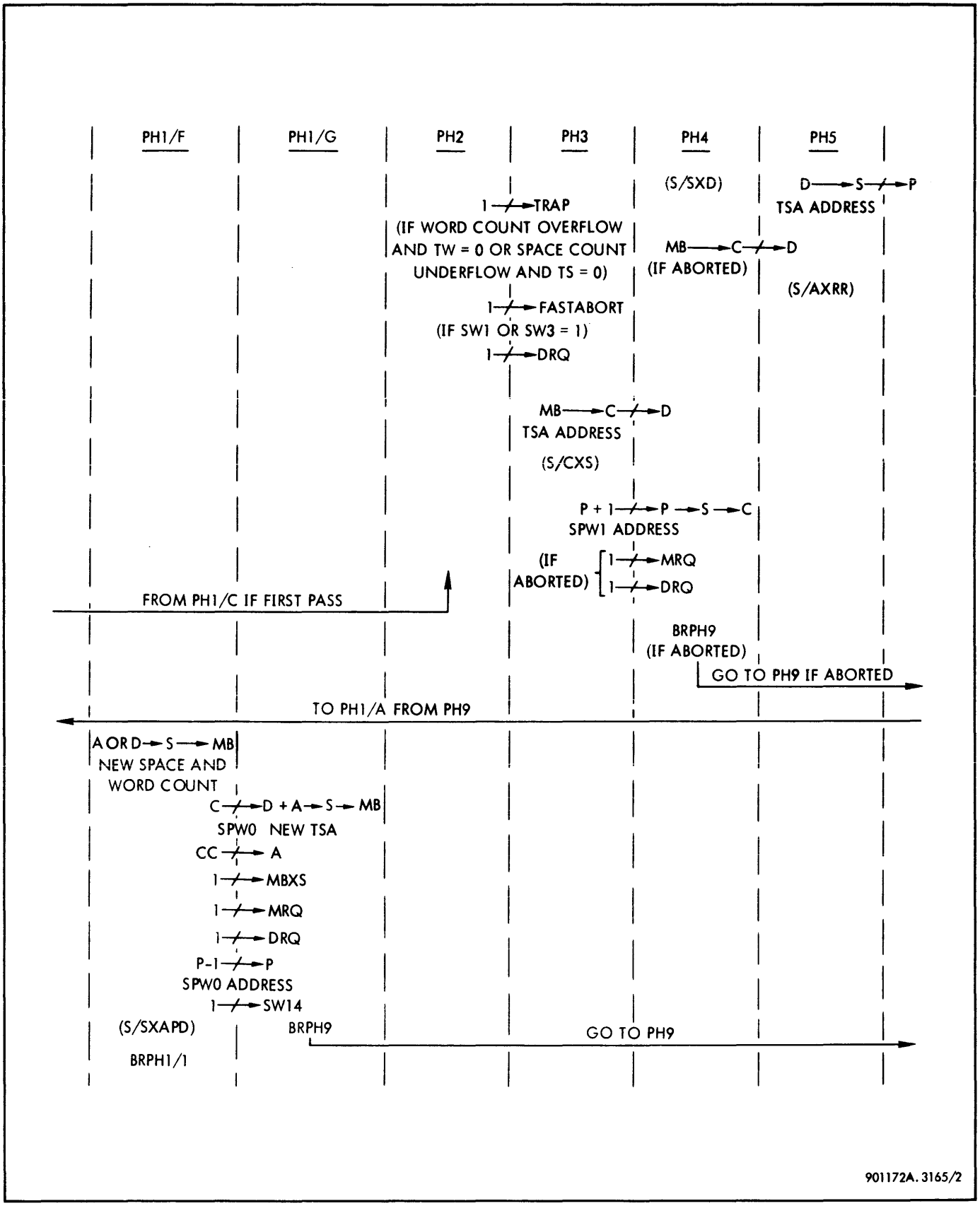


Figure 3-173. Push Multiple Instruction, Phase Sequence Diagram (Sheet 2 of 3)

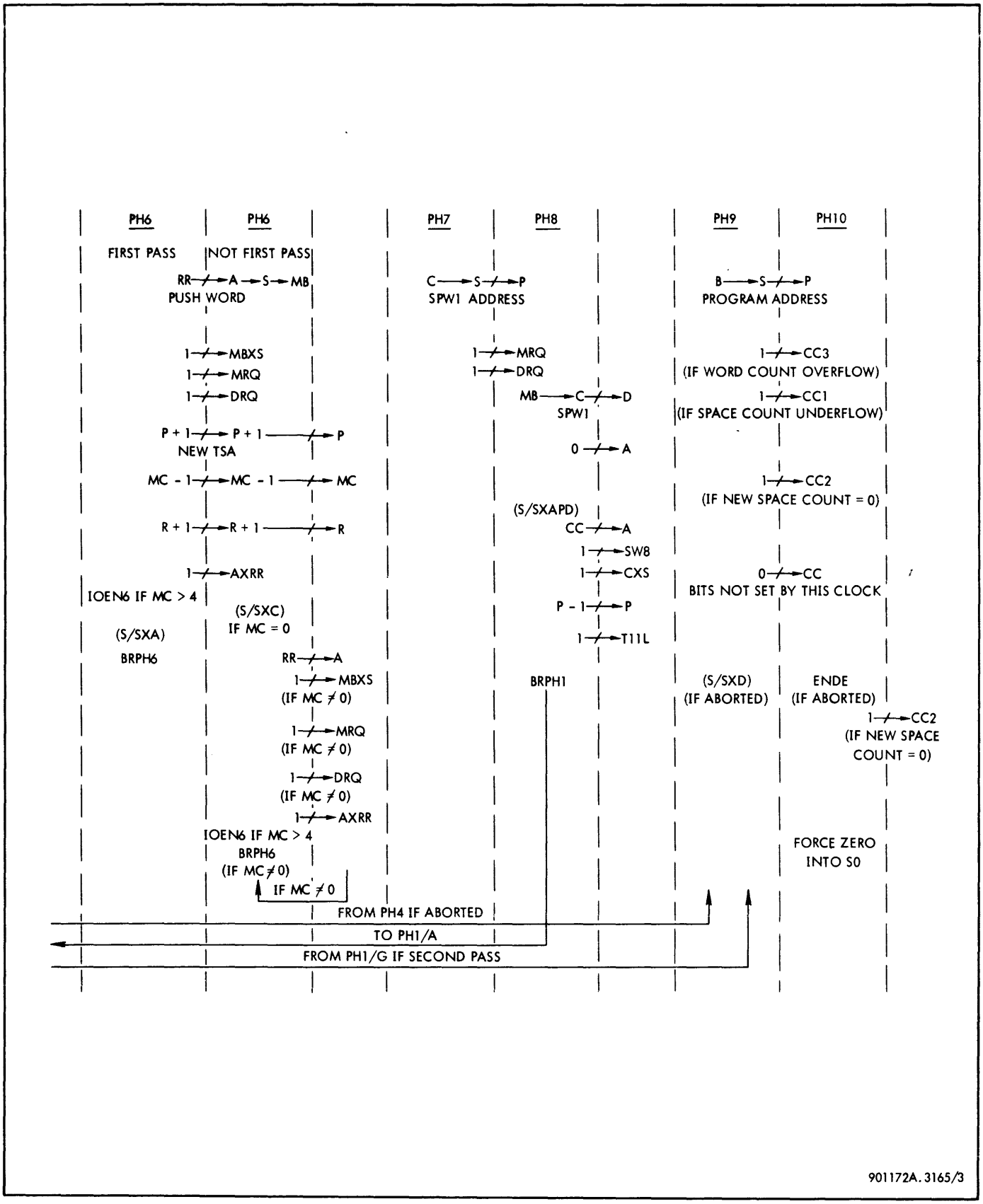


Figure 3-173. Push Multiple Instruction, Phase Sequence Diagram (Sheet 3 of 3)

Table 3-70. Push Multiple Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(C) : SPW1		Stack pointer doubleword 1
	(D) : SPW1		Stack pointer doubleword 1
	(B) : Program address		Address of next instruction in sequence
	(P) : SPW0 address		Location of bits 0 through 31 of stack pointer doubleword
	(A) : CC (number of words)		A-register contains number of words
	(MC) : CC (number of words)		Macro-counter set to number of words
	Preset conditions with PRE3		
	Enable signal (S/SXAPD)	(S/SXAPD) = FAST/C (PRE3 + ...) + ...	Preset adder for A plus D in PH1/A
	Set flip-flop SW8	S/SW8 = BRSW8 NRESET/A BRSW8 = FAST PRE3 + ...	
	Reset flip-flop NT11L	S/NT11L = N(S/T11L) (S/T11L) = FAST PRE3 + ... R/NT11L = ...	Set clock T11L for PH1/A
PH1/A T11L	One clock long D + A ⟶ S	PH1/A = PH1 SW8 Adder logic set at last PREP clock	Add number of words to word count in SPW1 to check for overflow
	Force a zero into S16	S16INH = FAST PH1/A	Inhibit TW
	Set SW3 if word count overflows	S/SW3 = (S/SW3) (S/SW3) = (A16 ⊕ K16) FAST PH1/A	Word count overflows into adder bit 16
	Set SW5 if TS is 1	S/SW5 = (S/SW5) (S/SW5) = FAST PH1/A D0 + ...	Trap-on-space inhibit bit is in D0
	Set SW6 if TW is 1	S/SW6 = (S/SW6) (S/SW6) = FAST PH1/A D16 + ...	D16 contains trap-on-word inhibit bit TW
	Down align D-register	DXDR8 = FAST PH1/A + ...	Shift D-register 8 bits right as first half of 16-bit down alignment
	Set flip-flop SW9	S/SW9 = SW8 STEP815 STEP815 = NBRSW8 NBRSW10 NBRSW11 NBRSW12 NBRSW13 NBRSW15	
	Reset flip-flop NT8L	S/NT8L = N(S/T8L) (S/T8L) = FAST PH1 R/NT8L = ...	Set clock T8L for PH1/B
	Sustain PH1	BRPH1/1 = FAST PH1 N(NSW7 PH1/C) + ...	Hold PH1 for PH1/B
			Mnemonic: PSM (0B, 8B)

(Continued)

Table 3-70. Push Multiple Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH1/B	One clock long	PH1/B = PH1 SW9	
T8L	Down align D-register	DXDR8 = FAST PH1/B + ...	Shift D-register 8 bits right to complete 16-bit down alignment. Space count is now in D17 through D31
	Enable signal (S/SXDMA)	(S/SXDMA) = FAST/C PH1/B	Preset adder for D minus A
	Set flip-flop SW10	S/SW10 = SW9 STEP815	
	Reset flip-flop NT11L	S/NT11L = N(S/T11L) (S/T11L) = FAST PH1/B R/NT11L = ...	Set clock T11L for PH1/C
	Sustain PH1	BRPH1/1 = FAST PH1 N(NSW7 PH1/C) + ...	Hold PH1 for PH1/C
PH1/C	One clock long	PH1/C = PH1 SW10	
T11L	D - A ⟶ S	Adder logic set at PH1/B clock	Subtract number of words from space count in D17 through D31 for underflow check only
	Force a zero into S16	S16INH = FAST PH1/C	Inhibit TS
	Set SW1 if space count underflows	S/SW1 = (S/SW1) (S/SW1) = (A16 ⊕ K16) FAST PH1/C + ...	Space count underflows into adder bit 16
	Set SW2 if new space count = 0	S/SW2 = (S/SW2) (S/SW2) = N(A16 ⊕ K16) S1631Z FAST PH1/C + ...	New space count = 0 if bits 16 through 31 of S-register = 0
	Set flip-flop SW7	S/SW7 = (S/SW7) (S/SW7) = FAST PH1/C NSW7 + ...	
	Set flip-flop MRQ	S/MRQ = (S/MRQ/3) + ... (S/MRQ/3) = FAST PH1/C + ... R/MRQ = ...	Request for core memory cycle
	Reset flip-flop NMRQP1	S/NMRQP1 = N(S/MRQ/3) R/NMRQP1 = ...	Delay flip-flop for data release signal Go to PH2 if abort or first pass
PH2	One clock long		
T5L	Trap conditions:		
	Set flip-flop TRAP if word count overflows and TW = 0 or if space count underflows and TS = 0	S/TRAP = (S/TRAP) (S/TRAP) = FAST PH2 SW3 NSW6 + FAST PH2 SW1 NSW5	SW3 is word count overflow, SW1 is space count underflow, NSW6 ⟹ TW = 0, NSW5 ⟹ TS = 0
			Mnemonic: PSM (0B, 8B)

(Continued)

Table 3-70. Push Multiple Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH2 T5L (Cont.)	Abort if SW1 or SW3 is set	S/FASTABORT = FAST PH2 SW1 + FAST PH2 SW3 S/FASTF1 = SW3 + SW1	Instruction unconditionally aborted on overflow or underflow. Note that FASTABORT is built with two flip-flops, FASTF1 and FASTF2
	Set flip-flop DRQ	S/DRQ = (S/DRQ) (S/DRQ) = MRQP1 + ...	Data request, inhibits transmission of another clock until data release received from core memory
PH3	Sustained until data release		
DR	(MB0-MB31) ⟶ (C0-C31)	CXMB = DG (data gate)	Top of stack address (SPW0) from memory ⟶ C-register
	(C0-C31) ⟶/⟶ (D0-D31)	DXC = FAST/A PH3	Top of stack address ⟶/⟶ D-register
	If not aborted, reset flip-flop NCXS	S/NCXS = N(S/CXS) (S/CXS) = FAST/A PH3 NFASTF1 + ... R/NCXS = ...	Preset for S ⟶ C in PH4
	P + 1 ⟶/⟶ P	PUC31 = FAST/A PH3 + ...	Add 1 to SPW0 address to obtain SPW1 address
	Set flip-flop MRQ if instruction aborted	S/MRQ = (S/MRQ/2) + ... (S/MRQ/2) = FASTABORT PH3 + ... R/MRQ = ...	Request for core memory cycle
	Set flip-flop DRQ if instruction aborted	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = (S/MRQ/2) + ... R/DRQ = ...	Data request, inhibits transmission of another clock until data release from core memory
PH4	One clock long		
T5L	(P0-P31) ⟶ (S0-S31)	SXP = FAST PH4 NDIS + ...	Store SPW1 address in C-register
(DR if abort)	(S0-S31) ⟶ (C0-C31)	CXS set at PH3 clock	
	If instruction not aborted, enable signal (S/SXD)	(S/SXD) = FAST PH4 NBRPH9 + ...	Preset adder logic for D ⟶ S in PH5
	Abort conditions:		
	If SW1 or SW3 set, branch to PH9	BRPH9 = FAST PH4 (SW1 + SW3)	Branch to PH9 to set condition code
	(MB0-MB31) ⟶ (C0-C31)	CXMB = DG	Load SPW1 from memory into C-register
	(C0-C31) ⟶/⟶ (D0-D31)	DXC = FASTABORT PH4	Return SPW1 to D-register
			Mnemonic: PSM (0B, 8B)

(Continued)

Table 3-70. Push Multiple Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH5	One clock long		
T5L	(D0-D31) ⟶ (S0-S31)	Adder logic set at PH4 clock	Top of stack address (SPW0) ⟶/⟶ P-register
	(S0-S31) ⟶/⟶ (P0-P31)	PXS = FAST/A PH5 + ...	
	Reset flip-flop NAXRR	S/NAXRR = N(S/AXRR) (S/AXRR) = FAST/S PH5 + ... R/NAXRR = ...	Preset for transfer of private memory R contents ⟶/⟶ A-register in PH6
PH6	One clock long		
T5L 1st Pass	(RR0-RR31) ⟶/⟶ (A0-A31)	AXRR set at PH5 clock	Store private memory register R contents in A-register
	Set flip-flop MBXS	S/MBXS = (S/MBXS) (S/MBXS) = FAST/S PH6 NMCZ	Preset for transfer of A-register contents to core memory in second PH6
	Set flip-flop MRQ	S/MRQ = (S/MRQ) (S/MRQ) = (S/MBXS) + ... R/MRQ = ...	Request for core memory cycle
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = (S/MBXS) + ... R/DRQ = ...	Data request, inhibits transmission of another clock until data release from core memory
	P + 1 ⟶/⟶ P	PUC31 = FAST/S PH6 + ...	Upcount P-register to obtain new top of stack address
	R + 1 ⟶/⟶ R	RUC31 = FAST/S PH6	Upcount R-register for next sequential private memory address
	Reset flip-flop NAXRR	S/NAXRR = N(S/AXRR) (S/AXRR) = FAST/S PH5 + ... R/NAXRR = ...	Preset for transfer of private memory contents ⟶/⟶ A-register
	MC-1 ⟶/⟶ MC	MCD7 = FAST/M PH6 NIOEN + ...	Decrement number of words in macro-counter
	Enable signal (S/SXA)	(S/SXA) = FAST/S PH6 NMCZ	Preset adder logic for A ⟶ S in next PH6
	Enable signal IOEN6 if MC $\geq$ 4	IOEN6 = FAST/A IOEN6/1 PH6 IOEN6/1 = NMC0005Z	I/O service call enable
	Sustain PH6	BRPH6 = FAST/M PH6 NMCZ	Repeat PH6 to store contents of A-register in memory
			Mnemonic: PSM (0B, 8B)

(Continued)

Table 3-70. Push Multiple Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH6	Sustained until data release		
DR	(A0-A31) ⟶ (S0-S31)	Adder logic set at first PH6 clock	Store A-register contents in memory at new top of stack address
Not 1st Pass	(S0-S31) ⟶ (MB0-MB31)	MBXS set by first PH6 clock	
	Enable signal (S/SXC) if MC = 0	(S/SXC) = FAST/S PH6 OU0 MCZ + ...	Preset adder for C ⟶ S in PH7
	P + 1 ⟶/⟶ P	PUC31 = FAST/S PH6 + ...	Upcount P-register for new top of stack address
	R + 1 ⟶ R	RUC31 = FAST/S PH6 + ...	Upcount R-register for new private memory address
	MC - 1 ⟶/⟶ MC	MCD7 = FAST/M PH6 NIOEN + ...	Decrement number of words in macro-counter
	(RR0-RR31) ⟶/⟶ (A0-A31)	AXRR set at last PH6 clock	Store private memory R contents in A-register
	Set flip-flop MBXS if MC ≠ 0	S/MBXS = (S/MBXS) (S/MBXS) = FAST/S PH6 NMCZ	Preset for transfer of A-register contents to core memory
	Set flip-flop MRQ if MC ≠ 0	S/MRQ = (S/MRQ) (S/MRQ) = (S/MBXS) + ... R/MRQ = ...	Request for core memory cycle
	Set flip-flop DRQ if MC ≠ 0	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = (S/MBXS) + ... R/DRQ = ...	Data request, inhibits transmission of another clock until data release from core memory
	Reset flip-flop NAXRR	S/NAXRR = N(S/AXRR) (S/AXRR) = FAST/S PH5 + ... R/NAXRR = ...	Preset for transfer of private memory contents ⟶/⟶ A-register
	Enable signal IOEN6 if MC ≥ 4	IOEN6 = FAST/A IOEN6/1 PH6 IOEN6/1 = NMC0005Z	I/O service call enable if number of words to be loaded ≥ 4
	Sustain PH6 if MC ≠ 0	BRPH6 = FAST/M PH6 NMCZ	Repeat PH6 to store private memory contents in core memory until number of words = 0
PH7	One clock long		
T5L	(C0-C31) ⟶ (S0-S31)	Adder logic set at first PH6 clock	SPW1 address ⟶ S
	(S0-S31) ⟶/⟶ (P0-P31)	PXS = FAST/A PH7 + ...	SPW1 address ⟶/⟶ P
	Set flip-flop MRQ	S/MRQ = (S/MRQ/2) + ... (S/MRQ/2) = FAST/A PH7 + ... R/MRQ = ...	Request for memory cycle
			Mnemonic: PSM (0B, 8B)

(Continued)

Table 3-70. Push Multiple Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH7 T5L (Cont.)	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = (S/MRQ/2) + ... R/DRQ = ...	Data request, inhibits transmission of another clock until data release from memory
PH8 DR	Sustained until data release		
	(MB0-MB31) ⟶ (C0-C31)	CXMB = DG	SPW1 from core memory ⟶ C-register
	(C0-C31) ⟶/⟶ (D0-D31)	DXC = FAST/A PH8 + ...	SPW1 ⟶/⟶ D-register
	(CC1-CC4) ⟶/⟶ (A28-A31)	AXCC = FAST/M (PH8 + ...)	Number of words ⟶/⟶ A-register
	0 ⟶/⟶ (A0-A31)	AXZ = FAST (PH8 + ...)	
	Enable signal (S/SXAPD)	(S/SXAPD) = FAST/C (PH8 + ...) + ...	Preset adder for D plus A in PH1/A
	Set flip-flop SW8	S/SW8 = NRESET BRSW8 BRSW8 = FAST/A PH8 + ...	
	Reset flip-flop NCXS	S/NCXS = N(S/CXS) (S/CXS) = FAST/A PH8 + ... R/NCXS = ...	Preset for S ⟶ C in PH1/A
	P - 1 ⟶/⟶ P	PDC31 = FAST/A PH8 + ...	Decrement P-register to obtain SPW0 address
	Reset flip-flop NT11L	S/NT11L = N(S/T11L) (S/T11L) = FAST PH8 + ... R/NT11L = ...	Set clock T11L for PH1/A
	Branch to PH1/A	BRPH1 = FAST/A PH8 + ... S/PH1 = BRPH1 NCLEAR	
PH1/A T11L	One clock long	PH1/A = PH1 SW8 FAST	
	D + A ⟶ S	Adder logic for D plus A set at PH8 clock	Update word count by adding number of words to SPW1 in D-register. Gate onto sum bus
	Force a zero into S16	S16 = (K16 ⊕ PR16) SXADD NS16INH S16INH = FAST PH1/A + ...	S16 (bit 48 of SPW1) is trap-on-word inhibit bit TW, and not included in word count
	(S16-S31) ⟶ (C16-C31)	CXS set at PH8 clock	New word count into C-register bits 16 through 31
	Zeros ⟶ (C0-C15)	CXS/0 = CXS N(FAST PH1/A) CXS/1 = CXS N(FAST PH1/A)	S0-S15 not gated onto C0-C15 because CXS/0 and CXS/1 are low
	Down align D-register	DXDR8 = FAST PH1/A + ...	Shift D-register 8 bits right as first half of 16-bit down alignment
			Mnemonic: PSM (0B, 8B)

(Continued)

Table 3-70. Push Multiple Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH1/A T11L (Cont.)	Set flip-flop SW9	S/SW9 = SW8 STEP815	
	Reset flip-flop NT8L	S/NT8L = N(S/T8L) (S/T8L) = FAST PH1 + ... R/NT8L = ...	Set clock T8L for PH1/B
	Sustain PH1	BRPH1/1 = FAST PH1 N(NSW7 PH1/C)	
PH1/B T8L	One clock long	PH1/B = PH1 SW9	
	Down align D-register	DXDR8 = FAST PH1/B + ...	Shift D-register 8 bits right to complete 16-bit down alignment. Space count is now in D17-D31
	Enable signal (S/SXDMA)	(S/SXDMA) = FAST/C PH1/B + ...	Preset adder for D minus A in PH1/C
	Set flip-flop SW10	S/SW10 = SW9 STEP815	
	Reset flip-flop NT11L	S/NT11L = N(S/T11L) (S/T11L) = FAST PH1 + ... R/NT11L = ...	Set clock T11L for PH1/C
	Sustain PH1	BRPH1/1 = FAST PH1 N(NSW7 PH1/C) + ...	
PH1/C T11L	One clock long	PH1/C = PH1 SW10	
	D - A ⟶ S	Adder logic set for D minus A in PH1/B	Update space count by subtracting number of words from D17-D31
	Force a zero into S16	S16 = (K16 ⊕ PR16) PH1/A NS16INH S16INH = FAST PH1/C	S16 is now trap-on-space bit TS, and is not included in space count
	(S0-S31) ⟶ (A0-A31)	AXS = FAST PH1/C SW7	Store new space count in A-register
	(C0-C31) ⟶ (D0-D31)	DXC = FAST PH1/C + ...	Store new word count in D-register
	Reset flip-flop SW7	R/SW7 = (R/SW7) (R/SW7) = FAST PH1/C SW7	
	Set flip-flop MRQ	S/MRQ = (S/MRQ/3) + ... (S/MRQ/3) = FAST PH1/C R/MRQ = ...	Request for core memory cycle
			Mnemonic: PSM (0B, 8B)

(Continued)

Table 3-70. Push Multiple Sequence (Cont.)

Phase	Function Performed	Signals Involved			Comments
PH1/C T11L (Cont.)	Reset flip-flop NMRQP1	S/NMRQP1	=	N(S/MRQ/3)	Delay flip-flop for data release signal
		R/NMRQP1	=	...	
	Reset flip-flop NT8L	S/NT8L	=	N(S/T8L) + ...	Set clock T8L for PH1/D
		(S/T8L)	=	FAST PH1	
		R/NT8L	=	...	
	Set flip-flop SW11	S/SW11	=	SW10 STEP815	
	Sustain PH1	BRPH1/1	=	FAST PH1 N(NSW7 PH1/C) + ...	
PH1/D T8L	One clock long	PH1/D	=	PH1 SW11	
	Up align A-register	AXAL8	=	FAST PH1/D + ...	Shift A-register 8 bits left as first half of 16-bit up alignment
	Set flip-flop DRQ	S/DRQ	=	(S/DRQ) NCLEAR	Data request, inhibits transmission of another clock until data release received from core memory
		(S/DRQ)	=	MRQP1 + ...	
	Set flip-flop SW12	S/SW12	=	SW11 STEP815	
PH1/E DR	Sustained until data release	PH1/E	=	PH1 SW12	
	(MB0-MB31) ⟶ (C0-C31)	CXMB	=	DG	SPW0 ⟶ C-register
	Up align A-register	AXAL8	=	FAST PH1/E + ...	Shift A-register 8 bits left as second half of 16-bit up alignment. New space count is now in A1 through A15
	Enable signal (S/SXAORD)	(S/SXAORD)	=	FAST PH1/E + ...	Preset adder for A OR D ⟶ S in PH1/F
	Set flip-flop A0 if TS is 1 (SW5)	S/A0	=	FAST PH1/E SW5 AXAL8 + ...	Set trap-on-space inhibit bit if set in original SPW1
	Set flip-flop A16 if TW is 1 (SW6)	S/A16	=	FAST PH1/E SW6 AXAL8 + ...	Set trap-on-word inhibit bit if set in original SPW1
	Set flip-flop MBXS	S/MBXS	=	(S/MBXS)	Preset for transfer of A OR D to core memory in PH1/F
		(S/MBXS)	=	FAST PH1/E + ...	
	Set flip-flop MRQ	S/MRQ	=	(S/MRQ)	Request for core memory cycle
		(S/MRQ)	=	(S/MBXS) + ...	
		R/MRQ	=	...	
					Mnemonic: PSM (0B, 8B)

(Continued)

Table 3-70. Push Multiple Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH1/E DR (Cont.)	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = (S/MBXS) + ... R/DRQ = ...	Data request, inhibits transmission of another clock until data release from memory
	P + 1 → P	PUC31 = FAST PH1/E	Increment P-register to obtain SPW1 address
	Set flip-flop SW13	S/SW13 = SW12 STEP815	
PH1/F DR	Sustained until data release	PH1/F = PH1 SW13	
	A OR D → S	Adder logic set at PH1/E clock	New word count in D-register and new space count in A-register → S
	(S0-S31) → (MB0-MB31)	MBXS set by PH1/E clock	Store new space count and word count in core memory at SPW1
	Set flip-flop MBXS	S/MBXS = (S/MBXS) (S/MBXS) = FAST PH1/F + ... R/MBXS = ...	Preset for memory write
	Set flip-flop MRQ	S/MRQ = (S/MRQ) (S/MRQ) = (S/MBXS) + ... R/MRQ = ...	Request for core memory cycle
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = (S/MBXS) + ... R/DRQ = ...	Data request, inhibits transmission of another clock until data release received from core memory
	(C0-C31) → (D0-D31)	DXC = FAST PH1/F + ...	Top of stack address (SPW0) in C-register clocked into D-register
	Enable signal (S/SXAPD)	(S/SXAPD) = FAST/C (PH1/F + ...) + ...	Preset adder for D plus A in PH1/G
	P - 1 → P	PDC31 = FAST PH1/F + ...	Decrement P-register to obtain SPW0 address
	(CC1-CC4) → (A28-A31)	AXCC = FAST/M (PH1/F + ...) + ...	Number of words → A-register
	Set flip-flop SW14	S/SW14 = SW13 STEP815	
	Sustain PH1	BRPH1/1 = FAST PH1 N(NSW7 PH1/C) + ...	
			Mnemonic: PSM (0B, 8B)

(Continued)

Table 3-70. Push Multiple Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH1/G	Sustained until data release	PH1/G = SW14 PH1	
DR	D + A ⟶ S	Adder logic set at PH1/F clock	Add number of words to top of stack address in D-register to obtain new top of stack address
	(S0-S31) ⟶ (MB0-MB31)	MBXS set by PH1/F clock	Store new top of stack address in memory at SPW0 location
	Branch to PH9	BRPH9 = FAST PH1/G S/PH9 = BRPH9 NCLEAR + ... R/PH9 = ...	
PH9	One clock long		
T5L	(B0-B31) ⟶ (S0-S31)	SXB = PXSXB NDIS PXSXB = NFAFL NFAMDS PH9	Program address ⟶ P-register via sum bus
	(S0-S31) ⟶ (P0-P31)	PXS = PXSXB	
	Set condition code:		
	Set CC3 if word count overflow and TW = 1 (SW6)	S/CC3 = (S/CC3/1) + ... (S/CC3/1) = FAST PH9 SW3	SW3 indicates word count overflow. If TW were 0, instruction would have trapped and not reached PH9
	Set CC1 if space count underflow and TS = 1 (SW5)	S/CC1 = (S/CC1/1) + ... (S/CC1/1) = FAST PH9 SW1	SW1 indicates space count underflow. If TS were 0, instruction would have trapped and not reached PH9
	Set CC2 if new space count = 0	S/CC2 = (S/CC2/1) + ... (S/CC2/1) = (FASTNABORT PH9) SW2 + ...	If instruction is successfully completed and stack is full, CC2 is set
	Place zeros in condition code flip-flops not set by this instruction	R/CC = FAST PH9 + ...	Places inputs on reset sides of CC1 through CC4 so that they will be reset if not set by this instruction
	Enable signal (S/SXD) if instruction aborted	(S/SXD) = FASTABORT PH9	Preset adder for D ⟶ S in PH10
			Mnemonic: PSM (0B, 8B)

(Continued)

Table 3-70. Push Multiple Sequence (Cont.)

Phase	Function Performed	Signals Involved			Comments
PH10	Sustained until data release				
DR	Normal ENDE				
	If instruction aborted:				
	Correct CC4	S/CC4	=	(S/CC4/2) + . . .	Set CC4 if original word count (in D-register) = 0
		(S/CC4/2)	=	(FASTABORT ENDE) S1631Z	
	Correct CC2	S/CC2	=	(S/CC2/4) + . . .	Set CC2 if original space count (in D-register) = 0
		(S/CC2/4)	=	S0007Z S0815Z (FASTABORT ENDE)	
	Force ones into S16 and S0	SGTZ	=	N(FASTABORT ENDE)	To prevent setting CC3
		S16	=	N(FASTABORT ENDE)	S16 is TW inhibit bit; S0 is TS inhibit bit
		S0	=	N(FASTABORT ENDE)	Neither should be checked for 0
					Mnemonic: PSM (0B, 8B)

PULL MULTIPLE (PLM; 0A, 8A). The PLM instruction loads a sequential set of private memory registers from the push-down stack defined by the stack pointer doubleword, which is located at the address specified in the reference address field of the PLM instruction. The number of words to be pulled is indicated by the condition code. If a total of 16 words are to be pulled from the stack, the initial value of the condition code is 0000. The private memory registers are treated as a circular set, with register 0 following register 15. The first private memory register to be loaded from the stack is the register specified in the R field of the instruction plus the condition code minus 1, and the contents of the current top of stack location become the contents of this register. The last private memory register to be loaded is the register specified in the R field of the instruction.

Registers R + CC-1 to register R are loaded in descending sequence, beginning with the contents of the location pointed to by the current top of stack address and ending with the contents of the location pointed to by the current top of stack address minus CC-1.

The current top of stack address is decremented by the value of the condition code to point to the new top of stack location. The space count is incremented by the value of the condition code, and the word count is decremented by the value of the condition code. The condition code is set as described under Stack Pointer Doubleword (page 3-438) to reflect the new status of the space count and word count.

If the space count or word count limits would be exceeded by the instruction, the instruction is aborted and a trap routine is entered if allowed by the TS or TW bit. The condition code is set as described under Stack Pointer Doubleword (page 3-438).

PULL MULTIPLE PHASE SEQUENCE. Preparation phases for the PLM instruction are the same as the general PREP phases for word instructions, paragraph 3-59. Figure 3-174 shows the simplified phase sequence for the PLM instruction. Table 3-71 lists the detailed logic sequence during all PLM execution phases. During the first pass through the phase 1 phases, word count underflow and space count overflow are checked in the adder and indicators are set, but the adder output is not used. The instruction branches from PH1/C to PH2 and obtains the top of stack address before PH6. The instruction loops through PH6, loading words from core memory

into private memory, the number of loops depending on the number of words to be pulled. When a zero value in the macro-counter indicates that the last word has been loaded, the instruction proceeds to PH7, and from PH8 branches back to PH1/A. From PH1/A to PH1/G, the new top of stack address, new space count, and new word count are calculated and stored in core memory in the stack pointer doubleword. After PH1/G, PH9 is entered to obtain the address of the next instruction, and PH10 enables the ENDE operation to take place.

If the condition code at the beginning of the instruction contains 0000, indicating that all 16 private memory registers are involved in the pull operation, bit 3 of the macro-counter is set at the time the condition code is transferred to the A-register, thereby establishing 10000 as the number of words.

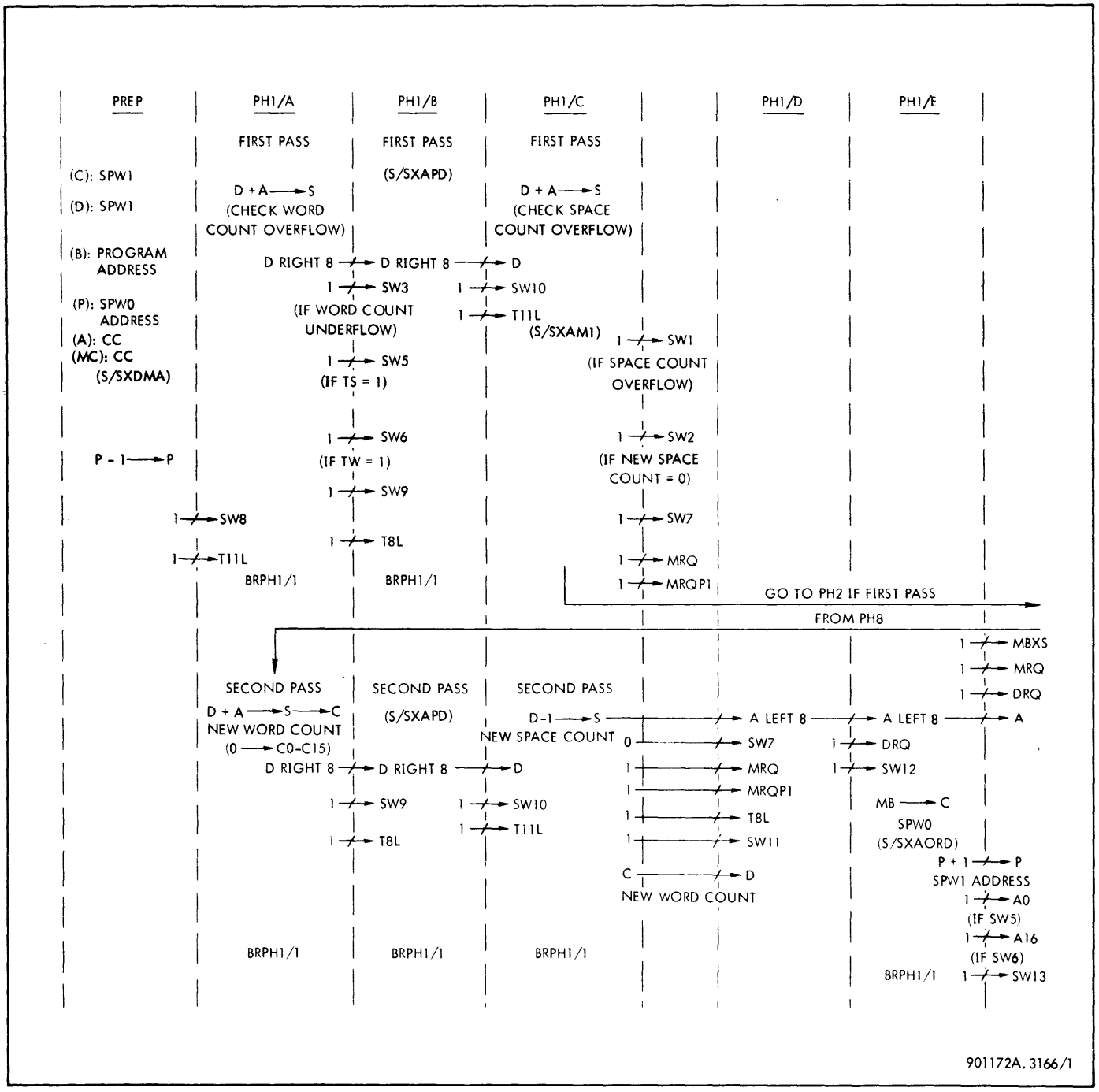


Figure 3-174. Pull Multiple Instruction, Phase Sequence Diagram (Sheet 1 of 3)

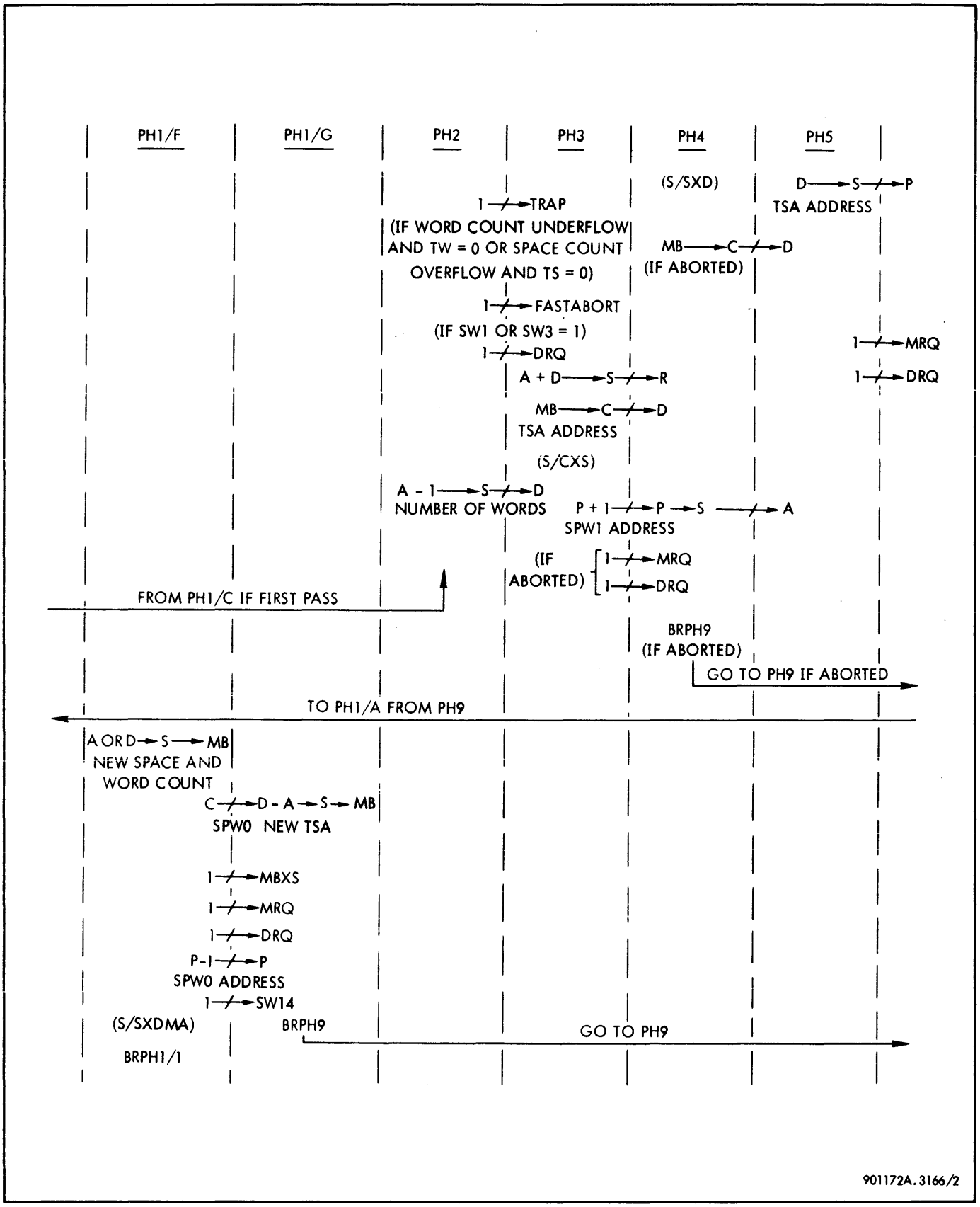


Figure 3-174. Pull Multiple Instruction, Phase Sequence Diagram (Sheet 2 of 3)

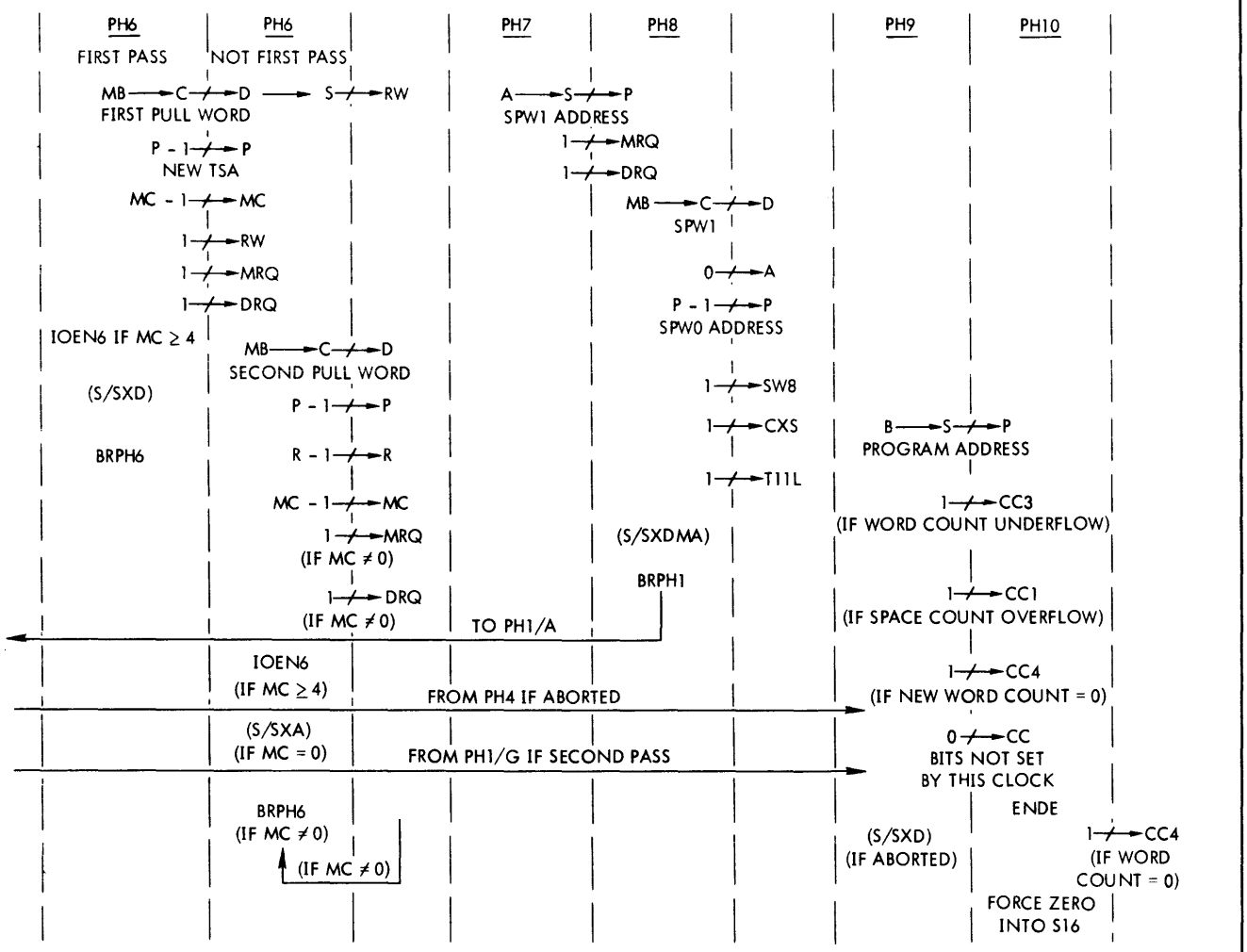


Figure 3-174. Pull Multiple Instruction, Phase Sequence Diagram (Sheet 3 of 3)

Table 3-71. Pull Multiple Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(C) : SPW1		Stack pointer doubleword 1
	(D) : SPW1		Stack pointer doubleword 1
	(B) : Program address		Address of next instruction in sequence
	(P) : SPWD address		Location of bits 0-31 of stack pointer doubleword
			Mnemonic: PLM (0A, 8A)

(Continued)

Table 3-71. Pull Multiple Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PREP (Cont.)	(A) : CC (number of words)		A-register contains number of words
	(MC) : CC (number of words)		Macro-counter set to number of words
	Preset conditions with PRE3:		
	Enable signal (S/SXDMA)	(S/SXDMA) = FUPLM (PRE3 + ...) + ...	Preset adder for D minus A in PH1/A
	Set flip-flop SW8	S/SW8 = BRSW8 NRESET/A BRSW8 = FAST PRE3 + ...	
	Reset flip-flop NT11L	S/NT11L = N(S/T11L) (S/T11L) = FAST PRE3 + ...	Set clock T11L for PH1/A
PH1/A T11L	One clock long	PH1/A = PH1 SW8	
	D - A→S	Adder logic set at last PREP clock	Subtract number of words from word count in SPW1 to check for underflow
	Force a zero into S16	S16INH = FAST PH1/A	Inhibit TW
	Set SW3 if word count underflows	S/SW3 = (S/SW3) (S/SW3) = (A16 ⊕ K16) FAST PH1/A + ...	Word count underflows into adder bit 16
	Set SW5 if TS is 1	S/SW5 = (S/SW5) (S/SW5) = FAST PH1/A D0 + ...	Trap-on-space inhibit bit is in D0
	Set SW6 if TW is 1	S/SW6 = (S/SW6) (S/SW6) = FAST PH1/A D16 + ...	D16 contains trap-on-word inhibit bit TW
	Down align D-register	DXDR8 = FAST PH1/A + ...	Shift D-register 8 bits right as first half of 16-bit down alignment
	Set SW4 if word count = 0	S/SW4 = (S/SW4) (S/SW4) = N(A16 ⊕ K16) S1631Z FAST PH1/A	New word count = 0 if S16-S31 = 0
	Set flip-flop SW9	S/SW9 = SW8 STEP815 STEP815 = NBRSW8 NBRSW10 NBRSW11 NBRSW12 NBRSW13 NBRSW15	
	Reset flip-flop NT8L	S/NT8L = N(S/T8L) (S/T8L) = FAST PH1 R/NT8L = ...	Set clock T8L for PH1/B
	Sustain PH1	BRPH1/1 = FAST PH1 N(NSW7 PH1/C)	Hold PH1 for PH1/B
			Mnemonic: PLM (0A, 8A)

(Continued)

Table 3-71. Pull Multiple Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH1/B	One clock long	PH1/B = PH1 SW9	
T8L	Down align D-register	DXDR8 = FAST PH1/B + ...	Shift D-register 8 bits right to complete 16-bit down alignment. Space count is now in D17 through D31
	Enable signal (S/SXAPD)	(S/SXAPD) = FUPLM PH1/B + ...	Preset adder for D plus A
	Set flip-flop SW10	S/SW10 = SW9 STEP815	
	Reset flip-flop NT11L	S/NT11L = N(S/T11L) (S/T11L) = FAST PH1/B R/NT11L = ...	Set clock T11L for PH1/C
	Sustain PH1	BRPH1/1 = FAST PH1 N(NSW7 PH1/C) + ...	Hold PH1 for PH1/C
PH1/C	One clock long	PH1/C = PH1 SW10	
T11L	D + A ⟶ S	Adder logic set at PH1/B clock	Add number of words to space count in D17 through D31 for overflow check only
	Set SW1 if space count overflows	S/SW1 = (S/SW1) (S/SW1) = (A16 ⊕ K16) FAST PH1/C + ...	Space count overflows into adder bit 16
	Set flip-flop SW7	S/SW7 = (S/SW7) (S/SW7) = FAST PH1/C NSW7 + ...	
	Set flip-flop MRQ	S/MRQ = (S/MRQ/3) + ... (S/MRQ/3) = FAST PH1/C + ... R/MRQ = ...	Request for core memory cycle
	Reset flip-flop NMRQP1	S/NMRQP1 = N(S/MRQ/3) R/NMRQP1 = ...	Delay flip-flop for data release signal
	Enable signal (S/SXAM1)	(S/SXAM1) = FUPLM PH1/C NSW7 + ...	Preset adder for A minus 1 in PH2. Go to PH2 if abort or first pass
PH2	One clock long		
T5L	Trap conditions: Set flip-flop TRAP if word count underflows and TW = 0 or if space count overflows and TS = 0	S/TRAP = (S/TRAP) NRESET (S/TRAP) = FAST PH2 SW3 NSW6 + FAST PH2 SW1 NSW5	SW3 is word count underflow, SW1 is space count overflow, NSW6 ⟹ TW = 0, NSW5 ⟹ TS = 0
	A - 1 ⟶ S	Adder set at PH1/C clock	Subtract 1 from number of words in preparation for finding starting register
			Mnemonic: PLM (0A, 8A)

(Continued)

Table 3-71. Pull Multiple Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH2 T5L (Cont.)	(S0-S31)—/→(D0-D31)	DXS = FUPLM PH2 + ...	Hold number of words minus 1 in D-register
	(R0-R31)—/→(A0-A31)	AXR = FUPLM PH2	Prepare to find starting register
	Abort if SW1 or SW3 is set	S/FASTABORT = FAST PH2 SW1 + FAST PH2 SW3 S/FASTF1 = SW3 + SW1	Instruction unconditionally aborted on overflow or underflow. Note that FASTABORT is built with two flip-flops, FASTF1 and FASTF2
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = MRQP1 + ... R/DRQ = ...	Data request, inhibits transmission of another clock until data release received from core memory
	Enable signal (S/SXAPD)	(S/SXAPD) = FUPLW PH2 + ...	Preset adder for A plus D in PH3
PH3	Sustained until data release		
DR	A + D——→S	Adder preset at PH2 clock	Add number of words minus 1 to private memory address to determine starting register
	(S0-S31)—/→(R0-R31)	RXS = FUPLW PH3 + ...	Place private memory starting address in R-register
	(MB0-MB31)——→(C0-C31)	CXMB = DG (data gate)	Top of stack address from memory——→C-register
	(C0-C31)—/→(D0-D31)	DXC = FAST/A PH3	Top of stack address —/→D-register
	P + 1—/→P	PUC31 = FAST/A PH3 + ...	Add 1 to SPW0 address to obtain SPW1 address
	Set flip-flop MRQ if instruction aborted	S/MRQ = (S/MRQ/2) + ... (S/MRQ/2) = FASTABORT PH3 + ... R/MRQ = ...	Request for core memory cycle
	Set flip-flop DRQ if instruction aborted	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = (S/MRQ/2) + ... R/DRQ = ...	Data request, inhibits transmission of another clock until data release from core memory
PH4	One clock long		
T5L	(P0-P31)——→(S0-S31)	SXP = FAST PH4 NDIS	Hold SPW1 address in A-register
(DR if abort)	(S0-S31)—/→(A0-A31)	AXS = FAST PH4	
			Mnemonic: PLM (0A, 8A)

(Continued)

Table 3-71. Pull Multiple Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH4 T5L (DR if abort) (Cont.)	If instruction not aborted, enable signal (S/SXD) Abort conditions: If SW1 or SW3 set, branch to PH9 (MB0-MB31)⟶(C0-C31) (C0-C31)⟶(D0-D31)	(S/SXD) = FAST PH4 NBRPH9 BRPH9 = FAST PH4 (SW1 + SW3) CXMB = DG DXC = FASTABORT PH4	Preset adder logic for D⟶S in PH5 Branch to PH9 to set condition code Load SPW1 from memory into C-register Return SPW1 to D-register
PH5 T5L	One clock long (D0-D31)⟶(S0-S31) (S0-S31)⟶(P0-P31) Set flip-flop MRQ Set flip-flop DRQ	Adder logic set at PH4 clock PXS = FAST/A PH5 + ... S/MRQ = (S/MRQ/2) + ... (S/MRQ/2) = FAST/L PH5 R/DRQ = ... S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = S/MRQ/2 R/DRQ = ...	Top of stack address (SPW0)⟶P-register Request for core memory cycle Data request, inhbits transmission of another clock until data release from memory
PH6 DR 1st Pass	Sustained until data release (MB0-MB31)⟶(C0-C31) (C0-C31)⟶(D0-D31) Set flip-flop RW P - 1⟶P MC - 1⟶MC Enable signal (S/SXD) Set flip-flop MRQ	CXMB = DG DXC = FAST/L PH6 + ... S/RW = (S/RW) + ... (S/RW) = FAST/L PH6 NMCZ + ... PDC31 = FAST/L PH6 OU0 + ... MDC7 = FAST/M PH6 NIOEN + ... (S/SXD) = FAST/L PH6 NMCZ + ... S/MRQ = (S/MRQ/2) + ... (S/MRQ/2) = FAST/L PH6 NMCZ + ... R/MRQ = ...	Load first pull word from top of stack address in memory⟶C-register Place first pull word in D-register for transfer to private memory Prepare to write into private memory Decrement P-register to obtain new top of stack address Decrement macro-counter by 1 Preset adder logic for D⟶S in second PH6 Request for core memory cycle
			Mnemonic: PLM (0A, 8A)

(Continued)

Table 3-71. Pull Multiple Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH6 DR 1st Pass (Cont.)	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = (S/MRQ/2) + ... R/DRQ = ...	Data request, inhibits transmission of another clock until data release from memory
	Enable signal IOEN6 if MC $\geq$ 4	IOEN6 = IOEN6/1 PH6 + ... IOEN6/1 = NMC0005Z	I/O service call enable
	Sustain PH6	BRPH6 = FAST/M PH6 NMCZ + ...	Repeat PH6 to store contents of D-register in private memory
PH6	Sustained until data release		
DR Not 1st Pass	(D0-D31) $\longrightarrow$ (S0-S31)	Adder logic set at first PH6 clock	Transfer first pull word to private memory via S-register
	(S0-S31) $\not\longrightarrow$ (RW0-RW31) (MB0-MB31) $\longrightarrow$ (C0-C31)	RWXS = RW CXMB = DG	Read subsequent words from core memory and place in D-register
	(C0-C31) $\not\longrightarrow$ (D0-D31)	DXC = FAST/L PH6 + ...	
	P - 1 $\not\longrightarrow$ P	PDC31 = FAST/L PH6 OU0 + ...	Decrement P-register for address of next core memory word
	R - 1 $\not\longrightarrow$ R	RDC31 = FAST/L PH6 OU0 + ...	Decrement R-register for address of next private memory register
	Enable signal IOEN6 if MC $\geq$ 4	IOEN6 = IOEN6/1 PH6 + ... IOEN6/1 = NMC0005Z	I/O service call
	Enable signal (S/SXA) if MC = 0	(S/SXA) = FAST/L PH6 OU0 MCZ + ...	Preset adder for A $\longrightarrow$ S in PH7
	MC - 1 $\not\longrightarrow$ MC	MCD7 = FAST/M PH6 NIOEN + ...	Decrement macro-counter to obtain new number of words
	Set flip-flop MRQ if MC $\neq$ 0	S/MRQ = (S/MRQ/2) + ... (S/MRQ/2) = FAST/L PH6 NMCZ + ... R/MRQ = ...	Request for core memory cycle
	Set flip-flop DRQ if MC $\neq$ 0	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = (S/MRQ/2) + ... R/DRQ = ...	Data request, inhibits transmission of another clock until data release from memory
	Sustain PH6 if MC $\neq$ 0	BRPH6 = FAST/M PH6 NMCZ + ...	Repeat PH6 if more words are to be pulled
			Mnemonic: PLM (0A, 8A)

(Continued)

Table 3-71. Pull Multiple Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH7	One clock long		
T5L	(A0-A31) ⟶ (S0-S31)	Adder logic set at first PH6 clock	SPW1 address ⟶ S
	(S0-S31) ⟶/⟶ (P0-P31)	PXS = FAST/A PH7 + ...	SPW1 address ⟶/⟶ P
	Set flip-flop MRQ	S/MRQ = (S/MRQ/2) + ...	Request for memory cycle
		(S/MRQ/2) = FAST/A PH7 + ...	
		R/MRQ = ...	
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR	Data request, inhibits transmission of another clock until data release from memory
		(S/DRQ) = (S/MRQ/2) + ...	
		R/DRQ = ...	
PH8	Sustained until data release		
DR	(MB0-MB31) ⟶ (C0-C31)	CXMB = DG	SPW1 from core memory ⟶ C-register
	(C0-C31) ⟶/⟶ (D0-D31)	DXC = FAST/A PH8 + ...	SPW1 ⟶/⟶ D-register
	Zeros ⟶/⟶ (A0-A31)	AXZ = FAST (PH8 + ...)	Clear A-register for word count and space count
	Enable signal (S/SXDMA)	(S/SXDMA) = FUPLM (PH8 + ...) + ...	Preset adder for D minus A in PH1/A
	Set flip-flop SW8	S/SW8 = NRESET BRSW8	
		BRSW8 = FAST/A PH8 + ...	
	Reset flip-flop NCXS	S/NCXS = N(S/CXS)	Preset for S ⟶ C in PH1/A
		(S/CXS) = FAST/A PH8 + ...	
		R/NCXS = ...	
	P - 1 ⟶/⟶ P	PDC31 = FAST/A PH8 + ...	Decrement P-register to obtain SPW0 address
	Reset flip-flop NT11L	S/NT11L = N(S/T11L)	Set clock T11L for PH1/A
		(S/T11L) = FAST PH8 + ...	
		R/NT11L = ...	
	Branch to PH1/A	BRPH1 = FAST/A PH8 + ...	
		S/PH1 = BRPH1 NCLEAR	
PH1/A	One clock long	PH1/A = PH1 SW8 FAST	
T11L	D - A ⟶ S	Adder logic for D minus 1 set at PH8 clock	Update word count by subtracting number of words from SPW1 in D-register. Gate onto sum bus
			Mnemonic: PLM (0A, 8A)

(Continued)

Table 3-71. Pull Multiple Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH1/A T11L (Cont.)	Force a zero into S16	S16 = (K16 ⊕ PR16) SXADD NS16INH S16INH = FAST PH1/A + ...	S16 (bit 48 of SPW1) is trap-on-word inhibit bit TW, and not included in word count
	(S16-S31)⟶(C16-C31)	CXS set at PH8 clock	New word count into C-register bits 17 through 31
	Zeros⟶(C0-C15)	CXS/0 = CXS N(FAST PH1/A) CXS/1 = CXS N(FAST PH1/A)	S0-S15 not gated into C0-C15 because CXS/0 and CXS/1 are low
	Down align D-register	DXDR8 = FAST PH1/A + ...	Shift D-register 8 bits right as first half of 16-bit down alignment
	Set flip-flop SW9	S/SW9 = SW8 STEP815	
	Reset flip-flop NT8L	S/NT8L = N(S/T8L) (S/T8L) = FAST PH1 + ... R/NT8L = ...	Set clock T8L for PH1/B
	Sustain PH1	BRPH1/1 = FAST PH1 N(NSW7 PH1/C) + ...	
PH1/B T8L	One clock long	PH1/B = PH1 SW9	
	Down align D-register	DXDR8 = FAST PH1/B + ...	Shift D-register 8 bits right to complete 16-bit down alignment. Space count is now in D17-D31
	Enable signal (S/SXAPD)	(S/SXAPD) = FUPLM PH1/B + ...	Preset adder for D plus A in PH1/C
	Set flip-flop SW10	S/SW10 = SW9 STEP815	
	Reset flip-flop NT11L	S/NT11L = N(S/T11L) (S/T11L) = FAST PH1 + ... R/NT11L = ...	Set clock T11L for PH1/C
	Sustain PH1	BRPH1/1 = FAST PH1 N(NSW1 PH1/C) + ...	
PH1/C T11L	One clock long	PH1/C = PH1 SW10	
	D + A⟶S	Adder logic set for D plus A in PH1/B	Update space count by adding number of words to D17-D31
			Mnemonic: PLM (0A, 8A)

(Continued)

Table 3-71. Pull Multiple Sequence (Cont.)

Phase	Function Performed	Signals Involved			Comments
PH1/C T11L (Cont.)	Force a zero into S16	S16	=	NS16INH (...)	S16 is now trap-on-space inhibit bit TS and is not included in space count
		S16INH	=	FAST PH1/C	
	(S0-S31) —/→ (A0-A31)	AXS	=	FAST PH1/C SW7 + ...	Hold new space count in A-register
	(C0-C31) —/→ (D0-D31)	DXC	=	FAST PH1/C + ...	Hold new word count in D-register
	Reset flip-flop SW7	R/SW7	=	(R/SW7)	
		(R/SW7)	=	FAST PH1/C SW7 + ...	
	Set flip-flop MRQ	S/MRQ	=	(S/MRQ/3) + ...	Request for core memory cycle
		(S/MRQ/3)	=	FAST PH1/C	
		R/MRQ	=	...	
	Reset flip-flop NMRQP1	S/NMRQP1	=	N(S/MRQ/3)	Delay flip-flop for data release signal
		R/NMRQP1	=	...	
	Reset flip-flop NT8L	S/NT8L	=	N(S/T8L) + ...	Set clock T8L for PH1/D
		(S/T8L)	=	FAST PH1 + ...	
		R/NT8L	=	...	
	Set flip-flop SW11	S/SW11	=	SW10 STEP815	
	Sustain PH1	BRPH1/1	=	FAST PH1 N(NSW7 PH1/C) + ...	
PH1/D T8L	One clock long	PH1/D	=	PH1 SW11	
	Up align A-register	AXAL8	=	FAST PH1/D + ...	Shift A-register 8 bits left as first half of 16-bit up alignment
	Set flip-flop DRQ	S/DRQ	=	(S/DRQ) NCLEAR	Data request, inhibits transmission of another clock until data release received from core memory
		(S/DRQ)	=	MRQP1 + ...	
		R/DRQ	=	...	
	Set flip-flop SW12	S/SW12	=	SW11 STEP815	
PH1/E DR	Sustained until data release	PH1/E	=	PH1 SW12	
	(MB0-MB31) ——→ (C0-C31)	CXMB	=	DG	SPW0 (TSA) ——→ C-register
	Up align A-register	AXAL8	=	FAST PH1/E + ...	Shift A-register 8 bits left as second half of 16-bit up alignment. New space count is now in A1 through A15
	Enable signal (S/SXAORD)	(S/SXAORD)	=	FAST PH1/E + ...	Preset adder for A OR D ——→ S in PH1/F
					Mnemonic: PLM (0A, 8A)

(Continued)

Table 3-71. Pull Multiple Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH1/E DR (Cont.)	Set flip-flop A0 if TS is 1 (SW5)	S/A0 = FAST PH1/E SW5 AXAL8	Set trap-on-space inhibit bit if set in original SPW1
	Set flip-flop A16 if TW is 1 (SW6)	S/A16 = FAST PH1/E SW6 AXAL8 + ...	Set trap-on-word inhibit bit if set in original SPW1
	Set flip-flop MBXS	S/MBXS = (S/MBXS) (S/MBXS) = FAST PH1/E + ... R/MBXS = ...	Preset for transfer of A OR D to core memory in PH1/F
	Set flip-flop MRQ	S/MRQ = (S/MRQ) + ... (S/MRQ) = (S/MBXS) + ... R/MRQ = ...	Request for core memory cycle
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = (S/MBXS) + ... R/DRQ = ...	Data request, inhibits transmission of another clock until data release from memory
	P + 1 ⟶ P	PUC31 = FAST PH1/E + ...	Increment P-register to obtain SPW1 address
	Set flip-flop SW13	S/SW13 = SW12 STEP815	
	Sustain PH1	BRPH1/1 = FAST PH1.N(NSW7 PH1/C) + ...	
PH1/F DR	Sustained until data release	PH1/F = PH1 SW13	
	A OR D ⟶ S	Adder logic set at PH1/E clock	New word count in D-register and new space count in A-register ⟶ S
	(S0-S31) ⟶ (MB0-MB31)	MBXS set by PH1/E clock	Store new space count and word count in core memory at SPW1 location
	Set flip-flop MBXS	S/MBXS = (S/MBXS) + ... (S/MBXS) = FAST PH1/F + ... R/MBXS = ...	Preset for memory write
	Set flip-flop MRQ	S/MRQ = (S/MRQ) + ... (S/MRQ) = (S/MBXS) + ... R/MRQ = ...	Request for core memory cycle
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = (S/MBXS) + ... R/DRQ = ...	Data request, inhibits another clock until data release received from core memory
	(C0-C31) ⟶ (D0-D31)	DXC = FAST PH1/F + ...	Top of stack address (SPW0) in C-register clocked into D-register
	Enable signal (S/SXDMA)	(S/SXDMA) = FUPLM (PH1/F + ...) + ...	Preset adder for D minus A in PH1/G
	P - 1 ⟶ P	PDC31 = FAST PH1/F + ...	Decrement P-register to obtain SPW0 address
			Mnemonic: PLM (0A, 8A)

(Continued

Table 3-71. Pull Multiple Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH1/F DR (Cont.)	(CC1-CC4)⟶(A28-A31)	AXCC = FAST/M (PH1/F + ...) + ...	Number of words⟶/⟶ A-register
	Set flip-flop SW14	S/SW14 = SW13 STEP815	
	Sustain PH1	BRPH1/1 = FAST PH1 N(NSW7 PH1/C) + ...	
PH1/G DR	Sustained until data release	PH1/G = SW14 PH1	
	D - A⟶S	Adder logic set at PH1/F clock	Subtract number of words from top of stack address in D-register to obtain new top of stack address
	(S0-S31)⟶(MB0-MB31)	MBXS set by PH1/F clock	Store new top of stack address in memory at SPW0 location
	Branch to PH9	BRPH9 = FAST PH1/G	
		S/PH9 = BRPH9 NCLEAR + ...	
		R/PH9 = ...	
PH9 T5L	One clock long		
	(B0-B31)⟶(S0-S31)	SXB = PXSXB NDIS	Program address⟶/⟶P-register via sum bus
		PXSXB = NFAFL NFAMDS PH9	
	(S0-S31)⟶/⟶(P0-P31)	PXS = PXSXB	
	Set condition code:		
	Set CC3 if word count underflow and TW = 1 (SW6)	S/CC3 = (S/CC3/1) + ...	SW3 indicates word count underflow. If TW were 0, instruction would have trapped and not reached PH9
		(S/CC3/1) = FAST PH9 SW3 + ...	
		R/CC3 = ...	
	Set CC1 if space count overflow and TS = 1 (SW5)	S/CC1 = (S/CC1/1) + ...	SW1 indicates space count overflow. If TS were 0, instruction would have trapped and not reached PH9
		(S/CC1/1) = FAST PH9 SW1 + ...	
	Set CC4 if new word count = 0	S/CC4 = (S/CC4/1) + ...	If instruction is successfully completed and stack is empty, CC4 is set
		(S/CC4/1) = (FASTNABORT PH9) SW4 + ...	
		R/CC = FAST PH9 + ...	Reset inputs to CC flip-flops to reset those not set in this phase
			Mnemonic: PLM (0A, 8A)

(Continued)

Table 3-71. Pull Multiple Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH9 T5L (Cont.)	Enable signal (S/SXD) if instruction aborted	(S/SXD) = FASTABORT PH9	Preset adder for D ⟶ S in PH10
PH10 DR	Sustained until data release Normal ENDE If instruction aborted:		
	Correct CC2	(S/CC2/4) = (FASTABORT ENDE) S1631Z	Set CC2 if original space count (in D-register) = 0
	Correct CC4	(S/CC4/2) = (FASTABORT ENDE) S1631Z	Set CC4 if word count (in D-register) = 0
	Force zeros into S0, S16, and SGTZ	SGTZ = N(FASTABORT ENDE)	To prevent setting CC3
		S16 = N(FASTABORT ENDE)	S16 is TW inhibit bit. S0 is TS inhibit bit. Neither should be checked for 0
		S0 = N(FASTABORT ENDE)	
			Mnemonic: PLM (0A, 8A)

MODIFY STACK POINTER (MSP; 13, 93). The modify stack pointer instruction changes the stack pointer doubleword located at the address specified in the reference address field of the MSP instruction. The amount of change is determined by the contents of the private memory register specified in the R field of the instruction. The private memory word contains the signed modifier in bits 16 through 31; bits 0 through 15 are insignificant. A negative integer used as a modifier is expressed in two's complement form as a fixed-point halfword.

The modifier is algebraically added to the top of stack address, subtracted from the space count, and added to the word count in the stack pointer doubleword. If as a result of the addition the space count or word count would be decreased below zero or increased above $2^{15}-1$, the instruction is aborted. The operations performed in this case are described under Stack Pointer Doubleword (page 3-438). If the instruction is successfully executed, the condition code is set as described under Stack Pointer Doubleword.

MODIFY STACK POINTER PHASE SEQUENCE. Preparation phases for the MSP instruction are the same as the general PREP phases for word instructions, paragraph 3-59. Figure 3-175 shows the simplified phase sequence for the MSP instruction. Table 3-72 lists the detailed logic sequence during all execution phases of the instruction.

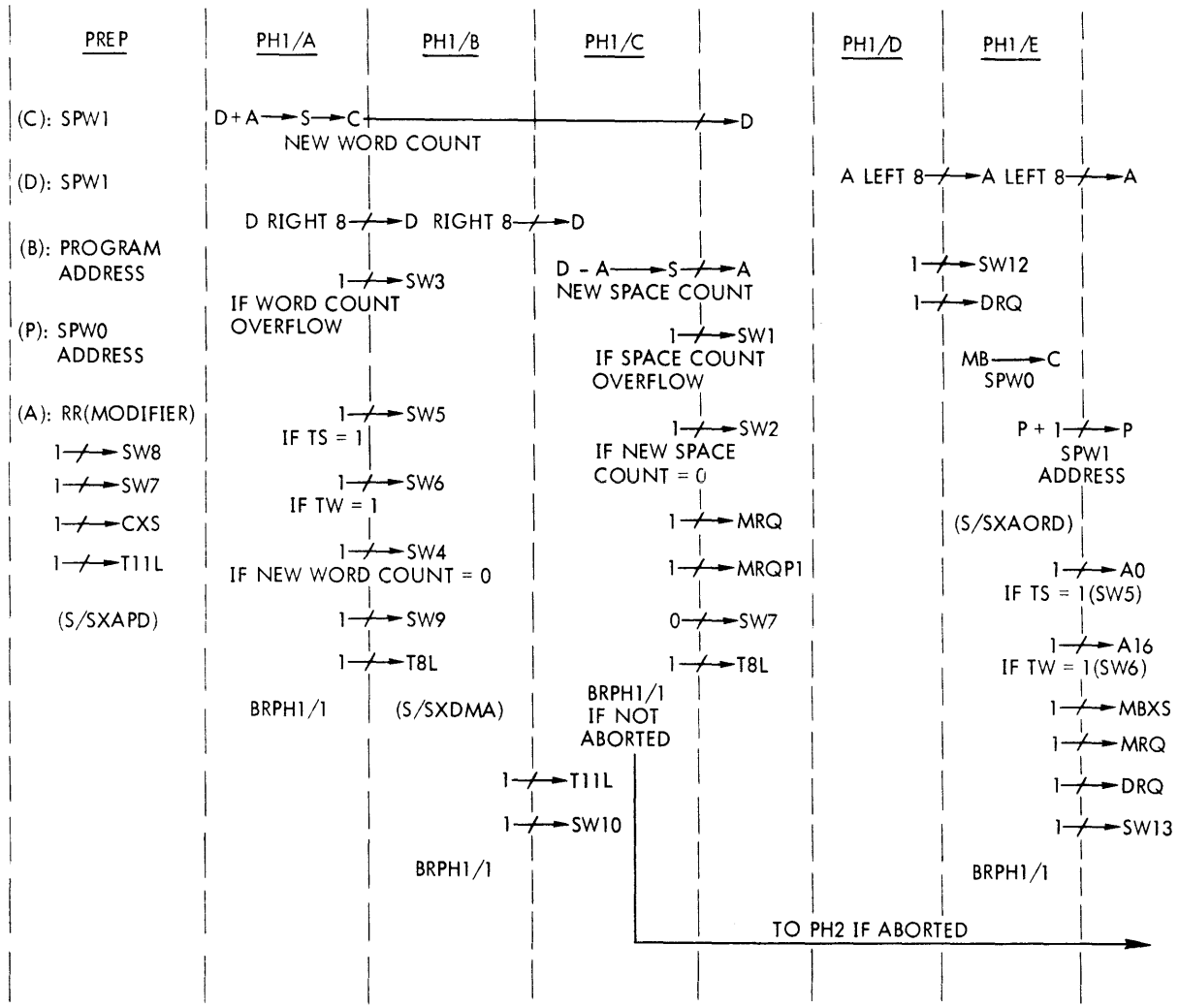


Figure 3-175. Modify Stack Pointer Instruction, Phase Sequence Diagram (Sheet 1 of 2)

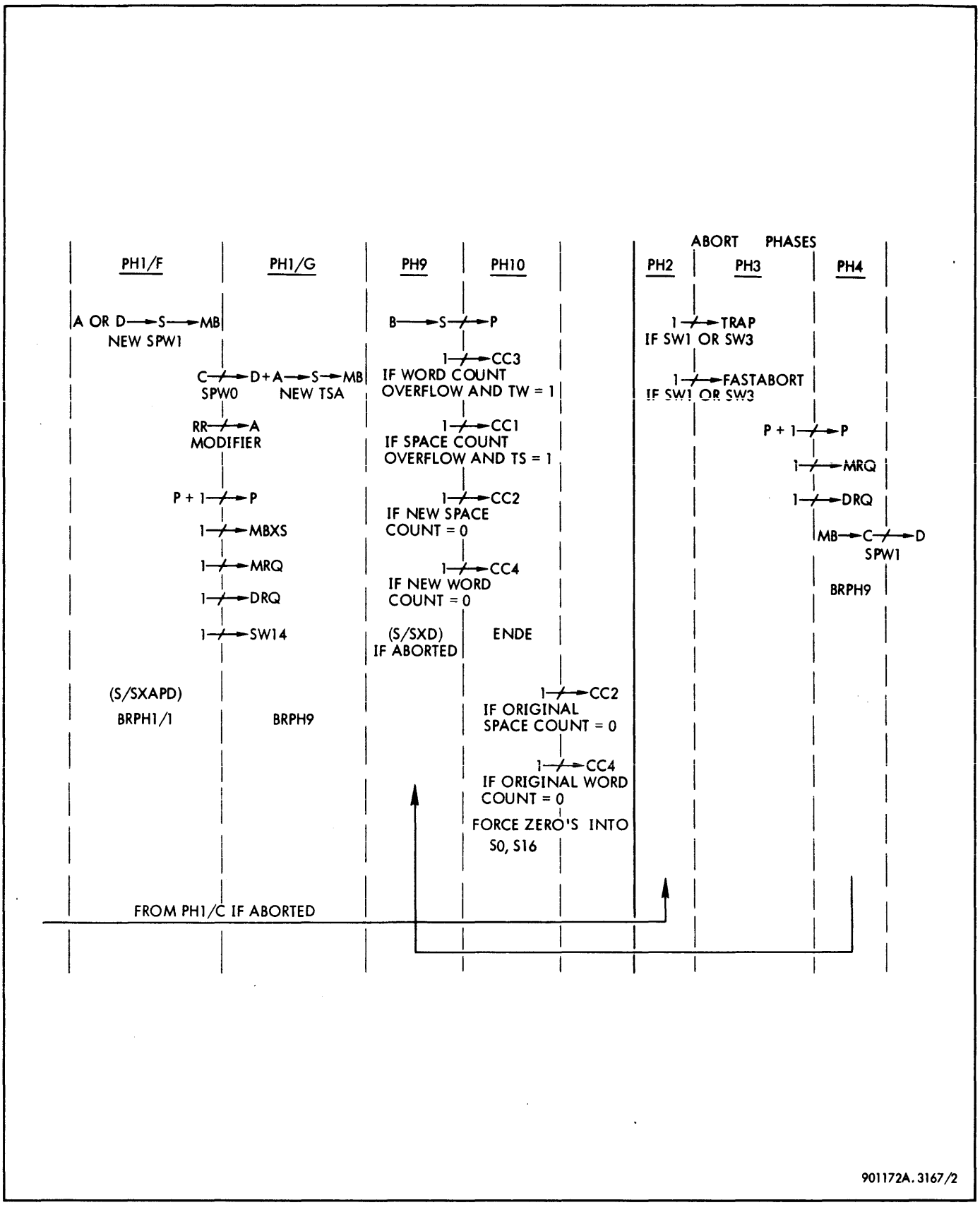


Figure 3-175. Modify Stack Pointer Instruction, Phase Sequence Diagram (Sheet 2 of 2)

Table 3-72. Modify Stack Pointer Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(C) : SPW1		
	(D) : SPW1		
	(B) : Program address		
	(P) : SPW0 address		
	(A) : RR (modifier)		
	Preset conditions with PRE3:		
	Enable signal (S/SXAPD)	(S/SXAPD) = FAST/C (PRE3 + ...) + ...	Preset adder for A plus D in PH1/A
	Set flip-flop SW8	S/SW8 = BRSW8 NRESET/A BRSW8 = FAST PRE3 + ...	
	Set flip-flop SW7	S/SW7 = (S/SW7) (S/SW7) = FAST PRE3 NO4	
	Reset flip-flop NCXS	S/NCXS = N(S/CXS) (S/CXS) = FAST PRE3 + ... R/NCXS = ...	Preset for S ⟶ C in PH1/A
	Reset flip-flop NT11L	S/NT11L = N(S/T11L) (S/T11L) = FAST PRE3 + ... R/NT11L = ...	Set clock T11L for PH1/A
PH1/A T11L	One clock long	PH1/A = PH1 SW8	
	D + A ⟶ S	Adder preset at last PREP clock	Add modifier to word count in SPW1
	(S0-S31) ⟶ (C0-C31)	CXS set at last PREP clock	Place new word count in C-register
	0 ⟶ (C0-C15)	CXS/0 = CXS N(FAST PH1/A) CXS/1 = CXS N(FAST PH1/A)	CXS/0 and CXS/1 are low
	Down align D-register	DXDR8 = FAST PH1/A + ...	Shift D-register 8 bits right as first half of 16-bit down alignment
	Set flip-flop SW3 if word count overflows	S/SW3 = (S/SW3) (S/SW3) = (A16 ⊕ K16) FAST PH1/A + ...	Word count overflows into adder bit 16
	Set flip-flop SW5 if TS is 1	S/SW5 = (S/SW5) (S/SW5) = FAST PH1/A D0 + ...	Trap-on-space inhibit bit is in D0
	Set flip-flop SW6 if TW is 1	S/SW6 = (S/SW6) (S/SW6) = FAST PH1/A D16 + ...	D16 contains trap-on-word inhibit bit TW
	Set flip-flop SW4 if word count = 0	S/SW4 = (S/SW4) (S/SW4) = N(A16 ⊕ K16) S1631Z FAST PH1/A + ...	New word count = 0 if S16-S31 contain zeros
	Set flip-flop SW9	S/SW9 = SW8 STEP815 STEP815 = NBRSW8 NBRSW10 NBRSW11 NBRSW12 NBRSW13 NBRSW15	
			Mnemonic: MSP (13, 93)

(Continued

Table 3-72. Modify Stack Pointer Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH1/A T11L (Cont.)	Reset flip-flop NT8L	S/NT8L = N(S/T8L) (S/T8L) = FAST PH1 R/NT8L = ...	Set clock T8L for PH1/B
	Sustain PH1	BRPH1/1 = FAST PH1 N(NSW7 PH1/C) + ...	Hold PH1 for PH1/B
PH1/B	One clock long	PH1/B = PH1 SW9	
T8L	Down align D-register	DXDR8 = FAST PH1/B + ...	Shift D-register right 8 bits as second half of 16-bit down alignment. Space count is now in D16-D31
	Set flip-flop SW10	S/SW10 = SW9 STEP815	
	Enable signal (S/SXDMA)	(S/SXDMA) = FAST/C PH1/B + ...	Preset adder for D minus A in PH1/C
	Reset flip-flop NT11L	S/NT11L = N(S/T11L) (S/T11L) = FAST PH1/B + ... R/NT11L = ...	Set clock T11L for PH1/C
	Sustain PH1	BRPH1/1 = FAST PH1 N(NSW7 PH1/C) + ...	Hold PH1 for PH1/C
PH1/C	One clock long	PH1/C = PH1 SW10	
T11L	D - A ⟶ S	Adder preset at PH1/B clock	Subtract modifier from space count
	(S0-S31) ⟶ (A0-A31)	AXS = FAST PH1/C SW7 + ...	Place new space count in A-register
	(C0-C31) ⟶/⟶ (D0-D31)	DXC = FAST PH1/C + ...	Transfer new word count to D-register
	Set SW1 if space count overflows	S/SW1 = (S/SW1) (S/SW1) = (A16 ⊕ K16) FAST PH1/C + ...	Space count overflows into adder bit 16
	Set SW2 if space count = 0	S/SW2 = (S/SW2) (S/SW2) = N(A16 ⊕ K16) S1631Z FAST PH1/C + ...	Space count = 0 if bits 16 through 31 of sum bus are 0
	Set flip-flop MRQ	S/MRQ = (S/MRQ/3) + ... (S/MRQ/3) = FAST PH1/C R/MRQ = ...	Request for core memory cycle
	Reset flip-flop NMRQP1	S/NMRQP1 = N(S/MRQ/3) R/NMRQP1 = ...	Delay data request one phase
			Mnemonic: MSP (13, 93)

(Continued)

Table 3-72. Modify Stack Pointer Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH1/C T11L (Cont.)	Reset flip-flop SW7	R/SW7 = (R/SW7) (R/SW7) = FAST PH1/C SW7 + ...	
	Reset flip-flop NT8L	S/NT8L = N(S/T8L) (S/T8L) = FAST PH1 R/NT8L = ...	Set clock T8L for PH1/D
	Sustain PH1 unless aborted or trapped	BRPH1/1 = FAST PH1 [N(NSW7 PH1/C) + (SW3 PH1/C) + (A16 ⊕ K16 PH1/C)]	Go to PH2 (which follows PH10 in this table) if aborted or trapped because of word count or space count overflow
PH1/D T8L	One clock long	PH1/D = PH1 SW11	
	Up align A-register	AXAL8 = FAST PH1/D + ...	Shift A-register 8 bits left as first half of 16-bit up alignment
	Set flip-flop DRQ	S/DRQ = MRQP1 + ...	Data request, inhibits transmission of another clock until data release received from core memory
	Set flip-flop SW12	S/SW12 = SW11 STEP815	
	Sustain PH1	BRPH1/1 = FAST PH1 N(NSW7 PH1/C) + ...	Hold PH1 for PH1/E
PH1/E DR	Sustained until data release	PH1/E = PH1 SW12	
	(MB0-MB31)⟶(C0-C31)	CXMB = DG	Read SPW0 from core memory
	Up align A-register	AXAL8 = FAST PH1/E	Shift A-register 8 bits left as second half of 16-bit up alignment. New space count is now in A0 through A15
	P + 1⟶P	PUC31 = FAST PH1/E	Increment P-register for SPW1 address
	Enable signal (S/SXAORD)	(S/SXAORD) = FAST PH1/E + ...	Preset for A OR D⟶S in PH1/G
	Set flip-flop A0 if TS is 1 (SW5)	S/A0 = FAST PH1/E SW5 AXAL8 + ...	Set trap-on-space inhibit bit if set in original SPW1
	Set flip-flop A16 if TW is 1 (SW6)	S/A16 = FAST PH1/E SW6 AXAL8 + ...	Set trap-on-word inhibit bit if set in original SPW1
			Mnemonic: MSP (13, 93)

(Continued)

Table 3-72. Modify Stack Pointer Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH1/E DR (Cont.)	Set flip-flop MBXS	S/MBXS = (S/MBXS) (S/MBXS) = FAST PH1/E + ... R/MBXS = ...	Preset for transfer of A OR D to core memory in PH1/F
	Set flip-flop MRQ	S/MRQ = (S/MRQ) + ... (S/MRQ) = (S/MBXS) + ... R/MRQ = ...	Request for core memory cycle
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = (S/MBXS) + ... R/DRQ = ...	Data request, inhibits transmission of another clock until data release received from core memory
	Set flip-flop SW13	S/SW13 = SW12 STEP815	
	Sustain PH1	BRPH1/1 = FAST PH1 N(NSW7 PH1/C) + ...	Hold PH1 for PH1/F
PH1/F DR	Sustained until data release	PH1/F = PH1 SW13	
	A OR D ⟶ S	(S/SXAORD) set at previous clock	New word count in D-register and new space count in A-register ⟶ S
	(S0-S31) ⟶ (MB0-MB31)	MBXS set at previous clock	Store new space count and word count in core memory at SPW1 location
	(C0-C31) ⟶/⟶ (D0-D31)	DXC = FAST PH1/F + ...	Transfer top of stack address (SPW0) to D-register
	(RR16-RR31) ⟶/⟶ (A16-A31)	AXRR/2 = AXRR/12 + ... AXRR/12 = FUMSP PH1/F NAXRR/6	Obtain modifier from private memory, place in A-register
	RR16 ⟶/⟶ A15	S/A15 = RR16 FUMSP PH1/F + ...	
	P - 1 ⟶/⟶ P	PDC31 = FAST PH1/F + ...	Decrement P-register to get SPW0 address
	Enable signal (S/SXAPD)	(S/SXAPD) = FAST/C (PH1/F + ...) + ...	Preset adder for A plus D in PH1/G
	Set flip-flop MBXS	S/MBXS = (S/MBXS) (S/MBXS) = FAST PH1/F + ... R/MBXS = ...	Preset for core memory write operation
	Set flip-flop MRQ	S/MRQ = (S/MRQ) + ... (S/MRQ) = (S/MBXS) + ... R/MRQ = ...	Request for core memory cycle
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = (S/MBXS) + ... R/DRQ = ...	Data request, inhibits transmission of another clock until data release received from core memory
			Mnemonic: MSP (13, 93)

(Continued

Table 3-72. Modify Stack Pointer Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH1/F	Set flip-flop SW14	S/SW14 = SW13 STEP815	
DR (Cont.)	Sustain PH1	BRPH1/1 = FAST PH1 N(NSW7 PH1/C) + ...	Hold PH1 for PH1/G
PH1/G	Sustained until data release	PH1/G = PH1 SW14	
DR	A + D⟶S	Adder preset by PH1/F clock	Add modifier in A-register to top of stack address in D-register and gate onto sum bus
	(S0-S31)⟶(MB0-MB31)	MBXS set by PH1/F clock	Store new top of stack address in core memory at SPW0 address
	Branch to PH9	BRPH9 = FAST PH1/G + ...	
PH9	One clock long		
T5L	(B0-B31)⟶(S0-S31)	SXB = PXSXB NDIS + ... PXSXB = NFAFL NFAMDS PH9	Program address⟶P-register via sum bus
	(S0-S31)⟶/(P0-P31)	PXS = PXSXB + ...	
	Set condition code:		
	Set CC3 if word count overflow and TW = 1 (SW6)	S/CC3 = (S/CC3/1) + ... (S/CC3/1) = FAST PH9 SW3 + ...	SW3 indicates word count overflow. If TW were 0, instruction would have trapped and not reached PH9
	Set CC1 if space count overflow and TS = 1 (SW5)	S/CC1 = (S/CC1/1) + ... (S/CC1/1) = FAST PH9 SW1 + ...	SW1 indicates space count overflow. If TS were 0, instruction would have trapped and not reached PH9
	Set CC2 if new space count = 0	S/CC2 = (S/CC2/1) + ... (S/CC2/1) = (FASTNABORT PH9) SW2 + ...	If instruction is successfully completed and stack is full, CC2 is set
	Set CC4 if new word count = 0	S/CC4 = (S/CC4/1) + ... (S/CC4/1) = FASTNABORT PH9 SW4	If instruction is successfully completed and word count = 0, CC4 is set
	Place zeros in condition code flip-flops not set	R/CC = FAST PH9 + ...	Places input on reset sides of CC1 through CC4 so that they will be reset if not set by this instruction
	Enable signal (S/SXD) if instruction aborted	(S/SXD) = FASTABORT PH9	Preset adder for D⟶S in PH10
			Mnemonic: MSP (13, 93)

(Continued)

Table 3-72. Modify Stack Pointer Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH10 DR	Sustained until data release Normal ENDE If instruction aborted:		
	Correct CC2	S/CC2 = (S/CC2/4) + ... (S/CC2/4) = S0007Z (FASTABORT ENDE)	Set CC2 if original space count (in D-register) = 0
	Correct CC4	S/CC4 = (S/CC4/2) + ... (S/CC4/2) = (FASTABORT ENDE) S1631Z	Set CC4 if original word count (in D-register) = 0
	Force zeros into S0, S16, and SGTZ	SGTZ = N(FASTABORT ENDE) S16 = N(FASTABORT ENDE) S0 = N(FASTABORT ENDE)	To prevent setting CC3 S16 is TW inhibit bit. S0 is TS inhibit bit. Neither should be tested for zero
PH2 T5L	If instruction is aborted (from PH1/C) One clock long Trap conditions:		
	Set flip-flop TRAP if word count overflows and TW = 0 or if space count overflows and TS = 0	S/TRAP = (S/TRAP) (S/TRAP) = FAST PH2 SW3 NSW6 + FAST PH2 SW1 NSW5	SW3 is word count overflow, SW1 is space count overflow, NSW6 ⟹TW = 0, NSW5⟹ TS = 0
	Abort if SW1 or SW3 is set	S/FASTABORT = FAST PH2 SW1 + FAST PH2 SW3 S/FASTF1 = SW3 + SW1	Instruction unconditionally aborted on space count or word count overflow. Note that FASTABORT is built with two flip-flops, FASTF1 and FASTF2
PH3 DR	Sustained until data release (memory access not applicable for this instruction)		
	Set flip-flop MRQ	S/MRQ = (S/MRQ/2) + ... (S/MRQ/2) = FASTABORT PH3 + ... R/MRQ = ...	Request for core memory cycle
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = (S/MRQ/2) + ... R/DRQ = ...	Data request, inhibits transmission of another clock until data release from core memory
			Mnemonic: MSP (13, 93)

(Continued)

Table 3-72. Modify Stack Pointer Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH3 DR (Cont.)	P + 1 → P	PUC31 = FAST PH3 + ...	Increment P-register to get SPW1 address
PH4	One clock long		
T5L	(MB0-MB31) → (C0-C31)	CXMB = DG	Load SPW1 from core memory into C-register
	(C0-C31) → (D0-D31)	DXC = FASTABORT PH4	Transfer SPW1 to D-register
	Branch to PH9	BRPH9 = FAST PH4 (SW1 + SW3)	Branch to PH9 to set condition code
			Mnemonic: MSP(13, 93)

LOAD MULTIPLE (LM; 2A, AA). The LM instruction loads a sequential set of words from core memory into a sequential set of private memory registers. The set of core memory words begins with the contents of the location specified in the reference address field of the LM instruction and follows in ascending order. The set of private memory registers begins with the register specified in the R field of the LM instruction and continues in ascending order. The number of words to be loaded is indicated by the condition code. If all 16 private memory registers are to be loaded, the initial value of the condition code is 0000. The private memory registers are treated as a circular set, with register 0 following register 15.

LOAD MULTIPLE PHASE SEQUENCE. Preparation phases for the LM instruction are the same as the general PREP phases for word instructions, paragraph 3-59. Figure 3-176 shows the simplified phase sequence for the LM instruction. Table 3-73 lists the detailed logic sequence during all LM execution phases. After the preparation phases, the instruction branches to PH6 to read the first word from core memory. In the second pass through PH6, the first word is loaded into private memory and the second word is read from core memory. The instruction continues looping through PH6 until a zero value in the macro-counter indicates that all of the words have been loaded. The instruction then enters PH9 to obtain the address of the next instruction and then proceeds to the ENDE operation in PH10.

If the condition code at the beginning of the instruction contains 0000, indicating that all 16 private memory registers are to be loaded, bit 3 of the macro-counter is set at the time the condition code is transferred to the A-register, thereby establishing 10000 as the number of words.

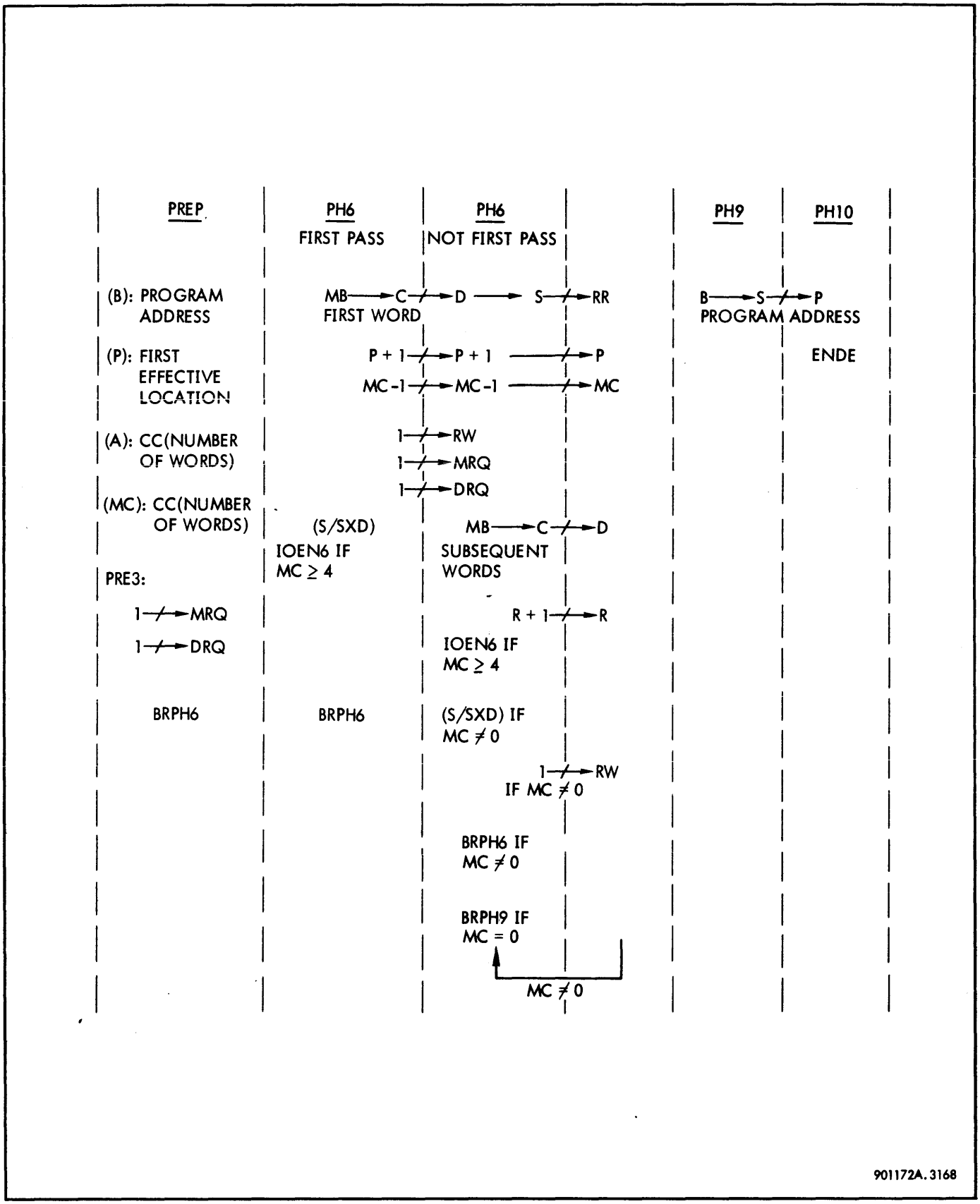


Figure 3-176. Load Multiple Instruction, Phase Sequence Diagram

Table 3-73. Load Multiple Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(B) : Program address		
	(P) : First effective location		
	(A) : CC (number of words)		
	(MC) : CC (number of words)		
	Preset conditions with PRE3:		
	Set flip-flop MRQ	S/MRQ = (S/MRQ/2) + ... (S/MRQ/2) = FAST/M PRE3 NOU0 OLA + ... R/MRQ = ...	Request for core memory cycle
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = (S/MRQ/2) + ... R/DRQ = ...	Data request, inhibits transmission of another clock until data release received from memory
	Branch to PH6	BRPH6 = FAST/M PRE3 NOU0 NANLZ + ...	
PH6	Sustained until data release		
DR 1st Pass	(MB0-MB31)⟶(C0-C31)	CXMB = DG	Read first word from core memory
	(C0-C31)⟶(D0-D31)	DXC = FAST/L PH6 + ...	Transfer first word to D-register for subsequent transfer to private memory
	P + 1⟶P	PUC31 = FAST/L PH6 NOU0 + ...	Increment P-register to obtain core memory location of second word
	MC - 1⟶MC	MCD7 = FAST/M NIOEN PH6 + ...	Decrement macro-counter for new number of words to be loaded
	Enable signal (S/SXD)	(S/SXD) = FAST/L PH6 NMCZ + ...	Preset adder logic for D⟶S in PH6 second pass
	Set flip-flop RW	S/RW = (S/RW) (S/RW) = FAST/L PH6 NMCZ + ... R/RW = ...	Prepare to write into private memory
	Set flip-flop MRQ	S/MRQ = (S/MRQ/2) + ... (S/MRQ/2) = FAST/L PH6 NMCZ + ... R/MRQ = ...	Request for core memory cycle
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = (S/MRQ/2) + ... R/DRQ = ...	Data request, inhibits transmission of another clock until data release received from memory
			Mnemonic: LM (2A, AA)

(Continued)

Table 3-73. Load Multiple Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH6 DR	Enable signal IOEN6 if MC $\geq$ 4	IOEN6 = IOEN6/1 PH6 + ... IOEN6/1 = NMC0005Z + ...	Enable I/O service call if number of words to be loaded $\geq$ 4
1st Pass (Cont.)	Sustain PH6	BRPH6 = FAST/M PH6 NMCZ + ...	Repeat PH6 to write into private memory and read another word
PH6	Sustained until data release		
DR	(D0-D31) $\longrightarrow$ (S0-S31)	Adder logic set at first PH6 clock	Load word into private memory
Not 1st Pass	(S0-S31) $\longrightarrow$ (RW0-RW31)	RWXS = RW	
	(MB0-MB31) $\longrightarrow$ (C0-C31)	CXMB = DG	Read another word from core memory
	(C0-C31) $\not\longrightarrow$ (D0-D31)	DXC = FAST/L PH6 + ...	Transfer word to D-register for subsequent transfer to private memory
	P + 1 $\not\longrightarrow$ P	PUC31 = FAST/L PH6 NOU0 + ...	Increment P-register to obtain next sequential core memory location
	R + 1 $\not\longrightarrow$ R	RUC31 = FAST/L PH6 NOU0 + ...	Increment R-register for next sequential private memory address
	MC - 1 $\longrightarrow$ MC	MCD7 = FAST/M NIOEN PH6 + ...	Decrement macro-counter for new number of words if no I/O interrupt
	Enable signal IOEN6 if MC $\geq$ 4	IOEN6 = IOEN6/1 PH6 + ... IOEN6/1 = NMC0005Z + ...	Enable I/O service call if number of words yet to be loaded $\geq$ 4
	Enable signal (S/SXD) if MC $\neq$ 0	(S/SXD) = FAST/L PH6 NMCZ + ...	Preset adder for D $\longrightarrow$ S if another word is to be loaded into private memory
	Set flip-flop RW if MC $\neq$ 0	S/RW = (S/RW) (S/RW) = FAST/L PH6 NMCZ R/RW = ...	Prepare to write into private memory if another word is to be loaded
	Sustain PH6 if MC $\neq$ 0	BRPH6 = FAST/M PH6 NMCZ + ...	Repeat PH6 if another word is to be loaded
	Branch to PH9 if MC = 0	BRPH9 = FAST/M PH6 NOU0 MCZ	Branch to PH9 if no more words are to be loaded
			Mnemonic: LM (2A, AA)

(Continued)

Table 3-73. Load Multiple Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH9	One clock long		
T5L	(B0-B31)⟶(S0-S31)	SXB = PXSXB NDIS	Address of next instruction in sequence ⟶ P-register via sum bus
	(S0-S31)⟶/⟶(P0-P31)	PXS = PXSXB	
		PXSXB = NFAFL NFAMDS PH9	
PH10	Normal ENDE		
			Mnemonic: LM (2A, AA)

STORE MULTIPLE (STM; 2B, AB). The STM instruction stores the contents of a sequential set of private memory registers into a sequential set of core memory locations. The set of private memory registers begins with the register specified in the R field of the STM instruction and follows in ascending order. The set of core memory locations begins with the location specified in the reference address field of the instruction and continues in ascending order. The number of words to be stored is indicated by the condition code immediately before the STM instruction. If all 16 private memory registers are involved in the operation, the initial value of the condition code is 0000. The private memory registers are treated as a circular set, with register 0 following register 15.

STORE MULTIPLE PHASE SEQUENCE. Preparation phases for the STM instruction are the same as the general PREP phases for word instructions, paragraph 3-59. Figure 3-177 shows the simplified phase sequence for the STM instruction. Table 3-74 lists the detailed logic sequence during all STM execution phases. After the preparation phases, the instruction branches to PH6 to get the first word from private memory. In the second pass through PH6, the first word is stored in core memory and the second word is obtained from private memory. The instruction continues looping through PH6 until a zero value in the macro-counter indicates that all the words have been stored. The instruction then enters PH9 to obtain the address of the next instruction and then proceeds to the ENDE operation in PH10.

If the condition code at the beginning of the instruction contains 0000, indicating that all 16 private memory registers are to be loaded, bit 3 of the macro-counter is set at the time the condition code is transferred to the A-register, thereby establishing 10000 as the number of words.

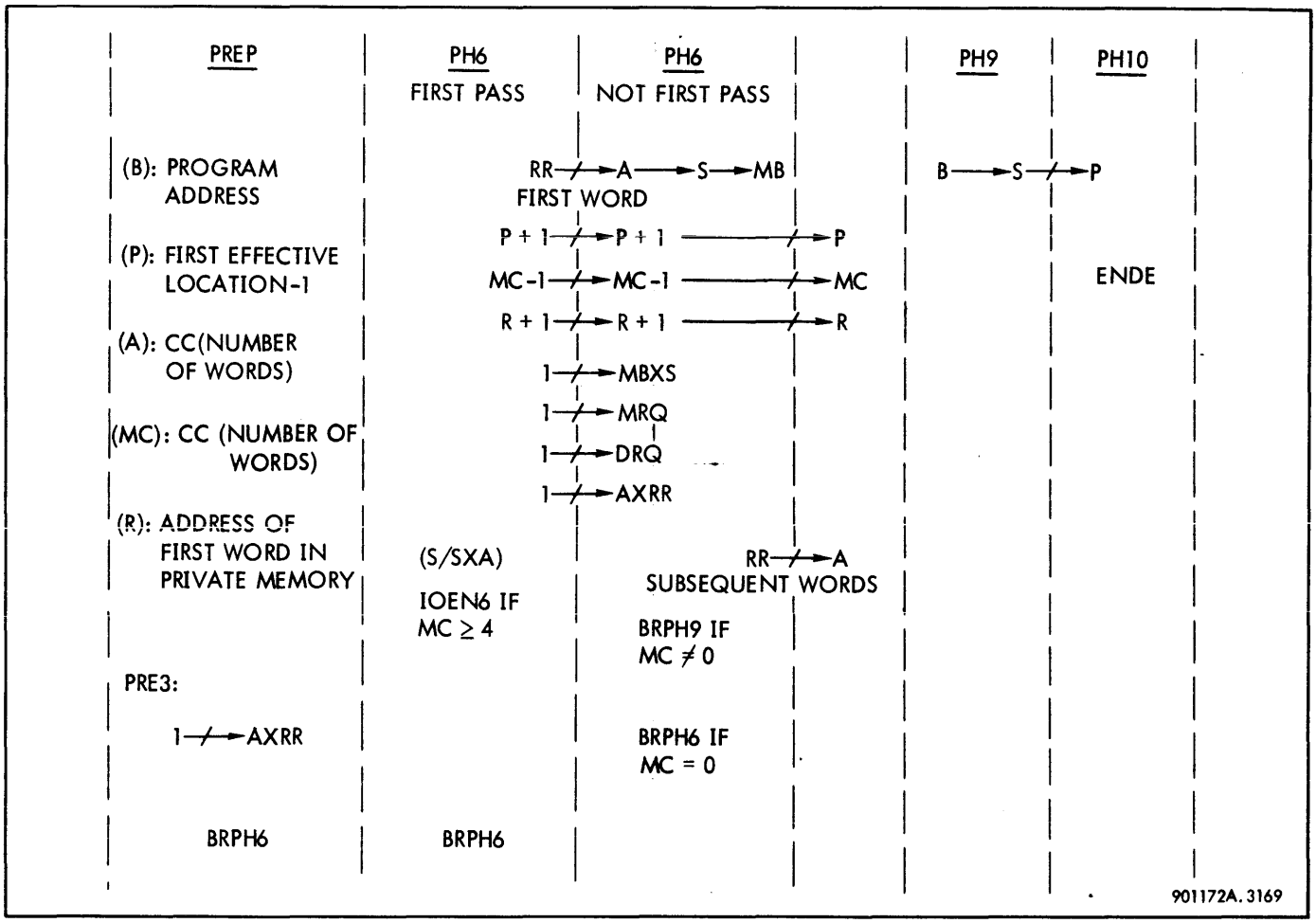


Figure 3-177. Store Multiple Instruction, Phase Sequence Diagram

Table 3-74. Store Multiple Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(B) : Program address		
	(P) : First effective location minus 1		
	(A) : CC (number of words)		
	(MC) : CC (number of words)		
	(R) : Location of first word		
	Preset conditions with PRE3:		
	Reset flip-flop NAXRR	S/NAXRR = N(S/AXRR) (S/AXRR) = FAST/M PRE3 NOU0 NOLA + ... R/NAXRR = ...	Preset for transfer of private memory contents → A-register in PH6
			Mnemonic: STM (2B, AB)

(Continued)

Table 3-74. Store Multiple Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PREP (Cont.)	Branch to PH6	BRPH6 = FAST/M PRE3 NOU0 NANLZ + ...	
PH6 T5L 1st Pass	One clock long		
	(RR0-RR31)—/→(A0-A31)	AXRR set at previous clock	Get first word from private memory
	P + 1—/→P	PUC31 = FAST/L PH6 NOU0 + ...	Increment P-register to obtain core memory location of first word
	R + 1—/→R	RUC31 = FAST/S PH6 + ...	Get address of second word in private memory
	MC - 1——→MC	MCD7 = FAST/M NIOEN PH6 + ...	Decrement macro-counter for new number of words to be loaded
	Enable signal (S/SXA)	(S/SXA) = FAST/S PH6 NMCZ + ...	Preset adder logic for A——→S in PH6 second pass
	Set flip-flop MBXS	S/MBXS = (S/MBXS) (S/MBXS) = FAST/S PH6 NMCZ R/MBXS = ...	Preset for transfer of A-register contents to core memory
	Set flip-flop MRQ	S/MRQ = (S/MRQ/2) + ... (S/MRQ/2) = (S/MBXS) R/MRQ = ...	Request for core memory cycle
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = (S/MBXS) R/DRQ = ...	Data request, inhibits transmission of another clock until data release received from memory
	Reset flip-flop NAXRR	S/NAXRR = N(S/AXRR) (S/AXRR) = FAST/S PH6 + ... R/NAXRR = ...	Preset for transfer of private memory contents —/→ A-register
	Enable signal IOEN6 if MC $\geq$ 4	IOEN6 = IOEN6/1 PH6 + ... IOEN6/1 = NMC0005Z + ...	Enable I/O service call if number of words to be stored $\geq$ 4
	Sustain PH6	BRPH6 = FAST/M PH6 NMCZ + ...	Repeat PH6 to transfer another word to core memory

(Continued)

Mnemonic: STM (2B, AB)

Table 3-74. Store Multiple Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH6 DR Not 1st Pass	Sustained until data release		
	(A0-A31)⟶(S0-S31)	Adder logic set at last PH6 clock	Write first word into memory
	(S0-S31)⟶(MB0-MB31)		
	(RR0-RR31)⇢(A0-A31)	AXRR set at previous clock	Read subsequent words from private memory
	P + 1⇢P	PUC31 = FAST/L PH6 NOU0 + ...	Increment P-register to obtain next sequential core memory location
	R + 1⇢R	RUC31 = FAST/L PH6 NOU0 + ...	Increment R-register for next sequential private memory address
	MC - 1⟶MC	MCD7 = FAST/M NIOEN PH6 + ...	Decrement macro-counter for new number of words if no I/O interrupt
	Reset flip-flop NAXRR	S/NAXRR = N(S/AXRR) (S/AXRR) = FAST/S PH6 + ... R/NAXRR = ...	Preset for transfer of subsequent words from private memory
	Enable signal IOEN6 if MC $\geq$ 4	IOEN6 = IOEN6/1 PH6 + ... IOEN6/1 = NMC0005Z + ...	Enable I/O service call if number of words yet to be stored $\geq$ 4
	Enable signal (S/SXA) if MC $\neq$ 0	(S/SXA) = FAST/S PH6 NMCZ + ...	Preset adder logic for A⟶S
	Sustain PH6 if MC $\neq$ 0	BRPH6 = FAST/M PH6 NMCZ + ...	Repeat PH6 if another word is to be stored
	Branch to PH9 if MC = 0	BRPH9 = FAST/M PH6 NOU0 MCZ	Branch to PH9 if no more words are to be stored
PH9 T5L	One clock long		
	(B0-B31)⟶(S0-S31)	SXB = PXSXB NDIS	Address of next instruction in sequence ⟶ P-register via sum bus
	(S0-S31)⇢(P0-P31)	PXS = PXSXB = PXSXB = NFAFL NFAMDS PH9	
PH10 DR	Sustained until data release Normal ENDE		
			Mnemonic: STM (2B, AB)

3-76 Family of Branch Instructions (FABRANCH)

BRANCH ON CONDITIONS SET (BCS; 69, E9). The BCS instruction forms the logical product (AND) of the R field of the instruction word and the current condition code. If the logical product is nonzero, the branch condition is satisfied, and the instruction pointed to by the effective address of the BCS instruction is executed. If the logical product is zero, the branch condition is not satisfied, and the next instruction in normal program sequence is executed. If the R field of the BCS instruction is 0000, the logical product is unconditionally zero. Therefore, the BCS can be used as a no-operation instruction by setting the R field to zero.

