OPERATION AND MAINTENANCE INSTRUCTIONS: ASC-1X CENTRAL PROCESSOR (CP)

TEXAS INSTRUMENTS
INCORPORATED

ASC

OPERATION AND MAINTENANCE INSTRUCTIONS: ASC-1X CENTRAL PROCESSOR (CP)

Equipment Group
P.O. Box 2909
Austin, Texas 78767

TEXAS INSTRUMENTS
INCORPORATED

930020-2
July 1973

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

SECTION V. MAINTENANCE

SECTION VI. PARTS LISTING

SECTION VII. DIAGRAMS

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (Continued)

LIST OF TABLES

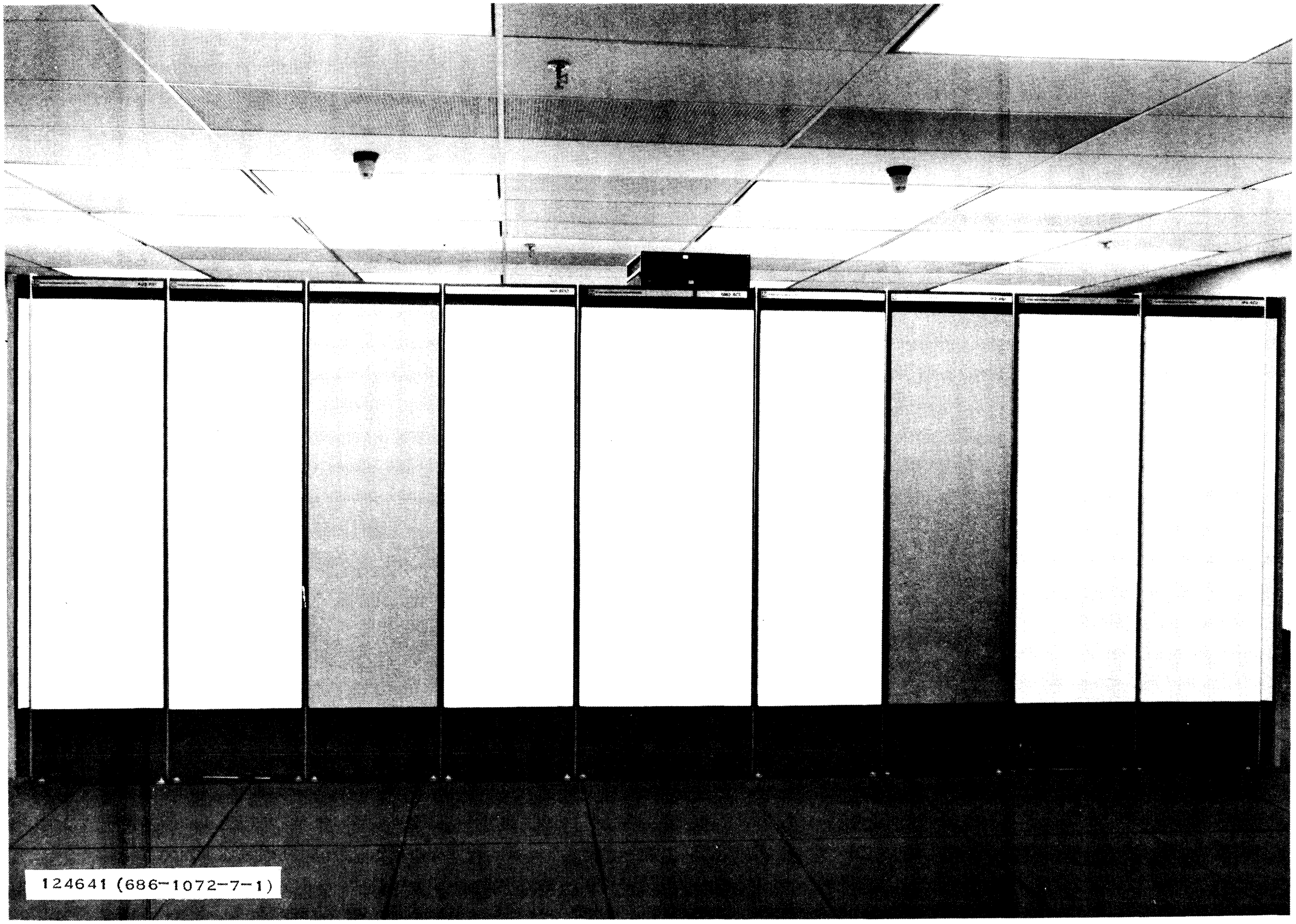


Figure 1-0. ASC Central Processor

SECTION I

GENERAL DESCRIPTION

1-1 GENERAL

This section describes the operation of the Central Processor (CP) of the Texas Instruments Advanced Scientific Computer (ASC). It includes a brief system overview of the ASC, a general functional block diagram description of the CP, a physical description of the CP, plus information about the instruction set and words used in the CP. Section 4 of this manual provides a detailed discussion of the CP theory of operation. Other useful charts and data are contained in the appendices to this manual. This manual applies to one-pipe CP configurations only.

1-2 PURPOSE

The ASC CP accesses program instructions from Central Memory, executes those instructions, and stores the results either within the CP or back into Central Memory. In performing this function, it also monitors program status to detect errors, branches and conflicts, and informs the Peripheral Processor if it is unable to continue a particular operation. The Peripheral Processor controls the selection of programs executed by the CP.

1-3 ASC SYSTEM OVERVIEW

Besides the CP, the ASC includes the following major units:

- Peripheral Processor (PP)
- Central Memory System (CM)
- System Clock
- Disc Storage System
- Magnetic Tape System
- Data Communications Channel
- Paper Peripheral Channels
- Operators Console
- Display Console
- Power System
- Maintenance System.

The relationship of these components is shown in figure 1-1. The CP interfaces directly with Central Memory for instruction and operand fetching, as well as for maintenance purposes. Initial programming sequences are determined by the PP, which also controls CP reaction to certain status conditions and calls. The CP, however, executes programs under its own control.

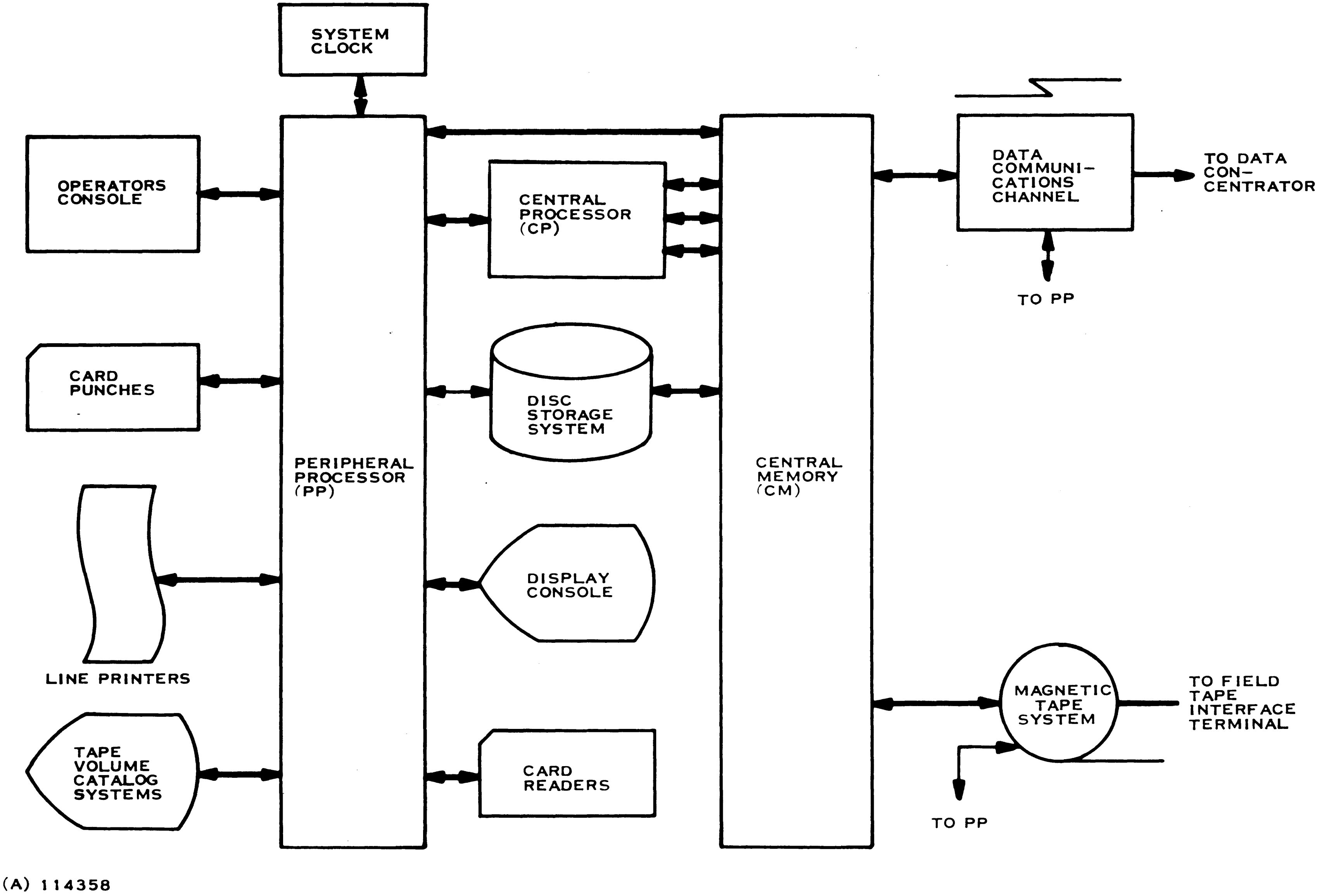


Figure 1-1. ASC Simplified Block Diagram

1-4 FUNCTIONAL DESCRIPTION

Three functional units comprise the Central Processor of the ASC: Instruction Processing Unit (IPU), Memory Buffer Unit (MBU), and Arithmetic Unit (AU). Each unit is a layered pipeline processor utilizing small, decentralized controllers. The IPU obtains instructions from Central Memory and develops operand addresses. The MBU performs memory fetch and data buffering functions for acquisition of operands. Three buffer levels in the MBU ensure a continuous data stream for vector operations. The MBU may also receive operands directly from the IPU. In either case, the MBU transfers the proper operands to the AU concurrently with control instructions for processing the operands. The AU performs the designated partial steps to satisfy the requested operation for the two operands. The result returns to the Register File in the IPU, or to the MBU for storage into Central Memory or for reprocessing as a new operand. The layered pipeline construction of the CP allows an instruction or group of two operands to be processed concurrently at each level of the pipeline, unless the layer is reserved by a previous operand or instruction. Figure 1-2 provides a block diagram of the CP. The following paragraphs briefly describe the function of each component in the diagram. Refer to section 4 of this manual for a more detailed discussion of CP operation.

1-5 ADDRESS REGISTERS AND CONTROL

Four address registers control the acquisition of instruction word octets (8-word groups) from Central Memory. These registers select the proper instruction word for processing, call up a new octet while the current one is being processed, and provide for branch address acquisition. During indirect addressing, the output of the Address Modification network updates the Output Address Register in this circuit for each new address developed by the network until the terminal effective address is reached. The other registers maintain the program address so that the program resumes when the effective address is reached.

1-6 REGISTER FILE

The Register File is a memory source contained within the IPU. These registers are loaded by program instructions with data from either memory or the AU output. The file consists of six sets of eight 32-bit registers (six octets). Each area in the file has a primary function, such as base addresses for developing effective addresses (15 words), general arithmetic use (16 words), seven index registers, and eight vector parameter registers to define the scope of a specific vector instruction. They may, however, be used for other processes.

1-7 INSTRUCTION FILES

Two instruction files, each containing one octet (eight words), supply a continuous source of instructions to the Instruction Register. The Address Registers and Control block controls loading and selection from these registers. It first loads one file and begins drawing instructions from the octet in that file. Address Control then loads the second file while the first one empties. Consecutive addresses supply a smooth transition from one file to the next. During indirect addressing, the effective address of an instruction from the

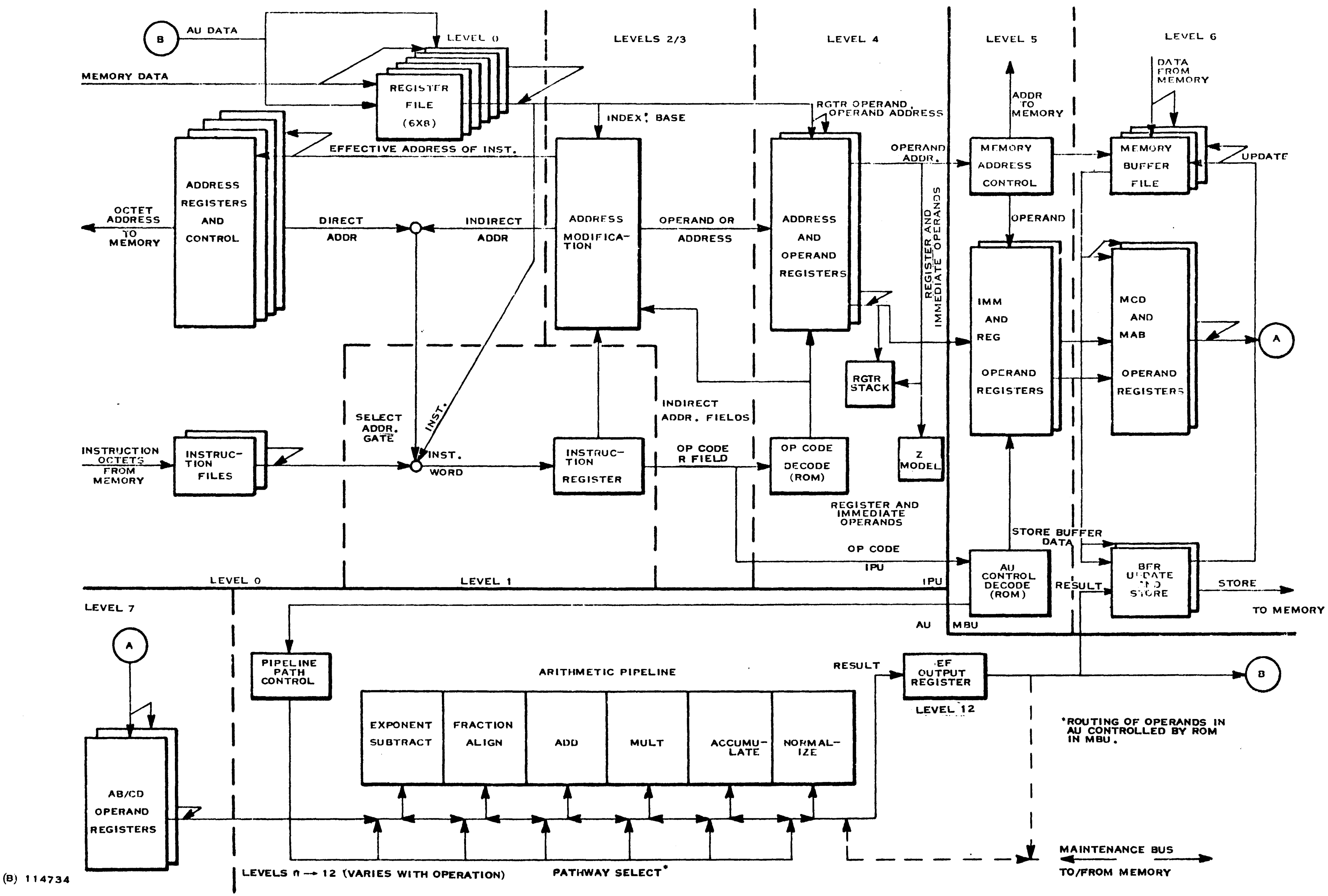


Figure 1-2. Central Processor Block Diagram

Address Modification block selects the output from the instruction files if the address is currently in the files.

1-8 INSTRUCTION REGISTER

The Instruction Register receives the selected word from the instruction file and holds it for processing. Depending on the instruction format, the register may contain address bits, address modifiers, and operand and/or an operation code. The register output drives instruction decode and address generation networks.

1-9 ADDRESS MODIFICATION

When the Instruction Register specifies either direct or indexed addressing, the Address Modification block performs the operations required to generate a new address. This block provides for base address (from Register File) plus displacement modification and/or addition of the contents of one of the seven index registers in the Register File. The circuit permits direct or indirect addressing with or without modification, or the development of an immediate operand. Operands, direct operand addresses, and terminal operand addresses transfer to the MBU to provide operands for the AU. If an indirect address develops, it returns to the Output Address Register to retrieve a new instruction word for further address generation. The modification hardware includes input registers for indexing, base address and displacement, an adder, plus a result holding and output register.

1-10 ADDRESS AND OPERAND REGISTERS

These two registers are the IPU output registers. They provide the MBU with either two operands, one operand and an operand address, or just one address.

1-11 IMM/REG REGISTERS

The Immediate (IMM) and Register (REG) Operand Registers receive operands from the IPU. During vector initialization, the IMM Register also transmits the vector parameters to the MBU Registers to set up the beginning vector conditions. Once a vector operation begins, neither of these registers is used until the next operation begins. Control signals generated within the MBU transfer data that is in these registers to the output registers of the MBU during scalar operations.

1-12 MEMORY ADDRESS CONTROL

This circuit supplies addresses to memory for storing results from the AU vector and store operations and for accessing new operand octets from memory for input to the AU. During scalar operations, operand addresses are supplied from the IPU. If the desired operand is already in the Memory Buffer File, the IPU sends only a 4-bit address to select the output from one of the file registers. If the operand is not in the buffer file, the IPU sends a full 21-bit address to fetch the octet containing the operand from Central Memory and load it into the buffer file before transferring the operand to the output register. During vector operations, Memory Address Control generates the address of each octet in the vector after the address is initially loaded by the IPU.

1-13 MEMORY BUFFER FILE

The Memory Buffer File consists of six octet buffers plus an octet receiver/synchronizer register. The buffers are arranged in two three-stage buffers with the output of the final stage available to the output registers. Inputs to the buffers may enter the final file to bypass the delay in the buffering sequence. During scalar operations, Memory Address Control can select the output from either buffer and transfer it to the MCD Operand Register. During vector operations each buffer set supplies a stream of operands to one of the MAB/MCD Operand Registers. Either buffer set may be modified by the result output from the Arithmetic Pipeline (update) during scalars.

1-14 MAB/MCD OPERAND REGISTERS

These registers supply two operands simultaneously to the AU for processing. The MAB Register receives register operands from the REG Register during scalar operations, and vector operands from the buffer file during vector operations. The MCD Register receives either immediate operands from the IMM Register or operands from either set of the buffer file during scalar operations. During vector operations the buffer file supplies a stream of operands to the MCD Register.

1-15 AU CONTROL DECODE

The AU Control Decode is a Read Only Memory (ROM) that designates to the AU which processes must be performed to accomplish the function specified by the Op Code. The decode circuit also supplies control signals to aid in selection of operands for the MAB/MCD registers.

1-16 BUFFER UPDATE AND STORE

The buffer update provides temporary retention of an octet of AU output. This octet may change the contents of the buffer file, or may be stored into Central Memory when the AU begins to produce results for a new octet.

1-17 AB/CD OPERAND REGISTERS

These registers are the input phase to the arithmetic pipeline. They receive two operands from the MBU and transfer them to the pipeline when the pipeline segment that performs the first operational step becomes available. Other inputs to these registers come from within the AU to provide a feedback path.

1-18 PIPELINE PATH CONTROL

This circuit follows the directions of the AU Control ROM in the MBU to perform the gating and sequencing functions required to develop a complete process in the pipeline.

1-19 ARITHMETIC PIPELINE

The Arithmetic Pipeline is a segmented arithmetic processor whose sequence is determined by the MBU ROM signals. Six segments of the pipe perform independent operations on up to six different sets of operands simultaneously. Each segment is a basic function that, combined in a specific order with other segments, performs arithmetic operations from scalar addition to complex vector operations on both fixed and floating point operands.

1-20 EF OUTPUT REGISTER

The EF Output Register receives a result from any segment of the pipeline, except the multiplication segment (output of multiplier is two partial products that must be added to produce a result). The output of this register may return to the Register File in the IPU (scalar operations), may update the data in the Memory Buffer File, or may be stored in memory (vectors and store operations).

1-21 GENERAL CHARACTERISTICS

Table 1-1 lists some of the general characteristics of the ASC Central Processor.

1-22 CP INSTRUCTION SET

The ASC Central Processor performs scalar and vector operations through a powerful array of instructions. The instruction set includes Load and Store functions, arithmetic scalar operations, scalar logical instructions, and branching capabilities. Two special instructions, VECT and VECTL, expand the ASC instructions into the vector mode by loading a new set of parameters into the IPU from the Vector Parameter File. The set of vector parameters includes a vector operation code. The function of the vector operation is defined by an additional set of vector instructions that can be loaded only through this vector mode. Table 1-2 lists the instructions in the normal ASC instruction set with their mnemonic code and operation code; figure 1-3 supplies a mapping of scalar Op Codes. Table 1-3 and figure 1-4 contain similar information for the vector mode instructions. Refer to the ASC programming manuals for a more detailed explanation of the uses of each instruction.

Table 1-1. Central Processor General Characteristics

Item	Characteristic
Construction	Layered pipeline
Word Size	16 bits (halfword) -fixed point only 32 bits (single word)-fixed or floating point 64 bits (doubleword) -floating point only
Instruction word size	32 bits (8 Op Code, 4 R-field, 4 T-field, 4 M-field, 12 N-field)
Memory address size:	
Octet	21 bits (sent to CM)
Word	24 bits (internal to CP)
Memory transfer size	1 octet (256 bits)
Number of memory paths	3: IPU (instruction fetch), MBU (operand fetch/store) AU (maintenance - Load/Store Details)
Operation Modes	2: Scalar and Vector
Control:	
Initiate/Terminate	Through CR File in the Peripheral Processor
Operating	Individual pipe level controllers in CP
CP Clock Period	65 nanoseconds
Processing Rate	1 result per clock as an upper limit

Table 1-2. Scalar Instruction Set

Mnemonic Code	Instruction	Operation Code	Operand Format
ST	Store arithmetic register, single length	24	R,@N,X
ST	Store base register, single length	28	R,@N,X
ST	Store index register or vector parameter register, single length	2C	R,@N,X
STLL	Store arithmetic left halfword into memory left halfword, indexed	25	R,@N,X
STRL	Store arithmetic register right half into memory left half, indexed	26	R,@N,X
STRR	Store arithmetic register right half into memory right half, indexed	2D	R,@N,X
STLR	Store arithmetic register left half into memory right half, indexed	29	R,@N,X
SPS	Store program status word	22	@N,X
STD	Store arithmetic register, double length	27	R,@N,X
STZ	Store zero, single length	20	@N,X
STZH	Store zero, half length	21	@N,X
STZD	Store zero, double length	23	@N,X
STN	Store negative, single length	34	R,@=N,X
STNH	Store negative, half length	35	R,@=N,X
STNF	Store negative, floating point	36	R,@=N,X
STND	Store negative, double length	37	R,@=N,X
STO	Store ones complement	2E	R,@=N,X
STOH	Store ones complement, half length	2A	R,@=N,X
STF	Store base register file, registers 1-7_{16}, M=0	2B	M,@N,X
STF	Store base register file, registers 8-F_{16}, M=1	2B	M,@N,X

Table 1-2. Scalar Instruction Set (Continued)

Mnemonic Code	Instruction	Operation Code	Operand Format
STF	Store arithmetic register file, registers $10\text{-}17_{16}$, M=2	2B	M,@N,X
STF	Store arithmetic register file, registers $18\text{-}1F_{16}$, M=3	2B	M,@N,X
STF	Store index register file, registers $20\text{-}27_{16}$, M=4	2B	M,@N,X
STF	Store vector parameter register file, registers $28\text{-}2F_{16}$, M=5	2B	M,@N,X
STFM	Store all register files, registers $1\text{-}2F_{16}$	2F	@N,X
L	Load arithmetic register single length word	14	R,@=N,X
L	Load base register single length	18	R,@=N,X
L	Load index register or vector parameter register single length	1C	R,@=N,X
LLL	Load arithmetic register left halfword from memory left halfword, indexed	15	R,@=N,X
LRL	Load memory left halfword, indexed, into arithmetic register right halfword	10	R,@=N,X
LRR	Load memory right halfword, indexed, into arithmetic register right halfword	1D	R,@=N,X
LLR	Load memory right halfword, indexed, into arithmetic register left halfword	19	R,@=N,X
LD	Load arithmetic register double length word	17	R,@=N,X
LI	Load immediate into arithmetic register single length	54	R,I,X
LI	Load immediate into index register, or vector parameter register single length	5C	R,I,X
LIH	Load immediate into arithmetic register half length	55	R,I,X
LM	Load magnitude fixed point single length - arithmetic register	3C	R,@=N,X

Table 1-2. Scalar Instruction Set (Continued)

Mnemonic Code	Instruction	Operation Code	Operand Format
LMH	Load magnitude fixed point half length - arithmetic register	3D	R,@=N,X
LMF	Load magnitude floating point single length - arithmetic register	3E	R,@=N,X
LMD	Load magnitude floating point double length - arithmetic register	3F	R,@=N,X
LN	Load negative fixed point single length (load twos complement) arithmetic register	30	R,@=N,X
LNH	Load negative fixed point half length - arithmetic register	31	R,@=N,X
LNF	Load negative floating point single length - arithmetic register	32	R,@=N,X
LND	Load negative floating point double length - arithmetic register	33	R,@=N,X
LNM	Load negative magnitude fixed point single length - arithmetic register	38	R,@=N,X
LNMH	Load negative magnitude fixed point half length - arithmetic register	39	R,@=N,X
LNMF	Load negative magnitude floating point single length - arithmetic register	3A	R,@=N,X
LNMD	Load negative magnitude floating point double length - arithmetic register	3B	R,@=N,X
LF	Load base register file, registers $1\text{-}7_{16}$, M=0	1B	M,@N,X
LF	Load base register file, registers $8\text{-}F_{16}$, M=1	1B	M,@N,X
LF	Load arithmetic register file, registers $10\text{-}17_{16}$, M=2	1B	M,@N,X
LF	Load arithmetic register file, registers $18\text{-}1F_{16}$, M=3	1B	M,@N,X
LF	Load index register file, registers $20\text{-}27_{16}$, M=4	1B	M,@N,X

Table 1-2. Scalar Instruction Set (Continued)

Mnemonic Code	Instruction	Operation Code	Operand Format
LF	Load vector parameter register file, registers 28-$2F_{16}$, M=5	1B	M,@N,X
LFM	Load all register files	1F	@N,X
XCH	Exchange - arithmetic register	1A	R,@N,X
LAM	Load arithmetic mask	12	@=N,X
LAC	Load arithmetic exception condition	13	@=N,X
LLA	Load look ahead	16	I
LO	Load arithmetic register with ones complement, single length	1E	R,@=N,X
A	Add to arithmetic register, fixed point, single length	40	R,@=N,X
A	Add to base register, fixed point, single length	60	R,@=N,X
A	Add to index or vector parameter register, fixed point, single length	62	R,@=N,X
AI	Add immediate to arithmetic register, fixed point, single length	50	R,I,X
AI	Add immediate to base register, fixed point, single length	70	R,I,X
AI	Add immediate to index or vector parameter register, fixed point, single length	72	R,I,X
AH	Add fixed point, half length - arithmetic register	41	R,@=N,X
AIH	Add immediate fixed point, half length - arithmetic register	51	R,I,X
AF	Add floating point, single length - arithmetic register	42	R,@=N,X
AFD	Add floating point, double length - arithmetic register	43	R,@=N,X

Table 1-2. Scalar Instruction Set (Continued)

Mnemonic Code	Instruction	Operation Code	Operand Format
AM	Add magnitude fixed point, single length - arithmetic register	44	R,@=N,X
AMH	Add magnitude fixed point, half length - arithmetic register	45	R.@=N,X
AMF	Add magnitude floating point, single length - arithmetic register	46	R,@=N,X
AMFD	Add magnitude floating point, double length - arithmetic register	47	R,@=N,X
S	Subtract fixed point, single length - arithmetic register	48	R,@=N,X
SI	Subtract immediate fixed point, single length - arithmetic register	58	R,I,X
SH	Subtract fixed point, half length - arithmetic register	49	R,@=N,X
SIH	Subtract immediate fixed point, half length - arithmetic register	59	R,I,X
SF	Subtract floating point, single length - arithmetic register	4A	R,@=N,X
SFD	Subtract floating point, double length - arithmetic register	4B	R,@=N,X
SM	Subtract magnitude fixed point, single length - arithmetic register	4C	R,@=N,X
SMH	Subtract magnitude fixed point, half length - arithmetic register	4D	R,@=N,X
SMF	Subtract magnitude fixed point, half length - arithmetic register	4E	R,@=N,X
SMFD	Subtract magnitude floating point, double length - arithmetic register	4F	R,@=N,X
M	Multiply fixed point, single length - arithmetic register	6C	R,@=N,X

Table 1-2. Scalar Instruction Set (Continued)

Mnemonic Code	Instruction	Operation Code	Operand Format
M	Multiply base register	68	R,@=N,X
M	Multiply index or vector parameter register	6A	R,@=N,X
MI	Multiply immediate fixed point, single length - arithmetic register	7C	R,I,X
MI	Multiply immediate to base register	78	R,I,X
MI	Multiply immediate to index or vector parameter register	7A	R,I,X
MH	Multiply fixed point, half length - arithmetic register	6D	R,@=N,X
MIH	Multiply immediate fixed point, half length - arithmetic register	7D	R,I,X
MF	Multiply floating point, single length - arithmetic register	6E	R,@=N,X
MFD	Multiply floating point, double length - arithmetic register	6F	R,@=N,X
D	Divide fixed point, single length - arithmetic register	64	R,@=N,X
DI	Divide immediate fixed point, single length - arithmetic register	74	R,I,X
DH	Divide fixed point, half length - arithmetic register	65	R,@=N,X
DIH	Divide immediate fixed point, half length - arithmetic register	75	R,I,X
DF	Divide floating point, single length - arithmetic register	66	R,@=N,X
DFD	Divide floating point, double length - arithmetic register	67	R,@=N,X
AND	AND - arithmetic register	E0	R,@=N,X

Table 1-2. Scalar Instruction Set (Continued)

Mnemonic Code	Instruction	Operation Code	Operand Format
ANDI	Immediate AND - arithmetic register	F0	R,I,X
OR	OR - arithmetic register	E4	R,@=N,X
ORI	Immediate OR - arithmetic register	F4	R,I,X
XOR	Exclusive OR - arithmetic register	E8	R,@=N,X
XORI	Immediate Exclusive OR - arithmetic register	F8	R,I,X
EQC	Equivalence - arithmetic register	EC	R,@=N,X
EQCI	Immediate equivalence - arithmetic register	FC	R,I,X
ANDD	AND - arithmetic register (double length)	E1	R,@=N,X
ORD	OR - arithmetic register (double length)	E5	R,@=N,X
XORD	Exclusive OR - arithmetic register (double length)	E9	R,@=N,X
EQCD	Equivalence - arithmetic register (double length)	ED	R,@=N,X
SA	Arithmetic shift, fixed point, single length - arithmetic register	C0	R,I,X
SAH	Arithmetic shift, fixed point, half length - arithmetic register	C1	R,I,X
SAD	Arithmetic shift, fixed point, double length - arithmetic register	C3	R,I,X
SL	Logical shift, single length - arithmetic register	C4	R,I,X
SLH	Logical shift, half length - arithmetic register	C5	R,I,X
SLD	Logical shift, double length - arithmetic register	C7	R,I,X

Table 1-2. Scalar Instruction Set (Continued)

Mnemonic Code	Instruction	Operation Code	Operand Format
SC	Circular shift, single length - arithmetic register	CC	R,I,X
SCH	Circular shift, half length - arithmetic register	CD	R,I,X
SCD	Circular shift, double length - arithmetic register	CF	R,I,X
RVS	Bit reversal, single length - arithmetic register	C6	R,I,X
C	Compare fixed point, single length - arithmetic register	C8	R,@=N,X
C	Compare index register, single length	CE	R,@=N,X
CI	Compare immediate, fixed point, single length - arithmetic register	D8	R,I,X
CI	Compare immediate, index register, single length	DE	R,I,X
CH	Compare fixed point, half length - arithmetic register	C9	R,@=N,X
CIH	Compare immediate, fixed point, half length - arithmetic register	D9	R,I,X
CF	Compare floating point, single length - arithmetic register	CA	R,@=N,X
CFD	Compare floating point, double length - arithmetic register	CB	R,@=N,X
CAND	Compare logical AND - arithmetic register (single length)	E2	R,@=N,X
CANDI	Compare immediate logical AND - arithmetic register (single length)	F2	R,I,X
COR	Compare logical OR, single length - arithmetic register	E6	R,@=N,X
CORI	Compare immediate logical OR, single length - arithmetic register	F6	R,I,X

Table 1-2. Scalar Instruction Set (Continued)

Mnemonic Code	Instruction	Operation Code	Operand Format
CANDD	Compare logical AND, double length - arithmetic register	E3	R,@=N,X
CORD	Compare logical OR, double length - arithmetic register	E7	R,@=N,X
IBZ	Increment, test, and branch on zero - arithmetic register	88	R,@=N,X
IBZ	Increment, test index, and branch on zero	8C	R,@=N,X
IBNZ	Increment, test and branch on non-zero - arithmetic register	89	R,@=N,X
IBNZ	Increment, test index, and branch on non-zero	8D	R,@=N,X
DBZ	Decrement, test, and branch on zero - arithmetic register	8A	R,@=N,X
DBZ	Decrement, test index, and branch on zero	8E	R,@=N,X
DBNZ	Decrement, test, and branch on non-zero - arithmetic register	8B	R,@=N,X
DBNZ	Decrement, test index, and branch on non-zero	8F	R,@=N,X
ISE	Increment, test, and skip on equal - arithmetic register	80	R,@=N,X
ISNE	Increment, test, and skip on not equal - arithmetic register	81	R,@=N,X
DSE	Decrement, test, and skip on equal - arithmetic register	82	R,@=N,X
DSNE	Decrement, test, and skip on not equal - arithmetic register	83	R,@=N,X
BCLE	Branch on arithmetic register less than or equal to	84	R,R,N

Table 1-2. Scalar Instruction Set (Continued)

Mnemonic Code	Instruction	Operation Code	Operand Format
BCLE	Branch on index less than or equal to	86	R,R,N
BCG	Branch on arithmetic register greater than	85	R,R,N
BCG	Branch on index greater than	87	R,R,N
PSH	Push word - arithmetic register	93	R,@N,X
PUL	Pull word - arithmetic register	97	R,@N,X
MOD	Modify - arithmetic register	9F	R,@N,X
BLB	Branch and load register with PC	98	R,@=N,X
BLX	Branch and load index register or vector parameter register	99	R,@=N,X
LEA	Load effective address - index register	56	R,@=N,X
LEA	Load effective address into base register	52	R,@=N,X
INT	Interpret - arithmetic register	92	R,@=N,X
XEC	Execute	96	@=N,X
FLFX	Convert floating point single length to fixed point single length - arithmetic register	A0	R,@N,X
FLFH	Convert floating point single length to fixed point half length - arithmetic register	A1	R,@N,X
FDFX	Convert floating point double length fixed point single length	A2	R,@N,X
FXFL	Convert fixed point single length to floating point single length	A8	R,@N,X
FXFD	Convert fixed point single length to floating point double length	AA	R,@N,X
FHFL	Convert fixed point half length to floating point single length	A9	R,@N,X

Table 1-2. Scalar Instruction Set (Continued)

Mnemonic Code	Instruction	Operation Code	Operand Format
FHFD	Convert fixed point half length to floating point double length	AB	R,@N,X
NFX	Normalize fixed point single length - arithmetic register	AC	R,@N,X
NFH	Normalize fixed point half length - arithmetic register	AD	R,@N,X

Mnemonic Code	Instruction	Operation Code	Assembler Supplies R Field	Operand Format
MCP	Monitor call and proceed	90		I,X
MCW	Monitor call and wait	94		I,X
VECT	Vector	B0	R = 1	@N,X
VECTL	Vector after loading vector file	B0	R = 0	@N,X

<u>Compare Code Branch Operation Code = 91</u>

BCC	Branch on compare code	91		M,@=N,X
NOP	Take next instruction	91	R = 0	@=N,X

<u>Comment</u>: Execution of data values or indirect address constants will have the effect of a no-operation if the first four bits of the word (operation code) are zeros.

BE	$(R) = (\alpha)$	91	R = 1	@=N,X
BG	$(R) \quad (\alpha)$	91	R = 2	@=N,X
BGE	$(R) \quad (\alpha)$	91	R = 3	@=N,X
BL	$(R) \quad (\alpha)$	91	R = 4	@=N,X
BLE	$(R) \quad (\alpha)$	91	R = 5	@=N,X
BNE	$(R) \neq (\alpha)$	91	R = 6	@=N,X
B	Unconditional branch		R = 7	@=N,X

Table 1-2. Scalar Instruction Set (Continued)

Mnemonic Code	Instruction	Operation Code	Assembler Supplies R Field	Operand Format
Logical Branch Operation Code = 91				
BCZ	All bits are zero	91	R = 1	@=N,X
BCO	All bits are one	91	R = 2	@=N,X
BCNM	Not mixed	91	R = 3	@=N,X
BCM	Mixed zeros and ones	91	R = 4	@=N,X
BCNO	Not all ones	91	R = 5	@=N,X
BCNZ	Not all zeros	91	R = 6	@=N,X
Result Code Branch Operation Code = 95				
BRC	Branch on result code	95		M,@=N,X
BZ	(R) = 0	95	R = 1	@=N,X
BPL	(R) > 0	95	R = 2	@=N,X
BZP	(R) ≥ 0	95	R = 3	@=N,X
BMI	(R) < 0	95	R = 4	@=N,X
BZM	(R) ≤ 0	95	R = 5	@=N,X
BNZ	(R) ≠ 0	95	R = 6	@=N,X
Logical Result Branch Operation Code = 95				
BRZ	All bits are zero	95	R = 1	@=N,X
BRO	All bits are one	95	R = 2	@=N,X
BRNM	Not mixed	95	R = 3	@=N,X
BRM	Mixed zeros and ones	95	R = 4	@=N,X
BRNO	Not all ones	95	R = 5	@=N,X
BRNZ	Not all zeros	95	R = 6	@=N,X

Table 1-2. Scalar Instruction Set (Continued)

Mnemonic Code	Instruction	Operation Code	Assembler Supplies R Field	Operand Format
Arithmetic Exception Branch Operation Code = 9D				
BAE	Branch on arithmetic exception	9D		M,@=N,X
BU	Floating point EXP underflow	9D	R = 1	@=N,X
BO	Floating point EXP overflow	9D	R = 2	@=N,X
BUO	Floating point EXP underflow or overflow	9D	R = 3	@=N,X
BX	Fixed point overflow	9D	R = 4	@=N,X
BXU	Fixed point overflow or floating EXP underflow	9D	R = 5	@=N,X
BXO	Fixed point overflow or floating EXP overflow	9D	R = 6	@=N,X
BXUO	Fixed point overflow or floating EXP overflow or underflow	9D	R = 7	@=N,X
BD	Divide check	9D	R = 8	@=N,X
BDU	Divide check or floating point EXP underflow	9D	R = 9	@=N,X
BDO	Divide check or floating point EXP overflow	9D	R = A	@=N,X
BDUO	Divide check or floating point EXP underflow or overflow	9D	R = B	@=N,X
BDX	Divide check or fixed point overflow	9D	R = C	@=N,X
BDXU	Divide check or fixed point overflow or floating point EXP underflow	9D	R = D	@=N,X
BDXO	Divide check or fixed point overflow or floating point EXP overflow	9D	R = E	@=N,X
BDXUO	Divide check or fixed point overflow or floating point EXP overflow or underflow	9D	R = F	@=N,X

Table 1-2. Scalar Instruction Set (Continued)

Mnemonic Code	Instruction	Operation Code	Assembler Supplies R Field	Operand Format
Branch on Execute Condition Operation Code = 9C				
BXEC	Branch on Execute branch condition true	9C	R = 1 or odd	@N,X

OP BITS 0–3

OP BITS 4–7

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	↑	LRL	STZ	LN	A	AI	A	AI	ISE	MCP	FLFX	VECT	SA		AND	ANDI
1			STZH	LNH	AH	AIH			ISNE	BCC	FLFH		SAH		ANDD	
2		LAM	SPS	LNF	AF	LEA	A	AI	DSE	INT	FDFX				CAND	CANDI
3		LAC	STZD	LND	AFD				DSNE	PSH			SAD		CANDD	
4		L	ST	STN	AM	LI	D	DI	BCLE	MCW			SL		OR	ORI
5		LLL	STLL	STNH	AMH	LIH	DH	DIH	BCG	BRC			SLH		ORD	
6	N	LLA	STRL	STNF	AMF	LEA	DF		BCLE	XEC			RVS		COR	CORI
7	O	LD	STD	STND	AMFD		DFD		BCG	PUL			SLD		CORD	
8	P	L	ST	LNM	S	SI	M	MI	IBZ	BLB	FXFL		C	CI	XOR	XORI
9		LLR	STLR	LNMH	SH	SIH			IBNZ	BLX	FHFL		CH	CIH	XORD	
A		XCH	STOH	LNMF	SF		M	MI	DBZ		FXFD		CF			
B		LF	STF	LNMD	SFD				DBNZ		FHFD		CFD			
C		L	ST	LM	SM	LI	M	MI	IBZ	BXEC	NFX		SC		EQC	EQCI
D		LRR	STRR	LMH	SMH		MH	MIH	IBNZ	BAE	NFH		SCH		EQCD	
E		LO	STO	LMF	SMF		MF		DBZ				C	CI		
F	↓	LFM	STFM	LMD	SMFD		MFD		DBNZ	MOD			SCD			

Figure 1-3. Scalar Op Code Map

Table 1-3. Vector Instruction Set

Mnemonic Code	Function	Operation Code
VA	Vector Add, fixed point, single length	40
VAH	Vector Add, fixed point, half length	41
VAF	Vector Add, floating point, single length	42
VAFD	Vector Add, floating point, double length	43
VAM	Vector Add magnitude, fixed point, single length	44
VAMH	Vector Add magnitude, fixed point, half length	45
VAMF	Vector Add magnitude, floating point, single length	46
VAMFD	Vector Add magnitude, floating point, double length	47
VS	Vector Subtract, fixed point, single length	48
VSH	Vector Subtract, fixed point, half length	49
VSF	Vector Subtract, floating point, single length	4A
VSFD	Vector Subtract, floating point, double length	4B
VSM	Vector Subtract magnitude, fixed point, single length	4C
VSMH	Vector Subtract magnitude, fixed point, half length	4D
VSMF	Vector Subtract magnitude, floating point, single length	4E
VSMFD	Vector Subtract magnitude, floating point, double length	4F
VM	Vector Multiply, fixed point, single length	6C
VMH	Vector Multiply, fixed point, half length	6D
VMF	Vector Multiply, floating point, single length	6E
VMFD	Vector Multiply, floating point, double length	6F
VDP	Vector dot product, fixed point, single length	68
VDPH	Vector dot product, fixed point, half length	69

Table 1-3. Vector Instruction Set (Continued)

Mnemonic Code	Function	Operation Code
VDPF	Vector dot product, floating point, single length	6A
VDPFD	Vector dot product, floating point, double length	6B
VD	Vector Divide, fixed point, single length	64
VDH	Vector Divide, fixed point, half length	65
VDF	Vector Divide, floating point, single length	66
VDFD	Vector Divide, floating point, double length	67
VSA	Vector Shift arithmetic, fixed point, single length	C0
VSAH	Vector Shift arithmetic, fixed point, half length	C1
VSAD	Vector Shift arithmetic, fixed point, double length	C3
VSL	Vector Shift logical, single length	C4
VSLH	Vector Shift arithmetic, half length	C5
VSLD	Vector Shift arithmetic, double length	C7
VSC	Vector Shift circular, single length	CC
VSCH	Vector Shift circular, half length	CD
VSCD	Vector Shift circular, double length	CF
VAND	Vector logical AND, single length	E0
VANDD	Vector logical AND, double length	E1
VOR	Vector logical OR, single length	E4
VORD	Vector logical OR, double length	E5
VXOR	Vector logical Exclusive OR, single length	E8
VXORD	Vector logical Exclusive OR, double length	E9
VEQC	Vector logical Equivalence, single length	EC
VEQCD	Vector logical Equivalence, double length	ED

Table 1-3. Vector Instruction Set (Continued)

Mnemonic Code	Function	Operation Code
VL	Vector search for largest arithmetic element, fixed point, single length	50
VLH	Vector search for largest arithmetic element, fixed point, half length	51
VLF	Vector search for largest arithmetic element, floating point, single length	52
VLFD	Vector search for largest arithmetic element, floating point, double length	53
VLM	Vector search for largest magnitude, fixed point, single length	54
VLMH	Vector search for largest magnitude, fixed point, half length	55
VLMF	Vector search for largest magnitude, floating point, single length	56
VLMFD	Vector search for largest magnitude, floating point, double length	57
VSS	Vector search for smallest arithmetic element, fixed point, single length	58
VSSH	Vector search for smallest arithmetic element, fixed point, half length	59
VSSF	Vector search for smallest arithmetic element, floating point, single length	5A
VSSFD	Vector search for smallest arithmetic element, floating point, double length	5B
VSSM	Vector search for smallest magnitude, fixed point, single length	5C
VSSMH	Vector search for smallest magnitude, fixed point, half length	5D
VSSMF	Vector search for smallest magnitude, floating point, single length	5E
VSSMFD	Vector search for smallest magnitude, floating point, double length	5F

Table 1-3. Vector Instruction Set (Continued)

Mnemonic Code	Function	Operation Code
VC	Vector arithmetic comparison, fixed point, single length	D0
VCH	Vector arithmetic comparison, fixed point, half length	D1
VCF	Vector arithmetic comparison, floating point, half length	D2
VCFD	Vector arithmetic comparison, floating point, double length	D3
VCAND	Vector Logical Comparison using AND, single length	E2
VCANDD	Vector Logical Comparison using AND, double length	E3
VCOR	Vector Logical Comparison using OR, single length	E6
VCORD	Vector Logical Comparison using OR, double length	E7
VMG	Vector Merge, single words	D8
VMGH	Vector Merge, halfwords	D9
VMGD	Vector Merge, doublewords	DB
VO	Vector order single words, fixed point	D4
VOH	Vector order halfwords, fixed point	D5
VOF	Vector order single words, floating point	D6
VOFD	Vector order doublewords, floating point	D7
VPP	Vector peak, fixed point, single length	DC
VPPH	Vector peak, fixed point, halflength	DD
VPPF	Vector peak, floating point, single length	DE
VPPFD	Vector peak, floating point, double length	DF
VFLFX	Vector floating to fixed point conversion, single length	A0
VFDFX	Vector floating to fixed point conversion, double to single lengths	A2

Table 1-3. Vector Instruction Set (Continued)

Mnemonic Code	Function	Operation Code
VFLFH	Vector floating to fixed point comparison, single to half lengths	A1
VFXFL	Vector fixed to floating point conversion, single lengths	A8
VFHFL	Vector fixed to floating point conversion, half to single lengths	A9
VFXFD	Vector fixed to floating point conversion, single to double lengths	AA
VFHFD	Vector fixed to floating point conversion, half to double lengths	AB
VNFX	Vector normalize, fixed point, single length	AC
VNFH	Vector normalize, fixed point, half length	AD
VSEL	Select single words from vector $\vec{A}$	B0
VSELH	Select halfwords from Vector $\vec{A}$	B1
VSELD	Select doublewords from Vector $\vec{A}$	B3
VREP	Replace single words in Vector $\vec{C}$	B8
VREPH	Replace halfwords in Vector $\vec{C}$	B9
VREPD	Replace doublewords in Vector $\vec{C}$	BB

OP BITS 0-3

OP BITS 4-7					4	5	6	7	8	9	A	B	C	D	E	F
0					VA	VL					VFLFX	VSEL	VSA	VC	VAND	
1					VAH	VLH					VFLFH	VSELH	VSAH	VCH	VANDD	
2					VAF	VLF					VFDFX			VCF	VCAND	
3					VAFD	VLFD						VSELD	VSAD	VCFD	VCANDD	
4					VAM	VLM	VD						VSL	VO	VOR	
5					VAMH	VLMH	VDH						VSLH	VOH	VORD	
6					VAMF	VLMF	VDF							VOF	VCOR	
7					VAMFD	VLMFD	VDFD						VSLD	VOFD	VCORD	
8					VS	VSS	VDP				VFXFL	VREP		VMG	VXOR	
9					VSH	VSSH	VDPH				VFHFL	VREPH		VMGH	VXORD	
A					VSF	VSSF	VDPF				VFXFD					
B					VSFD	VSSD	VDPFD				VFHFD	VREPD		VMG		
C					VSM	VSSM	VM				VNFX		VCS	VPP	VEQC	
D					VSMH	VSSMH	VMH				VNFH		VCSH	VPPH	VEQCD	
E					VSMF	VSSMF	VMF							VPPF		
F					VSMFD	VSSMFD	VMFD						VCSD	VPPFD		

Note: Blank Boxes represent illegal Op Codes.

Figure 1-4. Vector Op Code Map

1-23 INSTRUCTION FORMAT

The instruction word of the Central Processor contains 32 bits and is divided into five fields (see figure 1-5):

Field Name	Bit Positions	Field Size	Function
OP	0-7	8	Operation Code
R	8-11	4	Register address
T	12-15	4	Address modifier tag
M	16-19	4	Base address designator
N	20-31	12	Displacement address

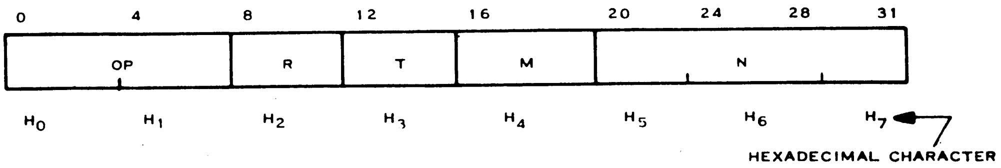


Figure 1-5. ASC Instruction Word Format

- Op-Field. The Op-Field specifies the machine instruction to be executed.
- R-Field. The R-Field addresses one of 16 registers from the arithmetic, base, or index register group.
- T-Field. The T-Field is an address modifier tag that has the following interpretation:

T	Addressing Type	Virtual Address, α, of Memory Operand
0	Direct address	N + (M)
1-7	Indexed address	N + (M) + (T)
8	Indirect	(N + (M))
9-F	Indexed indirect address	(N + (M) + (T - 8))

A symbol or expression enclosed by parentheses () represents "the contents of."

The T-field (figure 1-6) may be decomposed into an I-bit and an X-field where the most significant I-bit designates indirect addressing and the 3-bit X-field specifies one of seven index registers used in the indexing operation. The index registers are physically assigned to register file address locations 21 through 27 (hexadecimal). A special set of index instructions are used to load, store, modify, and test the index registers.

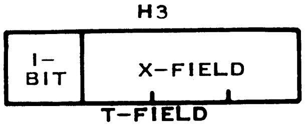


Figure 1-6. T-Field Subdivision

Displacement indexing is provided such that the indexing operation is compatible with word size; i.e., the index registers are automatically aligned according to word size. If an index register contains the value K, the Kth element of an array is accessed, whether it is a halfword, singleword, or doubleword.

- M-Field. The M-field is a base register designator. It is used to extend the addressing range capability of the ASC to a potential 16.7 million words. The M-field selects one of fifteen 24-bit base registers to be added to the N-field displacement before indexing or indirect addressing. No base addressing is used when M equals 0.

- N-Field. The N-field is the address displacement relative to the base address contained in M.

The M- and N-fields also may be interpreted as immediate operands when immediate instructions are specified by the operation code.

1-24 DATA FORMATS

Four data format representations may be used in the ASC:

- Fixed point, single length, 32-bit word (see figure 1-7).

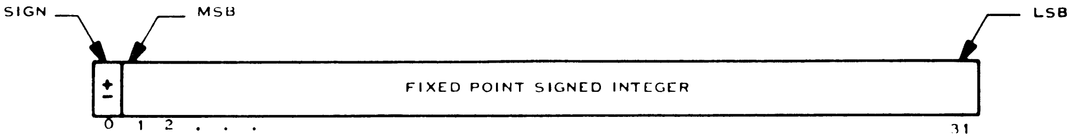


Figure 1-7. 32-bit, Fixed Point Data Word Format

The sign bit is zero for positive numbers and one for negative. Negative numbers are represented in twos complement notation. The binary point is to the right of the least significant bit (LSB), particularly for multiplication or division. The result after addition is the same as though two binary fractions were added.

- Fixed point, half length, 16-bit word (two half length words are shown in figure 1-8).

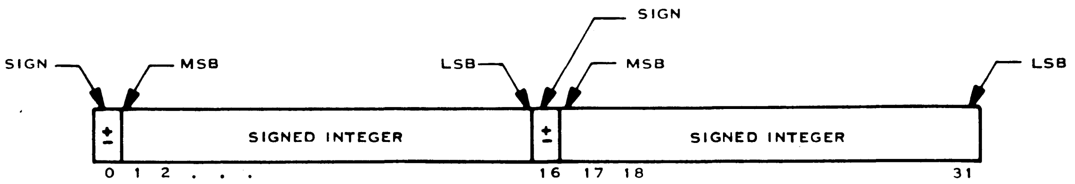


Figure 1-8. 16-bit, Fixed Point Data Word Format

The sign bit is zero for positive numbers and one for negative. Negative numbers are represented in twos complement notation. The binary point is to the right of the LSB. Numbers are in fixed point signed integer notation.

- Floating point, single length, 32-bit word (see figure 1-9).

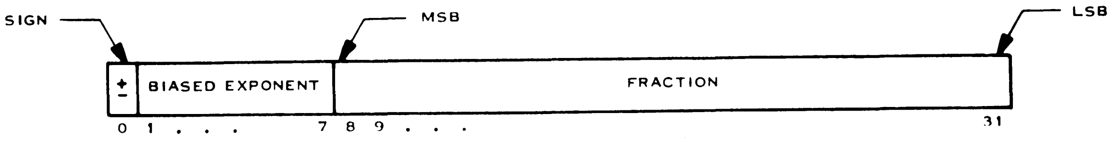


Figure 1-9. 32-bit, Floating Point Data Word Format

The sign bit is zero for positive numbers and one for negative. Sign and magnitude representation is used for the fractional portion, bits 0, 8 through 31. The binary point is to the left of the MSB of the fraction (between bits 7 and 8).

The biased hexadecimal exponent has the range 00_{16} to $7F_{16}$, which covers the base 16 exponent range 16^{-64} to 16^{+63}.

If the value 40 hex is subtracted from the biased exponent, a number is obtained which in signed integer twos complement notation (sign in bit position 1) can be converted to its equivalent decimal value. Sixteen raised to this decimal power gives a number which when multiplied by the fraction produces the number that was represented in floating point notation.

Examples:

Floating point		Decimal value
4110	0000	$(1/16) \times 16^1 = 1$
4210	0000	$(1/16) \times 16^2 = 16$
C110	0000	$-(1/16) \times 16^1 = -1$

Floating point		Decimal value
7FF0	0000	$(15/16) \times 16^{+63}$
0010	0000	$(1/16) \times 16^{-64} = 16^{-65}$

By definition:

0000	0000	zero
7FFF	FFFF	+ ∞
FFFF	FFFF	- ∞
7F00	0000	Indefinite (machine generated)
XX00	0000	Indefinite (dirty zero)

- Floating point, double length, 64-bit word (see figure 1-10).

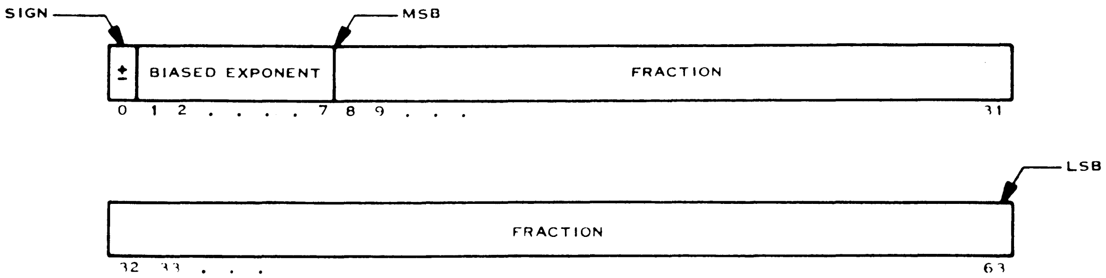


Figure 1-10. 64-bit, Floating Point Data Word Format

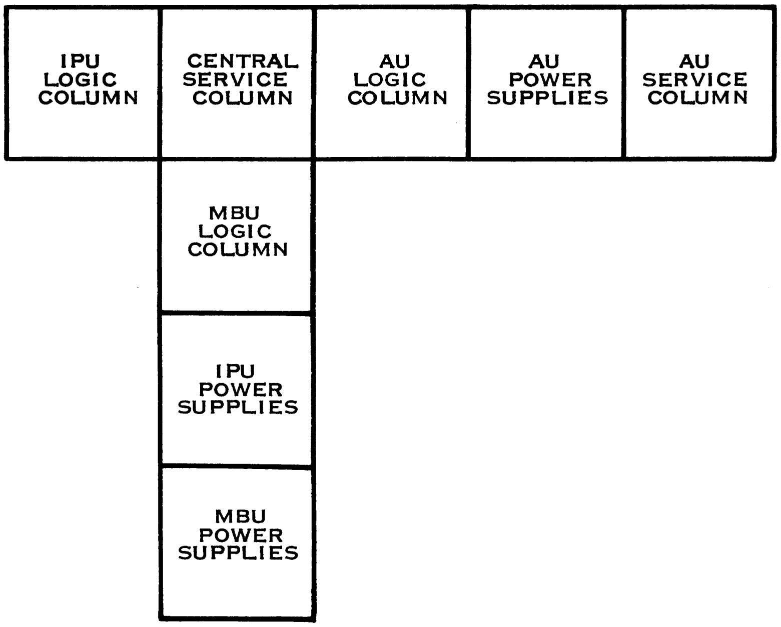
The sign bit is zero for positive numbers and one for negative. Sign and magnitude representation is used for the fractional portion, bits 0, 8 through 63. The binary point is to the left of the MSB of the fraction (between bits 7 and 8).

The biased hexadecimal exponent has the range 00_{16} to $7F_{16}$, which covers the base sixteen exponent range 16^{-64} to 16^{+63}.

Subtracting the value 40_{16} from the biased exponent yields a number which, in signed integer twos complement notation (sign in bit position 1), can be converted to its equivalent decimal value. Sixteen raised to this power gives a number which when multiplied by the fraction produces the number that was represented in floating point notation.

1-25 PHYSICAL DESCRIPTION

The ASC Central Processor in a one-pipe configuration is housed in a series of eight vertical logic and service columns. Figure 1-11 illustrates a typical layout for these columns; their actual arrangement may be changed to meet the physical requirements of the particular site. Each vertical logic column (IPU, MBU or AU) contains a three motherboard chassis capable of accepting up to 66 logic cards. Three other vertical columns containing mounting space for power



(A) 115135

Figure 1-11. Typical ASC Central Processor One-Pipe Configuration

supplies provide the dc power requirements for their respective CP unit. Between the three main logic columns is a service column that contains the connector panels for the cable connections between the logic columns. In addition to this central service column, other non-logic service columns provide electrical output busses and water input plumbing for the CP cooling system.

1-26 COOLING SYSTEM

The Central Processor cooling system consists of a combination of forced air and circulated, cooled water to dissipate heat generated by the logic circuits. This cooling system is represented schematically in figure 1-12. The cold plate between each set of logic cards is a copper plate with small tubes running through it. Cooled water pumped through these tubes absorbs heat from the air surrounding the cold plate and carries the heat away from the logic card area to a heat exchanger. The heat exchanger releases the heat to the surrounding air and returns the cooled water to the logic chassis to complete the cycle. A blower assembly in each logic column aids cooling by circulating room temperature air past the logic cards.

1-27 LOGIC CIRCUITS

Central Processor logic is implemented on 9-1/2 inch by 7-1/2 inch printed circuit boards using Emitter Coupled Logic (ECL) integrated circuit packages. A 272-pin connector on one end of the circuit board mates with a corresponding

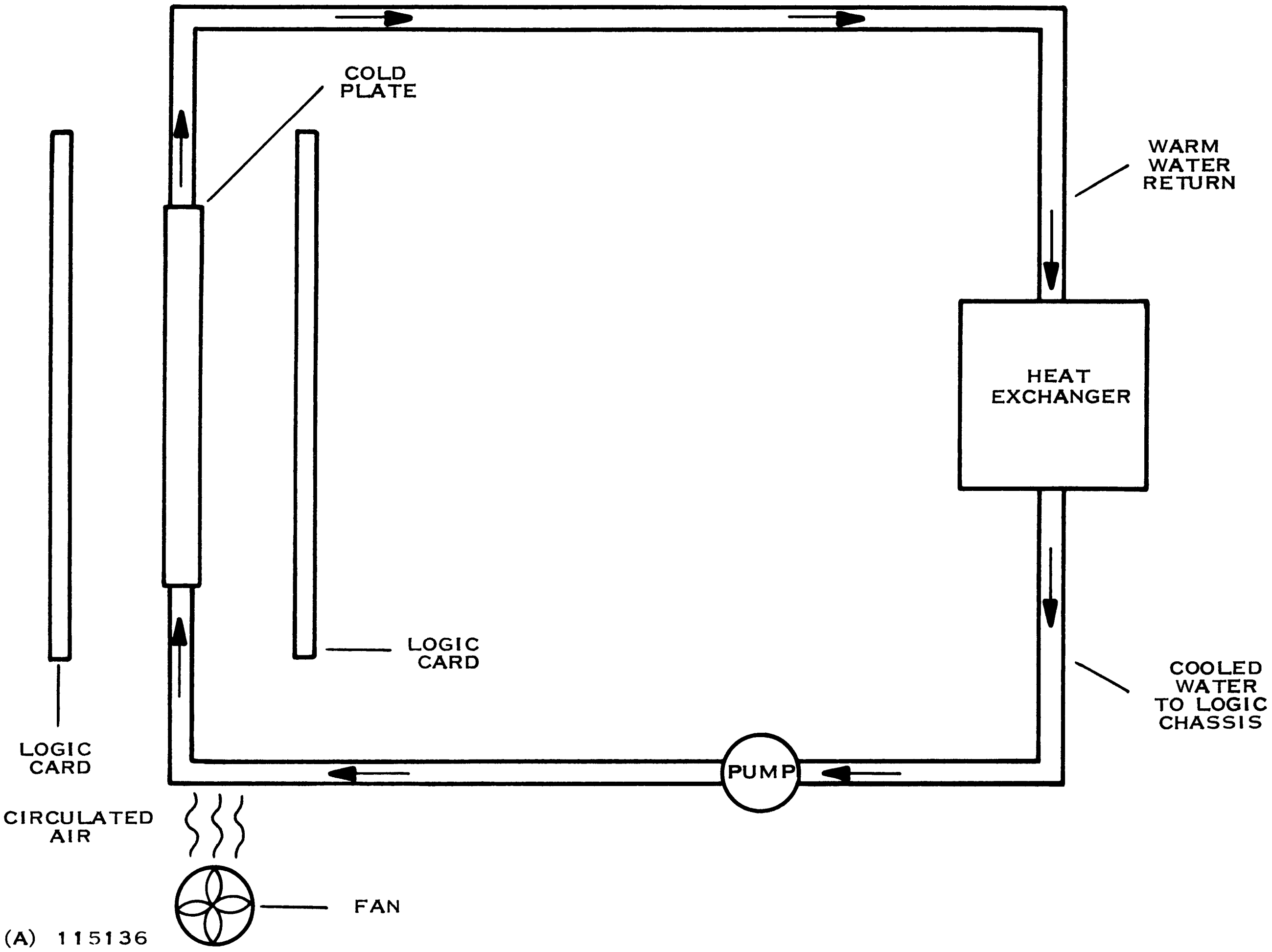


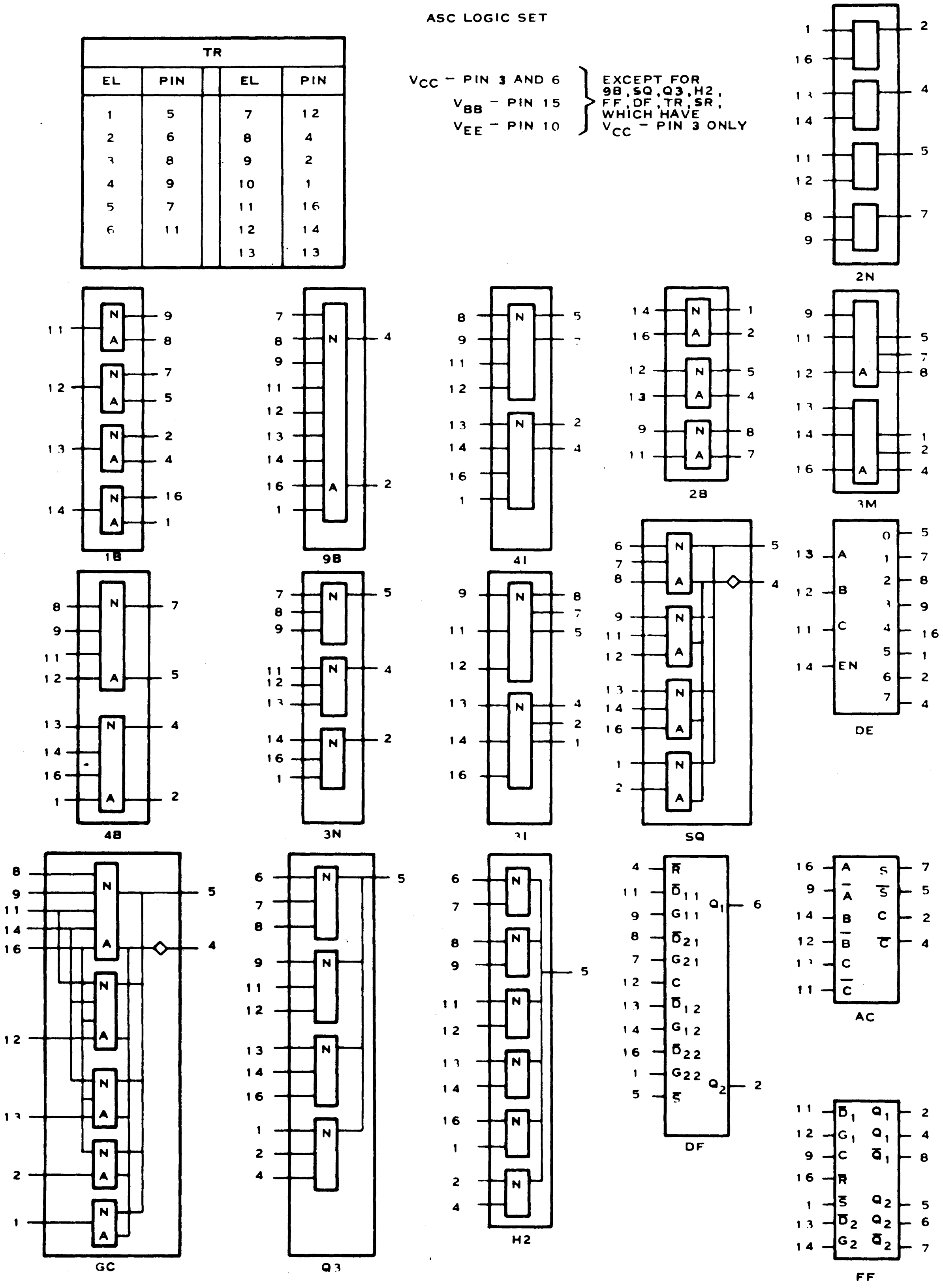
Figure 1-12. Schematic Representation of CP Cooling System

receptacle in one of three motherboards in a vertical logic column. The motherboard supplies inter-chassis wiring connections plus a bus of common signals, bias voltages and ground. Figure 1-13 illustrates the different logic circuits in the ECL logic set. Table 1-4 defines the function of each of these logic circuits. Refer to section VI, Parts Listing, of this manual for a listing of the logic cards by chassis location.

ASC LOGIC SET

TR			
EL	PIN	EL	PIN
1	5	7	12
2	6	8	4
3	8	9	2
4	9	10	1
5	7	11	16
6	11	12	14
		13	13

V_{CC} – PIN 3 AND 6
V_{BB} – PIN 15
V_{EE} – PIN 10
} EXCEPT FOR 9B, SQ, Q3, H2, FF, DF, TR, SR, WHICH HAVE V_{CC} – PIN 3 ONLY



(B) 109045

Figure 1-13. ECL Circuits

Table 1-4. ECL Circuit Types

Type	Title
1B	Four single to double ended converters
2N	Four 2-input inverting gates
2B	Three 2-input complementary gates
3N	Three 3-input inverting gates
4B	Two 4-input complementary gates
9B	9-input complementary gate
3I	Two 3-input, 3-output inverting gates
3M	Two 3-input, 3-ouput non-inverting gates
4I	Two 4-input, 2-output inverting gates
SQ	Three 3-input, One 2-input gates with dotted complementary outputs
GC	Four bit group carry gate structure
Q3	Four 3-input with dotted inverted output
DE	Three bit decoder with enable
AC	Full sum-carry with complementary outputs
H2	Six 2-input with dotted inverted outputs
FF	Two single-input gated clocked latches
DF	Two 2-input gated clocked latches
TR	Termination resistors (40Ω)
SR	Termination resistors (80Ω)
TE	TTL/ECL level converters W/ECL enable
ET	ECL/TTL level converters
OD	TTL output drivers
RS	Termination resistors (400Ω pulldown)
RD	Termination resistors (80Ω TTL)

Table 1-4. ECL Circuit Types (Continued)

Type	Title
AD	Two 2-input ECL/MOS level converters
DD	Four single-input TTL/MOS level converters
MA	MOS 256 X 8 memory array
2S	Two 2-input line receivers

SECTION II

INSTALLATION

2-1 GENERAL

Installation information is not provided in this publication. Refer to the ASC System Installation manual, Texas Instruments part number 929980-1.

SECTION III
OPERATING INSTRUCTIONS

3-1 GENERAL

Operating instructions are not included in this publication. Refer to the ASC Operator's Manual, Texas Instruments part number 931433-1.

SECTION IV

PRINCIPLES OF OPERATION

4-1 GENERAL

The ASC Central Processor is a layered pipeline processor. As such, the CP contains distinct levels, or stages, in the development of an instruction in the IPU, of operands in the MBU, and of results in the AU. Each of these levels can hold and simultaneously operate on a separate instruction or set of operands, unless the level has been reserved by a previous instruction. The IPU contains five levels for instruction development (levels 0-4), the MBU has an input and an output level for operand selection (levels 5 and 6), and the AU has a minimum of two levels (input and output). The number of effective levels in the AU varies with the operations being performed. Figure 4-1 illustrates the basic components of the Central Processor, their interconnections, and their relation to the levels of the CP pipe. The following theory discussion centers around this block diagram and explains the major functions of each block in the Central Processor. Additional maintenance data is included in the appendices to this manual. Detailed controller flowcharts and discussion follows the block diagram description.

4-2 IPU LEVEL 0

Level 0 of the IPU generates addresses to central memory to request instruction octets (eight word groups), receives the octets from memory, and selects one word instructions from the octets for transfer to the Instruction Register (IR) in level 1. The addressing portion consists of the Look-Ahead (LA) Register, the Present Address (PA) Register, the Output Address (OA) Register and the Branch Address (BA) Register. These registers ensure that the correct address will be in OA to access the next octet of instructions for the IPU. The Memory Interface File (KCM) and the two Current Instruction Files (KA and KB) receive and hold instruction octets from memory so that the selection circuits may access words from the octets. The File and Word select circuits use the address in PA to select an instruction from either KA or KB. While instructions are being drawn from either KA or KB, the other unused file can receive a new octet from memory. This latter file can then supply the next series of instructions without delay to the IPU. The following paragraphs describe the function of each of these level 0 components.

4-3 LOOK-AHEAD REGISTER (LA)

LA is a 24-bit register that normally holds the address of the octet that is currently being requested from memory. When central memory accepts that request, the output from LA is fed through an adder to increase the address by eight to form the address of next octet in sequence. This new octet address enters the OA register for transfer to central memory, and also the LA register for the next look ahead cycle. At the start of an instruction sequence, the first address to be fetched from memory is in the P3 register (P3 receives this address during initial CP loading, since the addressing registers at level 0 are used to load the CP with the new program). To initiate the new program the address in P3 transfers into OA, LA and PA. The IPU issues a memory request for the octet indicated by the address in OA, and transfers the address in LA through the adder to OA and LA. The address in PA selects an instruction from the octet when it returns from memory. LA continues to supply addresses through the adder to OA until the end of the program sequence if no cycle interruptions occur.

4-4 CYCLE INTERRUPTIONS. The normal processing cycle for the LA register may be broken by either a branch instruction, a Load Look-Ahead (LLA) instruction, or an instruction hazard at level 3 of the IPU. When a branch instruction reaches level 3 of the IPU and the address of the branch target is not already in the pipe, the address of the new instruction transfers from the AR register in level 3 to LA, OA and PA so that instructions from the branch path may be accessed from memory and loaded into the IPU.

An LLA instruction prepares the IPU for a branch back to a point in the program sequence occupied by the LLA. When the LLA reaches level 3 of the IPU, the address of the LLA in the P3 register is stored into the BA register. When the indicated branch instruction enters the pipe, the address in BA transfers to LA and OA to fetch the octet containing the LLA from memory and continue to access instructions from that instruction path.

If an instruction reaches level 3 of the IPU and a hazard has occurred that makes the instruction invalid, the address of that instruction is transferred from P3 to LA and OA to re-fetch that instruction octet from memory to obtain valid information for that instruction. When memory returns the valid instruction, the look ahead cycle continues in the normal manner.

4-5 OUTPUT COMPARE. The output of the LA Register feeds two compare circuits. One network uses the output to determine if a far range instruction hazard exists in the LA octet. The other network determines if the LA octet contains the object address of a branch or execute instruction or an indirect address. Refer to the discussion of these networks for further explanation of the comparisons.

4-6 LOAD LOOK-AHEAD COUNTER

The Load Look-Ahead Counter is a 12-bit, decrementing counter used only during a Load Look-Ahead instruction. When the LLA instruction reaches Level 3 of the IPU, the N field of that instruction enters the LLA Counter. The N field specifies the number of instructions to be executed before the required branch occurs. The counter then decrements by one for each instruction that reaches Level 1 of the pipe. When the LLA count minus the number of active IPU levels (at the time of the LLA) is equal to zero, the counter transfers the address in the Branch Address Register to the Look-Ahead Register (LA) request to memory. Refer to the Load Look-Ahead controller discussion for a flow chart and theory of the look-ahead process.

4-7 BRANCH ADDRESS REGISTER (BA)

BA is a 24-bit register that is used only during a Load Look-Ahead operation. When the LLA instruction reaches Level 3 of the IPU, the instruction address at that level transfers from the P3 Register to BA. BA then holds that address until the LLA Counter transfers the address to the LA register.

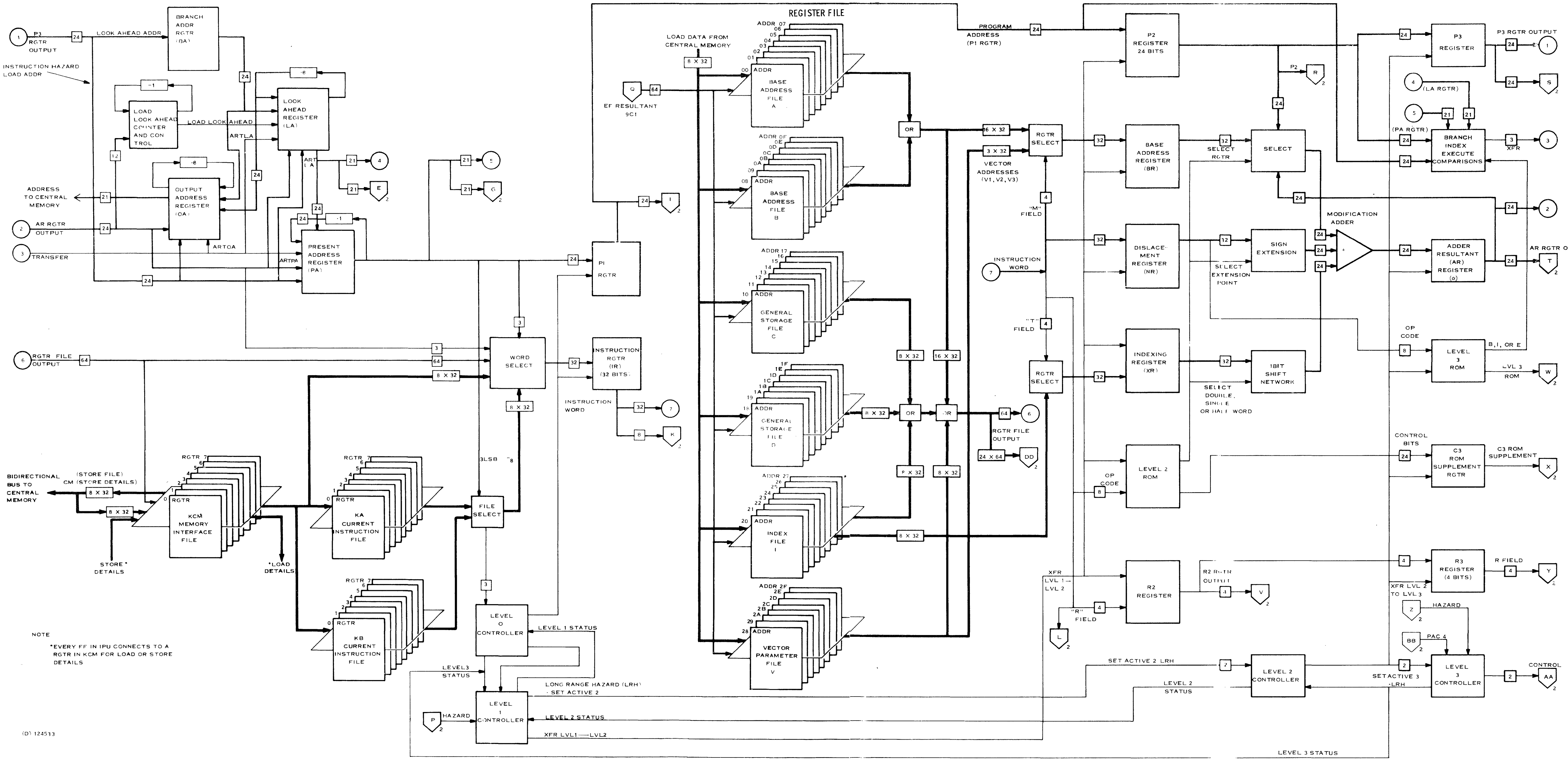


Figure 4-1. Central Processor Block Diagram (Sheet 1 of 5)

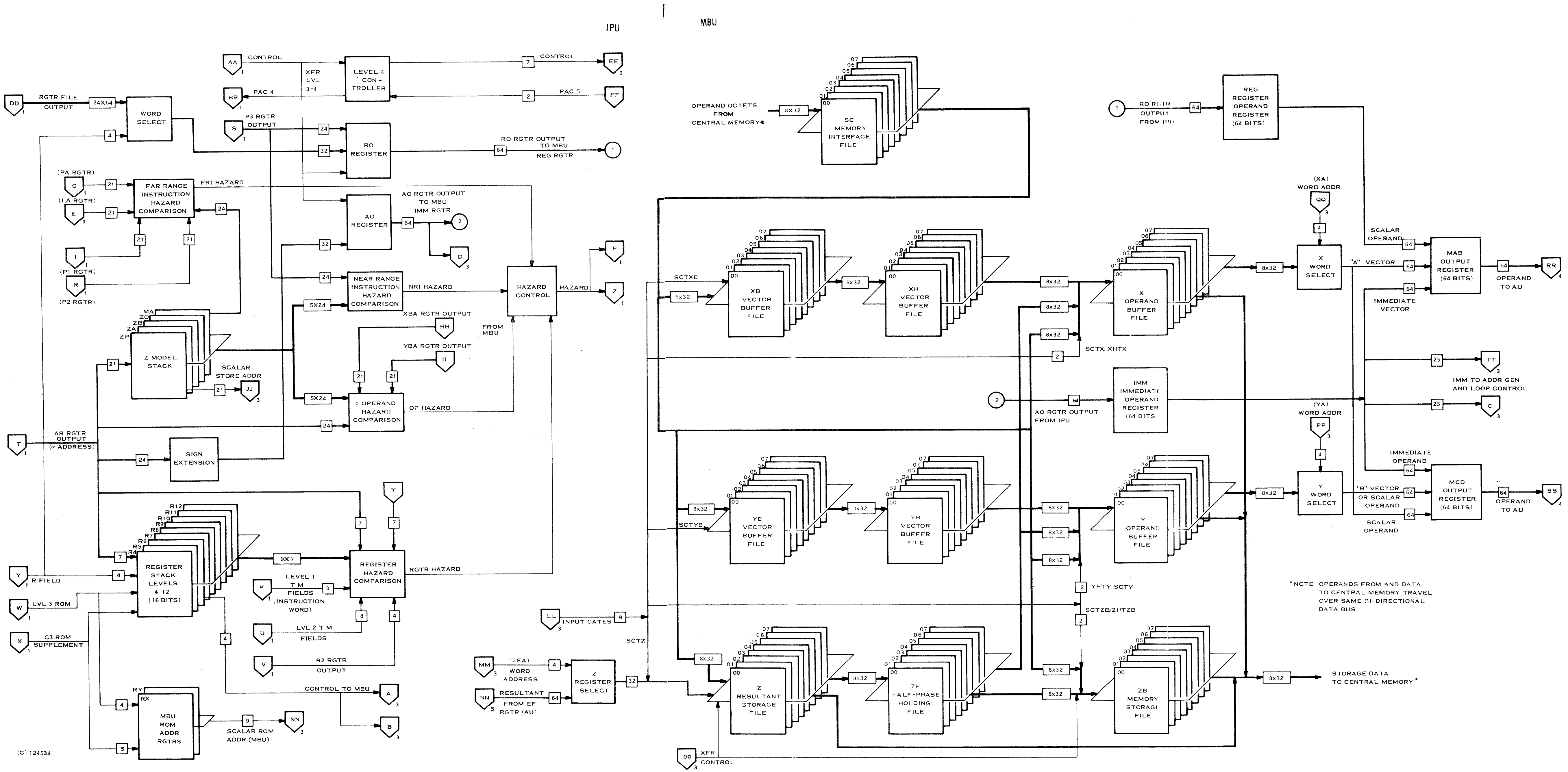


Figure 4-1. Central Processor Block Diagram (Sheet 2 of 5)

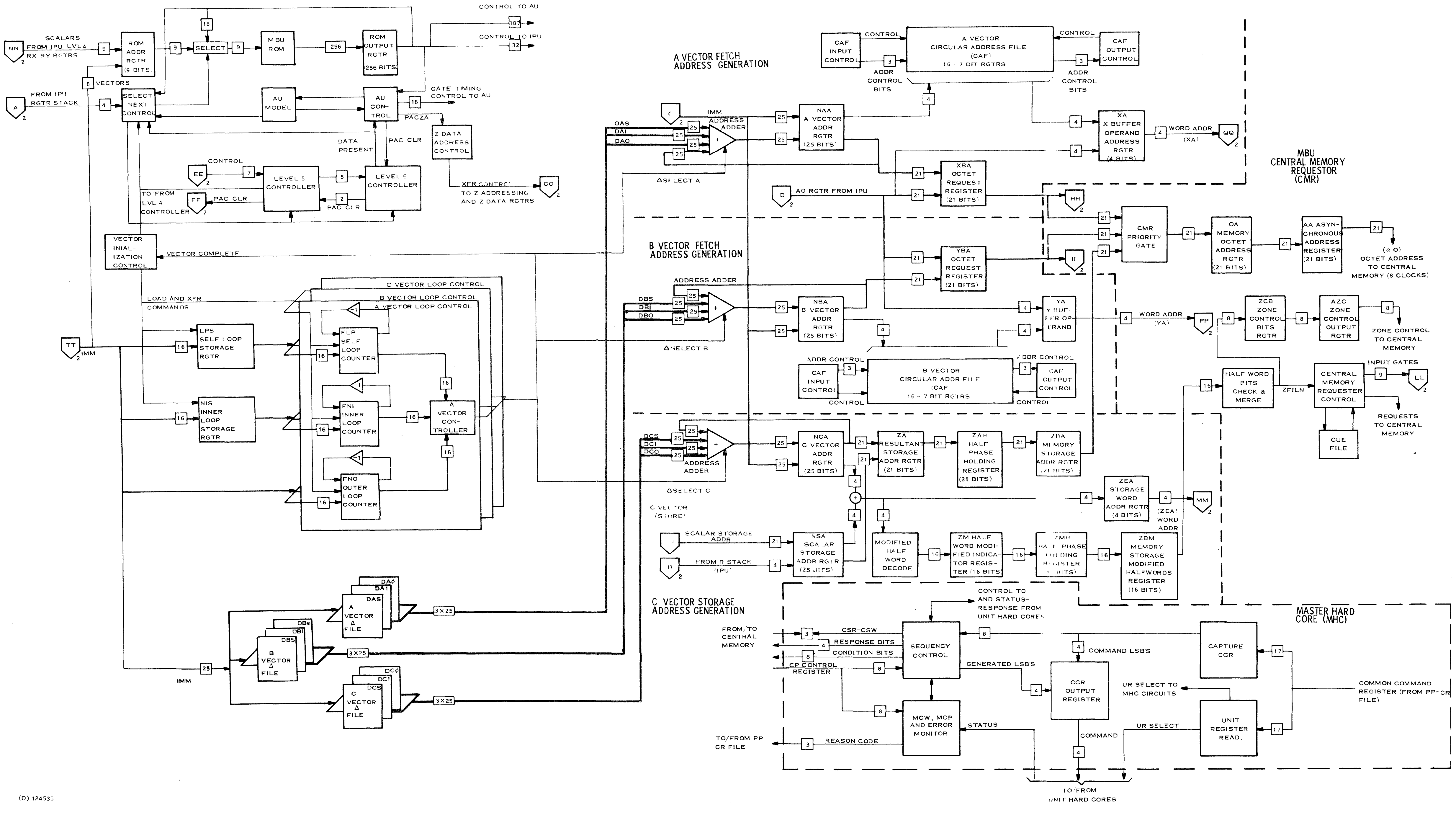

SCALARS
FROM IPU LVL 4
RX RY RGTRS
NN
ROM ADDR RGTR (9 BITS)
SELECT
MBU ROM
256
ROM OUTPUT RGTR 256 BITS
CONTROL TO AU
CONTROL TO IPU
18
32
VECTORS
FROM IPU RGTR STACK
SELECT NEXT CONTROL
AU MODEL
AU CON-TROL
GATE TIMING CONTROL TO AU
PACZA
DATA PRESENT
PAC CLR
Z DATA ADDRESS CONTROL
XFR CONTROL TO Z ADDRESSING AND Z DATA RGTRS
OO
CONTROL
EE
LEVEL 5 CONTROLLER
LEVEL 6 CONTROLLER
TO FROM LVL 4 CONTROLLER
FF
PAC CLR
VECTOR INIAL-IZATION CONTROL
VECTOR COMPLETE
C VECTOR LOOP CONTROL
B VECTOR LOOP CONTROL
A VECTOR LOOP CONTROL
LOAD AND XFR COMMANDS
TT
IMM
LPS SELF LOOP STORAGE RGTR
FLP SELF LOOP COUNTER
NIS INNER LOOP STORAGE RGTR
FNI INNER LOOP COUNTER
A VECTOR CON-TROLLER
FNO OUTER LOOP COUNTER
A VECTOR Δ FILE
DA0 DA1 DAS
B VECTOR Δ FILE
DB0 DB1 DBS
C VECTOR Δ FILE
DC0 DC1 DCS
3X25
IMM
(D) 124535
A VECTOR FETCH ADDRESS GENERATION
DAS DAI DAO
IMM
ADDRESS ADDER
ΔSELECT A
NAA A VECTOR ADDR RGTR (25 BITS)
CAF INPUT CONTROL
A VECTOR CIRCULAR ADDRESS FILE (CAF) 16 - 7 BIT RGTRS
CAF OUTPUT CONTROL
ADDR CONTROL BITS
XA X BUFFER OPERAND ADDRESS RGTR (4 BITS)
WORD ADDR (XA)
QQ
XBA OCTET REQUEST REGISTER (21 BITS)
AO RGTR FROM IPU
MBU CENTRAL MEMORY REQUESTOR (CMR)
HH
CMR PRIORITY GATE
OA MEMORY OCTET ADDRESS RGTR (21 BITS)
AA ASYN-CHRONOUS ADDRESS REGISTER (21 BITS)
(2 O) OCTET ADDRESS TO CENTRAL MEMORY (8 CLOCKS)
B VECTOR FETCH ADDRESS GENERATION
DBS DBI DBO
ADDRESS ADDER
ΔSELECT B
NBA B VECTOR ADDR RGTR (25 BITS)
YBA OCTET REQUEST REGISTER (21 BITS)
II
YA Y BUFFER OPERAND
WORD ADDR (YA)
PP
ZCB ZONE CONTROL BITS RGTR
AZC ZONE CONTROL OUTPUT RGTR
ZONE CONTROL TO CENTRAL MEMORY
B VECTOR CIRCULAR ADDR FILE (CAF 16 - 7 BIT RGTRS
INPUT GATES
HALF WORD BITS CHECK & MERGE
ZFILN
CENTRAL MEMORY REQUESTER CONTROL
LL
CUE FILE
REQUESTS TO CENTRAL MEMORY
DCS DCI DCO
ADDRESS ADDER
ΔSELECT C
C VECTOR (STORE)
NCA C VECTOR ADDR RGTR (25 BITS)
ZA RESULTANT STORAGE ADDR RGTR (21 BITS)
ZAH HALF-PHASE HOLDING REGISTER (21 BITS)
ZBA MEMORY STORAGE ADDR RGTR (21 BITS)
ZEA STORAGE WORD ADDR RGTR (4 BITS)
(ZEA) WORD ADDR
MM
SCALAR STORAGE ADDR
FROM R STACK (IPU)
NSA SCALAR STORAGE ADDR RGTR (25 BITS)
MODIFIED HALF WORD DECODE
ZM HALF WORD MODI-FIED INDICA-TOR REGIS-TER (16 BITS)
ZMH HALF PHASE HOLDING REGISTER (16 BITS)
ZBM MEMORY STORAGE MODIFIED HALFWORDS REGISTER (16 BITS)
C VECTOR STORAGE ADDRESS GENERATION
CONTROL TO AND STATUS-RESPONSE FROM UNIT HARD CORES
MASTER HARD CORE (MHC)
FROM/TO CENTRAL MEMORY
CSR-CSW
RESPONSE BITS
CONDITION BITS
CP CONTROL REGISTER
SEQUENCY CONTROL
COMMAND LSBS
GENERATED LSBS
CCR OUTPUT REGISTER
CAPTURE CCR
UR SELECT TO MHC CIRCUITS
COMMON COMMAND REGISTER (FROM PP-CR FILE)
MCW, MCP AND ERROR MONITOR
STATUS
UR SELECT
UNIT REGISTER READ.
TO/FROM PP CR FILE
REASON CODE
COMMAND
TO/FROM UNIT HARD CORES

Figure 4-1. Central Processor Block Diagram (Sheet 3 of 5)

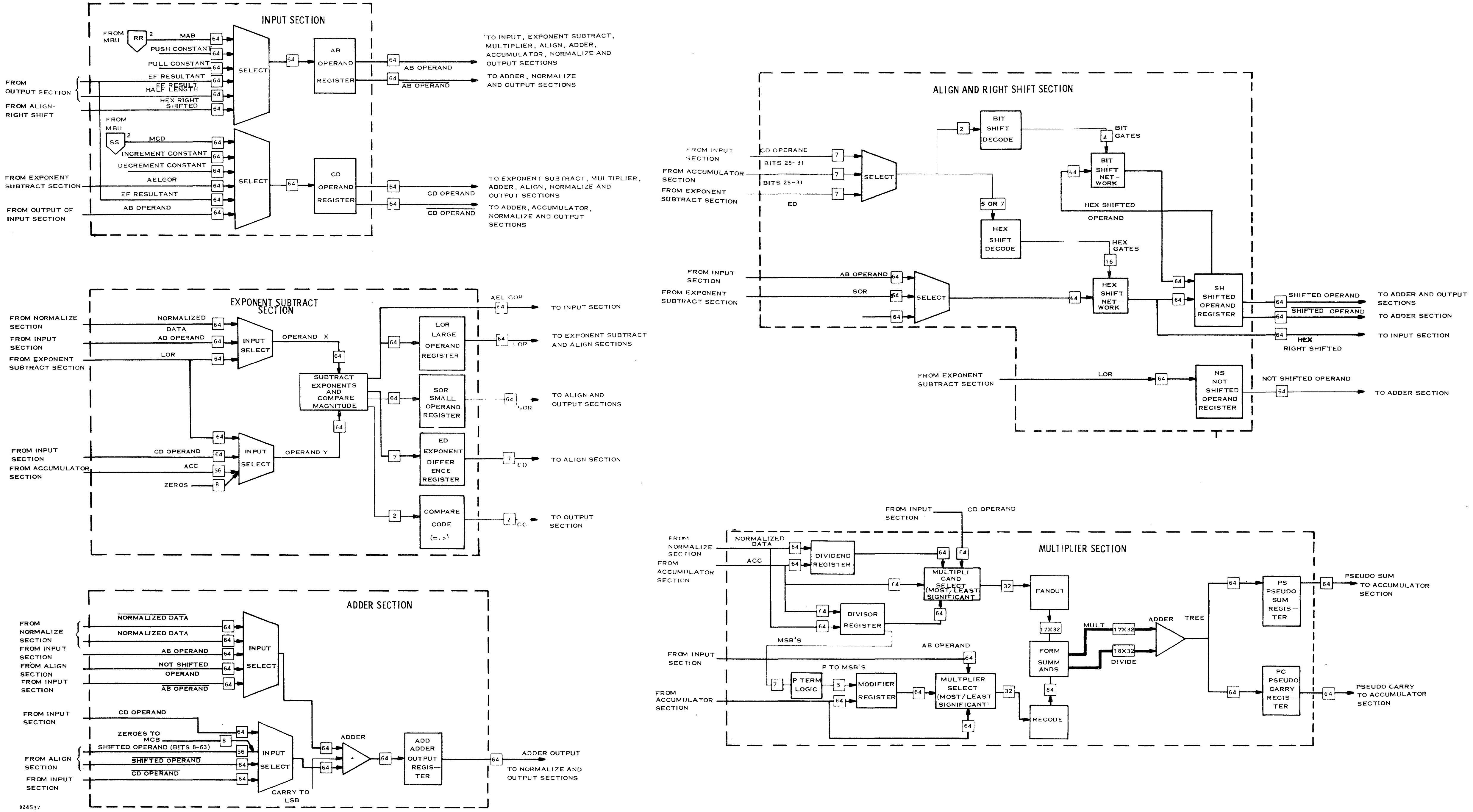


Figure 4-1. Central Processor Block Diagram (Sheet 4 of 5)

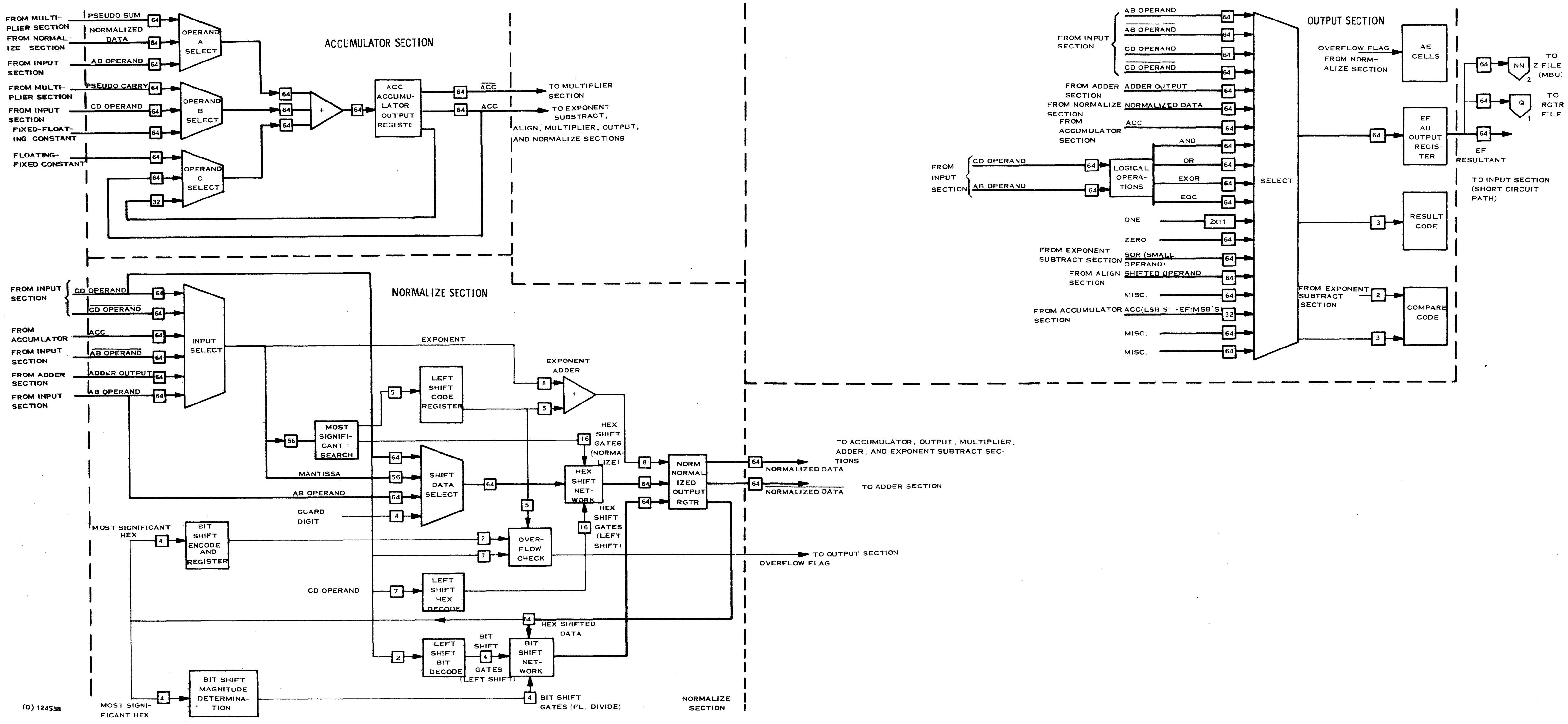


Figure 4-1. Central Processor Block Diagram (Sheet 5 of 5)

4-8 PRESENT ADDRESS REGISTER (PA)

The PA Register is a 24-bit address register that holds the address of the next word to be transferred from the instruction file to the Instruction Register (IR). The address in PA increments by one word address each time a word enters the Instruction Register. The three least significant bits of the PA Register select the word from the Instruction File during normal instruction processing. These bits also determine when the last bit in an octet has been accessed and are, therefore, used to gate input to the PA Register and to toggle the File Select network from one instruction file to the other. As a new instruction enters IR, the word address in PA transfers to P1 Register to accompany the instruction through the IPU.