Branch on Conditions Set Phase Sequence. Preparation phases for the BCS instruction are the same as the general PREP phases for the word instructions, described in paragraph 3-59. Figure 3-178 shows the simplified phase sequence for the BCS instruction during execution, and table 3-75 lists the detailed logic sequence during all BCS execution phases.

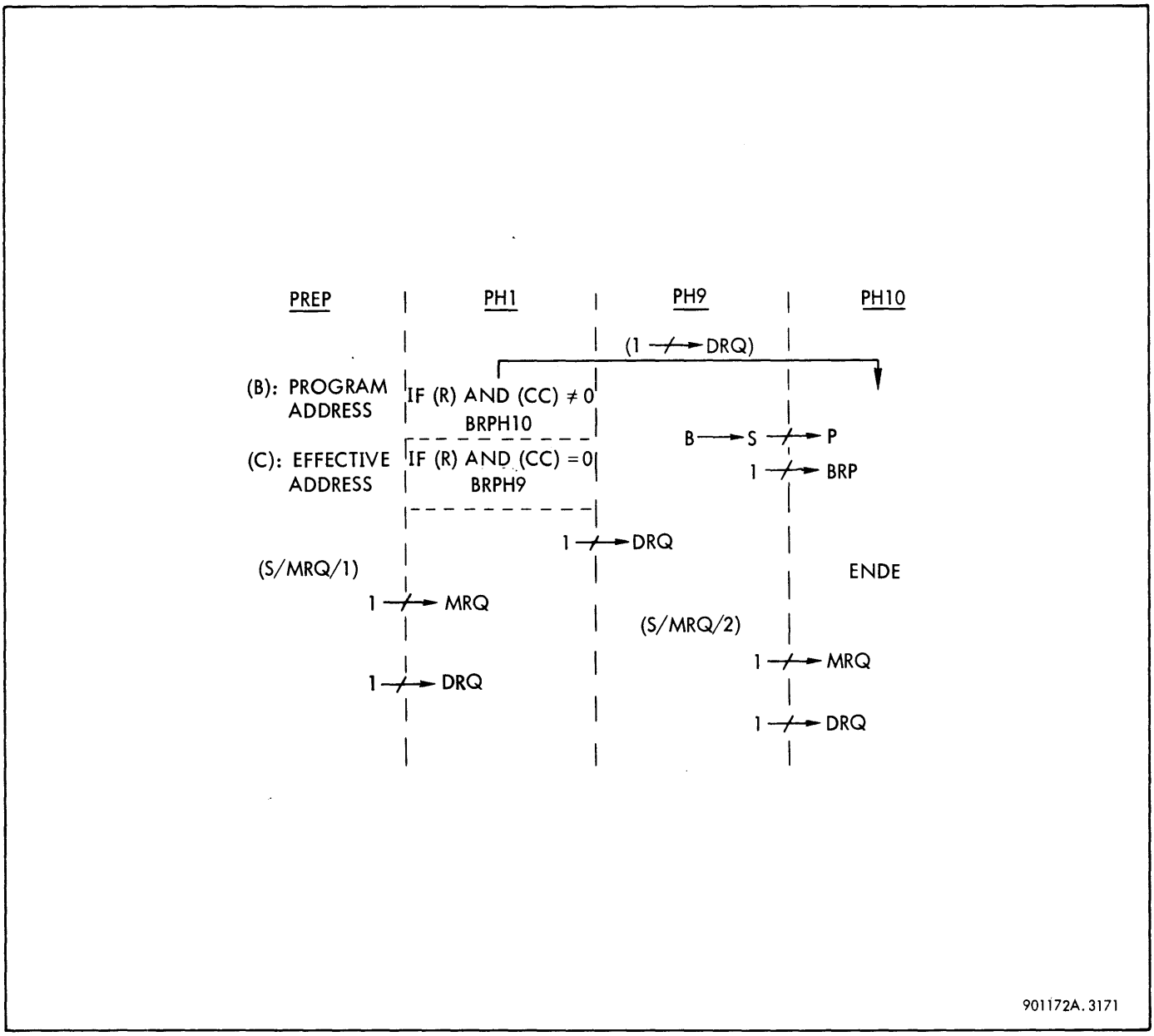


Figure 3-178. BCS Instruction, Phase Sequence Diagram

Table 3-75. Branch on Conditions Set Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(B) : Program address		Address of next instruction in sequence
	(P) : Effective address		Address of next instruction if branch conditions satisfied
	Set flip-flop MRQ	S/MRQ = (S/MRQ/1) + ...	Memory request set for all branch instructions. This memory request is for the instruction in the effective address in case the branch condition is satisfied. If the branch condition is satisfied, PH9 is skipped and memory request must have been made previously
		S/MRQ/1 = FABRANCH NANLZ PRE/12 + ...	
		FABRANCH = O1 O2 NO3 + ...	
		R/MRQ = ...	
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR	
		(S/DRQ) = (S/DRQ/2) + ...	
		(S/DRQ/2) = FABRANCH PRE3	Inhibits transmission of another clock until data release received from memory
PH1	One clock long		
DR	Compare contents of R field of BCS instruction with condition code and branch	BRPH9 = FUBCS PH1 N(R CC) + ...	Branch to PH9 if logical product of (R) and (CC) is zero. Branch condition not satisfied
		FUBCS = OU6 OL9	
		S/PH9 = BRPH9 NCLEAR + ...	
		R/PH9 = ...	
		BRPH10 = FUBCS PH1 (R CC) + ...	Branch to PH10 if logical product of (R) and (CC) is not zero. Branch condition satisfied
		S/PH10 = BRPH10 NCLEAR + ...	
		R/PH10 = ...	
		(R/CC) = CC1 R28 + CC2 R29 + CC3 R30 + CC4 R31	Logical product of R field and condition code
PH9	Sustained until DR		Requirement for DR is result of unconditional MRQ in PREP and DRQ in PH1
DR	(MB0-MB31) ⟶ (C0-C31)	CXMB = DG = /DG/	Instruction in effective address. Meaningless if this phase is entered since branch condition has not been satisfied
		SXB = PXSXB NDIS + ...	
			Mnemonic: BCS (69, E9)

(Continued)

Table 3-75. Branch on Conditions Set Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH9 DR (Cont.)	(B0-B31) ⟶ (S0-S31)	PXSXB = NFAFL NFAMDS PH9 + ...	Store program address in P-register
	(S15-S31) ⟶/ (P15-P31)	PXS = PXSXB + ...	
	Set flip-flop BRP	S/BRP = PXSXB + ... R/BRP = PRE1 NFAIM + ...	Signifies that program address is in the P-register
	Set flip-flop MRQ	S/MRQ = (S/MRQ/2) + ... (S/MRQ/2) = PSXSB + ... R/MRQ = ...	Memory request for next instruction in sequence
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = (S/MRQ/2) + ... R/DRQ = ...	Inhibits transmission of another clock until data release received from memory
PH10 DR	ENDE functions	See table 3-18	If entered from PH1, next instruction is from effective address in P-register at end of PREP. If entered from PH9, next instruction is from program address
			Mnemonic: BCS (69, E9)

BRANCH ON CONDITIONS RESET (BCR; 68, E8). The BCR instruction forms the logical produce (AND) of the R field of the instruction word and the current condition code. If the logical product is zero, the branch condition is satisfied, and the instruction pointed to by the effective address of the BCR instruction is executed. If the logical product is nonzero, the branch condition is not satisfied, and the next instruction in normal program sequence is executed. If the R field of the BCR instruction is 0000, the logical product is unconditionally zero. Therefore, the BCR instruction can be used as an unconditional branch instruction by setting the R field to zero.

Branch on Conditions Reset Phase Sequence. Preparation phases for the BCR instruction are the same as the general PREP phases for word instructions, described in paragraph 3-59. Figure 3-179 shows the simplified phase sequence for the BCR instruction during execution, and table 3-76 lists the detailed logic sequence during all BCR execution phases.

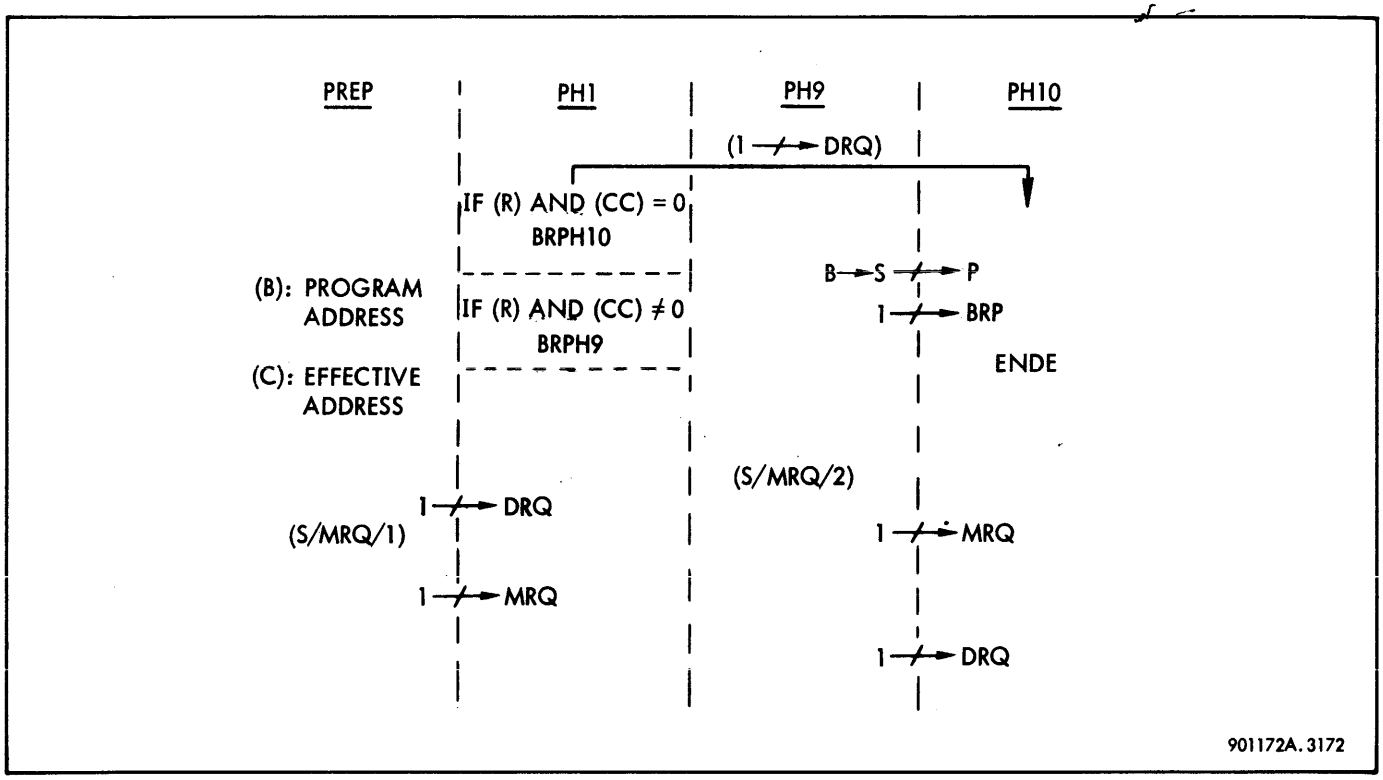


Figure 3-179. BCR Instruction, Phase Sequence Diagram

Table 3-76. Branch on Conditions Reset Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(B) : Program address		Address of next instruction in sequence
	(P) : Effective address		Address of next instruction if branch conditions satisfied
	Set flip-flop MRQ	S/MRQ = (S/MRQ/1) + ...	Memory request set for all branch instructions. This memory request is for the instruction in the effective address in case the branch condition is satisfied and PH10 is entered from PH1
		(S/MRQ/1) = FABRANCH NANLZ PRE/12 + ...	
		FABRANCH = O1 O2 NO3 + ...	
		R/MRQ = ...	
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR	
		(S/DRQ) = (S/DRQ2) + ...	
		(S/DRQ/2) = FABRANCH PRE3	
			Mnemonic: BCR (68, 'E8)

(Continued)

Table 3-76. Branch on Conditions Reset Sequence (Cont.)

Phase	Function Performed	Signals Involved			Comments
PH1	One clock long				
DR	Compare contents of R field of BCR instruction with condition code, and branch	BRPH9	=	FUBCR PH1 (R CC) + ...	Branch to PH9 if logical product of (R) and (CC) is not zero. Branch condition not satisfied
		FUBCR	=	OU6 OL8	
		S/PH9	=	BRPH9 NCLEAR + ...	
		R/PH9	=	...	
		BRPH10	=	FUBCR PH1 N(R CC) + ...	Branch to PH10 if logical product of (R) and (CC) is zero. Branch condition satisfied
		S/PH10	=	BRPH10 NCLEAR + ...	
		R/PH10	=	...	
		(R CC)	=	CC1 R28 + CC2 R29 + CC3 R30 + CC4 R31	Logical product of R field and condition code
PH9	Sustained until DR				Requirement for DR result of unconditional MRQ in PREP and DRQ in PH1
DR	(MB0-MB31)⟶(C0-C31)	CXMB	=	DG = /DG/	Instruction in effective address. Meaningless if this phase is entered since branch condition has not been satisfied
		SXB	=	PXSXB NDIS + ...	
	(B0-B31)⟶(S0-S31)	PXSXB	=	NFAFL NFAMDS PH9 + ...	Stores program address in P-register
	(S15-S31)⟶/(P15-P31)	PXS	=	PXSXB + ...	
	Set flip-flop BRP	S/BRP	=	PXSXB + ...	Signifies that program address is in P-register
		R/BRP	=	PRE1 NFAIM + ...	
	Set flip-flop MRQ	S/MRQ	=	(S/MRQ/2) + ...	Memory request for next instruction in sequence
		(S/MRQ/2)	=	PXSXB + ...	
		R/MRQ	=	...	
	Set flip-flop DRQ	S/DRQ	=	(S/DRQ) NCLEAR	Inhibits transmission of another clock until data release received from memory
		(S/DRQ)	=	(S/MRQ/2) + ...	
		R/DRQ	=	...	
					Mnemonic: BCR (68, E8)

(Continued)

Table 3-76. Branch on Conditions Reset Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH10 DR	ENDE functions	See table 3-18	If entered from PH1, next instruction is from effective address in P-register at end of PREP. If entered from PH9, next instruction is from program address
			Mnemonic: BCR (68, E8)

BRANCH AND LINK (BAL; 6A, EA). The BAL instruction determines the effective address, loads the address of the next instruction in normal sequence into bit positions 15 through 31 of private memory register R, and clears bit positions 0 through 14 to zero. The effective address then replaces the address of the next instruction in normal sequence, and the instruction pointed to by the effective address of the BAL instruction is executed.

If the effective address of the BAL instruction is a non-existent memory address, the computer aborts execution of the BAL instruction and traps to location X'40'. In this case, the instruction address stored by the XPSD instruction in location X'40' is the address of the BAL instruction.

Branch and Link Phase Sequence. Preparation phases for the BAL instruction are the same as the general PREP phases for word instruction, described in paragraph 3-59. Figure 3-180 shows the simplified phase sequence for the instruction during execution, and table 3-77 lists the detailed logic sequence during all BAL execution phases.

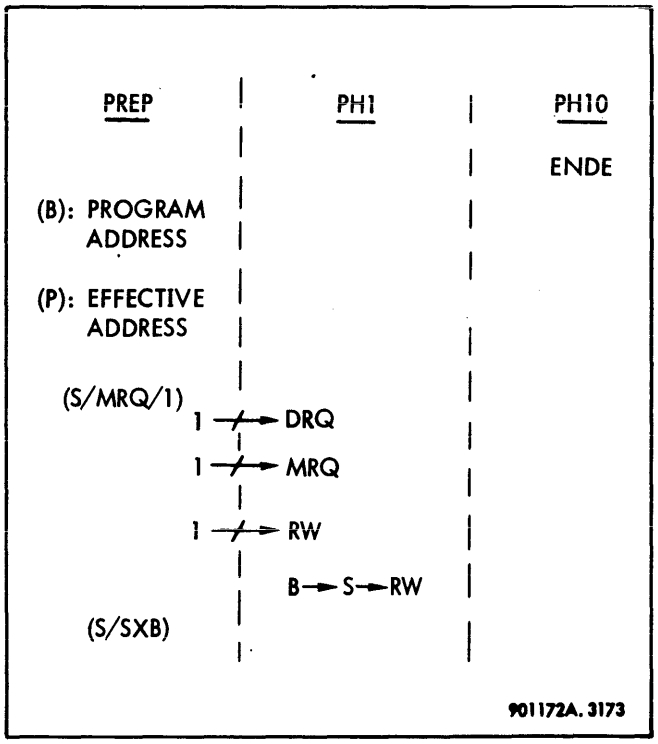


Figure 3-180. BAL Instruction, Phase Sequence Diagram

Table 3-77. Branch and Link Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(B) : Program address		Address of next instruction in sequence
	(P) : Effective address		Address of next instruction to be used
	Enable signal (S/SXB)	(S/SXB) = FUBAL PRE3 + ...	Preset logic for B⟶S in PH1
		FUBAL = OU6 OLA	
	Set flip-flop RW	S/RW = (S/RW/1) = (S/RW) + ...	Prepare to store next instruction in sequence in private memory
		(S/RW) = FUBAL NANLZ PRE3 + ...	
		R/RW = ...	
	Set flip-flop MRQ	S/MRQ = (S/MRQ/1) + ...	Memory request for instruction at effective address
		(S/MRQ/1) = FABRANCH NANLZ PRE/12 + ...	
		FABRANCH = O1 O2 NO3 + ...	
		R/MRQ = ...	
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR	Inhibits transmission of another clock until data release received from memory
		(S/DRQ) = (S/DRQ/2) + ...	
		(S/DRQ/2) = FABRANCH PRE3	
PH1	One clock long		
DR	(B0-B31)⟶(S0-S31)	Adder logic set at last PREP clock	Store program address in private memory register R
	(S0-S31)⟶/(RW0-RW31)	RWXS/0-RWXS/3 = RW + ...	
		RW = Set at last PREP clock	
	Branch to PH10	BRPH10 = FUBAL PH1 + ...	
		S/PH10 = BRPH10 NCLEAR + ...	
		R/PH10 = ...	
PH10			
DR	ENDE functions	See table 3-18	Execute instruction in effective address
			Mnemonic: BAL (6A, EA)

BRANCH ON DECREMENTING REGISTER (BDR; 64, E4). The BDR instruction decrements the contents of private memory register R by one. If the result is a positive value, the branch condition is satisfied, and the instruction pointed to by the effective address of the BDR instruction is executed. If the result is zero or a negative value, the branch condition is not satisfied, and the next instruction in normal program sequence is executed.

If the effective address of the BDR instruction is a non-existent memory address and the result of decrementing private memory register R is a positive value, the computer aborts execution of the BDR instruction and traps to location X'40'. In this case, private memory register R contains the value that existed just before execution of the BDR instruction, and the instruction address stored by the XPSD instruction in location X'40' is the address of the aborted BDR instruction.

Branch on Decrementing Register Phase Sequence. Preparation phases for the BDR instruction are the same as the general PREP phases for word instructions, described in paragraph 3-59. Figure 3-181 shows the simplified phase sequence for the BDR instruction during execution, and table 3-78 lists the detailed logic sequence during all BDR execution phases.

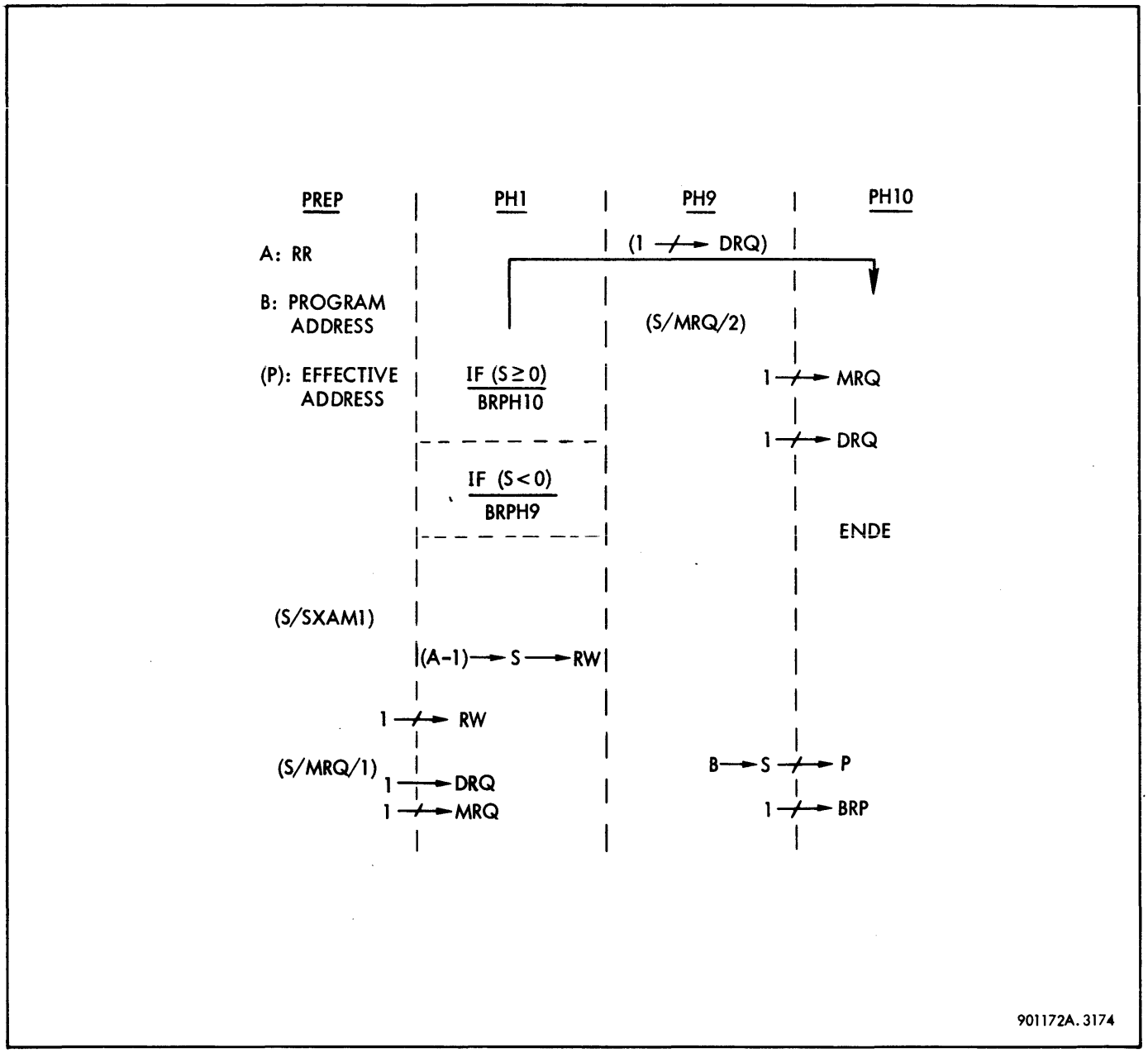


Figure 3-181. BDR Instruction, Phase Sequence Diagram

Table 3-78. Branch on Decrementing Register Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(A) : RR		Contents of private memory register R
	(B) : Program address		Address of next instruction in sequence
	(P) : Effective address		Address of next instruction if branch conditions satisfied
	Enable signal (S/SXAM1)	(S/SXAM1) = FUBDR PRE3 + ... FUBDR = OU6 OL4	Preset adder for (A-1) ⟶S in PH1
	Set flip-flop RW	S/RW = (S/RW/1) = (S/RW) + ... (S/RW) = FUBDR NANLZ PRE3 + ... R/RW = ...	Prepare to store (A-1) in private memory register R
	Set flip-flop MRQ	S/MRQ = (S/MRQ/1) + ... (S/MRQ/1) = FABRANCH NANLZ PRE/12 + ... R/MRQ = ...	Memory request set for all branch instructions. This memory request is for the instruction in the effective address in case the branch condition is satisfied and PH10 is entered from PH1
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = (S/DRQ/2) + ... (S/DRQ/2) = FABRANCH PRE3	Inhibits transmission of another clock until data release received from memory
PH1	One clock long		
DR	(A0-A31) – 1⟶(S0-S31)	Adder logic set at last PREP clock	Store (A-1) in private memory register R
	(S0-S31) ⟶/⟶(RW0-RW31)	RWXS/0-RWXS/3 = RW + ... RW = Set at last PREP clock	
	Set PH10 if (S0-S31) is greater than zero	S/PH10 = FUBDR PH1 SGTZ + ... SGTZ = (S0 + S1 + ... + S31) N(S0 NFACOMP) + ... R/PH10 = ...	S is greater than zero if (S0-S31) contains at least one 1, and S0 = 0. Branch condition satisfied
	Set PH9 if (S0-S31) is zero or less than zero	S/PH9 = FUBDR PH1 NSGTZ + ... R/PH9 = ...	Branch condition not satisfied
			Mnemonic: BDR (64, E4)

(Continued)

Table 3-78. Branch on Decrementing Register Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH9 DR	Sustained until DR		Requirement for DR result of unconditional MRQ in PREP and DRQ in PH1.
	(MB0-MB31)⟶(C0-C31)	CXMB = DG = /DG/	Instruction in effective address meaningless in this phase
	(B0-B31)⟶(S0-S31)	SXB = PXSXB NDIS + ...	Stores program address in P-register
		PXSXB = NFAFL NFAMDS PH9 + ...	
	(S15-S31)⟶(P15-P31)	PXS = PXSXB + ...	
	Set flip-flop BRP	S/BRP = PXSXB + ...	Signifies that program address is in P-register
		R/BRP = PRE1 NFAIM + ...	
	Set flip-flop MRQ	S/MRQ = (S/MRQ/2) + ...	Memory request for next instruction in sequence
		(S/MRQ/2) = PXSXB + ...	
		R/MRQ = ...	
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR	Inhibits transmission of another clock until data release received from memory
		(S/DRQ) = (S/MRQ/2) + ...	
		R/DRQ = ...	
PH10 DR	ENDE functions	See table 3-18	If entered from PH1, next instruction is from effective address in P-register at PREP. If entered from PH9, next instruction is from program address
			Mnemonic: BDR (64, E4)

BRANCH ON INCREMENTING REGISTER (BIR; 65, E5). The BIR instruction increments the contents of private memory register R by one. If the result is a negative value, the branch condition is satisfied, and the instruction pointed to by the effective address of the BIR instruction is executed. If the result is zero or a positive value, the branch condition is not satisfied, and the next instruction in normal program sequence is executed.

If the effective address of the BIR instruction is a nonexistent memory address, and the result of incrementing the contents of private memory register R is negative, the computer aborts execution of the BIR instruction and traps to location X'40'. In this case, private memory register R still contains the value that existed just before execution of the BIR instruction, and the instruction address stored by the XPSD instruction in location X'40' is the address of the aborted BIR instruction.

Branch on Incrementing Register Phase Sequence. Preparation phases for the BIR instruction are the same as the general PREP phases for word instructions, described in paragraph 3-59. Figure 3-182 shows the simplified phase sequence for the instruction during execution, and table 3-79 lists the detailed logic sequence for all BIR execution phases.

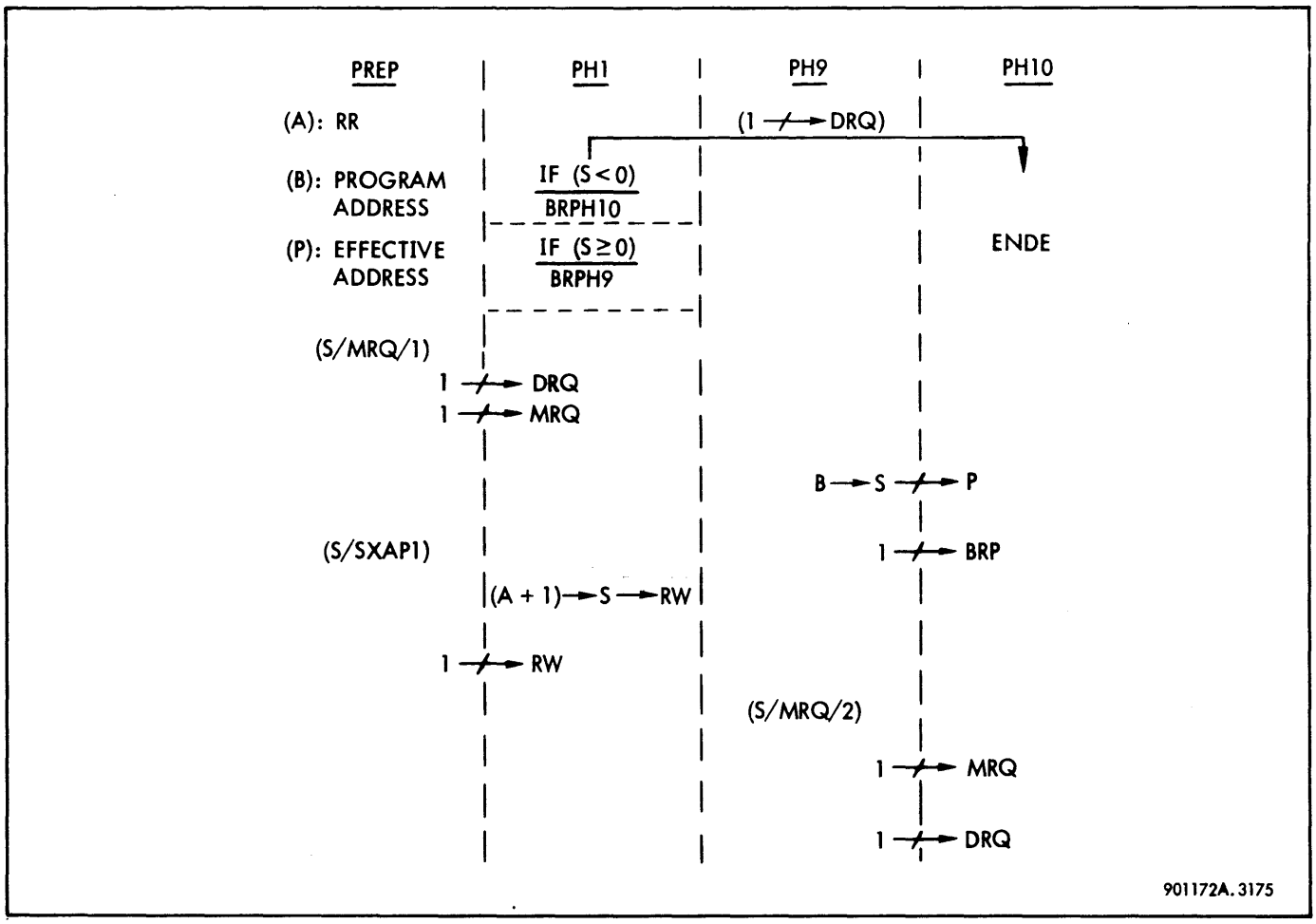


Figure 3-182. BIR Instruction, Phase Sequence Diagram

Table 3-79. Branch on Incrementing Register Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(A) : RR		Contents of private memory register R
	(B) : Program address		Address of next instruction in sequence
	(P) : Effective address		Address of next instruction if branch conditions satisfied
	Set flip-flop MRQ	S/MRQ = (S/MRQ/1) + ...	Memory request set for all branch instructions. This memory request is for the instruction in the effective address in case the branch condition is satisfied and PH10 is entered from PH1
		(S/MRQ/1) = FABRANCH NANLZ PRE/12 + ...	
		R/MRQ = ...	
			Mnemonic: BIR (65, E5)

(Continued)

Table 3-79. Branch on Incrementing Register Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PREP (Cont.)	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = (S/DRQ/2) + ... (S/DRQ/2) = FABRANCH PRE3	Inhibits transmission of another clock until data release received from memory
	Enable signal (S/SXAP1)	(S/SXAP1) = FUBIR PRE3 + ... FUBIR = OU6 OL5	Preset adder for (A + 1) ⟶ S in PH1
	Set flip-flop RW	S/RW = (S/RW/1) = (S/RW) + ... (S/RW) = FUBIR NANLZ PRE3 + ... R/RW = ...	Prepare to store (A + 1) in private memory register R
PH1	One clock long		
DR	(A0-A31) + 1 ⟶ (S0-S31)	Adder logic set at last PREP clock	Store (A + 1) in private memory register R
	(S0-S31) ⟶/⟶ (RW0-RW31)	RWXS/0-RWXS/3 = RW + ... RW = Set at last PREP clock	
	Branch to PH9 if (A + 1) positive or zero	BRPH9 = FUBIR PH1 NS0 + ... S/PH9 = BRPH9 NCLEAR + ... R/PH9 = ...	Sign bit (S0) is 0 for (A + 1) positive or zero. Branch condition not satisfied
	Set PH10 if (A + 1) negative	S/PH10 = FUBIR PH1 NBRPH9 + ...	If not BRPH9, sign bit is 1, indicating negative number. Branch condition satisfied
PH9	Sustained until DR		Requirements for DR result of unconditional MRQ in PRE and DRQ in PH1
DR			
	(MB0-MB31) ⟶ (C0-C31)	CXMB = DG = /DG/	Instruction in effective address. Meaningless in this phase
	(B0-B31) ⟶ (S0-S31) (S15-S31) ⟶/⟶ (P15-P31)	PXSXB = NFAFL NFAMDS PH9 + ...	Stores program address in P-register
	Set flip-flop BRP	S/BRP = PXSXB + ... R/BRP = PRE1 NFAIM + ...	Signifies that program address is in the P-register
	Set flip-flop MRQ	S/MRQ = (S/MRQ/2) + ... (S/MRQ/2) = PSXSB + ... R/MRQ = ...	Memory request for next instruction in sequence
			Mnemonic: BIR (65, E5)

(Continued)

Table 3-79. Branch on Incrementing Register Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH9 DR (Cont.)	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = (S/MRQ/2) + ... R/DRQ = ...	Inhibits transmission of another clock until data release received from memory
PH10	ENDE functions	See table 3-18	If entered from PH1, next instruction is effective address in P-register. If entered from PH9, next instruction is from program address
			Mnemonic: BIR (65, E5)

EXECUTE (EXU; 67, E7). The EXU instruction causes the computer to execute the instruction in the location pointed to by the effective address of the EXU instruction (subject instruction). The subject instruction is performed exactly as if it, instead of the EXU instruction, were initially accessed. If the subject instruction is another EXU instruction, the computer executes the new subject instruction. A sequence of EXU instructions will be processed until an instruction other than an EXU is accessed. After the final effective instruction is executed, the computer returns to the next instruction in sequence after the initial EXU instruction, unless the effective instruction is a branch instruction, which results in transfer to a different location.

If an interrupt activation occurs between the beginning of an EXU instruction and the last interruptible point of the effective instruction, the computer processes the interrupt-servicing routine for the active interrupt level and then returns program control to the EXU instruction. A program is interruptible after every instruction access, including accesses made with the EXU instruction. The effective instruction is interrupted in the normal manner for its type of instruction.

If a trap condition occurs between the beginning of an EXU instruction and the completion of the effective instruction, the computer traps to the appropriate location. The instruction address stored by the XPSD in the trap location is the address of the EXU instruction.

Execute Phase Sequence. Preparation phases for the EXU instruction are the same as the general PREP phases for word instruction, described in paragraph 3-59.

Figure 3-183 shows the simplified phase sequence for the instruction during execution, and table 3-80 lists the detailed logic sequence during all EXU execution phases.

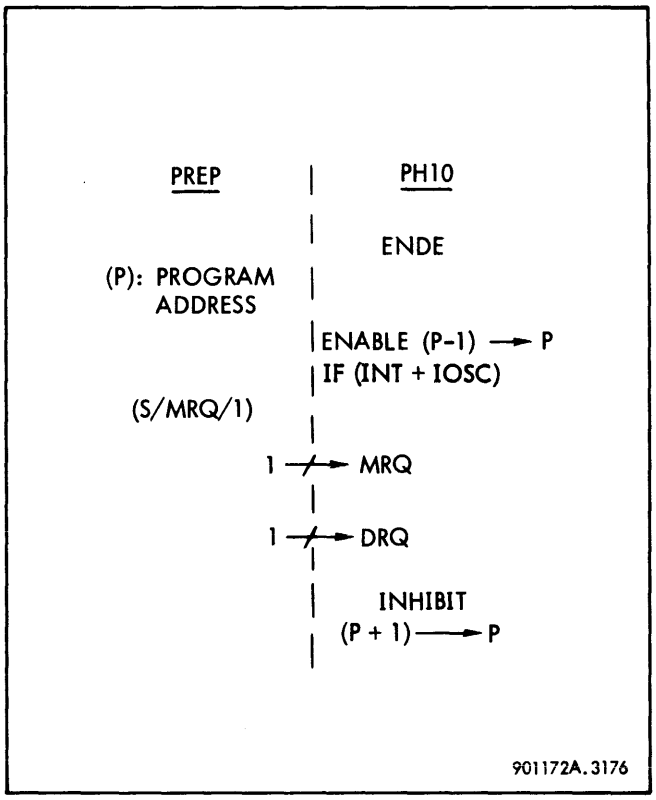


Figure 3-183. EXU Instruction, Phase Sequence Diagram

Table 3-80. Execute Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(P) : Program address		Address of next instruction in sequence
	Set flip-flop MRQ	S/MRQ = (S/MRQ/1) + ...	Memory request for subject instruction addressed by EXU instruction
		(S/MRQ/1) = FABRANCH NANLZ PRE/12 + ...	
		R/MRQ = ...	
	Branch to PH10	BRPH10 = FUEXU NANLZ PRE3 + ...	
		FUEXU = OU6 OL7	
		S/PH10 = BRPH10 NCLEAR + ...	
		R/PH10 = ...	
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR	Inhibits transmission of another clock until data release received from memory
		(S/DRQ) = BRPH10 + ...	
		R/DRQ = ...	
PH10	ENDE functions, with following exceptions:	See table 3-18	
	Inhibit (P + 1)—/→P	PUC31 = N(FUEXU ENDE) PH10 NHALT NIOSC NINT NKAHOLD	PUC31 false because (FUEXU ENDE) true, unless (INT + IOSC)
	Enable downcount	PDC31 = FUEXU ENDE (INT + IOSC)	(P - 1)—/→P if I/O service call pending or interrupt pending. The EXU will be executed again after the I/O service call or interrupt routine is processed
			Mnemonic: EXU (67, E7)

3-77 Family of Call Instructions (FACAL)

CALL 1 THROUGH CALL 4 (CAL1 THROUGH CAL4; 04 THROUGH 07, 84 THROUGH 87). CAL1 through CAL4 cause a trap to memory locations X'48' through X'4B', respectively, for the next instruction in sequence. The instruction in the trap location must be an exchange program status doubleword (XPSD) instruction. The R field of the CAL instruction word is ORed with CC1 through CC4 of the new program status doubleword. The R field value may also be used to modify the instruction address portion of the new program status doubleword. Both of these actions are discussed in the description of the XPSD instruction, paragraph 3-78. Execution of a CALL instruction involves storing the R field and enabling the INTRAP phases; further actions are then the same as the normal TRAP sequence discussed in paragraph 3-30. Table 3-81 lists the detailed logic sequence during all CAL execution phases. Preparation phases for CAL are the same as the general PREP phases for word instructions, described in paragraph 3-59.

Table 3-81. CAL1 Through CAL4 Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP: (B) : Program address		Not used
PH1 T5L	(R28-R31)→(TRACC1-TRACC4)	S/TRACC1 = FACAL PH1 NTRAP NSTRAP R28 ⋮ ⋮ S/TRACC4 = FACAL PH1 NTRAP NSTRAP R31 FACAL = OU0 (NO4 O5) R/TRACC1-R/TRACC4 = (S/TRAP) + ... (S/TRAP) = FACAL PH1 + ...	TRACC1 through TRACC4 are used to set the condition code flip-flops during execution of the XPSD instruction
	Set flip-flop TR28	S/TR28 = FACAL PH1NTRAP NSTRAP + ... R/TR28 = (S/TRAP) + ...	During INTRAP phases, TR28 through TR31→P28 through P31 to give least significant hexadecimal digit of trap location
	Set flip-flop TR30 if CAL3 or CAL4	S/TR30 = FACAL PH1 NTRAP NSTRAP O6 + ... R/TR30 = (S/TRAP) + ...	
	Set flip-flop TR31 if CAL2 or CAL4	S/TR31 = FACAL PH1 NTRAP NSTRAP O7 R/TR31 = (S/TRAP) + ...	

INST	TR 28	TR 29	TR 30	TR 31	DIGIT
CAL1	1	0	0	0	X'8'
CAL2	1	0	0	1	X'9'
CAL3	1	0	1	0	X'A'
CAL4	1	0	1	1	X'B'

Most significant hexadecimal digit is always 4

TR28 through TR31 may also modify instruction address during XPSD instruction

Mnemonic: CAL1-CAL4 (04-07, 84-87)

(Continued)

Table 3-81. CAL1 Through CAL4 Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH1 T5L (Cont.)	Set flip-flop TRAP	S/TRAP = (S/TRAP) NRESET (S/TRAP) = FACAL PH1 + ... R/TRAP = (R/TRAP) = FAPSD PH5 + ...	
	Set flip-flop INTRAP	S/INTRAP = (S/INTRAP) NRESET (S/INTRAP) = N(PCP2 NKRUN + INTRAP + DCSTOP) NPCPACT (S/TRAP) R/INTRAP = (R/TRAP) + ...	Preliminary actions before going into INTRAP phases. INTRAP phases are now entered
	Set flip-flop INTRAP1	S/INTRAP1 = (S/INTRAP) NRESET R/INTRAP1 = INTRAP2 + ...	
	Set flip-flop INTRAP2	S/INTRAP2 = (S/INTRAP) + ... R/INTRAP2 = ...	
	Inhibit setting PRE1	PREIEN = N(S/TRAP) N(S/INTRAP) NIOSC NHALT	
	Clear	CLEAR = (S/INTRAP) + ...	
			Mnemonic: CAL1-CAL4 (04-07, 84-87)

3-78 Family of Program Status Doubleword Instructions (FAPSD)

LOAD PROGRAM STATUS DOUBLEWORD (LPSD; 0E, 8E). The LPSD instruction replaces bits 0 through 39 of the current program status doubleword with bits 0 through 39 of the effective doubleword of the instruction address. Bits 56 through 59 of the program status doubleword are conditionally replaced.

General. A program status doubleword is stored in memory as a 64-bit word in two consecutive memory locations. The current program status doubleword (PSD) is stored in flip-flops and registers of the Sigma 5. The correspondence between the two storage locations is indicated in table 3-82.

Table 3-82. Program Status Doubleword Storage

Doubleword Bits	Flip-Flops	Content and Mnemonic	
0-3	CC1-CC4	Condition code (CC)	PSW1
4		Zero	
5	FS	Floating significant mask (FS)	
6	FZ	Floating zero mask (FZ)	
7	NFN	Floating normalize mask (FN)	
8	NMASTER	Master/slave mode control (MS)	
9		Zero	
10	DM	Decimal fault trap mask (DM)	
11	AM	Fixed-point arithmetic overflow trap mask (AM)	
12-14		Zeros	
15-31	P15-P31	Instruction address (IA)	

(Continued)

Table 3-82. Program Status Doubleword Storage (Cont.)

Doubleword Bits	Flip-Flops	Content and Mnemonic	
32-33		Zeros	PSW2
34-35	WK0, WK1	Write key (WK)	
36		Zero	
37	CIF	Counter interrupt group inhibit (CI)	
38	II	I/O interrupt group inhibit (II)	
39	EI	External interrupt group inhibit (EI)	
40-55		Zeros	
56-59	RP24-RP27	Register pointer (RP)	
60-63		Zeros	

Conditional Operations. If bit position 8 of the LPSD instruction contains a one, bits 56 through 59 of the current program status doubleword (register pointer bits) are replaced by bits 56 through 59 of the effective doubleword (bits 24 through 27 of PSW2). If bit position 8 of the LPSD instruction contains a zero, the register pointer bits of the current PSD are not changed.

If bit position 10 of the LPSD instruction contains a one, the highest priority interrupt level currently in the active state is reset to either the armed or the disarmed state. The interrupt level is armed if bit 11 of the LPSD instruction contains a one, or is disarmed if bit 11 of the LPSD instruction contains a zero. If bit 10 of the LPSD instruction contains a zero, no interrupt level is affected in any way.

Load Program Status Doubleword Phase Sequence. Preparation phases for the LPSD instruction are the same as the general PREP phases for doubleword instructions, paragraph 3-59. Figure 3-184 shows the simplified phase sequence for the LPSD instruction during execution. Table 3-83 lists the detailed logic sequence during all LPSD execution phases.

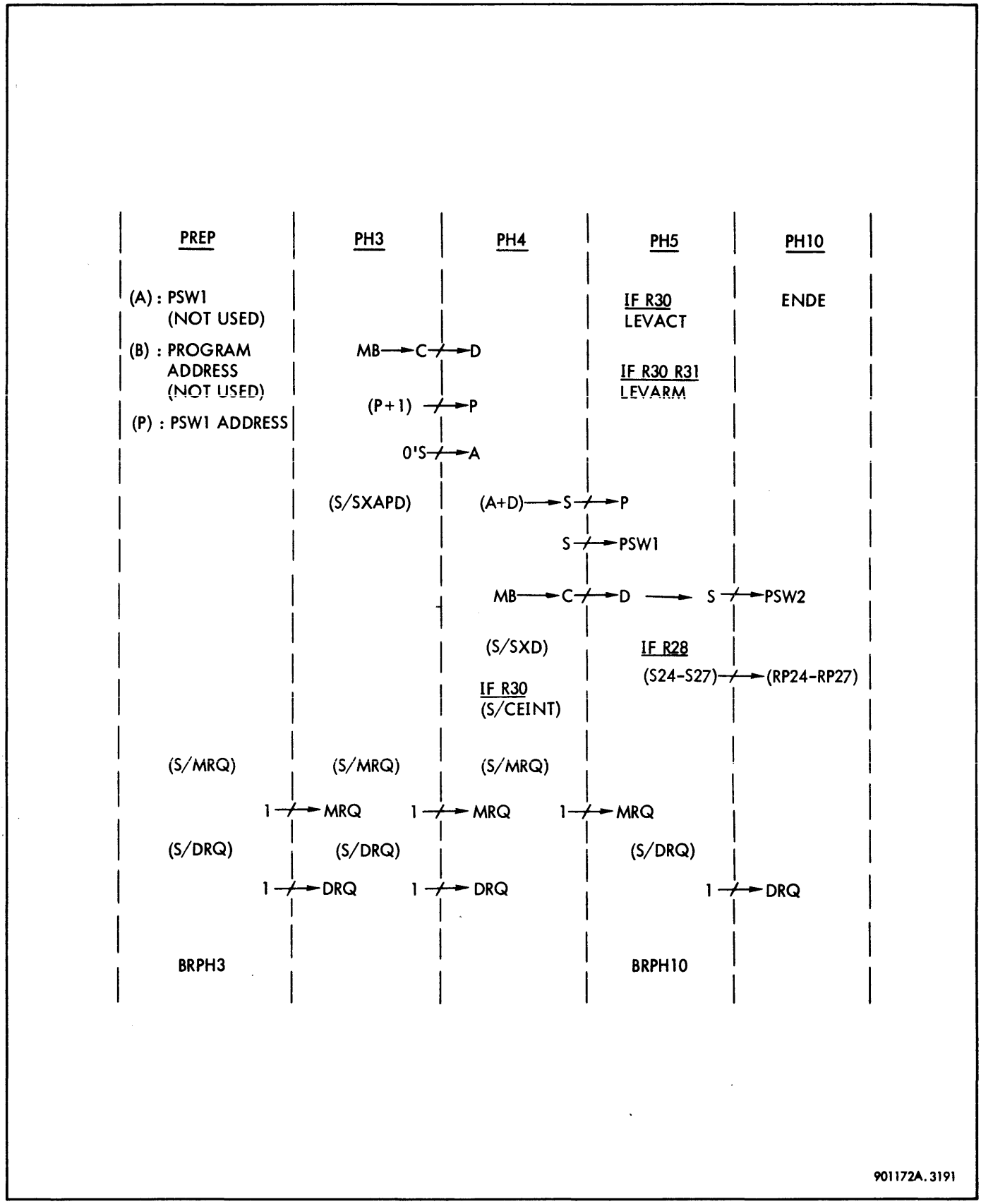


Figure 3-184. Load Program Status Doubleword Instruction, Phase Sequence Diagram

Table 3-83. Load Program Status Doubleword Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(A) : PSW1 (bits 0-3, 5-8, 10, 11)		Current PSW1 (not used)
	(B) : Program address		Address of next instruction in sequence (not used)
	(P) : PSW1 address		Address of first word of program status doubleword to be loaded
	Set flip-flop MRQ	S/MRQ = (S/MRQ) = (S/MRQ/2) + ... (S/MRQ/2) = FAPSD (PRE/34 + PH2) + ... FAPSD = OU0 O6 (O4 O5) R/MRQ = ...	Memory request for MB ⟶ C transfer in PH3
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = (S/MRQ/2) + ... R/DRQ = ...	Inhibits transmission of another clock until data release received from memory
	Branch to PH3	BRPH3 = FAPSD NO7 NANLZ PRE3 + ...	
PH3 DR	Sustained until DR		
	(MB0-MB31) ⟶ (C0-C31)	CXMB = DG = /DG/	Transfer addressed PSW1 to C-register
	(C0-C31) ⟶/⟶ (D0-D31)	DXC = FAPSD PH3 + ...	Transfer addressed PSW1 to D-register
	Enable signal (S/SXAPD)	(S/SXAPD) = FAPSD PH3 + ...	Preset adder for (A + D) ⟶ S in PH4
	Clear A-register	AXZ = FAPSD PH3 + ...	(A + D) becomes (0 + D)
	Upcount P-register	PUC31 = FAPSD (PH1 + PH3) + ...	Store address of PSW2 in P-register
	Set flip-flop MRQ	S/MRQ = (S/MRQ) = (S/MRQ/2) + ... (S/MRQ/2) = FAPSD PH3 + ... R/MRQ = ...	Core memory request for addressed PSW2
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = (S/MRQ/2) + ... R/DRQ = ...	Inhibits transmission of another clock until data release received from memory
PH4 DR	Sustained until DR		
	(MB0-MB31) ⟶/⟶ (C0-C31)	CXMB = DG = /DG/	Transfer addressed PSW2 to C-register
			Mnemonic: LPSD (0E, 8E)

(Continued)

Table 3-83. Load Program Status Doubleword Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH4 DR (Cont.)	(C0-C31)—/→(D0-D31)	DXC = FAPSD PH4 + ...	Transfer addressed PSW2 to D-register
	(A0-A31) + (D0-D31)——→ (S0-S31)	Adder logic set at PH3 clock	A-register cleared, therefore effectively a D——→S transfer
	(S15-S31)—/→(P15-P31)	PXS = FAPSD PH4 + ...	Store PSW2 address (AI) in P-register
	Set flip-flop CEINT if R30	S/CEINT = FPASD PH4 NO7 R30	Clock enable interrupt
		R/CEINT = ...	
	Store bits 0-3, 5-8, 10, and 11 of PSW1 in flip-flops	S/CC1 = S0 CCXS/0 + ...	Condition code
		CCXS/0 = PSW1XS + ...	
		PSW1XS = FAPSD PH4 + ...	
		R/CC1 = (R/CC1) = (R/CC) + ...	
		(R/CC) = CCXS/0 + ...	
		S/CC2 = S1 CCXS/0 + ...	
		S/CC3 = S2 CCXS/0 + ...	
		S/CC4 = S3 CCXS/0 + ...	
		R/CC2 = R/CC3 = R/CC4 = (R/CC) + ...	
		S/FS = S5 PSW1XS + ...	Floating significant mask
		R/FS = PSW1XS + ...	
		S/FZ = S6 PSW1XS + ...	Floating zero mask
		R/FZ = PSW1XS + ...	
		S/FNF = S7 PSW1XS + ...	Floating normalize mask
		R/FNF = ...	
		S/NMASTER = S8 PSW1XS + ...	Master/slave mode control
		R/NMASTER = PSW1XS + ...	
		S/DM = S10 PSW1XS + ...	Decimal fault trap mask
		R/DM = PSW1XS + ...	
		S/AM = S11 PSW1XS + ...	Fixed point arithmetic overflow trap mask
		R/AM = PSW1XS + ...	
	Enable signal (S/SXD)	(S/SXD) = FAPSD PH4 + ...	Preset adder for D——→S transfer in PH5
	Set flip-flop MRQ	S/MRQ = (S/MRQ) = (S/MRQ/1) + ...	Core memory request for addressed PSW2
		(S/MRQ/1) = FAPSD PH4 + ...	
		R/MRQ = ...	
			Mnemonic: LPSD (0E, 8E)

(Continued)

Table 3-83. Load Program Status Doubleword Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH5	One clock long		
T5L	(D0-D31) ⟶ (S0-S31)	Adder logic set at PH4 clock	Transfer addressed PSW2 to sum bus
	Store bits 34, 35, 37, 38, and 39 of program status doubleword (bits 2, 3, 5, 6, and 7 of PSW2)	S/WK0 = S2 PSW2XS PSW2XS = FAPSD PH5 + ... R/WK0 = PSW2XS	Write key bit 0
		S/WK1 = S3 PSW2XS R/WK1 = PSW2XS	Write key bit 1
		S/CIF = (S/CIF/2) + ... = S5 PSW2XS R/CIF = PSW2XS/1 + ...	Counter interrupt bit
		PSW2XS/1 = PSW2XS NO7	For LPSD, PSW2XS/1 = PSW2XS
		S/II = S6 PSW2XS/1 R/II = (R/I) = PSW2XS/1 + ...	I/O interrupt bit
		S/EI = S7 PSW2XS + ... R/EI = PSW2XS/1 + ...	External interrupt bit
	If R28, store register pointer bits	S/RP24 = S24 RPXS + ... ⋮ ⋮ S/RP27 = S27 RPXS + ... RPXS = PSW2XS R28 + ... R/RP24 = R/RP25 = R/RP26 = R/RP27 = RPXS	If R28 not set, register pointer bits (56-59) of program status doubleword not changed
	Enable signal LEVACT (if R30)	LEVACT = FAPSD PH5 NO7 R30 + ...	Clear highest priority interrupt in active state
	Enable signal LEVARM (if R30 R31)	LEVARM = FAPSD PH5 NO7 R30 R31 + ...	Arm interrupt level
	Reset flip-flop TRAP	R/TRAP = (R/TRAP) (R/TRAP) = FAPSD PH5 + ...	
	Reset flip-flops TRACC/1 through TRACC/4	R/TRACC/1 = (R/TRACC) R/TRACC/4 = (R/TRACC) (R/TRACC) = FAPSD PH5	
PH6	Branch to PH10	BRPH10 = FAPSD PH6 + ...	
DR	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = BRPH10 + ... R/DRQ = ...	Inhibits transmission of another clock until data release received from memory
PH10 DR	ENDE functions	See table 3-18	
			Mnemonic: LPSD (0E, 8E)

EXCHANGE PROGRAM STATUS DOUBLEWORD (XPSD; 0F, 8F). The XPSD instruction stores the entire current program status doubleword (PSD) and replaces the current PSD with a new PSD. Bits 0 through 35 of the current PSD are unconditionally replaced by bits 0 through 35 of the new PSD. Bits 37 through 39 and bits 56 through 59 of the current PSD are conditionally modified.

General. A program status doubleword is stored in memory as a 64-bit word in two consecutive memory locations. The current PSD is stored in flip-flops and registers of the Sigma 5. The relation between the two storage locations is indicated in table 3-82.

Standard Operations. Word 1 (PSW1) of the current PSD is stored in the location pointed to by the effective address of the XPSD instruction. The P-register count is then incremented by one, and word 2 (PSW2) of the current PSD is stored in the next consecutive location. The P-register count is incremented once more, and the PSW1 of the new PSD is fetched from memory and stored in flip-flops and registers of the Sigma 5. (Refer to bits 0 through 31 in table 3-82.) The P-register count is incremented once again, PSW2 of the new PSD is fetched from memory, and bits 32 through 36 are stored. The contents of bits 37 through 63 of the new PSD are dependent upon additional data.

Conditional Operations. If bit position 8 of the XPSD instruction contains a one, bits 56 through 59 of the current PSD (register pointer bits) are replaced by bits 56 through 59 of the new PSD (bits 24 through 27 of the new PSW2). If bit 8 of the XPSD instruction contains a zero, the current register pointer bits are not changed. An OR operation is performed between bits 37 through 39 of the current PSD and the corresponding bits in the new PSD fetched from memory. For these bit positions, if the bit in the new PSD is a zero, the corresponding bit of the current PSD is stored in the new PSD without change. If the bit in the new PSD is a one, the corresponding bit of the new PSD is set to 1. For example:

Current PSD	New PSD	Stored PSD (final value)
101	000	101
101	010	111
001	100	101
000	110	110
011	101	111

Trap Operations. If the XPSD instruction is executed because of a nonallowed operation or a CAL instruction, the operations illustrated in figure 3-185 are performed. Information stored in flip-flops TRACC1 through TRACC4 causes flip-flops CC1 through CC4 to be set, and results in arithmetic operations or logic operations during execution of the XPSD instruction.

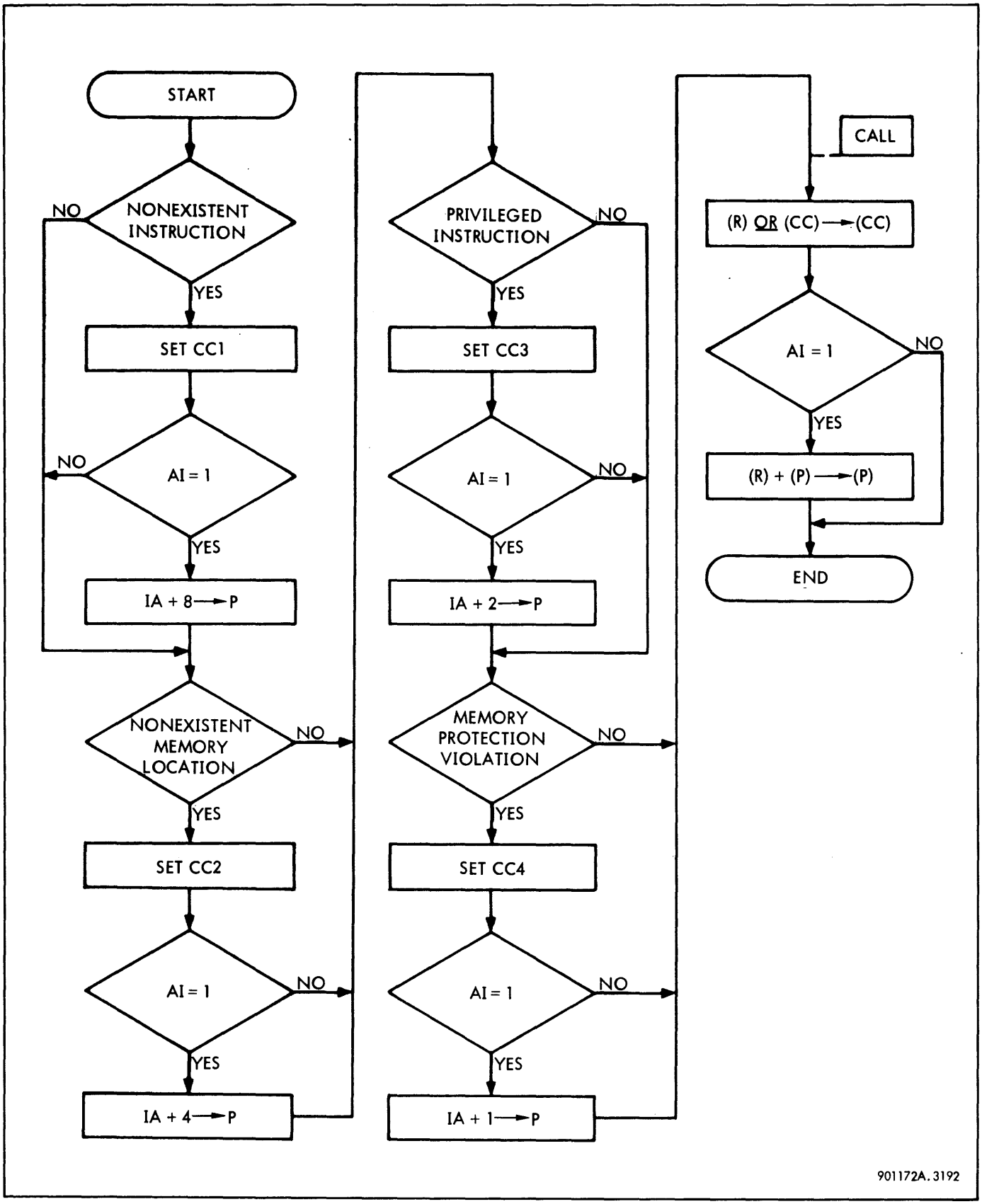


Figure 3-185. Exchange Program Status Doubleword Instruction, Flow Diagram

Exchange Program Status Doubleword Phase Sequence. Preparation phases for the XPSD instruction are the same as the general PREP phases for doubleword instructions, paragraph 3-59. Figure 3-186 shows the simplified phase sequence for the XPSD instruction during execution. Table 3-84 lists the detailed logic sequence during all XPSD execution phases.

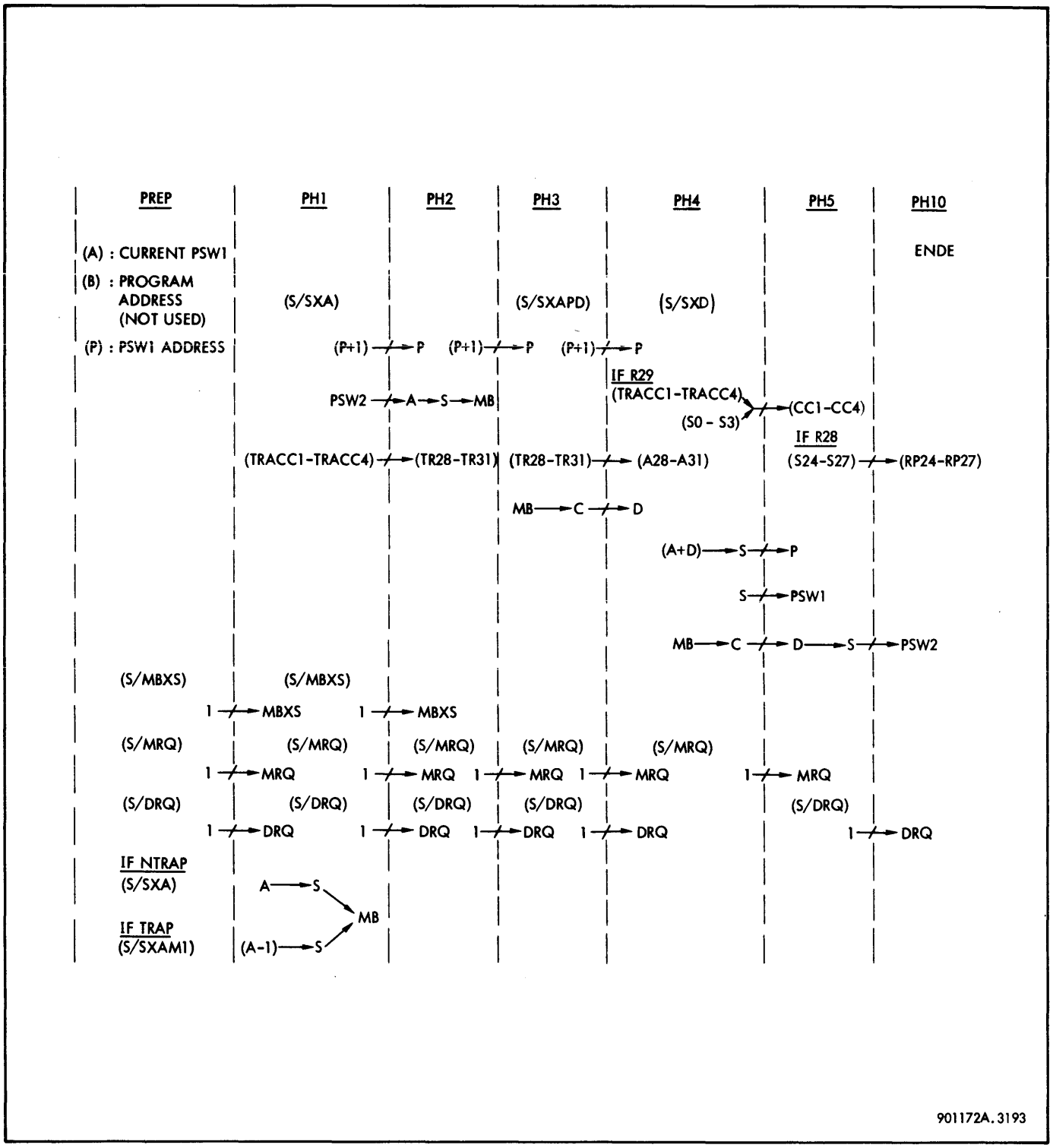


Figure 3-186. Exchange Program Status Doubleword Instruction, Phase Sequence Diagram

Table 3-84. Exchange Program Status Doubleword Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(A) : PSW1 (bits 0-3, 5-8, 10, and 11)	S/A0 = CC1 AXPSW1 + ...	Condition code
		S/A1 = CC2 AXPSW1 + ...	
		S/A2 = CC3 AXPSW1 + ...	
		S/A3 = CC4 AXPSW1 + ...	
		AXPSW1 = FAPSD PRE3 + ...	
		FAPSD = OU0 O6 (O4 O5)	
		R/A0-R/A3 = AX/0 + ...	
		AX/0 = AX + ... = AXZ + ...	
		AXZ = FAPSD PRE3 + ...	
		S/A5 = FS AXPSW1 + ...	Floating significant mask
		S/A6 = FZ AXPSW1 + ...	Floating zero mask
		S/A7 = FNF AXPSW1 + ...	Floating normalize mask
		S/A8 = NMASTER AXPSW1 + ...	Master/slave mode control
		S/A10 = DM AXPSW1 + ...	Decimal fault trap mask
		S/A11 = AM AXPSW1 + ...	Arithmetic trap mask
		R/An = AX/m = AX + ... = AXZ + ...	
	(B) : Program address		Address of next instruction in sequence (not used)
	(P) : PSW1 address		Address into which PSW1 of current PSD is to be stored
	Set flip-flop MBXS	S/MBXS = (S/MBXS)	Preset for S→MB in PH1
		(S/MBXS) = FAPSD PRE3 O7 + ...	
		R/MBXS = ...	
	Set flip-flop MRQ	S/MRQ = (S/MRQ) = (S/MRQ/2) + ...	Memory request for S→MB transfer in PH1
		(S/MRQ/2) = FAPSD (PRE/34 + PH2) + ...	
		R/MRQ = ...	
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR	Inhibits transmission of another clock until data release received from memory
		(S/DRQ) = (S/MRQ/2) + ...	
		R/DRQ = ...	
	If NTRAP:		
	Enable signal (S/SXA)	(S/SXA) = FAPSD PRE3 NTRAP + ...	Preset adder for A→S in PH1
			Mnemonic: XPSD (0F, 8F)

(Continued)

Table 3-84. Exchange Program Status Doubleword Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PREP (Cont.)	If TRAP:		
	Enable signal (S/SXAM1)	(S/SXAM1) = FAPSD PRE3 TRAP + ...	Preset adder for (A - 1) ⟶S in PH1
PH1 DR	Sustained until DR		
	If NTRAP:		
	(A0-A31) ⟶(S0-S31)	Adder logic set at last PREP clock	Transfer address bits (15-31) without change
	If TRAP:		
	(A0-A31) - 1 ⟶(S0-S31)	Adder logic set at last PREP clock	Decrement address bits (15-31) and transfer
	(S0-S31) ⟶(MB0-MB31)	MBXS = Set at last PREP clock	Transfer current PSW1 from A-register to memory (with or without decrement)
	Upcount P-register	PUC31 = FAPSD (PH1 + PH3) + ...	Store current PSW2 address in P-register
	Store bits 34, 35, 37-39, and 56-59 of current PSD (bits 2, 3, 5-7, and 24-27 of PSW2)	S/A2 = WK0 AXPSW2 + ...	Write key bit 0
		S/A3 = WK1 AXPSW2 + ...	Write key bit 1
		S/A5 = CIF AXPSW2 + ...	Counter interrupt bit
		S/A6 = II AXPSW2 + ...	I/O interrupt bit
		S/A7 = EI AXPSW2 + ...	External interrupt bit
		S/A24 = RP24 AXPSW2 + ...	Register pointer bits
		⋮ ⋮	
		S/A27 = RP27 AXPSW2 + ...	
		AXPSW2 = FAPSD PH1 + ...	
		R/An = AXm + ... = AX + ...	
		AX = AXPSW2 + ...	
	(TRACC1-TRACC4) ⟶ (TR28-TR31)	S/TR28 = NSTRAP (S/TR28) + ...	TRACC1-TRACC4 contain code stored by CAL instruction or by response to a nonallowed operation
		(S/TR28) = TRACC1 FAPSD PH1	
		R/TR28 = (R/TR) = (R/TRACC/1) + ...	
		(R/TRACC/1) = FAPSD PH5	
		S/TR29 = TRACC2 FAPSD PH1	
		S/TR30 = TRACC3 FAPSD PH1	
		S/TR31 = TRACC4 FAPSD PH1	
		R/TR29 = R/TR30 = R/TR31 = (R/TR)	
			Mnemonic: XPSD (0F, 8F)

(Continued)

Table 3-84. Exchange Program Status Doubleword Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH1	Enable signal (S/SXA)	(S/SXA) = FAPSD PH1	Preset for A⟶S in PH2
DR (Cont.)	Set flip-flop MBXS	S/MBXS = (S/MBXS) = FAPSD PH1 + ... R/MBXS = ...	Preset for S⟶MB in PH2
	Set flip-flop MRQ	S/MRQ = (S/MRQ) = (S/MBXS) + ... R/MRQ = ...	Memory request for S⟶MB transfer in PH2
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = (S/MBXS) + ... R/DRQ = ...	Inhibits transmission of another clock until data release received from memory
PH2	Sustained until DR		
DR	(A0-A31)⟶(S0-S31)	Adder logic set at PH1 clock	Transfer current PSW2 to sum bus
	(S0-S31)⟶(MB0-MB31)	MBXS = Set at PH1 clock	Transfer current PSW2 to memory
	Set flip-flop MRQ	S/MRQ = (S/MRQ) = (S/MRQ/2) + ... (S/MRQ/2) = FAPSD (PRE/34 + PH2) + ... R/MRQ = ...	Core memory request for PSW1 of new PSD
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = (S/MRQ/2) + ... R/DRQ = ...	Inhibits transmission of another clock until data release received from memory
	Upcount P-register	PUC31 = FAPSD PH2 + ...	Store address of new PSW1 in P-register
PH3	Sustained until DR		
DR	(MB0-MB31)⟶(C0-C31)	CXMB = DG = /DG/	Transfer new PSW1 to C-register
	(C0-C31)⇢(D0-D31)	DXC = FAPSD PH3 + ...	Transfer new PSW1 to D-register
	Enable signal (S/SXAPD)	(S/SXAPD) = FAPSD PH3 + ...	Preset adder for (A + D) ⟶S in PH4
	Upcount P-register	PUC31 = FAPSD (PH1 + PH3)	Store address of new PSW2 of PSD in P-register
	(TR28-TR31)⇢(A28-A31)	S/A28 = TR28 AXTR + ... AXTR = FAPSD PH3 TRAP R29 O7 R/A28 = AX/3 + ... = AX + ... AX = AXZ + ... = FAPSD PH3 + ...	Code stored in TR28-TR31 from TRACC1-TRACC4 transferred to A-register, and other A-register flip-flops reset (if bit 9 of XPSD instruction a 1)
			Mnemonic: XPSD (0F, 8F)

(Continued)

Table 3-84. Exchange Program Status Doubleword Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH3 DR (Cont.)		S/A29 = TR29 AXTR + ... S/A30 = TR30 AXTR + ... S/A31 = TR31 AXTR + ...	
	Set flip-flop MRQ	S/MRQ = (S/MRQ) = (S/MRQ/2) + ... (S/MRQ/2) = FAPSD PH3 + ... R/MRQ = ...	Core memory request for PSW2 of new PSD
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = (S/MRQ/2) + ... R/DRQ = ...	Inhibits transmission of another clock until data release received from memory
PH4	Sustained until DR		
DR	(MB0-MB31)⟶(C0-C31)	CXMB = DG = /DG/	Transfer PSW1 of new PSD to C-register
	(C0-C31)⟶(D0-D31)	DXC = FAPSD PH4 + ...	Transfer PSW1 of new PSD to D-register
	(A0-A31) + (D0-D31)⟶(S0-S31)	Adder logic set at PH3 clock	Add code stored in A28-A31 to AI of PSW1
	(S15-S31)⟶(P15-P31)	PXS = FAPSD PH4 + ...	Store AI in P-register
	Set condition code flip-flops	S/CC1 = CCXTRACC TRACC1 + S0 CCXS/0 + ... CCXTRACC = FAPSD PH4 O7 TRAP CCXS/0 = PSW1XS + ... = FAPSD PH4 + ... R/CC1 = (R/CC1) = (R/CC) + ... (R/CC) = (CCXS/0) + ...	Each CC flip-flop stores corresponding bit of PSW1 of new PSD if NTRAP, or stores bit from corresponding TRACC if TRAPP
		General Equations:	
	TRACC1 + S0⟶CC1 TRACC2 + S1⟶CC2 TRACC3 + S2⟶CC3 TRACC4 + S3⟶CC4	S/CCn = (S/CCn/3) + ... = (S/CCn/1) + ... (S/CCn/1) = CCXTRACC TRACCn + ... S/CCn = Sm CCXS/0 + ... (CCXS/0) = PSW1XS + ... R/CCn = (R/CCn) = (R/CC) + ... (R/CC) = (CCXS/0) + ...	
			Mnemonic: XPSD (0F, 8F)

(Continued)

Table 3-84. Exchange Program Status Doubleword Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH4 DR (Cont.)	Store bits 5-8, 10, and 11 of PSW1 of new PSD in flip-flops	S/FS = S5 PSW1XS + ...	Floating significant mask
		R/FS = PSW1XS + ...	
		S/FZ = S6 PSW1XS + ...	Floating zero mask
		R/FZ = PSW1XS + ...	
		S/FNF = S7 PSW1XS + ...	Floating normalize mask
		R/FNF = PSW1XS + ...	
		S/NMASTER = S8 PSW1XS + ...	Master/slave mode control
		R/NMASTER = PSW1XS + ...	
		S/DM = S10 PSW1XS + ...	Decimal fault trap mask
		R/DM = PSW1XS + ...	
		S/AM = S11 PSW1XS + ...	Arithmetic trap mask
		R/AM = PSW1XS + ...	
	Enable signal (S/SXD)	(S/SXD) = FAPSD PH4 + ...	Preset adder for D⟶S transfer in PH5
	Set flip-flop MRQ	S/MRQ = (S/MRQ) = (S/MRQ/1) + ...	Core memory request for next instruction in sequence
		(S/MRQ/1) = FAPSD PH4 + ...	
		R/MRQ = ...	
PH5	One clock long		
T5L	(D0-D31)⟶(S0-S31)	Adder logic set at PH4 clock	Transfer new PSW2 to sum bus
	Store bits 34, 35, and 37-39 of new PSD (bits 2, 3, and 5-7 of PSW2)	S/WK0 = S2 PSW2XS	Write key bit 0
		R/WK0 = PSW2XS	
		S/WK1 = S3 PSW2XS	Write key bit 1
		R/WK1 = PSW2XS	
		PSW2XS = FAPSD PH5 + ...	
	For bits 5-7, perform OR operation between input data and stored data	S/CIF = (S/CIF/2) + ... = S5 PSW2XS	Counter interrupt bit
		R/CIF = PSW2XS/1 + ...	
		PSW2XS/1 = PSW2XS NO7	Enable OR operation
		S/II = S6 PSW2XS + ...	I/O interrupt bit
		R/II = (R/1) = PSW2XS/1	
		S/EI = S7 PSW2XS + ...	External interrupt bit
		R/EI = PSW2XS/1	
			Mnemonic: XPSD (0F, 8F)

(Continued)

Table 3-84. Exchange Program Status Doubleword Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH5 T5L (Cont.)	Store new register pointer bits if bit 8 of XPSD instruction was 1 (now stored in R28)	S/RP24 = S24 RPXS S/RP25 = S25 RPXS S/RP26 = S26 RPXS S/RP27 = S27 RPXS RPXS = PSW2XS R28 + ... R/RP24-R/RP27 = RPXS	If bit 8 is a 0, no change in register pointer bits
	If TRAP, reset flip-flops	R/INTRAP = (R/INTRAP) − (R/TRAP) + ... (R/TRAP) = FAPSD PH5 + ... R/TRAP = (R/TRAP) R/TRACC1-R/TRACC4 = FAPSD PH5	
	Branch to PH10	BRPH10 = FAPSD PH5 + ...	
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = BRPH10 + ... R/DRQ = ...	Inhibits transmission of another clock until data release received from memory
PH10 DR	ENDE functions		
			Mnemonic: XPSD (0F, 8F)

3-79 Move to Memory Control (MMC; 6F, EF)

GENERAL. The memory protection feature of the Sigma 5 computer includes a block of 256 write-locks. Each write-lock consists of two flip-flops. Each two-bit configuration in a write-lock, and the configuration in the write-key of the program status doubleword, control a page of core memory, or 512 memory locations. All 256 write-locks control the maximum core memory of 128K locations. The 512-bit block of write-locks is byte-addressable, that is, four write-locks at a time may be addressed and their contents modified or read. The first group of four write-locks has an address of 0000000_2 and controls pages 0 through 3 of core memory. The last group of four write-locks has an address of 111111_2 and controls pages 252 through 255. Figure 3-187 shows the write-lock block and its relationship to core memory. The memory protection feature is discussed in detail in paragraph 3-59.

The MMC instruction loads one or more words from core memory into the write-lock block, thereby modifying 16 or more write-locks. The words to be loaded, taken together, are called the lock image.

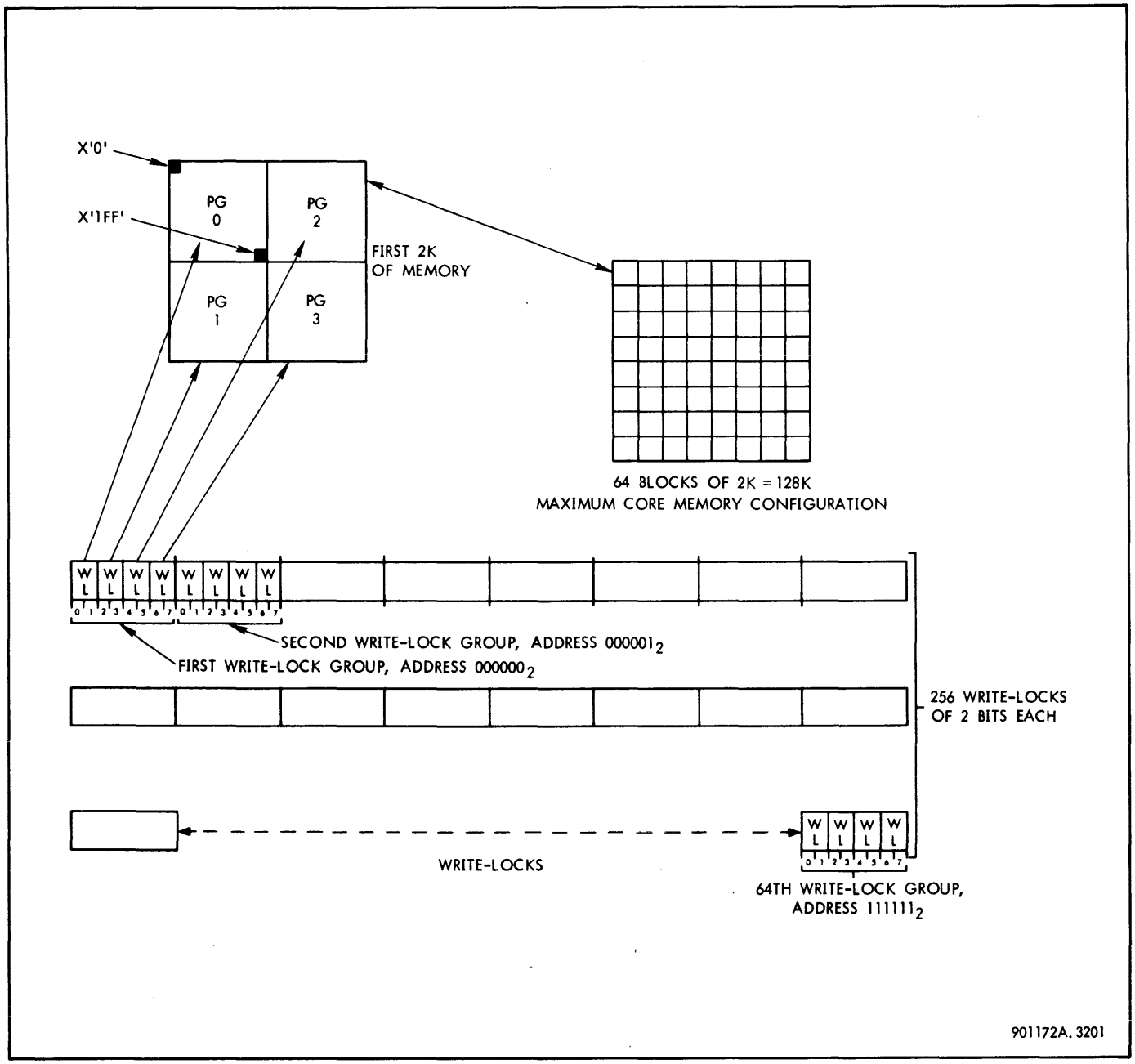


Figure 3-187. Write-Lock Configuration

INSTRUCTION FORMAT. The R field of the instruction word defines a pair of private memory registers that contain additional data for the instruction as shown below. The R field of the instruction word must be even for correct results.

The lock image address, in private memory register R (figure 3-188), is the address of the first core memory word to be loaded into the write-locks. The count field of private memory register Ru1 (figure 3-188) contains the number of words to be loaded into the write-locks. If this field contains all zeros, 256 words from core memory are to be loaded; otherwise one through 255 words are loaded. If the count field specifies a value greater than 16, the write-locks are loaded in circular fashion and some or all of the write-locks are overwritten. The control start field of private memory register Ru1 points to the address of the first four write-locks to be modified. An example of MMC is shown in figure 3-189.

MOVE TO MEMORY CONTROL PHASE SEQUENCES. Preparation phases for MMC are the same as the general PREP phases for word instructions. Figure 3-190 shows the simplified phase sequence for the instruction during execution, and table 3-85 lists the detailed logic sequence during all MMC execution phases.

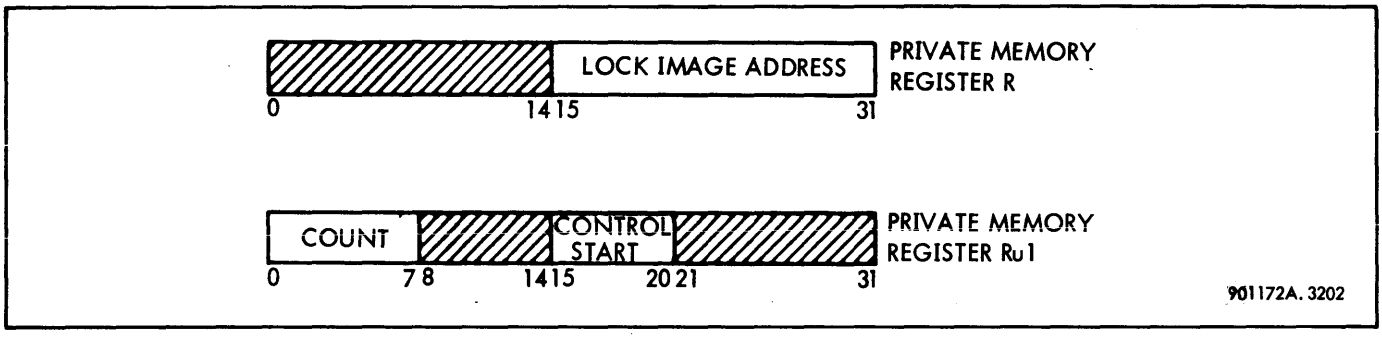


Figure 3-188. Contents of Private Memory Registers R and Ru1

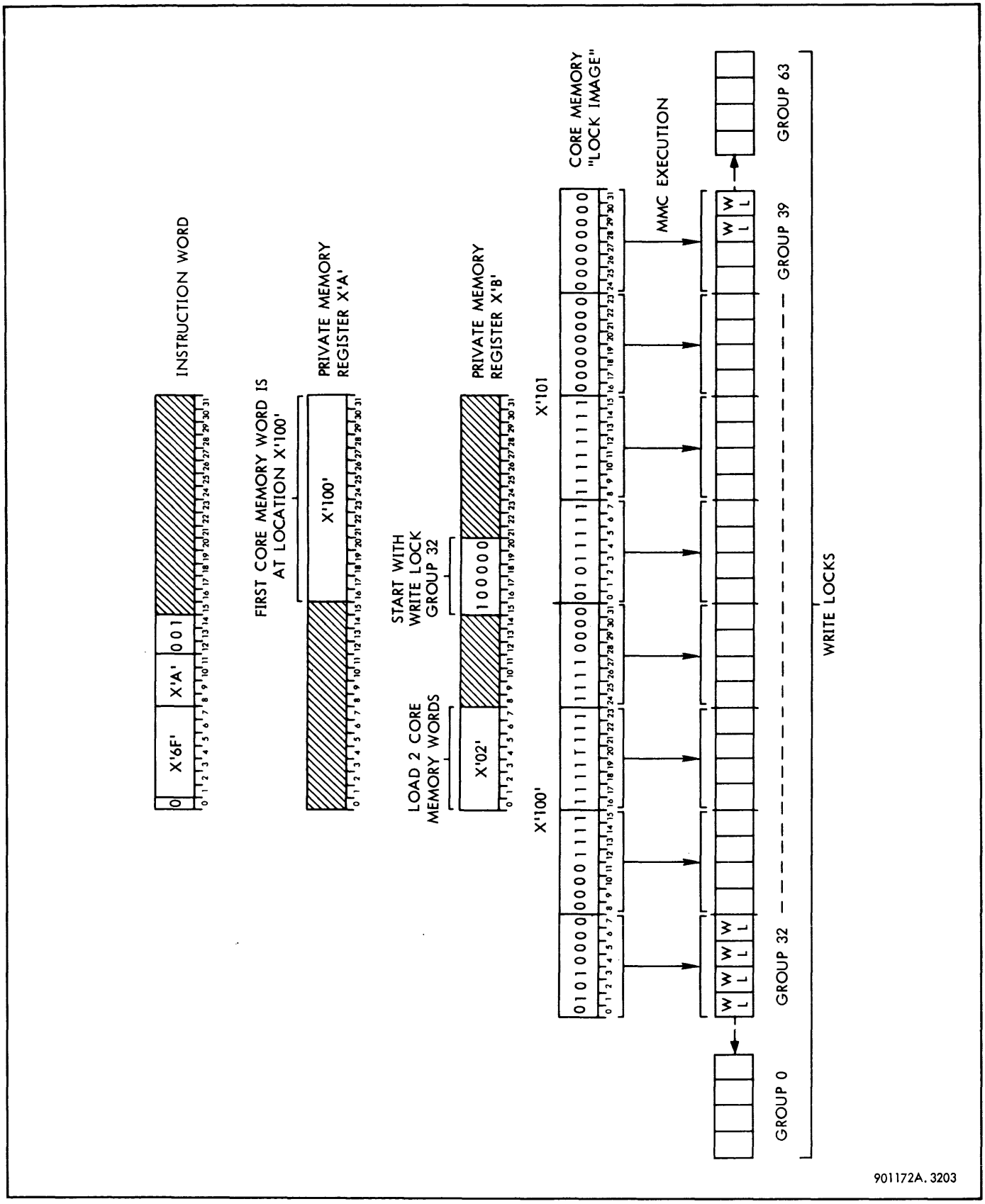


Figure 3-189. Move to Memory Control Example

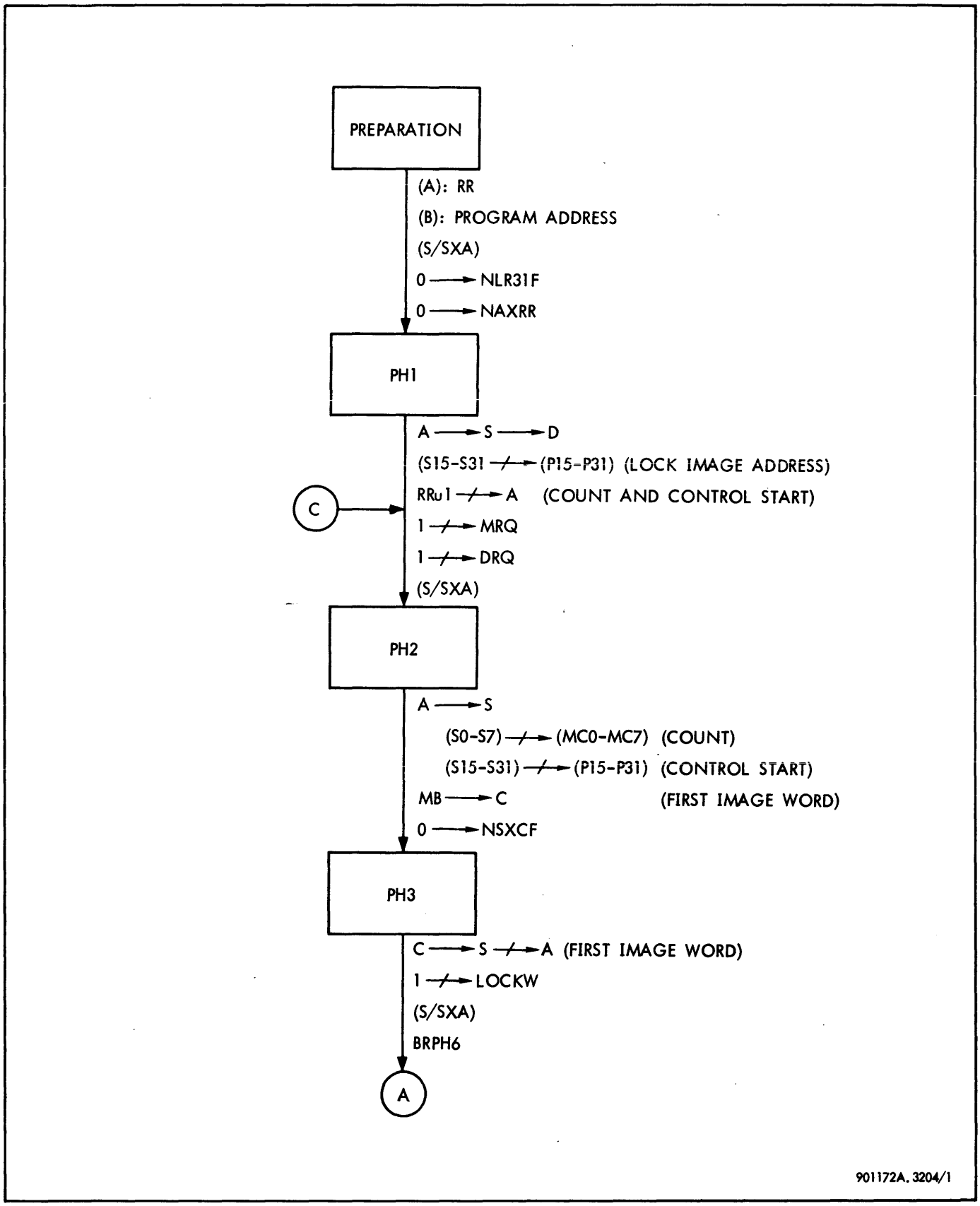


Figure 3-190. Move to Memory Control, Flow Diagram (Sheet 1 of 3)

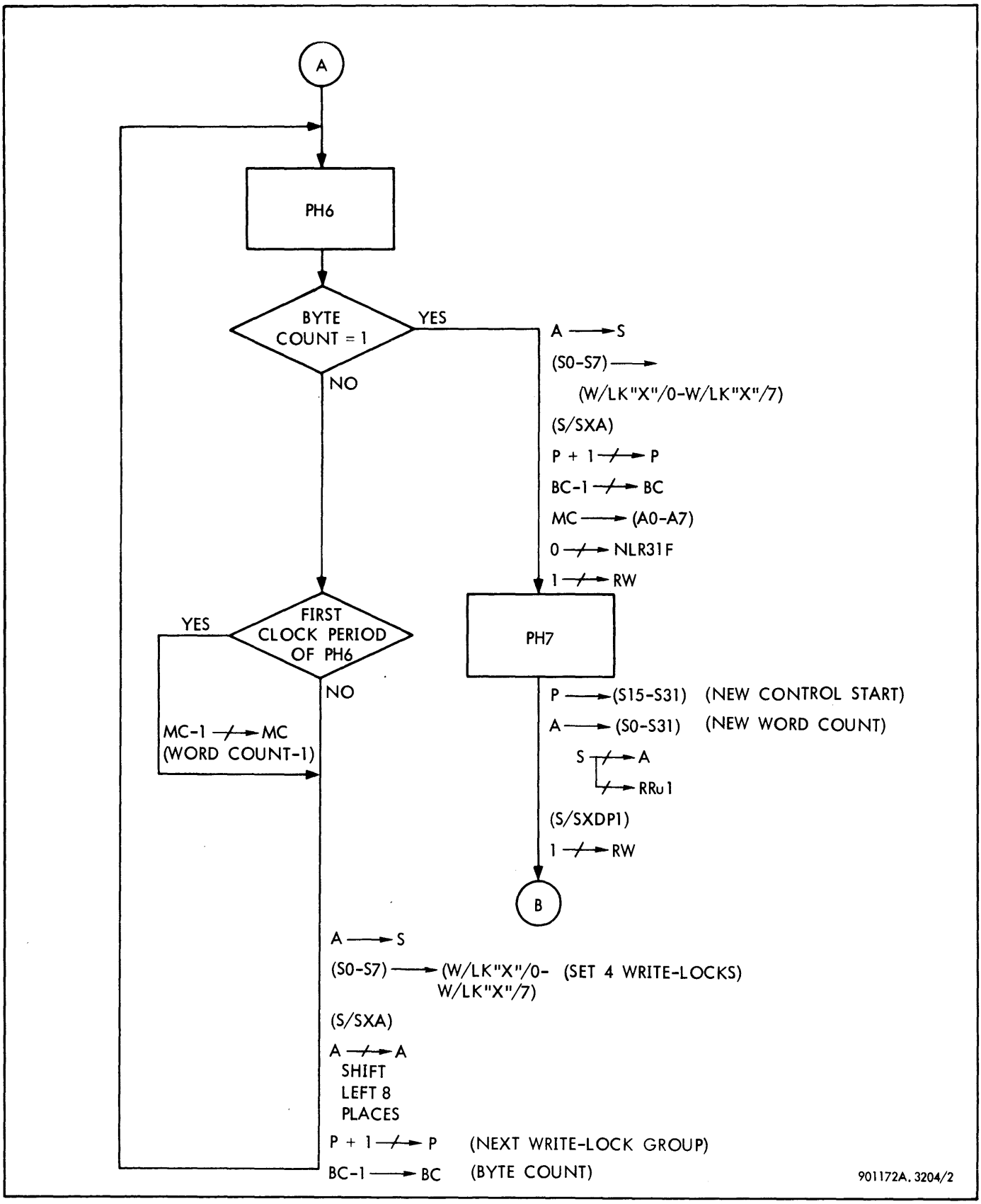


Figure 3-190. Move to Memory Control, Flow Diagram (Sheet 2 of 3)

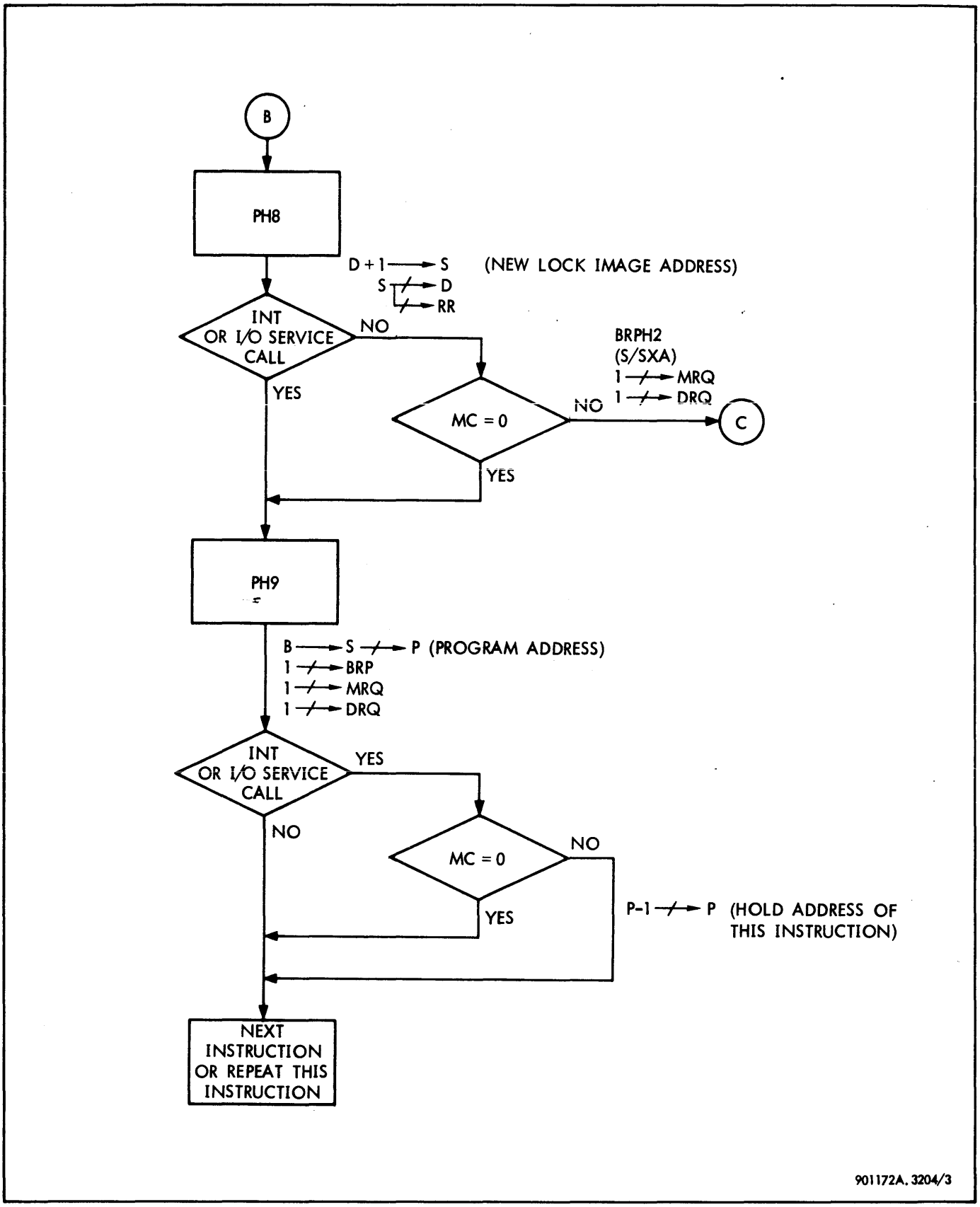


Figure 3-190. Move to Memory Control, Flow Diagram (Sheet 3 of 3)

Table 3-85. Move to Memory Control Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(A) : RR		Contents of private memory register R. The lock image address points to the first word of the lock image to be loaded into memory control registers
	(B) : Program address		Next instruction in sequence
	Enable signal (S/SXA)	(S/SXA) = FUMMC PRE3 + ... FUMMC = OU6 OLF	Preset adder for A⟶S in PH1
	Reset flip-flop NLR31F	S/NLR31F = N(S/LR31) (S/LR31) = FUMMC NANLX PRE3 + ... R/NLR31F = ...	Force a one onto private memory address line LR31 to select private memory register Ru1 in PH1
	Reset flip-flop NAXRR	S/NAXRR = N(S/AXRR) (S/AXRR) = FUMMC PRE3 + ... R/NAXRR = ...	Prepare to transfer contents of private memory register Ru1 to A-register
PH1	One clock long		
T5L	(A0-A31)⟶(S0-S31)	Adder logic set at last PREP clock	Transfer address of first word of lock image to P- and D-registers
	(S0-S31)⟶(D0-D31)	DXS = FUMMC PH1 + ...	
	(S15-S31)⟶(P15-P31)	PXS = FUMMC PH1 + ...	
	(RR0-RR31)⟶(A0-A31)	AXRR = Set at last PREP clock	Transfer contents of private memory register Ru1 to A-register. Private memory register Ru1 contains word count and control start
	Set flip-flop MRQ	S/MRQ = (S/MRQ/2) + ... (S/MRQ/2) = FUMMC PH1 + ... R/MRQ = ...	Core memory request for first word of control image

Mnemonic: MMC (6F, EF)

(Continued)

Table 3-85. Move to Memory Control Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH1 T5L (Cont.)	Set flip-flop DRQ	S/DRQ = (S/MRQ/2) + ... R/DRQ = ...	Inhibits transmission of another clock until data release signal received from core memory
	Enable signal (S/SXA)	(S/SXA) = FUMMC PH1 + ...	Preset adder for A ⟶ S in PH2
PH2	Sustained until data release		
DR	(A0-A31) ⟶ (S0-S31)	Adder logic set at PH1 clock	
	(S0-S7) ⟶/ (MC0-MC7)	MCXS = FUMMC PH2	Transfer word count to macro-counter
	(S15-S31) ⟶/ (P15-P31)	PXS = FUMMC PH2 + ...	Transfer control start to P-register
	(MB0-MB31) ⟶ (C0-C31)	CXMB = DG = /DG/	Transfer first image word to C-register
	Reset flip-flop NSXCF	S/NSXCF = N(S/SXC) N(FAST/S PH6 NMCZ) (S/SXC) = FUMMC PH2 + ... R/NSXCF = ...	Preset logic for C ⟶ S in PH3
PH3	One clock long		
T5L	(C0-C31) ⟶ (S0-S31)	SXC = NDIS SXCF + ... SXCF = Set at PH2 clock	Transfer first image word to A-register
	(S0-S31) ⟶/ (A0-A31)	AXS = FUMMC PH3 + ...	
	Set flip-flop LOCKW	S/LOCKW = FUMMC PH3 + ... R/LOCKW = ...	LOCKW enables writing into memory control registers
	Enable signal (S/SXA)	(S/SXA) = FUMMC PH3 + ...	Preset adder for A ⟶ S in PH4
	Branch to PH6	BRPH6 = FUMMC PH3 + ... S/PH6 = BRPH6 NIOEN NCLEAR + ... R/PH6 = ...	

Mnemonic: MMC (6F, EF)

(Continued)

Table 3-85. Move to Memory Control Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH6 T8L	Four clocks long. During first three clock periods perform the following operations:		
	(A0-A31) ⟶ (S0-S31)	Adder logic set at PH3 clock or previous PH6 clock	
	(S0-S7) ⟶ (W/LK'X'/0-W/LK'X'/7)	W/LK'X'/0 = S0 ⋮ ⋮ W/LK'X'/7 = S7	Byte 0 of sum bus is transferred to four write-locks pointed to by P15-P20. 'X' is 1, 2, 3, or 4. K/ is clock enable
		K/LK0-K/LK3 = LOCKW CK-32P43	
	Enable signal (S/SXA)	(S/SXA) = FUMMC PH6 + ...	Preset adder for A ⟶ S in next clock period
	(A8-A31) ⟶/⟶ (A0-A23)	AXAL8 = FUMMC PH6 N(BC = 1) + ...	Shift A-register left eight bit positions; next byte of image to byte 0 location
	Zeros ⟶/⟶ (A24-A31)		
	Increment (P15-P20) by one	PUC20 = FUMMC PH6	Point to next group of four write-locks
	Decrement byte count in BC0, BC1 by one	BCDC1 = FUMMC PH6 + ...	Count at beginning of PH6 is 00_2. BC0, BC1 are decremented with each clock of PH6. 01_2 signals the end of PH6, since on next clock all four bytes from the first image word will have been written into 16 write-locks
	Sustain PH6	BRPH6 = FUMMC PH6 N(BC = 1) + ...	
		S/PH6 = BRPH6 NIOEN NCLEAR + ...	
		IOEN = Not enabled for MMC PH6	
		R/PH6 = ...	
	Decrement macro-counter by one at first clock of PH6	MCDC7 = FUMMC PH6 BCZ + ...	Macro-counter now contains the original word count minus 1. PH6 will be re-entered after PH8 if word count is not zero

Mnemonic: MMC (6F, EF)

(Continued)

Table 3-85. Move to Memory Control Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH6 T8L (Cont.)	At last clock (BC = 1) perform the following operations:		
	(A0-A31)⟶(S0-S31)	Adder logic set at third PH6 clock	
	(S0-S7) ⟶ (W/LK'X'/0-W/LK'X'/7)	W/LK'X'/0 = S0 ⋮ ⋮ W/LK'X'/7 = S7 K/LK0-K/LK3 = LOCKW CK-32P43	Last byte of current image word transferred to write-lock write lines. 'X' is 1, 2, 3, or 4. K/ is clock enable. Write-lock address lines go to P15-P20
	Enable signal (S/SXA)	(S/SXA) = FUMMC PH6 + ...	Preset adder for A⟶S in PH7
	Increment (P15-P20) by one	PUC20 = FUMMC PH6	Point to next group of four write-locks. This group will be modified during next PH6 (if any)
	Decrement byte count in BC0, BC1 by one	BCDC1 = FUMMC PH6 + ...	BC0, BC1 now hold 00_2 for next PH6 (if any)
	(MC0-MC7)⟶(A0-A7)	AXMC = FUMMC PH6 (BC = 1)	Transfer updated word count to A-register
	Reset flip-flop NLR31F	S/NLR31F = N(S/LR31) (S/LR31) = FUMMC PH6 (BC = 1) + ... R/NLR31F = ...	Force a one on private memory address line LR31 to select private memory register Ru1 in PH7
	Set flip-flop RW	S/RW = (S/RW/1) (S/RW/1) = FUMMC PH6 (BC = 1) + ... R/RW = ...	Prepare to write new word count and new control start into private memory register Ru1
	Branch to PH7	NBRPH6 = (BC = 1) + ... S/PH7 = PH6 NBR NIOEN + ... R/PH7 = ...	
			Mnemonic: MMC (6F, EF)

(Continued)

Table 3-85. Move to Memory Control Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH7	One clock long		
T8L	(P15-P31)⟶(S15-S31)	SXP = FUMMC PH7 NDIS + ...	New control start in (P15-P20) and new count in (A0-A7) are merged into the A-register and private memory register Ru1
	(A0-A31)⟶(S0-S31)	Adder logic set at last PH6 clock	
	(S0-S31) ⟶/ (A0-A31)	AXS = FUMMC PH7 + ...	
	(S0-S31)⟶(RW0-RW31)	RWXS/0-RWXS/3 = RW + ...	
		RW = Set at last PH6 clock	
	Enable signal (S/SXDP1)	(S/SXDP1) = FUMMC PH7 + ...	Preset adder for D + 1 ⟶ S in PH8
	Set flip-flop RW	S/RW = FUMMC PH7 + ...	Prepare to write new lock image address in private memory register R
		R/RW = ...	
	Enable clock T8L	T8EN = NT5EN NT11L N(SXADD/1 RW) N(RW REU) N(REU AXRR)	
		NT5EN = RW + ...	
PH8	One clock long		
T8L	(D0-D31) + 1⟶(S0-S31)	Adder logic set at PH7 clock	Increment lock image address to point to next control image word
	(S0-S31)⟶(D0-D31)	DXS = FUMMC PH8 + ...	Transfer updated lock image address to D-register, P-register, and private memory register R
	(S0-S31)⟶(P15-P31)	PXS = FUMMC PH8 + ...	
	(S0-S31)⟶(RW0-RW31)	RWXS/0-RWXS/3 = RW + ...	
		RW = Set at PH7 clock	
	Enable clock T8L	T8EN = NT5EN NT11L N(SXADD/1 RW) N(RW REU) N(REU AXRR)	
		NT5EN = RW + ...	

Mnemonic: MMC (6F, EF)

(Continued)

Table 3-85. Move to Memory Control Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH8 T8L (Cont.)	If macro-counter count does not equal zero and no interrupt or I/O service call is pending, perform the following functions:		Count ≠ 0 indicates that more words are to be loaded
	Branch to PH2	BRPH2 = FUMMC PH8 N(INT + IOSC) NMCZ	Branch to PH2 to load next image word
	Enable signal (S/SXA)	(S/SXA) = FUMMC BRPH2 + ...	Preset adder for A⟶S in PH2
	Set flip-flop MRQ	S/MRQ = (S/MRQ/2) + ... (S/MRQ/2) = FUMMC BRPH2 + ... R/MRQ = ...	Core memory request for next lock image word
	Set flip-flop DRQ	S/DRQ = (S/MRQ/2) + ... R/DRQ = ...	Inhibits transmission of another clock until data release signal from core memory
	If interrupt is present or I/O service call is pending, or if count in macro-counter is zero, branch to PH9	S/PH9 = PH8 NBR + ... R/PH9 = ...	Interrupt or I/O service call will be processed at PH10
PH9	One clock long		
T5L	(B0-B31)⟶(S0-S31) (S15-S31)⟶(P15-P31)	SXB = PXSXB NDIS + ... PXSXB = NFAFL NFAMDS PH9 PXS = PXSXB + ...	Transfer program address to P-register from temporary storage in B-register
	Set flip-flop BRP	S/BRP = PXSXB + ... R/BRP = PRE1 NFAIM + ...	Indicates that program address is in P-register
	Set flip-flop MRQ	S/MRQ = (S/MRQ/2) + ... (S/MRQ/2) = PXSXB NINTRAP2 + ... R/MRQ = ...	Core memory request for next instruction in sequence
	Set flip-flop DRQ	S/DRQ = (S/MRQ/2) + ... R/DRQ = ...	Inhibits transmission of another clock until data release from core memory

Mnemonic: MMC (6F, EF)

(Continued)

Table 3-85. Move to Memory Control Sequence (Cont.)

Phase	Function Performed	Signals	Comments
PH10	Sustained until data release		
DR	Decrement program address in P-register to point to this instruction if interrupt or I/O service call is processed and not all image words were loaded	PDC31 = FUMMC PH10 NMCZ (INT + IOSC) ENDE	MMC instruction will be executed as many times as necessary to load all image words
	ENDE functions	See table 3-18	

Mnemonic: MMC (6F, EF)

3-80 Wait (WAIT; 2E, AE)

GENERAL. The WAIT instruction halts the sequential operation of the CPU until an interrupt activation occurs or until the computer operator manually moves the COMPUTE switch on the Processor Control Panel from RUN to IDLE and back to RUN or to STEP.

If an interrupt is activated while the CPU is halted, the interrupt subroutine will be carried out; the instruction executed after completion of the interrupt routine will be the next instruction in sequence after the WAIT instruction.

If the COMPUTE switch is moved from RUN to IDLE and back to RUN or STEP while the CPU is halted, normal instruction execution will proceed with the next instruction in sequence after WAIT.

Wait Phase Sequence. Preparation phases for WAIT are the same as the general PREP phases for word instructions. Table 3-86 lists the detailed logic sequence during all WAIT execution phases. Execution of a WAIT instruction, however, also involves enabling the PCP phases to halt CPU operations. PCP phases are discussed in paragraph 3-56.

Table 3-86. Wait Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(P) : Program address		Address of next instruction in sequence
	Set flip-flop MRQ	S/MRQ = (S/MRQ/1) + ...	Memory request for next instruction in sequence. This instruction will be decoded and held in the CPU during ENDE and PCP phases. The PREP and execution phases of this instruction will not be entered until completion of PCP phases
		(S/MRQ/1) = FUWAIT PRE3 + ...	
		FUWAIT = OU2 OLE	
		R/MRQ = ...	
PH1	One clock long		
T5L	Set flip-flop HALT	S/HALT = FUWAIT PH1 + ...	Setting HALT enables branch to PCP phases
		R/HALT = NKAS PCP2 + ...	
	Branch to PH10	BRPH10 = FUWAIT PH1 + ...	
		S/PH10 = BRPH10 NCLEAR + ...	
		R/PH10 = ...	
	Set flip-flop DRQ	S/DRQ = BRPH10 NCLEAR + ...	Inhibits transmission of another clock until data release is received from core memory
		R/DRQ = ...	
			Mnemonic: WAIT (2E, BE)

(Continued)

Table 3-86. Wait Sequence (Cont.)

Phase	Function Performed	Signals Involved			Comments
PH10 DR	One clock long				
	Enable signal ENDE	ENDE	=	PH10 EXC	
	(MB0-MB31)⟶(C0-C31)	CXMB	=	DG = /DG/	Next instruction ⟶ C-register
	(C0-C31)⟶(D0-D31)	DXC	=	PH10 + ...	Next instruction ⟶ D-register
	(C1-C7)⟶(O1-O7)	OXC	=	PH10 + ...	Opcode of instruction ⟶O-register
	(C8-C11)⟶(R28-R31)	RXC	=	PH10 + ...	R field of instruction ⟶R-register
	Inhibit incrementing program address	PUC31	=	N(FUEXU ENDE) PH10 NHALT NIOSC NINT NKAHOLD + ...	Preserve address of this instruction
	Enable signal (S/SXD)	(S/SXD)	=	PH10 + ...	Not used. Enabled again in PCP1
	Set flip-flop IOEN if I/O service call	S/IOEN	=	IOSC PH10 NIOINH + ...	I/O service call inhibits branch to PCP phases
	Interrupt enable	IEN	=	KRUN PH10 NIOSC	Interruptible point. Interrupt causes branch to INTRAP phases
	Set flip-flop LRXD	S/LRXD	=	OXC + ...	For index operations in PREP phases
		R/LRXD	=	...	
	Reset flip-flop NAXRR	S/NAXRR	=	N(S/AXRR)	For index operations in PREP phases
		(S/AXRR)	=	PH10 + ...	
		R/NAXRR	=	...	
	Inhibit branch to PRE1	PRE1EN	=	N(S/TRAP) N(S/INTRAP) NIOSC NHALT	PRE1 is entered after PCP phases
	Clear and reset functions	CLEAR	=	PH10 + ...	
		RESET/A	=	CLEAR + ...	
	Branch to PCP1	BRPCP1	=	NFUEXU HALT/1 ENDE NIOSC + ...	
		S/PCP1	=	BRPCP1 + ...	
		R/PCP1	=	...	
					Mnemonic: WAIT (2E, BE)

3-81 Family of Direct Instructions (FARWD)

GENERAL. The two direct instructions, Read Direct and Write Direct, enable the computer to transmit and receive a full word of data at a time without the use of an input/output channel. A special set of address, data, and control lines are provided for this direct communication with other elements (analog-to-digital converters, digital counters, etc.) of the Sigma 5 system. The Read Direct instruction requests data from the other element, and the Write Direct instruction transmits data to the other element.

READ DIRECT (RD; 6C, EC). The RD instruction operates in one of two modes, depending on the state of bits 16 through 19 of the instruction word. If any of these bits contain a logical 1, the computer operates in the external mode, communicating directly with other system elements without the aid of an input/output unit. The signals are carried on the read direct/write direct (RD/WD) lines consisting of 16 address lines, 32 data lines, two condition code lines, and various control lines. If bits 16 through 19 contain zeros, the computer performs internal control operations.

External Mode. If bits 16 through 19 of the instruction word contain X'2' through X'F', the CPU presents bits 16 through 31 of the effective address to other elements of the Sigma 5 system on the RD/WD address lines. Bits 16 through 31 of the effective address identify a specific system element that is to return two condition code bits and a maximum of 32 data bits to the CPU. The significance and number of data bits depend on the selected element. If the R field of the instruction word is nonzero, the returned data is loaded into the private memory register specified by the R field. If the R field is zero, the returned data is ignored. Bits CC3 and CC4 of the condition code portion of the program status doubleword are set by the addressed element regardless of the value of the R field.

Internal Mode. If bits 16 through 19 of the instruction word contain zeros, the condition code is set according to the states of the four SENSE switches on the processor control panel. If a particular SENSE switch is set, the corresponding condition code bit is set to one; if a SENSE switch is reset, the corresponding condition code bit is reset to zero.

If the RD instruction specifies the internal mode and bit 27 contains a one, the states of the eight memory fault indicators, one for each core memory module, are read. A memory fault indicator is set when a parity error or overtemperature condition occurs in its corresponding module. If the R field of the instruction word is nonzero, bit positions 0 through 23 of the private memory register specified by the R field are reset to zeros, and bit positions 24 through 31 are set according to the current states of the memory fault indicators. Then the memory fault indicators are reset. If the R field is zero, the memory fault indicators and the contents of the private memory register specified by the R field remain unchanged. In either case, the condition code is set according to the states of the SENSE switches.

RD Phase Sequence. Preparation phases for RD are the same as the general PREP phases for word instructions, paragraph 3-59.

Figure 3-191 shows the simplified phase sequence for the RD instruction. Table 3-87 lists the detailed logic sequence during the execution phases.

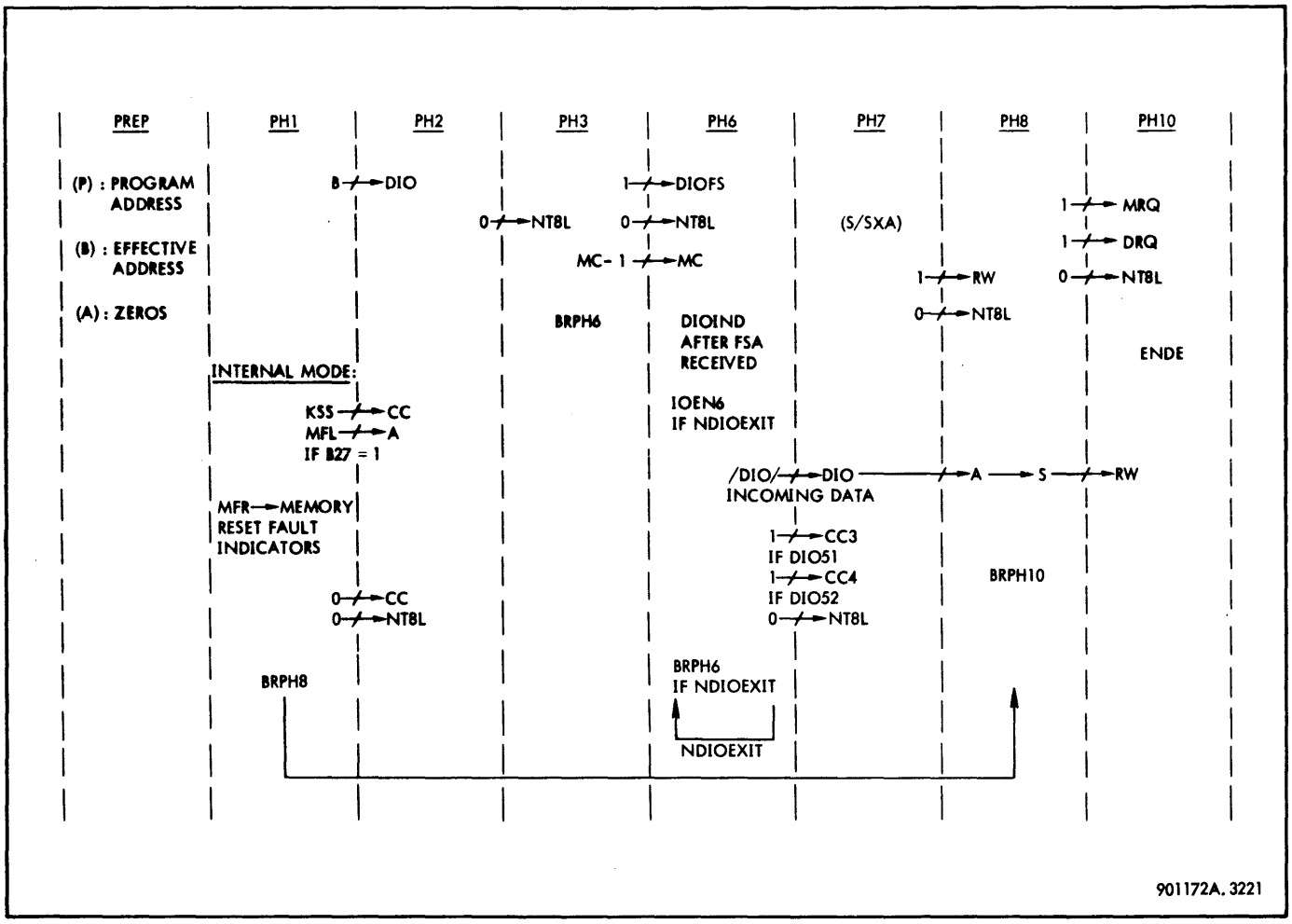


Figure 3-191. Read Direct Instruction, Phase Sequence Diagram

Table 3-87. Read Direct Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP: (P) : Program address (B) : Effective address (A) : Zeros	AXRRINH = FARWD OLC PRE3 + ... FARWD = OU6 (O4 O5) + ...	Enter zeros into A when AXRR is performed in PRE1
	Reset condition code flip-flops	R/CC = FARWD PRE3 + ...	Prepare to read SENSE switches or receive condition code from other element
			Mnemonic: RD (6C, EC)

(Continued)

Table 3-87. Read Direct Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH1	One clock long		
T5L	(B16-B31)—/→(DIO32/1-DIO47/1)	DIOXB = FARWD PH1	Present bits 16 through 31 of effective address to other system element on RD/WD lines
	Internal Mode (B16-B19 = 0):		
	(KSS1-KSS4)—/→(CC1-CC4)	CCXRWD = FARWD B1619Z PH1	Set condition code according to PCP SENSE switches
	(MFL0-MFL7)—/→(A24-A31)	AXPARITY = RDXMFI RDXMFI = OLC CCXRWD B27/1	Set A24 through A31 according to memory fault indicators if B27 = 1
	Enable signal (S/SXA) if R ≠ 0	(S/SXA) = FARWD PH1 NRZ + ...	Preset adder for A——→S in PH8
	Reset memory fault indicators if R ≠ 0	MFR = RDXMFI NRZ + ...	MFR set to memory via cable driver
	Reset flip-flop NT8L	S/NT8L = N(S/T8L) (S/T8L) = FARWD NPREP R/NT8L = ...	Set clock T8L for PH2 or PH8
	Branch to PH8	BRPH8 = FARWD B1619Z PH1 S/PH8 = BRPH8 NCLEAR + ... R/PH8 = ...	
PH2	One clock long		
T8L	Reset flip-flop NT8L	S/NT8L = N(S/T8L) (S/T8L) = FARWD NPREP R/NT8L = ...	Set clock T8L for PH3
PH3	One clock long		
T8L	Reset flip-flop NDIOFS	R/NDIOFS = (S/DIOFS) (S/DIOFS) = FARWD PH3 /DIO48/ = DIOFS	Set function strobe, to be transmitted to other element via RD/WD control line DIO48
	MC-1—/→MC	MDC7 = FARWD PH3 + ...	Decrement macro-counter from 00000000 to 11111111 to make instruction interruptible
	Reset flip-flop NT8L	S/NT8L = N(S/T8L) (S/T8L) = FARWD NPREP + ... R/NT8L = ...	Set clock T8L for PH6
			Mnemonic: RD (6C, EC)

(Continued)

Table 3-87. Read Direct Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH3 T8L (Cont.)	Branch to PH6	BRPH6 = FARWD PH3 + ... S/PH6 = BRPH6 NIOEN NCLEAR + ... R/PH6 = ...	
PH6 T8L	One clock long Enable signal IOEN6 until DIOEXIT goes true	IOEN6 = FARWD PH6 NEWDM NDIOEXIT NMC0005Z	I/O service call enable
	Enable signal DIOIND when function strobe acknowledge is received	DIOIND = NDIOT2 DIOT3 S/DIOT3 = FSA + DIOIND R/DIOT3 = ... FSA = DIO49 + ...	FSA is function strobe acknowledge received from other system element
	(/DIO0/-/DIO31/)—/→ (DIO0/1-DIO31/1)	DIOXDIO = DIOIND	Receive 32 data bits from selected element via RD/WD lines
	Set flip-flops CC3 and CC4 according to DIO51 and DIO52	S/CC3 = (S/CC3/1) + ... (S/CC3/1) = DIO51 DIOIND + ... S/CC4 = (S/CC4/1) + ... (S/CC4/1) = DIO52 DIOIND + ...	Receive two condition code bits from selected element via RD/WD lines
	Reset flip-flop NT8L	S/NT8L = N(S/T8L) (S/T8L) = FARWD NPREP R/NT8L = ...	Set clock T8L for PH7
	Sustain PH6 until DIOEXIT	BRPH6 = FARWD PH6 NDIOEXIT DIOEXIT = DIOT2 NDIOT1 S/DIOT2 = NIOACT S/DIOT1 = DIOIND R/DIOT1 = NDIOT3 (See figure 3-192)	If no I/O action, DIOEXIT rises four clock times after FSA goes true. If I/O action takes place, NDIOEXIT is delayed until the I/O action is complete
PH7 T8L	One clock long (DIO0/1-DIO31/1)—/→(A0-A31)	Adder logic set at PH7 clock	Transfer incoming data bits to A-register
	Enable signal (S/SXA)	(S/SXA) = FARWD PH6 NDIOEXIT NRZ	Preset adder for A——→S in PH8
	Set flip-flop RW	S/RW = (S/RW) (S/RW) = FARWD PH7 OLC NRZ	Prepare to write into private memory
	Reset flip-flop NT8L	S/NT8L = N(S/T8L) (S/T8L) = FARWD NPREP R/NT8L = ...	Set clock T8L for PH8
			Mnemonic: RD (6C, EC)

(Continued)

Table 3-87. Read Direct Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
	(Entered from PH1 if internal mode)		
PH8	One clock long		
T8L	(A0-A31) ⟶ (S0-S31)	Adder logic set at PH7 clock	Write incoming data or memory fault indicator bits in private memory via sum bus
	(S0-S31) ⟶/⟶ (RW0-RW31)	RWXS = RW	
	Set flip-flop MRQ	S/MRQ = (S/MRQ/1) + ... (S/MRQ/1) = FARWD PH8 + ... R/MRQ = ...	Request for core memory cycle
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = BRPH10 + ... R/DRQ = ...	Data request, inhibiting transmission of another clock until data release received from memory
	Branch to PH10	BRPH10 = FARWD PH8 + ... S/PH10 = BRPH10 NCLEAR + ... R/PH10 = ...	
PH10 DR	Sustained until data release ENDE functions		
			Mnemonic: RD (6C, EC)

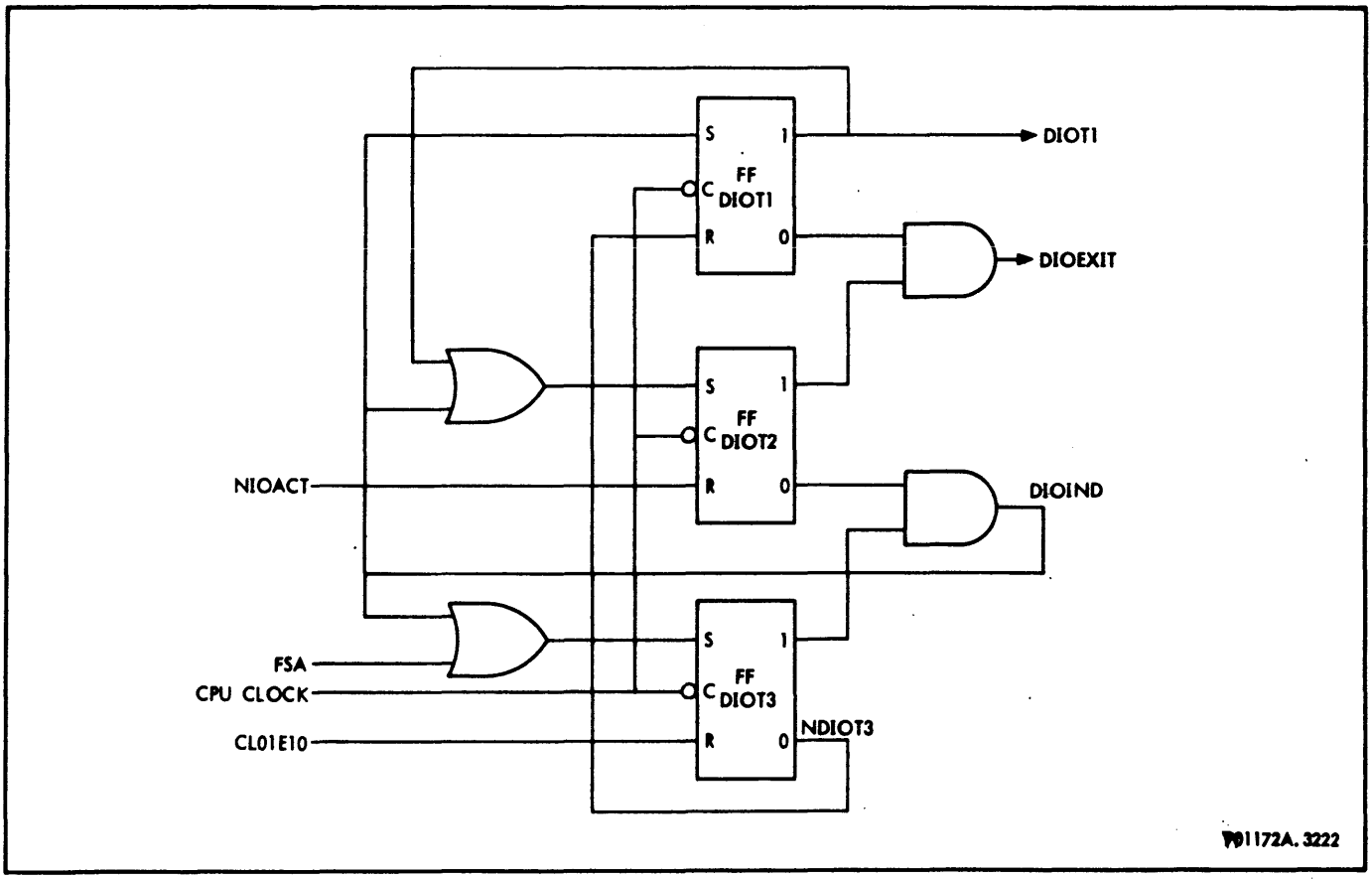


Figure 3-192. DIO Timing Flip-Flops, Simplified Logic Diagram

WRITE DIRECT (WD; 6D, ED). The WD instruction operates in one of three modes depending on the state of bits 16 through 19 of the instruction word. If this field contains X'3' through X'F', the computer operates in the external mode, communicating directly with other system elements without the aid of an input/output unit. The signals are carried on the read direct/write direct (RD/WD) lines consisting of 16 address lines, 32 data lines, two condition code lines, and various control lines. If bits 16 through 19 of the instruction word contain X'1', the interrupt control mode is entered to alter the various states of the individual interrupt levels in the CPU interrupt system. If bits 16 through 19 contain zeros, the computer performs internal control operations.

External Mode. In the external mode, the computer presents bits 16 through 31 of the effective address to other elements of the Sigma 5 system on the RD/WD address lines. These bits identify a specific element of the Sigma 5 system that is to receive control information from the CPU. If the R field of the WD instruction is nonzero, the 32-bit contents of the private memory register specified by the R field are transmitted to the specified element on the RD/WD data lines. If the R field is zero, 32 zeros are transmitted to the specified element. The specified element may return information to set bits 3 and 4 of the condition code.

Interrupt Mode. If bits 16 through 19 of the WD instruction contain 0001, the states of the interrupt levels in the CPU interrupt system are changed according to the states of bits 16 through 31 of the private memory register specified by the R field of the instruction. Bit position 16 of register R contains the selection bit for the highest priority (lowest numbered) interrupt level within the group, and bit position 31 of register R contains the selection bit for the lowest priority (highest numbered) interrupt level within the group. Bits 28 through 31 of the effective address specify the identification number of the group of interrupt levels to be controlled by the instruction. Bits 21 through 23 of the effective address contain a function code that specifies the type of control to be used.

Internal Mode. If bits 16 through 19 of the WD instruction contain zeros, the program is in the internal computer control mode. In this mode the condition code is set according to the states of the four SENSE switches on the processor control panel. If a particular SENSE switch is set, the corresponding condition code bit is set to a one; if a SENSE switch is reset, the corresponding condition code bit is reset to zero.

If bit positions 26 and 27 of the internal mode WD instruction contain ones, the interrupt inhibit bits in the program status doubleword (bits 37 through 39) are set, if in the zero state, according to bits 29 through 31 of the WD instruction. If any or all of bits 29 through 31 of the effective address are ones, the corresponding inhibit bits in the program status doubleword are set to ones. The current state of an inhibit bit is not affected if the corresponding bit position in the effective address contains a zero.

If bit position 26 of the internal mode instruction word contains a one and bit position 27 contains a zero (bit 25 not set) the interrupt inhibit bits of the program status doubleword are reset, if in the one state, according to bits 29 through 31 of the WD instruction. If any or all of bits 29 through 31 of the effective address are ones, the corresponding inhibit bits in the program status doubleword are reset to zero. The current state of an inhibit bit is not affected if a corresponding bit position of the effective address contains a zero.

When bit positions 25 and 31 of the internal mode WD instruction contain ones, the ALARM indicator on the maintenance section of the processor control panel is set. The ALARM indicator is reset with an internal mode WD instruction containing a one in bit position 25.

The CPU program-controlled-frequency flip-flop (MUSIC) is toggled with an internal mode WD instruction containing ones in bit positions 25 and 30. In response to the instruction, the flip-flop toggles.

An AUDIO signal is generated from the ALARM and MUSIC flip-flops and connected through the AUDIO switch on the processor control panel to the computer speaker. When flip-flop ALARM is set, a 1000-hz AUDIO signal is generated if the PCP COMPUTE switch is in the RUN position. The MUSIC flip-flop generates an AUDIO signal with a frequency determined by the rate at which the MUSIC flip-flop is toggled.

The integral IOP inhibit signal, set when the watchdog timer runout trap is activated, is reset by an internal mode WD instruction with ones in bit positions 25, 26, and 29 (or by manual control).

WD Phase Sequence. Preparation phases for WD are the same as the general PREP phases for word instructions, paragraph 3-59.

Figure 3-193 shows the simplified phase sequence for the WD instruction. Table 3-88 lists the detailed logic sequence during the execution phases.

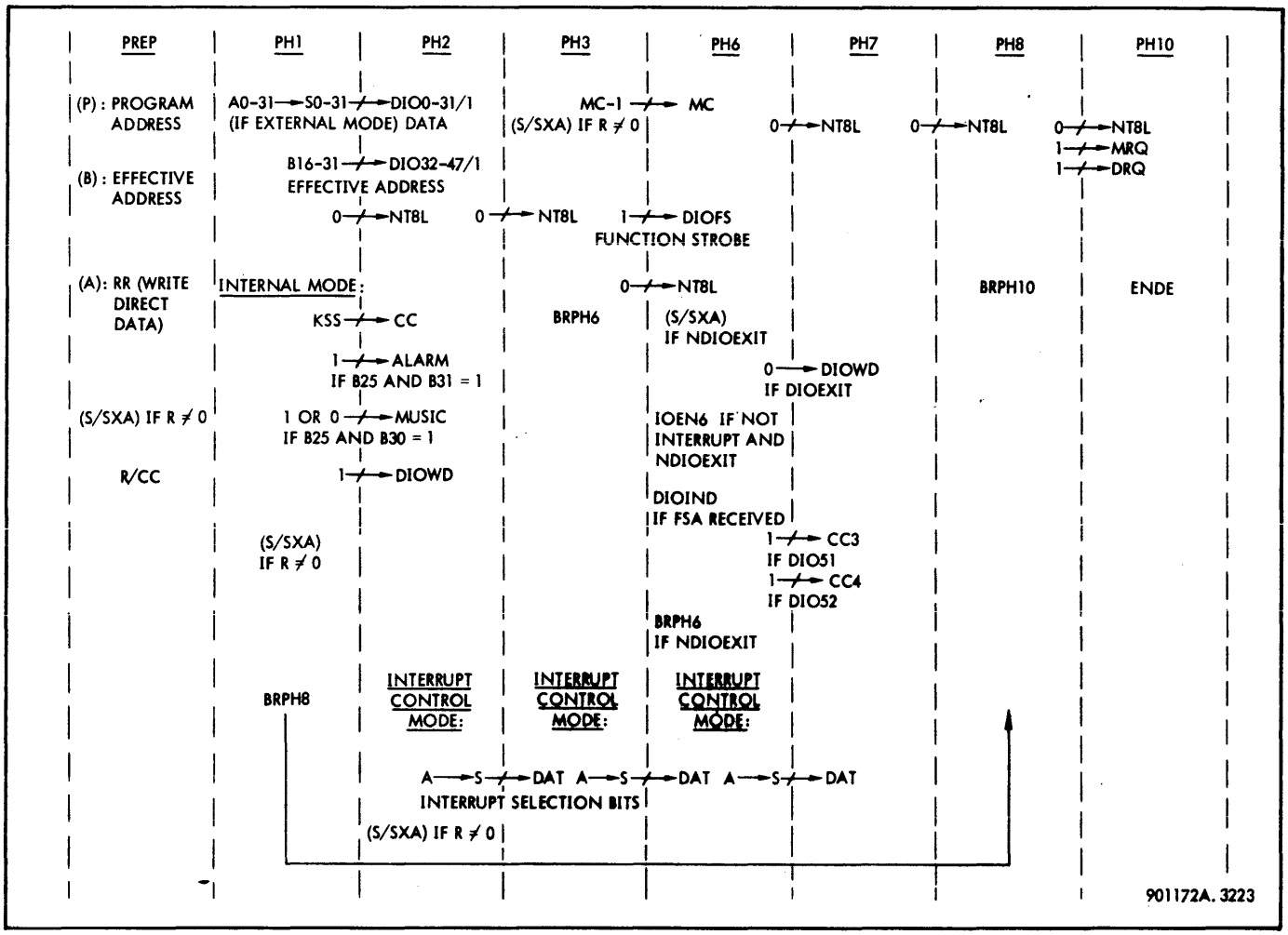


Figure 3-193. Write Direct Instruction, Phase Sequence Diagram

Table 3-88. Write Direct Sequence

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(P) : Program address		Address of next instruction
	(B) : Effective address		Mode and function
	(A) : Contents of private memory register R		Interrupt selection bits
	Enable signal (S/SXA) if R ≠ 0	(S/SXA) = FARWD (PRE/34 + PH2) NRZ + ...	Preset adder for A → S in PH1
		FARWD = OU6 (O4 O5) + ...	
	Reset condition code flip-flops	R/CC = FARWD PRE3 + ...	Prepare to read SENSE switches or receive code from other element
			Mnemonic: WD (6D, ED)

(Continued)

Table 3-88. Write Direct Sequence (Cont.)

Phase	Function Performed	Signals Involved			Comments
PH1 T5L	One clock long (A0-A31)——►(S0-S31)	Adder logic set at last PREP clock			Present contents of private memory register R or zeros to other element via bits 0 through 31 of RD/WD lines
	(S0-S31)—/—►(DIO0/1-DIO31/1)	DIOXS	=	FARWD PH1 NB1619Z	
	(B16-B31)—/—►(DIO32/1-DIO47/1)	DIOXB	=	FARWD PH1	Present effective address to other element or interrupt system via bits 32 through 47 of RD/WD lines
	Reset flip-flop NT8L	S/NT8L	=	N(S/T8L)	Set clock T8L for PH2
		(S/T8L)	=	FARWD NPREP + ...	
		R/NT8L			
	Internal mode (B16-B19 = 0): $\overline{(KSS1\text{-}KSS4)}$—/—►(CC1-CC4)	CCXRWD	=	FARWD B1619Z PH1	Set condition code according to PCP SENSE switches
	Set counter, input/output, and external interrupt group inhibit flip-flops in program status doubleword	S/CIF	=	(S/CIF/1) + ...	Set interrupt group inhibit specified by B29 through B31 if B27 = 1. Reset specified inhibit if B27 = 0. B26 specifies setting interrupt inhibits or resetting IOP inhibit
		(S/CIF/1)	=	INHXWD B29 B27	
		R/CIF	=	(R/CIF)	
		(R/CIF)	=	INHXWD B29 + ...	
		S/II	=	(S/II) + ...	
		(S/II)	=	INHXWD B30 B27	
		R/II	=	(R/II)	
		(R/II)	=	INHXWD B30 + ...	
		S/EI	=	(S/EI)	
		(S/EI)	=	INHXWD B31 B27	
		R/EI	=	(R/EI)	
		(R/EI)	=	INHXWD B31 + ...	
		INHXWD	=	CCXRWD OLD B26	
	Set or reset ALARM indicator on processor control panel	S/ALARM	=	WDINT B31	Alarm also resets when PCF is toggled (B25 true)
		WDINT	=	CCXRWD OLD B25	
		R/ALARM	=	WDINT + RESET	
	Toggle program-controlled-frequency flip-flop (PCF)	S/MUSIC	=	WDINT B30 NMUSIC	1000-hz audio signal transmitted to speaker when ALARM is set and COMPUTE switch is in RUN position. PCF output also transmitted to speaker as AUDIO signal
		R/MUSIC	=	WDINT B30	
	Transmit 1-hz alarm or music to speaker through AUDIO switch	AUDIO	=	MUSIC NALARM + ALARM KRUN 1KC	
	Reset direct input/output write flip-flop NDIOWD	(R/NDIOWD)	=	(S/DIOWD)	Prepare to transmit information to other system element in interrupt mode
		(S/DIOWD)	=	FARWD PH1 OLD	
	Enable signal (S/SXA)	(S/SXA)	=	FARWD (PH1 + PH3) NRZ + ...	Preset adder for A——►S in PH2
					Mnemonic: WD (6D, ED)

(Continued)

Table 3-88. Write Direct Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH1	Branch to PH8 if internal mode	BRPH8 = FARWD B1619Z PH1 + ...	
T5L (Cont.)		S/PH8 = BRPH8 NCLEAR + ...	
		R/PH8 = ...	
	Set flip-flop NIOWD	S/NIOWD = WDINT B29 + ...	Reset integral IOP inhibit if B25, B26, and B29 are true
	Interrupt control mode (B19 = 1):		
PH2	One clock long		
T8L	(A0-A31)⟶(S0-S31)	Adder preset at PH1 clock	Transfer contents of private memory register R to interrupt system via DAT lines. Bit 19 in instruction word specifies interrupt control mode
	(S16-S31)⟶(DAT16-DAT31)	DAT_n = S_n EWDM	
		EWDM = NB16 NB17 NB18 B19 DIOWD	
	Enable signal (S/SXA)	(S/SXA) = FARWD (PRE/34 + PH2) NRZ + ...	Preset adder for A⟶S in PH3
	Reset flip-flop NT8L	S/NT8L = N(S/T8L)	Set clock T8L for PH3
		(S/T8L) = FARWD NPREP + ...	
		R/NT8L = ...	
PH3	One clock long		
T8L	(A0-A31)⟶(S0-S31)	Adder preset at PH2 clock	Transfer contents of R-register to interrupt system if interrupt control mode
	(S16-S31)⟶(DAT16-DAT31)	DAT_n = S_n EWDM	
	Enable signal (S/SXA)	(S/SXA) = FARWD (PH1 + PH3) NRZ + ...	
	MC-1⟶MC	MDC7 = FARWD PH3 + ...	Decrement macro-counter from 00000000 to 11111111 to make instruction interruptible
	Reset flip-flop NDIOFS	R/NDIOFS = (S/DIOFS)	Transmit function strobe on DIO48
		(S/DIOFS) = FARWD PH3	
		/DIO48/ = DIOFS	
	Reset flip-flop NT8L	S/NT8L = N(S/T8L)	Set clock T8L for PH6
		(S/T8L) = FARWD NPREP + ...	
		R/NT8L = ...	
	Branch to PH6	BRPH6 = FARWD PH3 + ...	
		S/PH6 = BRPH6 NIOEN NCLEAR + ...	
		R/PH6 = ...	
			Mnemonic: WD (6D, ED)

(Continued)

Table 3-88. Write Direct Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH6	One clock long		
T8L	(A0-A31)⟶(S0-S31)	Adder logic set at PH3 clock	Transfer contents of R-register to interrupt system
	(S16-S31)⟶(DAT16-DAT31)	DAT_n = S_n EWDM	
	Enable signal (S/SXA)	(S/SXA) = FARWD PH6 NDIOEXIT NRZ + ...	Continue to present R-register contents to interrupt system until DIOEXIT
	Enable signal IOEN6	IOEN6 = FARWD PH6 NEWDM NDIOEXIT NMC0005Z	I/O service call enable if not interrupt control mode
	Set flip-flop NDIOWD	S/NDIOWD = DIOEXIT	Clear direct I/O write flip-flop
	Enable signal DIOIND when function strobe acknowledge is received	DIOIND = NDIOT2 DIOT3 S/DIOT3 = FSA + DIOIND (See figure 3-193) R/DIOT3 = ... FSA = DIO49	FSA is function strobe acknowledge received from other system element
	Set flip-flops CC3 and CC4 according to DIO51 and DIO52	S/CC3 = (S/CC3/1) (S/CC3/1) = DIO51 DIOIND + ... S/CC4 = (S/CC4/1) (S/CC4/1) = DIO52 DIOIND + ...	Receive two condition code bits from selected element via RD/WD lines
	Reset flip-flop NT8L	S/NT8L = N(S/T8L) (S/T8L) = FARWD NPREP + ... R/NT8L = ...	Set clock T8L for PH7
	Sustain PH6 until DIOEXIT	BRPH6 = FARWD PH6 NDIOEXIT DIOEXIT = DIOT2 NDIOT1 S/DIOT2 = NIOACT S/DIOT1 = DIOIND R/DIOT1 = NDIOT3 (See figure 3-193)	If not I/O action, DIOEXIT rises four clock times after FSA goes true. If I/O action takes place, NDIOEXIT is delayed until the action is completed
PH7	One clock long		
T8L	Reset flip-flop NT8L	S/NT8L = N(S/T8L) (S/T8L) = FARWD NPREP + .. R/NT8L = ...	Set clock T8L for PH8
			Mnemonic: WD (6D, ED)

(Continued)

Table 3-88. Write Direct Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH8	One clock long	S/MRQ = (S/MRQ/1) + ...	
T8L	Set flip-flop MRQ	(S/MRQ/1) = FARWD PH8 + ...	Request for core memory cycle
		R/MRQ = ...	
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR	Data request, inhibiting transmission of another clock until data release received from memory
		(S/DRQ) = BRPH10	
		R/DRQ = ...	
	Branch to PH10	BRPH10 = FARWD PH8 + ...	
		S/PH10 = NCLEAR BRPH10 + ...	
		R/PH10 = ...	
PH10	Sustained until data release		
DR	ENDE functions		
			Mnemonic: WD (6D, ED)

3-82 Family of Input/Output Instructions (FAIO)

GENERAL. The Sigma 5 CPU uses this family of instructions to communicate with standard peripheral devices such as line printers, card readers, and tape punches. If execution of an input/output instruction is attempted while the computer is in the slave mode (bit 8 of the current program status doubleword is a one), the computer aborts the instruction unconditionally and traps to location X'40'. Indirect addressing and indexing are performed in the same way as for the other instructions. With the exception of the AIO instruction, the 11 low-order bits of the effective address constitute the I/O address. For the AIO instruction, the device that initiated the interrupt call returns its 11-bit I/O address as part of the status response. Following is a list of instructions that comprise the FAIO:

SIO - Start Input/Output
TIO - Test Input/Output
TDV - Test Device
HIO - Halt Input/Output
AIO - Acknowledge Input/Output Interrupt

SIO INSTRUCTION. This instruction is used to initiate an input/output operation in the addressed device. In response, and based on the contents of the R field, the addressed IOP returns zero, one, or two words of status and condition codes CC1 and CC2. Also, the addressed IOP examines contents of private memory register 0 for address of first command doubleword in core memory. Figure 3-194 shows the structure of the instruction word, status format, distribution of data in the applicable registers, command doubleword format, and the significance of the condition codes.

HIO INSTRUCTION. This instruction is used to halt an input/output operation in the addressed device. If the device is in the interrupt pending condition, the condition is cleared. Information shown in figure 3-194 also applies to the HIO instruction, with the following exceptions:

a. The contents of private memory register 0 are not examined.

b. There is no command doubleword associated with an HIO instruction.

c. Condition codes are interpreted as follows:

1. CC1 CC2 = Address not recognized

2. NCC1 CC2 = Address recognized but device controller was busy at the time of the HIO instruction

d. FUSIO is false.

TIO INSTRUCTION. This instruction tests the current status of the addressed device, device controller, and IOP. No operation is initiated or terminated. Information shown in figure 3-194 also applies to the TIO instruction, with the following exceptions:

a. The contents of private memory register 0 are not examined.

b. There is no command doubleword associated with a TIO instruction.

c. Condition codes are interpreted as follows:

1. CC1 CC2 = Address not recognized

2. CC1 NCC2 = IOP busy

3. NCC1 CC2 = SIO cannot be accepted

d. FUSIO is false.

TDV INSTRUCTION. This instruction tests conditions in the addressed device not obtainable by means of a TIO instruction. Operation of the device, device controller, and IOP are not affected by this instruction. Information shown in figure 3-194 also applies to the TDV instruction, with the following exceptions:

a. The contents of private memory register 0 are not examined.

b. There is no command doubleword associated with a TDV instruction.

c. Condition codes are interpreted as follows:

1. CC1 CC2 = Address not recognized

2. CC1 NCC2 = IOP busy

3. NCC1 CC2 = Device-dependent condition exists

d. FUSIO is false.

AIO INSTRUCTION. This instruction is executed in response to an interrupt call issued by any device controller and is used to determine which device controller raised the interrupt call and for what purpose. In response, the highest priority device controller with an interrupt pending returns its address and condition codes CC1 and CC2, which specify the type of interrupt. Figure 3-195 illustrates the structure of the instruction word, the status format, and the significance of condition codes CC1 and CC2.

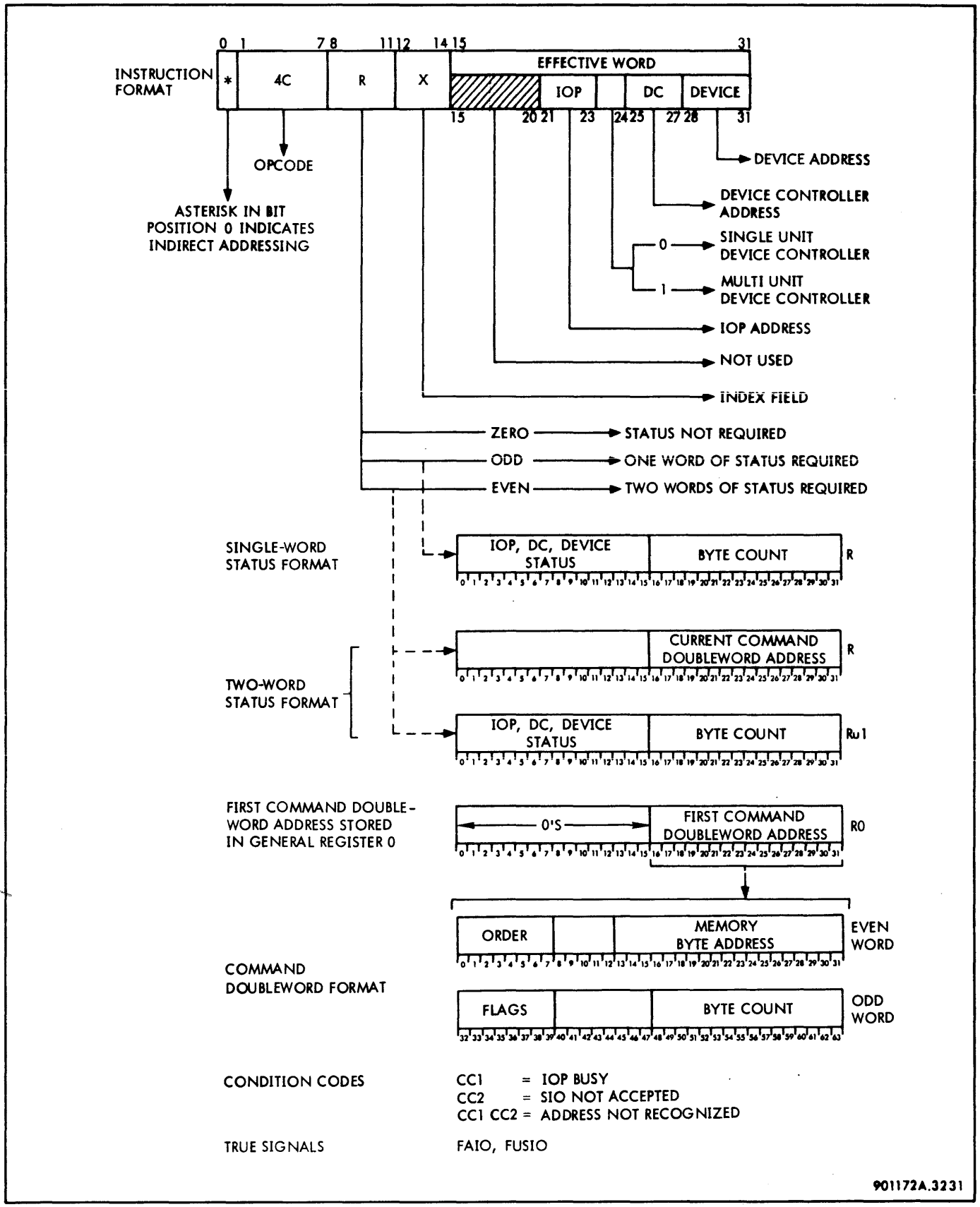


Figure 3-194. Start Input/Output Instruction Format

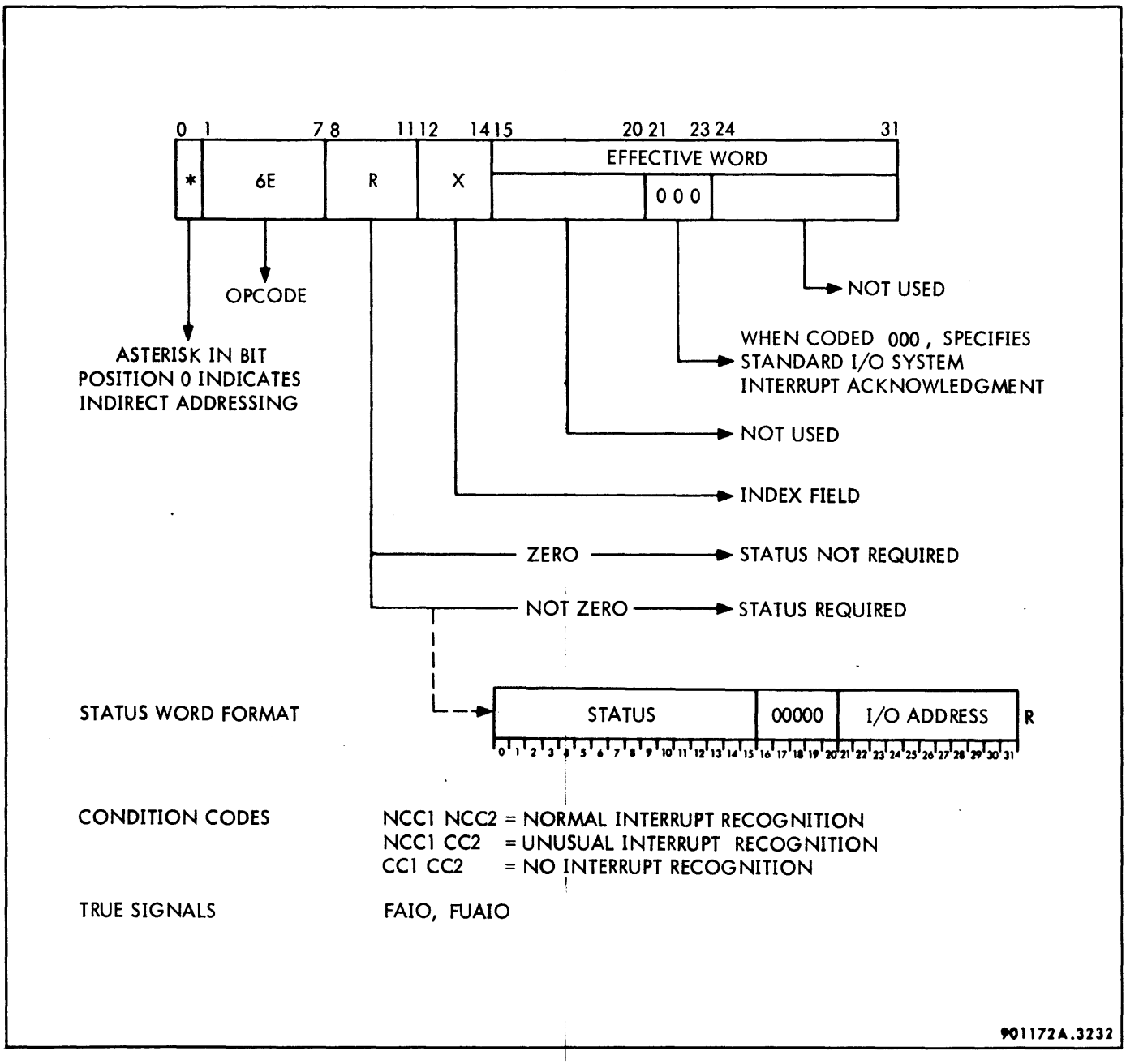


Figure 3-195. Acknowledge Input/Output Interrupt Instruction Format

I/O PHASE SEQUENCE CHARTS. Preparation phases for the I/O instructions are the same as the general PREP phases described in paragraph 3-59. The execution phases of the I/O instructions are described in four phase sequence charts and illustrated in associated flow diagrams as follows:

Instruction	Type of IOP	Sequence Chart	Flow Diagram
SIO, HIO TIO, TDV	MIOP	Table 3-89	Figure 3-196
SIO, HIO, TIO, TDV	Integral	Table 3-90	Figure 3-197
AIO	MIOP	Table 3-91	Figure 3-198
AIO	Integral	Table 3-92	Figure 3-199

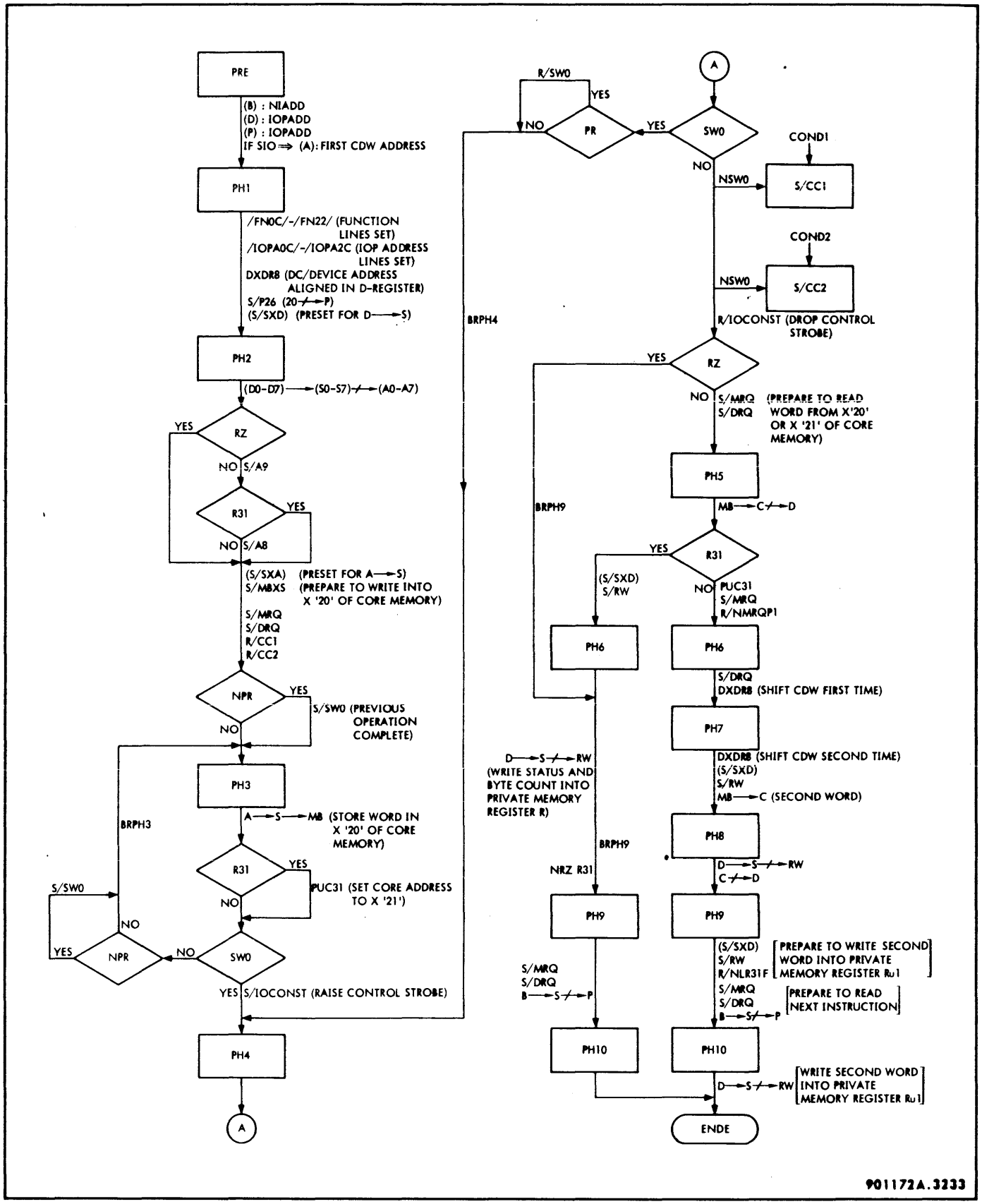


Figure 3-196. SIO, HIO, TIO, TDV, Flow Diagram for MIOP

Vol 8 ms 5

Table 3-89. SIO, TIO, TDV, HIO Sequence for MIOP

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP: (B): Program address (D): IOPADD (P): IOPADD If SIO, (A): RR		 New instruction address IOP, device controller, device address First command doubleword (CDW) address
PH1 T5L	One clock long Opcode transferred from O-register to function lines (O2, 6, 7) ⟶ (FNC0C, 1C, 2C) IOP address transferred from the P-register to the address lines: (P21, 22, 23) ⟶ (SW5, 6, 3) (D24-D31) ⟶/ (D0-D7)	SIO TIO TDV HIO /FNC0C/ 0 0 0 0 /FNC1C/ 0 0 1 1 /FNC2C/ 0 1 0 1 S/SW5 = (S/SW5) + ... (S/SW5) = FAIO PH1 P21 + ... FAIO = OU4 (O4 O5) R/SW5 = (R/SW5) S/SW6 = (S/SW6) + ... (S/SW6) = FAIO PH1 P22 R/SW6 = RESET/A S/SW3 = (S/SW3) + ... (S/SW3) = FAIO PH1 P23 R/SW3 = RESET/A DXDR8 = (FAIO PH1) + ...	 Specify the type of instruction to be executed Specify one of eight IOP's Align device controller/device address by means of a right circular shift. Bits 0 through 7 of the D-register will be transferred to the A-register during PH2
			Mnemonic: SIO (4C, CC) TIO (4D, CD) TDV (4E, CE) HIO (4F, CF)

(Continued)

Not our E

Table 3-89. SIO, TIO, TDV, HIO Sequence for MIOP (Cont.)

Phase	Function Performed	Signals Involved			Comments
PH1 T5L (Cont.)	20 ⟶ P	S/P26	=	(S/P26) + ...	Preset P-register to X'20' by forcing a 1 into bit 26 and resetting the other 16 bits. During PH2 a word is transmitted to the addressed MIOP via location X'20' in core memory
		(S/P26)	=	(FAIO PH1) + ...	
		R/P26	=	PX + ...	
		PX	=	FAIO PH1 + ...	
	Enable signal (S/SXD)	(S/SXD)	=	(FAIO PH1) + ...	Preset adder for D ⟶ S in PH2
PH2 T5L	One clock long				
	(D0-D7) ⟶ (S0-S7)	Adder logic set at PH1 clock			
	(S0-S7) ⟶ (A0-A7)	AXS/0	=	AXS/4 + ...	Transfer device controller/device address to the A-register
		AXS/4	=	AXS/2 ...	
		AXS/2	=	(FAIO PH2) + ...	
	If R field is not zero (NRZ), set flip-flop A9	S/A9	=	(S/A9) IOAXST + ...	Generate R portion of word to be stored in location X'20' of core memory
		(S/A9)	=	(FAIO PH2) NRZ + ...	
		IOAXST	=	(FAIO PH2) + ...	
		R/A9	=	AX/1	
	If R field is not zero and even (NR31), set flip-flop A8	S/A8	=	(S/A8) IOAXST + ...	
		(S/A8)	=	(FAIO PH2) NR31 NRZ + ...	
		R/A8	=	AX/1	
	Enable signal (S/SXA)	(S/SXA)	=	(FAIO PH2) + ...	Preset adder logic for A ⟶ S in PH3
	Set flip-flop MBXS	S/MBXS	=	(S/MBXS)	Prepare to transfer contents of A-register to core memory
		(S/MBXS)	=	(FAIO/1 PH2) (NIOPADD + ...)	
		NIOPADD	=	SW5 + SW6 + SW3	Indicates multiplexing IOP
		R/MBXS	=	...	
	Set flip-flop MRQ	S/MRQ	=	(S/MRQ)	Memory request for transferring contents of A-register
		(S/MRQ)	=	(S/MBXS) + ...	
		R/MRQ	=	...	
	Set flip-flop DRQ	S/DRQ	=	(S/DRQ) NCLEAR-2	Inhibits transmission of another clock until data release is received from core memory
		(S/DRQ)	=	(S/MBXS) + ...	
		R/DRQ	=	...	
	Reset flip-flops CC1 and CC2	R/CC1	=	(R/CC1)	CC1 and/or CC2 are set in PH4 if specified by conditions in the addressed MIOP
		(R/CC1)	=	(R/CC1/1) + ...	
		(R/CC1/1)	=	(FAIO PH2) + ...	
					Mnemonic: SIO (4C, CC) TIO (4D, CD) TDV (4E, CE) HIO (4F, CF)

(Continued)

Table 3-89. SIO, TIO, TDV, HIO Sequence for MIOP (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH2 T5L (Cont.)		R/CC2 = (R/CC2) (R/CC2) = (R/CC2/1) + ... (R/CC2/1) = (FAIO PH2) + ...	
	If NPR, set flip-flop SW0	S/SW0 = (S/SW0) NCLEAR + ... (S/SW0) = (FAIO PH2) NPR + ... R/SW0 = (R/SW0)	Early detect of NPR, indicating that previous operation has been completed. If NSW0, NPR is checked again in PH3
PH3 DR or T5L	One or more clocks long, depending on the state of flip-flop SW0. First clock controlled by data release signal DR. Subsequent clocks, if any, are T5L		
	(A0-A31) ⟶ (S0-S31)	Adder logic set at PH2 clock	Transfer contents of A-register into location X'20' of core memory
	(S0-S31) ⟶ (MB0-MB31)	MBXS = Set at PH2 clock	
	If R field odd (R31), increment P-register	PUC31 = (FAIO PH3) R31 + ...	Set core memory address to X'21'. When R is odd, the addressed MIOP places a single word of status in location X'21' of the core memory
	If flip-flop SW0 was not set in PH2, set SW0 when PR goes low	S/SW0 = (S/SW0) NCLEAR-2 (S/SW0) = (FAIO PH3) NPR + ... R/SW0 = (R/SW0)	Wait for NPR from previous operation
	If NSW0, enable signal BRPH3	BRPH3 = FAIO PH3-B NSW0-1 + ...	Sustain PH3 until flip-flop SW0 gets set
	If SW0, set flip-flop IOCONST	S/IOCONST = (S/IOCONST) (S/IOCONST) = (FAIO PH3) SW0 R/IOCONST = (R/IOCONST) + ...	Raise control strobe before entering PH4
PH4 T5L or T8L	Two or more clocks, depending on the state of flip-flop SW0. First clock T5L. Subsequent clocks, if any, T5L, except for the last clock. Last clock T8L		
	If PR, reset flip-flop SW0	R/SW0 = (R/SW0) (R/SW0) = (FAIO PH4) PR + ...	Wait until addressed MIOP returns PR signal in response to the control strobe signal
			Mnemonic: SIO (4C, CC) TIO (4D, CD) TDV (4E, CE) HIO (4F, CF)

(Continued)

Table 3-89. SIO, TIO, TDV, HIO Sequence for MIOP (Cont.)

Phase	Function Performed	Signals Involved			Comments
PH4 T5L or T8L (Cont.)	If SW0, enable signal BRPH4	BRPH4	=	(FAIO PH4) SW0 NSW2	Sustain PH4 while flip-flop SW0 is in the set state
	If NSW0, and R field is zero (RZ), enable signal BRPH9	BRPH9	=	(FAIO PH4) NSW0 RZ + ...	If R is zero, status is not required
	If NSW0, and R field is not zero (NRZ), set flip-flops MRQ and DRQ	S/MRQ	=	(S/MRQ/2) + ...	Memory request for reading status word from X'20' (R even) or X'21' (R odd) of core memory
		(S/MRQ/2)	=	(FAIO PH4) (NRZ NSW0) + ...	
		R/MRQ	=	...	
		S/DRQ	=	(S/DRQ) NCLEAR	Inhibits transmission of another clock until data release signal is received from core memory
		(S/DRQ)	=	(S/MRQ/2) + ...	
		R/DRQ	=	...	
	If NSW0, set flip-flops CC1 and/or CC2 if specified	S/CC1	=	(FAIO PH4) NSW0 COND1 + ...	Setting of CC1 and CC2 is controlled by conditions in the addressed MIOP
		R/CC1	=	(R/CC1)	
		S/CC2	=	(FAIO PH4) NSW0 COND2	
		R/CC2	=	(R/CC2)	
	If NSW0, reset flip-flop IOCONST	R/IOCONST	=	(R/IOCONST) + ...	Drop control strobe in response to PR
		(R/IOCONST)	=	(FAIO PH5) NSW0 + ...	
PH5 NSW0 DR	One clock long				
	(MB0-MB31) ⟶ (C0-C31)	CXMB	=	DG = /DG/	Transfer word from location X'20' (R even) or X'21' (R odd) of core memory into the C-register and then to the D-register
	(C0-C31) ⟶/ (D0-D31)	DXC-0 thru DXC-3	=	DXC	
		DXC	=	(FAIO PH5) NSW0 + ...	
	If R even (NR31), increment P-register and set flip-flop MRQ and reset flip-flop NMRQP1	PUC31	=	(FAIO/1 PH5) NR31 NSW0 + ...	Prepare to read second of two status words out of core memory. Location of this word is X'21'
		S/MRQ	=	(S/MRQ)	
		(S/MRQ)	=	(S/MRQ/3) + ...	
		(S/MRQ/3)	=	(FAIO PH5) NSW0 NR31	
		R/MRQ	=	...	
		S/NMRQP1	=	N(SMRQ/3) + ...	Delays setting of flip-flop DRQ
		R/NMRQP1	=	...	

Mnemonic: SIO (4C, CC)
TIO (4D, CD)
TDV (4E, CE)
HIO (4F, CF)

(Continued)

Table 3-89. SIO, TIO, TDV, HIO Sequence for MIOP (Cont.)

Phase	Function Performed	Signals Involved			Comments
PH5 NSW0 DR (Cont.)	If R is odd (R31), enable signal (S/SXD) and set flip-flop RW	(S/SXD)	=	(FAIO PH5) NSW0 R31 + ...	Preset adder for D ⟶ S in PH6
		S/RW	=	(S/RW/1) + ...	Prepare to transfer status word into private memory register R
		(S/RW/1)	=	(S/RW) + ...	
		(S/RW)	=	(FAIO PH5) NSW0 R31	
		R/RW	=	...	
PH6 T5L or T8L	One clock long. Clock is T5L if R is even, T8L if R is odd				
	If R is even, set flip-flop DRQ and shift D-register 8 places to the right	S/DRQ	=	(S/DRQ) NCLEAR	Inhibits transmission of another clock until data release signal received from core memory
		(S/DRQ)	=	MRQP1	
		R/DRQ	=	...	
		DXDR8	=	(FAIO PH6) + ...	First step of CDW address alignment. Meaningful only if R is even
	If R is odd:				
	(D0-D31) ⟶ (S0-S31)	Adder logic set at PH5 clock			Transfer IOP status and byte count from D-register to private memory register R
	(S0-S31) ⟶/ (RW0-RW31)	RWXS/0-RWXS/3	=	RW	
	Enable signal BRPH9	BRPH9	=	(FAIO PH6) NSW0 R31 + ...	If R is odd, transfer of additional status information will not be performed
PH7 DR	One clock long				
	Enable signal DXDR8	DXDR8	=	(FAIO PH7) + ...	Second and final step of CDW address alignment
	(MB0-MB31) ⟶ (C0-C31)	CXMB	=	DG = /DG/	Transfer second status word from location X'21' of core memory to the C-register. During PH8 contents of C-register will be clocked into the D-register
					Mnemonic: SIO (4C, CC) TIO (4D, CD) TDV (4E, CE) HIO (4F, CF)

(Continued)

Table 3-89. SIO, TIO, TDV, HIO Sequence for MIOP (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH7 DR (Cont.)	Enable signal (S/SXD)	(S/SXD) = (FAIO PH7) + ...	Preset adder for D ⟶ S in PH8
	Set flip-flop RW	S/RW = (S/RW/1) + ... (S/RW/1) = (S/RW) + ... (S/RW) = (FAIO PH7) + ... R/RW = ...	Prepare to write CDW address into private memory register R
PH8 T8L	One clock long		
	(D0-D31) ⟶ (S0-S31)	Adder logic set at PH7 clock	Transfer CDW address from D-register to private memory register R
	(S0-S31) ⟶/⟶ (RW0-RW31) (R)	RWXS/0-RWXS/3 = RW	
	(C0-C31) ⟶/⟶ (D0-D31)	DXC-0 thru DXC-3 = DXC	Transfer second word of status (IOP status and byte count) into the D-register
		DXC = (FAIO PH8) + ...	
PH9 T8L	One clock long		
	If R field not zero (NRZ) and even (NR31), enable signal (S/SXD), set flip-flop RW, and reset flip-flop NLR31F	(S/SXD) = IOBR9 NSW0 + ...	Preset adder for D ⟶ S in PH10
		IOBR9 = FAIO/1 NR31 NRZ PH9	
		S/RW = (S/RW/1)	Prepare to write IOP status and byte count into private memory register Ru1
		(S/RW/1) = (S/RW) + ...	
		(S/RW) = IOBR9 NSW0 + ...	
		R/RW = ...	
		S/NLR31F = N(S/LR31)	Force a one on private memory address line LR31 during PH10 to select private memory register Ru1
		(S/LR31) = (FAIO PH9 NSW0) + ...	
		R/NLR31F = ...	
			Mnemonic: SIO (4C, CC) TIO (4D, CD) TDV (4E, CE) HIO (4F, CF)

(Continued)

Table 3-89. SIO, TIO, TDV, HIO Sequence for MIOP (Cont.)

Phase	Function Performed	Signals Involved			Comments
PH9 T8L (Cont.)	(B0-B31) ⟶ (S0-S31)	SXB	=	PXSXB NDIS	Transfer next instruction address to P-register
		PXSXB	=	NFAFL NFAMDS PH9	
	(S15-S31) ⟶/⟶ (P15-P31)	PXS	=	PXSXB + ...	
	Set flip-flops MRQ and DRQ	S/MRQ	=	(S/MRQ)	Prepare to read next instruction from core memory
		(S/MRQ)	=	(S/MRQ/2) + ...	
		(S/MRQ/2)	=	PXSXB NINTRAP2 + ...	
		R/MRQ	=	...	
		S/DRQ	=	(S/DRQ) NCLEAR	Inhibits transmission of another clock until data release is received from core memory
		(S/DRQ)	=	(S/MRQ/2) + (S/DRQ/2) + ...	
		(S/DRQ/2)	=	PH9 + ...	
		R/DRQ	=	...	
PH10 DR	One clock long				
	If R not zero (NRZ) and even (NR31):				
	(D0-D31) ⟶ (S0-S31)	(S/SXD)	=	PH10 + ...	Transfer IOP status and byte count from D-register to private memory register Ru1
	(S0-S31) ⟶/⟶ (RW0-RW31) (Ru1)	RWXS/0-RWXS/3	=	RW	
	ENDE functions				

Mnemonic: SIO (4C, CC)
TIO (4D, CD)
TDV (4E, CE)
HIO (4F, CF)

OUR IOP

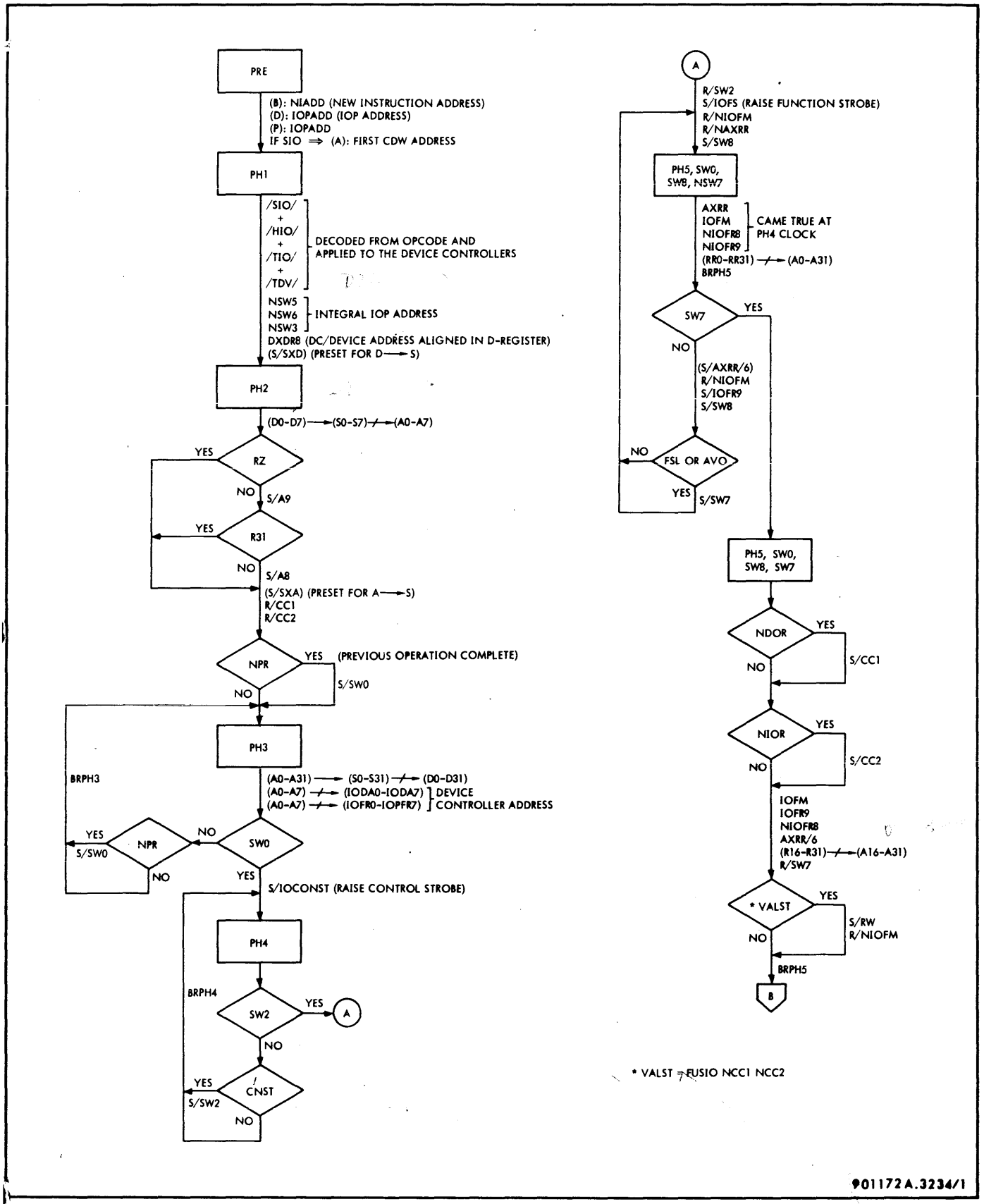


Figure 3-197. SIO, HIO, TIO, TDV Flow Diagram for Integral IOP (Sheet 1 of 2)

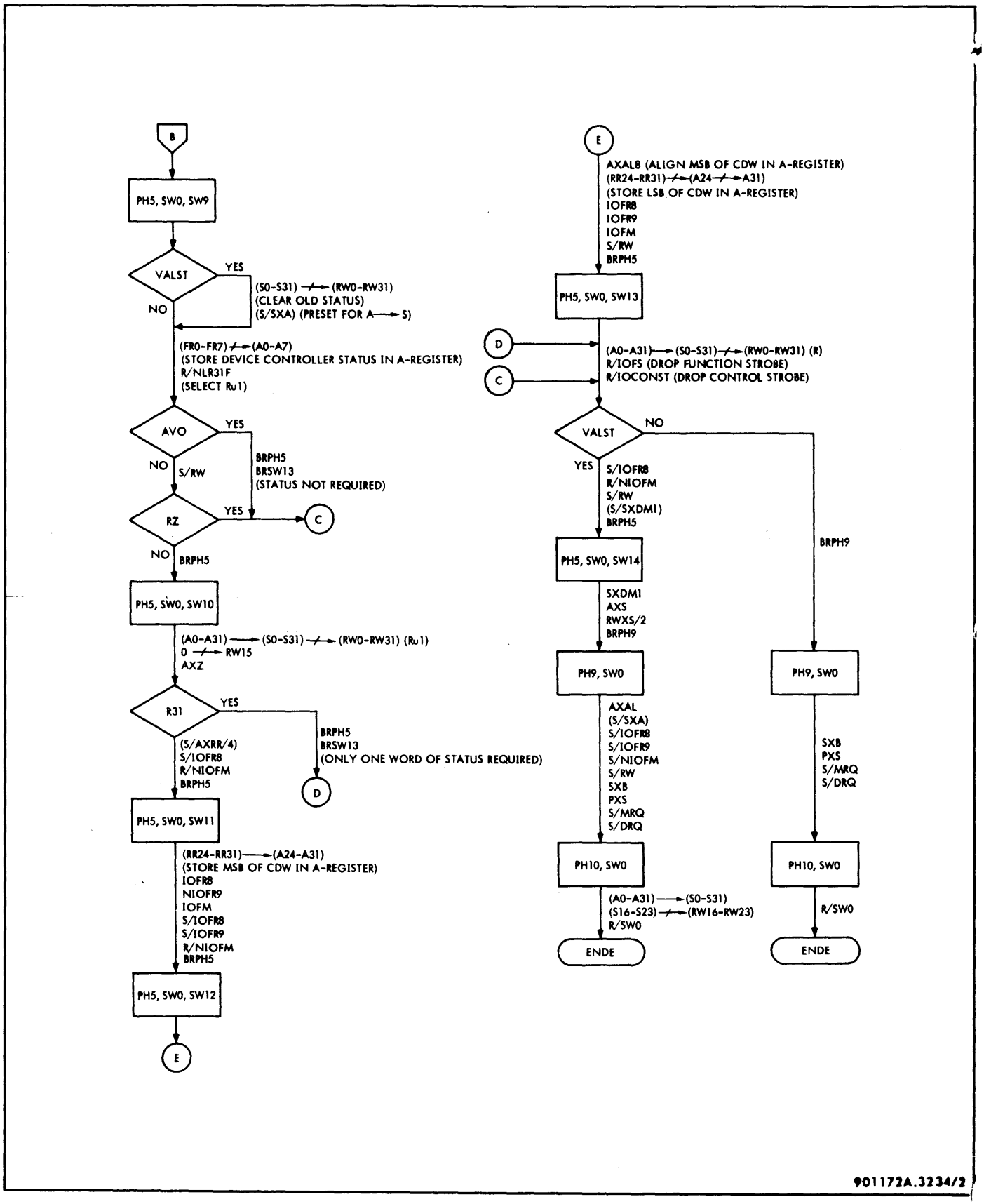


Figure 3-197. SIO, TIO, TDV, HIO Flow Diagram for Integral IOP (Sheet 2 of 2)

Table 3-90. SIO, TIO, TDV, HIO Sequence for Integral IOP

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP:		
	(B): Program address		New instruction address
	(D): IOPADD		IOP, device controller, device address
	(P): IOPADD		
	If SIO, (A) : RR		First command doubleword (CDW) address
PH1 T5L	One clock long		
	Opcode decoded from the contents of the O-register, causing the appropriate function indicator line to be raised	/SIO/ = FUSIO NIOCON NPH10	By means of the function indicator lines the integral IOP notifies the appropriate device controller of the type of function to be performed
		FUSIO = OU4 OLC	
		/HIO/ = O6 O7 NIOCON NPH10	
		/TDV/ = NO2 O6 NO7 NIOCON NPH10	
		/TIO/ = NO6 O7 NIOCON NPH10	
		IOCON = IOSC + IOIN	Service call pending
	(P21, 22, 23) —/→ (SW5, 6, 3)	S/SW5 = (S/SW5) + ...	Integral IOP is selected when SW5, SW6, and SW3 are false
		(S/SW5) = FAIO PH1 P21	
		R/SW5 = (R/SW5)	
		S/SW6 = (S/SW6) + ...	
		(S/SW6) = FAIO PH1 P22 + ...	
		R/SW6 = RESET/A	
		S/SW3 = (S/SW3) + ...	
		(S/SW3) = FAIO PH1 P23	
		R/SW3 = RESET/A	
	(D24-D31) —/→ (D0-D7)	DXDR8 = (FAIO PH1) + ...	Align device controller/device address by means of a right circular shift. Bits 0 through 7 of the D-register will be transferred to the A-register during PH2
	Enable signal (S/SXD)	(S/SXD) = (FAIO PH1) + ...	Preset adder for D → S in PH2
			Mnemonic: SIO (4C, CC) TIO (4D, CD) TDV (4E, CE) HIO (4F, CF)

(Continued)

Table 3-90. SIO, TIO, TDV, HIO Sequence for Integral IOP (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH2 T5L	One clock long		
	(D0-D7) ⟶ (S0-S7)	Adder logic set at PH1 clock	
	(S0-S7) ⟶/ (A0-A7)	AXS/0 = AXS/4 AXS/4 = AXS/2 AXS/2 = (FAIO PH2) + ...	Transfer device controller/device address to A-register
	If R field is not zero (NRZ), set flip-flop A9	S/A9 = (S/A9) IOAXST (S/A9) = (FAIO PH2) NRZ + ... IOAXST = (FAIO PH2) + ... R/A9 = AX/1	Generate R portion of word to be stored in D-register
	If R is not zero and even (NR31), set flip-flop A8	S/A8 = (S/A8) IOAXST (S/A8) = (FAIO PH2) NRZ NR31 R/A8 = AX/1	
	Enable signal (S/SXA)	(S/SXA) = (FAIO PH2) + ...	Preset adder for S ⟶ A in PH3
	Reset flip-flops CC1 and CC2	R/CC1 = (R/CC1) (R/CC1) = (R/CC1/1) + ... (R/CC1/1) = (FAIO PH2) + ... R/CC2 = (R/CC2) (R/CC2) = (R/CC2/1) + ... (R/CC2/1) = (FAIO PH2) + ...	Flip-flops CC1 and/or CC2 are set during PH5 SW8 SW7 if specified by conditions in the selected device controller
	If NPR, set flip-flop SW0	S/SW0 = (S/SW0) NCLEAR + ... (S/SW0) = (FAIO PH2) NPR + ... R/SW0 = (R/SW0)	Early detect of NPR, indicating that previous operation has been completed. If NSW0, NPR is checked again during PH3
PH3 T5L	One or more clocks long, depending on the state of flip-flop SW0		
	(A0-A31) ⟶ (S0-S31)	Adder logic set at PH2 clock	
	(S0-S31) ⟶/ (D0-D31)	DXS = FAIO PH3	
			Mnemonic: SIO (4C, CC) TIO (4D, CD) TDV (4E, CE) HIO (4F, CF)

(Continued)

Table 3-90. SIO, TIO, TDV, HIO Sequence for Integral IOP (Cont.)

Phase	Function Performed	Signals Involved			Comments
PH3 T5L (Cont.)	(A0-A7) —/→ (IODA0-IODA7)	IODAXA	=	(FAIO PH3) + ...	Transfer device controller/device address to the IODA-register. From here the information is transmitted on lines /DA0/ through /DA7/ to the device controllers associated with the integral IOP
	(A0-A7) —/→ (IOFR0-IOFR7)	IOFRXA	=	FAIO PH3	Transfer device controller/device address to the IOFR-register. Information stored in this register is used to select the appropriate IOFM-register
	If flip-flop SW0 was not set in PH2, set SW0 when PR goes low	S/SW0	=	(FAIO PH3) NPR + ...	Wait for NPR from previous operation
		R/SW0	=	(R/SW0)	
	If NSW0, enable signal BRPH3	BRPH3	=	FAIO PH3 NSW0 + ...	Sustain PH3 until flip-flop SW0 gets set
	If SW0, set flip-flop IOCONST	S/IOCONST	=	(S/IOCONST)	Raise control strobe before entering PH4. IOCONST will be reset at the end of PH5 SW13
		(S/IOCONST)	=	(FAIO PH3) SW0	
		R/IOCONST	=	(R/IOCONST) + ...	
PH4 T5L	Two or more clocks, depending on /CNST/				
	Set flip-flop SW2	S/SW2	=	(S/SW2)	Wait for /CNST/ to be returned through the IOP priority cable. /CNST/ is derived from IOCONST
		(S/SW2)	=	(FAIO PH4) IOPADD CNST + ...	
		CNST	=	/CNST/ NIOPOP (IOCONST + ...)	
		IOPADD	=	NSW5 NSW6 NSW3	Indicates integral IOP
	Enable signal BRPH4	BRPH4	=	(FAIO PH4) SW0 NSW2	Sustain PH4 until flip-flop SW2 has been set
	If flip-flop SW2 is set:				
	Reset flip-flop SW2	R/SW2	=	(R/SW2)	
		(R/SW2)	=	FAIO PH4 SW2 + ...	
					Mnemonic: SIO (4C, CC) TIO (4D, CD) TDV (4E, CE) HIO (4F, CF)

(Continued)

Table 3-90. SIO, TIO, TDV, HIO Sequence for Integral IOP (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH4 T5L (Cont.)	Set flip-flop IOFS	S/IOFS = (S/IOFS) (S/IOFS) = FAIO PH4 SW2 + ... R/IOFS = (R/IOFS)	Raise function strobe to device controllers Will be reset during PH5 SW13
	Reset flip-flops NIOFM and NAXRR	S/NAXRR = N(S/AXRR) (S/AXRR) = (S/AXRR/2) + ... (S/AXRR/2) = (FAIO/1 PH4) SW2 + ... R/NAXRR = ...	Prepare to read byte address and IOP status from IOP fast memory, area 00. Byte address will only be stored temporarily and then replaced by the byte count
		S/NIOFM = N(S/IOFM) (S/IOFM) = (S/AXRR/2) + ... R/NIOFM = ...	Select IOP fast memory registers
	Set flip-flop SW8	S/SW8 = NRESET/A BRSW8+... BRSW8 = (FAIO PH4) SW2 + ... R/SW8 = ...	Used to define the first two subphases in PH5
PH5 SW0 SW8 NSW7 T5L	One or more clocks, depending on the state of flip-flop SW7		
	Enable signal BRPH5	BRPH5 = (FAIO PH5 SW0) NSW14 (VALST + NSW13) VALST = FUSIO NCC1 NCC2	Sustain PH5 during integral IOP sequence through subphase SW13 if not SIO, and through SW14 if SIO and valid start. Valid start occurs if during an SIO the addressed device controller returns NCC1 and NCC2, i.e., CC1 and CC2 will remain reset
	Maintain flip-flop SW8 in set state	S/SW8 = NRESET/A BRSW8+... BRSW8 = FAIO PH5 SW8 NSW7 + ... R/SW8 = ...	Sustain subphase SW8 while flip-flop SW7 is in reset state
			Mnemonic: SIO (4C, CC) TIO (4D, CD) TDV (4E, CE) HIO (4F, CF)

(Continued)

Table 3-90. SIO, TIO, TDV, HIO Sequence for Integral IOP (Cont.)

Phase	Function Performed	Signals Involved			Comments
PH5 SW0 SW8 NSW7 T5L (Cont.)	Set flip-flop SW7	S/SW7	=	(S/SW7)	Wait for either FSL or AVO response from device controller system. FSL signifies that one of the device controllers recognized the address; AVO signifies that the addressed device controller is not present in system
		(S/SW7)	=	FAIO PH5 SW8 NSW7 (FSL + AVO)+...	
		R/SW7	=	(R/SW7)	
	(RR0-RR31) —/→ (A0-A31)	AXRR	=	Preset at PH4 clock	Load word from I/O fast memory register, area 00 into A-register
		IOFM	=	Preset at PH4 clock	
		NIOFR8	=	Reset during previous operation	
		NIOFR9	=	Reset during previous operation	
	Maintain flip-flop NIOFM in the reset state and set flip-flop IOFR9	S/NIOFM	=	N(S/IOFM)	Prepare to read byte count from IOP fast memory register, area 01
		(S/IOFM)	=	(S/AXRR/6)'+...	
		(S/AXRR/6)	=	(FAIO PH5) SW8 NSW7 NFUMH+...	
		R/NIOFM	=	...	
		S/IOFR9	=	(S/IOFR9 IOPOP)	
		(S/IOFR9)	=	(S/AXRR/6) +...	
		R/IOFR9	=	...	
PH5 SW0 SW8 SW7 T5L	One clock long Set flip-flops CC1 and CC2, if specified	S/CC1	=	FAIO PH5 SW8 SW7 NDOR +...	Setting of CC1 and CC2 is controlled by conditions in the addressed device controller
		R/CC1	=	(R/CC1)	
		S/CC2	=	FAIO PH5 SW8 SW7 NIOR +...	
		R/CC2	=	(R/CC2)	
	(RR24-RR31) —/→ (A24-A31)	AXRR/3	=	AXRR/13 +...	Load bytes 3 and 2 (bits 16 through 31) from I/O fast memory register to A-register.
		AXRR/13	=	AXRR/6	
		AXRR/6	=	FAIO PH5 SW7 SW8	
					Mnemonic: SIO (4C, CC) TIO (4D, CD) TDV (4E, CE) HIO (4F, CF)

(Continued)

Table 3-90. SIO, TIO, TDV, HIO Sequence for Integral IOP (Cont.)

Phase	Function Performed	Signals Involved			Comments
PH5 SW0 SW8 SW7 T5L (Cont.)	(RR16-RR23) → (A16-A23)	AXRR/2	=	AXRR/12 + . . .	Area of I/O fast memory register is 01, as defined by IOFR9 NIOFR8. Bits 16 through 31 contain the byte count and replace the byte address previously stored in A-register
		AXRR/12	=	AXRR/6 + . . .	
		IOFM	=	Logic set during preceding subphase	
		IOFR9	=	Logic set during preceding subphase	
		NIOFR8	=	Reset during previous operation	
	If SIO, address recognition, and SIO accepted, set flip-flop RW and maintain flip-flop NIOFM in a reset state	S/RW	=	(S/RW/1)	Prepare to write zeros into I/O fast memory, area 00, to clear the old status
		(S/RW/1)	=	(S/RW) + . . .	
		(S/RW)	=	(S/RW/2) + . . .	
		(S/RW/2)	=	SIOSP/1 DOR IOR + . . .	
		SIOSP/1	=	FUSIO PH5 SW8 SW7	
		R/RW	=	. . .	
		S/NIOFM	=	N(S/IOFM)	
		(S/IOFM)	=	(S/RW/2) + . . .	
		R/NIOFM	=	. . .	
	Reset flip-flop SW7	R/SW7	=	(R/SW7)	
		(R/SW7)	=	FAIO PH5 SW8 SW7+. . .	
	Set flip-flop SW9	S/SW9	=	SW8 STEP815 + . . .	Branch to SW9
		STEP815	=	NBRSW8 NBRSW10 NBRSW11 NBRSW13 NBRSW15 NRESET/A	
		R/SW9	=	. . .	
PH5 SW0 SW9 T8L	One clock long If RW was set during the preceding subphase:				
	(S0-S31) → (RW0-RW31)	RWXS/0-RWXS/3	=	RW + . . .	Transfer zeros to I/O fast memory register, area 00
	(FR0-FR7) → (A0-A7)	AXFR	=	(FAIO/1 PH5) SW9 + . . .	Load device controller status supplied on FR lines to A-register
					Mnemonic: SIO (4C, CC) TIO (4D, CD) TDV (4E, CE) HIO (4F, CF)

(Continued)

Table 3-90. SIO, TIO, TDV, HIO Sequence for Integral IOP (Cont.)

Phase	Function Performed	Signals Involved			Comments
PH5 SW0 SW9 T8L (Cont.)	If R field is zero or if AVO was returned by the device controller system, enable signal BRSW13	BRSW13	=	(FAIO PH5 SW9) AVO + (FAIO/1 PH5) SW9 RZ + ...	Advance to PH5 SW13. If either of these two conditions exists, the contents of the A-register will not be transferred to the private memory register
	Reset flip-flop NLR31F	S/NLR31F	=	N(S/LR31)	Force a one into private memory address line LR31 during PH5 SW10 to select private memory register Ru1
		(S/LR31)	=	(FAIO/1 PH5) SW9 + ...	
		R/NLR31F	=	...	
	If R field is not zero (NRZ), set flip-flop RW	S/RW	=	(S/RW/1) + ...	Prepare to write status and byte count into private memory register Ru1
		(S/RW/1)	=	(S/RW) + ...	
		(S/RW)	=	(FAIO/1 PH5) SW9 NRZ + ...	
		R/RW	=	...	
	If SIO, and valid start, enable signal (S/SXA)	(S/SXA)	=	FAIO PH9 SW0 VALST + ...	Preset adder logic for A ⟶ S in PH5 SW10
		VALST	=	FUSIO NCC1 NCC2	
	Set flip-flop SW10	S/SW10	=	SW9 STEP815	Branch to SW10
		R/SW10	=	...	
PH5 SW0 SW10 T8L	One clock long				
	(A0-A31) ⟶ (S0-S31)	Adder logic set during previous clock			Load status and byte count into private memory register Ru1
	(S0-S31) ⟶/ (RW0-RW31) (Ru1)	RWXS/0-RWXS/3	=	RW	
	Zero ⟶/ RW15	NRW15	=	NRW15XZ + ...	A zero in bit 15 indicates that the integral IOP is not a selector IOP
		NRW15XZ	=	FAIO/1 PH5 SW10	
	Enable signal AXZ	AXZ	=	(FAIO PH5) SW10 + ...	Reset A-register to zero
	If R field is odd (R31), enable signal BRSW13	BRSW13	=	(FAIO/1 PH5) SW10 R31 + ...	If odd R field, only one word of status is required
	If R field is even (NR31), enable signal (S/AXRR/4), set flip-flop IOFR8, and reset flip-flop NIOFM	S/IOFR8	=	(S/IOFR8)	Prepare to read most significant byte of CDW from I/O fast memory register, area 10
		(S/IOFR8)	=	(S/AXRR/4) + ...	
		(S/AXRR/4)	=	(FAIO/1 PH5) SW10 NR31 + ...	
		R/IOFR8	=	...	
					Mnemonic: SIO (4C, CC) TIO (4D, CD) TDV (4E, CE) HIO (4F, CF)

(Continued)

Table 3-90. SIO, TIO, TDV, HIO Sequence for Integral IOP (Cont.)

Phase	Function Performed	Signals Involved			Comments
PH5 SW0 SW10 T8L (Cont.)		S/NIOFM (S/IOFM) R/NIOFM	= = =	N(S/IOFM) (S/AXRR/4) + ...	Select IOP fast memory registers
	Set flip-flop SW11	S/SW11 R/SW11	= =	SW10 STEP815 + ...	Branch to SW11
PH5 SW0 SW11 T5L	One clock long (RR24-RR31) → (A24-A31)	AXRR/3 AXRR/13 IOFR8	= = =	AXRR/13 + ... IOFR8 NRW + ... Set during preceding subphase	Transfer most significant byte of CDW from area 10 of the I/O fast memory register to the A-register
	Set flip-flop IOFR9, maintain flip-flop IOFR8 in the set state, and maintain flip-flop NIOFM in the reset state	S/IOFR8 (S/IOFR8) (S/AXRR/4) R/IOFR8 S/IOFR9 (S/IOFR9) (S/AXRR/6) R/IOFR9 S/NIOFM (S/IOFM) R/NIOFM	= = = = = = = = = = =	(S/IOFR8) (S/AXRR/4) + ... FAIO/1 PH5 SW11 + ... (S/IOFR9) IOPOP (S/AXRR/6) + ... FAIO/1 PH5 SW11 ... N(S/IOFM) (S/AXRR/4) + ...	Prepare to read least significant byte of CDW from I/O fast memory, area 11 Select IOP fast memory registers
	Set flip-flop SW12	S/SW12 R/SW12	= =	SW11 STEP815 + ...	Branch to SW12
PH5 SW0 SW12 T8L	One clock long (A24-A31) → (A16-A23)	AXAL8	=	FAIO PH5 SW12 + ...	Shift most significant byte of CDW in A-register 8 places to the left
	(RR24-RR31) → (A24-A31)	AXRR/3 AXRR/13 IOFR8 IOFR9	= = = =	AXRR/13 + ... IOFR8 NRW + ... Set during preceding subphase Set during preceding subphase	Load least significant byte of I/O fast memory register, area 11, into A-register
					Mnemonic: SIO (4C, CC) TIO (4D, CD) TDV (4E, CE) HIO (4F, CF)

(Continued)

Table 3-90. SIO, TIO, TDV, HIO Sequence for Integral IOP (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH5 SW0 SW12 T8L (Cont.)	If R field not zero, enable signal (S/SXA), and set flip-flop RW	(S/SXA) = (FAIO PH5) SW12 + ...	Preset adder for A ⟶ S in PH5 SW13
		S/RW = (S/RW/1) + ...	Prepare to write CDW into private memory register R
		(S/RW/1) = (S/RW) + ...	
		(S/RW) = FAIO PH5 SW12 NRZ + ...	
		R/RW = ...	
	Set flip-flop SW13	S/SW13 = SW12 STEP815 + ...	Branch to SW13
		R/SW13 = ...	
PH5 SW0 SW13 T8L or T5L	One clock long; T5L if RZ, T8L if NRZ NAV0		
	Reset flip-flop IOCONST	R/IOCONST = (R/IOCONST) + ...	Drop control strobe
		(R/IOCONST) = FAIO PH5 SW13 + ...	
	Reset flip-flop IOFS	R/IOFS = (R/IOFS)	Drop function strobe
		(R/IOFS) = FAIO PH5 SW13 + ...	
	If R field not zero (NRZ):		
	(A0-A31) ⟶ (S0-S31)	Adder logic set during preceding subphase	Load CDW address into private memory register R
	(S0-S31) ⟶/ (RW0-RW31) (R)	RWXS = RW	
		RW = Set during preceding subphase	
	If not valid start (NVALST), enable signal BRPH9	BRPH9 = FAIO PH5 SW0 SW13 NVALST	If no address recognition or SIO not successful, branch to PH9
		NVALST = N(FUSIO NCC1 NCC2)	
	If SIO and valid start:		
	Enable signal (S/SXDM1)	(S/SXDM1) = FAIO PH5 SW13 VALST + ...	Preset adder logic for D-1 ⟶ S
	Set flip-flop IOFR8	S/IOFR8 = (S/IOFR8)	Prepare to transfer contents of D-register minus 1 to A-register, and byte 2 of D-register to I/O fast memory register, area 10
		(S/IOFR8) = (S/RW/4) + ...	
		(S/RW/4) = FAIO PH5 SW13 VALST + ...	
		R/IOFR8 = ...	

Mnemonic: SIO (4C, CC)
TIO (4D, CD)
TDV (4E, CE)
HIO (4F, CF)

(Continued)

Table 3-90. SIO, TIO, TDV, HIO Sequence for Integral IOP (Cont.)

Phase	Function Performed	Signals Involved			Comments
PH5 SW0 SW13 T8L or T5L (Cont.)	Reset flip-flop NIOFM	S/NIOFM	=	N(S/IOFM)	
		(S/IOFM)	=	(S/AXRR/4) + ...	
		R/NIOFM	=	...	
	Set flip-flop RW	S/RW	=	(S/RW/1)	
		(S/RW/1)	=	(S/RW) + ...	
		(S/RW)	=	(S/RW/4) + ...	
		R/RW	=	...	
	Set flip-flop SW14	S/SW14	=	SW13 STEP815 + ...	Branch to SW14
		R/SW14	=	...	
PH5 SW0 SW14 T5L	One clock long				
	(D-1) ⟶ (S0-S31)	SXDM1	=	Adder logic set during preceding subphase	Load byte 2 of the D-register in the I/O fast memory register, area 10. Byte 2 is the most significant byte of the next CDW address. Load the contents of the D-register into A-register. The A-register now contains the next CDW address minus 1
	(S0-S31) ⟶/⟶ (A0-A31)	AXS	=	FAIO PH5 SW14 + ...	
	(S16-S23) ⟶/⟶ (RW16-RW23)	RWXS/2	=	RW + ...	
		RW	=	Set during preceding subphase	
	Enable signal BRPH9	BRPH9	=	FAIO SW5 SW14 SW0 + ...	Branch to PH9
PH9 SW0 T8L	One clock long				
	Enable signals AXAL8-0 thru AXAL8-2	AXAL8-0 thru AXAL8-2	=	AXAL 8	Shift contents of A-register 8 places to the left
		AXAL 8	=	FAIO SW0 PH 9 +...	
	If SIO and valid start (VALST):				
	Enable signal (S/SXA)	(S/SXA)	=	FAIO PH9 SW0 VALST + ...	Preset adder logic for A ⟶ S in PH10
		S/IOFR8	=	(S/IOFR8)	Prepare to transfer byte 2 of the A-register to the I/O fast memory register, area 11
		(S/IOFR8)	=	(S/RW/4) + ...	
		(S/RW/4)	=	FAIO PH9 SW0 VALST + ...	
		R/IOFR8	=	...	
					Mnemonic: SIO (4C, CC) TIO (4D, CD) TDV (4E, CE) HIO (4F, CF)

(Continued)

Table 3-90. SIO, TIO, TDV, HIO Sequence for Integral IOP (Cont.)

Phase	Function Performed	Signals Involved			Comments
PH9 SW0 T8L (Cont.)	Set flip-flop IOFR9	S/IOFR9	=	(S/IOFR9) IOPOP	
		(S/IOFR9)	=	(S/RW/3) + ...	
		(S/RW/3)	=	FAIO PH9 SW0 VALST + ...	
		R/IOFR9	=	...	
	Reset flip-flop NIOFM	S/NIOFM	=	N(S/IOFM)	Selects IOP fast memory registers
		(S/IOFM)	=	(S/RW/3) + (S/RW/4) + ...	
		R/NIOFM	=	...	
	Set flip-flop RW	S/RW	=	(S/RW/1)	
		(S/RW/1)	=	(S/RW) + ...	
		(S/RW)	=	(S/RW/3) + (S/RW/4) + ...	
		R/RW	=	...	
	(B0-B31) ⟶ (S0-S31)	SXB	=	PXSXB NDIS	Transfer next instruction address to P-register
		PXSXB	=	NFAFL NFAMDS PH9	
	(S15-S31) ⟶/ (P15-P31)	PXS	=	PXSXB + ...	
	Set flip-flops MRQ and DRQ	S/MRQ	=	(S/MRQ)	Prepare to read next instruction from core memory
		(S/MRQ	=	(S/MRQ/2) + ...	
		(S/MRQ/2)	=	PXSXB NINTRAP2+...	
		R/MRQ	=	...	
		S/DRQ	=	(S/DRQ) NCLEAR	Inhibits transmission of another clock until data release is received from core memory
		(S/DRQ)	=	(S/MRQ/2) + (S/DRQ/2) + ...	
		(S/DRQ/2)	=	PH9 + ...	
		R/DRQ	=	...	
PH10 SW0 DR	One clock long				
	(A0-A31) ⟶ (S0-S31)	Adder logic set at PH9 clock			Load byte 2 of the A-register in the I/O fast memory register, area 11. Combined, area 10 and area 11 now contain the next CDW address minus 1. During IOPH3 SW10 of the order-out sequence, the CDW address is automatically incremented by 1
	(S16-S23) ⟶/ (RW16-RW23)	RWXS/2	=	RW + ...	
		RW	=	Set at PH9 clock	
	Reset flip-flop SW0	R/SW0	=	(R/SW0)	
		(R/SW0)	=	RESET/A + ...	
		RESET/A	=	CLEAR + ...	
		CLEAR	=	PH10 + ...	
	ENDE functions				
					Mnemonic: SIO (4C, CC) TIO (4D, CD) TDV (4E, CE) HIO (4F, CF)

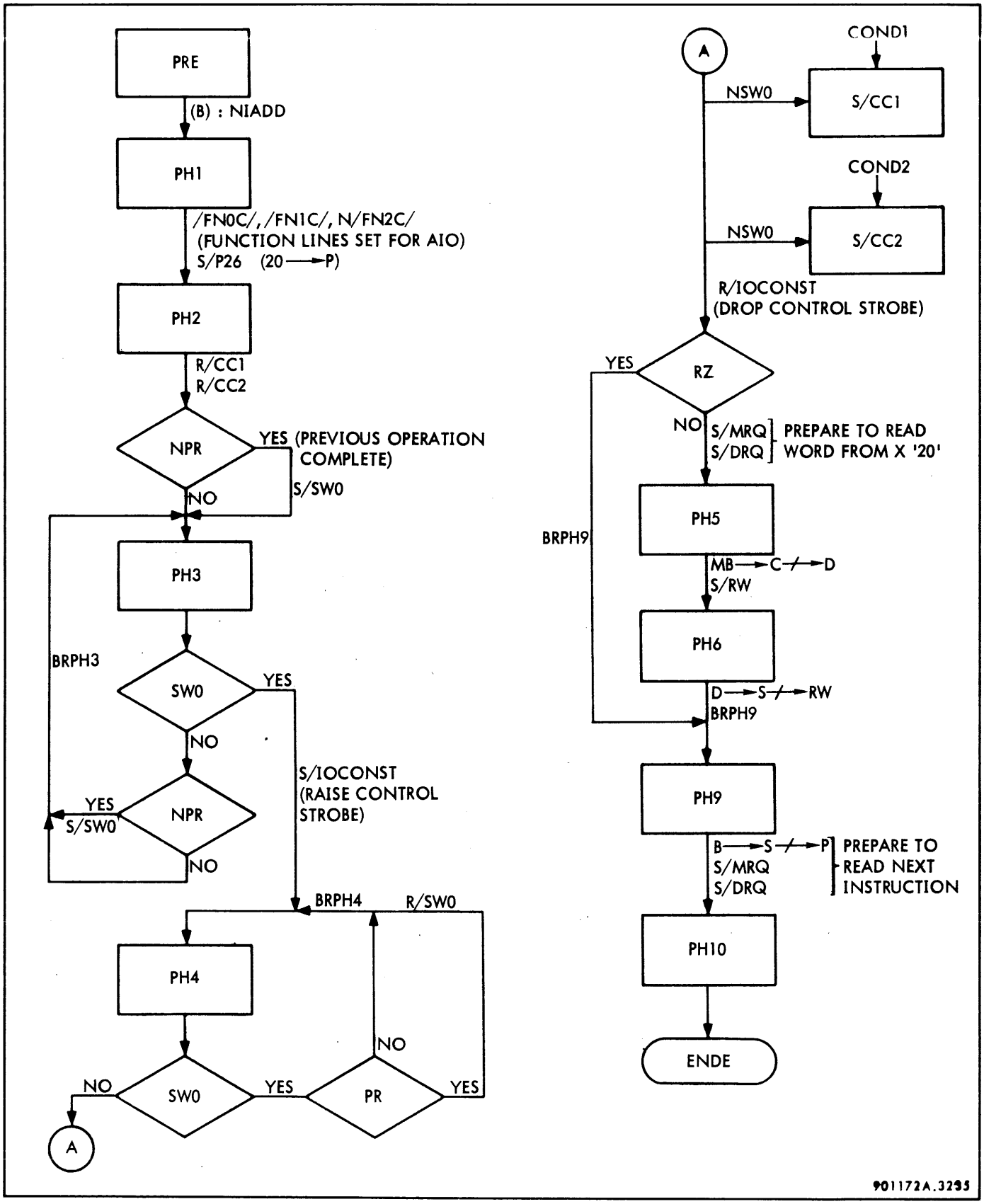


Figure 3-198. AIO Instruction Flow Diagram for MIOP

Table 3-91. AIO Sequence, MIOP

Phase	Function Performed	Function Performed	Comments
PREP	At end of PREP: (B): NIADD (A): RR (not used)		 Next instruction address Contents of private memory register R. Not used in this instruction
PH1 T5L	One clock long Opcode transferred from O-register to function lines: O2 ⟶ /FNC0C/ O6 ⟶ /FNC1C/ O7 ⟶ /FNC2C/ 20 ⟶ P	 /FNC0C/ = O2 /FNC1C/ = O6 /FNC2C/ = O7 S/P26 = (S/P26) + ... (S/P26) = (FAIO PH1) + ... R/P26 = PX + ... PX = FAIO PH1 + ...	 Specify AIO instruction Preset P-register to X'20' by forcing a 1 into bit 26 and resetting the other 16 bits. During PH6 a word is transferred from location X'20' of core memory into the C-register
PH2 T5L	One clock long Reset flip-flops CC1 and CC2 If NPR, set flip-flop SW0	 R/CC1 = (R/CC1) (R/CC1) = (R/CC1/1) + ... (R/CC1/1) = (FAIO PH2) + ... R/CC2 = (R/CC2) (R/CC2) = (R/CC2/1) (R/CC2/1) = (FAIO PH2) + ... S/SW0 = (S/SW0) NCLEAR (S/SW0) = (FAIO PH2) NPR + ... R/SW0 = (R/SW0)	 Early detect of NPR, indicating that previous operation has been completed
PH3 T5L	One or more clocks, depending on the state of flip-flop SW0 If flip-flop SW0 was not set in PH2, set SW0 when PR goes low	 S/SW0 = (S/SW0) NCLEAR (S/SW0) = (FAIO PH3) NPR + ... R/SW0 = (R/SW0)	 Wait for NPR from previous operation
			Mnemonic: AIO (6E, EE)

(Continued)

Table 3-91. AIO Sequence, MIOP (Cont.)

Phase	Function Performed	Signals Involved			Comments
PH3 T5L (Cont.)	If NSW0, enable signal BRPH3	BRPH3	=	FAIO PH3 NSW0 +...	Sustain PH3 until flip-flop SW0 gets set
	If SW0, set flip-flop IOCONST	S/IOCONST	=	(S/IOCONST)	Raise control strobe before entering PH4
		(S/IOCONST)	=	(FAIO PH3) SW0	
		R/IOCONST	=	(R/IOCONST) +...	
PH4 T5L or T8L	Two or more clocks, depending on the state of flip-flop SW0. First clock T5L. Subsequent clocks, if any, T5L, except for the last clock. Last clock T8L				
	If SW0, enable signal BRPH4	BRPH4	=	(FAIO PH4) SW0 NSW2	Sustain PH4 while flip-flop SW0 is in the set state
	If PR, reset flip-flop SW0	R/SW0	=	(R/SW0)	Wait until MIOP system returns PR signal in response to the control strobe signal
		(R/SW0)	=	(FAIO PH4) PR +...	
	If NSW0, and R field is zero (RZ), enable signal BRPH9	BRPH9	=	FAIO PH4 NSW0 RZ +...	If R is zero, status is not required
	If NSW0, and R field is not zero (NRZ), set flip-flops MRQ and DRQ	S/MRQ	=	(S/MRQ/2) +...	Memory request for reading status and IOP/device controller address from location X'20' of core memory
		(S/MRQ/2)	=	FAIO PH4 NRZ NSW0 +...	
		R/MRQ	=	...	
		S/DRQ	=	(S/DRQ) NCLEAR	Inhibits transmission of another clock until data release signal is received from core memory
		(S/DRQ)	=	(S/MRQ/2) +...	
		R/DRQ	=	...	
	If NSW0, set flip-flops CC1 and/or CC2 if specified	S/CC1	=	(FAIO PH4) NSW0 COND1 +...	Setting of CC1 and CC2 is controlled by conditions specified by the applicable device controller. If normal interrupt recognition, CC1 and CC2 are not set
		R/CC1	=	(R/CC1)	
		S/CC2	=	(FAIO PH4) NSW0 COND2 +...	
		R/CC2	=	(R/CC2)	
	If NSW0, reset flip-flop IOCONST	R/IOCONST	=	(R/IOCONST) +...	
		(R/IOCONST)	=	(FAIO PH5) NSW0 +...	Drop control strobe in response to PR
					Mnemonic: AIO (6E, EE)

(Continued)

Table 3-91. AIO Sequence, MIOP (Cont.)

Phase	Function Performed	Signals Involved			Comments
PH5 NSW0 DR	One clock long (MB0-MB31) ⟶ (C0-C31) (C0-C31) ⟶/⟶ (D0-D31)	CXMB DXC-0 thru DXC-3 DXC	= = =	DG = /DG/ DXC (FAIO PH5) NSW0 + ...	Transfer word from location X'20' of core memory into the C-register and then to the D-register
	Enable signal (S/SXD)	(S/SXD)	=	(FUAIO PH5) NSW0 + ...	Preset adder for D ⟶ S in PH6
	Set flip-flop RW	S/RW (S/RW/1) (S/RW) R/RW	= = = =	(S/RW/1) (S/RW) + ... (FUAIO PH5) NSW0+	Prepare to transfer status word into private memory register R
PH6 T8L	One clock long (D0-D31) ⟶ (S0-S31) (S0-S31) ⟶/⟶ (RW0-RW31) (R)	Adder logic set at PH5 clock RWXS/0 thru RWXS/3	= =	RW	Transfer status and IOP/device controller address to private memory register R
	Enable signal BRPH9	BRPH9	=	(FAIO PH6) FUAIO + ...	Information exchange between MIOP/device controller and the CPU completed. Branch to PH9
PH9 T5L	One clock long (B0-B31) ⟶ (S0-S31)	SXB PXSXB	= =	PXSXB NDIS NFAFL NFAMDS PH9	Transfer next instruction address to P-register
	(S15-S31) ⟶/⟶ (P15-P31)	PXS	=	PXSXB + ...	
	Set flip-flops MRQ and DRQ	S/MRQ (S/MRQ) (S/MRQ/2) R/MRQ S/DRQ (S/DRQ) (S/DRQ/2) R/DRQ	= = = = = = =	(S/MRQ) (S/MRQ/2) + ... PXSXB NINTRAP + ... (S/DRQ) NCLEAR (S/MRQ/2) + (S/DRQ/2) + ... PH9 + ...	Prepare to read next instruction from core memory Inhibits transmission of another clock until data release signal is received from core memory
PH10 DR	ENDE functions				
					Mnemonic: AIO (6E, EE)

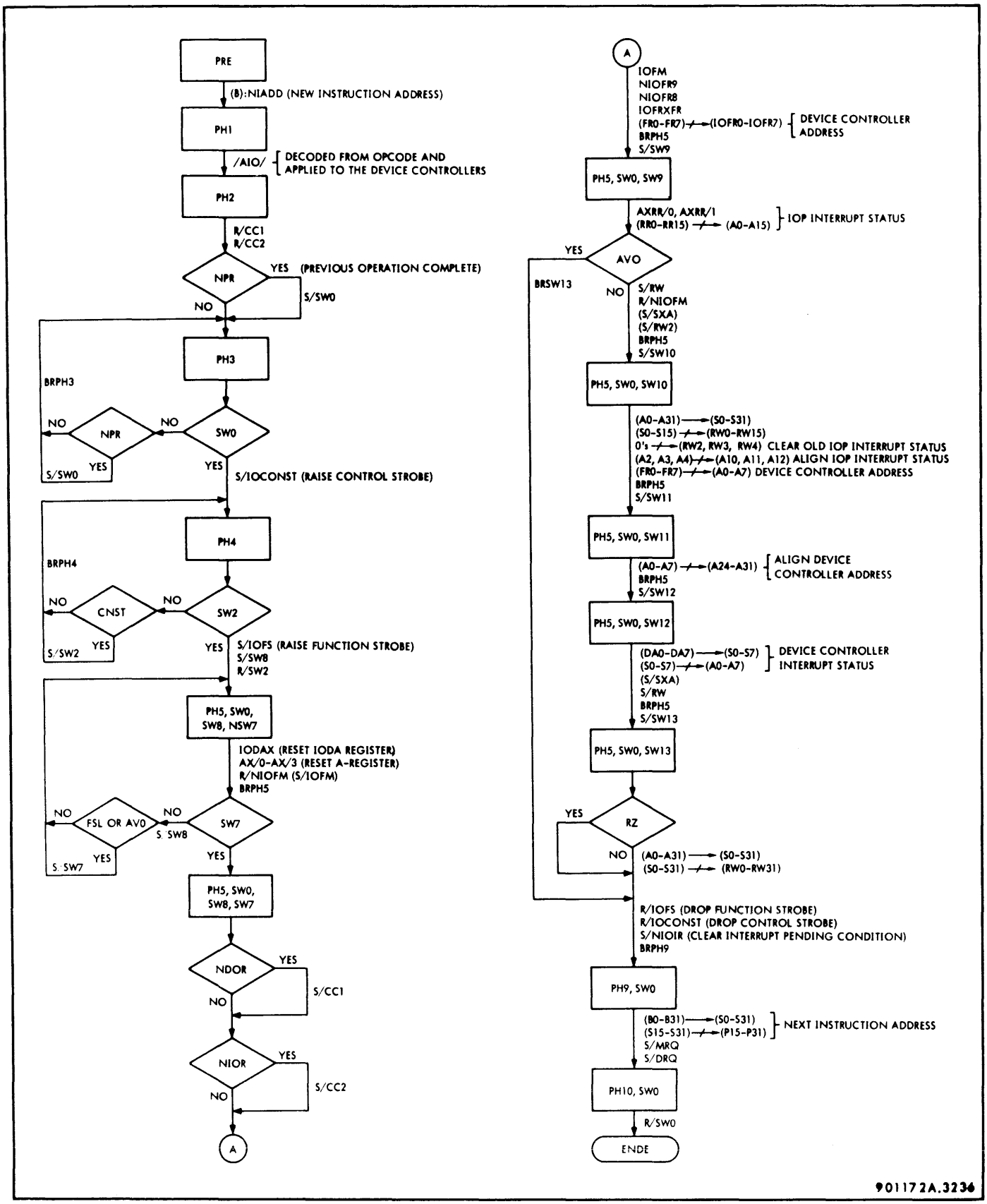


Figure 3-199. AIO Instruction Flow Diagram for Integral IOP

Table 3-92. AIO Sequence for Integral IOP

Phase	Function Performed	Signals Involved	Comments
PREP	At end of PREP: (B): NIADD (A): RR (not used)		 Next instruction address Contents of private memory register R. Not used in this instruction
PH1 T5L	Opcode decoded from the contents of the O-register, raising function indicator line /AIO/	/AIO/ = FUAIO FUAIO = OU6 OLE	Function indicator AIO is transmitted on a common line to all device controllers associated with the integral IOP. The device controller with an interrupt pending will respond by returning its address, condition codes, and status
PH2 T5L	One clock long Reset flip-flops CC1 and CC2	R/CC1 = (R/CC1) (R/CC1) = (R/CC1/1) + ... (R/CC1/1) = (FAIO PH2) + ... R/CC2 = (R/CC2) (R/CC2) = (R/CC2/1) + ... (R/CC2/1) = (FAIO PH2) + ...	Flip-flops CC1 and/or CC2 are set in PH5 SW8 NSW7, if specified by conditions in the device controller with an interrupt pending
	If NPR, set flip-flop SW0	S/SW0 = (S/SW0) NCLEAR (S/SW0) = (FAIO PH2) NPR + ... R/SW0 = (R/SW0)	Early detect of NPR, indicating that previous operation has been completed
PH3 T5L	One or more clocks, depending on the state of flip-flop SW0 If flip-flop SW0 was not set during PH2, set flip-flop SW0 when PR goes low	S/SW0 = (S/SW0) NCLEAR (S/SW0) = (FAIO PH3) NPR + ... R/SW0 = (R/SW0)	 Wait for NPR from previous operation
	If NSW0, enable signal BRPH3	BRPH3 = FAIO PH3 NSW0 + ...	Sustain PH3 until flip-flop SW0 gets set
			Mnemonic: AIO (6E, EE)

(Continued)

Table 3-92. AIO Sequence for Integral IOP (Cont.)