4-9 PA INPUTS. The PA Register is normally loaded from the LA Register when the three LSB's of PA are all 1's. However, the output from P3 can enter PA at the start of an instruction sequence of if an instruction hazard is detected at Level 3 of the IPU. The output from the AR register can also load PA during a branch instruction, an execute instruction, or for indirect addressing, providing that the object address of the operation is contained in the current octet as determined by the Branch, Execute, Indirect comparison network at Level 3.

4-10 OUTPUT ADDRESS REGISTER (OA)

The Output Address Register is a 24-bit register that relays 21-bit octet addresses to Central Memory for data transfer to/from KCM. All memory accesses from the IPU must transmit an address to memory through the OA register. Three input paths to the OA register provide addressing capability for all IPU communication to Central Memory.

4-11 P3 REGISTER OUTPUT. During an instruction sequence start-up, or if an instruction hazard is detected at Level 3 of the pipe, the contents of the P3 Register (Level 3 Program Address Register) transfer into the OA Register. P3 holds either the first address of the instruction sequence in the case of a start-up operation, or the address of the instruction that must be re-fetched due to an instruction hazard. In either case, the OA Register transmits that address to Central Memory to begin the instruction sequence.

4-12 LA REGISTER OUTPUT. During normal instruction processing, new instruction addresses enter the OA Register through the octet adder circuit (+8) from the Look-Ahead Register. The LA register provides a continuous source of instruction addresses to be fetched from memory.

4-13 AR REGISTER OUTPUT. For indirect addresses or for branch or execute instructions, the output of the AR Register may transfer to the OA Register if the required address is not already in the pipe. Comparison circuits at Level 3 determine if it is necessary to access memory for the desired word.

4-14 LOAD/STORE DETAILS. The OA Register transmits sequential addresses to memory to Load or Store the contents of the IPU from or into memory. The details instruction from the peripheral processor loads the OA Register with a pointer address that points to indicate the address of the first octet of the details map in memory. A partial adder then increments the address by one octet (addition of 8) to provide sequential octet addresses to memory.

4-15 KCM MEMORY INTERFACE FILE

KCM is an octet register file containing eight 32-bit registers. This file performs a buffer function between ASC Central Memory and the IPU registers and flip-flops. All IPU data transfer operations to and from Central Memory must pass through KCM. KCM holds the data until it can be synchronized with clock pulses for orderly transfer through the IPU, or until memory accepts the data to be stored.

4-16 INSTRUCTION PROCESSING. During instruction processing, the KCM file receives instruction octets from Central Memory and transfers the octets to one of the two current instruction files: KA or KB. The KCM file is transferred to whichever current instruction file is not being accessed by the current instruction address. For indirect addressing, a direct path from KCM to the instruction word select circuit by-passes the current instruction files to avoid alteration of the files. This path allows an instruction from Central Memory to be loaded directly into the Instruction Register.

4-17 LOAD/STORE DETAILS. Each bit in the KCM file connects directly to numerous bits throughout the IPU for use in a Load or Store Details operation. Each octet of the details map in Central Memory transfers sequentially to the KCM file (Load Details). The position of the octet in the details map determines which of the KCM output paths will be enabled for each bit of the octet until all flip-flops and registers in the IPU reflect the condition specified in the details map. The transfer path is similar, but in the opposite direction for a Store Details operation. Certain Details paths are also used in Load/Store Status or Intermediate commands. The process is the same for these operations as for the Details operation, but limited in scope.

4-18 STORE FILE. The Store File operation passes through KCM for transfer to Central Memory. The output from the Register File fills KCM and the octet transfers to memory. Load File enters data into KCM and then to the Register File.

4-19 KA/KB CURRENT INSTRUCTION FILES

The Current Instruction Files are two octet files containing eight 32-bit registers each. During normal operation they receive alternate, synchronized octets from KCM that contain instruction words to be accessed by the IPU. The first octet enters the KA instruction file. While addresses are selecting words from the KA file, KCM loads the next octet into the KB file. This alternate loading process allows the IPU to proceed uninterrupted through an instruction sequence without the delay required to access a new octet from Central Memory. This time advantage is lost, however, when a Branch instruction jumps to an instruction that is not resident in either the KA or the KB file.

4-20 FILE SELECT

The File Select circuit controls the sequencing of the Current Instruction Files and relays the file status to the Level 0 Controller. When the first instruction octet from memory enters KA, the File Select circuit gates the output from the KA registers to the Word Select network. File Select then monitors the three least significant bits (LSB) from the Present Address Register and enables the KB registers to the Word Select network on the clock after the three LSB's of PA are all ones (hexadecimal 7). File selection alternates in a like manner until the instruction set is complete or a Branch instruction alters the order of instruction processing. File Select also notifies the Level 0 Controller when either instruction file is full and which file is selected. This enables the controller to determine if valid data is available for transfer to Level 1.

4-21 WORD SELECT

The Word Select circuit enables the proper instruction word from either KA or KB to be transfered to the Instruction Register. The Look-Ahead Controller determines when the transfer will take place. Only one octet is active to the input of the word select circuits at any one time. During sequential instruction fetching, the file select circuit supplies one octet to the word select network. When the object address of an indirect address or an Execute instruction is not resident in the KA or KB files, either the KCM octet or an octet from the Register File supplies inputs to the word select network, depending upon the origin of the instruction octet. The select circuit then monitors the three LSB's from either the PA register (sequential instruction acquisition) or the AR register (indirect addressing or Execute instruction). These bits designate a particular word within the active octet.

4-22 LEVEL 0 CONTROLLER

The Level 0 Controller monitors the status of the instruction files to determine if valid data is present in Level 0, checks the status of the Level 1 Controller to determine if that level can accept a new instruction, and receives instruction status from Level 3 to determine if an instruction in Level 3 affects the actions required by Level 0. Level 0 Controller then issues a transfer signal to gate the Level 0 instruction and program address into the Level 1 registers. Refer to the Level 0 Controller flowchart and description later in this section for a detailed representation of controller functions.

4-23 IPU LEVEL 1

Level 1 of the IPU pipe is a passive level. It receives an instruction word from the Level 0 selection network and holds it until Level 2 is ready to accept the new instruction. While in Level 1, the instruction is checked for an indirect address or an Execute instruction, either of which disables instruction reception for Level 1 until the object of those functions passes through Level 1. The following paragraphs describe the major components of Level 1.

4-24 P1 REGISTER

The P1 Register is a 24-bit register that holds the address of the instruction currently in the Instruction Register of Level 1 of the IPU. The address transfers into P1 from the PA register when the instruction enters the Instruction Register and leaves P1 when the Level 1 Controller gates the instruction to Level 2.

4-25 INSTRUCTION REGISTER (IR)

The Instruction Register is a 32-bit register that receives an instruction word that has been selected from the instruction file, from Central Memory directly through KCM, or from the output of the Register File. IR holds the instruction until Level 1 Controller transfers it to Level 2. If IR contains an Execute instruction, or one containing an indirect address, Level 1 Controller prevents further instructions from entering the Instruction Register until the object of that instruction is retrieved from memory and passes into the Instruction Register.

4-26 LEVEL 1 CONTROLLER

The Level 1 Controller monitors the hazard detection circuit to detect a far range hazard, checks the status of Level 2 Controller to determine if that level can accept a transfer, and samples the instruction in Level 3 to determine its effect on Level 1. The controller then gates the contents of Level 1 into Level 2 and sets the active bit in the Level 2 Controller. Refer to the Level 1 Controller flowchart and description later in this section for a complete representation of the controller's functions.

4-27 REGISTER FILE

The Register File is a storage area in the IPU that is loaded by either a direct memory transfer or from the output of the Arithmetic Unit of the Central Processor. The file consists of forty-eight 32-bit registers grouped into six octets. The octets are designated by the letters A, B, C, D, I and V, and respond to the hexadecimal addresses 01 through 2F if the "M" field of the addressing instruction is equal to zero. The output of the Register File is available to three levels of the IPU pipe: Level 0 for indirect addressing and Execute instructions, Level 2 for base addresses and indexing, and Level 4 for operands and vector parameters except X, Y, and Z addresses. The following paragraphs provide an outline of the contents and function of the octets in the Register File.

4-28 BASE ADDRESS FILE, A AND B

Octets A and B of the Register File (addresses 01 through 0F) are used for base addressing. Their output is selected by the 4-bit "M" field in the instruction containing base addressing. Since an "M" field of zero indicates no base addressing is to be done, Register File address 00 is inaccessible by this network. No register resides in location 00 of the Register File.

4-29 GENERAL STORAGE FILE, C AND D

Octets C and D of the Register File (addresses 10 through 1F) provide general storage for arithmetic operations or for quick access by instructions. These files can be loaded directly from memory to provide a source of instructions or operands to the IPU.

4-30 INDEX FILE, I

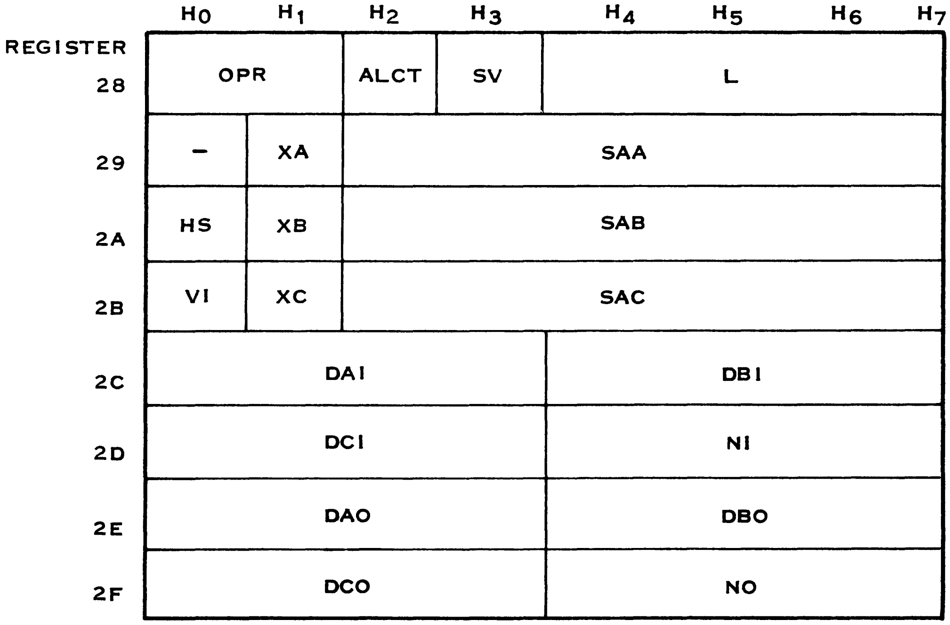
Octet I of the Register File (addresses 20 through 27) holds the index registers for indexing an address of an instruction. The T field of that instruction selects the proper register from the Index File to be used in the indexing process. Since a T field of zero indicates that no indexing will be performed, address 20 of the I File is inaccessible to the indexing network. Address 20 provides an additional general storage register.

4-31 VECTOR PARAMETER FILE, V

Octet V of the Register File (addresses 28 through 2F) supplies eight words that define the parameters used in a vector operation. A vector instruction results in reading the entire contents of the file. The words of the file are assigned as shown in figure 4-2. Table 4-1 defines the word fields. Words 29, 2A, 2B of the file, the starting addresses of the vectors, enter Level 2 of the IPU pipe for possible address modification. The remaining five words enter directly into Level 4 for transfer to the MBU.

4-32 IPU LEVEL 2

Level 2 of the IPU pipe is a selection and holding level in preparation for address modification. If index or base plus displacement addressing is indicated by the incoming instruction, this level channels the proper index and base address values into their respective holding registers, modifies the register



REGISTER	H_0	H_1	H_2	H_3	H_4	H_5	H_6	H_7
28	OPR		ALCT	SV	L			
29	–	XA	SAA					
2A	HS	XB	SAB					
2B	VI	XC	SAC					
2C	DAI				DBI			
2D	DCI				NI			
2E	DAO				DBO			
2F	DCO				NO			

114314

Figure 4-2. Vector Parameter File Format

Table 4-1. V-File Field Descriptions

Reg	Hex Character	Field	Description
28	H_0,H_1	ØPR	Operation code
28	H_2	ALCT	Arith. & Log. Comparison Term
28	H_3	SV	Single-valued vector
28	H_4-H_7	L	Vector dimension
29	H_1	XA	Initial index A
2A	H_1	XB	Initial index B
2B	H_1	XC	Initial index C
29	H_2-H_7	SAA	Starting address A
2A	H_2-H_7	SAB	Starting address B
2B	H_2-H_7	SAC	Starting address C
29	H_0-H_7	(29)	Immediate operand A
2A	H_0-H_7	(2A)	Immediate operand B
2A	H_0	HS	Halfword starting address
2B	H_0	VI	Vector increment direction
2C	H_0-H_3	DAI	$\pm\Delta A_i$, inner loop
2C	H_4-H_7	DBI	$\pm\Delta B_i$, inner loop
2D	H_0-H_3	DCI	$\pm\Delta C_i$, inner loop
2D	H_4-H_7	NI	Inner loop count
2E	H_0-H_3	DAØ	$\pm\Delta A_\emptyset$, outer loop
2E	H_4-H_7	DBØ	$\pm\Delta B_\emptyset$, outer loop
2F	H_0-H_3	DCØ	$\pm\Delta C_\emptyset$, outer loop
2F	H_4-H_7	NØ	Outer loop count

outputs as required by the operation to be performed, and places the resulting 24-bit words for input to the Level 3 Modification Adder. The following paragraphs describe functions of the major components of Level 2 of the IPU pipe.

4-33 LEVEL 2 CONTROLLER

The Level 2 Controller monitors the hazard detection circuit to detect a far range hazard, checks the status of Level 3 Controller to determine if that level can accept a transfer, and samples the instruction in Level 3 to determine its effect on Level 2. The controller then gates the contents of Level 2 into Level 3 after any necessary address modification has been performed and sets the active bit in the Level 3 Controller. Other control functions performed by this circuit are determined by the specific operation being processed by the pipe. Refer to the Level 2 Controller flowcharts and description later in this section for a complete representation of controller functions.

4-34 LEVEL 2 ROM

The Level 2 ROM receives the 8-bit operation code of the instruction word as it enters Level 2. Depending upon the Op Code, the ROM generates 32 control bits that are used in Level 2 to control instruction processing or that transfer to the C3 ROM Supplement Register in Level 3. Refer to appendix A of this manual for a listing of the ROM output bits.

4-35 R2 REGISTER

The R2 Register is a 4-bit register that receives the R field bits of the incoming instruction word and transfers them to Level 3 when the instruction enters Level 3. The output of this register is also used in Level 4 hazard detection logic to find a register hazard.

4-36 INDEXING REGISTER (XR)

The Indexing Register is a 32-bit register that receives input from one of the seven index registers in the Register File. If the instruction entering Level 2 from Level 1 indicates that indexing will be required, the 4-bit T field of that instruction selects one register in the I File for transfer to the Indexing Register. The output from XR enters a shift network. Control bits from the Level 2 ROM indicate whether the index word will be left-shifted one bit (doubleword addresses), right-shifted one bit (half-word addresses), or remain unaltered (single word addresses). The output from the shift network enters the modification adder.

4-37 DISPLACEMENT REGISTER (NR)

The Displacement Register is a 32-bit register that receives the instruction word from the Level 1 Instruction Register. Only part of the instruction is used by the displacement circuit, however. The instruction word from the Displacement Register enters a sign extension circuit. Control bits from the Level 2 ROM then determine one of two possible places for sign extension to occur. The LSB's of the resulting 24-bit word that enters the modification adder contain either the N field (bits 20 to 31) of the instruction word, or both the M and the N field (bits 16 to 31) of the instruction word. The remaining bits to the left of these fields are the result of sign extension.

4-38 P2 REGISTER

The P2 Register is a 24-bit register that holds the address of the instruction that currently resides in IPU Level 2. The address enters the P2 Register when the Level 1 Controller transfers the instruction into Level 2 and leaves the P2 Register when the Level 2 Controller transfers the instruction into Level 3. The output of the P2 Register may also transfer to the AR Register in Level 3 through the address modification network. The two comparison networks, hazard and branch, examine the contents of P2.

4-39 BASE ADDRESS REGISTER (BR)

The Base Address Register is a 32-bit register that receives the base address word from file A or B of the Register File. If the instruction word that enters

Level 2 contains an M field that is not zero, the M field bits select the output from one of the Register File registers and transfer that 32-bit word to the Base Address Register. When selected for base addressing, the Base Address Register inputs to the address modification adder. The Base Address Register is also used to transfer the first three words of the Vector Parameter File through the address modification network, and into the MBU. A select network at the output of this register allows control signals from the Level 2 ROM to select either the Base Address Register output or the output from the P2 register as the base address used in the modification addition.

4-40 IPU LEVEL 3

IPU Level 3 develops the effective address of the operand to be sent to the MBU. It receives input from the Level 2 instruction registers, adds the applicable base, displacement, and/or index, and holds the resultant address for use in Level 4. Level 3 also checks for hazards and reprocesses an instruction if a hazard exists concerning that instruction. The following paragraphs describe the function of the major blocks in Level 3.

4-41 MODIFICATION ADDER

The Modification Adder is a 32-bit (24 effective bits) parallel adder circuit with a double-level look-ahead, carry determination circuit. The adder receives inputs from the Base Address, Displacement, and Indexing Registers and adds them to form one 24-bit resultant that transfers to the Adder Resultant (AR) register when the Level 2 Controller enables the transfer. A feedback path from the AR register to the adder allows for incrementing the AR register to provide continuous octet addresses to Central Memory for Load File Multiple or Store File Multiple instructions.

4-42 ADDER RESULTANT (AR) REGISTER

The AR Register is a 32-bit (24 effective bits) register that receives the modified operand address from the Modification Adder. The output from this register may load the Level 0 addressing registers during a branch operation, indirect addressing, or an execute instruction. A feedback path to the Modification Adder provides for incrementing the address in the AR register for loading or storing multiple Register Files. If the address that enters the AR Register is an effective address of an operand (address) or an immediate operand, the contents of AR transfer to the AO Register in Level 4 under control of the Level 3 Controller. The address from AR also enters the Z model Stack (Store operation) or the Register Stack and is available to the hazard detection circuits in Level 4.

4-43 P3 REGISTER

The P3 Register is a 24-bit register that contains the address of the instruction that is currently in Level 3 of the IPU pipe. It receives the address from the P2 Register when the instruction enters the AR Register after undergoing any indicated modifications. The output of this register can be used to load the BA Register in Level 0, or can be transferred to the RO Register in Level 4 as a direct operand. In all cases, the output from this register is available to the hazard detection circuits in Level 4.

4-44 LEVEL 3 ROM

The Level 3 ROM receives the Operation Code portion of the instruction word from Level 2 as that instruction word enters Level 3 through the Modification Adder. The 8-bit Op Code produces a 32-bit output from the Level 3 ROM. This output, in conjunction with the C3 Register output, provides control bits for coordination of Level 3 processes and supplies bits to complete the address stored in the Register Stack in Level 4. If the Op Code indicates a branch, indirect or execute instruction, the Level 3 ROM triggers a comparison circuit for those operations. Refer to appendix B in this manual for a map of the contents of the Level 3 ROM.

4-45 ROM SUPPLEMENT REGISTER (C3)

The C3 Register is a 24-bit register that stores control bits from the Level 2 ROM to be used as control bits in supplement to those produced by the Level 3 ROM. The control bits enter the C3 Register when the Level 2 Controller transfers the particular instruction into Level 3. The output is immediately available to the Level 3 circuits for gating and control purposes. C3 output bits also transfer to the Register Stack in Level 4 to complete the address stored in that stack.

4-46 R3 REGISTER

The R3 Register is a 4-bit register that receives the R field bits of the incoming instruction word and transfers them to the Register Stack in Level 4 when the operand or operand address from the AR Register transfers to Level 4. The output of this register also selects a word from the Register File to enter into the RO Register in Level 4 as one of the operands needed by the MBU for transmission to the AU.

4-47 LEVEL 3 CONTROLLER

The Level 3 Controller monitors the hazard detection circuits to determine if a hazard exists for the instruction that is now in Level 3. If the hazard bit sets, the operand or address in the AR Register may not be valid. This condition causes the instruction to be re-addressed by transferring the contents of the P3 Register to the Level 0 Addressing Registers to begin processing that instruction again. The instructions currently in Levels 1 and 2 will also be re-addressed by the addressing registers following fetching of the Level 3 instruction from its memory location.

In addition, the Level 3 Controller monitors the status of the Level 4 Controller to determine if Level 4 can accept a transfer and then gates the contents of Level 3 to Level 4. Refer to the Level 3 Controller flowcharts and description later in this section for a detailed representation of the controller's functions.

4-48 BRANCH, INDIRECT, EXECUTE COMPARISONS

Whenever a Branch or Execute instruction or an address requiring indirect processing reaches Level 3 of the IPU pipe, the IPU must examine the addresses of the instructions currently in the pipe registers to determine if a new memory fetch will be necessary to obtain the desired word. The Branch, Indirect, Execute Comparison circuit performs this function in the following sequence (refer to figure 4-3):

- Compare AR with P2 (24 bits). If AR = P2, transfer Level 2 to Level 3.
- Compare AR with P1 (24 bits). If AR = P1, sequence Level 1 to Level 3.
- Compare AR octet with PA octet (21 bits). If equal, force AR to PA and LA to access new word (Branch), or use AR to select from Current Instruction File (Indirect or Execute).
- Compare AR octet with LA octet (21 bits). If equal, force AR to PA and LA to begin new sequence (Branch), or use AR to select from waiting Current Instruction File (Indirect or Execute).
- If all comparisons fail, transfer AR to OA, LA and PA to access new octet (Branch), or transfer AR to OA and use AR to select word from KCM (Indirect or Execute).

4-49 IPU LEVEL 4

IPU Level 4 is the IPU output level to the MBU. It includes an address and an operand output register, word selection logic, and a controller. Also included within the Level 4 circuits, but not solely operational within Level 4, are the hazard detection circuits. These circuits protect the IPU from processing potentially faulty instructions or operands. The following paragraphs briefly describe the major components included in Level 4 of the IPU.

4-50 LEVEL 4 CONTROLLER

The Level 4 Controller monitors the status of MBU Level 5 to determine if the MBU is ready to accept new data from the IPU. If the MBU can accept a transfer and the active bit in Level 4 Controller is set, the Level 4 Controller enables the output from the RO and AO Registers, along with control signals, to the MBU. Refer to the Level 4 Controller flowcharts and description later in this section for a complete representation of the Level 4 Controller functions.

4-51 REGISTER STACK

The Register Stack stores the resultant storage addresses of the operands in each level of the CP from IPU Level 4 through AU Level 12 (nine levels maximum). The stack is used for any instruction that passes through the AU and has a storage destination in the Register File. The Register Stack registers contain the storage address of the result as well as control bits. The destination address is normally developed from the output of the R3 Register and specific control bits from the Level 3 ROM and ROM Supplement Register (C3). However, during a Store (R) into α when α is less than or equal to 2F (in the Register File), the output from the AR Register in Level 3 supplies the destination address to the

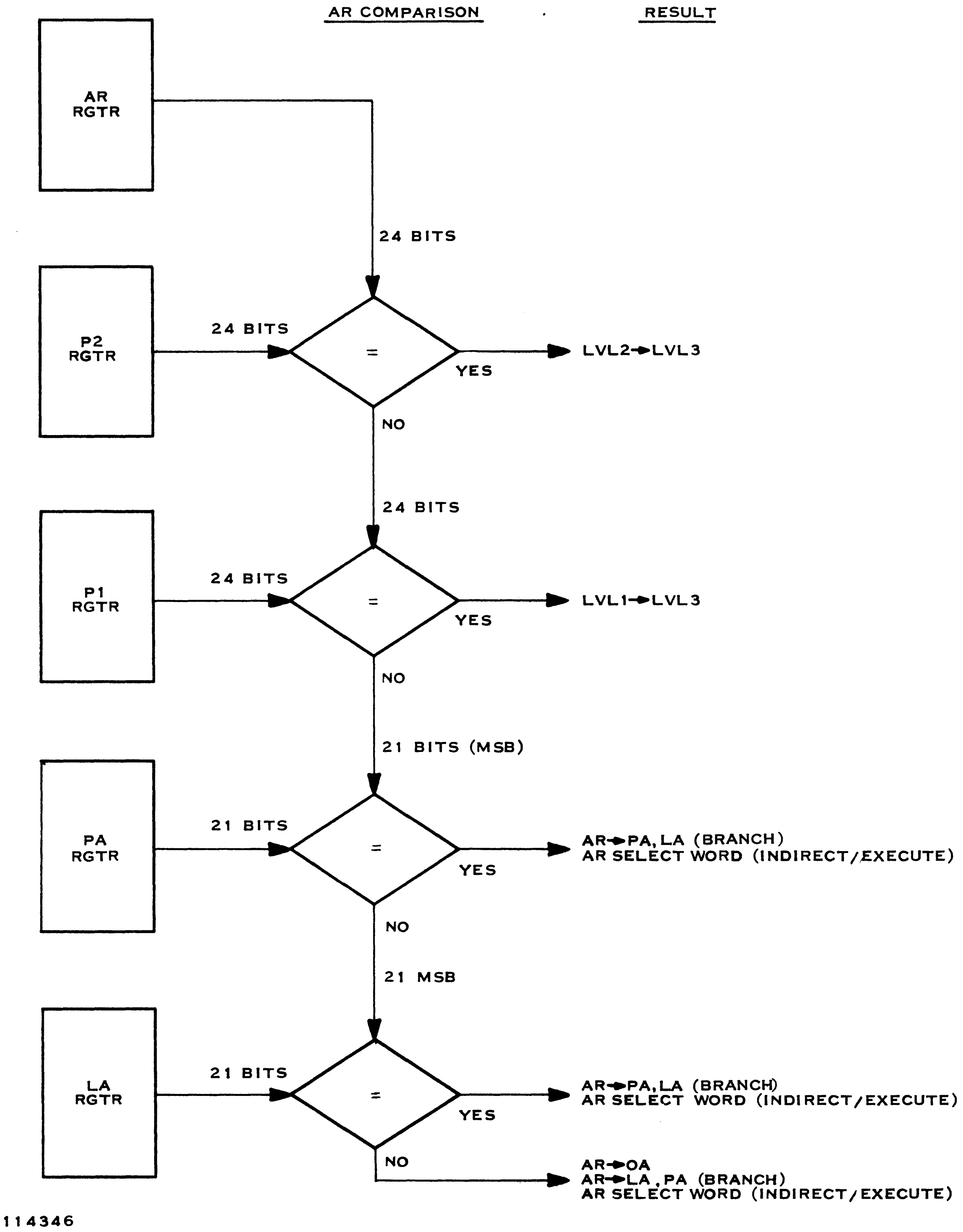


Figure 4-3. Branch, Indirect, Execute Comparisons

Register Stack. The output from the Register Stack is used for hazard detection. The Register Hazard Comparison circuit compares the contents of the Register Stack with various addresses in the IPU pipe to determine whether an instruction in the pipe will draw from a location that is to be modified by the operands preceding it in the pipe. Refer to the Register Hazard Comparison description for a more detailed discussion of the comparison circuitry.

4-52 REGISTER HAZARD COMPARISON

A register hazard exists when an instruction in the IPU accesses a register in the Register File and that register will be modified by an operation being processed elsewhere in the CP pipe. The contents of that register will not be valid data until the modification operation in the pipe is complete and the result has been stored in the Register File. The Register Hazard Comparison circuit prevents access to a register in the Register File until any instruction that modifies that register has cleared the CP pipe. The comparison circuit performs this safeguard function through the series of register comparisons illustrated in figure 4-4.

As an instruction enters Level 1 of the IPU pipe, the compare circuit monitors both the T field (index register select) and the M field (base register select) and compares these fields with the contents of the Register Stack registers to determine if the T or M field registers will be modified by an instruction in Levels 4 through 12 of the IPU. Registers R2 and R3 are also compared with the two fields to determine if the instruction in Level 2 or 3 will modify the Register File register. The AR Register in Level 3 is also compared with the T and M fields of Level 1 to detect a hazard during a Load Register File operation, where the AR Register holds the address of the register in the Register File to be loaded. If the instruction passes these tests, it moves to Level 2. If not, the instruction must wait until the hazard condition drops before it can transfer to Level 2 to select the registers from the Register File.

At Level 2 the comparison circuit checks only the AR Register in Level 3 for a hazard against the T and M fields of the instruction at Level 2. This comparison checks an address that was not generated when the instruction was in Level 1. If the instruction passes this test, it may move to Level 3.

Since the output of the R3 Register may select a Register File register for input to the R0 Register in Level 4, and the AR Register may select a Register File register during indirect addressing, these two registers are checked for a hazard conflict at Level 3 before being allowed to access the Register File. The addresses specified by these two registers are compared with the addresses stored in the Register Stack to detect a hazard condition.

4-53 AO REGISTER

The AO Register is a 64-bit register that receives the output from the AR Register in Level 3. This output may be either the memory address of an operand for use by the MBU or a direct operand (either immediate or from Register File) for transfer to the AU. Before entering the AO Register, the AR Register output (24 bits) undergoes a sign extension process to create a 64-bit input to the AO Register. The AO Register transfers its 64-bit word to the MBU when directed by the Level 4 Controller.

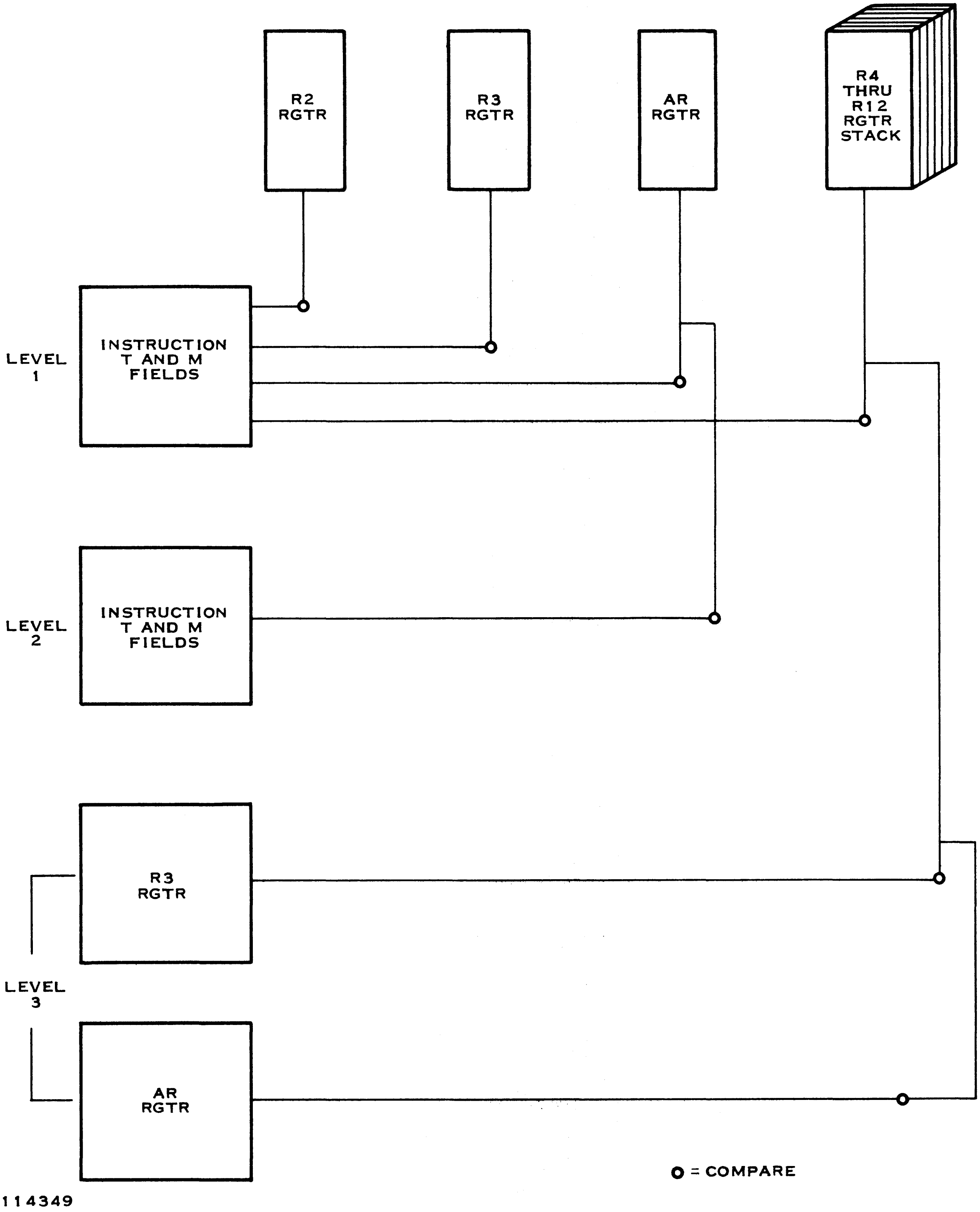


Figure 4-4. Register Hazard Comparisons

4-54 Z MODEL STACK

The Z Model is a 5-register stack that contains the destination address of all Store operations to Central Memory in the CP pipe. The address input is from the AR Register in Level 3 and enters the Z Model only if the Op Code of the instruction specifies a Store operation. The address then moves through the stack registers as the operand moves through the CP pipe. The registers in the Z Model correspond to positions within the pipe as follows:

- ZP Register. Contains the destination address of a Store operation that is currently in pipe levels 4 through 12.
- ZA Register. Contains the destination address of a Store operation that is in the MBU Z Register, having been processed by the CP pipe.
- ZB Register. Contains the destination address of a Store operation that has transferred from the Z File to the ZB File in the MBU, and is no longer available for X and Y update.
- ZO Register. Contains the destination address of a Store operation that is being sent to Central Memory.
- MA Register. Contains the destination address of a Store operation that is in the Memory Control Unit, but has not been written into its addressed location of the Memory Module.

The output from the Z Model is used to determine if a requested operand from memory is to be changed by a Store operation currently in the pipe (operand hazard). The Instruction and Operand Hazard Comparison circuits determine if any hazards exist with respect to the contents of the Z Model.

4-55 α OPERAND HAZARD COMPARISON

An Operand hazard exists when the operand addressed by the AR Register is about to be altered by a Store instruction that is farther along in the CP pipe. The hazard indicates that if the operand is acquired at the present moment, before the Store instruction is complete, a faulty operand may be obtained from memory. To avoid accessing a faulty operand, the Operand Hazard Comparison monitors the Z Model and compares its contents with the address indicated by the AR Register. This comparison is illustrated in figure 4-5.

An additional comparison is performed by this circuit to indicate whether a Z to X or Y update is necessary. The address in the ZA Register is compared with the addresses of the octets in the X and Y Buffers of the MBU. If the address in the ZA Register is within the octet in either the Y or the X Buffer, the IPU may choose to update the information in the buffers with the resultant data found in the MBU Z Register.

4-56 NEAR RANGE INSTRUCTION HAZARD COMPARISON

An Instruction hazard exists when an instruction that has been accessed by the IPU is to be altered by a Store instruction that is already in the pipe. The Near Range Instruction Hazard Comparison circuit detects an imminent instruction hazard by comparing the store operation address record in the Z Model with the address of the instruction that is about to be executed in Level 3 of the

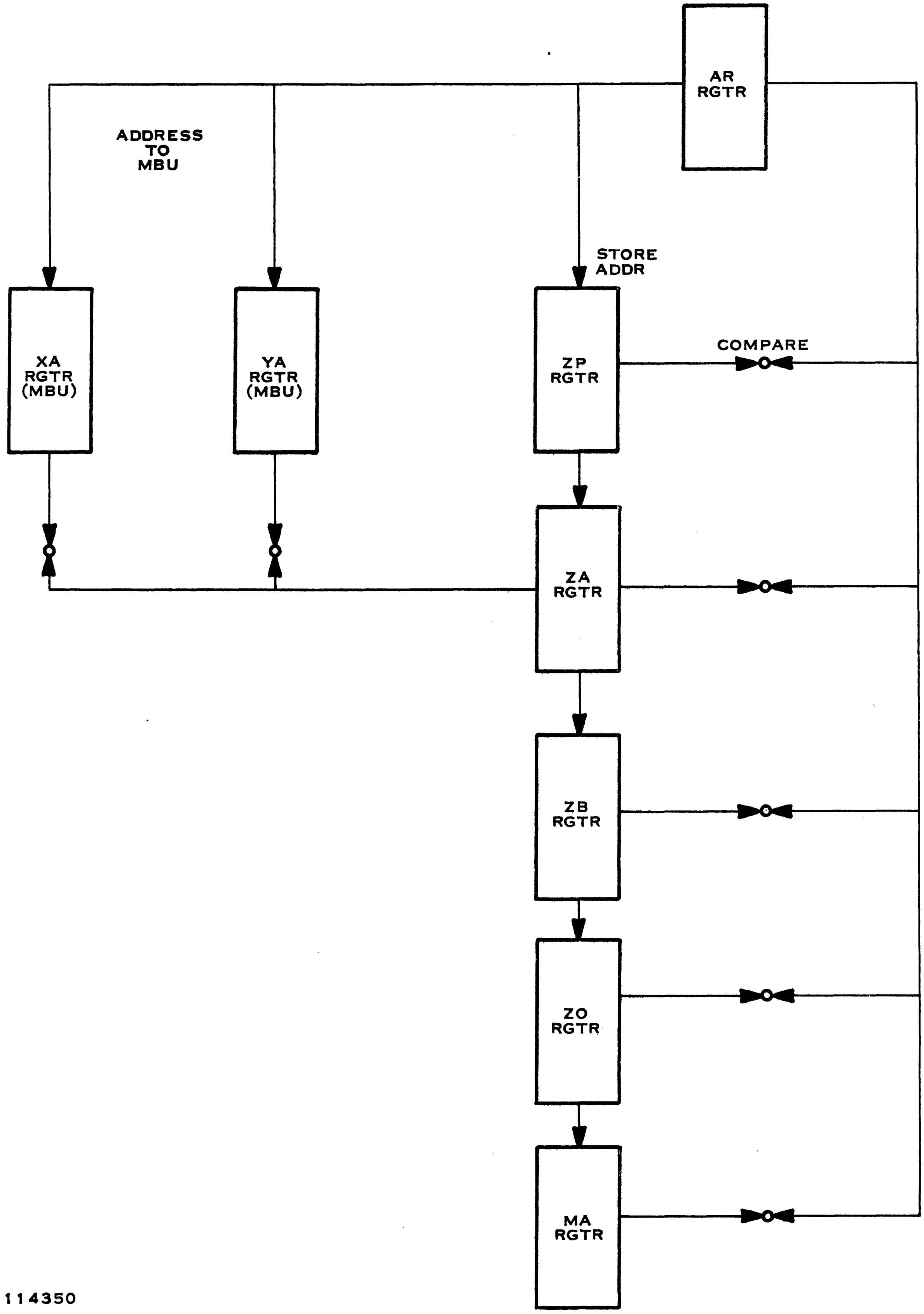
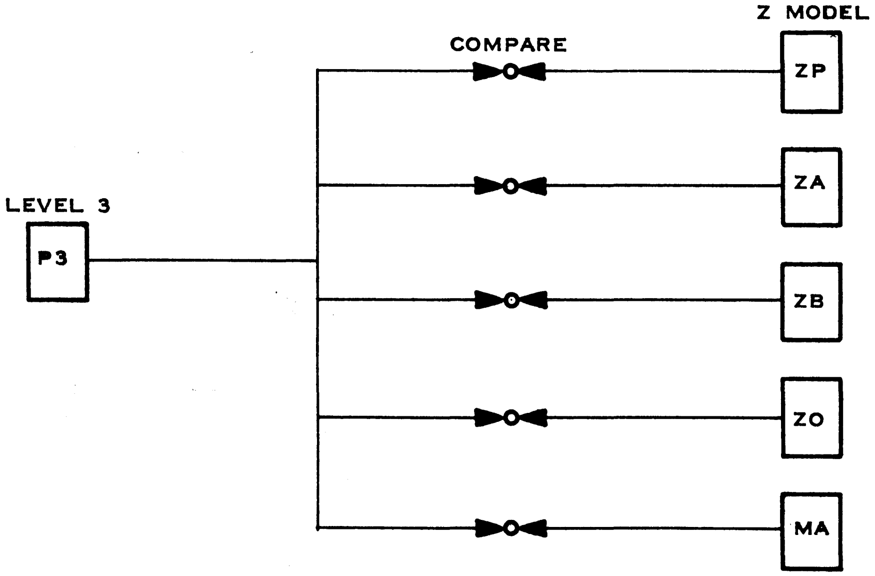


Figure 4-5. Operand Hazard Comparisons

IPU pipe. This address is contained in the P3 Register. If a near range hazard is detected by this comparison, the hazard flag is set. When the offending store instruction is finished, P3 transfers its contents to LA, OA, and PA to begin another pass at the instruction in memory. The near range comparison is illustrated in figure 4-6.

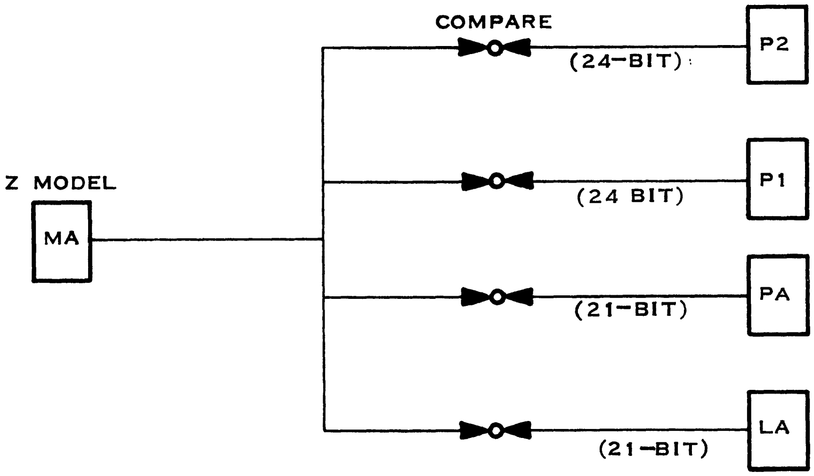
4-57 FAR RANGE INSTRUCTION HAZARD COMPARISON

A far range instruction hazard indicates that an instruction in the IPU pipe before Level 3 has been fetched from a memory location that is being changed by a previous store instruction that is writing into memory. This condition means that the instruction in the pipe is not valid. To detect a far range hazard, the comparison circuit monitors the address in the MA Register of the Z Model and compares that address with the addresses of the instructions in the P1 and P2 Registers and the octet address contained in PA and the LA Registers (refer to figure 4-7). Detection of a far range instruction hazard has no immediate effect on the IPU, as the invalid instruction may be disregarded by a branch, skip, or other diversion before it reaches Level 3. Instead of an immediate reaction, a far range hazard flag sets in the controller corresponding to the invalid instruction. This flag passes from controller to controller as the instruction moves through the IPU levels. When the instruction reaches Level 3, the Level 3 Controller checks the far range hazard flag. If that flag is set, the controller loads the P3 Register into PA, LA, and OA Registers to restart the instruction sequence.



114351

Figure 4-6. Near Range Instruction Hazard Comparisons



114352

Figure 4-7. Far-Range Instruction Hazard Comparisons

4-58 RO REGISTER

The RO Register is a 64-bit register that holds an operand for transmission to the MBU. The 4-bit R field from the R3 Register selects one word from the Register File for entry into the RO Register when the Level 3 Controller transfers information from Level 3 into Level 4. The RO Register may also be loaded from the P3 program address register for operations using a direct operand contained in the address registers. The output from the RO Register transfers to the MBU for input to the AU after the MBU fetches the second operand from memory.

4-59 MEMORY BUFFER UNIT (MBU)

The Memory Buffer Unit (MBU) receives addresses or immediate operands from the IPU. If the word is an address, the MBU requests the octet containing that address from central memory, and extracts the proper operand from that octet. In either case the MBU forwards the operand, immediate or addressed, to the AU for processing. The two operands within the MBU can be up to 64 bits long. During vector operations, the MBU buffers up to three octets from memory for each of two input buffers so that a steady input of data to the AU is ensured. The components of the MBU are illustrated in the detailed block diagram of the central processor in figure 4-1. The following paragraphs describe the function of each of those components within the MBU.

4-60 MEMORY INTERFACE FILE (SC)

The MBU Memory Interface File (SC) receives all operands from Central Memory that the MBU transfers to the AU. The file is an eight-register (octet) group with 32 bits to each register. It receives data directly from the memory data lines and holds that data until it is synchronized with the CP clock pulses. The clock pulses then transfer the data through the remainder of the MBU, subject to gating signals from the MBU controllers. The output from the SC File may enter one of many places in the MBU. During scalar operations and during vector operations when the Vector Buffer Files are empty, the SC output transfers directly into either the X or Y operand buffer. When a vector operation is in progress and the X or Y operand buffers are full, the SC output enters either the YB or the XB Vector Buffer File. Two paths supply data to the Z Storage Files also. One path provides fill-in for partially filled words for storage into memory (ZB), while the other path is used exclusively during a Load Details operation (Z).

4-61 VECTOR BUFFER FILES (XB, XH, YB, YH)

Each Vector Buffer File consists of eight 32-bit operand registers. During vector operations, the files provide continuous operands to the X and Y Operand Files for operand streaming into the AU. Two Vector Buffer Files supply two stages of octet buffering for each of the two Operand Files. Cue and control bits from the Central Memory Requester control entry into the Vector Buffer Files. The individual vector controllers (figure 4-1) gate the data between the files. When an octet arrives in the SC File, CMR determines which vector stream addressed that particular octet and gates the octet into either the X or Y data stream. The octet may enter either the B level or directly into the operand file, depending on the status of the operand file. If the octet enters the B level and the next buffer is clear (XH or YH) on the next clock pulse, a gate from the vector controller transfer the new octet into the H buffer. When the corresponding Operand Buffer File empties, another gate and a clock pulse transfer the octet from the H level to the Operand File.

4-62 OPERAND BUFFER FILES (X, Y)

Both Operand Buffer Files consist of eight 32-bit registers. These files supply operands to the MBU output registers during both scalar and vector operations. The files receive their input octet from three sources: the SC Interface File, the XH or YH Vector Buffer Files during vector operations, and the ZH Holding File. This last source of operands is used when the Z pipe contains modified entries for storage in the octet that is resident in either of the Operand Buffer Files. Flag bits record the halfwords that have changed in the octet so that only the changed portions of the octet transfer to the Operand Buffer Files during this update procedure. The output from these files is available to the MAB and MCD output registers through a selection network for input to the AU.

4-63 X AND Y WORD SELECT

The word select circuits receive inputs from their respective Operand Buffer File and use a 4-bit word address (figure 4-1), Address Generation Circuit to select a half, single or doubleword entry from the o perand octet. During vector operations, the output from the X select circuit is sent only to the MAB register and the output from the Y select circuits drives only the MCD register. No crossover of operands is possible. In scalar mode, both select circuits supply operands exclusively to the MCD register.

4-64 MAB/MCD OUTPUT REGISTERS

The MAB and MCD registers are two 64-bit registers that supply operands to the AU for processing. All operands, whether scalar or vector must pass through these registers for transmission to the AU.

4-65 SCALAR DATA PATHS. During scalar processing, the MAB Register receives operands exclusively from the REG Register. This register supplies operands from the IPU Register File if a register operand is required. The MAB register may not be used during a scalar operation if no register operands are needed. The MCD Register, however, has three sources of operands during a scalar operation. It may receive operands from Central Memory through either the X or the Y Operand Buffers, or an immediate operand from the IMM Register.

4-66 VECTOR DATA PATHS. In vector mode each output register has one main operand source: the X Operand File for the MAB Register and the Y Operand File for the MCD Register. Either output register, however, may receive an immediate vector through the IMM Register.

4-67 REG REGISTER

The REG Register is a 64-bit register that receives operands directly from the IPU during scalar mode operation. REG then holds the operand until a corresponding operand from Central Memory, or an immediate operand is available, and the Level 5 Controller enables a transfer from Level 5 to Level 6 of the CP pipe. The contents of the REG register then transfer to the MAB Register at Level 6 of the MBU.

4-68 IMM REGISTER

The IMM Register is a 64-bit register that receives immediate operands from the IPU and transfers them to the Level 6 output registers at the direction of the Level 5 Controller. The IMM Register output may transfer only to the MCD Register during a scalar operation. However, during a vector operation, the IMM Register loads the vector parameters into the MBU vector processor circuits, and may also be used to load an immediate, one-value vector into either the MAB or the MCD output register.

4-69 Z REGISTER SELECT

The Z Register Select circuit receives the resultant data from the AU and routes it to a particular register in the Z Resultant Storage File. A 4-bit element address from the Z address generation circuit (figure 4-1) designates the particular register for storing the resultant data. This circuit and the Z File pipe are used for scalar store operations and all vector operations only. All other scalar operations store results in the IPU Register File.

4-70 Z RESULTANT STORAGE FILE

The Z File consists of eight 32-bit registers that receive input data from the AU for storage in Central Memory. The Z File represents one contiguous octet of Central Memory, the actual address of which is controlled by the Z Address

Generation circuit (figure 4-1). The entire octet, therefore, may not be filled in any one operation if storage addresses do not indicate continuous memory locations. Whenever the storage addresses begin storing into a new octet, the Z Address Generation circuit transfers the contents of the Z File to the ZB File for storage and begins storing a new octet in the Z File. This transfer takes place regardless of the full status of the Z File. The input line to the Z File from the SC File is used during a Load Details operation only.

4-71 ZH HALF PHASE HOLDING FILE

The ZH File is an eight-register file used exclusively for a transfer delay between the Z File and the ZB File to avoid a premature transfer of contents from Z to ZB. Two clock pulses control the timing of transfers between the files, Phase 0 and Phase 1 clocks. Phase 0 enables input to the Z and the ZB files (along with other gating pulses from the control logic). The Phase 1 clock occurs at equal intervals to the Phase 0 clock, but the pulses are 180 degrees out of phase with the Phase 0 clock. Phase 1, therefore, represents a "half-phase" pulse with respect to the Phase 0 clock. This half-phase clock transfers the contents of the Z File into the ZH File, exclusive of all other control signals. ZH, therefore, always reflects the contents of the Z File after a half-clock delay. Because of this delay, ZH provides a stabilized output to the ZB File. At the end of an octet in the Z File, Phase 0 clock can simultaneously transfer the octet in the Z File, as reflected in ZH into the ZB File, and begin storing data for a new octet in the Z File. One half-clock later, Phase 1 clock changes the ZH File to reflect the new octet in the Z File.

If the octet in ZH is to be stored and the X or Y Operand File is currently using that octet, the changed words in the ZH octet transfer to the X or Y File to update that information before the store gate transfers the octet to the ZB File. This update path is also available if the operand address indicates an access to a word that is in the Z File. CMR will transfer the changed words from ZH to the requesting file, X or Y, to update the new octet as it enters X or Y from SC.

4-72 ZB MEMORY STORAGE FILE

The ZB File consists of eight 32-bit registers. The output from these registers supplies data to Central Memory for storage in octet transfers. Central Memory cannot store halfwords without destroying the second half of the word stored in that memory location. The ZB File receives either full or partially full octets from the Z File through the ZH File when address control begins storing AU results into a new octet. If the transfer contains a modified halfword that does not have a corresponding halfword to form a single word store, the Central Memory Requester circuit addresses memory to read the contents of the octet from its memory location into the SC Memory Interface File. Cue bits in CMR transfer the unmodified halfwords in the octet from the SC File to the ZB File to complete the storage octet. CMR then transfers the changed words into memory to replace the information stored at that location. If the ZB File receives only single words from the Z File, CMR stores the words of that octet without the fill-in process.

4-73 ROM ADDRESS REGISTER

The ROM Address Register is a 9-bit register that holds control bits from the IPU to designate the starting address of the next ROM sequence for AU gating control. During scalar operations, the nine bits have the following sources:

- Bit 0 - Designates one of two halves of the ROM. May be set to allow a new sequence without modifying the remaining address bits.

- Bits 1-4 - The four most significant bits of the current instruction Op Code.

- Bits 5-8 - A recoding of the four least significant bits of the current instruction Op Code.

The ROM Address Register receives this input during the Level 4 to Level 5 address transfer and holds the address bits until the Select Next Controller gates the output into the ROM circuits.

During vector operations, the ROM Address Register is loaded from the Vector Parameter File in the IPU. At the start of a vector operation, the operation code in the first word of the Vector Parameter File enters the ROM Address Register from the IMM Register. Since the operation code is only eight bits, the ninth bit is held at a constant "1" level.

4-74 ROM ADDRESS SELECT

Control bits from the Select Next Controller monitor the progress of the ROM sequence and designate to the Select circuit which address source to gate into the ROM circuit. The address may be derived either from the ROM Address Register or from the output of the ROM itself. The first source is used to initiate a ROM sequence; the second source continues the ROM sequence by supplying succeeding addresses from the ROM output to access the next ROM word. The ROM supplies two 9-bit addresses during sequencing (β_1 and β_2). The Select circuit may choose either of these addresses, depending upon conditions monitored by the Select Next Controller.

4-75 MBU ROM

The output from the MBU ROM controls the gating of operations through the AU pipe. The ROM supplies a 256-bit output from a 9-bit address input. In addition to AU Control bits, the ROM produces two 9-bit addresses that feed back to the Select circuit to address the next output from the ROM during sequences of more than one ROM instruction. An output map of the function codes is provided on site in computer printout form.

4-76 ROM OUTPUT REGISTER

The ROM Output Register is a 256-bit register that receives the code from the ROM and holds it for relay to the AU gating circuits. The contents of this register change each clock time to follow the output of the MBU ROM.

4-77 SELECT NEXT CONTROLLER

Select Next monitors the status of the MBU ROM sequence and determines when a new sequence may be started for the next instruction to enter the AU pipe. In streams of similar operations, a new operand may be entered into the AU before the previous operand has completed processing. The precise point of entry is determined by the nature of the operation in the pipe. When a new operation can begin in the pipe without disturbing the one in the pipe, Select Next enters the new Op Code into the ROM. The ROM then produces gate signals for both instructions concurrently. Refer to the flowchart and description of the Select Next Controller that appears later in this section.

4-78 AU CONTROL

The AU control circuit produces timing signals to the AU pipe that coordinate the gating signals produced by the MBU ROM. It receives control signals from the ROM, status from the Level 6 Controller, and a reflection of the contents of the AU pipe from the AU Model. AU control then generates the proper enable pulses to the AU to route data to the proper pipe segment at the proper time in the ROM sequence. AU control also supplies the Level 6 Controller with a Path Ahead Clear (PAC) indication so that the controller can gate data from Level 6 into the AU pipe. Refer to the AU Control flowchart and description that appears later in this section.

4-79 AU MODEL

The AU Model consists of a series of AU status flip-flops that mirror the current condition of each level of the AU pipe. Timing and control signals from the AU Controller set and clear the respective level busy bits in the model to assure a current status picture in the model at all times. The output from the model supplies status signals to the AU Controller and to the Select Next Controller to aid in coordination of AU operations and instructions.

4-80 Z DATA AND ADDRESS CONTROL

This circuit controls the transfer of data between the Z, ZH, and ZB Files and the corresponding transfer of addresses from the NSA to the ZBA Registers through the ZA Register. To determine the timing of these transfers, the circuit monitors the status of data in the AU pipe to discover when a new octet of addresses will be started in the Z File. This circuit also controls the introduction of a new storage address into the NSA Register by monitoring status commands from the IPU and gating the new address from the Z Stack in the IPU when the operand and instruction reaches Level 7 of the CP pipe. To provide immediate destination selection for resultant AU data, the addresses in the address registers precede their respective data by one clock time period as they pass through the files of the Z pipe. Flowcharts for the address and data flow through the Z pipe appear later in this section along with description of the major decision paths.

4-81 LEVEL 5 CONTROLLER

The Level 5 Controller receives status and transfer bits from the Level 4 Controller in the IPU, gates the IPU addresses into the Level 5 Registers, returns status bits to Level 4 and provides Level 6 with information for transfer coordination into Level 6. The Select Next Controller also provides a gating input to the Level 5 Controller to enable the controller to transfer new data into the next level of the pipe. A flowchart and description of the scalar input cycle appears later in this section. Vector inputs are under control of the separate vector controllers.

4-82 LEVEL 6 CONTROLLER

The Level 6 Controller receives transfer control signals from the Level 5 Controller, path status signals from the AU Controller, and a transfer enable signal from the Select Next Controller. By combining these signals, the Level 6 Controller determines when to transfer operands and ROM addresses such that no data or ROM control bits are lost. The Level 6 Controller also returns data present indications to the AU Controller to indicate an active level state. The logic flow of the Level 6 Controller in the scalar mode and vector mode is discussed later in this section.

4-83 INNER LOOP STORAGE REGISTER (NIS)

NIS is a 16-bit storage register that receives the inner loop count portion of the Vector Parameter File at the beginning of a vector operation. NIS then holds this count for restoration to the inner loop counters when a new inner loop is begun. The count held in NIS is used by the inner loop counters for all three vectors.

4-84 SELF LOOP COUNT REGISTER (LPS)

LPS is a 16-bit storage register that receives the Length portion of the Vector Parameter File at the beginning of a vector operation. LPS then holds this count for restoration to the self-loop counters when a new self-loop is begun. The count held in LPS is used by the self-loop counters for all three vectors.

4-85 VECTOR INITIALIZATION CONTROL

Vector Initialization Control performs the gating functions required to begin vector processing in the MBU/AU and also clears out the units at the completion of a vector to prepare the pipe for the next operation. At the start of a vector operation, this circuit distributes the sections of the Vector Parameter File from the IMM Register to their proper destinations within the vector control and generation circuits, starts the address generators, performs the first operand fetch of the vector, and aligns the first operands for processing. At that point vector control and generation circuits take control of the operation and run until completion.

When the vector is complete, the Initialization Control circuit again assumes control to store the last results of the vector into Central Memory. Initialization Control also cycles the MBU ROM to a NOP (no operation) condition and sequences the AU pipe to clear the gate and control flip-flops. This clean-up operation prevents alteration of the next operation that will be processed by the pipe. Refer to the flowchart and discussion of the Vector Initialization Controller that appears later in this section.

4-86 VECTOR LOOP CONTROL

Vector Loop Control consists of a 16-bit decrementing counter for each of the three address loops plus a vector controller. The controller monitors counter status and the status of the address generation network corresponding to that vector. It then controls counter decrementing and input gating to the address generator. Each of the three possible vectors, A, B, and C, have a separate and independent loop control circuit.

4-87 SELF LOOP COUNTER (FLP). FLP receives the 16-bit length field of the Vector Parameter File from LPS at the beginning of a vector operation. Control pulses from the vector controller then decrement the count in the counter each time an address is generated by the address generation circuits. If the FLP received a count of zero (NOP), or all loop counters receive a count of one (unit vector) to start the vector, the vector requires no address generation. When the counter reaches a count of one, it signals the vector controller that the FLP is "1". If additional self loops are to be executed, the vector controller loads the FLP with the original self-loop count from the LPS holding register.

4-88 INNER LOOP COUNTER (FNI). FNI receives the 16-bit inner loop count field of the Vector Parameter File from the NIS register at the beginning of a vector operation. If the inner loop count is equal to zero or one, the inner loop will not be executed; the self loop will be executed once. If the inner count is greater than one, control pulses from the vector controller decrement the count in the counter each time a new self-loop of addresses begins. When the count in the counter reaches a one, the counter signals the vector controller that the FNI is equal to "1". If additional inner loops are to be executed, the vector controller loads the FNI with the original inner loop count from the NIS holding register.

4-89 OUTER LOOP COUNTER (FNO). FNO receives the 16-bit outer loop count field of the Vector Parameter File directly from the IMM Register at the beginning of a vector operation. If the outer loop count is equal to one or zero, the outer loop will not be executed; only the self-loop and inner loops will run to completion. If the outer loop count is greater than one, control pulses from the vector controller decrement the count in the counter each time a new inner loop of addresses begins. When the count in the counter reaches a one, the counter signals the vector controller that FNO is equal to "1". Another cycle of inner and self-loops is performed to complete the vector. A new outer loop count entered into the FNO counter indicates the beginning of a new vector.

4-90 VECTOR CONTROLLER. The Vector Controller supplies increment pulses to the loop counters and gating signals to the address generation circuit to select increment values between addresses. At the beginning of a vector operation, the controller checks the value in the FLP counter. If this value is a zero (NOP) or a one (unit vector), the incoming vector requires no address generation. If the self-loop count is greater than one, the controller enables the address generation adder to increment the address. Each new address produced by the generation circuit causes the controller to decrement the FLP counter. When FLP reaches a "1" count, the controller disables the increment input to the address adder and inspects the FNI counter.

If the inner loop count in the FNI counter is equal to zero or one, vector processing is finished. If the inner loop count is greater than one, the controller enables the inner loop displacement input to the address adder to produce the starting address of the next self-loop and decrements FNI. The controller then loads the self-loop count into FLP and enables the address adder to increment through the self-loop again. This process continues until both FLP and FNI contain a count of one. At that point the controller disables the address adder and inspects the FNO counter.

If the outer loop count in FNO is equal to zero or one, the vector operation is complete. If the outer loop count is greater than one, the controller enables the address adder to add the outer loop displacement to the last address and decrements the outer loop counter. Both the inner loop count and the self-loop count restore to FNI and FLP, respectively, and the self- and inner loop sequence repeats. When the outer loop counter reaches a count of one, a final repetition of the self- and inner loop sequence completes the vector.

When the controller determines that the vector is finished, it disables the address generation circuit and informs the Vector Initialization Controller of the status. Vector Initialization Control then stores the remaining results in memory and clears the pipe for the next instruction.

4-91 MBU UNIT HARD CORE

MBU Unit Hard Core is the maintenance function and context switching controller for the MBU. It receives maintenance and switching commands from the PP through Master Hard Core. It then performs the specific operation independent from the other two unit hard cores in the CP. When the operation is complete, MBU UHC reports the completed status to MHC. Among the functions performed by Unit Hard Core are Load and Store Details or Intermediate operations, and Unit Register Read when the register to be transferred to the PP is in the MBU Unit Hard Core. Refer to the discussion of maintenance commands for flowcharts and explanation of the hard core operations.

4-92 VECTOR ADDRESS GENERATION (A/B VECTORS)

The address generation circuits for the A and B vectors are functionally identical. Each circuit produces addresses to access vector components from memory for their respective vectors. At the start of a vector operation, Vector Initialization Control loads the vector starting address field into the Vector Address Register (NAA/NBA). The octet address of this word transfers to the Octet Request Register (XBA/YBA). If it is a new octet, the vector controller generates a request to the Central Memory Requester (CMR). The 4-bit word address within an octet is stored into the vector Circular Address File (CAF).

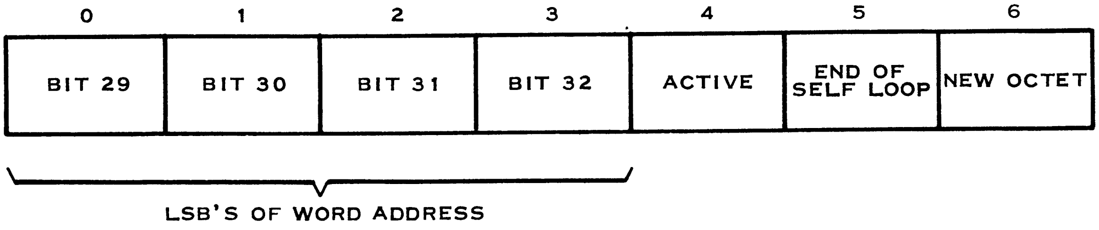
The word addresses are then retrieved sequentially (first in - first out) to select from their corresponding octet for input to the MAB/MCD Registers. As each word address enters CAF, the entire address feeds back through the address adder to create the next address in the vector. The following paragraphs describe each major component of the address generation circuit. Refer to the Address Generation Controllers discussion later in this section for description and flowcharts of the process.

4-93 VECTOR ADDRESS REGISTER (NAA/NBA). This register receives the 25-bit vector starting address field from the Vector Parameter File at the start of a vector operation. This field enters from the IMM Register under control of the Vector Initialization Controller. The Vector Address Register then holds that address until modified by input from the Address Adder or a new starting address field. The 21 most significant bits of this register (octet address) transfer to the Octet Request Register for transmission to memory. The four least significant bits are stored in the CAF for use in routing the proper operand to the data stream. The output from the register is then applied to the input of the Address Adder for possible modification.

4-94 ADDRESS ADDER. The Address Adder performs address modification on the address from the Vector Address Register to form a stream of addresses to access the component operands of a vector. Any one of three modification inputs may be added to the contents of the Vector Address Register to produce the next address to be accessed. Selection of the particular input is under control of the Vector Controller in the Vector Loop Control circuit. If the loop counters indicate a self-loop is to be performed, the controller enables the DAS input. This input is a fixed "1", so that it produces an increment or decrement of one, depending upon the sign. Similarly, for changing the address to begin a new self-loop sequence (inner loop) the DAI input is added to the address; for beginning a new inner loop sequence (outer loop) the DAO input is added to the address. The output from the adder transfers to the Vector Address Register to supply the next address in the stream.

4-95 OCTET REQUEST REGISTER (XBA/YBA). The Octet Request Register receives the 21-bit octet address from the Vector Address Register and makes it available to the Central Memory Requester for transmission to Central Memory. The individual vector controller monitors the contents of this register and produces a request to CMR when a new octet address enters the register. When CMR accepts this request and transfers the octet address to the OA Register, the Octet Request Register is free to accept a new address. During scalar operations, a direct path from the IPU A0 Register loads the Octet Request Register.

4-96 CIRCULAR ADDRESS FILE (CAF). CAF is a file of 16, 7-bit registers. Together with its input and output controllers, it keeps track of all vector components whose addresses have been requested from memory but have not been used by the operand selection circuit that loads the MAB and/or MCD Registers. The file has a capacity of 16 unused operands, so that when this limit is reached, the address generation circuit is disabled until one of the operands is used, creating a vacancy in the file. The seven bits of the file words are divided as illustrated in figure 4-8. The four most significant bits in the word are the element address bits from the Vector Address Register. Each time a new address enters the Vector Address Register, the four least significant bits of the address transfer to the CAF and fill the four MSB's of the next vacant file word. When the bits enter the file, the input controller sets bit 4 of the



0	1	2	3	4	5	6
BIT 29	BIT 30	BIT 31	BIT 32	ACTIVE	END OF SELF LOOP	NEW OCTET

(A)115817

Figure 4-8. Typical [redacted] Word

file word to indicate that the element address is active. When the word is used, the output controller clears bit 4 to indicate that that word is now vacant. Bit 5 is a control bit that indicates that the word is the end of a self-loop. This bit is set every time that the Vector Controller enables the DAI or DAO input to the Address Adder. Bit 6 is set by the input controller to indicate the first word of a new octet. When the octet address in the Octet Request Register differs from the octet address in the Vector Address Register, the controller sets bit 6. The words remain stored in sequence in the file until an operand request removes the address from the file.

Each time the MBU needs a new operand to send to the AU, the output controller transfers one of the file words to the Buffer Operand Address Register (XA/YA) and clears the active bit in the file corresponding to the word that was removed. The output controller assures that the file words are removed in exactly the same order that they were put into the file so that they remain matched with their proper octets that arrive from memory. Refer to the controller discussion later in this section for a flowchart and description of the address output cycle.

4-97 BUFFER OPERAND ADDRESS REGISTER (XA/YA)

The Buffer Operand Address Register is a 4-bit register whose output selects the proper word from an octet in the X or Y Operand Buffer for transfer into the MAB or MCD output registers. This register receives its element address from either the CAF in the case of vector streams, or directly from the IPU AO Register during scalar operations.

4-98 C VECTOR AND STORAGE ADDRESS GENERATION

This network generates memory addresses for storing the resultant vector (C) from a vector operation and processes the storage address of a scalar Store operation. For either operation, the addresses are coordinated with their proper data octets in the Z Pipe so that the address is sent to memory when the corresponding data octet is in the ZB File. In addition, this network records the modification status of each halfword in the Z pipe files for use in an update of the X or Y Buffer Files. The following paragraphs describe the major component circuits of this network. Refer to the detailed theory discussion later in this section for a flowchart of the address generation circuit.

4-99 C VECTOR ADDRESS REGISTER (NCA). NCA receives the 25-bit C Vector starting address field from the Vector Parameter File at the beginning of a vector operation. The Vector Initialization Controller enables this input through the IMM Register during vector start-up. The output of NCA is used to generate further addresses for storing the vector and also transfers the address to the CMR through the ZA/ZBA registers for use in storing the resultant vector in Central Memory.

4-100 ADDRESS ADDER. The Address Adder modifies the address in the Vector Address Register, NCA, to form a stream of addresses for storing the resultants of a vector operation. Any one of three modification inputs may be added to the contents of NCA to produce the next address for resultant storage. Selection of the particular input is under control of the C Vector Controller in the C Vector Loop Control circuit. If the loop counters indicate a self-loop is to be performed, the controller enables the DCS input. This input is a fixed "1" that produces an increment or decrement of one to the address in the register. Similarly, for changing the address to begin a new self-loop sequence (inner loop) the output from the DCI Register is added to the address; to begin a new inner loop sequence (outer loop), the DCO Register output is added to the address. The output from the adder transfers to the C Vector Address Register to supply the next address in the stream to the CMR.

4-101 SCALAR STORAGE ADDRESS REGISTER (NSA). NSA is a 25-bit register that contains the storage address of a scalar operation resultant that is currently in the AU pipe. A 21-bit input from the IPU ZP Register forms the octet address portion (most significant bits) of the storage address. Four bits from the IPU Register Stack (level R6) become the element address that is used to store the word within the octet contained in the Z File. The address enters NSA when the respective operands enter the AU pipe and leaves NSA to transfer to ZA and ZEA one clock before the resultant of the operation leaves the AU EF Register.

4-102 RESULTANT STORAGE ADDRESS REGISTER (ZA). ZA is a 21-bit octet address register. The contents of this register indicate the destination storage address of the octet that is currently in the Z File. The address enters ZA from NSA (scalar) or NCA (vector) one clock before the resultant begins storing into that octet in the Z File. The output of the register then transfers to the ZAH Register on the following Phase 1 clock pulse.

4-103 HALF PHASE HOLDING REGISTER (ZAH). ZAH is a 21-bit octet address register that reflects the address in ZA after a one-half clock delay. Refer to the discussion of the ZH Holding File for a description of the half phase relationship of the two registers. The output of ZAH transfers to the ZBA Register when a new octet address enters the ZA Register.

4-104 MEMORY STORAGE ADDRESS REGISTER (ZBA). ZBA is a 21-bit octet address register that contains the address of the octet that is currently in the ZB File awaiting transfer to Central Memory. The address transfers to ZBA from ZAH one clock before the corresponding octet transfers from the ZH File to the ZB File. An address that enters ZBA immediately results in a memory request to the CMR. Since store requests have priority over all other memory requests, CMR immediately begins a memory cycle to store the contents of the ZB File into the memory location contained in the ZBA Register.

4-105 HALFWORD MODIFIED INDICATOR REGISTER (ZM). ZM is a 16-bit register that is used to keep a record of those halfwords in the Z File that have been modified by resultants from the AU. Each bit in ZM corresponds to one-half word in the Z File. When an entry is made into a particular halfword in the Z File, the corresponding indicator bit in the ZM Register is set. The Modified Halfword Detect circuit monitors each element address sent to the Z File and generates the signal that sets the control bit in ZM.

4-106 HALF PHASE HOLDING REGISTER (ZMH). ZMH is a 16-bit register that reflects the contents of the ZM Register after a one-half clock delay. The output bits gate the changed halfwords into the X or Y File during a Z to X (Y) update.

4-107 MEMORY STORAGE MODIFIED HALFWORDS REGISTER (ZBM). ZBM is a 16-bit register that stores the modification indicator bits for the octet of data in the ZB File. It receives its input from the ZMH Register one clock before the data file transfers from ZH to ZB. If an update of X or Y is needed during the time that its corresponding octet is still in ZH, the output of ZBM may be used to gate the modified words of that file to the corresponding words in the X or Y Files. Since memory can only consider whole word storage, the output of this register is merged into eight whole-word-modified indicator bits before being transferred to CMR as zone control bits.

4-108 STORAGE WORD ADDRESS REGISTER (ZEA). ZEA is a 4-bit register that receives the element address from either NCA (vector) or NSA (scalar). The output of this register directs the output from the AU EF Register into the proper word of the octet currently residing in the Z File. ZEA receives a new input address as soon as the old address has performed its gating function.

4-109 CENTRAL MEMORY REQUESTER (CMR)

CMR performs the coordination, bookkeeping and transfer functions required to store or fetch octets in Central Memory. It assures that octets read from memory are routed to the data file that requested that particular octet. For store functions it also assures that only valid full words of data are stored into memory to avoid destroying information in Central Memory. Three addressing circuits provide requests for memory access to CMR. If conflicts occur between these requests, CMR resolves the conflict. Refer to the controller discussion that appears later in this section for flowcharts of CMR cycles.

4-110 CMR PRIORITY GATE. This circuit monitors memory requests generated by the vector controllers during vector operation, or from the IPU during scalar operation, and resolves any conflicts that may occur over simultaneous memory requests. Since storage requests have no address buffer, storage requests involving the ZBA Register address always receive the highest priority for memory requests. Conflicts involving the addresses from XBA and YBA are resolved alternately; that is, a request for access to the location in XBA receives access to memory for the first conflict with a request from the Y pipe, but the second conflict is resolved in favor of the Y pipe request. The Priority Gate then transfers the address from the highest priority channel to the OA Address Register.

4-111 MEMORY OCTET ADDRESS REGISTER (OA). OA is a 21-bit register that receives memory addresses from one of three input registers and holds that address until CMR Control completes processing the request. Once the address enters OA, the output is immediately available to the AA Register.

4-112 ASYNCHRONOUS ADDRESS REGISTER (AA). AA is a 21-bit address register that provides request addresses to Central Memory. Transfer of an address from OA into AA depends only upon Central Memory's acceptance of the address previously held in AA. AA is not dependent upon clock pulses.

4-113 HALFWORD BITS CHECK AND MERGE. This circuit receives the 16 "halfword modified" flags from the ZBM Register and processes them for two purposes. First, the circuit checks each pair of bits that represents the two halves of a whole word. If only one of the pair of bits is set, it indicates that only half of that word contains valid data in the ZB File. The other half, being unwritten into, contains unknown and undesirable data. The circuit, therefore, signals the CMR Controller that a Z Fill-in process is required (refer to CMR Controller description). The circuit then examines the bit pairs again and inputs a "1" bit to the ZCB Register for each pair of bits having at least one bit set. This produces a set of flags in ZCB that indicates which whole words in the ZB File have been changed.

4-114 ZONE CONTROL BIT REGISTERS (ZCB/AZC). ZCB and AZC are 8-bit registers that hold the zone control bits corresponding to the octet residing in the ZB File. Each zone control bit corresponds to one word in that octet. A set bit indicates that its word in ZB contains data that resulted from the AU pipe operations. A cleared bit indicates that no data is stored in that word. These bits enter ZCB when the address of the storage data enters the OA Register. AZC is an asynchronous register so that the control bits transfer to AZC as soon as Central Memory accepts the last group of control bits. Central Memory stores only those words in the ZB File whose corresponding zone control bit is set in the AZC Register. Valid data stored in memory is thereby not destroyed by writing unmodified (blank) words from ZB over the valid data.

4-115 CMR CONTROL. CMR Control coordinates all memory requests, routes the resulting octets to the requesting file, and assures that only valid data is stored into Central Memory. During a fetch from memory, CMR gates the request address to memory and sends a 2-bit code to the Cue File that indicates the destination for the octet when it returns from memory (X, Y or Z pipes). When an octet enters the SC File from memory, CMR gates the octet to the proper file by accessing the cue bits in the Cue File.

At the start of a memory store operation in CMR, the controller inspects the halfword bits check circuit to determine if the octet to be stored contains any half complete words. If all the words in the octet are complete, the octet is written into memory using the zone control bits from the AZC Register to determine which words in the octet are to be written. If, however, one or more of the words in the octet are only partially filled with new data, CMR Control initiates a Z Fill-in operation.

Z Fill-in results in supplying complete words to memory. CMR Control issues a fetch request to Central Memory for the octet in the location corresponding to the address of the octet to be stored. When the octet enters the SC File, CMR Control transfers it to the ZB File using the cue bits from the Cue File. The "halfword modified" control bits from the ZBM Register prevent the incoming octet from replacing those halfwords that are already in the ZB File. The incoming octet, therefore, only stores into the vacant halfwords of the ZB File. When the ZB File is complete, CMR Control issues a request to memory to store the octet in ZB. The zone control bits sent with it to Central Memory, however, allow only the modified words to be stored, preventing alteration of valid data that is already stored in memory.

4-116 CUE FILE. The Cue File is a 2-bit circular file with an eight-entry capacity. Each time the CMR Controller sends a memory request for a read from memory, it also generates a 2-bit code, designating the destination of the requested octet, and enters that code in the Cue File. The cue codes are as follows:

00 = octet to X Buffer Files

01 = octet to Y Buffer Files

10 = octet to ZB File (Z Fill-in).

Since memory requests are processed by Central Memory on a first-in, first-out basis, no tag bits are required to identify the cue entries with their octet. As octets return from memory, the cue entries are accessed from the Cue File in the order that they were stored. They are then used to gate the incoming octet to the proper file.

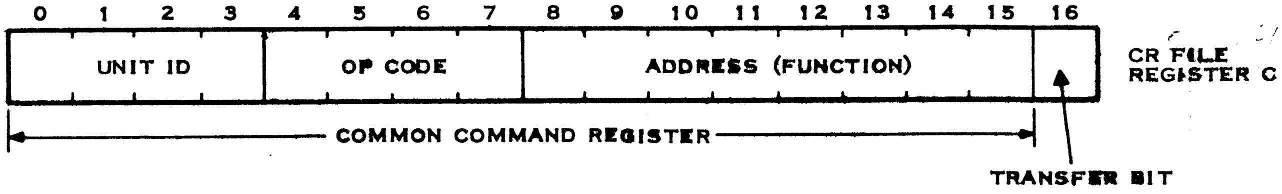
4-117 MASTER HARD CORE (MHC)

Master Hard Core (MHC) is the communications port and control channel between the Peripheral Processor (PP) and the Central Processor (CP). MHC contains connections to the Unit Hard Core (UHC) for each of the three units of the CP, the IPU, the MBU and the AU. These connections carry maintenance commands and switch commands to the UHC's and status, response and data feedback lines from the UHC's. Figure 4-1, the CP Block Diagram, illustrates the major circuits involved in Master Hard Core. The following paragraphs describe the functions of each of these components. Refer to the discussion of maintenance commands for further detail regarding MHC.

4-118 CAPTURE COMMON COMMAND REGISTER (CAPTURE CCR)

Capture CCR monitors the Common Command Register in the PP Communication Register File (CR File) in an asynchronous mode. The CCR (figure 4-9) is a 16-bit portion of the CR File Register C_{16}. It contains a 4-bit unit identifier, a 4-bit Op Code, and 8 bits of address that further define the function to be performed. In order for this register to be active, the Transfer Bit (bit 16 of CR File Register C) must be set.

Capture CCR first monitors the Transfer Bit for an active PP command. When the Transfer Bit sets, Capture CCR examines CCR bits 0-7 to determine if the active command is intended for the CP. If these bits represent a hexadecimal



(A)115836

Figure 4-9. Common Command Register and Transfer Bit

41, Capture CCR transfers the eight least significant bits of CCR to a holding register and clears the Transfer Bit in the PP. The function code transfers to Sequence Control, Sequence Control decides whether to gate the four least significant bits to the CCR Output Register, or if it must generate a new code to the UHC's.

4-119 MCW, MCP AND ERROR MONITOR

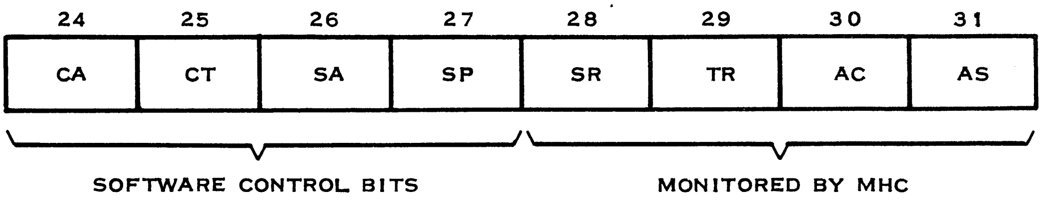
This circuit receives error indications from all three units of the CP, examines the IPU for a Monitor Call and Wait (MCW) or a Monitor Call and Proceed (MCP) instruction, and checks the 8-bit CP Control Register in the CR File. From these inputs, the circuit determines when switching of CP contents should be performed. If the switch is not possible, it generates a three bit reason code to the PP. Sequence Control enables the reason code to the PP, or performs the actual switching operation when the Monitor indicates that the switch is necessary.

4-120 CP CONTROL REGISTER. The CP Control Register consists of eight flag bits residing in bits 24-31 of the CR File Register A (see figure 4-10). Table 4-2 lists each of these bits and their definitions. These bits are used by the PP to condition the responses of the CP Master Hard Core to status conditions.

4-121 CP SWITCHES. Besides the switch initiated by an MCW instruction, the Monitor circuit checks the three CP units for the following error conditions and performs a switch when it detects an error if permitted by the PP:

- System Error (SYSERR). Terminate (TR) or Parity (PE) Errors
- Error Condition (ERR). Protect Violation (PV), or Illegal Operation (IL) or Arithmetic Exception (AE).

4-122 MONITOR RESPONSE. When the Monitor detects an error or call, it responds to the PP by setting bits in the CP Response Register (see figure 4-11), and initiates an operation in the Sequence Control circuit if the operation is enabled. If the context switch cannot be performed, it sets a reason code in



115839

Figure 4-10. CP Control Register

Table 4-2. CP Control Byte Bit Definitions

Bit	Name	Function
CA	CP Available	Set by Master Controller when no CP step is primed. Indicates the need to poll for activity on the CP execution queue. Reset when a CP step is primed.
CT	CP Test	Set by Master Controller to indicate CP control is being relinquished to MCD. When set, the Master Controller will not respond to any other activity in the CP Response or Condition bytes.
SA	Step Active	Set by Master Controller when a CP step is initiated. Reset when a step terminates and no step is primed for execution.
SP	Step Primed	Set by Master Controller to switch a CP step into execution. Reset when CP begins executing step and no other step is ready for priming.
SR	System Reset	Set by Master Controller to initiate a CP reset. Must be reset before any other CP action is taken.
TR	Terminate Request	Set by the Master Controller to terminate outstanding CCR or Automatic Switches in the CP. Reset by the MC.
AC	Allow Call	Set by the Master Controller to permit Automatic MCP and MCW calls. Reset to inhibit these calls. Should be reset anytime a CCR command is used that invalidates the next job step status defined by pointers 16, 17, and 28.
AS	Allow Switch	Set by the Master Controller to permit automatic MCW and Error context switching. Reset to inhibit these switches.

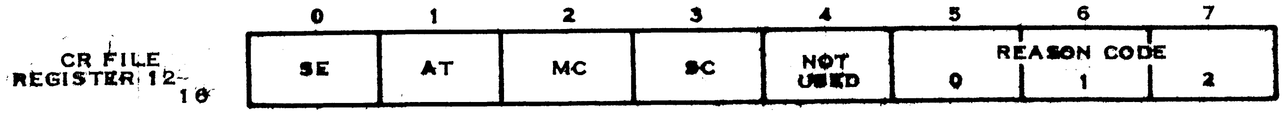


Figure 4-11. CP Response Byte

the CP Response Register to tell the PP why the switch is not performed. Table 4-3 defines the bits of the CP Response Byte, including a decoding of the Reason Code bits. Refer to the controller descriptions later in this section for a flowchart of the PP status checking cycle of the response bits.

4-123 SEQUENCE CONTROL

Sequence Control monitors error and call status from the Monitor circuit, status and response bits from the three CP units, and eight control bits from the CR File CP Control Register. It also receives the 8-bit function code from the Capture CCR circuit. From these inputs, Sequence Control determines the required steps to carry out the function prescribed by the PP. In performing this function, Sequence Control generates condition and response bits to inform the PP of current conditions. Four response bits transfer to the CR File CP Response Register. These bits are described under the Monitor circuit description. Eight condition bits, generated by Sequence Control transfer to the CP Condition Register in the CR File, Register 12_{16}. Table 4-4 lists and defines these condition bits; figure 4-12 illustrates the CP Condition Byte.

In addition to status reports, Sequence Control generates control signals to the CP Unit Hard Cores to coordinate performance of the PP Functions. If the hard core function originates from a CP error, Sequence Control produces a 4-bit code that transfers to the CCR Output Register in lieu of a PP produced command. Sequence Control flowcharts are included in the controller discussion later in this section.

4-124 CCR OUTPUT REGISTER

CCR Output is a 4-bit register that holds the operation code required by the CP Unit Hard Cores to perform any maintenance or switch function. The input may be directly from the Capture CCR circuit. However, when an error produces a switch condition, Sequence Control generates the 4-bit input to the CCR Output Register. Input from Capture CCR is a transfer of the four least significant bits of the CCR command.

Table 4-3. CP Response Byte Bit Definitions

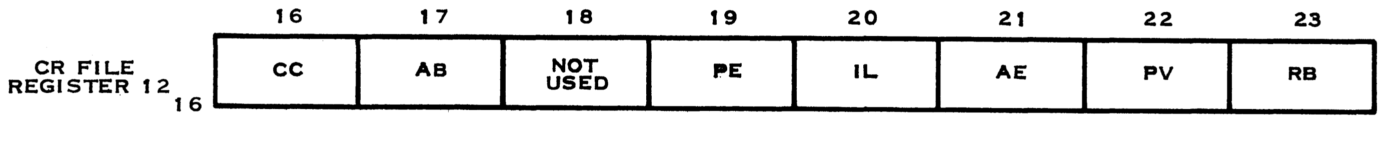
Bit	Name	Function
SE	System Error	Set by CP to indicate a Parity Error during normal CP operation. Reset by Master Controller.
AT	Attention	Set by the CP to indicate an abnormal termination (as defined by the Condition Byte) of an automatic call or switch (as defined by the MC and SC bits). Reset by the Master Controller.
MC	Message Complete	Set by the CP to indicate the completion of an MCP or MCW. Reset by the Master Controller after operation on the message.
SC	Switch Complete	Set by the CP to indicate the completion of an MCW or Error switch. Reset by the Master Controller after priming the next switch.
RZ(0-2)	Reason Codes	Set by the CP to inform the Master Controller of the following context switch conditions:

CODE	INTERPRETATION
000	NOOP
001	MCP inhibited by MC or SC bits being set.
010	MCW inhibited by MC or SC bits being set.
011	Error switch inhibited by MC or SC bits being set.
100	MCW inhibited by AC = 0.
101	MCP inhibited by AC = 0.
110	MCW inhibited as AS = 0.
111	Error switch inhibited by AS = 0.

The Master Controller resets these bits and sets the CP Run Bit via a CCR command after preparing for the indicated condition.

Table 4-4. CP Condition Byte Bit Definitions

Bit	Name	Function
CC	Command Complete	Set by the CP to indicate the completion of the last requested CP CCR command. Reset by the Master Controller.
AB	Abnormal	Set by the CP to indicate the last requested CP CCR command terminated abnormally Reset by the Master Controller.
PE	Parity Error	Set by the CP to inform the Master Controller of a CM Parity Error. Reset by the Master Controller.
IL	Illegal Operation	Set by the CP to inform the Master Controller that an illegal operation code forced or attempted to force an Error context switch. Reset by the Master Controller.
AE	Arithmetic Exception	Set by the CP to indicate that an arithmetic exception forced or attempted to force an Error context switch. Reset by the Master Controller.
PV	Protect Violation	Set by the CP to indicate that a CM protect violation forced or attempted to force an Error context switch. Reset by the Master Controller.
RB	Run Bit	Continuously gated CR bit reflecting the state of the CP's internal run bit.



(A)115846

Figure 4-12. CP Condition Byte

4-125 UNIT REGISTER READ

Unit Register Read performs the maintenance function of addressing a hard core (unit or master)register or series of flip-flops, so that the contents of the selected register are transferred to the CR File for inspection by the PP. All bits selected in this manner transfer to the CP Unit Register in the CR File, Register A (bits 8-15). The selected byte transfers directly from its respective Unit Hard Core to the Unit Register.

4-126 AU INPUT

The Input section of the AU is pipe level 7 of the Central Processor. It receives the original operand inputs from the MBU, plus numerous other inputs that are developed within the AU and fed back for further processing. Of significant interest is the additional input from the EF Output Register. This input supplies the 'short-circuit' feedback that is used for performing consecutive instructions on the same set of operands. It allows the result of one operation to be used as the input for the next operation without the delay that would be required if the first result were stored and then re-accessed. Control signals from the AU Control ROM in the MBU select the proper input and transfer it to the AB or CD Operand Registers. These 64-bit registers hold the operands until further control signals route the output to the AU level corresponding to the first operation to be performed.

4-127 EXPONENT SUBTRACT

The Exponent Subtract section determines which of the two input operands is larger and, in the process, performs a 7-bit subtraction of the operand exponent bits. The circuit is used to input properly ordered data and a right shift count to the Aligner and Right Shift section for aligning the smaller operand to the larger. It may also be used in simple arithmetic compare operations to designate the larger operand. The following paragraphs summarize the functional blocks of the Exponent Subtract section. Figure 4-13 provides a flowchart of the comparison logic.

4-128 INPUT SELECT

Control bits from the AU Control ROM in the MBU select the proper pair of inputs for a designated operation. Each input is 64-bits; however, during half or single word operation, only 16 or 32 bits may be active, leaving the remaining bits ineffective.

4-129 SUBTRACT EXPONENTS AND COMPARE MAGNITUDE

This circuit performs a comparison of the magnitude of the input operands by subtracting the exponent portion of the data word. If the result of the subtraction (X - Y) is positive, then operand X is greater than operand Y. Operand X is then gated to the LOR Register; operand Y, to the SOR Register; and the "greater than" compare code bit is set. If the result of the subtraction is zero, the circuit compares the magnitude of the two mantissas to determine which operand is larger. If the two mantissas are equal, the "equal to" compare code

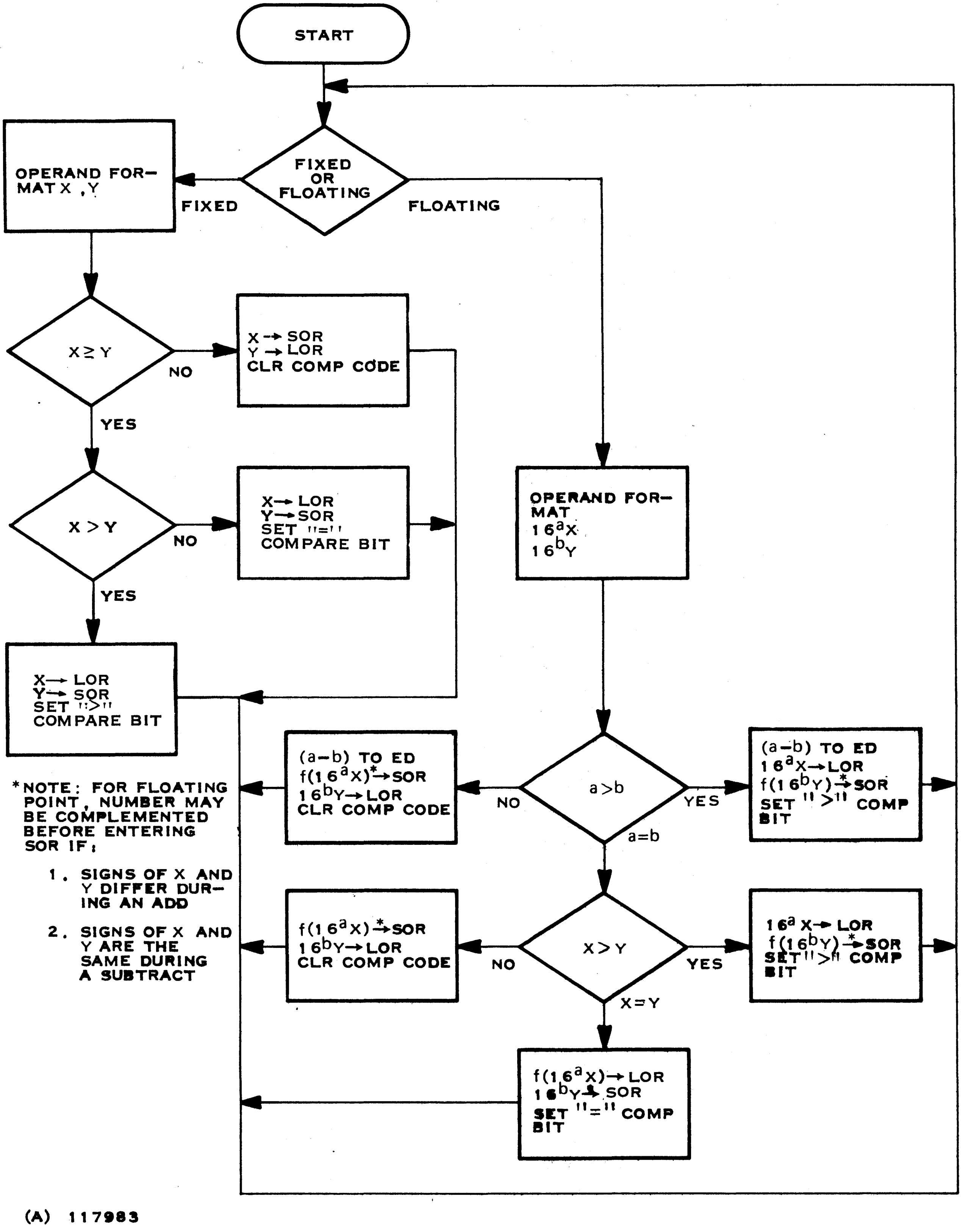


Figure 4-13. Exponent Subtract and Compare Logic Flowchart

bit sets, the X operand transfers to the LOR Register, and the Y operand transfers to the SOR Register. If they are unequal, either no bits or the "greater than" compare code bit sets; the incoming operands are routed to the proper output registers. If the result of the subtraction is negative, operand Y is greater than operand X. Operand X transfers to the SOR Register and Y enters the LOR Register. No compare bit sets, indicating that operand X is neither greater than nor equal to operand Y. The result of the exponent subtraction always enters the ED Register for output to the next stage of the AU pipe.

This circuit also complements floating point numbers in preparation for effective subtraction. That is, if the signs of X and Y are different during a Floating Point Add, the operand will be complemented (one's complement) before entering the SOR Register. Similarly, if these signs of X and Y are the same during a Floating Point Subtract, the operand will also be complemented before entering the SOR Register. The carry-in bit to complete the two's complement is added in the Adder section of the pipe.

4-130 LOR REGISTER

The LOR Register is a 64-bit register that holds the larger of the two input operands for output to the alignment section of the pipe. Control signals from the Subtract and Compare circuit select which operand to transfer into the LOR Register.

4-131 SOR REGISTER

The SOR Register is a 64-bit register that holds the smaller of the two input operands for output to the alignment section of the pipe. Control signals from the Subtract and Compare circuit select which operand to transfer into the SOR Register. The contents of SOR are right shifted in the Align circuit to make the exponent portion of the two operands (SOR and LOR) equal.

4-132 ED REGISTER

The ED Register is a 7-bit register that receives the result of the exponent subtraction process. The output of this register indicates the magnitude of the difference between the SOR and LOR operands and, therefore, indicates how many bits the SOR operand must be right shifted to align with the LOR operand.

4-133 COMPARE CODE

The Compare Code consists of two flag bits. One bit is set to indicate that the X operand is larger than the Y operand; the other bit sets to indicate that the two operands are equal. If neither bit is set, the X operand is neither larger than nor equal to the Y operand and must, therefore, be smaller than the Y operand.

4-134 ALIGN AND RIGHT SHIFT

This section is used for all floating point add instructions or for any right shift instruction. It performs floating point alignment in one pass through its circuitry and processes fixed point right shifts in two passes. A not-shifted holding stage is also supplied to maintain coordination of two corresponding operands as they pass through the section. The following paragraphs describe the major functional divisions of the Align and Right Shift circuit.