Phase	Function Performed	Signals Involved			Comments
PH3 T5L (Cont.)	If SW0, set flip-flop IOCONST	S/IOCONST	=	(S/IOCONST)	Raise control strobe before entering PH4
		(S/IOCONST)	=	(FAIO PH3) SW0	
		R/IOCONST	=	(R/IOCONST) + ...	Will be reset during PH5 SW13
PH4 T5L	Two or more clocks, depending on /CNST/				
	Set flip-flop SW2	S/SW2	=	(S/SW2)	Wait for /CNST/ to be returned through the IOP priority cable. /CNST/ is derived from IOCONST
		(S/SW2)	=	(FUAIO PH4) IOIR CNST + ...	
		CNST	=	/CNST/ NIOPOP (IOCONST + ...)	
		IOIR	=	NFF	IOIR indicates that an interrupt is pending, i.e., the applicable device controller has raised interrupt call line /IC/
		S/NIOR	=	IC	
		IC	=	/IC/	
		R/NIOR	=	NFUAIO	
	Enable signal BRPH4	BRPH4	=	(FAIO PH4) SW0 NSW2	Sustain PH4 until flip-flop SW2 has been set
	If flip-flop SW2 is set:				
	Set flip-flop IOFS	S/IOFS	=	(S/IOFS)	Raise function strobe to device controllers
		(S/IOFS)	=	FAIO PH4 SW2 + ...	
		R/IOFS	=	(R/IOFS)	Will be reset during PH5 SW13
	Reset flip-flop SW2	R/SW2	=	(R/SW2)	
		(R/SW2)	=	FAIO PH4 SW2	
	Set flip-flop SW8	S/SW8	=	NRESET/A BRSW8	Used to define the first two subphases in PH5
		BRSW8	=	(FAIO PH4) SW2 + ...	
		R/SW8	=	...	
PH5 SW0 SW8 NSW7 T5L	One or more clocks long, depending on the state of flip-flop SW7				
	Enable signal BRPH5	BRPH5	=	(FAIO PH5 SW0) NSW14 (NSW13 + ...)	Sustain PH5 during integral IOP sequence through subphase SW13
					Mnemonic: AIO (6E, EE)

(Continued)

Table 3-92. AIO Sequence for Integral IOP (Cont.)

Phase	Function Performed	Signals Involved			Comments
PH5 SW0 SW8 NSW7 T5L (Cont.)	Maintain flip-flop SW8 in the set state	S/SW8	=	NRESET/A BRSW8	Sustain subphase SW8 while flip-flop SW7 is in the reset state
		BRSW8	=	FAIO PH5 SW8 NSW7 + ...	
		R/SW8	=	...	
	Set flip-flop SW7	S/SW7	=	(S/SW7)	Wait for either an FSL or AVO response from the device controller system. FSL signifies that the device controller with an interrupt pending has responded to AIO FS. AVO signifies that the device controller which originally had an interrupt pending has in the meantime dropped its interrupt call
		(S/SW7)	=	FAIO PH5 SW8 NSW7 (FSL + AVO)	
		R/SW7	=	(R/SW7)	
	Enable signal IODAX	IODAX	=	(R/IODA)	Clear the IODA-register
		(R/IODA)	=	FUAIO PH5 SW8 NSW7 + ...	
	Enable signals AX/0 through AX/3	AX/0 thru AX/3	=	AX + ...	Clear the A-register
		AX	=	AXRR + ...	
		AXRR	=	NFF	
		S/NAXRR	=	N(S/AXRR)	
		(S/AXRR)	=	(S/AXRR/2) + ...	
		(S/AXRR/2)	=	FUAIO PH5 SW8 + ...	
		R/NAXRR	=	...	
	Reset flip-flop NIOFM	S/NIOFM	=	N(S/IOFM)	Prepare to read IOP interrupt status from IOP fast memory register, area 00. IOFM selects IOP fast memory registers
		(S/IOFM)	=	(S/AXRR/2) + ...	
		R/NIOFM	=	...	
PH5 SW0 SW8 SW7 T5L	One clock long Set flip-flops CC1 and/or CC2, if specified	S/CC1	=	(FAIO PH5) SW8 SW7 NDOR + ...	Setting of flip-flops CC1 and CC2 is controlled by conditions in the device controller with the interrupt pending
		R/CC1	=	(R/CC1)	
		S/CC2	=	(FAIO PH5) SW8 SW7 NIOR + ...	
		R/CC2	=	(R/CC2)	
					Mnemonic: AIO (6E, EE)

(Continued)

Table 3-92. AIO Sequence for Integral IOP (Cont.)

Phase	Function Performed	Signals Involved			Comments
PH5 SW0 SW9 T5L (Cont.)	Enable signal (S/SXA)	(S/SXA)	=	FAIO PH5 SW9 + ...	Preset adder logic for A⟶S in PH5 SW10
	Reset flip-flop NIOFM	S/NIOFM	=	N(S/IOFM)	Select IOP fast memory registers
		(S/IOFM)	=	(S/RW/2) + ...	
		R/NIOFM	=	...	
	Set flip-flop SW10	S/SW10	=	SW9 STEP815 + ...	Branch to SW10
		R/SW10	=	...	
PH5 SW10 SW0 T8L	One clock long (A0-A31) ⟶ (S0-S31)	SXA	=	Adder logic set at PH5 SW9 SW0 clock	Transfer contents of A-register to the sum bus
		NIOER8	=	Reset at PH5 SW9 SW0 clock	
		NIOFR9	=	Reset at PH5 SW9 SW0 clock	
	(S0-S15) ⟶/⟶ (RW0-RW15)	RWXS/0	=	RWXS/1 = RW + ...	Transfer contents of sum bus to IOFM register, area OO
		RW	=	Set PH5 SW9 SW0 clock	
	Zeros ⟶/⟶ (RW2, RW3, RW4)	RW2	=	S2 RWXS/0 N(FUAIO PH5 SW10)	Clear the old IOP interrupt status
		RW3	=	S3 RWXS/0 N(FUAIO PH5 SW10)	
		RW4	=	S4 RWXS/0 N(FUAIO PH5 SW10)	
	(A2, A3, A4) ⟶/⟶ (A10, A11, A12)	S/A10	=	A2 IOAXST + ...	Align IOP interrupt status in A-register
		IOAXST	=	IOINTST + ...	
		IOINTST	=	FUAIO PH5 SW10 + ...	
		S/A11	=	A3 IOAXST + ...	
		S/A12	=	A4 IOAXST + ...	
		R/A10-A12	=	AX/1	
	(FR0-FR7) ⟶/⟶ (A0-A7)	AXFR	=	FUAIO PH5 SW10 + ...	Transfer device controller address to the A-register
		AXZ	=	FAIO PH5 SW10 + ...	
	Set flip-flop SW11	S/SW11	=	SW10 STEP815 + ...	Branch to SW11
		R/SW11	=	...	
PH5 SW0 SW11 T5L	One clock long (A0-A7) ⟶/⟶ (A24-A31)	AXAR24	=	FUAIO PH5 SW11 + ...	Align device controller address in A-register
	Set flip-flop SW12	S/SW12	=	SW11 STEP815 + ...	
		R/SW12	=	...	
					Mnemonic: AIO (6E, EE)

(Continued)

Table 3-92. AIO Sequence for Integral IOP (Cont.)

Phase	Function Performed	Signals Involved			Comments
PH5 SW0 SW8 SW7 T5L (Cont.)	(FR0-FR7) —/→ (IOFR0-IOFR7)	IOFRXFR IOFRX	= =	(FUAIO P5 8 7) + ... (FUAIO P5 8 7) + ...	Transfer device controller address to the IOFR-register. Information stored in this register is used to select the appropriate IOFM-register
	Reset flip-flop NAXRR	S/NAXRR (S/AXRR) (S/AXRR/2) R/NAXRR	= = = =	N(S/AXRR) (S/AXRR/2) + ... FUAIO PH5 SW8 ...	Preset AXRR for transferring RR ——→ A in PH5 SW9
	Reset flip-flop NIOFM	S/NIOFM (S/IOFM) R/NIOFM	= = =	N(S/IOFM) (S/AXRR/2) + ...	Select IOP fast memory registers
	Set flip-flop SW9	S/SW9 STEP815 R/SW9	= = =	SW8 STEP815 + ... NBRSW8 NBRSW10 NBRSW11 NBRSW12 NBRSW13 NBRSW15 NRESET/A ...	Branch to SW9
PH5 SW0 SW9 T5L	One clock long (RR0-RR15) —/→ (A0-A15)	AXRR/0 AXRR IOFM NIOFR8 NIOFR9	= = = = =	AXRR/1 = AXRR NAXRRINH Set at PH5 SW8 SW7 clock Set at PH5 SW8 SW7 clock Reset during previous operation Reset during previous operation	Transfer IOP interrupt status to the A-register
	If AV0, enable signal BRSW13	BRSW13	=	(FAIO PH5) SW9 AV0 + ...	Branch to SW13
	Set flip-flop RW	S/RW (S/RW/1) (S/RW) (S/RW/2) R/RW	= = = = =	(S/RW/1) (S/RW) + ... (S/RW/2) + ... FUAIO SW9 PH5 + ...	Prepare to clear old IOP interrupt status
					Mnemonic: AIO (6E, EE)

(Continued)

Table 3-92. AIO Sequence for Integral IOP (Cont.)

Phase	Function Performed	Signals Involved			Comments
PH5 SW0 SW12 T5L	One clock long				
	(DA0-DA7) ⟶ (S0-S7)	SXDA	=	FUAIO PH5 SW12 + ...	Transfer device controller interrupt status to the A-register
	(S0-S7) ⟶/ (A0-A7)	AXS/0	=	AXS/4	
		AXS/4	=	AXS/2 + ...	
		AXS/2	=	FUAIO PH5 SW12 + ...	
	Enable signal (S/SXA)	(S/SXA)	=	FAIO PH5 SW12	Preset adder for A ⟶ S in PH5 SW13
	If R field is not zero (NRZ), set flip-flop RW	S/RW	=	(S/RW/1)	Prepare to transfer contents of A-register to private memory register R
		(S/RW/1)	=	(S/RW) + ...	
		(S/RW)	=	FAIO PH5 SW12 NRZ + ...	
		R/RW	=	...	
	Set flip-flop SW13	S/SW13	=	SW12 STEP815 + ...	Branch to SW13
		R/SW13	=	...	
PH5 SW0 SW13 T8L	One clock long				
	(A0-A31) ⟶ (S0-S31)	Adder logic set at PH5 SW12 clock			Transfer contents of A-register to private memory register R
	(S0-S31) ⟶/ (RW0-RW31) (R)	RWXS/0-RWXS/3	=	RW + ...	
		RW	=	Set at PH5 SW12 clock	
	Reset flip-flop IOFS	R/IOFS	=	(R/IOFS) + ...	Drop function strobe
		(R/IOFS)	=	FAIO PH5 SW13	
	Reset flip-flop IOCONST	R/IOCONST	=	(R/IOCONST) + ...	Drop control strobe
		(R/IOCONST)	=	FAIO PH5 SW13 + ...	
	Set flip-flop NIOIR	S/NIOIR	=	NIC + ...	Clear interrupt pending condition when device controller drops interrupt call
		IC	=	/IC/	
		R/NIOIR	=	NFUAIO	
					Mnemonic: AIO (6E, EE)

(Continued)

Table 3-92. AIO Sequence for Integral IOP (Cont.)

Phase	Function Performed	Signals Involved	Comments
PH5 SW0 SW13 T8L (Cont.)	Enable signal BRPH9	BRPH9 = FAIO PH5 SW0 SW13 NVALST + ... NVALST = NFUSIO + ...	Instruction complete branch to PH9 SW0
PH9 SW0 T5L	One clock long (B0-B31) ⟶ (S0-S31)	SXB = PXSXB NDIS PXSXB = NFAFL NFAMDS PH9	Transfer next instruction address to P-register
	(S15-S31) ⟶/⟶ (P15-P31)	PXS = PXSXB + ...	
	Set flip-flops MRQ and DRQ	S/MRQ = (S/MRQ) (S/MRQ) = (S/MRQ/2) + ... (S/MRQ/2) = PXSXB NINTRAP2 + ...	Prepare to read next instruction from core memory
		R/MRQ = ...	
		S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = (S/MRQ/2) + (S/DRQ/2) + ...	Inhibits transmission of another clock until data release is received from core memory
		(S/DRQ/2) = PH9 + ...	
		R/DRQ = ...	
PH10 SW0 DR	Reset flip-flop SW0	R/SW0 = (R/SW0) (R/SW0) = RESET/A + ... RESET/A = CLEAR + ... CLEAR = PH10-E + ...	
	ENDE functions		
			Mnemonic: AIO (6E, EE)

3-83 GLOSSARY OF TERMS

Glossaries of signal names for the CPU, Floating-Point, and Memory are listed in tables 3-93, 3-94, and 3-95, respectively. These glossaries define the main signals used in the Sigma 5 system. The glossary signals are identical to those found in the Sigma 5 logic equations (SDS drawing number 133263) except that the signals in the logic equations may be suffixed by a dash, followed by a number or letter. This suffix defines the driver used in the hardware and does not affect the signal logically. Other prefixes, suffixes, and conventions used in both the signal glossaries and the logic equations are shown below.

Prefixes

N	Not. Same as bar or overscore
S/	Set input to flip-flop
R/	Reset input to flip-flop
C/	Clock input to flip-flop
E/	Erase input to flip-flop (dc reset)
F/	Force input to flip-flop (dc set)
W/	Data write input to high-speed memory
L/	Address line to high-speed memory
K/	Read/write control to high-speed memory

Suffixes

/	Related logic signal. Example: XX/B is a logic signal related to logic signal XX
W	Usually means "one"
Z	Usually means "zero"
-U or /U	"Upper" bit positions (47-71). Floating-point only
-L or /L	"Lower" bit positions (0-31). Floating-point only

Symbols and Conventions

⟹	Implies
⟶	Transfer to
⟶/	Clock transfer to
/XX/	Cable signal

Table 3-93. Glossary of CPU and Integral IOP Signals

Signal	Definition
A0-A31	Bits 0 through 31 of A-register
A00	One-bit extension to most significant end of A-register
A0L-A7L A21L-A31L	Logic used for setting up bootstrap program during the time the LOAD switch is activated
A31XP32 A31XP33	P32 to be transferred to A31 P33 to be transferred to A31
ABO	Abort requested memory operation, and trap to location X'40'
/ABOC/	Abort signal to memory
ABOT	Abort timing pulse from delay line 2 (DL2/110)
ACCL/1	AC clock pulse derived from DL1
ACCLG	AC clock generate. Buffered latch used to retain clock pulse as it comes out of DL3 until another clock pulse is started down DL1
ADBDB	Arm and disable or disable interrupts
ADC3	Downcount A-register; begin looking at A3
ADMATCH	Address match between KSP15-31 and P15-31
ADNH	Memory address not here flip-flop
ADNHCL	Memory address not here clock. Timing pulse derived from DL3. Implies that sufficient time has elapsed for memory to have recognized the address
ADNHL	Logic term used for setting ADNH flip-flop
AEADB	Arm and enable or arm and disable interrupts
AEENLE	Arm and enable or enable or load enable interrupts
AH	Memory address here signal
(NAH AHCL)	Memory address not here and address recognition time
/AHC/	Memory address here signal from port C
AIB	Control flip-flop used in interrupt logic. Used during enter-active and leave-active interrupt level states

(Continued)

Table 3-93. Glossary of CPU and Integral IOP Signals (Cont.)

Signal	Definition
AIEI	Control flip-flop used by interrupt logic. Used when interrupt level enters active state
AIE2	Control flip-flop used by interrupt logic. Used when interrupt level leaves active state
/AIO/	Acknowledge IO interrupt request
ALARM	Flip-flop which causes AUDIO indicator to go on if COMPUTE switch is set to RUN and AUDIO switch is ON
AM	Arithmetic trap mask bit. Part of PSW1
AM/L	ARITH TRAP light indicator on PCP panel
ANLZ	Analyze
(ANLZ IA)	Analyze and indirect address
AR	Memory address release signal
/ARC/	AR from port C
ARE	Action-response signal from interrupt logic. Notifies CPU that action to interrupts has been accepted, and CPU can start clock and continue. Used in conjunction with CEINT
ARMCTR	Arm counter interrupts
ARMIO	Arm IO interrupts
ARMOVD	Arm override interrupts. Note that basic interrupts are divided into override, counter, and IO groups
/ASC/	Acknowledge service call
AUC3	Upcount A-register. Begin looking at A3
AUDIO	Signal sent to PCP speaker
AUDIO/L	AUDIO indicator on PCP panel
AVO	Available output priority signal. Generated when a function is not accepted
AX	Reset A-register. Overridden if a set term is present
AXAL8	Shift A-register left eight places
AXAR16	Shift A-register right 16 places (similarly AXAR8, AXAR24)

Table 3-93. Glossary of CPU and Integral IOP Signals (Cont.)

Signal	Definition
AXCC	Condition codes transferred to A-register: $CC_{1-4} \not\rightarrow A_{28-31}$, $CCZ \not\rightarrow A27$
	$CCZ \Rightarrow CC1 + CC2 + CC3 + CC4 = 0$
AXDIO	Transfer DIO data to A-register
AXFC	Transfer condition codes and floating control to A_{24-31}
AXFR	Transfer function response lines (FRn) to A-register
AXK	Transfer data switches (KSn) from PCP to A-register
AXLOAD	Logic term used to enable data to A-register during load procedure
AXMC	Transfer macro-counter (MC) to A-register: $MC_{0-7} \not\rightarrow A_{0-7}$
AXNR	$NR_{28-31} \not\rightarrow A_{28-31}$
AXPARITY	Transfer memory fault indicators to A-register: $MFL_{0-7} \not\rightarrow A_{24-31}$
AXPSW1	Transfer PSW1 to A-register
AXPSW2	Transfer PSW2 to A-register
AXR	$R_{28-31} \not\rightarrow A_{28-31}$
AXRR	$RR_{00-31} \not\rightarrow A_{00-31}$
AXRRINH	Inhibit RR to A transfer, or inhibit reading fast memory
AXS	Transfer sum bus to A-register
AXSL1	Transfer sum bus shifted left one position to A-register
AXSR1	Transfer sum bus shifted right one position to A-register
AXTR	$TR_{28-31} \not\rightarrow A_{28-31}$
AXZ	Put all zeros into the A-register
B0-B31	Bits 0 through 31 of B-register
B0001EN/1	Enables the two upper bits of B during multiply and double register shift
B0031Z	B0 through B31 contain zeros

(Continued)

Table 3-93. Glossary of CPU and Integral IOP Signals (Cont.)

Signal	Definition
(B31-BC31)	(B31 ⊕ BC31)
BC31	One-bit extension to least significant end of B-register
(BC = 1)	Byte count equals one
BC0, BC1	Byte count flip-flops
NBCDC0 BCDC1	Logic used for decrementing byte counter
NBCX	Logic used to reset byte counter
BCZ	Contents of byte counter equal zero
NBR	Not branch. When high allows a binary progression from one execution phase to the next (e.g., PH6 to PH7, PH1 to PH2)
BRP	Flip-flop used to keep track of location of program address. BRP = 1, program address in P-register BRP = 0, program address in B-register
BRPCP1 BRPCP5	Branch to PCP1 and PCP5, respectively
BRPH1	Branch to PH1
BRPHn	Branch to PHn
BRPRE4	Branch to PRE4
BRSW8	Branch to SW8
BRSWn	Branch to SWn
BX	Reset B-register
BXB-0 BXB-1	Logic which effects $B \xrightarrow{/} B_{0-15}$. Useful at BXP time
BXBGND-2 BXBGND-3	Inhibit transfer of $B \xrightarrow{/} B_{16-31}$. Used in conjunction with BXB, BXP
BXBL1	Shift B left one position
BXBR2	Shift B right one position
BXFP	Transfer $FP \xrightarrow{/} B_{00-31}$. FP ⇒ floating-point
BXP	Transfer (P) to B
BZC	Busy signal generated by counter interrupt group
BZI	Busy signal generated by IO interrupt group
BZO	Busy signal generated by override interrupt group

Table 3-93. Glossary of CPU and Integral IOP Signals (Cont.)

Signal	Definition
N(R/CC)	(R/CC)⇒Reset CC_{1-4}
CC1, CC2, CC3, CC4	Four-bit condition code register. Part of PSW1
CCXRWD	Enable setting of sense switches to be used to set condition codes. $KSS \xrightarrow{/} CC_{1-4}$
CCXTRACC	$TRACC \xrightarrow{/} CC_{1-4}$
CCZ	Contents of condition codes equal zero
CEINT	Flip-flop used to inhibit clock enable (CLEN). Used in conjunction with interrupts and watchdog timer. During watchdog timer runout, CEINT ensures that a clock has not just been sent down the delay line. During interrupt processing, CEINT inhibits clock until ARE is received from interrupt logic
CIF	Inhibit counter interrupt group flip-flop. Part of PSW2
CK-n	(CK-n)⇒fast-memory clock. n is a point of distribution of fast-memory clock
CK/n	Logic name given to the output of a fast-memory clock driver, where $1 \leq n \leq 12$
CL-n	(CL-n)⇒CPU ac clock. 01E01 is a point of distribution of CPU ac clock
CL/n	Logic name given to CPU ac clock driver, where $1 \leq n \leq 12$
CLEARMEM	Write zeros throughout core memory. KCPURESET and KSYSR must be activated simultaneously for CLEARMEM to be true
CLEN	Clock enable. Must be true for an ac clock to be generated in delay line
CLFP/n	Ac clock for floating point. FP⇒ floating point. $1 \leq n \leq 12$
CLIS	1 mc clock transmitted to external IOP
CNA, CNB	Control flip-flops used in basic interrupt logic. Used during write direct mode of communication with basic interrupt logic

(Continued)

Table 3-93. Glossary of CPU and Integral IOP Signals (Cont.)

Signal	Definition
CNLK	Flip-flop used as interlock so that only one interrupt request can be made for each time the interrupt button on the PCP is activated
CNLN7 CNLN8	Address lines generated by counter interrupt group
CNST	Control strobe generated for use by the IOP
COND1 COND2	Data used to set CC1 and CC2. IOP generates COND1 and COND2 to indicate whether or not an instruction is acceptable
CPUL1, CPUL2, CPUL3, CPUL4	Flip-flop outputs used to set IS2, IS3, IS4, and IS5. IS2-IS5 correspond to the count pulse interrupt levels of the override group
/CPURST/	Reset signal used by external interrupts
NCROSSCL	This term being low will inhibit CPU ac clock, because crossover clock has been requested
CROSSADD	Crossover address. Fast memory register has been addressed
CROSS	Combination of CROSSADD and memory request has been made
CROSSDCL	Crossover clock taken from DL1 (DL1/170)
CROSSD	Disables ac clock
CROSSEN	Crossover enable. Enables LR—/—►P
CROSSENR	Enable crossover read
CXMB	Enable memory bus (MB) to C-register
CXRR	Enable fast memory register data to C-register
CXS	Enable sum bus data to C-register
D0-D31	Bits 0 through 31 of D-register
DA0-DA7	Data lines between IOP and device controller
DAP	Odd parity line between IOP and device subcontroller
DARM	Disarm selected levels in basic interrupt (pertains to all three groups)
DASW4	(DATAIN + DATAOUT) NSW4
DAT16-DAT31	16 bits of data presented to interrupt logic during write direct mode

Table 3-93. Glossary of CPU and Integral IOP Signals (Cont.)

Signal	Definition
DATAIN	IOP has been requested to read data
DATAOUT	IOP has been requested to write data
DCCL/1	Clock to be used on C-register. Used in HOLDC logic
DCSTOP	Signal to stop CPU if address switches match memory address, and KADDRSTOP switch is on
DG	Data gate signal from memory
DIO0-DIO54	Direct input/output lines. (See Interface Design Manual for purpose of individual lines)
DIOEXIT	Direct input/output exit signal
DIOFS	Direct input/output function strobe
DIOIND	Direct input/output indicator. Used to enable DIO51 and DIO52 to set CC3 and CC4
DIOT1, DIOT2, DIOT3	Flip-flops used to accept FSA and generate DIOIND and DIOEXIT
DIOWD	Signifies that direct input/output function is a Write Direct
DIOX	Reset DIO register bits 0 through 31
DIOXB	Reset DIO register bits 32 through 47
DIOXDIO	Enable direct input/output data lines to DIO-register
DIOXS	Enable sum bus to DIO-register
DIS	Display. Allows a register other than the sum bus to be displayed
DIT/1	Divide iteration signal
DIVOVER	Divide overflow
DL1/040	40 nsec tap on delay line 1
DL2/050	50 nsec tap on delay line 2
DL3/080	80 nsec tap on delay line 3
DM	Decimal trap mask bit
DOR	Data order request. DOR = 1 implies order, DOR = 0 implies data. During an instruction, DOR is used to set condition code 1
DR	Data release from core memory

(Continued)

Table 3-93. Glossary of CPU and Integral IOP Signals (Cont.)

Signal	Definition
DR/1	Data release latch. Used to force a data release for crossover, to force a data release if address not here (ADNH), and to save DR if DR is received from memory before DRQ has been set
/DRC/	DR from memory port C
DRQ	Data request flip-flop (data from memory)
DRQAC	Combination of DRQ and ac clock. Hold term for DR/1 latch
DX	Reset D-register
DXC	Transfer C-register to D-register
DXCL1	Transfer C to D left one bit position
DXDR8	Shift D right eight places
DXS	Transfer sum bus to D-register
DXZ	Put all zeros into the D-register
ECPUL1, ECPUL2, ECPUL3	External count pulse (CPUL) request to count pulse interrupt levels 1, 2, 3
ED	End data line. Indicates last data or order byte is being transmitted
EI	External interrupt inhibit flip-flop. Part of PSW2
ENCNTR	Enable counter interrupt group request
ENIO	Enable IO interrupt group request
ENOVRD	Enable override interrupt group request
ENDE	End of execution
/ENXSTRI/	Enter exit strobe. Pertains to interrupt logic
/ES/	End service line. Indicates last byte of service is being transmitted
EWDM	Enable write direct mode. Pertains to interrupt logic
EXC	Execution flip-flop. Set when preparation phase is entered
FAADD	Family of Add instructions
FAARITH	Family of Arithmetic instructions
FABRANCH	Family of Branch instructions
FABYTE	Family of Byte instructions
FACAL	Family of Call instructions

Table 3-93. Glossary of CPU and Integral IOP Signals (Cont.)

Signal	Definition
FACOMP	Family of Compare instructions
FADIV	Family of Divide instructions
FADIVH	Family of Divide Halfword instructions
FADW	Family of Divide Word instructions
FAFL	Family of Floating point instructions
FAHW	Family of Halfword instructions
FAILL	Family of Illegal instructions
FAIM	Family of Immediate instructions
FAIO	Family of Input/Output (IO) instructions
FALCF	Family of Load Conditions and Floating Control instructions
FALCFP	FALCF or Function of Load Register Pointer
FALOAD	Family of Load instructions
FALOGIC	Family of Logic instructions
FAMDS	Family of Multiply, Divide, or Shift instructions
FAMDST	IFAST/L or IFAMDS
FAMT	Family of Modify and Test instructions
FAMUL	Family of Multiply instructions
FAMULNH	Family of Multiply-not-halfword instructions
FANIMP	Family of Nonimplemented instructions
FAPRIV	Family of Privileged instructions
FAPSD	Family of Program Status Doubleword instructions
FARWD	Family of Read Direct/Write Direct instructions
FASEL	Family of Select instructions
FASH	Family of Shift instructions
FASTABORT	Family of Store Abort instructions
FAST/L	Family of Pull Word, Pull Multiple, Load Multiple instructions
FAST/S	Family of Push Word, Push Multiple, Store Multiple instructions
FAST/A	Family of Pull or Push Word, Pull or Push Multiple instructions
FAST/B	Family of Load Multiple or Store Multiple instructions

(Continued)

Table 3-93. Glossary of CPU and Integral IOP Signals (Cont.)

Signal	Definition
FASTORE	Family of Store instructions
FASUB	Family of Subtract instructions
FAW	Family of Word instructions
FAWORDW	Family of Word or Doubleword instructions
FCXS	Transfer sum bus to condition code flip-flops and floating control flip-flops
FL1, FL2, FL3	Flag register
FMCL	Fast memory clock
FNC0, FNC1, FNC2	Function lines to IOP. (See Interface Design Manual for coding of lines)
FNF	Normalize mask bit, part of PSW1
FNL0, FNL1, FNL2	Function lines to interrupt logic. Decoded to determine function requested by write direct instruction
FNORM	Floating point normalize
FORCL	Force clock
FORCLEN	Force clock enable
FORCLG	Force clock gate signal
FP0-FP31	Floating point data lines 0 through 31
FPCLEN	Floating point clock enable
FPCON	Floating point connect
NFPOPTION	Not floating point option
NFPRR	Not floating point result ready
NFPXS	FPXS. Transfer sum bus to floating-point box
FR0-FR7	Function response lines. Pertains to IOP
FS	Function strobe
FSA	Function strobe acknowledge
NFSHEX	Not floating shift exit
FSL	Function strobe leading acknowledge
FUAIO	Function of AIO
FUANLZ	Function of Analyze
FUAWM	Function of Add Word to Memory
FUBAL	Function of Branch and Link
FUBCR	Function of Branch on Conditions Reset
FUBCS	Function of Branch on Conditions Set

Table 3-93. Glossary of CPU and Integral IOP Signals (Cont.)

Signal	Definition
FUBDR	Function of Branch on Decrementing Register
FUBIR	Function of Branch on Incrementing Register
NFUCS	Not Function of Compare Selective
FUDW	Function of Divide Word
FUEXU	Function of Execute
FUINT	Function of Interpret
FULAD	Function of Load Absolute Doubleword
FULRP	Function of Load Register Pointer
FUMH	Function of Multiply Halfword
FUMI	Function of Multiply Immediate
FUMMC	Function of Move to Memory Control
FUMSP	Function of Modify Stack Pointer
FUMTHOVER	Function of Modify and Test Halfword Overflow
FUMTSIGN	Function of Modify and Test Sign adjustment
FUPLW	Function of Pull Word
FUPLM	Function of Pull Multiple
FUPSW	Function of Push Word
FUS	Function of Shift
FUSF	Function of Shift Floating
FUSIO	Function of SIO
G0-G31	Generate terms from adder
G00	Extension to most significant end of generate logic
/GATCLK/	Gated clock. Used by external interrupts
GCLK	Gated clock. Used by basic interrupts
GND1101	Ground signal on frame 1, row 1, module location 01
GND2110	Ground signal on frame 2, row 1, module location 10
/GPADR0/-/GPADR3/	Group address data. Defines which external chassis of interrupts is addressed by WD instruction
GRP0	Group 0 (basic interrupts)
GXAD	Generate AD

(Continued)

Table 3-93. Glossary of CPU and Integral IOP Signals (Cont.)

Signal	Definition
GXAND	Generate AND
GXNAD	Generate NAD
HALT	Flip-flop that causes CPU to stop in PCP2 (PCP2 = idle phase)
/HBZC/	Busy signal transmitted by override group to interrupts of lower priority
/HBZI/	Busy signal transmitted by counter group to interrupts of lower priority
/HBZE/	Busy signal transmitted by I/O group to interrupts of lower priority
/HIO/	Halt I/O
/HOF/	Halt on parity error signal transmitted to memory
HOLDC	Hold term used on C-register latches
/HRQBZC/	Higher requesting or busy signal transmitted by override group
/HRQBZI/	Higher requesting or busy signal transmitted by counter group
/HRQBZE/	Higher requesting or busy signal transmitted by I/O group
/HRQBZI/	Higher requesting or busy signal transmitted by counter group
IA	Indirect address
IC	Leave active state signal to counter group
IBI	Leave active state signal to I/O group
IBO	Leave active state signal to override
IC	Interrupt request from internal I/O
IEC	Enter active state signal to counter group
IEI	Enter active state signal to I/O group
IEO	Enter active state signal to override group
IEN	CPU interrupt enable
IFAM	IFAST/S or IFAST/L or IFAMDS
IFAMDS	(FAMDS and NIPH10) or PCP2
IFAST/L IFAST/S	(FAST/L and NIPH10 and NPCP2)
II	Inhibit I/O interrupt group, part of PSW2

Table 3-93. Glossary of CPU and Integral IOP Signals (Cont.)

Signal	Definition
IN0-IN15	Enable flip-flops contained in basic interrupts
INDX	Index
INHXWD	Transfer write direct data to interrupt inhibit bits
INT	Interrupt request flip-flop used by CPU logic
INT0-INT8	Interrupt subroutine address lines received by CPU from interrupt logic
INT9	Interrupt request received by CPU from interrupt logic
INTRAP, INTRAP1, INTRAP2	Interrupt/trap sequence phase flip-flops
K/IOmBn	Where $0 \leq n \leq 4$. Clock to IO fast memory where $1 \leq m \leq 4$
L/IOmBn	Address lines to IO fast memory
w/IOmBn	Data lines to IO fast memory
IOACT	Internal IO active
IOAXST	Align I/O status in A-register
IOBO	Abort IO operation. No recognition due to AVO
IOCON	Internal I/O connect
IOCONST	I/O control strobe
IODA0-IODA7	I/O data register. Used for terminal order data out and regular data out
IODAP	Parity bit for IODA-register
IODAX	Clear IODA-register
IODAXA	Transfer A-register to IODA-register
IODC	I/O data chain
IOEN	I/O enable. Goes true when permissible for I/O to interrupt CPU
IOEN6	I/O enable during execution PH6
IOENIN	IO is enabled and FSL has been received
IOFF	Power-off interrupt request from power fail-safe monitor
IOFM	I/O fast memory. True when reading from or writing to internal I/O fast memory

(Continued)

Table 3-93. Glossary of CPU and Integral IOP Signals (Cont.)

Signal	Definition
IOFR0-IOFR9	I/O function response register. Outputs decoded to define fast memory address
IOFRX	Clear IOFR-register
IOFRXA	Transfer A to IOFR
IOFRXFR	Transfer FR to IOFR
IOFS	I/O function strobe
IOIN	I/O in. Accepts FSL
IOINH	Inhibit I/O, because of ABORT, processing INTRAP sequence, or ADNH
IOINTST	Used to align interrupt status in A-register, status was returned in response to an AIO instruction
IOIR	Flip-flop that receives interrupt call (IC) from internal IOP. IOIR is put on IR line through a cable driver. IR is used to set common I/O interrupt level in basic interrupt chassis
IOLN7-IOLN8	Interrupt address lines 7 and 8 brought high by I/O interrupt group
/IONEN/	ION enable. Power-on enable from power fail-safe monitor
/IONN/	Power-on signal from power fail-safe monitor
IOPA0-IOPA2	IOP address lines 0, 1, and 2
IOPADD	Internal IOP is addressed
IOPC	IO parity check
NIOPEX	No external IOP in system
IOPG	IO parity generator on most significant byte of A-register
IOPH0-IOPH3	Internal IO phases 0 through 3
IOPOP	Internal IOP is plugged in
IOR	Input-output line during IOP service. IOR = 1 output, IOR = 0 input
IORB	I/O read backward
IOSC	Internal IO service call flip-flop
IOTRIN	I/O transfer-in-channel
IOWD	Watchdog Timer runout during I/O operation
IP0-IP15	Sixteen arm flip-flops of basic interrupt logic

Table 3-93. Glossary of CPU and Integral IOP Signals (Cont.)

Signal	Definition
IPH10	IOSC interrupted CPU during execution PH10
IR	IOP interrupt request. Used to set common IOP interrupt level in basic interrupt
IS0-IS15	Sixteen request flip-flops of basic interrupt logic
NISIN0	N(IS0 IN0)
NISNIP0	N(IS0 NIP0)
IX	Index flip-flop
IXAL	Index alignment flip-flop
K0-K31	32 carry bits of sum bus
K00	Extension to most significant end of carry logic
KADDRSTOP	Address stop signal from PCP panel
NKAHOLD	Not address hold
KAS/1 NKAS/2 NKAS/B	If KAS/1 is true and NKAS/2 is false, one of the following PCP switches is activated: DATA ENTER, DATA CLEAR, STORE SELECT ADDR, STORE INSTR ADDR, INSERT PSW1, INSERT PSW2, COMPUTE STEP, COMPUTE RUN, DISPLAY SELECT ADDR, DISPLAY INSTR ADDR, INSTR ADDR INCREMENT, or LOAD. If NKAS/B is true, none of the above listed switches are activated
KC	Signal from PCP, low during no clock or continuous clock
NKC/B	Signal from PCP, high during no clock or continuous clock
KCLEAR/B	Data clear signal from PCP
KCLRPSW1	Clear PSW1 signal from PCP
KCLRPSW2	Clear PSW2 signal from PCP
NKCLRPSW/B	Not clear PSW signal from PCP
KCONT	True if PARITY ERROR MODE switch is in CONT position
KCPURESET	True if CPU RESET switch is activated
KD	True if REGISTER DISPLAY switch is ON and CLOCK MODE switch is not in CONT position
NKDI	True if REGISTER SELECT switch is not in the EXT position

(Continued)

Table 3-93. Glossary of CPU and Integral IOP Signals (Cont.)

Signal	Definition
KDISPLAK/B	Display contents of SELECT ADDRESS switches
KDISPLAQ/B	Display contents of INSTRUCTION ADDRESS indicators
KENTER/B	Enter data signal
KFILL/B	True if LOAD switch is activated
KHOP	Halt on parity error
KINCRE/B	Increment instruction address
KINLVSEL	True if INTERLEAVE SELECT switch is in the DIAGNOSTIC position
KINTRP	True if INTERRUPT switch is activated
KIORESET	True if I/O RESET switch is activated
KPSW1/B	True if INSERT PSW1 switch is set
KPSW2/B	True if INSERT PSW2 switch is set
KRUN	True if COMPUTE switch is in RUN position
KS0-KS31	32 DATA switches, true if particular switch is in the 1 position
KSC	Low during CONT CLOCK
KSP15-KSP31	17 SELECT ADDRESS switches, true if particular switch is in the 1 position
KSS1-KSS4	SENSE switches, true if particular switch is in the 1 position
KSTEP/B	True if the COMPUTE switch is in the STEP position
KSTORK/B	Store in SELECT ADDRESS location
KSTORQ/B	Store in INSTRUCTION ADDRESS location
KSXA, KSXB, KSXC, KSXD, KSXS	REGISTER SELECT switch signals, select A, B, C, D, or sum bus
KSYSR	True if SYSTEM RESET switch is activated
KUA21-KUA31	True as a function of UNIT ADDRESS switch
KWDTR	True if WATCHDOG TIMER switch is in the OVERRIDE position
/LB15/-/LB31/	Address lines to core memory
LCK0-LCK1	Write lock decoding used to cause abort
LEVACT	Leave Active State signal to interrupt logic

Table 3-93. Glossary of CPU and Integral IOP Signals (Cont.)

Signal	Definition
LEVARM	When exiting an interrupt level, leave level in armed state
LIN00-LIN08	Nine address lines associated with an interrupt request to CPU
LINREQ	Interrupt request from external interrupts
LIO3-LIO7	Address lines to internal I/O fast memory
LK0, LK1, LK2, LK3	Signals which are decoded to define a particular location in fast memory which is used for write lock data
K/LKn	Where $0 \leq n \leq 3$. Clock to fast memory used for write lock data
W/LKn/m	Where $0 \leq n \leq 3$, m $0 \leq 5$. Write data lines to write lock fast memory
LOCK0-LOCK7	Data output from write lock fast memory modules
LOCKW	Enable clock to fast memory write locks
/LR23/-/LR31/	Address lines to CPU fast memory
LRXD	Transfer $D_{12-14} \rightarrow LR_{29-31}$, where D_{12-14} equals index register selection
LRXR	Transfer $R_{28-31} \rightarrow LR_{28-31}$
LRXZ	Put all zeros into the LR lines
MASTER	Flip-flop denoting slave mode when MASTER = 0
NMB0CRO-NMB3CRO	CRO ⇒ memory address is a crossover address or address of fast memory register. MB0-MB3 ⇒ Bytes 0 through 3. NMB0CRO being low, implies write byte 0 data in fast memory
MBXS	Transfer sum bus to memory bus (MB) data lines
MB0-MB31	Memory data bits 0 through 31
MC0-MC7	Eight-bit macro-counter, used to keep record of multiply iterations, etc.
MCDC3, MCDC7	Decrement macro-counter. If MCDC3 is true, macro-counter will be decremented by 10_{16}. If MCDC7 is true, macro-counter will be decremented by one
MCX	Clear macro-counter
MCXNPL1	Transfer NP left one to MC (i.e., NP26 → MC1)
MCXPL2	Transfer P left two to MC (i.e., P26 → MC0)

(Continued)

Table 3-93. Glossary of CPU and Integral IOP Signals (Cont.)

Signal	Definition
MCXS	Transfer $S_{0-7} \rightarrow MC_{0-7}$
MCZ	Contents of macro-counter equal zero
MFL0-MFL7	Memory fault lights 0 through 7
MFR	Memory fault reset signal
MIT	Multiply iteration signal
/MQC/	Memory request to port C
/MR/	Memory reset signal
MRC	Flip-flop set if memory request out. If DRQ set and no memory request was made, NMRC = 1, in this case DRQ. NMRC generates CLEN
MRCL	Memory request clock
MRQ	Memory request flip-flop set by CPU
MRQP1	Flip-flop which causes DRQ to be set on clock following the clock which set MRQ
MUSIC	Flip-flop used to drive speaker on PCP panel
/MW0/-/MW3/	Write byte lines to core memory
O1-O7	Seven-bit opcode register
ODINST	Order-in status enable
OL0-OLF	Decode of bits 4 through 7 of opcode register (O lower)
ORAB	Override memory requests to ports A and B, giving highest priority to port C
ORDERIN, ORDEROUT	Order in, order out. Applicable during internal IOP operations
ORDSW4	Implies order or switch 4 (SW4). SW4 ⟹ data chain
OU0-OU7	Decode of bits 1 through 3 of opcode register (O upper)
OVERIND	Overflow indicator flip-flop
OVLN6-OVLN8	Interrupt address lines 6, 7, 8 driven by override interrupt group
OX	Clear O-register
OXC	Transfer $C_{1-7} \rightarrow O_{1-7}$
P15-P31	Seventeen-bit address register
P32-P33	Two additional bits of address registers, used for byte count, etc.

Table 3-93. Glossary of CPU and Integral IOP Signals (Cont.)

Signal	Definition
P32HOLD, P33HOLD	Hold P32 and P33 at their current value
PARITYOK	Parity OK signal from memory port C
PBAHOLD	Hold byte address in P32 and P33 at current value
PC	Parity check signal received by internal IOP from a device controller
PCP1-PCP6	Processor control panel phases 1 through 6
PCPACT	Processor is active in PCP phases
PDC18, PDC22, PDC25, PDC29	Decrement P counter. PDC18 explained as follows: P19-P33 = 0, then decrement P15-P18 by one
PE	Parity error from memory port C
PEINT	Flip-flop which accepts PEM and is used to set parity error interrupt level
PEM	Parity error in memory latch
PH1-PH10	CPU execution phases 1 through 10
/POKC/	Parity OK from C port. Used to generate PARITY OK signal
PON	Power-on request to power-on interrupt level
PR	Proceed signal from external IOP
PR0-PR31	Propagate signals for sum bus
PR00	Extension to most significant end of propagate logic
/PRC/	Proceed signal on cable. Used to generate PR
PRE1-PRE4	CPU preparation phases 1 through 4
PREIO	Preparation for IO service
PREOPER	Signal true for those instructions which require reading contents of effective address
NPREP	CPU not in PRE1, 2, 3, or 4
PRETR	PRE-TRAP. Flip-flop denoting when CPU may TRAP out of preparation phases
PRI	Proceed signal
PROBEOVER	Probe for overflow
PROBOVER/H	Probe for overflow, halfword

(Continued)

Table 3-93. Glossary of CPU and Integral IOP Signals (Cont.)

Signal	Definition
PRXAD	Enable propagation of AD
PRXAND	Enable propagation of AND
PRXNAD	Enable propagation of NAD
PRXNAND	Enable propagation of NAND
PSW1XS	Transfer sum bus to PSW1
PSW2XS	Transfer sum bus to PSW2
PUC18	Upcount P-register. Add one to bit 18 level of P-register
PULLUP	Source is a terminator. Provides additional drive input to clock drivers
NPX	Not clear P-register
PXINT	Transfer interrupt address to P-register. Also used as source of enter active state signal to interrupt logic
PXK	Transfer address switches to P (i.e., store select address, display select address)
PXS	Transfer sum bus to P-register
PXSXB	Transfer B to P via the sum bus
PXTR	Transfer TRAP address to P
R0	Interrupt level 0 requesting service
R2	Interrupt level 2 requesting service
NR01	Interrupt levels 0 and 1 are not requesting service
NR23	Interrupt levels 0 through 3 are not requesting service
R28-R31	Four-bit R-register. Used to retain the R field of instructions
RDC31	Decrement contents of R-register
RDXMFI	Read and reset MEMORY FAULT indicators
REIP1	Interrupt level 1 is requesting service. Level 0 is not active or requesting
REIP3	Interrupt level 3 is requesting service. Levels 0 through 2 are not active or requesting
REN	Reset enable flip-flops signal to interrupt logic
REU	Register extension unit
RIO	Reset I/O

Table 3-93. Glossary of CPU and Integral IOP Signals (Cont.)

Signal	Definition
RP24-RP27	Four-bit register pointer (RP) register
K/RP0B0	Clock signal to CPU internal fast memory, byte 0
L/RP0B0	CPU internal fast memory address line
W/RP0B0	CPU internal fast memory data write line
RR0-RR31	CPU internal fast memory read data lines
RPXS	Transfer sum bus to register pointer (RP) register
RQBZC	Requesting or busy signal from counter interrupt group
RQBZI	Requesting or busy signal from I/O interrupt group
RQBZO	Requesting or busy signal from override interrupt group
/RRW0/-/RRW31/	Read/write data signals on cable from CPU to internal fast memory
RS	Request service strobe
RSA	Request service acknowledge
RSCLEN	RS clock enable
/RST/	Reset signal to internal IOP
RTC	Real-time clock signal. Generated by power monitor
RTO9	Request terminal order
RUC31	Upcount R_{28-31}
RW	Write signal to fast memory
RW0-RW31	Write-data lines to internal CPU or internal IOP fast memory
RW15XZ	Zero RW15
RX	Clear R_{28-31}
RXC	Transfer C to R_{28-31}
RXS	Transfer sum bus to R_{28-31}
RZ	Contents of R_{28-31} equal zero
RWB0-RWB3	Read/write byte lines to fast memory
S0-S31	32 sum bus bits
NSCINH	Not service call inhibit
SC	Service call

(Continued)

Table 3-93. Glossary of CPU and Integral IOP Signals (Cont.)

Signal	Definition
SCL	Single clock latch
SFT	Shift
SGTZ	Sum greater than zero
/SIO/	Start IO
SPIM	Sign-pad immediate. Extend sign of data for immediate instructions
SPW	Sign-pad with ones
SPZ	Sign-pad with zeros
SR8, 9, 10, 11, and 13	Set term to IS flip-flops of interrupt levels 8, 9, 10, 11, and 13, respectively
/ST/	Start signal from power monitor
START	Derived from /ST/
STEP815	Signal which allows switches to progress in a binary fashion from SW8 through SW15
STRAP	SET-TRAP signal caused by watchdog timer runout
SW0-SW15	16 switch signals which help to define certain states of the CPU
SWK1, 2, 3, 4, 5, 6, 12	Logical decoding of functions performed by the PCP switches
N(S/SXAEORD)	Not transfer the exclusive OR of A and D to sum bus
N(S/SXAMD)	Not transfer (A minus D) to sum bus
N(S/SXAORD)	Not transfer (A or D) to sum bus
N(S/SXAP1)	Not transfer (A plus one) to sum bus
SXBF	SXB flip-flop
SXB	Transfer B to sum bus
SXDA	Transfer DA data lines to sum bus
NSYSR	Not (system reset or start)
T5, T8, T11	CPU clock pulses. Nominal values: T5 = 220 nsec T8 = 290 nsec T11 = 420 nsec
T5EN	Enable T5 clock pulse
T8EN	Enable T8 clock pulse (T11 is automatically selected if T5 or T8 are not enabled)
/TDV/	Test device
/TIO/	Test I/O

Table 3-93. Glossary of CPU and Integral IOP Signals (Cont.)

Signal	Definition
TESTS	Signal which enables the testing of the contents of the sum bus
TODATA	Terminal order data
TORDIN	Terminal order in
TR28-TR31	Four-bit TRAP address register
(R/TR)	Reset TR-register
TRACC1-TRACC4	Four-bit TRAP accumulator register. Holds least significant hex digit of call instruction trap address, as well as data for setting CC1-4 for certain bytes of TRAP conditions
TRAP	Flip-flop which is set when CPU attempts to perform an illegal operation
TRIG	Enable signal which gates the setting of IS flip-flops (TRIG ⟹ trigger)
VALST	Valid start
VDATAIN	Valid data in
VORDER	Valid order
WAIT/L	Wait indicator on PCP panel
WCT1-WCT6	Six-bit flip-flop register used for watchdog timer accumulator
WDINT	Internal write direct
WDTA	Flip-flop set when watchdog accumulator has reached the length of time when an operation should have been completed
WDTR	Watchdog timer reset. One of the terms used to set WDTRAC
WDTRAC	Reset watchdog timer accumulator. This allows count to start over
WK0-WK1	Write key flip-flops
ZXX	Set IOWD because watchdog timer ran out during an I/O operation
128KC	128 kilocycles per second clock
n KC	n kilocycles per second clock
1 MC	1 megacycle per second clock
N1MCS	Clock pulse formed from the combination of N1MC 2MC
500 CPS	500 cycles per second

Table 3-94. Glossary of Floating Point Signals

Signal	Definition
A47-A31	A-register (57 bits, multipurpose)
A2831XB	B3128 → A2831 (for postnormalizing in multiply – see AXSL4)
A4851Z	A4851 = 0 (for normalization logic)
A5255Z	A5255 = 0 (for normalization prediction logic)
ALM	Right align memory operand [augend (EW) in ADD/SUB]
ALR	Right align register operand [addend (R) in ADD/SUB]
ASN	A-register is simple-normalized: i.e., $1/16 \leq A < 1$ or $-1 \leq A < -1/16$ } (A47 thru A51 are not equal to one another) or forced high if FN = 1 in add/sub (when interrogated) to inhibit normalization
ASPP	Add/subtract preparation (mantissas are aligned, therefore prepare to add or subtract)
AX	Enable A4731 (via -U, /L – see below)
AXL	Enable A0031 only
AX-U	Enable A4771 (= AX)
AX/L	Enable A0031 (= AX + AXL)
AXFP	FP0031 → A0031 (via -4 thru -7)
AXS	S4731 → A4731 (via -1 thru -7)
AXSL1	S4831 → A4730, B48 → A31 (via -1 thru -7)
AXSL4	S5131 → A4727, 0's → A2831 except where A2831XB is high (via -1 thru -7)
AXSL4/1	High speed version of AXSL4 used for control logic
AXSR2	S4629 → A4831, S45 → A47 (via -1 thru -7) (S/A47/2) = (G46 + PR46 NK46) BXBL2 = AXSR2
AXSR4	S4627 → A5031, 0's → A4749 (via -1 thru -7)
AXSR4/1	High speed version of AXSR4 used for control logic
B48-B31	B-register (56 bits, multipurpose)
BX	Enable B4831 (via -U, -L)
BXBL1	B4931 → B4830 (via -U, -L) (K46 → B31 if long DIV; K46 → B71 if short div)
BXBL2	B5031 → B4829, S3130 → B3031 (via -U, -L)
BXFP	FP3100 → B4807, B4871 → B0831 (via /U, /L – see below) (Multiply functions where B-register is played backward)
BXFPU	FP3108 → B4871, 0's → B0031
BXFP/L	FP0700 → B0007, B4871 → B0831 (= BXFP)
BXFP/U	FP3108 → B4871 (= BXFP + BXFPU)
C46-C31	Buffers for DXDL1 and DXDR1 logic
S/CC1/FP	Set CC1 in CPU
S/CC2/FP	Set CC2 in CPU

(Continued)

Table 3-94. Glossary of Floating Point Signals (Cont.)

Signal	Definition
D46-D31	D-register (58 bits, multipurpose)
DIT	Divide iterations (excluding final 2 clocks of PH8)
DIT/1	Divide iterations (excluding final clock of PH8)
DIV	Divide (O decoding)
DPP	Divide preparation (operands are simple-normalized, therefore prepare to divide)
DSN	D-register is simple-normalized i.e., $1/16 \leq D < 1$ or $-1 \leq D < -1/16$ (D47 thru D51 are not equal to one another)
DX	Enable D4631 (via -U, /L – see below)
DX-U	Enable D4671 (= DX)
DX/L	Enable D0031 (= DX + PH4)
DXA	A4731—/→D4731 (A47—/→D46)
DXDL1	D4831—/→D4730, sustain D46, 0—/→D31 (via -U, -L and C4631) (DX2—/→D)
DXDR1	D4630—/→D4731, sustain D46 (via -U, -L and C4631) (DX-1/2—/→D)
DXS	S4631—/→D4631 (via -U, /L – see below)
DXS-U	S4671—/→D4671 (= DXS)
DXS/L	S0031—/→D0031 (= DXS + PH4)
E0-E7	E-register (8 bits, for exponent processing)
E0003Z	E0003 = 0
E0407Z	E0407 = 0
EDC3	Downcount E0003 [inhibited if $E < -96_{10}$ to prevent false overflow indication arising from certain cases of unrecoverable underflows (e.g., $00000001_{16} \times 00000001_{16}$)]
EDC7	Downcount E0407
EUC3	Upcount E0003 [inhibited if $E > 96_{10}$ to prevent false underflow indication arising from certain cases of unrecoverable overflows (e.g., $7FFFFFFF_{16} \div 00000001_{16}$)]
EX	Enable E0007
EXFM64	F minus 64—/→E
EXNE	Invert E0007
EXNFM64	(Inverted F) minus 64—/→E
F0-F7	F-register (8 bits, primarily iterations counter)
FDC3	Downcount F0003
FDC7	Downcount F0407
FEOF	Floating exponent overflow ($E \geq 64$) (result $\neq 0$)
FEUF	Floating exponent underflow ($E < -64$) (result $\neq 0$) N(significance trap with FZ = 0)
FN/FNF	FN flag in CPU PSW (called FNF in hardware). FN = 1 inhibits normalization in add/sub
FP0-FP31	Bidirectional bus between CPU and box for transmission of data

(Continued)

Table 3-94. Glossary of Floating Point Signals (Cont.)

Signal	Definition
FPCON	Floating point box control (from CPU). Starts box by setting PH1, and stores sign of EW in MWN during PH1
FPDIS	Floating point display. Substitutes information to be displayed onto the FP bus in place of normal logic. Also contributes to SDIS logic
FPR	Floating polarity reversed. When high indicates that the sign of an intermediate result is opposite to that of the final result
FPRR	Floating point result ready. Signals the CPU that the results are to be available on the FP bus starting with the next clock (which is PH9 in the box)
FPX	High when the box is feeding the FP bus
FPXMISC	Miscellaneous signals ⟶ FP0031 for display purposes
FPXSL	S0031 ⟶ FP0031
FPXSU	S47 ⟶ FP0, $\overline{E1}$ ⟶ FP1, E0207 ⟶ FP0207, S4871 ⟶ FP0831
FS	FS flag in CPU PSW. FS = 1 causes trap if > 2 postnormalizing shifts are needed or if result = 0 in add/sub
FX	Enable F0007
FXD	D0007 —/→ F0007
FXNA	Inverted A0007 —/→ F0007 (storing inverted A0007 instead of true outputs is for signal loading only and has no logical significance)
FZ	FZ flag in CPU PSW. FZ = 1 causes trap on underflow instead of store zero
G46-G31	Generate terms in adder
G0003, etc.	Group generate terms in carry system (high when a carry is generated out of the specified bit range)
GXAD	A D ⟶ G if NSDIS (via /7, /A thru /E) (flip-flops set by the S/SX ... terms)
GXAND	A $\overline{D}$ ⟶ G if NSDIS (via /7, /A thru /E)
K46-K31	Adder carries (none can be high unless SXADD NSDIS) Special cases: K15: Output directly enabled by SXADD to assure early turnoff of higher order carries derived from K15 (for benefit of S = 0 test following an add) K31: Input carry for 2's complementing (= PRXNAND NSDIS) K71: Can be forced high by special input to G0003 = K31 PH10 NFPRD for cases where only bits 4771 are to be 2's complemented
KFPXMISC	Switch signal raising FPXMISC if FPDIS
KFPXSL	Switch signal raising FPXSL if FPDIS
KFPXSU	Switch signal raising FPXSU if FPDIS
KSXA	Switch signal raising PRXAD/'s and PRXAND/'s if SDIS (for A ⟶ PR ⟶ S)
KSXB	Switch signal raising SXB if SDIS (for B ⟶ S)
KSXD	Switch signal raising PRXAD/'s and PRXNAD/'s if SDIS (for D ⟶ PR ⟶ S)
M1	2^1 of multiplier bit pair } Polarity matched to that of multiplicand to produce product
M2	2^0 of multiplier bit pair }
MIT	Multiply iterations (excluding final clock of PH7)
MUL	Multiply (O decoding)

(Continued)

Table 3-94. Glossary of Floating Point Signals (Cont.)

Signal	Definition
MWN	Memory word negative. Flip-flop that stores sign of the EW operand
O2, O6, O7	Opcode bits from CPU. Define particular floating point instruction
PH1-PH10	Phase flip-flops
PR46-PR31	Propagate terms in adder
PR0003, etc.	Group propagate terms in carry system. (PR0003 means PR00-PR03 are all high)
PRXAD	A D ⟶ PR if NSDIS (via /7, /A thru /E)
PRXAND	A ND ⟶ PR if NSDIS (via /7, /A thru /E)
PRXNAD	NA D ⟶ PR if NSDIS (via /7, /A thru /E)
PRXNAND	NA ND ⟶ PR if NSDIS (via /7, /A thru /E) ⟵ PRXNAND NSDIS ⇒ 1 ⟶ K31
	(Flip-flops set by the S/SX ... terms)
R31	R31 from CPU. Register address add, used to determine product length in multiply
RTZ	Result is zero flip-flop. Detects zero result in mantissa
S/RW/FP	Sets RW flip-flop in CPU, causing write into CPU scratch-pad
S46-S31	Sum bus bits. (S45 is synthesized – see AXSR2)
S0031XFP	FP0031 ⟶ S0031 (via SXFP/4, /A)
S4607XFP	FP0 ⟶ S4647 (sign), FP0831 ⟶ S4871 (mantissa – MSW), FP0007 ⟶ S0007 (exponent) (via SXFP/4, /U)
SDIS	S display. Substitutes A, B, or D for normal logic on S bus; also kills all carries
SW0, 1, 2	General purpose control flip-flops
S/SXA	Preset A ⟶ S (i.e., S/PRXAD, PRXAND) (A ⟶ PR)
SXADD	The S bus is performing an arithmetic operation where carries are involved (= PRXNAND + GXAD)
S/SXAMD	Preset A - D ⟶ S (i.e., S/PRXAD, PRXNAND, GXAND) [N(A ⊕ D) ⟶ PR, A ND ⟶ G, 1 ⟶ K31]
S/SXAMD/1	S/SXAMD unconditionally
S/SXAMD/2	S/SXAMD if conditions do not call for S/SXAPD
S/SXAPD	Preset A + D ⟶ S (i.e., S/PRXAND, PRXNAD, GXAD) [(A ⊕ D) ⟶ PR, A D ⟶ G]
S/SXAPD/1	S/SXAPD unconditionally
S/SXAPD/2	S/SXAPD as a condition of signals demanding a minimum number of logic levels. When S/SXAPD/2 is high, S/SXAPD reduces to: DIT (K46 ⊕ MWN ⊕ SXADD) [(Divide: = MWN on 1st clock, then = (K46 = MWN)] + PH6 NO6 N(K46 ⊕ PR46) [(Add/sub: = (S46 = 0)]
S/SXAVA	Preset |A| ⟶ S (i.e., S/SXA if A47 = 0, or S/SXMA if A47 = 1)
S/SXAVD	Preset |D| ⟶ S (i.e., S/SXD if D46 = 0, or S/SXMD if D46 = 1)
SXB	B4831 ⟶ S4831, 0's ⟶ S4647 (via -L, -U)
S/SXD	Preset D ⟶ S (i.e., S/PRXAD, PRXNAD (D ⟶ PR)
SXFP/4	FP0007 ⟶ S0007 (= S4607XFP + S0031XFP)
SXFP/A	FP0831 ⟶ S0831 (= S0031XFP)
SXFP/U	FP0 ⟶ S4647, FP0831 ⟶ S4871 (= S4607XFP)

(Continued)

Table 3-94. Glossary of Floating Point Signals (Cont.)

Signal	Definition
S/SXMA	Preset -A⟶S (i.e., S/PRXNAD, PRXNAND) (NA⟶PR, 1⟶K31)
S/SXMD	Preset -D⟶S (i.e., S/PRXAND, PRXNAND) (ND⟶PR, 1⟶K31)
SZL	S0031 = 0
SZU	S4771 = 0
TRAP	TRAP to X'44' and inhibit write into scratch-pad in CPU Conditions: Underflow (exp. $< 64_{10}$) (Result $\neq 0$) (FZ = 1) + Overflow (exp. $\geq 64_{10}$) (Result $\neq 0$) + Divide by zero (SW1 = 1 when interrogated) + Significance trap (FS = 1) (FN = 0) (top 3 hexes of unnormalized |result| in add/sub = 0)

Table 3-95. Glossary of Memory Signals

Signal	Definition
00YNIP-32YNIP	Y negative inhibit driver signals. Generated by the true condition of TNYI and the reset output of the respective M-register flip-flops
00YPIP-32YPIP	Y positive inhibit driver signals. Generated by the true condition of TPYI and the reset output of the respective M-register flip-flops
3YNC0N-3YNC7N	Y negative current predrive elements. Decode bits L19, L20, and L21 of the address register
3YNV0N-3YNV7N	Y negative voltage predrive elements. Decode bits L18, L22, and L24 of the address register
3YPC0N-3YPC7N	Y positive current predrive elements. Decode bits L19, L20, and L21 of the address register
3YPV0N-3YPV7N	Y positive voltage predrive elements. Decode bits L18, L22, and L24 of the address register
(4K)	True when the memory size switches are in the configuration NS0 NS1. Used in address and interleave logic
(12K)	True when the memory size switches are in the configuration S0 NS1. Used in the address here and interleave logic
ABOA, ABOB, ABOC	Abort signals from the CPU to ports A, B, and C. Used to override a write operation to prevent changing the contents of a memory location
ADA, ADB, ADC	Port priority signals. Used to indicate which port has access decision
ADACO, ADBCO, ADCCO	Intermediate port logic signals. Used to gate various timing signals to the CPU and IOP
ADADG	Port A data gate enable signal
ADAM, ADBM, ADCMB, ADCMI	Amplified versions of ADAS, ADBS, and ADC
ADAMW, ADBMW, ADCMW	Amplified versions of ADAS, ADBS, and ADC

(Continued)

Table 3-95. Glossary of Memory Signals (Cont.)

Signal	Definition
ADAS, ADBS	Amplified versions of ADA and ADB
ADBDG	Port B data gate enable signal
ADCDG	Port C data gate enable signal
/AHA/, /AHB/, /AHC/	Address here signals as they appear from each port cable driver. True whenever a memory module responds to an implemented address configuration
AHA, AHB, AHC	Address here signals as they appear in the internal memory logic. True when the requested address (post-map, post-interleave) compares with the setting of the starting address switches for that particular memory module (bits 15-19)
AP	Almost parity. Third level parity signal
APA	Port A priority signal. True when AHA and MQA have been received and memory is not busy. Used to trigger the port delay line, causing IPD to go true
APB	Port B priority signal. True when AHB and MQB have been received and memory is not busy. Same function as APA
APE	Almost parity error. Fourth level parity signal
/ARA/, /ARB/, /ARC/	Address release signals as they appear in the interface. Used to allow CPU and IOP to drop their address lines. Generated by the memory logic when an address has been entered into the address register
CFA, CFB	Control signals for ports A and B. Used to allow memory to set up for a cycle for A or B while memory is busy
DECENP	Sense preamplifier selection enable signals
DG	Data gate enable signal. True during a read process
/DGA/	Data gate signal from port A telling requesting unit that memory data output lines are active and may be sampled
DGA0-DGA7	Port A data output gates. Gate output of M-register onto data lines for port A
/DGB/	Data gate signal from port B. Same function as /DGA/
DGB0-DGB7	Port B data output gates. Same function as DGA0-DGA7
/DGC/	Data gate signal from port C. Same function as /DGA/
DGC0-DGC7	Port C data output gates. Same function as DGA0-DGA7
/DRA/, /DRB/, /DRC/	Data release signals as they appear in the interface. Perform different functions relating to data, depending upon whether the memory operation is read, write, or write partial
/EDRA/, /EDRB/, /EDRC/	Early data release signals as they appear in the interface. May or may not be present, depending upon whether the memory operation is read, write, or write partial
HALT	Generated whenever the following conditions exist: either the power fail-safe and reset (PFSR) is true, or halt on fault (HOF) and memory fault (MF), ANDed together, are both true. The HALT signal is used to cause memory busy (MB) to stay true and ignore any further memory requests
HOF	Halt on fault signal. Generated by the CPU whenever it is desirable to halt memory when a memory parity error occurs
IPD	Initiate port delay signal. Used to trigger the Port Delay (PORTDL). IPD is used in port A and B only for access decision
L18-L31	L-register outputs. Used for X-Y selection. Bits 15-17 do not go into the L-register. Instead, they are used for address here (AH), mapping, and interleaving to determine which of the eight possible memory modules is to be selected

(Continued)

Table 3-95. Glossary of Memory Signals (Cont.)

Signal	Definition
L18SEN, L19SEN	Duplicate logic of L18 and L19. Used to drive the preamplifier selection signals PASL0-PASL7 where they appear as L18J and L19J
/LA15/, /LA31/	Port A address lines as they appear at the input to the port A cable receivers
LA15-LA31	Port A address lines as they appear at the output of the port A cable receivers. LA20-LA29 are direct inputs to the L-register
LA16S-LA19S	Port A memory selection signals. May be interleave modified by LA30 and LA31. Inputs to the starting address comparison logic
LA18L-LA19L	Special 4K and 8K address lines as they appear at the input to the L-register
LA30L, LA31L	Port A memory selection signals. May be interleave modified by LA16-LA19. Inputs to the L-register
/LB15/, /LB31/	Port B address lines as they appear at the input to the port B cable receivers. Also the CPU address lines as they appear at the output of the CPU cable drivers
LB15-LB31	Port B address lines. Same function as LA15-LA31
LB16S-LB19S	Port B memory selection signals. Same function as LA16S-LA19S
LB18L, LB19L	Port B special address lines. Same function as LA18L-LA19L
LB30L, LB31L	Port B memory selection signals. Same function as LA30L-LA31L
/LC15/ - /LC31/	Port C address lines. Same function as /LA15/-/LA31/
LC15-LC31	Port C address lines. Same function as LA15-LA31
LC16S-LC19S	Port C memory selection signals. Same function as LA16S-LA19S
LC18L, LC19L	Port C special address lines. Same function as LA18L-LA19L
LC30L, LC31L	Port C memory selection signals. Same function as LA30L-LA31L
LXA--	Port A transfer signals for address lines into the L-register
LXB--	Port B transfer signals for address lines into the L-register
LXC--	Port C transfer signals for address lines into the L-register
LXL	Source of clear and latch signals for the L-register
LXL--	L-register latch signals generated by LXL
/LX15/-/LX31/	IOP address lines as they appear at the output of the IOP cable drivers
M00-M31	M-register flip-flops. Accept data inputs from ports A, B, and C, or from core memory discriminator outputs. Each complete memory block (4, 8, 12, or 16K) has its own M-register
M32	Parity flip-flop. Set during a read restore or partial-write operation if the word from memory contains an even number of ones. Also set during a partial or full write if the data to be strobed into core memory has an even number of ones
M32XP	Parity flip-flop transfer signal. Used to set flip-flop M32 during parity generation (partial or full write)
/MA00/-/MA31/	Port A delay lines. Input-output of cable receiver/drivers
MA00-MA31	Port A data lines. Inputs to the M-register
MB	Memory busy signal. True during the time memory is in the process of satisfying a memory request. Also kept true during a memory halt condition to prevent any new memory requests from being honored
/MB00/-/MB31/	Port B data lines. Input-output of cable receiver/drivers

(Continued)

Table 3-95. Glossary of Memory Signals (Cont.)

Signal	Definition
MB00-MB31	Port B data lines. Inputs to the M-register
/MC00/-/MC31/	Port C data lines. Input-output of cable receiver/drivers
MC00-MC31	Port C data lines. Inputs to the M-register
MD00P-MD31P	Sense amplifier/discriminator outputs from core memory. Inputs to the M-register
MD32P	Sense amplifier/discriminator output from parity bit in core memory. Input to parity flip-flop
MF	Memory fault signal. Used to gate MFL00-MFL07 memory fault signals
/MFL00/-/MFL07/	Memory fault lamp signals. Used to specify in which memory module a memory fault (typically a parity error) occurred. These signals appear only on port C
MFR	Memory fault reset signal. Generated by the CPU and used to reset MF
MI	Memory initiate signal. Used to begin a memory cycle when the address here and memory request signals are both true
MQA, MQB, MQC	Memory request signals from external units as they appear in the memory logic
/MNN/	Margins not normal signal as it appears in the interface. Generated by any one of the PT16 power supplies in the system if its associated margin switch is not in the normal position. The end effect is to extinguish the NORMAL MODE indicator on the Processor Control Panel
MR	Memory reset signal from the CPU (where it appears as MRS). Resets control elements in core memory. Do not confuse this signal with the MR signaled by the CPU as a memory request
MW0-MW3	Byte presence indicator flip-flops. Determine which memory operation is to take place. If all flip-flops are reset, a read-restore operation occurs. If all flip-flops are set, a full-write operation occurs. If neither of these conditions exists, a partial-write operation occurs
MW0A-MW3A	Write-byte signals to port A from an external unit. Used to set the byte presence indicator flip-flops
MW0B-MW3B	Write-byte signals to port B from an external unit. Used to set the byte presence indicator flip-flops
MW0C-MW3C	Write-byte signals to port C from an external unit. Used to set the byte presence indicator flip-flops
MXA0-MXA3	Port A transfer signals between the M-register set input and the port A data lines
MXB0-MXB3	Port B transfer signals between the M-register set input and the port B data lines
MXC0B-MXC3B	Port C transfer signals between the M-register set input and the port C data lines
MXC0I-MXC31	Port C transfer signals between the M-register reset input and the port C data lines
MXD0B-MXD3B	Core memory discriminator transfer signals between the M-register set input and the discriminator outputs
MXD0I-MXD31	Core memory discriminator transfer signals between the M-register reset input and the discriminator outputs
MXM0-MXM3	M-register clear and latch signals
MXM32	Parity flip-flop clear and latch signal
N0, N1, N2	Memory number switches. Used to control the MEMORY FAULT lamps on the Processor Control Panel

(Continued)

Table 3-95. Glossary of Memory Signals (Cont.)

Signal	Definition
NIL	Interleave logic. True if interleaving is not established
NTSSTB	Not time for sense strobe signal. When false, causes the strobe signals, SAST0-3, to go false and to strobe the preamplifier outputs into the sense amplifiers
ORIL	Override interleave signal. Generated by the Processor Control Panel and used to disable interleaving
ORSP	Override slow port signal. Generated by the CPU to cause port C to have the highest priority. Locks out ports A and B even though the CPU may not have a memory request pending
PASL0-PASL7	Sense preamplifiers selection signals. Enable the proper preamplifiers by decoding address bits L18, L19, and L23
PE	Parity error signal. True if a parity error is detected during a read-restore or partial-write operation. Also called a fifth level parity signal
/PEA/, /PEB/, /PEC/	Parity error signals as they appear at the output of the individual port cable drivers
PF00-PF30	Parity first level gates
PFSR	Power fail-safe and reset signal. Can go true as a result of receiving MR (memory reset) from the CPU, or ST (start) from the power fail-safe circuits. Used to reset MF
POK	Parity OK flip-flop. Used to signal external unit that parity check was satisfactory on word just received from memory. Signal is ANDed with port logic to develop /POKA/, /POKB/, and /POKC/
PORTDL	Port delay line. Triggered by IPD, which generates TP00 through TP100 in 20 nsec steps. Generated whenever port A or B receives a memory request, unless CFA or CFB is active
PORTDL2	Port delay line. Triggered by TP100, which generates TP120 through TP200 in 20 nsec steps
PS00-PS27	Parity second level gates
RD	Read signal. Generated whenever all four byte lines are false
READDL	Read delay line. Triggered by MI to generate TR000 through TR620 in 20 nsec steps. Used to control read portion of a memory cycle
RESMW	Latch signal for byte presence indicator flip-flops MW0-3
S0, S1	Memory size switches. Used to establish size of memory module
S8	Interleave switch. True for 8K
S16	Interleave switch. True for 16K
S32	Interleave switch. True for 32K
S64	Interleave switch. True for 64K
SAST0-SAST3	Sense amplifier strobe signals
SPA00P-SPA07P SPA00N-SPA07N	Sense preamplifier outputs for byte 0
SPA08P-SPA15P SPA08N-SPA15N	Sense preamplifier outputs for byte 1
SPA16P-SPA23P SPA16N-SPA23N	Sense preamplifier outputs for byte 2
SPA24P-SPA32P SPA24N-SPA32N	Sense preamplifier outputs for byte 3

(Continued)

Table 3-95. Glossary of Memory Signals (Cont.)

Signal	Definition
/SRAA/, /SRAB/, /SRAC/	Second request allowed signals as they appear in the interface. Used to signal external unit that another memory request may be issued
ST	Start signal. Generated by the power-on circuit, which is true for at least 300 ms
TNXC	Time for negative X current. True during the read portion of a memory cycle if L22 ≠ L25 and during the write portion of a memory cycle if L22 = L25. Used to enable selected negative X voltage predrive switches
TNXV	Time for negative X voltage. True during the read portion of a memory cycle if L22 = L25 and during the write portion of a memory cycle if L22 ≠ L25. Used to enable selected negative X voltage predrive switches
TNYC	Time for negative Y current. True during the read portion of a memory cycle if the sum of L22, L23, and L25 is odd and during the write portion of a memory cycle if the sum is even. Used to enable selected negative Y current predrive switches
TNYI	Time for negative Y inhibit. True during the write portion of a memory cycle if the sum of L22, L23, and L25 is even. Used to short-circuit those Y current switches where a zero is to be generated in core memory
TNY10-TNY13	Amplified versions of TNYI
TNYV	Time for negative Y voltage. True during the read portion of a memory cycle if the sum of L22, L23, and L25 is even and during the write portion of a memory cycle if the sum is odd
TPXC	Time for positive X current. True during the read portion of a memory cycle if L22 = 25 and during the write portion of a memory cycle if L22 ≠ L25
TPXV	Time for positive X voltage. True during the read portion of a memory cycle if L22 ≠ L25 and during the write portion of a memory cycle if L22 = L25
TPYC	Time for positive Y current. True during the read portion of a memory cycle if the sum of L22, L23, and L25 is even and true during the write portion of a memory cycle if the sum is odd
TPYI	Time for positive Y inhibit. True during the write portion of a memory cycle if the sum of L22, L23, and L25 is odd
TYPI0-TPYI3	Amplified version of TPYI
TPYV	Time for positive Y voltage. True during the read portion of a memory cycle if the sum of L22, L23, and L25 is odd and during the write portion of a memory cycle if the sum is even
WF	Write full signal. True whenever all the byte presence indicator flip-flops are set
WP	Write partial signal. True whenever some (but not all) of the byte presence indicator flip-flops are set
WRITEDL	Write delay line. Triggered by TR160 during a read-restore or full-write operation and by TR560 during a write-partial operation. Used to control the write portion of a memory cycle
X, NX	Current direction control signals for the X selection. X is true when L22 = L25. NX is true when L22 ≠ L25
X8	Interleave logic: interleave size is 8K
X161, X162	Interleave logic: interleave size is 16K
X32	Interleave logic: interleave size is 32K
X641, X642	Interleave logic: interleave size is 64K

(Continued)

Table 3-95. Glossary of Memory Signals (Cont.)

Signal	Definition
XNCD0-XNCD3	X negative current predrive elements. Decode L19 and L25
XNCK0-XNCK3	X negative current predrive elements. Decode L26 and L27. XNCD0-3 and XNCK0-3 form a matrix for the X negative current predrive system
XNVD0-XNVD7	X negative voltage predrive elements. Decode L18, L28, and L29
XNVK0-XNVK3	X negative voltage predrive elements. Decode L30 and L31. XNVD0-7 and XNVK0-3 form a matrix for the X negative voltage predrive system
XPCD0-XPCD3	X positive current predrive elements. Decode L19 and L25
XPCK0-XPCK3	X positive current predrive elements. Decode L26 and L27. XPCD0-3 and XPCK0-3 form a matrix for the X positive current predrive system
XPVD0-XPVD7	X positive voltage predrive elements. Decode L18, L28, and L29
XPVK0-XPVK3	X positive voltage predrive elements. Decode L30 and L31. XPVD0-7 and XPVK0-3 form a matrix for the X positive voltage predrive system
Y, NY	Current direction control signals for the Y selection. Y is true if the sum of L22, L23, and L25 is even. NY is true if the sum is odd

This page left blank intentionally

3-84 POWER FAIL-SAFE

3-85 General

The Sigma 5 power fail-safe feature detects primary power application and primary power failure in the CPU and provides reset and interrupt signals to initiate startup and shutdown sequences at appropriate times. This feature also supplies power to execute the transfer of data during the interrupt operation. Power fail-safe sequences are initiated under the following conditions:

a. When power is initially supplied to the CPU.

b. When a complete power failure is detected.

c. When a short-term power failure is detected.

If power returns to an acceptable level, normal operation resumes automatically. During power fail-safe shutdown, information in certain volatile flip-flop registers is stored in core memory to prevent critical program data loss. When power is restored, the information in core memory is returned to the volatile flip-flop registers so that the program can resume at or near the interrupted point. Core memory serves as the storage device during power fail-safe operation since the cores are nonvolatile and retain information without the presence of power.

The power fail-safe feature is composed of two major components: the power fail-safe interrupts, which initiate the save and recovery programs, and the power monitor assembly, which monitors the primary power source.

3-86 Interrupts

When a power failure occurs, the power fail-safe feature notifies the CPU by means of a power-off interrupt. Sufficient energy is stored in the Sigma 5 power supply system to maintain dc power for the duration of a short power failure subroutine. When primary power resumes, a power-on interrupt causes the CPU to enter a recovery subroutine that restores the CPU to the state existing before the lapse of power.

The interrupt memory locations are X'50' for the power-on interrupt and X'51' for the power-off interrupt. The power-on interrupt is the highest priority interrupt in the system; power-off interrupt has second highest priority. Both of these interrupt levels are always enabled; they cannot be disarmed, disabled, inhibited, or triggered under program control.

3-87 Power Monitor Assembly

Figure 3-200 is a simplified block diagram of the power monitor assembly, a standard equipment item in Sigma 5, which consists of three standard modules: the WT21 regulator and independent power supply, the WT22 line detector, and the AT13 line driver.

The WT21 applies regulated dc voltages to the WT22 line detector and AT13 line driver. It also supplies unregulated voltages to the WT22 line driver.

The WT22 line detector performs the function of detecting a power failure and indirectly providing the necessary signals to the CPU for a startup or shutdown sequence.

The AT13 line driver is basically a cable driver used to drive the output signals of the WT22.

Although the primary power sources are optional, depending on user requirements, the primary power source shown in the simplified block diagram is single phase 120 Vac.

The application of primary power to the Sigma 5 system power supplies provides the power fail-safe feature with the voltage necessary to power the WT21, WT22, and AT13 modules. These standard dc voltages are provided by the internal power supply in the WT21 regulator. The power fail-safe feature receives 120 Vac and 60 Vdc power when primary power is applied. The 120 Vac power is transmitted to the WT21 regulator, which in turn is converted to regulated +4 Vdc, +8 Vdc, and -8 Vdc and unregulated +24 Vdc and +50 Vdc. These voltages are routed to the WT22 and AT13 circuits.

During three-phase operation, the 60 Vdc power output from the PT14 power supply is monitored directly to the WT22 line detector, which senses this input to determine whether it is within acceptable limits.

The requirements for a startup or shutdown sequence are governed by the WT22 line detector, which contains the basic sensing circuits within the power fail-safe feature. Detection of an out-of-tolerance voltage by the WT22 line detector generates the necessary logic signals to the AT13 line driver for a fail-safe shutdown. A subsequent return of voltage within tolerance generates the necessary logic signals to the AT13 line driver for a fail-safe startup.

REAL-TIME CLOCK. The real-time clock circuit on the WT22 module generates a stable clock frequency synchronized to the line frequency. This real-time clock is not an integral part of the power fail-safe feature and is located on the WT22 primarily for purposes of convenience.

INPUT REQUIREMENTS. Note that the input power source will vary according to user requirements. For information on the various power sources, refer to the section on Power Distribution.

THREE-PHASE INPUT DETECTION. When three-phase detection is required, one phase supplies power to the power monitor. The presence of three phases is detected by sensing the presence of a three-phase rectified but unfiltered 60 Vdc signal supplied by the PT14 power supply.

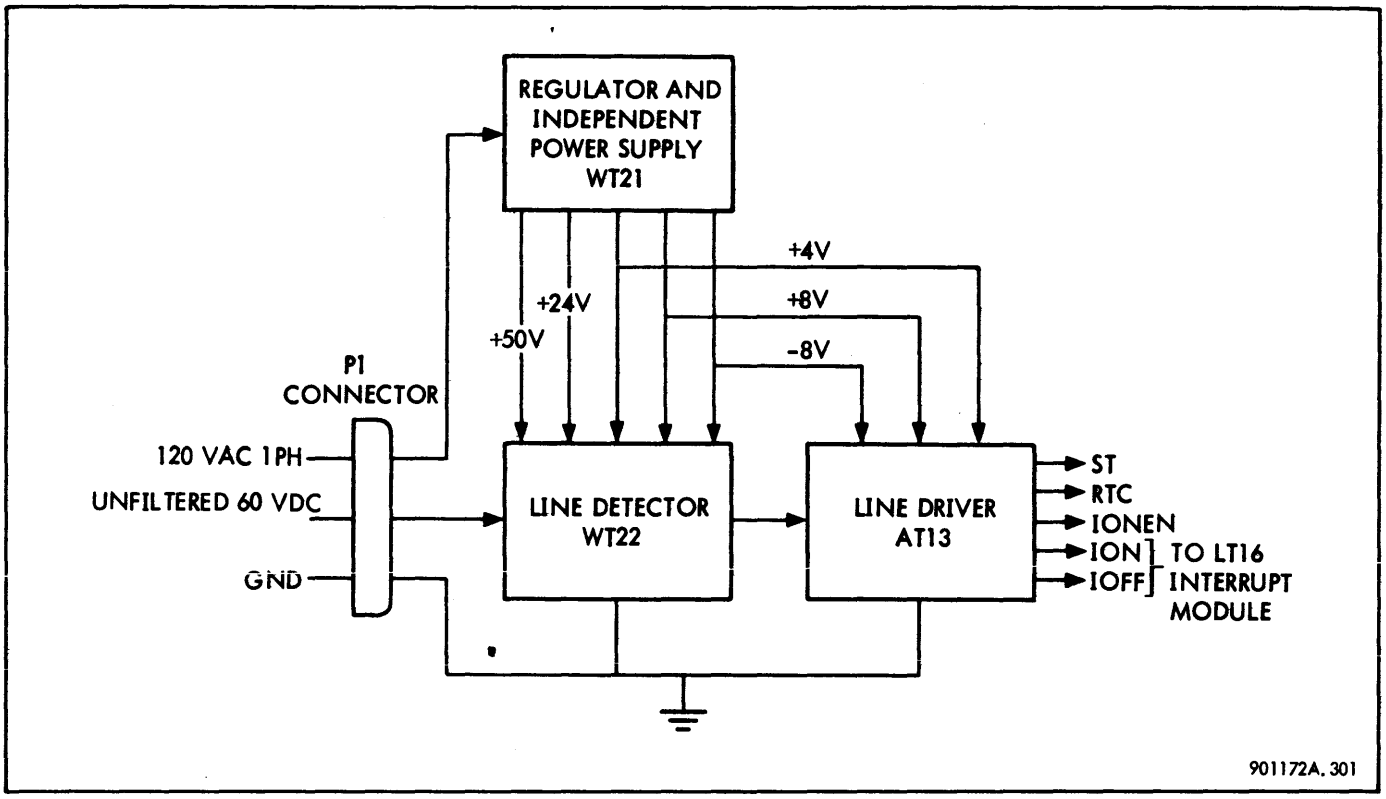


Figure 3-200. Power Monitor Assembly, Simplified Block Diagram

SINGLE-PHASE DETECTION. Single-phase detection is provided for by a simple rewiring in the power monitor. When rewired, this standard 110 Vac line is the only external input to the power monitor assembly.

INTERNAL POWER SUPPLY. The power monitor has its own internal power supply capable of delivering power to the WT21, WT22, and AT13 modules. This supply comes into operation when external power is applied. When power is shut off this internal supply outlasts the dc supplies in the computer, thereby keeping logic signals in their appropriate state as power decays and the power-off subroutine is executed.

PARALLEL OPERATION. The power monitor is capable of paralleling its output with the output of other power monitors. This is necessary, since several power monitors may be used to monitor individual lines and power supplies in a system. Therefore, if more than one power monitor is used in a given installation, the equivalent logic outputs of the power monitors are ORed together.

OUTPUT SIGNALS. There are five output signals from the AT13 line driver: ST, the master reset signal, ION, which initiates the startup sequence, IOFF, which initiates the shutdown sequence, IONEN, the ION enable signal, which performs an AND function for the output of the power monitor assembly, and RTC, the real-time clock signal, which is a clock synchronized to the line frequency, but is not used directly in the power fail-safe feature.

CIRCUIT DISCRIPTION. Figure 3-201 is a functional schematic of the power monitor assembly. Input power to the internal power supply on the WT21 regulator is shown to be single phase 110/120 Vac from pins 1 and 2 of the P1 connector. This input power is transmitted to transformer T1, which is in the internal power supply of the WT21 regulator package. Diodes CR1 through CR4 comprise a full-wave bridge rectifier and provide the dc inputs to the +8 Vdc and +4 Vdc regulator drivers. Diodes CR5 and CR6 act as a full-wave rectifier providing ac input to -8 Vdc regulator circuit.

WT21 REGULATOR. The WT21 (figure 3-202) is the voltage regulator-driver used to supply regulated dc to the WT22 power monitor and AT13 line driver. The WT21 contains a +8 Vdc regulator-driver, +4 Vdc regulator-driver, and -8 Vdc regulator. The +8 volt and +4 volt drivers are used with external pass transistors. The -8 volt regulator contains the pass transistor located on the module. The -8 volt regulator contains a rectifier circuit to allow operation from ac inputs at pins 14 and 18. The +8 and +4 volt regulator-drivers require dc input voltages. Two additional bridge rectifying circuits are located on the WT21 to provide 24 Vdc and 50 Vdc.

Filter capacitors for input filtering to the series regulators are not located on the module; however, provisions are made for external connections. Surge resistors are located on the WT21 to prevent damage to the external rectifiers that supply current to the +8 Vdc and +4 Vdc regulators.

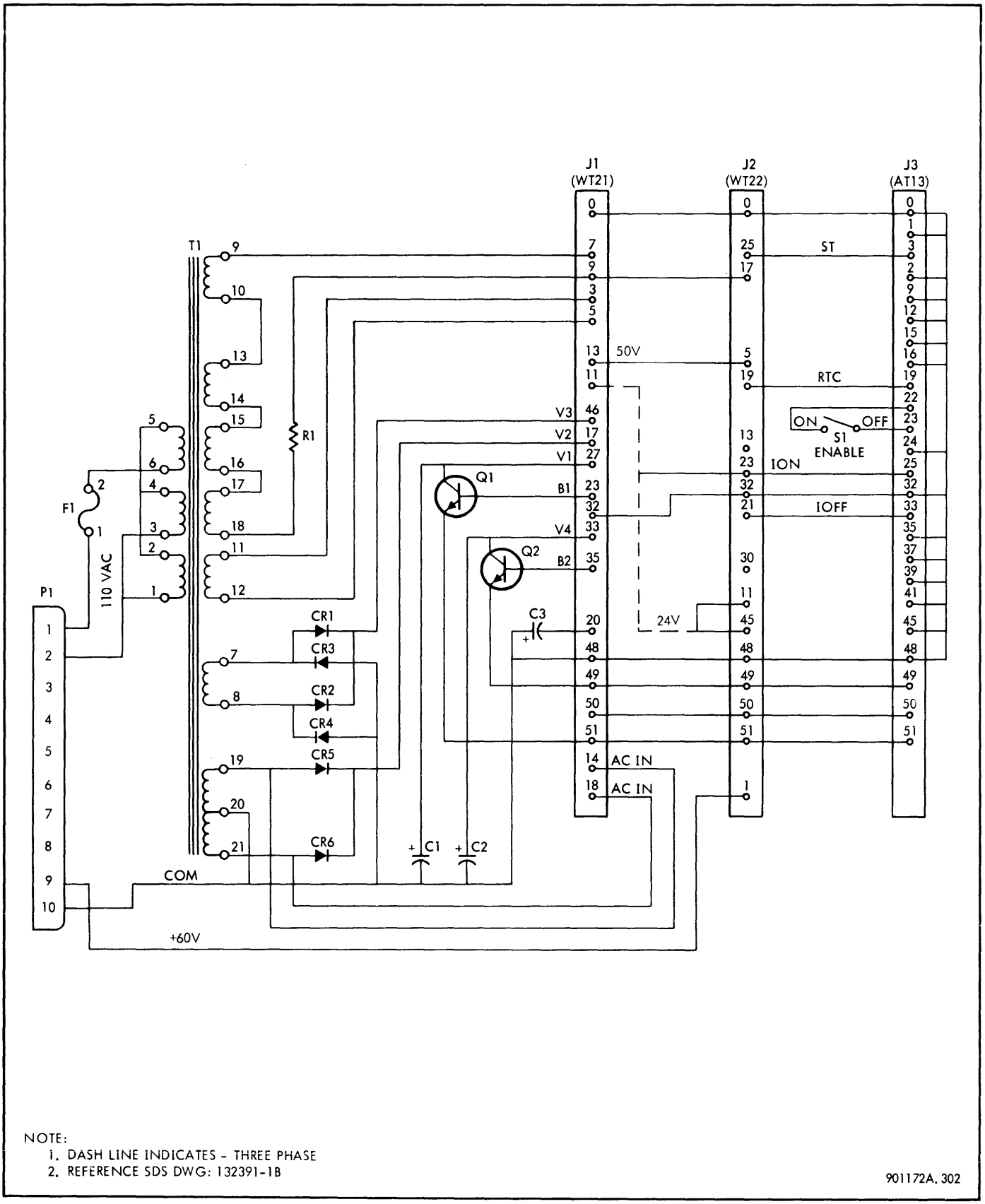
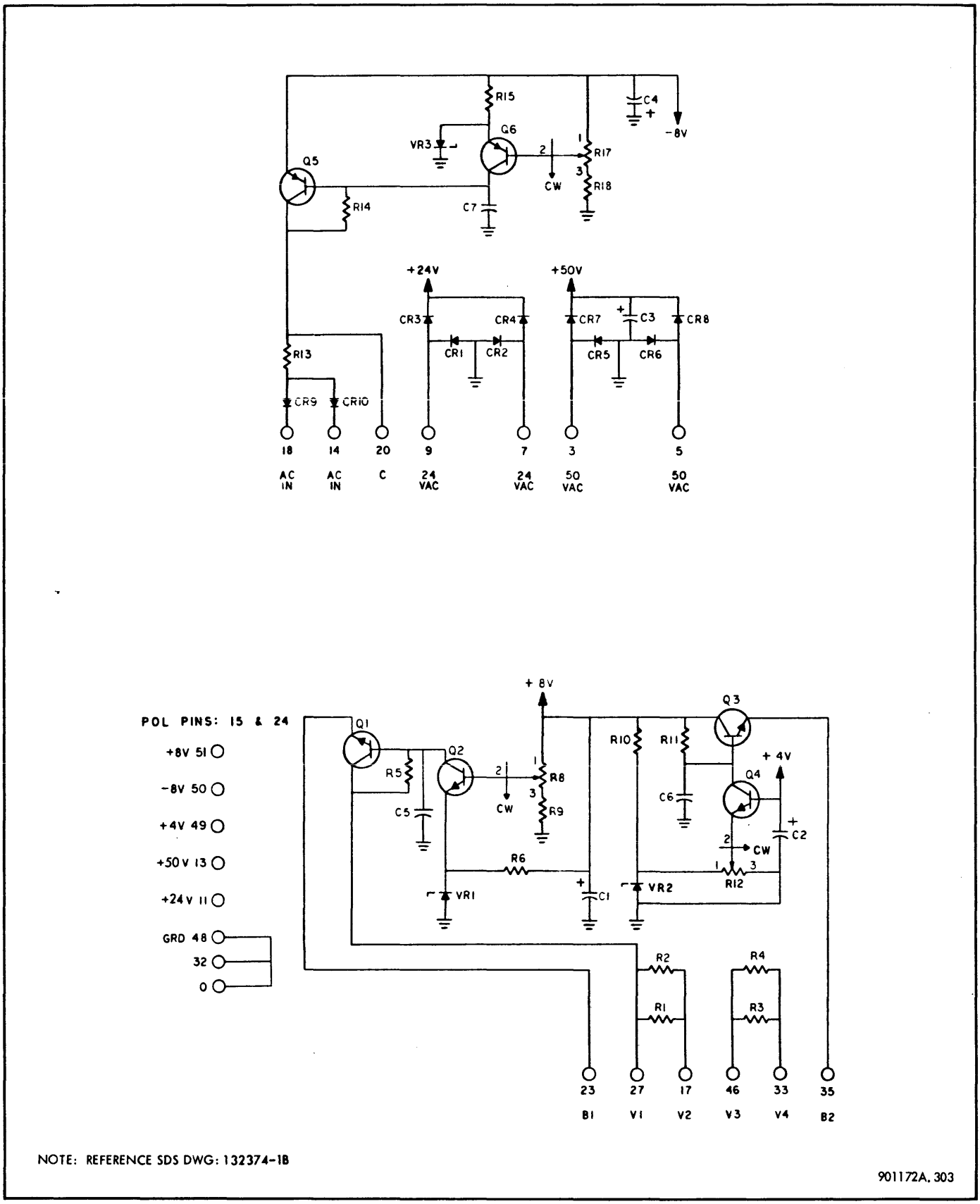


Figure 3-201. Power Monitor, Functional Schematic Diagram



NOTE: REFERENCE SDS DWG: 132374-1B

901172A.303

Figure 3-202. WT21 Regulator, Schematic Diagram

R1 through R4 are the surge protection resistors. CR1 through CR4 and CR5 through CR8 rectify the ac input voltages, which are then used to operate the power monitor with 24 Vdc and 50 Vdc.

+8 Vdc Regulator. Q2 and Q1 are the sense and drive transistors used in the +8 Vdc regulator. The input of the regulator is pin 17 (V2). Pin 27 (V1) is brought out to connect to an external filter capacitor. Voltage adjustment is accomplished by controlling the current in Q2. This current is determined by the emitter voltage, which is the reference voltage, and the sampled base voltage as adjusted by R8. The Q1 emitter output drives the external pass transistor.

+4 Vdc Regulator. Q3 and Q4 are the drive and sense transistors for the +4 volt regulator. Collector voltage to Q3 is derived from the +8 volt supply. Drive voltage to Q3 also comes from the +8 volts, therefore providing preregulation for the +4 volt regulator. Q3 drives a power transistor external to the WT21. The base of Q4 is connected to the +4 volt output. Current is controlled by changing the reference voltage, R12, at the emitter of Q4.

-8 Vdc Regulator. The -8 volt regulator, including pass transistor Q5, is located on the WT21. CR9 and CR10 provide a negative supply voltage when ac is applied to pins 18 and 14. Pin 20 is the common and external capacitor connection. R17 provides voltage control of the output.

WT22 LINE DETECTOR. Figure 3-203 is a block diagram of the WT22, figure 3-204 is the WT22 schematic, and figure 3-205 shows the WT22 waveforms. Basic timing and input power for the WT22 are as follows:

Delay time D	Adjustable from 5 to 20 ms (set @ 10 ms)
ION time A	300 ms ± 10%
Power failure detection time	< 3 ms
Input power	+8 Vdc @ 40 ma
	+4 Vdc @ 30 ma
	+60 Vdc @ 50 ma for 3Ø detection
	+22 Vdc @ 10 ma for 1Ø detection
	+50 Vdc @ 35 ma for 1Ø detection

The WT22 is the line detector module that provides all output signals for the power monitor. This is the basic unit within the power fail-safe feature. The principal function of the WT22 is to detect a power failure and provide the necessary reset and interrupt signals for the CPU. These signals initiate startup and shutdown sequences when power comes on and goes off.

Startup Sequence (see figure 3-204). When power is first turned on, the ST flip-flop is dc set by V1 (+8 Vdc), charging C8, causing ST to go high. VD, the sampled voltage, is also applied and charges C4 through R13. The voltage across C4 is determined by the magnitude of VD through R13, as well as the time constant R13-C4. This time constant determines the time after power is applied that the threshold sensing circuit will trigger the ION pulse. As shown in figure 3-205, the occurrence of ION is time A, or the approximate time necessary for all dc power supplies in the computer to stabilize. The ION pulse resets the flip-flop and ST falls to 0. If ST is 0, the reset of the flip-flop is high and prevents C4 from charging by holding the NOR gate on.

Shutdown Sequence. When the line detection circuit indicates power failing it generates the IOFF signal. The IOFF signal is delayed by the period D shown in figure 3-205. D is the approximate maximum time dc power supplies will remain within regulation after a power failure. IOFF is applied to the clock input of the ST flip-flop, and since the set is held high, the flip-flop sets when IOFF returns to 0, which is at the conclusion of period D. IOFF pulses will continue to be generated as long as power is below the acceptable threshold level.

IOFF pulses and NST prevent C4 from charging in case of a short power interrupt, as shown in figure 3-205. As long as IOFF produces a pulse, C4 will discharge, preventing an ION pulse until C4 charges up again. During this time ST is held high by IOFF, setting the flip-flop continuously.

ST will go false when ION goes true and there are no IOFF pulses present. This means that any time an IOFF pulse occurs, the entire startup sequence will take place. When power returns, the line detection circuit will generate IOFF pulses whenever the line voltage drops below threshold. Threshold oscillation is prevented by a preset hysteresis. IOFF and ION, as determined by their respective threshold settings, may be set 10v apart; for example, IOFF will be present at 80v line and ION will occur at 90v line. ST will go high when the line drops below 80v and will remain high until the line raises above 90v.

Real-Time Clock Signal (RTC). In addition to the ST, ION and IOFF signals, the WT22 generates a real-time clock pulse (RTC) synchronized to the line frequency. By selecting the 1F or 2F term on the module this pulse will be at the line frequency or twice the line frequency.

Circuit Description. The line detection circuit is that part of the WT22 module which detects an ac line failure. The detection scheme is slightly different for single phase and three phase operation; however, both phases are detected by the WT22.

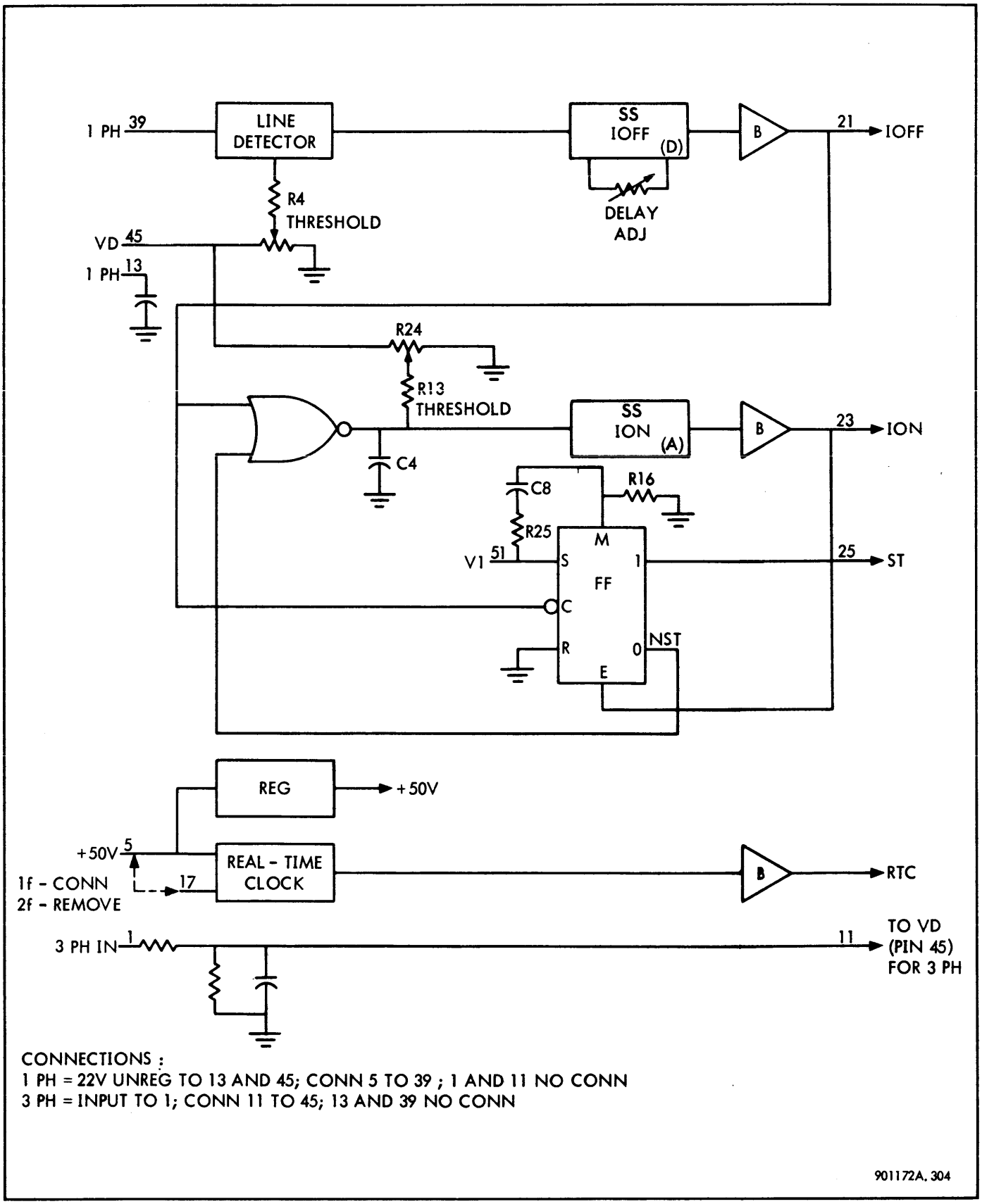
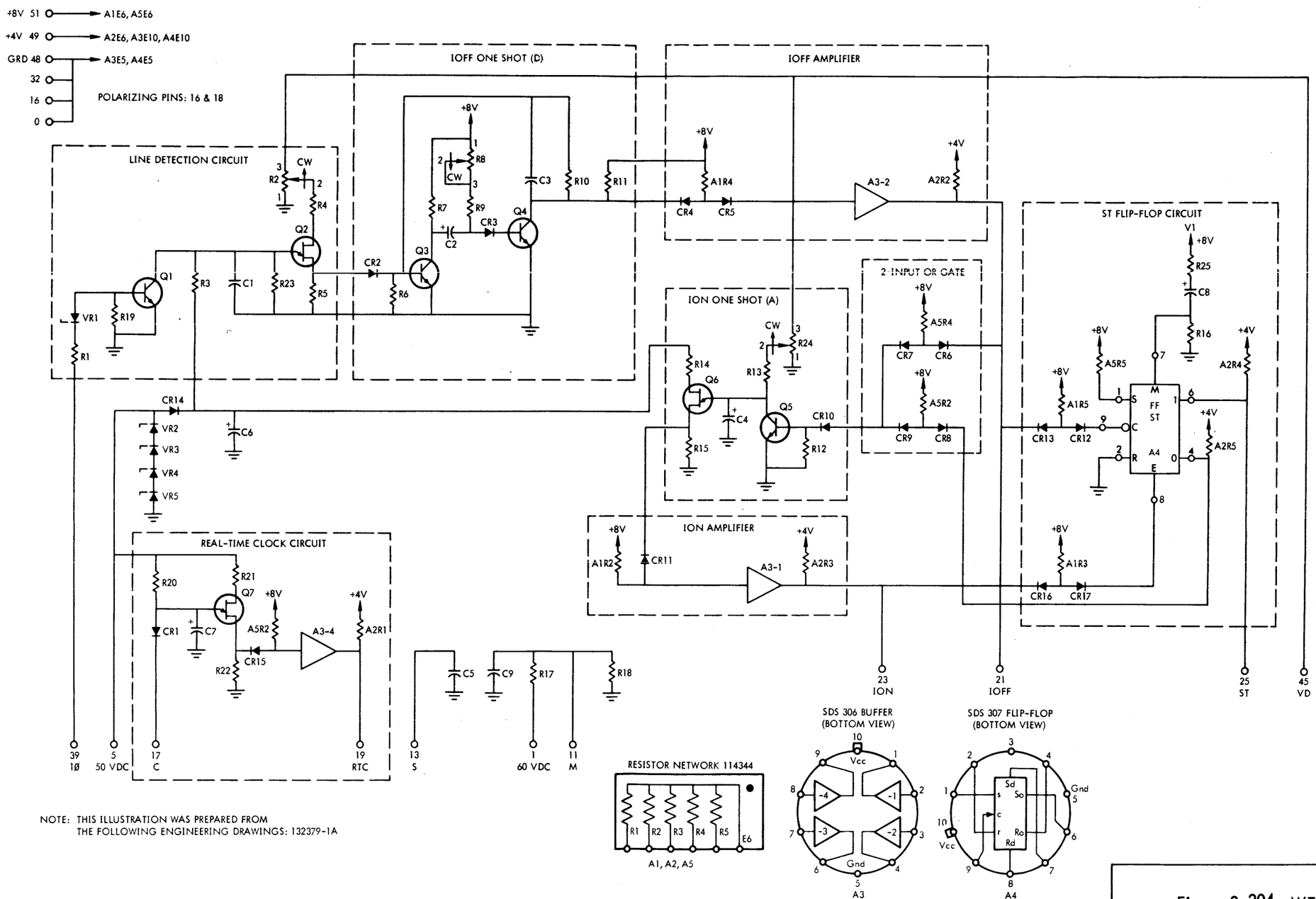


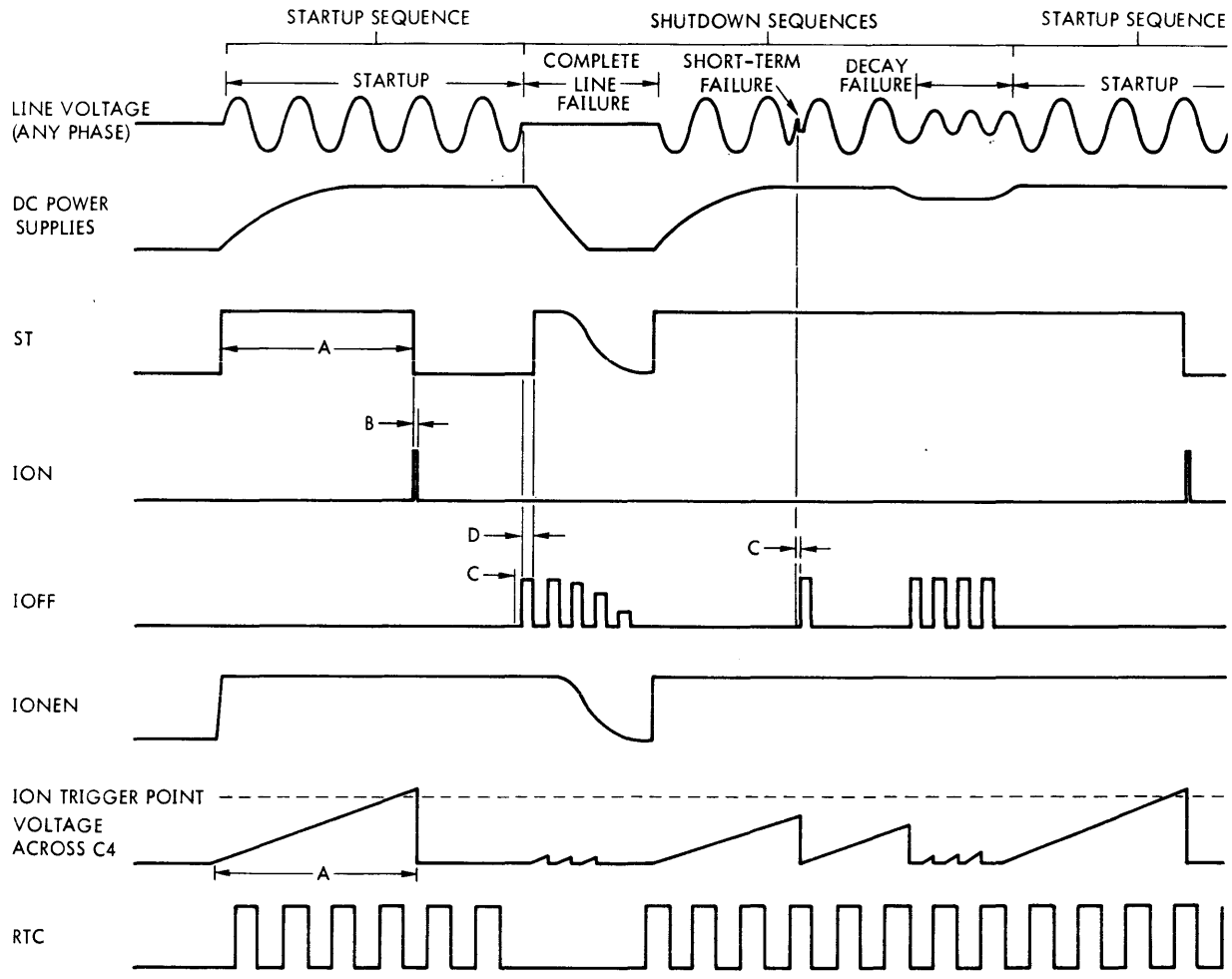
Figure 3-203. WT22 Line Detector, Block Diagram



NOTE: THIS ILLUSTRATION WAS PREPARED FROM THE FOLLOWING ENGINEERING DRAWINGS: 132379-1A

Figure 3-204. WT22 Line Detector, Schematic Diagram

901172A.305



LEGEND

A. INITIAL TIME FOR DC SUPPLIES TO STABILIZE IS APPROXIMATELY 300 MS
B. 5-100 μSEC AFTER ST FALLS
C. RESPONSE OF IOFF AFTER COMPLETE LINE FAILURE <1.0 MS
D. TIME BETWEEN IOFF AND ST OR MAXIMUM TIME POWER SUPPLIES WILL REMAIN WITHIN SPECIFICATIONS AFTER A POWER FAILURE IS 5 TO 20 MS

901060A.31105

Figure 3-205. Power Fail-Safe Waveforms

Single-phase operation is shown in figure 3-206. C1 cannot charge to level Vp if an ac signal is present at Ein, the single-phase input. Ein is generated by an unfiltered dc signal that is clamped to provide a steep rise at the zero crossing. This rise time determines the minimum response time of the IOFF pulse. C1 must charge to Vp, and Vp is determined by setting potentiometer R2. The time constant of R3C1 (T2) is longer than T1 as the voltage charges to Vp. In addition, the base voltage, VD, is derived from an unregulated source so it will decrease with line voltage, causing Vp to be at a lower point, Vp = n VBB where n is 0.7 and Vp is the firing point of the unijunction transistor Q2. If power drops out completely, V_B decreases immediately and T1 then equals or exceeds T2 and triggers Q2 in less than one-quarter of a cycle. If power goes down slowly below threshold, Q2 will fire at a worst-case condition of one-half cycle caused by the decrease in VBB; however, this is only if power drops slowly below threshold and is not a worst-case condition. This happens because in this case the power supplies will take longer to come out of regulation.

For three-phase operation the threshold is set within the dc range of the multiphase signal, as shown in figure 3-207. This unfiltered signal supplies the VBB source voltage in three-phase operation. If any phase falls below threshold the unijunction transistor will trigger. Since the voltage is now sampled at six times the line frequency, response time will be faster than in single-phase operation. The additional transistor across C1 is not used and is therefore disconnected in three-phase operation by selection of the proper input connections to the WT22 module.

Pin Connections. For single-phase detection, connect pin 5 to pin 35; connect pin 13 to pin 45. For three-phase detection, connect pin 11 to pin 45.

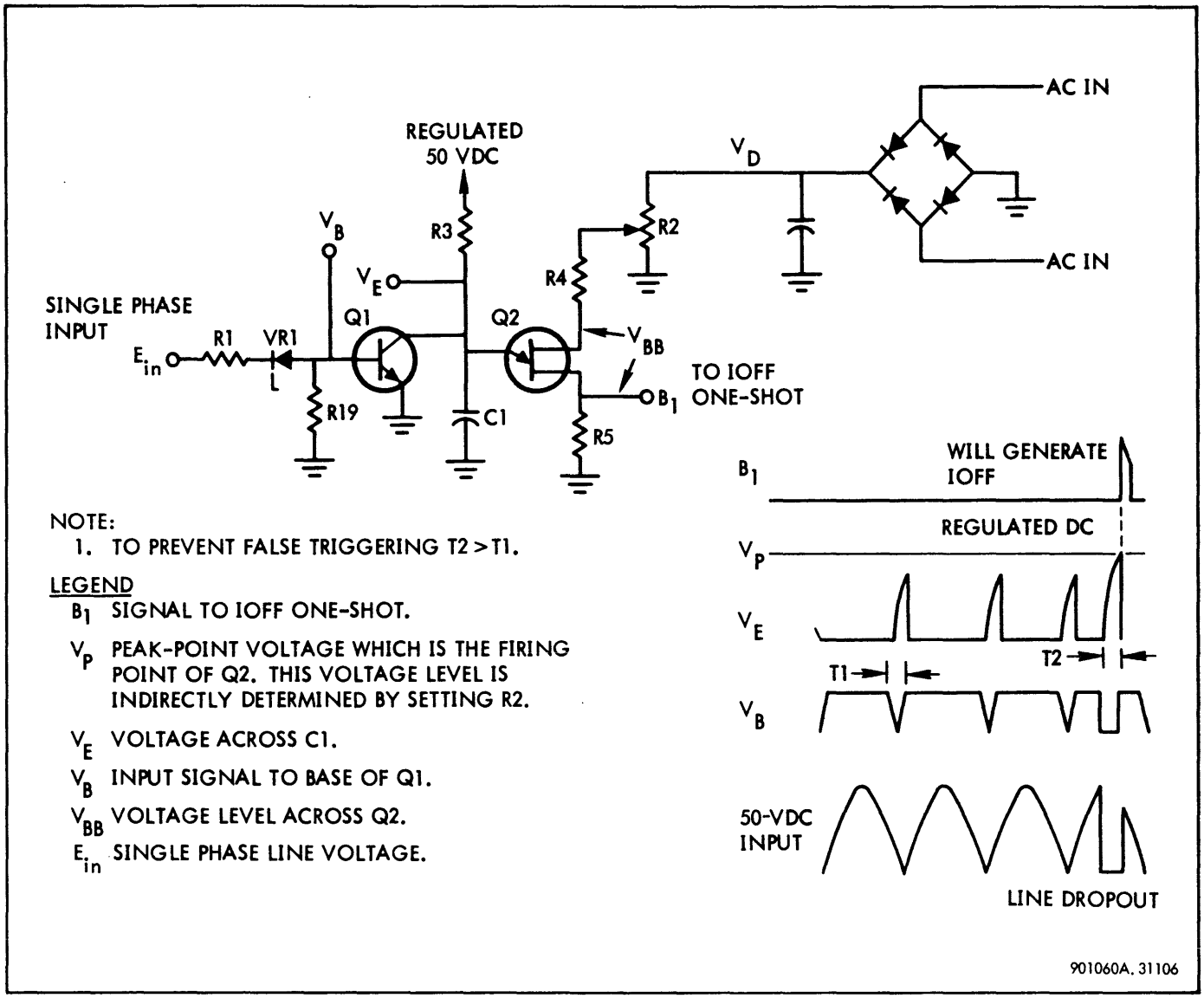


Figure 3-206. Single-Phase Detection

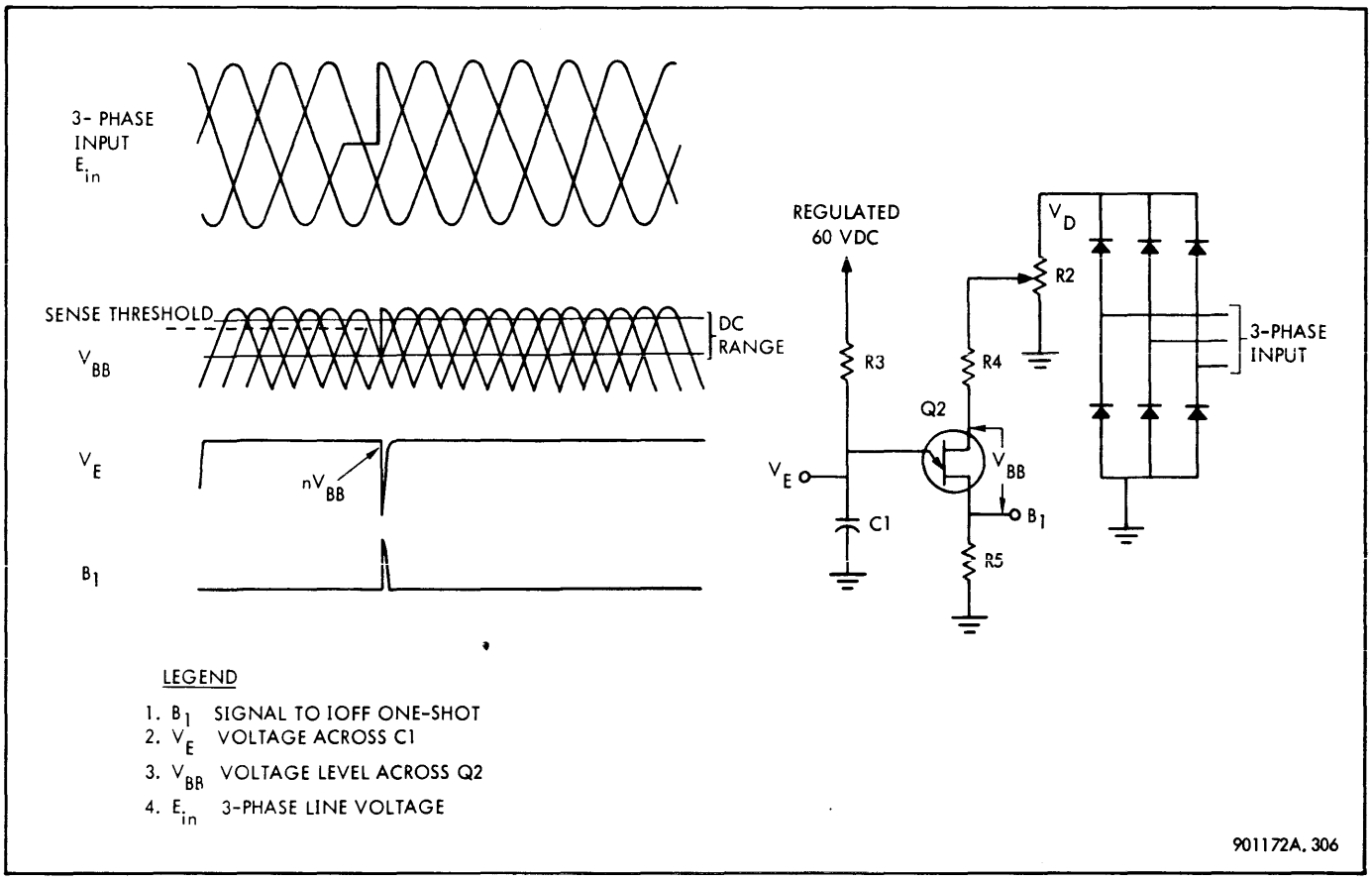


Figure 3-207. Three-Phase Detection

The ION one-shot is used to generate the ION pulse. This occurs when there are no IOFF pulses and the ST flip-flop is reset with ST low if the line is above the ION threshold. The one-shot, shown in figure 3-208, consists of a unijunction transistor threshold circuit, an inverter, and a 2-input NOR gate. If pulses are applied to D, the emitter voltages will not reach Vp on Q6; likewise, if R is positive Q5 will conduct and C4 will not charge. This provides the inhibit function of the one-shot. C4 will charge only if power is on and no IOFF pulses are present, which is the startup routine. When Q6 fires it produces a pulse across R15 that is used to generate ION.

The real-time clock puts out pulses at the line frequency or twice the line frequency, as shown in figure 3-209. Q7 derives its interbase voltage VBB from a clamped, high voltage, unfiltered, dc source. R20C7 is set to be longer than one cycle. As C7 charges, VBB suddenly reduces, and Q7 fires when Vp = n VBB and produces a pulse at B1. If single frequency pulses are required, Q7 must not fire every half cycle. C7 is prevented from charging by diverting the current through R20 through CR1 every other cycle.

POWER MONITOR LOGIC. There are five power monitor logic signals put out by the AT13 logic module. The signals and their cable pins are as follows:

Signal	Cable Pin
ST	04
RTC	07
IONEN	08
ION	09
IOFF	10

The ION and IOFF signals are input to the LT16 interrupt module.

Startup Routine. The following steps outline the logic signals that make up the power monitor assembly startup routine.

a. ST is the master reset signal that is true during the time when power on-off transitions are occurring. When power is applied this signal comes true as soon as possible (determined by the internal power supply). ST remains high initially as the power supplies in the computer stabilize. This time is determined by ION occurring.

b. ION occurs only if the line voltage is above a preset level, which is the ION threshold. ION is then generated approximately 900 ms after power is turned on.

When ION occurs, ST falls to zero. ION should outlast ST by more than 2 μsec but less than 100 μsec.

c. IONEN is a true signal as long as a power monitor is operating. This signal is necessary only when using more than one power monitor per system. It is available on an AT13 cable driver-receiver where the receivers and drivers are connected externally. This signal from a driver of one power monitor connects to the receiver of another, thus cascading the signals. IONEN then becomes an AND function which will only be true if all power monitors are operative.

As shown in figure 3-201, the IONEN switch, S1, is left open if only one power monitor is used in a system. If more than one is used, the IONEN switches are closed on all power monitors except the first switch in the cable scheme. This will be the IONEN switch closest to the cable terminator, and it is always left open. With S1 open, the IONEN signal will be high as long as primary power is applied to the power monitor. IONEN is ANDed with ION to produce the PON signal for the LT16 interrupt module so that the power-on interrupt subroutine can be initiated.

RTC (Real-Time Clock). A clock pulse that is jitter-free and synchronized to the line frequency is one of the outputs. This output is arranged so that one RTC signal will not be paralleled with other RTC signals by the interconnecting cables. One of the isolated receiver-drivers on the AT13 is used for this purpose. This precaution is necessary since these RTC signals may be on different phases of the line.

Shutdown Routine. The following steps outline the logic signals that make up the power monitor assembly shutdown routine.

a. IOFF is a signal that sets an interrupt channel indicating to the CPU that the line voltage is below a preset threshold. This interrupt initiates a shutdown subroutine that stores all volatile data into core storage before the master reset signal, ST, causes a cessation of memory operations. The IOFF pulse should be greater than 2 μsec, but less than 20 ms. The delay between a power failure and the IOFF pulse going true should be minimized (less than 2 ms for single phase, less than 1 ms for three phase).

b. ST will go true, after a delay time, when power fails. This delay time is determined by the amount of time it takes for the external dc supplies in the computer to fall below their specified tolerances. This delay time or the time between IOFF occurring and ST going true should be adjusted to as long a duration as possible to allow maximum time to store data before shutting down the input to the memory. The delay is adjustable between 5 and 20 ms; however, it is set at 10 ms.

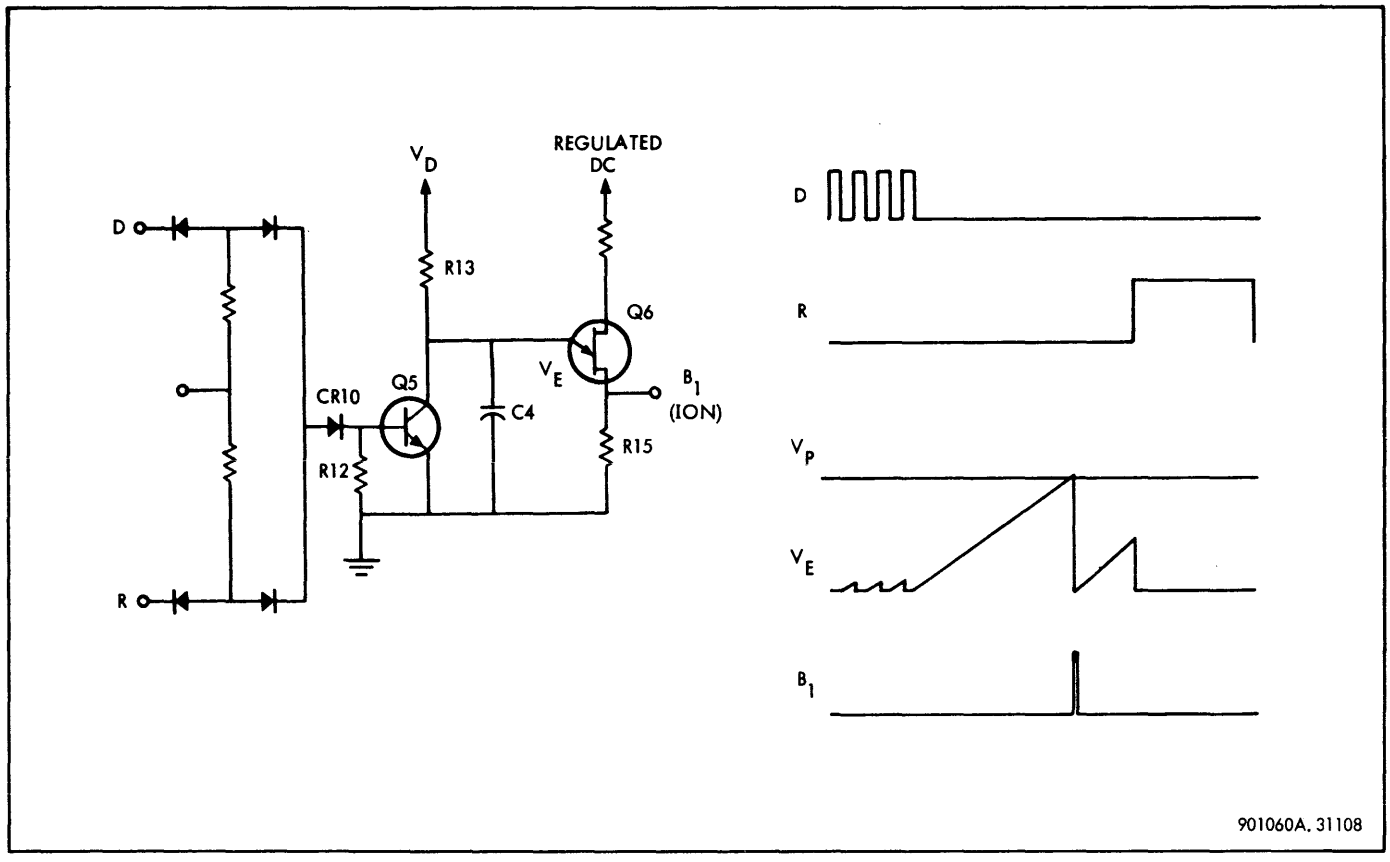


Figure 3-208. ION One-Shot Operation

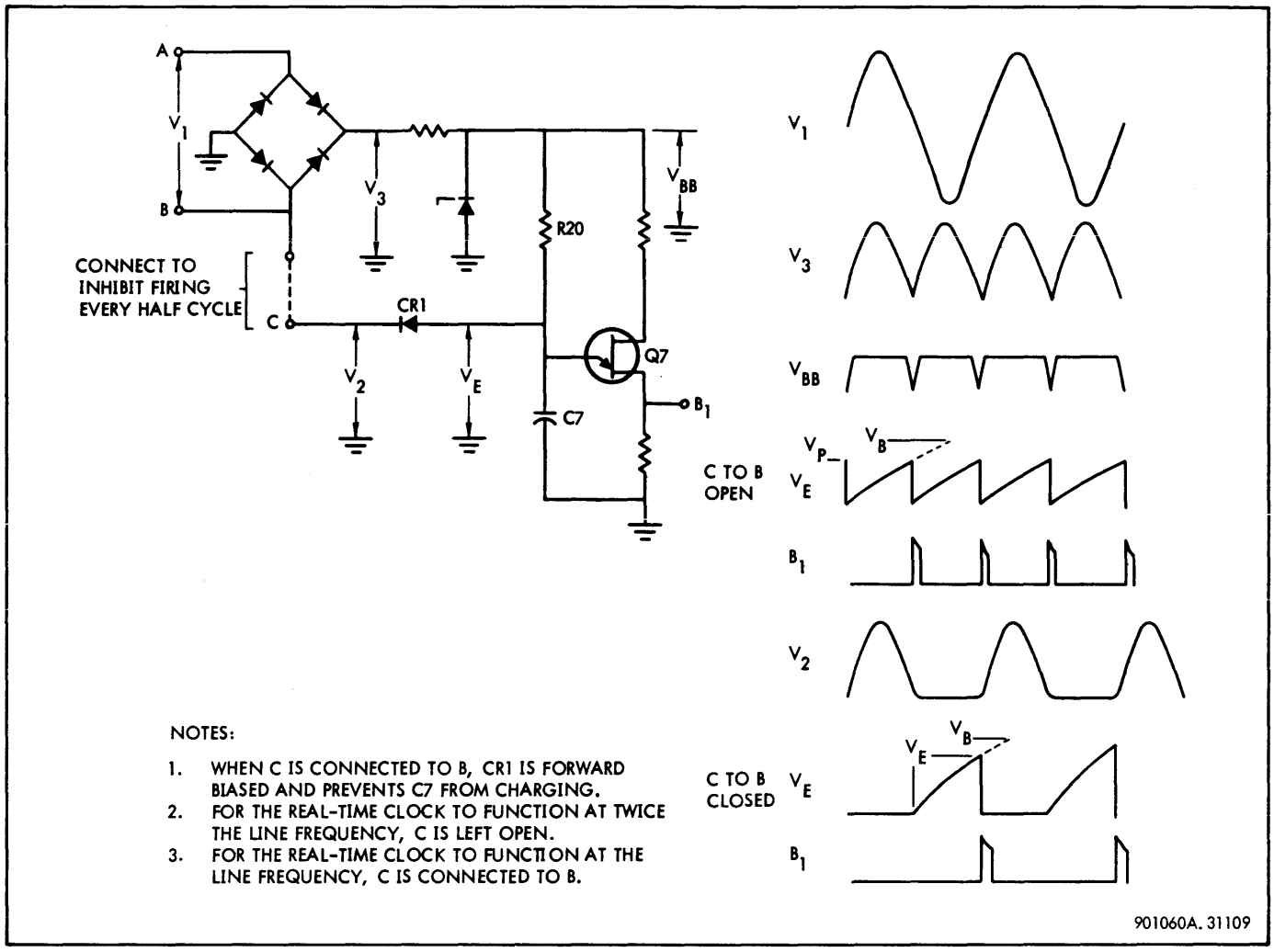


Figure 3-209. Real-Time Clock Operation

3-88 FLOATING POINT UNIT

The floating point unit consists of five registers, an adder, and control logic for the execution of the following Sigma 5 instructions:

Floating Add Short, code 3D

Floating Add Long, code 1D

Floating Subtract Short, code 3C

Floating Subtract Long, code 1C

Floating Multiply Short, code 3F

Floating Multiply Long, code 1F

Floating Divide Short, code 3E

Floating Divide Long, code 1E

The floating point registers are similar to the CPU arithmetic registers with the following exceptions:

a. The floating point registers contain 25 or 26 additional flip-flops.

b. Two additional registers, the E- and F-registers, are included to handle exponents.

c. Hexadecimal shift logic is included for normalizing.

A block diagram of the floating point unit is shown in figure 3-210. The floating point unit and the CPU communicate by means of 32 bidirectional data lines, FP0 through FP31, and by control signals. The CPU transmits the following control signals to the floating point unit:

FS (floating significance)

NFZ (not floating zero)

FNF (floating normalize)

R31 (bit 31 of R-register)

O2, O6, O7 (bits 2, 6, and 7 of O-register)

FPCON (enables connection of floating point unit)

FPDIS (enables display of floating point register on PCP)

Signals from the floating point unit to the CPU are:

N(S/CC1/FP), N(S/CC2/FP) (to set condition code flip-flops)

NFPRR (not floating point result ready)

Clock signals for the floating point unit flip-flops are derived from CPU delay line 1. The clocks are designated CLFP/1 through CLFP/2 in the CPU and are generated at the same time as the CPU ac clock signals. The floating point clock signals are enabled by signals FPCLEN/1, FPCLEN/2, and NCROSCL.

The functions of the floating point registers are described in the paragraphs below. The detailed functions of the registers during instruction execution are described in the floating point instruction sequence charts.

3-89 A-Register

The A-register is used to hold the augend during addition and subtraction, the multiplier and then the product during multiplication, and the numerator and then the quotient during division. Left shifting for normalizing takes place in the A-register four bits at a time.

The inputs to the A-register and their enabling signals are shown in figure 3-211.

3-90 B-Register

The B-register holds the multiplier in reverse order and then the product during multiplication, and receives the quotient during division. This register also serves as a counter during postnormalizing.

The inputs to the B-register and their enabling signals are shown in figure 3-212.

3-91 D-Register

The D-register holds the addend and then the result in addition and subtraction, the multiplicand in multiplication, and the denominator in division. The inputs to the D-register and their enabling signals are shown in figure 3-213.

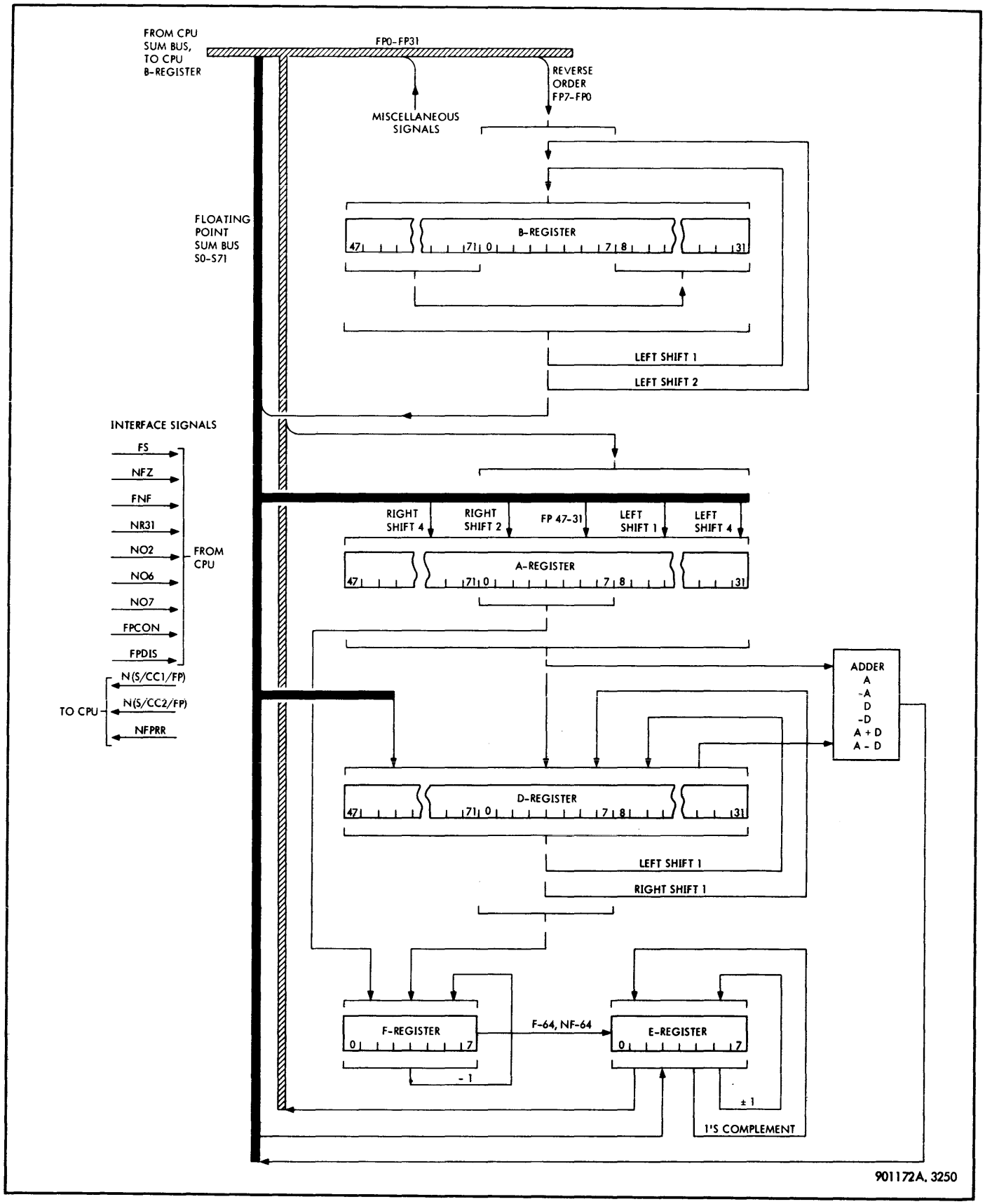


Figure 3-210. Floating Point Unit, Block Diagram

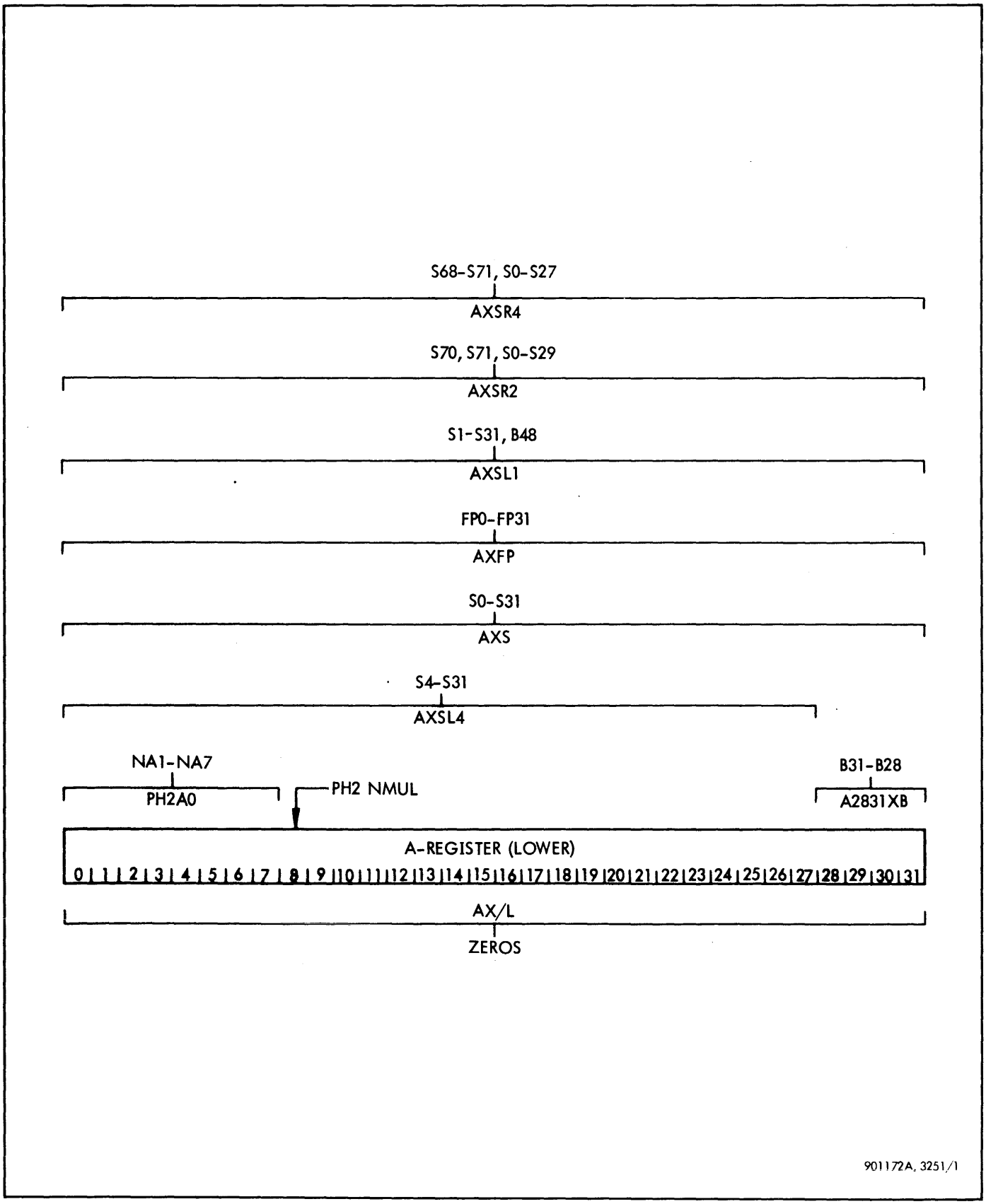
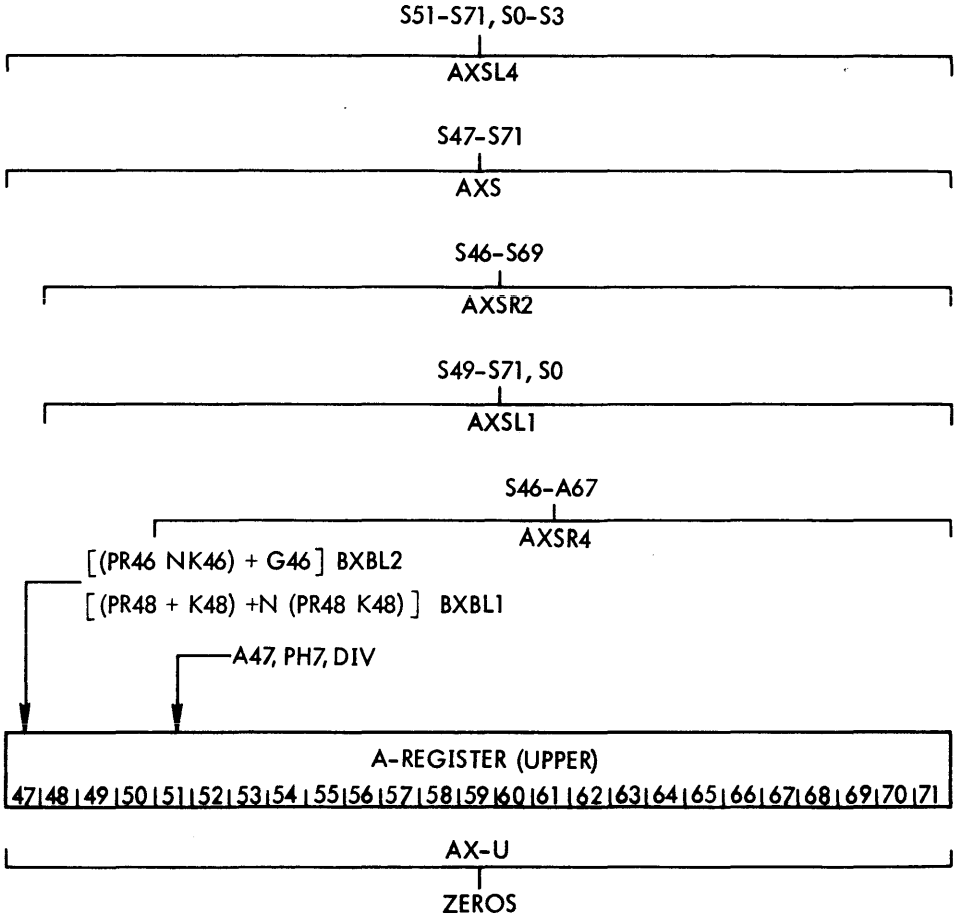


Figure 3-211. Floating Point A-Register Inputs and Enabling Signals (Sheet 1 of 2)



901172A.3251/2

Figure 3-211. Floating Point A-Register Inputs and Enabling Signals (Sheet 2 of 2)

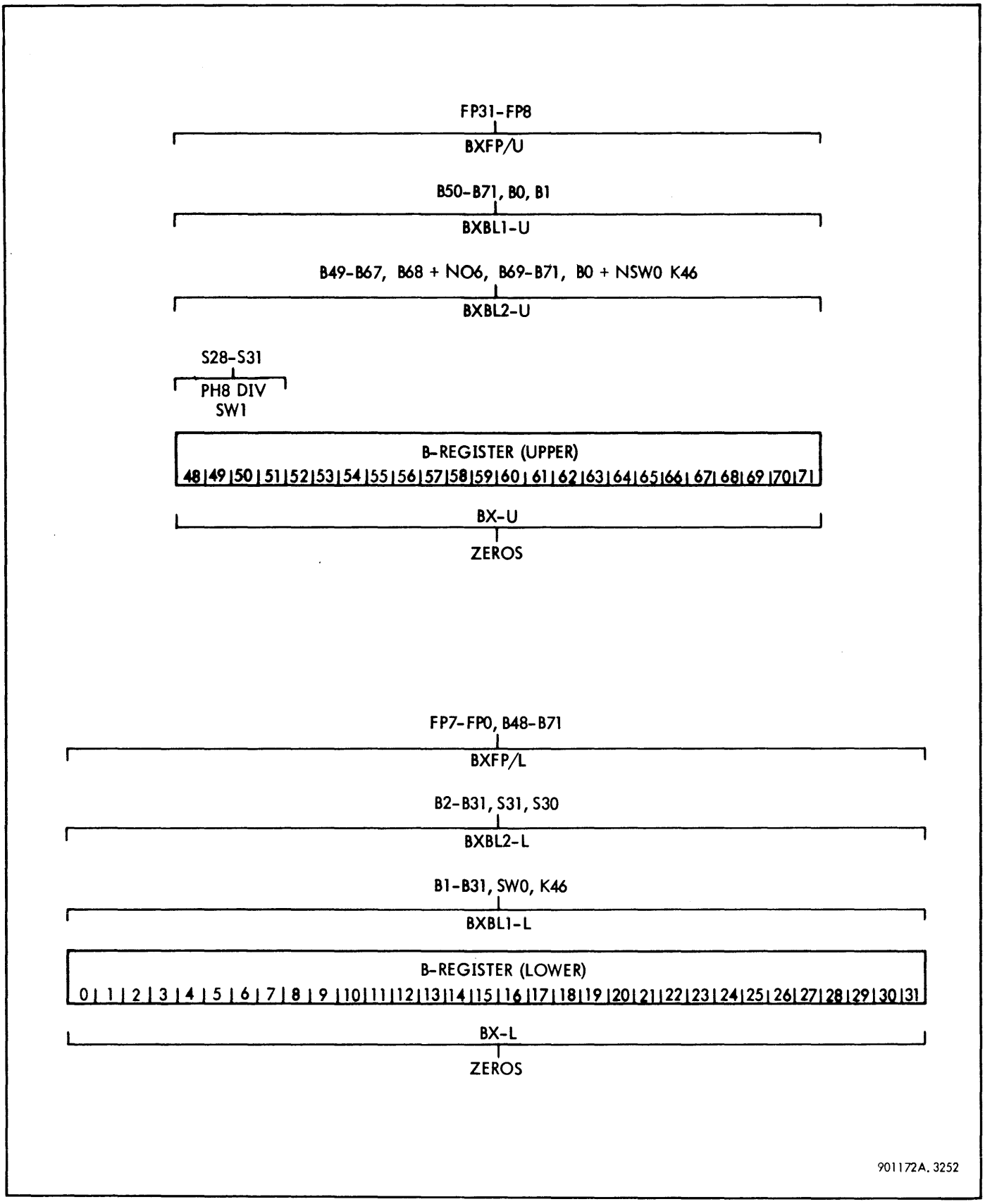


Figure 3-212. Floating Point B-Register Inputs and Enabling Signals

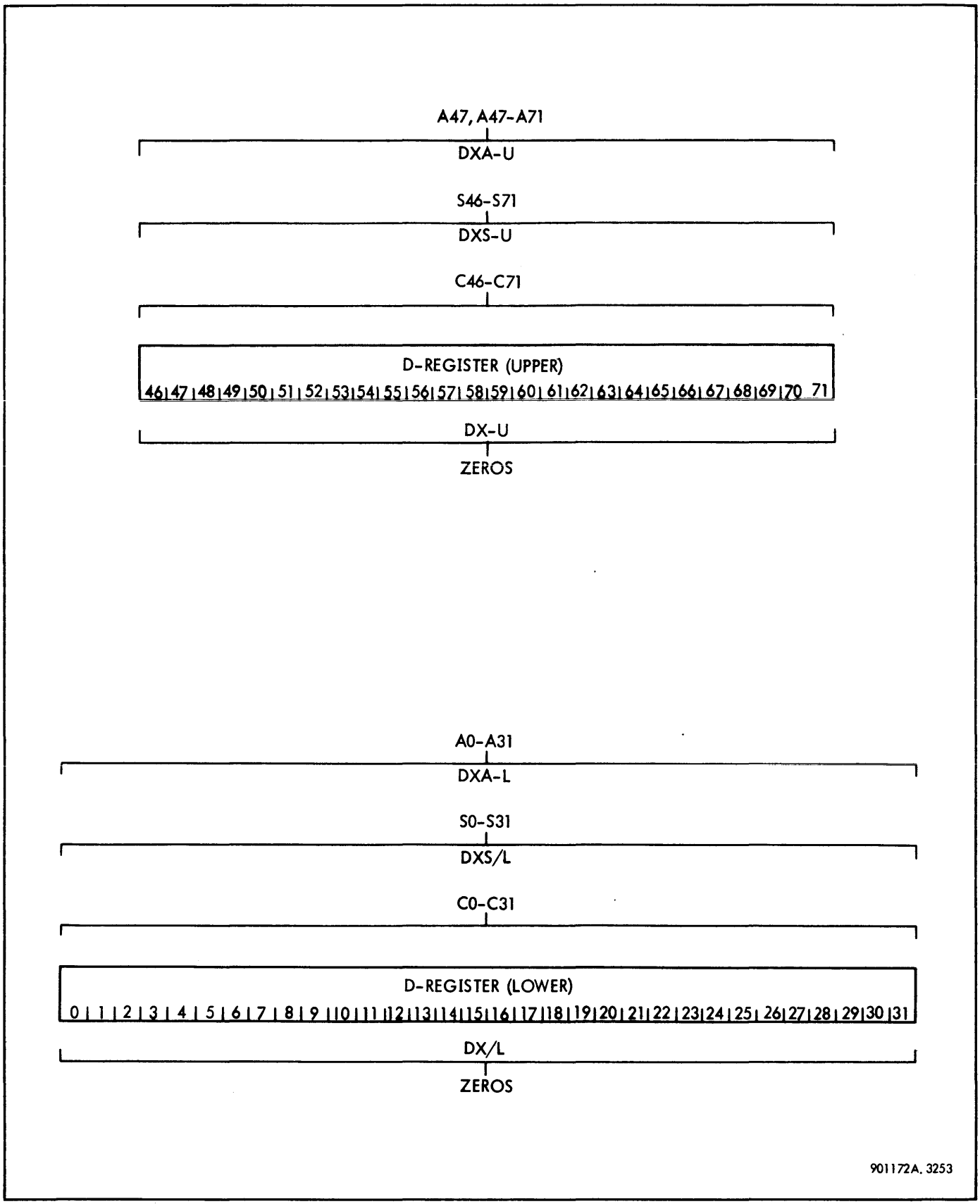


Figure 3-213. Floating Point D-Register Inputs and Enabling Signals

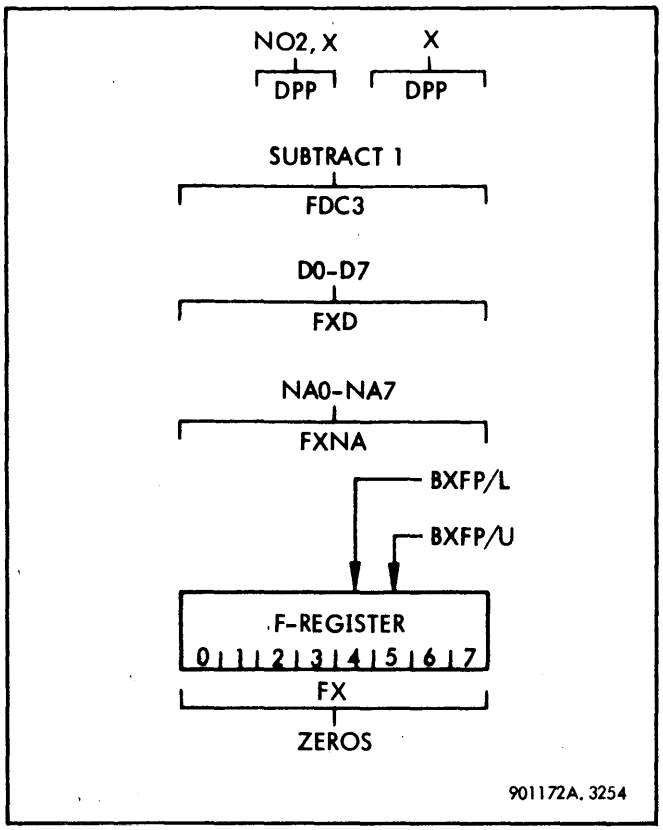


Figure 3-214. Floating Point F-Register Inputs and Enabling Signals

3-92 F-Register

The F-register is used as an exponent buffer during floating addition and subtraction while the E-register is involved in pre-alignment logic. In multiplication and division, the F-register is used as an iteration counter, subtracting one from the count with each iteration. The inputs to the F-register and their enabling signals are shown in figure 3-214.

3-93 E-Register

The E-register receives the unbiased exponent for all floating point operations and is used as an alignment counter during addition and subtraction. The inputs to the E-register and their enabling signals are shown in figure 3-215.

3-94 Adder

The adder in the floating point unit operates in the same manner as the adder in the arithmetic and control unit. The same type of adder preset terms are used; for example, S/SXA to transfer the contents of the A-register to the sum bus, and S/SXAPD to add the contents of the A- and D-registers. Two preset terms not found in the CPU adder are used in the floating point adder: S/SXAuA to place the absolute value of the A-register contents on the sum bus, and S/SXAuD to place the absolute value of the D-register contents on the sum bus.

The preset terms set repeater flip-flops as in the CPU adder. These flip-flops produce propagate terms PRXAD, PRXAND, PRXNAD, and PRXNAND. Generate terms GXAD and GXAND and carry terms K0 through K71 are also developed as a result of the preset logic, as in the CPU adder. The propagate, generate, and carry signals are combined in a parallel adder configuration to place on the sum bus the results of the adder functions shown in figure 3-210.

3-95 Floating Point Display

The contents of the floating point registers may be displayed in the CPU DISPLAY indicators by placing the REGISTER SELECT switch in the EXT position and operating switches on the ST14 toggle switch module in location 6A in the floating point unit. To display the contents of registers A, B, D, the sum bus outputs, or a set of miscellaneous signals, S1-1 through S1-5 on the switch module are set as shown in table 3-96. Switches S1-1 through S1-5 are the five switches on the front of the module, S1-5 on the top and S1-1 on the bottom. The information displayed in the miscellaneous (FPXMISC), sum bus lower (FPXSL), and sum bus upper (FPXSU) positions is shown in figure 3-216.

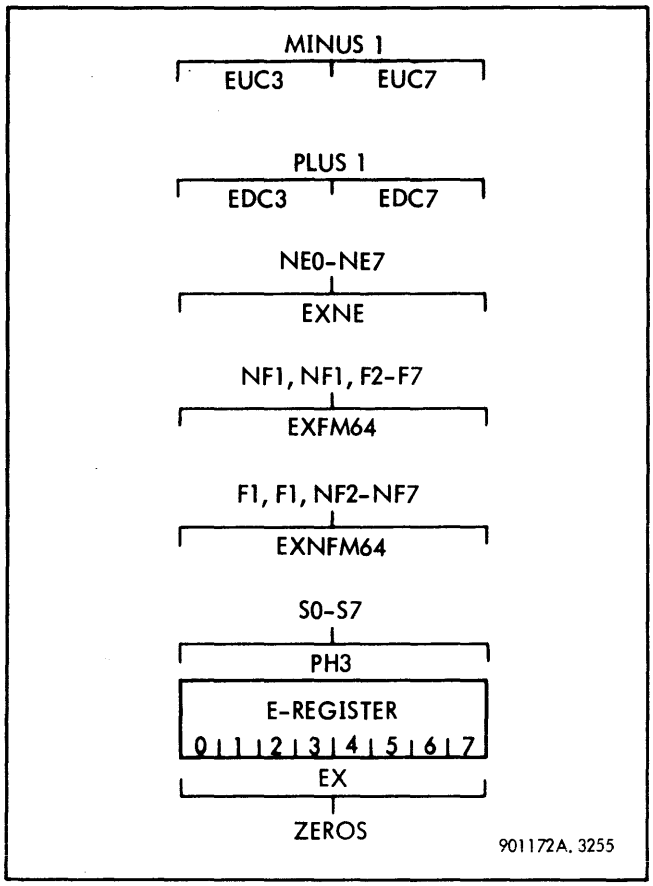


Figure 3-215. Floating Point E-Register Inputs and Enabling Signals

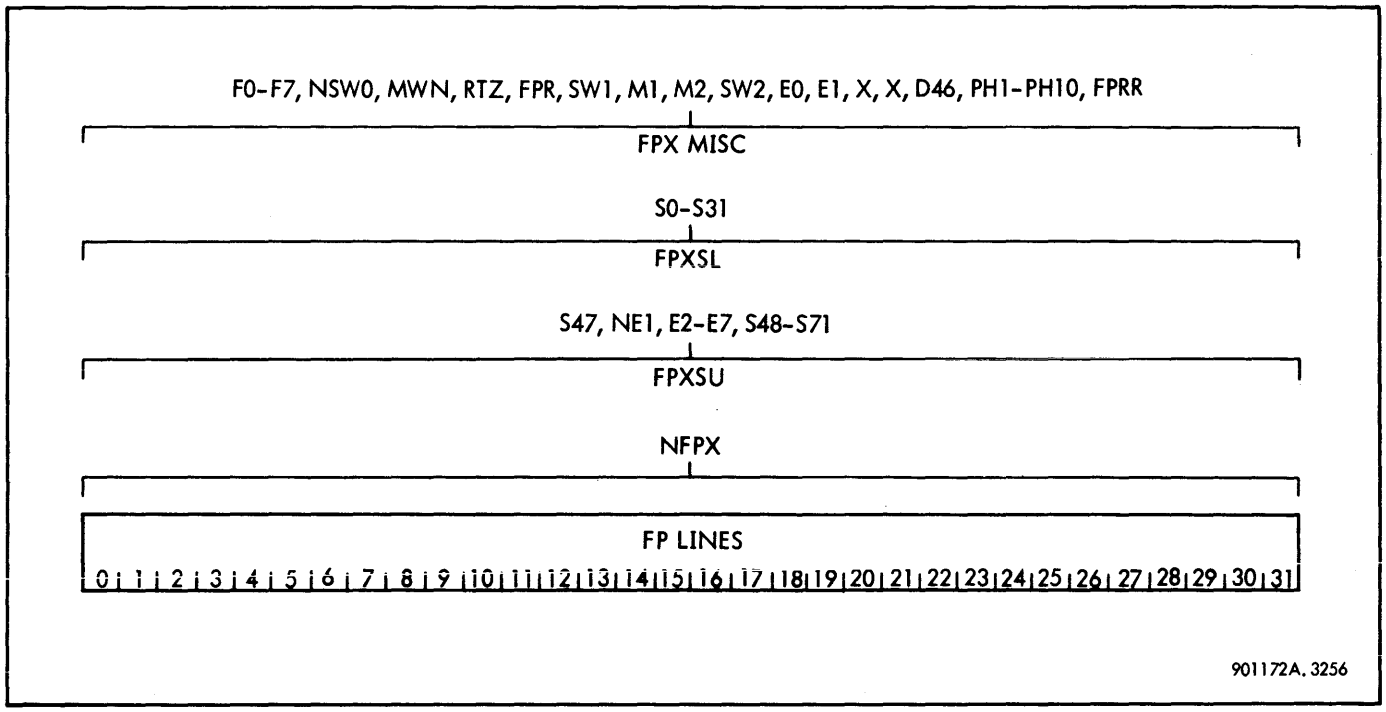


Figure 3-216. Data on Floating Point Lines and Gating Terms

Table 3-96. Switch Positions for Floating Point Information Display

SWITCH POSITIONS					INFORMATION DISPLAYED
S1-5	S1-4	S1-3	S1-2	S1-1	
Down	X	X	X	X	Miscellaneous
Up	Down	Down	Down	Down	Sum bus, lower
Up	Down	Down	Down	Up	Sum bus, upper
Up	Up	Down	Down	Down	A-register, lower
Up	Up	Down	Down	Up	A-register, upper
Up	Down	Up	Down	Down	B-register, lower
Up	Down	Up	Down	Up	B-register, upper
Up	Down	Down	Up	Down	D-register, lower
Up	Down	Down	Up	Up	D-register, upper

When the REGISTER SELECT switch is in the EXT position, signal NKDI in the CPU is true, gating the FP lines into the PCP DISPLAY indicators. When the REGISTER SELECT switch is in the A, B, C, D, or S position, signal KDI is true, gating CPU sum bus information into the DISPLAY indicators.

A logic diagram of the display switches is shown in figure 3-217. The switch outputs are used to gate the desired information onto the FP lines. When signal KFPXSL or KFPXSU is true, the lower or upper portion of the sum bus contents is gated directly onto the FP lines with the equations

FPXSL = FPDIS KFPXSL

FPXSU = FPDIS KFPSU

FP0-FP31 = S0-S31 FPSL

FP8-FP31 = S48-S71 FPSU

The contents of the B-register are placed on the sum bus when KSXB is true with the equations

SXB = KSXB SDIS

S0-S71 = B0-B71 SXB

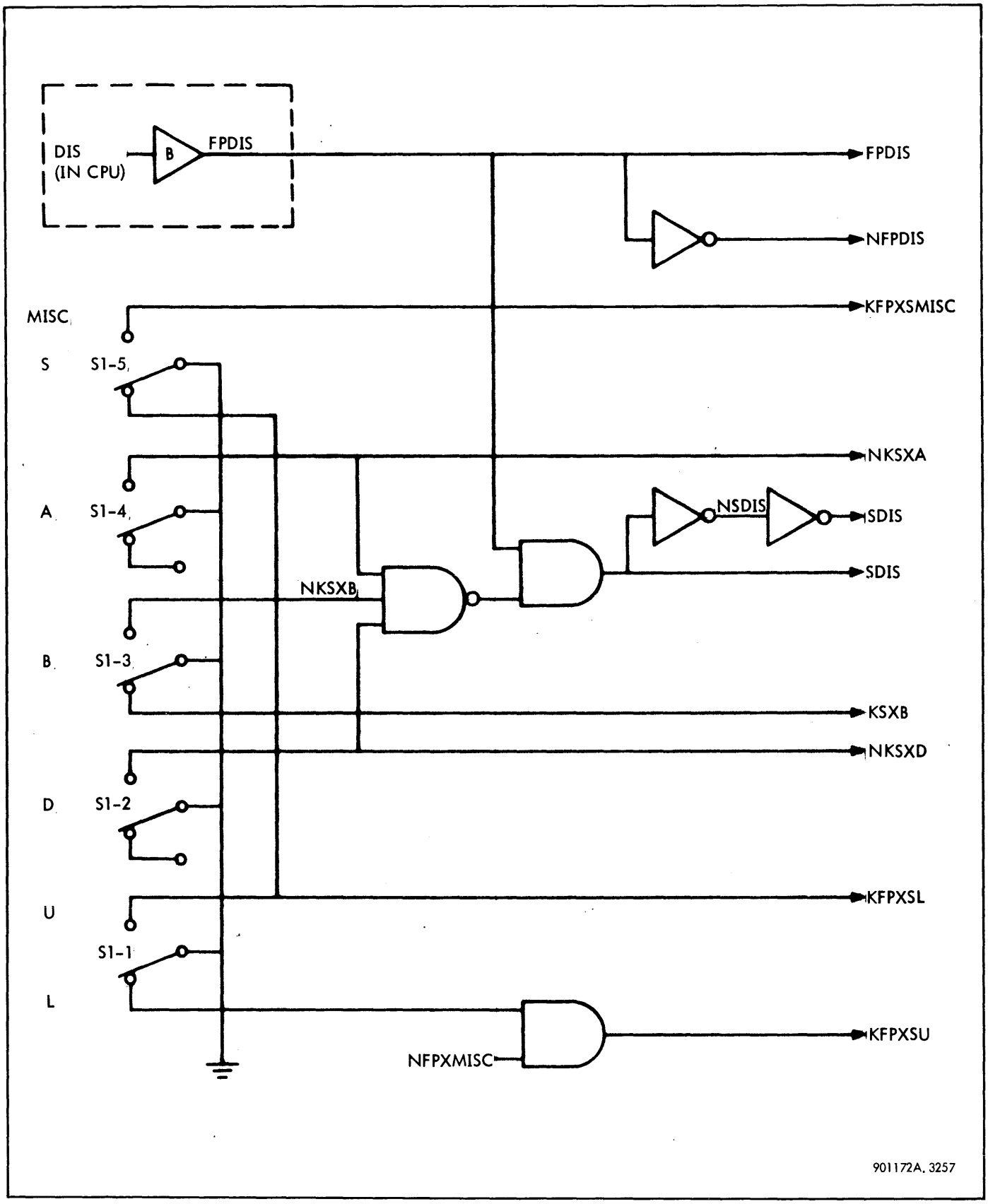


Figure 3-217. Floating Point Display Switches, Logic Diagram

The miscellaneous signals are displayed as a result of enabling signal FPXMISC with the equation

FPXMISC = FPDIS KFPXSMISC

The A-register and D-register contents are placed on the sum bus by way of the adder when NKSXA or NKSXD is false. The adder propagate terms are generated as follows:

PRXAD/ = SDIS (KSXA + KSXD) + . . .

PRXAND/ = SDIS KSXA + . . .

PRXNAD/ = SDIS KSXD + . . .

The PRXNAND term, the G terms, and K31 are all qualified by NSDIS.

Using bit 12 as an example, figure 3-218 shows the transfer of data between the CPU and the floating point unit by means of the FP lines.

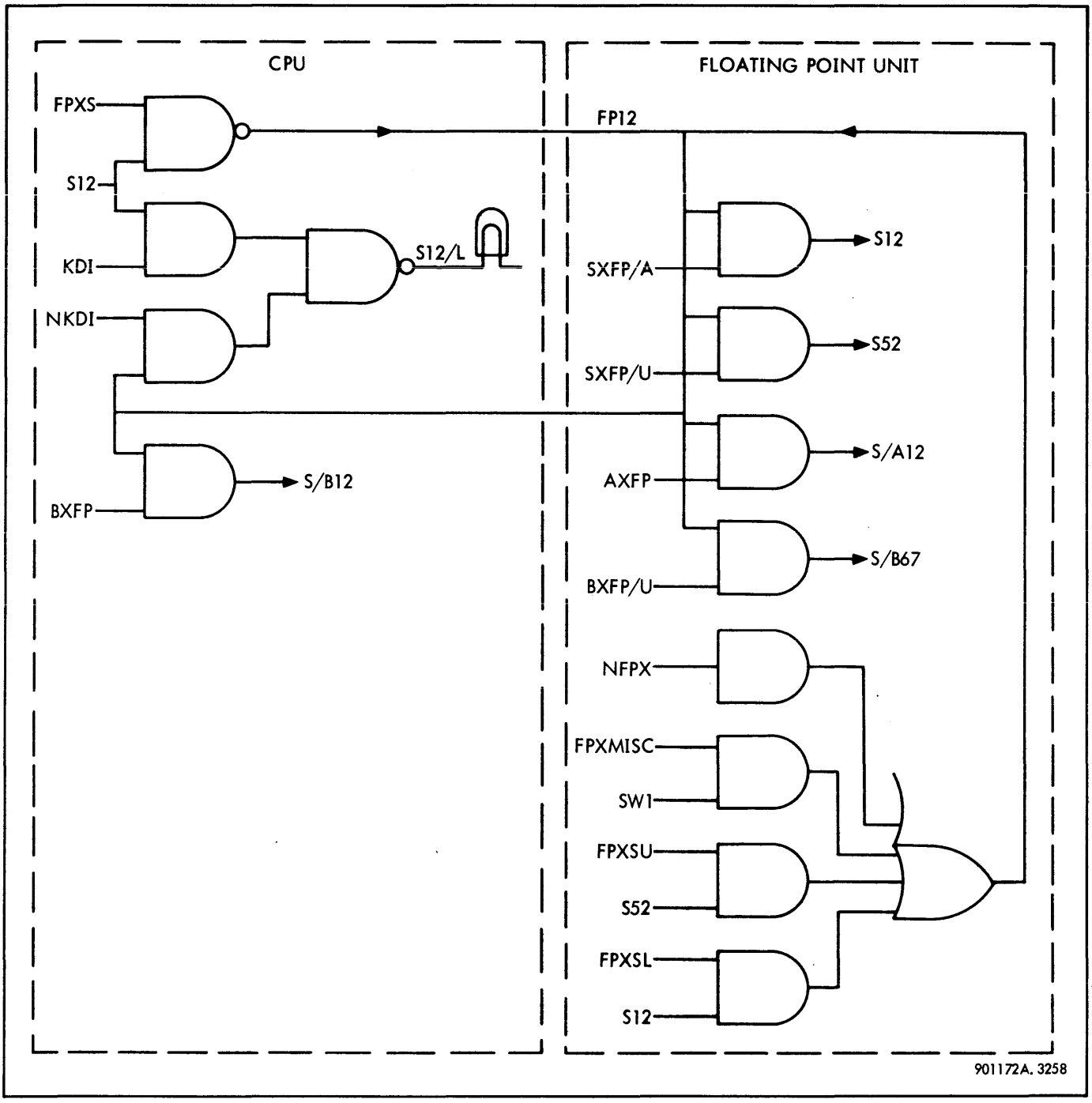


Figure 3-218. Floating Point Bit 12, Logic Diagram

3-96 PROCESSOR CONTROL PANEL (PCP)

The Sigma 5 Processor Control Panel, with its switches, indicators and displays, is shown in figure 2-1. The PCP is divided into two separate functional sections. The upper section (labeled MAINTENANCE SECTION) is reserved for maintenance controls and indicators, and the lower section (not labeled) contains the controls and indicators for the computer operator.

3-97 Control Switches

The PCP control switches, their designators, logic names, and true or false logic levels, as well as their functions, are given in table 3-97. The COMPUTE switch must be set to the IDLE position before the control switches, except in the case of the POWER and INTERRUPT pushbuttons and the CONTROL MODE, ADDR STOP, and INSTR ADDR switches, will function.

Three logic signals are not directly associated with any single control switch, but, rather, are the result of several combinations of switch settings. These logic signals are KAS/1, KAS/2, and NKAS/B. The following switch logic describes the conditions under which these signals are true.

KAS/1 = NFILL (DATA CLEAR + DATA ENTER)
+ NFILL (STORE INST ADDRESS + STORE SEL ADDRESS)
+ NFILL (KPSW1 + KPSW2)
+ NFILL (COMPUTE RUN + COMPUTE STEP)
+ NFILL (DISPLAY INST ADDRESS + DISPLAY SEL ADDRESS)
+ NFILL (INST ADDRESS INCREMENT)
+ FILL (DATA CNTR STORE CNTR INSERT CNTR COMPUTE IDLE DISPLAY CNTR INSTR ADDR CNTR)*
+ CONTROL MODE LOCK

KAS/2 = NFILL COMPUTE RUN
+ NFILL DISPLAY INST ADDR
+ DATA (ENTER + CLEAR) NFILL
+ NFILL INSERT PSW2
+ NFILL STORE INST ADDRESS + SEL ADDRESS
+ FILL
+ CONTROL MODE LOCK

NKAS/B = NFILL (DATA CNTR STORE CNTR INSERT CNTR COMPUTE IDLE DISPLAY CNTR INSTR ADDRESS CNTR)*

*Where CNTR = switch in center position

3-98 Indicators

The PCP indicators, their designators, and their associated lamp drivers are listed in table 3-98.

3-99 PCP Phase Sequencing

Most control operations carried out by the PCP require one or more PCP phase sequences. These phase sequences are controlled by six flip-flops, PCP1 through PCP6. The logic for the PCP phase flip-flops is given in the sequence charts for the individual PCP functions.

3-100 CLOCK MODE Switch

When the program is sequencing normally, the CLOCK MODE switch is in CONT and the clock enable signal, CLEN, is not inhibited, since switch signal KSC is false. When the switch is in the center position, however, KSC is true and the clock enable signal is inhibited. The equation for clock enable signal CLEN is as follows:

$$CLEN = N\,[(NCEINT\ KSC)\ NSC2]$$
$$N\,[(NCEINT\ KSC)\ SCL] + \ldots$$

If the switch is set to SINGLE STEP, KC goes true, causing SC1 to set on the next 1-MHz clock with the equation

$$S/SC1 = KSC\ KC$$

Flip-flop SC2 then sets on the next 1-MHz clock with the equation

$$S/SC2 = SC1$$

Signal CLEN is enabled momentarily when SC2 is set. The first ac clock generated sets latch SCL, which inhibits further clocks by disabling signal CLEN. The SCL latch resets when SC2 is reset:

$$SCL = SCL\ SC2 + SC2\ NCEINT\ CL$$

When the CLOCK MODE switch is returned to the center position, NKC/B is true and SC1 is reset with the equation

$$R/SC1 = NKC/B\ SCL$$

Flip-flop SC2 is then reset with the equation

$$R/SC2 = NSC1$$

At this point, signal CLEN is again inhibited.

3-101 CONTROL MODE Switch

The CONTROL MODE switch is a two-position key lock. When the switch is in LOCAL, all controls and indicators on the PCP are operative. Except for the POWER and INTERRUPT pushbuttons and the SENSE and AUDIO switches, when the switch is in LOCK the gates associated with most control panel switches are inhibited and retain the functional status that was occupied when the CONTROL MODE switch was set to the LOCK position.

The switches listed in table 3-99 are interlocked to the states indicated when the CONTROL MODE switch is in the LOCK position.

Table 3-97. PCP Control Switches

Switch Name	Designator	Logic Name	Switch Position	Logic Level	Function
CONTROL MODE	S3	None	LOCAL		Supplies +8v and -8v local voltages to PCP
			LOCK		Interlocks COMPUTE switch to RUN (KRUN/B true), WATCHDOG TIMER switch to NORMAL (KWDTR false), INTERLEAVE SELECT switch to NORMAL (KINLVSEL false), PARITY ERROR MODE switch to CONT (KHOP false), and CLOCK MODE switch to CONT (KSC false). The POWER, INTERRUPT, AUDIO, and SENSE switches remain operative. All other switches on the PCP are disabled. All indicators on the PCP continue to indicate the various computer states. Setting the CONTROL MODE switch to LOCK prevents unauthorized persons from disrupting a program by switch manipulation
WATCHDOG TIMER	S12	KWDTR	NORMAL	False	Allows watchdog timer runout trap
			OVERRIDE	True	Inhibits watchdog timer runout trap
INTERLEAVE SELECT	S11	KINLVSEL	NORMAL	False	Memory interleaving in effect
			DIAGNOSTIC	True	Memory interleaving not in effect
PARITY ERROR MODE	S10	KCONT	HALT	False	In HALT, halt when parity error occurs
			CONT	True	In CONT, interrupt when parity error occurs, but do not halt
		KHOP	HALT	True	
			CONT	False	
SENSE 1	S9	KSS1	1	True	Sense switches. Data from these switches can be read into the Condidion Codes (CC1-CC4) by a read direct or write direct instruction
SENSE 2	S8	KSS2	1	True	
SENSE 3	S7	KSS3	1	True	
SENSE 4	S6	KSS4	1	True	
CLOCK MODE	S5	KC	CONT	False	Three-position switch. Center position inhibits all ac clocks in CPU. CONT position allows continuous ac clocks. SINGLE STEP momentary position provides one clock each time switch is moved to SINGLE STEP position
			Center	False	
			SINGLE STEP	True	
		NKC/B	CONT	True	
			Center	True	
			SINGLE STEP	False	
		KSC	CONT	False	
			Center	True	
			SINGLE STEP	True	

(Continued)

Table 3-97. PCP Control Switches (Cont.)

Switch Name	Designator	Logic Name	Switch Position	Logic Level	Function
REGISTER DISPLAY	S4	KD	ON Off	True False	When ON, permits REGISTER SELECT switch to display selected register in DISPLAY indicators. KD will be true only if REGISTER DISPLAY switch is ON and the CLOCK MODE switch is not in CONT
REGISTER SELECT	S1	KDI	A B C D S EXT	True True True True True False	Selects register whose contents will be transferred to sum bus
		KSXB	B	True	Force B0-B31 to sum bus for display if KDI
		KSXD	D	True	Force D0-D31 to sum bus for display if KDI
		KSXS	S	True	Display contents of sum bus S0-S31 if KDI
		KSXA	A	True	Force A0-A31 to sum bus for display if KDI
		KSXC	C	True	Force C0-C31 to sum bus for display if KDI
AUDIO	S2	None	ON		Closes speaker circuit to allow an audio alarm when alarm flip-flop is set
POWER	S19	None			Supplies or removes ac power to power supplies PT14, PT15, PT16, and PT17. Causes signal ST (START) to initialize system. When power is supplied to or removed from the system PON or IOFF signals in the optional power monitor cause interrupts
CPU RESET/ CLEAR	S18	KCPURESET NKCPURESET/B	Pressed	True False	Initializes CPU. If pressed simultaneously with SYSTEM RESET/CLEAR switch, the CPU and the IOP are initialized and core memory is cleared to 0's
I/O RESET	S17	KIORESET	Pressed	True	Initializes all I/O operations. All peripheral devices are halted, and all status and control indicators in the I/O system are reset. Does not affect the current operations of the CPU

(Continued)

Table 3-97. PCP Control Switches (Cont.)

Switch Name	Designator	Logic Name	Switch Position	Logic Level	Function
LOAD	S16	KFILL/B	Pressed	True	Pressing the LOAD switch (with COMPUTE in IDLE) forces a bootstrap program to be entered in memory locations X'20' through X'29'
UNIT ADDRESS	S15A			Encoded	The three UNIT ADDRESS thumbwheel switches are used in the load operation to designate from left to right the input/output processor, the device controller, and the device. The address designated by the UNIT ADDRESS switches is stored into memory location X'25' when the LOAD switch is set
		KUA21		2^{10}	
		KUA22	0-7	2^{9}	
		KUA23		2^{8}	
	S15B			Encoded	
		KUA24		2^{7}	
		KUA25	0-F	2^{6}	
		KUA26		2^{5}	
		KUA27		2^{4}	
	S15C			Encoded	
		KUA28		2^{3}	
		KUA29	0-F	2^{2}	
		KUA30		2^{1}	
		KUA31		2^{0}	
SYSTEM RESET/ CLEAR	S14	KSYSR/B	Pressed	True	Initializes the CPU and all I/O functions. If pressed simultaneously with CPU RESET/CLEAR switch, core memory is cleared to all 0's
		NKSYSR		False	
INTERRUPT	S13	KINTRP	Pressed	True	Causes an interrupt to location X'5D' if PSW2 bit 6 (flip-flop II) is a 0
		NKINTRP/B		False	
INSERT	S21	KPSW1/B	PSW1	True	Enters the contents of the DATA switches into PSW1 or PSW2 if COMPUTE is in IDLE
			PSW2	False	
		KPSW2/B	PSW1	False	
			PSW2	True	
STORE	S23	KSTORK/B	SELECT ADDR	True	In SELECT ADDR stores the current value of the DISPLAY indicators into the location pointed to by the SELECT ADDRESS switches
			INSTR ADDR	False	
		KSTORQ/B	INSTR ADDR	True	In INSTR ADDR stores the current value of the DISPLAY indicators into the location pointed to by the INSTRUCTION ADDRESS indicators
			SELECT ADDR	False	

(Continued)

Table 3-97. PCP Control Switches (Cont.)

Switch Name	Designator	Logic Name	Switch Position	Logic Level	Function
DATA	S43	KCLEAR/B	CLEAR	True	Resets the DISPLAY indicators (D-register)
			ENTER	False	
		KENTER/B	ENTER	True	Enters the contents of the DISPLAY indicators according to the states of the 32 DATA switches.
			CLEAR	False	
INSTR ADDR	S20	KINCRE/B	INCREMENT	True	Momentary position. Causes the instruction address in the P-register to count up by 1
			HOLD	False	
		NKAHOLD	HOLD	False	Inhibits the P-register from counting
			INCREMENT	True	
DISPLAY	S22	KDISPLAK/B	SELECT ADDR	True	Displays in the DISPLAY indicators the contents of the location pointed to by the SELECT ADDRESS switches
			INSTR ADDR	False	
		KDISPLAQ/B	INSTR ADDR	True	Displays in the DISPLAY indicators the contents of the location pointed to by the INSTRUCTION ADDRESS indicators
			SELECT ADDR	False	
COMPUTE	S42	KRUN/B	RUN	True	With COMPUTE in RUN the CPU sequences normally through the program. With COMPUTE in IDLE the CPU waits in PCP2. When COMPUTE is set in the momentary STEP position from IDLE, the CPU executes the current instruction, reads the next instruction, and returns to PCP2 and waits
			IDLE	False	
			STEP	False	
		KSTEP/B	RUN	False	
			IDLE	False	
			STEP	True	
SELECT ADDRESS	S24 . . . S40	KSP31 . . . KSP15	1 0	True False	The 17 SELECT ADDRESS switches are used with the ADDR STOP switch to select the address at which the program is to be halted; with the STORE switch to select the address of a memory location to be altered; and with the DISPLAY switch to select the address of a memory location to be displayed
ADDR STOP	S41	KADDRSTOP	ON	True	When this switch is ON the CPU halts when the value of the memory address register equals the value set in the SELECT ADDRESS switches. At the halt, the instruction in the location pointed to by the INSTRUCTION ADDRESS indicators appears in the DISPLAY indicators. This instruction is the one that would have been executed next had the halt not occurred. With JUMP instr. the comp will stop even if the address insertion will not be needed. (conditions for jumps are not met) It will stop at the next instruction after conditional jump Ref: 25(6) Handbook

(Continued)

Table 3-97. PCP Control Switches (Cont.)

Switch Name	Designator	Logic Name	Switch Position	Logic Level	Function
DATA 0	S75	KS0	1 0	True False	The 32 DATA switches are used to enter a new value into PSW1 or PSW2 when used with the INSERT switch, or to enter a new value in the DISPLAY indicators when used with the DATA switch
.	.	.	.	.	
.	.	.	.	.	
.	.	.	.	.	
DATA 31	S44	KS31	1 0	True False	
CLEAR PSW1	S77	KCLR PSW1	Up	True	Clears contents of PSW1
CLEAR PSW2	S76	KCLR PSW2	Up	True	Clears contents of PSW2

Table 3-98. PCP Indicators

Indicator Name	Designator	Lamp Driver Origin
INSTRUCTION ADDRESS	DS39 . . . DS55	P31/L . . . P15/L
TRAP ARITH	DS56	AM/L
MODE SLAVE	DS59	MASTER/L
FLOAT MODE NRMZ ZERO SIG	 DS60 DS61 DS62	 FNF/L FZ/L FS/L
CONDITION CODE 1 2 3 4	 DS66 DS65 DS64 DS63	 CC1/L CC2/L CC3/L CC4/L
POINTER	DS29 . . . DS32	RP27/L . . . RP24/L

(Continued)

Table 3-98. PCP Indicators (Cont.)

Indicator Name	Designator	Lamp Driver Origin
INTRPT INHIBIT		
EXT	DS34	EI/L
I/O	DS35	II/L
CTR	DS36	CIF/L
WRITE KEY	DS38	WK0/L
	DS37	WK1/L
DISPLAY	DS67	S31/L
	:	:
	DS98	S0/L
POWER	DS28	+8v
NORMAL MODE	DS25	Special from PT16
RUN	DS24	RUN/L
WAIT	DS23	WAIT/L
INTERRUPT	DS22	CPI/L
MEMORY FAULT	DS21	MFL0/L
	:	:
	DS14	MFL7/L
ALARM	DS13	ALARM/L
PHASES	DS12	PRE4/L
PREPARATION	DS11	PRE2/L
	DS10	PRE1/L
PCP	DS9	PCP4/L
	DS8	PCP2/L
	DS7	PCP1/L
EXECUTION	DS6	PH8/L
	DS5	PH4/L
	DS4	PH2/L
	DS3	PH1/L
INT/TRAP	DS2	INTRAP2/L
	DS1	INTRAP1/L

Table 3-99. Control Mode Lock Switch Status

Switch	Interlock State
COMPUTE	RUN
WATCHDOG TIMER	NORMAL
INTERLEAVE SELECT	NORMAL
PARITY ERROR MODE	CONT
CLOCK MODE	CONT

Note

Unpredictable results may occur if the CONTROL MODE switch is actuated while the COMPUTE switch is not in RUN

3-102 WATCHDOG TIMER Switch

When the WATCHDOG TIMER switch is in NORMAL, the watchdog timer counter is reset (flip-flops WCT1-WCT6 set to all ones) by signal WDTR at each interruptible period during program execution.

$$S/WDTR = IEN + PH10 + \ldots$$

If the watchdog timer, which counts up by ones each microsecond, reaches a count of 42 without being reset, a watchdog timer runout condition exists and the program traps to location X'46'.

When the WATCHDOG TIMER switch is in OVERRIDE, WDTR is held true, and the watchdog timer flip-flops are constantly held to all ones and cannot count.

$$WDTR = KWDTR + KSC + NKRUN + PCPACT$$

3-103 INTERLEAVE SELECT Switch

When the INTERLEAVE SELECT switch is in NORMAL, core memory interleaving is in effect. When the switch is in DIAGNOSTIC, memory interleaving is not in effect. Normally this switch is in DIAGNOSTIC only when the operator is performing memory diagnostic programs. Interleave logic is found in the core memory.

3-104 AUDIO Switch

The AUDIO switch in the ON position connects the PCP speaker to either flip-flop MUSIC or flip-flop ALARM. If ALARM is true, and the AUDIO switch is ON, the speaker will emit a 1-kHz signal. The ALARM and MUSIC flip-flops are set or reset by the write direct instruction.

3-105 SENSE Switches

The four SENSE switches on the PCP operate in either the local or lock control modes.

3-106 REGISTER DISPLAY Switch

The REGISTER DISPLAY switch is used with the REGISTER SELECT 12-position switch to display the contents of the CPU internal registers. The logic signal KD generated by the REGISTER DISPLAY switch is true only when the REGISTER DISPLAY switch is ON and the CLOCK MODE switch is not in CONT.

3-107 REGISTER SELECT Switch

The REGISTER SELECT switch is used to display the following information under the conditions noted:

a. Contents of the CPU registers or sum bus, as selected by positions A, B, C, D, and S on the panel above the switch when the REGISTER DISPLAY switch is ON. The display is in the DISPLAY indicators.

b. Contents of the floating point unit registers or sum bus, as selected by switches on the floating point unit (paragraph 3-95) when the REGISTER SELECT switch is in the EXT position. The display is in the DISPLAY indicators.

c. Integral IOP information as shown in table 2-2 when the REGISTER SELECT switch is set at EXT. The display is in the INSTRUCTION ADDRESS and EXECUTION, PCP, and PREPARATION PHASES indicators.

The DISPLAY indicators are lighted by lamp drivers S0/L through S31/L, which receive their inputs from the sum bus, S0 through S31, or the floating point unit, FP0 through FP31 as follows:

$$S0/L\text{-}S31/L = S0\text{-}S31\ KDI + FP0\text{-}FP31\ NKDI$$

where NKDI is true when the REGISTER SELECT switch is in the EXT position.

The contents of the CPU B- and C-registers are gated onto the sum bus as follows:

$$S0\text{-}S31 = B0\text{-}B31\ SXB + C0\text{-}C31\ SXC + \ldots$$

$$SXB = KSXB\ DIS + \ldots$$

$$SXC = KSXC\ DIS + \ldots$$

$$DIS = NKSXS\ KD\ KSC\ NSC1$$

Signals KSXB and KSXC are the outputs of the REGISTER SELECT switch in the B and C positions respectively, signal KD is true when the REGISTER DISPLAY switch is ON, and KSC is true when the CLOCK MODE switch is in the center position.

The A- and D-register outputs are placed on the sum bus by way of the adder propagate signals, PR0 through PR31.

REGISTER SELECT switch outputs KSXA and KSXD, true when the switch is in the A and D positions respectively, are inverted and used in the adder enable terms PRXAD, PRXNAD, PRXAND, and PRXNAND in such a way that the propagate term for each bit will contain the A- or D-register information when the switch is in the appropriate position. The propagate logic is explained in detail in the discussion on the CPU adder.

If the REGISTER SELECT switch is in the S position, the REGISTER DISPLAY switch is off, or the CLOCK MODE switch is in the CONT position, signal NKSXS, KD, or KSC respectively is false, driving signal DIS false. In this case, any information that happens to be on the sum bus is displayed in the DISPLAY indicators.

The integral IOP information is displayed by way of the following lamp drivers:

INSTRUCTION ADDRESS	
P16/L-P25/L	IOFR0-IOFR9
P26/L	IOFM
EXECUTION PHASES	
PH1/L-PH4/L	IOPH0-IOPH3
PCP PHASES	
PCP4/L	IOSC
PREPARATION PHASES	
PRE4/L, PRE2/L, PRE1/L	SW4/LP, SW2/LP, SW1/LP. States of SW9-SW15, binary coded from 001 (SW9) to 111 (SW15)

The I/O information is gated onto the appropriate lamp driver lines by signal NKDI, which is true when the REGISTER SELECT switch is in the EXT position. Typical equations are as follows:

P19/L = P19 KDI + IOFR3 NKDI

PCP4/L = PCP4/LP KDI + IOSC NKDI

3-108 I/O RESET Switch

The I/O RESET switch generates signal KIORESET. KIORESET is gated with NKAS/B-1 to produce the I/O reset signal /RIOC/. Signal KIORESET is interlocked to the false state when the CONTROL MODE switch is in LOCK. The I/O RESET switch does not affect the current operation of the CPU.

3-109 UNIT ADDRESS Switches

The three UNIT ADDRESS switches are used with the LOAD switch to enter into the initial bootstrap load routine the address of the device, device controller and the IOP from which the data is to be read into memory. The UNIT ADDRESS switches decode the hexadecimal numbers to their binary representations.

3-110 INTERRUPT Switch

The CPU INTERRUPT switch generates the signal KINTRP, which causes the CPU to interrupt to location X'5D'. Unless PSW2 bit 6 is a zero and the interrupt level is armed, the interrupt will not occur.

3-111 SELECT ADDRESS Switches

The 17 SELECT ADDRESS switches are used with the ADDR STOP switch to select the address at which the program is to be halted. They are used with the STORE switch to select the address of a memory location to be altered, and are also used with the DISPLAY switch to select the address of a memory location to be displayed.

3-112 DATA Switches

The 32 DATA switches are used to alter the contents of PSW1 or PSW2 when used with the INSERT switch, or to change the value of the DISPLAY indicators (D-register) when used with the DATA switch.

3-113 Entering PCP Phases (See figure 3-219.)

The PCP phases are entered when signal HALT/1 is true and phase 10 of the current instruction is reached.

S/PCP1 = BRPCP1 + ...

BRPCP1 = HALT/1 PH10 NFUEXU NIOSC N(INT IEN) N(S/TRAP)

Signal HALT/1 goes true under the following conditions:

a. The COMPUTE switch is set to IDLE and phase PRE1 of the current instruction is reached.

S/HALT = NKRUN PRE1 NFUEXU + ...

b. A wait instruction has been executed.

S/HALT = FUWAIT PH1 + ...

c. The ADDR STOP switch is ON and the value in the SELECT ADDRESS switches is equal to the address on the memory address lines.

HALT/1 = DCSTOP + ...

DCSTOP = MR ADMATCH KADDRSTOP NIOACT + ...

d. A trap or interrupt has occurred and the instruction being executed is not a modify and test or exchange program status doubleword instruction.

S/HALT = INTRAP PRETR N(FAMT + XPSD)

e. Power is applied to or removed from the system.

S/PCP2 = RESET/KS + ...

RESET/KS = RESET NKCPURESET/B

RESET = SYSR + ...

SYSR = START + ...

START = /ST/ (from power monitor)

Each of the above conditions, except condition e, causes the CPU to enter PCP1. When dc power is applied to or removed from the system, signal START forces the CPU directly to PCP2. Phase PCP1 is entered at the end of any instruction, except execute, if signal HALT/1 is true and no trap or interrupt is active.

The program goes from PCP1 to PCP2 with the equations

R/PCP1 = ...

S/PCP2 = PCP1 + RESET/KS

The program remains in PCP2 until a control switch is operated.

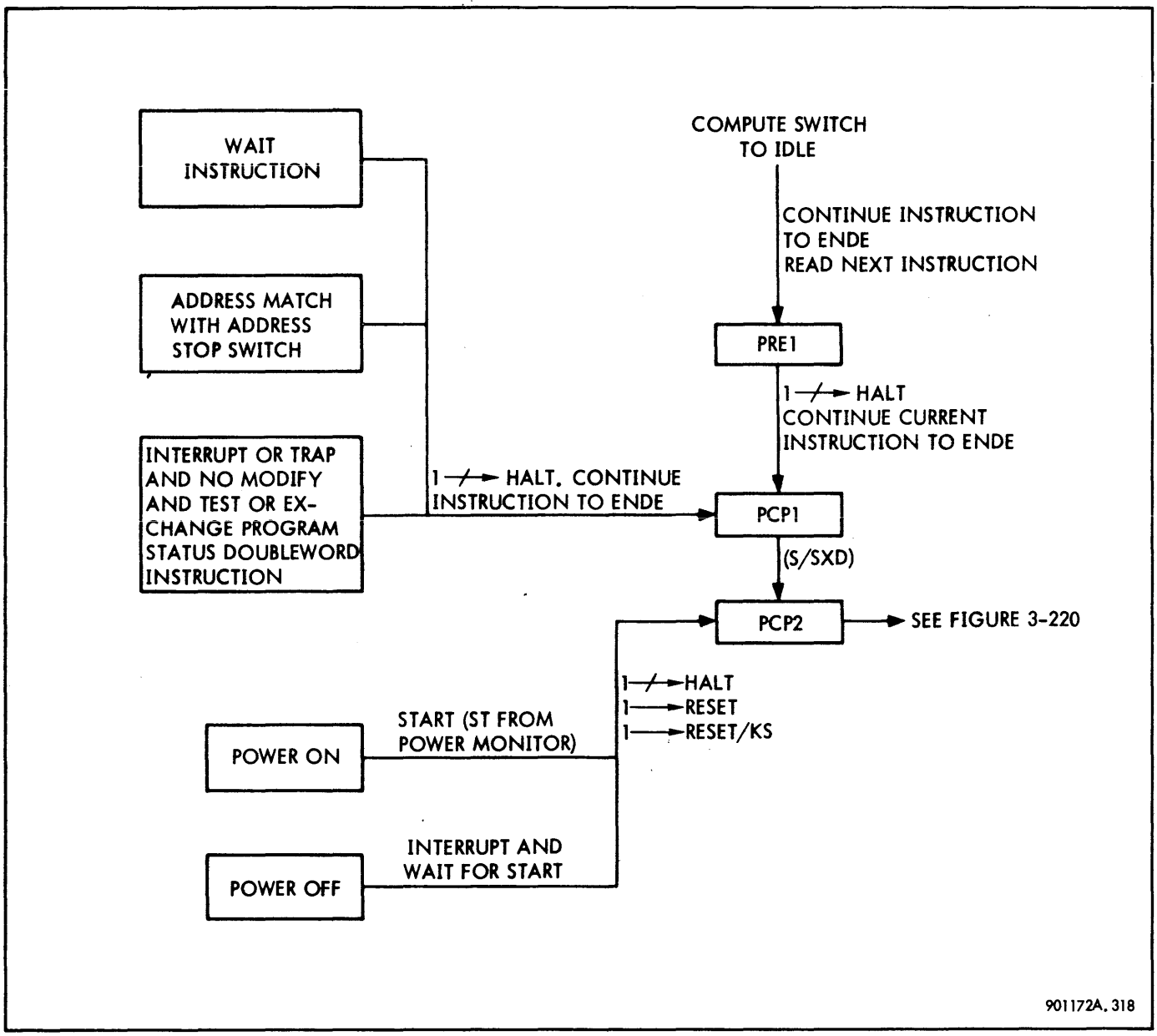


Figure 3-219. Entering PCP Phases

The PCP phase sequencing following PCP2 is described under the various functions of the control switches.

An overall diagram of the PCP sequencing beyond the idle phase is shown in figure 3-220.

3-114 Reset Function

To reset the CPU by pressing either the CPU RESET/CLEAR switch or the SYSTEM RESET/CLEAR switch, the COMPUTE switch must first be placed in the IDLE position. The logical sequence of events when either of these switches is pressed is shown in table 3-100 and figure 3-221.

3-115 Clear PSW1, PSW2 Function

When the CPU is in PCP2 because the COMPUTE switch is in IDLE, setting the CLEAR switch to PSW1 or PSW2 clears program status doubleword 1 or 2 respectively. Signal KCLRPSW1 or KCLRPSW2 is generated at the switch output and signal PSW1XS or PSW2XS is developed:

PSW1XS = KCLRPSW1 NIOCON NKAS/B + ...

PSW2XS = KCLRPSW2 NIOCON NKAS/B + ...

Zeros are placed on the sum bus by inhibiting signal S/SXD, which is normally true during PCP2:

S/SXD = NKCLRPSW/B PCP2/1 NRESET/C NIOCON

Signal NKCLRPSW/B goes low when either of the CLEAR switches is operated.

The control flip-flops in PSW1 are cleared as follows:

R/CC1 -CC4 = CCXS/0

CCXS/0 = PSW1XS

R/FS, R/FZ, R/FNF, R/NMASTER, R/AM = PSW1XS

The P-register (instruction address) is cleared by transferring the zeros on the sum bus into the P-register with PXS:

PXS = PSW1XS + ...

The control flip-flops in PSW2 are cleared as follows:

R/WK0, R/WK1, R/CIF, R/II, R/EI = PSW2XS + ...

The register pointer in PSW2 is cleared by transferring the zeros on the sum bus into the RP-register as follows:

R/RP24 - RP27 = RPXS

RPXS = PSW2XS

3-116 STEP or RUN from Idle Operation

When the CPU is idling in PCP2 because the COMPUTE switch is in IDLE, setting the switch to STEP causes the CPU to enter PCP3 and branch to phase 10, then perform the instruction execution. After execution, the signal BRPCP1 goes true and the CPU sequences to PCP1 and PCP2 where it again remains in the idle state.

Table 3-100. Reset Sequence Chart

Phase	Function Performed	Signals Involved	Comments
	Switches and signals involved:		
	CPU RESET/CLEAR ⇒	KCPURESET, KCPURESET/B	
	or		
	SYSTEM RESET/CLEAR ⇒	KSYSR, KSYSR/B	
	COMPUTE in IDLE ⇒	NKAS/B, NKRUN (necessary for either switch operation)	
			Mnemonic: RESET

(Continued)

Table 3-100. Reset Sequence Chart (Cont.)

Phase	Function Performed	Signals Involved			Comments
PCP2	Idle phase – sustained until control switch operated				
	Reset HALT flip-flop	R/HALT	=	PCP2 NKAS/B + ...	Prepare logic for setting PCP3
	Inhibit interrupts during idle phase	(S/INTRAP)	=	N(PCP2 NKRUN)	Inhibit setting of first of interrupt phase sequence flip-flops
	X'02000000' ⟶ (D0-D31)	S/D6	=	RESET/C + ...	Place in D-register a load conditions and floating control immediate instruction with zeros in bits 10 and 11 to produce a no operation instruction
		RESET/C	=	RESET/B NIOCON NMRC + ...	
		RESET/B	=	N(KSYSR/B KCPURESET/B) NKAS/B (KCPURESET/B + KSYSR/B)	
		DX	=	DXZ + ...	
		DXZ	=	RESET	
		RESET	=	SYSR + (KCPURESET RESET/B · NIOCON)	
		SYSR	=	KSYSR RESET/B + ...	
	0 ⟶ PSW1 (except P)	PSW1XS	=	RESET + ...	Sum bus contains zeros because no adder preset has been made
	0 ⟶ PSW2	PSW2XS	=	RESET + ...	
	X'25' ⟶ (P15-P31)	S/P26, S/P29, S/P31	=	RESET/C + ...	Set program address to X'25'
		R/P15-P31	=	PX	
		PX	=	RESET + ...	
	Set flip-flop BRP	S/BRP	=	RESET/C + ...	Indicates that program address is in P-register
	0 ⟶ interrupt arm and enable flip-flops	E/IS0-E/IS15	=	RESET	Reset interrupt levels to disarmed and disabled state
		E/IP0-E/IP15	=	RESET	
		R/IN0-R/IN15	=	REN	
		REN	=	RESET + ...	
	Reset ALARM indicator	ALARM/L	=	ALARM	Turn off alarm indicator on panel
		R/ALARM	=	RESET	
					Mnemonic: RESET

(Continued)

Table 3-100. Reset Sequence Chart (Cont.)

Phase	Function Performed	Signals Involved	Comments
PCP2 (Cont.)	SYSTEM RESET/CLEAR ⇒ RESETIO	RESETIO = SYSR + ...	Initialize input/output system
	/MFR/	/MFR/ = RESET + ...	Send signal to memory to reset memory fault indicators
	/MR/	/MR/ = RESET/KS RESET/KS = RESET NKCPURESET	Send signal to memory to initialize memory control logic
	Sustain PCP2 until control switch activated	S/PCP3 = PCP2/1 NIOCON NDCSTOP (CLEARMEM + INT KRUN + NHALT KAS/1 KAS/2)	
			Mnemonic: RESET

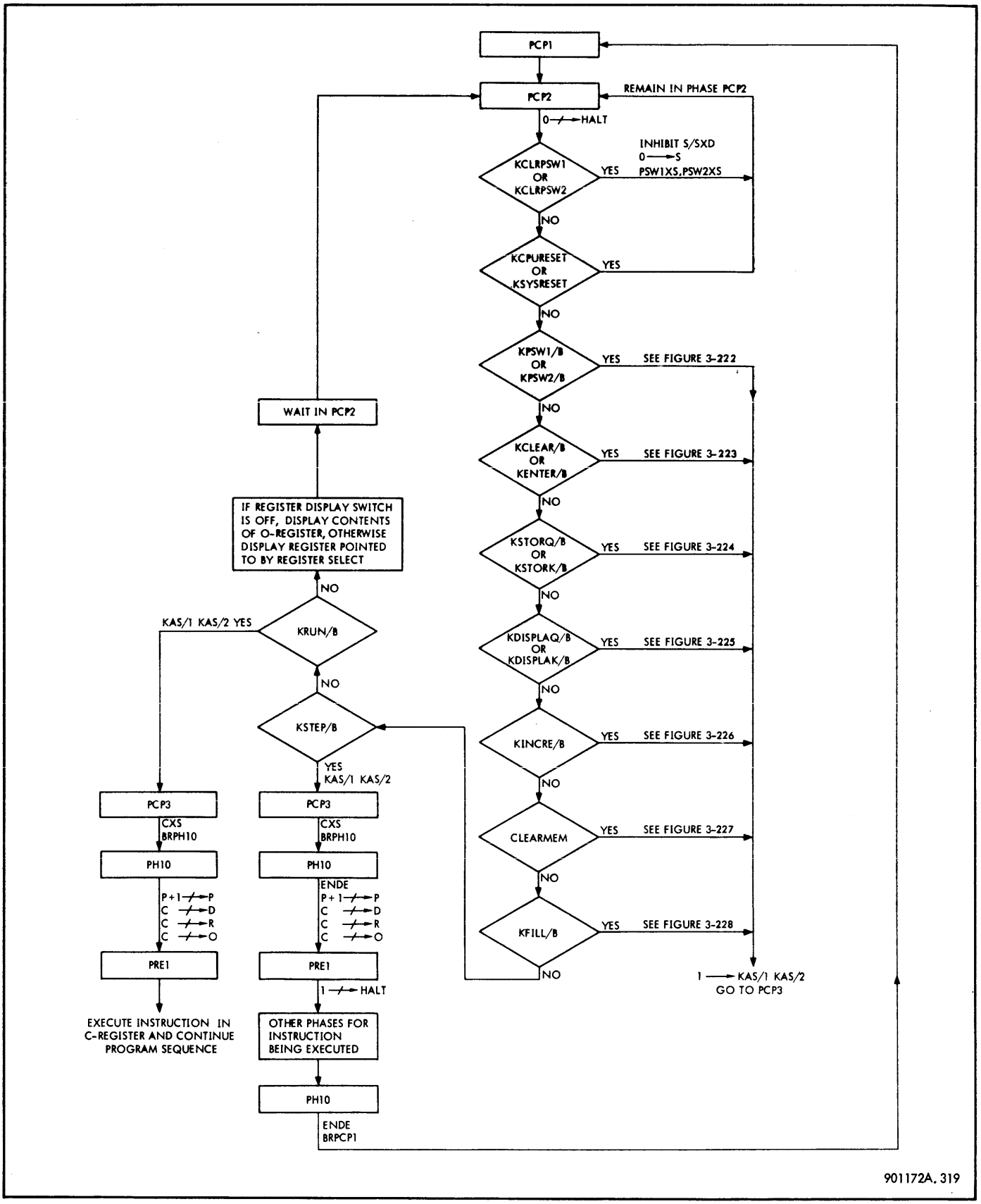


Figure 3-220. PCP Sequencing Beyond Idle State

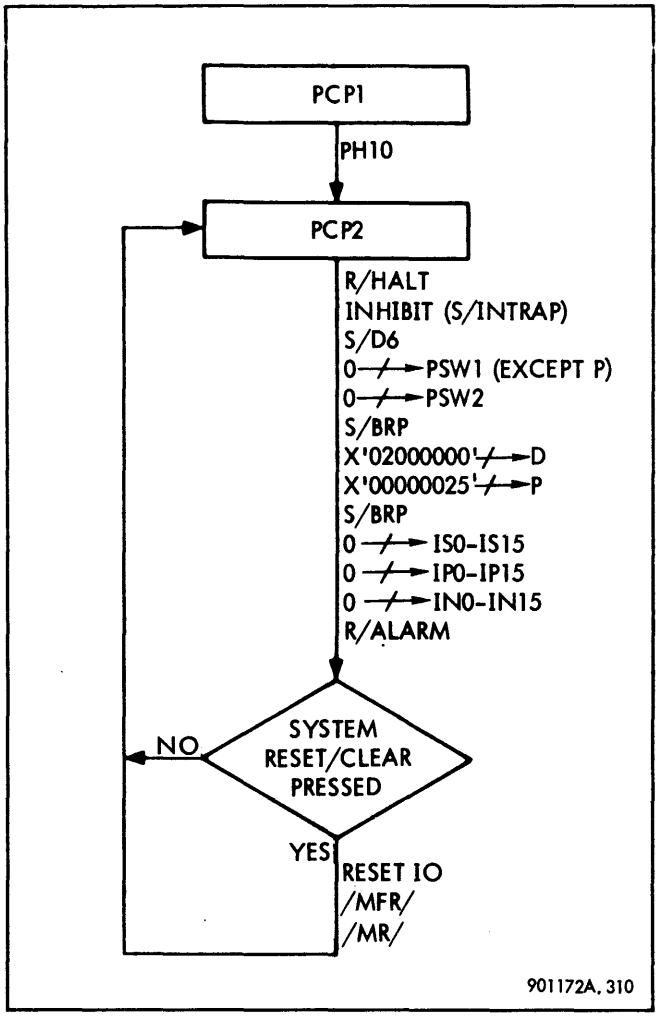


Figure 3-221. CPU RESET/CLEAR and SYSTEM RESET/ CLEAR, Flow Diagram

When the COMPUTE switch is moved from IDLE to RUN, the CPU sequences to PCP3, branches to phase 10, and continues to sequence through the program in its normal manner. The sequence of operations when this switch is operated is shown in figure 3-219.

3-117 INSERT Function (See figure 3-222.)

If in idle phase PCP2 the INSERT switch is placed in PSW1, a program status word PSW1 will be entered according to the settings of the DATA switches. If the INSERT switch is set to PSW2, the program status word PSW2 will be entered according to the settings of the DATA switches. The DATA switches can only set or cause no change in the corresponding bits of PSW1 and PSW2. If a reset is required in any bit, the contents of PSW1 or PSW2 must be cleared before entering new data. The logic sequence for the INSERT function is provided in table 3-101.

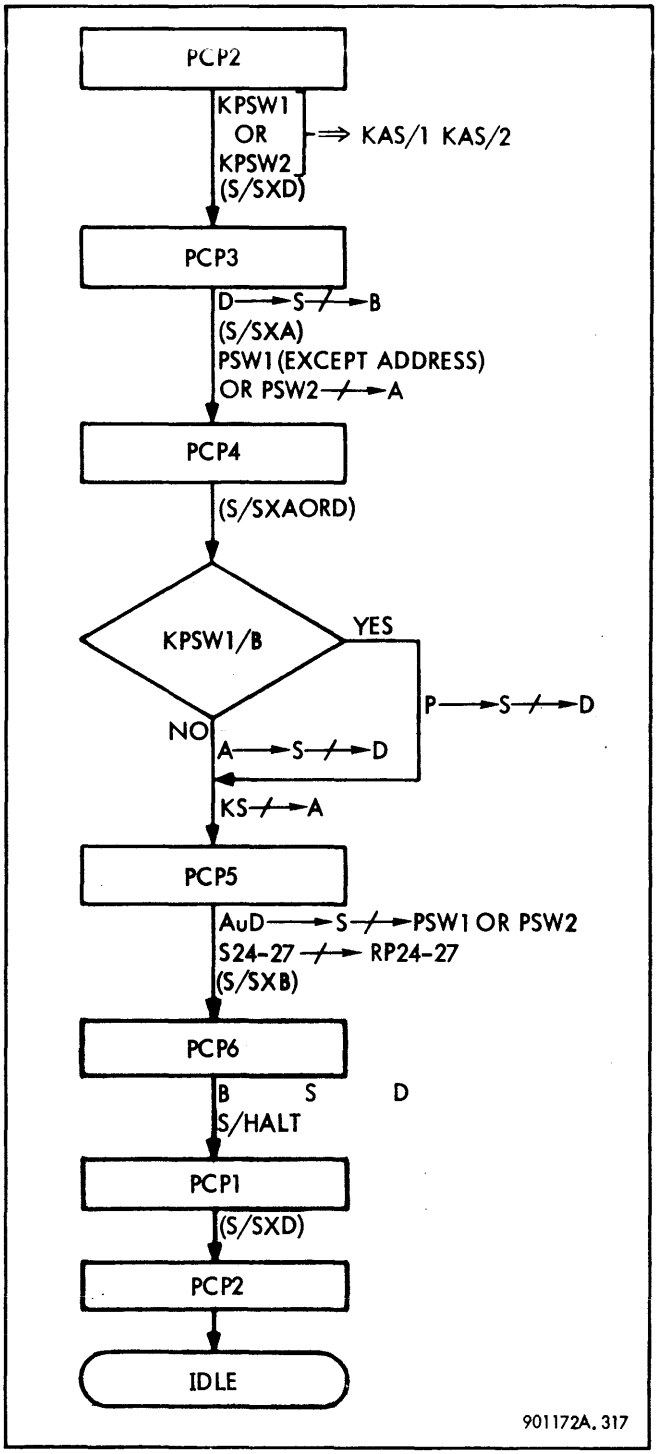


Figure 3-222. Insert PSW1/Insert PSW2, Flow Diagram

3-118 DATA ENTER/CLEAR Function (See figure 3-223.)

When the DATA switch is set to ENTER, the states of the 32 DATA switches are transferred to the D-register and are displayed in the 32 DISPLAY indicators. When the DATA switch is set to CLEAR, zeros are transferred to the D-register. If after data has been transferred from the DATA

Table 3-101. Insert PSW1/Insert PSW2 Sequence

Phase	Function Performed	Signals Involved	Comments
PCP2	Idle phase sustained until KAS/1 KAS/2		
	Reset HALT flip-flop	R/HALT = PCP2 NKAS/B + ...	Enable program to proceed to PCP3
	KPSW1 or KPSW2 ⟹ KAS/1, KAS/2		
	Enable signal (S/SXD)	(S/SXD) = PCP2/1 NRESET/C NKCLRPSW/B NIOCON + ...	Preset adder for D ⟶ S in PCP3
		PCP2/1 = PCP2 NPCP3	
	Set flip-flop PCP3	S/PCP3 = (NHALT KAS/1 KAS/2+...) PCP2/1 NIOCON NDCSTOP	
PCP2/3	One clock long		
	(D0-D31) ⟶ (S0-S31)	Adder preset at PCP2 clock	Transfer instruction currently in D-register to B-register
	(S0-S31) ⟶/ (B0-B31)	BXS = PCP3 SWK12 + ...	
		SWK12 = SWK1 + SWK2	
		SWK2 = KPSW1 + KPSW2	
	Enable signal (S/SXA)	(S/SXA) = PCP3 + ...	Preset adder for A ⟶ S in PCP4
	Insert PSW1 ⟹ PSW1 (bit 0-bit 11) ⟶/ (A0-A11)	AXPSW1 = KPSW1/B PCP3 + ...	Condition code, floating control bits, MS, DM, AM ⟶ A-register to save current PSW1
	0 ⟶/ A bits not being set	AX = AXZ + ...	Enable reset inputs to A-register
		AXZ = PCP3 + ...	
	INSERT PSW2 ⟹ PSW2 (bit 2-bit 27) ⟶/ (A2-A27)	AXPSW2 = KPSWZ/B PCP3 + ...	Write key, inhibits, register pointer ⟶ A-register to save current PSW2
	NIOFS ⟹ RESET IOSC if set	R/IOSC = PCP3 NIOFS + ...	Reset internal I/O service call flip-flop if no IO function strobe
		R/PCP2 = PCP3	

(Continued)

Table 3-101. Insert PSW1/Insert PSW2 Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PCP4	One clock long		
	Enable signal (S/SXAORD)	(S/SXAORD) = PCP4 + ...	Preset adder for AuD ⟶ S in PCP5
	INSERT PSW1 ⟹ (P15-P31) ⟶ (S15-S31)	SXP = PCP4 KPSW1/B NDIS + ...	Transfer PSW1 or PSW2 to D-register
	(A0-A31) ⟶ (S0-S31)	Adder logic preset in PCP3	If PSW1, A15-A31 is empty, and S15-S31 comes from P-register (program address). If PSW2, all information going into D comes from A-register but only A2-A7 and A24-A27 contain useful information
	(S0-S31) ⟶/⟶ (D0-D31)	DXS = PCP4 SWK12 + ...	
	(KS0-KS31) ⟶/⟶ (A0-A31)	AXK = PCP4 SWK2 + ...	Manually entered information from DATA switches ⟶ A-register. KS0-KS31 are DATA switch outputs and are true when corresponding switches are in up position. Clear A-register flip-flops not set by switches.
	Enable A-register reset inputs	AX = AXK + ...	
PCP5	One clock long		
	(A0-A31) or (D0-D31) ⟶ (S0-S31)	Adder logic preset in PCP4	Sets PSW1 and PSW2 flip-flops if corresponding DATA switches are set to 1. Causes no change where data switches are not set. (To enter zeros where the original PSW contained ones, the PSW must first be cleared with the PSW1 or PSW2 CLEAR switch.)
	INSERT PSW1 ⟹ (S0-S3) ⟶/⟶ (CC1-CC4); (S5-S8) ⟶/⟶ FS, FZ, FNF, NMASTER; (S10, S11) ⟶/⟶ DM, AM	PSW1XS = PCP5 KPSW1/B + ...	
	INSERT PSW2 ⟹ (S2, S3) ⟶/⟶ WK0, WK1; (S5-S7) ⟶/⟶ CI, II, EI	PSW2XS = PCP5 KPSW2/B + ...	
	(S24-S27) ⟶/⟶ (RP24-RP27)	RPXS = PSW2XS + ...	
	Enable signal (S/SXB)	(S/SXB) = PCP5 NBRPCP5	Preset adder logic for B ⟶ S in PCP6

(Continued)

Table 3-101. Insert PSW1/Insert PSW2 Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PCP6	One clock long (B0-B31) ⟶ (S0-S31)	Adder logic set in PCP5	Return current instruction to D-register
	(S0-S31) ⟶/⟶ (D0-D31)	DXS = PCP6 SWK12	
	Set flip-flop HALT	S/HALT = PCP6 + ...	Halt computer
PCP1	One clock long Enable signal (S/$\overline{SXD}$)	(S/$\overline{SXD}$) = PCP1 + ...	Preset adder logic for D ⟶ S in PCP2
PCP2	Idle D ⟶ S ⟶ Display indicators	Preset in PCP1	

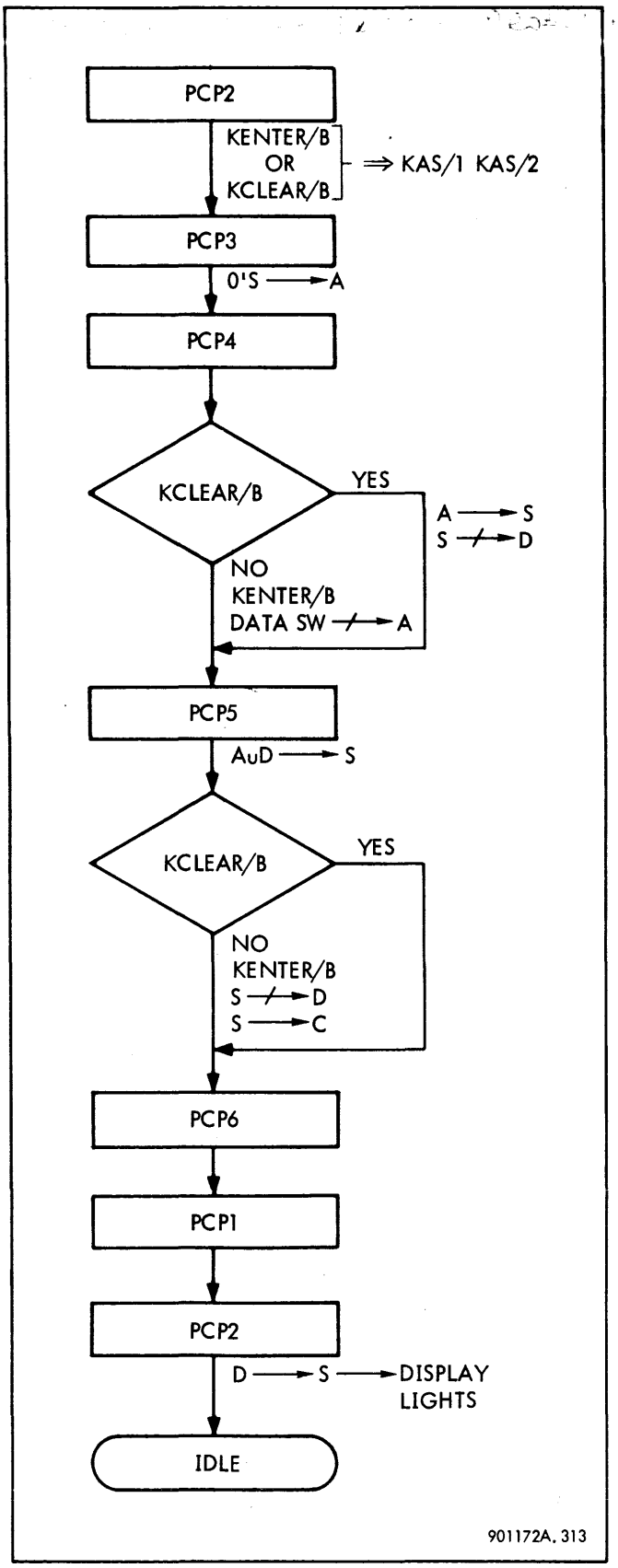


Figure 3-223. DATA ENTER/DATA CLEAR, Flow Diagram

switches into D the COMPUTE switch is set to either RUN or STEP, the contents of the D-register will be accepted as the next instruction to be executed. The logic sequence for the DATA ENTER/CLEAR function is given in table 3-102.