4-135 SELECT

The Align and Right Shift circuit has two select circuits. One select circuit supplies a 64-bit operand to the Hex Shift circuit for processing. The other select circuit provides a parameter input that indicates the magnitude of the required shift. This parameter may be specified by bits 25-31 of the CD Operand Register for a fixed point right shift, by bits 25-31 of the Accumulator Output Register for a fixed-floating or a floating-fixed conversion, or by the 7-bit output of the ED Register for a floating point add operation. The ED Register input is a base 16 number. It specifies only a shift of 4-bit multiples, or hex shift. The other inputs are base 2 numbers and specify both a hex shift and a bit shift. For this reason, floating point add alignment requires only one pass through the circuit for hex shifting, whereas the other shift operations require both a hex shift and a bit shift to complete the operation.

4-136 HEX SHIFT DECODE

This circuit receives the shift select bits from the select circuit and generates the required gating signals to perform the specified hex shift. Since there are 16 possible hexadecimal shifts for a 64-bit word, 16 separate gate signals are required to allow for any specified right shift. These 16 gates specify hex shifts from zero to 15 hexadecimal characters to the right.

4-137 BIT SHIFT DECODE

This circuit monitors the two least significant bits of the shift select network. During operations requiring a bit shift in addition to a hex shift, this circuit produces one of four possible gate signals to shift the hex shifted operand between one and four bits to the right.

4-138 SHIFT SEQUENCE

The Shift Sequence involves the bit shift network, the hex shift network, and the SH Shifted Operand Register. The incoming data word is first hex shifted, and the hex shifted result is stored in the SH Register. This output is available to the adder circuit for a floating point add at this time. If, however, a bit shift is required, the hex shifted operand in SH is fed back into the shift circuit gates. The bit shifted result then appears in SH.

Depending upon which hex shift gate bit is active, any one of 16 inputs to the ARSH16 FF (figure 4-14) may be enabled. The next clock pulse will transfer that enabled input to the ff and provide a hex shifted output. If a bit shifted result is required, one of the bit shift gate bits will enable the output from one of four of the SH Register ff's to be fed back as an input to the SH Register. The second clock period transfers that bit into the SH Register to complete the shift operation.

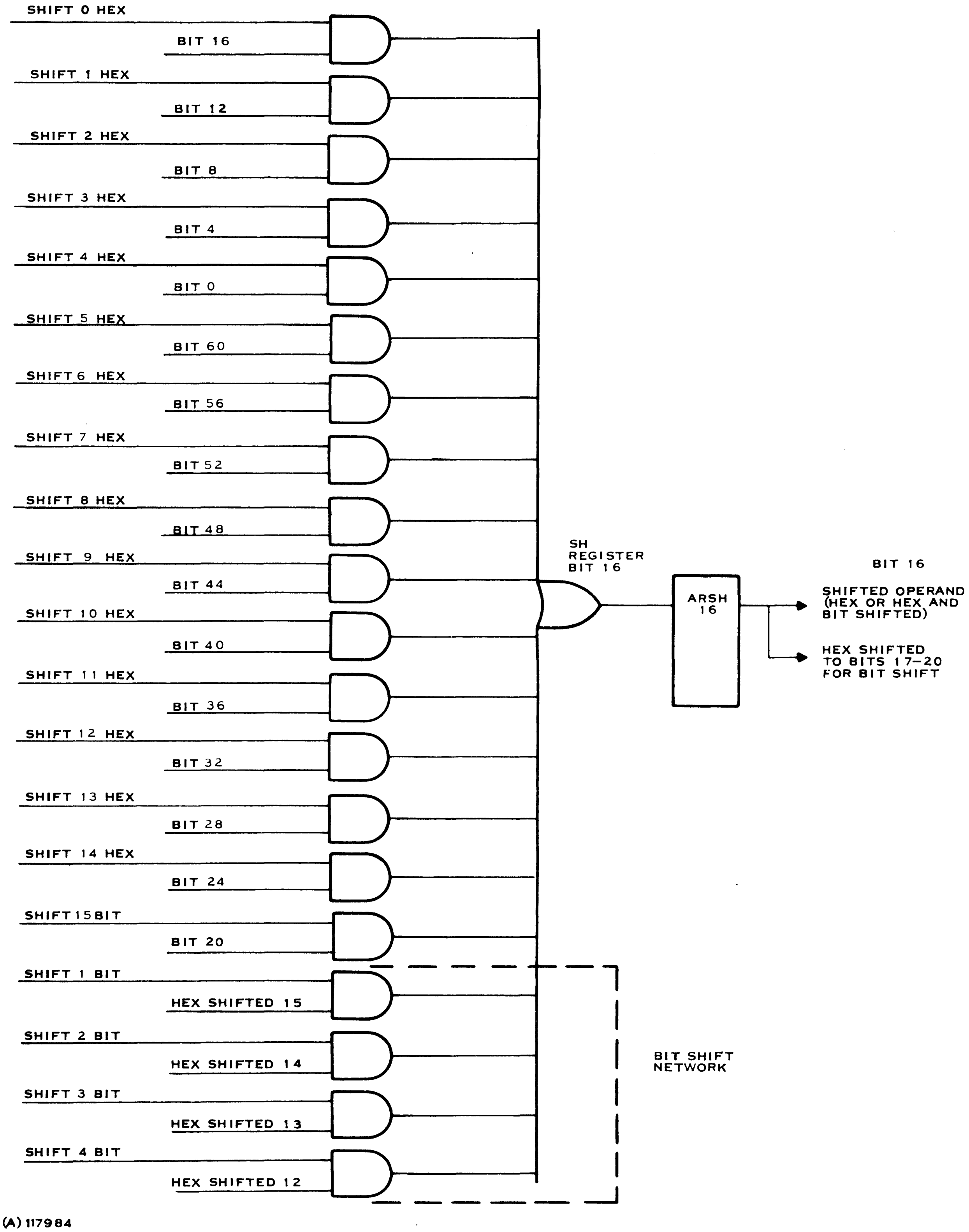


Figure 4-14. Simplified Right Shift Network (Bit 16 of Operand)

4-139 NOT SHIFTED REGISTER (NS)

The pipeline structure of the AU requires that both operands of an operation be at the same level of the pipe at all times to avoid confusion or loss of data. Therefore, even though the larger operand of a floating point add requires no action of the Align and Right Shift section, a holding stage must be provided for that operand while the smaller operand is being hex shifted. For this purpose, NS receives the output from the LOR Register and holds that operand until the hex shift of the smaller operand is complete. Both operands then transfer to the Add section of the pipe for the addition portion of the floating point add instruction.

4-140 ADDER SECTION

The Adder section performs both addition and subtraction operations on either fixed or floating point operands. It is capable of processing two 64-bit operands during one clock period. The following paragraphs describe the functions of the major components of the AU Adder Section.

4-141 INPUT SELECT

Two select gates provide the pair of operands to the adder circuit. Control signals from the AU Control ROM in the MBU designate the proper gate signals to the Select circuits. For floating point adds and subtracts, the input from the Align and Right Shift section is selected. The Input section of the AU supplies operands for fixed point operations. If the adder is to perform a subtraction, control signals select the input corresponding to the complement of the subtrahend. This number will be a simple one's complement; an additional carry input to the adder creates the two's complement input required for a subtract operation.

4-142 ADDER

The adder performs 64-bit addition on two operands in a parallel mode. Two-level, look-ahead logic determines the carry of the total operation and adds it to the partial sum to form the sum that is transferred to the ADD Register. For subtraction operations, one of the input operands is in one's complement form. An extra bit is added to the least significant bit position of the addition to develop an equivalent to a two's complement for performing the subtraction.

4-143 ADDER OUTPUT REGISTER (ADD)

The Adder Output Register is a 64-bit register that receives the resultant sum or difference of an adder operation and forwards that answer to the next level of the AU pipe as determined by control signals from the AU Control ROM in the MBU. The result from floating point operations is sent to the Normalize section before being placed into the EF Output Register. Fixed point results transfer directly to the EF Register.

4-144 ACCUMULATOR

The Accumulator is a special purpose adder section used to total the output of the Multiplier section for vector dot products and for other functions that require a running total. The following paragraphs describe the basic functions performed by the accumulator.

4-145 OPERAND SELECT

The Accumulator has three operand select circuits that provide inputs to the adder portion of the circuit. Two or three of the select circuits may be enabled at one time. Control signals from the AU Control ROM in the MBU enable the select circuits to select the proper operands. One of the select gates provides a wrap-around path from the result sum in the ACC Register so that new incoming data may be added to the contents of ACC to form an accumulated total in ACC.

4-146 ADDER

The Accumulator adder is a 64-bit parallel adder with double level look-ahead logic for determining the carry of the addition operation. The adder has three inputs, all of which may be active at one time. Refer to the Multiplier section description for information regarding a similar 3-input adder tree used in that section. The output of the adder is stored in the ACC Register for output to the next level in the AU.

4-147 ACCUMULATOR OUTPUT REGISTER (ACC)

ACC is a 64-bit register that receives the output from the accumulator adder and holds that output until a new result is entered from the adder. The output from this register is available to other stages of the AU as directed by the AU Control bits from the ROM in the MBU. A feedback path from this register to the input select circuit allows the contents of ACC to be added to incoming data to form an accumulated total.

4-148 OUTPUT

The AU output section gates all AU results to either the MBU for vectors and store operations or to the IPU Register File. It receives the output signals from all AU sections except the Multiplier section. Control signals from the AU Control ROM in the MBU select the input to this section and determine the destination of its output. This section also performs basic Boolean logical functions and reports status conditions to the IPU. The following paragraphs describe the major components of the AU Output section.

4-149 LOGICAL OPERATIONS

The Output section receives operands directly from the Input section to perform four Boolean logical operations on them. These operations are always performed on all operands that pass through the Input section. The output corresponding to the particular function need only be selected to place the logical result into the EF Output Register. The logical operations performed by this circuit are Logical AND, Logical OR, Exclusive OR, and Equivalence.

4-150 OUTPUT SELECT

The Output Select circuit uses control signals from the MBU ROM to select the proper resultant to place into the EF Output Register. The circuit may select the output from any AU section, except the Multiplier whose output is a partial result that must be added by the Accumulator to be meaningful. Three miscellaneous word inputs provide for transferring status bits or other messages to the IPU or to the MBU for storage into Central Memory. Other inputs provide a word of all zeroes, two 11-bit entries of ones, and 32 least significant bits of the ACC Register for input to the 32 most significant bits of EF.

4-151 EF REGISTER

The AU Output Register (EF) is a 64-bit register that holds the resultant output of the AU process until it is either transferred to the Register File in the IPU (scalar operations) or to the MBU for storage into memory (vectors and store operations). The output of this register is also fed back to the AU input section for use by the next following instruction if that instruction addresses the register address of the result in the EF Register. This "short circuit" path saves the time required to fetch the result from its register storage location if the instruction immediately following it will require that same result as an operand.

4-152 COMPARE CODE

The Compare Code performs two functions of comparison. In both cases it provides three mutually exculsive flags to the IPU for determining whether two operands are equal or if one is larger than the other. The first usage receives the comparison flag bits from the Exponent Subtract section and uses those bits to set the X greater than Y, less than Y, or equal to Y flag. Only one flag can be set at one time. The second function monitors the logical compare circuit and sets the compare code to indicate if the result of that comparison is mixed ones and zeroes, all ones, or all zeroes.

4-153 RESULT CODE

The Result Code is a 3-bit, mutually exclusive flag set that indicates whether the result contained in the EF Register after an arithmetic operation is positive, negative, or equal to zero. During logical operations, these flags indicate if the result is all ones, all zeroes, or mixed ones and zeroes.

4-154 ARITHMETIC EXCEPTION CELLS (AE)

This series of flags monitors various error bits in the AU and sets a flag if an arithmetic exception occurs. Types of arithmetic exceptions include: fixed point overflow, floating point overflow, floating point underflow, and divide check (attempt to divide by zero).

4-155 NORMALIZE SECTION

The Normalize section is employed for both floating point add instructions and fixed point left shift instructions. Divisors are routed through this section to assure bit normalized inputs for divide instructions. The section performs essentially the same as the Align and Right Shift section, except that this section must also determine the length of a hexadecimal floating point shift during normalization. During left shifts, however, the instruction word specifies the length of shift, so that the operation of this circuit is then analogous to the Align and Right Shift section. This circuit also employs a 7-bit adder to update the exponent for normalizing processes. The following paragraphs describe the major functional blocks of the Normalize section.

4-156 INPUT SELECT

The Normalize section has two input select circuits. The first circuit determines which input will be entered into the normalize logic to designate the length of shift required to normalize the input. Control bits from the MBU ROM control the selection of the input word. The output of this select circuit is divided into two parts: the exponent and the mantissa.

The second select circuit determines the input to the shifter. The input may be the mantissa output from the first select circuit during a normalize operation. In this case a 4-bit guard digit is also added to the mantissa to avoid loss of data. During a fixed point left shift, the output from the input section is selected to the shifter. For these instructions, the AB operand is the number to be shifted and the CD operand contains the shift parameters.

The guard digit added in the second select circuit consists of the four least significant bits and is used to avoid the loss of one hexadecimal digit of accuracy resulting from truncation prior to double length addition or subtraction. Four bits are sufficient, since the only times normalization may produce a loss of accuracy, it requires a shift of only one hexadecimal digit. Normalized operands are required for the guard digit to be of maximum use. For example, when multiplying two normalized operands, the fractions will be between 2^{-4} and 2^{-1}. The result will be between 2^{-8} and 2^{-2}. Therefore, the result will always require no more than one 4-bit shift to normalize the fraction to between 2^{-4} and 2^{-1}. During an addition, if the exponents are equal, no alignment is required. Therefore, the guard digit is not necessary. If the exponents differ by one, the guard digit will retain significant information. Finally, if the exponents differ by more than one, it can be shown that the result to be normalized will require no more than one hexadecimal shift. Thus, the guard digit contains information that can be retrieved.

4-157 MOST SIGNIFICANT 1 SEARCH

This logic searches the incoming 56-bit mantissa, beginning with the most significant hexadecimal digit, for the first 1 bit in the number. While searching, it totals the number of hex digits that have been checked until the 1 bit is found. This total defines the number of left hex shifts required to normalize the number. From this total the circuit enables one of 16 hex shift gates in the Hex Shift Network to perform the hex shift for normalization. In order to adjust the exponent to fit the normalized mantissa, the shift count is fed to the Left Shift Code Register, where the count is added to the exponent.

4-158 LEFT SHIFT CODE REGISTER

This register is 5-bit holding register that stabilizes the shift code determined by the Significant 1 Search circuit and inputs that code into the exponent adder. The five bits in this register correspond to the five most significant bits of the exponent. The output from this register is also used during shift instructions to determine if the requested shift produces an overflow.

4-159 EXPONENT ADDER

The Exponent Adder is a 7-bit adder that receives the exponent portion of the data to be normalized and adds the hex shift count to it. The result represents the adjusted exponent that corresponds to the shifted mantissa. An additional flip-flop parallel to the adder holds the sign bit during the addition, so that the sign bit remains unmodified.

4-160 LEFT SHIFT HEX DECODE

During a Left Shift instruction, the CD operand from the AU Input section specifies the parameters of the left shift. This circuit receives the 7-bit shift count from that operand and enables one of 16 possible hex shift gates to the Hex Shift Network after decoding the incoming shift count to determine which shift gate to activate. The shift count parameter input to this circuit is also used to detect a possible overflow in the shift network.

4-161 HEX SHIFT NETWORK/BIT SHIFT NETWORK

Both shift networks are electrically integrated, but functionally separate. That is, although both shift networks employ the same type of circuitry and funnel through the same gates into the NORM Register, the hex shift must be performed before the bit shift. Refer to figure 4-15 for a simplified representation of the shift circuitry for one bit of the operand.

An incoming operand to be left shifted, whether for normalization or for bit shifting, must first pass through the hex shift network. One of 16 hex shift gates, turned on by the Significant 1 Search or the Left Shift Hex Decode circuit, enables an input gate to each bit of the NORM Register corresponding to the number of hexadecimal digits involved in the shift. For normalization, processes, except bit normalization, the process is then complete. The output of the NORM Register is available to the circuit requiring normalized data.

During a left shift or bit normalization, the output of the NORM Register is fed back through a four-gate input to the NORM Register. One of four select signals enables one of the four input gates for each bit of the operand, resulting in a shift of zero to three bits to the left. The bit shifted operand is then available to other AU levels from the output of the NORM Register.

4-162 NORMALIZED OUTPUT REGISTER (NORM)

NORM is a 64-bit register that receives the results of the normalizer shift networks and holds them for output to the other levels of the AU, or to the bit shift network in the Normalize section. Inputs enter directly from the shift gates without enabling pulses, so that when an operand passes through the shift gates, it immediately enters the NORM Register. During floating point normalize operations, eight input bits (one sign bit plus seven exponent bits) from the Exponent Adder fill-in the exponent portion of the floating point data word in the NORM Register. The mantissa portion of the data word (56-bits)

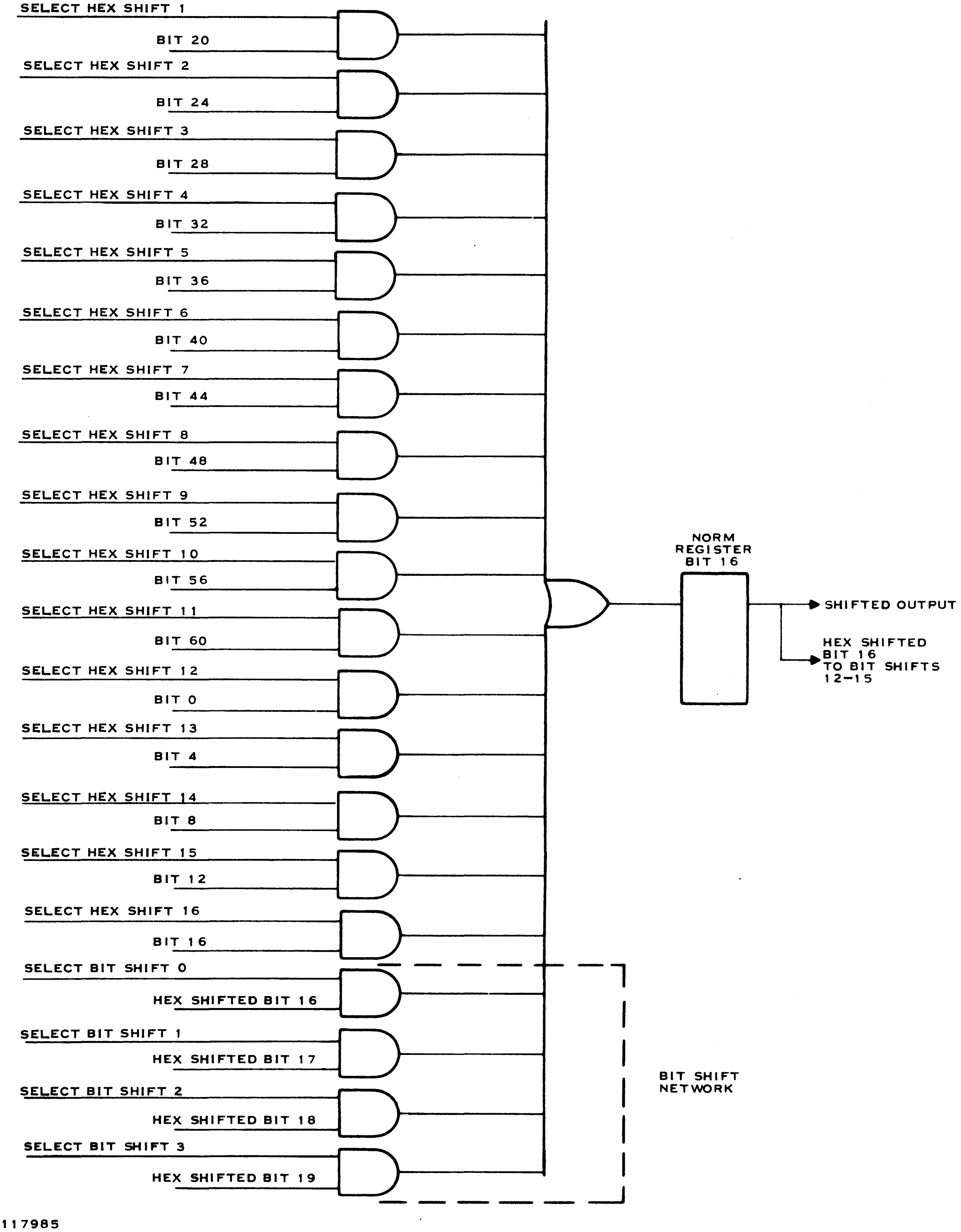


Figure 4-15. Simplified Left Shift (Normalize) Network (Bit 16 of Operand)

passes through the Hex Shift Network before entering the NORM Register. During left shift operations, the entire 64-bit word for the NORM Register enters from the Hex Shift Network, or from the Bit Shift Network following the second pass.

4-163 LEFT SHIFT BIT DECODE

Bit Decode receives the two least significant bits of the CD Operand Register and generates one of four input gating signals to the bit shift network. The input gating signals result in enabling a left shift of between zero and three bits. By combining the 4-bit shift with a hex shift from zero to 16 hexadecimal digits, any magnitude of left shift may be performed from zero to 63 bits.

4-164 BIT SHIFT MAGNITUDE DETERMINATION

During bit normalization, the Normalize section must determine the magnitude of the bit shift. To accomplish this function, a magnitude determination inspects the most significant hexadecimal digit of the hex shifted operand to locate the most significant 1 bit in that hex digit. By counting the number of zeroes preceding that 1 bit, this circuit determines the magnitude of left shift required to bit normalize the number. This circuit then generates one of four input gate signals to the Bit Shift Network to produce the required bit shift.

4-165 BIT SHIFT ENCODE AND REGISTER

This circuit monitors the most significant hexadecimal digit of the NORM Register during a left shift operation and produces a 2-bit code that indicates the number of bit shifts that can be performed on that number before an overflow will occur. The 2-bit code indicates a shift from 0 to 3 digits to the left, depending upon the position of the most significant one bit in the hexadecimal digit. A 2-bit register in this circuit holds the shift code for input to the Overflow Check circuit.

4-166 OVERFLOW CHECK

This circuit determines if an overflow of significant data occurs during a left shift instruction. By the nature of a normalize operation, no overflow can occur during that process. The seven bits of the CD operand that indicate the magnitude of the left shift enter this circuit at the beginning of the operation. Control signals then route the AB Operand through the Significant 1 Search network in addition to transferring it to the Hex Shift Network. The Search circuit creates a 5-bit code that indicates the maximum possible hex shift before data will be lost due to an overflow. This function is identical to determining the hex shift required for normalization. Overflow Check compares this code with the requested hex shift in the CD Operand. If the requested shift is greater than the maximum shift, the Overflow Flag sets. If a bit shift is to be performed, the Bit Shift Encode and Register circuit supplies a 2-bit code that specifies the maximum bit shift allowable before an overflow will occur. Overflow Check compares this code against the requested bit shift and sets the Overflow Flag if the requested shift is greater than the allowable shift. In any case, performance of the shift is not interrupted. An Arithmetic Exception cell in the Output section of the AU informs the IPU of the error.

4-167 MULTIPLIER SECTION

The Multiplier section of the AU performs both multiplication and division on 32-bit operands for the ASC Central Processor. Doubleword operands are processed as two single words; the results are then combined to form the doubleword result. Division is performed as a series of reciprocal multiplications. The following paragraphs describe the functional blocks that perform the multiplication. Following the block diagram descripton are two sections that discuss the theory of the multiplication and division processes in this section.

4-168 DIVIDEND REGISTER

The Dividend Register is a 64-bit register that holds the number to be divided during division. One of two inputs may supply the dividend to this register. The Normalizer section output supplies initial input for all division operations. The Accumulator input allows for loading the result of one multiply or divide immediately into the dividend register to start a new iteration.

4-169 DIVISOR REGISTER

The Divisor Register is a 64-bit register that holds the number to divide into the dividend. One of two inputs supply operands to this register: the normalizer section of the AU supplies normalized floating point inputs, while the Accumulator section supplies results from a previous multiplication or divide for starting a new iteration of the division process.

4-170 P-TERM LOGIC

The P-Term Logic uses the seven most significant bits of the divisor as an address to access a table location within the logic. The table contains 5-bit numbers that are approximations of the reciprocal of the input address. Therefore, each input of seven bits from the Divisor Register results in placing five bits into the Modifier Register that are an approximation of the reciprocal of the divisor. This reciprocal is accurate to five bits so that the maximum estimation error is less than or equal to 0.00001_2. The estimation error is eliminated in significance through several iterations in the division operation. The 5-bit P-term enters the five most significant bits of the Modifier Register; the remaining bits of the register are zeroes.

4-171 MODIFIER REGISTER

The Modifier Register is a 64-bit holding register for input to the Recode circuit during a division. At the start of a divide, the 5-bit approximate reciprocal of the divisor enters the Modifier Register in the five most significant bits. The multiplier multiplies the divisor and then the dividend by the fraction in the Modifier Register as the first steps in the production of a quotient. The Modifier Register also stores intermediate multiplication terms throughout the division process.

4-172 MULTIPLICAND/MULTIPLIER SELECT

The Multiplicand/Multiplier Select circuits provide 32-bit inputs to the Fanout and Recode circuits, respectively, during both multiply and divide operations. During multiplication, the AB operand input to the Multiplier Select and the CD Operand input to the Multiplicand Select circuits are enabled. These are, however, 64-bit inputs. Control signals from the AU Control ROM direct these circuits to enable either the most or the least significant half of the 64-bit input for output to their respective circuits.

During division, three inputs are available to the fanout circuit, depending upon the stage of the division process that has been reached. Refer to the discussion of the division process in the Multiplier circuit for the particular gating sequence. Three inputs are also available to the Recode circuit during a divide: two through the Modifier Register and one directly from the complemented output of the Accumulator Register. The gating of these signals is also discussed in the division process explanation.

4-173 RECODE

The Recode circuits inspect the incoming 32-bit multiplier word in three bit segments, as illustrated in figure 4-16. Sixteen separate recode circuits (R_0 - R_{15}) are required for a single word multiply; an additional circuit (R_{DV}) is used to generate a fraction summand during a division. R_{DV} checks the most significant bit of the multiplier. If it is a one it copies the multiplicand into the DV summand; if it is a zero, it complements the multiplicand and enters it into the DV summand. Each of the other recode circuits operates on its respective bits identically to the other decode circuits to activate one or none of four control lines to the Form Summand circuit. In general form, Recode circuit R_N monitors the 3-bit segment XYZ of the incoming multiplier word (Y is an even power of 2). The equivalent equations in table 4-5 define the states of X, Y and Z that produce each of four output control signals and the void state of no output control signals.

4-174 FANOUT

The Fanout circuit receives the 32-bit Multiplicand word from the Multiplicand Select circuit and duplicates it 16 times. The output from this circuit supplies 16 identical 32-bit words to the Summand Formation circuit to be used in

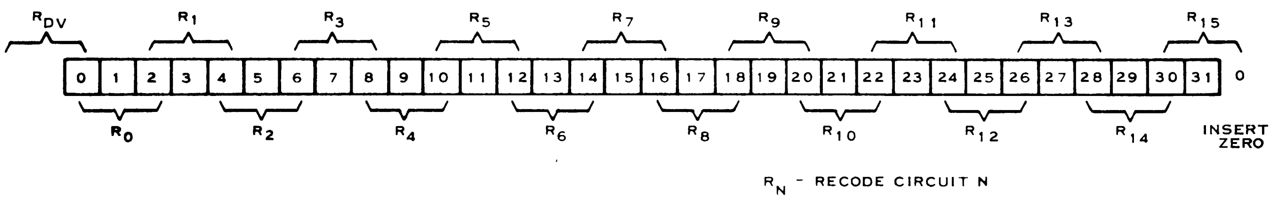


Figure 4-16. Multiplier Word Recode Bit Assignments

Table 4-5. Recode Output Control Signal Definitions

Signal	Equation	Control Function
R_NP1	$= \overline{X}\ (Y \oplus Z)$	Copy Multiplicand into Summand N
R_NP2	$= \overline{X}*Y*Z$	Left shift Multiplicand by one bit and enter shifted word into Summand N
R_NM1	$= X\ (Y \oplus Z)$	Enter the two's complement of the Multiplicand into Summand N
R_NM2	$= X*\overline{Y}*\overline{Z}$	Form the two's complement of the Multiplier, left shift the complement by one bit, and enter the result into Summand N.*
Void (all signals are zero)	$= X*Y*Z\ +$ $\overline{X}*\overline{Y}*\overline{Z}$	Load zeroes into Summand N.

*Refer to Summand circuit description for a refinement of the complement process.

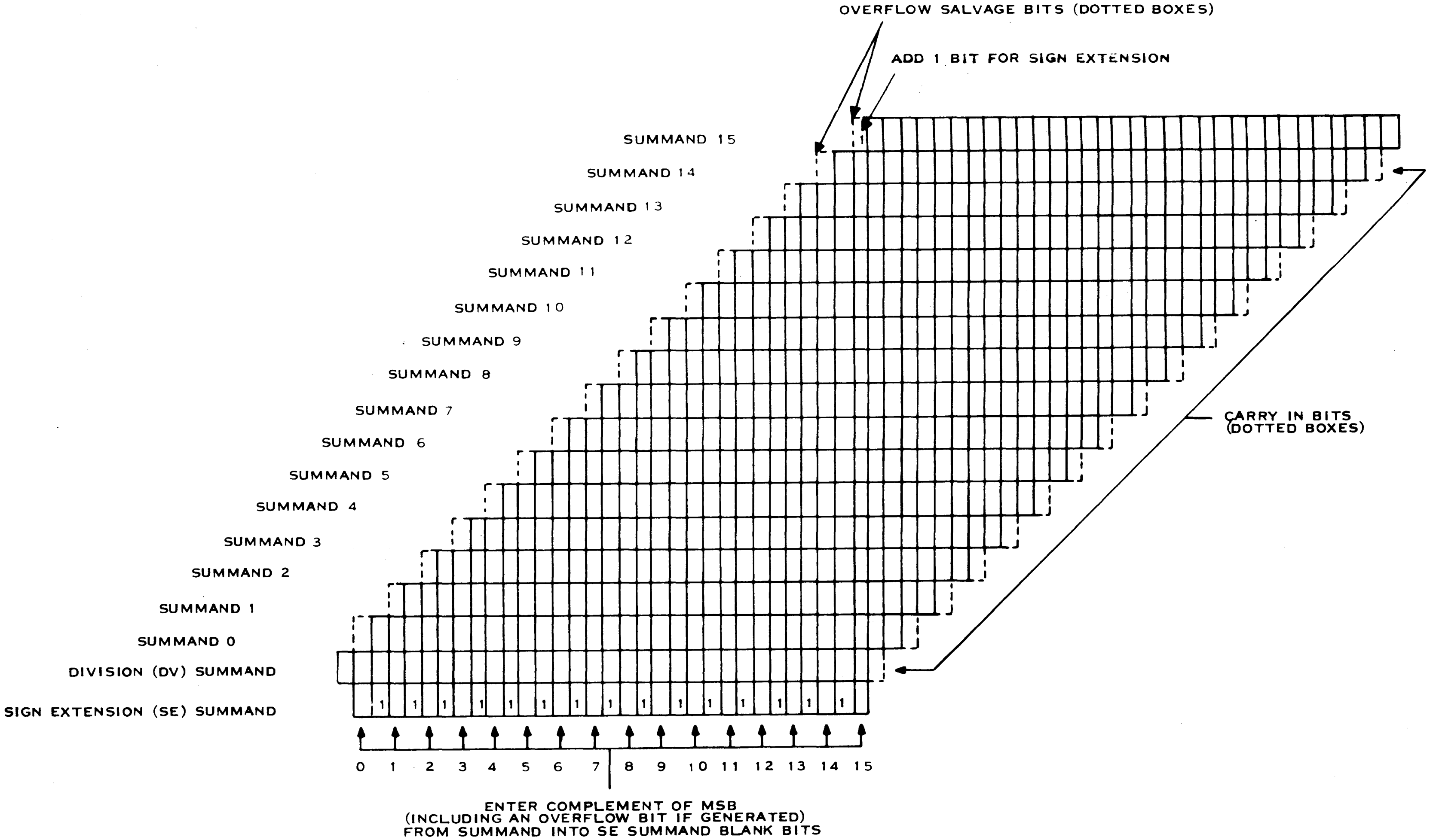
creating the first 16 summands for input to the Adder Tree. Each of these words is a copy of the Multiplicand. During division, a 17th fanout is produced to generate the division summand in the Form Summands circuit.

4-175 FORM SUMMANDS

This circuit receives the Multiplicand words from the fanout circuit and manipulates each word, as directed by the recode bits before placing the words into their respective adder tree inputs. Figure 4-17 illustrates the arrangement of the 18 possible summands in a figurative addition array. The summand circuits are gating devices and contain no registers for the individual summands.

4-176 OVERFLOW SALVAGE. During a left shift, a one bit from the data word may be shifted out of the summand. To avoid losing this bit, an extra bit position preceding each data word catches the shifted bit. This bit is not, however, added in the Adder Tree. It is significant only in its use to fill the Sign Extension summand, so that a shift will not change the effective sign of any summand.

4-177 SIGN EXTENSION SUMMAND. To avoid the requirement of extensive hardware to add sign extension bits of the summands, a single Sign Extension Summand represents the sum of all sign extension bits. Certain bits in the Summand are


OVERFLOW SALVAGE BITS (DOTTED BOXES)
ADD 1 BIT FOR SIGN EXTENSION
SUMMAND 15
SUMMAND 14
SUMMAND 13
SUMMAND 12
SUMMAND 11
SUMMAND 10
SUMMAND 9
SUMMAND 8
SUMMAND 7
SUMMAND 6
SUMMAND 5
SUMMAND 4
SUMMAND 3
SUMMAND 2
SUMMAND 1
SUMMAND 0
DIVISION (DV) SUMMAND
SIGN EXTENSION (SE) SUMMAND
CARRY IN BITS (DOTTED BOXES)
1
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
ENTER COMPLEMENT OF MSB (INCLUDING AN OVERFLOW BIT IF GENERATED) FROM SUMMAND INTO SE SUMMAND BLANK BITS

(A) 117986

Figure 4-17. Summand Array

fixed at a one value; the remaining bits are governed by the complement of the most significant bit of the modified multiplicand in each preceding Summand. The sign bit will be in the overflow salvage bit if a shift occurred. Otherwise, it will be the MSB of each summand. An additional one bit, added to the least significant bit of the SE Summand, completes the sign extension total. This bit occupies the overflow salvage bit of summand 15. Refer to paragraphs 4-185 through 4-187 for further explanation of the sign extension algorithm.

4-178 DIVISION SUMMAND. During a divide operation a 17th multiplicand enters from the fanout circuit and is acted upon by the Division Recode bits (R_{DV}) to form the Division Summand. R_{DV} can specify only a load multiplicand ($R_{DV}P1$) or a load complemented multiplicand ($R_{DV}M1$). Addition of the Division Summand to the summand addition produces a multiplication by an unsigned fraction rather than a signed whole number.

4-179 TWO'S COMPLEMENT. To perform a two's complement, the Summand circuit forms the one's complement (invert each bit) of the word and adds one to the least significant bit. The mechanics of the operation are slightly different for each of the two complementing operations. The R_NM1 recode bit (enter two's complement) enters the one's complement of the 31 most significant bits and the true value of bit 31 into the proper Summand. If bit 31 is a zero, a carry would have been generated by the addition of one to bit 31. The circuit then enters a "carry-in" one bit into the bit 32 position of the next Summand. This results in adding that bit to bit 30 of the first summand when the Adder Tree forms the Pseudo-Sum and Pseudo-Carry.

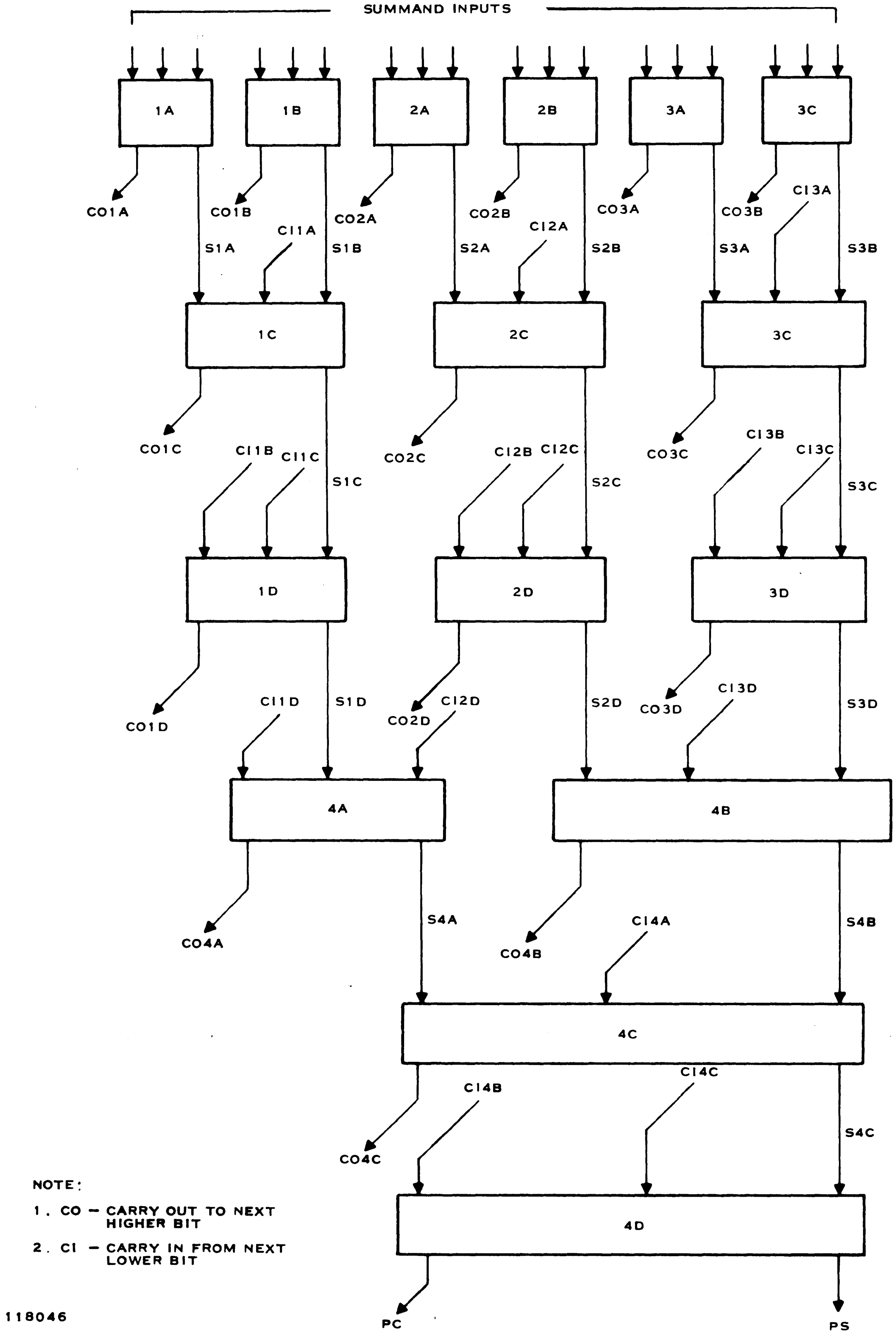
The R_NM2 recode bit (enter two's complement and left shift) actually loads the one's complement into the proper Summand and shifts it left one bit. To complete the two's complement, it loads a one bit into the "carry-in" bit of the next Summand so that when the Adder Tree adds that column, the carry-in bit will be added to the shifted LSB of the multiplicand.

4-180 ADDER TREE

The Multiplier Adder Tree is an 18-input adder circuit that receives the 32-bit Summand words from the Form Summands circuit and produces two 64-bit results: a pseudo-sum and a pseudo-carry. Figure 4-18 provides a simplified circuit diagram of the Adder Tree for one bit of the addition process. The inputs represent a 1-bit vertical slice from the Summand array. There will be a similar circuit for each vertical column in the array, except that most will not be as extensive as the sample circuit due to a fewer number of elements in the column. The output from this adder in the form of two 64-bit words is not in itself meaningful. These two data words must be totaled in the Accumulator section of the AU to form the final product or quotient.

4-181 PSEUDO-SUM (PS) REGISTER

PS is a 64-bit register that receives one of the resultant words from the Adder Tree during any operation that passes through the Multiplier section. The output from this register always goes to the Accumulator section of the AU to be added to the output from the PC Register.



NOTE:

1. CO — CARRY OUT TO NEXT HIGHER BIT
2. CI — CARRY IN FROM NEXT LOWER BIT

118046

Figure 4-18. Simplified Adder Tree Block Diagram

4-182 PSEUDO-CARRY (PC) REGISTER

PC is a 64-bit register that receives one of the resultant words from the Adder Tree during a multiply or divide operation. The output from this register always goes to the Accumulator section of the AU to be added to the output from the PS Register.

4-183 MULTIPLICATION THEORY

The AU Multiplier section multiplies two 32-bit operands to form two 64-bit partial results that are added to provide a final 64-bit product in the Accumulator section. Each 32-bit operand may be represented in the form:

$$2^n a_n + 2^{n-1} a_{n-1} + 2^{n-2} a_{n-2} + \cdots + 2^2 a_2 + 2a_1 + a_0$$

or in summation notation (for a 32-bit operand):

$$\sum_{n=0}^{31} 2^n a_n$$

Where a can be either a one or a zero bit coefficient, and n indicates the power of 2 position of that bit. The product of this operand and another 32-bit operand, B, is represented as:

$$\sum_{n=0}^{31} 2^n a_n B$$

One classical computer multiplication process performs this operation through a series of additions and register shifts. This process, for 32-bit operands, would require 32 add/shift cycles, or 32 partial Summands, depending on the desirability of execution time versus hardware bulk. The AU improves these extremes through the use of a recoding algorithm that reduces the number of Summands required for a high-speed multiply.

4-184 ALGORITHM DERIVATION

An equivalent binary expansion form of the multiplier is:

$$2^{2n+1} a_{2n+1} + 2^{2n} a_{2n} + 2^{2n-1} a_{2n-1} + \cdots + 2^2 a_2 + 2a_1 + a_0$$

This representation is equivalent to the first notation, but has the advantage of differentiating between odd and even terms. The summation can, therefore, be written defining half as many index values for n:

$$\sum_{n=0}^{15} 2^{2n+1} a_{2n+1} + 2^{2n} a_{2n}$$

By multiplying each odd term of the expansion by an equivalent of one, (2 - 1), a more useful expression evolves:

$$(2 - 1)\ 2^{2n+1}a_{2n+1} + 2^{2n}a_{2n} + (2 - 1)\ 2^{2n-1}a_{2n-1} + \cdots$$

$$+ 2^{2}a_{2} + (2 - 1)\ 2a_{1} + a_{0}$$

or, through simplification and replacement of each $2(2^{2n+1})$ with 2^{2n+2},

$$2^{2n+2}a_{2n+1} - 2^{2n+1}a_{2n+1} + 2^{2n}(a_{2n} + a_{2n-1}) - 2^{2n-1}a_{2n-1} + \cdots$$

$$+ 2^{2}(a_{2} + a_{1}) - 2a_{1} + a_{0}$$

which can be written in summation form as (for a 32-bit operand):

$$2^{32}a_{31} + \sum_{n=0}^{15} - 2^{2n+1}a_{2n+1} + 2^{2n}a_{2n} + 2^{2n}a_{2n-1}$$

The term $2^{32}a_{31}$ preceding the summation is actually a repetition of the sign bit, a_{31}, or a sign extension term that overflows the modulus of the multiplier. This term is, therefore, discarded to produce the expression

$$\sum_{n=0}^{15} - 2^{2n+1}a_{2n+1} + 2^{2n}a_{2n} + 2^{2n}a_{2n-1}$$

that is the equivalent of the binary expansion for the 32-bit operand. Multiplication of the multiplicand by this representation of the multiplier requires only 16 summands (terms) instead of 32 for the normal expansion. An inspection of the multiplication using this term demonstrates the algorithm used in the AU multiplier.

To multiply multiplicand B by the multiplier A in the new format, each bit position of B is multiplied by the term

$$-2^{2n+1}a_{2n+1} + 2^{2n}a_{2n} + 2^{2n}a_{2n-1}$$

to form the 16 Summands, one for each value of n,needed to form the product. If B is represented by the expression

$$\sum_{k=0}^{31} 2^{k}b_{k}$$

then each bit of the nth Summand will be determined as follows:

$$\text{Bit k of B} = 2^k b_k$$

$$\text{times } n^{th} \text{ term of A} = \underline{-2^{2n+1}a_{2n+1} \quad + 2^{2n}a_{2n} \quad + 2^{2n}a_{2n-1}}$$

$$2^{2n+k^{th}} \text{ bit} = -a_{2n+1}b_k 2^{2n+k+1} + a_{2n}b_k 2^{2n+k} + a_{2n-1}b_k 2^{2n+k}$$

or, $$-2a_{2n+1}b_k 2^{2n+k} + a_{2n}b_k 2^{2n+k} + a_{2n-1}b_k 2^{2n+k}$$

Which can be simplified to: $(-2a_{2n+1} + a_{2n} + a_{2n-1})\ b_k 2^{2n+k}$

This expression indicates that the Nth Summand can be determined by multiplying each b_k (k from 0-31) by the constant inside parentheses and placing the resulting bits in the 2^{2n+k} bit position. The constant is determined by inspecting the even term (a_{2n}) and its neighboring odd terms (a_{2n-1}, a_{2n+1}) of the multiplier with respect to the relation: constant = $-2a_{2n+1} + a_{2n} + a_{2n-1}$. The Recode circuit of the AU Multiplier performs this inspection to yield the results outlined in table 4-6.

Table 4-6. Recode Circuit Data Analysis

Incoming Bits a_{2n+1}, a_{2n}, a_{2n-1}	Resulting Constant (C)	Required Action to perform multiplication (C)*(b_k)
000, or 111	0	Fill Summand with zeroes.
110, or 101	-1	Perform two's complement of B and enter into Summand.
010, or 001	+1	Copy each b_k into corresponding bit positions of Summand.
011	+2	Left shift the B operand one bit and enter into Summand.
100	-2	Complement B, left shift it one bit, and enter result into Summand.

4-185 AU DIVISION THEORY

The Multiplier section performs division by multiplying the dividend by the reciprocal of the divisor. That is, the division Y ÷ X is performed as the equivalent multiplication:

$$1/X \cdot Y.$$

Both X and Y are the mantissa portion of normalized floating point numbers.. The Exponent Subtract section processes the exponent portion of the data. This division process, however, requires finding the reciprocal of X.

The AU approximates this reciprocal by providing a table of reciprocals (P-Term Logic) that is accurate to four binary places. This table, therefore, yields a reciprocal that is accurate within $\pm 0.00001_2$. For discussion, the Greek letter (Δ) represents the absolute value of this error. The reciprocal table provides a first approximation of the reciprocal of:

$$1/X \pm \Delta$$

to provide an approximation of the quotient (Q), through multiplication, of:

$$Y/X = Q \approx Y(1/X \pm \Delta)$$
$$Q \approx Q(1 \pm X\,\Delta).$$

Since X is normalized, it is of the form:

0.1---
0.01--
0.001-
or
0.0001

Therefore, the product ($X\,\Delta$) can be no greater than Δ. For this worst case, as the error approaches Δ, the value of X in the product $X\Delta$ can be disregarded to yield an approximation of Q:

$$Q \approx Q(1 \pm \Delta).$$

To reach a more acceptable value of Q, further refinement is necessary. By multiplying the approximate quotient by an approximation of one, this refinement is achieved. This operation produces:

$$Q \approx Q(1 \pm \Delta) \quad (1 \mp \Delta) = Q(1 - \Delta^2).$$

The error is reduced from $\pm\Delta$ (4-bit accuracy) to $-\Delta^2$ (8-bit accuracy). Extending this process develops the following approximations:

$$Q \approx Q(1 - \Delta^2)\ (1 + \Delta^2) = Q(1 - \Delta^4)$$

and

$$Q \approx Q(1 - \Delta^4)\ (1 + \Delta^4) = Q(1 - \Delta^8).$$

This last step reduces the error to Δ^8, (32-bit accuracy), which is not significant for single word (32 bit) operands. Double length operands require an additional step:

$$Q \approx Q(1 - \Delta^8)\ (1 + \Delta^8) = Q(1 - \Delta^{16})$$

to provide a value of Q that is accurate to 64 bits, or 2^{-65}.

Figure 4-19 illustrates the paths utilized by the AU to accomplish the division process. The vertical sequence on the left of the figure derives the multipliers for each step of the refinement process; the right column produces the approximations of Q. Notice that both the Accumulator and the Multiplier sections of the AU are in continuous use until a solution produced, each one being used alternately by one of the two derivation cycles.

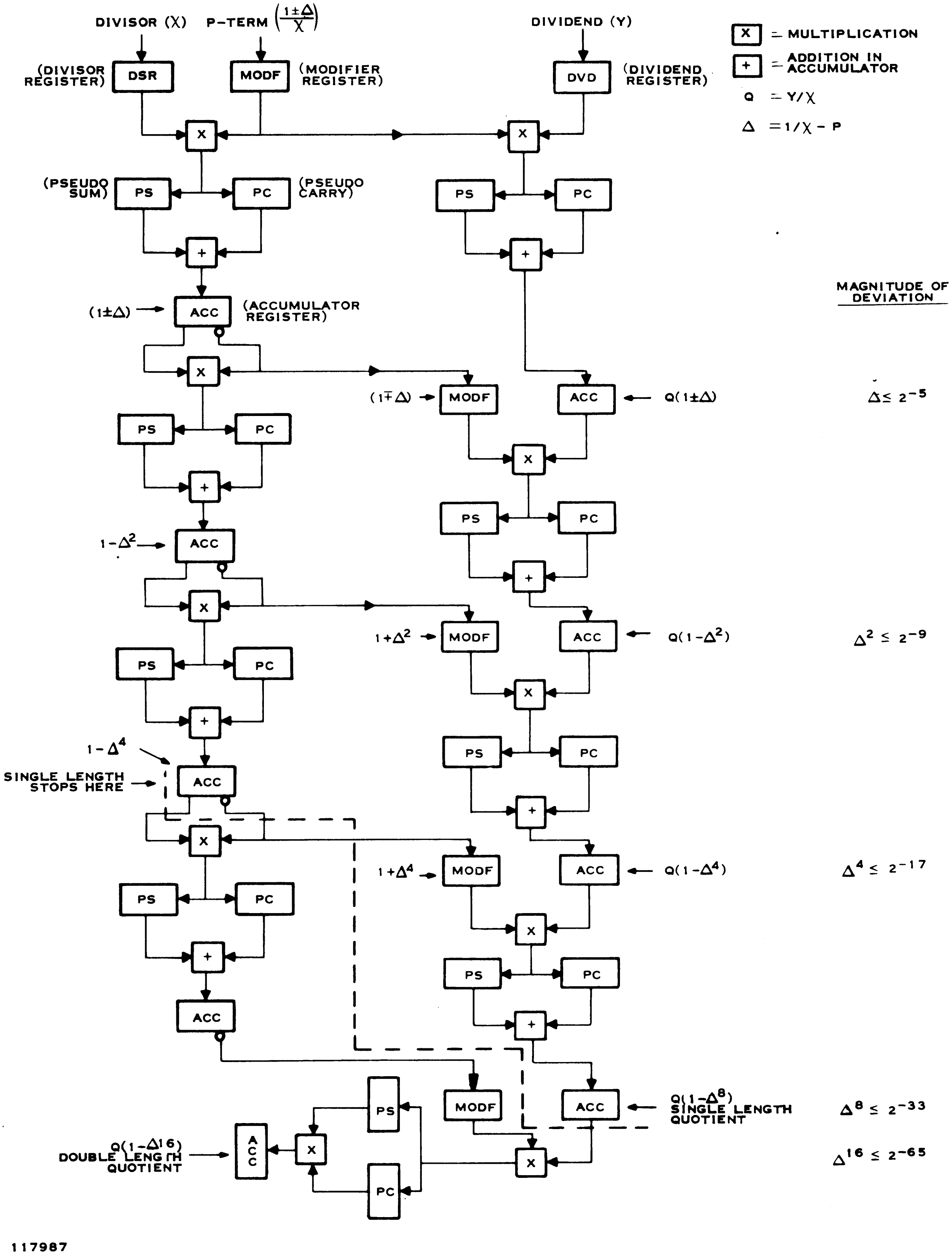


Figure 4-19. CP Hardware Utilization-Division Process

4-186 SIGN EXTENSION ALGORITHM

The sign extension algorithm allows the replacement of 272 Summands and their associated adder tree logic with a single 32-bit Summand that uses existing inputs to the adder tree. The algorithm determines the sum of all sign extension bits by inspecting the most significant bit (or overflow bit if the summand was shifted) of each summand. The resulting sum of the sign extension bits is determined by complementing the most significant bit, including overflow, of Summand 15 and placing the result in the least significant bit of the SE Summand. The complemented MSB's of the remaining summands enter every other bit of the SE Summand in sequence, while one bits fill the intervening bit positions of the SE Summand. Finally, an additional one bit is added to the LSB of the SE Summand to complete the process (this bit actually occupies the "overflow salvage" bit position of Summand 15).

4-187 TWO'S COMPLEMENT FORMATION

Before discussing the derivation of the sign extension algorithm, two methods of forming the two's complement of a binary number must be reviewed. The first method forms the two's complement of a number by starting with the LSB of the number and copying all bits of the number up to and including the first one bit. After that point, all bits of the number are inverted. That is, copy the LSB. If that bit is a one, complement the remaining bits of the number. If it is a zero, copy the next bit, inspecting it in a like manner.

The second method of two's complementing forms the two's complement by creating the one's complement of each bit of the number and then adding one to the LSB of the complemented number and performing any carry addition that may be required.

4-188 ALGORITHM DERIVATION

NOTE

> The following derivation considers only a three-entry sign extension. Expansion of this explanation to a 16-entry extension as employed in the AU is valid, but for simplicity of presentation is not covered in this discussion.

Consider the sign extension bits from Summands 0, 1 and 2. Using letters to represent these bits, they would appear in the Summand array as:

```
        CCCCCC----
        BBBB------
        AA--------
    ______________

  Total =  abcdef
```

By inspection of the addition and application of Boolean logical functions, the sum of the sign extension bits (abcdef) can be expressed in terms of the sign extension bits A, B, and C:

$f = C$
$e = C$
$d = B \oplus C$ *
$c = B + C$
$b = A \oplus (B + C)$
$a = A + B + C$

($\oplus$ is the symbol for exclusive OR).

This expression does not define a process for directly determining the sum. To further simplify this expression, the relation, $A \equiv (\bar{A})$, will be used. That is, a number is always equal to the complement of the complement of that number.

Forming the two's complement of abcdef, using the first method outlined above, produces a new number, UVWXYZ, whose individual bits are defined as follows:

$Z = f$ (copy first bit)

$Y = \bar{e}f + e\bar{f}$ (if f = 1, complement e, if not, then copy e)
$\quad = e \oplus f$

$X = \bar{d}(e + f) + d(\overline{e + f})$ (complement d if any preceding bit is a one; otherwise, copy d)
$\quad = d \oplus (e + f)$

$W = \bar{c}(e + f + d) + c(\overline{d + e + f})$ (similar to d)
$\quad = c \oplus (d + e + f)$

$V = b \oplus (c + d + e + f)$ (similar to d)

$U = a \oplus (b + c + d + e + f)$ (similar to d)

*d = 1 if B or C is a one, but not if both are ones. If B and C are ones, d = 0 and a carry is generated to the next bit, c.

Similarly,

c = 1 if B or C is a one, or if both are a one. If both are ones, however, the sum includes a carry from bit d, and in turn generates a carry to bit b.

Substituting the equivalent forms of abcdef into the definitions for UVWXYZ yields the following relationships:

$$Z = C$$

$$Y = C \oplus C = 0$$

$$X = (B \oplus C) \oplus (C + C) = (B \oplus C) \oplus C = B$$

$$W = (B + C) \oplus [(B \oplus C) + C] = (B + C) \oplus (B + C) = 0$$

$$\begin{aligned} V &= [A \oplus (B + C)] \oplus [(B + C) + (B + C)] \\ &= [A \oplus (B + C)] \oplus (B + C) = A \end{aligned}$$

$$\begin{aligned} U &= (A + B + C) \oplus \{[A \oplus (B + C)] + (B + C)\} \\ &= (A + B + C) \oplus (A + B + C) = 0 \end{aligned}$$

These expressions indicate that the two's complement of the sum of the sign extension bits may be represented as:

0A0B0C

Forming the two's complement of this representation (using the second method explained above) returns to an equivalent expression of the sum of the sign extension bits:

$$\begin{array}{r} 1\ \overline{A}\ 1\ \overline{B}\ 1\ \overline{C} \\ +\ 1 \end{array}$$

This expression defines the algorithm used to fill the Sign Extension Summand in the Summand array. Extending the SE Summand expression to 32-bits expands it to the right by adding bits of alternate ones and complemented sign extension bits. The additional one bit is always added to the least significant bit of the sum.

4-189 AU UNIT HARD CORE

Unit Hard Core performs context switches, power down sequences, and other maintenance functions, as instructed by the CP Master Hard Core. Refer to the maintenance command discussion for further information about AU hard core. Refer to Appendix C of this manual for map diagrams of the AU details words.

4-190 CONTROLLER DESCRIPTIONS AND FLOWCHARTS

The following pages contain information concerning the control circuits within the ASC Central Processor. Each control circuit is represented by a flowchart that outlines the decision paths within the controller. Text accompanying the flowcharts explains these paths in relation to the functions performed and signals generated by the controller.

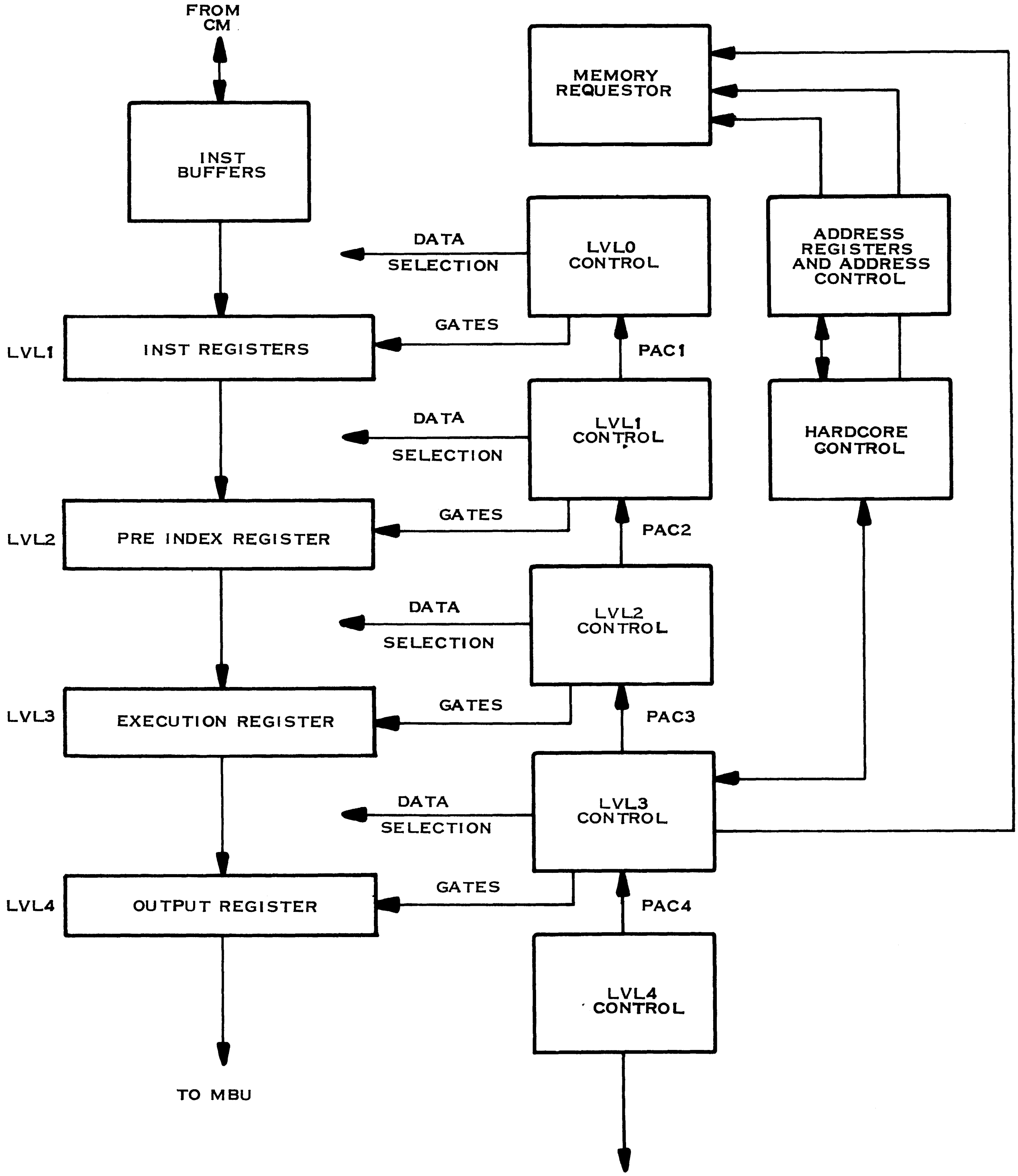
IPU control can be viewed as shown in figure 4-20. The control of each level of the pipeline through the IPU is treated independently in the flowcharts that accompany each control circuit description. Each level controller serves three primary functions: data selection, register gating, and generation of the "path ahead clear" (PAC) function for that level. The data selection and manipulation required between one level and the next does not affect the progress of an instruction from one level to the next. The control circuits determine when data is gated into the next level.

In general, the flowcharts for level 0 control through level 4 control indicate the following actions:

1. The gating of data into the succeeding level is usually indicated by the statement "LVLn ⟶ LVL_{n+1}", and is accompanied by setting the activity bit at level n+1, "1 ⟶ A_{n+1}". In general, this action occurs if the path ahead is clear into level n+1, level n is active, and hazards do not exist.
2. The progression of inactivity to a succeeding level is usually indicated by resetting the activity bit at level n+1 "0 ⟶ A_{n+1}". In general, this action occurs if the path ahead is clear into level n+1, but level n is inactive or a hazard exists.
3. The communication between a given controller and other controllers is indicated by output statements; for example, "PAC_n". This statement normally occurs if level n is inactive or if level n is active and is passing its data to level n+1.
4. Changing the primary control cells at a given level constitutes a state change at that level. Each state is enclosed with a dotted line on the flowcharts. State changes, then, are indicated by the progression through the flowchart from one enclosed area to another. These state changes normally occur when the subsequent actions at a given level are dependent on current conditions at that level.
5. State changes at a given level are sometimes accomplished by manipulation of secondary control cells. These cells are referred to as flags or counters. Changes in these flags are indicated by statements such as "1 ⟶ HOLD FLAG", or "DEC COMP CTR."

4-191 INSTRUCTION FLOW

The data residing at a given level of the IPU is usually an instruction in a partially decoded and developed condition. This data usually passes from one level to the next, never occupying more than one level at any given time. For such cases, the flowchart for each pipe level can easily be studied independently. There are, however, several situations for which the levels in the IPU do not operate independently. The following paragraphs supply an overview of the IPU objectives in order to more easily understand the flowcharts for these situations.



(A)124698

Figure 4-20. IPU Control

4-192 INDIRECT ADDRESSING. As an indirect instruction proceeds through the IPU, it reserves the level 1 through level 3. Thus, when an indirect instruction reaches level 3, levels 1 and 2 are inactive. Level 3 control makes a request for the indirect cell via the IPU memory bus and then becomes inactive. When the indirect cell is available, it enters level 1, proceeds through level 2, and finally replaces the address associated with the original instruction which still resides at level 3. When the indirect cell enters level 3, the activity bit at level 3 is set again, and the level 3 controller is again aware of the instruction. If the address is still indirect, then levels 1 and 2 were again reserved as the indirect cell passed·through. As the terminal indirect cell progresses through levels 1 and 2, these levels revert to their usual condition, and take no further part in indirect cycling. Indirect addresses never advance down the pipe beyond level 3.

4-193 EXECUTE INSTRUCTION. As an execute instruction (XEC) passes through the IPU to level 3 it reserves levels 1 and 2 of the pipe in a manner similar to an indirect instruction. Level 3 control makes a request for the object of the XEC, sets the XEC flag, and becomes inactive. When the object of the XEC instruction reaches level 3, all trace of the original instruction is gone except that the XEC flag is set and the address register at level 3 contains the address of the XEC. The object instruction is performed as if it had been in the program string in the position of the XEC. The XEC flag alters skips, branches, and calls such that the program string is not altered. At the conclusion of the instruction the XEC flag is reset.

4-194 SKIPS. Skips produce a SKIP signal from level 3. Each upstream level control observes pipe activity in the upper levels and the instruction to be skipped is inactivated if it is in the pipe. If it is not yet in the pipe, then this fact is recorded by the level 1 control. When the instruction does appear in level 1, it is discarded by the level 1 controller.

4-195 BRANCHES. A branch instruction produces commands from level 3 to the upstream levels. Each upstream level inactivates those instructions which are in the pipe but which are not desired because of the branch. If the instruction to which the branch is taken is not in the pipe upstream from level 3, then the branch address in level 3 is accepted by address control, and the pipe remains inactive through level 3 until the new instruction stream can be fetched and started down the pipe.

Indirect branches reserve levels 1 and 2 of the pipe as do all indirect instructions. However, indirect cycling does not begin until and unless the branch test is satisfied.

4-196 STORE FILE AND LOAD FILE INSTRUCTIONS. Store File and Load File instructions reserve level 2 as they pass down the pipe to level 3. This block at level 2 eliminates special and extensive hazard detection logic which would be required if an instruction were at level 2 during execution of file instructions at level 3.

Memory requests required for execution of the file instructions are initiated by level 3 control via the IPU memory bus. Normal instruction flow resumes through level 2 after the file instruction.

4-197 PUSH, PULL INSTRUCTIONS. Push and Pull instructions reserve level 2 as they pass down the pipe to level 3. The Push or Pull instruction occupies level 3 while the address of the stack parameters advances to the MBU. The MBU fetches/and transfers them to the AU for modification and testing. When available from the AU, the stack pointer is accepted into level 2, advances to level 3, and proceeds down the pipe, appearing to be a load (Pull) or a store (Push). As the pointer moves into level 3, normal instruction flow into level 2 resumes.

If the AU test indicates termination should result, level 3 control terminates the operation and resumes normal instruction flow. If termination is not necessary, then the address of the stack parameters advances to the MBU and the MBU stores the modified values.

4-198 LOAD LOOK AHEAD CONTROLLER

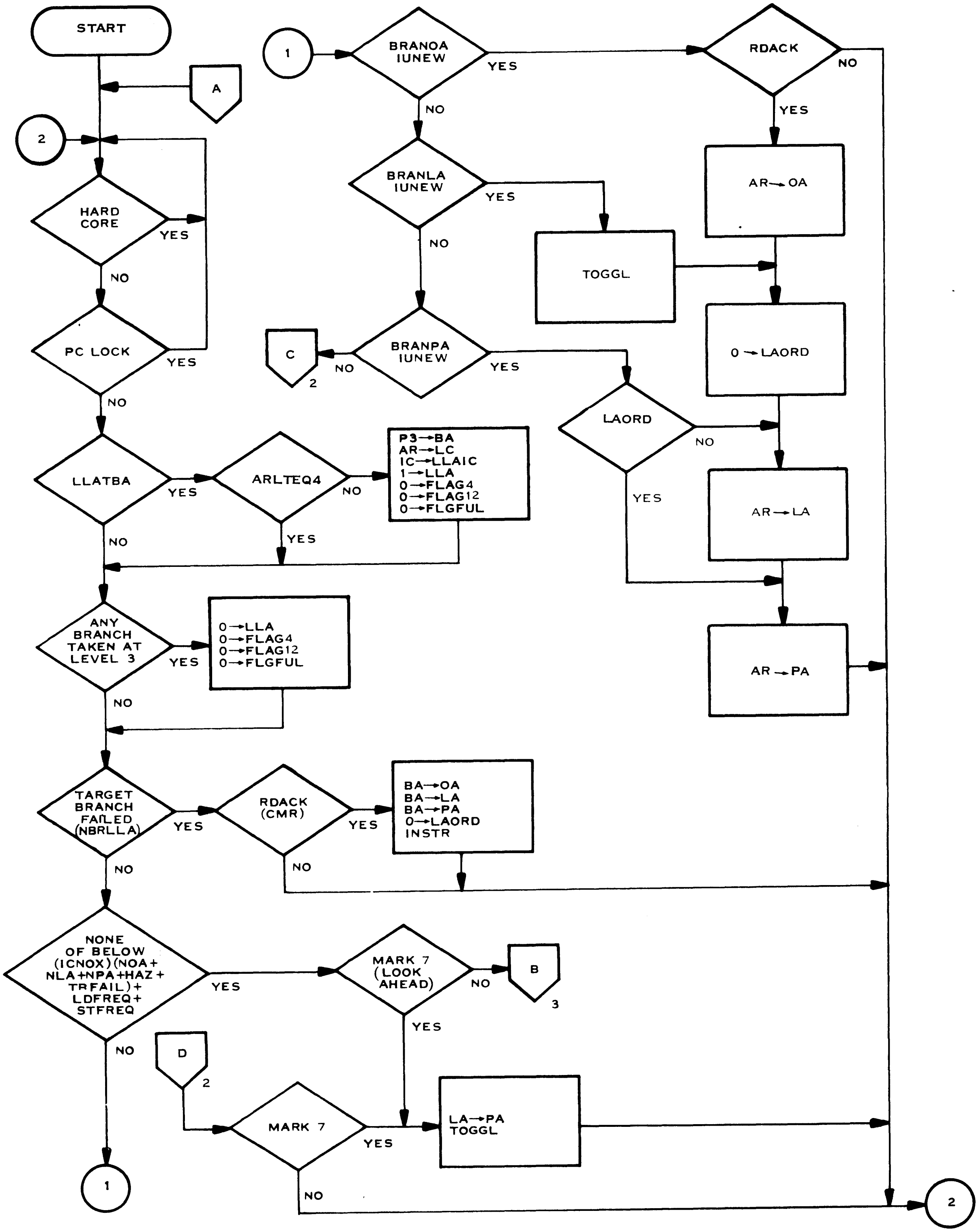
The Load Look Ahead controller produces gating and control signals required to load each address register in level 0 of the IPU, and the IR register in level 1. The controller monitors the status of instructions octets in the IPU, and by loading the address registers at the proper time, ensures that instructions will be available to IR with the minimum possible delay. During normal instruction sequencing, the controller loads the address of the look ahead octet (the next sequential octet after the current octet) into OA, so that the IPU may fetch that octet from memory and place it into the look ahead buffer (KA or KB). If a branch enters the pipe that has been preceded by an LLA instruction, the controller fills the pipe following the branch instruction with instructions from the branch path. When the branch occurs, the instructions in the branch path will be immediately available. A branch that is not preceded by an LLA creates a delay by requiring a new memory fetch. The following paragraphs describe the operation of the Load Look Ahead controller with reference to the flowchart in figure 4-21. The paragraphs follow the same order as the logic flow through the chart, and explain the major decision paths that are possible within the controller.

4-199 CONTROLLER TIMING

The LLA controller is composed of combinational logic, and as such, has no timing chain, sequence of events, or formal states. All of the question blocks illustrated in the flowchart are examined simultaneously during each control cycle to enable only one path through the controller. When the control clock pulse occurs, all of the action blocks on the indicated path are executed simultaneously. This type of timing means that actions upstream from other decision blocks in the flowchart do not affect the decision block. Also, since all actions occur simultaneously, all action statements refer to conditions at the start of the control cycle.

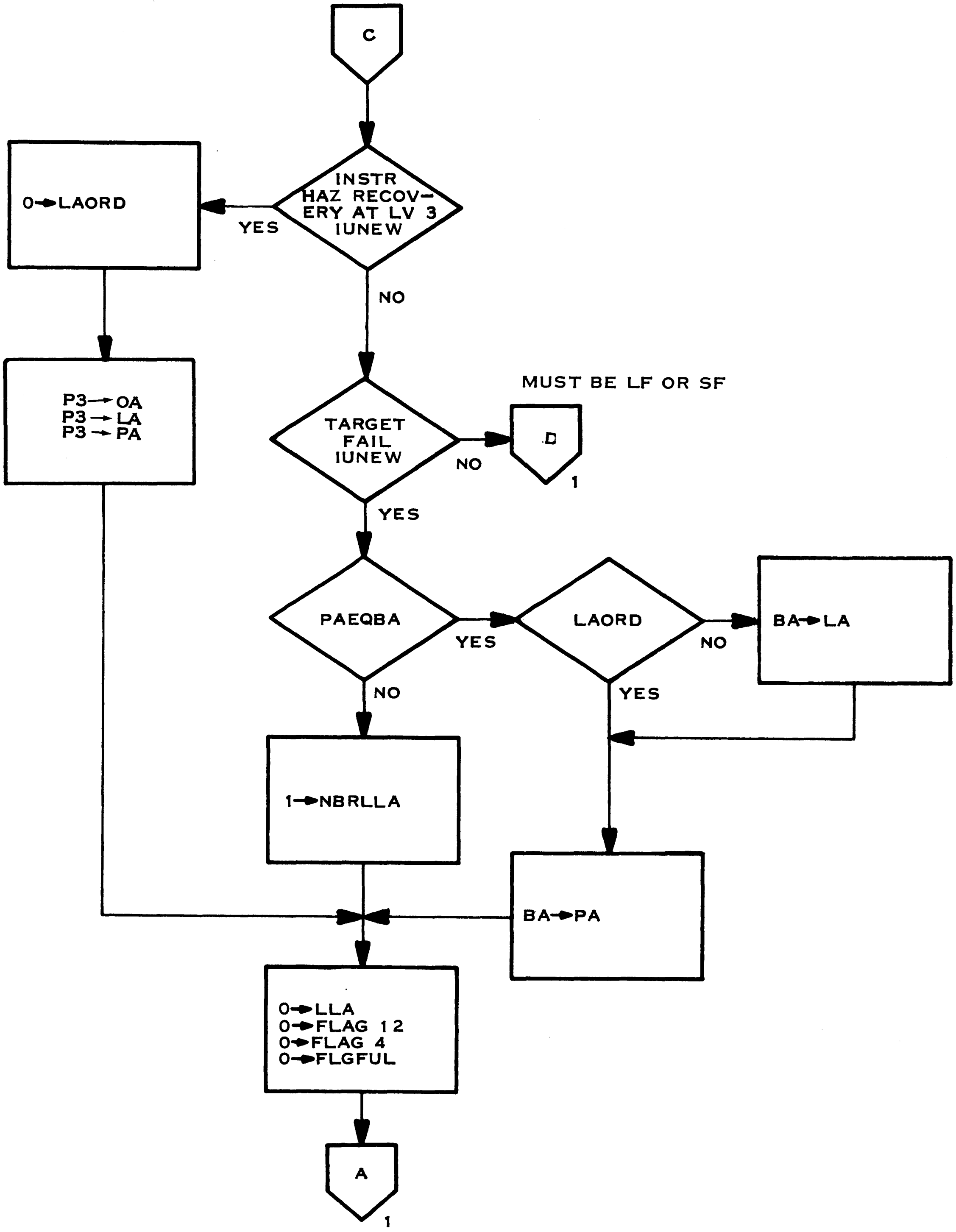
4-200 START

Before the LLA controller can check any of the status conditions, it ensures that the CP is not performing a maintenance command (hard core) and that the CP is not disabled (PC LOCK). If both of these conditions are satisfied, the controller is enabled to perform the remaining inspections.



114299A

Figure 4-21. Load Look-Ahead Controller Flowcharts (Sheet 1 of 5)



114300

Figure 4-21. Load Look-Ahead Controller Flowcharts (Sheet 2 of 5)

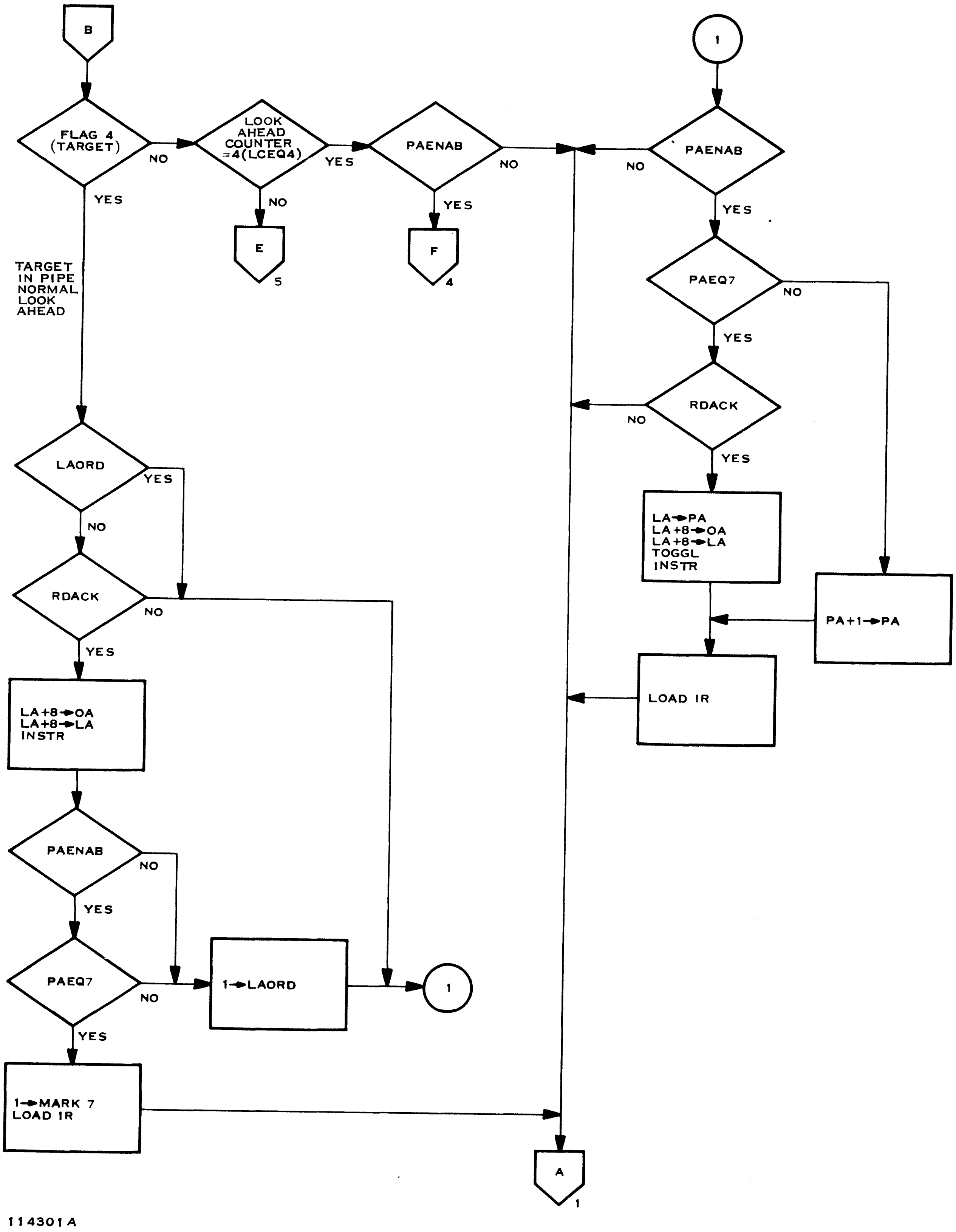
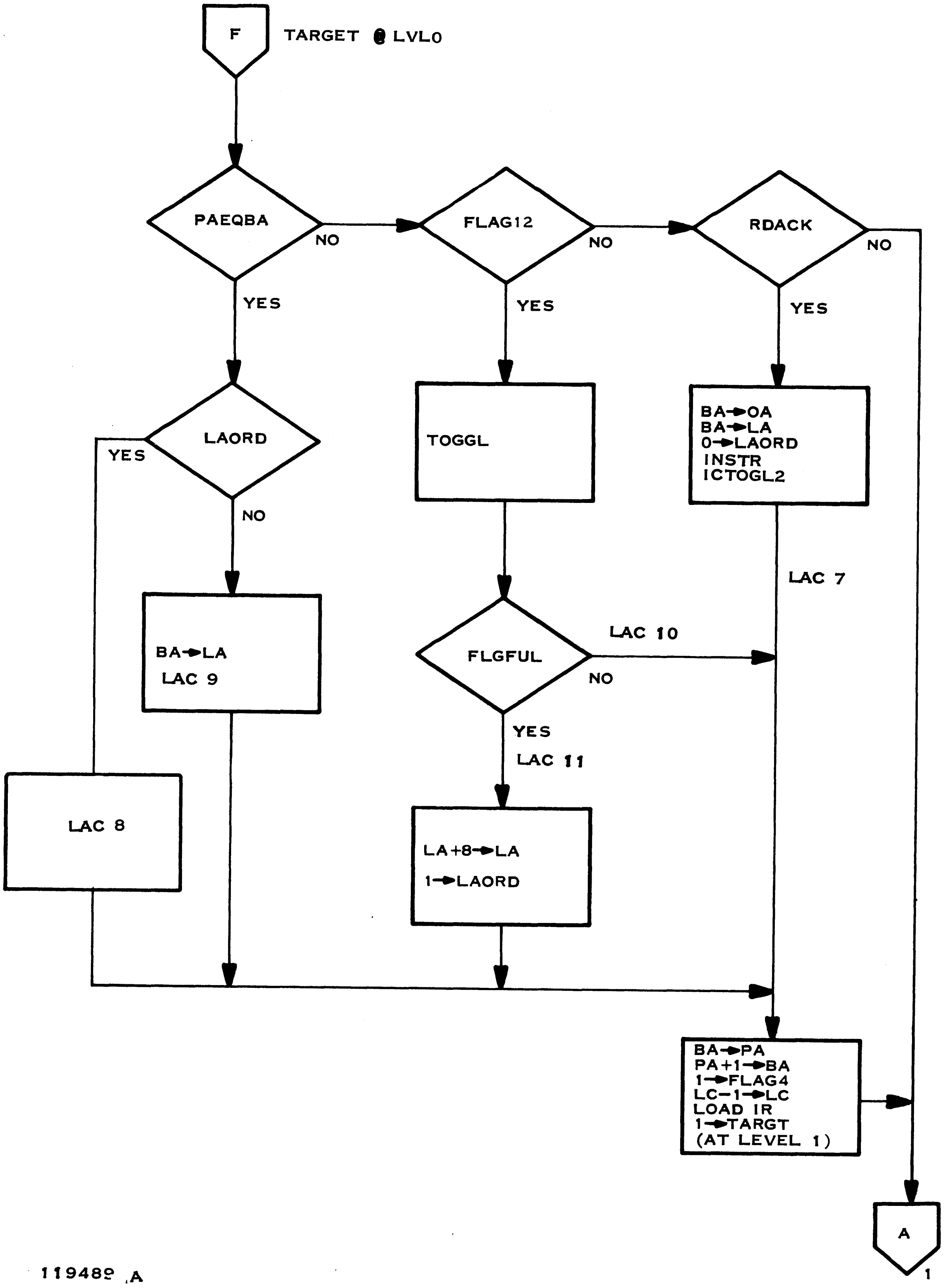


Figure 4-21. Load Look-Ahead Controller Flowcharts (Sheet 3 of 5)



119489 A

Figure 4-21. Load Look-Ahead Controller Flowcharts (Sheet 4 of 5)

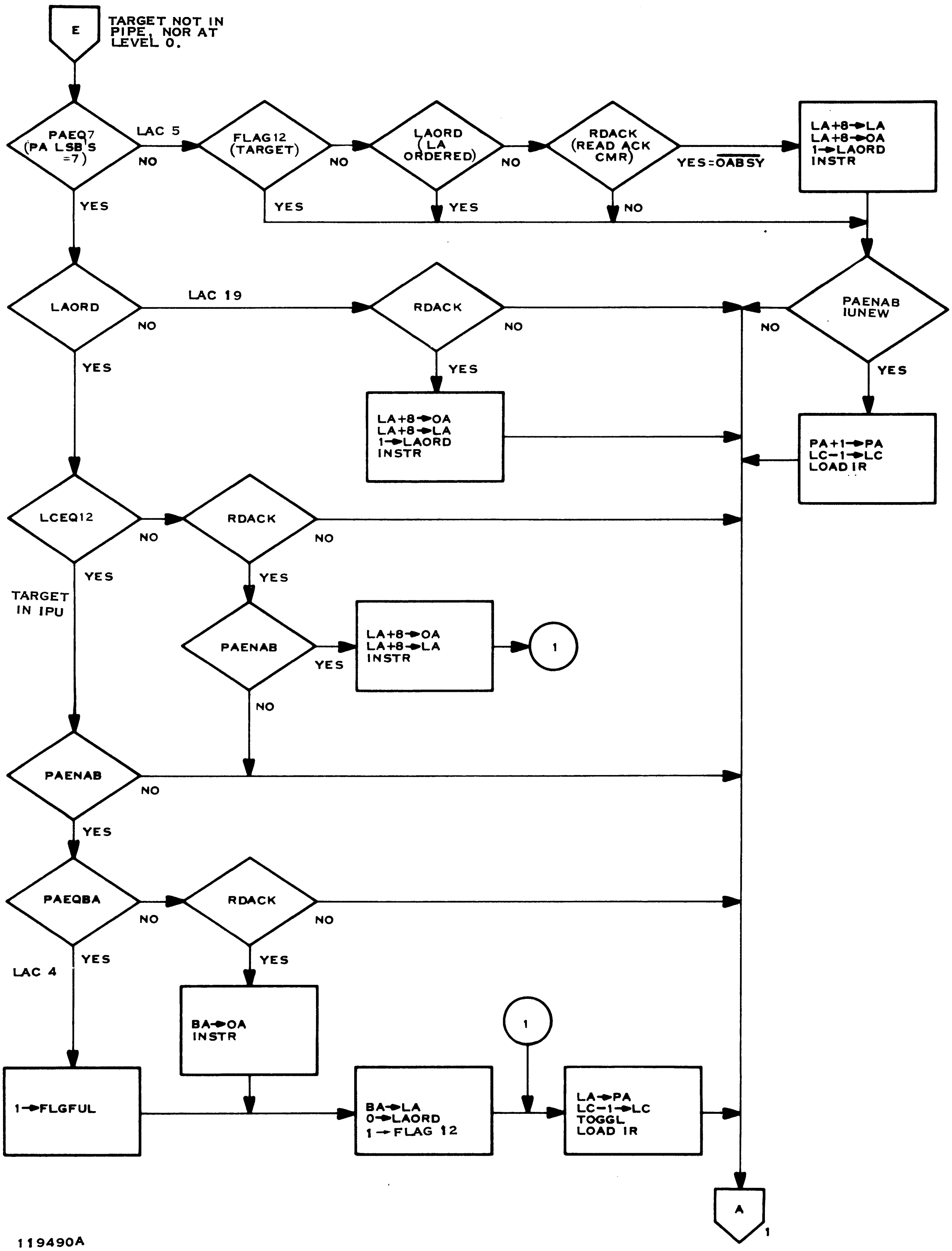


Figure 4-21. Load Look-Ahead Controller Flowcharts (Sheet 5 of 5)

4-201 LLA AT LEVEL 3

If an LLA instruction is at level 3 (LLATBA), the controller checks the count in the AR register. If the count is not less than or equal to 4, the indicated branch instruction is not in the pipe. Therefore, the controller transfers the LLA address from P3 to BA and the number of instructions between the LLA and the branch from AR to LC. It then sets the LLA in progress flag, clears the branch progress flags and loads the number of active levels into the the active level count register (LLAIC). The control path returns to the main path whether or not the counter is equal to 4.

4-202 BRANCH TAKEN AT LEVEL 3

If a branch has been taken at level 3, then any flags relative to branch position are no longer valid. The controller clears these flags so that they will not create false indications during the control cycle.

4-203 TARGET BRANCH FAILED

If a targeted branch is skipped over or fails to branch when it reaches level 3 (due to conditional requirements of the branch), the controller must recover the original instruction stream since it has prepared the IPU for the branch path. To recover, the controller ensures that the Central Memory Requester (CMR) is ready to perform a memory fetch (RDACK). If this condition is met, the controller transfers the recovery address from BA to OA, LA and PA, clears the LA ordered flag (LAORD) and initiates a memory request cycle (INSTR). The recovery address is one greater than the address of the branch instruction. This address was transferred into BA when the branch instruction entered level 1 of the IPU.

4-204 TARGET IN PIPE

If no condition is present that will cause the contents of the pipe to be negated (branch to OA, LA or PA, instruction hazard, target branch fail, store file or load file) and the last instruction used was not the last instruction of an octet (not MARK 7), the controller checks the condition of Flag 4 to determine if a targeted branch has entered the pipe (sheet 3). If flag 4 is set, the look ahead octet has not been requested from memory (LAORD), and CMR is ready to issue a read request to memory, the controller adds eight to the address in LA and transfers that new address to OA and LA. The controller then initiates a memory request cycle (INSTR) for the new octet and checks whether IR can accept a new instruction (PAENAB). If IR is not ready, the controller sets the LA Ordered flag to indicate that the look ahead octet has been requested and returns to the beginning of the control cycle for the next clock pulse. If IR is ready and the next instruction from the current octet will be the last instruction in that octet, (PAEQ7), the controller sets MARK 7 to indicate that the current buffer has been used, and transfers the last instruction from the current buffer into IR before returning to the start of the control cycle.

If IR is ready but the next instruction will not be the last in the current octet, the controller sets the LA Ordered flag, increments the address in PA and loads a new instruction into IR. If the LA Ordered flag was set when the controller entered this segment of the control cycle, the controller checks to see if the IR register can accept a new instruction. If IR is ready, the controller increments the address in PA and loads IR if the last word transferred into IR was not the last word of an octet (not PAEQ7). If the last instruction is being taken from the octet, the controller adds eight to the address in LA and transfers that new address to the OA and LA registers. It then toggles the KA/KB output pointer to select the next octet, initiates a new memory request for the look ahead octet (INSTR), and moves the address previously in LA to PA in preparation for the next instruction transfer to IR.

4-205 TARGET ENTERING PIPE

If Flag 4 is not set (the targeted branch is not in the pipe), the controller checks the look ahead counter together with the number of active levels at the start of the LLA to determine if the target is about to enter level 1 of the pipe (sheet 3). If the target branch is entering level 1 with the next control clock, and the IR register is ready to accept a new instruction (PAENAB), the controller checks the status of LLA addresses and flags (sheet 4). If the address of the current octet contained in PA is equal to the address of the branch path contained in BA, then the controller does not need to order a new octet to prepare for the branch. The controller determines if the look ahead octet has been ordered (LAORD) from memory. If LAORD is not set, the controller transfers the address in BA to LA to ensure that the next octet ordered from memory will be the correct octet to continue the branch path.

If PA is not equal to BA, the controller checks the state of Flag 12 to determine if the octet for the branch path is contained in the IPU look ahead buffer, or has been ordered from memory and will be loaded into the look ahead buffer. If Flag 12 is set, the controller toggles the output pointer that selects the output from KA or KB so that the look ahead buffer will supply instructions to IR for subsequent control cycles. In addition, if Flgful is set, indicating that the branch path is in the previously used octet, the controller loads the address of the next octet into LA (LA + 8) and sets the LA ordered flag to indicate that the look ahead octet is resident in the IPU.

If PA is not equal to BA and Flag 12 is not set, the controller must fetch a new octet from memory to provide instructions from the branch path. If CMR is ready to perform a read request to memory (RDACK), the controller transfers the address of the branch path from BA into OA and LA, clears the LA Ordered flag to indicate that a new memory request is required to access the look ahead octet for the branch path, initiates a request to memory for the look ahead octet (INSTR), and sets ICTOGL2 to select the new buffer to supply instructions to IR

Regardless of which of the above paths the controller follows through the Target at Level 0 branch of the logic (if RDACK = 1), the controller transfers the address in BA into PA to load instructions from the branch path into the pipe following the branch instruction, and transfers an address that is one greater than the current address (PA + 1) into BA so that the controller may recover the current instruction path if the branch is not taken. The controller then sets flag 4 to indicate that the branch is in the pipe, decrements the look ahead counter, transfers the branch instruction from the current buffer into IR, and sets the target at level one flag (TARGT) to indicate the position of the branch instruction.

4-206 NORMAL LOOK AHEAD CYCLES

If a target is not in the pipe or in level 0, and the current octet is not exhausted (PAEQ7 is false), the controller determines that a target is not in the IPU (Flag 12), the look ahead octet has not been requested from memory (LAORD) and that CMR is ready to perform a read request. If all of these conditions are met, the controller adds eight to the address in LA and loads that value into LA and OA, sets the LA Ordered flag, and initiates a memory request cycle. Regardless of the result of the above inspection, the controller increments the address in PA to select the next instruction from the current octet, decrements the count in the look ahead counter (if there is a count), and transfers the currently addressed instruction from the current buffer to the IR register if IR can accept a new instruction.

When the current instruction will be the last instruction in the current buffer (PA = 7), the controller ensures that the look ahead octet has been requested from memory. If LAORD is not set, the controller cannot supply further instructions to IR until the look ahead octet has been requested. The controller therefore, loads LA + 8 into LA and OA, initiates a memory fetch cycle, and sets the LA Ordered flag so that during the next control cycle the controller will be able to load a new instruction into IR.

If the LA Ordered flag has been set, the controller determines if a targeted branch is in the look ahead octet (LCEQ12). If no target is in the look ahead octet, CMR is ready for a read request, and IR can accept a new instruction, the controller adds eight to the address in LA, enters that address into OA and LA and initiates a memory fetch cycle for that octet. The controller also transfers the previous address in LA into PA to designate the first instruction from the next octet, toggles the output pointer to select the buffer containing that octet, decrements any count in the look ahead counter, and transfers the last instruction in the old buffer into IR.

4-207 TARGET IN LOOK AHEAD BUFFER

If the targeted branch instruction is not in the pipe or in level 0, the next instruction drawn from the current instruction octet will be the last instruction in that octet (PAEQ7), and the look ahead octet has been ordered from memory, the controller determines if the targeted branch is in that look ahead octet. To locate the branch, the controller examines the look ahead counter together with the number of active levels at the start of the LLA operation. If the LC count plus the number of active levels is less than or equal to 12, then the branch is in the look ahead buffer. If this examination determines that the target is in the look ahead buffer (LCEQ12), and IR can accept a new instruction (PAENAB), the controller compares the address in the BA register (starting address of the branch path) with the current address in PA. If the two octet address portions compare, the branch path initiates in the current octet. The controller therefore, sets the FLGFUL flag to prevent the octet from being destroyed by new memory fetches. If PA is not equal to BA, the controller transfers the address in BA into OA and initiates a memory request for that octet so that the controller can supply instructions from the branch path to follow the branch instruction into the pipe. For either condition of the PA - BA comparison, the controller transfers the address in BA into LA for generation of the look ahead octet for the branch path, clears the LA Ordered flag so that the look ahead octet will be requested after flag and sets, and sets FLAG12 to indicate that the target branch instruction is in the IPU. The controller also transfers the address in LA into PA to select instructions from the next octet, toggles the output pointer to select the buffer containing the next octet, decrements the count in the look ahead counter, and transfers the last instruction from the current octet into IR.

4-208 CONTENTS OF PIPE NOT USEFUL

Seven conditions, each of which causes the contents of the IPU pipe to become useless for further processing and requires that the IPU ignore these instructions are: Branch to PA, LA or OA, instruction hazard at level 3, target fail, store file or load file. The following paragraphs explain the controllers reaction to each of these conditions.

4-209 BRANCH TO OA. If a branch instruction reaches level 3 and the octet containing the branch path is not within the IPU, the controller must fetch that octet from memory before it can continue processing instructions. Since the instruction currently in the pipe are from the old instruction path, those instructions are disabled. If CMR can accept a read request, the controller clears the LA ordered flag to indicate that previous preparations for a look ahead octet are no longer valid and transfers the branch address from AR into OA to be sent to memory to fetch the branch octet, into LA to generate the next look ahead address, and into PA so that instructions from the branch path will be loaded into the pipe when the new octet returns from memory.

4-210 BRANCH TO LA. If a branch at level 3 references an address equal to the LA address, the branch path will be in the look ahead buffer, KA or KB. Since the level 3 controller checks the resident address registers in reverse order, i.e., P2, P1, PA and then LA, LA must be different from PA in order to produce a branch to LA. Therefore, LA has been ordered from memory. To access the word from LA, the target address in AR is transferred to PA and the KA/KB output selection pointer is toggled to choose the unused buffer. To ensure that LA contains the correct address and has not been altered since the branch determination, the address in AR is transferred to LA to be incremented for the next look ahead octet. The LA Ordered flag clears to ensure that the look ahead octet will be requested during the next control cycle.

4-211 BRANCH TO PA. If a branch at level 3 references an address equal to the the PA address, the branch path is in the current buffer. To access the first instruction in that path from the current buffer, the branch address in AR transfers to PA. In addition if the LA Ordered flag is not set, the controller transfers the address in AR to LA to ensure that the next octet fetched from memory will be the next sequential octet following the branch octet.

4-212 INSTRUCTION HAZARD RECOVERY. If an instruction hazard has occurred at level 3, the controller must re-access the octet from which the current instruction was drawn to obtain the new information. The address of the hazarded instruction is in the P3 register. To prepare for a memory fetch to access the octet, the address in P3 transfers into OA. In addition, the controller loads the address into PA to select the instruction from the octet when it returns from memory, and into LA to form the next look ahead address. At this point the address in PA is the same as the address in LA, indicating that the next octet has not yet been requested. Therefore, the LA Ordered flag clears. The controller also clears all of the branch status flags (FLAG4, FLAG12 and FLGFUL) since the change in instruction sequence negates the effect of these flags. The next control cycle initiates the request to memory for the octet containing the hazarded instruction.