3-119 STORE INSTR ADDR/SELECT ADDR Function (See figure 3-224.)

The STORE switch is operative only while the CPU is in the idle state, PCP2. When the STORE switch is set to INSTR ADDR, the contents of the D-register are stored into the memory address currently in the P-register. When the STORE switch is set to SELECT ADDR, the contents of the D-register are stored into the address specified by the settings of the 17 SELECT ADDRESS switches. The logic sequence for the STORE INSTR ADDR/SELECT ADDR function is given in table 3-103.

3-120 DISPLAY INSTR ADDR/SELECT ADDR Function (See figure 3-225.)

If the DISPLAY switch is set to INSTR ADDR in PCP2 idle state, the CPU reads into the D-register the contents of the memory location pointed to by the P-register. If the DISPLAY switch is set to SELECT ADDR in PCP2 wait state, the CPU reads into the D-register the contents of the memory location whose value is equal to the value of the 17 SELECT ADDRESS switches. The logic sequence for the DISPLAY INSTR ADDR/SELECT ADDR function is given in table 3-104.

3-121 INSTR ADDR HOLD/INCREMENT Function

During normal program execution the INSTR ADDR switch is in the center position, and signals KAHOLD and KINCRE/B are false. When this switch is in the center position the contents of the P-register are incremented at the end of each instruction execution (ENDE).

With the INSTR ADDR switch in HOLD and the COMPUTE switch set to RUN, the CPU will repeatedly execute the instruction addressed by the INSTRUCTION ADDRESS display (P-register), and will not sequence to the next instruction.

With the COMPUTE switch in IDLE, moving the INSTR ADDR switch to INCREMENT will cause the current instruction address to be counted up by one, as shown in figure 3-226 and the contents of this updated address to be displayed in the DISPLAY indicators. Thus, the operator can display the contents of sequential memory locations by repeatedly moving the INSTR ADDR switch to INCREMENT. The logic sequence for the INCREMENT function is given in table 3-105.

3-122 Clear Memory Function

When the CPU RESET/CLEAR and the SYSTEM RESET/CLEAR pushbuttons are pressed sumultaneously, signal CLEAR MEM is true, and all core memory locations are cleared to zero.

Table 3-102. DATA ENTER/CLEAR Sequence

PCP Phase	Function Performed	Signals Involved	Comments
PCP2	Go to PCP3	S/PCP3 = (NHALT KAS/1 KAS/2 + ...) PCP2/1 NIOCON NDCSTOP	
PCP2/3	One clock long 0's —/→ A	AX = AXZ + ... AXZ = PCP3 + ...	
PCP4	One clock long KCLEAR/B ⇒ (S0-S31) —/→ (D0-D31) KENTER/B ⇒ Data SW —/→ A Preset S ——→ C Preset A or D ——→ S	 DXS = PCP4 KCLEAR/B + ... AXK = PCP4 KENTER/B + ... (S/CXS) = PCP4 KENTER/B + ... S/SXAORD = PCP4 + ...	
PCP5	One clock long Enter Data ⇒ (S0-S31) —/→ (D0-D31) (S0-S31) ——→ (C0-C31) (A0-A31) or (D0-D31) ——→ (S0-S31) Preset B ——→ S	 DXS = PCP5 KENTER/B + ... S/SXB = PCP5 NBRPCP5 + ...	 Preset in PCP4
PCP6	One clock long (B0-B31) ——→ (S0-S31) Set HALT flip-flop	 S/HALT = PCP6 + ...	 Preset In PCP5
PCP1	One clock long Preset D ——→ S for PCP2	 (S/SXD) = PCP1 + ...	
PCP2	Idle (D0-D31) ——→ (S0-S31) ——→ display indicators Reset HALT flip-flop	 R/HALT = PCP2 NKAS/B + ...	 Preset during PCP1 No control switch action

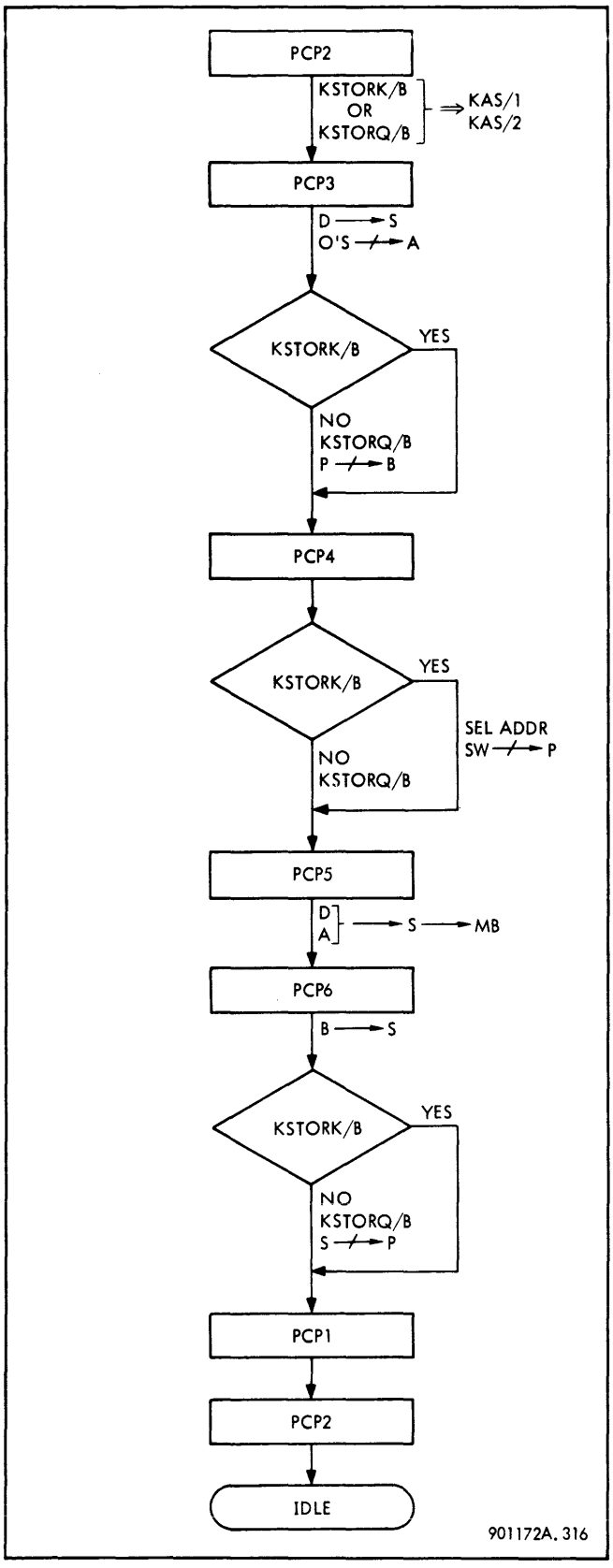


Figure 3-224. STORE INSTR ADDR/STORE SELECT ADDR, Flow Diagram

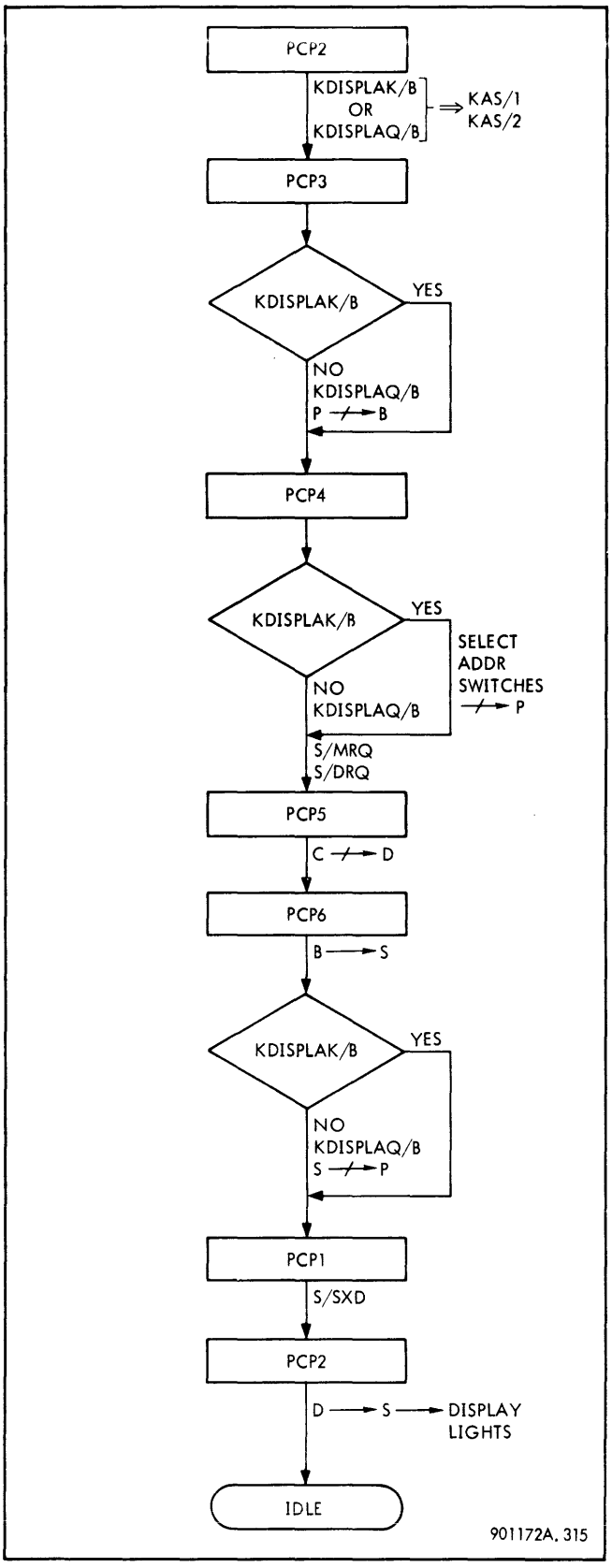


Figure 3-225. DISPLAY SELECT ADDR/DISPLAY INSTR ADDR, Flow Diagram

Table 3-103. Store INSTR ADDR/STORE SELECT ADDR Sequence

PCP Phase	Function Performed	Signals Involved	Comments
PCP2	Enable signal (S/SXD)	(S/SXD) = PCP2/1 NRESET/C NKCLRPSW/B NIOCON	Preset adder for D ⟶ S in PCP3
	Go to PCP3	S/PCP3 = (NHALT KAS/1 KAS/2 + ...) PCP2/1 NIOCON NDCSTOP	
PCP2/3	One clock long (D0-D31) ⟶ (S0-S31)	Adder logic preset in PCP2	For display PRESET during PCP2
	0's ⟶ A	AX = AXZ + ... AXZ = PCP3 + ...	
	(P16-P31) ⟶/⟶ (B16-B31)	BXP = PCP3 SWK5 + ... R/PCP2 = PCP3	Transfer address in PSW1 to B-register
PCP4	Enable signal (S/SXAORD)	(S/SXAORD) = PCP4 + ...	Preset adder logic for A or D ⟶ S in PCP5
	STORE SELECT ADDR ⟹		
	Address switches ⟶/⟶ P	PXK = PCP4 SWK5	Transfer address switch outputs to P-register
	Set flip-flop MBXS	S/MBXS = (S/MBXS) + ... (S/MBXS) = PCP4 SWK3 + ...	Prepare for memory write
	Set flip-flop DRQ	S/DRQ = (S/MBXS) + ...	Data request. Inhibits transmission of another clock until data release received from memory
PCP5	One clock long (A0-A31) or (D0-D31) ⟶ (S0-S31)	Adder logic preset in PCP4	Store address in D-register in instruction address or address pointed to by SELECT ADDRESS switches
	(S0-S31) ⟶ (MB0-MB31)	MBXS set in PCP4	
	Preset B ⟶ S for PCP6	S/SXB = PCP5 NBRPCP5 + ...	
PCP6	One clock long (B0-B31) ⟶ (S0-S31)	Preset in PCP5	Return program address to P-register
	(S0-S31) ⟶/⟶ (P0-P31)	PXS = PCP6 SWK5 + ...	

Table 3-104. DISPLAY INSTR ADDR/DISPLAY SELECT ADDR Sequence

PCP Phase	Function Performed	Signals Involved	Comments
PCP2	(D0-D31) ⟶ (S0-S31) (S0-S31) ⟶ display indicators Go to PCP3	(S/SXD) = PCP2/1 NRESET/C NKCLRPSW/B NIOCON S/PCP3 = (NHALT KAS/1 KAS/2 + ...) PCP2/1 NIOCON NDCSTOP	For display
PCP3	(P0-P31) ⟶/⟶ (B0-B31)	BXP = PCP3 SWK5 + ...	Save instruction address in P-register
PCP4	DISPLAY SELECT ⟹ Enable signal PXK Set flip-flop MRQ Set flip-flop DRQ	 PXK = PCP4 SWK5 S/MRQ/2 = PCP4 SWK4 + ... S/DRQ = S/MRQ/2 + ...	 Transfer to P-register address selected by SELECT ADDR switches Request for memory cycle Data request. Inhibits transmission of another clock until data release received from memory
PCP5	One clock long (MB0-MB31) ⟶ (C0-C31) (C0-C31) ⟶/⟶ (D0-D31) Preset B ⟶ S	 CXMB = DG (data gate) DXC = PCP5 SWK4 + ... S/SXB = PCP5 NBRPCP5 + ...	Read contents of program address or address pointed to by SELECT ADDRESS switches from memory into D-register by way of C-register
PCP6	One clock long (B0-B31) ⟶ (S0-S31) (S0-S31) ⟶ (P0-P31)	 PXS = PCP6 SWK5 + ...	Return program address to P-register if replaced by selected address
PCP1	One clock long Preset D ⟶ S	S/SXD = PCP1 + ...	Prepare to display D-register contents
PCP2	Idle	R/HALT = PCP2 NKAS/B + ...	

Table 3-106. Clear Memory Sequence

Phase	Function Performed	Signals Involved	Comments
	Switches and signals involved: CPU RESET/CLEAR ⟹ KCPURESET, KCPURESET/B and SYSTEM RESET/CLEAR ⟹ KSYSR, KSYSR/B COMPUTE in IDLE ⟹ NKAS/B, NKRUN (necessary for clear memory operation)		
PCP2	Idle phase — sustained until CPU RESET/CLEAR and SYSTEM RESET/CLEAR are pressed		
	Reset HALT flip-flop	R/HALT = (R/HALT) (R/HALT) = PCP2 NKAS/B	
	Inhibit interrupts during idle phase	(S/INTRAP) = N(PCP2 NKRUN)	Inhibit setting of first of interrupt phase sequence flip-flops
	Enable signal (S/SXD)	(S/SXD) = PCP2/1 NRESET/C NKCLRPSW/B NIOCON + ...	Preset adder logic for D ⟶ S in PCP3
	Set flip-flop PCP3	S/PCP3 = (S/PCP3) (S/PCP3) = PCP2/1 NIOCON NDCSTOP (CLEARMEM + ...) CLEARMEM = NKAS/B KSYSR/B KCPURESET/B	Go to PCP phase 3
PCP3	One clock long		
	(D0-D31) ⟶ (S0-S31)	Adder logic preset in PCP2	Transfer next instruction in D-register to B-register
	(S0-S31) ⟶/ (B0-B31)	BXS = PCP3 SWK12 + ... SWK12 = SWK1 + SWK2 SWK1 = CLEARMEM + ...	
	NIOFS = Reset IOSC	R/IOSC = PCP3 NIOFS	Cancel internal I/O service call enable
	0 ⟶/ (A0-A31)	AX = AXZ + ... AXZ = PCP3 + ...	Clear A-register
	Enable signal (S/SXA)	(S/SXA) = PCP3 + ...	Preset adder logic for A ⟶ S in PCP4

(Continued)

Table 3-106. Clear Memory Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PCP4	One clock long		
	Enable signal (S/SXAORD)	(S/SXAORD) = PCP4 + ...	Preset adder logic for A or D ⟶ S in PCP5
	(A0-A31) ⟶ (S0-S31)	Adder logic preset in PCP3	Clear D-register by transferring zeros in A-register to D-register
	(S0-S31) ⟶/⟶ (D0-D31)	DXS = PCP4 SWK12 + ...	
	Set flip-flop MBXS	S/MBXS = (S/MBXS)	Prepare for memory write
		(S/MBXS) = PCP4 SWK1	
	Set flip-flop DRQ	S/DRQ = (S/MBXS) + ...	Data request. Inhibits transmission of another clock until data release received from core memory
PCP5	Sustained until switches released		
	(A0-A31) or (D0-D31) ⟶ (S0-S31)	Adder logic preset in PCP4	Place zeros on memory bus (A- and D-registers both contain zeros)
	(S0-S31) ⟶/⟶ (MB0-MB31)	Memory write preset in PCP4	
	P + 1 ⟶/⟶ P	PUC31 = PCP5 SWK1	Add 1 to P-register each PCP5 to address all memory locations
	Set flip-flop MBXS	S/MBXS = (S/MBXS)	Preset for memory write to write zeros into each addressed memory location as PCP5 repeats
		(S/MBXS) = PCP5 SWK1 + ...	
	Set flip-flop DRQ	(S/DRQ) = (S/MBXS) + ...	Data request. Inhibits transmission of another clock until data release received from core memory with each memory access

(Continued)

Table 3-106. Clear Memory Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PCP5 (Cont)	Enable signal (S/SXA)	(S/SXA) = BRPCP5 + ...	Preset adder logic for A ⟶ S with each repetition of PCP5
	Enable signal (S/SXB)	(S/SXB) = PCP5 NBRPCP5	Preset adder logic for B ⟶ S in last PCP5
	Repeat PCP5 as long as both switches are pressed	BRPCP5 = PCP5 CLEARMEM + ...	All memory locations are cleared while switches are pressed
	Set flip-flop PCP6 when switches are released	S/PCP6 = PCP5 NBRPCP5	Go to PCP phase 6
PCP6	(B0-B31) ⟶ (S0-S31)	Adder logic preset in PCP5	Return next instruction to D-register
	(S0-S31) ⟶/ (D0-D31)	DXS = PCP6 SWK12 + ...	
	Set HALT flip-flop	S/HALT = (S/HALT)	Halt computer
		(S/HALT) = PCP6 + ...	
PCP1	Enable Signal (S/SXD)	(S/SXD) = PCP1 + ...	Preset adder logic for D ⟶ S in PCP2
PCP2	Idle phase (D0-D31) ⟶ (S0-S31) ⟶ DISPLAY indicators	Preset in PCP1	

Table 3-107. Load Sequence

Phase	Function Performed	Signals Involved	Comments
PCP2	Signals true:		
	KFILL/B, KAS/1, KAS/2, SWK1	SWK1 = KFILL/B + ...	
	Idle phase		
	Reset HALT flip-flop	R/HALT = (R/HALT)	
		(R/HALT) = PCP2 NKAS/B + ...	
	Inhibit signal (S/INTRAP)	(S/INTRAP) = N(PCP2 NKRUN)	Inhibit interrupts during idle phase

(Continued)

Table 3-107. Load Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PCP2 (Cont.)	Enable signal (S/SXD)	(S/SXD) = PCP2/1 NRESET/C NKCLRPSW/B NIOCON	Preset adder logic for D ⟶ S in PCP3
	Set flip-flop PCP3	S/PCP3 = PCP2/1 NIOCON NDCSTOP (NHALT KAS/1 KAS/2 + ...)	Go to PCP phase 3
PCP3	One clock long		
	(D0-D31) ⟶ (S0-S31)	Adder logic preset in PCP2	Save next instruction in B-register
	(S0-S31) ⟶/ (B0-B31)	BXS = PCP3 SWK12	
		SWK12 = SWK1 + SWK2	
	NIOFS ⟹ Reset IOSC	R/IOSC = PCP3 NIOFS	Reset IO service call if no function strobe
	Enable signal (S/SXA)	(S/SXA) = PCP3 + ...	Preset adder logic for S ⟶ A in PCP4
		R/PCP2 = PCP3	
PCP4	One clock long		
	Set flip-flop MBXS	S/MBXS = (S/MBXS)	Prepare for memory write
		(S/MBXS) = PCP4 SWK1	
	Set flip-flop MRQ	S/MRQ = (S/MBXS) + ...	Request for core memory cycle
	Set flip-flop DRQ	S/DRQ = (S/MBXS) + ...	Data request. Inhibits transmission of another clock until data release received from memory
	(A0-A31) ⟶ (S0-S31)	Adder logic preset in PCP3	Zeros transferred from A-register to D-register
	(S0-S31) ⟶/ (D0-D31)	DXS = PCP4 SWK12	
	Enable signal (S/SXAORD)	(S/SXAORD) = PCP4 + ...	Preset adder logic for A or D ⟶ S in PCP5

(Continued)

Table 3-107. Load Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PCP4 (Cont.)	Set X'20' in P-register	S/P26 = PCP4 KFILL/B R/P15-P31 = PX PX = PCP4 KFILL/B	Address location X'20' to load first word of bootstrap program
PCP5	One clock long		
	(A0L-A31L) —/→ (A0-A31)	AXLOAD = PCP5 KFILL/B	Load bootstrap program into A-register. A0L-A-31L logic decodes P-register contents to set correct instruction in A-register for each bootstrap address from X'20' to X'29'. When (P) = X'24', indicating that next location to be loaded is X'25', A21L-A31L contain outputs of UNIT ADDRESS switches, KUA21-KUA31
	First pass ⟹		
	(A0-A31) or (D0-D31) ⟶ (S0-S31)	Adder logic set in PCP4	Write zeros into location X'20'
	Not last pass ⟹ Enable signal (S/SXA)	(S/SXA) = BRPCP5	Preset adder logic for A ⟶ S in next PCP5
	Not first pass ⟹ (A0-A31) —/→ (S0-S31)	Adder logic preset in previous PCP5	Place successive A-register contents on sum bus to be written in memory
	(S0-S31) ⟶ (MB0-MB31)	Memory write preset in PCP4 or previous PCP5	Load bootstrap program into memory
	Set flip-flop MBXS	S/MBXS = (S/MBXS) (S/MBXS) = PCP5 SWK1 + ...	Prepare for memory write in next PCP5
	Set flip-flop MRQ	S/MRQ = (S/MRQ/2) + ... (S/MRQ/2) = BRPCP5 + ...	Request for core memory cycle in next PCP5
	Set flip-flop DRQ	S/DRQ = (S/DRQ) (S/DRQ) = (S/MBXS) + ...	Data request. Inhibits transmission of another clock until data release received from memory

(Continued)

Table 3-107. Load Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
PCP5 (Cont.)	P + 1 → P	PUC31 = PCP5 SWK1	Add one to P-register. Contents during each loop through PCP5 to address successive bootstrap locations
	Sustain PCP5 until P-register contains X'00000029'	BRPCP5 = PCP5 N(P28 P31) KFILL/B	Location X'29' is last bootstrap location
	Last pass ⟹ Set flip-flop PCP6	S/PCP6 = PCP5 NBRPCP5	Go to PCP phase 6
PCP6	(B0-B31) → (S0-S31)	Adder logic preset in last PCP5	X'02000000' into D-register making next instruction a "no operation" (LCFI with zeros in bits 10 and 11)
	(S0-S31) → (D0-D31)	DXS = PCP6 + ...	
	Set HALT flip-flop	S/HALT = (S/HALT) (S/HALT) = PCP6 + ...	
	Set X'25' in P-register	S/P26, S/P29, S/P31 = RESET/C	Location X'26' is first executed instruction in bootstrap program.
		RESET/C = PCP6 KFILL/B + ...	
		R/P15-P31 = PX PX = PCP6 KFILL/B	One is added to P-register contents in PH10 when COMPUTE switch is set to RUN
	Set X'02000000' in D-register	S/D6 = RESET /C RESET/C = PCP6 KFILL/B	Ensure that no operation instruction is in D-register
PCP1	Enable signal (S/SXD)	(S/SXD) = PCP1 + ...	Preset adder logic for D → S in PCP2
PCP2	Idle (D0-D31) → (S0-S31) → DISPLAY indicators	Preset in PCP1	

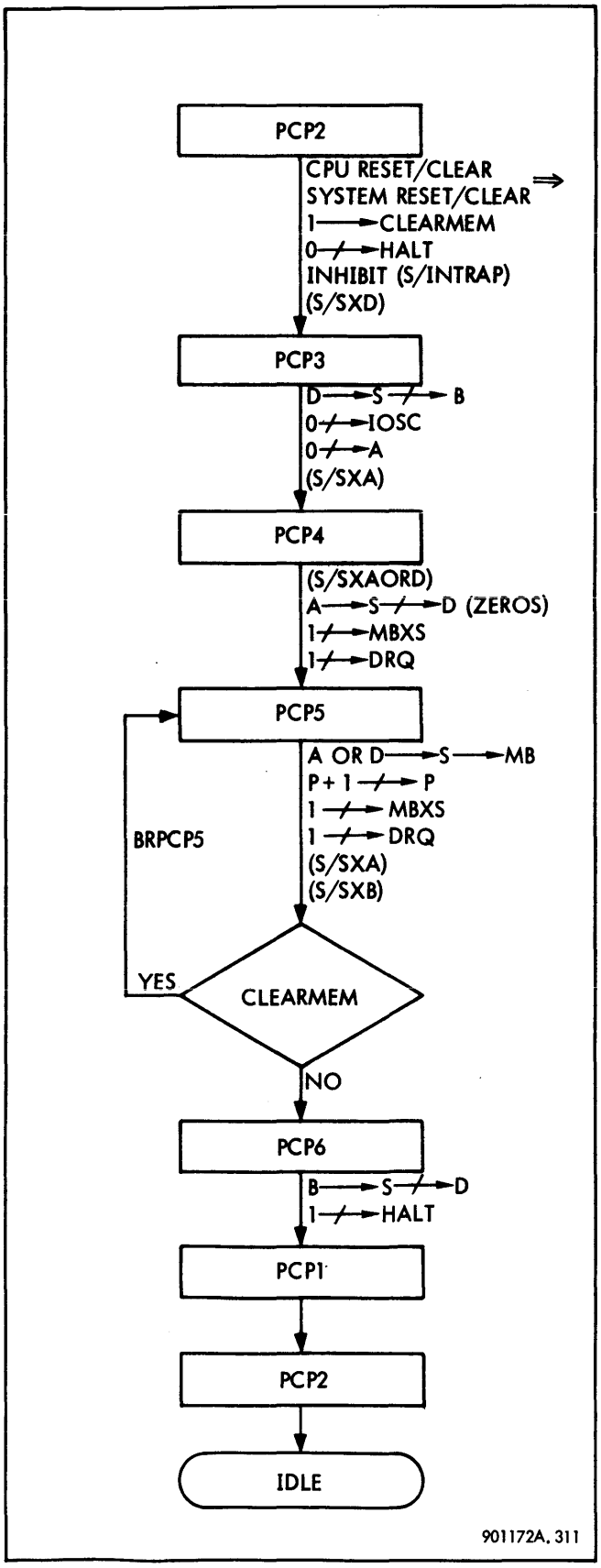


Figure 3-227. Clear Memory, Flow Diagram

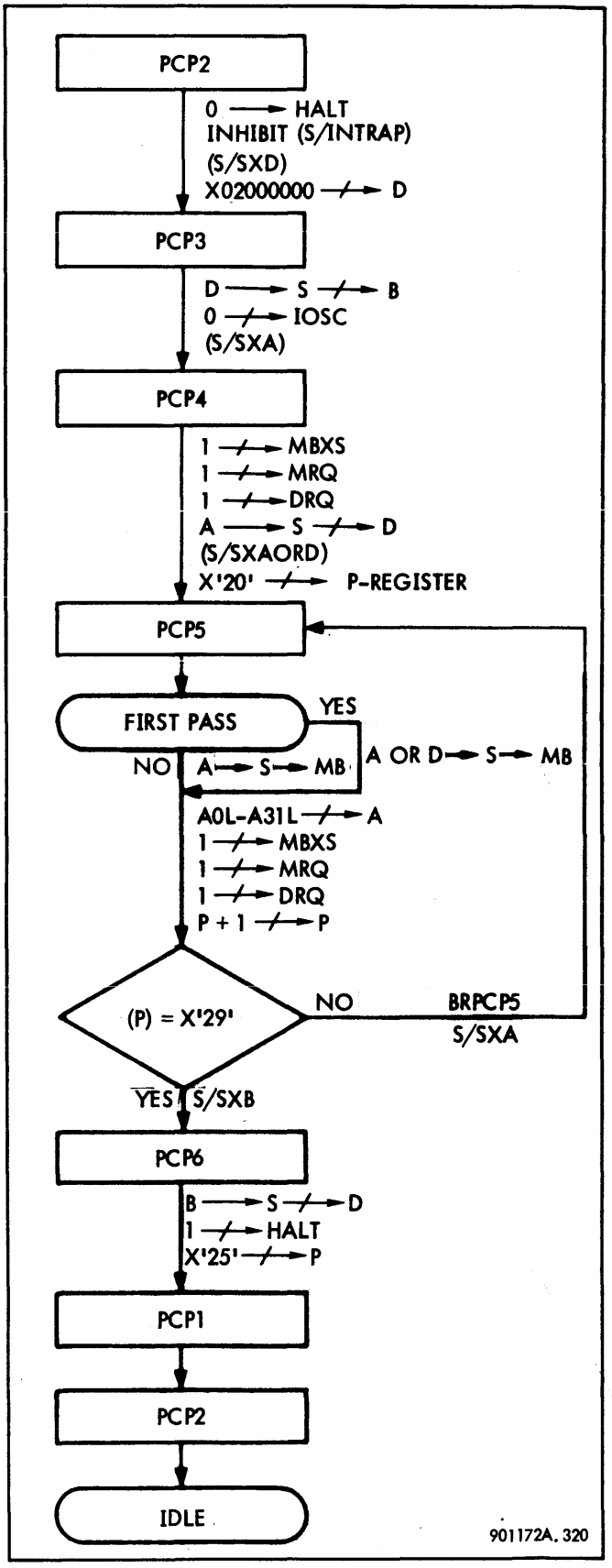


Figure 3-228. Load, Flow Diagram

3-126 INTEGRAL INPUT/OUTPUT PROCESSOR

3-127 General

The Sigma 5 integral IOP controls data transfer between core memory and one or more peripheral devices. To do this, the integral IOP uses standard CPU registers and circuits together with registers and circuits which have only an I/O function. Since portions of the integral IOP are an integral part of the CPU, the term integral IOP refers to a functional rather than a physical unit.

Figure 3-229 shows the functional circuit groups included in the integral IOP. Blocks with heavy lines denote circuit groups used only for I/O purposes.

3-128 Address and Priority Assignment

IOP address 000 is assigned exclusively to the integral IOP. Interrupt priority is determined by the relative position of an IOP within the interrupt priority chain; the integral IOP may be placed at the discretion of the user, at any level within the priority chain. The integral IOP is not involved in memory priority, since it cannot access memory independently, but only by normal CPU channels.

3-129 Capabilities

Eight I/O channels are standard equipment with the Sigma 5 integral IOP; that is, the integral IOP can service eight device controllers. Additional I/O channels are available, in 8-channel increments, as an option. The maximum number of I/O channels, including the eight standard channels, is 32. For the remainder of this discussion it is assumed, unless stated otherwise, that the integral IOP has a full complement of 32 I/O channels.

Each increment of eight I/O channels is termed a group, and labeled 1 through 4. Of these, group 1 controls multidevice device controllers, each capable of handling 16 devices. Therefore, the maximum number of devices that can be accommodated by the integral IOP is 152, as illustrated in figure 3-230.

3-130 I/O Fast Memory IOFM

GENERAL. The I/O fast memory consists of 32 channel registers, distributed among four 8-channel groups. Each group is made up of five FT25 fast memory modules. Group 1 is typical and is illustrated in figure 3-231. Each channel register is dedicated to a device controller and contains 80 bits. To form the 80 bits, each channel register is distributed twice across the five FT25 modules; 40 bits are contained in the top half channel and 40 bits in the bottom half channel. Channel 0 in figure 3-231 is detailed to show byte distribution, and channel 7 is detailed to show byte information assignment. The bit index in figure 3-231 defines bit information assignment.

Each channel register is divided into four memory access areas, labeled 0 through 3; area selection is controlled by address bits IOFR8 and IOFR9. Areas 0 and 1 each contain four bytes in a channel (bytes 0 through 3), and areas 2 and 3 each contain one byte in a channel (byte 4). Group and channel selection is controlled by address bits LI03 through LI07 according to the codes shown in the group and channel selection charts in figure 3-231.

FT25 FAST MEMORY MODULE. Figure 3-232 is a simplified logic diagram of a typical FT25 fast memory module as used in the I/O fast memory. The module depicted in figure 3-232 represents byte 0 for both upper and lower half channels of channels 0 through 7 in group 1. The basic unit of memory is memory element SDS 304. There are 16 memory elements in an FT25 module, labeled A1 through A16, each with an 8-bit storage capacity. Data distribution is as follows: memory element A1 stores eight bit 0's, one for each of upper half channels 0 through 7; memory element A2 stores bit 1's for upper half channels 0 through 7; memory element A9 stores bit 0's for lower half channels 0 through 7; and so on.

To write into the fast memory module, the information code is placed on the fast memory input (RW) lines and applied as data inputs. After entering the module, the input lines are changed to write I/O lines with designations applicable to each module. In the example shown in figure 3-232 the input line designations are W/IO1B0/X, indicating that this module represents byte 0's for all 16 half channels in group 1. When clock signal K/IOB0/0 and I/O enable term IOFM are both high, the information contained in the data input lines is stored in this module within the half channel specified by the address lines. The address configuration shown in figure 3-232 selects either half channel of channel 0 in group 1; the state of IOFR9 determines which half channel is selected.

Data is read out of the module without the use of the clock signal. When I/O enable term IOFM is high, the contents of the half channel selected by the address lines are placed on data output lines RR0 through RR7 and become available for use.

In figure 3-232 the data input lines shown are RW0 through RW7. These same input lines are connected to three additional FT25 modules (byte 0 group 2, byte 0 group 3, and byte 0 group 4). Similarly, input lines RW8 through RW15 are connected to the four byte 1 modules and input lines RW24 through RW31 are connected to the four byte 3 modules. Data input lines RW16 through RW23 are shared by the four byte 2 modules with the four byte 4 modules; when address bit IOFR8 is false, the byte 4 modules cannot be accessed; when address bit IOFR8 is true, only the byte 4 modules can be accessed.

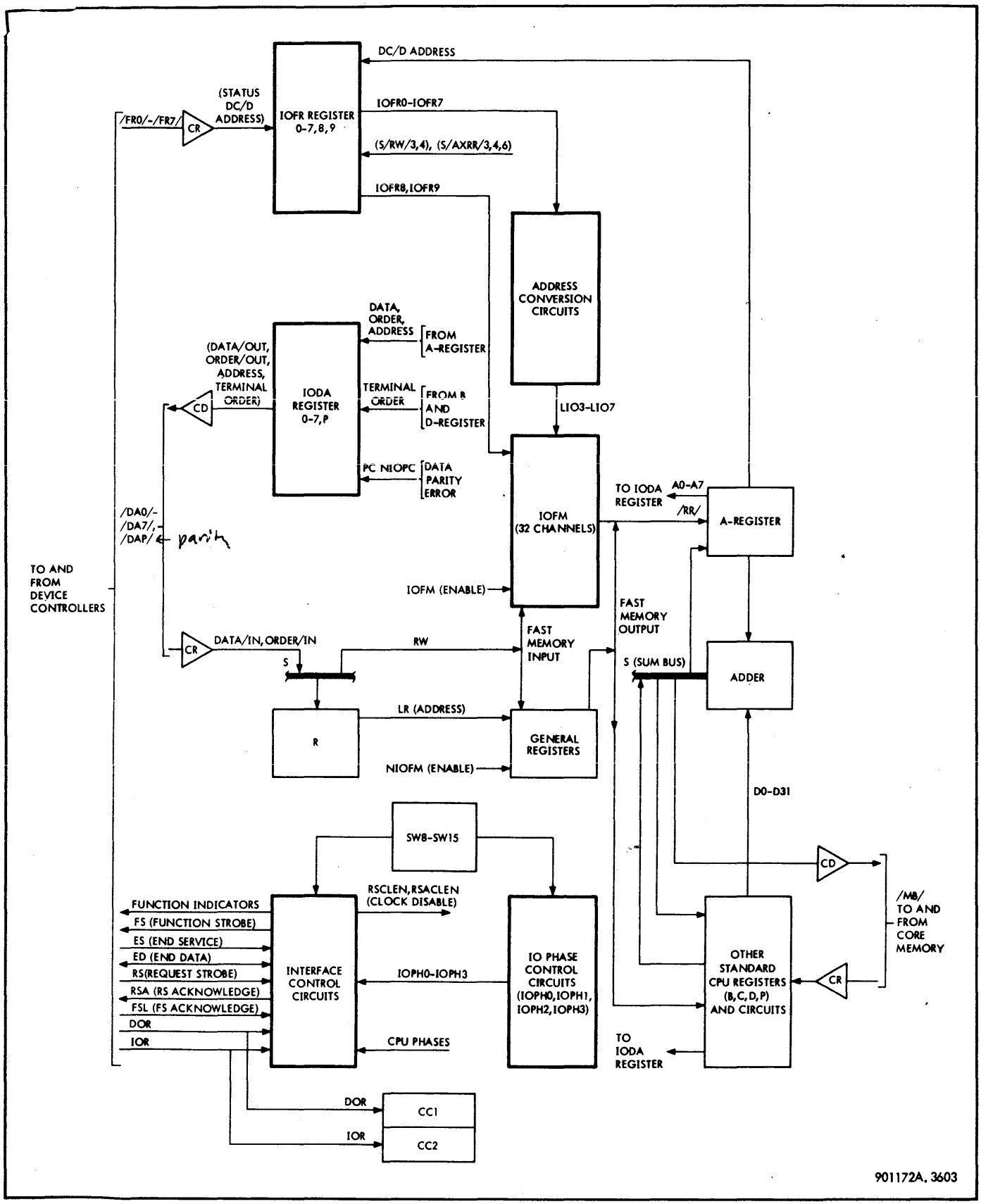


Figure 3-229. Integral IOP, Functional Block Diagram

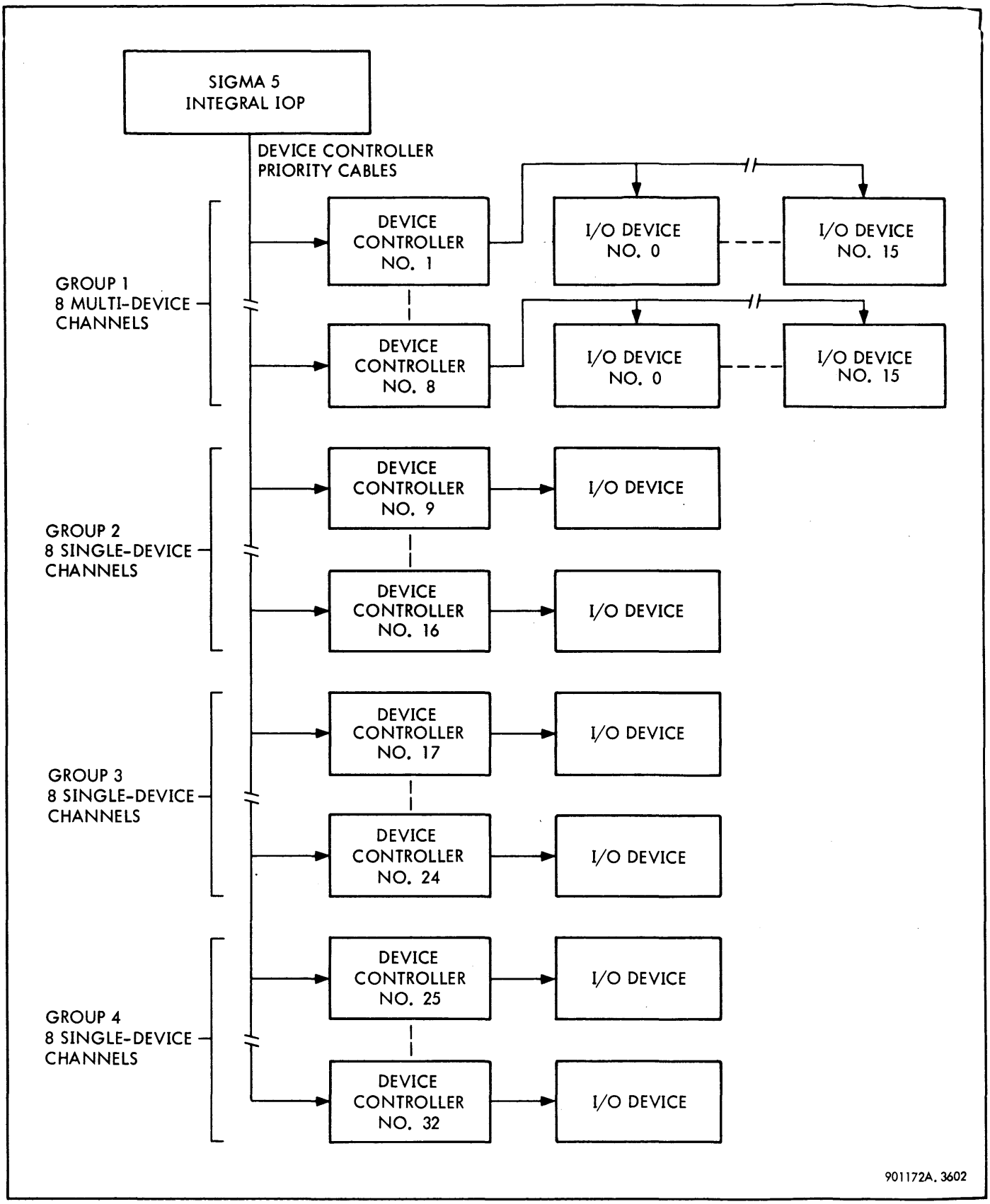
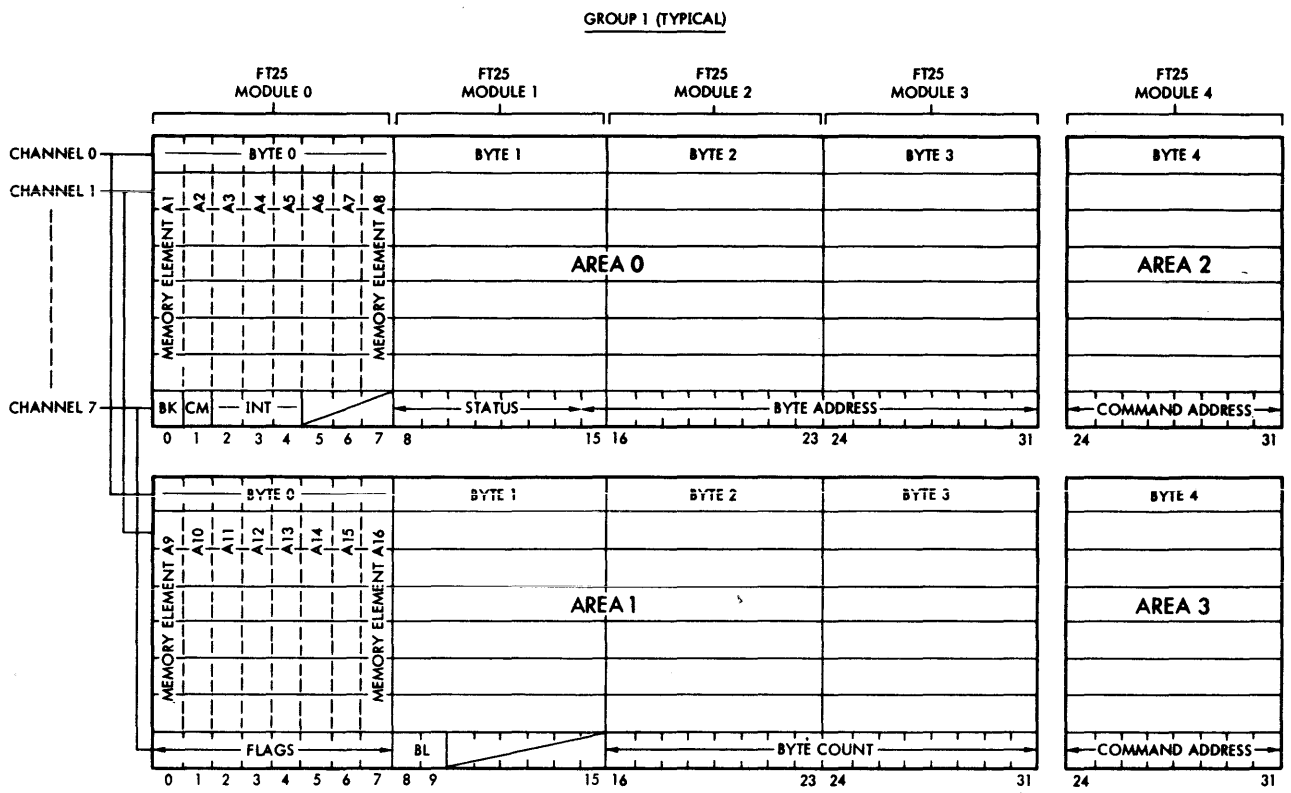


Figure 3-230. Integral IOP, Device Controller/Device Configuration



GROUP SELECTION

GROUP	SIGNAL CODE	
	LI03	LI04
1	0	0
2	0	1
3	1	0
4	1	1

CHANNEL SELECTION

CHANNEL	SIGNAL CODE		
	LI05	LI06	LI07
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

AREA SELECTION

AREA	SIGNAL CODE	
	IOFR8	IOFR9
0	0	0
1	0	1
2	1	0
3	1	1

BIT INDEX

AREA 0

BITS	
0	BK - BACKWARD READ
1	CM - CHAINING MODIFIER
2	ZERO COUNT
3	CHANNEL END
4	UNUSUAL END
5-7	NOT USED
8	INCORRECT LENGTH
9	TRANSMISSION ERROR
10	TRANSMISSION MEMORY ERROR
12	CHAINING MEMORY ERROR
13	CHANNEL END
14	IOP HALT
15-31	BYTE ADDRESS (MS BIT = 15, LS BIT = 31)

INTERRUPT CAUSES: bits 2, 3, 4

STATUS (OPERATIONAL): bits 8, 9, 10, 12, 13, 14

AREA 1

BITS	
0	DATA CHAINING
1	INTERRUPT ON ZERO COUNT
2	COMMAND CHAINING
3	INTERRUPT ON CHANNEL END
4	HALT ON TRANSMISSION ERROR
5	INTERRUPT ON UNUSUAL END
6	SUPPRESS INCORRECT LENGTH
7	SKIP
8, 9	BYTE LEVEL (BIT 8 = 2^1, BIT 9 = 2^0)
10-15	NOT USED
16-31	BYTE COUNT (MS BIT = 15, LS BIT = 31)

FLAGS: bits 0, 1, 2, 3, 4, 5, 6, 7

AREA 2

BITS	
24-31	COMMAND ADDRESS, MS BYTE (MS BIT = 24, LS BIT = 31)

AREA 3

BITS	
24-31	COMMAND ADDRESS, LS BYTE (MS BIT = 24, LS BIT = 31)

901172A.3600

Figure 3-231. I/O Fast Memory, Group Organization

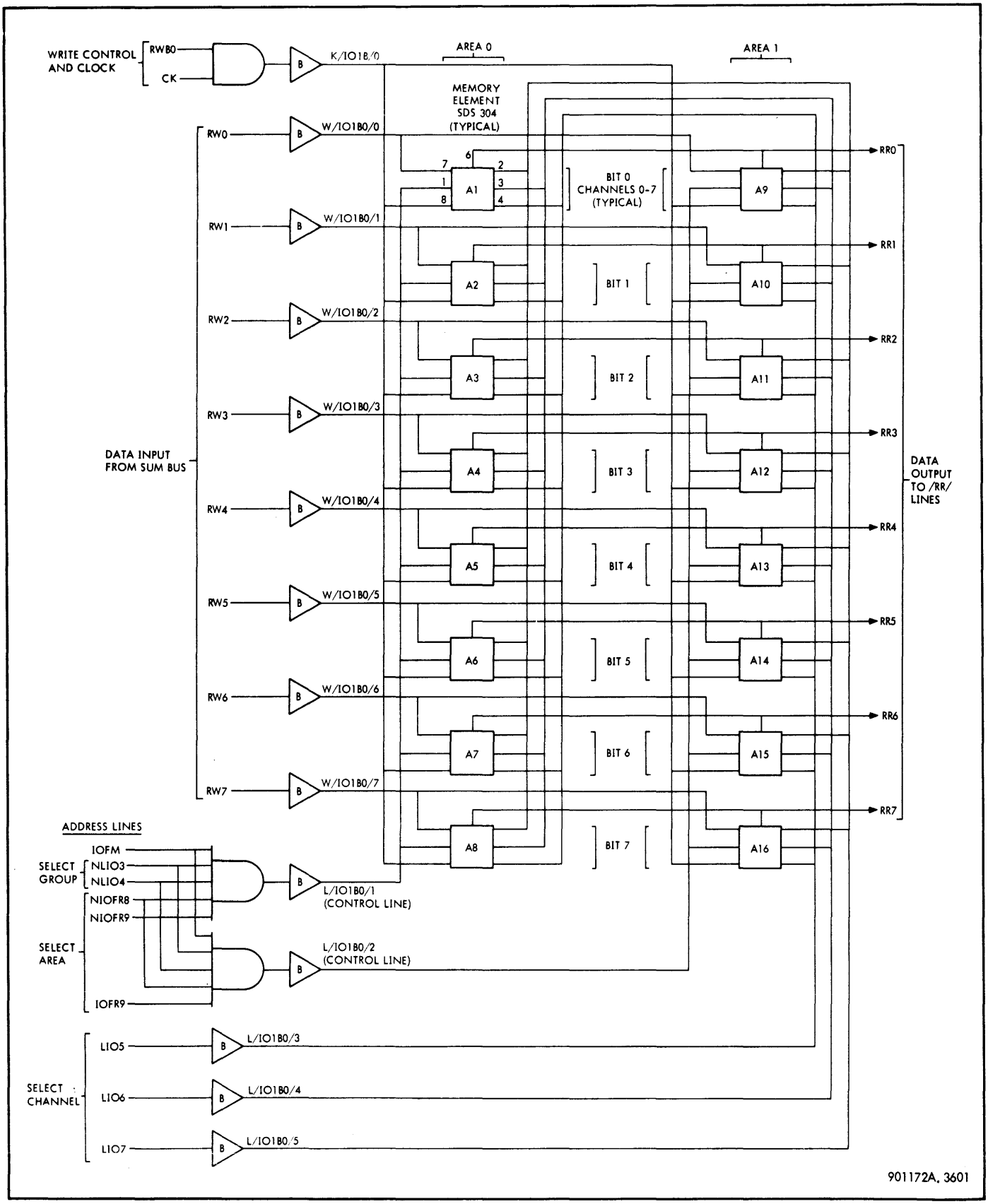


Figure 3-232. Fast Memory Module FT25, Logic Diagram

3-131 I/O Address Register IOFR

The I/O address register is a 10-bit flip-flop register used for selecting the group number, channel number, and area in the IOFM register. Group and channel number selection are controlled by the eight high order bits IOFR0 through IOFR7. Area selection is controlled by the two low order bits IOFR8 and IOFR9.

Input to the eight high order bits is obtained either from the A-register or from the function response lines, as follows:

S/IOFR0 = A0 IOFRXA + FR0 IOFRXFR
⋮ ⋮ ⋮
S/IOFR7 = A7 IOFRXA + FR7 IOFRXFR

C/IOFR0-IOFR7 = CL

R/IOFR0-IOFR7 = IOFRX

IOFRXA = FAIO PH3 (Execution phase of an SIO, HIO, TIO, or TDV instruction when A-register contains device controller/device address)

IOFRXFR = IOENIN + FUAIO P5·8·7

IOENIN = Service call processing phase when device controller places its address on the function response lines FR0-FR7 in response to an ASC

FUAIO P5·8·7 = Execution phase for an AIO instruction when device controller places its address on function response lines FR0-FR7 in response to an interrupt query

IOFRX = FAIO PH3 + FUAIO P5·8·7 + IOENIN

The two low order bits, IOFR8 and IOFR9, are controlled by general transfer terms used for transferring data to and from the IOFM register, as follows:

S/IOFR8 = (S/IOFR8)

(S/IOFR8) = (S/AXRR/4) + (SRW/4)

C/IOFR8 = CL

R/IOFR8 = ...

S/IOFR9 = (S/IOFR9) IOPOP

(S/IOFR9) = (S/AXRR/3) + (SRW/3) + (S/AXRR/6)

C/IOFR9 = CL

R/IOFR9 = ...

(S/AXRR/3) = IOPH1 SW8 ORDSW4
+ IOPH1 SW8 NIODC DASW4
+ IOPH0 SW9
+ IOPH0 SW8 IOPH10
+ IOPH0 SW11 IFAST

(S/AXRR/4) = FAIO/1 PH5 (SW11 + SW10 NR31)
+ IOPH0 SW12 DOR IOR
+ IOPH1 SW8 DASW4 IODC
+ IOPH3 SW8

(S/AXRR/6) = FAIO/1 PH5 (SW11 + SW8 NSW7 NFUMH)
+ IOPH3 SW8

(S/RW/3) = IOPH0 SW10 NIOPH10
+ FAIO PH9 SW0 VALST
+ IOPH1 SW12
+ IOPH1 SW11 NSW3 NIFAM
+ IOPH3 SW12

(S/RW/4) = FAIO PH5 SW13 VALST
+ FAIO PH9 SW0 VALST
+ IOPH3 (SW11 + SW12)

3-132 I/O Data Register IODA

The I/O data register is a 9-bit flip-flop register used for transmitting one byte of information (data, address, order, or terminal order) to the device controller. Eight of the nine bits, IODA0 through IODA7, store the actual byte of information to be transmitted. The ninth, IODAP, is the data parity bit.

Input to the four high order bits, IODA0 through IODA3, is obtained either from the A-register or, in the case of a terminal order, from flags and status bits stored in the D- and B-registers, as follows:

S/IODA0 = (S/IODA0) + TODATA SW0 D1 + A0 IODAXA

(S/IODA0) = (S/B3) + (S/B4)

(S/B3) = TORDIN (...)

(S/B4) = TORDIN (...)

TODATA = Terminal order condition during the phase sequence of any of the four service cycles

TORDIN = Terminal order condition during an order-in phase sequence

IODAXA = Term used for transferring data from the A-register
= IOPH3 SW15 NSW4 + FAIO PH3 + DATAOUT IOPH0 SW13 ND7

IOPH3 SW15 NSW4 = Phase during order-out phase sequence when A-register stores order

FAIO PH3 = Execution phase during an I/O instruction when A-register stores device controller/device address

DATAOUT IOPH3 SW13 ND7 = Phase during data-out phase sequence when A-register stores data, and the skip flag is not high

R/IODA0 = IODAX

IODAX = (R/IODA)

(R/IODA) = FUAIO PH5 SW8 NSW7 + TODATA + IOENIN + IODAXA

C/IODA0 = CL

S/IODA1 = TODATA SW0 ND0 + A1 IODAXA

C/IODA1 = CL

R/IODA1 = IODAX

S/IODA2 = TODATA D2 + A2 IODAXA

C/IODA2 = CL

R/IODA2 = IODAXA

S/IODA3 = TODATA B4 + A3 IODAXA

C/IODA3 = CL

R/IODA3 = IODAXA

Input to bits IODA4 through IODA7 is obtained from the A-register only:

S/IODA4 = A4 IODAXA
⋮ ⋮
S/IODA7 = A7 IODAXA

R/IODA4 = R/IODA5-IODA7 = IODAXA

C/IODA4 = C/IODA5-IODA7 = CL

Parity flip-flop IODAP has a dual function: one for data-out and another for data-in. During data-out, odd parity is established by parity generator term IOPG. Flip-flop IODAP assumes the state of IOPG and, accordingly, drives the data parity line /DAP/. During data-in, flip-flop IODAP is set if the byte received from the device controller does not pass parity and a parity check is required. The complete logic for IODAP is as follows:

S/IODAP = IOPH0 SW13 DATAOUT IOPG + IOPH0 SW14 DATAIN NIOPC PC

IOPG = Sum of true data bits is even

NIOPC = Byte from device controller did not pass parity

PC = Parity check required, as specified by device controller

R/IODAP = IOPH0 SW13

C/IODAP = CL

3-133 Address Conversion Circuits

The address conversion circuits sample bits IOFR0 through IOFR7 of the address register and, accordingly, provide a 5-bit output, LIO3 through LIO7, to the I/O fast memory. Bits LIO3 and LIO4 are used to select one of four channel groups, and bits LIO5 through LIO7 select one of eight channels within the selected group. Bits IOFR8 and IOFR9 of the address register are applied directly to the I/O fast memory and are used to select the proper area. The bit conversion logic is as follows:

LIO3 = IOFR3 NIOFR0

NIOFR0 = Specifies a single-device device controller

LIO4 = IOFR4 NIOFR0

LIO5 = IOFR5 NIOFR0 + IOFR1 IOFR0

IOFR0 = Specifies a multidevice device controller

LIO6 = IOFR6 NIOFR0 + IOFR2 IOFR0

LIO7 = IOFR7 NIOFR0 + IOFR3 IOFR0

In the case of a single-device device controller (IOFR0 = 0), bits LIO3 through LIO7 follow bits IOFR3 through IOFR7, allowing IOFR3 and IOFR4 to control group selection and bits IOFR5 through IOFR7 to control channel selection. In the case of a multidevice device controller (IOFR0 = 1), channel selection is controlled by bits IOFR1 through IOFR3 via LIO5 through LIO7, and group 1 is selected automatically by the logic NLIO3 NLIO4 (see figure 3-231 for the group selection chart).

3-134 Instructions, Commands, Orders

See Sigma 5 Computer Reference Manual, SDS Publication No. 900959.

3-135 Integral IOP/Device Controller Interface

See Interface Design Manual, SDS Publication No. 900973.

3-136 Service Cycles

See Interface Design Manual, SDS Publication No. 900973.

3-137 I/O Phase Sequencing

GENERAL. Input/output operations fall into two categories: instructions, which are CPU-initiated, and service cycles, which are initiated when a device controller generates a service call. Instructions are processed in a sequence of CPU phases and subphases and are described in tables 3-89 through 3-92. The following paragraphs describe the device controller-initiated I/O operations.

I/O PHASES AND SUBPHASES. Service calls are processed in a sequence of connect phases, IOSC and IOEN NIOIN,

preliminary phase IOEN IOIN NIOPH1, and main phases IOPH0 through IOPH3. Each main phase is divided into from one to eight subphases, SW8 through SW15. The four main phases are not necessarily sequential and are sustained until reset. The eight subphases are normally sequential; branching terms are used to alter the sequence. The logic for the main phases and subphases is shown below.

```
S/NIOPH0 =   RESET/A + (R/IOPH0)
 (R/IOPH0)=  IOR IOPH0 SW12
             + IOPH0 SW15
             + IOPH0 SW14 DATAOUT
               NVDATAOUT
R/NIOPH0 =   (S/IOPH0)
 (S/IOPH0)=  IOENIN + (R/IOPH2)
             + (IOPH3 SW15)

S/NIOPH1 =   RESET/A + (R/IOPH1)
 (R/IOPH1)=  IOPH1 SW8 DASW4 IODC
R/NIOPH1 =   (S/IOPH1)
 (S/IOPH1)=  IOPH0 SW15
             + IOB0 IOENNIN
             + IOPH0 SW14 DATAOUT
               NVDATAOUT

S/NIOPH2 =   RESET/A + (R/IOPH2)
 (R/IOPH2)=  BCZ IOPH2 (SW14 + SW15)
R/NIOPH2 =   (S/IOPH2)
 (S/IOPH2)=  IOPH0 SW12 IOR NDOR

S/NIOPH3 =   RESET/A + IOPH3 SW15
R/NIOPH3 =   (S/IOPH3)
 (S/IOPH3)=  IOPH0 SW12 DOR IOR
             + IOPH1 SW8 IODC DASW4

S/SW8    =   NRESET/A BRSW8
R/SW8    =   ...
S/SW9    =   SW8 STEP815
 STEP815 =   NBRSW13 NBRSW15 NRESET/A
             NBRSW8 NBRSW10 NBRSW11
             NBRSW12
R/SW9    =   ...

S/SW10   =   NRESET/A BRSW11 + SW9 STEP815
R/SW10   =   ...

S/SW11   =   NRESET/A BRSW11 + SW10 STEP815
R/SW11   =   ...

S/SW12   =   NRESET/A BRSW12 + SW11
             STEP815
R/SW12   =   ...

S/SW13   =   NRESET/A BRSW13 + SW12
             STEP815
R/SW13   =   ...

S/SW14   =   SW13 STEP815
R/SW14   =   ...

S/SW15   =   NRESET/A BRSW15 + SW14
             STEP815
R/SW15   =   ...
```

INDICATORS SW0 THROUGH SW7. During the I/O phase sequencing, flip-flops SW0 through SW7 indicate the following I/O-related conditions, as follows:

SW0	=	Zero byte count
SW1	=	Order (when true) or data (when false)
SW2	=	Out (when true) or in (when false)
SW3	=	Terminal order condition
SW4	=	Data chaining condition
SW5	=	Transfer in channel condition
SW6	=	End data; line /ED/ follows SW6
SW7	=	End service; line /ES/ follows SW7

PHASE SEQUENCE CHARTS. The I/O phases associated with service cycles are described in eight phase sequence charts and one summary chart, as follows:

Table 3-108 – Service Call Connect Phases. Initiated when the first service call is received and ended when flip-flop IOIN is set. If two successive service calls are processed, the connect phases of the second service call overlap with the restoration phases of the first service call, as shown in table 3-115. A typical timing sequence of the service call connect phases is illustrated in figure 3-233.

Table 3-109 – I/O Setup Phase Sequence. Deals with the saving of the interrupted instruction, storing the address of the device controller that generated the SC, and storing the service cycle type specified by the device controller. Events described in this table are common to all service cycles, and occur before the events described in table 3-110, table 3-112, table 3-113, or table 3-114, as applicable.

Table 3-110 – Order-Out Phase Sequence. Describes a sequence of events applicable only to the order-out service cycle. Other events occurring during the order-out service

cycle, but which are common to all service cycles, are described in tables 3-109 and 3-115.

Table 3-111 – Data Chaining Phase Sequence. The sequence of events described in this table is similar to table 3-110, and is entered, under certain conditions, from a data-out or data-in phase sequence.

Table 3-112 – Data-Out Phase Sequence. Describes a sequence of events that occurs only during the data-out service cycle.

Table 3-113 – Data-In Phase Sequence. Describes a sequence of events that occurs only during a data-in service cycle.

Table 3-114 – Order-In Phase Sequence. Describes a sequence of events that occurs only during an order-in service cycle.

Table 3-115 – I/O Restoration Phase Sequence. Deals with the restoring of the interrupted instruction, and is common to all service cycles. Note that certain phases of this table, such as IOPH1 SW9, also appear in table 3-110. This is the same phase, and was placed in table 3-115 to avoid repeating events common to all service cycles.

Table 3-116 – I/O Abort Phase Sequence. Deals with conditions resulting from an aborted service call.

Table 3-117 – Summary of I/O Phase Sequences. Combines the phases listed in tables 3-109 through 3-115 in a chronological order, and describes the main events occurring in each phase. Where no entry appears in the general activities column, that phase is entered only during those service cycles that have an entry in the corresponding special activities column. For example, phases IOPH2 are entered only during a data-out service cycle. On the other hand, if an entry appears in the general activities column, that phase is entered in the course of every service cycle, regardless of whether or not there is an entry in the corresponding special activities column.

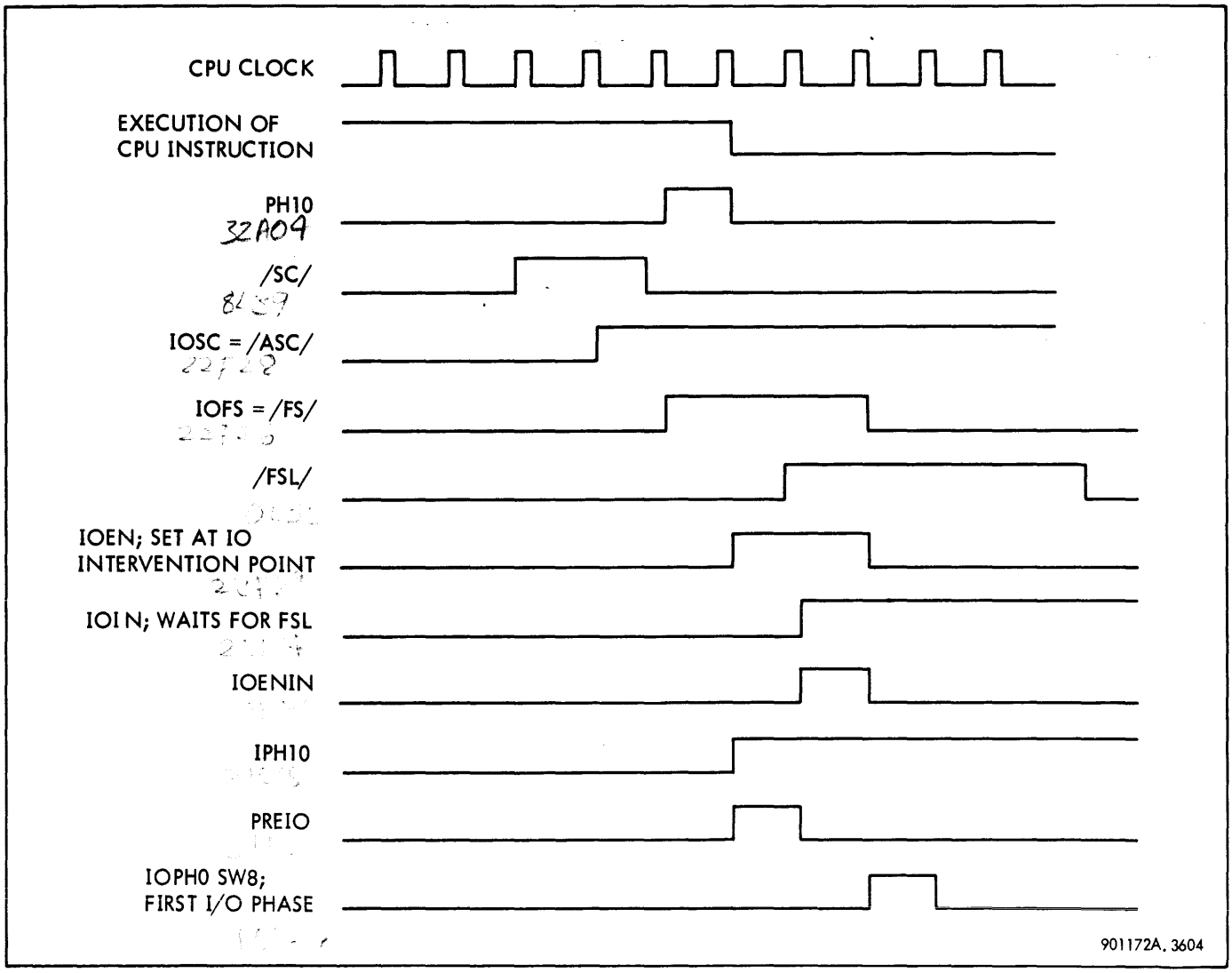


Figure 3-233. Service Call Connect Phases, Typical Timing Sequence

Table 3-108. Service Call Connect Phase Sequence

Phase	Function Performed	Signals Involved	Comments
ENTRY	Device controller may raise service call during any CPU phase	SC = /SC/	
	Integral IOP will defer SC acknowledgement if SC inhibit is present	SCINH = PCP2 RQBZO	Override interrupt busy
		+ PCP2/1 IOACT	I/O active during PCP idle phase
		+ PCPACT	PCP active
		+ IOWD	I/O watchdog timer runout
		+ FAIO NPH9 NPCP2	Execution phase of I/O instruction
		+ RESET/KS	KS reset
		+ N(IOPH1 SW9) IOACT	I/O is processing previous service call, but IOPH1 SW9 has not yet occurred
		+ PH10	
	If not SCINH, set flip-flop IOSC	S/IOSC = (S/IOSC)	Prepare to acknowledge service call
		(S/IOSC) = SC NSCINH IOPOP	
		IOPOP = Integral IOP option present	
		R/IOSC = PCP3 NIOFS	
		+ IOPH1 SW13 IOBO	
		+ RESETIO	
		+ IOPH0 SW8	
IOSC	Enable signal /ASC/	/ASC/ = IOSC	Acknowledge service call
	Set flip-flop IOFS	S/IOFS = (S/IOFS)	
		(S/IOFS) = IOSC NPCP3 + ...	
		R/IOFS = (R/IOFS)	
		+ IOPH1 SW13 IOBO	
		+ RESETIO	
	Enable signal IOCON	IOCON = IOSC + IOIN	IOP connect
	Inhibit RESETIO	RESETIO = RESIO + ...	
		RESIO = (NRCPUIOCON ...)	
		NRCPUIOCON = NIOCON + ...	
	If PCP2:		
	Inhibit (S/CXS)	(S/CXS) = (PCP2/1 NIOCON ...) + ...	

(Continued)

Table 3-108. Service Call Connect Phase Sequence (Cont.)

Phase	Function Performed	Signals Involved			Comments
IOSC (Cont.)	Inhibit (S/PCP3)	(S/PCP3)	=	(PCP2/1 NIOCON ...) + ...	
	Inhibit 0⟶PSW1	PSW1XS	=	(KCLRPSW1 NIOCON ...) + ...	
	Inhibit 0⟶PSW2	PSW2XS	=	(PSW2XS) + ...	
		(PSW2XS)	=	(KCLRPSW2 NIOCON ...) + ...	
	Inhibit (S/SXD)	(S/SXD)	=	(PCP2/1 NIOCON ...) + ...	
	If SC was obtained in PH10:				
	Inhibit P + 1⟶P	PUC31	=	(PH10 NIOSC ...) + ...	
	Set flip-flop NPRE1	S/NPRE1	=	N(S/PRE1)	
		(S/PRE1)	=	PRE1EN PH10 + ...	
		PRE1EN	=	(NIOSC ...)	
	Inhibit interrupt enable	IEN	=	(PH10 NIOSC ...)	
	Inhibit branch to PCP1	BRPCP1	=	(PH10 NIOSC ...)	
	Set flip-flop IPH10	S/IPH10	=	(S/IPH10)	Indicates CPU phase left at time SC was received. Used during IOPH1 SW13 as a reentry term
		(S/IPH10)	=	PH10 IOSC NIOINH	
		IOINH	=	INTRAP NPCP2	
				+ ADNH PH6	
				+ ABO PH6	
				+ (S/TRAP)	
		R/IPH10	=	(R/IPH10)	
	Set flip-flop IOEN at one of the four points at which I/O intervention is possible, whichever occurs first	S/IOEN	=	(S/IOEN)	Setting of flip-flop IOEN inhibits further CPU activities until I/O phase sequencing is completed
		(S/IOEN)	=	IOFS PCP2/1	PCP idle phase
				+ IOSC PH10 NIOINH	
				+ IOSC IOEN6 NIOINH	
				+ IOSC NIOINH IOPH1 SW11	This intervention point occurs during an I/O phase sequence when a previous SC is being processed. See IOPH1 SW11 of table 3-113
		IOEN6	=	(FAMDS + FAST/A	IOEN6 indicates that execution phase PH6 is in progress and that the instruction being executed qualifies for I/O intervention
				+ FAST/B + FAFL + FARWD)	
				NDIOEXIT (NFPRR NFSHEX	
				IOEN6/1 PH6)	
		IOEN6/1	=	N(MC005Z + FADIV CC2	
				+ EWDM)	

(Continued)

Table 3-108. Service Call Connect Phase Sequence (Cont.)

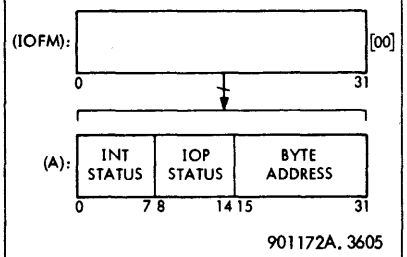
Phase	Function Performed	Signals Involved	Comments
IOEN NIOIN	Set flip-flop IOIN at the next clock after the device controller returned FSL	S/IOIN = (S/IOIN) (S/IOIN) = IOEN FSL NPH6 NIOINH R/IOIN = RESET/A	Setting of flip-flop IOIN indicates that FSL has been returned and SC is about to be processed
	If IFAST/L or IFAMDS, enable signal (S/SXA)	(S/SXA) = FAMDST PREIO + ... PREIO = IOEN NIOIN NPCP2 NIOINH FAMDST = (IFAST/L + IFAMDS) NPH6 IFAST/L = FAST/L NIPH10 NPCP2 IFAMDS = FAMD NIPH10 NPCP2	Prepare adder for A→/→S in IOEN NIOIN NIOPH1
	If device controller did not return FSL, reset flip-flop NIOB0 at the next clock following AVO	R/NIOB0 = (S/IOB0) (S/IOB0) = IOENNIN NIOINH AVO IOENNIN = IOEN NIOIN AVO S/NIOB0 = IOPH1 SW13 + RESET/A	Indicates I/O abort condition, as specified by AVO supplied by the device controller system; exit to phase IOEN NIOIN NIOPH1 IOB0 in table 3-116
	If IFAST/L, enable signal (S/CXS)	(S/CXS) = PREIO IFAST/L + ...	Prepare to transfer S→C in IOEN IOIN NIOPH1
	If IFAMDS, set flip-flop RW	S/RW = (S/RW/1) (S/RW/1) = (S/RW) + ... (S/RW) = IFAMDS PREIO	Prepare to transfer S→RW in IOEN IOIN NIOPH1

Table 3-109. I/O Setup Phase Sequence

Phase	Function Performed	Signals Involved	Comments
IOEN IOIN NIOPH1 T5L or T8L	One clock long. T8L if IFAMDS, T5L if IFAST/L		
	Reset flip-flops IODA0-IODA7	R/IODA0-IODA7 = IODAX IODAX = (R/IODA) (R/IODA) = IOENIN + ...	Clear IODA register
	(A0-A31)→(S0-S31)	SXA = Set at IOEN NIOIN clock	Transfer contents of A-register to sum bus
	If IFAMDS: (S0-S31)→(RW0-RW31) (R)	RWXS/0-RWXS/3 = RW + ... RW = Set at IOEN NIOIN clock	Transfer contents of sum bus to private memory register R

(Continued)

Table 3-109. I/O Setup Phase Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
IOEN IOIN NIOPH1 T5L or T8L (Cont.)	If IFAST/L:		
	(S0-S31)⟶(C0-C31)	CXS = Set at IOEN NIOIN clock	Transfer contents of sum bus to the C-register
	Reset flip-flop IOEN	R/IOEN = (R/IOEN) + ... (R/IOEN) = IOIN NIOPH1 + ...	
	Reset flip-flop IOFS	R/IOFS = (R/IOFS) + ... (R/IOFS) = IOIN NIOPH1 + ...	Drop function strobe
	(FR0-FR7)⟶(IOFR0-IOFR7)	IOFRXFR = IOENIN + ... IOENIN = IOEN IOIN NIOPH1 IOFRX = IOENIN + ...	Transfer device controller/device address to the IOFR register
	Enable signal (S/AXRR/2)	(S/AXRR/2) = IOENIN + ...	Prepare to transfer interrupt status, IOP status, and byte address from IOFM, area 00, to the A-register during IOPH0 SW8
	Reset flip-flop NIOFM	R/NIOFM = ... S/NIOFM = N(S/IOFM) (S/IOFM) = (S/AXRR/2) + ...	
	Maintain flip-flops IOFR8 and IOFR9 in a reset state	R/IOFR8 = ... R/IOFR9 = ...	
	Reset flip-flop NIOPH0	R/NIOPH0 = (S/IOPH0) (S/IOPH0) = IOENIN + ...	Prepare to exit to IOPH0 SW8
	Enable signal BRSW8	BRSW8 = IOENIN + ...	
IOPH0 SW8 T5L	One clock long		
	Reset flip-flop IOSC	R/IOSC = IOPH0 SW8 S/IOSC = SC (...)	Drop acknowledge service call signal ASC
	(RR0-RR31)⟶(A0-A31)	AXRR = Set at NIOPH1 IOEN IOIN clock	Transfer byte address, IOP status, and interrupt status from IOFM, area 00, to the A-register
	 901172A.3605	IOFM = Set at NIOPH1 IOEN IOIN clock NIOFR8 = Reset at NIOPH1 IOEN IOIN clock NIOFR9 = Reset at NIOPH1 IOEN IOIN clock	
	Disable signal PEM	PEM = PEM N(R/PEM) (R/PEM) = IOPH0 SW8 + ...	Erase previous parity error in memory condition

(Continued)

Table 3-109. I/O Setup Phase Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
IOPH0 SW8 T5L (Cont.)	Set flip-flop RSCLEN	S/RSCLEN = (S/RSCLEN) NCLEAR (S/RSCLEN) = IOPH0 SW8 + ... R/RSCLEN = ...	Disable clock at the end of this phase until the device controller returns RS
	If IFAST/L or IFAST/S and NIOPH10:		
	(B15-B31)⟶(S15-S31)	SXB-0 through SXB-3 = SXB SXB = SXBF NDIS + ... R/NSXBF = ... S/NSXBF = N(S/SXB) (S/SXB) = IOPH0 SW8 NIOPH10	Exchange contents of B-register with contents of P-register
	(S15-S31)⟶(P15-P31)	PXS = IOPH0 SW8 IFAST + ... IFAST = IFAST/L + IFAST/S	
	(P15-P31)⟶(B15-B31)	BXP = BXP/1 + ... BXP/1 = IOPH0 SW8 IFAST + ...	
	Enable signal (S/RW/2)	(S/RW/2) = IOPH0 SW8 NIOPH10	Prepare to transfer contents of B-register to IOFM, area 00, in IOPH0 SW9
	Reset flip-flop NIOFM	R/NIOFM = ... S/NIOFM = N(S/IOFM) (S/IOFM) = (S/RW/2)	
	Maintain flip-flops IOFR8 and IOFR9 in a reset state	R/IOFR8 = R/IOFR9 = ...	
	Reset flip-flop NSXBF	R/NSXBF = ... S/NSXBF = N(S/SXB) (S/SXB) = IOPH0 SW8 NIOPH10	Prepare adder for B⟶S in IOPH0 SW9
	Set flip-flop BRP	S/BRP = (S/BRP) + ... (S/BRP) = IOPH0 SW8 + ... R/BRP = (R/BRP)	Indicates that program address is now in P-register

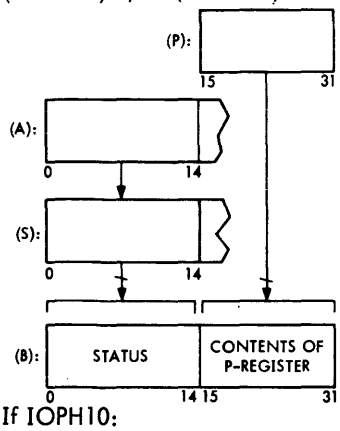
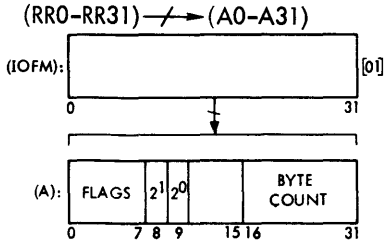
(Continued)

Table 3-109. I/O Setup Phase Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
IOPH0 SW8 T5L (Cont.)	If IOPH10		
	Enable signal (S/SXA)	(S/SXA) = IOPH0 SW8 IOPH10 + ... IOPH10 = NPCP2 IPH10 + ... IPH10 = Set at phase IOSC clock	Prepare to transfer contents of IOFM, area 01, to A-register
	Reset flip-flop NAXRR	R/NAXRR = N(S/AXRR) (S/AXRR) = (S/AXRR/3) + ... (S/AXRR/3) = IOPH0 SW8 IOPH10 + ...	
	Reset flip-flop NIOFM	R/NIOFM = ... S/NIOFM = N(S/IOFM) (S/IOFM) = (S/AXRR/3) + ...	
	Set flip-flop IOFR9	S/IOFR9 = (S/IOFR9) IOPOP IOPOP = Integral IOP option present (S/IOFR9) = (S/AXRR/3) + ... R/IOFR9 = ...	
	Maintain flip-flop IOFR8 in a reset state	R/IOFR8 = ...	
	Enable signal BRSW10	BRSW10 = IOPH0 SW8 IOPH10 + ...	Branch to IOPH0 SW10
IOPH0 SW9 T8L	Entered if NIOPH10 T8L minimum. Duration of clock controlled by device controller via RS		
	(B0-B31)⟶(S0-S31)	SXB = Set at IOPH0 SW8 clock	Transfer contents of B-register to sum bus
	(S0-S31)⟶(RW0-RW31)	RW = Set at IOPH0 SW8 clock IOFM = Set at IOPH0 SW8 clock NIOFR8 = Reset at IOPH0 SW8 clock NIOFR9 = Reset at IOPH0 SW8 clock	Transfer contents of sum bus to IOFM, area 00
	Enable signal (S/SXA)	(S/SXA) = IOPH0 SW9 + ...	Preset adder for A⟶S in IOPH0 SW10

(Continued)

Table 3-109. I/O Setup Phase Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
IOPH0 SW9 T8L (Cont.)	Reset flip-flop NIOFM	R/NIOFM = ... S/NIOFM = N(S/IOFM) (S/IOFM) = (S/AXRR/3) + ...	Prepare to transfer contents of IOFM, area 01, to the A-register
	Enable signal (S/AXRR/3)	(S/AXRR/3) = IOPH0 SW9 + ...	
	Set flip-flop IOFR9	S/IOFR9 = (S/IOFR9) IOPOP IOPOP = Integral IOP option present (S/IOFR9) = (S/AXRR/3) + ...	
		R/IOFR9 = ...	
	Reset flip-flop IOFR8	R/IOFR8 = ...	
	When RS, enable signal CLEN	CLEN = RSCLEN NRSA RS + ...	Enable clock when RS is obtained from device controller
IOPH0 SW10 T5L	One clock long. This phase entered either from IOPH0 SW8 or IOPH0 SW9. If entered from IOPH0 SW8, duration of clock controlled by device controller via RS		
	(A0-A31)⟶(S0-S31)	Adder logic set at IOPH0 SW8 clock or IOPH0 SW9 clock, as applicable	Transfer status from A-register through sum bus to the most significant half of B-register. Transfer contents of P-register to the least significant half of B-register
	(S0-S14)⟶(B0-B14)	BXS/0 = BXS/1 = BXP/2 + ...	
		BXP/2 = IOPH0 SW10 + ...	
	(P15-P31)⟶(B15-B31)	BXP = BXP/2	
			
	If IOPH10:		
	(RR0-RR31)⟶(A0-A31)	AXRR = Set at IOPH0 SW8 clock or IOPH0 SW9 clock	Transfer byte count and flags from IOFM, area 01, to the A-register
		IOFM = Set at IOPH0 SW8 clock or IOPH0 SW9 clock IOFR9 = Set at IOPH0 SW8 clock or IOPH0 SW9 clock NIOFR8 = Reset at IOPH0 SW8 clock	

901172A. 3606

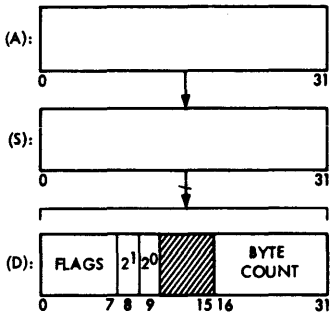
(Continued)

Table 3-109. I/O Setup Phase Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
IOPH0 SW10 T5L (Cont.)	Enable signal (S/SXA) Enable signal BRSW12	(S/SXA) = IOPH0 SW10 IOPH10 + ... BRSW12 = IOPH0 SW10 IOPH10	Preset adder for A⟶S in IOPH0 SW12, and branch to SW12
	If entered from IOPH0 SW8, enable signal CLEN when RS	CLEN = RSCLEN NRSA RS + ...	Enable clock when RS is obtained from device controller
	If NIOPH10		
	Enable signal (S/SXC)	(S/SXC) = IOPH0 SW10 NIOPH10 + ...	Prepare to transfer contents of C-register via sum bus to IOFM, area 01, in IOPH0 SW11
	Reset flip-flop NIOFM	R/NIOFM = ... S/NIOFM = N(S/IOFM) (S/IOFM) = (S/RW/3) + ... (S/RW/3) = IOPH0 SW10 NIOPH10 + ...	
	Set flip-flop IOFR9	S/IOFR9 = (S/IOFR9) IOPOP (S/IOFR9) = (S/RW/3) + ... R/IOFR9 = ...	
	Maintain flip-flop IOFR8 in a reset state	R/IOFR8 = ... S/IOFR8 = (S/IOFR8)	
IOPH0 SW11 T8L	Entered if NIOPH10 One clock long		
	(C0-C31)⟶(S0-S31)	SXC = SXCF NDIS + ... R/NSXCF = ... S/NSXCF = N(S/SXCF) (S/SXCF) = Came true at IOPH0 SW10 clock	Transfer contents of C-register via sum bus to IOFM, area 01
	(S0-S31)⟶(RW0-RW31) (01)	RWXS/0-RWXS/3 = RW + ... S/RW = (S/RW/1) + ... (S/RW/1) = (S/RW) + ... (S/RW) = (S/RW/3) + ... (S/RW/3) = Came true during IOPH0 SW10 clock	
	Enable signal (S/SXA)	(S/SXA) = IOPH0 SW11 + ...	Preset adder for A⟶S in IOPH0 SW12
IOPH0 SW12 T5L	One clock long (A0-A31)⟶(S0-S31)	SXA = Set at IOPH0 SW10 clock or IOPH0 SW11 clock, as applicable	Transfer contents of A-register via sum bus to D-register
	(S0-S31)⟶(D0-D31)	DXS = IOPH0 SW12 + ...	

(Continued)

Table 3-109. I/O Setup Phase Sequence (Cont.)

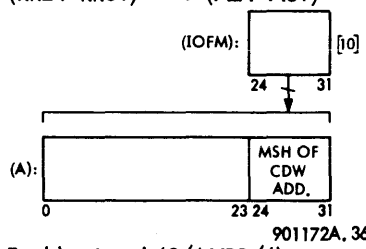
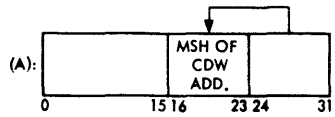
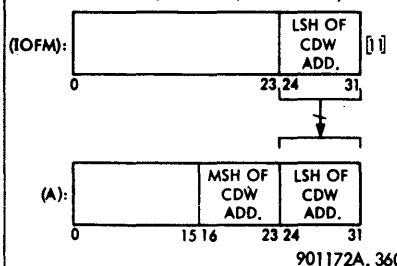
Phase	Function Performed	Signals Involved	Comments
IOPH0 SW12 T5L (Cont.)	 901172A.3607		
	If A8, set flip-flop P32	S/P32 = A8 IOPH0 SW12 + ...	Flip-flops P32 and P33 are the byte level indicators
		R/P32 = PX-3	
	If A9, set flip-flop P33	S/P33 = A9 IOPH0 SW12 + ...	
		R/P33 = PX-3	
	If DOR, set flip-flop SW1	S/SW1 = (S/SW1)	By means of DOR and IOR the device controller specifies one of four types of service it requires. This condition is stored in flip-flops SW1 and SW2 until IOPH1 and SW13
		(S/SW1) = DOR IOPH0 SW12 + ...	
		R/SW1 = RESET/A	
	If IOR, set flip-flop SW2	S/SW2 = (S/SW2)	
		(S/SW2) = IOR IOPH0 SW12 + ...	
		R/SW2 = (R/SW2)	
		DOR IOR 0 0 = data in 0 1 = data out 1 0 = order in 1 1 = order out	
	If order out		
	Enable signal (S/AXRR/4)	(S/AXRR/4) = IOPH0 SW12 DOR IOR + ...	If order out, prepare to transfer most significant half of CDW address from IOFM, area 10, to the A-register
	Reset flip-flop NIOFM	R/NIOFM = ...	
		S/NIOFM = N(S/IOFM)	
		(S/IOFM) = (S/AXRR/4) + ...	
	Set flip-flop IOFR8	S/IOFR8 = (S/IOFR8)	
		(S/IOFR8) = (S/AXRR/4) + ...	
		R/IOFR8 = ...	
	Reset flip-flop IOFR9	R/IOFR9 = ...	
	Set flip-flop NIOPH0	S/NIOPH0 = (R/IOPH0)	Advance to IOPH3 SW8, as required to process the order-out service cycle
		(R/IOPH0) = IOR IOPH0 SW12 + ...	
		R/NIOPH0 = (R/IOPH0) + RESET/A	

(Continued)

Table 3-109. I/O Setup Phase Sequence (Cont.)

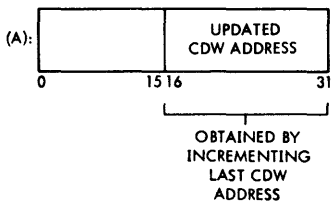
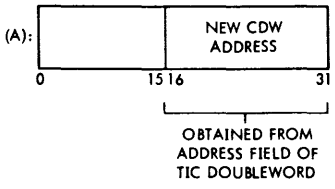
Phase	Function Performed	Signals Involved	Comments
IOPH0 SW12 T5L (Cont.)	Reset flip-flop NIOPH3	R/NIOPH3 = (S/IOPH3) (S/IOPH3) = IOPH0 SW12 DOR IOR S/NIOPH3 = IOPH3 SW15 + RESET/A	
	Enable signal BRSW8	BRSW8 = IOPH0 SW12 DOR IOR + ...	
	If data out		
	Set flip-flops MRQ and DRQ	S/MRQ = (S/MRQ) (S/MRQ) = (S/MRQ/2) + ... (S/MRQ/2) = IOPH0 SW12 NDOR IOR R/MRQ = ... S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = (S/MRQ/2) + ... R/DRQ = ...	Prepare to read first word of data from core memory Inhibits transmission of another clock until data release is received from core memory
	Set flip-flop NIOPH0 Reset flip-flop NIOPH2	S/NIOPH0 = (R/IOPH0) (R/IOPH0) = IOR IOPH0 SW12 + ... R/NIOPH2 = (S/IOPH2) (S/IOPH2) = IOPH0 SW12 NDOR IOR	Advance to IOPH2 SW13 as required to process the data-out service cycle
	If data in		
	Enable signal (S/SXDM1)	(S/SXDM1) = (IOPH0 SW12 NDOR NIOR) + ...	Prepare adder for D -1 → S in IOPH0 SW13
	Exit		
	If order out, to IOPH3 SW8 (see table 3-110)		
	If order in, to IOPH0 SW13 (see table 3-114)		
	If data out, to IOPH2 SW13 (see table 3-112)		
	If data in, to IOPH0 SW13 (see table 3-113)		

Table 3-110. Order-Out Phase Sequence

Phase	Function Performed	Signals Involved	Comments
IOPH3 SW8 T5L	One clock long (RR24-RR31)⟶(A24-A31)  901172A.3608	AXRR = Set at IOPH0 SW12 clock IOFR8 = Set at IOPH0 SW12 clock NIOFR9 = Set at IOPH0 SW12 clock IOFM = Set at IOPH0 SW12 clock	Transfer most significant half of command doubleword address from IOFM, area 10, to A-register
	Enable signal (S/AXRR/4)	(S/AXRR/4) = IOPH3 SW8 + ...	Prepare to transfer least significant half of command doubleword address from IOFM, area 11, to A-register
	Enable signal (S/AXRR/6)	(S/AXRR/6) = IOPH3 SW8 + ...	
	Set flip-flop IOFR8	S/IOFR8 = (S/IOFR8) (S/IOFR8) = (S/AXRR/4) + ... R/IOFR8 = ...	
	Set flip-flop IOFR9	S/IOFR9 = (S/IOFR9) IOPOP (S/IOFR9) = (S/AXRR/6) + ... R/IOFR9 = ...	
	Reset flip-flop NIOFM	R/NIOFM = ... S/NIOFM = N(S/IOFM) (S/IOFM) = (S/AXRR/6) + ...	
	Set flip-flop SW3	S/SW3 = (S/SW3) + ... (S/SW3) = IOPH3 + ... R/SW3 = RESET/A	Used as a qualifying term during IOPH1 SW12
IOPH3 SW9 T5L	One clock long Enable signal AXAL8  	AXAL8 = IOPH3 SW9 + ...	Shift contents of A-register 8 places to the left
	(RR24-RR31)⟶(A24-A31)  901172A.3609	AXRR = Set at IOPH3 SW8 clock IOFM = Set at IOPH3 SW8 clock IOFR8 = Set at IOPH3 SW8 clock IOFR9 = Set at IOPH3 SW8 clock	Transfer least significant half of command doubleword address from IOFM, area 11, into A-register
	Enable signal (S/SXAP1)	(S/SXAP1) = IOPH3 SW9 + ...	Preset adder logic for A + 1⟶A in IOPH3 SW10

(Continued)

Table 3-110. Order-Out Phase Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
IOPH3 SW9 T5L (Cont.)	Exit: to IOPH3 SW10 NB1 or IOPH3 SW10 B1, depending on the state of chaining modifier flip-flop B1		
IOPH3 SW10 NB1 T5L	One clock long Reason for entering this phase: a. From IOPH3 SW9, if order-out sequence was entered following an SIO instruction. b. From IOPH3 SW9, if order-out sequence was entered as a result of command chaining, and a chaining modifier was not present. c. From IOPH3 SW10 B1, if order-out sequence was entered as a result of command chaining, and a chaining modifier was present. d. From IOPH3 SW14, if bits 4 through 7 of the command doubleword specified transfer in channel. If not transfer in channel (A + 1)⟶(S)  If this phase was entered from IOPH3 SW14 (A0-A31)⟶(S0-S31)  2S⟶A 901172A.3610	SXAP1 = Set at IOPH3 SW9 clock or at IOPH3 SW10 B1 clock SXA = Set at IOPH3 SW14 clock AXSL1 = IOPH3 SW10 NB1 + ...	Increment CDW address by one. The term SXAP1 enables the A-register to function as a command address counter. To make the initial address come out correctly, it was decremented by one during PH5 SW13 of the SIO instruction Incrementation not required, since in this case the A-register contains the actual address of the new CDW A left shift is necessary to obtain the correct word address of the command doubleword

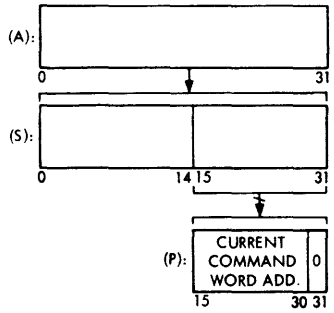
(Continued)

Table 3-110. Order-Out Phase Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
IOPH3 SW10 NB1 T5L (Cont.)	Enable signal (S/SXA) Exit: to IOPH3 SW11	(S/SXA) = IOPH3 SW10 NB1 + ...	Preset adder logic for A ⟶ S in IOPH3 SW11
IOPH3 SW10 B1 T5L	One clock long Reason for entering this phase: From IOPH3 SW9, when order-out sequence was entered as a result of command chaining, and chaining modifier was present		
	(A + 1) ⟶ S	SXAP1 = Set at IOPH3 SW9 clock	Increment command doubleword address by one
	(S0-S31) ⟶/⟶ (A0-A31)	AXS = IOPH3 SW10 B1 + ...	
	Reset flip-flop B1	R/B1 = (R/B1) (R/B1) = (R/B1/1) + ... (R/B1/1) = IOPH3 SW10 + ...	Erase chaining modifier condition
		S/B1 = (S/B1) IOPOP + ... (S/B1) = TORDIN A2	Chaining modifier B1 was originally set during the preceding order-in sequence if specified by the device controller. During IOPH0 SW13 of the order-in sequence A2 is controlled by bit DA2 via S2
	Enable signal (S/SXAP1)	(S/SXAP1) = IOPH3 SW10 B1 + ...	Preset adder logic for A + 1 ⟶ A in IOPH3 SW10 NB1
	Enable signal BRSW10	BRSW10 = IOPH3 SW10 B1 + ...	Command doubleword address is incremented twice, during this phase and during IOPH3 SW10 NB1, before core memory is accessed. This way, the next command in sequence is skipped
	Exit: to IOPH3 SW10 NB1		
IOPH3 SW11 T5L	One clock long (A0-A31) ⟶ (S0-S31)	SXA = Set at IOPH3 SW10 NB1 clock	Transfer contents of A-register via the sum bus to the P-register
	(S15-S31) ⟶/⟶ (P15-P31)	PXS = IOPH3 SW11 + ...	

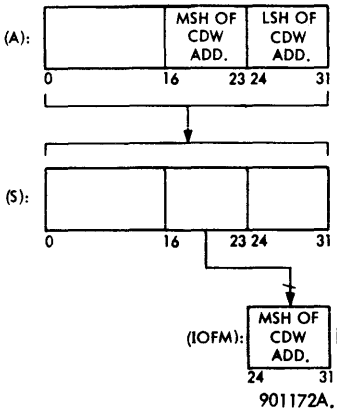
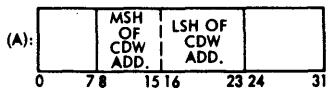
(Continued)

Table 3-110. Order-Out Phase Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
IOPH3 SW11 T5L (Cont.)	 901172A.3611		
	Set flip-flop MRQ and reset flip-flop NMRQP1	S/MRQ = (S/MRQ) (S/MRQ) = (S/MRQ/3) + ... (S/MRQ/3) = IOPH3 SW11 + ... R/MRQ = ... S/NMRQP1 = N(S/MRQ/3) + ... R/NMRQP1 = ...	Prepare to read even word of CDW from core memory Delays setting of flip-flop DRQ by one clock
	If NSW5, enable signal (R/PEM)	(R/PEM) = IOPH3 SW11 NSW5	If not transfer in channel, reset parity error in memory condition
	Enable signal AXSR1	AXSR1 = IOPH3 SW11 + ...	Change word address stored in A-register to doubleword address by means of a right shift
	Enable signal (S/SXA)	(S/SXA) = IOPH3 SW11 + ...	Preset adder for A⟶S in IOPH3 SW12
	Set flip-flop IOFR8 and maintain flip-flop IOFR9 in a reset state	S/IOFR8 = (S/IOFR8) (S/IOFR8) = (S/RW/4) + ... (S/RW/4) = IOPH3 SW11 + ... R/IOFR8 = ... R/IOFR9 = ... R/NIOFM = ... S/NIOFM = N(S/IOFM) (S/IOFM) = (S/RW/4) + ...	Prepare to transfer byte 2 of A-register via the sum bus to IOFM, area 10
	Set flip-flop RW	S/RW = (S/RW/4) + ... R/RW = ...	
IOPH3 SW12 T8L	One clock long (A0-A31)⟶(S0-S31) (S16-S23)⟶/⟶(RW16-RW23)	SXA = Set at IOPH3 SW11 clock RWXS/0-RWXS/3 = RW + ... RW = Set at IOPH3 SW11 clock	Load most significant half of current command doubleword address to IOFM, area 10

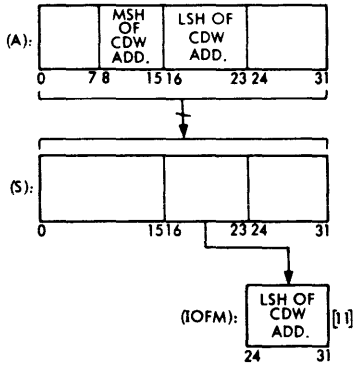
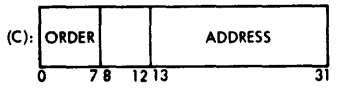
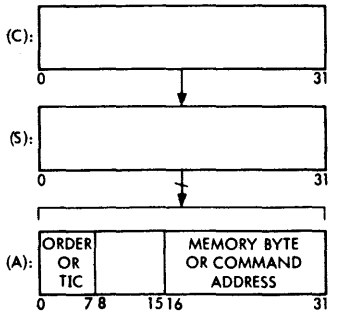
(Continued)

Table 3-110. Order-Out Phase Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
IOPH3 SW12 T8L (Cont.)	 901172A.3612	IOFR8 = Set at IOPH3 SW11 clock NIOFR9 = Reset at IOPH3 SW11 clock IOFM = Set at IOPH3 SW11 clock	
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = MRQP1 + ... R/DRQ = ...	Inhibits transmission of another clock until data release signal is received from core memory
	Set flip-flop RW	S/RW = (S/RW/1) (S/RW/1) = (S/RW) + ... (S/RW) = (S/RW/3) + (S/RW/4) + ... (S/RW/3) = IOPH3 SW12 + ... (S/RW/4) = IOPH3 SW12 + ... R/RW = ...	Prepare to transfer LSH of command doubleword address from A-register into IOFM, area 11
	Set flip-flops IOFR8 and IOFR9	S/IOFR8 = (S/IOFR8) (S/IOFR8) = (S/RW/4) + ... R/IOFR8 = ... S/IOFR9 = (S/IOFR9) IOPOP (S/IOFR9) = (S/RW/3) + ... R/IOFR9 = ...	
	Reset flip-flop NIOFM	R/NIOFM = ... S/NIOFM = N(S/IOFM) (S/IOFM) = (S/RW/3) + (S/RW/4) + ...	
	Enable signal AXAL8  901172A.3613	AXAL8 = IOPH3 SW12 + ...	Align LSH of command doubleword address by shifting the contents of the A-register 8 places to the left
	Enable signal (S/SXA)	(S/SXA) = IOPH3 SW12 + ...	Preset adder logic for A ⟶ S in IOPH3 SW13

(Continued)

Table 3-110. Order-Out Phase Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
IOPH3 SW13 DR	One clock long (A0-A31)⟶(S0-S31) (S16-S23)⟶(RW16-RW23) 	SXA = Set at IOPH3 SW12 clock RW = Set at IOPH3 SW11 clock IOFR8 = Set at IOPH3 SW12 clock IOFR9 = Set at IOPH3 SW12 clock IOFM = Set at IOPH3 SW12 clock	Load LSH of command doubleword address from A-register to IOFM, area 11
	(MB0-MB31)⟶(C0-C31) 	CXMB = DG = /DG/	Load even word of command doubleword from core memory into the C-register
	Enable signal (S/SXC)	(S/SXC) = IOPH3 SW13 + ...	Preset adder for C⟶S in IOPH3 SW14
	Enable signal PUC31  901172A.3614	PUC31 = IOPH3 SW13 + ...	Increment P-register by one to obtain the odd word of the CDW during IOPH3 SW14
IOPH3 SW14 T5L	One clock long (C0-C31)⟶(S0-S31) (S0-S31)⟶(A0-A31)  901172A.3615	SXC = Set at IOPH3 SW13 clock AXS = IOPH3 SW14 + ...	Load even command word into the A-register. If order, bits 13 through 31 contain the memory byte address. If TIC, bits 16 through 31 contain new command address
	Enable signal (S/SXA)	(S/SXA) = IOPH3 SW14 + ...	Prepare adder logic for A⟶S in IOPH3 SW15 or IOPH3 SW10, as applicable

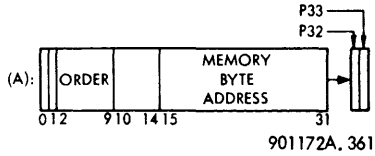
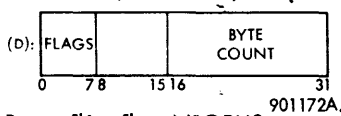
(Continued)

Table 3-110. Order-Out Phase Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
IOPH3 SW14 T5L (Cont.)	If IOTRIN (transfer in channel)		
	Set flip-flop SW5	S/SW5 = (S/SW5) + ... (S/SW5) = IOPH3 SW14 VORDER IOTRIN IOTRIN = C4 NC5 NC6 NC7 VORDER = (ORDEROUT + SW4) NSW5 NPEM NADNH SW4 = Data chaining	Store transfer in channel condition in flip-flop SW5
	Enable signal BRSW10	BRSW10 = IOPH3 SW14 VORDER IOTRIN + ...	Return to IOPH3 SW10 NB1 and obtain a new command from core memory, as specified by the address field of the A-register
	If NIOTRIN (not transfer in channel)		
	Set flip-flop MRQ	S/MRQ = (S/MRQ) (S/MRQ) = (S/MRQ/2) + ... (S/MRQ/2) = IOPH3 SW14 NIOTRIN R/MRQ = ...	Prepare to read odd command word from core memory
	Set flip-flop DRQ	S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = (S/MRQ/2) + ... R/DRQ = ...	Inhibits transmission of another clock until data release is received from core memory
	Reset flip-flop SW5	R/SW5 = (R/SW5) (R/SW5) = (R/SW5/1) + ... (R/SW5/1) = IOPH3 SW14 NIOTRIN	
IOPH3 SW15 DR	One clock long (A0-A7)—/→(IODA0-IODA7)	IODAXA = IOPH3 SW15 NSW4 + ...	Load order into the IODA register. From here the order is transmitted automatically, via data lines /DA0/-/DA7/, to the device controller
	(IODA0-IODA7)——→ (/DA0/-/DA7/)	/DA0/-/DA7/ = IODA0-IODA7	
	(A0-A31)——→(S0-S31)	SXA = Set at IOPH3 SW14 clock	Shift the contents of the A-register two positions to the right
	Enable signal AXSR2	AXSR2 = IOPH3 SW15 + ...	
	If A30, set flip-flop P32	S/P32 = A30 AXSR2 + ... R/P32 = PX + ...	Transfer two least significant bits of the A-register to P32 and P33, respectively
	If A31, set flip-flop P33	S/P33 = A31 AXSR2 + ... R/P33 = PX + ...	

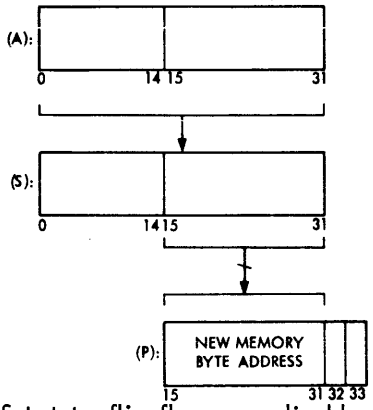
(Continued)

Table 3-110. Order-Out Phase Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
IOPH3 SW15 DR (Cont.)	 901172A.3616		
	(MB0-MB31) → C → (D0-D31)	CXMB = DG = /DG/ DXC = VORDER	Load odd word into the C-register and then to the D-register
	 901172A.3617		
	Reset flip-flop NIOPH0	R/NIOPH0 = (S/IOPH0) (S/IOPH0) = IOPH3 SW15 + ... S/IOPH0 = (R/IOPH0) + ...	Branch to IOPH0 SW15
	Set flip-flop NIOPH3	S/NIOPH3 = IOPH3 SW15 + ... R/NIOPH3 = (S/IOPH3)	
	Enable signal BRSW15	BRSW15 = IOPH3 SW15 + ...	
IOPH0 SW15 T5L	One clock long		
	Set flip-flop SW6	S/SW6 = (S/SW6) + ... (S/SW6) = (IOPH0 SW15) (SW1 + SW4) + ... R/SW6 = RESET/A	SW6 and NSW7 are used during IOPH1 SW8 to instruct the device controller to request a terminal order
	Reset flip-flop SW7	R/SW7 = (R/SW7) (R/SW7) = IOPH0 SW15 + ...	
	If VORDER, enable signal (S/SXA)	(S/SXA) = IOPH0 SW15 VORDER VORDER = (ORDEROUT + SW4) NSW5 NPEM NADNH	If VORDER, prepare adder for A → S in IOPH1 SW8
	Set flip-flop NIOPH0	S/NIOPH0 = (R/IOPH0) + ... (R/IOPH0) = IOPH0 SW15 + ... R/NIOPH0 = (S/IOPH0)	Advance to IOPH1 SW8
	Reset flip-flop NIOPH1	R/NIOPH1 = (S/IOPH1) (S/IOPH1) = IOPH0 SW15 + ...	
	Enable signal BRSW8	BRSW8 = IOPH0 SW15 + ...	
IOPH1 SW8 T8L	One clock long		
	Enable signal /ED/	/ED/ = SW6 = Set in IOPH0 SW15	Instruct device controller by means of /ED/ N/ES/ to request a terminal order. Meaningful only when RSA is raised
	Disable signal /ES/	N/ES/ = NSW7 = Reset in IOPH0 SW15	

(Continued)

Table 3-110. Order-Out Phase Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
IOPH1 SW8 T8L (Cont.)	Set flip-flop RSA	S/RSA = (S/RSA) (S/RSA) = IOPH1 SW8 ORDSW4 + ... ORDSW4 = SW1 + SW4 SW1 = Order out or order in SW4 = Data chaining E/RSA = NRS	Raise request strobe acknowledge signal to the device controller then drop it when device controller drops RS
	If VORDER (A0-A31)⟶(S0-S31) (S15-S31)⟶/(P15-P31)  Set status flip-flops as applicable	SXA = Set at IOPH0 SW15 clock PXS = IOPH1 SW8 VORDER + ... VORDER = NSW5 NPEM NADNH (ORDEROUT + SW4) NSW5 = Not transfer in channel NPEM = Not parity error in memory NADNH = Not address not here	Load new memory byte address into the P-register
	If read backward order, set flip-flop B0	S/B0 = (S/B0) IOPOP + ... (S/B0) = IOPH1 SW8 ORDEROUT IORB IORB = IODA4 IODA5 NIODA6 NIODA7 R/B0 = (R/B0)	
	If IOP control error, set flip-flop B13	S/B13 = (S/B13) IOPOP + ... (S/B13) = IOPH1 SW8 ORDEROUT SW5 R/B13 = BX/1	With no error, SW5 should be false. If SW5 is true, it indicates two consecutive transfers in channel
	If IOP memory error, set flip-flop B12	S/B12 = (S/B12) IOPOP + ... (S/B12) = IOPH1 SW8 ORDEROUT PEM R/B12 = BX/1	
	If IOP halt, set flip-flop B14	S/B14 = (S/B14) IOPOP + ... (S/B14) = (S/B12) + (S/B14/1) + ... (S/B14/1) = (S/B13) + (S/B11) + ... (S/B11) = ADNH IOIN (memory address error) R/B14 = BX/1	During order out an IOP halt can be caused either by a control error, memory parity error, or memory address error

901172A.3618

(Continued)

Table 3-110. Order-Out Phase Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
IOPH1 SW8 T8L (Cont.)	Set flip-flop RSCLEN	S/RSCLEN = (S/RSCLEN) NCLEAR (S/RSCLEN)= IOPH1 SW8 ORDSW4 + ... R/RSCLEN = ... CLEN = NRSCLEN + ...	Advance to IOPH1 SW9, then disable clock until device controller returns RS
IOPH1 SW9 T8L	One clock long. Duration of clock controlled by device controller via RS		
	Set flip-flops SW6 and SW7	S/SW6 = TODATA + ... TODATA = IOPH1 SW9 ORDSW4 + ... ORDSW4 = SW1 + SW4 R/SW6 = RESET/A S/SW7 = TODATA + ... R/SW7 = (R/SW7) + ...	Prepare to specify final byte exchange between the integral IOP and the device controller
	If D1 and SW0, set flip-flop IODA0	S/IODA0 = TODATA SW0 D1 + ... SW0 = Zero byte count D1 = Interrupt on zero byte count flag, stored in D-register R/IODA0 = IODAX	Assemble terminal order in IODA register. During an order-out sequence, IODA0 and IODA3 are the only two meaningful terminal order bits
	If B14, set flip-flop IODA3	S/IODA3 = B14 TODATA + ... B14 = IOP halt R/IODA3 = IODAX	
	If SW0 and ND0 (count done), set flip-flop IODA1	S/IODA1 = TODATA SW0 ND0 + ... ND0 = Not data chaining R/IODA1 = IODAX	
	If D2 (command chaining), set flip-flop IODA2	S/IODA2 = D2 TODATA + ... R/IODA2 = IODAX D2 = Command chaining	
	(IODA0-IODA7) ⟶ (/DA0/-/DA7/)	/DA0/-/DA7/ = IODA0-IODA7	From the IODA register the terminal order is transmitted automatically via data lines /DA0/-/DA7/ to the device controller
	When RS, enable signal CLEN	CLEN = RSCLEN NRSA RS + ...	Enable clock when RS is obtained from device controller

(Continued)

Table 3-110. Order-Out Phase Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
IOPH1 SW12 T5L	One clock long. Start of next clock controlled by device controller via NRS		
	Enable signal /ED/	/ED/ = SW6	Specify final byte exchange by means of /ED/ and /ES/
	Enable signal /ES/	/ES/ = SW7	
	Set flip-flop RSA	S/RSA = (S/RSA) (S/RSA) = IOPH1 SW12 SW3 SW3 = Set at IOPH3 SW8 clock E/RSA = NRS	Raise request strobe acknowledge signal to device controller, then drop it when device controller drops request strobe
	Set flip-flop RSACLEN	S/RSACLEN = (S/RSACLEN) NCLEAR (S/RSACLEN) = IOPH1 SW12 SW3 R/RSACLEN = ... CLEN = RSACLEN NRSA + ...	Delay start of next clock by setting flip-flop RSACLEN. Clock starts again when device controller drops RS, dc-resetting RSA. NRSA RSACLEN drive clock enable signal CLEN true. Falling edge of RS also disconnects device controller

Table 3-111. Data Chaining Phase Sequence

Phase	Function Performed	Signals Involved	Comments
IOPH3 SW8 T5L	This phase is identical to IOPH3 SW8 of the order-out phase sequence. See table 3-110 Conditions for entering this phase: a. From IOPH1 SW8 of the data-out phase sequence. See table 3-112 b. From IOPH1 SW8 of the data-in phase sequence. See table 3-113		
IOPH3 SW9 T5L	This phase is similar to IOPH3 SW9 of the order-out phase sequence, with the following additions: If parity error in memory, set flip-flop B10	S/B10 = (S/B10) IOPOP + ... (S/B10) = PEM NSW1 (IOPH3 SW9 SW4 + ...) PEM = Parity error in memory NSW1 = Data out or data in	Store transmission error condition

(Continued)

Table 3-111. Data Chaining Phase Sequence (Cont.)