4-213 TARGET FAIL. If a branch instruction failed to branch when it reached level 3, or was skipped over in the program sequence, a target fail condition exists. The look ahead controller must then revert to the program sequence that was abandoned when the branch instruction entered the pipe. At that time the address of the instruction following the branch instruction was stored into the BA register (PA + 1 to BA), and the controller began loading instructions from the branch path. Since the branch will not be taken, the controller must use the address in BA to fetch an instruction octet to continue the program. If the address in BA is in the PA octet, an immediate recovery is possible without the delay of a memory fetch. The contents of BA are transferred to PA to select the proper word from the buffer to be loaded into the pipe. If the LA octet has not been ordered from memory, BA also transfers to LA to ensure that the proper octet will be ordered for the next look ahead octet. If the BA address is not equal to PA, a memory request cycle is required to recover from the target fail. The controller sets the NBRLLA flag on this clock, so that the following clock will initiate a memory fetch for the correct octet. For either course of action, the controller clears all branch status flags since the target failure negates the progress of the LLA instruction.

4-214 LOAD OR STORE FILE. If the contents of the pipe have been negated by an operation that is not a branch to OA, LA or PA, is not an instruction hazard and is not a target fail, then the condition is a load or store file instruction. Either of these instructions require no further action of the look ahead controller. The controller ensures that the current octet has not been exhausted (MARK 7). If MARK 7 is set, the controller transfers the address in LA into PA and toggles the KA/KB output pointer to select the other buffer to supply instructions to IR.

4-215 LEVEL 0 CONTROLLER

The level 0 controller selects a word from one of the instruction buffer files, and transfers that word into the IR register at level 1 if level 1 is not active. If level 1 is active, the level 0 controller examines the state of the level 2 controller and either waits or clears the activity bit for level 1 depending upon the results of that examination. Included in the controller logic are allowances for branch, target fail and hazard conditions that abort the instruction sequence currently in the pipe. Since the controller clears the level 1 activity bit during one clock there is a minimum delay (or "bubble") of one clock time between instruction transfers to IR. The following paragraphs describe the major control paths depicted in the level 0 controller flowchart (see figure 4-22).

4-216 INDIRECT OR EXECUTE AT LEVEL 3

If the control circuit is enabled, the controller determines if there is an indirect or execute instruction currently at level 3. If either of these conditions exists, the controller must select the word designated by the alpha address at level 3 to be passed through the pipe to level 3. The controller therefore checks the level 1 activity bit to determine if a new instruction can be transferred into level 1. If the activity bit is set and level 1 is not busy due to an M or T hazard, the controller clears the level 1 activity bit since level 2 should have been reserved by the indirect or execute at level 3. That is, level 1 will move into level 2 without delay.

If the level 1 activity bit is not set, the controller checks the alpha address at level 3 to determine if the address is in the register file (alpha less than or equal to 2F). If the address is in the register file, the instruction at level 3 is not a branch (a branch cannot reference the register file), and the register inhibit flag is not set, the controller uses the alpha address to select a word from the register file as the object of the indirect or execute at level 3. If the reference is outside the register file, the controller determines if the object of the indirect or execute is in either of the two buffer files. If the alpha address is in the current buffer (alpha = PA), the controller enables the buffer indicated by KRTAG (KRTAG = KB; -KRTAG = KA) and uses the alpha address to select the object word from the buffer. If the alpha address is in the look ahead buffer (alpha = LA) the controller enables the buffer that is not indicated by KRTAG and uses the alpha address to select the object word from the buffer. If any of these local branch paths are taken, the controller completes the cycle by setting the level 1 activity bit, thereby transferring the selected word into IR. If the object of the indirect or execute is not within the IPU, the controller uses the alpha address to select a word from KCM, waits for Central Memory Requester (CMR) to place the required octet into KCM, and then sets the level 1 activity bit to transfer the selected word into IR.

4-217 FILE SELECT

If an indirect or execute is not currently at level 3, the controller inspects the state of the buffer output pointer, KRTAG. If this bit is clear (zero), the controller selects the KA buffer file to supply instructions to level 1; if this bit is set, the controller selects KB to supply instructions to level 1.

4-218 LEVEL 1 NOT ACTIVE

If level 1 is not active, then the controller can provide a new instruction to the IR register. The controller first inspects the state of the level 1 controller. If the level 1 controller is performing in the indirect at level 2, hazard, store file or the load file state, the level 0 controller returns to the start of the control sequence, since it cannot transfer a new instruction to IR if the level 1 controller is in any of these states. If after inspecting the state of the level 1 controller, the level 0 controller determines

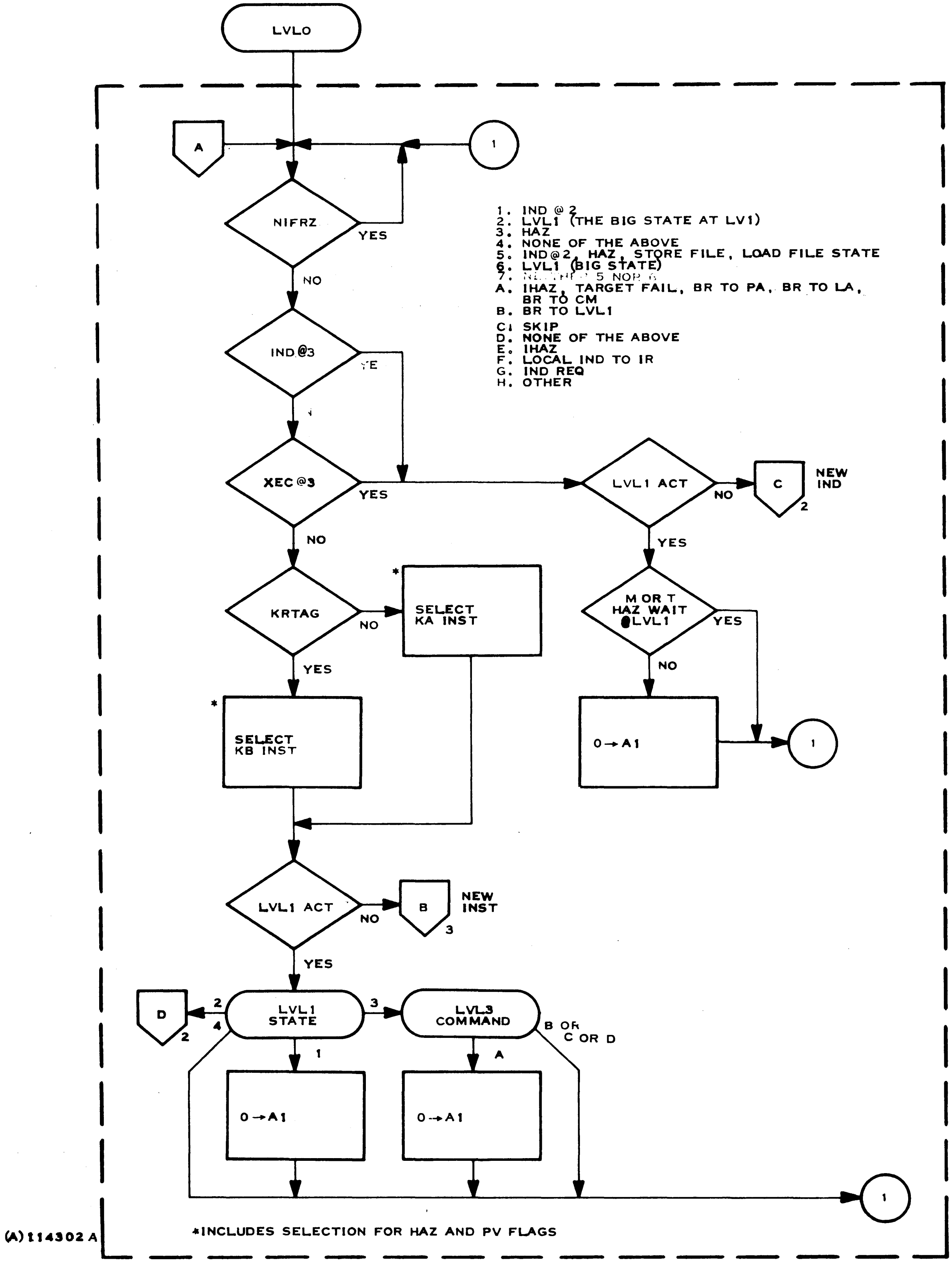


Figure 4-22. Level 0 Controller Flowchart (Sheet 1 of 3)

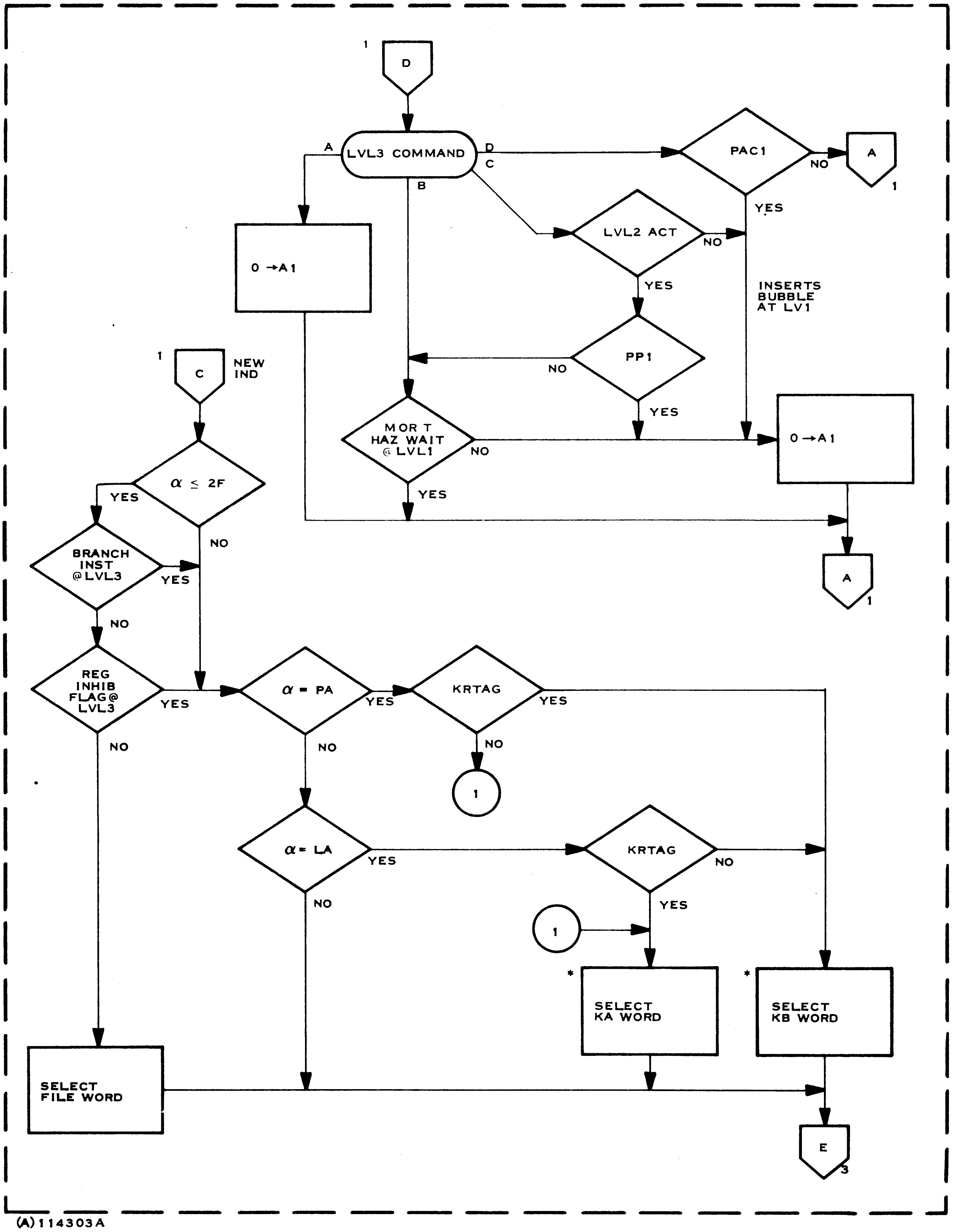
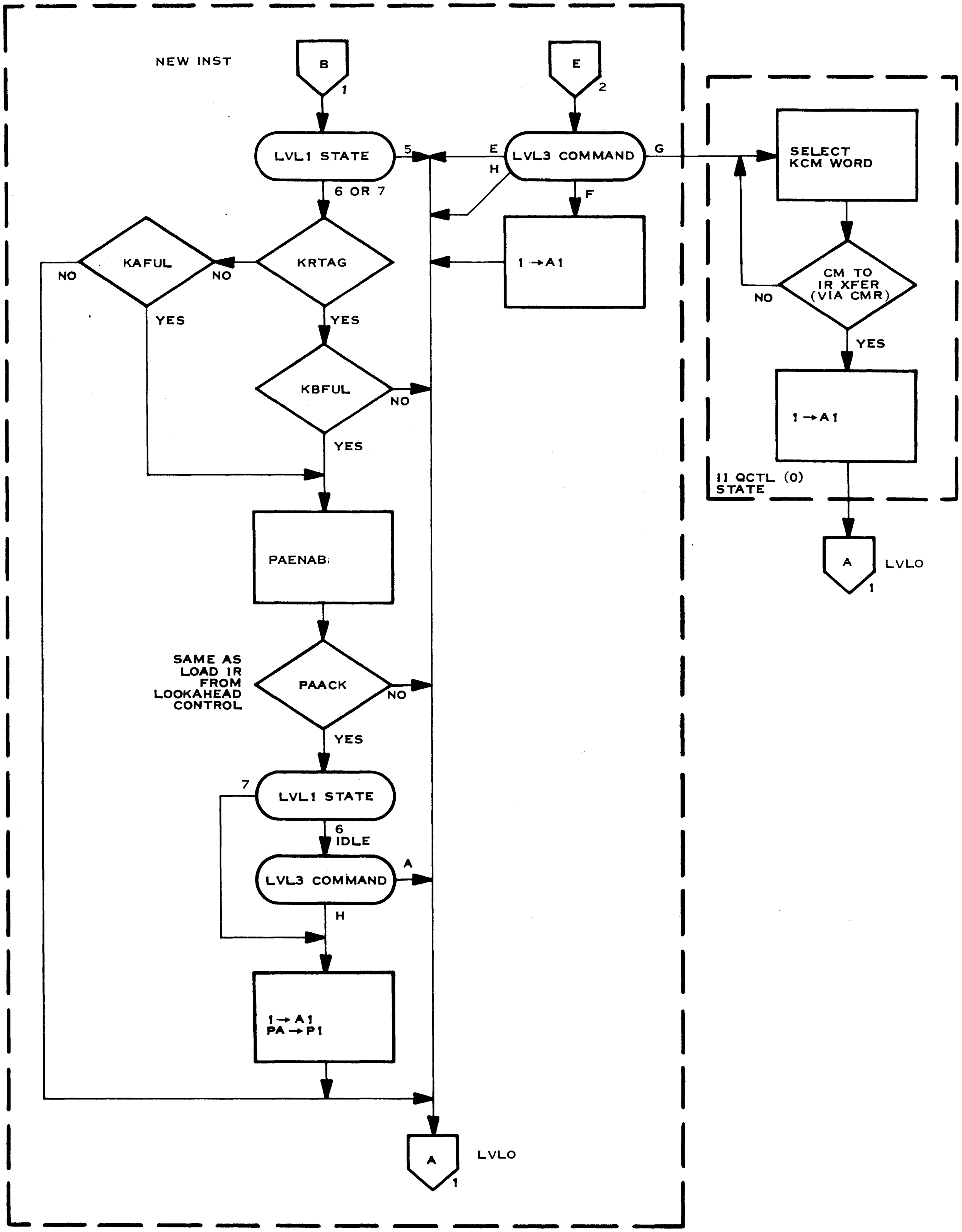


Figure 4-22. Level 0 Controller Flowchart (Sheet 2 of 3)



119488 A

Figure 4-22. Level 0 Controller Flowchart (Sheet 3 of 3)

that it can transfer a new instruction into IR, the controller ensures that the chosen buffer file contains a valid octet to provide instructions to IR. If the respective buffer full signal is active, the controller issues PAENAB to the look ahead controller indicating that the IR register can accept a new instruction and the selected buffer can supply a valid instruction word. The look ahead controller then generates a Load IR signal that appears as PAACK to the level 0 controller. When the level 0 controller detects this signal, and if the level 1 controller is in the Big State, the level 0 controller examines the command at level 3 to determine if that command will negate the instruction that was just transferred into IR. If that is the case, the controller returns to the beginning of the control cycle without setting the level 1 activity bit. If the instruction transferred into IR will not be negated, or if the level 1 controller was not in the Big State, level 0 controller sets the level 1 activity bit and transfers the address in PA into the P1 register. This sequence loads a new valid instruction into IR and its corresponding word address into P1.

4-219 LEVEL 1 ACTIVE

If level 1 is active, the controller cannot transfer a new instruction into IR. The controller then inspects the state of the level 1 controller to determine if the level 1 activity bit can be cleared during the next control clock pulse. If the level 1 controller is in the Indirect at level 2 state, or if it is in the hazard state and the instruction at level 3 negates the instruction in level 1, the level o controller clears the level 1 activity bit. Clearing the activity bit occurs on the control clock pulse as the instruction in level 1 passes to level 2. If the level 1 controller is in the hazard state, but the level 3 command does not negate the pipe contents, or if the level 1 controller is otherwise not in the Big State, the level 0 controller takes no further action and returns to the start of the control cycle. If the level 1 controller is in the Big State, the level 0 controller inspects the level 3 command to determine its course of action.

If the instruction at level 3 negates the contents of the pipe, the level 0 controller clears the level 1 activity bit on the next control clock. If the instruction at level 3 is a branch to level 1 and there is no M or T hazard wait at level 1, the controller clears the level 1 activity bit on the next control clock. An M or T hazard wait at level 1 returns the level 0 controller to the start of the cycle to await the next control cycle. If the Skip bit is active for the level 3 command, the controller checks to see if level 2 is active. If there is no instruction at level 2, the controller clears the level 1 activity bit. Similarly, if level 2 is active and there is no M or T hazard wait at level 1, the controller clears the level 1 activity bit. The answer to the "PP1" question block on the flowchart will always be "NO", since if the level 1 controller is in the PP1 state, there will be a Push instruction pointer word at level 2 and the pointer word always clears the level 2 activity bit when it enters level 2. Therefore, PP1 could not occur if level 2 is active.

4-220 LEVEL 1 CONTROLLER

The level 1 controller is an eleven state controller that determines when to transfer the level 1 register contents to the level 2 registers in the IPU, and indicates to the level 0 controller when a new instruction may be loaded into level 1 by transmitting "Path Ahead Clear" (PAC1). The initial state of the controller is the Big State. All other states, except DAV state, may be entered from the Big State; all other states may exit to the Big State. Figure 4-23 illustrates the interrelationship of the controller states and provides a flowchart of the logical progression through the control circuits. The following paragraphs describe the major decision paths within the controller as portrayed in the flowchart.

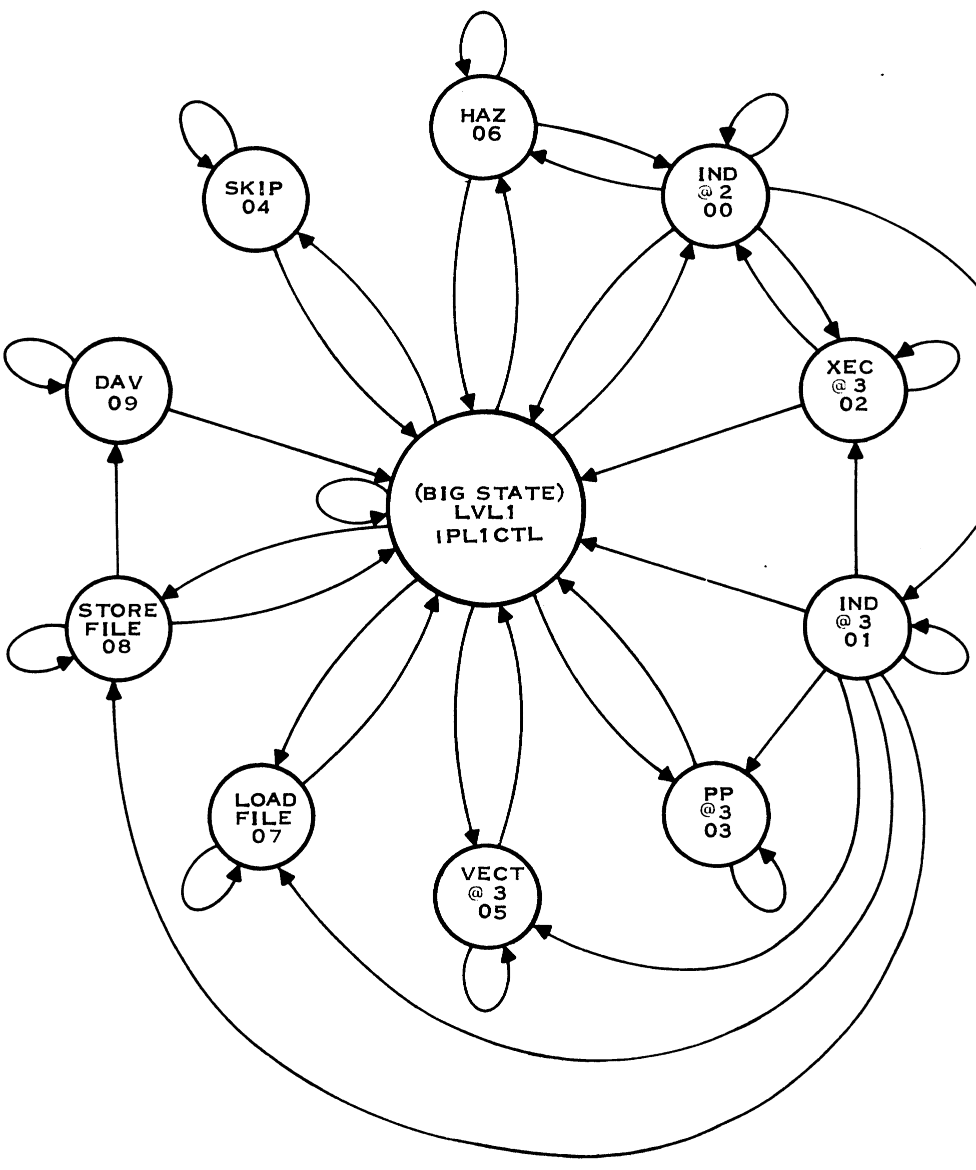
4-221 LEVEL 1 (BIG STATE)

The Big State of the level 1 controller monitors the instruction at level 3 and the conditions in the pipe to determine which of the ten other controller states, if any, must be entered to perform instruction transfer through the level 1 components of the IPU. Upon entering the Big State, the controller enables the outputs of the level 1 components to their respective destinations in level 2 of the IPU. When the level 1 to level 2 transfer occurs, these gates route the data to the correct level 2 components. The controller then examines the command at level 3 if no hazard or hold condition prohibits further action.

4-222 INSTRUCTION PATH CHANGE. If the level 3 command is an instruction or condition that negates the contents of the IPU pipe in favor of new instructions from memory or any portion of the IPU before level 1, (Target fail, hazard recovery, or branch to PA, LA or OA) the controller clears the level 2 activity bit at returns to the entry point of the state. This places a no-op at level 2, since the instruction that would have been transferred to level 2 is no longer useful.

4-223 SKIP. If the skip bit is set at level 3, the controller checks the level 2 activity bit. If the bit is not set (no instruction at level 2), the controller clears the level 2 activity bit again during the next transfer pulse. This results in transferring a no-op to level 2 in place of the instruction to be skipped. The controller also signals "Path Ahead Clear" (PAC) to the level 0 controller. If there is an instruction in level 1, the controller returns to the start of the Big State; if there is no instruction in level 1, the controller exits to the Skip state. If level 2 is active, but level 1 is not active, the controller transfers the activity of level 1 to level 2 during the next transfer cycle, signals PAC to level 0, and returns to the start of the control cycle. The answer to the PP1 question block will always be "NO" since if a push or pull instruction is at level 3, the pointer will be at level 2; the pointer always carries an inactive indication through the pipe. If both level 2 and level 1 are active, the skip command behaves identically to a branch to level 1 instruction.

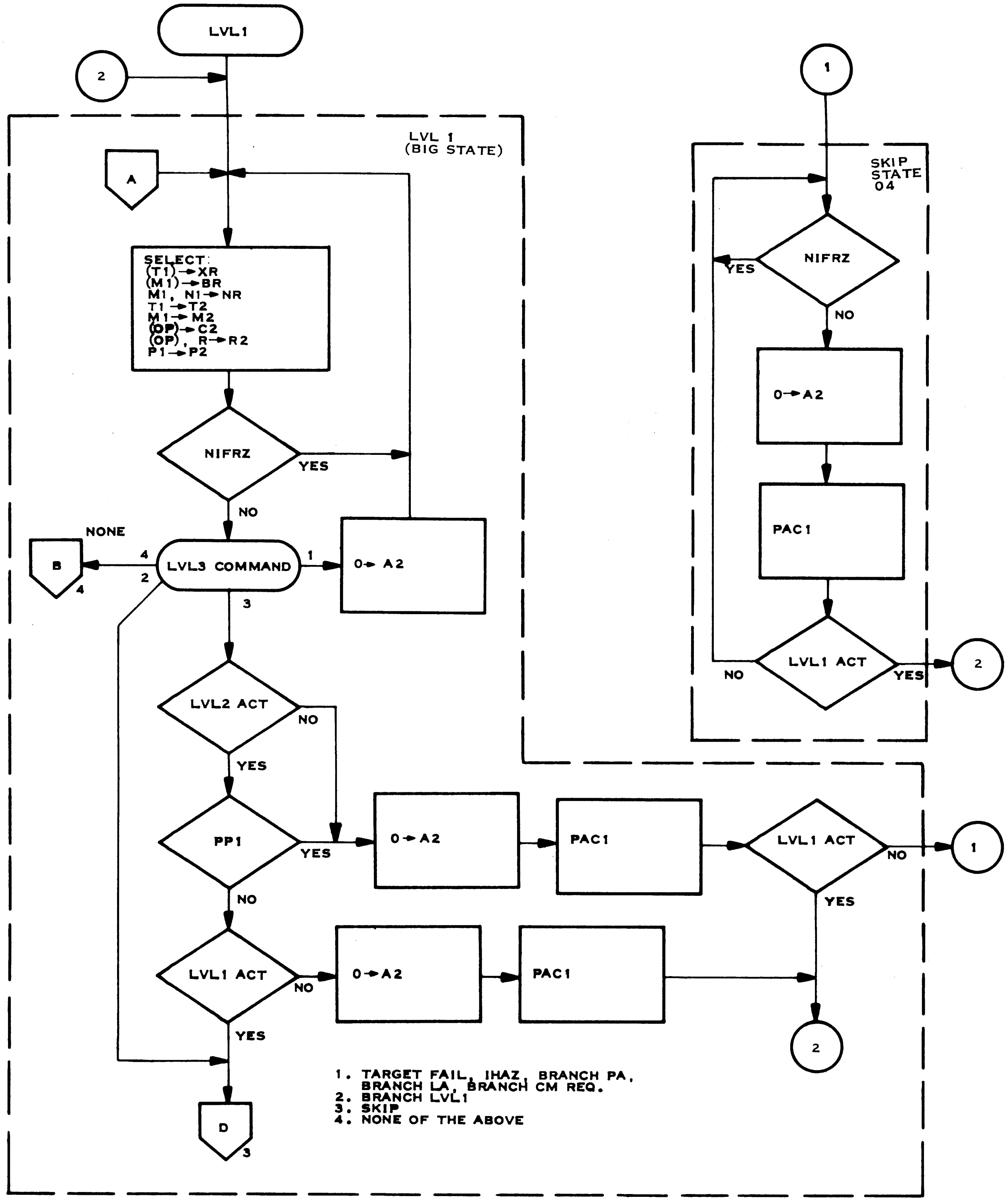
4-224 BRANCH TO LEVEL 1. If the instruction at level 3 specifies a branch to level 1 or a skip with both levels 1 and 2 active, the level 1 controller examines the level 1 instruction for hazards before forwarding it to level 2. The level 1 instruction must have no T or M hazard. If a hazard does exist, either the respective T or M field must be equal to zero, or the instruction must be in a format that does not use either the T or M hazard determination (T or M hazard free). If the instruction does not pass this inspection, the



IPQCTL (0-9)

(A)114304A

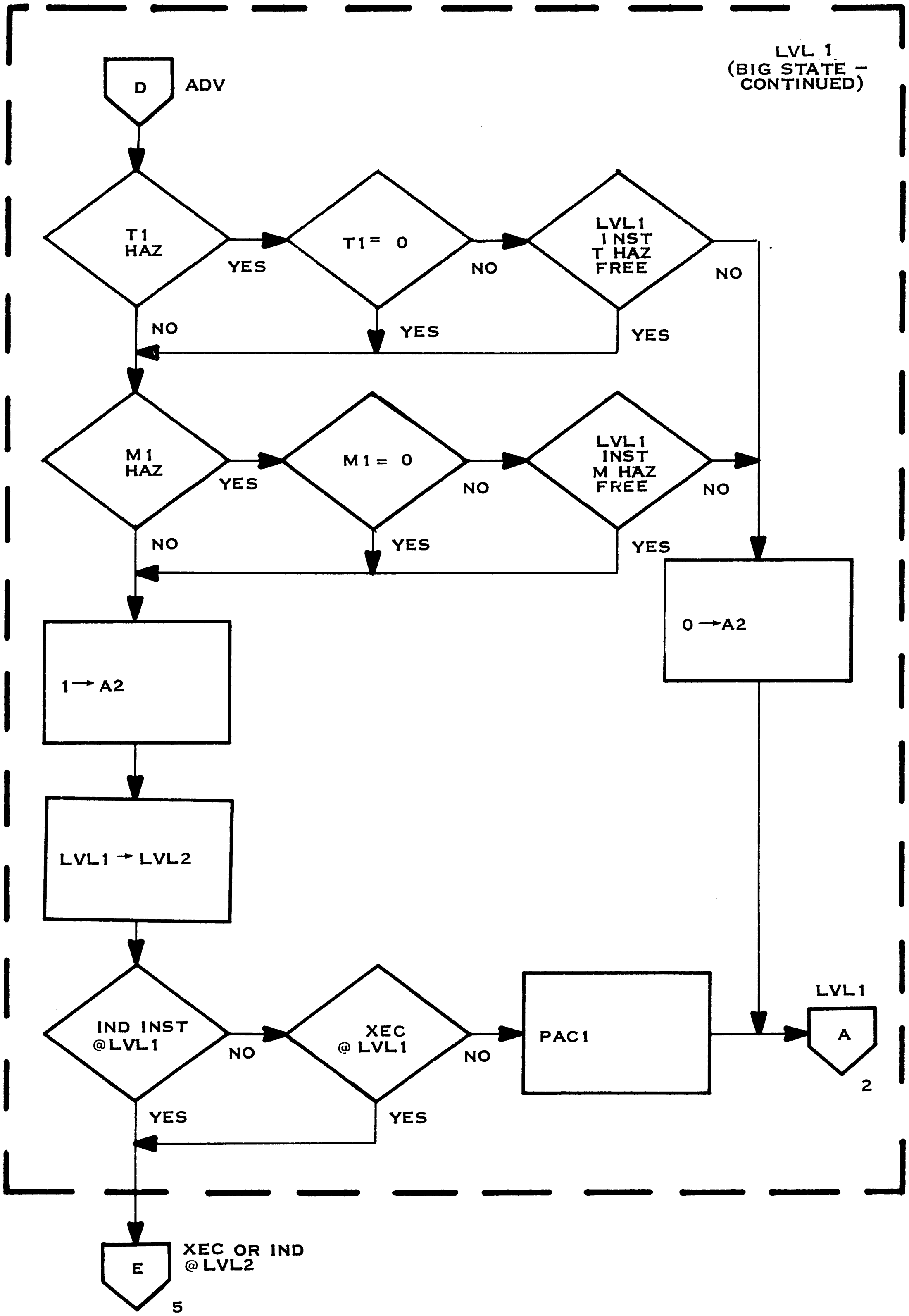
Figure 4-23. Level 1 Controller Flowchart (Sheet 1 of 12)



1. TARGET FAIL, IHAZ, BRANCH PA, BRANCH LA, BRANCH CM REQ.
2. BRANCH LVL1
3. SKIP
4. NONE OF THE ABOVE

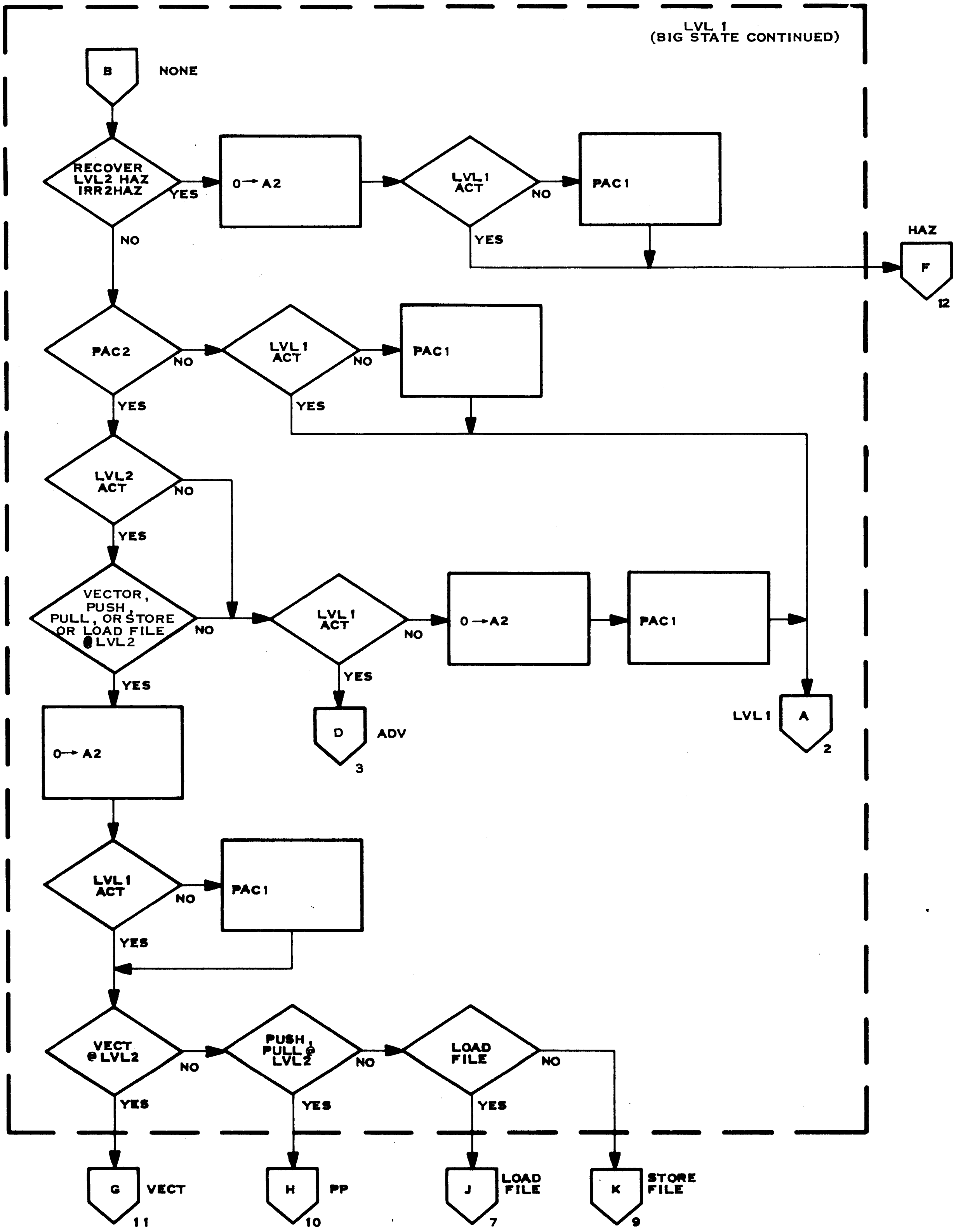
(A)114305A

Figure 4-23. Level 1 Controller Flowchart (Sheet 2 of 12)



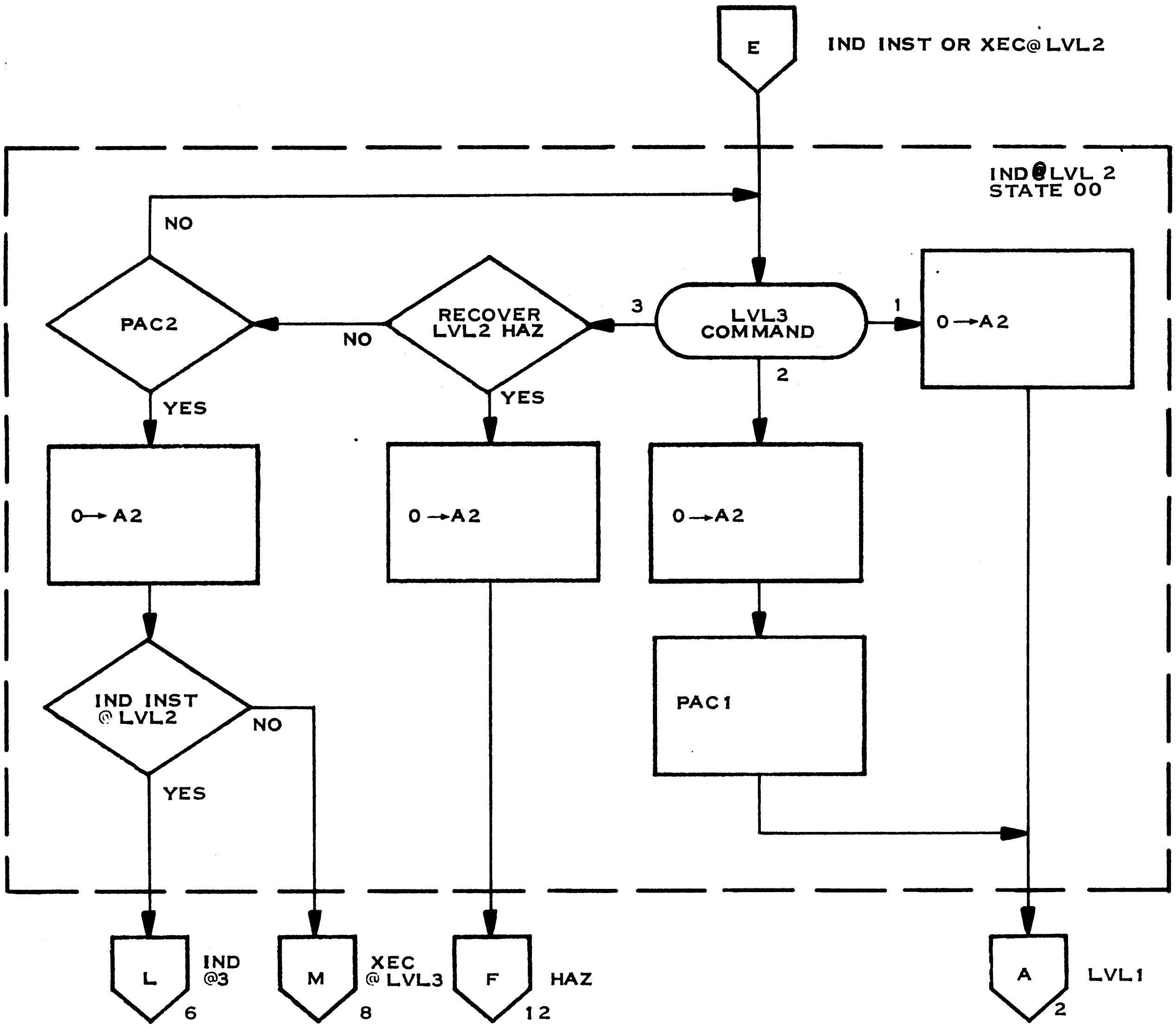
(A)114306A

Figure 4-23. Level 1 Controller Flowchart (Sheet 3 of 12)



(A)114307A

Figure 4-23. Level 1 Controller Flowchart (Sheet 4 of 12)



1. TARGET FAIL, IHAZ, BRANCH PA, BRANCH LA, BRANCH CM REQ.
2. SKIP
3. NONE OF THE ABOVE

(A)114308A

Figure 4-23. Level 1 Controller Flowchart (Sheet 5 of 12)

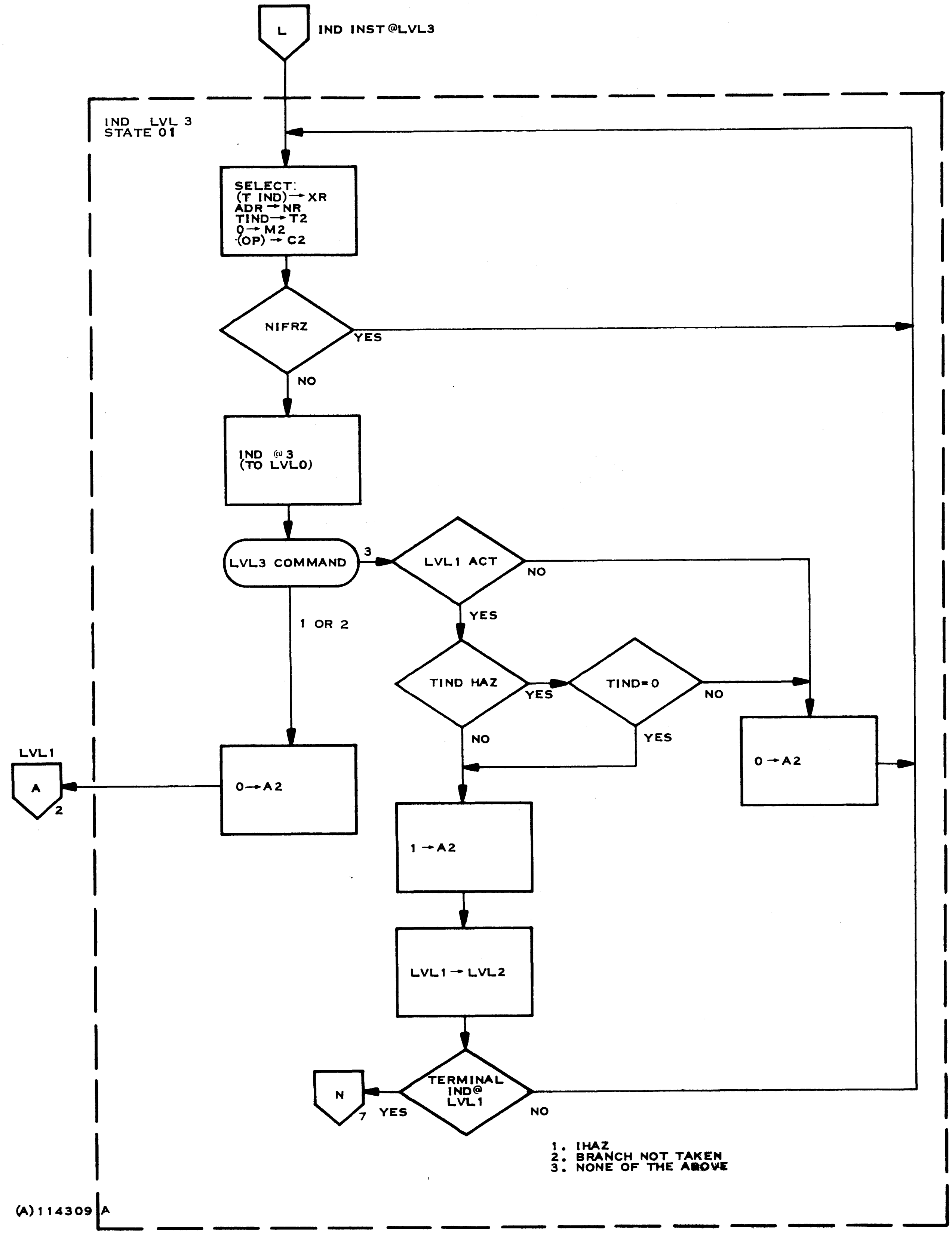


Figure 4-23. Level 1 Controller Flowchart (Sheet 6 of 12)

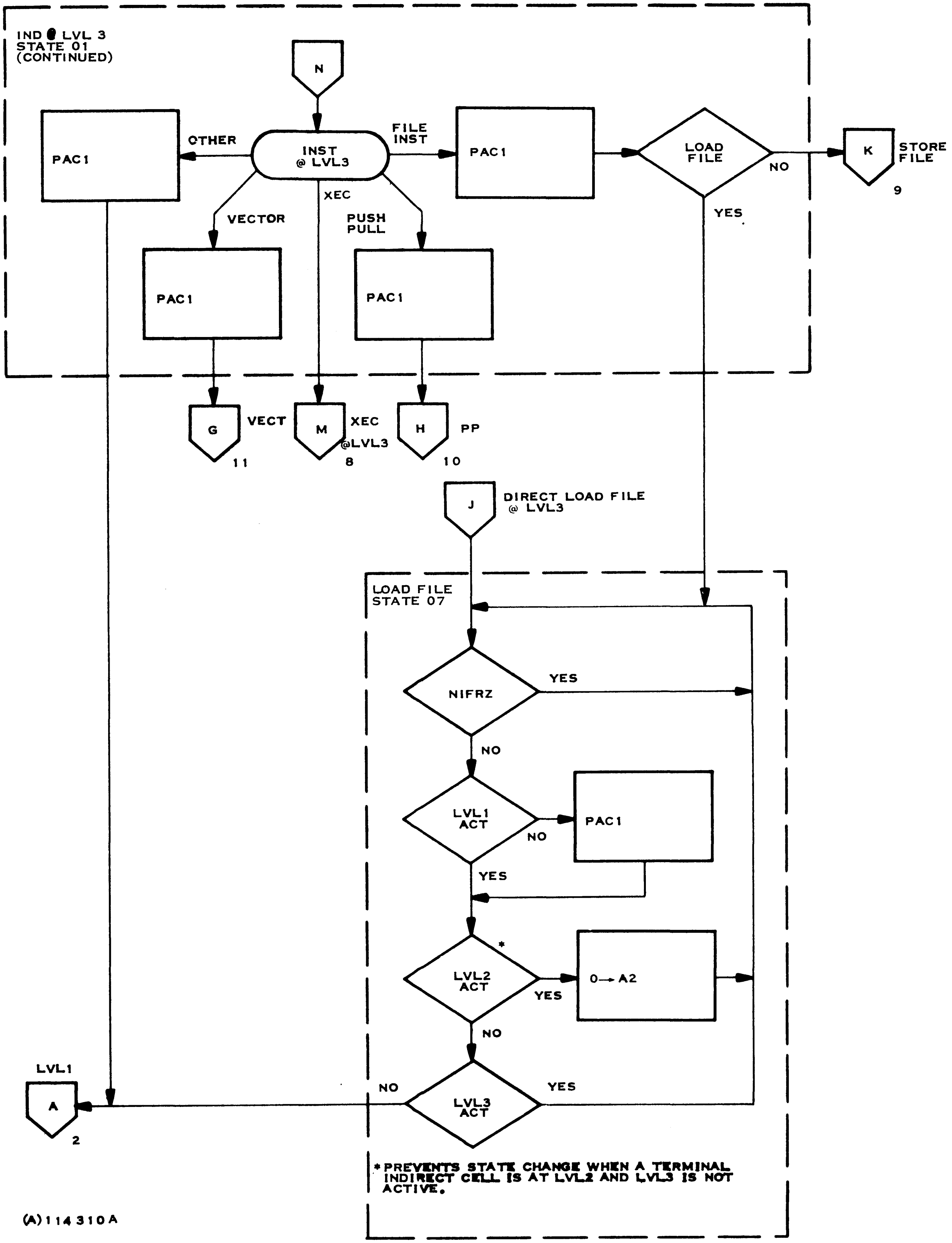
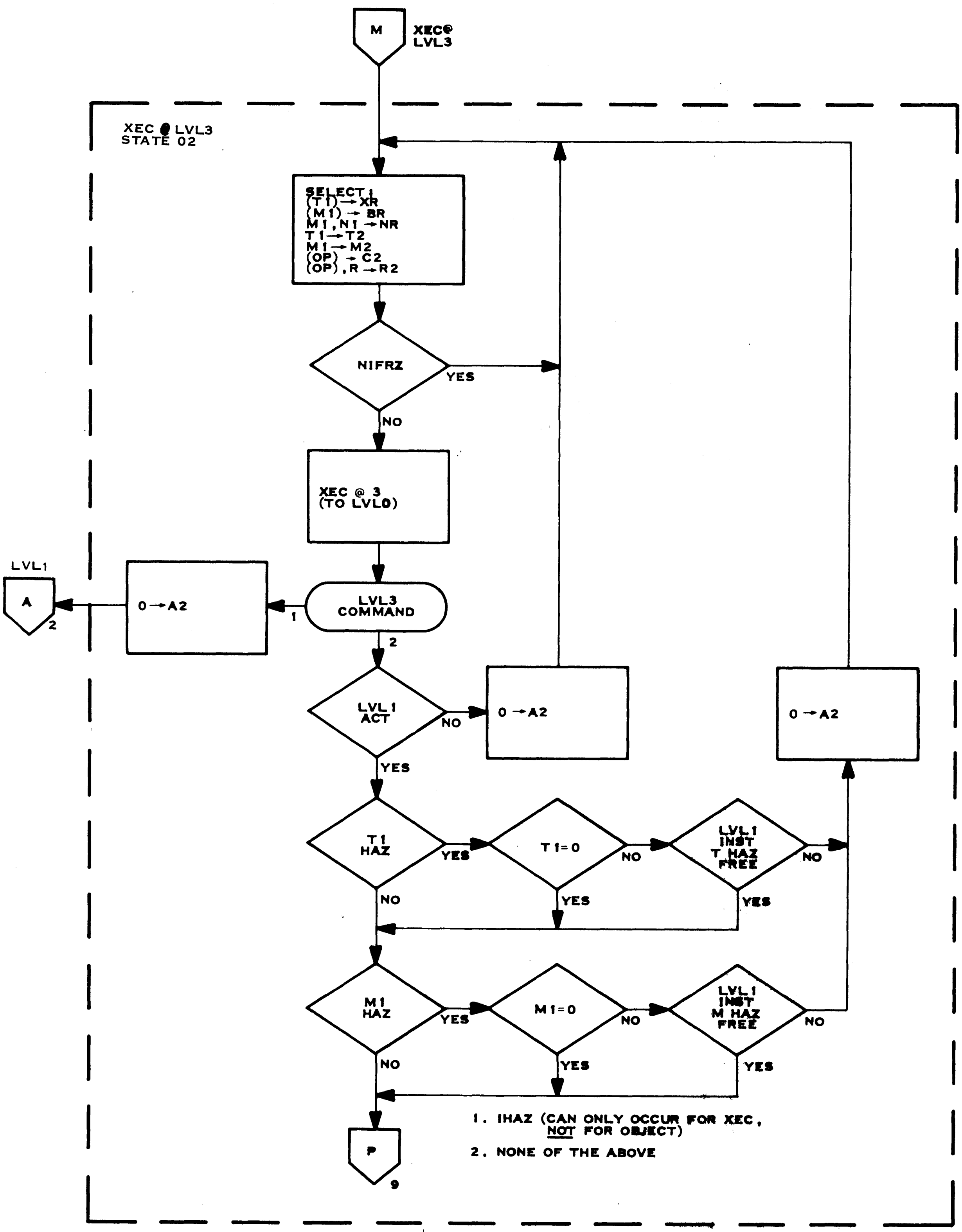


Figure 4-23. Level 1 Controller Flowchart (Sheet 7 of 12)



(A)114311A

Figure 4-23. Level 1 Controller Flowchart (Sheet 8 of 12)

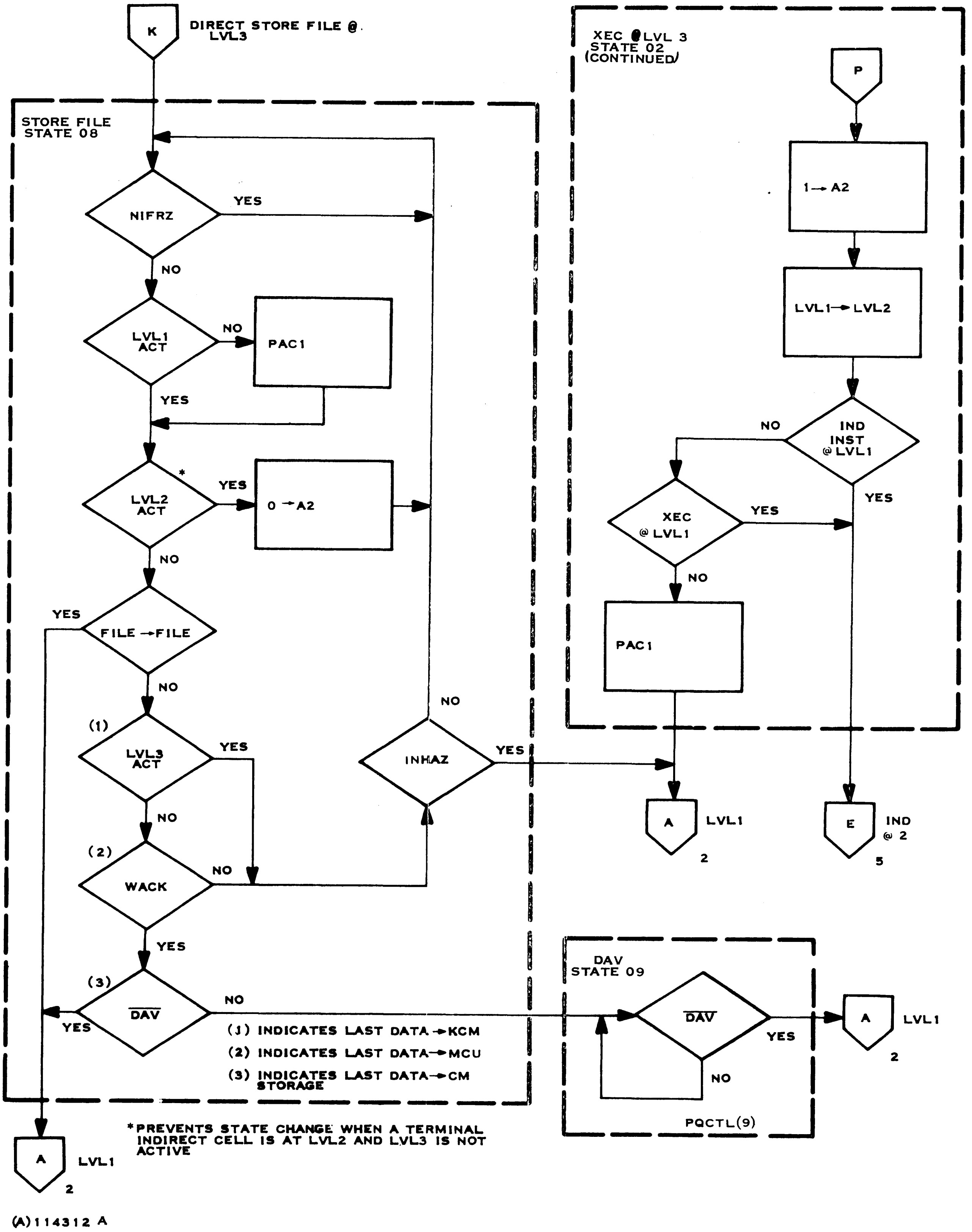
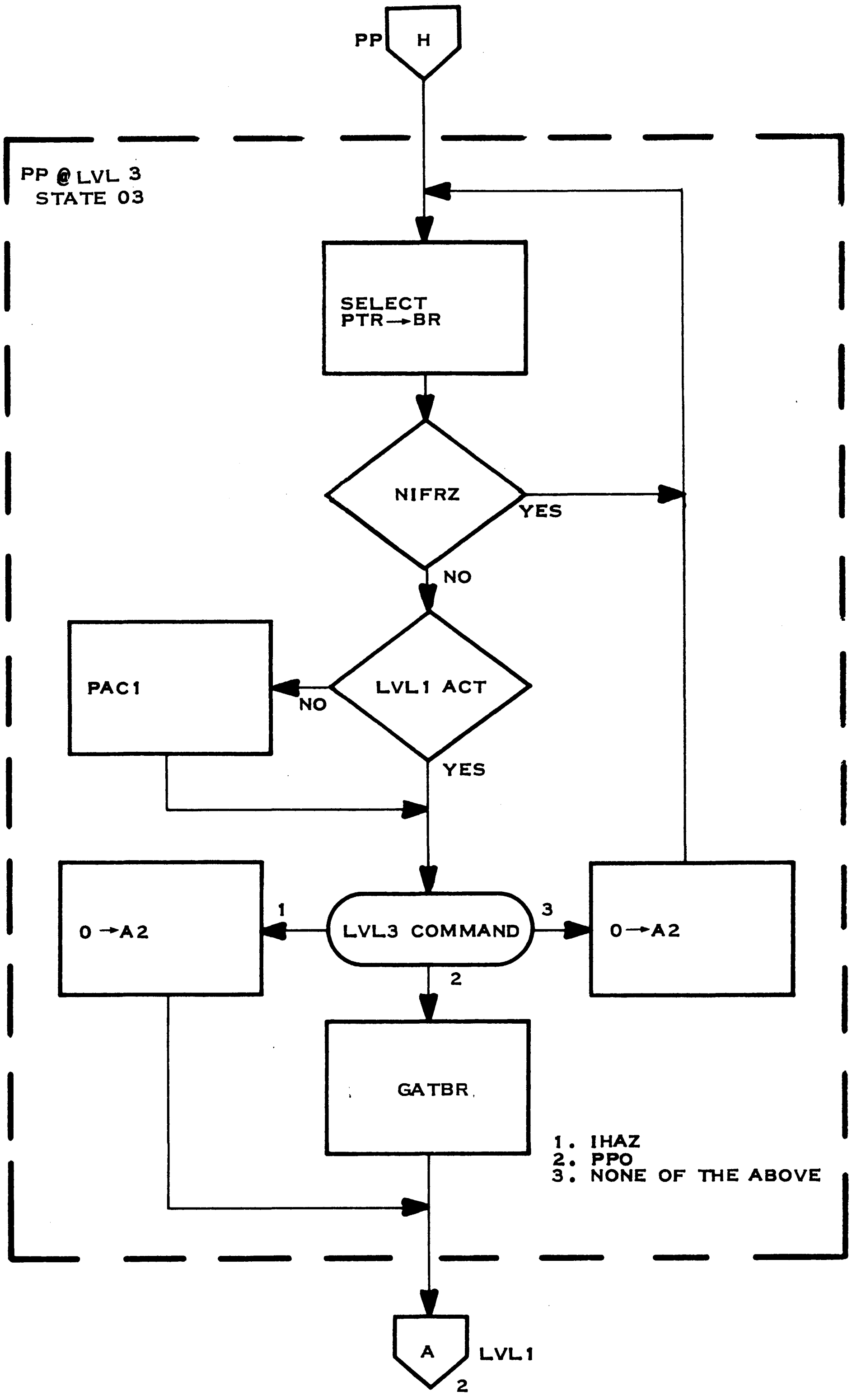
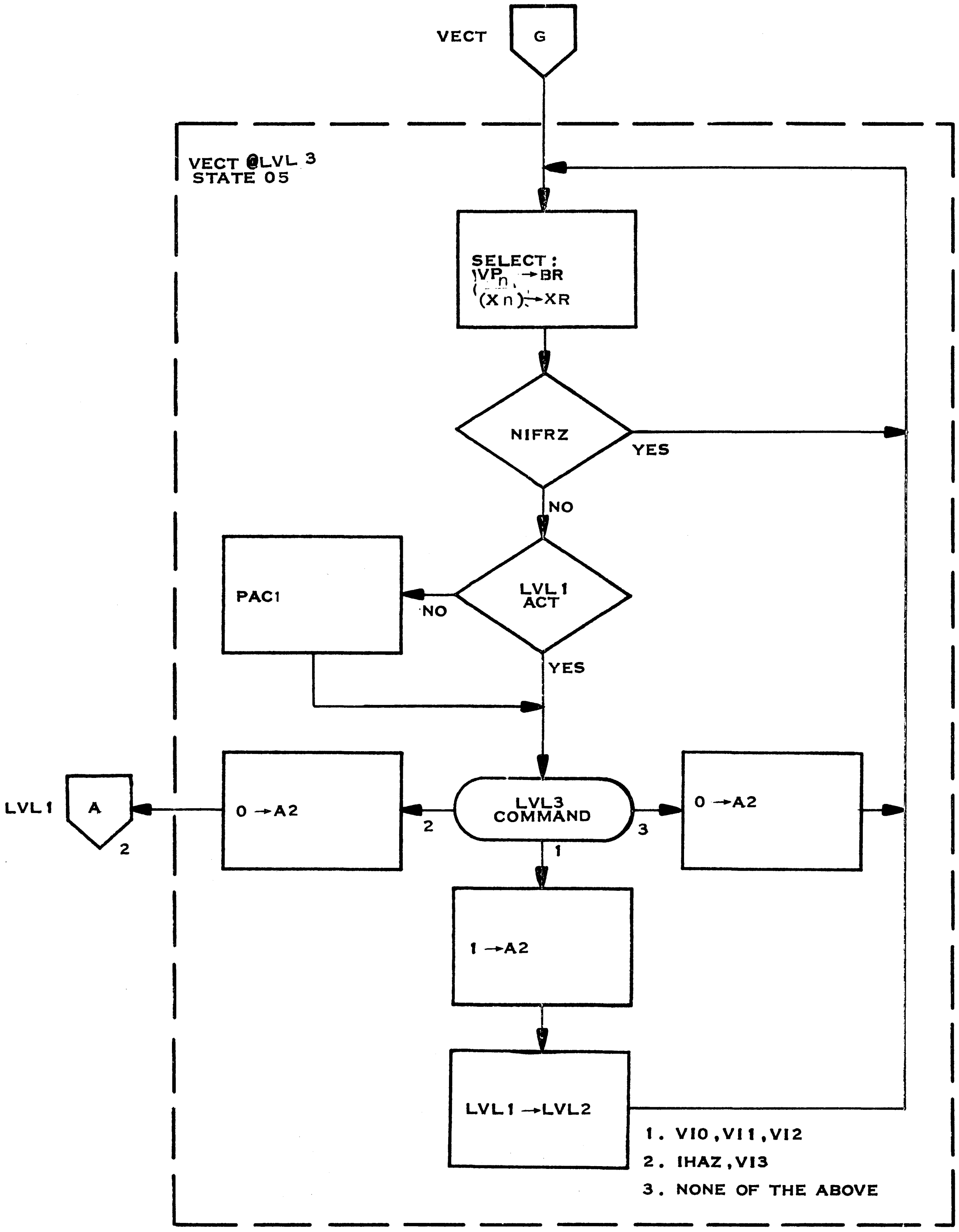


Figure 4-23. Level 1 Controller Flowchart (Sheet 9 of 12)



114313A

Figure 4-23. Level 1 Controller Flowchart (Sheet 10 of 12)



119492A

Figure 4-23. Level 1 Controller Flowchart (Sheet 11 of 12)

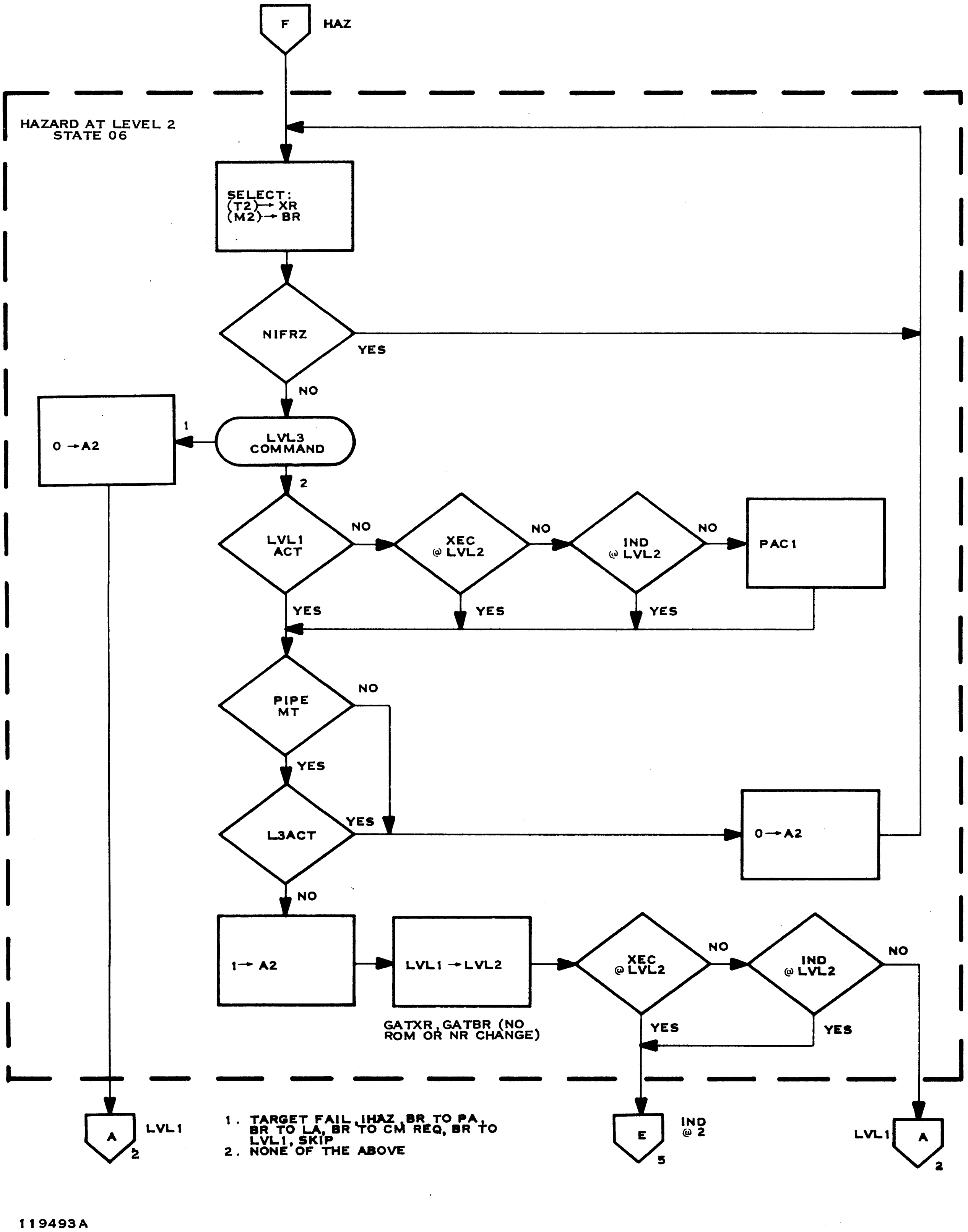


Figure 4-23. Level 1 Controller Flowchart (Sheet 12 of 12)

controller forwards a zero to the level 2 activity bit, holding the level 1 instruction at level 1 until the hazard clears. If the instruction is without conflict, the controller sets the level 2 activity bit and enables the transfer of the level 1 instruction to level 2 during the next transfer cycle. If the instruction to be transferred from level 1 to level 2 is either an indirect or an execute instruction, the controller enters the Indirect at level 2 state for the next control cycle. If the instruction is neither of these two types, the controller returns to the start of the Big State for the next control cycle.

4-225 RECOVER LEVEL 2 HAZARD. If a hazard occurs to the level 2 instruction such that the level 2 instruction must be refetched, the level 1 controller clears the level 2 activity bit, and if level 1 is not active, transmits PAC to the level 0 controller. The controller then enters the hazard state on the next control cycle.

4-226 NONE. When the controller has determined that no recoverable hazard exists at level 2, it checks the instruction type, activity bit at level 2 and the PAC signal from level 2 controller. If PAC is not present, the level 1 controller cannot perform any further functions. The controller returns to the start of the Big State, sending PAC1 to the level 0 controller if level 1 is not active. If level 2 is not active or if level 2 is active and the instruction is not a vector, push, pull, store file or load file, the controller checks the activity bit for level 1. If this bit is not set, the controller clears the level 2 activity bit during the next transfer cycle and generates PAC1 to the level 0 controller. If there is an instruction at level 1, the controller performs the inspection sequence previously described for the Branch to level 1 instruction before returning to the start of the Big State.

A vector, push, pull, store file or load file instruction at level 2 requires special action from the level 1 controller. If one of this type of instruction is present, the controller clears the level 2 activity bit during the next transfer cycle (transferring a no-op into level 2 while the instruction moves to level 3). If level 1 is not currently active, the controller also sends PAC to the level 0 controller. In either case the controller exits to either the Vector, Push-Pull, Load File or Store File state for the next control cycle.

4-227 SKIP STATE

The level 1 controller enters the skip state from the Big State and returns to that state when the requirements of the skip state have been satisfied. In this state, the controller transmits a no-op to level 2 and signals PAC1 to the level 0 controller. When an instruction enters level 1, setting the level 1 activity bit, the controller exits to the Big State.

4-228 INDIRECT OR EXECUTE AT LEVEL 2 STATE

The level 1 controller enters the indirect or execute at level 2 state from the Big State, the hazard state, or the Execute at level 3 state. It can return to any of these states plus the Indirect at level 3 state, or loop within itself waiting for PAC2. Upon entering this state the controller examines the command in level 3. If the instruction will negate the contents of the pipe (target fail, hazard, branch to PA, LA or OA), the controller transfers a zero to the level 2 activity bit and exits to the Big State. If the instruction is a skip, the controller clears the level 2 activity bit, generates a PAC1 signal to the level 0 controller so that a new instruction can be transferred into level 1 and exits to the Big State. If the level 3 command does not fall into either of these categories, the controller determines if there is a recoverable hazard at level 2. If a hazard does exist at level 2, the controller transfers a zero into the activity bit for level 2 and exits to the hazard state. If no hazard is at level 2, the controller transfers a zero to the activity bit for level 2 to clear the pipe for the impending instruction from the indirect or execute. The controller then exits to the Indirect at level 3 state or to the Execute at level 3 state, depending upon the type of instruction that has been transferred to level 3.

4-229 INDIRECT AT LEVEL 3 STATE

The level 1 controller enters this state from the Indirect at Level 2 state when the CP clock transfers the contents of level 2 to level 3. When the controller enters this state it enables the inputs to the level 2 registers that will be used to develop the required address from the indirect address. Then, if no status freeze condition exists, the controller transmits an indirect at level 3 indication to the level 0 controller and examines the command at level 3. If the instruction at level 3 is a hazard recovery or a branch (preceded by an LLA) that is not taken, the controller clears the level 2 activity bit during the next clock pulse and exits to the Big State. If neither of these conditions is present at level 3, the controller checks the level 1 activity bit. If this bit is not set, the controller clears the level 2 activity bit on the next clock pulse and loops within the Indirect at Level 3 State for the next control cycle. If level 1 contains an active instruction, and there is no hazard condition in the pipe that affects the register containing the index modifier for the indirect address, the controller sets the level 2 activity bit and enables the transfer of level 1 to level 2 during the next clock pulse. If the instruction at level 1 is not the last step in the indirect process, the controller loops within the Indirect at level 3 state until the terminal indirect is reached. At that point the controller again examines the instruction at level 3 to determine the exit destination for the next control cycle. If the instruction is an execute, the controller reserves level 1 for the upcoming instruction and exits to the Execute at Level 3 state. All other commands produce a PAC indication to the level 0 controller. The level 1 controller then exits to the Vector, PP, Load File, Store File or Big State depending upon the instruction type at level 3.

4-230 LOAD FILE STATE

The controller enters the Load File state from either the Indirect at Level 3 state or the Big State. When in the Load File state, the controller waits for the Load File instruction at level 3 to complete execution. If level 1 is not active, the controller generates a PAC indication to the level 0 controller so that a new instruction may transfer to level 1. If an instruction is at level 2, the controller loops within the Load File state and clears the level 2 activity bit during the next transfer pulse. This loop ensures that the controller will remain in the Load File state if level 3 is inactive due to waiting for development of the indirect address.

4-231 EXECUTE AT LEVEL 3 STATE

The level 1 controller enters this state from either the Indirect at Level 2 state or the Indirect at Level 3 state. When the controller enters this state it enables the inputs to the level 2 registers that will develop the instruction to be executed. Then, if no status freeze condition exists, the controller transmits an execute at level 3 indication to the level 0 controller and examines the command at level 3. If the execute instruction references an address whose contents will be changed by an instruction already in the pipe, the controller clears the level 2 activity bit during the next transfer pulse and returns to the Big State for the hazard condition to clear. If no hazard condition occurs, but level 1 is inactive, the controller clears the level 2 activity bit on the next clock pulse and loops within this state. If there is an active instruction in level 1, the controller ensures that the instruction (object of Execute) either has no T or M hazards, or if a hazard exists the hazard is ignored due to the corresponding field being zero or the instruction being in the format where a T or M field is not used. If these conditions are not met, the controller clears the level 2 activity bit and returns to the start of Execute at Level 3 state. If the level 1 instruction passes the hazard inspection, the controller sets the level 2 activity bit and enables the transfer of the level 1 instruction to level 2 during the next clock pulse. If that instruction is either an indirect or execute instruction, the controller exits to the Indirect at Level 2 state. If the instruction is neither an indirect nor an execute, the controller signals PAC1 to the level 0 controller and exits to the Big State.

4-232 STORE FILE STATE

The level 1 controller enters the store file state from either the Big State or from the Indirect at Level 3 state. This state is essentially a wait state that the controller enters while a Store File operation is being performed at level 3. If there is no instruction in level 1, the controller issues PAC1 to the level 0 controller to enable transfer of a new instruction into level 1. If level 2 is active, the controller clears the level 2 activity bit on the next transfer pulse and loops within the Store File state. This loop allows the controller to remain in the Store File state even though level 3 may be inactive while waiting for resolution of an indirect address. If the Store file operation is a transfer from one register file octet in the IPU to another register file octet, the controller immediately exits to the Big State since no wait will be required because a memory access is not involved. If the operation is not a file to file transfer, the controller waits until the level 3 activity bit clears and the Central Memory Requester (CMR) signals that it is ready for another write request (WACK), signifying that the last octet of the store file operation has been accepted by the MCU. When both of these conditions are satisfied, the controller exits to the Big State if the ¬DAV signal from the MCU has dropped, or to the DAV state if the ¬DAV signal has not dropped.

4-233 DAV STATE

The controller enters the DAV state from the Store File state. In this state the controller waits until the DAV signal from the MCU drops. DAV from the

MCU indicates that the write data from the IPU has been accepted but has not yet been stored. When the signal drops, the data has been stored into memory. The controller then returns to the Big State for the next control cycle.

4-234 PUSH - PULL STATE

The controller enters the PP state from either the Big State or the Indirect at Level 3 state. As the controller enters this state it enables the input to BR at level 2 so that when the transfer pulse occurs, the selected pointer associated with the Push or Pull at level 3 will be transferred into the BR register. If no status freeze conditions have occurred and level 1 is not active, the controller issues PAC1 to the level 0 controller. In either activity state at level 1, the controller inspects the state of the level 3 command. If an instruction hazard recovery is required, the controller clears the level 2 activity bit on the next clock pulse and returns to the Big State. If the level 3 command is a valid push or pull, the controller transfers the pointer into BR (GATBR) on the next clock pulse and returns to the Big state. If neither of these situations occur, the controller clears the level 2 activity bit and returns to the start of the PP state.

4-235 VECTOR STATE

The level 1 controller enters the Vect state from the Indirect at Level 3 state or from the Big State during Vector initialization. When the controller enters the Vect state it enables the first word from the vector parameter file into the BR register (VPn, where n = 0), and the corresponding index value into the XR register for formation of the initial vector values sent to the MBU. If no freeze conditions exist, the controller generates PAC1 to the level 0 controller if no instruction is currently in level 1 and then examines the command in level 3. If the command is one of the first three words in the vector parameter field, the controller transfers a 1 to the level 2 activity bit and produces the gating signals during the next clock pulse to transfer the next parameter into level 2 of the IPU. The controller returns to the start of the Vect state to repeat until vector file word three is in level 3. When word three of the vector parameter file reaches level 3, the controller clears the level 2 activity bit and exits to the Big State since all remaining vector parameter file words transfer directly to level 3 without passing through the modification network.

4-236 HAZARD STATE

The level 1 controller enters the Hazard state when an instruction at level 3 of the pipe modifies the register file location that is used to modify the instruction at level 2 of the pipe. Since the modification parameters for that instruction specified by its T and M fields have already been loaded into level 2 registers (BR and XR), the level 1 controller must replace these faulty values with the updated values when the hazard is resolved. Therefore, when the controller enters the hazard state, it enables the selection of the register specified by the T2 register to XR and the register specified by the M2 register into BR. If there is no status freeze condition, the controller then examines the instruction at level 3. If that instruction will abort the instruction stream that is in level 2 of the pipe (target fail, hazard recovery, Branch to PA, LA, OA or level 1, or a Skip instruction), then the level 1 controller does not need to update that instruction. The controller therefore, clears the level 2 activity bit and exits to the Big state. If the level 3 instruction is not in the above group, the controller inspects the level 1 activity bit. If that bit is not set and there is not an indirect or execute instruction at level 2, the controller generates PAC 1 to the level 0 controller. The controller then waits for the pipe to clear of all instructions and for level 3 to become inactive to ensure that the hazard has cleared. When the hazard condition no longer exists, the controller sets the level 2 activity bit and transfers the base and indexing parameters into XR and BR without changing the contents of any of the other level 2 registers. If the new instruction in level 2 is an indirect or execute, the controller exits to the Indirect at Level 2 state; if not, the controller returns to the Big State for the next control cycle.

4-237 LEVEL 2 CONTROLLER

The level 2 controller monitors status conditions from the level 3 controller and determines when the contents of level 2 will be transferred to level 3. When a status freeze condition occurs and a new instruction is being passed through level 2, the new instruction flag, NI, sets. The level 2 controller also communicates with the level 1 controller by indicating that the Path Ahead is Clear to level 2 (PAC2). The following paragraphs describe the major logic paths followed by the level 2 controller with reference to the flowcharts of the controller logic illustrated in figure 4-24.

4-238 SELECT ADDER INPUT

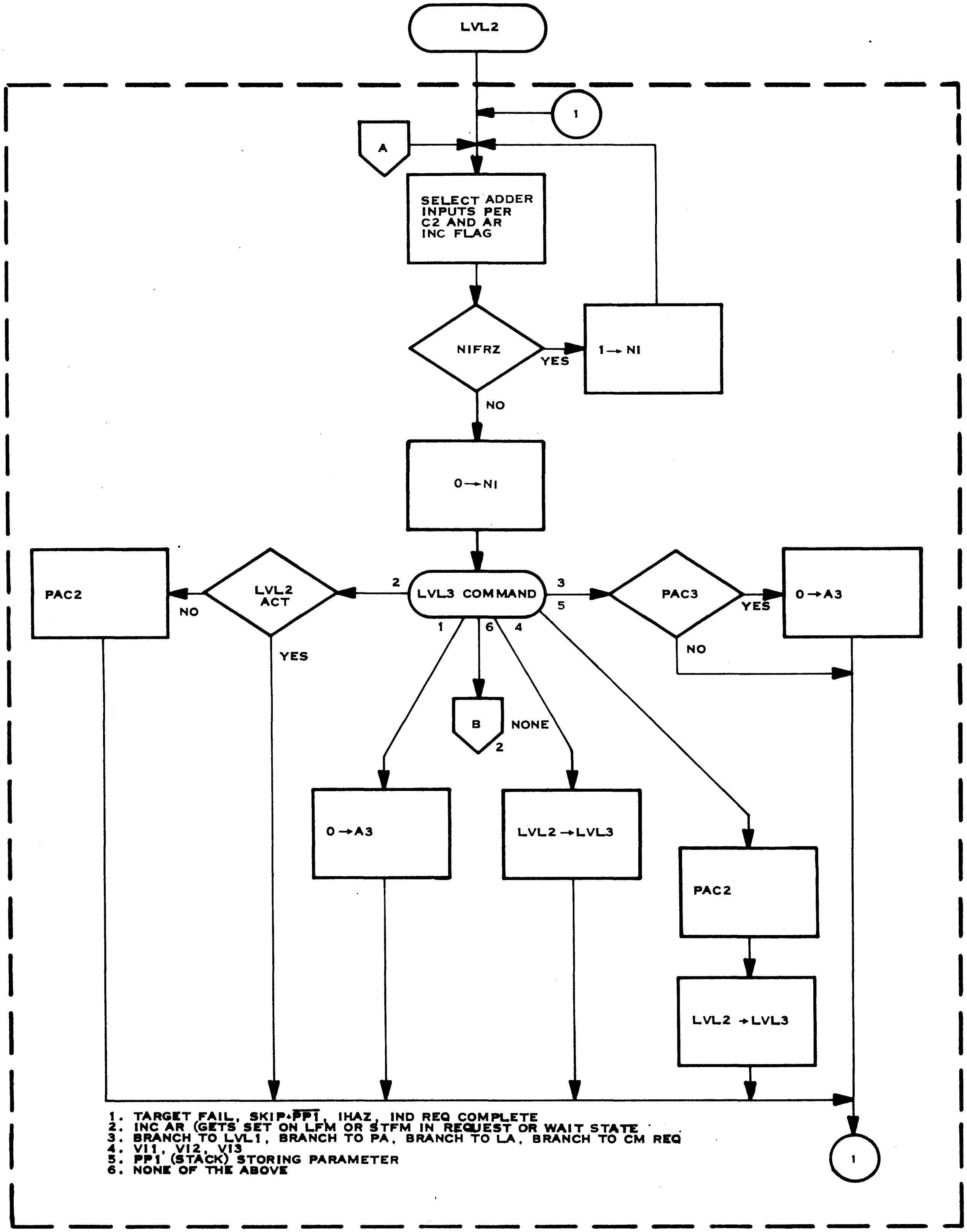
At the start of each control cycle, the level 2 controller examines the C2 output and the AR Increment flag to determine what inputs to the modification adder are required for development of the next instruction. AR increment is used during load file multiple and store file multiple instructions. The controller keeps NI set if a NI Freeze condition is at level 3. If NIFRZ is not present, the controller allows NI to clear and examines the command at level 3.

4-239 CONTENTS OF LEVEL 2 NOT USEFUL

If the instruction at level 3 either bypasses the instruction at level 2 (skip or branch) or indicates that the instruction at level 2 is not valid (Hazard recovery or indirect request complete), the controller clears the activity bit in level 3 during the next transfer cycle and returns to the start of the state. For branch instructions, clearing the level 3 activity bit is dependent upon PAC3 from the level 3 controller (path 3 on the flowchart). The remaining conditions do not require the PAC indication to clear the activity bit (path 1).

4-240 INCREMENT AR

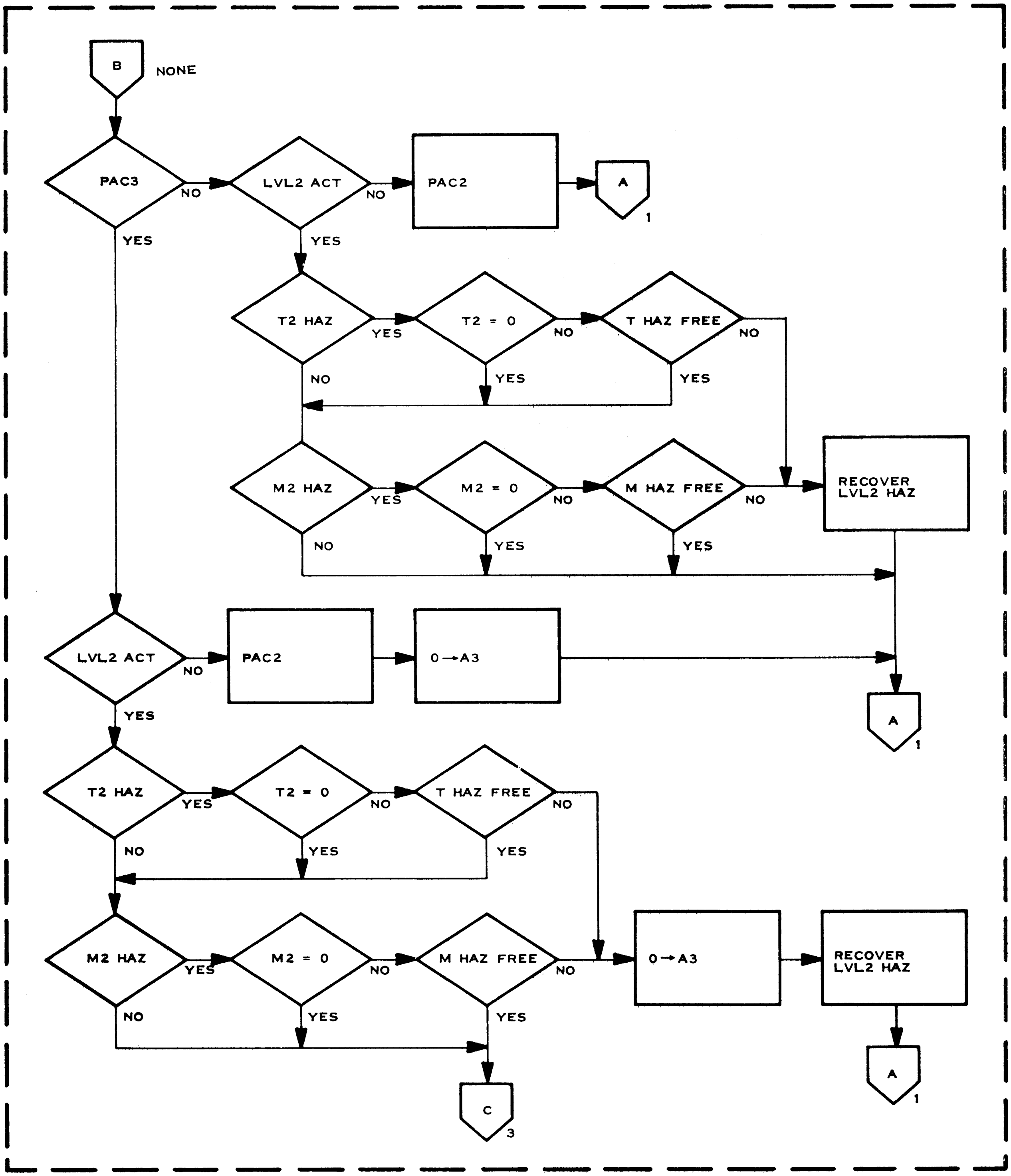
If a load file multiple or store file multiple instruction at level 3 has generated Increment AR, the controller returns to the start of the control cycle if valid data is in level 2. If level 2 is not active, the controller generates PAC2 to indicate its availability before returning to the start of the control cycle.



1. TARGET FAIL, SKIP+$\overline{\text{PP1}}$, IHAZ, IND REQ COMPLETE
2. INC AR (GETS SET ON LFM OR STFM IN REQUEST OR WAIT STATE
3. BRANCH TO LVL1, BRANCH TO PA, BRANCH TO LA, BRANCH TO CM REQ
4. VI1, VI2, VI3
5. PP1 (STACK) STORING PARAMETER
6. NONE OF THE ABOVE

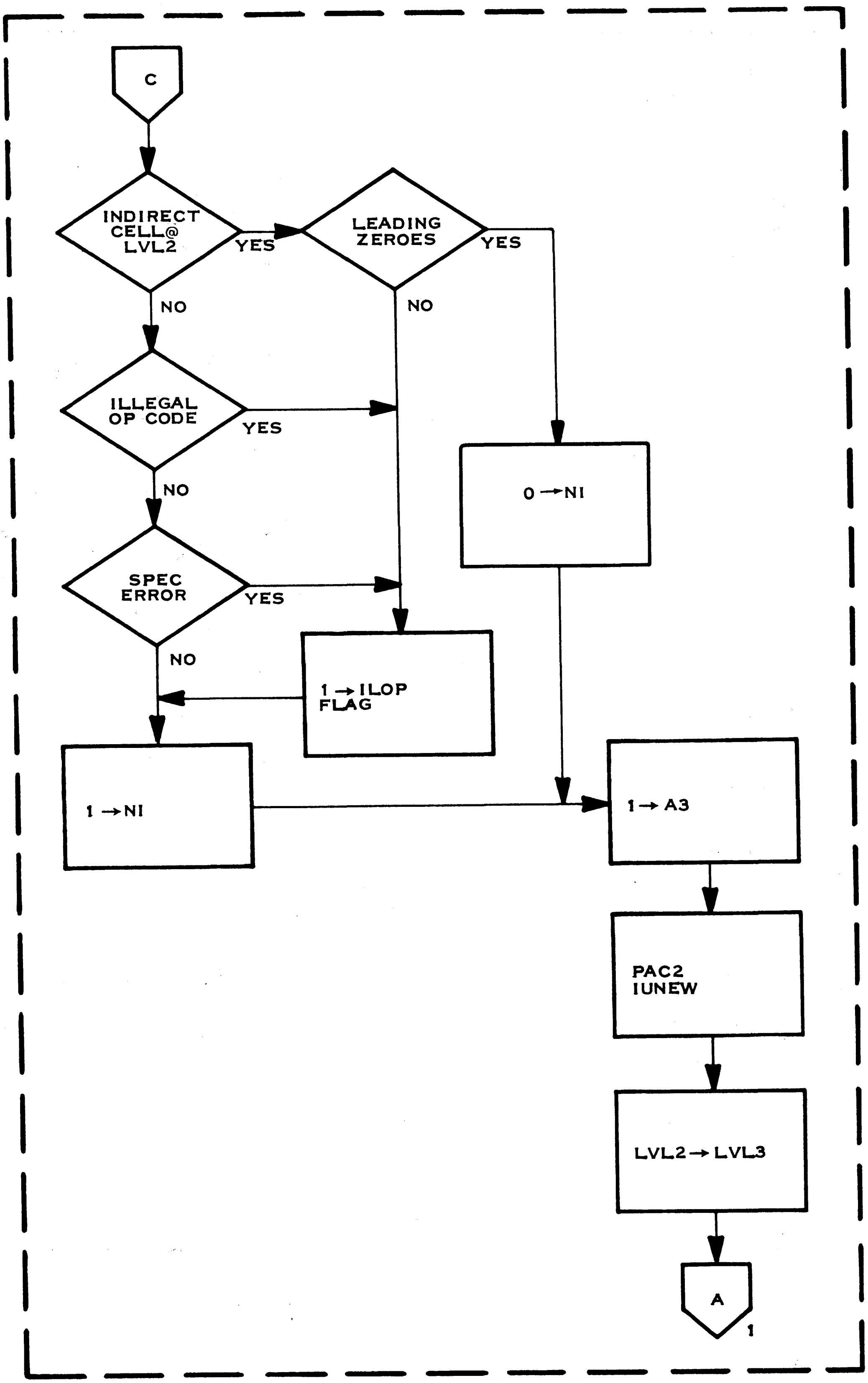
(A)114316 A

Figure 4-24. Level 2 Controller Flowchart (Sheet 1 of 3)



(A)114317A

Figure 4-24. Level 2 Controller Flowchart (Sheet 2 of 3)



119487A

Figure 4-24. Level 2 Controller Flowchart (Sheet 3 of 3)

4-241 VECTOR PARAMETER FILE

If any of the three words from the vector parameter file that define the three vector starting points is at level 3 (V1, V2 or V3 of the file), the controller enables transfer of the level 2 contents to level 3 on the next clock pulse. If the word at level 3 is the V3 word, however, the level 3 controller generates a level 3 to level 4 enable signal that transfers the next word of the vector parameter file (V4) directly to level 4, since the remaining words in the file cannot be modified by the IPU address modification network.

4-243 PUSH - PULL

If the instruction is a push or pull instruction, then level 2 contains the pointer for that instruction and the level 2 activity bit is clear. The controller enables the transfer of the pointer to level 3 on the next control clock and signals PAC2 to the level 1 controller before returning to the start of the control cycle for the next clock period.

4-243 NONE

If none of the preceding conditions exist at level 3, the controller checks the activity at both levels 2 and 3. If level 2 is not active, the controller issues PAC2 to the level 1 controller, and clears the level 3 activity bit on the next clock pulse if the level 3 controller has issued PAC3. If level 2 is active, the controller examines the contents of level 2 to determine if a T or M hazard exists for an instruction that is not hazard free (instruction format does not use T or M fields). If a hazard does exist that is valid for the instruction, the controller indicates to the level 1 controller that a level 2 hazard recovery must be performed. If PAC3 was active, the controller also clears level 3 activity upon detection of a level 2 hazard, since the instruction passed from level 2 to level 3 is not a valid instruction.

When the controller determines that no valid hazards are present, that level 2 is active, and level 3 can accept a new instruction (PAC3), the controller examines the level 2 instruction for an illegal operation code. If the instruction is an indirect cell that has a one bit in any of bits 0-3, is an illegal op code, or contains a specification error, the controller sets the Illegal Operation (ILOP) flag to cause an interrupt in program sequencing. Regardless of the outcome of the inspection for illegal op code, the controller sets the level 3 activity bit, enables the transfer of level 2 to level 3 and transmits PAC2 to the level 1 controller. If the instruction at level 2 was not an indirect cell that contained the correct leading zeroes in bits 0-3, the controller sets the New Instruction flag (NI) to the level 3 controller. The indirect cell does not set this flag since it is not a new instruction.

4-244 LEVEL 3 CONTROLLER

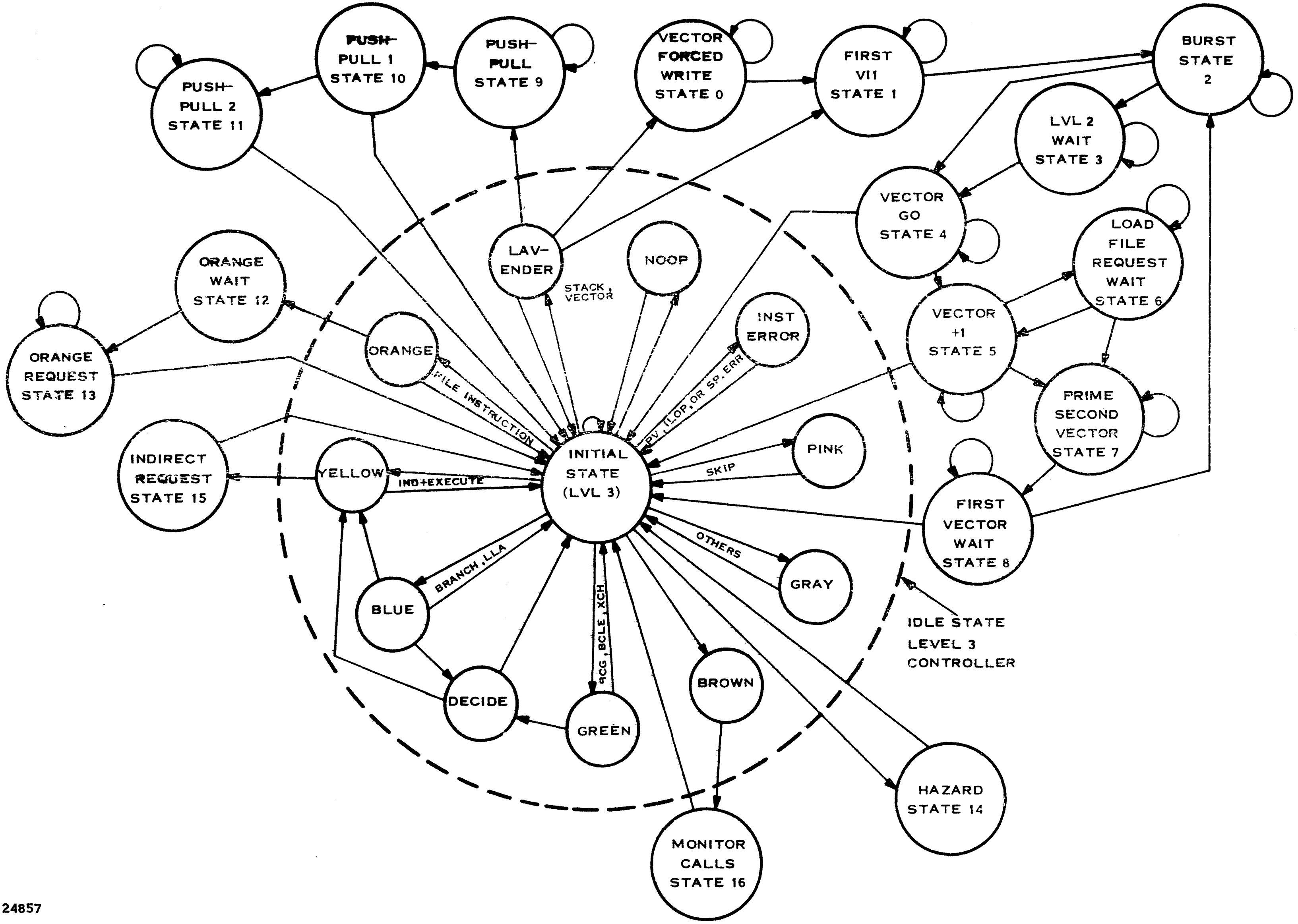
The level 3 controller decodes the instruction at level 3 of the IPU, determines the transfers necessary to perform the designated function, enables those transfers and establishes conditions in the MBU such that the operation will enter the pipe and be executed. In addition, the controller checks for hazards and short circuit paths available to ensure that the instruction is accurately and rapidly processed. The controller is an 18 state device. Figure 4-25 illustrates the relationship of these states and the 12 subcycles of the Idle state to each other. Each state represents logic that executes during one clock pulse. The flowcharts and description that follow explain the action and decisions in each state. Because the flowcharts represent actions that occur during the clock pulse, they are not in sequential order as arranged in the flowchart. Instead the controller examines the conditions that exist before the clock pulse, and enables execution of the action blocks determined by those conditions when the clock pulse occurs. In this description, the term "control cycle" applies to that period before the clock pulse during which the controller examines all decision blocks.

4-245 IDLE STATE

The Idle state of the level 3 controller consists of one initial cycle that examines conditions in level 3, determines the required responses to those conditions and chooses one of eleven subcycles of the Idle state or another state to provide that reaction. When the controller completes processing the instruction in one of the subcycles or states, the controller returns to the beginning of the Idle state for inspection of the next new instruction that enters level 3, or for re-examination of the previous instruction during control loops. The following paragraphs describe the major logic paths and decisions followed by the level 3 controller in the Idle state.

4-246 INITIAL STATE. The initial Idle state checks for hazard conditions, instruction violations, status freeze, and an instruction type to determine the proper path required by the level 3 controller to perform all inspections and transfers for each set of conditions. The logic flow of the initial state is illustrated in figure 4-26. If level 3 activity bit is not set at the start of the control cycle, the controller sets the PAC3 indication to the level 2 controller and loops for each clock until the activity bit sets (Sequence A is a no operation sequence). If level 3 activity is set, and the instruction in level 3 is a new instruction (NI flag set), the controller checks for a status freeze condition. If a freeze condition exists, the controller issues a new instruction freeze indication to the level 2 controller after data in the Z- buffer is stored into memory. If the storage octet is still in the ZA file in the MBU (ZA full) the controller initiates a forced write cycle to store that octet so that the hazard may be recovered. In either case the controller loops on each control cycle in which the status freeze condition is still present.

Once the status freeze condition clears, the controller checks for an instruction error. An instruction error indicates that the IPU encountered a protect violation when it tried to access the instruction in memory, that the instruction is an illegal op code, or that there is some type of specification error in the format of the instruction. If the controller detects any of these conditions, it exits to the Instruction Error subcycle.



(B) 124857

Figure 4-25. Level 3 Controller State Diagram

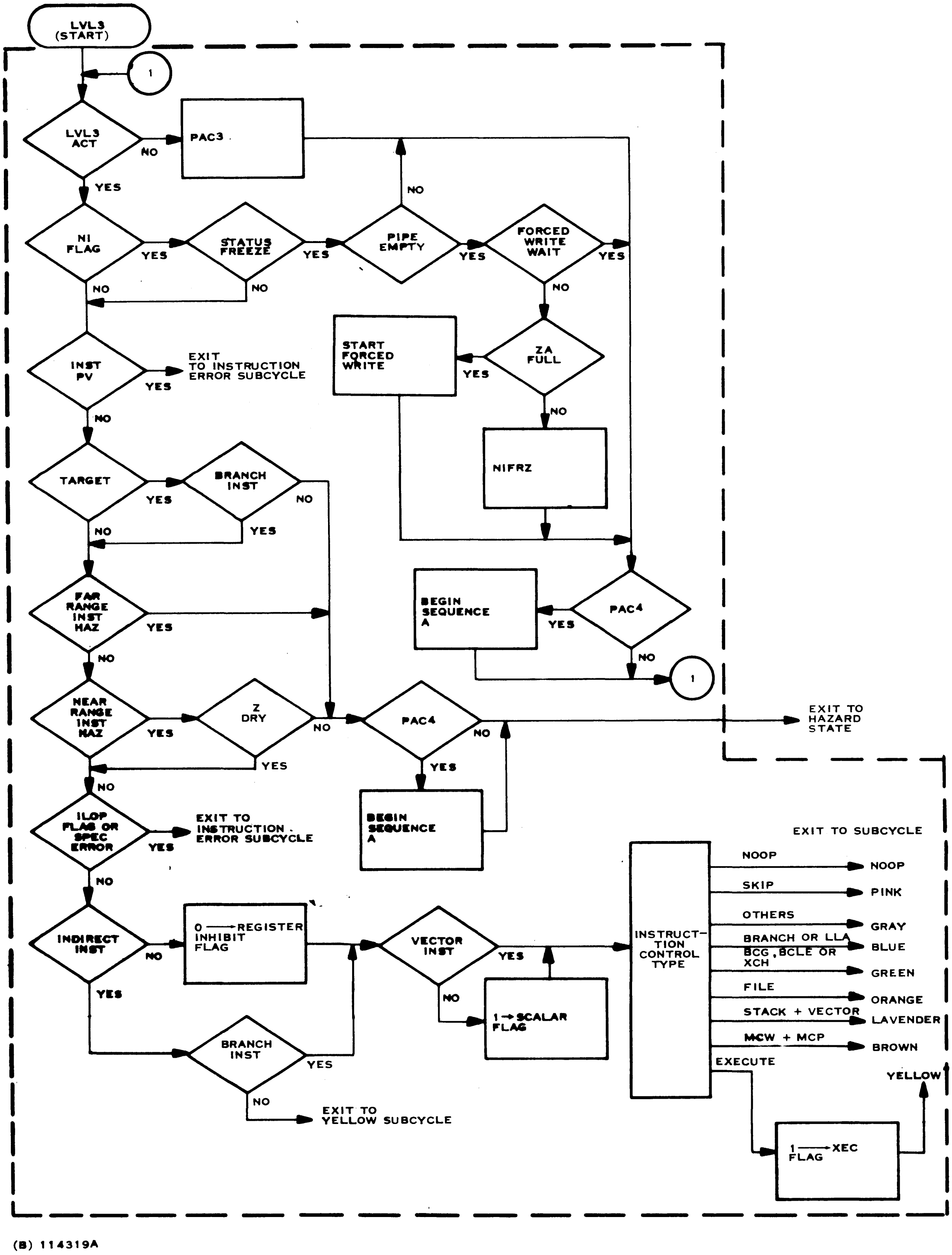


Figure 4-26. Initial Subcycle of Level 3 Idle State

If no instruction error occurs, the controller checks for a condition that requires the IPU to recover a previous instruction stream and abort the instructions that are currently in the pipe. These conditions are an LLA has prepared the pipe for a branch, but the target instruction does not branch when it reaches level 3 (Target and Not Branch Instruction), a far range instruction hazard, or a near range hazard when the Z buffers in the MBU contain information to be stored into memory. Any of these conditions causes the controller to exit to the Hazard state to produce the required signals for recovery from the fault condition.

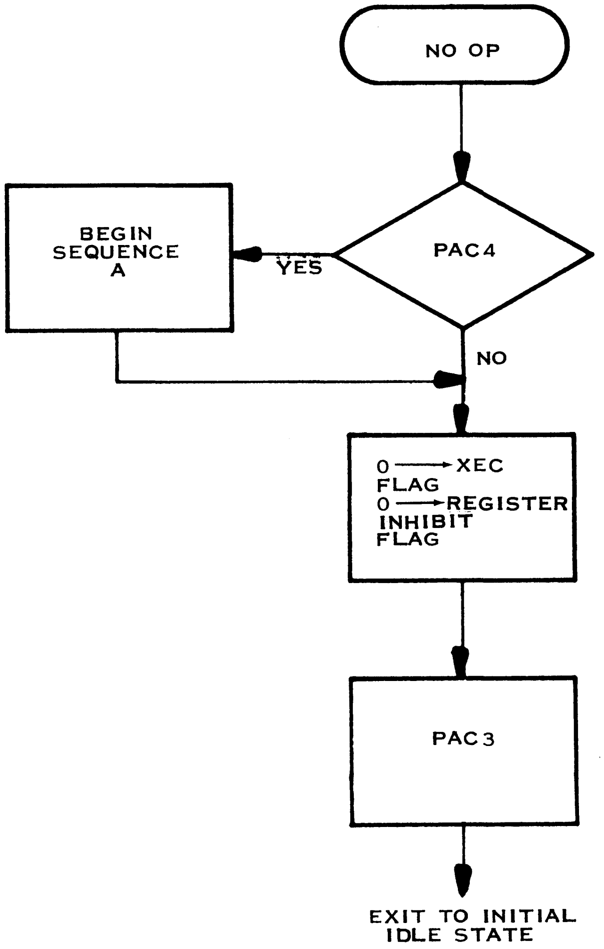
Having determined that no fault conditions exist at level 3, the controller inspects the instruction at level 3 and exits to the subcycle required to process that type of instruction. In performing this inspection the controller clears the Register Inhibit flag if the instruction is not an indirect instruction, and sets the Scalar flag if the instruction is not a vector. Clearing the Register Inhibit flag allows the next indirect instruction that enters level 3 to access the Register file for one level of indirect addressing. The controller then exits to the required subcycle.

4-247 NO OP. The controller enters the No Op subcycle when the instruction at level 3 is a no operation instruction. In this subcycle the controller clears the Execute flag and the Register Inhibit flag, and issues PAC3 to the level 2 controller before returning to the Initial Idle state for the next control cycle. Figure 4-27 illustrates the logic path that the controller follows through the No Op subcycle of the Idle state.

4-248 BROWN. The controller enters the Brown subcycle when the IPU executes a monitor call instruction. Either a monitor call and wait (MCW) or a monitor call and proceed (MCP) instruction causes the controller to enter the Brown subcycle. Brown is a transient subcycle in which the controller checks for PAC4 from level 4 controller before exiting to the Monitor Calls state (state 10). Figure 4-28 illustrates this decision path for the Brown subcycle. Sequence A is a no operation sequence.

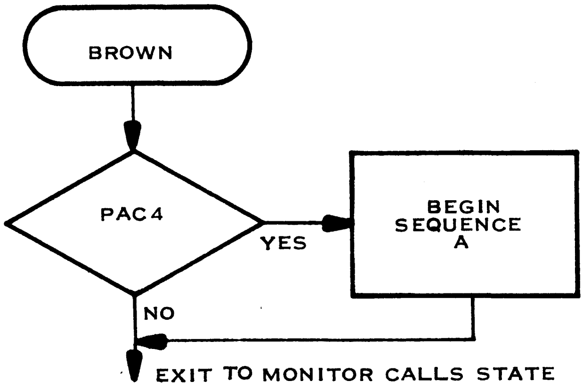
4-249 INSTRUCTION ERROR. The controller enters the Instruction Error subcycle when the Initial Idle state detects that a memory protect violation occurred when the instruction at level 3 was fetched from memory. In addition, if the instruction at level 3 is an illegal operation code, or contains format errors that will not allow the CP to execute the instruction properly, the controller enters the Instruction Error subcycle. In this cycle, the controller waits until the pipe empties into the Z buffer in the MBU. It then initiates a forced write operation to store those results into memory and signals the PP with either the PV or ILOP reason code bit to explain why the instruction is not executed. This empties the pipe of all successfully completed store instructions and terminates the program sequence until the PP or the user can rectify the error and restart the program. Figure 4-29 illustrates the decision paths contained in the Instruction Error subcycle.

4-250 YELLOW. The Yellow subcycle (figure 4-30) inspects conditions in the IPU to determine what transfers and gating signals should be enabled during indirect and execute instructions. The controller enters the Yellow subcycle from the Initial Idle state if it determines that the instruction is either an



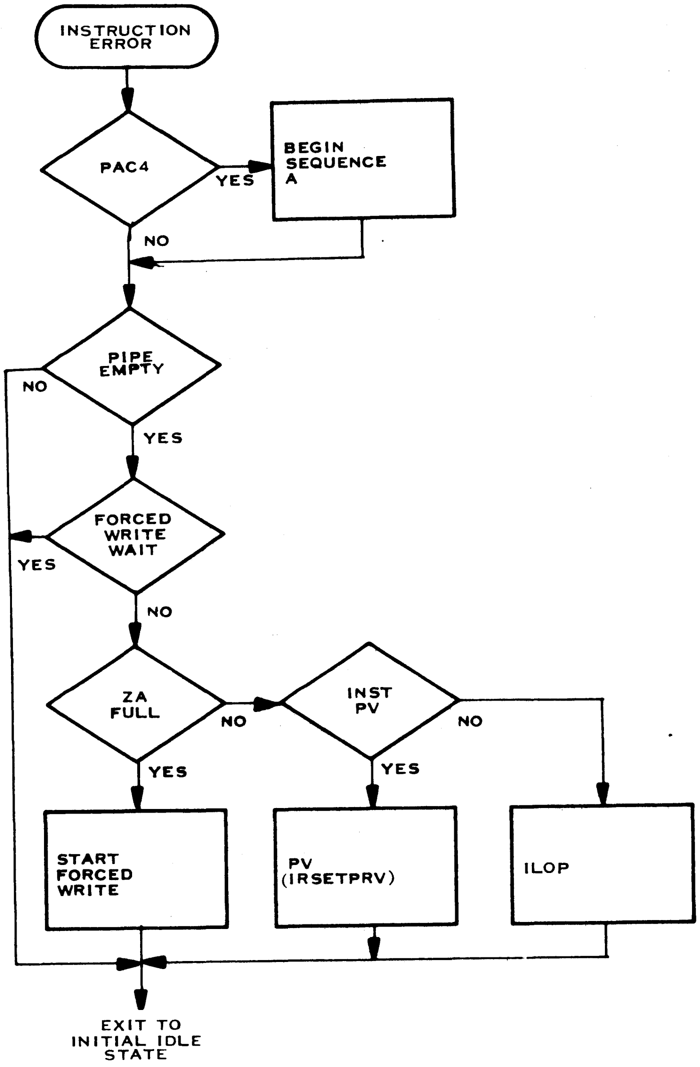
(A) 124858

Figure 4-27. No Op Subcycle of Level 3 Idle State



(A) 124859

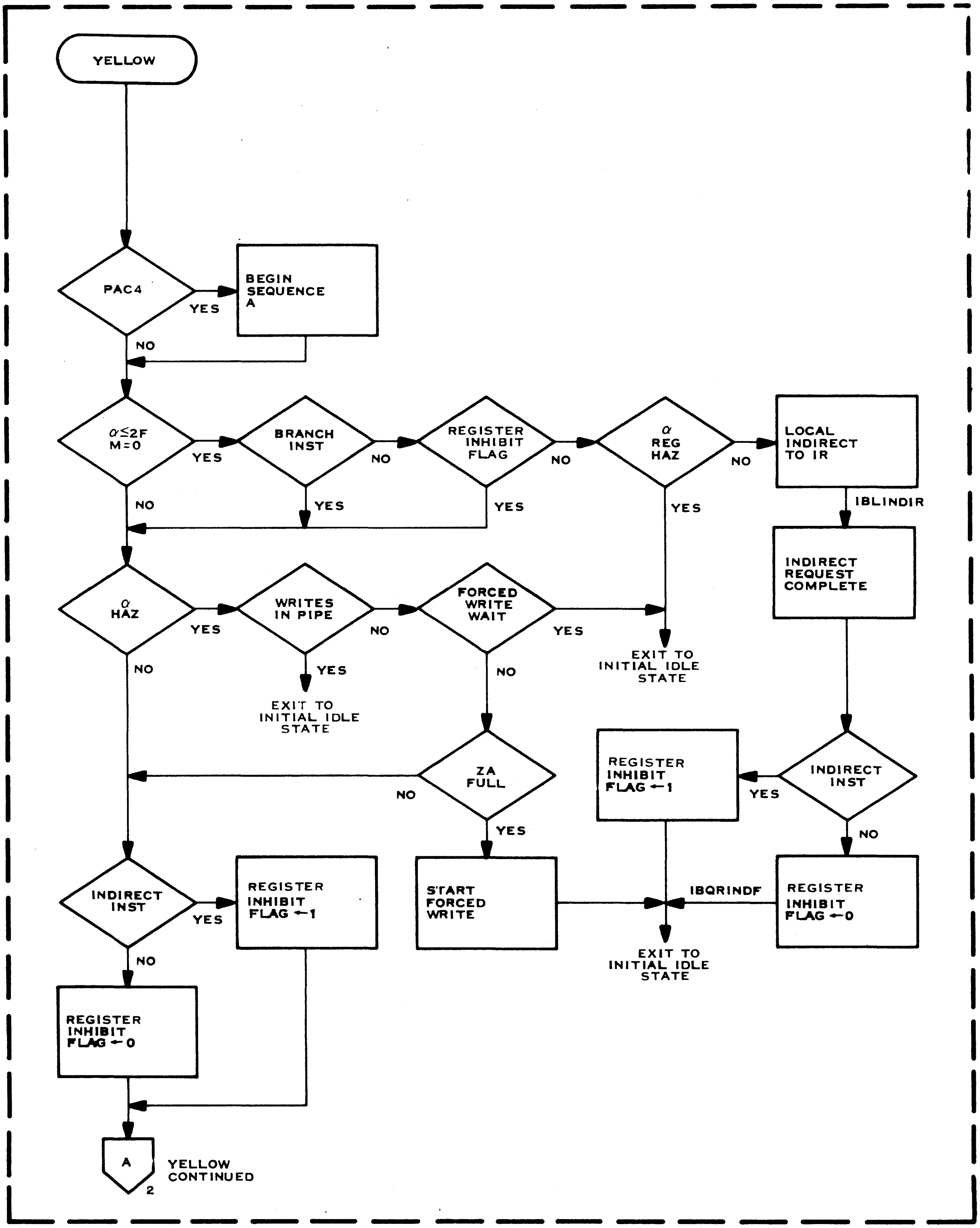
Figure 4-28. Brown Subcycle of Level 3 Idle State



(A) 124860

Figure 4-29. Instruction Error Subcycle of Level 3 Idle State

indirect instruction that is not a branch, or an execute instruction. The controller also enters the Yellow subcycle from the Blue subcycle if that subcycle detects an indirect instruction. In this subcycle the controller determines if the instruction references an address in the register file ($\alpha \leq 2F$, M=0). If it does, the controller ensures that the instruction is not a branch and that the Register Inhibit flag is not set. If both of these conditions are met and there is no alpha register hazard, the controller initiates a transfer from the register file to IR, signals that the indirect request is complete (for either an indirect or execute) and sets or clears the Register Inhibit flag depending upon the type of instruction. The Register Inhibit flag prevents accesses to the register file for more than one level of indirect addressing. The controller then exits to the Initial Idle state. If an alpha



(B) 114321A

Figure 4-30. Yellow Subcycle of Level 3 Idle State (Sheet 1 of 2)

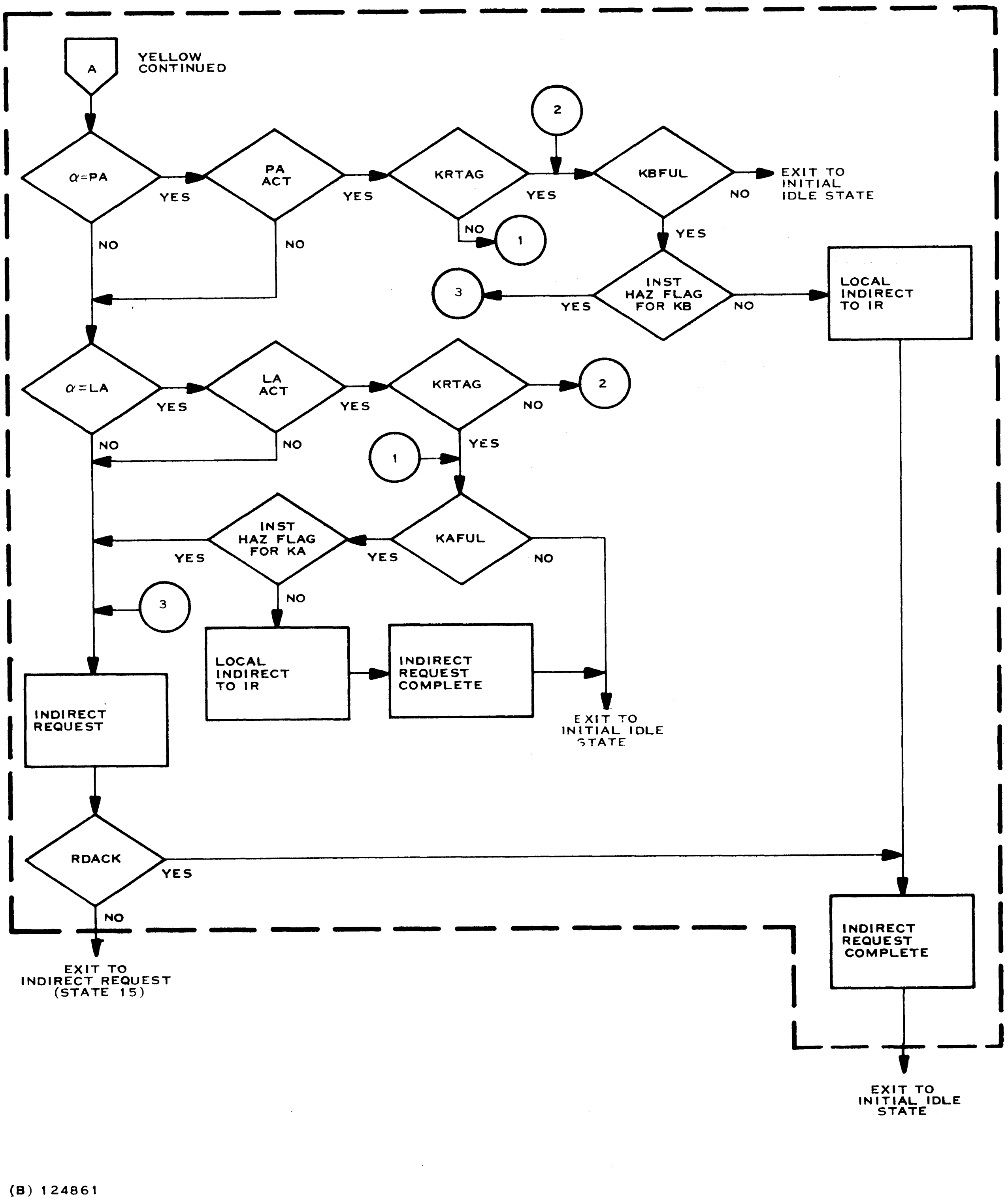


Figure 4-30. Yellow Subcycle of Level 3 Idle State (Sheet 2 of 2)

hazard was detected, the controller exits at that point to the Initial Idle state to wait for the hazard to clear before permitting access to the register file.

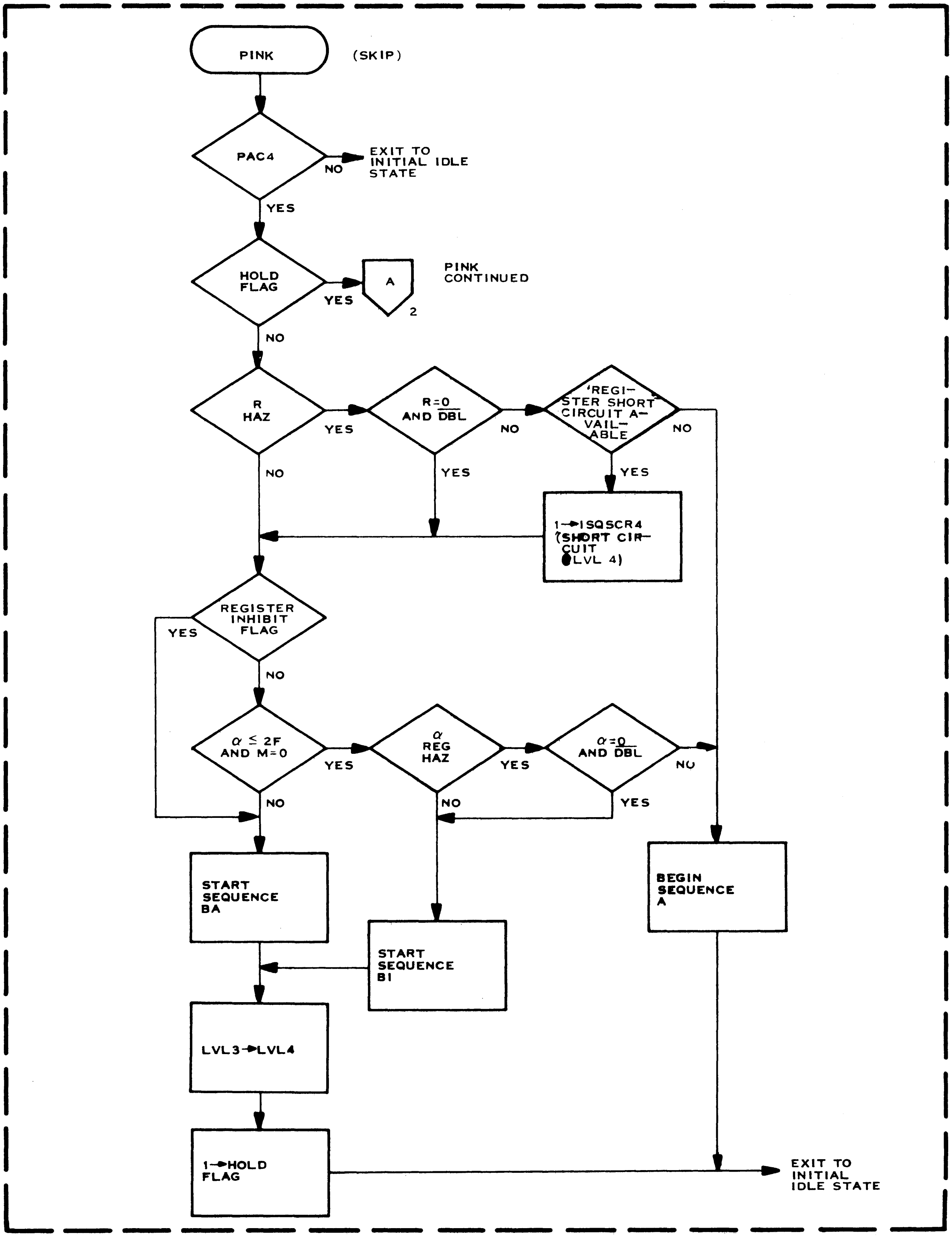
If the address referenced by the instruction is not in the register file, the controller checks for an alpha octet hazard. If a hazard exists and one or more instructions are still in the pipe that will be stored into memory, the controller exits to the Initial Idle state to wait for the hazard to clear. If no writes are in the pipe and a forced write is not in progress, the controller checks the contents of the ZA register. If the address in the IPU ZA register is an active address (indicating a valid octet of data in the MBU Z buffer), the controller initiates a forced write operation to transfer that octet to memory (refer to Forced Write controller), and returns to the Initial Idle state. If ZA is not full, the controller continues with the Yellow subcycle inspections.

If alpha is not in the register file and there is no alpha register hazard, the controller either sets or clears the Register Inhibit flag depending upon whether the instruction is an indirect or an execute, respectively. If the alpha octet address is equal to the address of the current octet and the current octet is valid, the controller performs a transfer from the current buffer (indicated by KRTAG) into IR as long as that buffer is full and no hazard exists for that buffer. If the buffer does not contain a valid octet, the controller returns to the Initial Idle state.

If alpha is not in the current octet but is in the look ahead octet, the controller ensures that the look-ahead octet is present in the IPU and then transfers the required instruction from that octet (indicated by KRTAG) to IR if no instruction hazard exists for the referenced buffer.

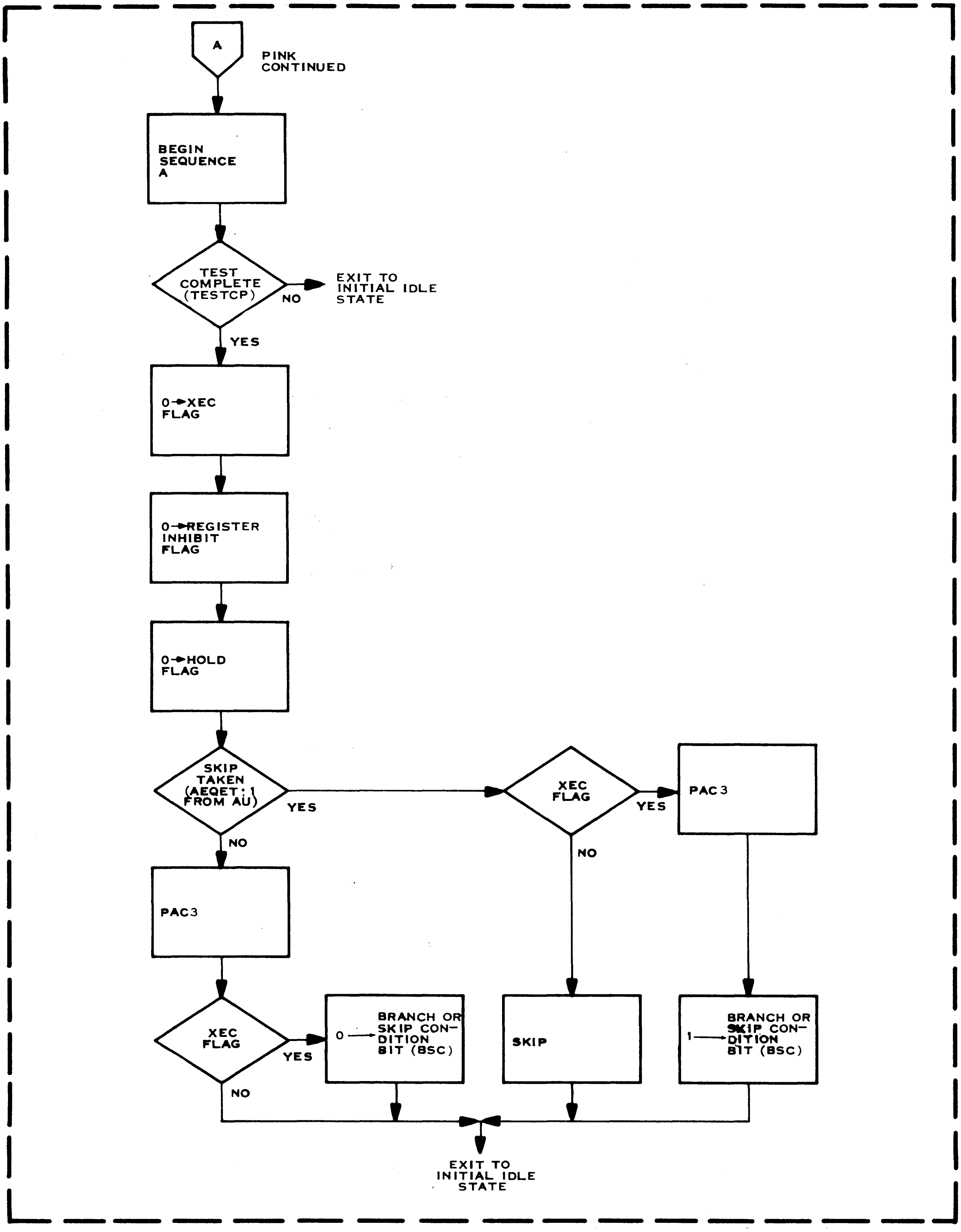
The controller initiates an indirect request to memory for the octet containing the required instruction if alpha is not in the register file, the current octet or in the look-ahead octet. If Central Memory Requester (CMR) is prepared to make the read request to memory, the control cycle is complete. The controller indicates that the indirect request is complete and returns to the Initial Idle state. If, however, CMR is not prepared to make the read request, the controller exits to the Indirect Request state to wait for CMR to issue the request for the required octet to memory.

4-251 PINK. The controller enters the Pink subcycle (figure 4-31) when it detects a skip condition at level 3 of the IPU. The controller performs two passes through the Pink subcycle to effect a skip of the next instruction. When the controller enters the Pink subcycle, the Hold flag is clear. The controller then checks for an R Hazard condition. If a hazard exists but the R-field of the instruction references register 00 of the register file and the object word is not a doubleword, the controller continues with the normal examination cycle. Register 00 of the register file is fixed at all zeroes and can therefore produce no hazards as long as the requested word is entirely within word 00. If the hazard is not for word 00, the controller determines if it can recover from the hazard by using the short circuit path in the AU so that a delay while waiting for the hazard to clear can be avoided. If the short circuit path can be used, the controller sets the Short Circuit at Level



(B) 114326A

Figure 4-31. Pink Subcycle of Level 3 Idle State (Sheet 1 of 2)


A
PINK CONTINUED
BEGIN SEQUENCE A
TEST COMPLETE (TESTCP)
NO
EXIT TO INITIAL IDLE STATE
YES
0→XEC FLAG
0→REGISTER INHIBIT FLAG
0→HOLD FLAG
SKIP TAKEN (AEQET: 1 FROM AU)
YES
NO
XEC FLAG
YES
NO
PAC3
PAC3
XEC FLAG
YES
NO
0 → BRANCH OR SKIP CONDITION BIT (BSC)
SKIP
1 → BRANCH OR SKIP CONDITION BIT (BSC)
EXIT TO INITIAL IDLE STATE

(B) 114327A

Figure 4-31. Pink Subcycle of Level 3 Idle State (Sheet 2 of 2)

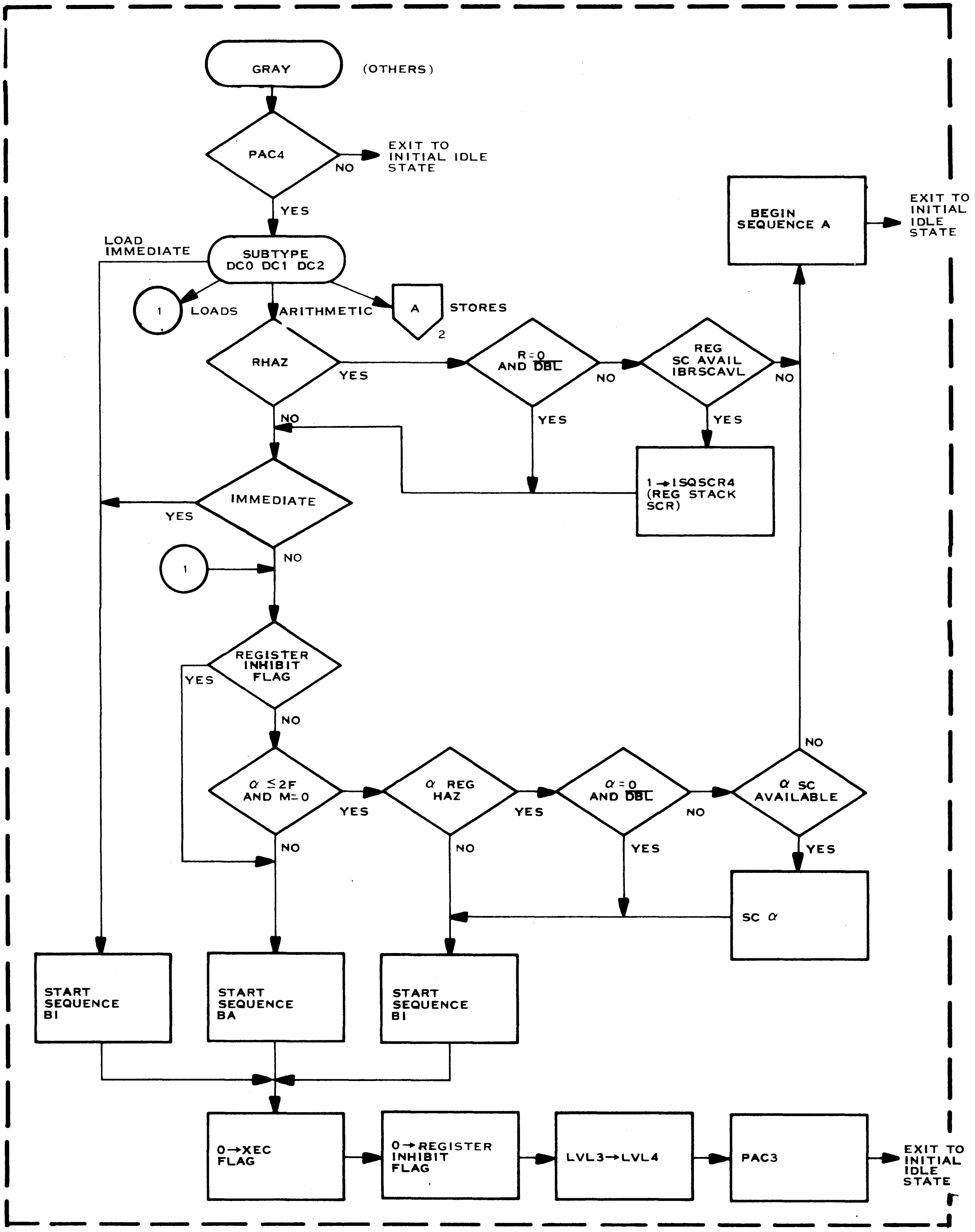
4 flag and continues with the normal examination cycle. If the short circuit path cannot be used, the controller returns to the Initial Idle state for the next clock cycle.

If no R hazard exists and the Register Inhibit flag is not set, the controller determines if the skip instruction references a value in the register file for skip determination comparison. If the value is in the register file and a hazard exists for the alpha operand, the controller determines if the hazard can be ignored due to a reference to register file word 00. If not, the controller returns to the Initial Idle state to wait for the hazard to clear. If no hazard exists or if that hazard can be ignored, the controller initiates sequence BI and enables the level 3 to level 4 transfer to load the test value into the pipe for processing, and sets the Hold flag before returning to the Initial Idle state. If the skip did not reference a register file address, the controller initiates sequence BA and enables a level 3 to level 4 transfer to load the test value into the pipe, and sets the Hold flag before returning to the Initial Idle state.

During the second pass through the Pink subcycle, the Hold flag is set. The controller then waits for the test to be completed to determine if the skip is to be executed. When the test is complete, the controller clears the Execute flag, the Register Inhibit flag and the Hold flag, and examines the indication from the AU to determine if the skip will be performed. If the AU indicates that the skip will be performed and the skip instruction was not the object instruction of an execute, the controller issues a SKIP signal to the other controllers so that the next instruction is bypassed. If the skip was part of an execute instruction, the controller does not perform the skip, but signals PAC3 to the level 2 controller and sets the Branch or Skip Condition bit (BSC) to indicate that the instruction referenced by the execute instruction was a branch or skip instruction (these instructions will not be performed when referenced by an execute instruction). If the AU indicates that the skip will not be taken, the controller issues PAC3 to the level 2 controller, and clears the BSC bit if the instruction was part of an execute operation. Regardless of the outcome of the AU test, the controller returns to the Initial Idle state after performing the specified operations.

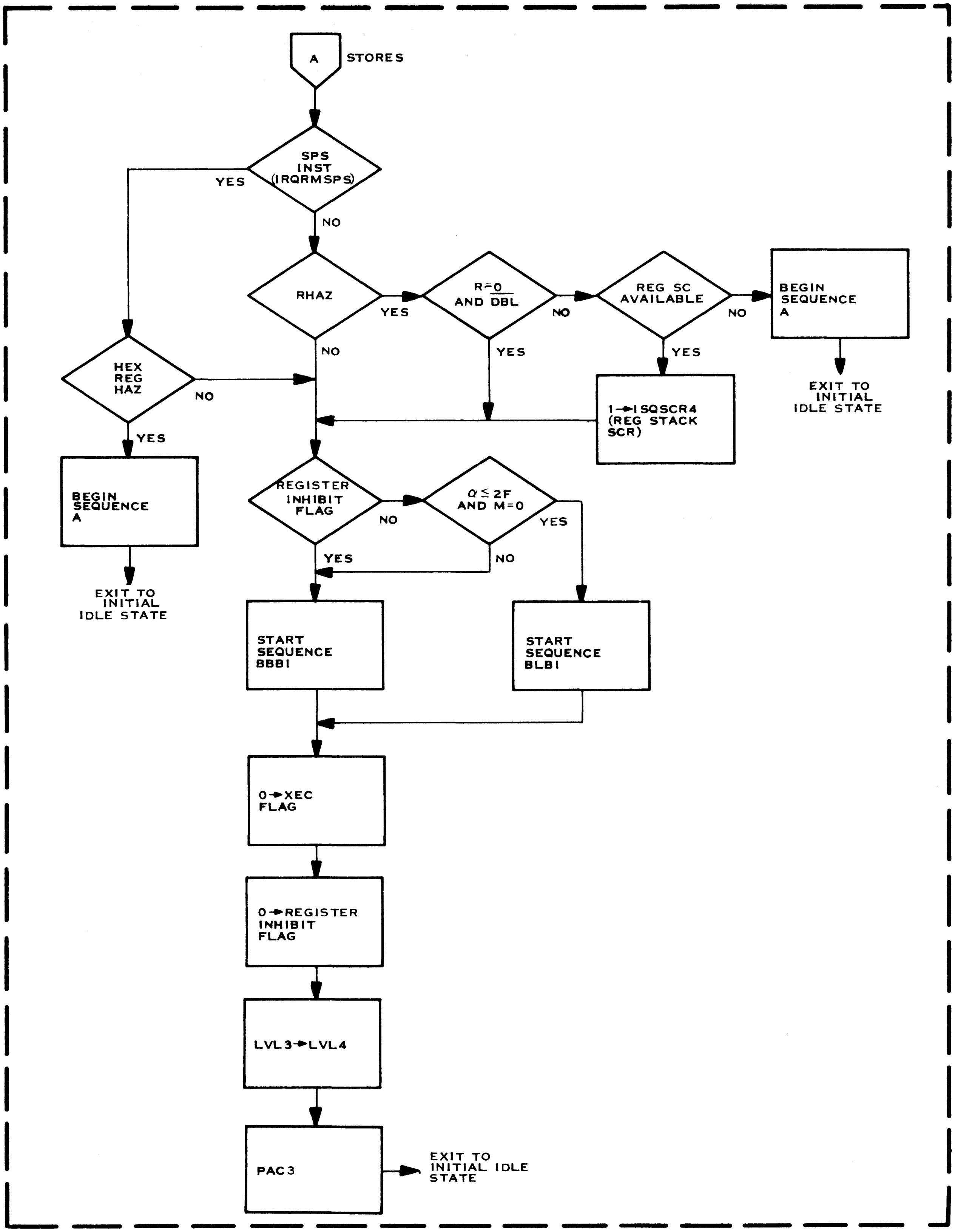
4-252 GRAY. The controller enters the Gray subcycle (figure 4-32) from the initial idle state when it ascertains that the instruction at level 3 is an immediate operand, a store operation, a conditional branch, an execute, a load operation, or an addition or subtraction command. In this subcycle the controller performs the inspections and transfers necessary to complete each of these operations at level 3. All operations within this subcycle are dependent upon the receipt of PAC4 from the level 4 controller. Until this indication is received, the controller loops between the Initial Idle state and the Gray subcycle.

If the instruction at level 3 is a load immediate instruction, the controller loads the immediate value into the pipe, (sequence BI and LVL3 → LVL4), clears the Execute flag and the Registe Inhibit flag, and signals PAC3 to the level 2 controller before returning to the Initial Idle state.



(B) 114323A

Figure 4-32. Gray Subcycle of Level 3 Idle State (Sheet 1 of 2)



(B) 114324A

Figure 4-32. Gray Subcycle of Level 3 Idle State (Sheet 2 of 2)

If the instruction at level 3 is a load operation and the Register Inhibit flag is not set, the controller determines if the load operation references a location in the register file to supply the quantity to be loaded. An access outside the register file causes the controller to initiate sequence BA to retrieve the object octet from memory and load it into the pipe through the MBU memory interface. If the quantity is in the register file, the controller determines if there is an instruction in the pipe that will change that quantity (alpha register hazard). If a hazard exists and the instruction does not reference word 00 of the register file, the controller checks to see if it can avoid a hazard delay by using the short circuit path in the AU to recover from the hazard. If not, the controller exits to the Initial Idle state to wait for the hazard to clear. If a short circuit path is available, the controller signals that the short circuit path should be used. The controller then initiates sequence BI to transfer the desired quantity into the pipe.

Regardless of the path taken during the load cycle inspection (unless an alpha register hazard created a controller loop cycle), the controller concludes the examination cycle by clearing the Execute flag and Register Inhibit flag, enabling the transfer of level 3 to level 4, and transmitting PAC3 to the level 2 controller before returning to the Initial Idle state.

If the instruction at level 3 is an arithmetic instruction or an arithmetic immediate instruction, the controller examines the hazard detection circuits for an R-field hazard. If a hazard is indicated, but the R-field references word 00 of the register file and the addressed word is not a doubleword, the controller ignores the hazard (Word 00 is fixed zeroes and cannot have a halfword or singleword hazard associated with it). If the hazard cannot be ignored, the controller determines if it can use the short circuit path in the AU to recover from the hazard and avoid the hazard delay time. If the short circuit path is available, the controller sets the Short Circuit at Level 4 flag. If not, the controller loops between the Initial Idle state and the Gray subcycle until the hazard clears. After the R-field hazard determination has been performed, the controller separates the immediate instructions from the arithmetic instructions. The arithmetic instructions follow the control path described for load operations above; the immediate operand follows the load immediate cycle described above.

For store operations at level 3 the controller checks for R-field hazards using a control cycle similar to addition, subtraction, or immediates unless the store operation is a store program status instruction. In that case the controller determines if any instruction is already in the pipe that will produce a change in the compare code, arithmetic exception, or result code bits, or if a Load Arithmetic Mask instruction is in the pipe. These conditions will change a portion of the program status doubleword and are termed "hex register hazard" conditions. If a hex register hazard exists, the controller loops between the Initial Idle state and the Gray subcycle until the hazard clears the pipe and has modified the cell in the status doubleword. If no hex register hazard exists, the controller continues with the store inspection cycle.

Having determined that no hex register hazards affect the store operation, the controller determines whether the store is intended for the register file. Two conditions must be satisfied to make this determination: the Register Inhibit flag must be clear, and the alpha address and M-field must indicate that the store is intended for the register fiel. If these conditions are true, the controller initiates sequence BLBI to store the specified material into the register file. If the store operation is not for the register file, the controller initiates sequence BBBI to store the specified material into central memory. Regardless of the storage destination of the material, the controller completes the control cycle by clearing the Execute and Register Inhibit flags, enabling the transfer of level 3 to level 4, and transmitting PAC3 to level 2 controller before exiting to the Initial Idle state.

4-253 ORANGE. The controller enters the Orange subcycle (figure 4-33) when it determines that a load or store file operation is in level 3 of the IPU. This determination is performed only in the Initial Idle state of the controller. The Orange subcycle divides the file instructions into multiple (those operations involving more than one register file octet), and single (those operations that load or store only one octet in the register file).

If the instruction is a load or store multiple, the controller ensures that all operations that store results into the register file (or the memory location used to load the file) have cleared the pipe. If the pipe is active or a forced write operation is in progress, the controller loops to the Initial Idle state to wait for the pipe to clear. If neither of these conditions exist but a valid octet of data is in the Z buffer in the MBU (ZA Full), the controller initiates a forced write operation to store that octet into memory before continuing with the control cycle. When the controller determines that the pipe is clear of all operations, it loads a value of "5" into the completion counter. During load file multiple operations in the Orange Request state, the controller checks the completion counter after a request has been made for an octet from memory. If the counter is not equal to zero, the controller decrements the counter and continues to make requests for the remaining octets. This procedure results in six octet requests to memory before the operation terminates.

For a store operation, the controller determines if the store will affect the octet contained in the MBU X or Y buffers by comparing the alpha address of the store instruction with the address in the XA and YA registers in the IPU. If either of the comparisons is true, the controller clears the activity bit associated with that address register so that succeeding instructions in the IPU will not draw operands from the unchanged values in the X or Y buffers. The controller then issues a store file request to CMR, and loads the request counter with a value of "4". In the Orange Request state the controller checks the contents of the request counter after each store file request is made to terminate the operation after six file octets have been stored into memory. If CMR is prepared to perform a write into memory on the next clock (WACK), the controller sets the AR Increment flag so that the next clock pulse will add eight to the address in the AR register (alpha address). The controller then exits to the Orange Wait state for one clock cycle and then to the Orange Request state. If CMR was not ready to perform a write request due to outstanding read requests to central memory, the controller loops between the Initial Idle state and the Orange subcycle until all requests have been honored.

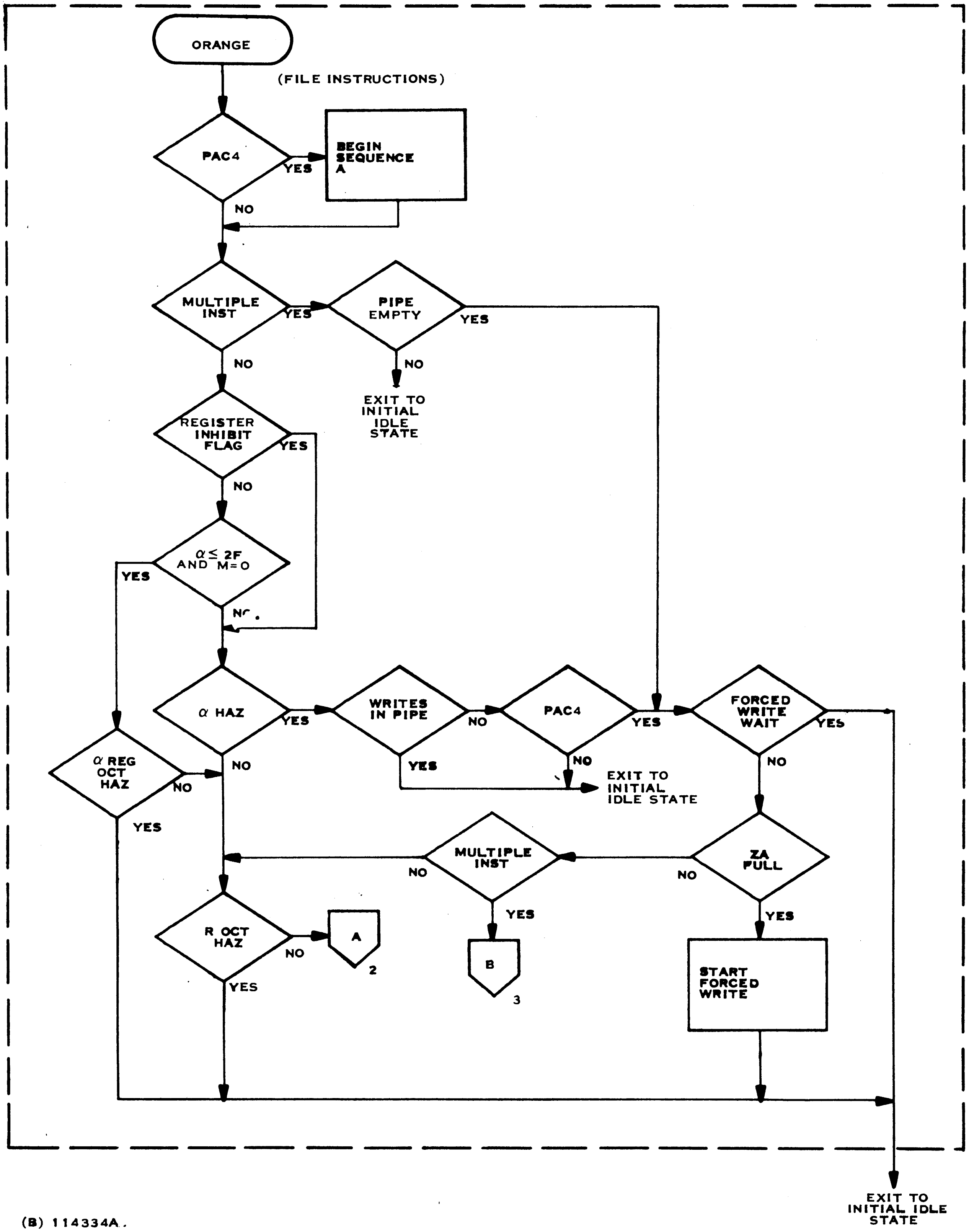
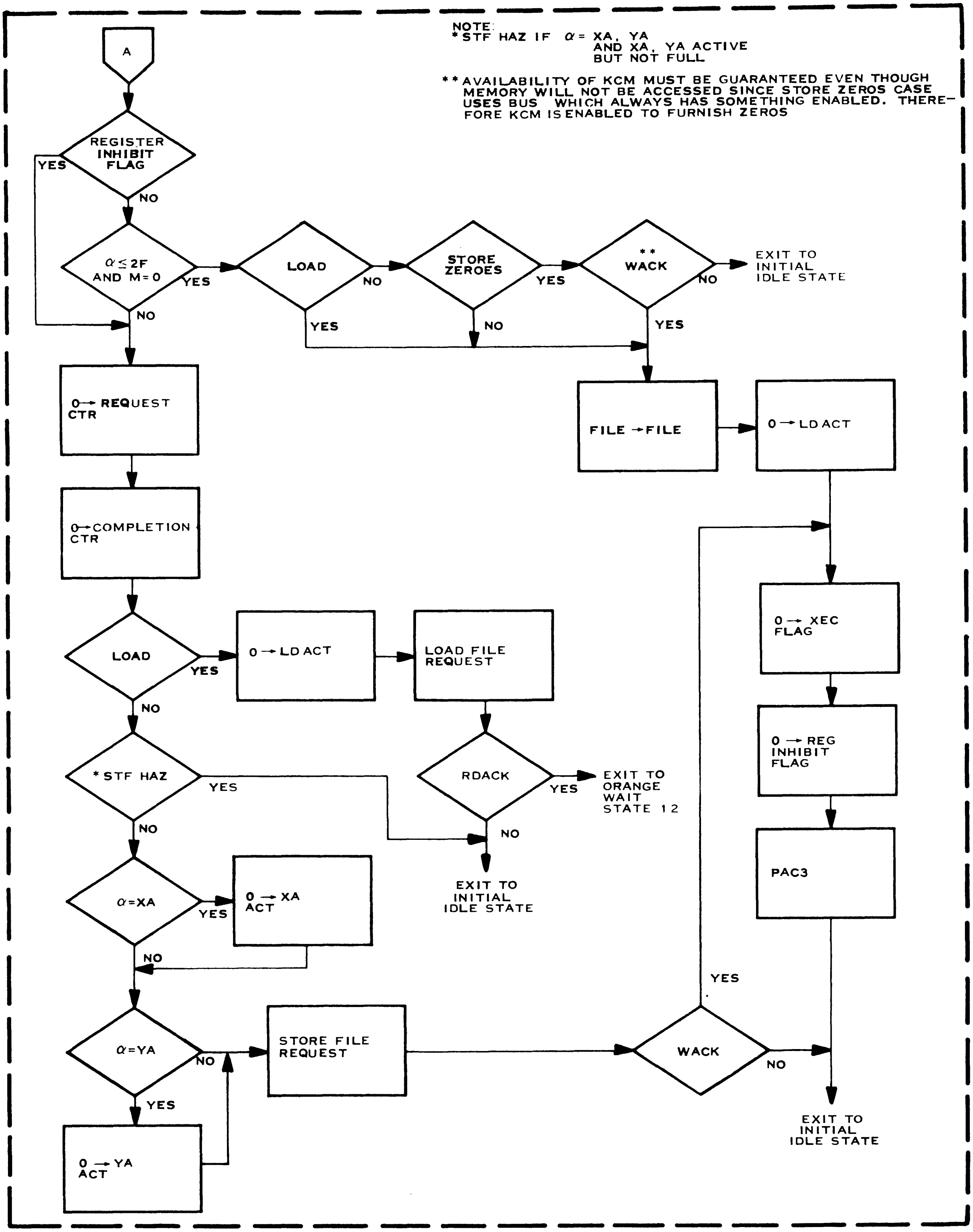


Figure 4-33. Orange Subcycle of Level 3 Idle State (Sheet 1 of 3)

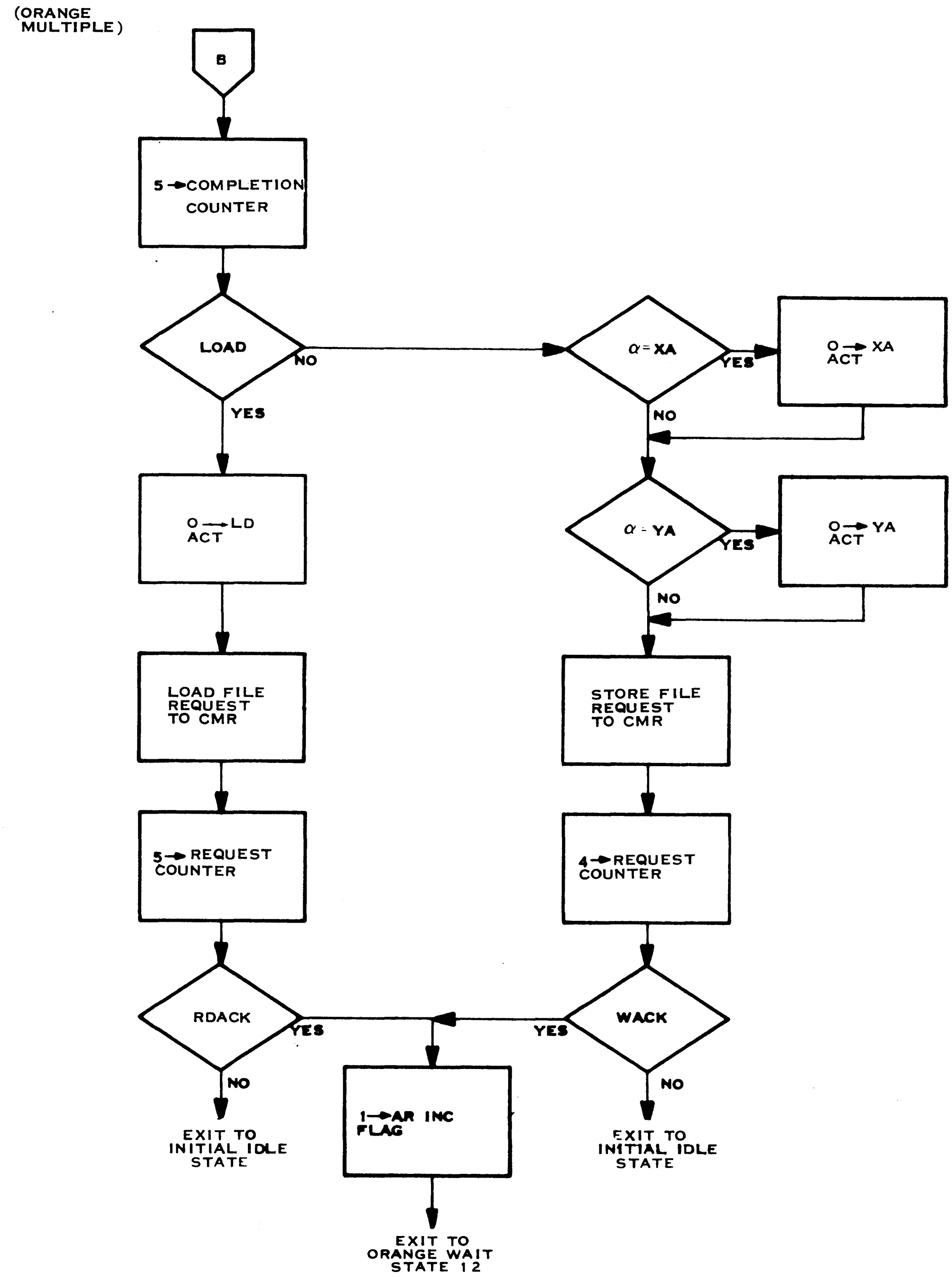


NOTE:
*STF HAZ IF α = XA, YA
AND XA, YA ACTIVE
BUT NOT FULL

**AVAILABILITY OF KCM MUST BE GUARANTEED EVEN THOUGH MEMORY WILL NOT BE ACCESSED SINCE STORE ZEROS CASE USES BUS WHICH ALWAYS HAS SOMETHING ENABLED. THEREFORE KCM IS ENABLED TO FURNISH ZEROS

(B) 114335A

Figure 4-33. Orange Subcycle of Level 3 Idle State (Sheet 2 of 3)



(B) 124862

Figure 4-33. Orange Subcycle of Level 3 Idle State (Sheet 3 of 3)

During Load File Multiple instructions, the controller clears the LD register activity bit. The LD (Last Destination) register contains the register file destination address of the last instruction sent down the pipe. Since the contents of the register file are being changed by the load file multiple instruction, this address is no longer useful. Clearing the LD activity bit, therefore, indicates that the address in LD is not a valid last destination address. The controller then issues a load file request to CMR, sets the request counter to a value of "5", and if CMR is prepared to process a read memory request (RDACK), sets the AR Increment flag. Setting this flag results in adding eight to the address in the AR register during the next clock cycle so that the address of the next file octet will be in the AR register. The request counter tracks the number of requests yet to be made to memory in the Orange Request state of the controller.

For single file instructions the controller determines if the instruction references an address in the register file. If alpha is in the register file, the controller checks to see that no operation in the pipe will change the values of the alpha address octet (alpha register octet hazard), and also that the octet in the register file specified by the R-field will not be changed by an operation in the pipe (R Octet hazard). The controller then enables a "file to file" transfer, clears the LD register activity bit, clears the Execute and Register Inhibit flags, and issues PAC3 to the level 2 controller before exiting to the Initial Idle state for the next control clock. If the operation is a store zeroes, the memory bus must be clear of all data (WACK) so that KCM can supply an input of all zeroes to the register file octet indicated by the alpha address.

If a single file instruction references an address outside of the register file, or the register inhibit flag is set, the controller determines if any operation in the pipe will affect the memory location designated by the alpha address (alpha hazard). If such a hazard exists, the controller loops between the initial idle state and the Orange subcycle until a forced write operation has emptied the pipe and stored the results in memory (ZA not full). Once the controller has determined that either no alpha hazard exists or that the hazard has cleared, it ensures that no hazard exists to the memory location specified by the R-field. The controller then loads a value of zero into the request counter and into the completion counter to prevent more than one octet from being affected by the operation. If the operation is a load, the controller clears the LD register activity bit, issues a load file request to CMR, and if CMR can process that request the controller exits to the Orange Wait state and then to the Orange Request state.

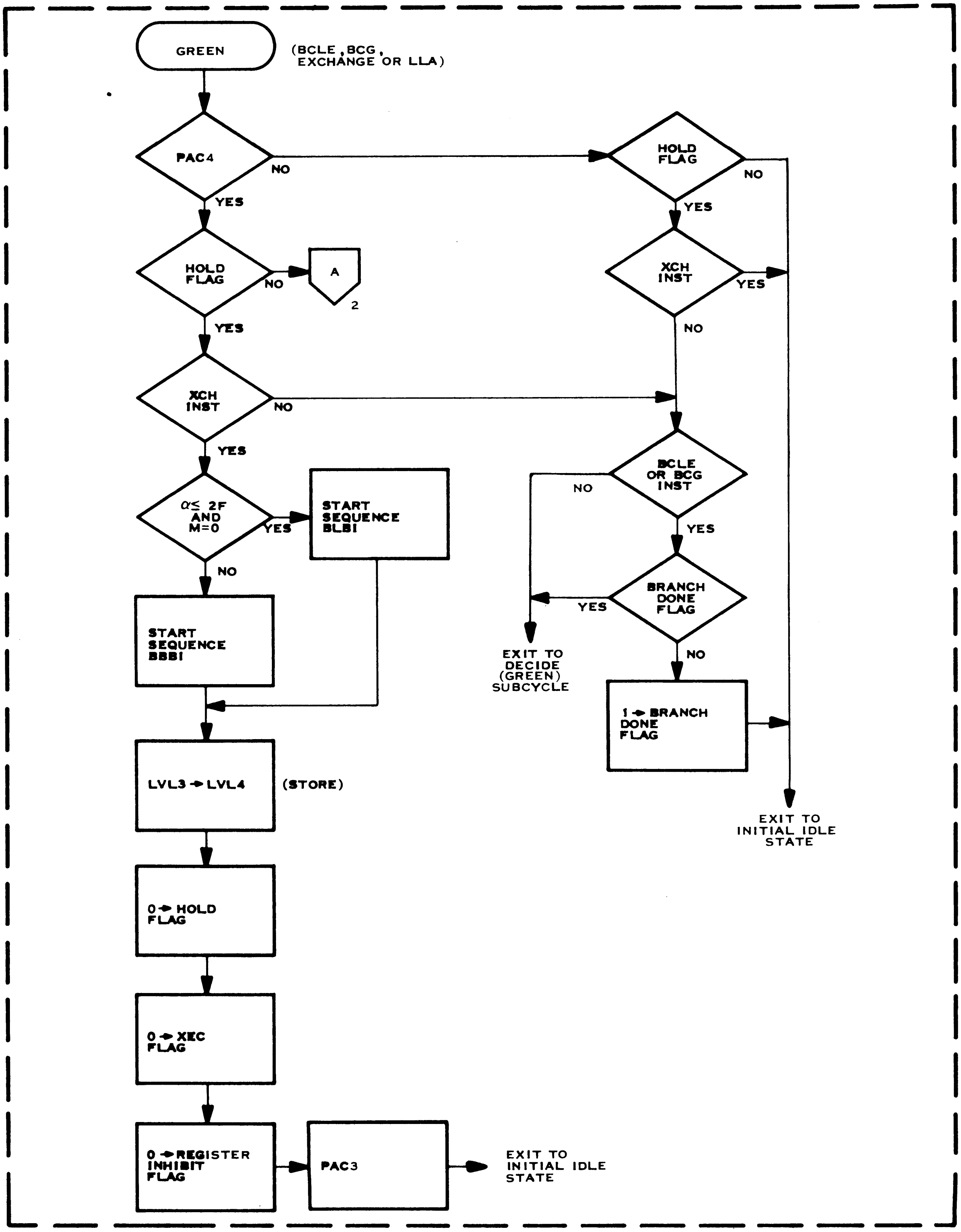
If the operation is a store file, the controller checks for a store file hazard. A store file hazard exists if a previous instruction caused the MBU to order an octet from memory (XA or YA active), that octet has not yet entered the MBU from memory (XA or YA not full), and the alpha address of the store file instruction references that octet for the store file destination. Under this condition the store file operation will write on top of the desired information in memory before it can be fetched for the previous instruction. To avoid destruction of the needed information, the controller loops between the Initial Idle state and the Orange subcycle until the hazard clears. When the hazard clears, the controller issues a store file request to CMR. If alpha references the address in either XA or YA, the controller also clears the respective register activity bit to indicate that the address no longer represents the memory location of the octet in the MBU. If CMR is not prepared to perform the write operation to memory (not WACK), the controller loops between the Initial Idle state and the Orange subcycle until WACK becomes active. At that time the controller clears the Execute and Register Inhibit flags and issues PAC3 to the level 2 controller before returning to the Initial Idle state for the next control cycle.

4-254 GREEN. The Green subcycle processes exchange instructions and conditional branches that use the IPU hardware to determine the branch condition (BCLE and BCG). The controller enters this subcycle from the Initial Idle state when the instruction at level 3 is determined to be within the described categories. Figure 4-34 illustrates the decision paths available within the Green subcycle. The controller may take one of two paths through the subcycle depending upon the type of instruction at level 3 of the IPU.

The two conditional branch instructions pass through the Green subcycle three times to complete the test and branch operation. During the first control cycle, the controller enters Green and the Hold flag is not set. This flag is set in the Green subcycle to track the number of times that the controller has passed through the subcycle. Since Hold is clear, the controller checks for hazards to the registers specified by both the R and T fields of the instruction. If any instruction in the pipe will alter the contents of the register specified by either of these quantities, the controller loops between the Initial Idle state and Green until all hazards have cleared. The controller then initiates sequence BI to transfer the test parameters from the areas specified in the instruction fields into the level 4 registers (R0 and A0) for the branch determination. The controller enables the level 3 to level 4 transfer and sets the Hold flag before returning to the Initial Idle state for the start of the second control cycle.

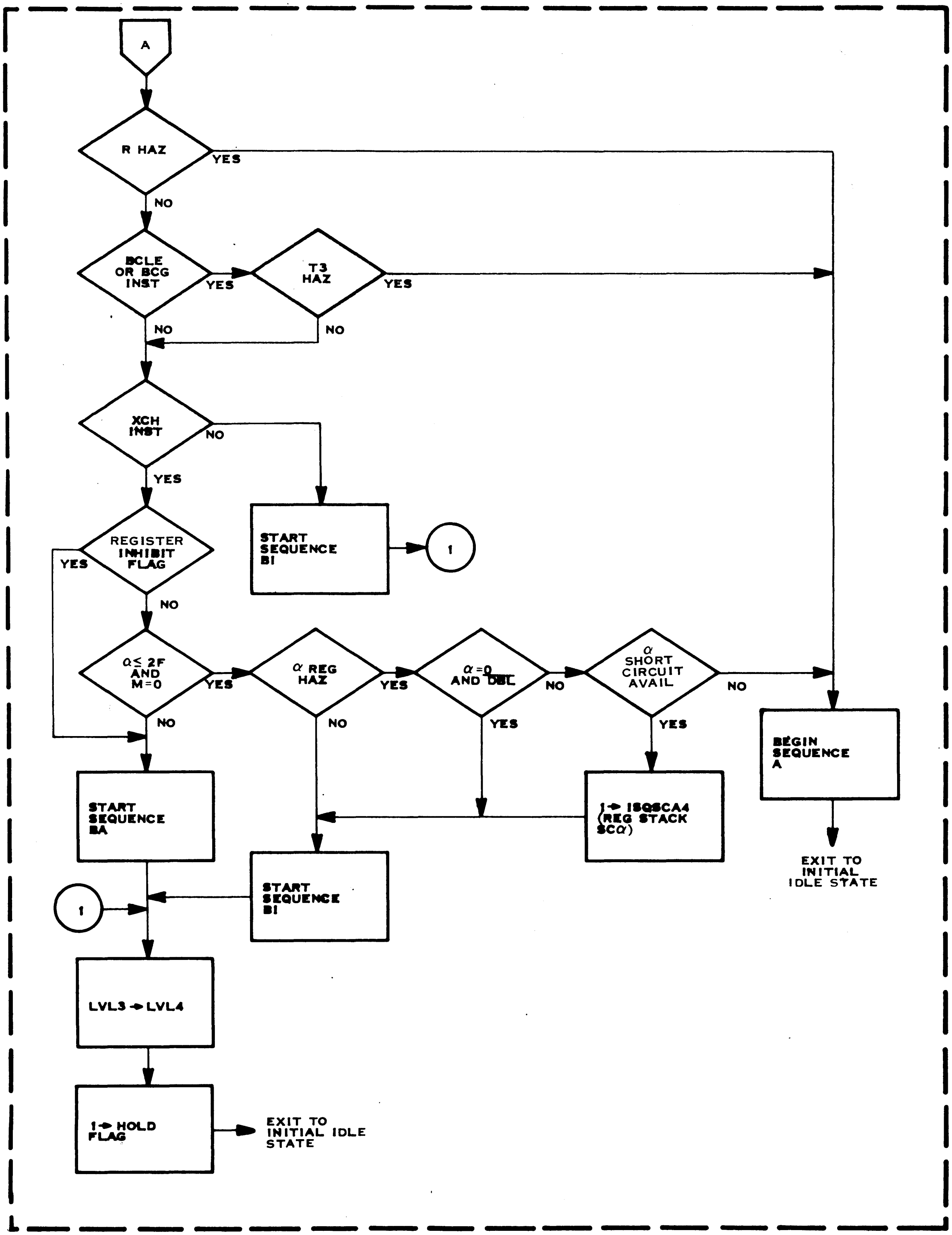
The second control cycle during BCG or BCLE instructions allows the IPU time to add the test parameters to decide whether to branch. The count value is contained in the R0 register, the index value is in the most significant half (32 bits) of A0, and the comparison value is in the least significant half of A0. The IPU adds R0 to the most significant half of A0 and the one's complement of the least significant half of A0. The carry out bit of this operation determines if the branch is to be taken. The second cycle allows time for this operation by checking the Branch Done flag. This flag is not set, so the controller sets that flag and returns to the Initial Idle state for the start of the next control cycle.

During the third control cycle, both the Hold flag and the Branch Done flag are set. The controller exits to the Decide (Green) subcycle to determine the results of the branch comparison, and either take or disregard the branch.



(B) 114332A

Figure 4-34. Green Subcycle of Level 3 Idle State (Sheet 1 of 2)



(B) 114333A

Figure 4-34. Green Subcycle of Level 3 Idle State (Sheet 2 of 2)

An exchange instruction interchanges the contents of a register file location (specified by the R-field) with the contents of a memory or register file location specified by the effective address, alpha. The controller passes through Green twice when processing an exchange instruction. The first pass accesses a memory location and sends the contents into the pipe for storage in the register file location designated by the R-field. The second pass, one clock later, draws the present contents from the register file location and sends them down the pipe for storage in the memory location vacated by the original value. Although having a store destined for the register file in the pipe at the same time that the register file location is stored into memory constitutes a hazard, the Green subcycle ignores that hazard condition so that the exchange may be accomplished in two clock cycles.

During the first pass through the Green subcycle, the Hold flag is not set. The controller then checks for a hazard to the register file location specified in the R-field. If a previous instruction in the pipe will change the contents of that location, the controller loops between the Initial Idle state and the Green subcycle until that instruction clears the pipe and the result is stored into the register file. When the hazard has cleared, the controller checks the Register Inhibit flag. If this flag is not set, the location specified by the alpha address may also be in the register file. If alpha is less than or equal to 2F and the M-field is zero, the alpha address specifies a register file location. In that case the controller determines if a previous instruction in the pipe will alter the contents of the referenced register. If no hazard exists, or if the alpha address references address 00 in a singleword or halfword transfer, the controller initiates sequence BI to load the value from the register file into the pipe to be stored in the designated register file location. If a hazard does exist, however, the controller determines if a short circuit path is available for recovery from the hazard. For a short circuit the controller sets the "short circuit at level 4" flag (ISQSCA4) and initiates sequence BI. The unchanged value will be transferred from the register file and started down the pipe. However, when the value reaches level 7 (input to the AU), the AU short circuit path will change the data to the correct value before it is stored into the register file. If no recovery is possible from the hazard, the controller loops between the Initial Idle state and the Green subcycle until the hazard clears the pipe and has been stored into the register file. If the alpha address is not in the register file, the controller instead initiates sequence BA to fetch the required data from a memory location and start it down the pipe to be stored into the register file. Regardless of the source of the data to be stored into the register file, the controller completes the control cycle by enabling the level 3 to level 4 transfer and setting the hold flag to designate that the first cycle has been completed. The controller returns to the Initial Idle state for the start of the next control cycle.

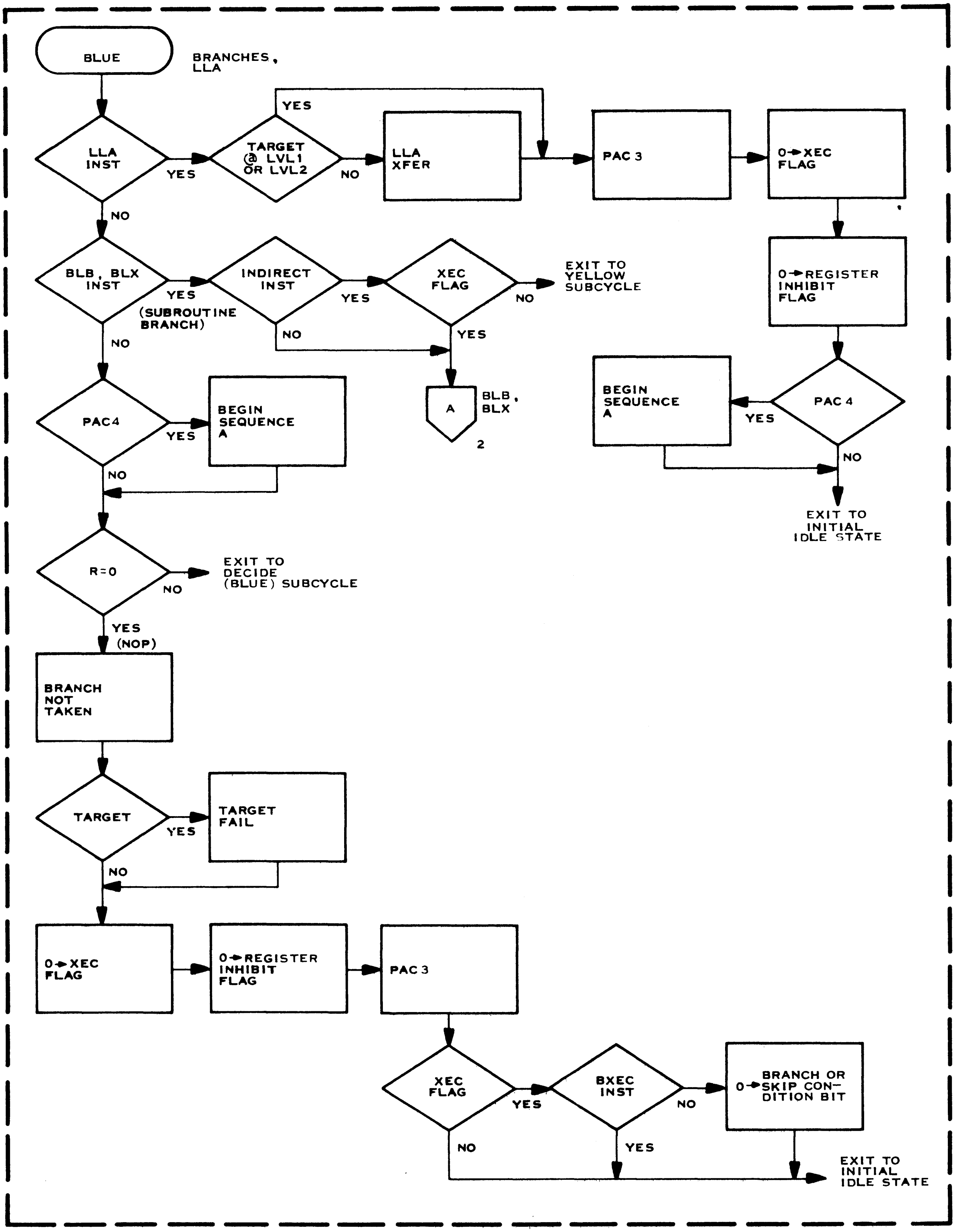
During the second pass through the Green subcycle, the Hold flag is set. If the alpha address (destination of the store portion of the exchange) is in the register file, the controller initiates sequence BLBI to fetch the word from the location in the register file specified by the R-field and transfer it into the pipe for storage into the register file location designated by alpha. If alpha designates a memory location, the controller initiates sequence BBBI to start the register file data into the pipe for storage into memory. In either case the controller enables the level 3 to level 4 transfer, initializes the controller by clearing the Hold, Execute and Register Inhibit flags, and transmits PAC3 to the level 2 controller before returning to the Initial Idle state for the next control cycle.

4-255 BLUE. The level 3 controller enters the Blue subcycle from the Initial Idle state when it determines that the instruction at level 3 is either a load look ahead (LLA) or a branch other than a conditional branch that can be decided in the IPU (BCLE or BCG - see Green). The Blue subcycle generates required gating and flag signals. The controller may then exit to either the Initial Idle state, to the Decide subcycle for conditional branches, or to the Yellow subcycle for indirect branches. Figure 4-35 illustrates the decision paths available within the Blue subcycle. The paths may be divided into three categories depending upon the instruction type: an LLA, a branch and store P3 in the register file (BLB or BLX), or some other type of branch instruction.

If the instruction at level 3 is an LLA, the controller determines if the branch instruction referenced by the LLA (target) is in level 1 or level 2 of the IPU. This determination is made by comparing the instruction count (difference of instruction sequence numbers between LLA and branch) portion of the LLA instruction with the number of active levels in the IPU. If the instruction count is "2", for example and both level 1 and level 2 are active, then the instruction in level 1 is the target. If the target is in the pipe, the level 3 controller does not issue an LLA Transfer signal to enable the look ahead controller. If the target is not in the pipe, the level 3 controller enables the load look ahead controller to transfer the address in P3 into BA for reference when the target does enter the pipe. In either case, the controller issues PAC3 to the level 2 controller to indicate that it is ready for the next instruction, and clears the Execute and Register Inhibit flags. The controller returns to the Initial Idle state for the start of the next control cycle.

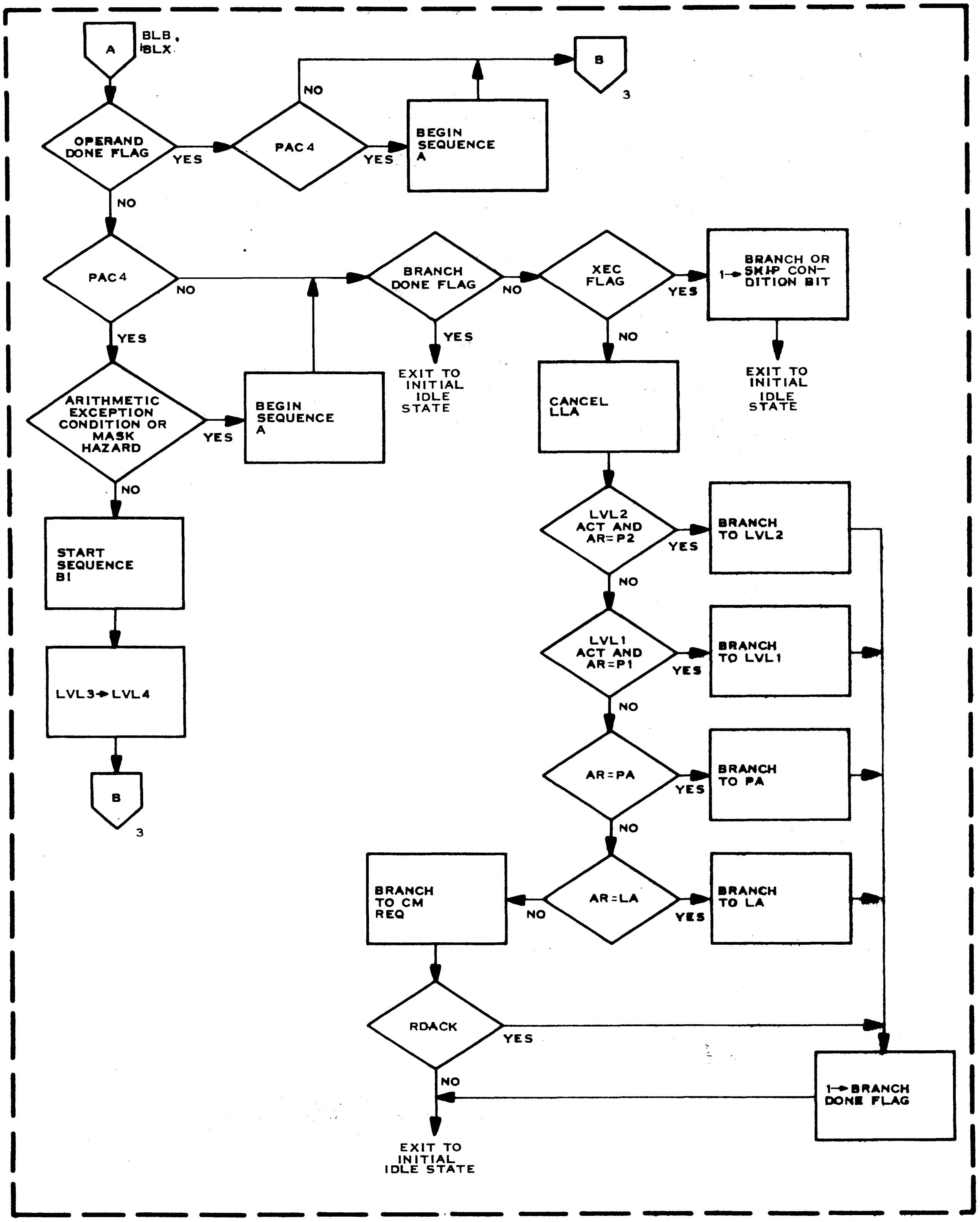
If the instruction at level 3 is an unconditional branch (either a BLB or a BLX), the controller determines if the instruction is an indirect instruction. If it is an indirect instruction and not a part of an Execute instruction, the indirect portion of the instruction must be satisfied before further processing of the branch can be performed. The controller therefore exits to the Yellow subcycle to process the indirect addressing involved in the instruction. If indirect addressing is not involved in the instruction, the controller continues with the BLB, BLX instruction path.

Two flags control the completion of the BLB, BLX decision path. The Operand Done flag indicates that the P3 address is being modified in the AU to a value of P3+1 and will be stored into the base or index register specified by the R-field. This value is used for return address at the end of the subroutine. The process also stores the AE mask and condition bits in the same register file location to reinstate these status conditions when the program returns from the subroutine. The Branch Done flag indicates that the source of the b branch path has been isolated, but either the lack of PAC4 from the level 4 controller or a hazard to one of the status word bits prevents the branch from executing. If either flag sets, the controller loops between the Initial Idle state and the BLB, BLX path until the circumstances that causes the flag clear.



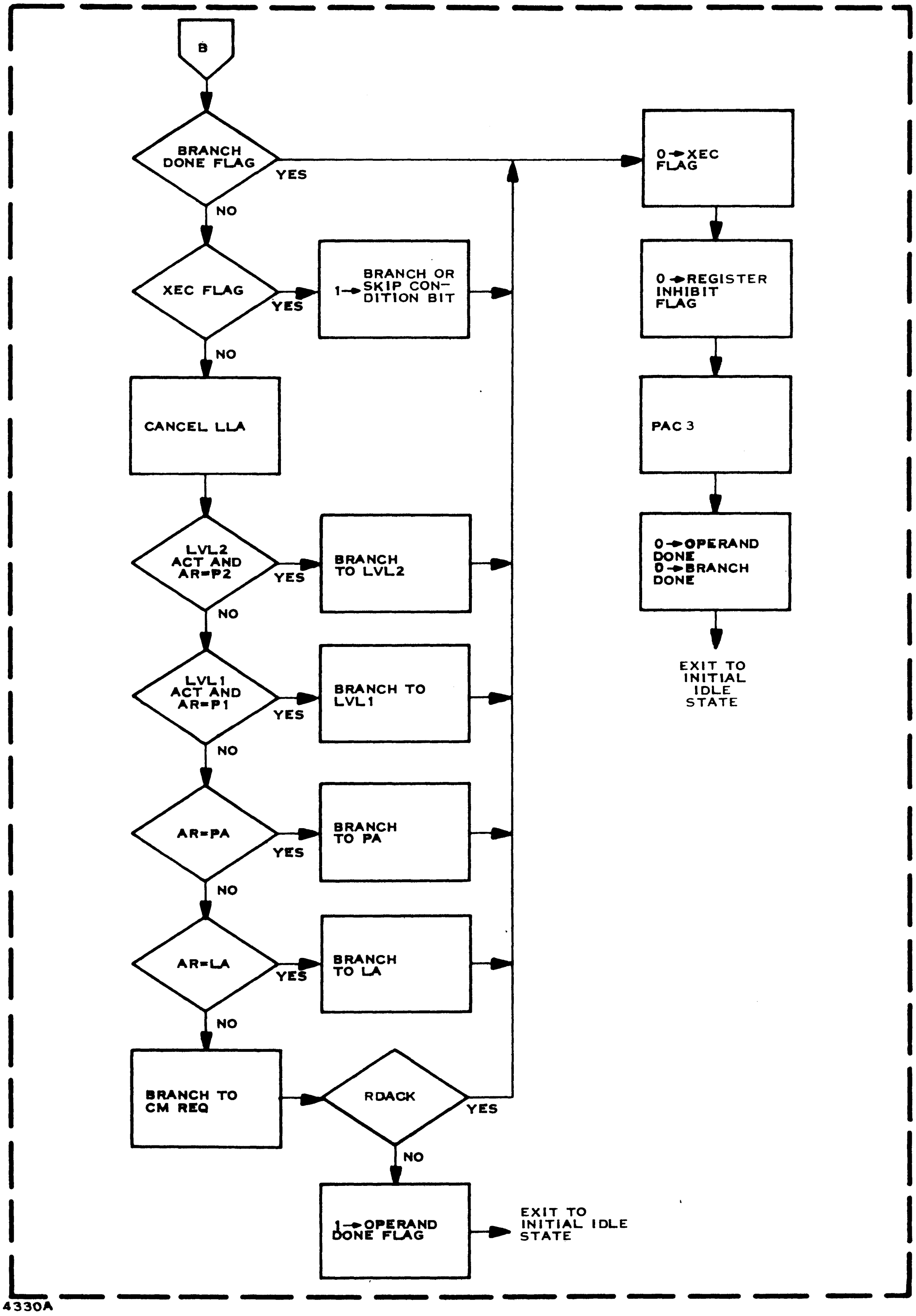
(B) 114328A

Figure 4-35. Blue Subcycle of Level 3 Idle State (Sheet 1 of 3)



(B) 114329A

Figure 4-35. Blue Subcycle of Level 3 Idle State (Sheet 2 of 3)



(B) 114330A

Figure 4-35. Blue Subcycle of Level 3 Idle State (Sheet 3 of 3)

When the controller enters the BLB, BLX path the Operand Done flag will not be set. If PAC4 is present from the level 4 controller, the pipe is examined for an instruction that will change the Arithmetic Exception or Mask condition bits. These bits are part of the program status word and are stored along with P3 into the designated register file doubleword when the BLB or BLX is performed. If these bits will be changed, the controller must wait until the hazard clears the pipe before storing status information. Under this circumstance, the controller checks the Branch Done flag to determine if this control path has previously been taken for this instruction. If the flag is set, the controller loops to the Initial Idle state. If it is not set, the controller examines the Execute flag. If the instruction is part of an execute operation, the actual branch will not be performed. Instead the controller sets the Branch or Skip Condition bit to indicate that a branch instruction was detected. This completes the execute operation and the controller returns to the Initial Idle state for the next control cycle. If the instruction is not an execute, the branch path will be taken. The controller cancels any LLA that may be in progress, since the branch will divert the instruction path from the path containing the target. The controller then determines if the branch address is in the IPU. For branches within the IPU the controller issues signals that inactivate all instructions between level 3 and the branch path so that the next active instruction that reaches level 3 will be the first instruction in the branch path. If the branch path is in central memory, the controller indicates to CMR that a memory request is required. When CMR is able to make the request, or if the branch was within the IPU, the controller sets the Branch Done flag to indicate that the branch path is ready to execute when the status word hazard clears or when level 4 can accept the branch path.

In the absence of any status word hazards the controller initiates sequence BI, and enables the level 3 to level 4 transfer to start the program status data down the pipe to be stored into the register file. The controller then examines the Branch Done flag to determine if a previous control cycle has prepared the branch path for execution. If this flag is not set, and the branch is not part of an execute instruction, the controller cancels any LLA in progress and determines if the branch is local (within the IPU) or in central memory. Local branches result in inactivating the intervening instructions so that the next instruction to reach level 3 will be the first instruction in the branch path. For a branch to a central memory location, the controller indicates to CMR that a memory request is required. If CMR is not ready to perform the fetch, the controller sets the Operand Done flag and loops between the Initial Idle state and the BLB, BLX path until CMR requests the new octet. When the controller is sure that the branch path is in the IPU and will supply the next active instruction to level 3, the controller clears the Execute, Register Inhibit, Branch Done and Operand Done flags, issues PAC3 to the level 2 controller, and returns to the Initial Idle state for the next control cycle.

If the instruction at level 3 is not an LLA, a BLB or BLX, the controller examines the R field of the instruction. If the R field is zero, the instruction specifies a no operation condition. If the R field is not zero, the instruction specifies a branch condition that must be resolved by exiting to the Decide (Blue) subcycle. A no operation condition results in issuing a Branch not Taken indication to the look ahead controller. If the branch instruction was a targeted branch of an LLA instruction, the no operation creates a target fail condition that requires the load look ahead controller to recover the previous instruction stream to replace the branch path that had been loaded into the pipe. The controller then clears the Execute and Register Inhibit flags, issues PAC3 to level 2 controller, and if the instruction was an execute, clears the Branch or Skip Condition bit in the program status word. This bit indicates that the branch was not taken when encountered by an execute instruction. The controller then returns to the Initial Idle state for the start of the next control cycle.

4-256 DECIDE. The controller enters the Decide subcycle from either the Green or the Blue subcycle to determine if a branch path will be taken. Instructions examined at the Green entry point to the Decide subcycle (see figure 4-36) are always conditional branches. The branch instructions examined from the Blue subcycle entry point, however, may be either conditional or unconditional. The instructions from the Green subcycle are not relative to the condition codes in the AU, since the branch decision comparison is performed in the IPU. For these reasons, two entries are possible into the Decide subcycle depending upon the original subcycle. Entry from the Blue subcycle involves additional determinations for a conditional branch, and if conditional, whether an instruction in the pipe may change the condition codes in the AU and thus affect the branch decision (hex register hazard). If such a hazard exists, the controller loops to the Initial Idle state until the hazard clears and the resulting condition codes are reflected in the AU bits. After these points have been cleared, the decision paths are identical for either a Green or a Blue instruction.

If the AU or IPU examination of the condition bits indicates that the branch should be taken, or if the branch is an unconditional branch (R-field = 7), the controller determines if the instruction is part of an execute instruction. If the execute flag is set, the controller indicates that the branch will not be taken, and if the instruction is not a BXEC (which is not recognized as a branch during execute instructions), the controller sets the Branch or Skip Condition bit to indicate that the branch was detected. If the Execute flag was not set, and the instruction does not require processing for an indirect address (Yellow subcycle), the controller inactivates any LLA instruction in progress since the branch will divert the instruction path from the target. The controller then determines if the branch is to any of the local IPU registers.

For a local branch, the controller issues a control signal that inactivates all instructions in the pipe between the branch path instruction and level 3 so that the next active instruction that reaches level 3 will be the first instruction in the branch path. If the branch is not local, the controller indicates to CMR that a memory request will be required, and loops to the Initial Idle state until CMR is able to make the request for the new octet. Once the branch path is in the IPU, the controller clears the control flags, issues PAC3 to level 2 controller, and exits to the Initial Idle state for the next control cycle.

If the branch will not be taken, the controller generates a signal that indicates to the other controllers that the branch is not taken. If the instruction is part of an execute sequence and is not a BXEC instruction, the controller clears the Branch or Skip Condition bit in the status doubleword to

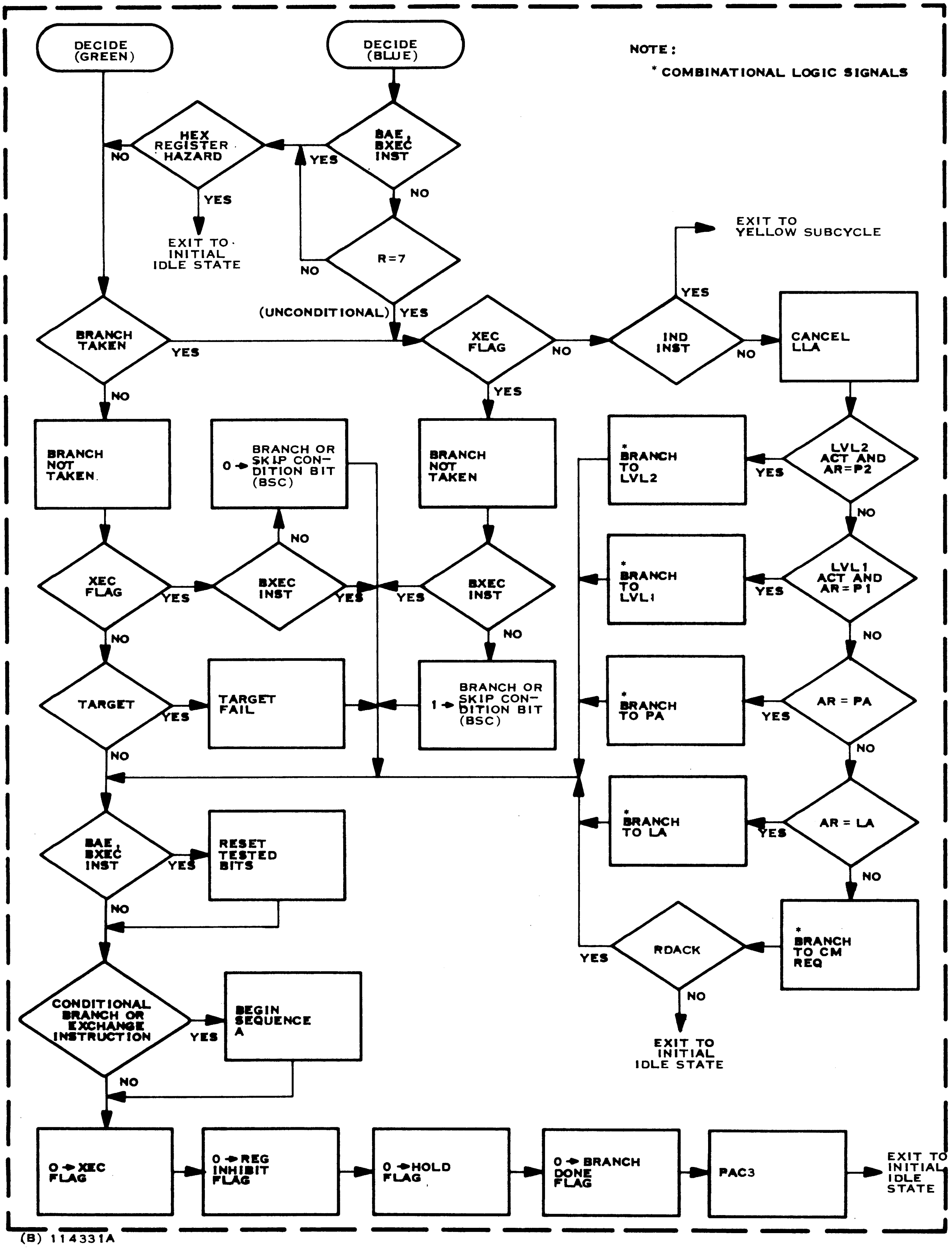


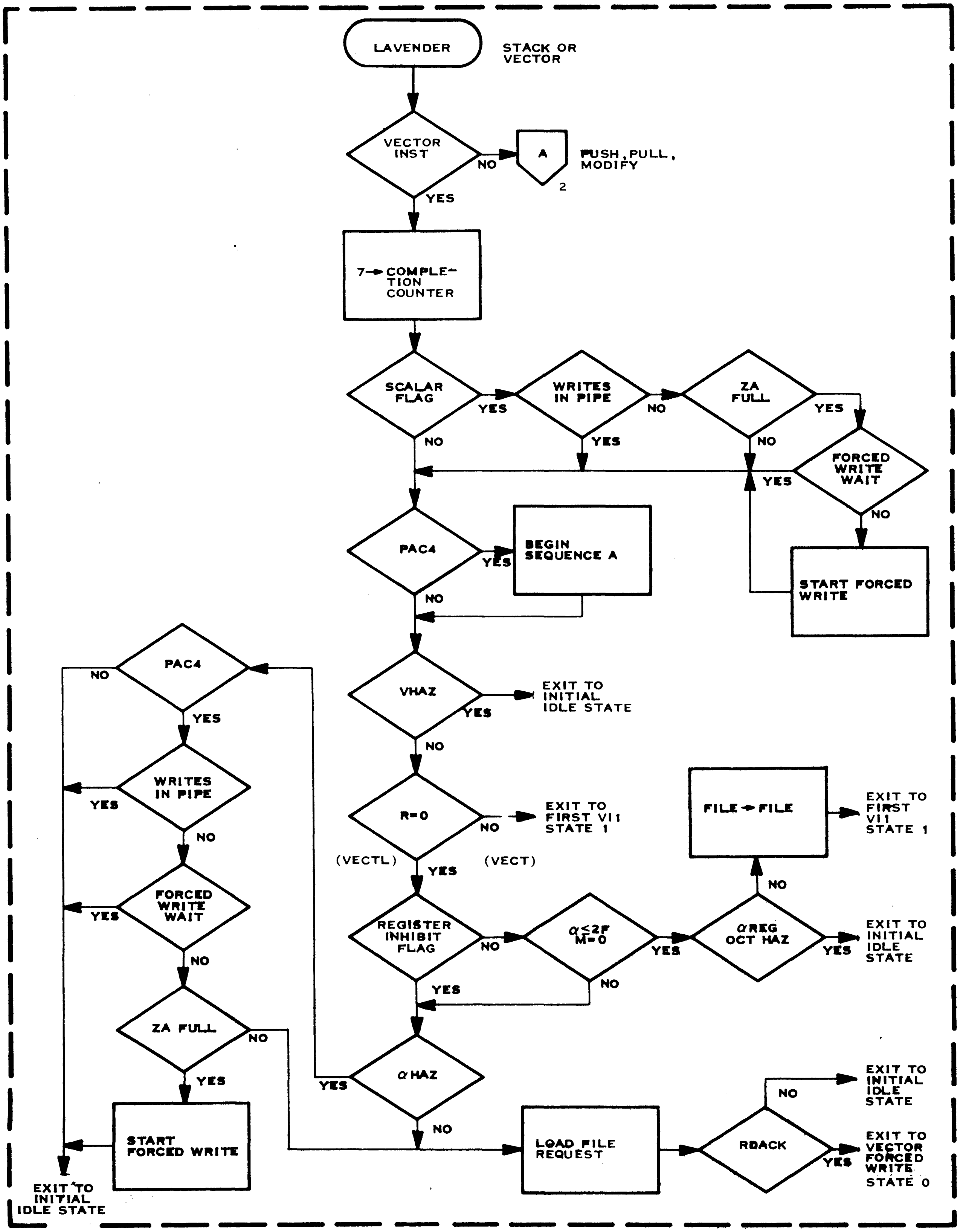
Figure 4-36. Decide Subcycle of Level 3 Idle State

indicate that the branch instruction was examined but not taken. If the instruction was not part of an execute sequence, the controller determines if the branch was targeted by a previous LLA instruction. If the instruction was a target, then the look ahead controller will have to recover the old instruction sequence to replace the branch sequence that it has loaded into the pipe following the targeted branch. The level 3 controller issues "Target Fail" to the look ahead controller to indicate that condition.