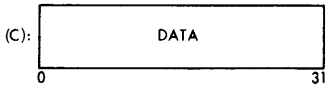
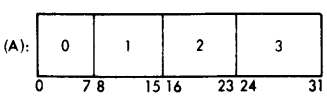
Phase	Function Performed	Signals Involved	Comments
IOPH3 SW9 T5L (Cont.)	If parity error in memory, and halt on transmission error flag is high, set flip-flop B14	SW4 = Data chaining R/B10 = BX/1 S/B14 = (S/B14) IOPOP + ... (S/B14) = (S/B10) D4 D4 = Halt on transmission error flag R/B14 = BX/1	Store IOP halt condition
IOPH3 SW10 NB0	This phase is similar to IOPH3 SW10 NB0 of the order-out phase sequence (see table 3-110), with the following exceptions: a. Reason for entering this phase: 1. From IOPH3 SW9 of the data chaining sequence 2. From IOPH3 SW14, if data bits 4 through 7 of the double-word specified transfer in channel b. SXAP1 = Set at IOPH3 SW9 clock		
IOPH3 SW11 T5L	This phase is identical to IOPH3 SW11 of the order-out phase sequence. See table 3-110		
IOPH3 SW12 T8L	This phase is identical to IOPH3 SW12 of the order-out phase sequence. See Table 3-110		
IOPH3 SW13 DR	This phase is identical to IOPH3 SW13 of the order-out phase sequence. See table 3-110		
IOPH3 SW14 T5L	This phase is identical to IOPH3 SW14 of the order-out phase sequence. See table 3-110		
IOPH3 SW15 DR	This phase is similar to IOPH3 SW15 (see table 3-110), with the following exception: Signal IODAXA does not come true (NSW4 is false) and data is not transferred to the IODA register		

(Continued)

Table 3-111. Data Chaining Phase Sequence (Cont.)

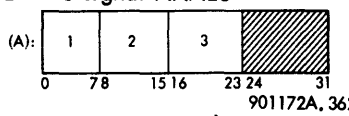
Phase	Function Performed	Signals Involved	Comments
IOPH0 SW15 T5L	This phase is identical to IOPH0 SW15 of the order-out phase sequence. See table 3-110		
IOPH1 SW8 T8L	This phase is identical to IOPH1 SW8 of the order-out phase sequence. See table 3-110		
IOPH1 SW9 T8L	This phase is similar to IOPH1 SW9 of the order-out phase sequence (see table 3-110), with the following exception: Terminal order bit IODA1 is also meaningful		
IOPH1 SW12 T5L	This phase is identical to IOPH1 SW12 of the order-out phase sequence. See table 3-110		

Table 3-112. Data-Out Phase Sequence

Phase	Function Performed	Signals Involved	Comments
IOPH2 SW13 DR	One clock long (MB0-MB31)⟶(C0-C31)  901172A.3619	CXMB = DG = /DG/	Load one word of data from core memory in the C-register
	P32⟶/⟶BC0	S/BC0 = IOPH2 SW13 P32 NPRE/34 + ... R/BC0 = (R/BC0)	Load byte level from P32 and P33 in byte level indicators BC0 and BC1, respectively
	P33⟶/⟶BC1	S/BC1 = IOPH2 SW13 P33 NPRE/34 + ... R/BC1 = (R/BC1)	
	Enable signal (S/SXC)	(S/SXC) = IOPH2 SW13 + ...	Prepare adder for C⟶S in IOPH2 SW14
IOPH2 SW14 T5L	One clock long (C0-C31)⟶(S0-S31) (S0-S31)⟶/⟶(A0-A31)  -1⟶/⟶(BC0-BC1) 901172A.3620	SXC = Set at IOPH2 SW13 AXS = IOPH2 SW14 + ...	Load data word from C-register via sum bus to A-register
		BCDC1 = IOPH2 SW14 + ...	Decrement byte level by one

(Continued)

Table 3-112. Data-Out Phase Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
IOPH2 SW14 T5L (Cont.)	If BCZ	BCZ = NBC0 NBC1	Test for BCZ; if true, exit to IOPH0 SW13. If BCZ occurs in this phase, byte 0 will be the first byte to be sent to the device controller in IOPH0 SW13. If NBCZ, advance to IOPH2 SW15
	Set flip-flop NIOPH2	S/NIOPH2 = (R/IOPH2) (R/IOPH2) = BCZ IOPH2 SW14 + ... R/NIOPH2 = (S/IOPH2)	
	Reset flip-flop NIOPH0	R/NIOPH0 = (S/IOPH0) (S/IOPH0) = (R/IOPH2) + ...	
	Enable signal BRSW13	BRSW13 = IOPH2 SW14 BCZ + ...	
	Enable signal (S/SXDM1)	(S/SXDM1) = IOPH2 SW14 BCZ + ...	Prepare adder for (D - 1) ⟶ D in IOPH0 SW13
IOPH2 SW15 T5L	One, two, or three clocks long, depending on the initial count stored in BC0-BC1, until BCZ is reached		
	Enable signal AXAL8  901172A.3621	AXAL8 = IOPH2 SW15 + ...	Shift contents of A-register 8 places to the left
	-1 ⟶ (BC0-BC1)	BCDC = IOPH2 SW15	Decrement byte level by one
	If BCZ	BCZ = NBC0 NBC1	Test for BCZ; if true, exit to IOPH0 SW13
	Set flip-flop NIOPH2	S/NIOPH2 = (R/IOPH2) (R/IOPH2) = BCZ IOPH2 SW15 + ...	
	Reset flip-flop NIOPH0	R/NIOPH0 = (S/IOPH0) (S/IOPH0) = (R/IOPH2) + ...	
	Enable signal BRSW13	BRSW13 = IOPH2 SW15 BCZ + ...	
	Enable signal (S/SXDM1)	(S/SXDM1) = IOPH2 SW15 BCZ + ...	Prepare adder for (D - 1) ⟶ D in IOPH0 SW13
	If not BCZ		
	Enable signal BRSW15	BRSW15 = IOPH2 SW15 NBCZ + ...	Sustain IOPH2 SW15 until BCZ is reached. During each iteration the most significant byte is shifted out of the A-register and BC0-BC1 is decremented by one. At BCZ, any bytes (or byte) still remaining in the A-register will eventually be transferred, one byte at a time, to the device controller

(Continued)

Table 3-112. Data-Out Phase Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
IOPH0 SW13 T8L	This phase is entered initially from IOPH2 SW14 or IOPH2 SW15. Subsequently, it is re-entered from IOPH0 SW14 each time the term VDATAOUT is true. The maximum number of iterations is four. When entered from IOPH0 SW14, duration of the clock is controlled by the device controller via RS		
	(D16-D31 minus 1)⟶ (S16-S31)	SXDM1 = Set at IOPH2 SW14, or IOPH2 SW15, or IOPH0 SW14 clock, as applicable	Decrement byte count by one
	(S16-S31)—/—►(D16-D31)	DXS/2 = DXS/3 = DXS/4 + DXS DXS/4 = IOPH0 SW13 NSW1 NSW1 = Data out or data in	
	Test for zero byte count	S1631Z = N(S16 + S17 + ... + S31)	
	If S1631Z, set flip-flop SW0	S/SW0 = S1631Z IOPH0 SW13 NSW1 + ... SW0 = Zero byte count	
	(A0-A7)—/—►(IODA0-IODA7)	IODAXA = DATAOUT IOPH0 SW13 ND7 ND7 = Skip flag not present /DA0/-/DA7/ = IODA0-IODA7	Transfer data byte to the device controller
	If sum of A0-A7 even, enable signal IOPG; otherwise, reset flip-flop IODAP IOPG—/—►IODAP⟶/DAP/	IOPG = True if sum of A0-A7 even /DAP/ = IODAP; S/IODAP = (S/IODAP) (S/IODAP) = IOPH0 SW13 DATAOUT IOPG + ... R/IODAP = IOPH0 SW13	Generate odd data parity by means of parity bit IOPG. Flip-flop IODAP is used in data-out to generate data parity bit; during data-in to store data parity fail condition
	Enable signal AXAL8 If this phase entered from IOPH0 SW14	AXAL8 = DATAOUT IOPH0 SW13 + ...	Align next data byte
	Increment P32-P33	PUC3033 = PUC33 + ... PUC33 = IOPH0 SW13 DATAOUT RSCLEN RSCLEN = Set at IOPH0 SW14 clock	Increment byte address
	When RS, enable signal CLEN	CLEN = RSCLEN NRSA RS NRS ⟹ E/RSA (not clocked)	Enable clock when RS is obtained from device controller

(Continued)

Table 3-112. Data-Out Phase Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
IOPH0 SW14 T8L	When VDATAOUT is true, this phase alternates with IOPH0 SW13 for a maximum of four iterations. Each iteration transmits one byte of data to the device controller		
	Enable signal (S/T8L)	(S/T8L) = IOPH0 SW14 DATAOUT	
	Test for VDATAOUT	VDATAOUT = IOPH0 SW14 DATAOUT	
		N(P32 P33)	Not word boundary
		N(SW0)	Not zero byte count
		NPEM	Not parity error in memory
		NADNH	Not address not here
		NED	Not end data
	If VDATAOUT		
	Enable signal BRSW13	BRSW13 = VDATOUT + ...	Return to IOPH0 SW13
	Set flip-flop RSA	S/RSA = (S/RSA)	Apply function strobe acknowledge signal to the device controller
		(S/RSA) = VDATAOUT + ...	
		E/RSA = NRS	
		/RSA/ = RSA	
	Set flip-flop RSCLEN	S/RSCLEN = (S/RSCLEN) NCLEAR	Disable clock at the end of this phase until the device controller returns RS
		(S/RSCLEN) = VDATAOUT + ...	
		R/RSCLEN = ...	
	Enable signal (S/SXDM1)	(S/SXDM1) = VDATAOUT + ...	Prepare adder for (D - 1) ⟶ D in IOPH0 SW13
	If NVDATAOUT		
	Enable signal BRSW8	BRSW8 = IOPH0 SW14 NVDATAOUT DATAOUT + ...	Branch to IOPH1 SW8
	Set flip-flop NIOPH0	S/NIOPH0 = (R/IOPH0) + ...	
		(R/IOPH0) = IOPH0 SW14 NVDATAOUT DATAOUT + ...	
	Reset flip-flop NIOPH1	R/NIOPH1 = (S/IOPH1)	
		(S/IOPH1) = IOPH0 SW14 NVDATAOUT DATAOUT + ...	
IOPH1 SW8 T8L	One clock long		
	1⟶(P15-P33)	PUC33 = IOPH1 SW8 DATAOUT NSW4 + ...	Increment byte address in P-register
		NSW4 = Not data chaining; true in IOPH1 SW8	

(Continued)

Table 3-112. Data-Out Phase Sequence (Cont.)

Phase	Function Performed	Signals Involved			Comments
IOPH1 SW8 T8L (Cont.)	Set flip-flops SW6 and SW7	S/SW6	=	(S/SW6) + ...	Preparation for disconnecting the device controller. If terminal order pending, SW7 will be reset before RSA is raised
		(S/SW6)	=	IOPH1 SW8 DASW4 + ...	
		DASW4	=	NSW1 NSW4	
		NSW1	=	Data out or data in	
		NSW4	=	...	
		R/SW6	=	RESET/A	
		S/SW7	=	(S/SW7) + ...	
		(S/SW7)	=	IOPH1 SW8 DASW4 + ...	
		R/SW7	=	(R/SW7)	
	If memory address error, set status bit B11	S/B11	=	(S/B11) IOPOP + ...	
		(S/B11)	=	ADNH IOIN	
	If data chaining and parity error in memory, set flip-flop B14	S/B14	=	(S/B14) IOPOP + ...	B14 = IOP halt status bit
		(S/B14)	=	(S/B12) + (S/B10) D4 + ...	
		(S/B12)	=	PEM IOPH1 SW8 SW4 + ...	
		(S/B10)	=	PEM NSW1 + ...	
		D4	=	Halt on transmission error flag	
		R/B11	=	BX/1	
	If zero byte count was detected in IOPH0 SW13, set status bit B2 if interrupt on zero byte count flag is high	S/B2	=	(S/B2) IOPOP + ...	
		(S/B2)	=	IOPH1 SW8 SW0 D1 DASW4	
		D1	=	Interrupt on zero byte count flag	
	Test for data chaining condition	IODC	=	SW0 (Zero byte count) D0 (Data chaining flag) NADNH (Not address not here)	
	If IODC and DASW4				
	Set flip-flop SW4	S/SW4	=	(S/SW4)	Store data chaining condition
		(S/SW4)	=	IOPH1 SW8 IODC DASW4 + ...	
		DASW4	=	NSW1 NSW4	
		R/SW4	=	RESET/A	
	Set flip-flop NIOPH1	S/NIOPH1	=	(R/IOPH1) + ...	Exit to IOPH3 SW8 of the data chaining phase sequence. See table 3-111
		(R/IOPH1)	=	IOPH1 SW8 IODC DASW4	
		R/NIOPH1	=	(S/IOPH1)	
	Reset flip-flop NIOPH3	R/NIOPH3	=	(S/IOPH3)	
		(S/IOPH3)	=	IOPH1 SW8 DASW4 IODC + ...	
		S/NIOPH3	=	RESET/A + ...	

(Continued)

Table 3-112. Data-Out Phase Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
IOPH1 SW8 T8L (Cont.)	Enable signal BRSW8	BRSW8 = IOPH1 SW8 IODC DASW4 + ...	
	Enable signal (S/AXRR/4)	(S/AXRR/4) = IOPH1 SW8 DASW4 IODC + ...	Prepare adder for RR ⟶A in IOPH3 SW8 of the data chaining phase sequence
	Reset flip-flop NIOFM	R/NIOFM = ... S/NIOFM = N(S/IOFM) (S/IOFM) = (S/AXRR/4) + ...	Select IOFM, area 10, for source of RR
	Set flip-flop IOFR8	S/IOFR8 = (S/IOFR8) (S/IOFR8) = (S/AXRR/4) + ... R/IOFR8 = ...	
	Maintain IOFR9 in a reset state	R/IOFR9 = ...	
IOPH1 SW9 T8L	One clock long Test for terminal order condition	RTO9 = (S/B10) (Parity error in memory) + B14 (IOP halt) + SW0 (Zero byte count) + SW3 (Terminal order)	
	If RTO9		Terminal order condition exists
	Set flip-flop SW3	S/SW3 = (S/SW3) + ... (S/SW3) = IOPH1 SW9 DASW4 RTO9 DASW4 = NSW1 NSW4 R/SW3 = RESET/A	Store terminal order condition
	Maintain flip-flop SW6 in a set state	S/SW6 = (S/SW6) + ... (S/SW6) = IOPH1 SW9 RTO9 DASW4 + ... R/SW6 = RESET/A	SW6 and NSW7 are used during IOPH1 SW10 to instruct the device controller to request a terminal order
	Reset flip-flop SW7	R/SW7 = (R/SW7) (R/SW7) = IOPH1 SW9 RTO9 DASW4 + ...	
	Enable signal (S/B10) if parity error in memory exits	(S/B10) = PEM NSW1 + ...	
	Set flip-flop B/14 if (S/B10) is true and halt on transmission error flag is high	S/B14 = (S/B14) IOPOP + ... (S/B14) = (S/B10) D4 + ... D4 = Halt on transmission error flag	

(Continued)

Table 3-112. Data-Out Phase Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
IOPH1 SW9 T8L (Cont.)	If NRTO9		If terminal order condition does not exist
	Set flip-flop RSA	S/RSA = (S/RSA) (S/RSA) = IOPH1 SW9 DASW4 NRTO9 + ... E/RSA = NRS	Raise request strobe acknowledge signal to device controller then drop it when device controller drops RS
	Set flip-flop RSACLEN	S/RSACLEN = (S/RSACLEN) NCLEAR (S/RSACLEN) = IOPH1 SW9 NRTO9 DASW4 + ... R/RSACLEN = ... CLEN = RSACLEN NRSA + ...	Delay start of next clock by setting flip-flop RSACLEN. Clock starts again when device controller drops RS, dc-resetting RSA. NRSA RSACLEN drive clock enable signal CLEN true
	Enable signal /ED/	/ED/ = SW6 = Set at IOPH1 SW8 clock	Specify final byte exchange by means of /ED/ /ES/. Meaningful to the device controller only if RSA is high. With /ED/ and /ES/ high, falling edge of RS disconnects device controller
	Enable signal /ES/	/ES/ = SW7 = Set at IOPH1 SW8 clock	
IOPH1 SW10 T5L	One clock long. This portion of the data-out phase sequence (other than I/O restoration) is meaningful only if terminal order conditions (RTO9) existed in IOPH1 SW9		
	Set flip-flop RSA	S/RSA = (S/RSA) (S/RSA) = IOPH1 SW10 DASW4 SW3 + ... E/RSA = NRS	Raise request strobe acknowledge signal to device controller, then drop it when device controller drops RS
	Set flip-flop RSCLEN	S/RSCLEN = (S/RSCLEN) NCLEAR (S/RSCLEN) = IOPH1 SW10 DASW4 SW3 + ... R/RSCLEN = ...	Advance to IOPH1 SW11 then disable clock until device controller returns RS
	Enable signal /ED/	/ED/ = SW6 = Set at IOPH1 SW9 clock	Instruct device controller to request terminal order by means of /ED/ and N/ES/. The state of these two signals is meaningful only when accompanied by RSA
	Disable signal /ES/	N/ES/ = NSW7 = Reset at IOPH1 SW9 clock	

(Continued)

Table 3-112. Data-Out Phase Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
IOPH1 SW11 T8L	One clock long. Duration of clock controlled by device controller via RS		
	Set flip-flop SW7	S/SW7 = TODATA + ... TODATA = IOPH1 SW11 DASW4 SW3 + ... R/SW7 = (R/SW7)	Prepare to specify final byte exchange by means of SW6 and SW7 via /ED/ and /ES/. SW6 was set in IOPH1 SW9
	Assemble terminal order byte in IODA register, as follows:	TODATA	
	Set flip-flop IODA0 if applicable	S/IODA0 = TODATA D1 SW0 + ... D1 = Interrupt on zero byte count flag SW0 = Zero byte count R/IODA0 = IODAX	Interrupt
	Set flip-flop IODA1 if applicable	S/IODA1 = ND0 SW0 TODATA + ... ND0 = Data chain flag is low R/IODA1 = IODAX	Count done
	Set flip-flop IODA2 if applicable	S/IODA2 = D2 TODATA + ... D2 = Command chain flag R/IODA2 = IODAX	Command chain
	Set flip-flop IODA3 if applicable	S/IODA3 = B14 TODATA + ... B14 = IOP halt status bit R/IODA3 = IODAX	IOP halt
IOPH1 SW12 T5L	One clock long		
	Enable signal /ED/	/ED/ = SW6 = Set at IOPH1 SW9 clock	Specify final byte exchange by means of /ED/ and /ES/
	Enable signal /ES/	/ES/ = SW7 = Set at IOPH1 SW11 clock	
	Set flip-flop RSA	S/RSA = (S/RSA) (S/RSA) = IOPH1 SW12 SW3 SW3 = Set at IOPH1 SW9 clock E/RSA = NRS	Raise request strobe acknowledge signal to device controller, then drop it when device controller drops RS

(Continued)

Table 3-112. Data-Out Phase Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
IOPH1 SW12 T5L (Cont.)	Set flip-flop RSACLEN	S/RSACLEN = (S/RSACLEN) NCLEAR (S/RSACLEN) = IOPH1 SW12 SW3 R/RSACLEN = ... CLEN = RSACLEN NRSA	Delay start of next clock by setting flip-flop RSACLEN. Clock starts again when device controller drops RS, dc-resetting RSA. NRSA RSACLEN drive clock enable signal CLEN true. Falling edge of RS also disconnects device controller

Table 3-113. Data-In Phase Sequence

Phase	Function Performed	Signals Involved	Comments
IOPH0 SW13 T8L	This phase is entered initially from IOPH0 SW12. Subsequently, it is reentered from IOPH0 SW14 each time the term VDATAIN is true. The maximum number of iterations is four. When entered from IOPH0 SW14, duration of the clock is controlled by the device controller via RS		
	(D16-D31 minus 1)⟶ (S16-S31)	SXDM1 = Set at IOPH0 SW12 clock (see table 3-502)	Decrement byte count by one
	(S16-S31)⟶/⟶(D16-D31)	DXS/2 = DXS/3 = DXS/4 + DXS DXS/4 = IOPH0 SW13 NSW1 + ... NSW1 = Data in or data out	
	Reset flip-flop IODAP	R/IODAP = IOPH0 SW13	Erase previous data parity fail condition
	Test for zero byte count	S1631Z = N(S16 + S17 + ... S31)	
	If S1631Z, set flip-flop SW0	S/SW0 = S1631Z IOPH0 SW13 NSW1 + ... SW0 = Zero byte count	
	First pass (NRSCLEN)		
	If N(P32 + P33), enable signal (S/CXS)	(S/CXS) = IOPH0 SW13 DATAIN NP32 NP33 NRSCLEN + ... NP32 NP33 during the first pass specifies that four bytes, one byte at a time, are to be received from the device controller	If P32 and P33 are both false, prepare to transfer first byte to the C-register in IOPH0 SW14

(Continued)

Table 3-113. Data-In Phase Sequence (Cont.)

Phase	Function Performed	Signals Involved			Comments
IOPH0 SW13 T8L (Cont.)	Second and subsequent passes (RSCLEN)				
	If NP32 and P33, enable signal AXAR8	AXAR8	=	IOPH0 SW13 DATAIN NP32 P33 + ...	Shift byte 0 in the A-register 8 places to the right
	If P32 and NP33, enable signal AXAR16	AXAR16	=	IOPH0 SW13 DATAIN P32 NP33 + ...	Shift byte 0 in the A-register 16 places to the right
	If P32 and P33, enable signal AXAR24	AXAR24	=	IOPH0 SW13 DATAIN P32 P33 + ...	Shift byte 0 in the A-register 24 places to the right
	If NB0				
	1 $\not\rightarrow$ (P32-P33)	PUC3033	=	PUC31 + PUC33	If when read backward status bit is not high, increment (P32-P33) by one
		PUC33	=	IOPH0 SW13 DATAIN RSCLEN NB0 + ...	
		NB0	=	Not read backward	
		RSCLEN	=	Set at IOPH0 SW14 clock	
	If B0				
	-1 $\not\rightarrow$ (P32-P33)	PSC3033	=	PDC31 + PDC33	If when read backward status bit is high, decrement (P32-P33) by one
		PDC33	=	IOPH0 SW13 DATAIN RSCLEN B0	
IOPH0 SW14 T8L	When VDATAIN is true, this phase alternates with IOPH0 SW13 for a maximum of four iterations. Each iteration transfers one byte of data from the device controller to the A-register				
	(DA0-DA7) $\rightarrow$ (S0-S7)	SXDA	=	IOPH0 SW14 DATAIN NDIS + ...	Transfer data byte via the sum bus to the A-register
	(S0-S7) $\not\rightarrow$ (A0-A7)	AXS/0	=	AXS/4 = AXS/2 + ...	
		AXS/2	=	IOPH0 SW14 DATAIN	
	If P32 and P33 were both false during fhe initial pass of IOPH0 SW13				
	(S0-S7) $\rightarrow$ (C0-C7)	CXS	=	Set at IOPH0 SW13 if NP32 NP33	Load first data byte in the C-register
	Set byte control flip-flops MBXS/0 through MBXS/3, as applicable	S/MBXS/0	=	IOPH0 SW14 DATAIN NP32 NP33	Allows byte 0 to be transferred to core memory in IOPH0 SW15
		R/MBXS/0	=	DRQ	
		S/MBXS/1	=	IOPH0 SW14 DATAIN NP32 P33 + ...	Allows byte 1 to be transferred to core memory in IOPH0 SW15
		R/MBXS/1	=	DRQ	

(Continued)

Table 3-113. Data-In Phase Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
IOPH0 SW14 T8L (Cont.)		S/MBXS/2 = IOPH0 SW14 DATAIN P32 NP33 + ...	Allows byte 2 to be transferred to core memory in IOPH0 SW15
		R/MBXS/2 = DRQ	
		S/MBXS/3 = IOPH0 SW15 DATAIN P32 P33 + ...	Allows byte 3 to be transferred to core memory in IOPH0 SW15
		R/MBXS/3 = DRQ	
	Byte distribution in the A- and C-registers during a typical data-in phase sequence is illustrated below. It is assumed that the initial byte address was NP32 NP33, and that the read backward status bit is false (NB0)		
	AT THE END OF THE FIRST PASS (NP32 NP33)	(A): BYTE 0 — 0 7 8 15 16 23 24 31	(C): BYTE 0 — 0 7 8
	AT THE END OF THE SECOND PASS (NP32 P33)	(A): BYTE 1 — 0 7 8 15 16 23 24 31	(C): BYTE 0 — 0 7 8
	AT THE END OF THE THIRD PASS (P32 NP33)	(A): BYTE 2, BYTE 1 — 0 7 8 15 16 23 24 31	(C): BYTE 0 — 0 7 8
	AT THE END OF THE FOURTH AND FINAL PASS (P32 P33)	(A): BYTE 3, BYTE 1, BYTE 2 — 0 7 8 15 16 23 24 31	(C): BYTE 0 — 0 7 8
			901172A.3622
	If PC, test for data parity	PC = /PC/	
		/PC/ = Used by device controller to specify that parity check is necessary	
	If parity checks, enable signal IOPC	IOPC = Sum of true data bits plus parity bit is odd	
	If parity fails, set flip-flop IODAP	S/IODAP = (S/IODAP)	
		(S/IODAP) = IOPH0 SW14 DATAIN NIOPC PC + ...	Flip-flop IODAP is used during data in to store a data parity fail condition, during data out to generate data parity bit DAP
		PC = Parity check required	
		NIOPC = Parity failed	
		R/IODAP = IOPH0 SW13	
	Test for VDATAIN	VDATAIN = IOPH0 SW14 DATAIN NED (IOPC + NPC)	Not end data Data parity checks, or data parity check not required

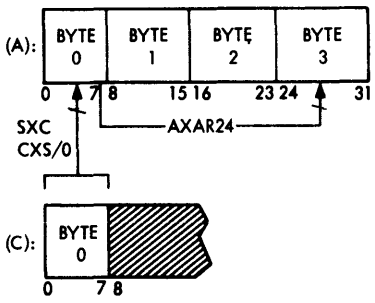
(Continued)

Table 3-113. Data-In Phase Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
IOPH0 SW14 T8L (Cont.)		N(NP32 NP33 B0)	Not word boundary on read backward
		N(P32 P33 NB0)	Not word boundary on read forward
	If VDATAIN		
	Set flip-flop RSA	S/RSA = (S/RSA) (S/RSA) = VDATAIN + ... E/RSA = NRS	Apply function strobe acknowledge signal to the device controller, then drop it when NRS
	Set flip-flop RSCLEN	S/RSCLEN = (S/RSCLEN) NCLEAR (S/RSCLEN) = VDATAIN + ... R/RSCLEN = ...	Disable clock until device controller returns RS
	Enable signal (S/SXDM1)	(S/SXDM1) = VDATAIN + ...	Prepare adder for (D -1) ⟶ D in IOPH0 SW13
	Enable signal BRSW13	BRSW13 = VDATAIN + ...	Return to IOPH0 SW13
	If NVDATAIN, enable signal (S/SXC)	(S/SXC) = IOPH0 SW14 NVDATAIN DATAIN + ...	Prepare adder for C⟶ S in IOPH0 SW15
IOPH0 SW15 T5L	One clock long If NB0 and NSW4		
	(C0-C7)⟶(S0-S7)	SXC = Set at IOPH0 SW14 clock	Transfer byte 0 from the C-register via the sum bus to the A-register
	(S0-S7)⟶/⟶(A0-A7)	AXS/0 = AXS/4 AXS/4 = AXS/2 + ... AXS/2 = IOPH0 SW15 DATAIN NSW4 NB0 NSW4 = Not data chaining NB0 = Read forward	
	Enable signal AXAR8, AXAR16, or AXAR24, as applicable	AXAR8 = IOPH0 SW15 DATAIN NSW4 NP32 P33 + ... AXAR16 = IOPH0 SW15 DATAIN NSW4 P32 NP33 + ... AXAR24 = IOPH0 SW15 DATAIN NSW4 P32 P33 + ...	Final byte alignment in A-register
	Enable signal (S/SXA)	(S/SXA) = IOPH0 SW15 DATAIN NSW4 + ...	
	If skip flag is not high enable signal (S/MBXS)	(S/MBXS) = IOPH0 SW15 DATAIN NSW4 ND7 + ...	Prepare to store data word in core memory in IOPH1 SW8

(Continued)

Table 3-113. Data-In Phase Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
IOPH0 SW15 T5L (Cont.)	Set flip-flops MRQ and DRQ	S/MRQ = (S/MBXS) + ... R/MRQ = ... S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = (S/MBXS) + ... R/DRQ = ...	
	Reset flip-flop NIOPH1	R/NIOPH1 = (S/IOPH1) (S/IOPH1) = IOPH0 SW15 + ... S/NIOPH1 = RESET/A + (R/IOPH1)	Advance to IOPH1 SW8
	Set flip-flop NIOPH0	S/NIOPH0 = (R/IOPH0) + ... (R/IOPH0) = IOPH0 SW15 + ... R/NIOPH0 = (S/IOPH0)	
	Enable signal BRSW8	BRSW8 = IOPH0 SW15 + ...	
	Final byte alignment in A-register at the end of this clock (continuation of example illustrated in IOPH0 SW14 portion of phase sequence chart)	 901172A.3623	
IOPH1 SW8 DR	One clock long (A0-A31)⟶(S0-S31) (S0-S31)⟶(MB0-MB31)	SXA = Set at IOPH0 SW15 clock MBXS/0-MBXS/3 = Applicable flip-flop (or flip-flops) set at IOPH0 SW14 clock	Store data word in core memory
	If NB0 1⟶(P15-P31)	PUC33 = IOPH1 SW8 DATAIN NSW4 NB0 + ... NB0 = Read forward NSW4 = Not data chaining	Increment byte address in P-register
	If B0 -1⟶(P15-P31)	PDC33 = IOPH1 SW8 DATAIN NSW4 B0 + ... B0 = Read backward	Decrement byte address in P-register

(Continued)

Table 3-113. Data-In Phase Sequence (Cont.)

Phase	Function Performed	Signals Involved			Comments
IOPH1 SW8 DR (Cont.)	Set flip-flops SW6 and SW7	S/SW6	=	(S/SW6) + ...	Preparation for disconnecting the device controller. If terminal order pending, SW7 will be reset before RSA is raised
		(S/SW6)	=	IOPH1 SW8 DASW4 + ...	
		DASW4	=	NSW1 NSW4	
		NSW1	=	Data in or data out	
		R/SW6	=	RESET/A	
		S/SW7	=	(S/SW7) + ...	
		(S/SW7)	=	IOPH1 SW8 DASW4 + ...	
		R/SW7	=	(R/SW7)	
	If memory address error, set status bit B11	S/B11	=	(S/B11) IOPOP + ...	
		(S/B11)	=	ADNH IOIN	
	If data parity error, set status bit B9	S/B9	=	(S/B9) IOPOP + ...	
		(S/B9)	=	IOPH1 SW8 DATAIN DAP + ...	
		R/B9	=	BX/1	
	If memory address error, or data priority error with halt on transmission flag high, set IOP halt flip-flop B14	S/B14	=	(S/B14) IOPOP + ...	
		(S/B14)	=	(S/B12) + (S/B10) D4 + ...	
		(S/B12)	=	PEM + IOPH1 SW8 SW4 + ...	
		PEM	=	Parity error in memory	
		SW4	=	Data chaining	
		(S/B10)	=	PEM NSW1 + ...	
		D4	=	Halt on transmission error flag	
	If zero byte count was detected in IOPH0 SW13 and interrupt on zero byte count flag is high, set flip-flop B2	S/B2	=	(S/B2) IOPOP + ...	
		(S/B2)	=	IOPH1 SW8 SW0 D1 DASW4	
		D1	=	Interrupt on zero byte count flag	
	Test for data chaining condition	IODC	=	SW0 D0 NADNH	
		SW0	=	Zero byte count; set in IOPH0 SW13	
		D0	=	Data chaining flag	
		NADNH	=	No memory address error	
	If IODC and DASW4				
	Set flip-flop SW4	S/SW4	=	(S/SW4)	Store data chaining condition
		(S/SW4)	=	IOPH1 SW8 IODC DASW4 + ...	
		R/SW4	=	RESET/A	

(Continued)

Table 3-113. Data-In Phase Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
IOPH1 SW8 DR (Cont.)	Reset flip-flop NIOPH3	R/NIOPH3 = (S/IOPH3) (S/IOPH3) = IOPH1 SW8 DASW4 IODC S/NIOPH3 = RESET/A + ...	Exit to IOPH3 SW8 of the data chaining phase sequence. See table 3-111
	Enable signal BRSW8	BRSW8 = IOPH1 SW8 IODC DASW4 + ...	
	Set flip-flop NIOPH1	S/NIOPH1 = (R/IOPH1) + ... (R/IOPH1) = IOPH1 SW8 IODC DASW4	
	Enable signal (S/AXRR/4)	(S/AXRR/4) = IOPH1 SW8 DASW4 IODC + ...	Prepare adder for RR ⟶A in IOPH3 SW8 of the data chaining phase sequence
	Reset flip-flop NIOFM	R/NIOFM = ... S/NIOFM = N(S/IOFM) (S/IOFM) = (S/AXRR/4) + ...	Select IOFM register, area 10, for source of RR
	Set flip-flop IOFR8	S/IOFR8 = (S/IOFR8) (S/IOFR8) = (S/AXRR/4) + ... R/IOFR8 = ...	
	Maintain flip-flop IOFR9 in the reset state	R/IOFR9 = ...	
	If NIODC, exit to IOPH1 SW9		
IOPH1 SW9 T8L	One clock long		
	Test for terminal order condition	RTO9 = (S/B10) (Parity error in memory) + B14 (IOP halt) + SW0 (Zero byte count) + SW3 (Terminal order)	
	If RTO9		Terminal order condition exists
	Set flip-flop SW3	S/SW3 = (S/SW3) + ... (S/SW3) = IOPH1 SW9 DASW4 RTO9 DASW4 = NSW1 NSW4 R/SW3 = RESET/A	Store terminal order condition
	Maintain flip-flop SW6 in a set state	S/SW6 = (S/SW6) + ... (S/SW6) = IOPH1 SW9 RTO9 DASW4 + ... R/SW6 = RESET/A	SW6 and NSW7 are used during IOPH1 SW10 to instruct the device controller to request a terminal order

(Continued)

Table 3-113. Data-In Phase Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
IOPH1 SW9 T8L (Cont.)	Reset flip-flop SW7	R/SW7 = (R/SW7) (R/SW7) = IOPH1 SW9 RTO9 DASW4 + ...	
	Enable signal (S/B10) if parity error in memory exits	(S/B10) = PEM NSW1 + ...	
	Set flip-flop B/14 if (S/B10) is true and halt on transmission error flag is high	S/B14 = (S/B14) IOPOP + ... (S/B14) = (S/B10) D4 + ... D4 = Halt on transmission error flag	
	If NRTO9		If terminal order condition does not exist
	Set flip-flop RSA	S/RSA = (S/RSA) (S/RSA) = IOPH1 SW9 DASW4 NRTO9 + ... E/RSA = NRS	Raise request strobe acknowledge signal to device controller, then drop it when device controller drops RS
	Set flip-flop RSACLEN	S/RSACLEN = (S/RSACLEN) NCLEAR (S/RSACLEN) = IOPH1 SW9 NRTO9 DASW4 + ... R/RSACLEN = ... CLEN = RSACLEN NRSA + ...	Delay start of next clock by setting flip-flop RSACLEN. Clock starts again when device controller drops RS, dc-resetting RSA. NRSA RSACLEN drive clock enable signal CLEN true
	Enable signal /ED/	/ED/ = SW6 = Set at IOPH1 SW8 clock	Specify final byte exchange by means of /ED/ /ES/. Meaningful to the device controller only if RSA is high. With /ED/ and /ES/ high, falling edge of RS disconnects device controller
	Enable signal /ES/	/ES/ = SW7 = Set at IOPH1 SW8 clock	
IOPH1 SW10 T5L	One clock long. This portion of the data-in phase sequence (other than I/O restoration) is meaningful only if terminal order conditions (RTO9) existed in IOPH1 SW9		
	Set flip-flop RSA	S/RSA = (S/RSA) (S/RSA) = IOPH1 SW10 DASW4 SW3 + ... E/RSA = NRS	Raise request strobe acknowledge signal to device controller, then drop it when device controller drops RS

(Continued)

Table 3-113. Data-In Phase Sequence (Cont.)

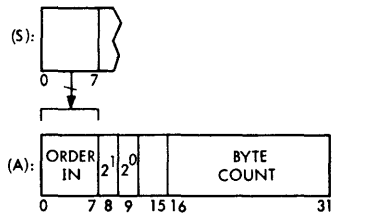
Phase	Function Performed	Signals Involved	Comments
IOPH1 SW10 T5L (Cont.)	Set flip-flop RSCLEN	S/RSCLEN = (S/RSCLEN) NCLEAR (S/RSCLEN) = IOPH1 SW10 DASW4 SW3 + ... R/RSCLEN = ...	Advance to IOPH1 SW11 then disable clock until device controller returns RS
	Enable signal /ED/	/ED/ = SW6 = Set at IOPH1 SW9 clock	Instruct device controller to request terminal order by means of /ED/ and N/ES/. The state of these two signals is meaningful only when accompanied by RSA
	Disable signal /ES/	N/ES/ = NSW7 = Reset at IOPH1 SW9 clock	
IOPH1 SW11 T8L	One clock long. Duration of clock controlled by device controller via RS		
	Set flip-flop SW7	S/SW7 = TODATA + ... TODATA = IOPH1 SW11 DASW4 SW3 + ... R/SW7 = (R/SW7)	Prepare to specify final byte exchange by means of SW6 and SW7 via /ED/ and /ES/. SW6 was set in IOPH1 SW9
	Assemble terminal order byte in IODA register, as follows:	TODATA	
	Set flip-flop IODA0 if applicable	S/IODA0 = TODATA D1 SW0 + ... D1 = Interrupt on zero byte count flag SW0 = Zero byte count R/IODA0 = IODAX	Interrupt
	Set flip-flop IODA1 if applicable	S/IODA1 = ND0 SW0 TODATA + ... ND0 = Data chain flag is low R/IODA1 = IODAX	Count done
	Set flip-flop IODA2 if applicable	S/IODA2 = D2 TODATA + ... D2 = Command chain flag R/IODA2 = IODAX	Command chain
	Set flip-flop IODA3 if applicable	S/IODA3 = B14 TODATA + ... B14 = IOP halt status bit R/IODA3 = IODAX	IOP halt
IOPH1 SW12 T5L	One clock long Enable signal /ED/ Enable signal /ES/	/ED/ = SW6 = Set at IOPH1 SW9 clock /ES/ = SW7 = Set at IOPH1 SW11 clock	Specify final byte exchange by means of /ED/ and /ES/

(Continued)

Table 3-113. Data-In Phase Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
IOPH1 SW12 T5L (Cont.)	Set flip-flop RSA	S/RSA = (S/RSA) (S/RSA) = IOPH1 SW12 SW3 SW3 = Set at IOPH1 SW9 clock E/RSA = NRS	Raise request strobe acknowledge signal to device controller, then drop it when device controller drops RS
	Set flip-flop RSACLEN	S/RSACLEN = (S/RSACLEN) NCLEAR (S/RSACLEN) = IOPH1 SW12 SW3 R/RSACLEN = ... CLEN = RSACLEN NRSA	Delay start of next clock by setting flip-flop RSACLEN. Clock starts again when device controller drops RS, dc-resetting RSA. NRSA RSACLEN drive clock enable signal CLEN true. Falling edge of RS also disconnects device controller

Table 3-114. Order-In Phase Sequence

Phase	Function Performed	Signals Involved	Comments
IOPH0 SW13 T5L	(/DA0/-/DA7/)⟶(S0-S7) (S0-S7)⟶(A0-A7)  901172A.3624	SXDA = NDIS ORDERIN IOPH0 SW13 + ... AXS/0 = AXS/4 AXS/4 = AXS/2 + ... AXS/2 = IOPH0 SW13 ORDERIN + ... ORDERIN = SW1 NSW2	Load order-in byte, supplied by the device controller via data lines /DA0/-/DA7/, into the A-register
	Enable signal BRSW15	BRSW15 = IOPH0 SW13 ORDERIN	Advance to IOPH0 SW15
IOPH0 SW15 T5L	One clock long Set status flip-flops as applicable If A0, set flip-flop B9	S/B9 = (S/B9) IOPOP + ... (S/B9) = ODINST A0 + ... ODINST = ORDERIN IOPH0 SW15 R/B9 = BX/1	A0 represents the transmission error bit of the order-in byte
	If A1, set flip-flop B8	S/B8 = (S/B8) IOPOP + ... (S/B8) = ODINST A1 R/B8 = BX/1	A1 represents the incorrect length bit of the order-in byte

(Continued)

Table 3-114. Order-In Phase Sequence (Cont.)

Phase	Function Performed	Signals Involved			Comments
IOPH0 SW15 T5L (Cont.)	If (S/B9) and D4, or (S/B8) and D4 and ND6, set flip-flop B14	S/B14	=	(S/B14) IOPOP + ...	D4 represents the halt on transmission error flag; D6 represents the suppress incorrect length flag
		(S/B14)	=	(S/B14/1) + ...	
		(S/B14/1)	=	(S/B8) D4 ND6 + (S/B9) D4	
	Set flip-flop IOPH0	S/NIOPH0	=	(R/IOPH0) + ...	Change to IOPH1 SW8
		(R/IOPH0)	=	IOPH0 SW15 + ...	
		R/NIOPH0	=	(S/IOPH0)	
	Reset flip-flop NIOPH1	R/NIOPH1	=	(S/IOPH1)	
		(S/IOPH1)	=	IOPH0 SW15 + ...	
		S/NIOPH1	=	(R/IOPH1) + RESET/A	
	Enable signal BRSW8	BRSW8	=	IOPH0 SW15 + ...	
	Set flip-flop SW6	S/SW6	=	(S/SW6) + ...	
		(S/SW6)	=	IOPH0 SW15 (SW1 + ...) + ...	
		R/SW6	=	RESET/A	
	Reset flip-flop SW7	R/SW7	=	(R/SW7)	SW6 and NSW7 are used during IOPH1 SW8 to instruct the device controller to request a terminal order
		(R/SW7)	=	IOPH0 SW15 + ...	
		S/SW7	=	(S/SW7) + TODATA	
	Set flip-flop SW3	S/SW3	=	ODINST + ...	Flip-flop SW3 stores a terminal order condition, and is used during IOPH1 SW12 as a qualifying term
		ODINST	=	ORDERIN IOPH0 SW15	
		R/SW3	=	RESET/A	
IOPH1 SW8 T8L	One clock long				
	Enable signal /ED/	/ED/	=	SW6	Instruct device controller by means of /ED/ and N/ES/ to request a terminal order
	Disable signal /ES/	N/ES/	=	NSW7	
	Set flip-flop RSA	S/RSA	=	(S/RSA)	Raise request strobe acknowledge to device controller, then drop it when device controller drops request strobe
		(S/RSA)	=	IOPH1 SW8 ORDSW4	
		ORDSW4	=	SW1 + SW4	
		E/RSA	=	NRS	
		/RSA/	=	RSA	
	Set flip-flop RSCLEN	S/RSCLEN	=	(S/RSCLEN) NCLEAR	Advance to IOPH1 SW9, then disable clock until device controller returns RS
		(S/RSCLEN)	=	IOPH1 SW8 ORDSW4 + ...	
		R/RSCLEN	=	...	
		CLEN	=	NRSCLEN + ...	

(Continued)

Table 3-114. Order-In Phase Sequence (Cont.)

Phase	Function Performed	Signals Involved			Comments
IOPH1 SW9 T8L	One clock long. Duration of clock controlled by device controller via RS				
	Set flip-flop SW6	S/SW6	=	TODATA + ...	Prepare to specify final byte exchange between integral IOP and device controller
		TODATA	=	IOPH1 SW9 ORDSW4 + ...	
		ORDSW4	=	SW1 + SW4	
		R/SW6	=	RESET/A	
	Set flip-flop SW7	S/SW7	=	TODATA + ...	
		R/SW7	=	(R/SW7) + ...	
	Set the following status flip-flops, as applicable				
	Set flip-flop B1	S/B1	=	(S/B1) IOPOP + ...	Bits 0 through 7 of the A-register contain the order-in byte supplied by the device controller. Bits 0 through 7 of the D-register contain flags originally obtained from the IOFM register
		(S/B1)	=	TORDIN A2	
		TORDIN	=	ORDERIN IOPH1 SW9	
		A2	=	Chaining modifier bit of order-in byte	
		R/B1	=	(R/B1)	
	Set flip-flop B3	S/B3	=	(S/B3) IOPOP + ...	
		(S/B3)	=	TORDIN A3 D3	
		A3	=	Channel end	
		D3	=	Interrupt on channel end	
		R/B3	=	BX	
	Set flip-flop B4	S/B4	=	(S/B4) + ...	
		(S/B4)	=	TORDIN A4 D5	
				+ TORDIN A0 A3 D4 D5	
				+ TORDIN A1 ND6 A3 D4 D5	
		A1	=	Incorrect length	
		A4	=	Unusual end	
		A0	=	Transmission error	
		D3	=	Interrupt on channel end	
		D4	=	Halt on transmission error	
		D5	=	Interrupt on unusual end	
		ND6	=	Not suppress incorrect length	
		R/B4	=	BX	
	If (S/B3), or (S/B4), set flip-flop IODA0	S/IODA0	=	(S/IODA0) + ...	Assemble terminal order in IODA register
		(S/IODA0)	=	(S/B3) + (S/B4)	
		R/IODA0	=	IODAX	

(Continued)

Table 3-114. Order-In Phase Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
IOPH1 SW9 T8L (Cont.)	If D2, set flip-flop IODA2	S/IODA2 = TODATA D2 + ... D2 = Command chaining R/IODA2 = IODAX	
	If B14, set flip-flop IODA3	S/IODA3 = TODATA B14 + ... B14 = IOP halt R/IODA3 = IODAX	
	(IODA0-IODA7) ⟶ (/DA0/-/DA7/)	/DA0/-/DA7/ = IODA0-IODA7 + ...	From the IODA register the terminal order is transmitted automatically by data lines /DA0/-/DA7/ to the device controller
	When RS, enable signal CLEN	CLEN = RSCLEN NRSA RS	Enable clock when RS is obtained from device controller
IOPH1 SW12 T5L	One clock long. Start of next clock controlled by device controller via NRS		
	Enable signal /ED/ Enable signal /ES/	/ED/ = SW6 /ES/ = SW7	Specify final byte exchange by means of /ED/ and /ES/
	Set flip-flop RSA	S/RSA = (S/RSA) (S/RSA) = IOPH1 SW12 SW3 SW3 = Set at IOPH0 SW15 clock E/RSA = NRS	Raise request strobe acknowledge signal to device controller, then drop it when device controller drops request strobe
	Set flip-flop RSACLEN	S/RSACLEN = (S/RSACLEN) NCLEAR (S/RSACLEN) = IOPH1 SW12 SW3 R/RSACLEN = ... CLEN = RSACLEN NRSA	Delay start of next clock by setting flip-flop RSACLEN. Clock starts again when device controller drops RS, dc-resetting RSA. NRSA and RSACLEN drive clock enable signal CLEN true. Falling edge of RS also disconnects device controller

Table 3-115. I/O Restoration Phase Sequence

Phase	Function Performed	Signals Involved			Comments
IOPH1 SW8 T8L	Enable signal (S/AXRR/3)	(S/AXRR/3)	=	IOPH1 SW8 (ORDSW4 + DASW4 IODC)	Prepare to transfer contents of IOFM, area 01, to the A-register
	Reset flip-flop NIOFM	R/NIOFM	=	...	
		S/NIOFM	=	N(S/IOFM)	
		(S/IOFM)	=	(S/AXRR/3) + ...	
	Set flip-flop IOFR9	S/IOFR9	=	(S/IOFR9) IOPOP	
		(S/IOFR9)	=	(S/AXRR/3) + ...	
	Maintain IOFR8 in a reset state	R/IOFR8	=	...	
		S/IOFR8	=	(S/IOFR8)	
IOPH1 SW9 T8L	(RR0-RR31)—/→(A0-A31)	AXRR	=	Set at IOPH1 SW8 clock	Transfer contents of IOFM, area 01, to the A-register
		IOFM	=	Set at IOPH1 SW8 clock	
		IOFR9	=	Set at IOPH1 SW8 clock	
		NIOFR8	=	Reset at IOPH1 SW8 clock	
	(B15-B31)——→(S15-S31)	SXB	=	Set at IOPH1 SW8 clock	Exchange contents of P-register with contents of B-register
	(S15-S31)—/→(P15-P31)	PXS	=	IOPH1 SW9 + ...	
	(P15-P31)—/→(B15-B31)	BXP	=	BXP/1 + ...	
		BXP/1	=	IOPH1 SW9 + ...	
	Set flip-flop BRP	S/BRP	=	(S/BRP) + ...	Indicates that P-register contains the program address
		(S/BRP)	=	IOPH1 SW9 + ...	
	If NIFAM NRTO9 Enable signal BRSW11	BRSW11	=	IOPH1 SW9 NIFAM NRTO9	If IOPH10, advance to IOPH1 SW11
		RTO9	=	SW3 + ...	
		IFAM	=	IFAST/S + IFAST/L + IFAMDS	
		NIFAM	⇒	IOPH10	
	Enable signal (S/RW/2)	(S/RW/2)	=	IOPH1 SW9 NRTO9	
	Reset flip-flop NIOFM	R/NIOFM	=	...	
		S/NIOFM	=	N(S/IOFM)	
		(S/NIOFM)	=	(S/RW/2) + ...	
	Enable signal (S/SXB)	(S/SXB)	=	IOPH1 SW9 NIFAM NRTO9	
	If SC, set flip-flop IOSC	S/IOSC	=	(S/IOSC)	If another service call is pending, acknowledge it at this point
		(S/IOSC)	=	SC NSCINH IOPOP	
		SCINH	=	N(IOPH1 SW9) IOACT + ...	
		R/IOSC	=	IOPH1 SW13 IOB0 + ...	

(Continued)

Table 3-115. I/O Restoration Phase Sequence (Cont.)

Phase	Function Performed	Signals Involved			Comments
IOPH1 SW9 T8L (Cont.)	If IFAM				
	Enable signal (S/AXRR)	(S/AXRR)	=	(S/AXRR/2) + ...	Prepare to transfer contents of IOFM, area 00, to the A-register
		(S/AXRR/2)	=	IOPH1 SW9 IFAM	
		IFAM	=	IFAST/L + IFAST/S + IFAMDS	
	Reset flip-flop NIOFM	R/NIOFM	=	...	
		S/NIOFM	=	N(S/IOFM)	
		(S/IOFM)	=	(S/AXRR/2) + ...	
	Maintain flip-flops IOFR8 and IOFR9 in the reset state	R/IOFR8	=	R/IOFR9 + ...	
		S/IOFR8	=	(S/IOFR8) + (S/AXRR/4) + ...	
		S/IOFR9	=	(S/IOFR9) IOPOP	
		(S/IOFR9)	=	(S/AXRR/3) + (S/AXRR/6) + ...	
	Enable signal (S/CXS)	(S/CXS)	=	IOPH1 SW9 IFAM + ...	Prepare to transfer contents of sum bus to the C-register
	Enable signal (S/SXA)	(S/SXA)	=	IOPH1 SW9 IFAM + ...	Preset adder for A⟶S in IOPH1 SW10
IOPH1 SW10 T5L	This phase entered from IOPH0 SW9 if IFAM				
	One clock long				
	(A0-A31) ⟶ (S0-S31)	SXA	=	Set at IOPH1 SW9 clock	Transfer contents of A-register via sum bus to C-register
	(S0-S31) ⟶ (C0-C31)	CXS	=	Set at IOPH1 SW9 clock	
	(RR0-RR31) ⟶ (A0-A31)	AXRR	=	Set at IOPH1 SW9 clock	Transfer contents of IOFM, area 00, to the A-register
		NIOFR8	=	Reset at IOPH1 SW9 clock	
		NIOFR9	=	Reset at IOPH1 SW9 clock	
	If IOSC, set flip-flop IOFS	S/IOFS	=	IOSC NPCP3 + ...	If flip-flop IOSC was set at the end of IOPH1 SW9, raise function strobe
	Enable signal (S/RW/2)	(S/RW/2)	=	IOPH1 SW10 + ...	Prepare to transfer new IOP status from B-register to IOFM, area 00
	Enable signal (S/SXB)	(S/SXB)	=	IOPH1 SW10	
	Reset flip-flop NIOFM	R/NIOFM	=	...	
		S/NIOFM	=	N(S/IOFM)	
		(S/NIOFM)	=	(S/RW/2) + ...	
	Maintain flip-flops IOFR8 and IOFR9 in their reset states	R/IOFR8	=	R/IOFR9 + ...	
		S/IOFR8	=	(S/IOFR8)	
		S/IOFR9	=	(S/IOFR9) IOPOP	

(Continued)

Table 3-115. I/O Restoration Phase Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
IOPH1 SW11 T8L One clock long			
	(B0-B31)⟶(S0-S31)	SXB = Set at IOPH1 SW9 clock or IOPH1 SW10 clock, as applicable	Transfer new IOP status from B-register via sum bus to IOFM, area 00
		RWXS/0-RWXS/1 = NRWXZ (RW-2 + ...)	
		RW-2 = RW NCROSSEN	
		RWXS/2 = RW-1 + ...	
		RW-1 = RW-2	
		RWXS/3 = RW-1 + ...	
	(S0-S31)⟶/⟶(RW0-RW31)	RW = Set at IOPH1 SW9 clock or IOPH1 SW10 clock, as applicable	
		IOFM = Set at IOPH1 SW9 clock or IOPH1 SW10 clock, as applicable	
		NIOFR8 = Not set at last clock	
		NIOFR9 = Not set at last clock	
	0⟶/⟶(D8-D15)	R/D8 = DX/1	Clear old byte count
		DX/1 = D0815XZ + ...	
		D0815XZ = IOPH1 SW11	
		R/D9 = DX/1 ⋮ ⋮ R/D15 = DX/1	
	If flip-flop IOSC was set in IOPH1 SW9,and IOPH1 SW10 was not entered, set flip-flop IOFS	S/IOFS = IOSC NPCP3 + ...	Raise function strobe
	If flip-flop IOSC was set in IOPH1 SW9, set flip-flop IOEN	S/IOEN = (S/IOEN)	IOPH1 SW11 is one of four interruptible points for I/O service. The other three are shown in phase IOEN NIOIN of table 3-108
		(S/IOEN) = IOPH1 SW11 IOSC NIOINH + ...	
		R/IOEN = (R/IOEN) + ...	
	If IFAM, enable signal (S/SXA)	(S/SXA) = IOPH1 SW11 IFAM + ...	Prepare adder for A⟶S in IOPH1 SW12
	If FAMDS, enable signal (S/AXRR)	(S/AXRR) = IOPH1 SW11 FAMDS + ...	Prepare to transfer contents of general register R to the A-register

(Continued)

Table 3-115. I/O Restoration Phase Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
IOPH1 SW11 T8L (Cont.)	If IFAST		
	Enable signal (S/AXRR)	(S/AXRR) = (S/AXRR/3) + ... (S/AXRR/3) = IOPH1 SW11 IFAST + ...	Prepare to transfer contents of IOFM, area 01, to the A-register
	Reset flip-flop NIOFM	R/NIOFM = ... S/NIOFM = N(S/IOFM) (S/IOFM) = (S/AXRR/3) + ...	
	Set flip-flop IOFR9	S/IOFR9 = (S/IOFR9) IOPOP (S/IOFR9) = (S/AXRR/3) + ... R/IOFR9 = ...	
	Leave flip-flop IOFR8 in a reset state	R/IOFR8 = ... S/IOFR8 = (S/IOFR8)	
	If IOPH10 and not SW3, enable signal BRSW13	BRSW13 = IOPH1 SW1 NIFAM NSW3 NIFAM ⇒ IOPH10	Branch to SW13
IOPH1 SW12	If IFAST		
	(A0-A31)⟶(S0-S31)	SXA = Set at IOPH1 SW11 clock	Transfer contents of A-register to P-register
	(S15-S31)⟶(P15-P31)	PXS = IOPH1 SW12 IFAST + ...	
	(P15-P31)⟶(B15-B31)	BXP = BXP/1 + ... BXP/1 = IOPH1 SW12 IFAST + ...	Transfer contents of P-register to B-register
	If IFAMDS		
	(A0-A31)⟶(S0-S31)	SXA = Set at IOPH1 SW11 clock	Transfer contents of A-register to B-register
	(S0-S31)⟶(B0-B31)	BXS/0 = BXS/1 = BXS + ... BXS = IOPH1 SW12 IFAMDS + ...	
	(RR0-RR31)⟶(A0-A31)	AXRR = Set at IOPH1 SW11 clock	Transfer contents of general register R to A-register
	If IFAST		
	(RR0-RR31)⟶(A0-A31)	AXRR = Set at IOPH1 SW11 clock IOFM = Set at IOPH1 SW11 clock IOFR9 = Set at IOPH1 SW11 clock NIOFR8 = Not set at IOPH1 SW11	Transfer contents of IOFM register, area 01, to A-register
	Enable signal (S/SXD)	(S/SXD) = IOPH1 SW12 + ...	Prepare adder for D⟶S in IOPH1 SW13
	Enable signal (S/RW)	(S/RW) = (S/RW/3) + ... (S/RW/3) = IOPH1 SW12 + ...	Prepare to transfer contents of sum bus to IOFM, area 01

(Continued)

Table 3-115. I/O Restoration Phase Sequence (Cont.)

Phase	Function Performed	Signals Involved			Comments
IOPH1 SW12 (Cont.)	Reset flip-flop NIOFM	R/NIOFM	=	...	
		S/NIOFM	=	N(S/IOFM)	
		(S/IOFM)	=	(S/RW/3) + ...	
	Set flip-flop IOFR9	S/IOFR9	=	(S/IOFR9) IOPOP	
		(S/IOFR9)	=	(S/RW/3) + ...	
		R/IOFR9	=	...	
	Leave flip-flop IOFR8 in the reset state	R/IOFR8	=	...	
		S/IOFR8	=	(S/IOFR8)	
IOPH1 SW13 NIOB0 T8L	One clock long				
	(D0-D31)⟶(S0-S31)	SXD	=	Set at IOPH1 SW12 clock	Transfer contents of D-register via sum bus to IOFM, area 01
	(S0-S31)⟶/⟶(RW0-RW31)	RWXS/0	=	RWXS/1 = NRWXZ (RW-2 + ...)	
		RW-2	=	RW NCROSSEN	
		RWXS/2	=	RW-1 + ...	
		RWXS/3	=	RW-1 + ...	
		RW-1	=	RW-2	
		RW	=	Set at IOPH1 SW12 clock	
		IOFM	=	Set at IOPH1 SW12 clock	
		IOFR9	=	Set at IOPH1 SW12 clock	
		NIOFR8	=	Not set at IOPH1 SW12 clock	
	P32⟶S8	S8	=	P32 S0809XP + ...	Transfer byte level bits via sum bus into IOFM, area 01, bit positions 8 and 9
		S0809XP	=	IOPH1 SW13 NDIS-4	
	P33⟶S9	S9	=	P33 S0809XP + ...	
	(C0-C31)⟶/⟶(D0-D31)	DXC	=	IOPH1 SW13 NBXBR2 + ...	Transfer contents of C-register to D-register
	If flip-flop IOEN was set in IOPH1 SW11 and FSL is returned by device controller, maintain flip-flop IOIN in a set state; otherwise, reset flip-flop IOIN	S/IOIN	=	(S/IOIN)	Setting of flip-flop IOIN prepares CPU to process new service call and inhibits all other CPU functions
		(S/IOIN)	=	FSL IOEN NIOINH NPH6	
		R/IOIN	=	RESET/A	
		RESET/A	=	IOPH1 SW13 + ...	

(Continued)

Table 3-115. I/O Restoration Phase Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
IOPH1 SW13 NIOB0 T8L (Cont.)	If flip-flop IOEN was set in IOPH1 SW11, but FSL was not returned by device controller, reset flip-flop IOIN and branch to IOEN NIOIN NIOPH1. See table 3-116	R/IOIN = RESET/A RESET/A = IOPH1 SW13 + ...	Indicates that device controller, which originally raised new service call, has in the meantime dropped its service call pending condition. During IOEN NIOIN INOPH1 the I/O operation is aborted
	Set flip-flop NIOPH1	S/NIOPH1 = RESET/A + ...	End I/O sequence
IOB0 + NIOEN	If flip-flop IOEN was not set in IOPH1 SW11 and NIPH10 and NPCP2:		
	and IFAST/L and OU0, enable signals PUC31 and RUC31	PUC31 = IOPH1 SW13 IFAST/L OU0 (NIOEN + ...) + ...	Increment P-register by one
		RUC31 = IOPH1 SW13 IFAST/L OU0 (NIOEN + ...) + ...	Increment private memory register R by one
	and IFAST/L and NOU0, enable signals PDC31 and RDC31	PDC31 = IOPH1 SW13 IFAST/L NOU0 (NIOEN + ...) + ...	Decrement P-register by one
		RDC31 = IOPH1 SW13 IFAST/L NOU0 (NIOEN + ...) + ...	Decrement private memory register R by one
	and IFAST/S, enable signals PDC31, RDC31, and (S/AXRR)	PDC31 = IOPH1 SW13 IFAST/S (NIOEN + ...) + ...	Decrement P-register by one
		RDC31 = IOPH1 SW13 IFAST/S (NIOEN + ...) + ...	Decrement private memory register by one
		(S/AXRR) = IOPH1 SW13 IFAST/S (NIOEN + ...) + ...	Prepare to transfer contents of private memory register to the A-register
	and IFAST/L, set flip-flops MRQ and DRQ	S/MRQ = (S/MRQ) (S/MRQ) = (S/MRQ/2) + ... (S/MRQ/2) = IFAST/L IOPH1 SW13 (NIOEN + ...) + ... R/MRQ = ...	Prepare to request next stack word from core memory in PH6
		S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = (S/MRQ/2) + ... R/DRQ = ...	Inhibits transmission of another clock until data release signal is received from core memory
	and FAMDST, enable signal (S/SXA)	(S/SXA) = FAMDST PREIO/1 + ... PREIO/1 = IOPH1 SW13 IOEN NPCP2	Preset adder for A⟶S in IOEN IOIN NIOPH1

(Continued)

Table 3-115. I/O Restoration Phase Sequence (Cont.)

Phase	Function Performed	Signals Involved	Comments
IOBO + NIOEN (Cont.)	Set flip-flop PH6	S/PH6 = BRPH6 NIOEN NCLEAR + ... BRPH6 = IOPH1 SW13 (S/PH6/IO) + ... (S/PH6/IO) = IOPH1 SW13 NIPH10 NPCP2 (NIOEN + ...) R/PH6 = ...	Exit from I/O phases and return to execution phase PH6
	Set flip-flops MRQ and DRQ	S/MRQ = (S/MRQ) (S/MRQ) = (S/MRQ/2) + ... (S/MRQ/2) = IOPH1 SW13 IPH10 NPCP2 (NIOEN + ...) + ... R/MRQ = ...	Prepare to request next instruction from core memory in PH10 Inhibits transmission of another clock until data release signal is received from core memory
		S/DRQ = (S/DRQ) NCLEAR (S/DRQ) = (S/MRQ/2) + ... R/DRQ = ...	
	Set flip-flop PH10	S/PH10 = BRPH10 NCLEAR + ... BRPH10 = IOPH1 SW13 IPH10 NPCP2 (NIOEN + ...) + ... R/PH10 = ...	Exit from I/O phases and return to execution phase PH10
	If IOBO, set flip-flop NIOB0	S/NIOB0 = IOPH1 SW13 + RESET/A	Erase I/O abort condition

Table 3-116. I/O Abort Phase Sequence

Phase	Function Performed	Signals Involved	Comments
IOEN NIOIN NIOPH1 NIOB0 T5L	One clock long		
	Enable signal IOENNIN	IOENNIN = IOEN NIOIN NIOPH1	Indicates I/O disable condition
	Reset flip-flop NIOB0	R/NIOB0 = (S/IOB0) (S/IOB0) = IOENNIN NIOINH AVO S/NIOB0 = IOPH1 SW13 + RESET/A	Stores I/O abort condition at clock following AVO. Signal AVO supplied by the device controller system following a new service call, specifies an unusual condition
IOEN NIOIN NIOPH1 IOB0 T5L	One clock long		
	Reset flip-flop NIOPH1	R/NIOPH1 = (S/IOPH1) (S/IOPH1) = IOENNIN IOB0 + ... S/NIOPH1 = (R/IOPH1) + RESET/A	Return to IOPH1 SW13 (See table 3-115.)
	Enable signal BRSW13	BRSW13 = AVO IOB0 IOENNIN + ...	

Table 3-117. Summary of I/O Phase Sequences

PHASE AND CLOCK	GENERAL ACTIVITIES (APPLICABLE TO ALL SERVICE CYCLES)	SPECIAL ACTIVITIES			
		Order Out and Data Chaining	Order In	Data Out	Data In
IOEN IOIN NIOPH1 T8L or T5L	I/O Setup S/IOPH0 S/SW8 R/IOEN R/IOFS (S/AXRR/2) S/IOFM DC/D Address To IOFR FR→IOFR IFAMDS⇒A→S S→RW(R) IFAST/L⇒A→S S→C				
IOPH0 SW8 T5L	I/O Setup RR→A [00] R/PEM R/IOSC S/RSCLEN IFAST/L⇒B→S or IFAST/S S→P P→B IOPH10⇒(S/SXA) BRSW10 S/IOFM S/IOFR9				
IOPH0 SW9 T8L (RS)	I/O Setup PCP2 or IFAMDS or IFAST/L or IFAST/S ⇒ B→S S→RW (00) CLEN				

(Continued)

Table 3-117. Summary of I/O Phase Sequences (Cont.)

(Continued)

PHASE AND CLOCK	GENERAL ACTIVITIES (APPLICABLE TO ALL SERVICE CYCLES)	SPECIAL ACTIVITIES			
		Order Out and Data Chaining	Order In	Data Out	Data In
IOPH0 SW10 T5L	I/O Setup A ⟶ S S ⟶/⟶ B P ⟶ B IOPH10 ⟹ RR ⟶/⟶ A [01] (S/SXA) BRSW12 CLEN NIOPH10 ⟹ (S/SXC) S/IOFM S/IOFR9				
IOPH0 SW11 T8L	I/O Setup C ⟶ S S ⟶/⟶ RW [01] (S/SXA)				
IOPH0 SW12 T5L	I/O Setup A ⟶ S S ⟶/⟶ D A8 ⟹ S/P32 A9 ⟹ S/P33 DOR ⟹ S/SW1 IOR ⟹ S/SW2	(S/AXRR/4) S/IOFM S/IOFR8 S/IOPH3 R/IOPH0 BRSW8		S/MRQ S/DRQ S/IOPH2 R/IOPH0	(S/SXDM1)
IOPH2 SW13 DR				MB ⟶ C P32 ⟶/⟶ BC0 P33 ⟶/⟶ BC0 (S/SXC)	
IOPH2 SW14 T5L				C ⟶ S S ⟶/⟶ A DECR ⟶/⟶ (BC0-BC1) BCZ ⟹ S/IOPH0 R/IOPH2 BRSW13 (S/SXDM1)	

Table 3-117. Summary of I/O Phase Sequences (Cont.)

PHASE AND CLOCK	GENERAL ACTIVITIES (APPLICABLE TO ALL SERVICE CYCLES)	SPECIAL ACTIVITIES			
		Order Out and Data Chaining	Order In	Data Out	Data In
IOPH0 SW15 T5L		S/SW6 (ED) R/SW7 (NES) VORDER ⇒ (S/SXA) R/IOPH0 S/IOPH1 BRSW8	A0 ⇒ S/B9 A1 ⇒ S/B8 (S/B8) (S/B9) D4 ND6 ⇒ S/B14 S/SW3 S/SW6 (ED) R/SW7 (NES) S/IOPH1 R/IOPH0 BRSW8		NB0 NSW4 ⇒ C → S → A NP32 P33 ⇒ AXAR8 P32 NP33 ⇒ AXAR16 P32 P33 ⇒ AXAR24 NSW4 ⇒ (S/SXA) ND7 ⇒ (S/MBXS), S/MRQ, S/DRQ R/IOPH0 S/IOPH1 BRSW8
IOPH1 SW8	I/O Restoration ⇒ (S/AXRR/3), S/IOFM, S/IOFR9	T8L ED NES S/RSA VORDER ⇒ A → S → P Read Backward ⇒ S/B0 IOP Control Error ⇒ S/B13 IOP Memory Error ⇒ S/B12 IOP Halt ⇒ S/B14 ADNH ⇒ S/B11 S/RSCLEN	T8L ED NES S/RSA S/RSCLEN	T8L 1 → P S/SW6 S/SW7 ADNH ⇒ S/B11 PEM + SW4 ⇒ S/B14 SW0 D1 ⇒ S/B2 IODC DASW4 ⇒ S/SW4, R/IOPH1, S/IOPH3, BRSW8, S/AXRR/4), S/IOFM, S/IOFR8	DR A → S → MB NB0 ⇒ 1 → P B0 ⇒ -1 → P S/SW6 S/SW7 ADNH ⇒ S/B11 DAP ⇒ S/B9 PEM + DAP D4 ⇒ S/B14 SW0 D1 DASW4 ⇒ S/B2 IODC DASW4 ⇒ S/SW4, R/IOPH1, S/IOPH3, BRSW8, (S/AXRR/4), S/IOFM, S/IOFR8

(Continued)

Table 3-117. Summary of I/O Phase Sequences (Cont.)

PHASE AND CLOCK	GENERAL ACTIVITIES (APPLICABLE TO ALL SERVICE CYCLES)	SPECIAL ACTIVITIES			
		Order Out and Data Chaining	Order In	Data Out	Data In
IOPH2 SW15 T5L				AXAL8 DECR—/→ (BC0-BC1) NBCZ ⇒ BRSW15 BCZ ⇒ S/IOPH0 R/IOPH2 BRSW13 (S/SXDM1)	
IOPH3 SW8 T5L		RR—/→ A [10] (S/AXRR/4) (S/AXRR/6) S/IOFR8 S/IOFR9 S/SW3 S/B10			
IOPH3 SW9 T5L		RR——→ A [11] AXAL8 (S/SXAP1)			
IOPH3 SW10 B1 T5L		A + 1——→ S S——→ A R/B1 (S/SXP1) BRSW10			
IOPH3 SW10 NB1 T5L		AXSL1 (S/SXA) SXAP1 ⇒ A + 1——→ S SXA ⇒ A——→ S			
IOPH3 SW11 T5L		A——→ S S——→ P S/MRQ S/MRQP1 AXSR1 (S/SXA)			

(Continued)

Table 3-117. Summary of I/O Phase Sequences (Cont.)

PHASE AND CLOCK	GENERAL ACTIVITIES (APPLICABLE TO ALL SERVICE CYCLES)	SPECIAL ACTIVITIES			
		Order Out and Data Chaining	Order In	Data Out	Data In
IOPH3 SW11 T5L (Cont.)		(S/RW/4) S/IOFM S/IOFR8 R/IOFR9 NSW5 ⟹ (R/PEM)			
IOPH3 SW12 T8L		A ⟶ S S ⟶/ RW [10] S/DRQ (S/RW/3) (S/RW/4) S/RW8 S/RW9 S/IOFM AXAL8 (S/SXA)			
IOPH3 SW13 DR		A ⟶ S ⟶/ RW [11] MB ⟶ C (S/SXC) P + 1 ⟶/ P			
IOPH3 SW14 T5L		C ⟶ S ⟶/ A (S/SXA) IOTRIN ⟹ S/SW5 BRSW10 NIOTRIN ⟹ S/MRQ S/DRQ R/SW5			
IOPH3 SW15 DR		A ⟶/ IODA ⟶ /DA/ A ⟶ S A30 ⟶/ P32 A31 ⟶/ P33 AXSR2 MB ⟶ C S/IOPH0 R/IOPH3 BRSW15			

(Continued)

Table 3-117. Summary of I/O Phase Sequences (Cont.)

PHASE AND CLOCK	GENERAL ACTIVITIES (APPLICABLE TO ALL SERVICE CYCLES)	SPECIAL ACTIVITIES			
		Order Out and Data Chaining	Order In	Data Out	Data In
IOPH0 SW13			T5L /DA/ ⟶ S ⟶/⟶ A BRSW15	T8L (D -1) ⟶ S ⟶/⟶ D S1631Z ⇒ S/SW0 A ⟶/⟶ IODA ⟶ /DA/ (A0-A7) Even ⇒ S/IODAP Odd ⇒ R/IODAP AXAL8 RSCLEN ⇒ 1 ⟶/⟶ (P32-P33) RS ⇒ CLEN	T8L (D -1) ⟶ S ⟶/⟶ D S1631Z ⇒ S/SW0 R/IODAP NRSCLEN, NP32 NP33 ⇒ (S/CXS) RSCLEN, NP32 P33 ⇒ AXAR8 RSCLEN, P32 NP33 ⇒ AXAR16 RSCLEN, P32 P33 ⇒ AXAR24 RSCLEN, NB0 ⇒ 1 ⟶/⟶ (P32-P33) RSCLEN, B0 ⇒ -1 ⟶/⟶ (P32-P33) RS ⇒ CLEN
IOPH0 SW14 T8L				VDATAOUT ⇒ BRSW13 S/RSA S/RSCLEN (S/SXDM1) NVDATAOUT ⇒ BRSW8 R/IOPH0 S/IOPH1	DA ⟶ S ⟶/⟶ A, NP32 NP33 ⇒ S/MBXS/0 NP32 P33 ⇒ S/MBXS/1 P32 NP33 ⇒ S/MBXS/2 P32 P33 ⇒ S/MBXS/3 CXS ⇒ S ⟶ C Parity Checks ⇒ IOPC Parity Fails ⇒ S/IODAP VDATAIN ⇒ S/RSA S/RSCLEN (S/SXDM1) BRSW13 NVDATAIN ⇒ (S/SXC)

(Continued)

Table 3-117. Summary of I/O Phase Sequences (Cont.)

PHASE AND CLOCK	GENERAL ACTIVITIES (APPLICABLE TO ALL SERVICE CYCLES)	SPECIAL ACTIVITIES			
		Order Out and Data Chaining	Order In	Data Out	Data In
IOPH1 SW9 T8L	I/O Restoration and processing of new service call: RR→A [01]; B→S→P; P→B; S/BRP; NIFAM, NRTO9 ⇒ BRSW11 (S/RW/2) S/IOFM (S/SXB); SC ⇒ S/IOSC; IFAM ⇒ (S/AXRR/2) S/IOFM (S/CXS) (S/SXA)	S/SW6 S/SW7 D1 SW0 ⇒ S/IODA0 ND0 SW0 ⇒ S/IODA1 D2 ⇒ S/IODA2 B14 ⇒ S/IODA3 RS ⇒ CLEN	S/SW6 S/SW7 A2 ⇒ S/B1 A3 D3 ⇒ S/B3 A4 D5 ⇒ S/B4 (S/B3) + (S/B4) ⇒ S/IODA0 D2 ⇒ S/IODA2 B14 ⇒ S/IODA3 RS ⇒ CLEN	RTO9 ⇒ S/SW3 S/SW6 R/SW7 PEM ⇒ S/B14 NRTO9 ⇒ S/RSA S/RSCLEN ED ES	RTO9 ⇒ S/SW3 S/SW6 R/SW7 PEM ⇒ S/B14 NRTO9 ⇒ S/RSA S/RSCLEN ED ES
IOPH1 SW10 T5L	I/O Restoration and processing of new service call: A→S→C; RR→A [00]; IOSC ⇒ S/IOFS; (S/RW/2); (S/SXB); S/IOFM			S/RSA S/RSCLEN ED NES	S/RSA S/RSCLEN ED NES
IOPH1 SW11 T8L	I/O Restoration and processing of new service call: B→S→RW [00]; 0→(D8-D15); IOSC, NIOFS ⇒ S/IOFS; IOSC ⇒ S/IOEN; IFAM ⇒ (S/SXA); NIFAM, NSW3 ⇒ BRSW13; FAMDS ⇒ (S/AXRR); IFAST ⇒ (S/AXRR) S/IOFM S/IOFR9			S/SW7 TODATA SW0 D1 ⇒ S/IODA0 SW0 ND0 ⇒ S/IODA1 D2 ⇒ S/IODA2 B14 ⇒ S/IODA3	S/SW7 TODATA SW0 D1 ⇒ S/IODA0 SW0 ND0 ⇒ S/IODA1 D2 ⇒ S/IODA2 B14 ⇒ S/IODA3

(Continued)

Table 3-117. Summary of I/O Phase Sequences (Cont.)

PHASE AND CLOCK	GENERAL ACTIVITIES (APPLICABLE TO ALL SERVICE CYCLES)	SPECIAL ACTIVITIES			
		Order Out and Data Chaining	Order In	Data Out	Data In
IOPH1 SW12 T5L	I/O Restoration and processing of new service call	ED ES S/RSA S/RSACLEN NRS ⇒CLEN	ED ES S/RSA S/RSACLEN NRS ⇒CLEN	ED ES S/RSA S/RSACLEN NRS ⇒CLEN	ED ES S/RSA S/RSACLEN NRS ⇒CLEN
IOPH1 SW13 NIOB0 T8L	I/O Restoration and processing of new service call				
IOB0 + NIOEN					

(Continued)

IFAST ⇒ A → S ⁄→ P; P ⁄→ B; RR ⁄→ A[01]

IFAMDS ⇒ A → S ⁄→ B; RR ⁄→ A[R]

S/SXD)

(S/RW/3)

S/IOFM

S/IOFR9

D → S ⁄→ RW [01]

P32 → S8

P33 → S9

C ⁄→ D

IOEN FSL ⇒ IOIN

R/IOPH1

IOEN FSL ⇒ IOIN

NIOEN + NFSL ⇒ R/IOIN

NIOEN, FAST/L, OU0 ⇒ (P + 1) ⁄→ P; (R + 1) ⁄→ R

NIOEN, FAST/L, NOU0 ⇒ (P - 1) ⁄→ P; (R - 1) ⁄→ R

NIOEN, FAST/S ⇒ (P - 1) ⁄→ P; (R - 1) ⁄→ R (S/AXRR)

IOEN, NPCP2, FAMDST ⇒ (S/SXA)

Table 3-117. Summary of I/O Phase Sequences (Cont.)

PHASE AND CLOCK	GENERAL ACTIVITIES (APPLICABLE TO ALL SERVICE CYCLES)	SPECIAL ACTIVITIES			
		Order Out and Data Chaining	Order In	Data Out	Data In
NIOB0 + IOEN (Cont.)	I/O Restoration and processing of new service call NIOEN, IFAST/L ⇒ S/MRQ, S/DRQ NIOEN, NIPH10, NPCP2 ⇒ S/PH6 NIOEN, IPH10 ⇒ S/MRQ, S/DRQ, S/PH10 IOB0 ⇒ R/IOB0				

3-138 POWER DISTRIBUTION

The following units supply and distribute power in the Sigma 5 CPU, memory, and peripheral equipment:

a. Main power distribution box

b. Power junction box

c. PT14 converter power supply

d. PT15 inverter power supply

e. PT16 logic power supply

f. PT17 memory power supply

g. PT18 interface power supply

3-139 Main Power Distribution Box

Primary power is supplied to the Sigma 5 system through the main power distribution box in the CPU. The power distribution box, as shown in figure 3-234, contains a LOCAL-OFF-REMOTE switch, five connectors, a contactor, terminal board, and power monitor assembly. Details of the power monitor assembly are presented under the description of the power fail-safe feature in this section.

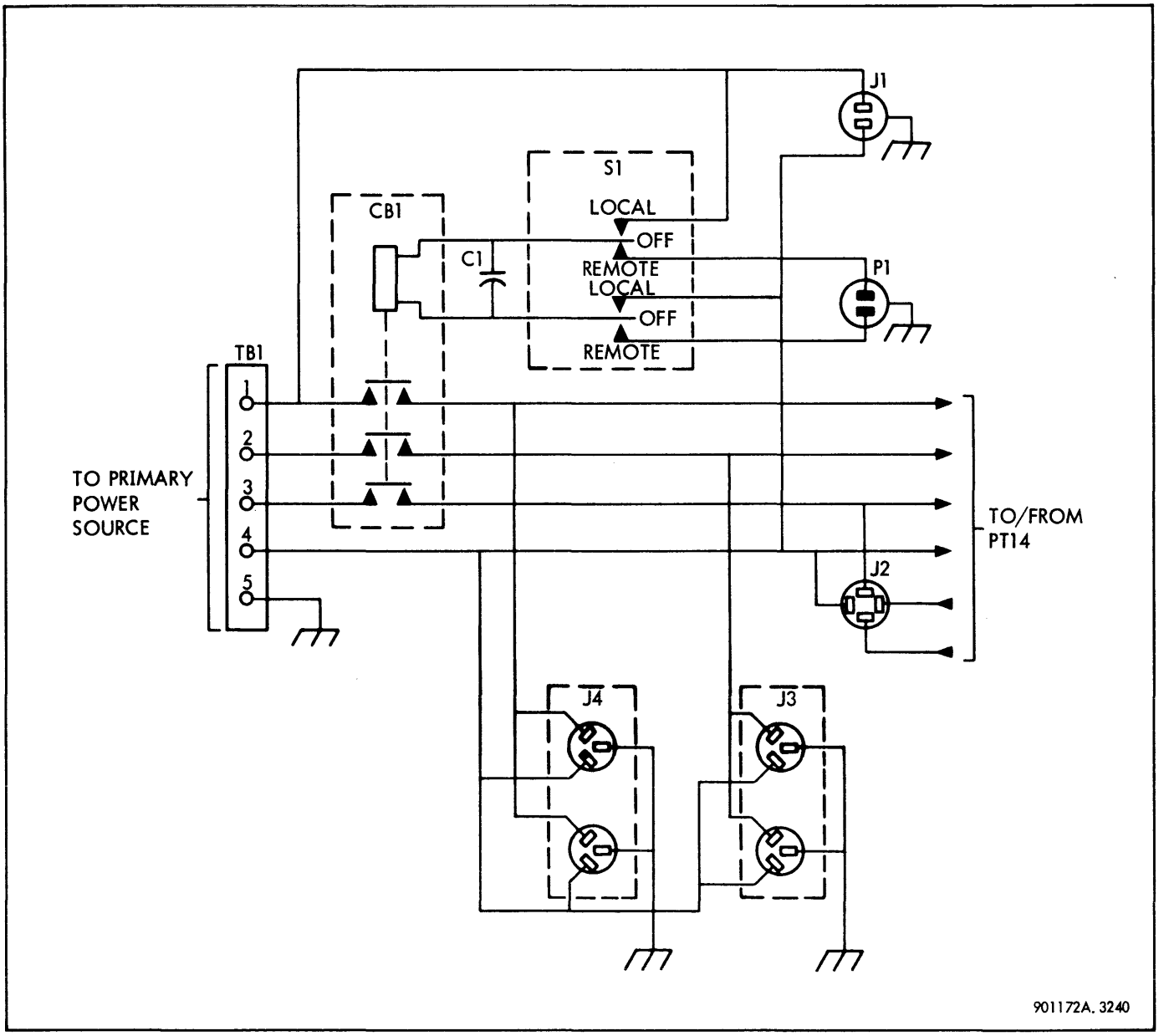


Figure 3-234. Main Power Distribution Box, Schematic Diagram

The LOCAL-OFF-REMOTE switch (S1, figure 3-234), allows the operator to select whether power shall be turned on and off at the PCP or at some unit of peripheral equipment. The solenoid-actuated contactor (CB1) is controlled through switch S1; this contactor connects the primary power source with the PT14 power supply and the five connectors in the main power distribution box. Duplex connectors J3 and J4 supply power at 120v/60 Hz to unit ventilating fans and to the power junction box. Connectors J1 and P1 connect local and remote power controls, respectively, to S1. The local power control is on the processor control panel; the remote power controls are on the peripheral equipment. Connector J2 supplies source power to the power monitor assembly. Terminal board TB1 serves as a connection point for primary power input.

3-140 Power Junction Box

The power junction box provides four 120v/60 Hz connectors and six 120v/2 kHz connectors. The 60 Hz power is derived from the main power distribution box; the 2 kHz power is obtained from the PT15 power supply.

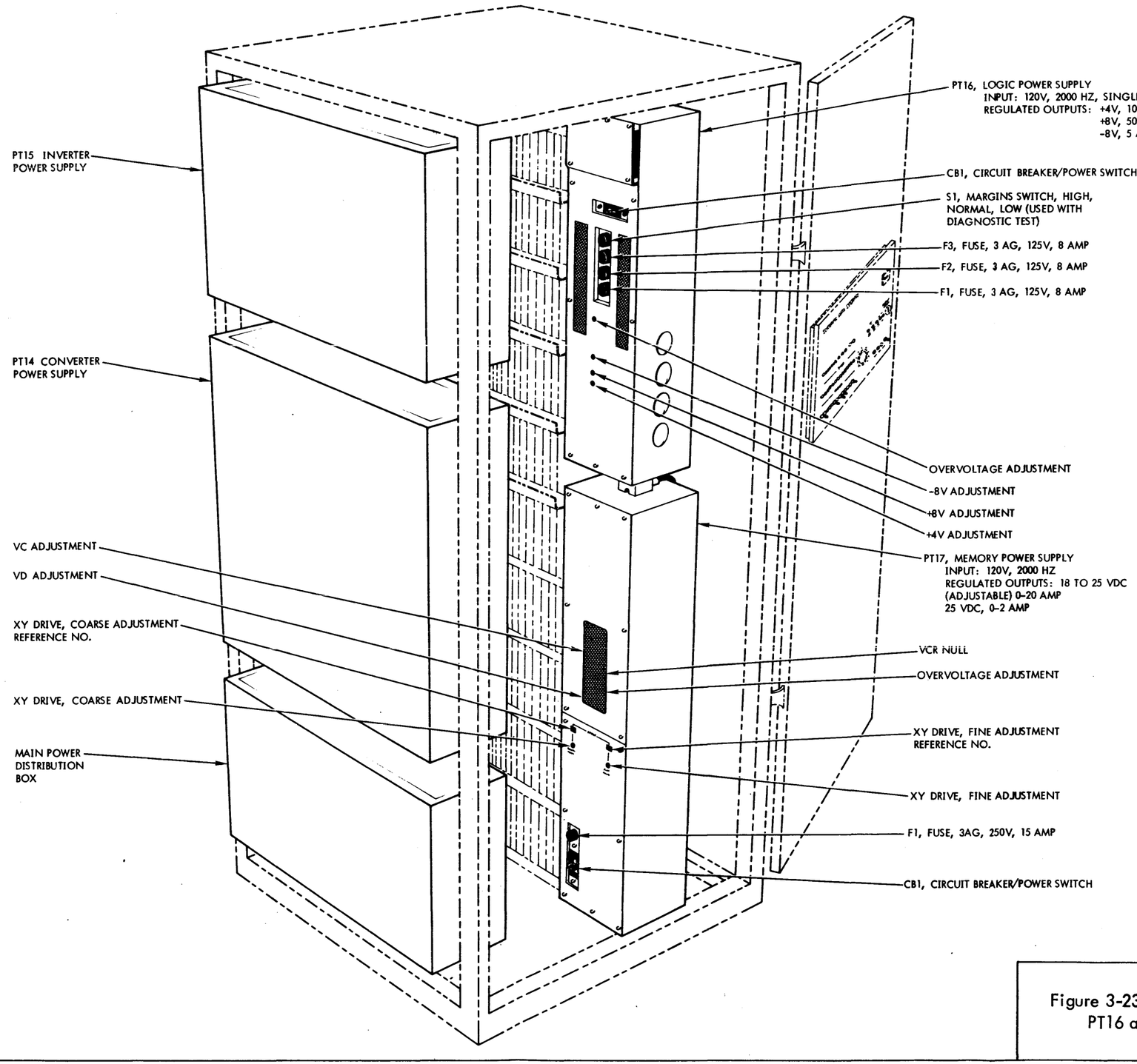
3-141 Power Supplies

The PT14/PT15 power supply combination changes the 60 Hz source power to 2 kHz, which serves as input to PT16, PT17, and PT18. Detailed descriptions and theories of operation for the PT-series power supplies are covered in associated technical manuals applicable to each power supply.

A power frequency of 2 kHz is used so that the individual low-voltage high-current power supplies may be small enough to be mounted on each frame. As a result, short, direct connections to the high-current loads are used.

The physical and electrical configuration of power supplies and the locations of voltage terminals are presented in a series of illustrations. Figure 3-235 emphasizes PT16 and PT17 details and shows the mounting of these power supplies relative to the PT14 and PT15 power supplies and main power distribution box. Figure 3-236 shows PT14 and PT15 power supplies, main power distribution box, and power junction box physical details. Figure 3-237 shows the CPU and memory backwiring, with terminals that provide ground, +4v, +8v, -8v, +21.5v, +24v, and +10.25v. Refer to table 3-118 for jacks and pins on which the voltages appear.

Power interconnection varies with the Sigma 5 configuration and peripheral equipment used. Therefore, typical power connections are shown in figure 3-238. One Sigma 5 cabinet and an accessory cabinet for optional equipment are shown as examples. More than one power junction box may be used when a greater number of outlets are required.



LEGEND

VC VOLTAGE, CONSTANT

VCR VOLTAGE, DIODE (CR) (USED TO NULL AN AMPLIFIER CONTROLLED BY TEMPERATURE SENSING DIODES)

VD VOLTAGE, DRIVE

Figure 3-235. Physical Details of Sigma 5 PT16 and PT17 Power Supplies

900630A. 359

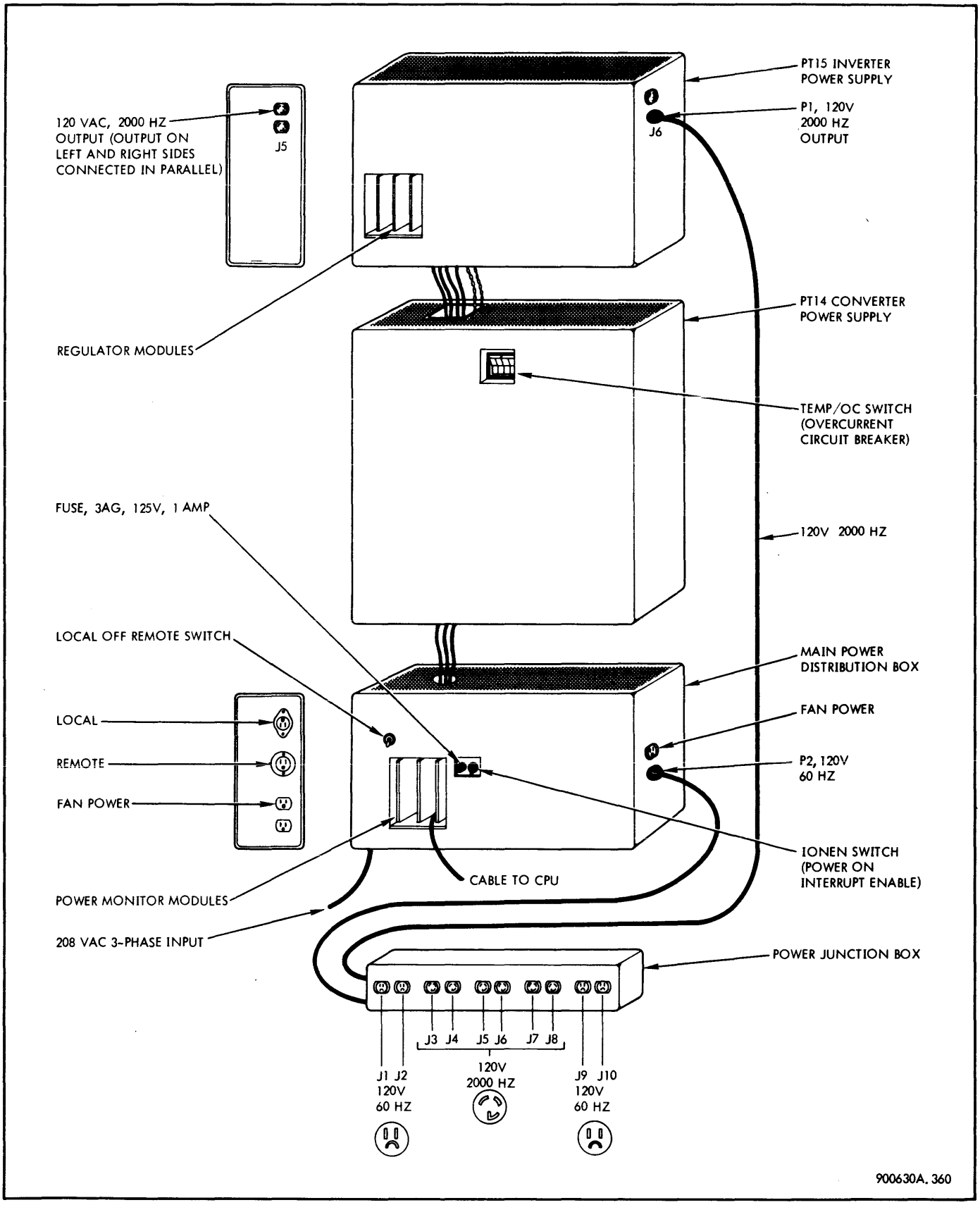


Figure 3-236. Physical Details of PT14 and PT15 Power Supplies, Main Power Distribution Box, and Power Junction Box

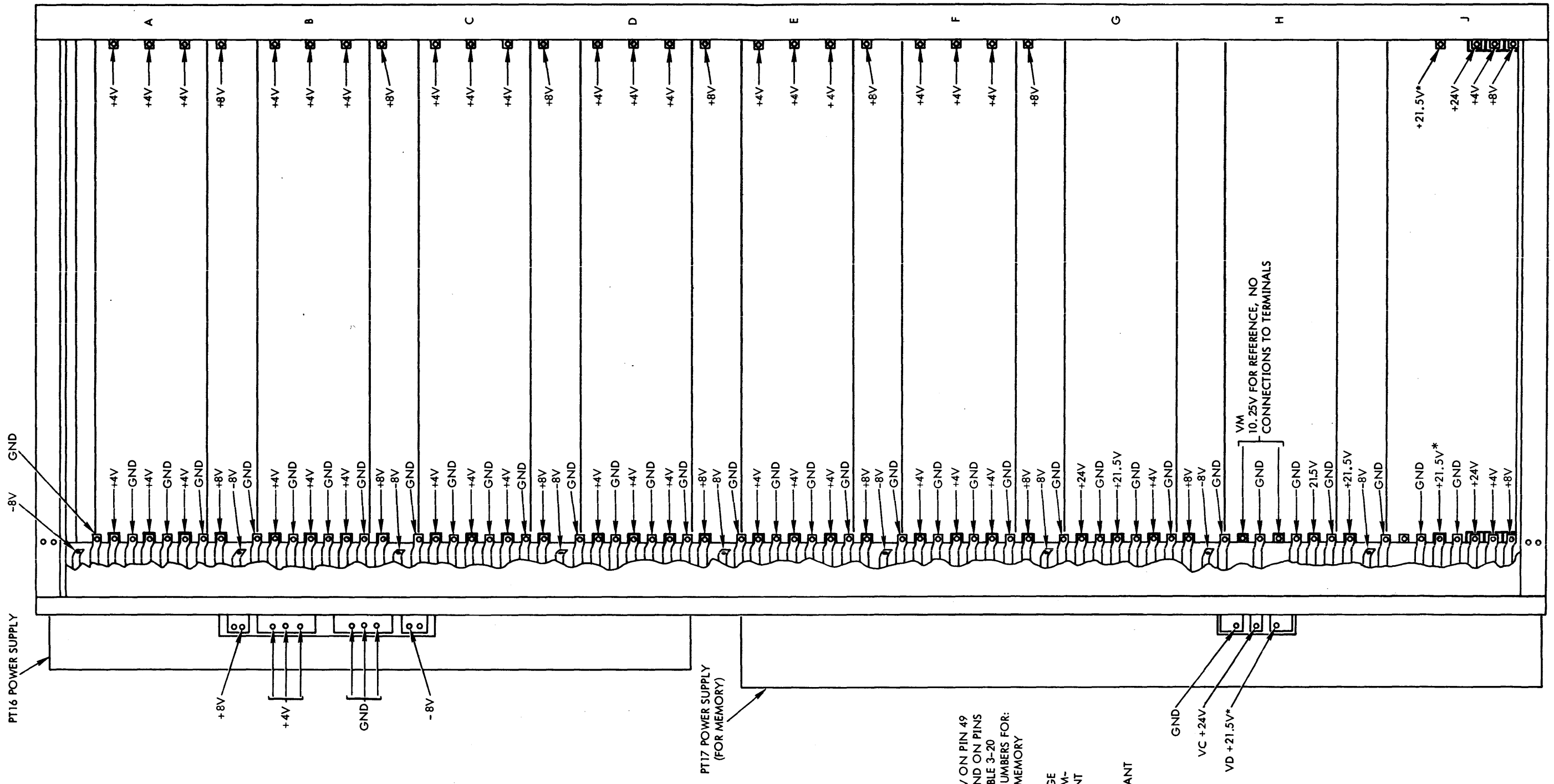


Figure 3-237. Voltage Terminals on Backwiring Boards and PT16 and PT17 Power Supplies

900630A. 361

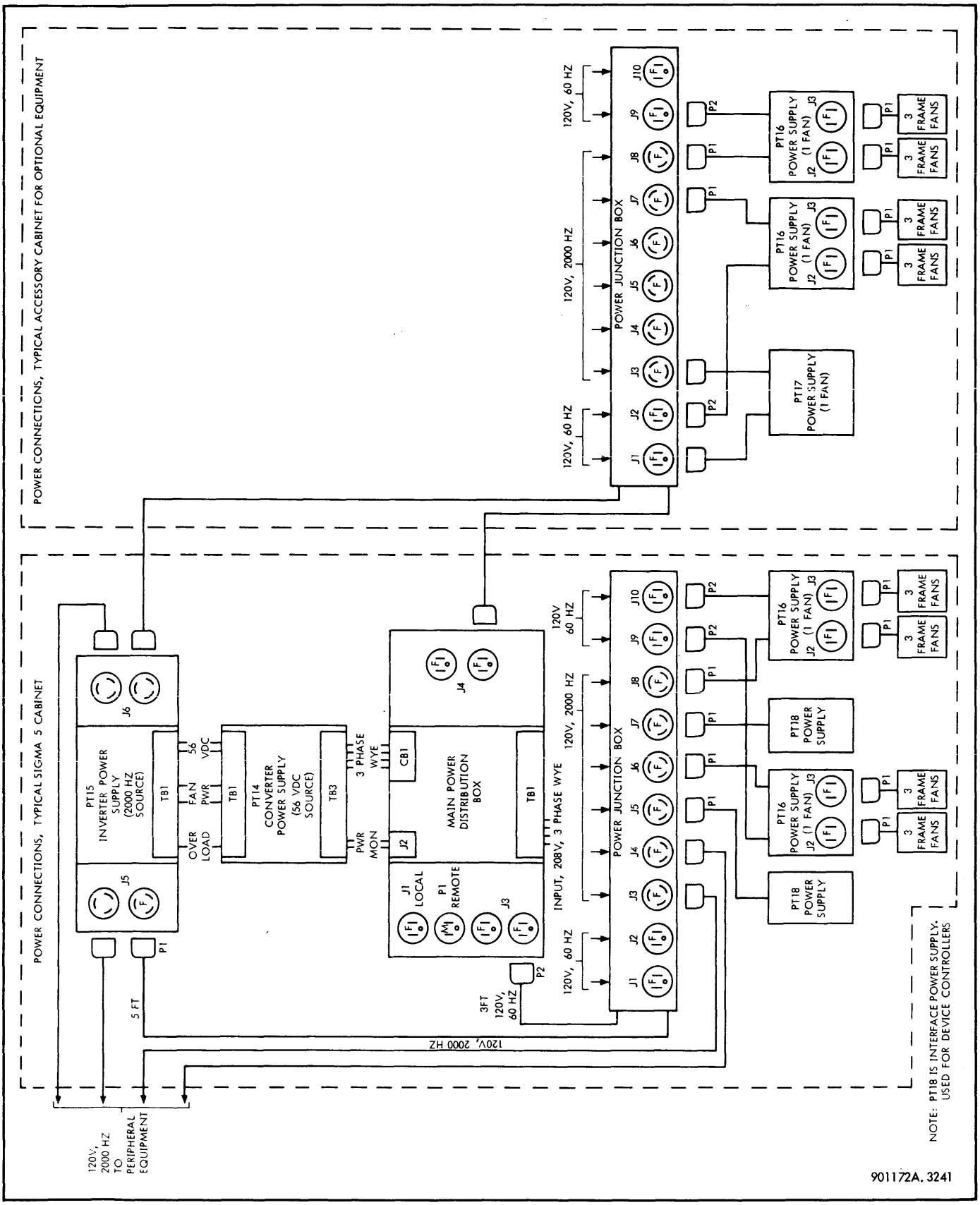


Figure 3-238. Typical Power Distribution Diagram

Table 3-118. Voltages on Pins and Jacks in Backwiring

Row	Ground	+4v	+8v	-8v	vc +24v	vd +21.5v	vm +10.25v
A	Pins 0,16,32, 48; all jacks	Pin 49; all jacks	Pin 51; all jacks	Pins 50; jacks 1, 2, 3,4,28,30,32			
B	Pins 0,16,32, 48; all jacks	Pin 49; all jacks	Pin 51; all jacks	Pin 50; jacks 1,3, 5,6,7,8,30,32			
C	Pins 0,16,32, 48; all jacks	Pin 49; all jacks	Pin 51; all jacks	Pin 50; jacks 1,3, 5,7,11,25,27			
D	Pins 0,16,32, 48; all jacks	Pin 49; all jacks	Pin 51; all jacks	Pin 4, jack 32			
E	Pins 0,16,32, 48; all jacks	Pin 49; all jacks	Pin 51; all jacks	Pin 50; jacks 2, 3, 22			
F	Pins 0,16,32, 48; all jacks	Pin 49; all jacks	Pin 51; all jacks	Pin 50; jacks 3 thru 12, 27 thru 32			
G	Pins 0,16,32, 48; jacks 1, 2,11,16 Pins 0,16,48; jacks 3 thru 8, 12,13,14 Pins 0,20,48; jacks 9,10 Pins 16,32; jack 15	Pin 49; jacks 3 thru 8, 11 thru 16	Pin 51; jacks 3 thru 8, 12 thru 16		Pin 1, jack 15	Pin 31; jacks 9,10 Pin 20; jack 15	Pin 21; jacks 9, 10
H	Pins 3, 20; jacks 1 thru 16					Pin 51; jacks 1 thru 16	Pin 21; jacks 1 thru 16
J	Pins 0,16,32, 48; jacks 1,4, 7,10,13,16 thru 32 Pins 0,20,48; jacks 2,3,5,6, 8,9,11,12,14, 15						

SECTION IV
MAINTENANCE AND PARTS LIST

4-1 MAINTENANCE

Maintenance requirements for the Sigma 5 computer depend upon the selection of optional features and device controllers included in the CPU and core memory. Reference documents for basic and optional features are identified in this section. For maintenance of an item of peripheral equipment and its controller, refer to the appropriate technical and programming manuals.

4-2 SPECIAL TOOLS AND TEST EQUIPMENT

Special tools and test equipment recommended for repair or maintenance of the Sigma 5 computer are listed in table 4-1.

4-3 PREVENTIVE MAINTENANCE

Preventive maintenance of the Sigma 5 computer consists of scheduled diagnostic testing in addition to visual inspection and routine maintenance. Because there are no mechanical devices in the Sigma 5, lubrication and mechanical adjustments are not required.

External surfaces of the Sigma 5 computer cabinets must be kept clean and free of dust. Doors and panels must close completely and be in reasonable alignment. Tops of cabinets must be cleared of all materials so that fan assemblies are able to expel the air taken in at the bottom of the cabinets.

The interior of cabinets must be free of wire cuttings, dust, and other foreign matter. No clip leads or push-on jumpers should be in use during normal operation, and all cables must be neatly dressed by sufficient clamps or routing. All chassis and frames must be properly bolted down, with all hardware in place.

The air filters (SDS part number 117427) should be checked for cleanliness periodically. They may be washed with water and detergent, and reinstalled.

Note

Do not spray the Sigma series filters with adhesive fluid, since it inhibits air flow.

Table 4-1. Special Tools and Test Equipment

Name	Manufacturer's Part No.	Manufacturer
P6010 IBM accessories and probe package	010-0186-00	Tektronix Beaverton, Oregon
Oscilloscope	453	
Wirewrap tool	14XA2	Gardner-Denver Grand Haven, Mich.
Wirewrap bit	502128	
Wirewrap sleeve	502129	
Wire unwrap tool	505084(LH)	
Module extractor	126668	SDS
Extender Module ZT10	117037	
Solder sucker	(None)	
Device controller simulator JK58	124300	

4-4 DIAGNOSTIC TESTING

Diagnostic test procedures for features of the Sigma 5 computer are described in the documents listed in table 4-2. The diagnostic test programs should be run at intervals not longer than those indicated in table 4-3. Diagnostic test procedures should be run with power supplies at normal, +10 percent level, and -10 percent level.

4-5 ELECTRONIC TESTING

For two- or three-port memories, make the following electrical performance monitoring measurements each time the MEDIC 75 diagnostic program is run.

a. Remove AT11 modules from 6C, 4D, and 8D to present continuous memory requests from all ports.

b. Place address switches for each port to a starting address of zero (see table 4-11).

c. Ground pins 8D-36 (signal ORAB), 4D-36 (signal ORAC), 6C-36 (signal ORBC), and 3D-35 (signal MR).

d. Check that memory cycles (period approximately 860 nsec) are initiated at the following pins by signal CFA.

Pin	Signal	Duration
29C-7	CFA	True approximately 350 nsec
29C-21	CFB	True approximately 60 nsec

e. Ground pin 29C-7 (signal CFA).

f. Check that signal CFB initiates memory accesses approximately every 860 nsec.

g. Ground pin 29C-21 (signal CFB).

h. Check that signal APA causes memory cycles with a period of 1.1 μsec (1100 nsec) at the following pins.

Pin	Signal	Duration
27D-18	APA	True for approximately 250 nsec
27D-2	APB	True for less than 60 nsec

i. Ground pin 27D-18 (signal APA).

j. Check that signal APB causes memory cycles approximately every 1100 nsec, and is true for approximately 250 nsec.

k. Ground pin 27D-2 (signal APB).

l. Check that there are no memory cycles, and that signal MI (pin 28D-2) is false.

m. Remove all grounds attached in steps c, e, g, i, and k, except the one on pin 3D-35 (signal MR).

n. Check that continuous memory cycles are generated from port C (indicating that signals MQA and MQB are locked out), that signal ADC (pin 29D-15) is true, and that MI (pin 28D-2) is true approximately every 860 nsec.

o. Ground pin 8D-36 (signal ORAB).

p. Check that port B initiates memory cycles and that signal CFB (pin 29C-21) is true for approximately 350 nsec.

q. Ground pin 4D-36 (signal ORAC).

r. Check that port A causes memory cycles and that signal CFA (pin 29C-7) is true for approximately 350 nsec.

s. Remove all grounds, and restore address switches to original value.

4-6 SWITCH SETTINGS

Modules ST14 and LT26, included in features of the Sigma 5 computer, require specific settings of switches to enable proper operation of the computer, as summarized in table 4-4. The reference designations for switches on these modules are indicated in figure 4-1 and figure 4-2.

Primary sources for switch position data are listed in table 4-4. Table 4-5 summarizes functions of switches associated with the memory. Tables 4-6 through 4-17 locate modules ST14 and LT26 as specified in module location charts for each feature. Basic and optional features of the Sigma 5 computer are normally assigned locations in accordance with the Sigma 5 System Installation Drawing (137112). However, locations in a specific installation should be verified by consulting documents included with the equipment.

Switches associated with the floating point feature permit display of data stored in the floating point registers on the CPU DISPLAY indicators when the REGISTER SELECT switch is in the EXT position. The type of data displayed is described in detail in Section III of this manual and is summarized in table 4-18.