Regardless of the control path taken through the Decide subcycle, when the branch determination has been made and satisfied, the controller exits by reinitializing the flags and cells used during the determination. It resets the condition bits used for determination during BAE and BXEC instructions, clears the Execute, Register Inhibit, Hold and Branch Done flags, issues PAC3 to the level 2 controller, and exits to the Initial Idle state for the next control cycle.

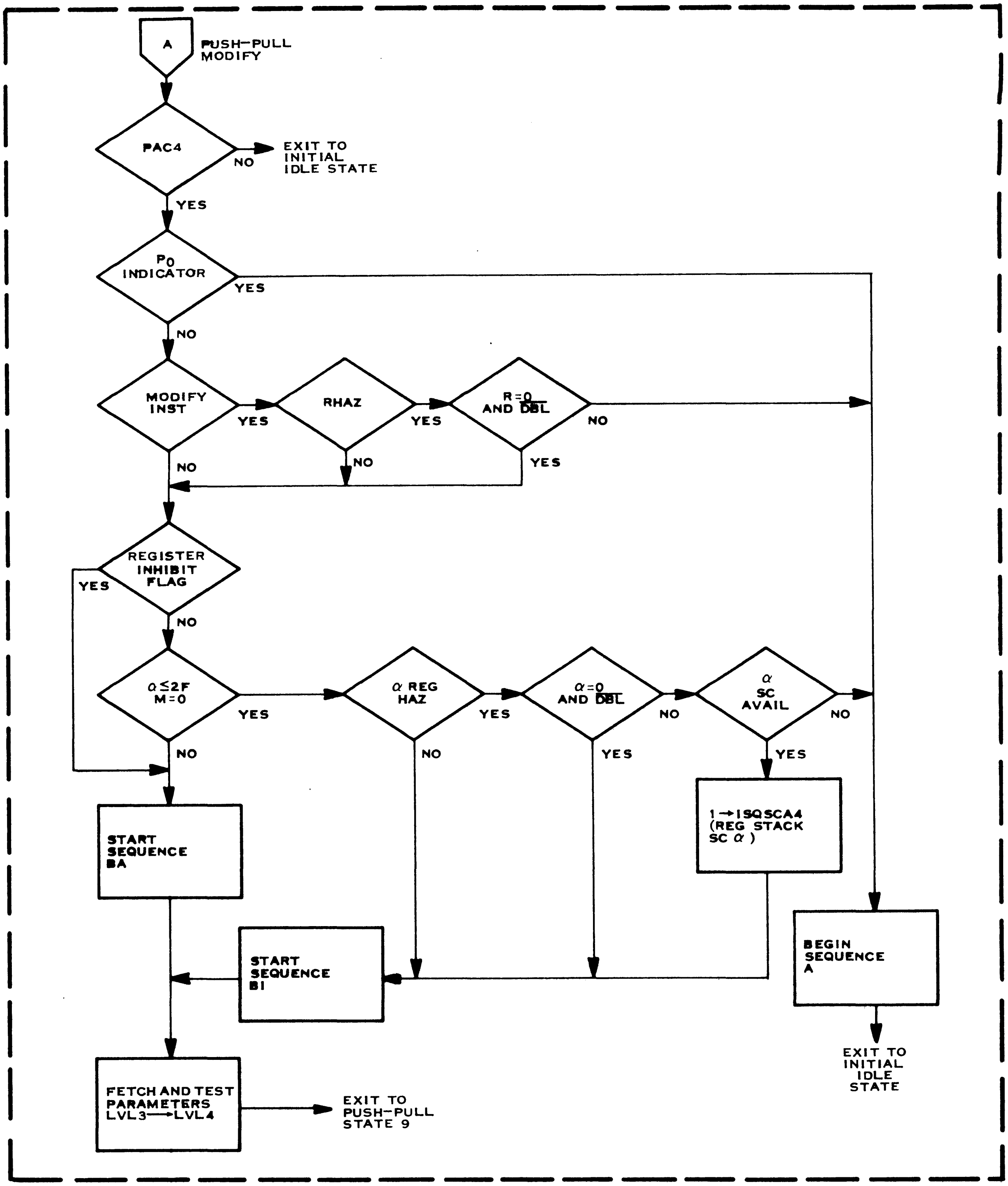
4-257 LAVENDER. The controller enters the Lavender subcycle (figure 4-37) from the Initial Idle state when the instruction at level 3 is either a vector initiate instruction (VECT or VECTL), or a stack instruction (PSH, PUL or MOD). The subcycle determines the type of instruction that is at level 3, performs the steps required to ensure that the pipe is prepared to process that instruction type, and exits to one of three operational states or returns to the Initial Idle state if the IPU cannot yet process the instruction. The Lavender subcycle is divided into two sections. One section ensures that the pipe is prepared to process vector initiation instructions; the other section is used for stack instructions.

If the instruction at level 3 is a vector initiation instruction, the controller loads a value of "7" into the completion counter. The counter decrements for each word pulled from the vector parameter file during the vector loading process so that the controller knows which word it is pulling (see Vector Burst state), and when the file is exhausted. If a scalar instruction is in the pipe, the controller ensures that the results will be stored by a forced write operation. The controller then examines the hazard detection circ uit to determine if an instruction in the pipe will change a value in the vector parameter file. If such a hazard exists, the controller loops to the Initial Idle state until the hazard clears. The controller next checks the R-field of the vector initiation instruction. If the R-field is not equal to zero, it must be equal to one if the instruction is a valid vector instruction. This value indicates that the instruction is VECT and that the desired initiation sequence is already in the vector parameter file. The controller exits to the First VI1 state to begin pulling the initiation parameters from the register file. If the R-field is equal to zero, the instruction is VECTL indicating that the vector parameter file must be loaded with a new octet before using the file to supply the vector initiation parameters. The controller then examines the Register Inhibit flag, the M-field of the instruction and the alpha address to determine if the location to supply information for the vector parameter file is within the register file. If the address is within the register file, the controller ensures that no instruction in the pipe will alter the contents of that location before enabling the transfer of the designated file to the vector parameter file.



(B) 114338A

Figure 4-37. Lavender Subcycle of Level 3 Idle State (Sheet 1 of 2)



(B) 114344A

Figure 4-37. Lavender Subcycle of Level 3 Idle State (Sheet 2 of 2)

Since the desired octet will be in the vector parameter file for the next control cycle, the controller exits to the First VII state as if the instruction were a VECT instruction. If the alpha address is not in the register file, the controller also checks for alpha hazards. If an instruction in the pipe will be stored into the octet indicated by the alpha address, the controller ensures that the value will be written into memory by a forced write, and exits to the Initial Idle state until the hazard clears. When the hazard clears, the controller issues a load file request to CMR for the octet indicated by the alpha address. If CMR is prepared to make that memory request (RDACK), the controller exits to the Vector Forced Write state to await the arrival of the octet from memory.

If the instruction at level 3 is a stack instruction (push, pull or modify), the controller examines the P_o indicator. This signal originates in the MBU ROM and will never be present at the start of a stack instruction sequence. If two stack instructions are adjacent, it is possible for the second stack instruction to see the P_o indicator from the first stack instruction. Therefore, a true P_o indicator during the first Lavender subcycle must be due to a preceeding stack instruction. The controller then determines if the instruction is a modify instruction and is therefore susceptible to R-field hazards. Since the R-field of a modify instruction selects a register file location whose contents will be used to modify a stack parameter, an instruction in the pipe that alters the selected register file location constitutes a hazard. If the selected register file location is not location "00" (all zeroes), the controller loops to the Initial Idle state until the hazard clears.

The controller then determines if the alpha address selects a register file location by examining the Register Inhibit flag, the M-field and the alpha address. If the address is not in the register file, the controller initiates sequence BA and enables the transfer of level 3 to level 4 to load the stack parameter into the pipe from central memory. If the alpha address is within the register file, the controller checks for a hazard condition. If an instruction in the pipe alters the register file location selected by the alpha address, the alpha address is not location 00, and no short circuit path is available to recover from the hazard, the controller loops to the Initial Idle state until the hazard exits from the pipe and is stored into the register file. However, if a short circuit path is available in the AU that will enable the CP to feed back the modifying instruction result to the input of the AU for replacement of the previous value, the controller need not wait for the hazard to clear the pipe. Instead, it sets the "short circuit at level 4" flag to indicate the condition. When the faulty value reaches the AU input level, it will be replaced by the updated value from the output of the AU. If no alpha hazard exists or it can be ignored or circumvented, the controller initiates sequence BI and enables the level 3 to level 4 transfer to load the stack parameters from the register file into the pipe. After the controller has started the parameters down the pipe, the controller exits to the Push-Pull state to continue the operation.

4-258 VECTOR FORCED WRITE STATE (STATE 0)

The level 3 controller enters state 0 (figure 4-38) from the Lavender subcycle of the Idle state when a VECTL instruction at level 3 required a memory request to load the vector parameter file before the parameter file could be used. The

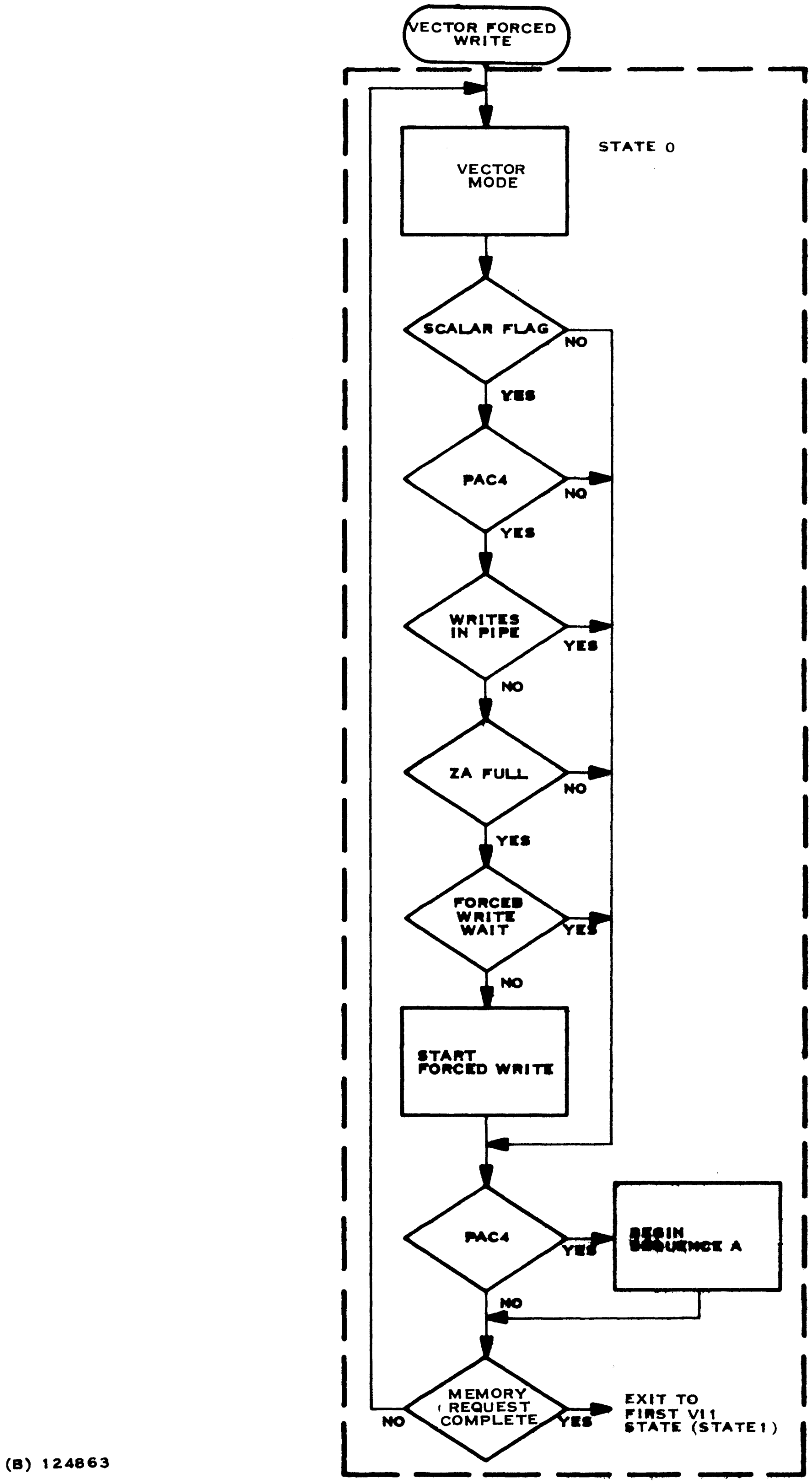


Figure 4-38. Vector Forced Write State of Level 3 Controller

controller remains in state 0 until the requested octet returns from memory and can be loaded directly into the vector parameter file. While in state 0, the controller ensures that the pipe is free of all scalar instructions by initiating a forced write operation to store the contents of the Z file in the MBU if valid data is in the Z file. The controller detects the presence of data in the Z file by examining the ZA Full bit in the Z Model of the IPU. When the required octet is stored into the vector parameter file, the controller exits to the First VI1 state to begin unpacking the vector parameter file.

4-259 FIRST VECTOR INITIATION (STATE 1)

The controller enters state 1 (figure 4-39) from either the Lavender subcycle of the Idle state, or from the Vector Forced Write state. When the controller enters state 1, the vector parameter file in the register file contains the set of specifications that will initiate the first vector operation. While in this state, the controller ensures that no scalar operation is still in the pipe and that the MBU Z file is free of valid data. During the first cycle through the state, the completion counter (set to "7" in the Lavender subcycle) is at a value of seven. The controller then ensures that the pipe is empty, decrements the count in the completion counter, and transfers word 1 of the vector parameter file into level 2 for input to the address modification circuits (words 1, 2 and 3 contain starting addresses of the three vectors that can be changed through address development).

During the second cycle through state 1, the completion counter is no longer at a value of seven. The controller then ensures that level 4 of the IPU and levels 5 and 6 in the MBU are not active and can therefore receive the parameters. The controller transfers word 0 from the vector parameter file to level 4 of the IPU (bypassing the address modification circuit), transfers level 2 to level 3 through the address modification circuits, and loads word 2 from the vector parameter file into level 2. Having initially loaded the IPU with the first three words of the vector parameter file, the controller exits to the Vector Burst state to continue unpacking the vector parameter file.

4-260 VECTOR BURST (STATE 2)

The controller enters state 2 from state 1 after the IPU levels have been loaded with the first three words in the vector parameter file. The contents of the IPU, therefore, is: word 0 at level 4 (to the MBU), word 1 at level 3 (having been modified as required), word 2 at level 2, and a value of six in the completion counter. The controller recycles through state 2 seven times while the vector parameter file is unpacked, word by word, and sent to the MBU. During each cycle the controller defines the Vector Mode of the IPU, clears the VI Start flag, decrements the completion counter, and holds the MBU from action by issuing Vector Wait. The LD Active flag is not needed during vector processing since all operations are stored into memory upon completion. XA and YA Active are also not needed during vector processing since the IPU cannot force another instruction into the MBU while it is processing a vector. During each cycle through the state, the controller samples the count in the completion counter to determine what action to perform during that control cycle. The flowchart for the Vector Burst state (figure 4-40) defines the transfers that are enabled for each of the decremented counts of the completion counter. When all of the vector parameter file has been transferred to the MBU, the controller examines the level 2 activity bit to determine if the next instruction in the program has reached level 2. If level 2 is active, the controller exits to the Vector Go state; if level 2 is not active, the controller exits to the Level 2 Wait state until a new instruction enters level 2.

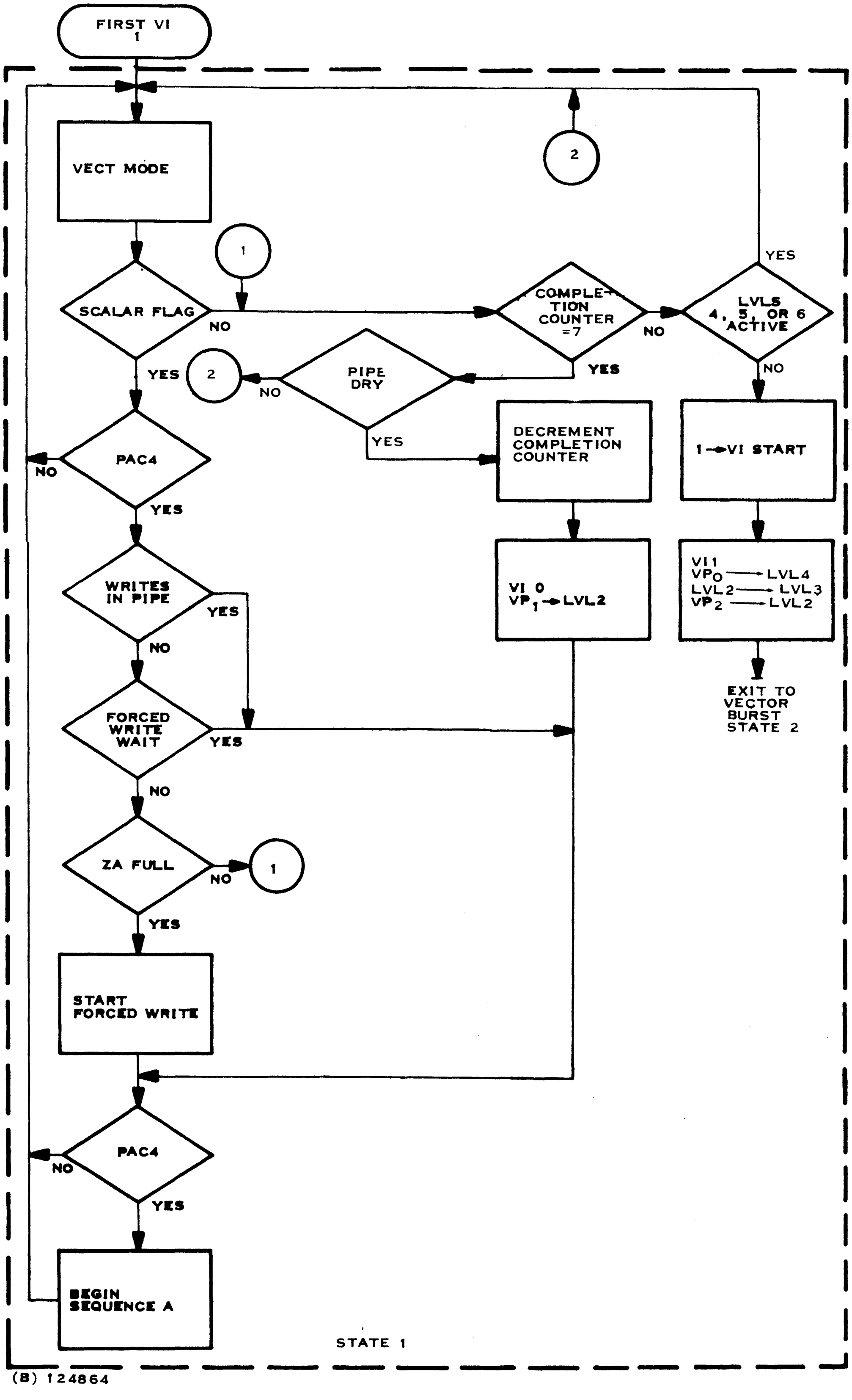
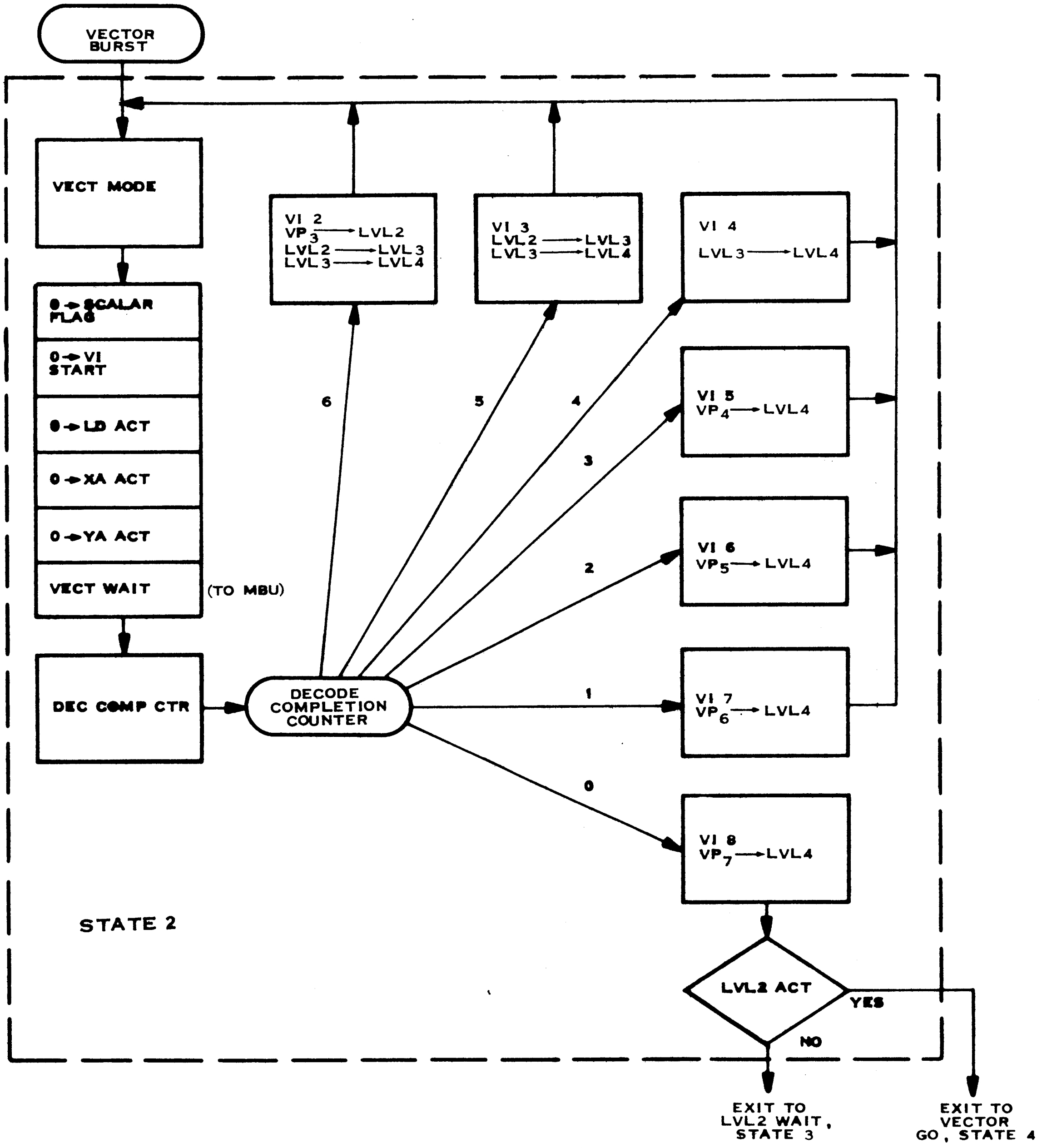


Figure 4-39. First Vector Initiation State of Level 3 Controller



(B) 124865

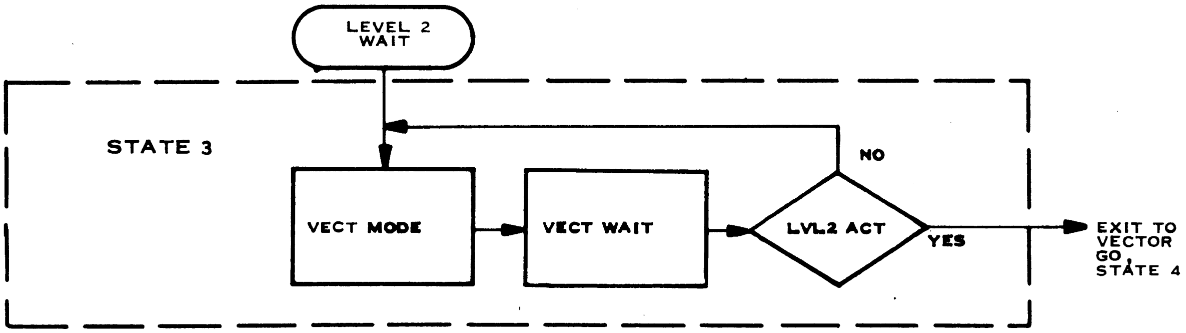
Figure 4-40. Vector Burst State of Level 3 Controller

4-261 LEVEL 2 WAIT (STATE 3)

The controller enters state 3 from state 2 after the vector parameter file has been loaded into the MBU to start a vector operation, but no new instruction is at level 2 of the IPU. Since the IPU can only detect an instruction hazard caused by storing vector results through comparison of the storage addresses with the address in P3 (near range hazard), the next instruction in the program sequence must be in level 3 before any of the vector results are stored. This will enable the IPU to compare the address of the instruction (P3) against each octet address that is stored during the vector operation. If the vector stores into the octet containing the new level 3 instruction, the instruction will have to be refetched to provide the correct information. For this reason, state 3 issues a Vector Wait signal to the MBU that inhibits the MBU from storing any vector results until the next instruction enters level 2. When level 2 becomes active, the controller exits to the Vector Go state. Figure 4-41 illustrates the Level 2 Wait cycle.

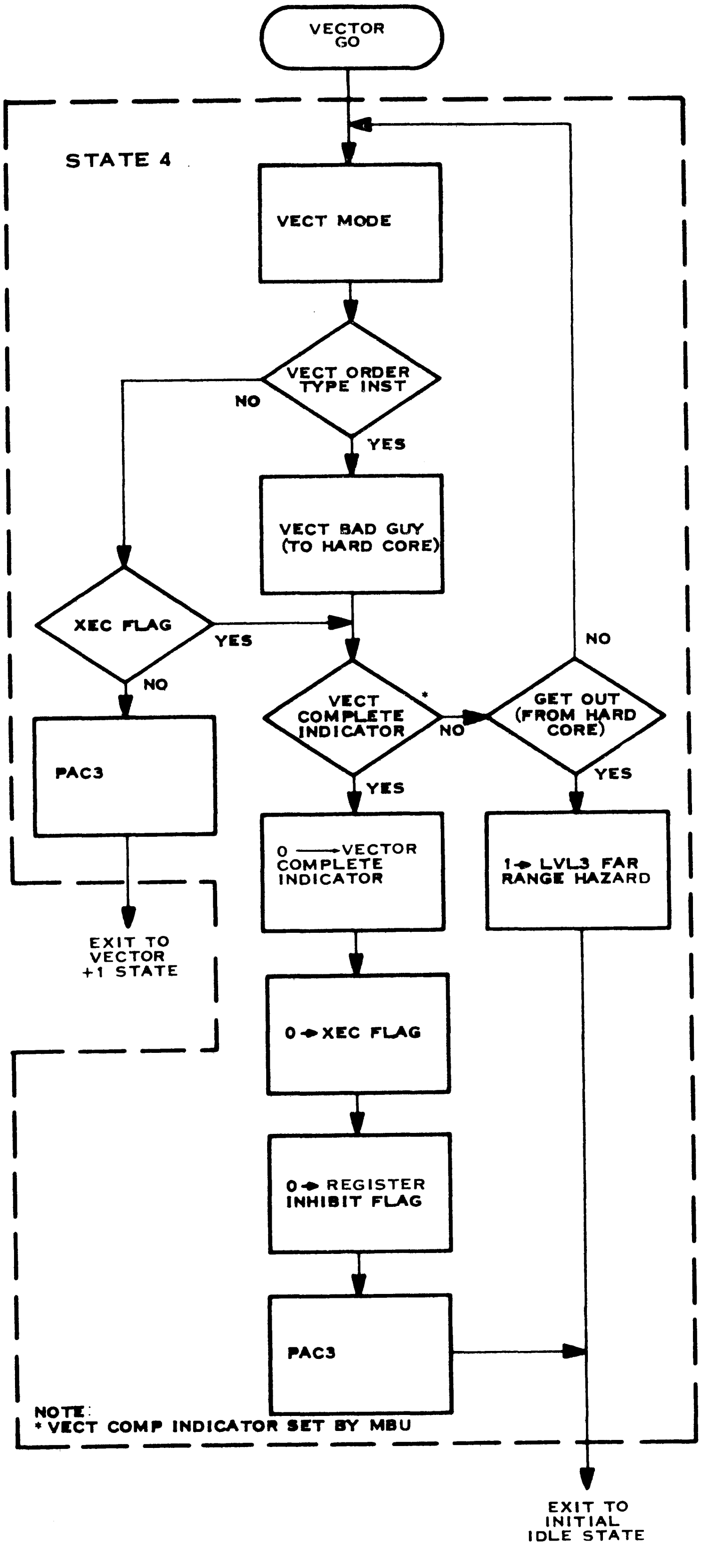
4-262 VECTOR GO (STATE 4)

The controller enters state 4 (figure 4-42) from either state 2 or state 3 when level 2 becomes active. As the controller enters this state, the Vector Wait signal that had been preventing the MBU from storing vector results drops. If the vector instruction being executed is not an order instruction and is not part of an Execute instruction, the controller sends PAC3 to the level 2 controller and exits to the Vector + 1 state. If, however, the instruction is a vector order (generating Vector Bad Guy to the hard core controller) or part of an execute, the controller waits in state 4 until the vector has completed or until the controller receives a Get Out signal from the hard core controller. If the MBU sets the Vector Complete indicator, the controller clears that indicator to acknowledge its receipt, clears the Execute and Register Inhibit flags and issues PAC3 to the level 2 controller before exiting to the Initial Idle state. The hard core controller generates Get Out when it receives a command from the CR file (in the PP) that requires a context switch. Get Out forces the MBU to wind down the vector currently in progress without evacuating the X or Y buffer files. The level 3 controller then sets the Level 3 Far Range Hazard flag so that the instruction that was executing will be started over when the instruction sequence begins again. The controller then exits to the Initial Idle state for the next control cycle.



(A) 124866

Figure 4-41. Level 2 Wait State of Level 3 Controller



(B) 124867

Figure 4-42. Vector Go State of Level 3 Controller

4-263 VECTOR + 1 (STATE 5)

The level 3 controller enters state 5 (figure 4-43) from either the Vector Go state (state 4) or the Load File Request Wait state (state 6). In either case a valid instruction has been transferred into level 3 and the controller examines that instruction in state 5 for fault indications involved with that instruction. When it enters from state 6, the controller has already been through state 5 once. The second entry causes the controller to loop within state 5 until the vector being run in the MBU is complete.

In state 5 the controller enables the next control clock to load a value of seven into the completion counter for use in loading the next vector instruction parameter set. The controller also checks for a memory protect violation, a near range or far range hazard, an alpha address hazard, an indirect instruction, a targeted branch, or an instruction that is not another vector or a load file. Any of these conditions prevents the controller from further processing of the instruction until the currently processing vector clears the pipe. The controller, therefore, waits for the MBU to set the Vector Complete indicator. It then clears that indicator to indicate recognition of the completion. The controller also clears the alpha address hazard flag since if the vector has completed storing into memory, the hazard has cleared and the IPU can use the alpha address to fetch a word from memory. The controller then returns to the Initial Idle state for the next control cycle. Notice that if a near range hazard was detected for the instruction at level 3, the controller set the Level 3 Far Range Instruction Hazard flag. Setting this flag causes the controller to refetch the level 3 instruction from memory when the executing vector clears the pipe.

If none of the discussed conditions prevent the controller from further processing the instruction at level 3, the controller determines if the instruction is another vector or a load file instruction. If it is a vector instruction, the controller examines the R-field of the instruction to determine if the instruction is a VECT (R=1) or a VECTL (R=0) instruction. If it is a VECT instruction, the controller exits to the Prime Second Vector state since the correct parameters are currently in the Vector Parameter file of the IPU. For a VECTL instruction the controller determines if the octet to be loaded into the Vector Parameter file is in the register file or in central memory. If it is in the register file, the controller enables the next clock pulse to transfer that file into the Vector Parameter file and exits to the Prime Second Vector state where it checks for alpha hazards. If the parameter file is to be loaded from central memory, the controller checks for alpha hazards, setting the alpha hazard flag if one exists, and issues a load file request to CMR. When CMR is able to process the memory request for the octet, it transmits RDACK to the controller, and the controller exits to the Load File Request Wait state.

If the instruction at level 3 is a load file instruction, the controller determines if it is a file to file load or a load from central memory. If it is a file to file load (a one clock operation), the controller waits for the vector to terminate before performing the transfer. This delay avoids the possibility that the vector operation might change the instruction at level 3 by producing an instruction hazard (storing into the P3 address in memory).

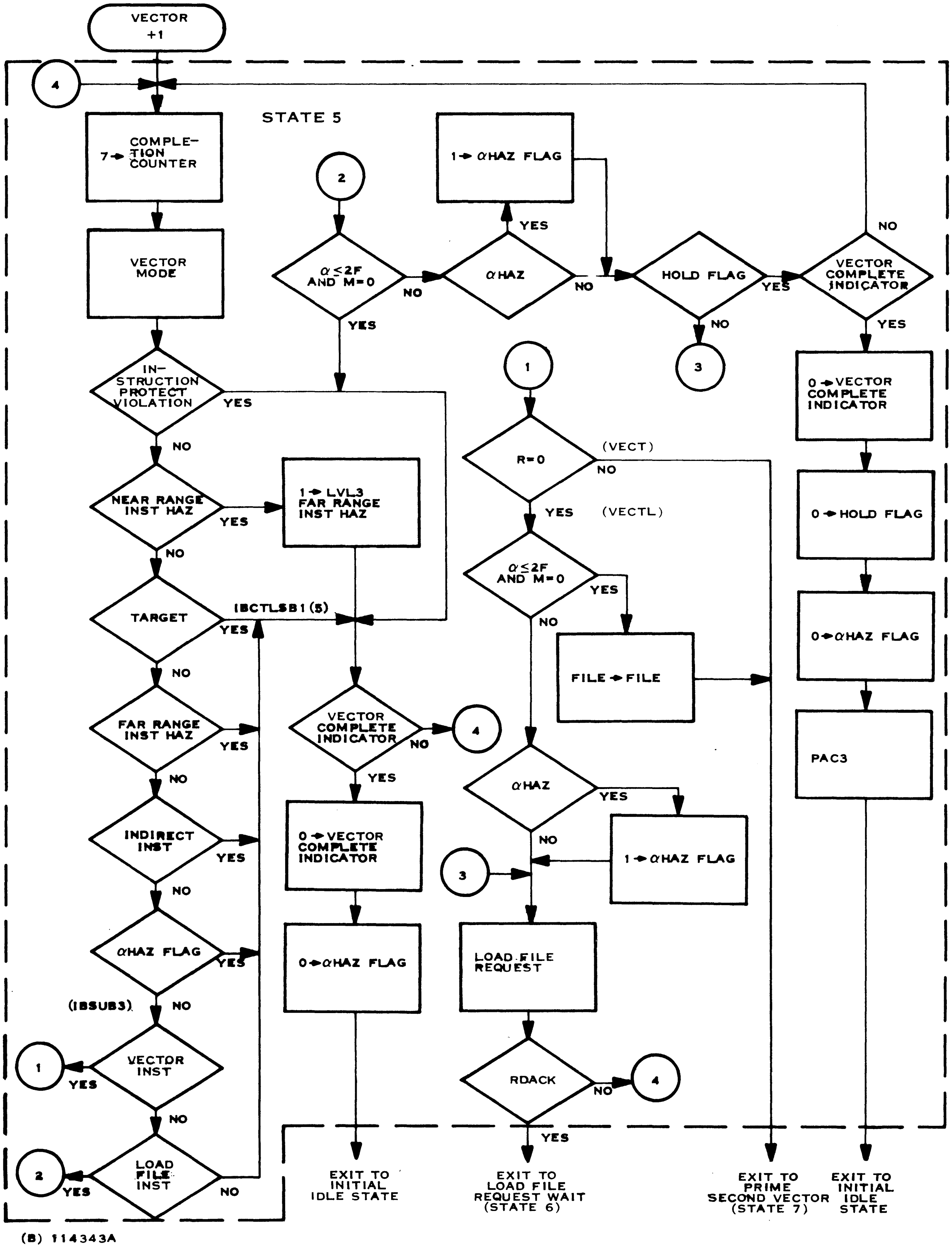


Figure 4-43. Vector +1 State of Level 3 Controller

Since the chance of this hazard developing is small, the controller initiates the load file operation if the instruction is a memory to file load. Choosing this alternative saves processor time by performing the memory fetch while still running the vector. Before issuing the memory request, the controller checks for alpha hazards, setting the Alpha Hazard flag if a hazard exits. If the Hold flag is not set, the controller issues the Load File Request to CMR and exits to the Load File Request Wait state when CMR accepts the request.

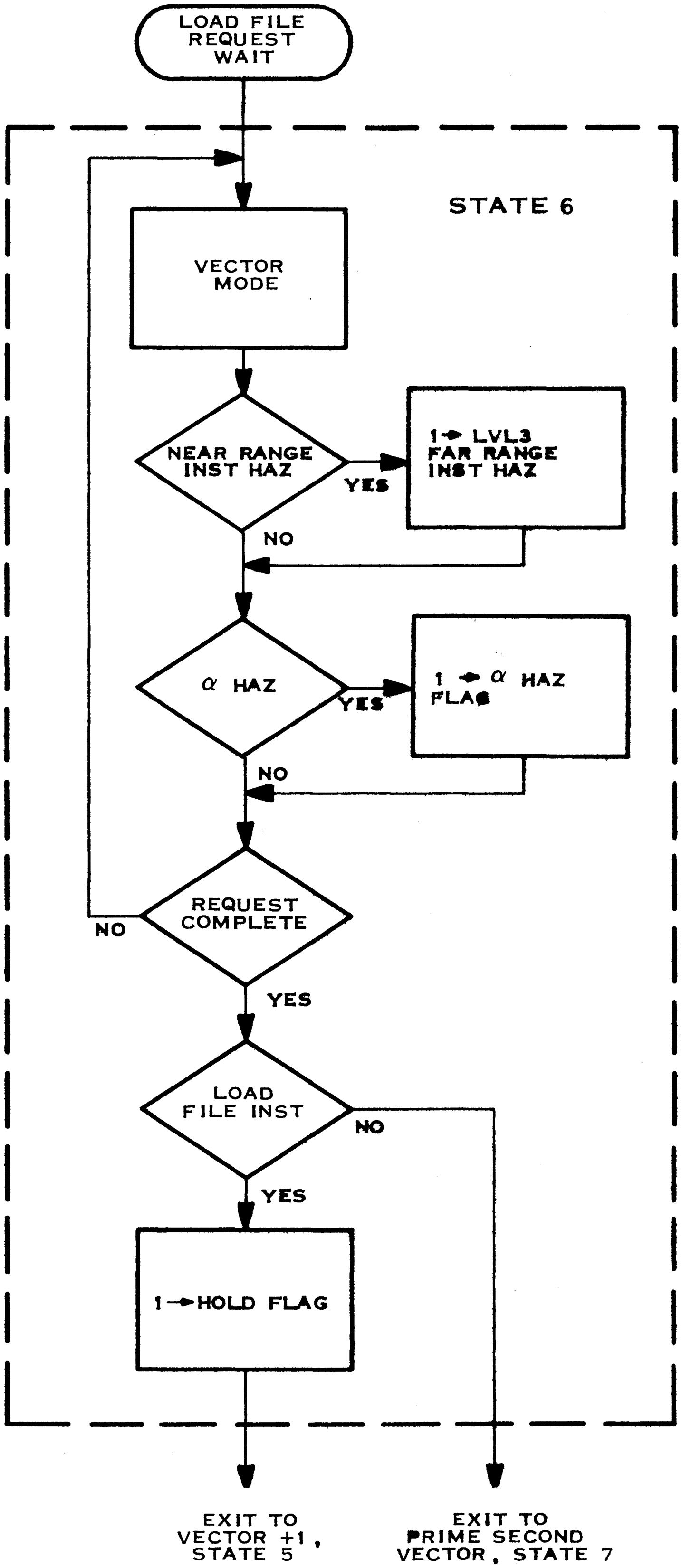
The Hold flag sets in the Load File Request Wait state when the controller exits that state to return to state 5. Therefore, if the Hold flag is set the load file instruction at level 3 has already been executed. The controller therefore loops within state 5 and continues to monitor the executing vector in the MBU for instruction hazards to the instruction that has been performed. When the vector terminates, the controller clears the Vector Complete indicator to recognize the termination, clears the Hold and Alpha Hazard flags, issues PAC3 to the level 2 controller and exits to the Initial Idle state for the next control cycle.

4-264 LOAD FILE REQUEST WAIT (STATE 6)

The controller enters the Load File Request Wait state (figure 4-44) from state 5 after it has initiated a load file request to CMR and CMR has transferred that request to central memory. The load file request may be part of a load file instruction or a vector operation requiring new parameters in the vector parameter file (VECTL). While in state 6 the controller monitors the storage addresses generated by the vector that is executing in the MBU to determine if a near range instruction hazard (current instruction is invalid) or an alpha address hazard (origin of load file is invalid) develops. If either hazard occurs, the controller sets the Level 3 Far Range Instruction Hazard flag or the Alpha Hazard flag, respectively, so that the controller will replace the invalid parameter when the vector terminates. When the memory request is complete, the controller determines which of the two instruction types produced the load file operation. If it was a Load File instruction, the controller sets the Hold flag to prevent the controller from returning to state 6, and exits to state 5 (Vector + 1). If a VECTL instruction initiated the load file operation, the controller exits to the Prime Second Vector state to load word 1 of the vector parameter file into the pipe in preparation for the next operation.

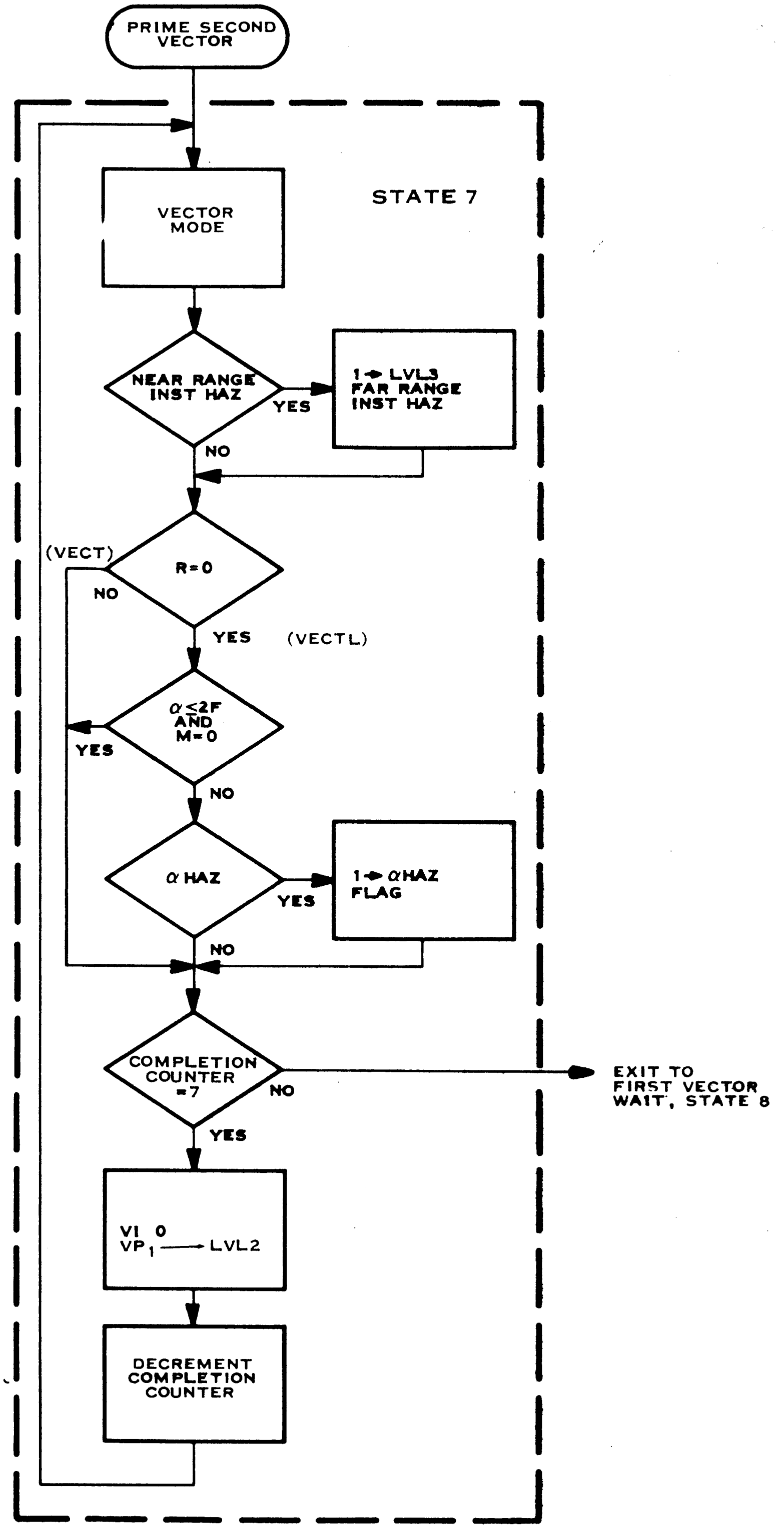
4-265 PRIME SECOND VECTOR (STATE 7)

The controller enters state 7 (figure 4-45) from either the Vector + 1 state (state 5) or the Load File Request Wait state (state 6) when the controller is sure that the proper set of parameters is in the vector parameter file. The controller remains in state 7 for one clock and then exits to the First Vector Wait state (state 8). While in state 7, the controller monitors the storage addresses generated by the vector running in the MBU. If a near range hazard occurs, invalidating the current instruction at level 3, the controller sets the Level 3 Far Range Instruction Hazard flag so that the instruction will be refetched from memory when the vector terminates. If the instruction at level 3 is a VECTL instruction, the controller also checks for an alpha hazard to



(A) 124868

Figure 4-44. Load File Request Wait State of Level 3 Controller



(B) 124869

Figure 4-45. Prime Second Vector State of Level 3 Controller

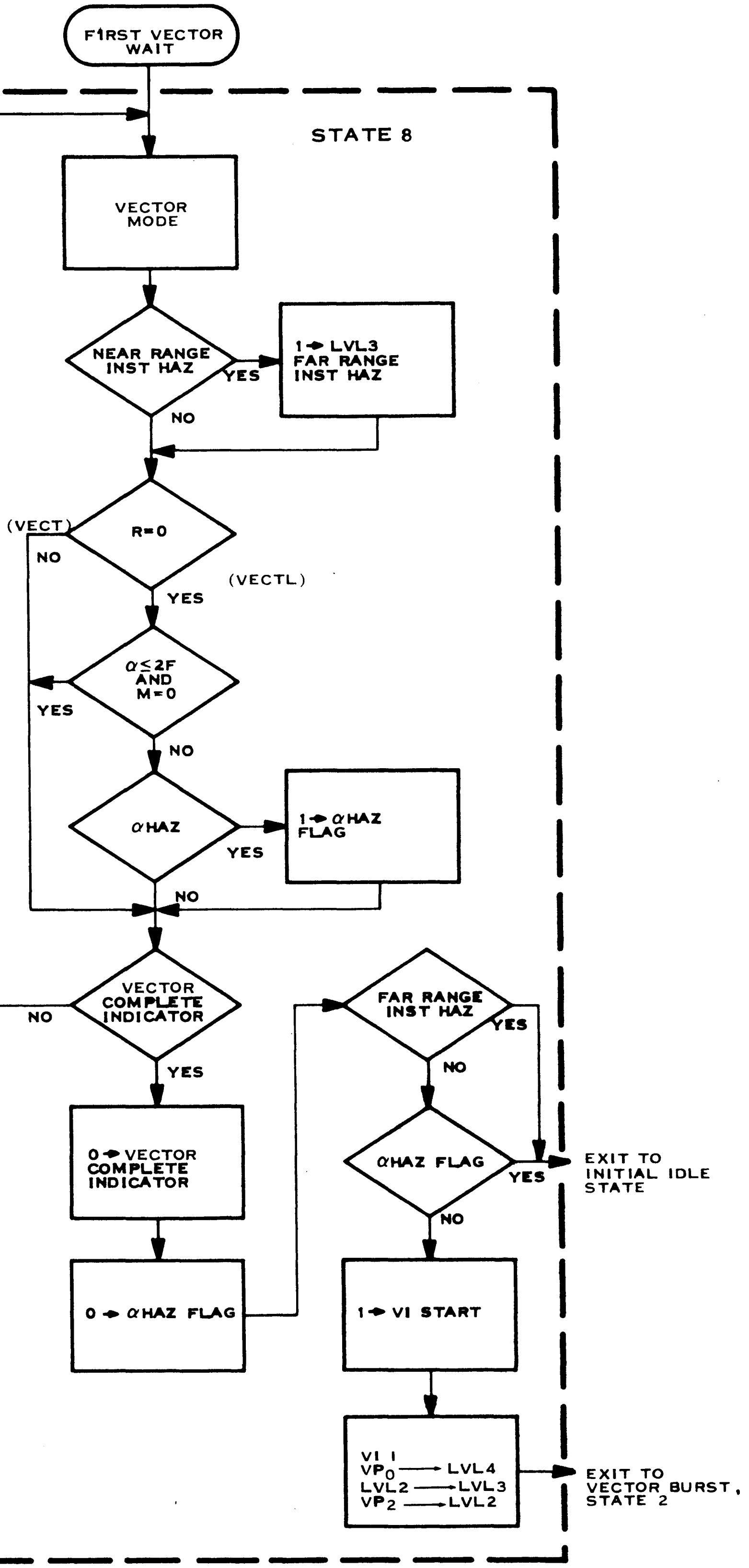
ensure that the vector parameters loaded during the VECTL preparation are still valid values. If an alpha hazard occurs, the controller sets the Alpha Hazard flag so that the parameters will be reloaded when the vector terminates. The first time through state 7, the completion counter is at a value of seven because it has not changed since set to 7 in the Vector + 1 state. The controller, therefore, transfers word 1 of the vector parameter file into level 2 and decrements the completion counter. During the next control cycle, the controller will repeat the instruction and alpha hazard inspections, and exit to the First Vector Wait state, since the completion counter will no longer be at a value of seven.

4-266 FIRST VECTOR WAIT (STATE 8)

The controller enters the First Vector Wait state (figure 4-46) from the Prime Second Vector state after having loaded the vector parameter if required and transferring word 1 of the vector parameter file into level 2 of the IPU. In state 8 the IPU remains ready while the controller monitors the storage addresses generated by the MBU to detect hazards. If a near range hazard occurs, the controller sets the Level 3 Far Range Instruction Hazard flag. If the instruction waiting at level 3 is a VECTL instruction (R=0) the controller checks for alpha address hazards to ensure that the vector parameter file was loaded with valid data. If an alpha hazard occurs, the controller sets the alpha hazard flag. When the MBU sets the Vector Complete Indicator, the controller enables the next clock pulse to clear both the Vector Complete Indicator (in response to the MBU) and the alpha hazard flag. It then makes a final examination to determine if the instruction at level 3 has incurred a hazard during the vector storage operation. If this far range hazard exists or the alpha hazard flag is set, the controller exits to the Initial Idle state to refetch the correct parameters from memory. If no hazards have occurred, the controller sets VI Start, transfers word 0 of the vector parameter file to level 4, enables word 1 from level 2 through the address modification circuits to level 3, and loads word 2 into level 2. The IPU is then prepared to begin processing the second vector parameter set. The controller exits to the Vector Burst state to unpack the vector parameter file and transfer it to the MBU.

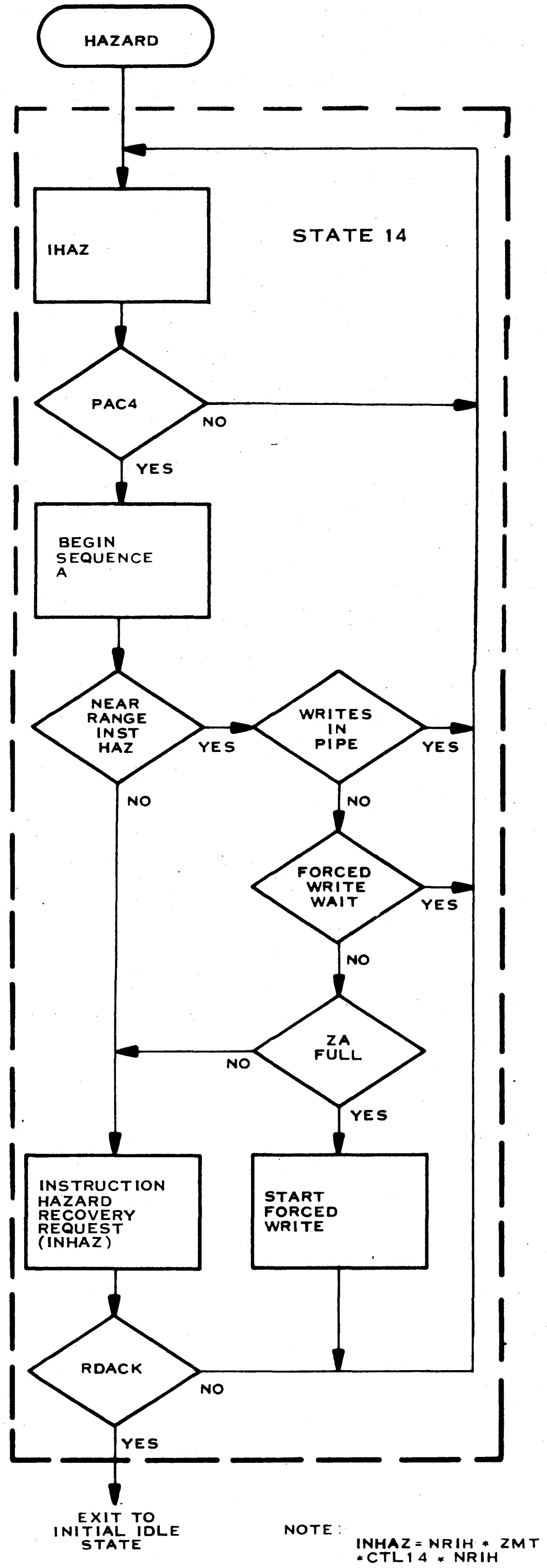
4-267 HAZARD (STATE 14)

The controller enters the Hazard state (figure 4-47) from the Initial Idle state when it detects a far range hazard (a store in the pipe that alters an instruction that is in pipe levels 0, 1 or 2), a near range hazard (a store in the pipe that alters the instruction in level 3), or a target of an LLA reached level 3 but the branch was not taken. Any of these cases require the IPU to perform a memory request to recover valid data for continuation of the instruction sequence. When the controller enters this state it generates IHAZ to indicate the presence of a hazard condition. If the hazard is a near range instruction hazard, the controller waits until all store operations are clear from the CP. Then, if the Z buffer in the MBU contains valid data (ZA Full), it initiates a forced write operation to write that data into memory and waits until the operation is complete (not Forced Write Wait). When the controller is sure that a memory request will produce valid data, it generates an instruc-hazard recovery request (INHAZ) to CMR and waits until CMR can perform the



(B) 124870

Figure 4-46. First Vector Wait State of Level 3 Controller



(B) 124871

Figure 4-47. Hazard State of Level 3 Controller

request to memory (RDACK). If RDACK is true, the request has been sent to memory. The controller exits to the Initial Idle state for the start of the next control cycle.

4-268 INDIRECT REQUEST (STATE 15)

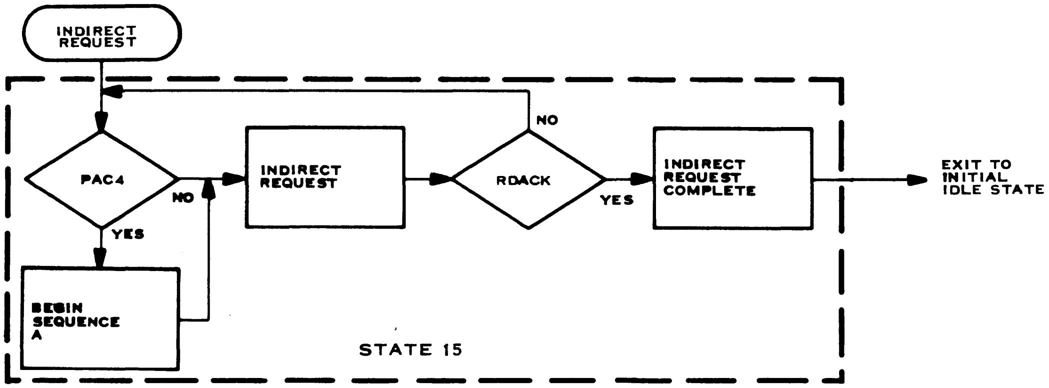
The controller enters the Indirect Request state (figure 4-48) from the Yellow subcycle of the Idle state when it determines that an indirect address cannot be satisfied within the IPU instruction registers, the controller had issued an Indirect Request to CMR, and CMR was not prepared to process the request when it was made. The controller remains in the Indirect Request state until RDACK becomes active, indicating that CMR has accepted the memory request and has forwarded it to central memory. Until that time, the controller maintains the Indirect Request signal to CMR at a high level. When CMR accepts the request, the controller signals Indirect Request Complete and exits to the Initial Idle state for the next control cycle.

4-269 ORANGE WAIT (STATE 12)

The controller enters the Orange Wait state (figure 4-49) from the Orange subcycle of the Idle state after the controller has issued a load file request or the store file request of a store file multiple to CMR, and CMR has forwarded that request to memory. In state 12 the controller increments the address in AR so that if another file operation is performed, the memory address used to fetch or store that file will be the next sequential octet to the one currently in use. If a load file memory request returns during the one clock cycle that the controller is in the Orange Wait state, the controller sets the Hold flag to mark that event for use in the Orange Request state. Since the request complete indication is dependent on a KCM to file transfer signal (KCMTFILE), the Hold flag is not set during store file operations. Store file multiple does not need a memory response to continue (except WACK from CMR). At the end of one clock cycle the controller exits to the Orange Request state.

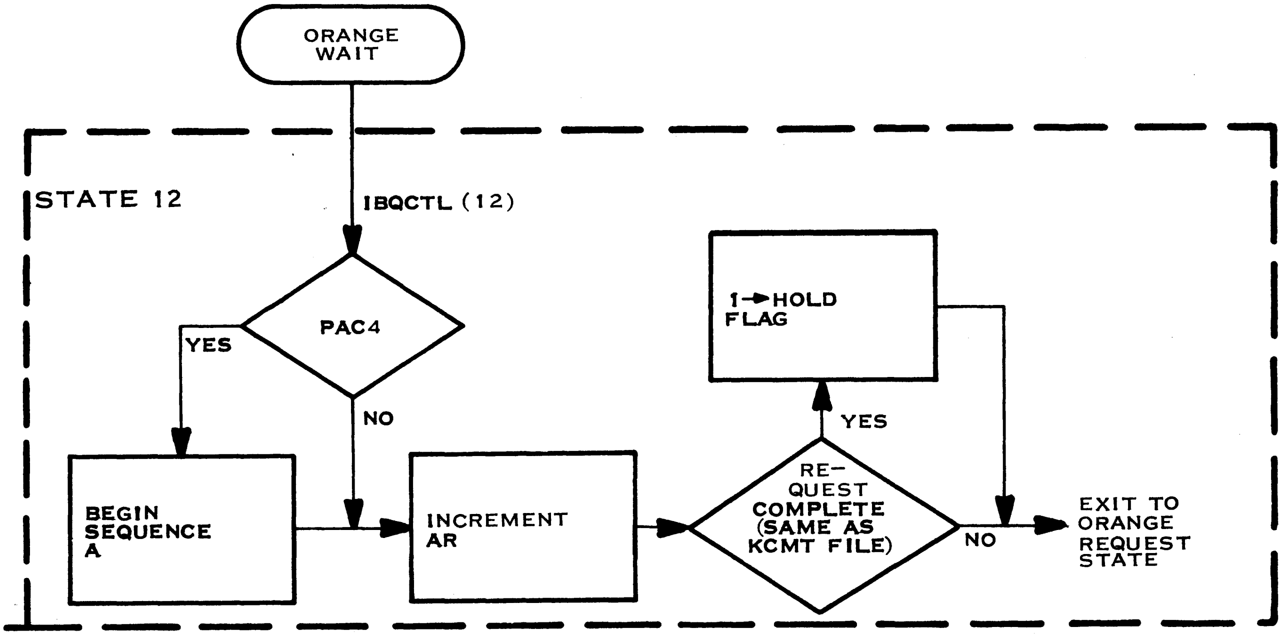
4-270 ORANGE REQUEST (STATE 13)

The controller enters the Orange Request state from the Orange Wait state. In the Orange Request state the controller generates load or store file requests



(A) 124872

Figure 4-48. Indirect Request State of Level 3 Controller

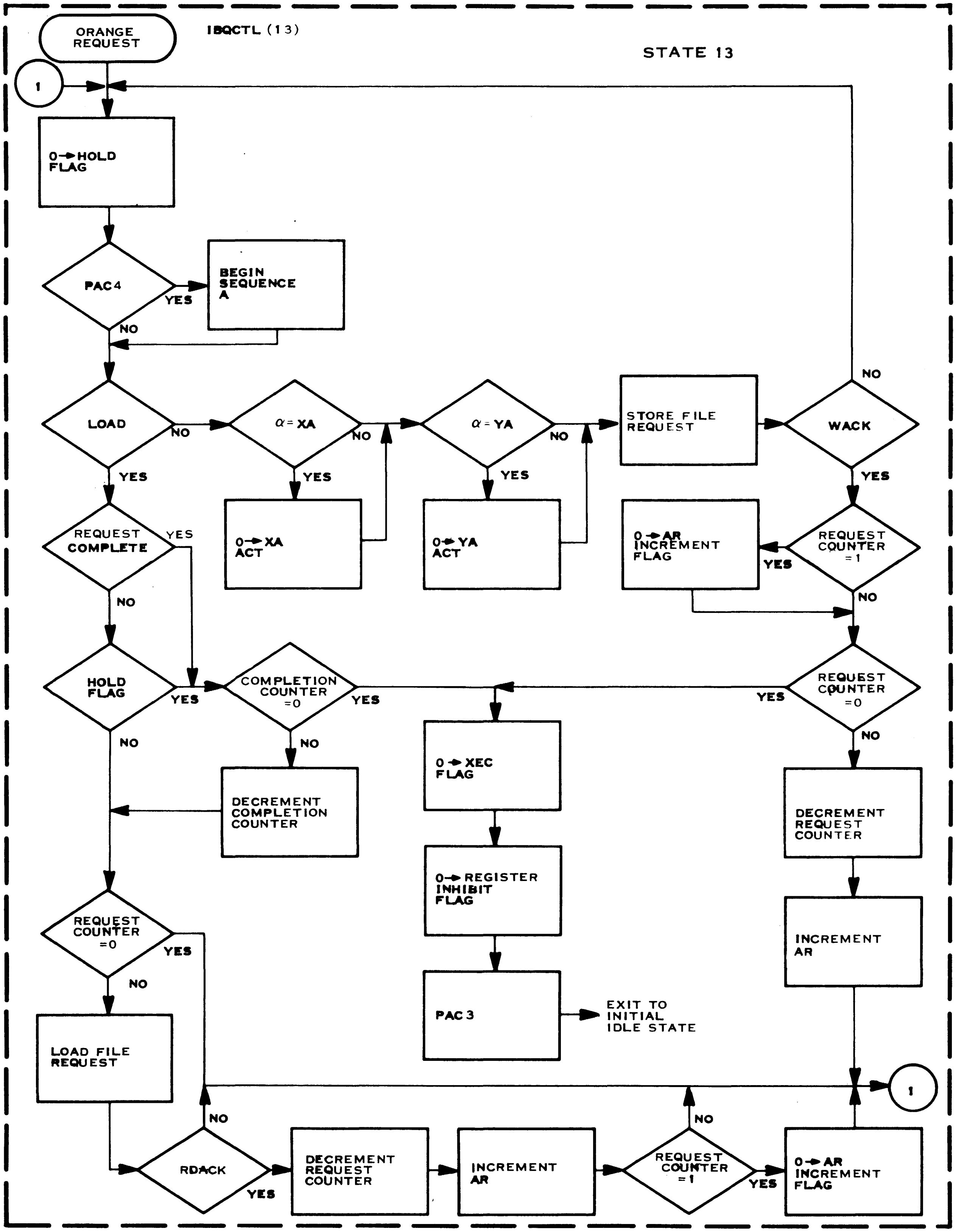


(B) 124873

Figure 4-49. Orange Wait State of Level 3 Controller

to complete both load and store file multiple operations, and also performs the terminating steps in a single load file operation after the octet has been loaded from memory into the designated register file location. The controller logic flow, illustrated in figure 4-50, is divided into two sequences; load file multiple and store file multiple. Single load file operations follow the load file multiple path. Single store file operations are completed entirely within the Orange subcycle of the Idle state.

4-271 LOAD FILE MULTIPLE. The Completion and Request Counters track the status of a load file multiple operation in the Orange Request state. The Request counter is initially loaded with a count of five in the Orange subcycle of the Idle state, and decrements once for each request that is sent to memory while the controller is in the Orange Request state. When the Request counter reaches a count of zero, six requests have been sent to memory (one in Orange, five in Orange Request) for octets to load the six register file octets. The Completion counter is initially loaded with a value of five in the Orange subcycle and decrements each time an octet returns from memory and is loaded into the register file. The fifth octet from memory decrements the counter from 1 to 0. After the sixth octet returns from memory, the controller checks the counter, determines that it is zero, and performs the exit steps to terminate the operation. Single file operations will have an initial count of zero in the Completion counter so that when the first octet returns from memory, the controller terminates the operation. Notice that if the Hold flag was set in the Orange Wait state, the controller clears that flag on the next clock (after having used it to indicate Request Complete). Therefore, the Hold flag will affect the logic flow only during the first cycle through the Orange Request state. During the load file operation when the Request counter



(B) 114337A

Figure 4-50. Orange Request State of Level 3 Controller

reaches a count of one, the controller clears the AR Increment flag to prevent the last request cycle from incrementing the address in AR. When the controller terminates the operation it clears the Execute and Register Inhibit flags, issues PAC3 to the level 2 controller, and exits to the Initial Idle state.

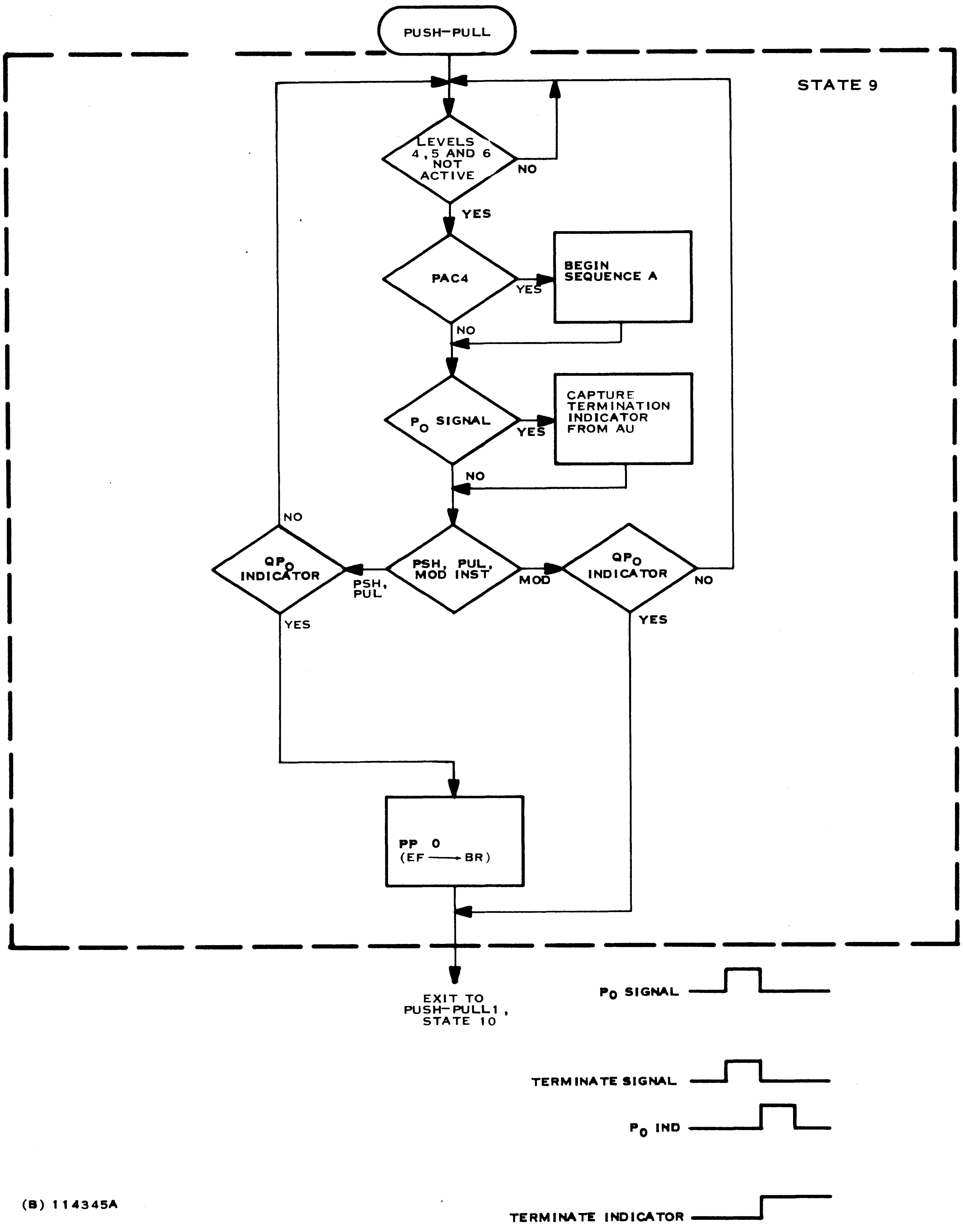
4-272 STORE FILE MULTIPLE. The Request counter is the only counter used to track the progress of a store file multiple since the IPU does not have to wait for returning data from memory for a store operation as it does during load operations. This counter is loaded with a value of 4 in the Orange subcycle and is decremented for each store file request generated in the Orange Request state. The fifth request to memory (one in Orange, four in Orange Request) decrements the counter from 1 to 0. During the following control cycle, the controller issues a request to store the sixth octet, determines that the counter is zero, and terminates the operation. Before exiting to the Initial Idle state, the controller clears the Execute and Register Inhibit flags and issues PAC3 to the level 2 controller. During each successive cycle through the Orange Request state, the controller compares the storage destination address (alpha) with the address of the octets in the X and Y buffers in the MBU (XA or YA). If one of the octets is stored into the location that the MBU octets were drawn from, the controller clears the associated activity bit to prevent succeeding instructions in the IPU from using the data in the changed buffer without refetching that data from memory.

4-273 PUSH-PULL (STATE 9)

The controller enters state 9 (figure 4-51) from the Lavender subcycle of the Idle state. In Lavender the controller loaded the stack parameters into the MBU and started them down the pipe for testing to see if the desired stack instruction will overflow the prescribed boundaries of the stack area in memory. The AU performs this comparison. In state 9 the controller waits for levels 4, 5 and 6 to become inactive, indicating that the stack parameters have entered the AU for testing (the entire parameter doubleword is processed at one time). The controller then loops within state 9 until two AU control ROM signals appear. The first signal, P_0, indicates that the comparison in the AU is complete. If the test of the parameters with respect to the stack instruction caused the result to overflow the boundaries of the stack (word count or space count went negative), the AU transmits a termination bit indicating that the instruction should be terminated and not executed. When P_0 appears, the controller enables the termination bit to set the Terminate Indicator in the IPU. The second AU signal is P_0 Indicator. This signal designates that the pointer for the stack has been modified, decremented or untouched, and is ready in the output stage of the AU. For push and pull instructions, the controller then enables the transfer of that pointer from the EF register to the BR register in level 2 of the IPU. Regardless of the instruction type, the P_0 Indicator moves the controller into the Push-Pull 1 state.

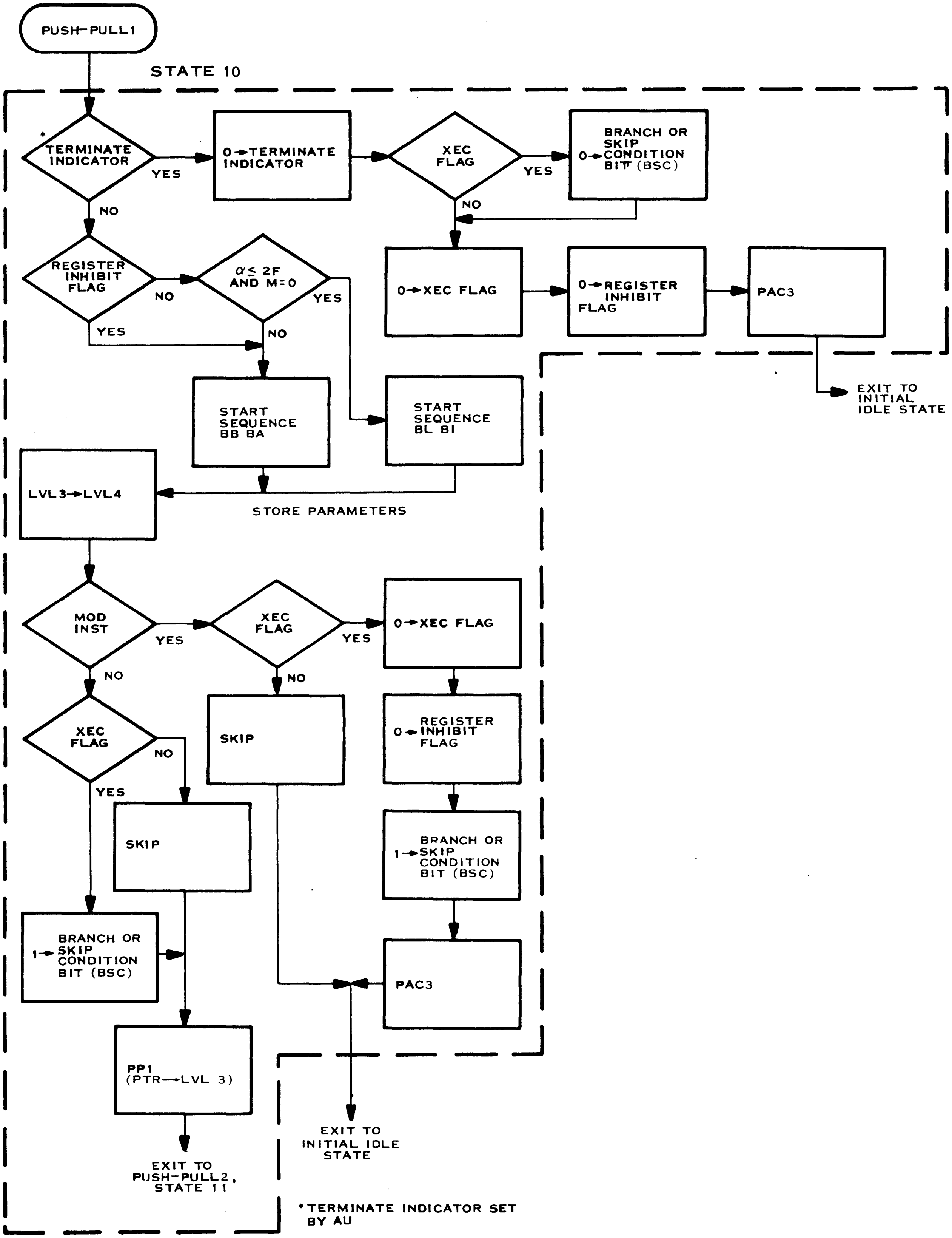
4-274 PUSH-PULL 1 (STATE 10)

The controller enters state 10 (figure 4-52) from state 9 after testing the parameters of a stack instruction to determine if the desired operation will produce an overflow of the designated memory area for the stack. If an overflow condition was detected, the Terminate Indicator will be set when the controller enters state 10. The controller then clears the Terminate Indicator,



(B) 114345A

Figure 4-51. Push-Pull State of Level 3 Controller



(B) 124874

Figure 4-52. Push-Pull 1 State of Level 3 Controller

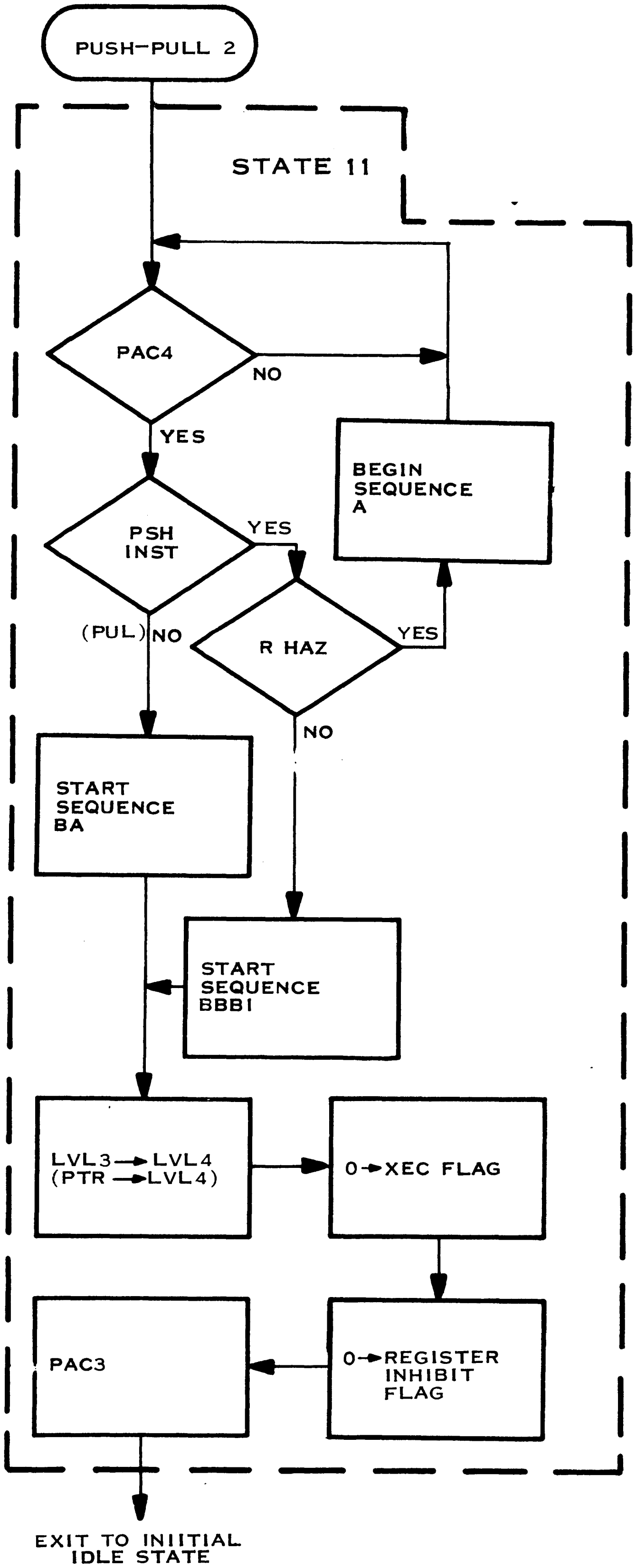
clears the Execute and Register Inhibit flags, and issues PAC3 to the level 2 controller. If the instruction was part of an execute operation, the controller also clears the Branch of Skip Condition bit to indicate that no valid branch or skip was encountered in processing the instruction.

If the Terminate Indicator was not set, the controller initiates a sequence (BBBA or BLBI), to load the parameters into the MBU a second time, pass them through the AU for the required modification, and store them into central memory. Enabling transfer of level 3 to level 4 during the next clock pulse starts the parameters down the pipe for storage in memory. If the instruction is a modification, this step is all that is required. The AU will modify the parameters and store them into memory using the contents of R0 at level 4 to modify the parameters. The controller then determines if the instruction is part of an execute sequence. If it is, the controller clears the Execute flag and Register Inhibit flag, sets the Branch or Skip Condition bit to indicate to the PP that a skip condition was encountered but not taken due to the execute, and issues PAC3 to the level 2 controller. If the Execute flag was not set, the controller issues a Skip signal so that the next instruction in the sequence will be skipped. The next instruction is typically an instruction that enables the program to recover if the parameter test produces an overflow condition. This instruction is therefore not needed if the Terminate Indicator was not set. The controller exits to the Initial Idle state for the next control cycle.

If the instruction was not a modify instruction, a further transfer through the pipe is required to store into or read from the stack. The controller inspects the Execute flag. If this flag is set, the controller sets the Branch or Skip Condition bit; if it is not set, the controller issues the Skip signal. In either case, the controller enables the next clock pulse to transfer the pointer from level 2 to level 3 to supply the storage address for the Push or Pull operation. The controller then exits to the Push-Pull 2 state to complete the process.

4-275 PUSH-PULL 2 (STATE 11)

State 11 (figure 4-53) completes the level 3 controller's responsibilities for Push or Pull instructions by initiating a store into the stack area of central memory (Push) or a load from the stack area of central memory (Pull). When the controller enters state 11, it waits for PAC4 from the level 4 controller so that level 4 will be vacant to receive the operation. If the instruction is a Pull, the controller initiates a sequence (BA) to load that value from the stack location indicated by the pointer into the MBU. If the instruction is a Push, the controller waits for any hazards to the register file location indicated by the R-field to clear before initiating a sequence (BBBI) to store that value into the stack location indicated by the pointer. The controller then enables the next clock pulse to transfer the instruction at level 3 to level 4, and clear the Execute and Register Inhibit flags. The controller issues PAC3 to the level 2 controller before exiting to the Initial Idle state for the start of the next control cycle.



(B) 124875

Figure 4-53. Push-Pull 2 State of Level 3 Controller

4-276 MONITOR CALLS (STATE 16)

The controller enters state 16 (figure 4-54) from the Brown subcycle of the Idle state when it determines that the instruction at level 3 of the IPU is either a Monitor Call and Proceed (MCP) or a Monitor Call and Wait (MCW). Either of these instructions require the CP to issue a call for further information to the PP, store a pointer in a predetermined location in memory, and either proceed with the program sequence (MCP) or stop the sequence, performing a context switch, until the information returns from the PP (MCW). The level 3 controller makes two cycles through the Monitor Calls state. The Hold flag keeps track of which cycle is being done.

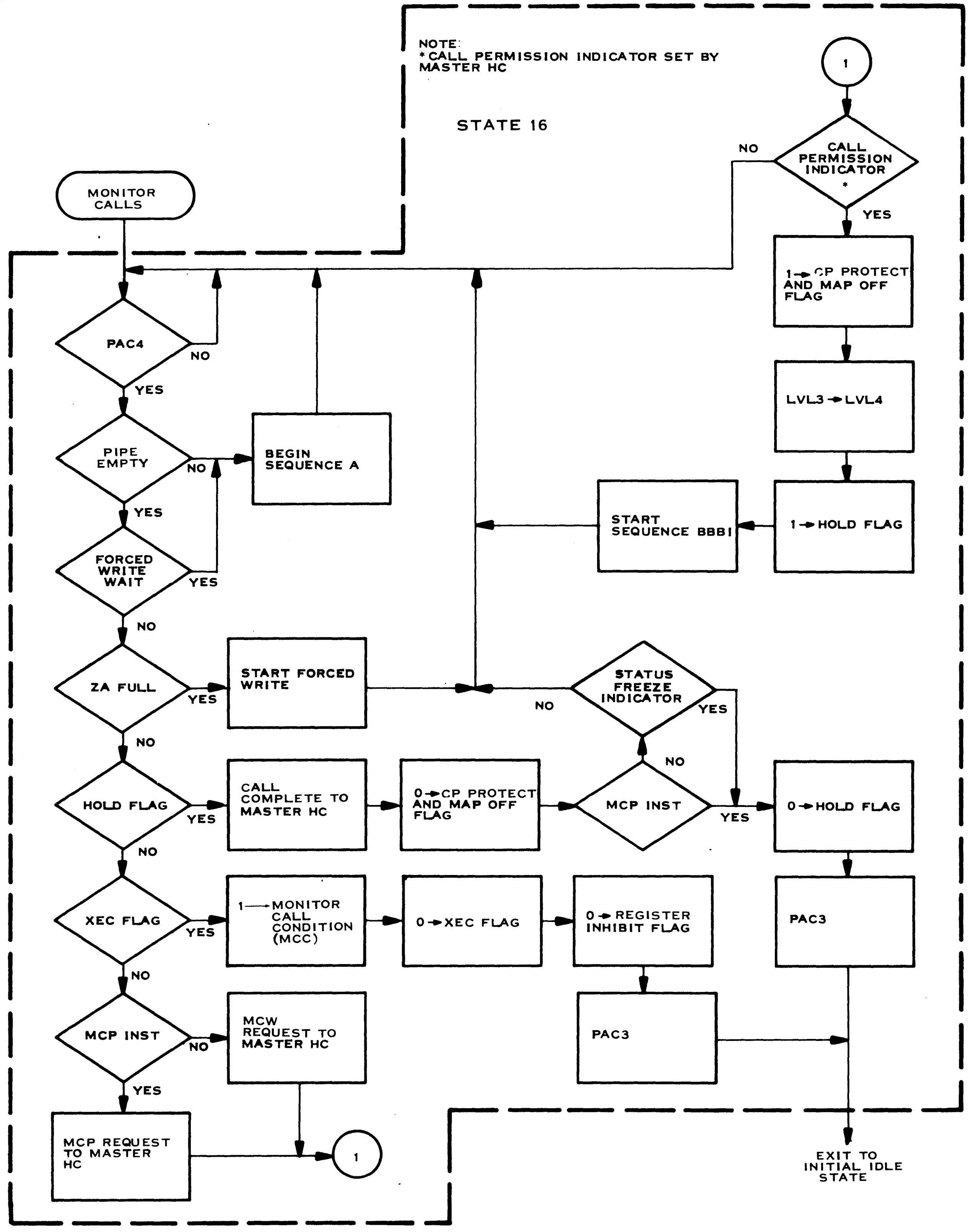
When the controller enters state 16, it ensures that the level 4 controller can accept a new instruction, that the pipe is empty, no forced write operation is in progress, and that no valid data is left in the Z buffer of the MBU for storage into memory (ZA not full). The first time through state 16, the Hold flag is not set. If the monitor call is part of an execute sequence, the call will not be performed. Instead the controller indicates that the call instruction was encountered by setting the monitor call condition (MCC) bit in the program status doubleword, clears the Execute and Register Inhibit flags, issues PAC3 and returns to the Initial Idle state. If the Execute flag is not set, the controller issues either a MCP or a MCW request to Master Hard Core and inspects the Call Permission indicator. If this bit is not set, the controller loops within state 16 until it sets. When Call Permission is on, the controller inhibits the map and protect parameters of the CP in the MCU by setting the CP Protect and Map Off flag. The controller also sets the Hold flag, initiates a sequence to store the pointer into memory, and enables a level 3 to level 4 transfer to start the store into the pipe. The controller then returns to the start of Monitor Calls for the next control cycle.

During the second pass through state 16, the controller waits until the pointer has cleared the pipe and has been stored into memory. Since the Hold flag is now set, the controller issues Call Complete to Master Hard Core, clears the CP Protect and Map Off flag, clears the Hold flag and issues PAC3 to the level 2 controller. If the instruction is a MCW, the controller must wait until the Status Freeze indicator sets, indicating the start of the context switch, before clearing the Hold flag, producing PAC3, and exiting to the Initial Idle state.

4-277 LEVEL 3 CONTROLLER COMMON SEQUENCES

To avoid extensive duplication in the level 3 controller flowcharts, several common logic sequences are merely referenced by an action block, i.e., "Start Sequence BA", etc. These sequences perform the steps within the controller to prepare the IPU to either load the MBU with a value, or store a value into memory or the register file by sending it down the pipe. Figure 4-55 illustrates the logic flow within the controller for each of these sequences. The following paragraphs describe the function performed by each sequence as well as some of the flags and control signals that appear in the flowchart.

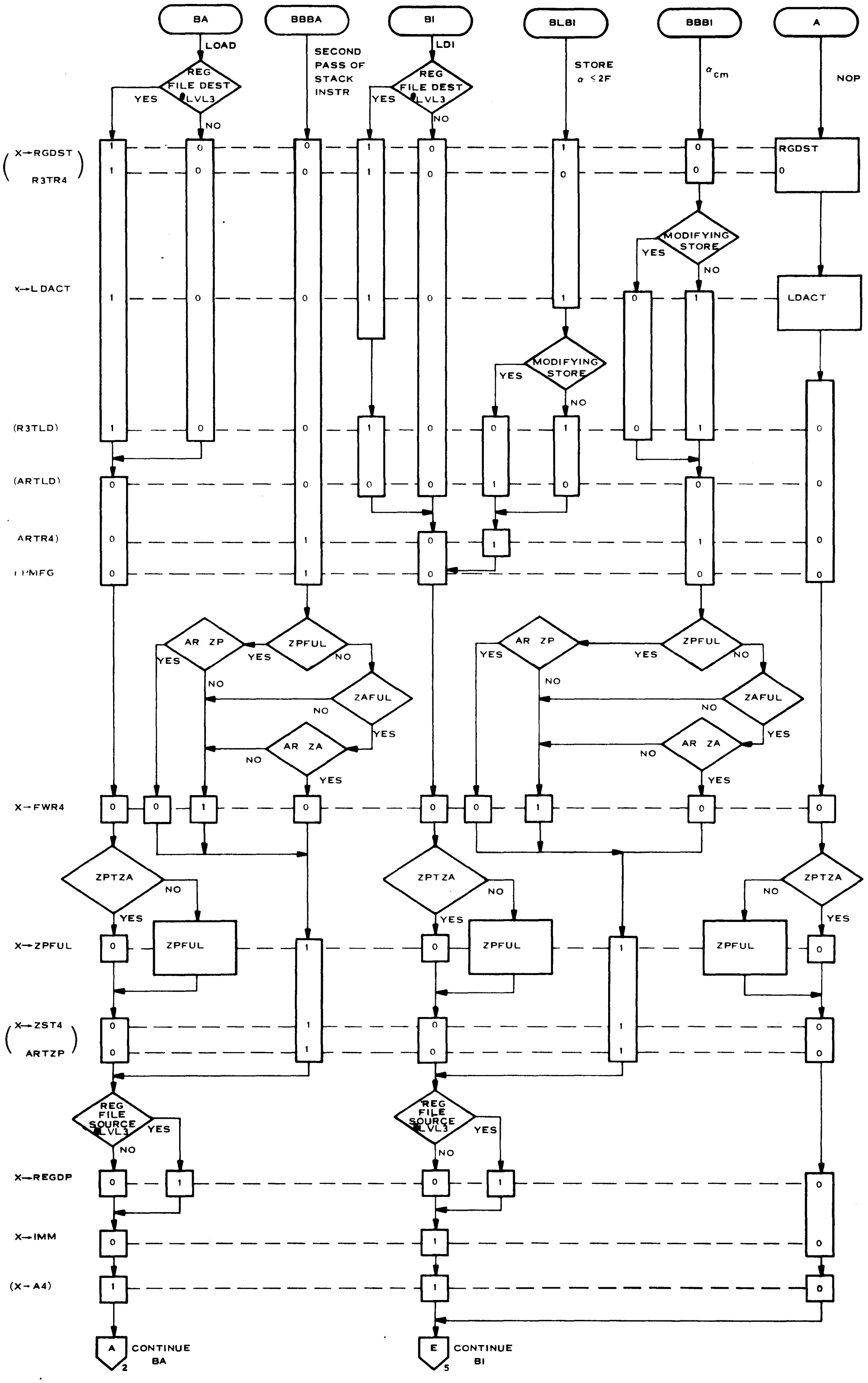
4-278 SEQUENCE BA. BA sets IPU conditions so that a value from central memory can be loaded into the MBU through the MBU's interface with memory (SC file to X or Y buffers). When the controller starts BA, it determines if the



(B) 124876

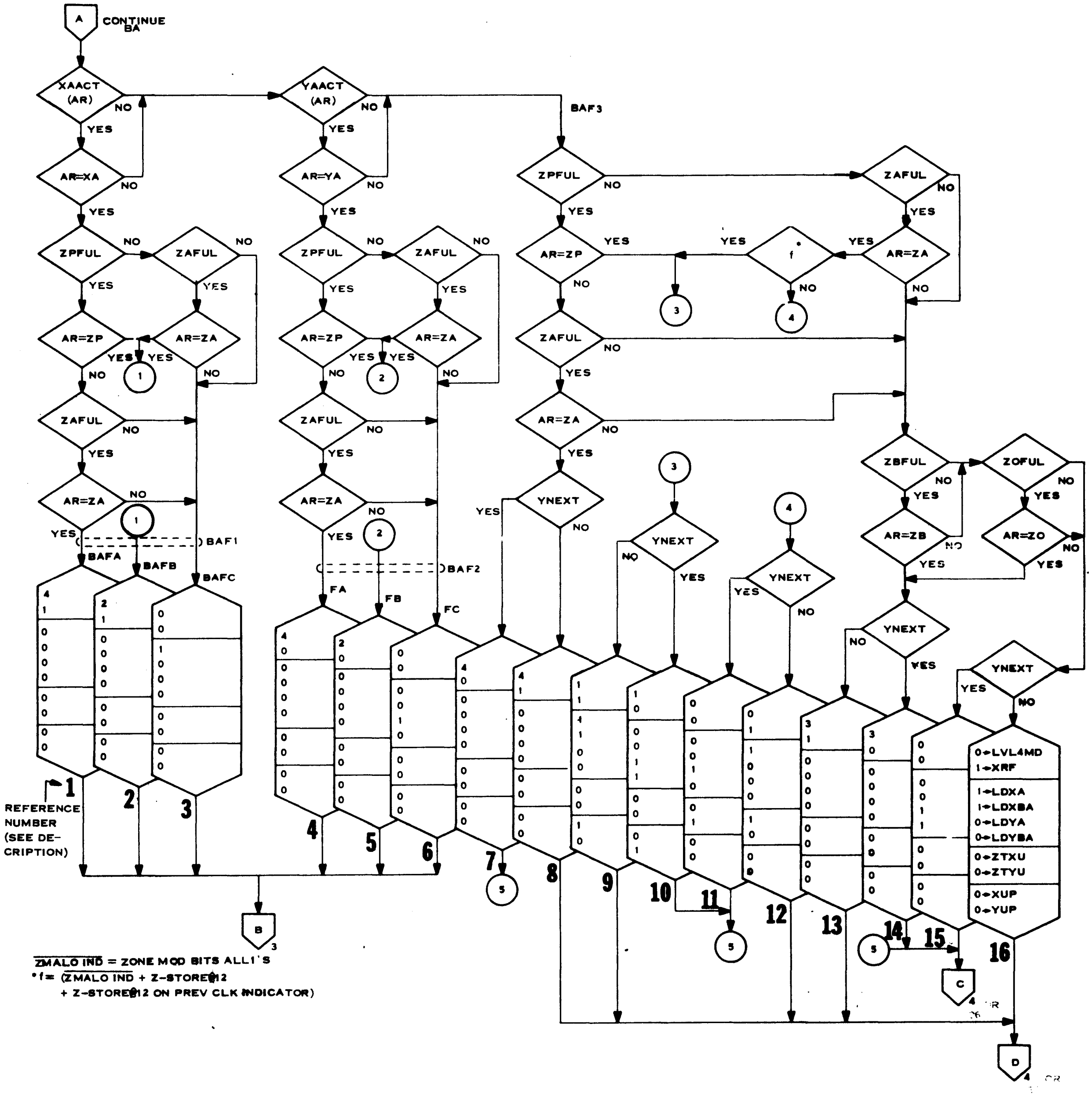
Figure 4-54. Monitor Calls State of Level 3 Controller

LEVEL 3 CONTROLLER COMMON SEQUENCES



(C)119479A

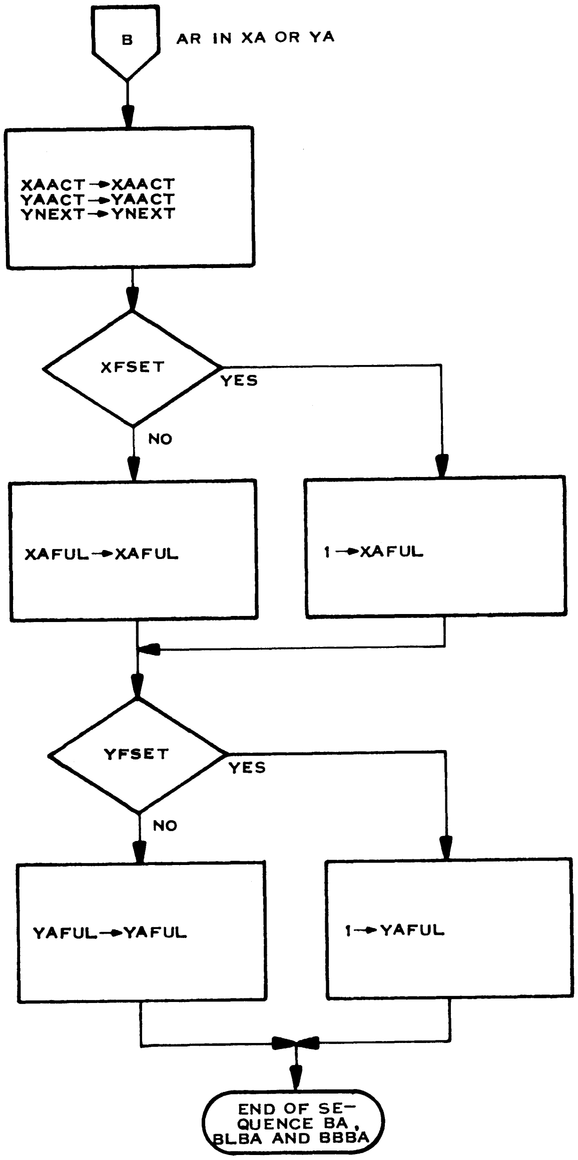
Figure 4-55. Level 3 Controller Common Sequences (Sheet 1 of 5)



ZMALO IND = ZONE MOD BITS ALL1'S

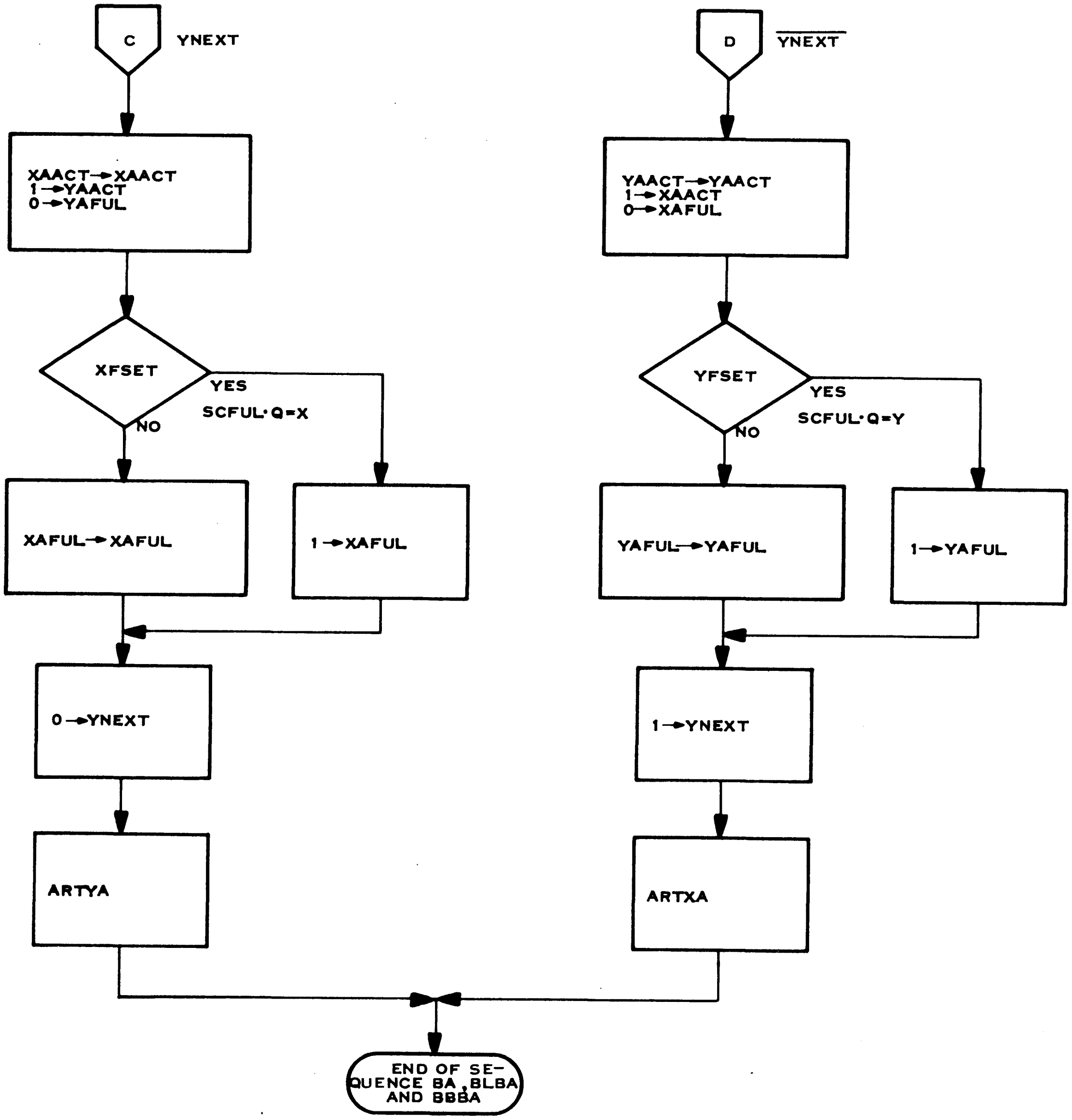
*f= (ZMALO IND + Z-STORE@12
+ Z-STORE@12 ON PREV CLK INDICATOR)

Figure 4-55. Level 3 Controller Common Sequences (Sheet 2 of 5)



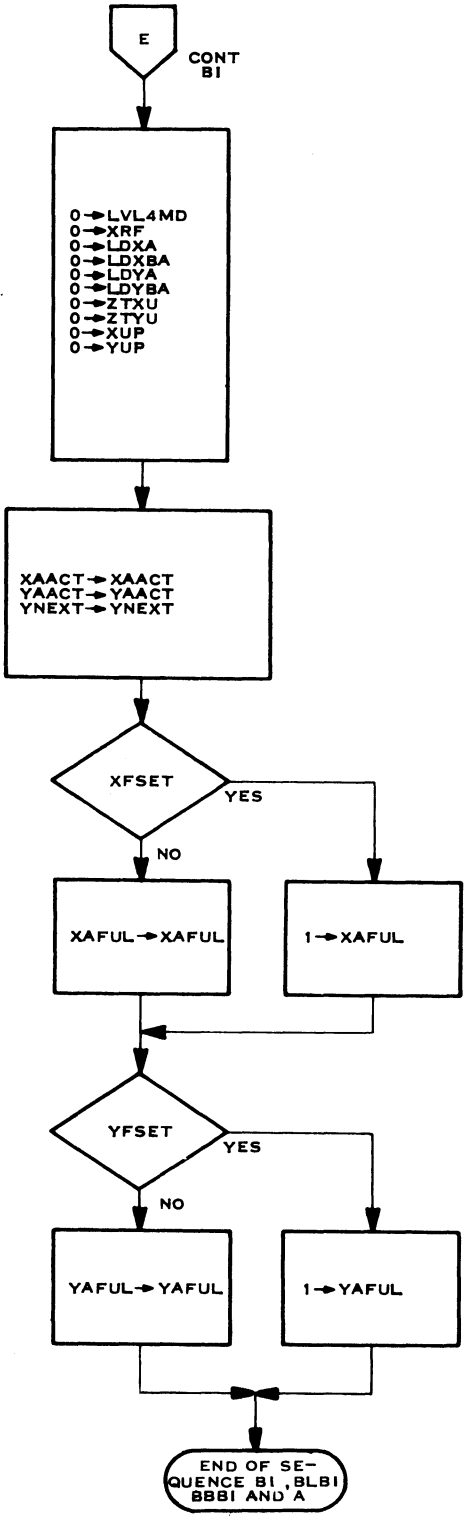
(B) 119482A

Figure 4-55. Level 3 Controller Common Sequences (Sheet 3 of 5)



(B) 119484A

Figure 4-55. Level 3 Controller Common Sequences (Sheet 4 of 5)



(B) 119481A

Figure 4-55. Level 3 Controller Common Sequences (Sheet 5 of 5)

instruction at level 3 specifies a storage area in the register file for the resultant of the operation. If it indicates a register file destination, the IPU must track that operation through the pipe to detect register hazard conditions. Therefore, the controller sets the Register Destination flag (IRQRGDST), enables the next clock to transfer the storage address from R3 to R4 to pass through the register stack and to LD to indicate the last destination address that was sent down the pipe. To indicate that the LD address is valid, the controller also sets LD Active. If the storage address is in central memory, the controller ignores the previous steps, clearing all flags and gates described above to zero. In either case, the controller sets Register Data Present if the data to be loaded is from the register file, sets level 4 activity bit (A4), and proceeds to the Continue BA flowchart. Since that portion of the sequence is shared with the BBBA sequence, Continue BA is described separately.

4-279 SEQUENCE BBBA. The controller uses sequence BBBA during the second pass of a stack instruction to load stack parameters into the MBU, start them down the pipe and store them back into memory with the same instruction. The first portion of BBBA prepares the pipe for the store portion of the operation. The sequence then joins sequence BA (refer to Continue BA) to establish conditions for the load operation.

When the controller begins BBBA it loads the destination address for the store portion of the operation into the register stack (ARTR4) so that the IPU can track the progress of the store operation as it proceeds down the pipe. The controller also sets a bit that indicates the store is part of a stack instruction, so that the parameters will be altered to reflect the new status of the stack as they pass through the AU (PPMFG). The controller then determines if the parameters will be stored into an octet currently in the MBU storage buffers or the pipe (ZA or ZP). If the parameters are to be stored into a new octet, the controller sets First Word in R4 (FWR4) to indicate the start of a new octet. This bit is part of the register stack and is passed down through the register stack as the store proceeds through the pipe. The controller also enables transfer of the storage address from AR to the ZP register, and sets ZPFUL to indicate a valid storage address in the ZP register. The Z Store at level 4 bit is also set to tag the operation as a memory store as it progresses through the pipe. The controller then sets level 4 activity bit and joins sequence BA to prepare for the load portion of the operation.

4-280 CONTINUE BA. This portion of the sequence examines the address contained the AR register, compares it with the octet addresses of files within the MBU and AU, and generates the signals that load the octet specified by AR into the selected buffer (X or Y) in the MBU and select the word from that octet for loading into the pipe. Continue BA recognizes sixteen possible relationships between the address in AR, the other addresses in the CP, and the selection of X or Y buffer. These sixteen combinations are illustrated in the common sequences flowchart. Table 4-7 summarizes the decisions performed in this segment of logic. It includes a reference number that corresponds to a number along the output of each of the sixteen action blocks in the flowchart. For each reference number the table lists the relationship of the AR address to the CP addresses, other conditions pertinent to the routing decision, the level 4 mode code sent to the level 4 controller, and those actions that must

Table 4-7. Continue BA Decode Results

Flowchart Reference Number	AR Address Contained IN	Other Conditions	Level 4 Mode (LUL4MO)	Required Action 3
1	XA and ZA	New Octet in Pipe	4	Wait for ZA→CM, fetch (AR) to X, Load (AR) from X
2	XA and ZP or ZA	Only one Active Octet in Pipe	2	Fetch (AR), update buffer (Z→X)
3	XA	-	0	Load (AR) from X buffer
4	YA and ZA	New Octet in Pipe	4	Wait for ZA→CM, fetch (AR) to Y, Load (AR) from Y
5	YA and ZP or ZA	Only one Active Octet in Pipe	2	Fetch (AR) to Y, update buffer (Z→Y)
6	YA	-	0	Load (AR) from Y buffer
7	ZA	New Octet in Pipe, Y Buffer Next	4	Wait for ZA→CM, fetch (AR) to Y, Load (AR) from Y
8	ZA	New Octet in Pipe, X Buffer Next	4	Wait for ZA→CM, fetch (AR) to X, Load (AR) from X
9	ZP or ZA	ZA is not a complete octet; X Buffer Next	1	Fetch (AR) to X, update buffer (Z→X), Load (AR) from X
10	ZP or ZA	ZA not complete, Y Buffer Next	1	Fetch (AR) to Y, update buffer (Z→Y), Load (AR) from Y
11	ZA	ZA complete, Y Buffer Next	0	Update Buffer (Z→Y), Load (AR) from Y
12	ZA	ZA complete, X Buffer Next	0	Update Buffer (Z→X), Load (AR) from X
13	ZB or ZO	X Buffer Next	3	Wait for data in memory, fetch (AR) to X, Load (AR) from X
14	ZB or ZO	Y Buffer Next	3	Wait for data in memory, fetch (AR) to Y, Load (AR) from Y
15	CM	Y Buffer Next	0	Fetch (AR) to Y, Load (AR) from Y
16	CM	X Buffer Next	0	Fetch (AR) to X, Load (AR) from X

NOTES: 1. AR is an address in the AR Register

2. (AR) is the contents of location AR

3. Some action items performed by Level 4 controller.

be performed to load the desired values into the pipe. To help understand the signals generated within this control segment, table 4-8 lists the acronyms used in the flowchart and their significance in the CP.

Table 4-8. Continue BA Acronyms

Acronym	Function
f	When f is not true, the contents of the Z buffer can be transferred in total for an update; i.e., the Z buffer octet contains eight complete words.
LDXA	Load XA - loads an operand from the current octet in the X buffer (MBU) into the pipe.
LDXBA	Load XBA - loads an operand from memory into the X buffer.
LDYA	Load YA - loads an operand from the current octet in the Y buffer (MBU) into the pipe.
LDYBA	Load YBA - loads an operand from memory into the Y buffer.
LVL4MD	Level 4 Mode - a two bit code sent to the level 4 controller that enables portions of the level 4 control logic relative to the conditions in the level 3 controller during the previous control cycle.
XRF	XR Flag - When set, indicates that the next octet from memory will be loaded into the X buffer; when clear, indicates that the next octet will be loaded into the Y buffer. Complement of YNEXT.
XUP	X Update required - Set to prevent the MBU from using an octet from memory until that octet has been updated by ZTXU.
YNEXT	Y buffer next - When set, indicates that the next octet from memory will be loaded into the Y buffer; when clear, indicates that the next octet will be loaded into the X buffer. Complement of XRF.
YUP	Y Update required - Set to prevent the MBU from using an octet from memory when it arrives in the Y buffer until that octet has been updated by a ZTYU.
ZTXU	Z to X Update - Set to enable the next clock to transfer the half words in the Z buffer whose zone modification bits are set into the corresponding half word locations in the X buffer.
ZTYU	Z to Y Update - Set to enable the next clock to transfer the half words in the Z buffer whose zone modification bits are set into the corresponding half word locations in the Y buffer.

4-281 END BA. Sequence BA has three possible termination paths depending upon the outcome of the decoding circuit in the Continue BA portion of BA. These termination paths are illustrated on sheets 3 and 4 of figure 4-55, the common sequences flowchart. If Continue BA discovered that the address in AR was contained in either XA or YA (reference numbers 1 through 6 on the Continue BA flowchart), the controller terminates the sequence with the series of events on sheet 3. Since no new octets are to be introduced into the MBU, the controller ensures that the activity bits for the XA and YA registers remain the same as they were before the sequence began. Similarly, YNEXT remains the same. The controller then determines if either X or Y will receive a new octet during the next clock (XFSET or YFSET). If either octet will receive new data, the controller enables the next clock to set the corresponding address full flip-flop (XAFUL or YAFUL) to indicate that the address in XA or YA references an octet that is now in the MBU. The controller then returns to the original control cycle.

If YNEXT was set when examined during Continue BA (reference numbers 7, 10, 11, 14 and 15), the controller terminates the sequence by setting YA activity and clearing YA full. These bits then indicate that an octet has been ordered from memory for the Y buffer but is not yet in the MBU. If the next clock will transfer a new octet into the X buffer, the controller enables the next clock to also set XAFUL to indicate that a valid octet corresponding to the address in XA is in the MBU. The controller then clears YNEXT so that the next octet ordered from memory will be routed to the X buffer, and transfers the address of the desired word from the AR register to the YA register (ARTYA).

If YNEXT was not set when examined during Continue BA (reference numbers 8, 9, 12, 13 and 16), the controller terminates the sequence by setting XA activity and clearing XA full. These bits then indicate that an octet has been ordered from memory for the X buffer, but is not yet in the MBU. If the next clock will transfer a new octet into the Y buffer, the controller sets YAFUL to indicate that a valid octet corresponding to the address in YA is in the MBU. The controller then sets YNEXT so that the next octet ordered from memory will be routed to the Y buffer, and transfers the address of the desired word from the AR register to the XA register (ARTXA).

4-282 SEQUENCE BLBI. BLBI establishes conditions in the IPU so that a value from the register file can be stored into another register file location. When the controller enters this sequence it sets RGDST to indicate that the store has a register file destination. The controller then sets LDACT, and depending upon the type of store operation, enables the next clock to transfer the contents of either R3 or AR into the LD register. LD is used to determine short circuit paths for the next instruction that enters level 3. If the store operation is a simple store that does not change the contents of the register being stored, then the register address contained in R3 is the address of that value when it reaches the AU Output stage where it can be selected for a short circuit path. The controller therefore enables that address to LD (R3TLD). If, however, the store operation will also modify the storage quantity (Store Negative, Store Ones Complement, etc.), the register address in R3 does not define the address of the value when it reaches the AU Output stage. Instead, the destination address of the store, contained in AR, must be used as the address of that value when it reaches the AU Output stage. Therefore, the controller enables ARTLD. In either case, the destination address in AR is loaded into R4 to be passed through the register stack for register hazard comparisons with subsequent instructions. The controller then sets Register Data Present since the value to be stored originated in the register file, sets IMM to indicate that an immediate operand value is to be transferred, and sets level 4 activity bit on the next clock. The controller then completes the sequence through the Continue BI portion. This flowchart portion is described separately since it is common to four sequences.

4-283 SEQUENCE BI. BI sets IPU conditions so that a value from the register file or an immediate operand can be loaded into the MBU through level 4 of the IPU. When the controller starts BI, it determines if the instruction at level 3 specifies a storage area in the register file for the resultant of the operation. If it does indicate a register file destination, the IPU must track that operation through the pipe to detect register hazard conditions. Therefore, the controller sets the Register Destination flag (IRQRGDST), enables the next clock to transfer the storage address from R3 to R4 for passing through the register stack and to LD to indicate the last destination address that was sent down the pipe for the register file. To indicate that this is a valid address, the controller also sets LD Active. If the storage address is in central memory, the controller ignores the previous steps, clearing all flags and gates described. In either case, the controller sets Register Data Present if data is taken from the register file for the MBU. The controller also sets IMM to indicate that an immediate operand value is to be transferred, and sets level 4 activity bit on the next clock. The controller then completes the sequence through the Continue BI portion. This flowchart portion is described separately since it is common to four sequences.

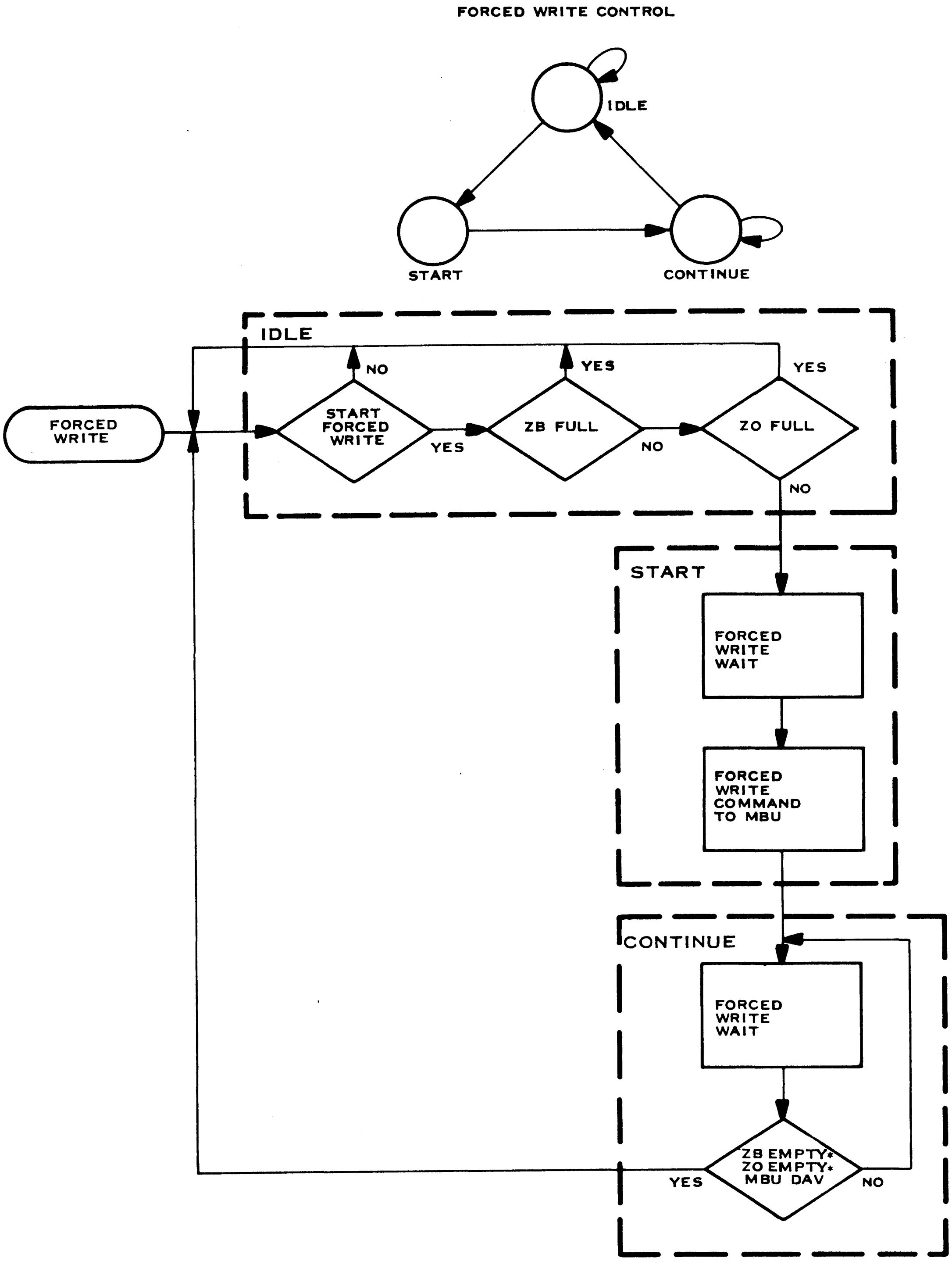
4-284 SEQUENCE BBBI. BBBI establishes conditions in the IPU so that a value from the register file can be stored into a location in central memory. When the controller enters this sequence, it clears the RGDST bit to indicate that the operation is intended for central memory. If the store operation is a simple store that does not modify the contents of the register being stored, the controller sets LDACT and enables the address of the register being stored to be transferred from R3 to LD for use in short circuit determination. If, however, the store operation will also modify the contents of the register being stored, the address in R3 will no longer represent the data that will eventually appear at the AU Output stage as a result of the store instruction. Consequently, the R3 address cannot be used for short circuit determination. In addition, since the address in AR is a central memory address, it is too large to fit into the 7 bit address space in LD. It too cannot be used to represent the address of the AU Output stage value. To perform a short circuit function for central memory stores, therefore, the level 3 controller must perform an X or Y update instead of a direct short circuit. In either case the controller enables the next clock to transfer the destination address from the AR register to R4 for transfer down the register stack. The controller then examines the addresses in ZA and ZP to determine if the store operation will be into an octet that is already present in the CP due to a previous instruction. If the store operation is for a new octet, the controller sets the First Word at R4 (FWR4) flag to indicate that the operation will be for the first word of a new octet. The controller also transfers the storage address from AR to the ZP register and sets ZPFUL to indicate a valid storage address in the ZP register. The Z Store at level 4 bit (ZST4) is also set to tag the operation as a memory store as it progresses through the pipe. Since the store operand originates in the register file, Register Data Present (REGDP) sets to indicate that the store involves data from the register file. The controller then sets IMM to indicate an immediate operand at level 3, and sets level 4 activity. The control cycle concludes with the Continue BI portion of the sequence. This portion is described separately since it is shared by four sequences.

4-285 SEQUENCE A. The controller enters sequence A when it returns to the idle state. This sequence performs no data transfers, but ensures that no transfers associated with the sequence charts will occur during that clock time. It is a No Operation sequence that sets all gating signals to zero, and maintains the existing flags in their present state. Sequence A joins BI in Continue BI to re-initialize the load gates.

4-286 CONTINUE BI. Continue BI provides the concluding portion of sequences BI, BLBI, BBBI and A. Since these sequences do not involve a load operation, Continue BI clears all load operation gates and enabling signals. The store operation of these sequences will not change the resident octets in the MBU, so the controller ensures that the buffer active and selection bits (XA Active, YA Active and YNEXT) do not change. The controller then determines if the next clock will load a new octet from memory into either the X or the Y buffer of the MBU (XFSET or YFSET). If either of these signals is true, the controller enables the next clock to set the XAFUL or YAFUL bit, indicating that the corresponding buffer contains a valid octet of data. The controller then returns to the control path that originated the sequence.

4-287 FORCED WRITE CONTROLLER

The forced write controller (figure 4-56), under direction of the level 3 controller, initiates an operation that stores the contents of the Z buffer in the MBU into memory. The forced write operation, therefore, effectively empties the pipe of currently executing octets. The forced write controller constantly waits in the Idle state until the level 3 controller issues a forced write command. The controller then waits until any MBU memory stores in progress have been sent to memory (ZB and ZO not full) before signaling level 3 controller to wait for the forced write to complete (Forced Write Wait) and sending a Forced Write command to the MBU. The controller then continues to hold Forced Write Wait high until the indicated store operation has been sent to central memory (ZB and ZO empty and MBU Data Available to MCU). The controller then returns to the Idle state to wait for another Forced Write command from level 3 controller.



(B) 119485A

Figure 4-56. Level 3 Forced Write Controller

4-288 LEVEL 4 CONTROLLER

The level 4 controller (figure 4-57) generates interface signals to the MBU to transfer the required operands into the MBU registers. The interface signals produced by the level 4 controller are the same as those controlled by the level 3 controller through the Continue BA common sequence. For a particular operation only one of the control circuits generates the required interface signals, depending upon the activity at level 4. Level 4 takes control if it contains an active instruction to be passed to the MBU and level 3 cannot proceed until level 4 completes its assignment (not PAC4). The signals generated by the level 4 controller are, however, dependent upon conditions in the IPU that were identified by the Continue BA sequence in the level 3 controller. These conditions are indicated by the level 4 mode (LVL4MD) bits from the level 3 controller.

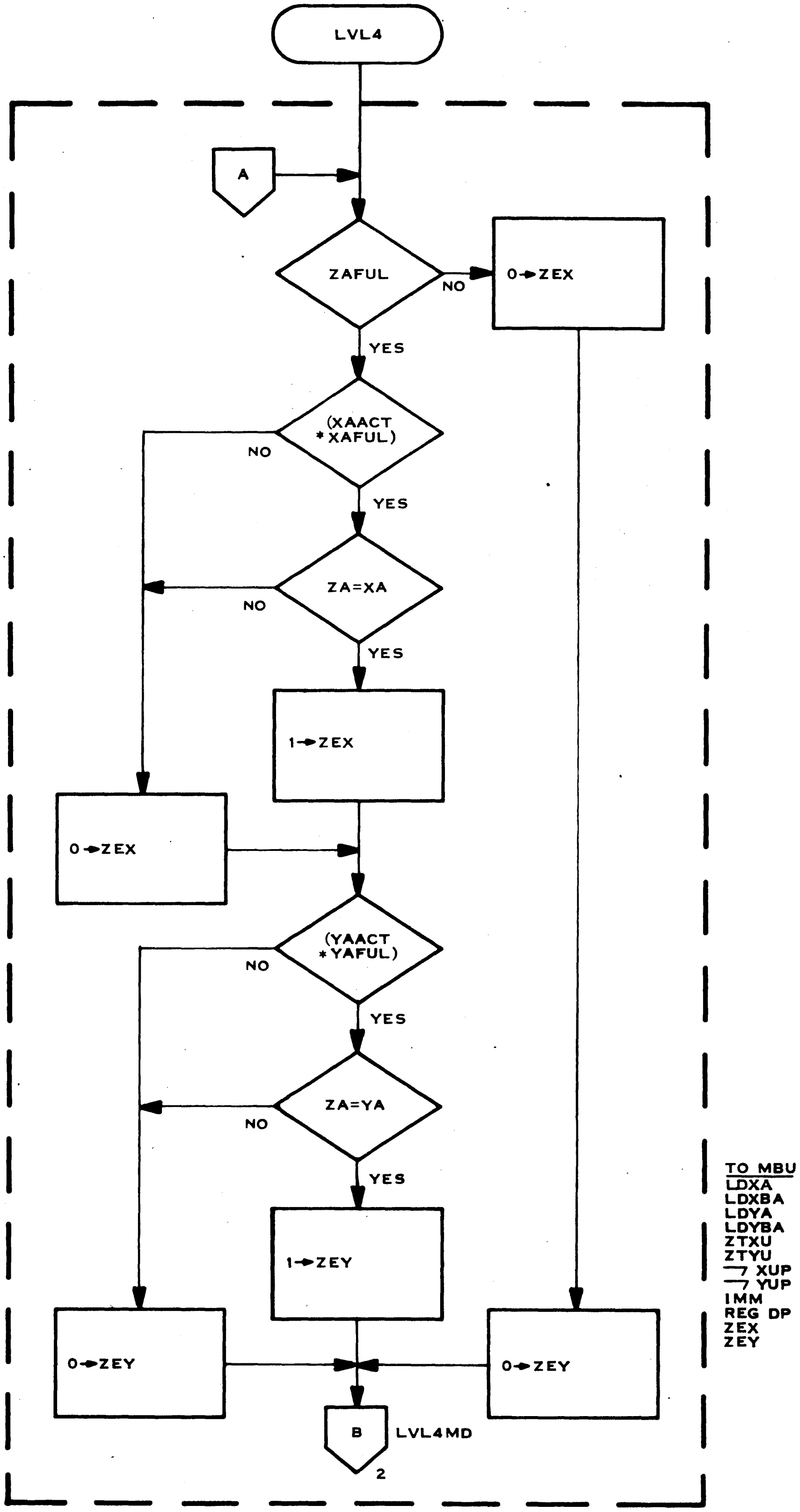
4-289 UPDATE ENABLE

At the start of the control cycle, the level 4 controller determines if a Z to X or a Z to Y update can be performed. If a valid octet is in the Z buffer (ZAFUL), the controller inspects the XA and YA registers. If either of these registers is both full and active, and contains the octet address of the octet in the Z buffer (ZA), then the controller sets the Z equal X (1 to ZEX) or the Z equal Y (1 to ZEY) indicators. These cells enable a Z to X update or a Z to Y update whenever the forced write controller forces the Z buffer to memory. An update operation transfers those halfwords that have been changed by an AU operation from the Z buffer to the selected operand buffer in the MBU. If neither condition is satisfied, the controller clears the two update enable bits. In either case, the controller inspects the level 4 mode bits from the level 3 controller to determine its next course of action.

4-290 MODE ZERO

Mode zero indicates to the level 4 controller that the level 3 controller has established conditions on the MBU interface such that the level 4 controller need perform no changes in the MBU interface bits. The level 4 controller need only transfer the instruction at level 4 to the MBU when conditions permit. In mode zero, the controller determines if the MBU will accept input from the IPU (PACMBI). If the MBU is not ready, and no active instruction is in level 4, the controller issues PAC4 to the level 3 controller to enable a new instruction into level 4, and returns to the start of the level 4 control cycle. If level 4 does contain an active instruction, the controller loops within the level 4 control cycle while maintaining the MBU interface bits in their preset condition until the MBU is ready to accept new input.

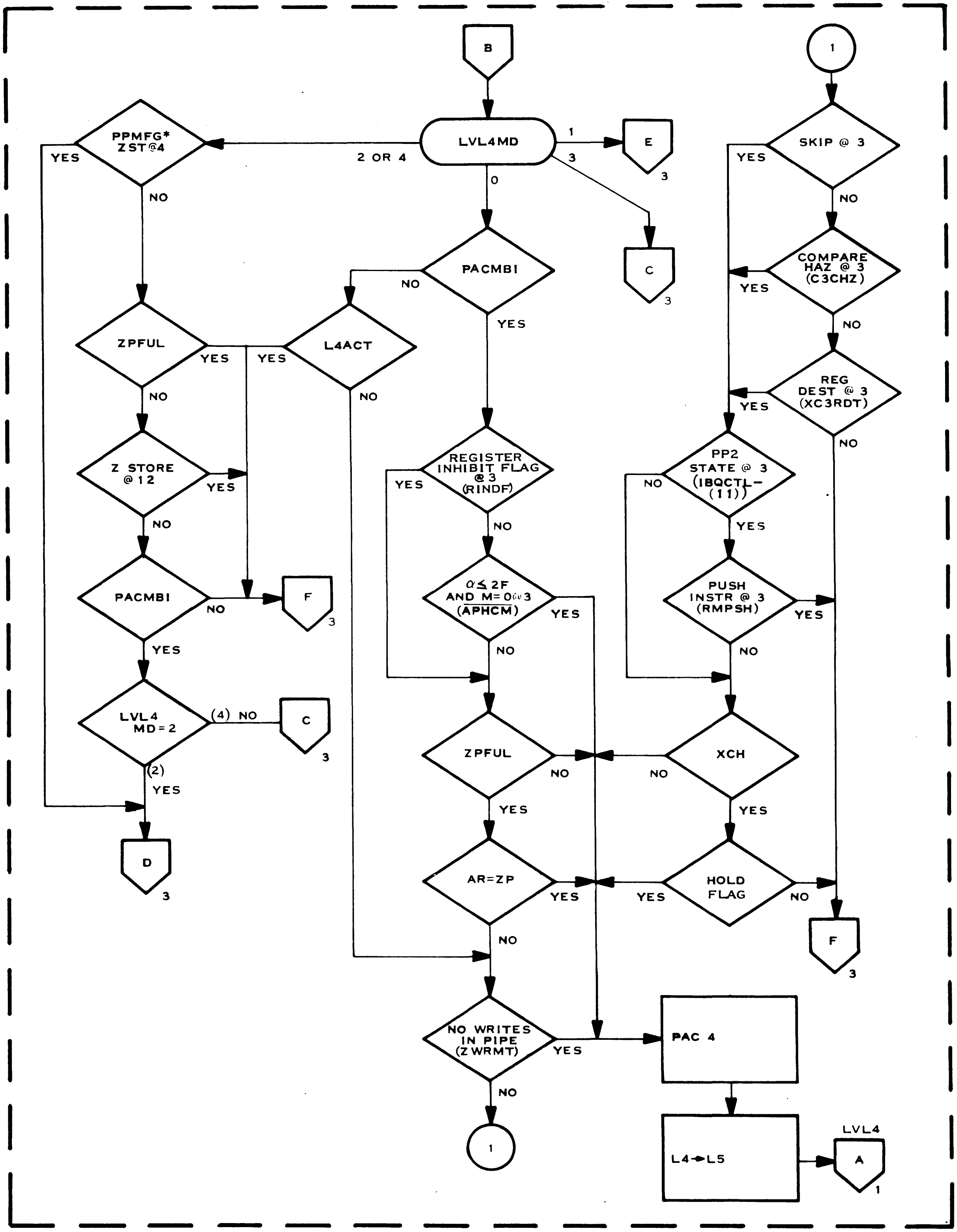
When the controller receives PACMBI, it examines conditions in the IPU to determine if the instruction can be transferred to the MBU. If the alpha operand is in the register file, the transfer will involve immediate operands only and the MBU interface with memory will not be required. The controller can, therefore, issue PAC4 and enable the transfer of level 4 to level 5 on the next clock. If the alpha operand is from memory, the controller determines if the level 4 instruction is a store by examining ZPFUL. A store from level 3 sets ZPFUL and places the store address in the ZP register when it transfers from level 3 to level 4. The controller determines if the storage



*
IF PAC4, THEN LVL3 CONTROLLER CONTROLS: LVL4MD, LDXA, LDXBA, LDYA, LDYBA, ZTXU, ZTYU, XUP, YUP. IF $\overline{\text{PAC4}}$, THEN LVL4 CONTROLLER CONTROLS THESE CELLS. DUAL INPUT CELLS ARE USED IN IMPLEMENTATION, PROVIDING INDEPENDENT STEERING EQUATIONS

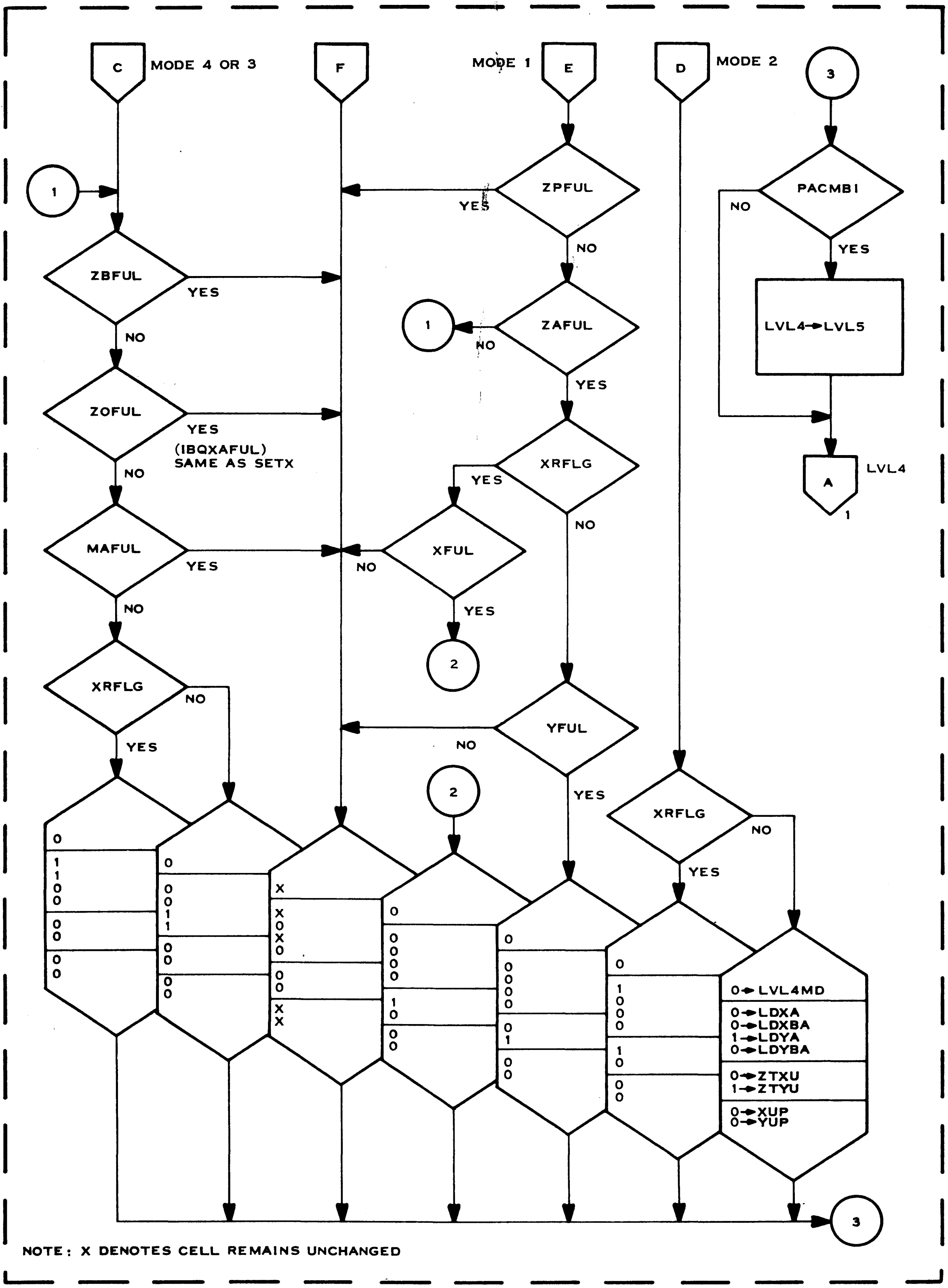
(B) 114347

Figure 4-57. Level 4 Controller Flowcharts (Sheet 1 of 3)



(B) 114348

Figure 4-57. Level 4 Controller Flowcharts (Sheet 2 of 3)



NOTE: X DENOTES CELL REMAINS UNCHANGED

(B) 119491

Figure 4-57. Level 4 Controller Flowcharts (Sheet 3 of 3)

address of the level 4 instruction is different from the storage address of the level 3 instruction. If a current address in the ZP register (ZPFUL) is different from the storage address of the new level 3 instruction (AR ≠ ZP), the controller must wait until the operation in the pipe is completely stored into the Z buffer before allowing the new instruction into the pipe, that is, before issuing PKU to the level 3 controller. Similarly, all write operations must be clear of the pipe and into the Z buffer before the new instruction can be entered into the MBU. The controller then examines the instruction at level 3 to ensure that it is not a store into memory. A store into memory (Z store) occurs if the instruction at level 3 is not a skip, not a compare instruction, and does not have a register file destination. In addition, a store into memory occurs if the level 3 controller is in state 11 and the instruction is a push, or if the instruction is an exchange and the Hold flag is not set. For all of these cases, the controller maintains the MBU interface control bits at their original states and loops within the level 4 mode zero until the pipe clears. When PACMBI is received, the contents of level 4 are transferred to the MBU, but an inactive PAC4 holds the level 3 instruction at level 3 until one of the above conditions clears. PAC4 is issued when the pipe clears of write operations. Notice that the store parameter portion of stack instructions does not affect the controller as the pipe is always clear of other operations at that time.

4-291 MODE ONE

Mode one indicates that the level 4 controller must perform a Z update to transfer current values from Z into the input buffer level of the MBU. The controller first waits for all operations to clear the pipe and be stored into the Z buffer (ZPFUL) not). Then, if Z contains a valid octet (ZAFUL), the controller examines XRFLG to determine which buffer, X or Y, will receive the update. If that buffer contains valid data (XFUL or YFUL), the controller sets the interface bit that produces the update transfer (ZTXU or ZTYU). If the buffer is not full, the controller loops until the X or Y octet returns from memory and is loaded into the buffer. Notice that if the Z buffer did not contain valid data (not ZAFUL), the controller jumps to the control path associated with modes 3 and 4 to fetch the desired octet from memory and load it into one of the operand buffers.

4-292 MODE TWO

Mode two indicates that the level 4 controller must first update an octet in the MBU operand buffers using data in the Z buffer, and then load a value from that operand buffer into the pipe. If the instruction at level 4 does not have the Push, Pull or Modify gate bit set (PPMFG) or is not a Z store portion of the stack instruction, the controller ensures that all data has been transferred from the pipe (not ZPFUL), that the value in the output of the AU is not being stored into the Z buffer, and that the MBU is ready to accept data from the IPU (PACMBI) before it enables the update to the selected operand buffer (ZTXU or ZTYU) and selects a value from that buffer for the pipe operand (LDXA or LDYA).

4-293 MODE THREE

Mode three indicates that the level 4 controller has missed a chance to use the Z to X or Z to Y update path because the Z buffer data has moved out to ZB; the update must come through the memory path. The level 4 controller must load an octet from memory into one of the operand buffers in the MBU, and then select a word from that octet to be loaded into the pipe. Before the controller can initiate the request to memory, it must be sure that the needed data must be stored into memory from the MBU. Therefore, the controller waits until ZBFUL (indicating an octet in the MBU output to memory), ZOFUL (indicating an octet that has been sent to memory) and MAFUL (indicating an octet that has been accepted by memory, but not stored) are all clear before initiating the memory request to load the needed octet of data into the operand buffer selected by the XRFLG flag.

4-294 MODE FOUR

In mode four, the level 4 controller must wait for the Z buffer octet to be stored into memory, initiate a request for that octet from memory to be loaded into one of the operand buffers, and select a value from that buffer to be loaded into the pipe. At the start of the mode four control cycle, the controller waits for the ZPFUL indication to drop, indicating that a new octet in the pipe has been transferred into the Z buffer, forcing the previous contents of the Z buffer into memory. When the MBU indicates it can receive new instructions (PACMBI), the controller can then continue with the control cycle. Because the desired octet is the one that has just been sent to memory, the controller must wait until that octet is stored into memory before performing the fetch for that octet. Therefore, the controller waits until ZPFUL, ZOFUL, and MAFUL become inactive before enabling a memory request for the octet, loading that octet into the buffer indicated by XRFLG, and selecting a value from that octet to be loaded into the pipe.

4-295 CONCLUSION

Regardless of the mode of operation in the level 4 controller, after successful completion of the cycle, the controller enables transfer of the instruction in level 4 to level 5 (MBU interface). All modes terminate the sequence by clearing the mode level to zero, so that on the succeeding clock pulse, the controller will enter mode zero operation and send PAC4 to the level 3 controller.

4-296 LEVEL 5 SCALAR INPUT CONTROLLER

The level 5 scalar input controller monitors the interface control bits from the IPU levels 3 and 4 and enables data transfers to perform the specified loading operations. The operations performed are further qualified by status inputs from level 5 and level 6 of the MBU. Figure 4-58 illustrates the control paths involved in the level 5 input controller. Since vector loading operations are handled by a different controller that is not dependent on the IPU-MBU interface control bits, this controller is responsible only for scalar loading of the MBU.

4-297 INPUT STAGE NOT ACTIVE (NOT MBIAC)

If level 5 (Memory Buffer Input, MBI) is not busy with an input operation, the controller can accept data and address inputs from the IPU. The IPU inputs can specify any one of three types of operations for the controller to perform: load an immediate operand from the IPU, load an operand from the X buffer, or load an operand from the Y buffer.

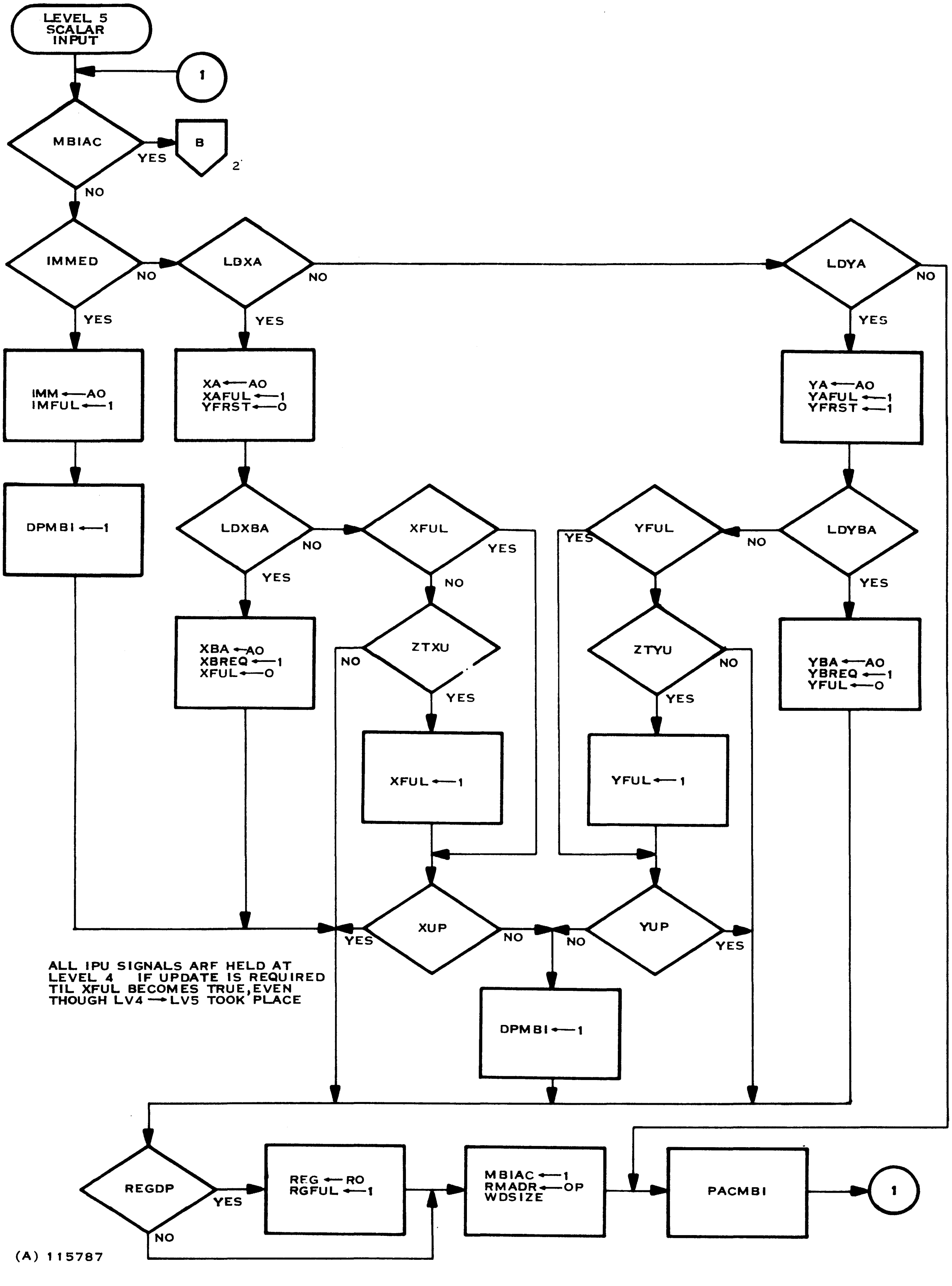
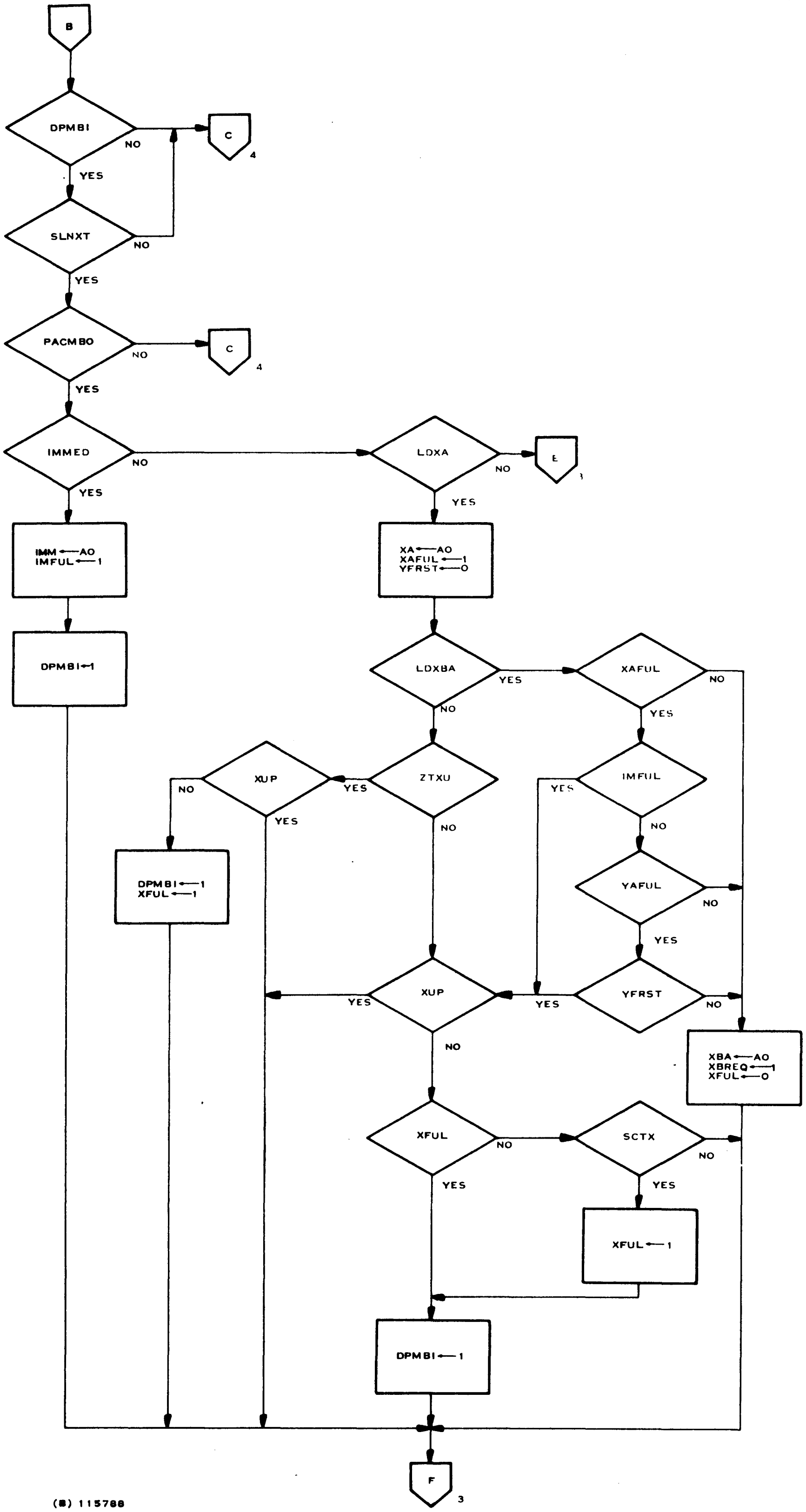
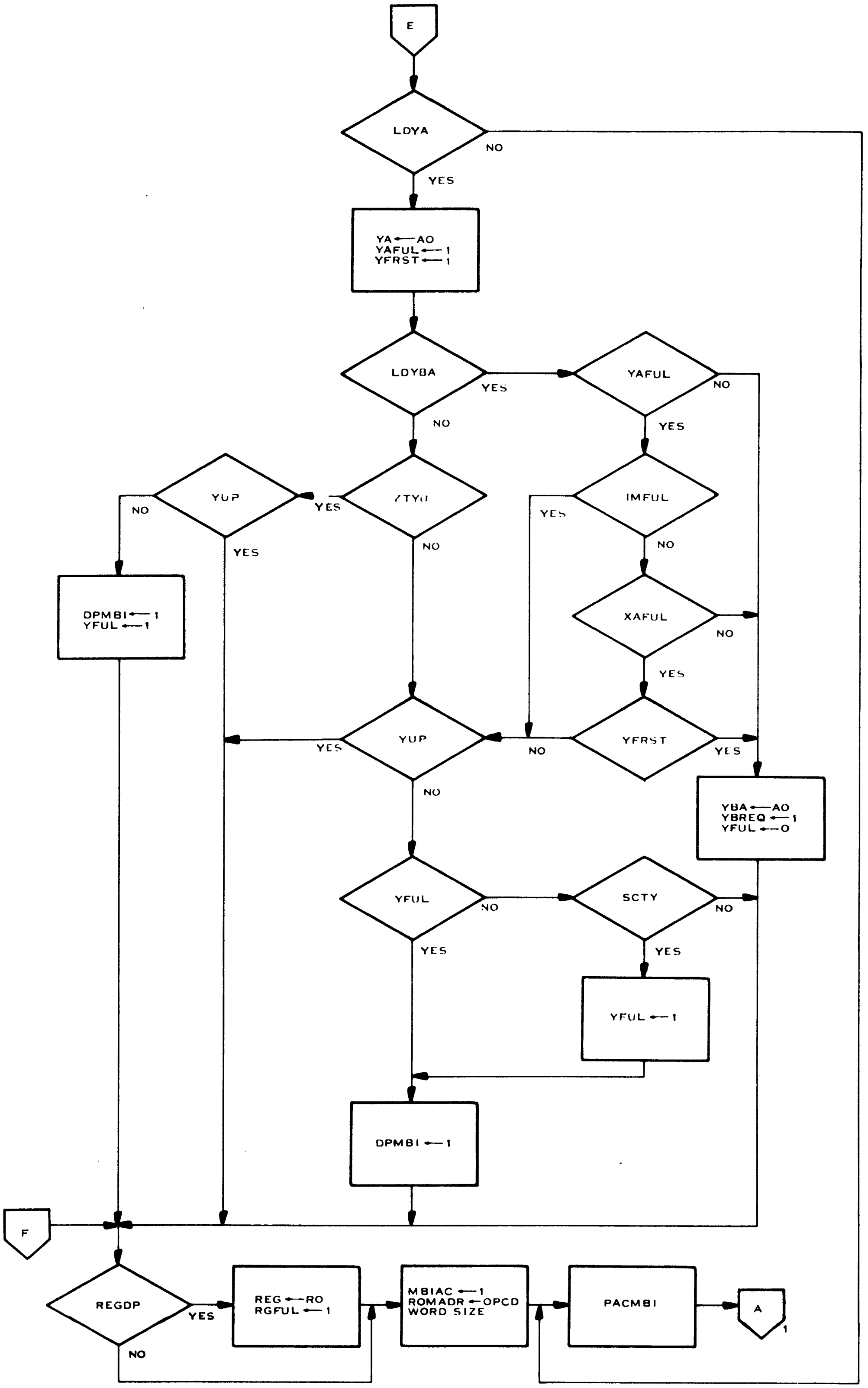


Figure 4-58. Level 5 Scalar Input Controller Flowchart (Sheet 1 of 5)



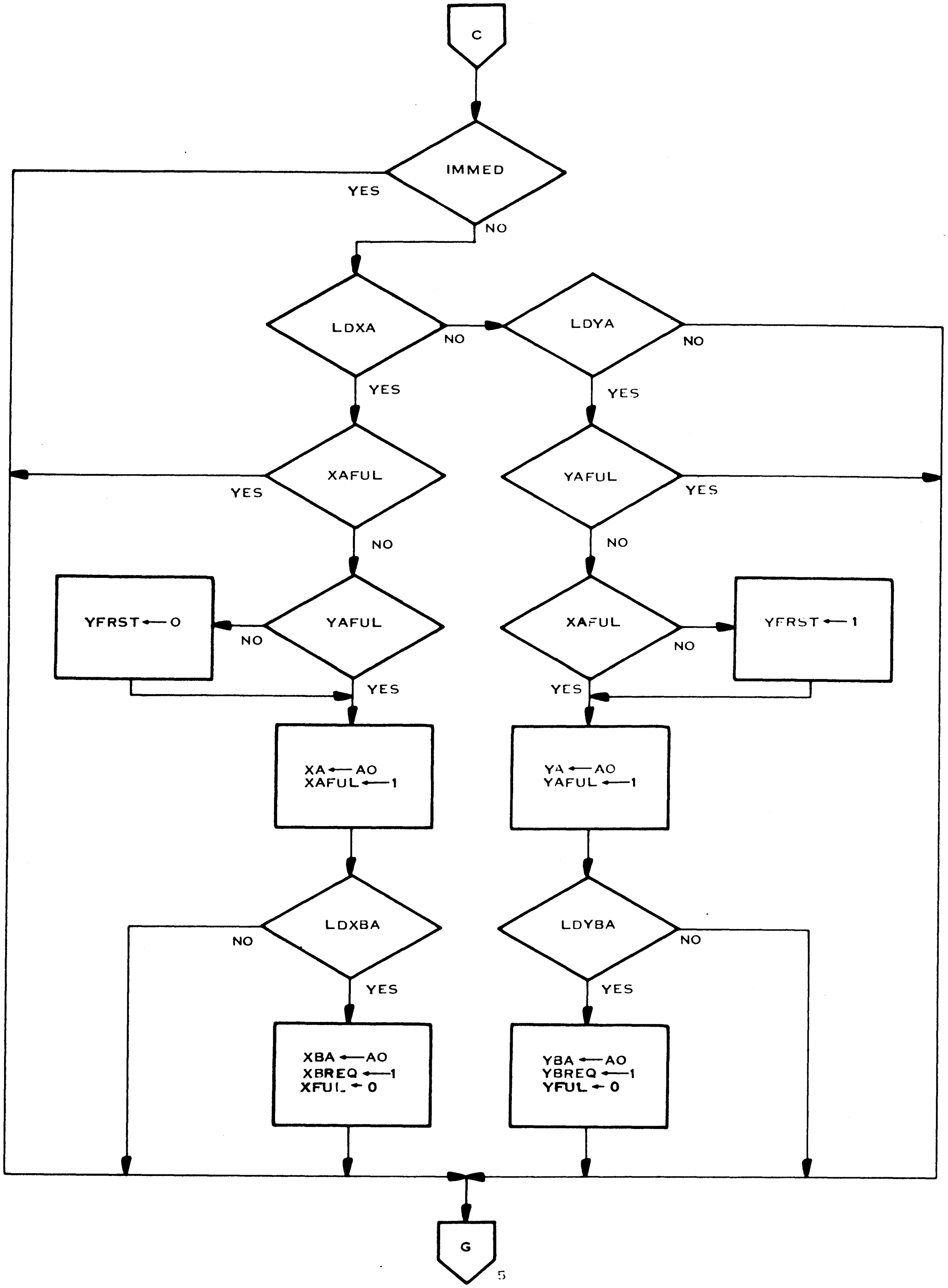
(B) 115788

Figure 4-58. Level 5 Scalar Input Controller Flowchart (Sheet 2 of 5)



(B) 125111

Figure 4-58. Level 5 Scalar Input Controller Flowchart (Sheet 3 of 5)



(B) 115789

Figure 4-58. Level 5 Scalar Input Controller Flowchart (Sheet 4 of 5)

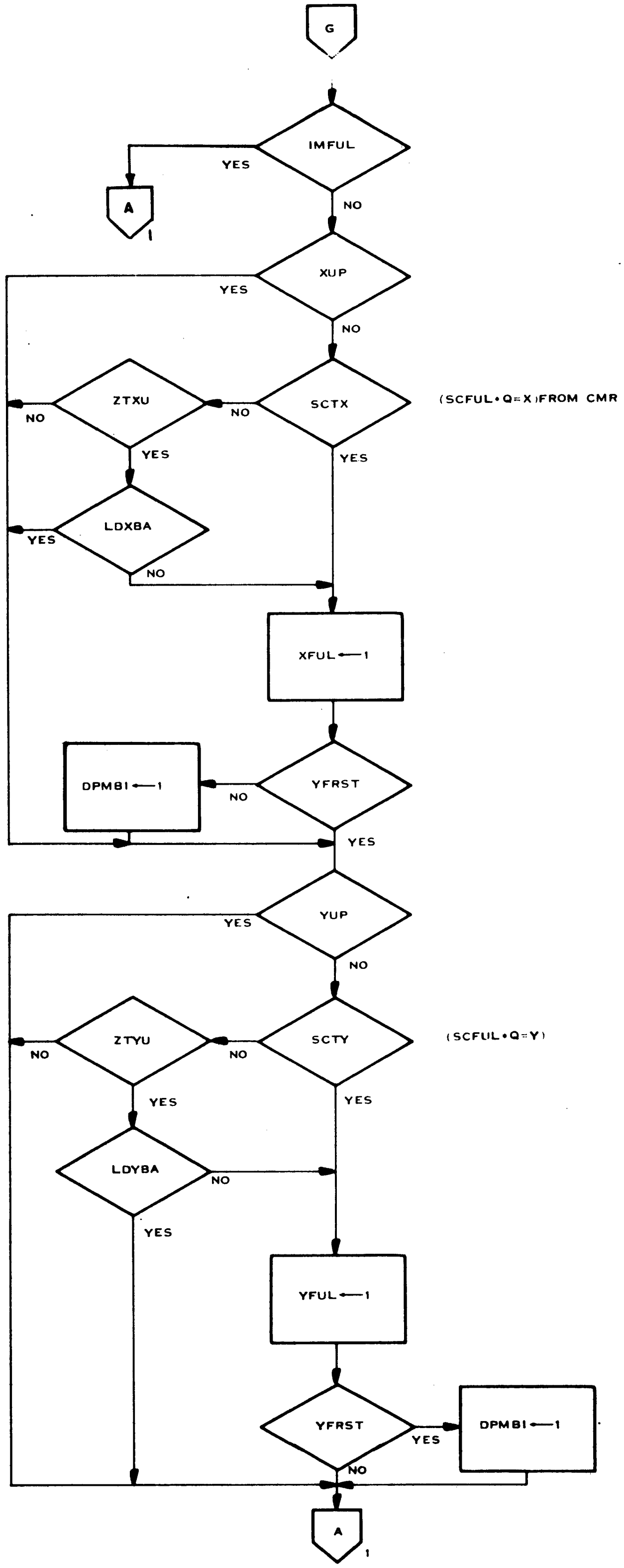


Figure 4-58. Level 5 Scalar Input Controller Flowchart (Sheet 5 of 5)

4-298 LOAD IMMEDIATE OPERAND. If the IMMED interface control bit is set from the IPU, then the A0 register in level 4 of the IPU contains data to be loaded into the pipe. The controller enables the transfer of the data in A0 directly into the IMM register in the MBU input stage, sets IMFUL to indicate that IMM contains valid data, and sets DPMBI to indicate the presence of data in the input stage to the MBU. The controller next examines the Register Data Present (REGDP) interface control bit to determine if the R0 register in the IPU also contains data to be placed into the pipe. If REGDP is set, the controller enables transfer of the data in R0 into the REG register in the input stage of the MBU, and sets RGFUL to indicate the presence of valid data in the REG register.

Regardless of the state of the REGDP bit, the controller sets MBIAC to indicate that the input stage contains valid operands to be transferred to level 6, enables the Op Code from the RX or RY register at IPU level 4 to be transferred to the ROM Address Register, allows the word size indicator from the IPU to select the operand word size to be used during the operation in the AU, and issues PACMBI to the IPU indicating that the MBU is ready to examine the next instruction.

4-299 LOAD FROM X BUFFER (LDXA). If the LDXA interface control bit is set, the level 5 controller must select a word from the X buffer and load it into the MAB output register at level 6. To prepare for this transfer, the controller transfers the address of the word from the A0 register into the XA register in the MBU. XA output will then be able to select the correct operand from the X buffer when level 5 to level 6 occurs. The controller then sets XAFUL to indicate that a valid address is in XA, and clears YFRST. YFRST is a pointer that indicates which buffer will receive the next input octet from memory during scalar operations. When set, YFRST indicates that the Y buffer will receive the input; when clear, YFRST indicates that the X buffer will receive the input. The controller must then determine if the X buffer contains the correct octet. If LDXBA is set, a memory octet must first be transferred into the X buffer before the word can be selected. The controller, therefore, transfers the address of the octet from the A0 register to XBA for input to the Central Memory Requester, sets XBREQ to indicate that CMR will have to perform a fetch for the octet, and clears XFUL to indicate that the octet currently in the X buffer is not the desired octet. The controller sets MBIAC as it leaves this control loop so that it waits for the octet to return from memory before performing any further operations with the level 5 data.

If LDXBA was not set, then no memory fetch will be required to load the X buffer. The controller then checks that the X buffer contains valid data (XFUL). If XFUL is set, or if it is clear and a Z to X update is required (ZTXU), thereby setting XFUL, the controller checks the XUP interface control bit. If this bit is set, the update operation has not yet completed. The controller will wait for XUP to clear before performing any additional operations with level 5 operands. If XUP is not set, then the data in X will not be changed by an update in progress or is being changed by ZTXU update and will, therefore, represent a valid octet. The controller sets DPMBI to indicate the presence of valid data in the input stage. Regardless of the path taken through the load from X sequence, the controller completes the cycle by loading REG with data from R0 and setting RGFUL (if the REGDP interface control bit is set), setting MBIAC to indicate activity at level 5, transferring the Op Code from RX to the ROM Address register, selecting the word size to be used for the operands from the word size indicator in the MBU, and generating PACMBI to indicate to the IPU that the controller can examine a new instruction.

4-300 LOAD FROM Y BUFFER (LDYA). The LDYA control paths are identical to the LDXA control paths, except that the action and decision blocks involve the status and registers associated with the Y buffer instead of the X buffer. Refer to the description of the Load From X Buffer (LDXA) control sequence for an understanding of the LDYA sequence.

4-301 INPUT STAGE ACTIVE (MBIAC)

If MBIAC is set when the controller begins the control cycle, then level 5 contains a valid operation to be performed. The controller can then enter one of two routines depending upon whether the operation at level 5 will be passed to level 6 during the next clock pulse. To determine this condition, the controller examines DPMBI (Data Present MBI), SLNXT (Select Next instruction) and PACMBO (PAC from level 6). If all of these signals are true, then the operation at level 5 will pass to level 6 during the next clock. If any one of these conditions is not met, the operation at level 5 will not pass to level 6.

4-302 TRANSFER OK. If the next clock pulse will transfer the operation in level 5 into level 6, then the controller can examine the incoming instruction and prepare to route it into level 6. The controller, therefore, determines if the instruction in the output of the IPU is an immediate operand, requires a load from the X buffer, or requires a load from the Y buffer.

For immediate operands, the controller enables the next clock pulse to transfer the contents of AO in the output stage of the IPU into the IMM register in level 5 of the MBU, and sets IMFUL to indicate the presence of valid data in the IMM register. The controller also sets DPMBI to indicate the presence of valid data in level 5, before proceeding to examine the REGDP control bit. If there is an operand in the RO register (REGDP set) the controller enables that operand into the REG register in the MBU and sets RGFUL to indicate the presence of data in that register. Regardless of the state of REGDP, the controller sets MBIAC to indicate the new operation in level 5, enables the op code portion of the instruction from RX or RY in level 4 to the ROM Address register in the MBU, and uses the word size indication from the IPU to select the operand size to be sent to the AU. The controller issues PACMBI to the IPU before returning to the start of the control cycle.

If the new instruction from the IPU requires a load into level 6 from either the X or the Y buffer (LDXA or LDYA), the controller transfers the address of the operand from the AO register in the IPU to the respective operand address register (XA or YA) to select the correct word from the output of the Buffer. The controller also sets the full flag (YAFUL or XAFUL) associated with that address register to indicate the presence of a valid address in that register and either sets or clears the YFRST pointer, depending upon whether the Y or the X buffer will receive the next operand from memory. If a memory request is required to load an octet into the selected buffer (LDYBA or LDXBA), YAFUL will not be set during the first control cycle for the instruction. The controller then indicates to CMR that a memory request is required (YBREQ or XBREQ), transfers the address from A0 into the operand address register (YBA or XBA) and clears YFUL to indicate that the octet in the Y buffer is not valid. If YAFUL is set when the controller takes the LDYBA path, then the instruction in level 5 is using the YA register or the controller has already made the memory request for the level 4 instruction. If YAFUL is set, the controller makes three other inspections to determine if it can request a new octet from memory to satisfy the LDYBA bit. If YAFUL and IMFUL are both true, then the instruction in level 5 is an immediate and the controller has already requested the octet for the instruction at level 4. Therefore, the controller checks the YFUL flag and sets DPMBI if the octet for the level 4 instruction has returned from memory. If YAFUL is set, but the instruction at level 5 is not an immediate operand, the controller checks the XAFUL flag. If XAFUL is not set, then the level 5 to level 6 transfer that will occur on the next clock pulse will select a word from the Y buffer, clearing YAFUL. The controller can therefore issue a memory request for an octet to be loaded into the Y buffer. If both YAFUL and XAFUL are set and level 5 is not immediate data, the controller checks the YFRST flag. If YFRST is not set, then the octet to be loaded into the Y buffer for this instruction has already been requested from memory (since the X buffer is first and XA corresponds to the operands in level 5, therefore YA corresponds to the instruction at level 4 and YAFUL indicates that the octet has been ordered). The controller then checks to see if the octet for the Y buffer has returned from memory (SCTY), and if so, it sets YFUL and DPMBI. The controller then sets MBIAC, transfers the op code from RY in the IPU to the ROM Address register and selects the word size of the operand to be used for the operation at level 4. If the REGDP (Register Data Present) flag is set, the controller enables the transfer of the register operand from R0 to the REG register and sets RGFUL. Regardless, the controller sends PACMBI to the level 4 controller so that not only will the next clock pulse transfer the operands of the level 5 operation into level 6, but it will also transfer the instruction at level 4 into level 5.

If a memory fetch is not required to furnish the operand for the instruction at level 4, (not LDYBA), the controller checks the YUP control bit from the level 4 controller. If this bit is set, the controller waits until YUP clears indicating that the Z-to-Y update has been performed and the Y buffer contains the correct data. When YUP clears during a Z-to-Y update (ZTYU), the controller sets DPMBI and YFUL to indicate that the correct data is ready in the Y buffer. If there is no Z-to-Y update and YUP is clear, the controller waits for data to return from memory, sets YFUL, and also sets DPMBI (if data has already returned from memory or was resident in the current buffer, YFUL will already be set). The controller then enables the next clock pulse to transfer level 4 into level 5 along with any register operand that may be in the R0 register, sets MBIAC and selects the operand word size.

4-303 TRANSFER NOT OK. If MBIAC is set at the start of a control cycle, but either DPMBI, SLNXT (select next), or PACMBO is not active, then the instruction at level 5 will not transfer to level 6 during the next clock. Therefore, the level 5 controller will not be able to accept a new instruction from level 4. It can, however, request the octet containing the operand from memory if a request is required. If the instruction at level 4 is an immediate operand, then no memory fetch will be needed. The controller, therefore, waits until the instruction at level 5 moves to level 6 before processing an immediate operand.

If the instruction at level 4 requires an operand from either the X or Y buffer (LDXA or LDYA), the controller can prepare the registers for that operation. The requirements of either an LDXA or an LDYA are identical; only one cycle will be described. For an LDXA, the controller first inspects the XAFUL flag to determine if the instruction at level 5 is using the XA register to select its operand from the X buffer. If that is the case, the controller may not load a new address into XA until the level 5 to level 6 transfer occurs. If XA is not full, the controller inspects YAFUL. If YAFUL is not set, the controller clears YFRST to indicate that, since the instruction at level 4 is the only instruction in levels 4 and 5, the X buffer will be the next buffer to receive a memory transfer. If YAFUL is set, the Y buffer will be used first so the controller leaves YFRST set to select that buffer. Regardless of the state of YAFUL, as long as XAFUL is not set, the controller enables the next clock to transfer the word address of the operand from A0 into the XA register and sets XAFUL to indicate the presence of a valid address in XA. If a memory request is required to fill the X buffer (LDXBA), the controller also enables the octet address in A0 to the XBA register, clears XFUL to indicate that the X buffer does not contain the desired octet, and issues XBREQ to CMR to indicate the need for a memory request. The controller then examines the X buffer flags to determine if the buffer contains valid data for the instruction to be executed. If the XUP bit is still set, the buffer requires a Z-to-X update and the present data is not correct. The controller then waits for XUP to clear. When XUP has cleared, the controller determines if an octet has returned from memory (SCTX) into the X buffer. If SCTX is true, the controller sets XFUL to indicate the presence of data in the X buffer. Also, if new data did not arrive, but a Z-to-X update was performed (ZTXU), the controller also sets XFUL. If YFRST is not set, indicating that the X buffer will be the next source of operands for the MBU, the controller also sets DPMBI to indicate that the input stage is prepared to transfer data to level 6.

4-304 LEVEL 6 CONTROLLER - SCALAR MODE

The level 6 controller (figure 4-59) operating in the scalar mode monitors signals from the level 5 and Select Next controllers to gate data from level 5 into level 6 for output to the AU. For a transfer to occur, the following signals must be present:

- DPMBI - Data Present in Memory Buffer Interface
- SLNXT - Select Next; indicates that the next instruction can be input to the AU
- DPMBO - Data Present in Memory Buffer Output stage

  or
- PACAUR - Path Ahead Clear in the AU Receiver section.

If these signals are present, the controller checks the level 5 register status flags. If an immediate operand is in level 5, the controller enables the contents of IMM into the MCD register, clears IMFUL and sets DPMBO. If a register operand is in level 5, the controller enables the contents of the

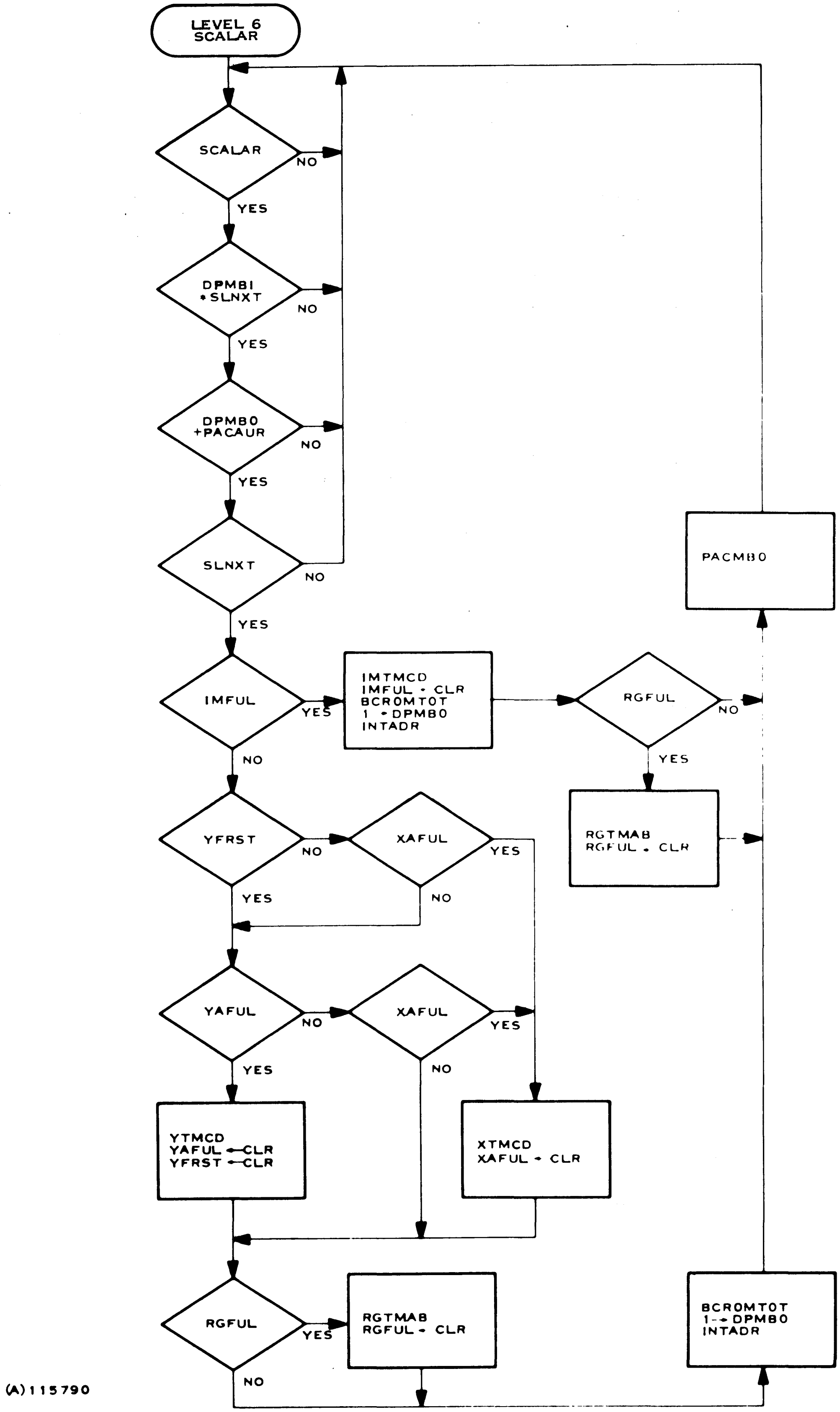


Figure 4-59. Level 6 Controller Flowchart - Scalar Mode

REG register into the MAB register. For X or Y operands, the controller checks the YFRST flag. If it is set, the controller examines the Y buffer flags first. In either case, if the first inspection does not result in a transfer into MCD, the controller will check the flags associated with the other buffer and transfer data from that buffer into MCD if valid data is present. When all required transfers are complete during a control cycle, the controller issues PACMBO to the level 5 controller.

4-305 LEVEL 6 CONTROLLER - VECTOR MODE

Level 6 vector mode control is illustrated in figure 4-60. Description of this circuit will be supplied at a later date.

4-306 SELECT NEXT CONTROLLER

The Select Next controller (figure 4-61) enables transfer of instructions into the AU, operation codes into the AU control ROM, and operands from level 5 to level 6 by setting the SLNXT control bit. Except for vector initiation, the controller is used for scalar operations only. Select Next (SLNXT) is set under the following conditions:

1. VECTOR INITIATION loads first vector address into the AU control ROM.

2. MBI INACTIVE

   - MBI Inactive and AU empty - enables level 5 to level 6.
   - MBI Inactive and AU active with a one clock operation - enables level 5 to level 6 since AU will be empty on next clock.
   - MBI Inactive, PACMBO and AU active during last clock of operation - enables level 5 to level 6 since AU will be empty on next clock.
   - Next ROM Code NO OP - No Op will have no effect on operation in AU; therefore level 5 to level 6 is enabled.

3. MBI ACTIVE and PACMBI

   - One clock operation at level 5 - enables level 5 to level 6 to prepare to insert one clock operation into pipe at completion of current operation.
   - Same group at level 4, first clock of same group and no short circuit (wait) at level 4 - allows transfer of instruction to prepare for interleave of operation in same group time.

4. MBI ACTIVE AND NOT PACMBI

   - AU Inactive - enter next operation into pipe for processing.
   - One clock at level 5 and in AU - Enter next operation since AU will be vacant after next clock.

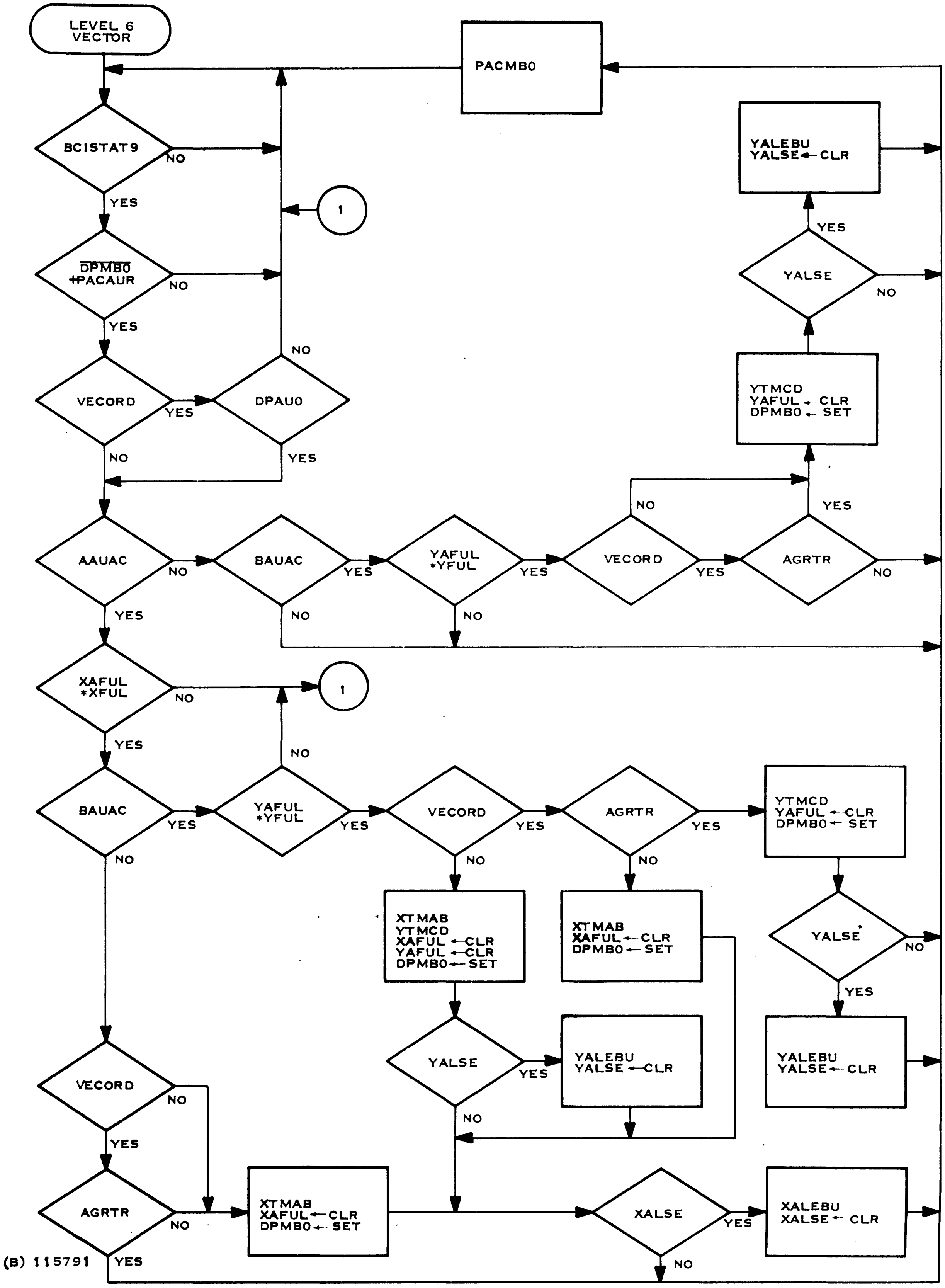
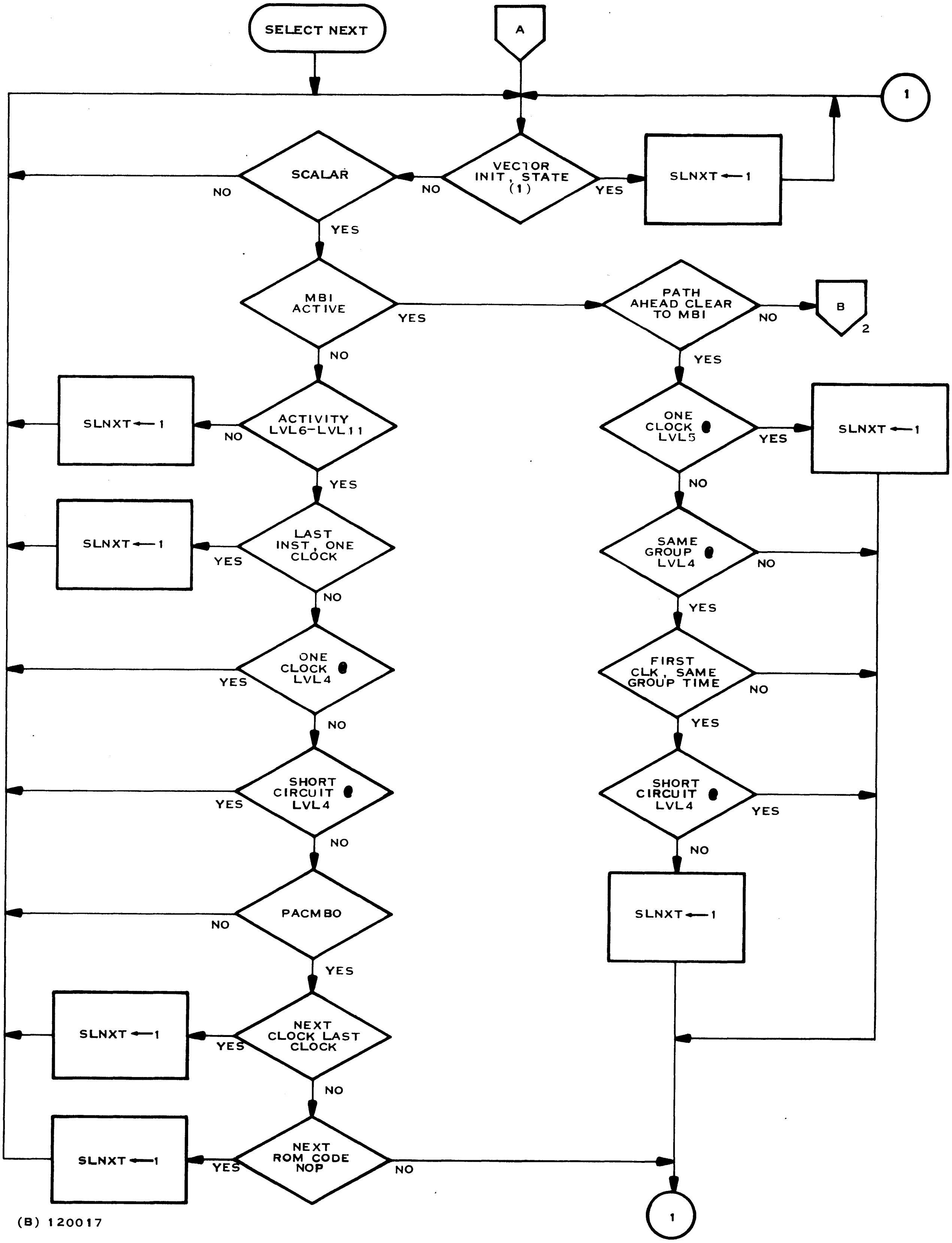


Figure 4-60. Level 6 Controller Flowchart - Vector Mode



(B) 120017

Figure 4-61. Select Next Flowchart (Sheet 1 of 2)

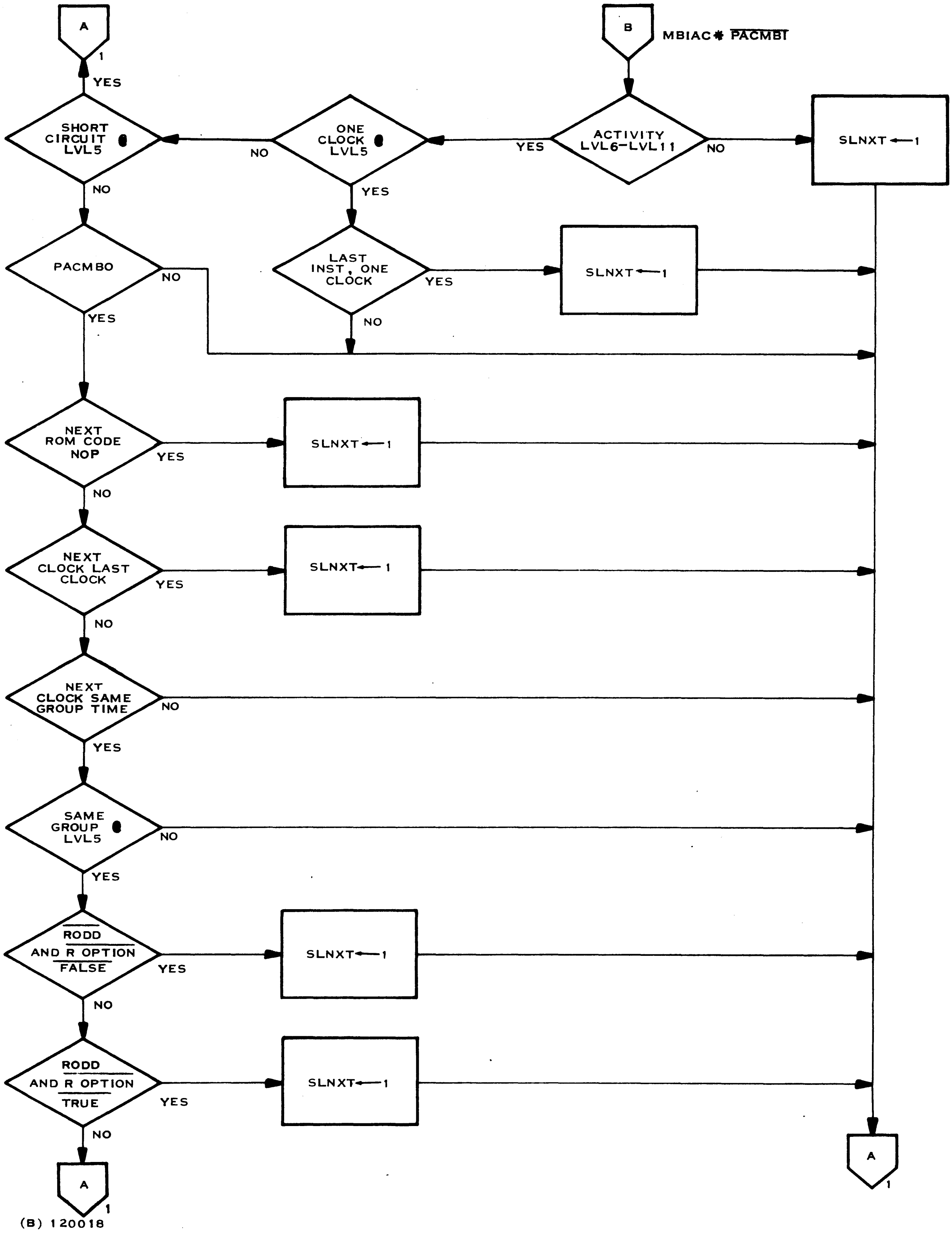


Figure 4-61. Select Next Flowchart (Sheet 2 of 2)

- AU active, PACMBO, Next ROM Code No Op - No Op will not affect AU, load MBO in preparation for next operation.

- AU active, PACMBO, Next clock last clock of current ROM instruction - AU operation is completing, load MBO in preparation for next operation.

- Same group, Same group time, and if single-length multiply, the result in the AU will be the same as the result of the new instruction at level 5 (see explanation below).

RODD indicates that the result of a single-length multiply at level 5 will be stored into an odd memory location, and therefore cannot be a doubleword result. R OPTION FALSE and R OPTION TRUE are AU ROM signals that indicate the word length of the result of a single-length multiply currently in the pipe. R OPTION FALSE indicates that the result will be single-length; R OPTION TRUE indicates that the result will be double-length. The Select Next controller; however, examines the complement of these two ROM signals so that non-multiply instructions need not be coded with ones. Therefore, if RODD is false (indicating an even storage address) and R OPTION FALSE is also false (indicating a double-length result), both operations specify the same length result. Similarly, if RODD is true and R OPTION TRUE is false (indicating a single-length result), both operations specify the same length result. The controller can then enable SLNXT so that the operands in level 5 can be inserted into the AU at the same group time to be overlapped with the current instruction in the AU. One of these two paths will always be true for same group operations other than single-length multiply.

4-307 CENTRAL MEMORY REQUESTER (CMR)

The Central Memory Requester (CMR) in the MBU (figure 4-62) receives requests for octets from either the level 5 controller or from the Z storage control circuit. CMR then decides priority for the requests, issues the requests to memory, and routes the returning octets from memory, through the SC Memory Interface File, into the buffer file (X, Y or Z) that requested the data. Data requested for the Z buffer is for a Z Fill-in (ZFILN) operation. That is, the data in the Z buffer contains some incomplete halfwords and cannot be stored into memory without filling the empty halfword positions with valid data from memory. The halfword modified flags in the ZBM register designate which halfwords in the Z buffer contain valid data, and prevent the incoming memory words from changing those halfword positions in ZB. CMR also issues storage requests to memory to write the contents of the ZB file into memory after a fill-in operation, or when the next word to be stored into the Z buffer will be in a new octet. The control circuits contain an examination state for each number of allowable outstanding requests to memory. Each cycle performs the same basic functions.