Table 4-2. Diagnostic Programming Manuals

Publication Number	Publication Title
900712	Sigma 5 and 7 Diagnostic Control Program
900825	Sigma 5 and 7 Memory (≥8K) Test (MEDIC 75)
900870	Sigma 5 and 7 CPU Diagnostic System (Verify)
900891	Sigma 5 and 7 CPU Diagnostic System (Pattern)
900898	Sigma 5 and 7 CPU Diagnostic System (Float)
900972	Sigma 5 and 7 Relocatable Diagnostic Program Loader
901071	Sigma 5 and 7 Memory Interleaving Test (MIT)
901076	Sigma 5 and 7 Systems Monitor
901126	Sigma 5 and 7 Multiplexor IOP Test
901134	Sigma 5 and 7 Interrupt Test
901135	Sigma 5 and 7 Power Fail-Safe Test
901136	Sigma 5 and 7 Real-Time Clock Test
901158	Sigma 5 and 7 Selector IOP Channel Test
901516	Sigma 5 and 7 CPU Diagnostic Program (Memory Protect)
901519	Sigma 5 CPU Diagnostic System (Suffix)
901523	Sigma 5 CPU Diagnostic System (Auto)

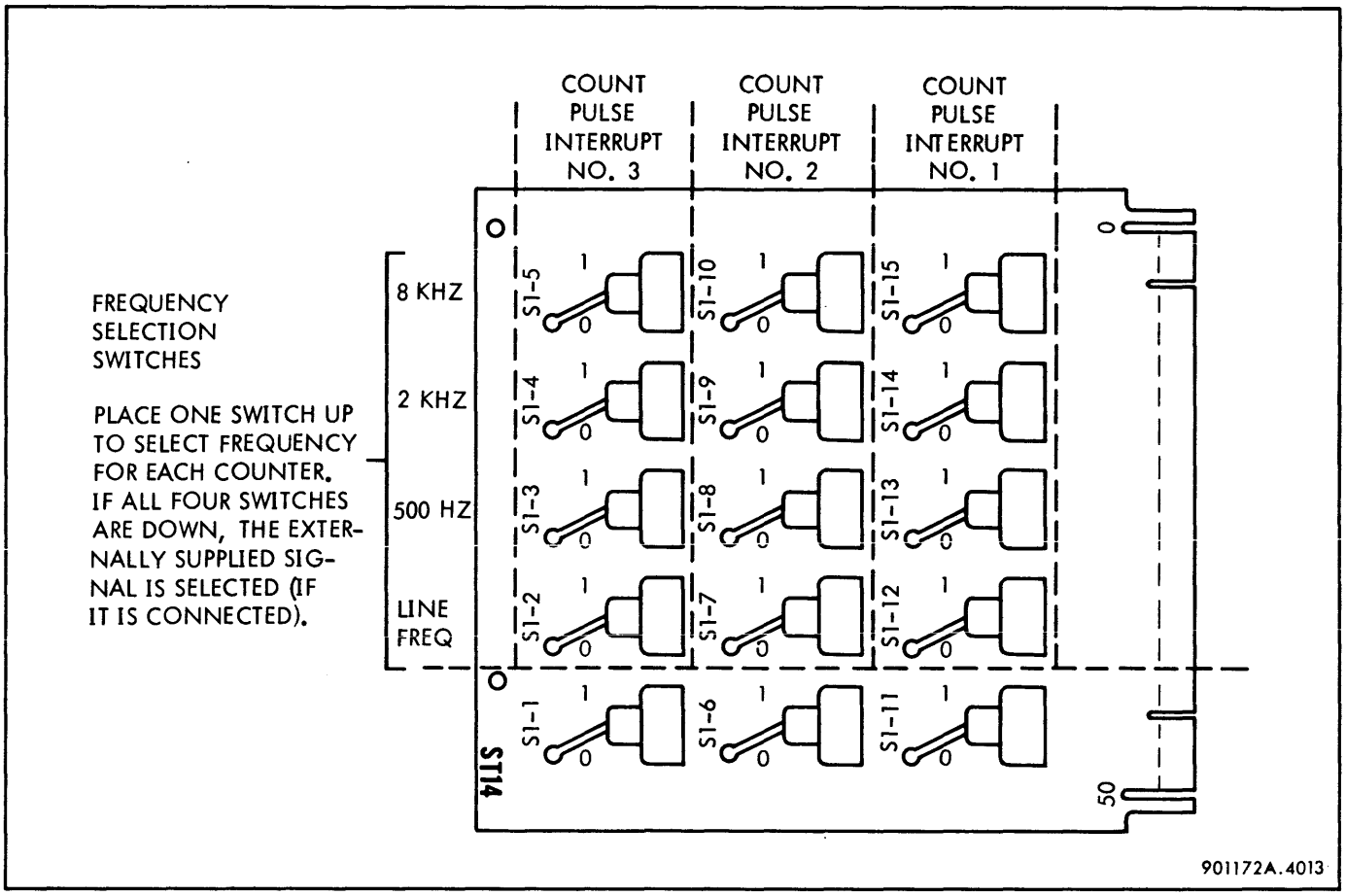


Figure 4-1. Address Selector Module ST14

Table 4-3. Diagnostic Programming Schedule

Feature	Publication	Interval
Central Processing Unit	*	2 weeks
4K Memory	†	2 weeks
Multiplexing IOP	901126	2 weeks
Real-Time Clock	901136	4 weeks
Power Fail-Safe	901135	8 weeks
Memory Protection	901516	2 weeks
Private Memory Register Extension	900891	2 weeks
Floating Point	900898	2 weeks
External Interrupt Chassis	901134	2 weeks
Memory Expansion (8K, 12K, 16K)	900825	2 weeks

Table 4-3. Diagnostic Programming Schedule (Cont.)

Feature	Publication	Interval
Two- to Three-Port Expansion	**	
Three- to Six-Port Expansion	**	
Additional Eight Subchannels	901126	2 weeks
Selector IOP	901158	4 weeks
Additional Selector Channel	901158	4 weeks

* CPU is tested by five programs (described in publications 900870, 900891, 900898, 901519, and 901523) which also test optional features.

† No test for 4K memory; see publication number 900825 for memory of 8K and greater.

** No test for expansion units. Malfunction detected as part of memory test.

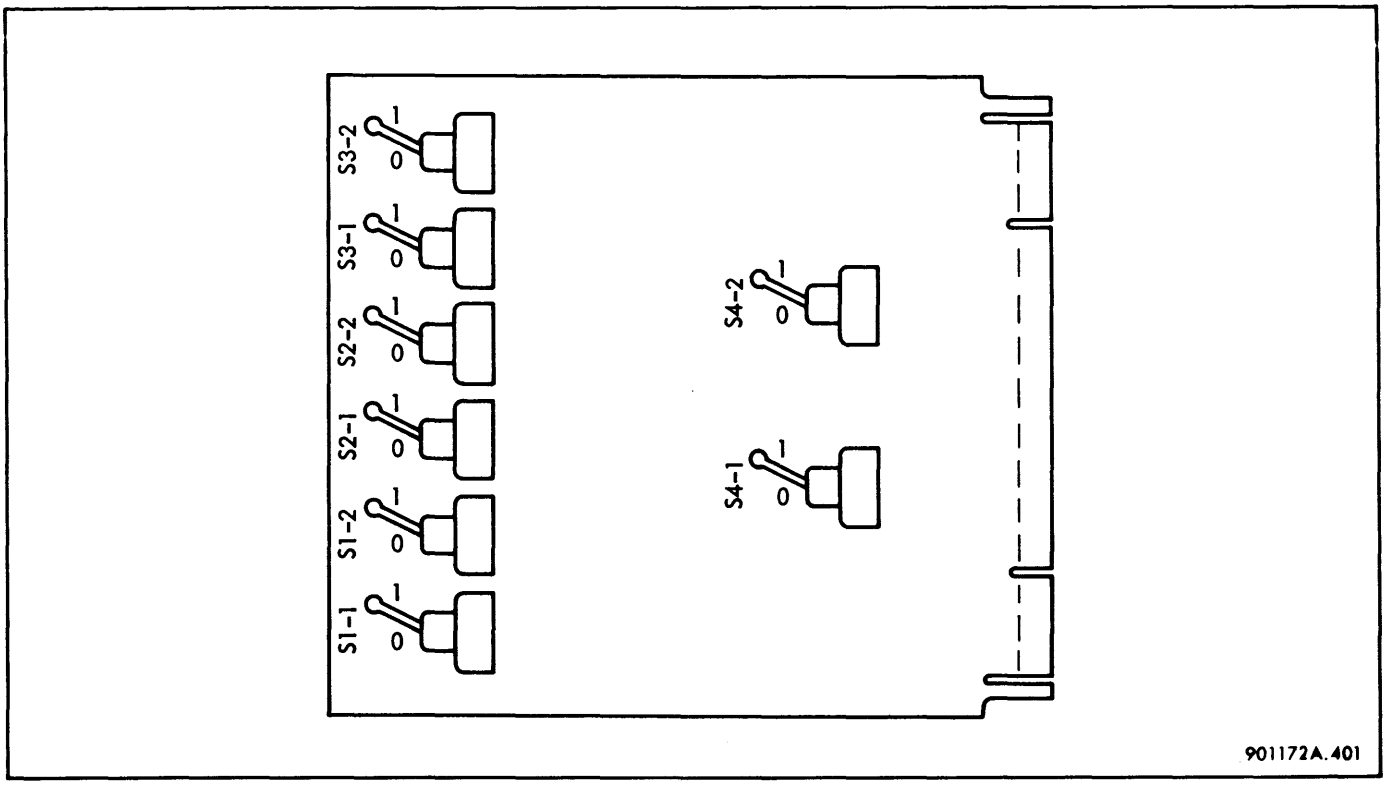


Figure 4-2. Switch Comparator LT26

Table 4-4. Switch Setting Data

Function	MLC	Installation	Module	Table
Real-Time Clock	135273	133279	ST14	4-6
Port Expansion	117652	127409	ST14	4-1
Memory Interleave	117652	127409	ST14	4-8
Memory Fault Number	117652	127409	ST14	4-9
Memory Size	117652	127409	ST14	4-10
MIOP Address	123656	123652	LT26	4-17
Priority Interrupt				
Most Significant Digit	129700	124469	LT26	4-14
Least Significant Digit	129700	124469	LT26	4-12
Register Extension Unit	124819	124816	LT26	4-13
SIOP Address	134000	133995	LT26	4-15
SIOP Bus Share	134000	133995	LT26	4-16
Floating Point	145613	136253	ST14	4-18
Starting Address	117652	127409	ST14	4-11

Table 4-5. Memory Setup Switches in ST14 Modules

Module Location	Function
20C	PORT EXPANSION: S1-1 S1-2 S1-3 S1-4 S1-5 STARTING ADDRESS – PORT C: S1-6 S1-7 S1-8 S1-9 S1-10 STARTING ADDRESS – PORT A: S1-11 S1-12 S1-13 S1-14 S1-15
21C	MEMORY SIZE: S1-1 S1-2 INTERLEAVE SIZE: S1-3 S1-4 S1-5 INTERLEAVE SIZE: S1-6 S1-7 MEMORY NUMBER: S1-8 S1-9 S1-10 STARTING ADDRESS – PORT B: S1-11 S1-12 S1-13 S1-14 S1-15

Table 4-6. Address Selector Module ST14 Switch Settings for Counters (Location 3K)

Counter 1	S1-12	S1-13	S1-14	S1-15
Counter 2	S1-7	S1-8	S1-9	S1-10
Counter 3	S1-2	S1-3	S1-4	S1-5
External Freq	0	0	0	0
500 Hz	1	0	0	0
2000 Hz	X	1	0	0
8000 Hz	X	X	0	1

Notes
1. X denotes that switch setting is irrelevant
2. Input counter 4 always wired to 500 Hz

Table 4-7. Switch Settings for ST14 Modules in Port Expansion (Location 20C)

Condition	Switch
Port A Expanded	S1-2 set to 1
Port A Not Expanded	S1-2 set to 0
Port B Expanded	S1-1 set to 1
Port B Not Expanded	S1-1 set to 0

Table 4-8. Switch Settings for ST14 Modules in Memory Interleave (Location 21C)

Interleave Size	S1-3	S1-4	S1-5	S1-6
None	0	0	0	0
8K	1	0	0	0
16K	0	1	0	0
32K	0	0	1	0
64K	0	0	0	1

Table 4-9. Switch Settings for ST14 Modules Which Determine Memory Fault Number (Location 21C)

Memory Number	S1-8	S1-9	S1-10
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0

Table 4-9. Switch Settings for ST14 Modules Which Determine Memory Fault Number (Location 21C) (Cont.)

Memory Number	S1-8	S1-9	S1-10
5	1	0	1
6	1	1	0
7	1	1	1

Note

Memory fault lights are numbered 1 through 8, so that light number is one greater than switch code.

Table 4-10. Switch Settings for ST14 Modules Which Indicate Memory Size (Location 21C)

Memory Size	S1-1	S1-2
4K	0	0
8K	0	1
12K	1	0
16K	1	1

Table 4-11. Starting Address in ST14 Modules

Port A (20 C)	S11	S12	S13	S14	S15
Port B (20 C)	S6	S7	S8	S9	S10
Port C (21 C)	S11	S12	S13	S14	S15
Memory Size					
0K	0	0	0	0	0
4K	0	0	0	0	1
8K	0	0	0	1	0
12K	0	0	0	1	1
16K	0	0	1	0	0
20K	0	0	1	0	1
24K	0	0	1	1	0
28K	0	0	1	1	1

(Continued)

Table 4-11. Starting Address in ST14 Modules (Cont.)

Port A (20 C)	S11	S12	S13	S14	S15
Port B (20 C)	S6	S7	S8	S9	S10
Port C (21 C)	S11	S12	S13	S14	S15
Memory Size					
32K	0	1	0	0	0
36K	0	1	0	0	1
40K	0	1	0	1	0
44K	0	1	0	1	1
48K	0	1	1	0	0
52K	0	1	1	0	1
56K	0	1	1	1	0
60K	0	1	1	1	1
64K	1	0	0	0	0
68K	1	0	0	0	1
72K	1	0	0	1	0
76K	1	0	0	1	1
80K	1	0	1	0	0
84K	1	0	1	0	1
88K	1	0	1	1	0
92K	1	0	1	1	1
96K	1	1	0	0	0
100K	1	1	0	0	1
104K	1	1	0	1	0
108K	1	1	0	1	1
112K	1	1	1	0	0
116K	1	1	1	0	1
120K	1	1	1	1	0
124K	1	1	1	1	1

Table 4-12. Switch Settings for LT26 Modules in Priority Interrupt (Least Significant Address Digit)

Module Location	Interrupt Level Address*	Switch Module (1J)	Switch Setting†
7J	X0	None	0
	X1	None	0

Table 4-12. Switch Settings for LT26 Modules in Priority Interrupt (Least Significant Address Digit) (Cont.)

Module Location	Interrupt Level Address*	Switch Module (1J)	Switch Setting†
8J	X2	S1-1	1
	X3	S1-2	1
9J	X4	S1-3	1
	X5	S1-4	1
10J	X6	S1-5	1
	X7	S1-6	1
14J	X8	S1-7	1
	X9	S1-8	1
15J	XA	S1-9	1
	XB	S1-10	1
16J	XC	S1-11	1
	XD	S1-12	1
17J	XE	None	0
	XF	S1-13	1

Notes

*X denotes the most significant digit of the address, and is determined by the group select switches in the priority interrupt chassis

†Switches corresponding to vacant module locations must be set to 0

Table 4-13. Switch Settings for LT26 Modules in Register Extension Units (Location 32A)

Register Extension Unit Assembly	S3-2	S3-1	S2-2
4 thru 7	0	0	1
8 thru 11	0	1	0
12 thru 15	0	1	1
16 thru 19	1	0	0
20 thru 23	1	0	1
24 thru 27	1	1	0
28 thru 31	1	1	1

Note

Positions of S2-1, S1-1, S4-1, and S4-2 irrelevant

Table 4-14. Switch Settings for LT26 Module in Location 30J of Priority Interrupt (Most Significant Address Digit)

Required Group No.	Address	S1-1	S2-1	S3-1	S4-1
2	60 - 6F	0	0	1	0
3	70 - 7F	0	0	1	1
4	80 - 8F	0	1	0	0
5	90 - 9F	0	1	0	1
6	A0 - AF	0	1	1	0
7	B0 - BF	0	1	1	1
8	C0 - CF	1	0	0	0
9	D0 - DF	1	0	0	1
A	E0 - EF	1	0	1	0
B	F0 - FF	1	0	1	1
C	100 - 10F	1	1	0	0
D	110 - 11F	1	1	0	1
E	120 - 12F	1	1	1	0
F	130 - 13F	1	1	1	1

Note

Settings of S1-2, S2-2, S3-2, and S4-2 irrelevant

Table 4-15. Switch Settings for LT26 Module in SIOP Unit (Location 8F)

Unit Address	S1-1	S2-1	S3-1	S4-1
0	0	0	0	X
1	0	0	1	X
2	0	1	0	X
3	0	1	1	X
4	1	0	0	X
5	1	0	1	X

Table 4-15. Switch Settings for LT26 Module in SIOP Unit (Location 8F) (Cont.)

Unit Address	S1-1	S2-1	S3-1	S4-1
6	1	1	0	X
7	1	1	1	X

Notes

1. S1-2 and S2-2 used for optional bus share
2. S4-1 must be 1 for last IOP in system, 0 for all others

Table 4-16. Switch Settings for LT26 Module Using Optional Bus Share with SIOP (Location 8F)

SIOP	S1-2	S2-2
First	1	1
Second	0	1

Table 4-17. Switch Settings for LT26 Module in MIOP Unit (Location 13C)

Unit Address	S1-1	S2-1	S3-1	S1-2*
0	0	0	0	X
1	0	0	1	X
2	0	1	0	X
3	0	1	1	X
4	1	0	0	X
5	1	0	1	X
6	1	1	0	X
7	1	1	1	X

*S1-2 must be 0 for the last MIOP in the system and 1 for all others

Table 4-18. Switch Settings for Display of Floating Point Register Information* (Location 6A)

SWITCH SETTINGS†					INFORMATION DISPLAYED
S1-5	S1-4	S1-3	S1-2	S1-1	
0	X	X	X	X	Miscellaneous
1	0	0	0	0	Sum Bus, Lower
1	0	0	0	1	Sum Bus, Upper
1	1	0	0	0	A-Register, Lower
1	1	0	0	1	A-Register, Upper
1	0	1	0	0	B-Register, Lower
1	0	1	0	1	B-Register, Upper
1	0	0	1	0	D-Register, Lower
1	0	0	1	1	D-Register, Upper

*REGISTER SELECT switch on PCP must be set to EXT and REGISTER DISPLAY switch must be set to ON

†X indicates that the switch position is irrelevant

4-7 CORRECTIVE MAINTENANCE

4-8 Wirewrap Techniques

Solderless wirewrap is done with the wirewrap tools listed in table 4-1. For detailed information about solderless wirewrap, see SDS Application Bulletin 64-51-07.

4-9 Power Supplies

Power supplies are installed on the frames of the Sigma 5 computer as described in section I. Reference documents for maintenance of the power supplies are listed in table 4-19.

4-10 PARTS LISTS

The tables and figures in this section list and illustrate replaceable parts of the Sigma 5 computer group, including the accessory cabinet, the central processing unit, and the memory cabinet, with optional equipment listed in typical arrangements.[1]

[1] Although typical arrangements are listed in this section, customer requirements would determine exact arrangements.

4-11 TABULAR LISTINGS (Tables 4-20 through 4-51)

The replaceable parts are arranged in tables of parts lists, starting with the listing of the main assemblies of the equipment, table 4-20. Each main assembly is then broken down into subassemblies or component parts. Breakdown by table continues until all replaceable parts down to a field-replaceable level have been listed and illustrated.

4-12 ILLUSTRATIONS (Figures 4-3 through 4-14)

Each parts list table has an accompanying illustration that indicates the parts described in the table and their locations in the assembly that has been listed.

4-13 PARTS LIST TABLES

Each parts list table is arranged in six columns as follows:

a. Figure and index number of the listed part.

b. Brief description of the part.

c. The reference designator of the part as shown on the schematic diagrams for that part.

d. Manufacturer's code for the part.

e. Manufacturer's part number for the part.

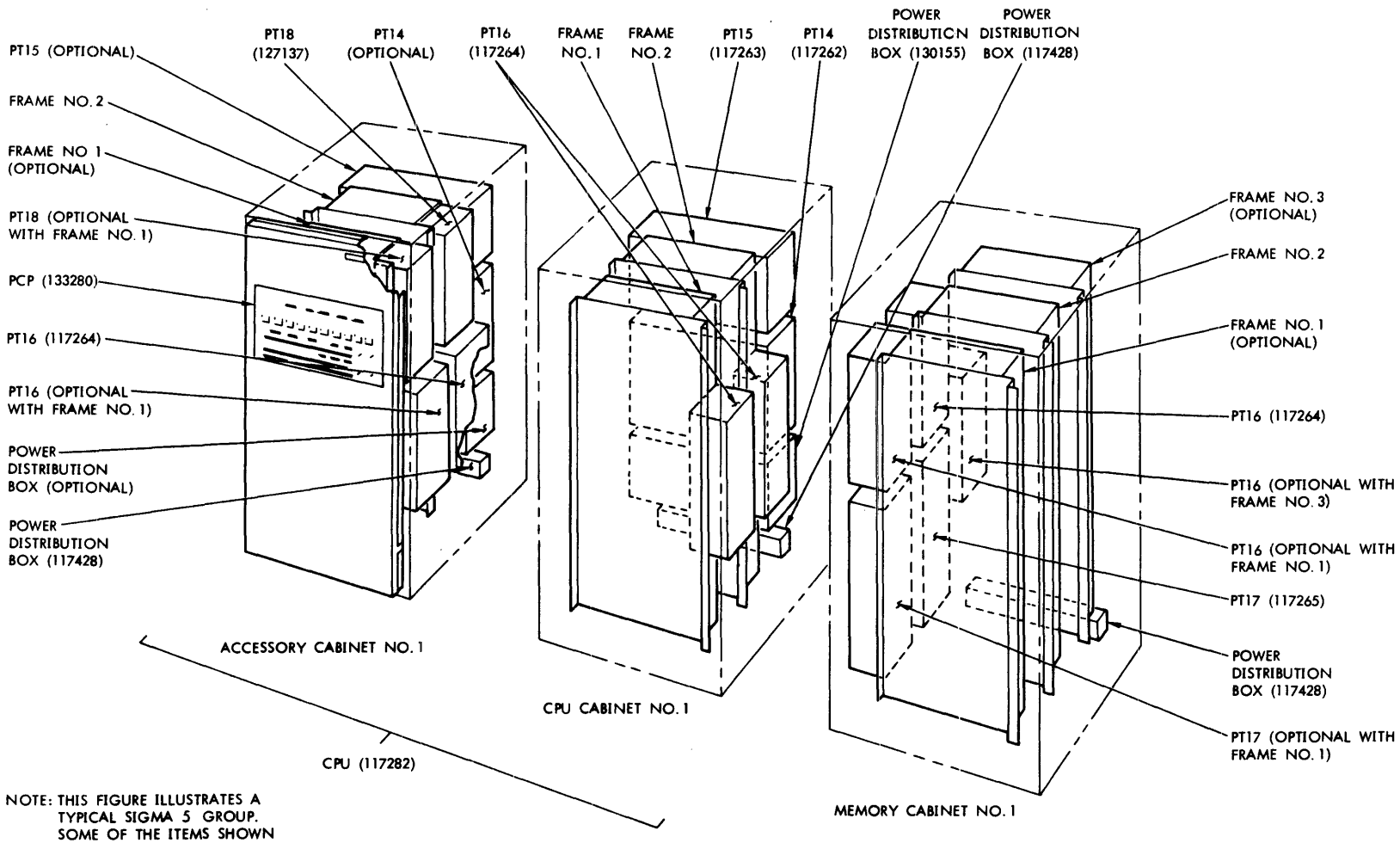
f. Quantity of the part used per assembly.

4-14 MANUFACTURER CODE INDEX (Table 4-52)

The manufacturers of parts listed in these tables are identified by code numbers. Their names and addresses may be found by consulting the manufacturer code index at the end of this section.

Table 4-19. Reference Documents for Sigma 5 Power Supplies

Power Supply	Assembly Drawing	Installation Drawing	Schematic Diagram	Technical Manual
PT14	117262	123310	123311	SDS 901078
PT15	117263	123310	123381	SDS 901078
PT16	117264	123352	123533	SDS 901080
PT17	117265	123636	123637	SDS 901079
PT18	127137	127156	127157	SDS 900866



NOTE: THIS FIGURE ILLUSTRATES A TYPICAL SIGMA 5 GROUP. SOME OF THE ITEMS SHOWN ARE OPTIONAL EQUIPMENT OR PART OF OPTIONAL EQUIPMENT

901172A.4001

Figure 4-3. Sigma 5 Computer Group

Table 4-20. Sigma 5 Computer Group, Replaceable Parts

Fig. & Index No.	Description	Reference Designator	Manufacturer	Part No.	Qty
4-3	Sigma 5 Computer Group		SDS		
	. Central Processing Unit With Integral IOP (see table 4-21 for parts breakdown)*		SDS	117282	1
	. Basic 4K Memory Module (see table 4-30 for parts breakdown)		SDS	132546	1-8
	. Options:				
	. . Real-Time Clock (see table 4-32 for parts breakdown)		SDS	117616	1
	. . Power Fail-Safe (see table 4-33 for parts breakdown)		SDS	117612	1
	. . Memory Protection Feature (see table 4-34 for parts breakdown)		SDS	134101	1
	. . Additional Register Block (see table 4-35 for parts breakdown)		SDS		1-15
	. . Floating Point Arithmetic (see table 4-39 for parts breakdown)		SDS	134099	1
	. . Interrupt, two level (see table 4-40 for parts breakdown)		SDS	132206	1-112
	. . Interrupt Control Chassis (see table 4-41 for parts breakdown)		SDS	117330	1-14
	. . Additional Groups of 8 Multiplexor Channels for Integral IOP (see table 4-42 for parts breakdown)		SDS	134077	1-3
	. . Memory Expansion Kit, 4K to 8K (see table 4-43 for parts breakdown)		SDS	117638	1-4
	. . Memory Expansion Kit, 8K to 12K (see table 4-44 for parts breakdown)		SDS	117639	1-4
	. . Memory Expansion Kit, 12K to 16K (see table 4-45 for parts breakdown)		SDS	117640	1-4
	. . Two-Way Access (see table 4-46 for parts breakdown)		SDS	129463	1-8
	. . Three-Way Access (see table 4-47 for parts breakdown)		SDS	128125	1-8
	. . Port Expander F (see table 4-48 for parts breakdown)		SDS	130625	1-4

*The Sigma 5 group may consist of a Central Processing Unit without integral IOP, Model 8202, in place of the Central Processing Unit with integral IOP Model 8201. Modules required for the Integral IOP are shown in figure 4-8.

Table 4-20. Sigma 5 Computer Group, Replaceable Parts (Cont.)

Fig. & Index No.	Description	Reference Designator	Manufacturer	Part No.	Qty
4-3 (Cont.)					
	. . Port Expander S (see table 4-49 for parts breakdown)		SDS	130626	1-4
	. . IOP/DC Expansion Kit (see SDS publication No. 901515 for parts breakdown)		SDS	117618	
	. . I/O Processor (see SDS publication No. 901515 for parts breakdown)		SDS	117610	
	. . Selector I/O Processor "A" (see SDS publication No. 901515 for parts breakdown)		SDS	117620	
	. . Selector I/O Processor "B" (see SDS publication No. 901515 for parts breakdown)		SDS	117620	
	. . External Interface Feature (see table 4-50 for parts breakdown)		SDS	137086	1
	. . External IOP Interface Feature (see table 4-51 for parts breakdown)		SDS		1

Table 4-21. Central Processing Unit With Integral IOP, Replaceable Parts

Fig. & Index No.	Description	Reference Designator	Manufacturer	Part No.	Qty
4-3	Central Processing Unit With Integral IOP (see table 4-20 for next higher assembly)		SDS	117282	Ref
	CPU Cabinet No. 1:				
	• Frame No. 1 (see table 4-22 for parts breakdown)				1
	• Frame No. 2 (see table 4-22 for parts breakdown)				1
	. Power Distribution Assembly (see table 4-25 for parts breakdown)		SDS	130155	1
	. Power Supply, PT14 (see SDS publication No. 901078 for parts breakdown)		SDS	117262	1
	. Power Supply, PT15 (see SDS publication No. 901078 for parts breakdown)		SDS	117263	1
	. Power Supply, PT16 (see SDS publication No. 901080 for parts breakdown)		SDS	117264	2
	. Power Distribution Box Assembly (see table 4-27 for parts breakdown)		SDS	117428	1
	. Module Assembly (see table 4-28 for parts breakdown)		SDS	146275	1
	Accessory Cabinet No. 1:				
	• Frame No. 2 (see table 4-22 for parts breakdown)				1
	. Power Supply, PT18 (see SDS publication No. 900866 for parts breakdown)		SDS	127137	1
	. Power Supply, PT16 (see SDS publication No. 901080 for parts breakdown)		SDS	117264	1
	. Processor Control Panel (see table 4-29 for parts breakdown)		SDS	133280	1
	. Power Distribution Box Assembly (see table 4-27 for parts breakdown)		SDS	117428	1

Table 4-22. Frame Assembly, Replaceable Parts

Fig. & Index No.	Description	Reference Designator	Manufacturer	Part No.	Qty
4-4	Frame Assembly (see table 4-21 for next higher assembly)		SDS		Ref
	. Fan, Top, Assembly (see table 4-23 for parts breakdown)		SDS	123943	1
	. Fan, Bottom, Assembly (see table 4-24 for parts breakdown)		SDS	117320	1

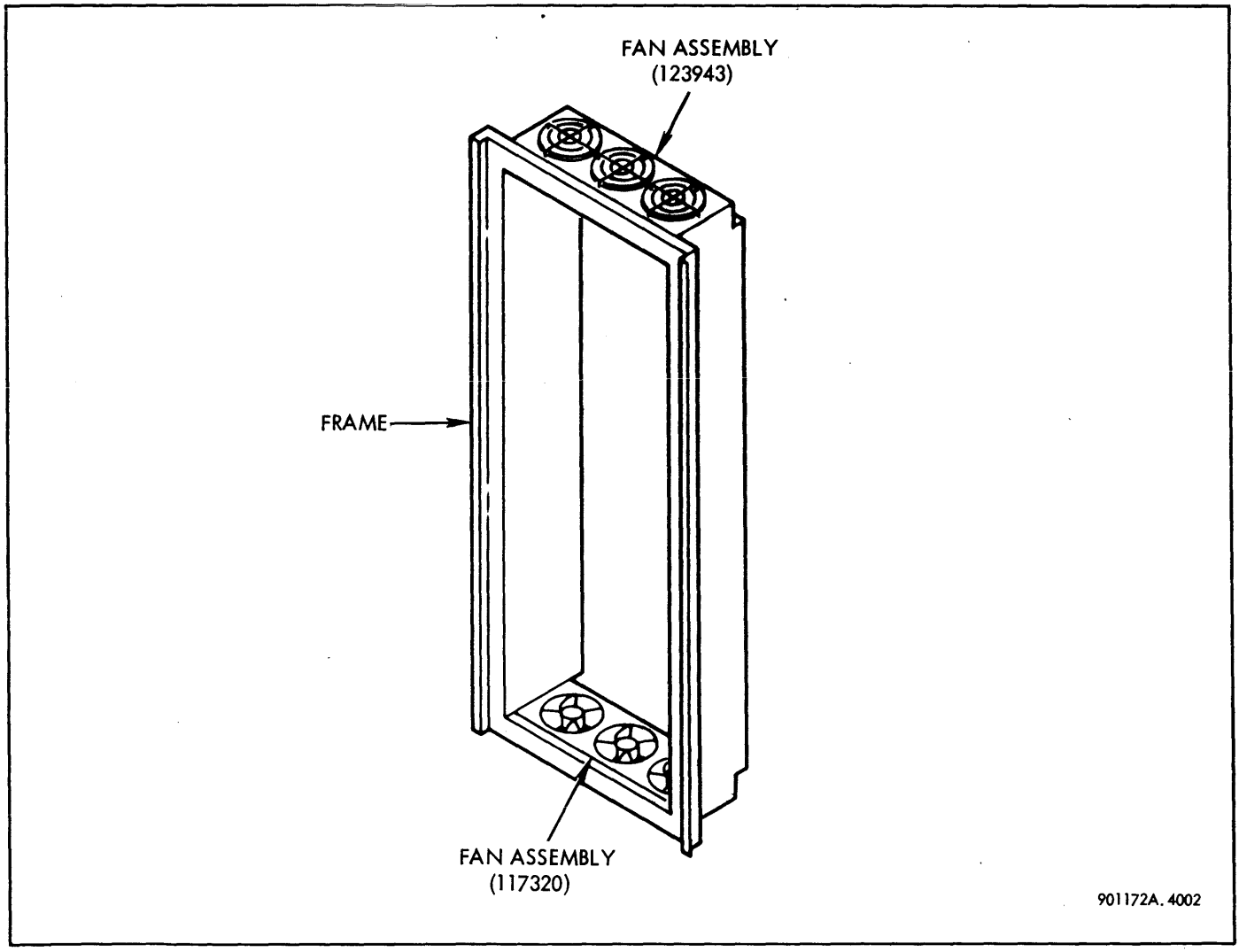


Figure 4-4. Frame Assembly With Fan Arrangement

Table 4-23. Fan, Top, Assembly, Replaceable Parts

Fig. & Index No.	Description	Reference Designator	Manufacturer	Part No.	Qty
4-4	Fan, Top, Assembly (see table 4-22 for next higher assembly)		SDS	123943	Ref
	. Fan, electric		139	104052	3
	. Cord, ac		378	126374-001	1

Table 4-24. Fan, Bottom, Assembly, Replaceable Parts

Fig. & Index No.	Description	Reference Designator	Manufacturer	Part No.	Qty
4-4	Fan, Bottom, Assembly (see table 4-22 for next higher assembly)		SDS	117320	Ref
	. Fan, electric		139	104052	3
	. Cord, ac		378	126374-001	1

Table 4-25. Power Distribution Assembly, Replaceable Parts

Fig. & Index No.	Description	Reference Designator	Manufacturer	Part No.	Qty
4-5	Power Distribution Assembly (see table 4-21 for next higher assembly)		SDS	130155	Ref
-1	. Contactor, 3-pole	CB1	211	130422-001	1
-2	. Switch, toggle, 3-position	S1	54	130462	1
-3	. Outlet, duplex, female	J3, J4	106	127672	2
-4	. Block, terminal	TB1	107	109432-007	5
-5	. Inlet, flanged, male	P1	365	127675	1
-6	. Connector, female	J1	365	101430	1
-7	. Socket, female	J2	51	100544	1
-8	Power Monitor Assembly (see table 4-26 for parts breakdown)		SDS	132389	1

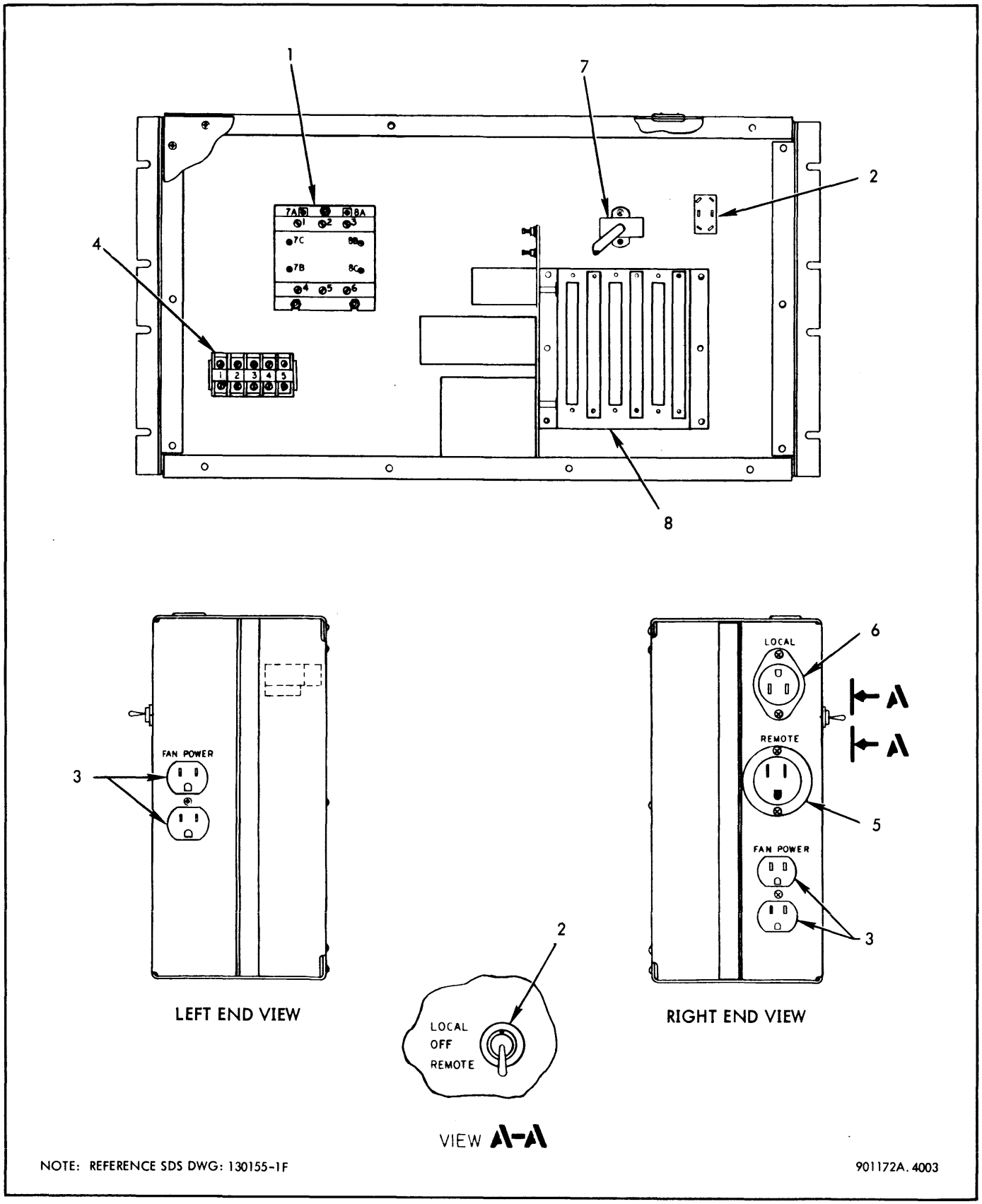


Figure 4-5. Power Distribution Assembly

Table 4-26. Power Monitor Assembly, Replaceable Parts

Fig. & Index No.	Description	Reference Designator	Manufacturer	Part No.	Qty
4-6	Power Monitor Assembly (see table 4-25 for next higher assembly)		SDS	132389	Ref
-1	. Fan, electric		139	104052	1
-2	. Connector, solder tail	J1, J2, J3	356	117874	3
-3	. Transistor, SDS 225, 01, 02		1	107820	2
-4	. Post, extractor, fuse	XF1	49	100331	1
-5	. Switch, subminiature toggle, spdt	S1	54	107396	1
-6	. Diode, rectifier, SDS 125	CR1, CR2, CR3 CR4, CR5, CR6	211	123939	6
-7	. Transformer, power supply	T1	145	117115	1
-8	. Resistor, wirewound, 20w	R1	45	101155-102	1
-9	. Capacitor, electrolytic	C2	20	108474-019	1
-10	. Capacitor, electrolytic	C1, C3	20	108474-004	2
-11	. Plug, 10 pin	P1	51	100532	1
-12	. Cable Driver Assembly, AT13		SDS	125260	1
-13	. Detector Assembly, WT22		SDS	131183	1
-14	. Regulator Assembly, WT21		SDS	131181	1
-15	. Fuse, 3 AG, slow burning	F1	48	124865-011	1

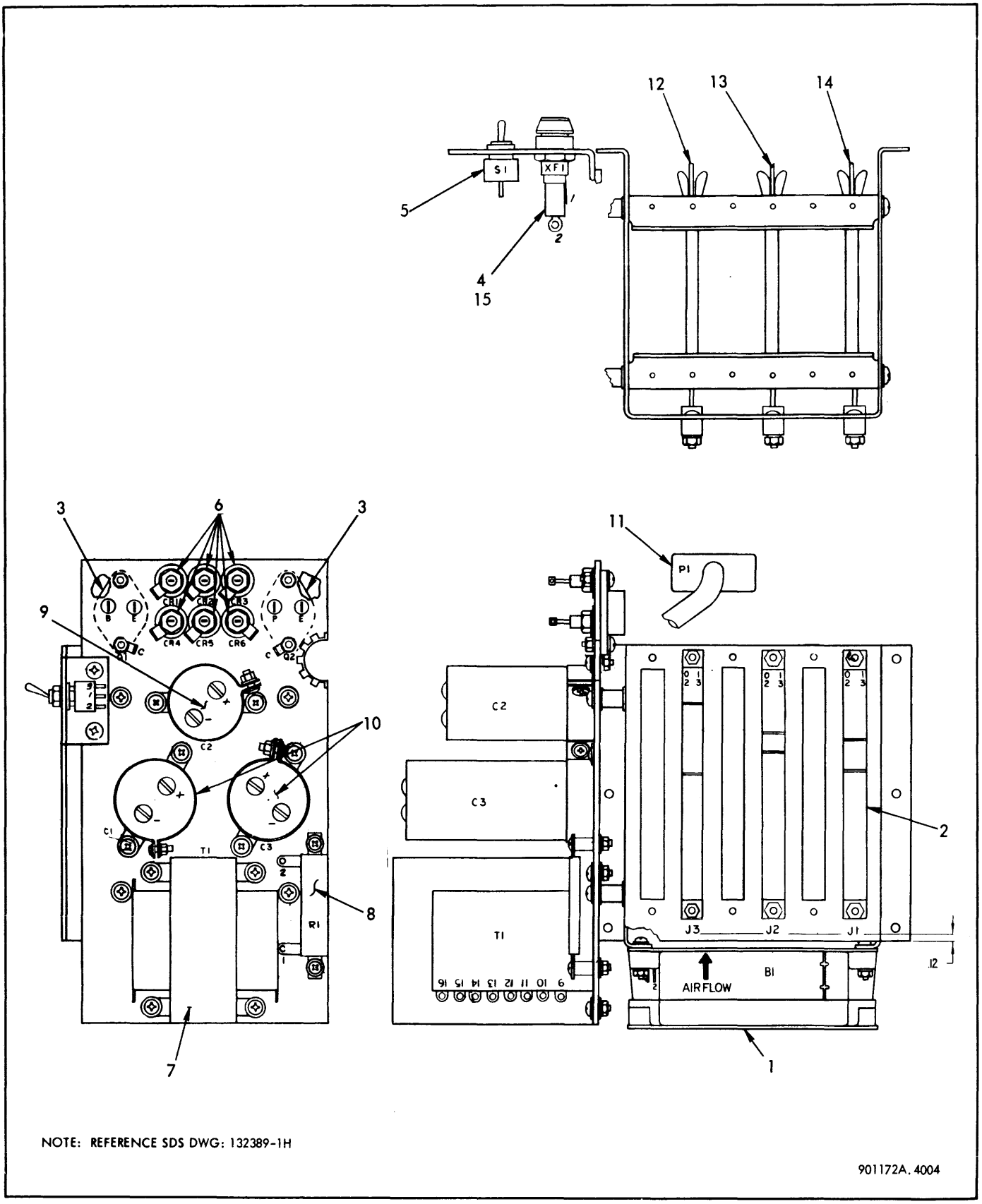


Figure 4-6. Power Monitor Assembly

Table 4-27. Power Distribution Box Assembly, Replaceable Parts

Fig. & Index No.	Description	Reference Designator	Manufacturer	Part No.	Qty
4-7	Power Distribution Box Assembly (see table 4-21 for next higher assembly)		SDS	117428	Ref
-1	. Power Distribution Box		SDS	126846	1
-2	. Receptacle, female (twist lock)		106	127677	3
-3	. Receptacle, female		106	127672	2
-4	. Connector, male		365	127679	1
-5	. Connector, male		365	127674	1

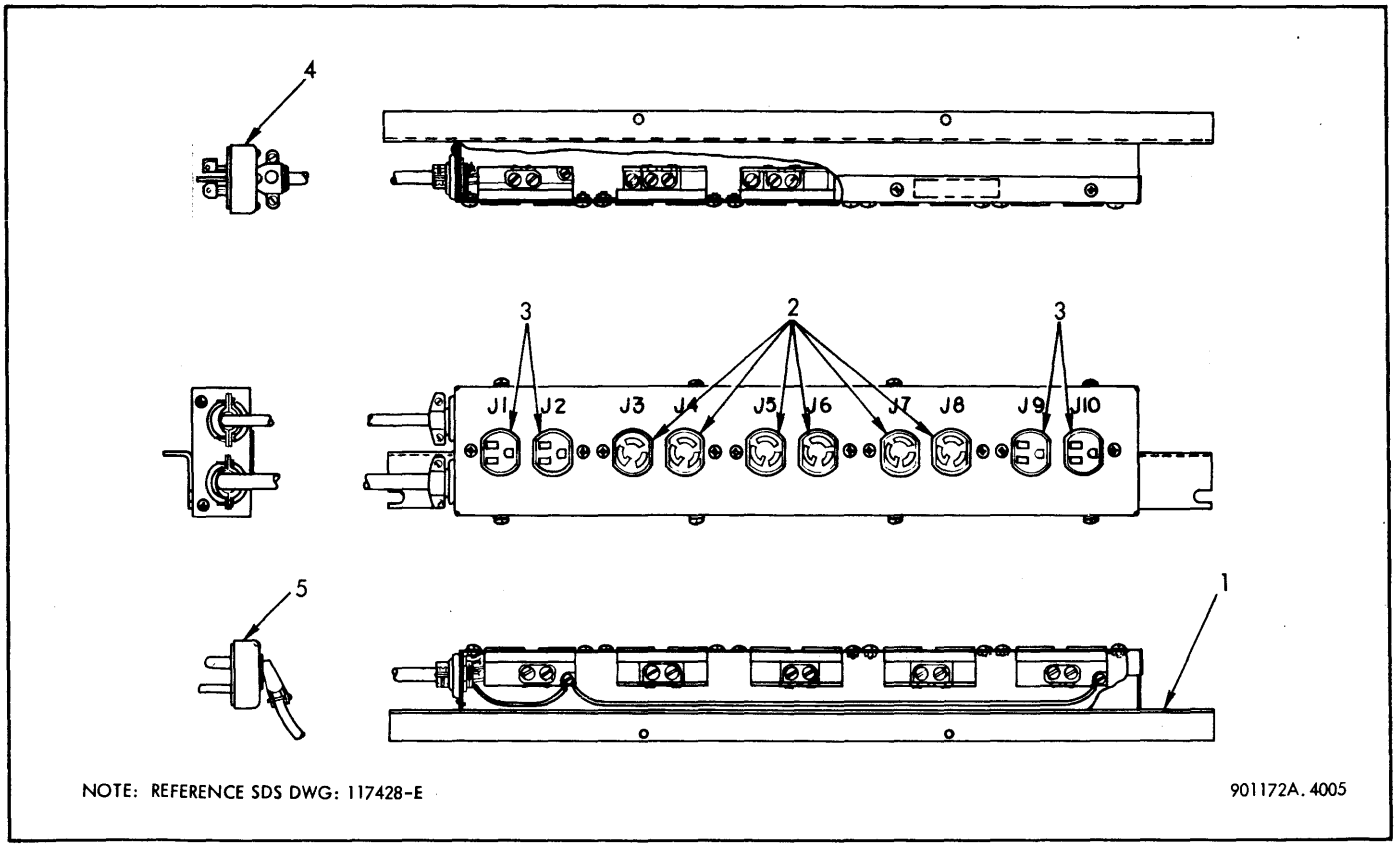


Figure 4-7. Power Distribution Box Assembly

Table 4-28. Module Assembly, Replaceable Parts

Fig. & Index No.	Description	Reference Designator	Manufacturer	Part No.	Qty
4-8	Module Assembly (see table 4-21 for next higher assembly)		SDS	146275	Ref
	. Cable Receiver, printed wire assembly, AT10		SDS	123018	1
	. Cable Driver Receiver, printed wire assembly, AT11		SDS	123019	4
	. Cable Driver, printed wire assembly, AT12		SDS	124629	2
	. Cable Driver, printed wire assembly, AT21		SDS	127797	7
	. Clock Driver, printed wire assembly, AT23		SDS	128166	4
	. Clock Driver, printed wire assembly, AT13		SDS	125260	2
	. Buffered AND/OR Gate, printed wire assembly, BT10		SDS	116056	9
	. Band Gate, printed wire assembly, BT11		SDS	116029	22
	. Gated Buffer, printed wire assembly, BT16		SDS	125262	21
	. Gated Buffer, printed wire assembly, BT17		SDS	126330	2
	. Band Gate, printed wire assembly, BT18		SDS	126613	7
	. Buffered Matrix, printed wire assembly, BT13		SDS	116407	1
	. Clock Oscillator, printed wire assembly, CT16		SDS	133694	1
	. Delay Line, printed wire assembly, DT14		SDS	127319	3
	. Register Flip-Flop, printed wire assembly, FT17		SDS	124628	6
	. Counter Flip-Flop, printed wire assembly, FT18		SDS	124634	10
	. Universal Flip-Flop, printed wire assembly, FT22		SDS	124713	24
	. Fast Access Memory, 16X18, printed wire assembly, FT25		SDS	126743	4
	. Buffered Latch No. 3, printed wire assembly, FT26		SDS	126856	1
	. Register Flip-Flop, printed wire assembly, FT41		SDS	133251	1
	. Gate Expander No. 1, printed wire assembly, GT10		SDS	124750	2

(Continued)

Table 4-28. Module Assembly, Replaceable Parts (Cont.)

Fig. & Index No.	Description	Reference Designator	Manufacturer	Part No.	Qty
4-8 (Cont.)	. Gate Expander No. 2, printed wire assembly, GT11		SDS	125271	6
	. Gate Expander, printed wire assembly, GT12		SDS	133375	8
	. Delay Line Sensors, printed wire assembly, HT15		SDS	127391	2
	. Gated Delay Line Sensors, printed wire assembly, HT16		SDS	128011	1
	. Inverter Matrix, printed wire assembly, IT13		SDS	117000	1
	. Gated Inverter, printed wire assembly, IT16		SDS	125264	33
	. Gated Inverter, printed wire assembly, IT17		SDS	126331	4
	. Gated Inverter, printed wire assembly, IT20		SDS	126747	18
	. NAND Gate, printed wire assembly, IT25		SDS	128190	29
	. NAND Gate, printed wire assembly, IT18		SDS	126372	1
	. NAND Gate, printed wire assembly, IT26		SDS	128192	1
	. Buffer Inverter No. 1, printed wire assembly, LT13		SDS	123016	5
	. Buffer Inverter No. 2, printed wire assembly, LT14		SDS	123017	12
	. Priority Interrupt, printed wire assembly, LT16		SDS	123379	4
	. Carry No. 1, printed wire assembly, LT18		SDS	123590	5
	. Logic Element, printed wire assembly, LT20		SDS	124717	8
	. Logic Element, printed wire assembly, LT21		SDS	126615	4
	. Clock Logic, printed wire assembly, LT29		SDS	127643	1
	. Adder No. 3, printed wire assembly, LT42		SDS	133383	17
	. SW Module, printed wire assembly, ST14		SDS	123008	1
	. Time Base Selector, printed wire assembly, ST29		SDS	129460	1

(Continued)

Table 4-28. Module Assembly, Replaceable Parts (Cont.)

Fig. & Index No.	Description	Reference Designator	Manufacturer	Part No.	Qty
4-8 (Cont.)	. Terminator Module, printed wire assembly, XT10		SDS	116527	40
	. Clock Term, printed wire assembly, XT18		SDS	132009	4
	. Cable Plug-Clock, printed wire assembly, ZT23		SDS	128164	5
	. Jumper Module, printed wire assembly, ZT50		SDS	139244	1

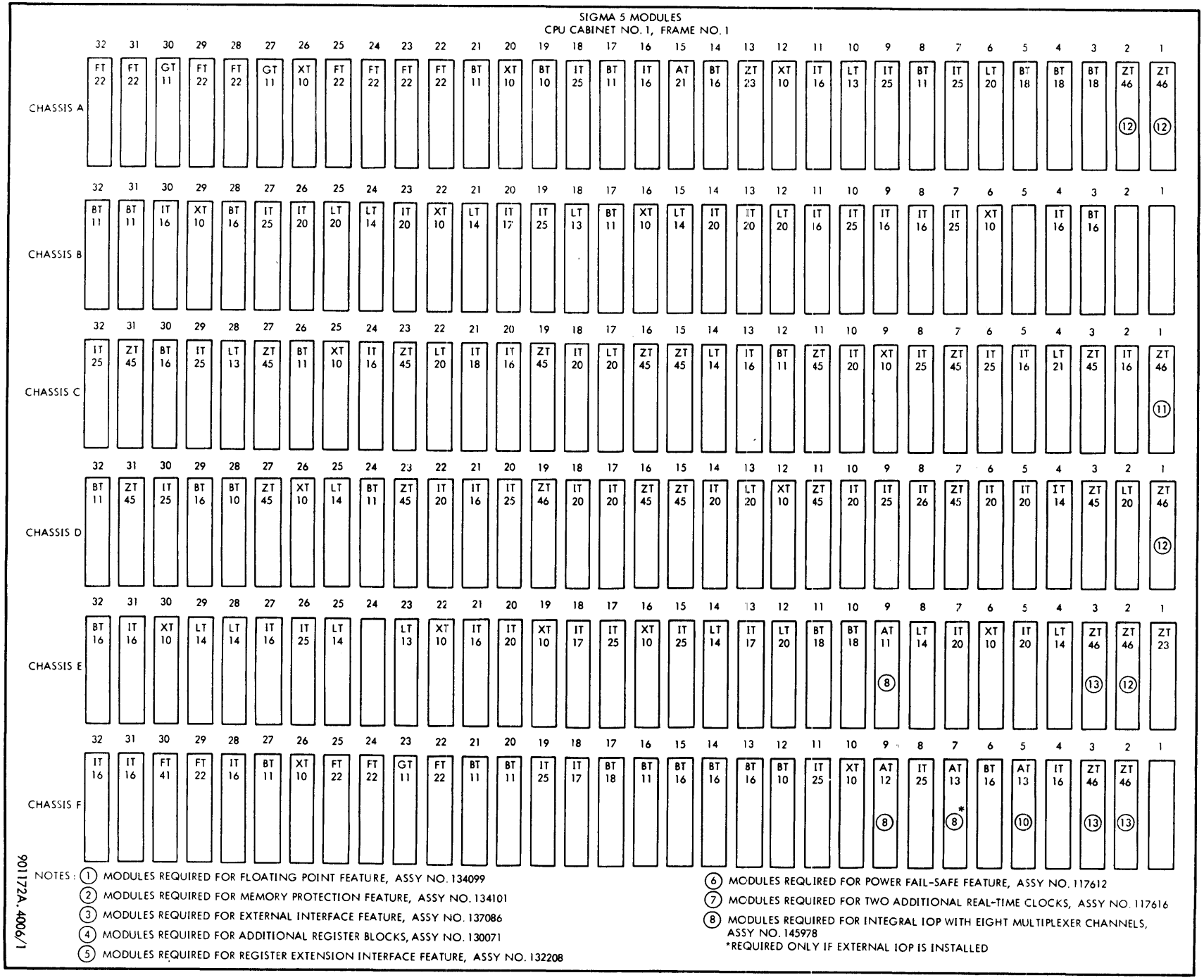


Figure 4-8. Module Assembly, CPU Cabinet No. 1, Frame 1 (Sheet 1 of 2 sheets)

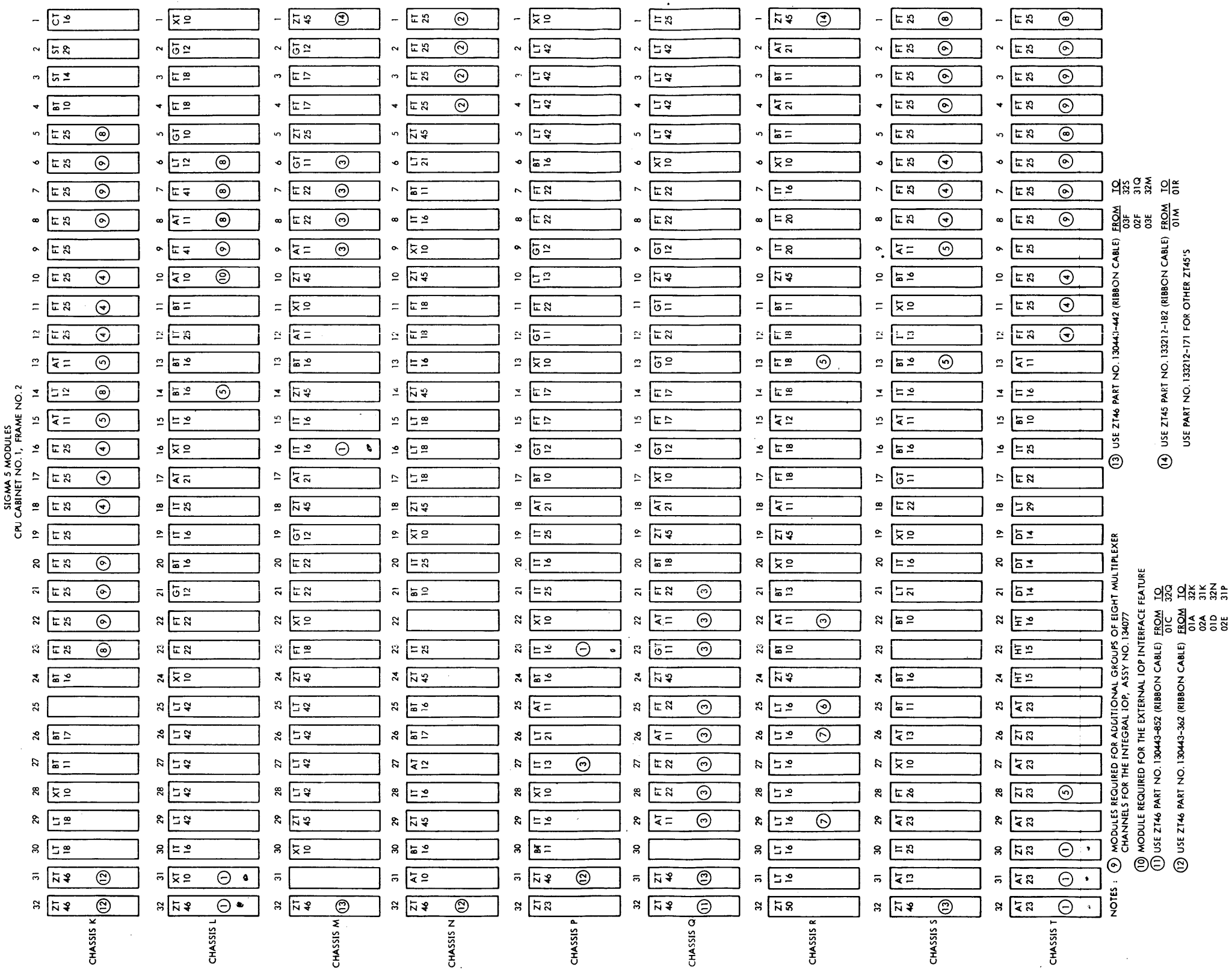


Figure 4-8. Module Assembly, CPU Cabinet No. 1, Frame 1 (Sheet 2 of 2 sheets)

901172A.4006/2

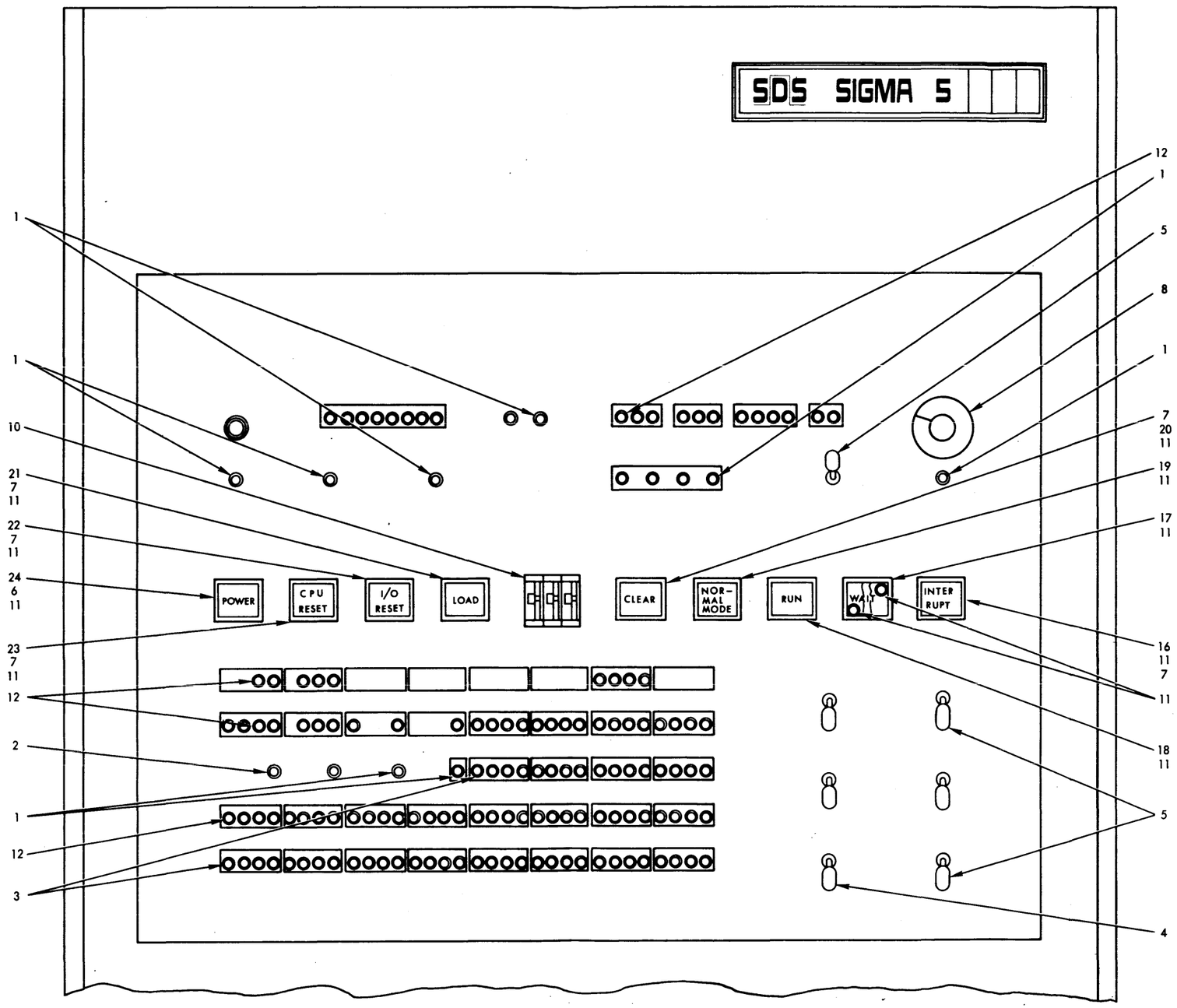
Table 4-29. Processor Control Panel Assembly, Replaceable Parts

Fig. & Index No.	Description	Reference Designator	Manufacturer	Part No.	Qty
4-9	Processor Control Panel Assembly (see table 4-21 for next higher assembly)		SDS	133280	Ref
-1	Switch, lever, single state	S2, S4, S6 thru S12, S41, & S41	384	124406-001	11
-2	Switch, lever	S76, S77	384	124406-003	2
-3	Switch, lever, 8 station	S24 thru S39 S44 thru S75	384	124404	6
-4	Switch, lever	S21, S22, S23, S43	156, 384	126993	4
-5	Switch, lever	S5, S20, S42	156, 384	126994	3
-6	Switch, alternating action dpdt	S19	162, 203, 381	111455	1
-7	Switch, momentary, dpdt	S13, S14, S16, S17, S18	162, 203, 381	111459	5
-8	Switch, rotary	S1	55, 208	115928	1
-9	Switch, rotary	S3	SDS	133967	1
-10	Switch, thumbwheel, 1-6 position	S16	82, 140, 387	126600-003	1
-11	Lamp, miniature, incandescent		83, 84, 211, 382	102266	18
-12	Lamp, miniature, incandescent		56, 63, 104	123710	88
-13	Receptacle, female	J32	365	101430	1
-14	Speaker, miniature	SP1	379, 380	108042	1
-15	Receptacle, male	J31	365	127675	1
-16	Lampholder (INTERRUPT)	DS22	162, 203, 381	116284-002	1
-17	Lampholder (WAIT)	DS23	162, 203, 381	116284-003	1
-18	Lampholder (RUN)	DS24	162, 203, 381	116284-004	1
-19	Lampholder (NORMAL MODE)	DS25	162, 203, 381	116284-005	1
-20	Lampholder (CLEAR)	DS26	162, 203, 381	116284-006	1
-21	Lampholder (LOAD)	DS27	162, 203, 381	116284-007	1
-22	Lampholder (I/O RESET)	DS99	162, 203, 381	116284-008	1
-23	Lampholder (CPU RESET)	DS100	162, 203, 381	116284-009	1

(Continued)

Table 4-29. Processor Control Panel Assembly, Replaceable Parts (Cont.)

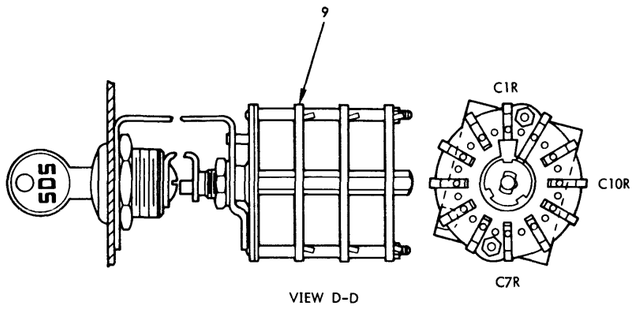
Fig. & Index No.	Description	Reference Designator	Manufacturer	Part No.	Qty
4-9 (Cont.)					
-24	Lampholder (POWER)	DS28	162, 203, 381	116284-010	1
-25	Lamp Driver, printed wire assembly, QT14		SDS	132055	1
-26	Console Interface, printed wire assembly, NT26		SDS	134936	7
-27	Cable Plug-Clock, printed wire assembly, ZT23		SDS	124164	1
-28	Block, terminal	TB1	107	109432-009	1
-29	Connector, one pin		221	130811	2



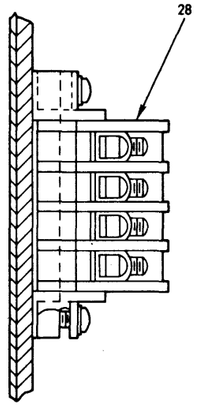
NOTE: REFERENCE SDS DWG: 133280-1B

Figure 4-9. Processor Control Panel Assembly (Sheet 1 of 2 sheets)

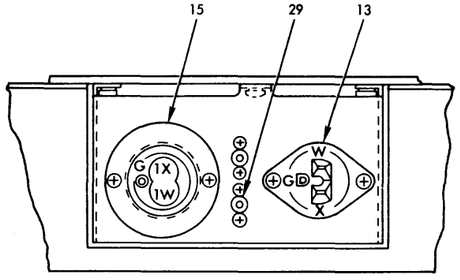
901172A.4007/1



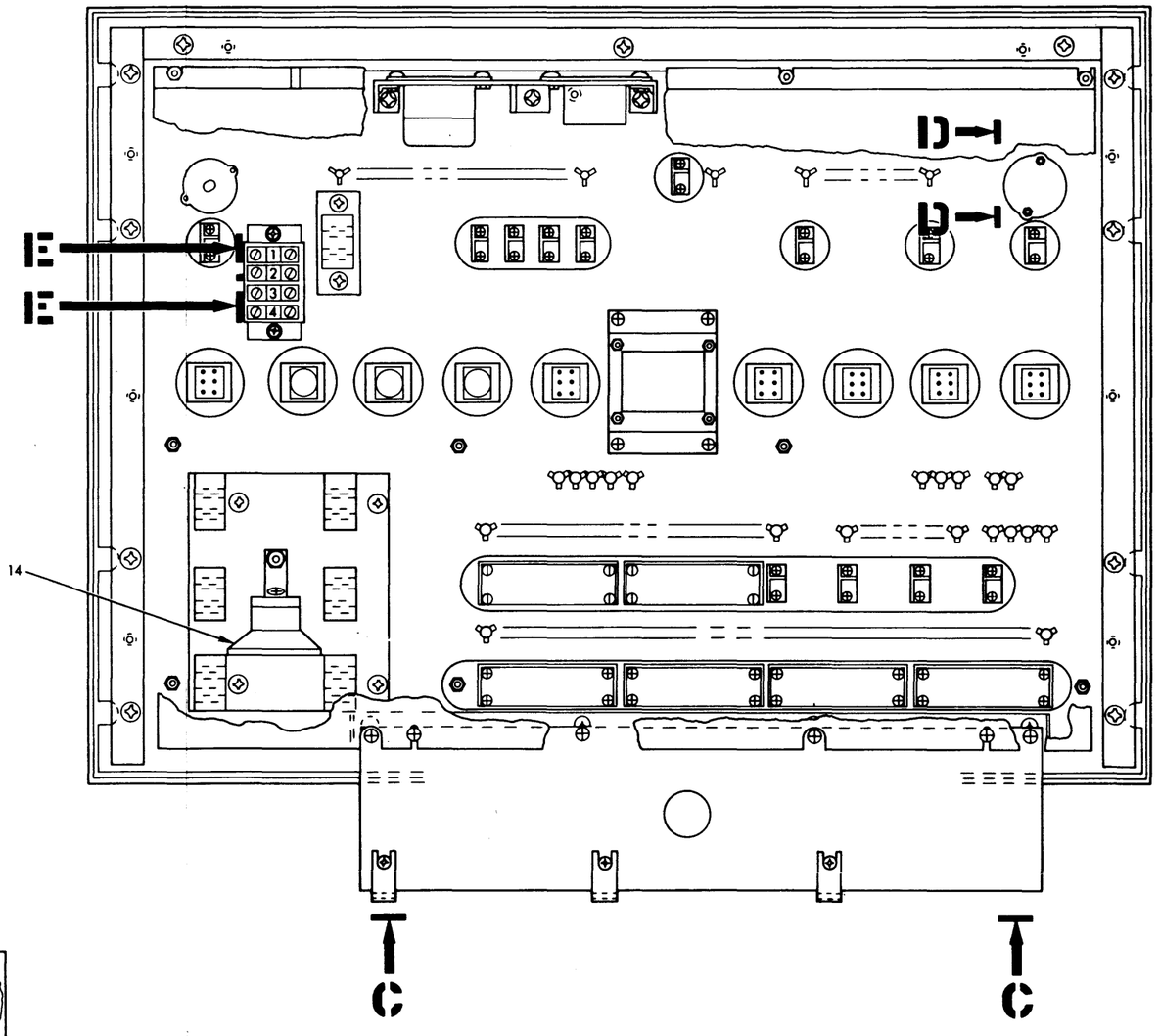
VIEW D-D

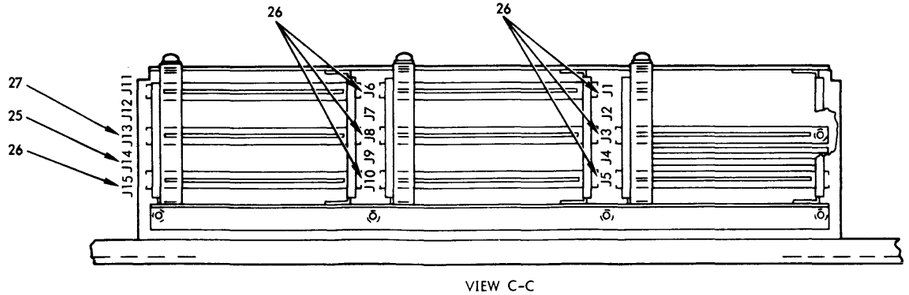


SECTION E-E



VIEW B-B





VIEW C-C

NOTE: REFERENCE SDS DWG: 133280-2B, 3B

Figure 4-9. Processor Control Panel Assembly
(Sheet 2 of 2 sheets)
901172A.4007/2

Table 4-30. Memory Module, Basic 4K, Replaceable Parts

Fig. & Index No.	Description	Reference Designator	Manufacturer	Part No.	Qty
4-3	Memory Module, basic 4K (see table 4-20 for next higher assembly))		SDS	132546	Ref
	. Frame No. 1 (see table 4-22 for parts breakdown)*		SDS		1
	. Frame No. 2 (see table 4-22 for parts breakdown)*		SDS		1
	. Power Supply, PT16 (see SDS publication No. 901080 for parts breakdown)		SDS	117264	1
	. Power Supply, PT17 (see SDS publication No. 901079 for parts breakdown)		SDS	117265	1
	. Module Assembly (see table 4-31 for parts breakdown)		SDS		1
	. Memory Cabinet† Assembly, power distribution box (see table 4-27 for parts breakdown)		SDS	117428	1

*The first basic memory block (up to 16K with memory increments, assemblies 117638, 117639, and 117640) is contained in frame 2 of memory cabinet 1. The next memory block is contained in frame 1 of memory cabinet 1. Additional memory blocks, up to 8 total, are contained in memory cabinets 2 through 4

†Additional memory cabinets are added as required

Table 4-31. Module Assembly, Replaceable Parts

Fig. & Index No.	Description	Reference Designator	Manufacturer	Part No.	Qty
4-10	Module Assembly (see table 4-30 for next higher assembly)		SDS		Ref
	. Core Diode Module Assembly		SDS	111550	1
	. Core Diode Module Assembly		SDS	111549	3
	. Cable Receiver, AT10		SDS	123018	1
	. Cable Driver Receiver, AT11		SDS	123019	6
	. Rejection Gate, AT16		SDS	126611	2
	. Cable Driver Receiver, AT31		SDS	133053	1
	. Gated Buffer, BT16		SDS	125265	6
	. Fast Buffer, BT22		SDS	127393	11
	. Buffered AND/OR Gate, BT24		SDS	130967	3
	. Band Gate, BT25		SDS	130947	1
	. Delay Line, DT11		SDS	126963	2
	. Buffered Latch No. 2, FT37		SDS	130942	3
	. Buffered Latch No. 3, FT38		SDS	130952	7
	. Memory Sense Amplifier, HT11		SDS	123010	6
	. Delay Line Sensor, HT15		SDS	127391	3
	. Memory Preamplifier, HT26		SDS	131633	6
	. Gated Inverter, IT14		SDS	126617	6
	. Gated Inverter, IT16		SDS	125264	3
	. NAND/NOR Gate, IT24		SDS	128188	2
	. NAND Gate, IT25		SDS	128190	2
	. Logic Element, LT19		SDS	123915	1
	. Logic Element with inverter, LT20		SDS	124717	1
	. Logic Element with buffer, LT21		SDS	126615	5
	. Parity Generator, LT34		SDS	130958	9

(Continued)

Table 4-31 Module Assembly, Replaceable Parts (Cont.)

Fig. & Index No.	Description	Reference Designator	Manufacturer	Part No.	Qty
4-10 (Cont.)	. Memory Switch A, ST10		SDS	123005	10
	. Memory Switch B, ST11		SDS	123006	16
	. Toggle Switch Module, ST14		SDS	123008	2
	. Memory Preamp Selector, ST15		SDS	123012	1
	. Voltage Regulator, ST17		SDS	131292	1
	. Inhibit Driver, ST21		SDS	132153	6
	. Memory Driver, ST22		SDS	132159	1
	. Strobe Generator, ST34		SDS	130902	2
	. Terminator Module, XT10		SDS	116257	16
	. Resistor Module C, XT13		SDS	127791	9
	. Resistor Module D, XT14		SDS	127793	1
	. Cable Intra-frame Assembly, ZT35		SDS	132411-171	3
	. Coaxial Cable Connection		SDS	115832	2
	. Resistor Connector Assembly		SDS	127315	2

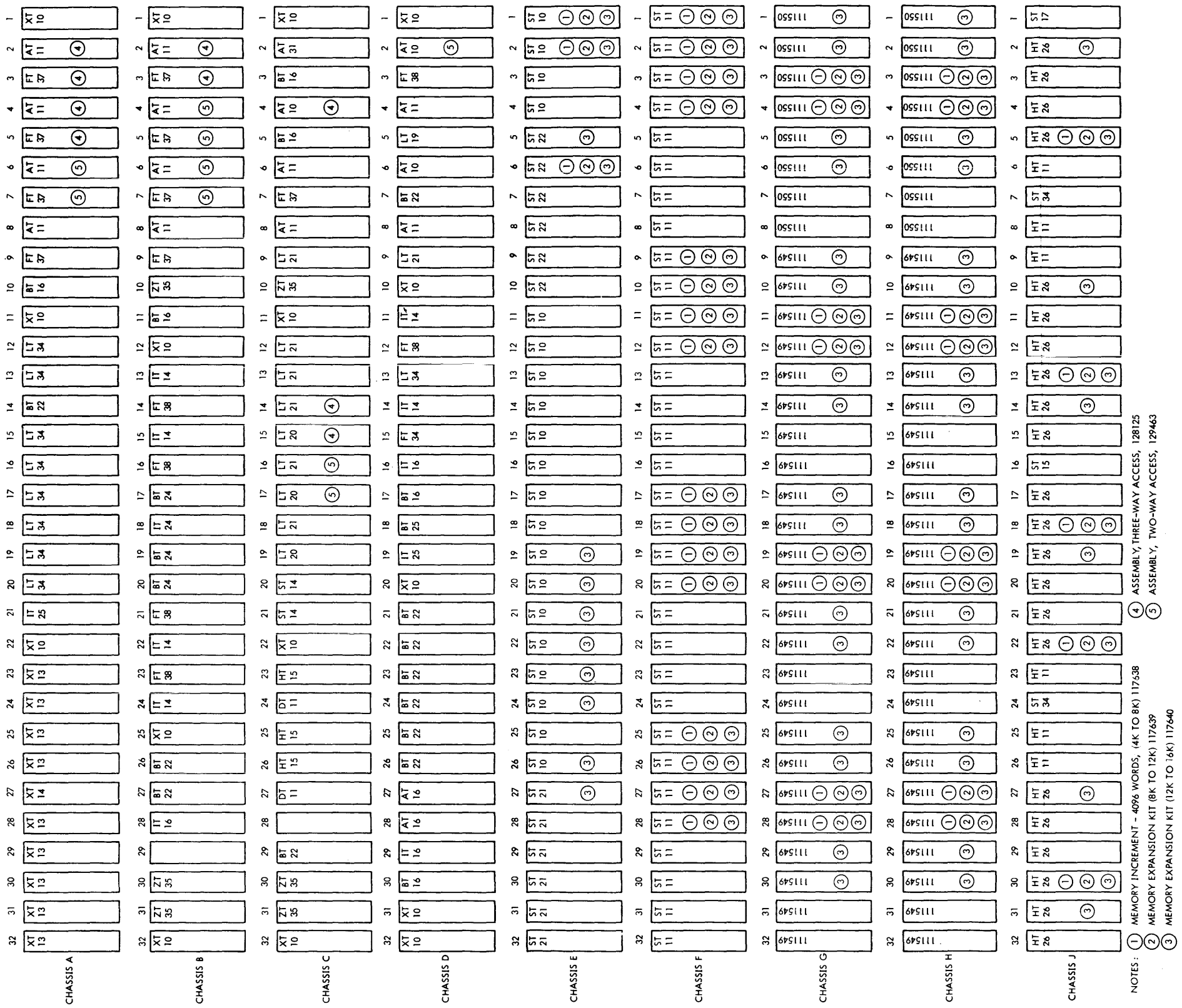


Figure 4-10. Module Assembly, Memory Cabinet, Frames 1 and 2

901172A.4008

Table 4-32. Real-Time Clock, Replaceable Parts

Fig. & Index No.	Description	Reference Designator	Manufacturer	Part No.	Qty
4-8	Real-Time Clock (see table 4-20 for next higher assembly)		SDS	117616	Ref
	. Printed Wire Assembly, LT16		SDS	123379	2

Table 4-33. Power Fail-Safe, Replaceable Parts

Fig. & Index No.	Description	Reference Designator	Manufacturer	Part No.	Qty
4-8	Power Fail-Safe (see table 4-20 for next higher assembly)		SDS	117612	Ref
	. Printed Wire Assembly, LT16		SDS	123379	1

Table 4-34. Memory Protection Feature, Replaceable Parts

Fig. & Index No.	Description	Reference Designator	Manufacturer	Part No.	Qty
4-8	Memory Protection Feature (see table 4-20 for next higher assembly)		SDS	134101	Ref
	. Fast Access Memory, FT25		SDS	123743	4
	. NAND/NOR Gate, IT24		SDS	128188	1

Table 4-35. Additional Register Block, Replaceable Parts

Fig. & Index No.	Description	Reference Designator	Manufacturer	Part No.	Qty
4-8	Additional Register Block (see table 4-20 for next higher assembly)		SDS		Ref
	. High-Speed Register Page (see table 4-36 for parts breakdown)		SDS	117621	1
	. Register Extension Unit (see table 4-37 for parts breakdown)		SDS	130071	1*
	. Register Extension Unit Interface (see table 4-38 for parts breakdown)		SDS	132208	1†

*The first three additional register blocks (0 to 3) require only one high-speed register page assembly to supplement the additional register blocks. The next four additional register blocks (4 to 7) require one to four register page assemblies and one register extention unit, as do the (8-11) blocks and the (12 to 15) blocks.

†The register extension unit interface is added with the first register extension unit only.

Table 4-36. High-Speed Register Page, Replaceable Parts

Fig. & Index No.	Description	Reference Designator	Manufacturer	Part No.	Qty
4-8	High-Speed Register Page (see table 4-35 for next higher assembly)		SDS	117621	Ref
4-11*	. Printed Wire Board Assembly, FT25		SDS	126743	4

*The first three high-speed register page modules are installed in the CPU and are shown in figure 4-8. Additional modules are installed in the register extension units, assembly No. 130071, and are shown in figure 4-11.

Table 4-37. Register Extension Unit, Replaceable Parts

Fig. & Index No.	Description	Reference Designator	Manufacturer	Part No.	Qty
4-11	Register Extension Unit (see table 4-35 for next higher assembly)		SDS	130071	Ref
	. Printed Wire Board Assembly, AT11		SDS	123019	5
	. Printed Wire Board Assembly, BT16		SDS	125262	2
	. Printed Wire Board Assembly, IT16		SDS	125264	1
	. Printed Wire Board Assembly, LT26		SDS	126982	1
	. Printed Wire Board Assembly, XT10		SDS	116257	3

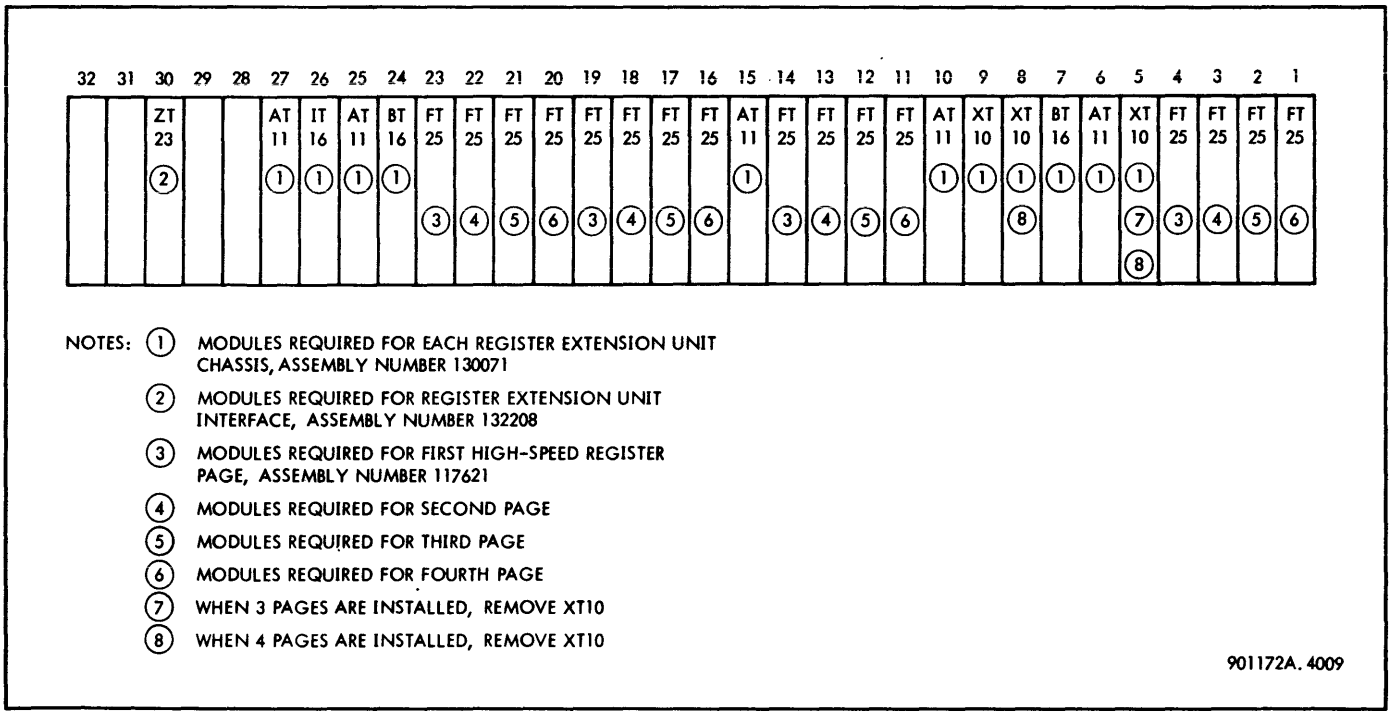


Figure 4-11. Module Assembly, Register Extension Unit, Register Interface, High-Speed Register Page

Table 4-38. Register Extension Unit Interface, Replaceable Parts

Fig. & Index No.	Description	Reference Designator	Manufacturer	Part No.	Qty
4-8	Register Extension Unit Interface (see table 4-35 for next higher assembly)		SDS	132208	Ref.
4-11	. Printed Wire Board Assembly, AT11		SDS	123019	4
	. Printed Wire Board Assembly, AT23		SDS	128166	1
	. Cable Module Assembly, ZT23		SDS	128164	2

Table 4-39. Floating Point Arithmetic, Replaceable Parts

Fig. & Index No.	Description	Reference Designator	Manufacturer	Part No.	Qty
4-8 4-12	Floating Point Arithmetic (see table 4-20 for next higher assembly)		SDS	134099	Ref.
	. Cable, single condition coaxial		SDS	128147-372	14
	. Printed Wire Board Assembly, AT23		SDS	128166	2
	. Printed Wire Board Assembly, BT10		SDS	116056	3
	. Printed Wire Board Assembly, BT11		SDS	116029	2
	. Printed Wire Board Assembly, BT16		SDS	125262	3
	. Printed Wire Board Assembly, BT18		SDS	126613	1
	. Printed Wire Board Assembly, FT18		SDS	124634	4
	. Printed Wire Board Assembly, FT22		SDS	124713	10
	. Printed Wire Board Assembly, FT26		SDS	126856	4
	. Printed Wire Board Assembly, FT41		SDS	133251	7
	. Printed Wire Board Assembly, GT11		SDS	124881	2
	. Printed Wire Board Assembly, GT12		SDS	133375	7
	. Printed Wire Board Assembly, IT16		SDS	125264	9
	. Printed Wire Board Assembly, IT17		SDS	126331	1
	. Printed Wire Board Assembly, IT25		SDS	128190	3
	. Printed Wire Board Assembly, IT26		SDS	128192	1
	. Printed Wire Board Assembly, LT18		SDS	123590	9
	. Printed Wire Board Assembly, LT20		SDS	124717	2
	. Printed Wire Board Assembly, LT42		SDS	133383	29
	. Printed Wire Board Assembly, ST14		SDS	123008	1
	. Printed Wire Board Assembly, XT10		SDS	116257	11
	. Printed Wire Board Assembly, ZT25		SDS	128164	1
	. Ribbon Cable Assembly, ZT46		SDS	133204-113	1

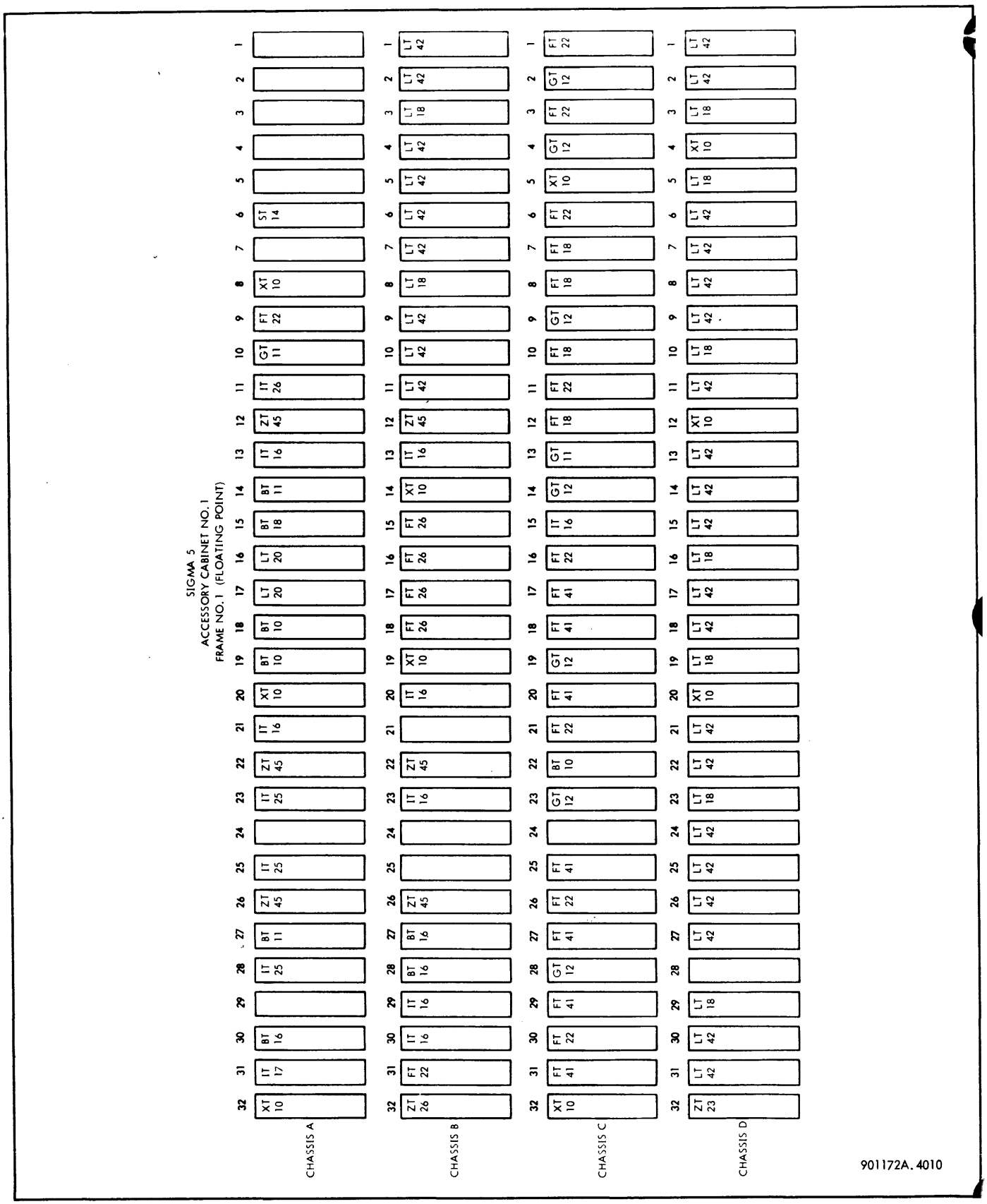


Figure 4-12. Module Assemblies, Accessory Cabinet No. 1, Frame 1, Floating Point

Table 4-40. Interrupt, 2 Level Assembly, Replaceable Parts

Fig. & Index No.	Description	Reference Designator	Manufacturer	Part No.	Qty
4-13	Interrupt, 2 Level Assembly (see table 4-20 for next higher assembly)		SDS	132206	Ref.
	. Printed Wire Board Assembly, LT16		SDS	123379	1

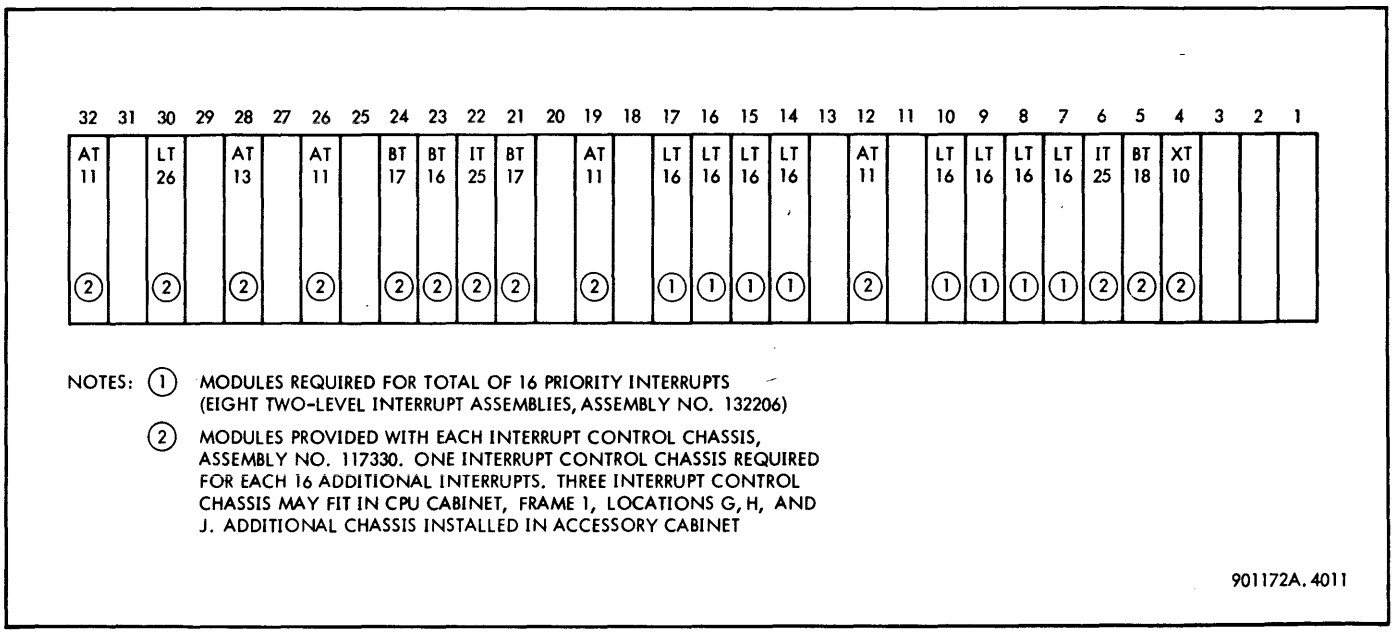


Figure 4-13. Module Assembly, Interrupt Control Chassis

Table 4-41. Interrupt Control Chassis, Replaceable Parts

Fig. & Index No.	Description	Reference Designator	Manufacturer	Part No.	Qty
4-13	Interrupt Control Chassis (see table 4-20 for next higher assembly)		SDS	117330	Ref.
	. Printed Wire Board Assembly, AT11		SDS	123019	4
	. Printed Wire Board Assembly, AT13		SDS	125260	1
	. Printed Wire Board Assembly, BT16		SDS	125262	1
	. Printed Wire Board Assembly, BT17		SDS	126330	2
	. Printed Wire Board Assembly, BT18		SDS	126613	1
	. Printed Wire Board Assembly, LT25		SDS	128190	2
	. Printed Wire Board Assembly, LT26		SDS	126982	1
	. Printed Wire Board Assembly, ST14		SDS	123008	1
	. Printed Wire Board Assembly, XT10		SDS	116982	1

Table 4-42. Additional Groups of Eight Multiplexer Channels for Integral IOP, Replaceable Parts

Fig. & Index No.	Description	Reference Designator	Manufacturer	Part No.	Qty
4-8	Additional Groups of Eight Multiplexer Channels for Integral IOP (see table 4-20 for next higher assembly)		SDS	134077	Ref.
	. Printed Wire Board Assembly, FT25		SDS	126743	4

Table 4-43. Memory Expansion Kit, 4K to 8K, Replaceable Parts

Fig. & Index No.	Description	Reference Designator	Manufacturer	Part No.	Qty
4-10	Memory Expansion Kit, 4K to 8K (see table 4-20 for next higher assembly)		SDS	117638	Ref.
	. Core Diode Module Assembly		SDS	111549	3
	. Core Diode Module Assembly		SDS	111550	1
	. Memory Preamplifier		SDS	131633	5
	. Memory Switch A, HT26		SDS	123005	2
	. Memory Switch B, ST11		SDS	123006	16
	. Memory Driver, ST22		SDS	132159	1

Table 4-44. Memory Expansion Kit, 8K to 12K, Replaceable Parts

Fig. & Index No.	Description	Reference Designator	Manufacturer	Part No.	Qty
4-10	Memory Expansion Kit, 8K to 12K, (see table 4-20 for next higher assembly)		SDS	117639	Ref.
	. Core Diode Module Assembly		SDS	111549	3
	. Core Diode Module Assembly		SDS	111550	1
	. Memory Switch A, ST10		SDS	123005	8
	. Memory Driver, ST22		SDS	132159	1
	. Memory Preamplifier, HT26		SDS	131633	6

Table 4-45. Memory Expansion Kit, 12K to 16K, Replaceable Parts

Fig. & Index No.	Description	Reference Designator	Manufacturer	Part No.	Qty
4-10	Memory Expansion Kit, 12K to 16K (see table 4-20 for next higher assembly)		SDS	117640	Ref.
	. Core Diode Module Assembly		SDS	111549	3
	. Core Diode Module Assembly		SDS	111550	1
	. Memory Preamplifier, HT26		SDS	131633	5

Table 4-46. Two-Way Access, Replaceable Parts

Fig. & Index No.	Description	Reference Designator	Manufacturer	Part No.	Qty
4-10	Two-Way Access (see table 4-20 for next higher assembly)		SDS	129463	Ref.
	. Cable Receiver, AT10		SDS	123018	1
	. Cable Driver Receiver, AT11		SDS	123019	3
	. Buffered Latch No. 2a, FT37		SDS	130942	3
	. Logic Element with Inverter, LT20		SDS	124717	1
	. Logic Element with Buffer, LT21		SDS	126615	1

Table 4-47. Three-Way Access, Replaceable Parts

Fig. & Index No.	Description	Reference Designator	Manufacturer	Part No.	Qty
4-10	Three-Way Access (see table 4-20 for next higher assembly)		SDS	128125	Ref.
	. Cable Receiver, AT10		SDS	123018	1
	. Cable Driver Receiver, AT11		SDS	123019	3
	. Buffered Latch, FT37		SDS	130942	3
	. Logic Element with Inverter, LT20		SDS	124717	1
	. Logic Element with Buffer, LT21		SDS	126615	1

Table 4-48. Port Expander F Assembly, Replaceable Parts

Fig. & Index No.	Description	Reference Designator	Manufacturer	Part No.	Qty
4-14	Port Expander F Assembly* (see table 4-20 for next higher assembly)		SDS	130625	Ref.
	. Memory Cabinet, Frame No. 3		SDS	117264	1
	. Power Supply, PT16 (see SDS publication No. 901080 for parts breakdown)				
	. Cable Plug Module Assembly		SDS	133763	5
	. Cable Receiver Assembly, AT10		SDS	123018	4
	. Cable Driver Receiver Assembly, AT11		SDS	123019	16
	. Rejection Gate, printed wire assembly, AT16		SDS	126611	2
	. Gated Buffer, printed wire assembly, BT15		SDS	117389	1
	. Fast Buffer, printed wire assembly, BT22		SDS	127393	2
	. Buffered AND/OR Gate, printed wire assembly, BT24		SDS	130967	1
	. Buffered Latch No. 3, printed wire assembly, FT26		SDS	126856	1
	. Buffered Latch No. 2a, printed wire assembly, FT37		SDS	130942	14
	. Buffered Latch No. 3a, printed wire assembly, FT38		SDS	130952	7
	. Gated Inverter, printed wire assembly, IT15		SDS	117375	1
	. Gated Inverter, printed wire assembly, IT16		SDS	125264	6
	. Logic Element with inverter, printed wire assembly, LT20		SDS	124717	4
	. Logic Element with buffer, printed wire assembly, LT21		SDS	126615	4
	. Address Selector, printed wire assembly, ST14		SDS	123008	2
	. Terminator Module, printed wire assembly, XT10		SDS	116257	10
	. Cable Plug Module Assembly	P252-P253	SDS	133763-201	2
	. Cable Plug Module Assembly	P252-P253	SDS	133763-301	2
	. Cable Plug Module Assembly	P252-P253	SDS	133763-401	1

*Port expander F is the first port expander installed in a memory cabinet and is used to expand the first block of memory (frame 2) in the cabinet.

Table 4-49. Port Expander S Assembly, Replaceable Parts

Fig. & Index No.	Description	Reference Designator	Manufacturer	Part No.	Qty
4-14	Port Expander S Assembly* (see table 4-20 for next higher assembly)		SDS	130626	Ref.
	. Memory Port Expander S Assembly		SDS	133651	1
	. Cable Plug Module Assembly		SDS	133763	5
	. Rejection Gate, printed wire assembly, AT16		SDS	126611	2
	. Gate Buffer, printed wire assembly, BT15		SDS	117389	1
	. Fast Buffer, printed wire assembly, BT22		SDS	127393	2
	. Buffered AND/OR Gate, printed wire assembly, BT24		SDS	130967	1
	. Buffered Latch No. 3a, printed wire assembly, FT38		SDS	130952	7
	. Gated Inverter, printed wire assembly, IT15		SDS	117375	1
	. Logic Element with inverter, printed wire assembly, LT20		SDS	124717	4
	. Logic Element with inverter, printed wire assembly, LT21		SDS	126615	4
	. Address Selector, printed wire assembly, ST14		SDS	123008	2
	. Terminator Module, printed wire assembly, XT10		SDS	116257	3
	. Ribbon Cable, printed wire assembly, ZT45		SDS	133212-171	2
	. Cable Plug Modules, printed wire assembly	P252-P253	SDS	133763-601	3
	. Cable Plug Modules, printed wire assembly	P252-P253	SDS	133763-651	2

*Port expander S is the second port expander installed in memory cabinet and is used to expand the second block of memory (frame 1) in the cabinet.

Table 4-50. External Interface Feature, Replaceable Parts

Fig. & Index No.	Description	Reference Designator	Manufacturer	Part No.	Qty
4-8	External Interface Feature (see table 4-20 for next higher assembly)		SDS	137086	Ref.
	. Cable Driver Receiver Assembly, AT11			123019	4
	. Cable Driver Assembly, AT12			124629	1
	. Universal Flip-Flop Assembly, FT22			124713	6
	. Gate Expander No. 1 Assembly, GT11			124881	2
	. Inverter Matrix Assembly, IT13			117000	1

Table 4-51. External IOP Interface Feature, Replaceable Parts

Fig. & Index No.	Description	Reference Designator	Manufacturer	Part No.	Qty
4-8	External IOP Interface (see table 4-20 for next higher assembly)		SDS		Ref.
	. Printed Wire Board Assembly, AT13		SDS	125260	1

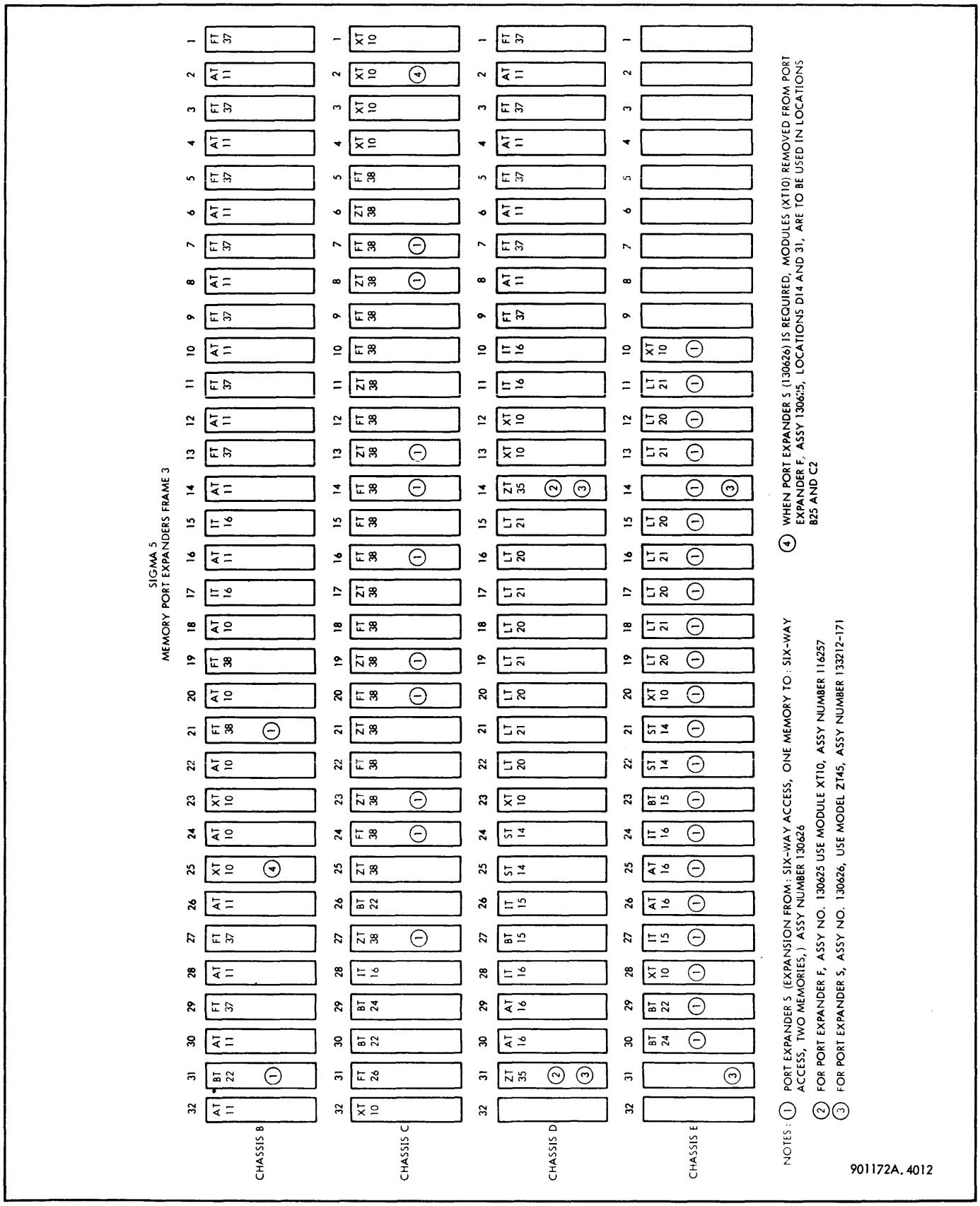


Figure 4-14. Assemblies, Memory Port Expanders, Frame 3

Table 4-52. Manufacturer Code Index

Code No.	Name	Address
1	Motorola Semiconductor Products, Inc.	P. O. Box 2953, Phoenix, Ariz. 85002
7	RCA, Electronic Components & Devices	415 S. 5th St., Harrison, N. J. 07029
8	Silicon Transistor Corp.	150 Glen Cove Rd., Carle Place, N. Y. 11514
20	Sangamo Electric Co.	Box 359, 1301 N. Eleventh St., Springfield, Ill. 62705
23	Sprague Electric Co.	481 Marshall St., N. Adams, Mass. 01248
25	General Electric Co., Capacitor Dept.	P. O. Box 158, Irmo, S. C. 29063
45	Dale Electronics, Inc.	1342 28th Avenue, Columbus, Neb. 68601
48	Littlelfuse, Inc.	800 E. Northwest Hwy., Des Plaines, Ill. 60016
49	Bussman Manufacturing Div. McGraw-Edison Co.	University at Jefferson, St. Louis, Mo. 63107
51	Cinch Manufacturing Co.	1026 S. Homan Avenue, Chicago, Ill. 60624
53	Ohmite Manufacturing Co.	3635 Howard St., Skokie, Ill. 60076
54	Cutler-Hammer, Inc.	321 N. 12th St., Milwaukee, Wisc. 53201
55	Centralab Electronics	900 E. Keefe Ave., Milwaukee, Wisc. 63201
56	Eldema Corp.	18435 Susana Rd., Compton, Calif. 90221
63	Transitron Electronic Corp.	168-182 Albion St., Wakefield, Mass. 01881
82	Elco Corp.	Maryland Rd. & Computer Ave., Willow Willow Grove, Md. 19090
83	Chicago Miniature Lamp Works	Dept. E, 4433 Ravenswood Ave. Chicago, Ill. 60640
84	General Electric Co., Miniature Lamp Dept.	Nela Park, Cleveland, Ohio 44112
104	Dialight Corp.	60 Stewart Ave,, Brooklyn, N. Y. 11237
106	Arrow-Hart & Hegeman Electric Co.	103 Hawthorne St., Hartford, Conn. 06106
107	Allen-Bradley Co.	1201 Second St., Milwaukee, Wisc. 53204
121	Astro Dynamics, Inc.	2nd Ave., Northwest Industrial Pk., Burlington, Mass.
139	Rotron Manufacturing Co.	Woodstock, N. Y. 12498
140	The Digitran Co.	855 S. Arroyo Pkwy., Pasadena, Calif. 91105

Table 4-52. Manufacturer Code Index (Cont.)

Code No.	Name	Address
145	Malco Manufacturing Co., Inc.	4025 W. Lake St., Chicago, Ill. 60624
156	Capitol Machine & Switch Co.	36 Balmforth St., Danbury, Conn. 06813
162	Honeywell, Micro Switch Div.	11 W. Spring St., Freeport, Ill. 61033
175	Ward Leonard Electric Co.	75 South St., Mt. Vernon, N. Y. 10550
194	P. R. Mallory & Co., Inc.	3029 E. Washington St., Indianapolis, Ind. 46206
203	Master Specialities Co.	15020 Figueroa, Gardena, Calif. 90247
204	Alco Electronic Products, Inc.	3 Wolcott Ave., Lawrence, Mass. 01843
208	Oak Manufacturing Co.	E. Crystal Lake Ave., Dept, EL, Crystal Lake, Ill. 60014
211	Westinghouse Electric Corp., Lamp Div.	MacArthur Blvd., Bloomfield, N. J. 07003
244	Hardwick, Hindle Products	Huntington, Ind. 46750
340	Bryant Electric	1421 State, Bridgeport, Conn. 06600
365	Harvey Hubbell, Inc.	Narvey Street & Boxtwick, Bridgeport, Conn.
376	C & K Components	103 Morse St., Newton, Mass. 02158
377	Standard Tool & Manufacturing Co.	738 Schuyler Ave., Lyndhurst, N. J.
378	Electric Parts Manufacturing Co., Inc.	508-10 25th St., Union City, N. J. 07087
381	Korry Manufacturing Co.	233 8th St., N., Seattle, Wash. 98109
382	Union Carbide	270 Park Avenue, N. Y., N. Y. 10017
383	Pass and Seymour, Inc.	Solvay Station, Syracuse, N. Y. 13209
384	Switchcraft, Inc.	5533 N. Elston Ave., Chicago, Ill. 60630
385	Lectrohm, Inc.	5560 Northwest Hwy, Chicago, Ill. 60600
387	Cycle-Dyne, Inc.	134-20 Jamaica Ave., Jamaica, N. Y. 11418