If a hardcore operation must be performed, the controller checks to see if an outstanding request has returned from memory. If SCFUL is set (new octet in SC) and the queue indicates that the octet is for a Z buffer fill-in operation (CUE = 10), the controller allows the next clock to transfer the contents of SC into the ZB file, clears the ZFILN flag, and enables the halfword

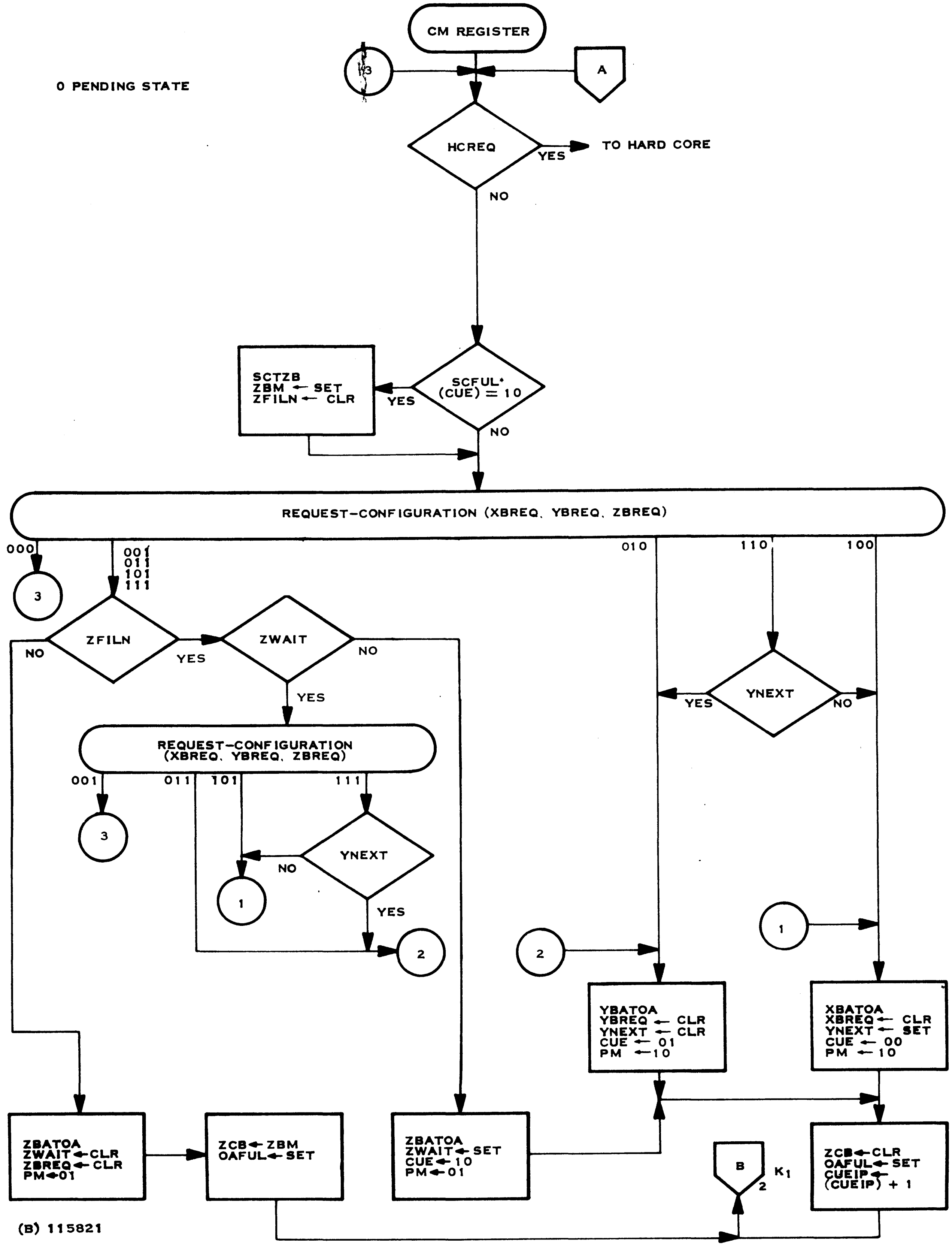
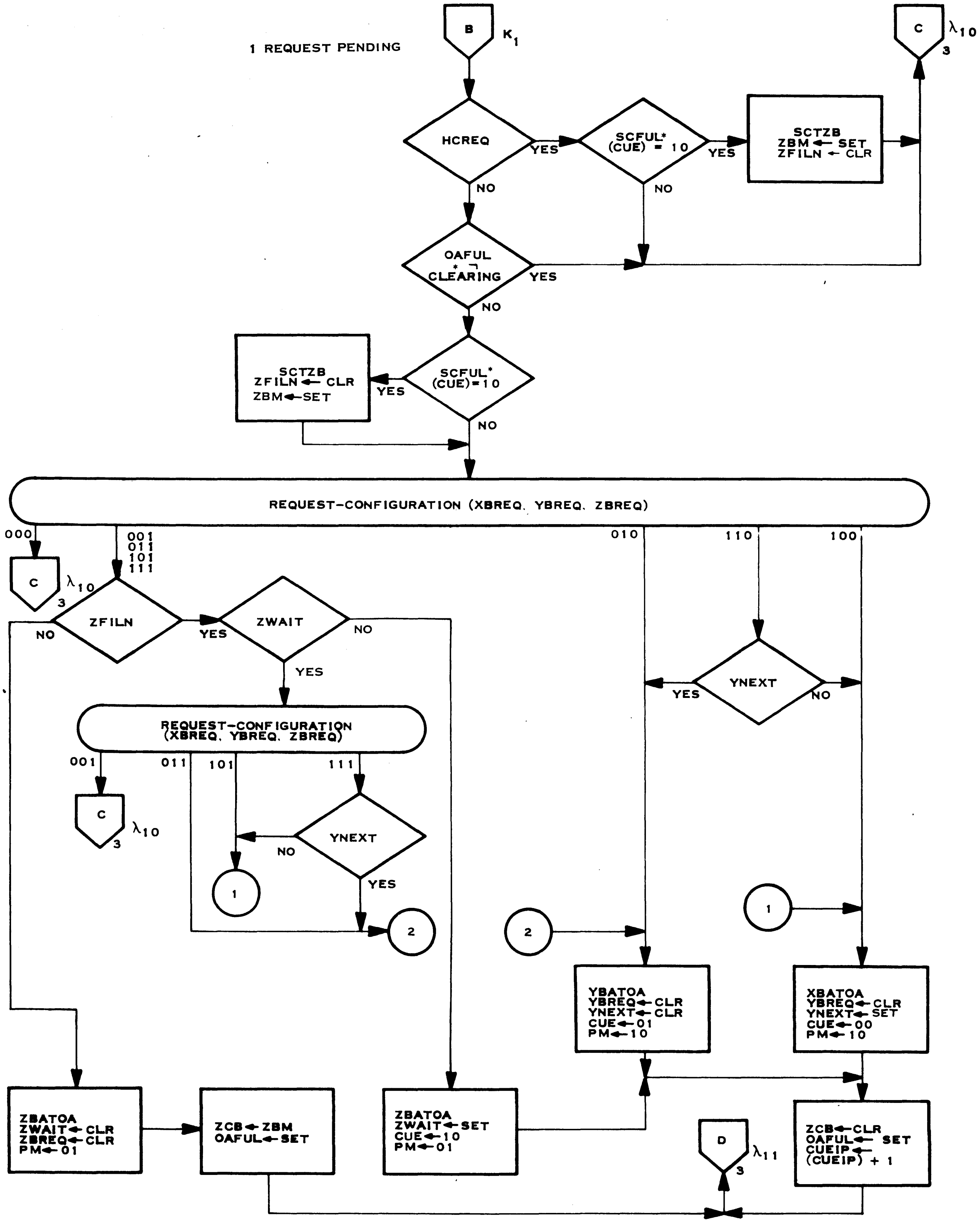
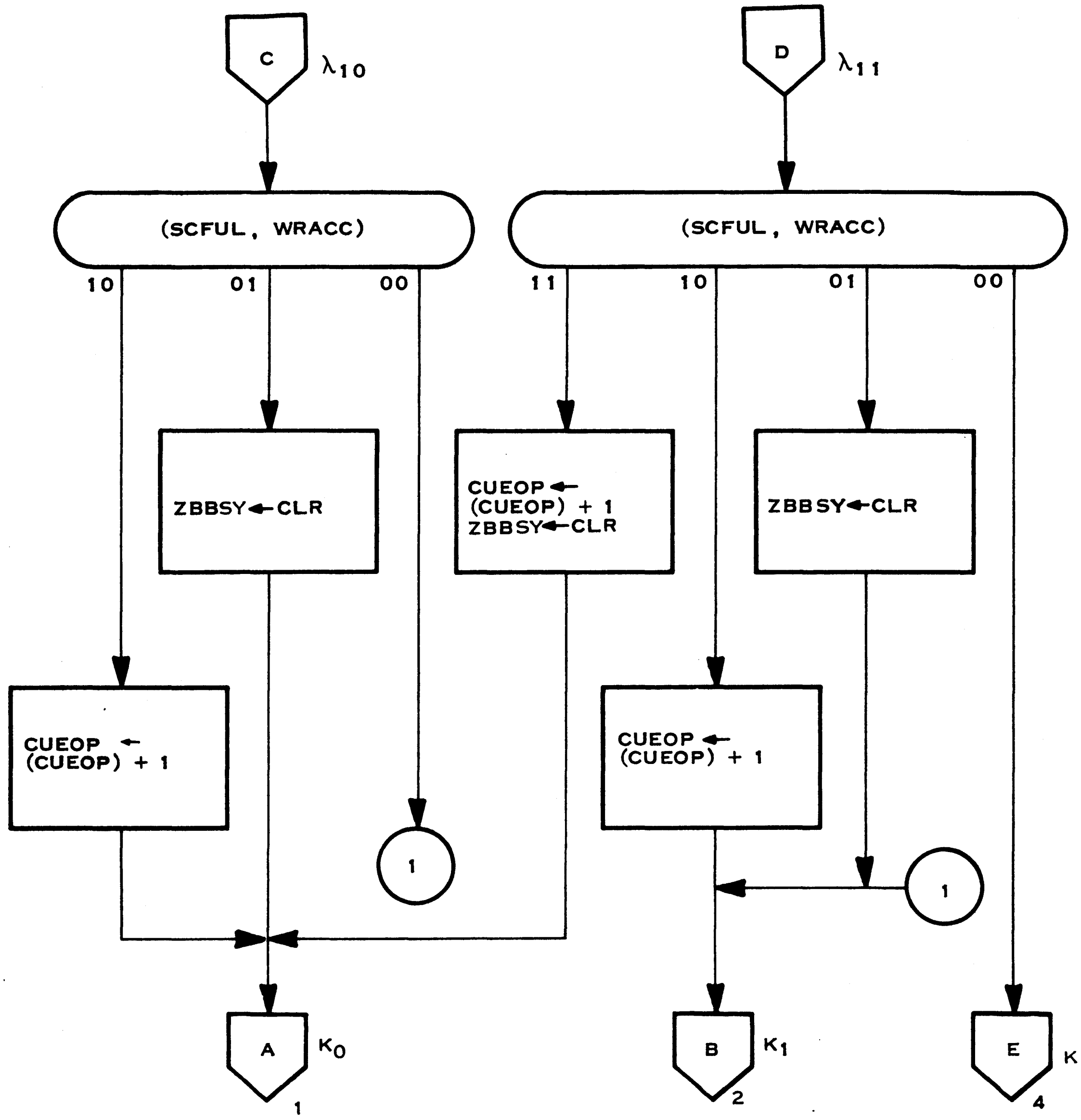


Figure 4-62. CM Requester Flowchart (Sheet 1 of 15)



(B) 115822

Figure 4-62. CM Requester Flowchart (Sheet 2 of 15)



(A) 115823

Figure 4-62. CM Requester Flowchart (Sheet 3 of 15)

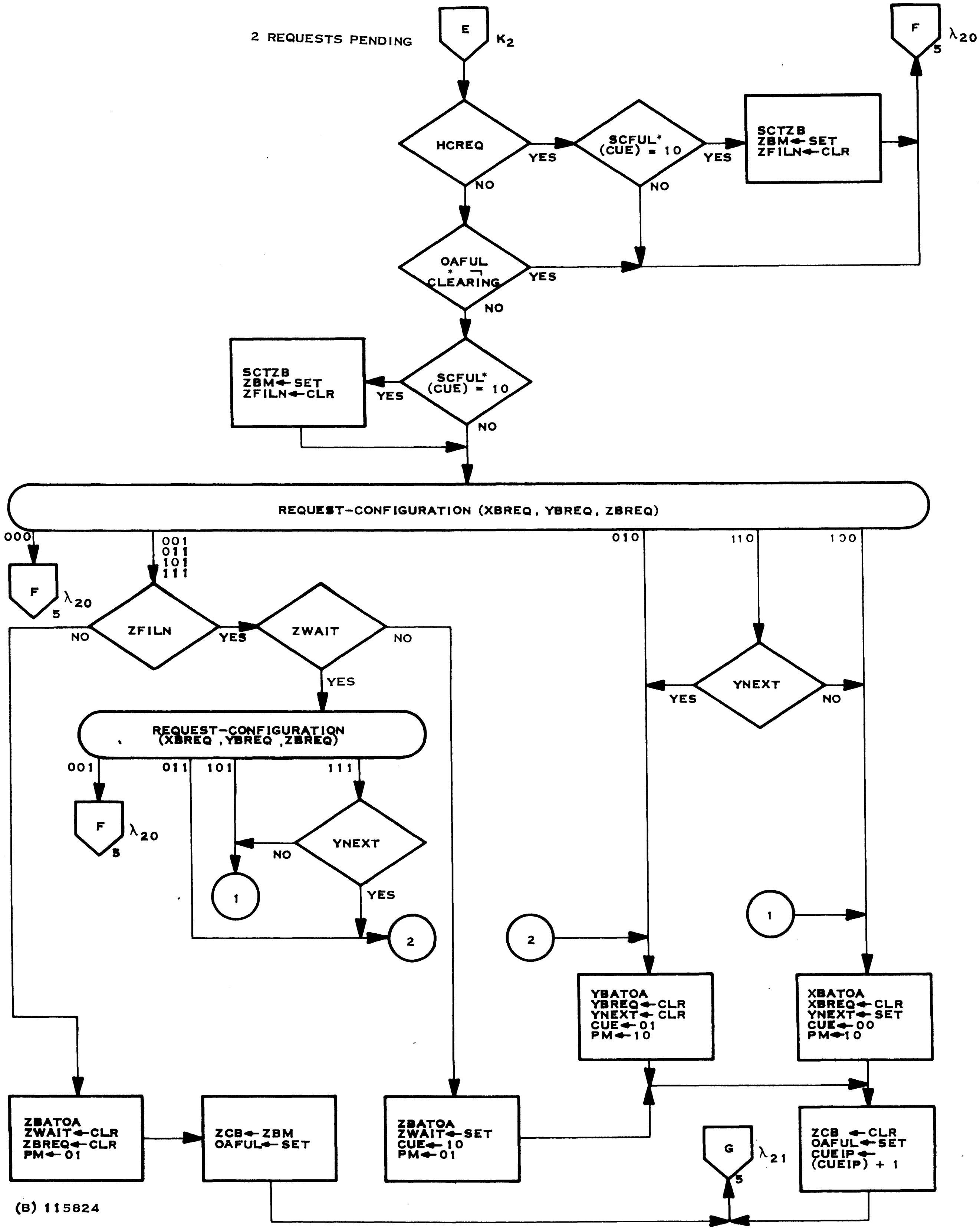
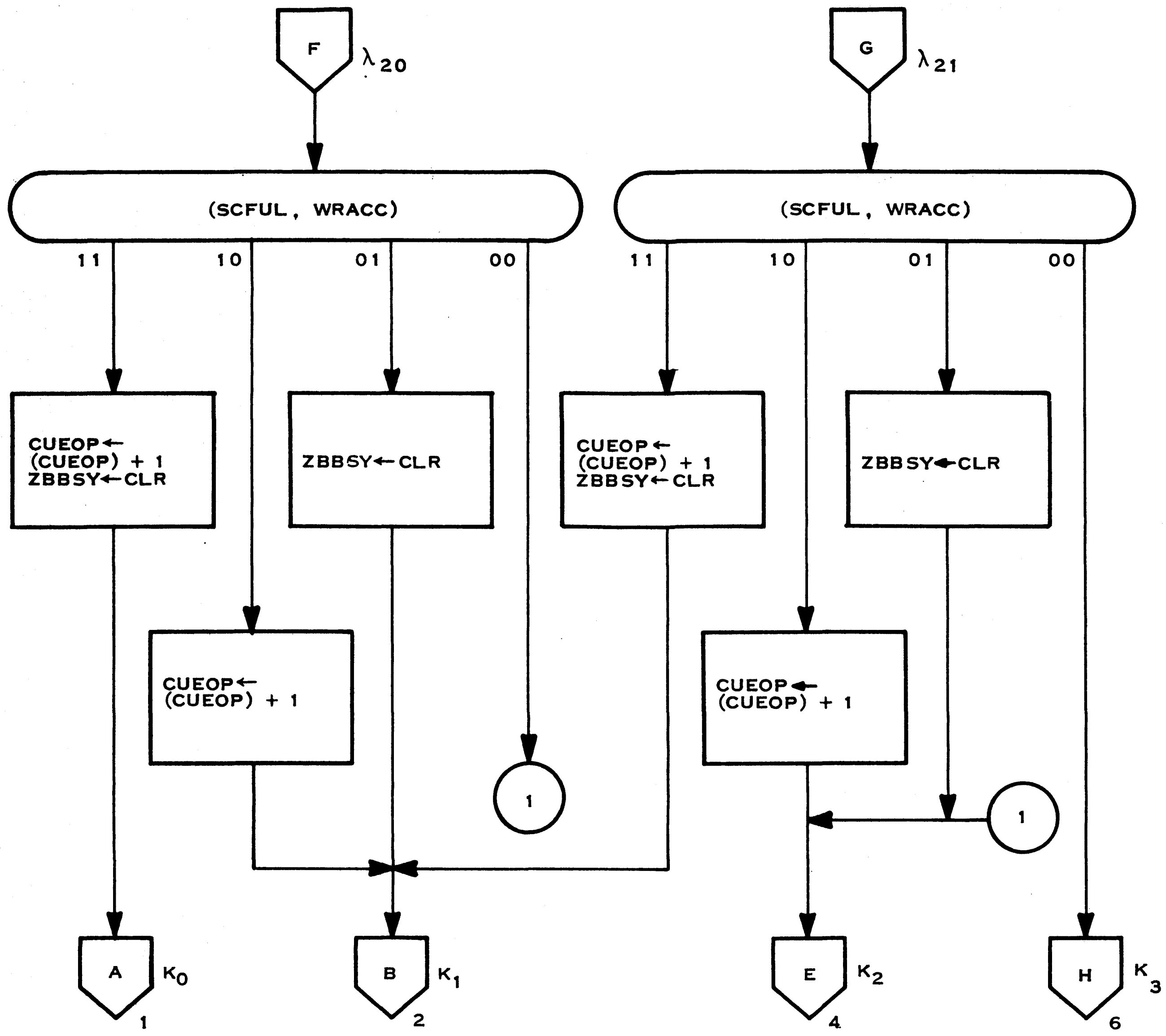
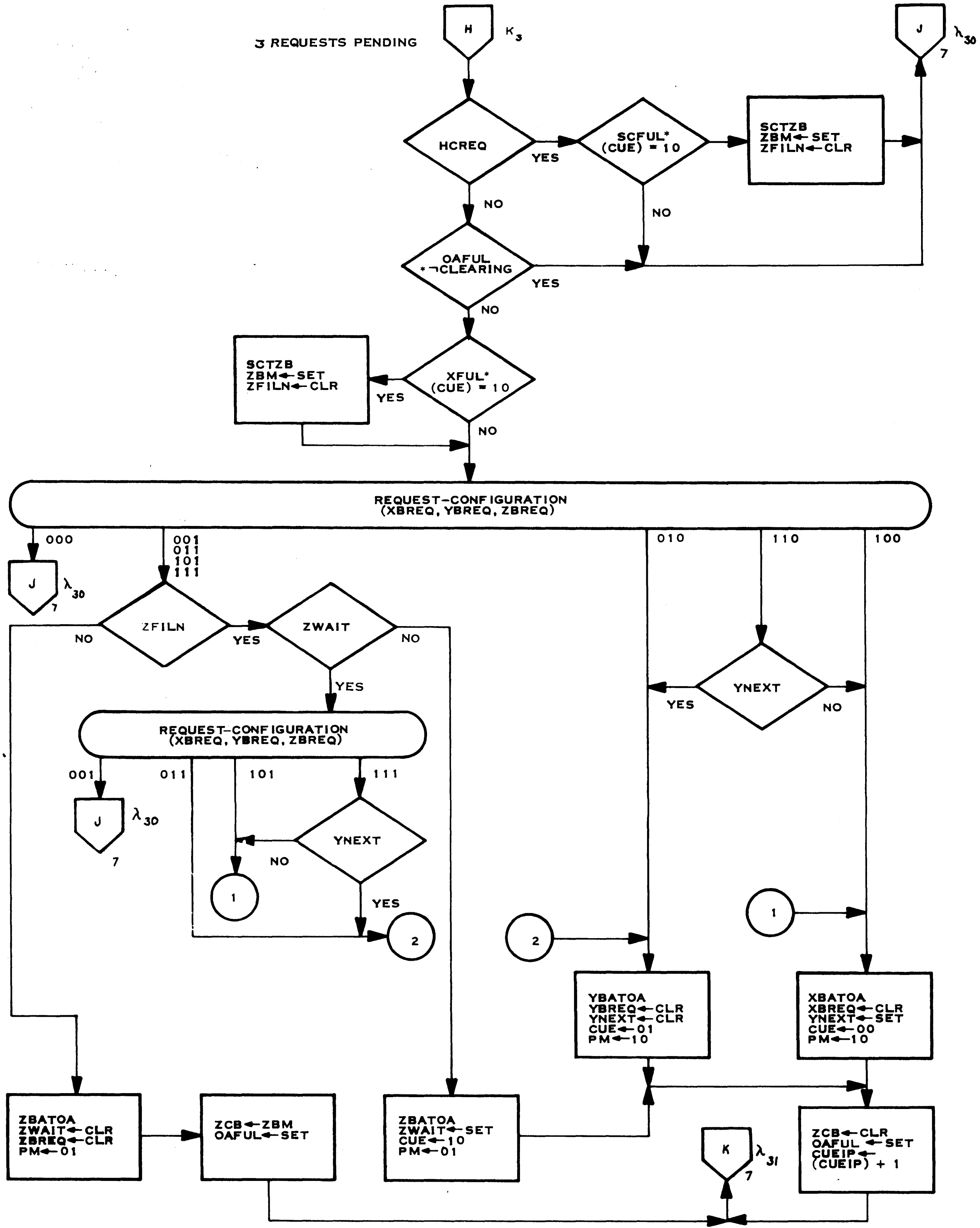


Figure 4-62. CM Requester Flowchart (Sheet 4 of 15)



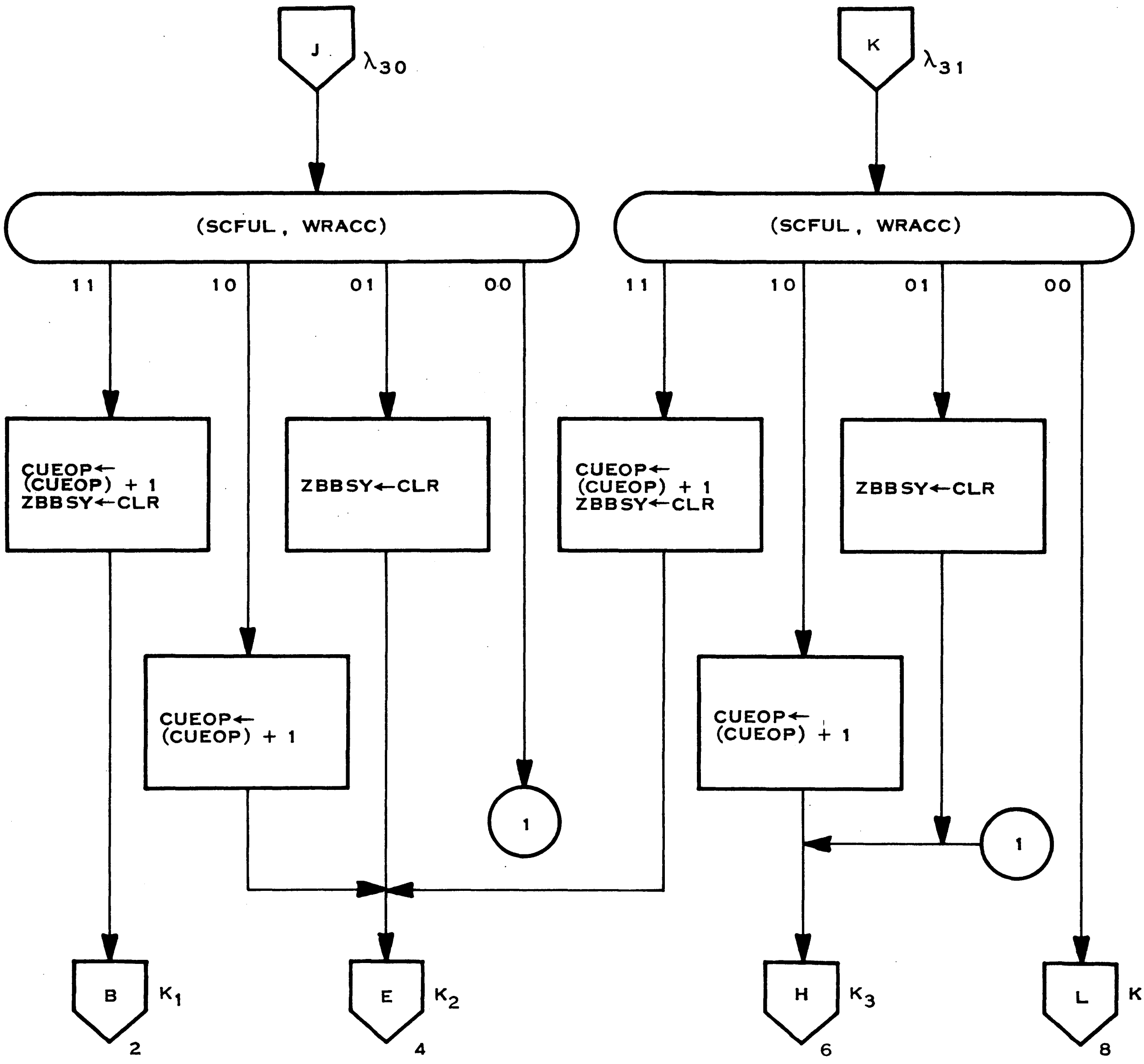
(A) 115825

Figure 4-62. CM Requester Flowchart (Sheet 5 of 15)



(B) 115826

Figure 4-62. CM Requester Flowchart (Sheet 6 of 15)



(A) 115827

Figure 4-62. CM Requester Flowchart (Sheet 7 of 15)

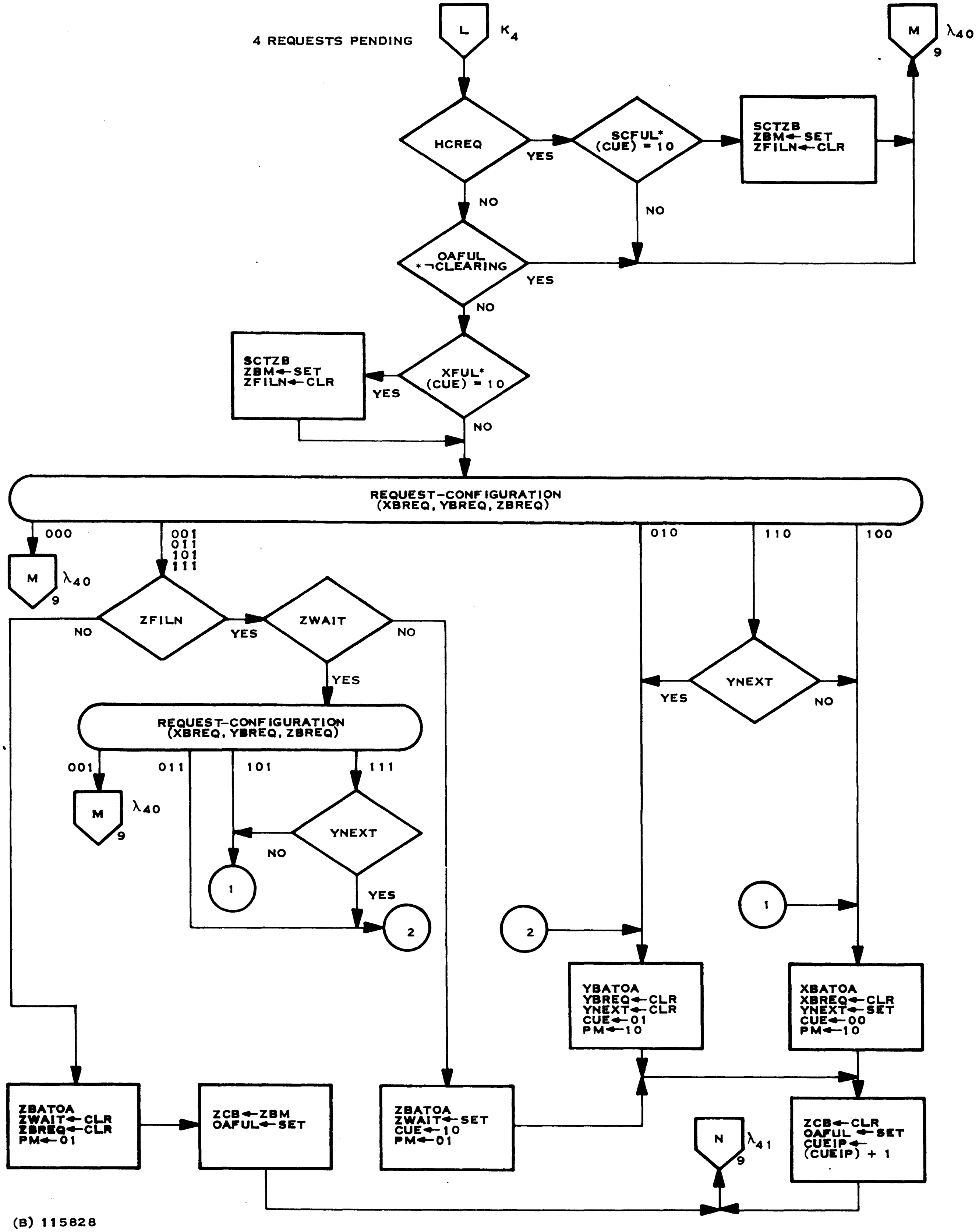
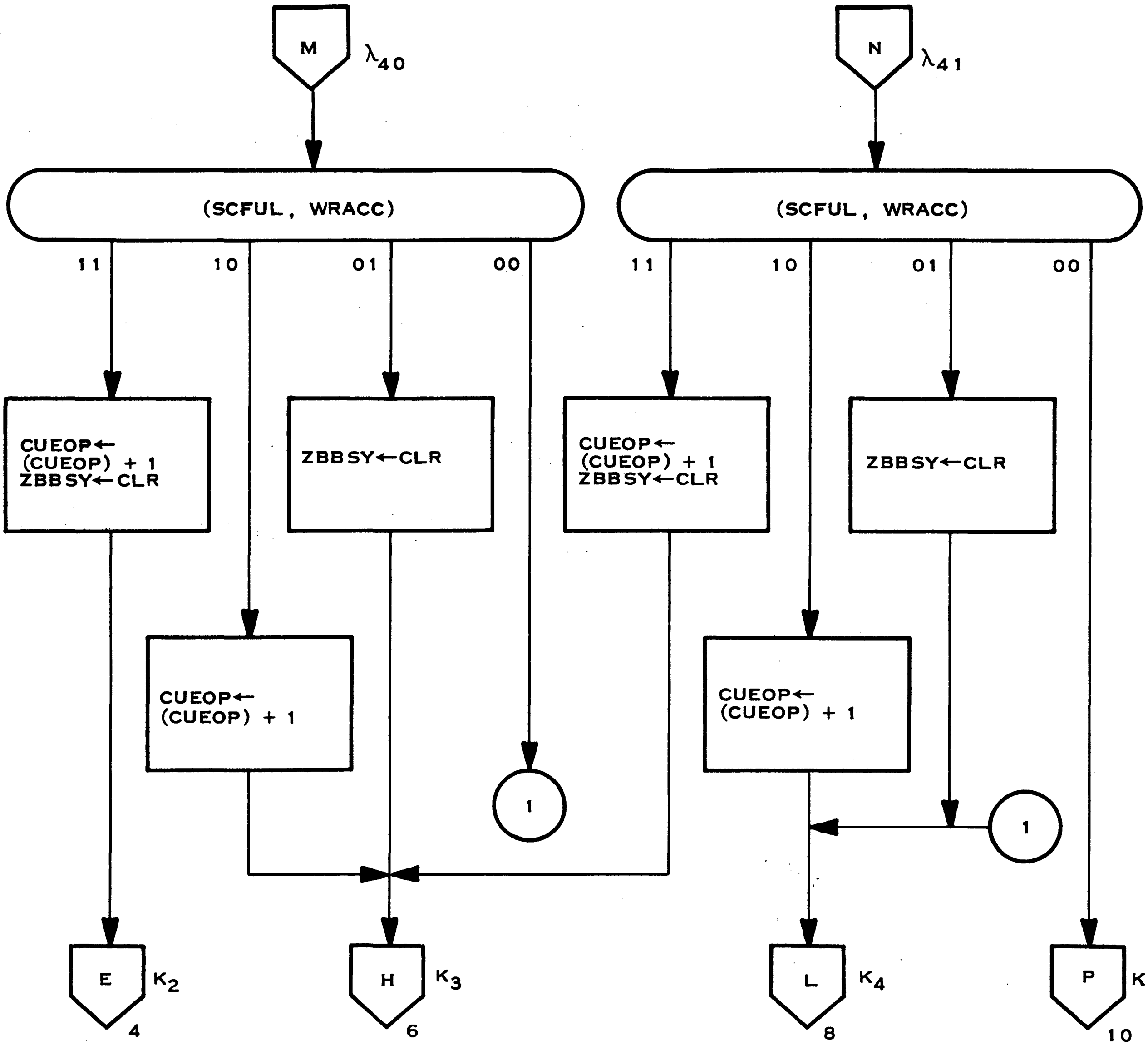
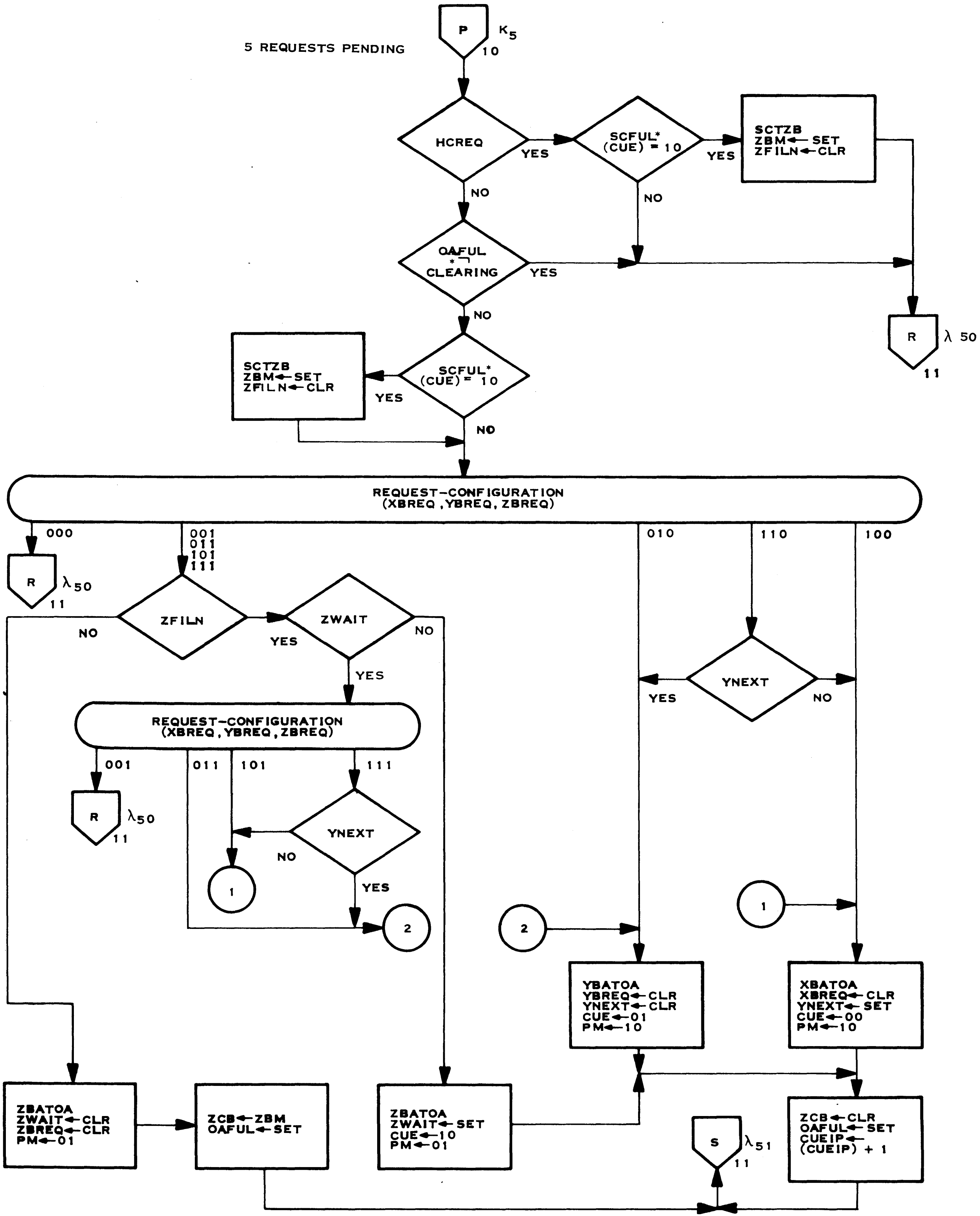


Figure 4-62. CM Requester Flowchart (Sheet 8 of 15)



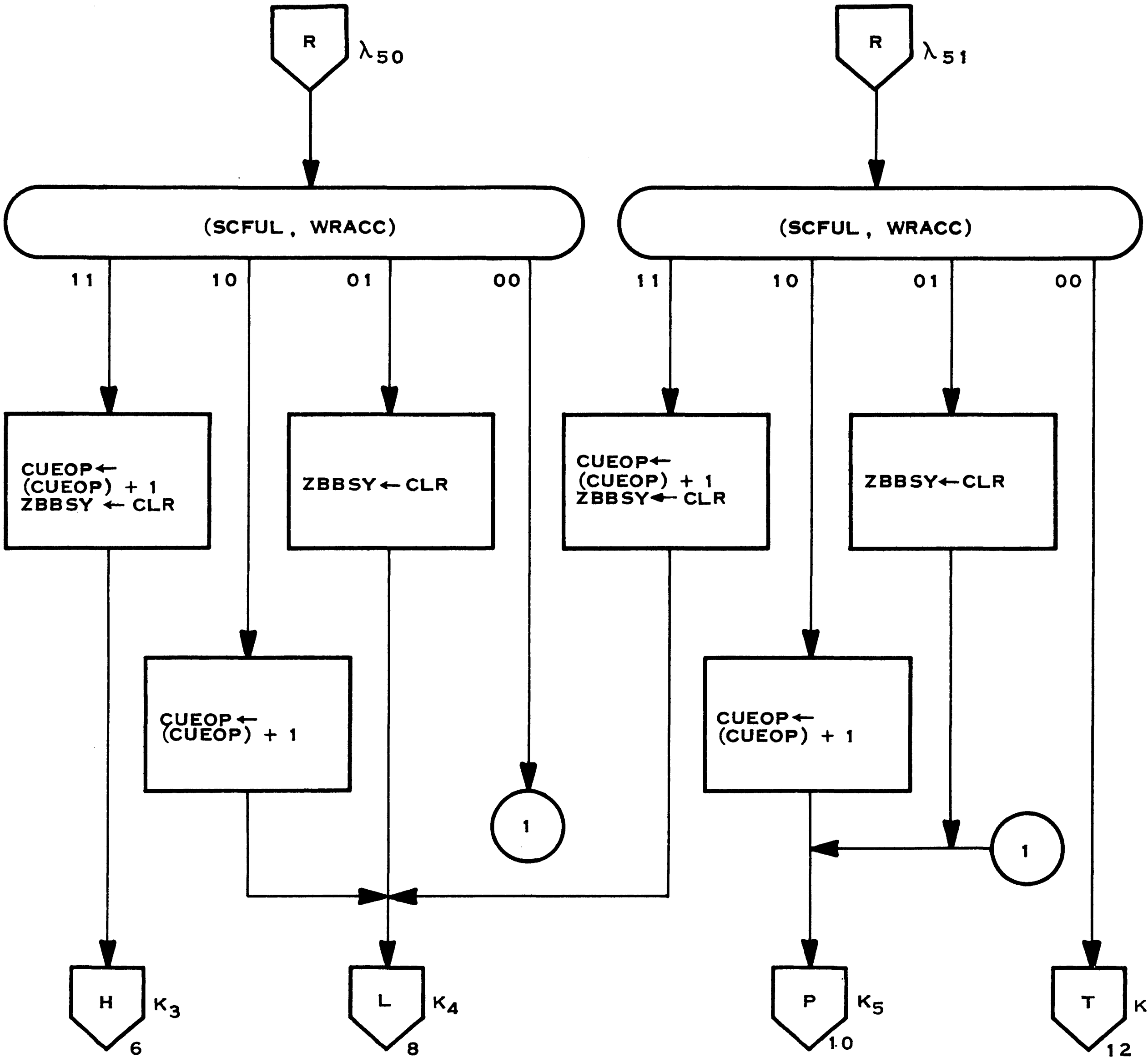
(A) 115829

Figure 4-62. CM Requester Flowchart (Sheet 9 of 15)



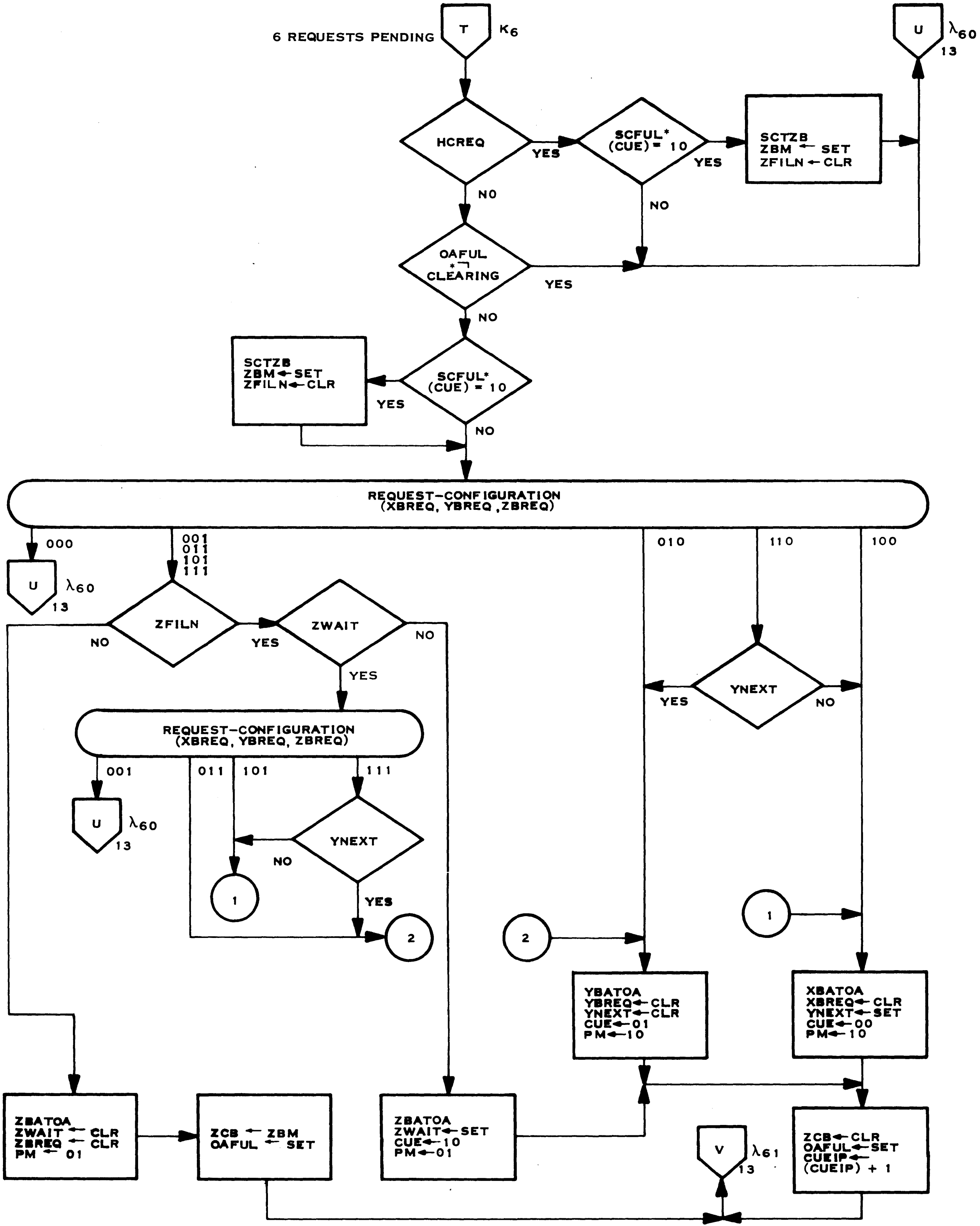
(B) 115830

Figure 4-62. CM Requester Flowchart (Sheet 10 of 15)



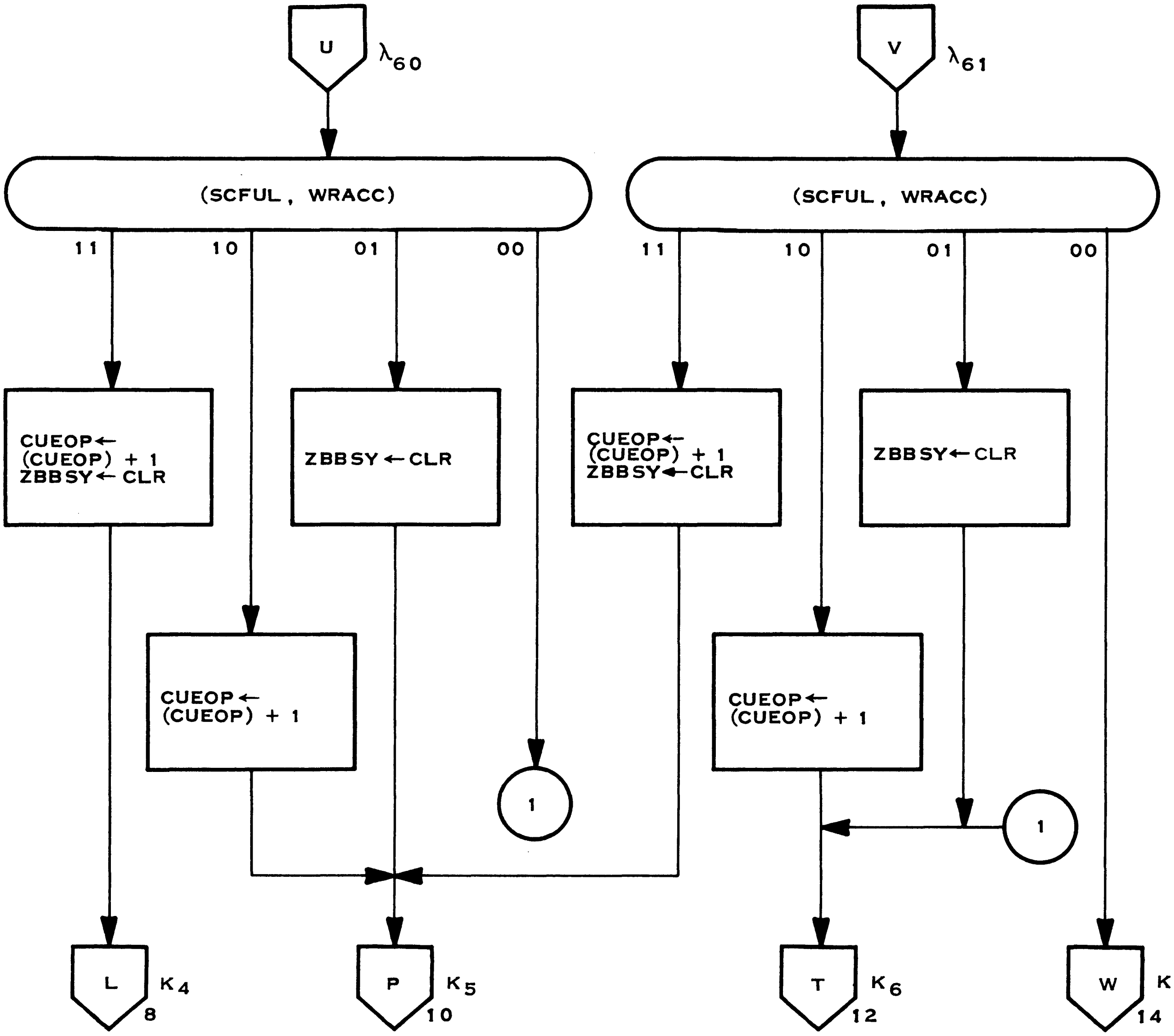
(A) 115831

Figure 4-62. CM Requester Flowchart (Sheet 11 of 15)



(B) 115832

Figure 4-62. CM Requester Flowchart (Sheet 12 of 15)



(A) 115833

Figure 4-62. CM Requester Flowchart (Sheet 13 of 15)

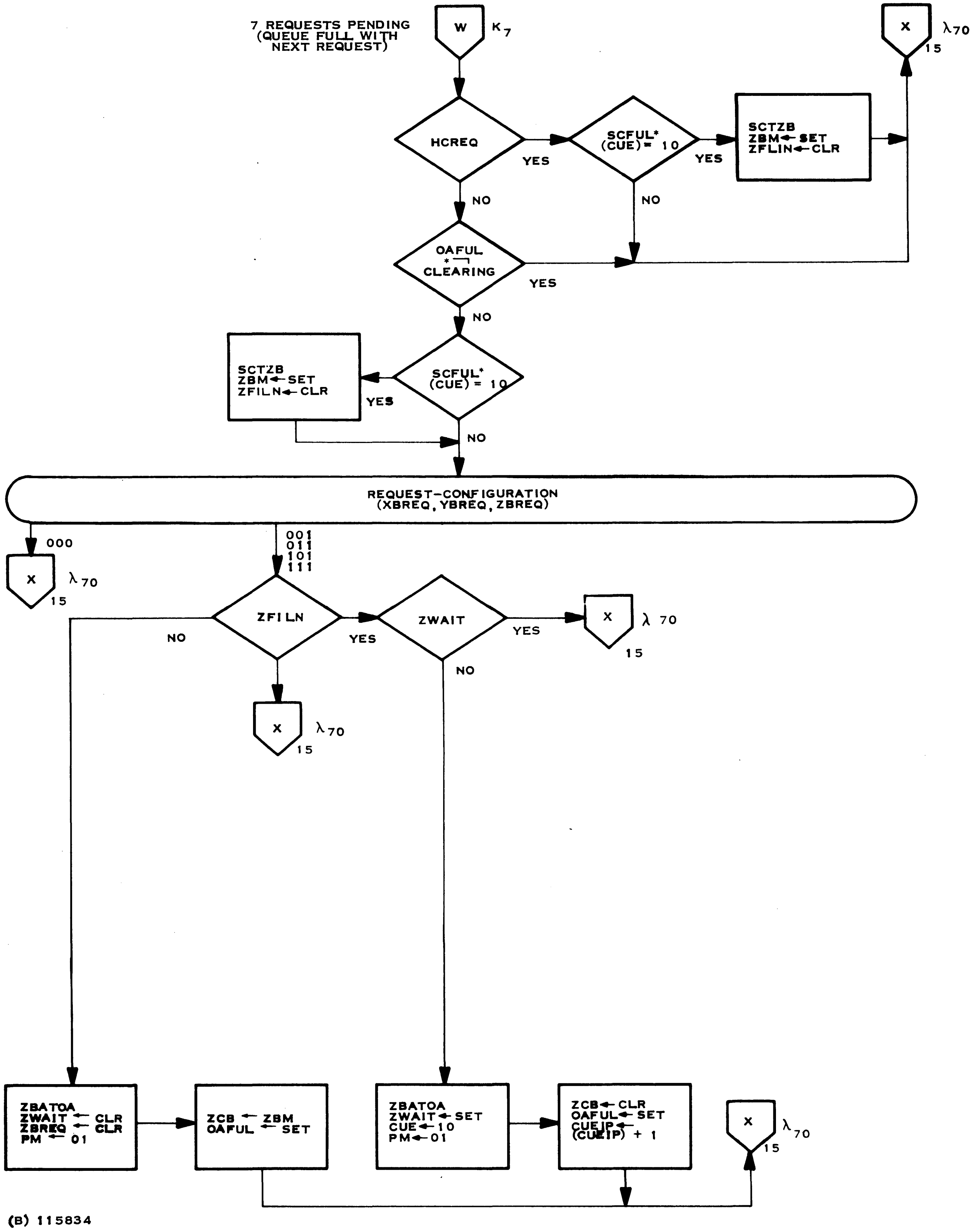
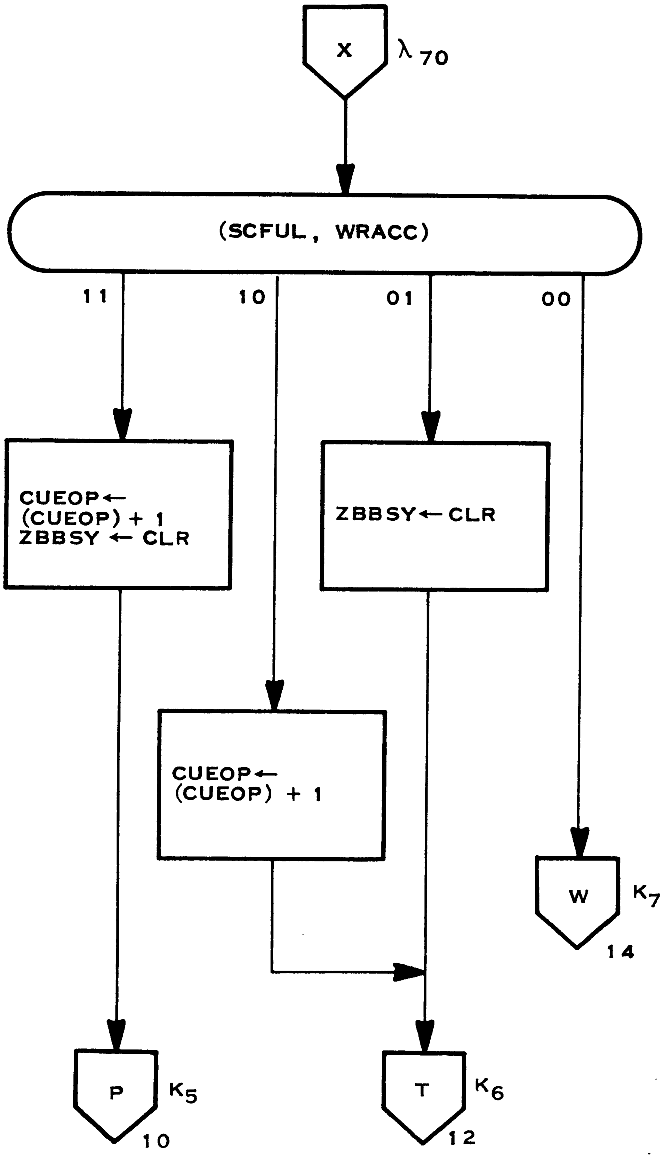


Figure 4-62. CM Requester Flowchart (Sheet 14 of 15)



(A) 115835

Figure 4-62. CM Requester Flowchart (Sheet 15 of 15)

modified flags to transfer the changed halfwords from the Z buffer into ZB to update the octet for storage into memory. The controller relinquishes control to MBU Unit Hard Core for the next clock cycle.

If the OA register is available to transfer another address to central memory, the controller also checks for a Z Fill-in operation and performs the transfers described above. The controller then checks for a request that originates in the Z buffer stream since Z requests must be made immediately due to lack of a buffer stage for the Z data. If there is no ZFILN operation, the controller transfers the address in ZBA to OA to designate the storage area in memory to be used, clears ZWAIT and ZBREQ, and sets the protect mode bits to a "01", indicating that the request is to be governed by the write protection parameters in the MCU. The controller then sets the zone control bits in ZCB that correspond to any of the halfword-modified flags in ZBM, and sets OAFUL to indicate that a request is being made to memory.

If ZBREQ is set and a ZFILN operation is required, the controller examines ZWAIT to determine if the Z buffer data will be stored or held. If ZWAIT is set, the controller continues examining the requests from the X and Y buffers. If ZWAIT is not set, the controller sets ZWAIT to indicate that a ZFILN fetch is in progress, enables the next clock to transfer the address in ZBA to the OA register for transfer to memory, sets the queue code to "10", indicating a destination of the Z buffer for the returning octet, and sets the protect mode bits to "01" to indicate to the MCU that this particular read operation is to be governed by the write protect bits (since the final result will be a write into that location). The controller then clears ZCB indicating a read operation to the MCU, sets OAFUL to indicate that a memory request is in progress, and increments the queue input pointer for the next operation.

If no Z buffer operation is to be performed during this clock period, the controller examines the XBREQ, YBREQ and YNEXT signals. YNEXT resolves conflicts between X and Y data streams when simultaneous requests appear from both (YNEXT selects YBREQ; not YNEXT selects XBREQ). For either request, the controller transfers the memory address from the corresponding address register (XBA or YBA) into the OA register for transfer to memory, clears the request flag (XBREQ or YBREQ), toggles the YNEXT indicator so that the other buffer request will be selected during the next conflict, sets the protect mode bits to "10" to designate the read protect parameters to the MCU, and loads the corresponding code into the queue to indicate the origin of the request and ultimate destination of the octet from memory (X buffer = 00, Y buffer = 01). CMR then clears the zone control bits for the read operation, sets OAFUL to indicate that a memory operation is in progress, and increments the queue input pointer (CUEIP) to select the queue position for the next operation.

Regardless of the path through the decode cycle, the controller completes the control cycle by inspecting the SCFUL and WRACC flags. If SCFUL is set, the controller increments the queue output pointer (CUEOP) since the code has been used to select the destination of the new octet. If WRACC is present (write acknowledgement from the MCU), the last write operation is complete. The controller, therefore, clears ZBBSY indicating that the ZB buffer no longer contains needed data and that a new octet may be transferred into ZB for storage into memory.

4-308 PP RESPONSE POLLING OF THE CP

The Peripheral Processor (PP) monitors the CP response bits in its CR File, Register 12 (hexadecimal). These response bits are System Error (SE), Abnormal Termination (AT), Message Complete (MC) and Switch Complete (SC). If the CP sets one of these bits, the PP performs a set of actions to recover from the condition that produced the response bit. Figure 4-63 illustrates the polling loop that the PP follows. The following paragraphs describe the PP response to each of the conditions.

4-309 SYSTEM ERROR

If the CP detects a parity error during normal operation, it sets the SE response bit. The PP then checks the PE bit in the condition byte to determine if the parity error occurred during a memory fetch. If PE is set, the PP indicates a parity error termination, issues a reset command to the CP, loads a new job into the CP via an exchange intermediate CCR command, and sets the CP run bit to start the new job into operation. If PE is not set, the PP resets the terminate request bit in the control byte, loads a new job into the CP and sets the Run bit to start the new job into operation.

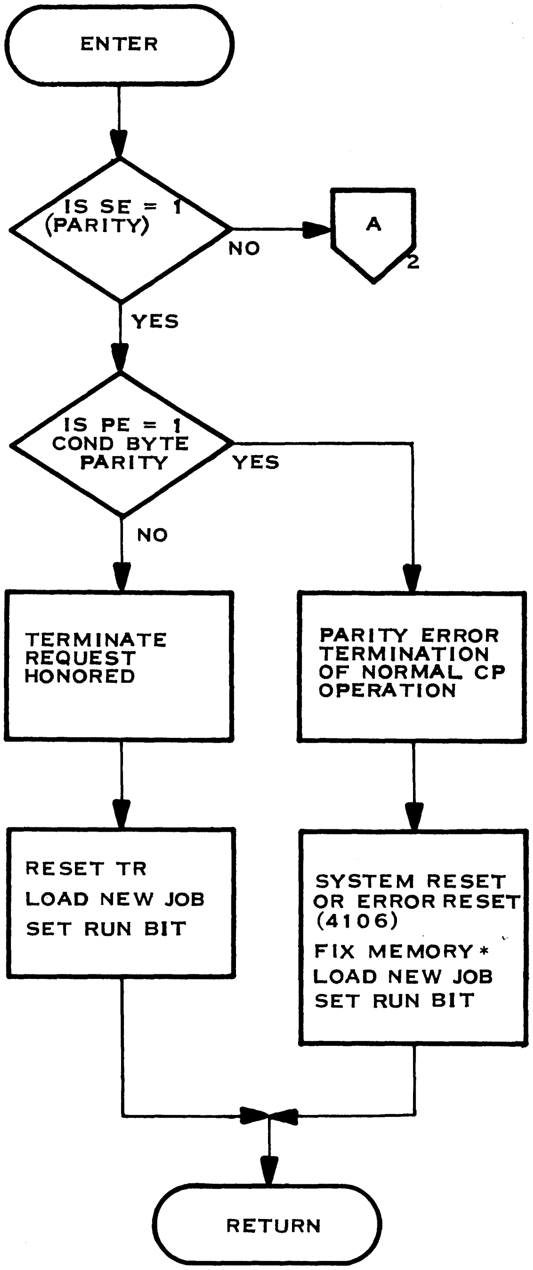
4-310 ABNORMAL TERMINATION

If a maintenance command executing in the CP encountered an error condition and terminated abnormally, the CP will set the AT bit in the CP response byte of the PP CP File. If this bit sets, the PP inspects the SC and MC bits to determine the condition prior to termination.

If SC and MC are both clear, the CP is inactive between jobs and is ready to receive new commands. If MC is clear but SC is set, and error switch was in progress when the CP terminated the sequence. If MC is set, a monitor call resulted in program termination. The condition of the SC bit indicates which call was present: SC = 1 indicates a MCW operation; SC = 0 indicates a MCP operation. The PP then determines what conditions caused the termination by inspecting the PE and PV condition byte bits. If PE is set, then a memory parity error or other system error caused the termination. The condition in memory must be corrected before the job can run in that area of memory. The PP loads a new job into the CP and sets the Run bit to start the CP on the new job. If the PV bit is set, then a memory operation encountered a memory protect violation causing the CP program to terminate. The memory protect parameters in the MCU must be adjusted to allow the program to access the required area in memory. The PP loads a new job into the CP and sets the Run bit to start the operation in the CP. If neither of the two condition byte bits are set, the termination resulted from a termination request issued by the PP. The PP resets the Termination Request bit (TR). loads a new job into the PP and sets the Run bit to start the new job into the CP.

4-311 NORMAL TERMINATION

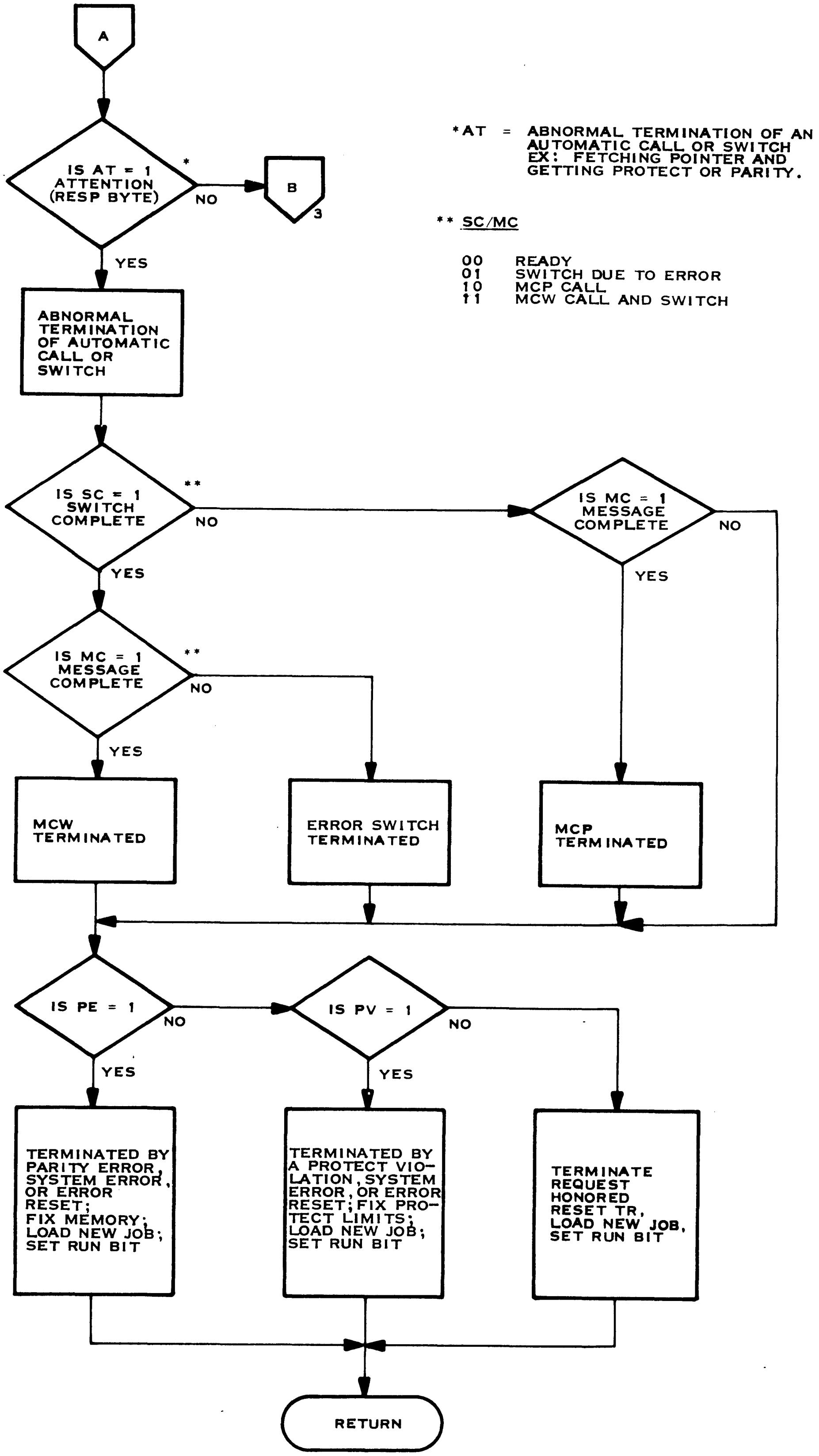
If no system error or abnormal termination occurs, the PP inspects the SC and MC response bits to determine if a switch operation or call instruction executed properly. If neither of these two bits is set, some condition has


ENTER
IS SE = 1 (PARITY)
NO
A
2
YES
IS PE = 1 COND BYTE PARITY
YES
NO
TERMINATE REQUEST HONORED
PARITY ERROR TERMINATION OF NORMAL CP OPERATION
RESET TR
LOAD NEW JOB
SET RUN BIT
SYSTEM RESET OR ERROR RESET (4106)
FIX MEMORY*
LOAD NEW JOB
SET RUN BIT
RETURN

*FIX MEMORY—IF PARITY OCCURS, IT IS ASSUMED THAT MEMORY IS BROKEN AND REQUIRES REPAIR AND THEREFORE NO ATTEMPT TO SWITCH IS MADE.

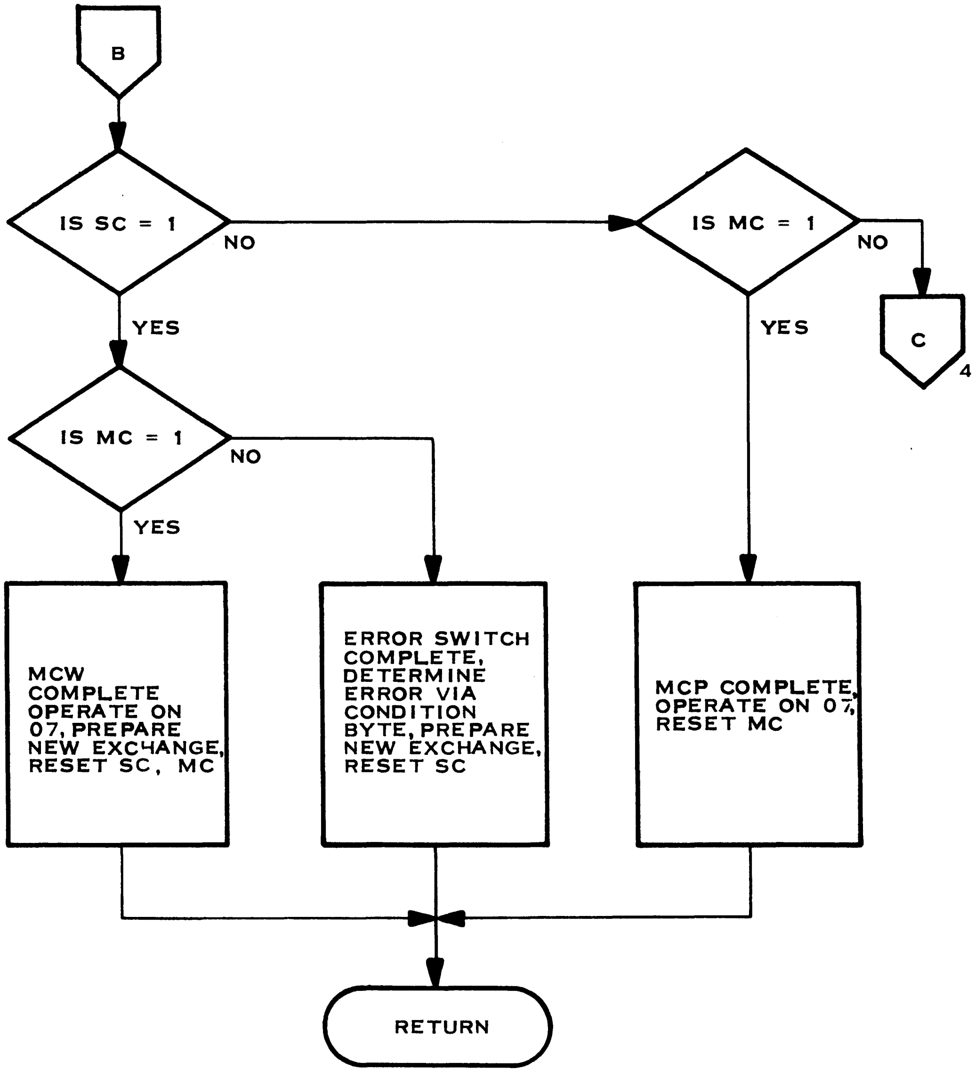
(A)115842

Figure 4-63. PP Automatic Interrupt or Polling Loop of CP Status (Sheet 1 of 4)


A
IS AT = 1 ATTENTION (RESP BYTE) *
NO
B
3
YES
ABNORMAL TERMINATION OF AUTOMATIC CALL OR SWITCH
*AT = ABNORMAL TERMINATION OF AN AUTOMATIC CALL OR SWITCH EX: FETCHING POINTER AND GETTING PROTECT OR PARITY.
** SC/MC
00 READY
01 SWITCH DUE TO ERROR
10 MCP CALL
11 MCW CALL AND SWITCH
IS SC = 1 SWITCH COMPLETE **
NO
YES
IS MC = 1 MESSAGE COMPLETE
NO
YES
IS MC = 1 MESSAGE COMPLETE **
NO
YES
MCW TERMINATED
ERROR SWITCH TERMINATED
MCP TERMINATED
IS PE = 1
NO
YES
IS PV = 1
NO
YES
TERMINATED BY PARITY ERROR, SYSTEM ERROR, OR ERROR RESET; FIX MEMORY; LOAD NEW JOB; SET RUN BIT
TERMINATED BY A PROTECT VIOLATION, SYSTEM ERROR, OR ERROR RESET; FIX PROTECT LIMITS; LOAD NEW JOB; SET RUN BIT
TERMINATE REQUEST HONORED RESET TR, LOAD NEW JOB, SET RUN BIT
RETURN

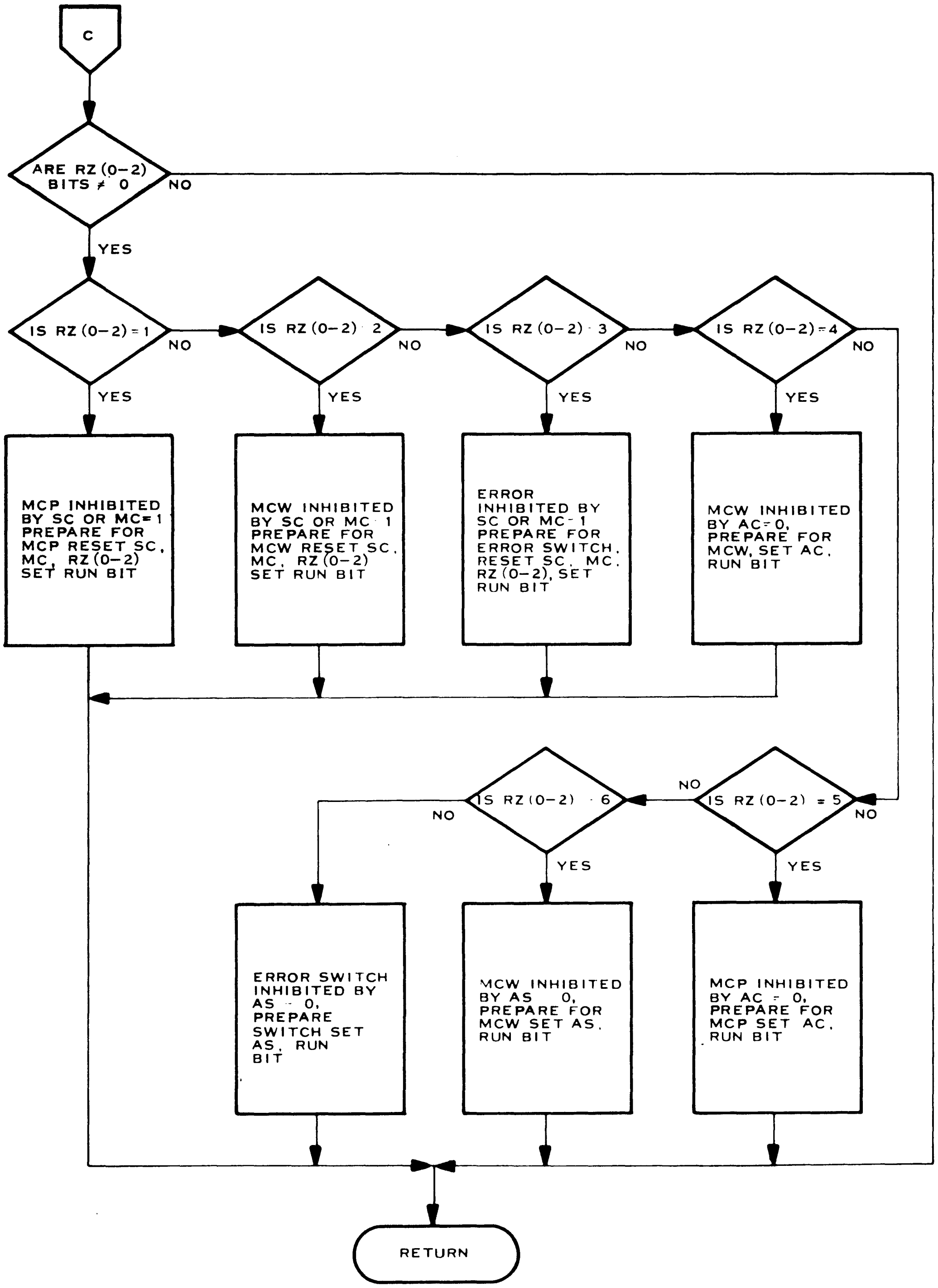
(A)115843

Figure 4-63. PP Automatic Interrupt or Polling Loop of CP Status (Sheet 2 of 4)


B
IS SC = 1
NO
IS MC = 1
NO
C
4
YES
YES
IS MC = 1
NO
YES
MCW COMPLETE OPERATE ON 07, PREPARE NEW EXCHANGE, RESET SC, MC
ERROR SWITCH COMPLETE, DETERMINE ERROR VIA CONDITION BYTE, PREPARE NEW EXCHANGE, RESET SC
MCP COMPLETE, OPERATE ON 07, RESET MC
RETURN

(A)115844

Figure 4-63. PP Automatic Interrupt or Polling Loop of CP Status (Sheet 3 of 4)


C
ARE RZ (0–2) BITS ≠ 0
NO
YES
IS RZ (0–2) = 1
NO
YES
IS RZ (0–2) = 2
NO
YES
IS RZ (0–2) = 3
NO
YES
IS RZ (0–2) = 4
NO
YES
MCP INHIBITED BY SC OR MC = 1 PREPARE FOR MCP RESET SC, MC, RZ (0–2) SET RUN BIT
MCW INHIBITED BY SC OR MC = 1 PREPARE FOR MCW RESET SC, MC, RZ (0–2) SET RUN BIT
ERROR INHIBITED BY SC OR MC = 1 PREPARE FOR ERROR SWITCH, RESET SC, MC, RZ (0–2), SET RUN BIT
MCW INHIBITED BY AC = 0, PREPARE FOR MCW, SET AC, RUN BIT
IS RZ (0–2) = 6
NO
YES
IS RZ (0–2) = 5
NO
NO
YES
ERROR SWITCH INHIBITED BY AS = 0, PREPARE SWITCH SET AS, RUN BIT
MCW INHIBITED BY AS = 0, PREPARE FOR MCW SET AS, RUN BIT
MCP INHIBITED BY AC = 0, PREPARE FOR MCP SET AC, RUN BIT
RETURN

(A)115845

Figure 4-63. PP Automatic Interrupt or Polling Loop of CP Status (Sheet 4 of 4)

prevented completion of the operation. The PP inspects the reason code bits to determine the cause. If SC is set, but MC is not, then the CP has completed an error switch operation. The PP must examine the condition byte to determine the nature of the error, prepare a new job for exchange operations in case the current job in the CP requires PP intervention, and reset the SC bit. If MC is set, then a monitor call operation has successfully completed. The condition of the SC bit indicates which of the two monitor calls was completed. In either case, the PP pulls the call pointer from memory location 07 and performs the required operation to fulfill the CP's requirements. The PP also resets the MC bit, and, if the operation was an MCW, prepares a new exchange to be ready in case the current program in the CP completes or requests context switching via another MCW operation.

4-312 CAPTURE CCR

The capture CCR logic is part of the master hard core circuitry that monitors the transfer bit (TB) and unit code of the Common Command Register (CCR) in the Peripheral Processor Communications Register File. The flowchart in figure 4-64 illustrates the control cycle. If TB sets, the PP is issuing a command to one of the system devices. The controller then inspects the unit code of the command to determine if the command is intended for the CP. A code of 41 (in hexadecimal) specifies a CP command. The controller then inspects the Request Present flag (QRPF) to determine if another request is currently being processed by the hard core logic. If this flag is set, the controller must wait until it clears before proceeding. When the flag clears, the controller activates the Reset TB (RSTB) and Gate CCR (GCCR) lines to the CR File to transfer the new command into the CP. When the PP returns a recognition of the transfer (TBRL), the controller deactivates the two signals and activates the Request Present indicator (RP) to indicate to the hard core logic that a new command is resident. The controller then ensures that the TB has not yet reset.

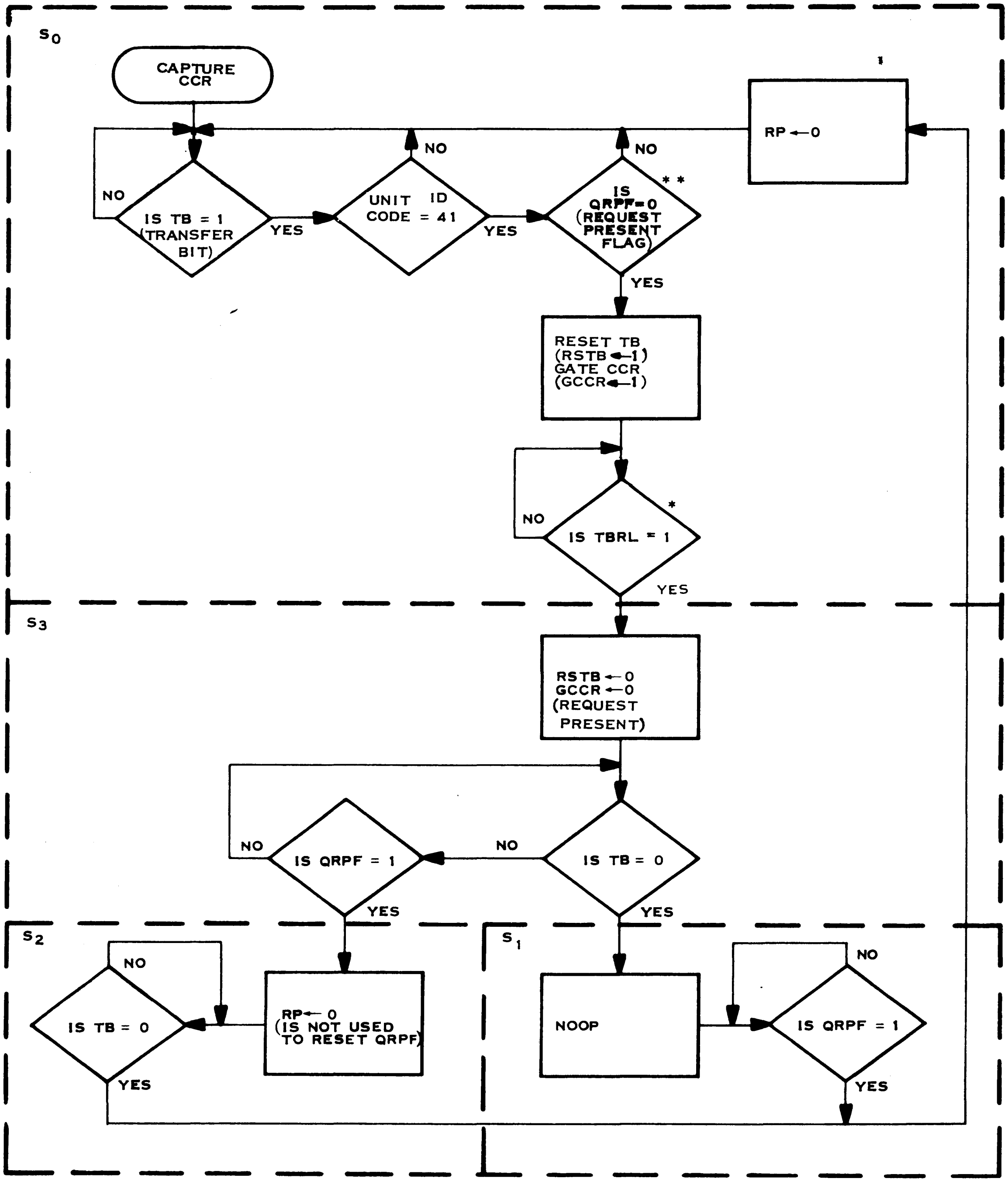
NOTE

> Due to the long clock period of the CR File with respect to the CP, the CP has several clock periods before TB resets. Therefore, if TB is reset at this point, it should not have been set. This feature also provides a time-out that negates the CCR command if QRPF does not set within a reasonable time.

If TB is still set, the controller waits for the hard core logic to set QRPF as a result of RP being active. QRPF indicates an active command in the hard core logic. When QRPF sets, the controller drops RP and waits for TB to clear. When TB clears, the cycle is complete and the controller is ready to begin the cycle again.

4-313 ERROR MONITOR

The error monitor logic, illustrated in figure 4-65, determines the conditions in the CP hard core interface and sets the reason code bits to the PP to indicate the status of a context switch in the CP. If an error occurs in the



NOTE: THIS IS ALL ASYNCHRONOUS LOGIC. (1→ XXX REQUIRES NO CLOCK)

*TBRL —LATCH OUTPUT ON CCR COOKIE SHEET THAT LOOKS AT ALL TB RESET LINES AND WHEN TBR CLK OCCURS IT GETS SET SO THAT THE MHC KNOWS HOW LONG TO HOLD RSTB = 1 DUE TO CCR 500 NS CLOCK

**QRPF —GETS RESET VIA COMMAND COMPLETE IN SYNCHRONOUS LOGIC STATE 7.

(A)115837

Figure 4-64. Capture CCR Logic Flowchart

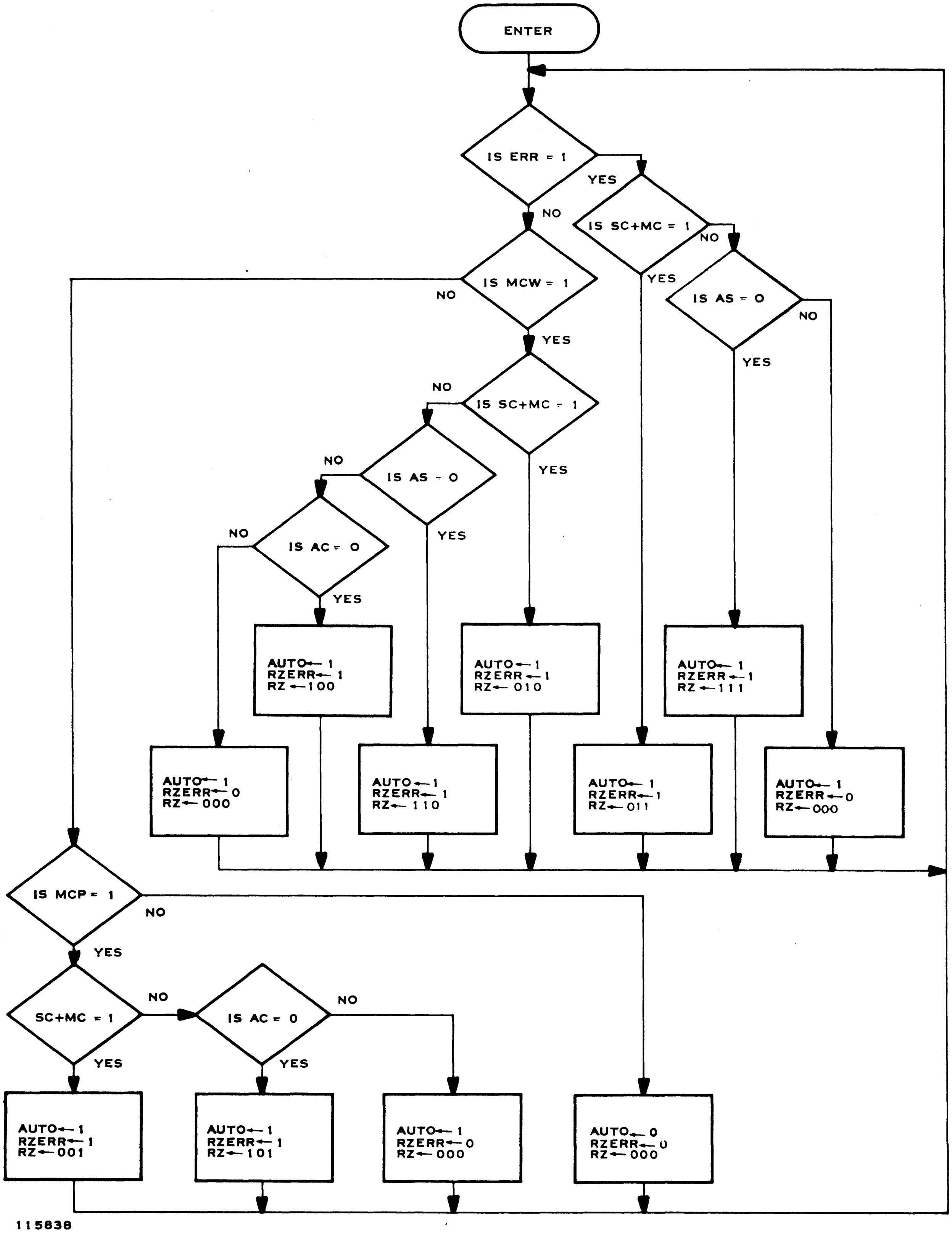


Figure 4-65. Monitor Flowchart

program currently executing in the CP, the program should be switched out of the CP and a new program loaded into the CP to conserve processor time. The monitor, therefore, checks the SC and MC bits from the PP. If either of these bits is set, then the PP has not as yet recovered from the last program switch to ready a new program for the current switch. The controller sets the reason code bits to "011" to indicate this condition. If however, the PP has prepared a new program, the controller examines the AS control bit from the PP. If Allow Switch is a zero, the PP is prohibiting a context switch. The controller sets the reason code bits to "111" to indicate that an error has occurred, but has not been switched out of the CP. If AS is not zero, the controller clears the reason code bits and allows hard core logic to initiate the context switch.

The decision paths are similar for MCW or MCP instructions in the program sequence. These instructions are also dependent upon Allow Call (AC) being set, so an inspection of that bit is also made. Since an MCP does not initiate an immediate switch of programs, the AS bit does not have to be active for successful completion of the instruction. If no error, MCW or MCP is encountered during the control cycle, the controller clears the reason code bits and returns to the start of the control cycle for the next clock.

4-314 SEQUENCE CONTROL

Sequence control monitors the program progress in the CP, initiates context switches when errors occur, decodes CCR commands from the CR file and issues commands to the unit hard core controllers to execute the commands. Before initiating any operation, sequence control examines the control byte from the CR file to determine if that operation is permitted. The sequence control flowchart appears in figure 4-66 and table 4-9 defines the acronyms used in the flowchart.

4-315 STATE 0

State 0 of sequence control monitors the conditions in the CP during normal processing and waits for a CCR command from the PP. If a program error, MCW, MCP or system error occurs during normal processing, the controller exits to the state that performs the required steps for that condition. If a CCR command enters from the PP, the controller decodes the command and exits to the state required to initiate that command. During normal operation, the controller monitors the Run Bit to ensure that normal operation is in progress. It then checks the System Error indication and exits to state 1 if a system error has occurred. If no system error occurs, the controller checks AUTO from the Error Monitor circuit. If this signal is active, the controller determines which condition caused AUTO (by examining the other indicator bits from the Error Monitor) and exits to the proper state. If AUTO is not set, the controller examines QRPF. If this flag is set, then the Capture CCR logic has detected and captured a CCR command from the PP. The controller determines that the PP has not terminated the request, and then decodes the command. If the command is a simple one-step command (lock or unlock PC, set or reset all registers, or reset error cells), the controller exits to state 2 to enable the unit hard core controllers to perform their functions. If the command is an intermediate or status command, the controller ensures that the CP has not

Table 4-9. Sequence Control Acronyms

Term	Function
AB	Abnormal - Condition byte bit to PP that indicates that the last CCR command terminated abnormally.
ABORT	Signal to unit hard core controllers that prevents them from further processing of any CCR command.
ABTERM	Signal from the unit hard core controllers that indicates an abnormal termination of a unit command.
AS	Allow Switch - CP Control Byte bit from PP that enables automatic switching for MCW and errors.
AT	Abnormal Termination - Response Byte bit that indicates to the PP that a switch or call terminated abnormally.
AUTO	Signal from Error Monitor that indicates an MCW, MCP or an error condition in the CP.
CC	Command Complete - Condition Byte bit that indicates the completion of the last CCR command.
CCRO	CCR Output Register - Register in master hard core that supplies instruction codes to unit hard core controllers.
CHKCMD	Check Command - Internal signal indicating that the CCR command is either a status or intermediate maintenance transfer (CCR codes 4108 through 410D).
CLRINP	Signal that clears the Hard Core In Progress flag in the unit hard core controllers.
CLRREQ	Clears the unit hard cores Hard Core Requirement flag.
CSR	Context Switch Response - signal from PP indicating that map and protect parameters are set up in MCU for a switch.
CSW	Context Switch - signal to PP indicating a requirement for new map and protect parameters in the MCU for a context switch.
ERRF	Internal flag that indicates a program error switch is being performed.
EXCHCMD	Exchange Command - internal signal that indicates that the CCR command involves both a load and a store (exchange) - CCR codes 410A and 410D.
EXCHF	Internal flag that indicates that the hard core logic is processing an exchange command from the PP.

Table 4-9. Sequence Control Acronyms (Continued)

Term	Function
HCCALL	Call permission indicator to level 3 controller to enable IPU to write the call pointer.
HCINIT	Hard Core Initiate - Starts hard core operation in the unit hard core controllers.
MC	Message Complete - Response Byte bit that indicates that the CP has completed an MCW or MCP.
MCWF	Internal flag indicating that the controller is performing an MCW operation.
MEMCMD	Internal signal that indicates that the CCR command is either a load or store operation - CCR codes 4100, 4101, 410E and 410F.
QRPF	Request in Progress flag - Set by the Capture CCR logic indicating that a CCR command is active in the CP. Cleared at the completion of command by Sequence control.
RB	Run Bit - When set, enables CP processing.
RIPF	Internal flag indicating that the command in progress will reset the run bit.
RZF	Internal flag indicating that an error reason code will be sent to the PP.
SC	Switch Complete - Response Byte bit indicating that the CP has completed an MCW or error switch.
SE	System Error - Response Byte bit indicating that a parity error occurred during normal processing.
SETREQ	Set Hard Core Requirement - Sets the Hard Core Requirement flags in the unit hard core controllers so that the units will wind down operations in preparation for a maintenance command.
SIMCMD	Simple Command - Internal signal indicating that the CCR command is a basic command requiring no complex operations - CCR codes 4102 through 4107.
STRB	Set Run Bit - Internal signal indicating a Set Run Bit CCR command.
SYSERR	System Error - Internal signal indicating that a parity error has occurred during CP processing or the PP has set TR.
TR	Terminate Request - Control Byte bit that instructs the CP to cease processing the current CCR command.

Table 4-9. Sequence Control Acronyms (Continued)

Term	Function
UNCMP	Unit Complete - Internal signal that indicates that all CP unit hard core controllers have completed their operations for the current CCR command.
WAIT	Prevents memory requests to keep the CP in a zero request pending state.
ZROPND	Zero Pending - Indicates that all memory requests have been satisfied by returning data from memory (no outstanding requests).

recently completed a context switch that the PP has not recognized (SC or MC set). If this is the case, the program that the PP requested status or intermediate information for is no longer in the CP. The controller, therefore, sets command complete, abnormal termination, clears the request present flag and transfers the condition byte to the PP. If the CCR command is a load or store operation, the controller exits to state 4 to process the request. If the command is a Set Run Bit, the controller sets the Run Bit and command complete, and clears the Request Present flag. If none of the above commands are present, the controller defaults to a clear run bit command which is performed in state 1.

4-316 STATE 1

Four conditions cause the controller to enter state 1. These conditions are:

1. System error during normal operation.
2. Error reason code to be sent to PP.
3. Reset Run Bit command from PP.
4. System error during a call operation.

For each of the above causes, the controller sets the Hard Core Requirement flag in each of the unit hard core controllers so that they will conclude the operations that are currently being processed. If a reason code is to be sent to the PP, the controller also updates the reason code in the reason buffer so that the correct information will be sent to the PP, and sets the RZF flag to indicate that the code will be sent. The RIPF flag is set if the Run Bit will be cleared by the command in progress. The controller then waits for ZROPND to indicate that all outstanding memory requests have returned from memory. During the waiting period, the controller monitors the system error indicator and disables the unit hard core controllers if a system error occurs. The controller then decodes the states of the three indicators; SYSERR, RZF and RIPF, to determine what actions to perform. RZF and RIPF are mutually exclusive flags, whereas, SYSERR could have occurred while waiting for ZROPND. The decode and the resulting actions are listed in the flowchart (figure 4-66). The actions are enabled for the next clock following the decode (state 7).

4-317 STATE 2

The controller enters State 2 when it decodes the CCR command from the PP to be a simple operation requiring no complex transfers in the CP. The controller then issues HCINIT to the unit hard core controllers to start each of the units into their respective sequences, and gates the CCR code into the master hard core's CCR Output register. The controller then waits until each unit hard core controller returns an operation complete indication before it exits to state 7 to issue Command Complete to the PP and clear the Request Present flag (QRPF).

4-318 STATE 3

The controller enters state 3 during call commands. In this state the controller waits while the level 3 controller writes the call message into the designated location (location 07) in memory. When the controller enters state 3, it sets HCCALL to the level 3 controller to enable it to write the message into memory. If the command is an MCW, the controller also sets the MCWF flag. The controller then waits for Call Complete indicating that the message has been written. During the waiting period, the controller monitors the error indicators to ensure that no system or program errors occur. A system error causes the controller to exit to state 1 to terminate the operation. If a program error occurs, the controller may terminate the operation in state 1 if the Allow Switch bit is not set. If AS is set, the controller exits to state 4 to begin a context switch that loads a new program into the CP. If the message is written into memory without error, the controller examines the MCWF flag to determine if it should perform a context switch (MCW), or if it should continue to process the same program (MCP). For an MCP, the controller exits to state 7 to set Message Complete to the PP. For an MCW the controller exits to state 4 to begin the context switch.

4-319 STATE 4

The controller enters state four under four conditions, each of which requires some type of memory transfer (load, store or exchange). When the controller enters state 4, it sets HCINIT prepare the unit hard core controller for the transfer operation, sets WAIT to prevent them from initiating any memory requests, and transfers a code into the CCR Output register for transfer to the UHC's ("0" for context switches, or CCR code for the direct CCR commands). The Exchange flag sets if the operation is an exchange. The controller then waits for all outstanding memory requests to return from memory. During the waiting period, if a system error condition occurs, the controller sends Abort to the unit hard core controllers to halt performance of the CCR command. When ZROPND becomes active, the controller ensures that no system errors have occurred, that a program error has not occurred, or if ERR is set, that the operation is a context switch (ERRF or MCWF). The controller then exits to state 5 for context switch or exchange operations and to state 6 for load or store operations.

4-320 STATE 5

The controller enters state 5 to perform an exchange of CP contents. In this state, the controller sends CSW to the PP to ensure that the map and protect parameters have been established in the MCU. When the PP responds with CSR, indicating that the parameters for the new program are ready, the controller exits to state 6. When the controller enters state 5, it also clears the WAIT flag so that the unit hard core controllers can begin the store portion of the exchange while the sequence controller is waiting for CSR.

4-321 STATE 6

When the controller enters state 6, it clears the WAIT flag and CSW to the PP if CSW was enabled in state 5. Clearing WAIT allows the unit hard core controllers to perform the designated memory transfer operation. For context switches, the new program is loaded while the controller is in state 6; for loads or stores, that operation is performed in state 6. When each of the unit hard core controllers has completed its transfer operation, the controller determines if any of the units terminated abnormally and issues the status and transfer commands required for each of the conditions. These commands are illustrated in the flowchart for sequence control (figure 4-66).

4-322 STATE 7

State 7 enables the status and transfer commands that the controller has determined are required (through examination of the operations in the other states of the controller). These transfer commands and status reports are illustrated in the sequence control flowchart. After enabling these commands, the controller determines if the operation completed by hard core was an error context switch (ERRF). If not, the controller clears all control flags and returns to the initial monitor cycle in state 0. If ERRF is set, the controller exits to state 8 to reinitialize the CP hardware flags.

4-323 STATE 8

The controller enters state 8 from state 7 after completion of an error switch operation. Since an error produced the switch operation and a new program has entered the CP, the system error cells in each of the CP units must be cleared so that they will not affect the operation of the new program. To produce this effect, the controller loads a code of "06" (Reset system error cells) into the CCR Output register, and enables the unit hard cores by setting Hard Core Initiate (HCINIT). The controller remains in state 8 until the unit hard core controllers have completed the operation (UNCMP). The controller then returns to the initial monitor cycle in state 0.

4-324 OTHER CONTROL CIRCUITS

Figures 4-67 through 4-75 illustrate the control cycles for other controllers within the CP. Description of these circuits will be added at a later date.

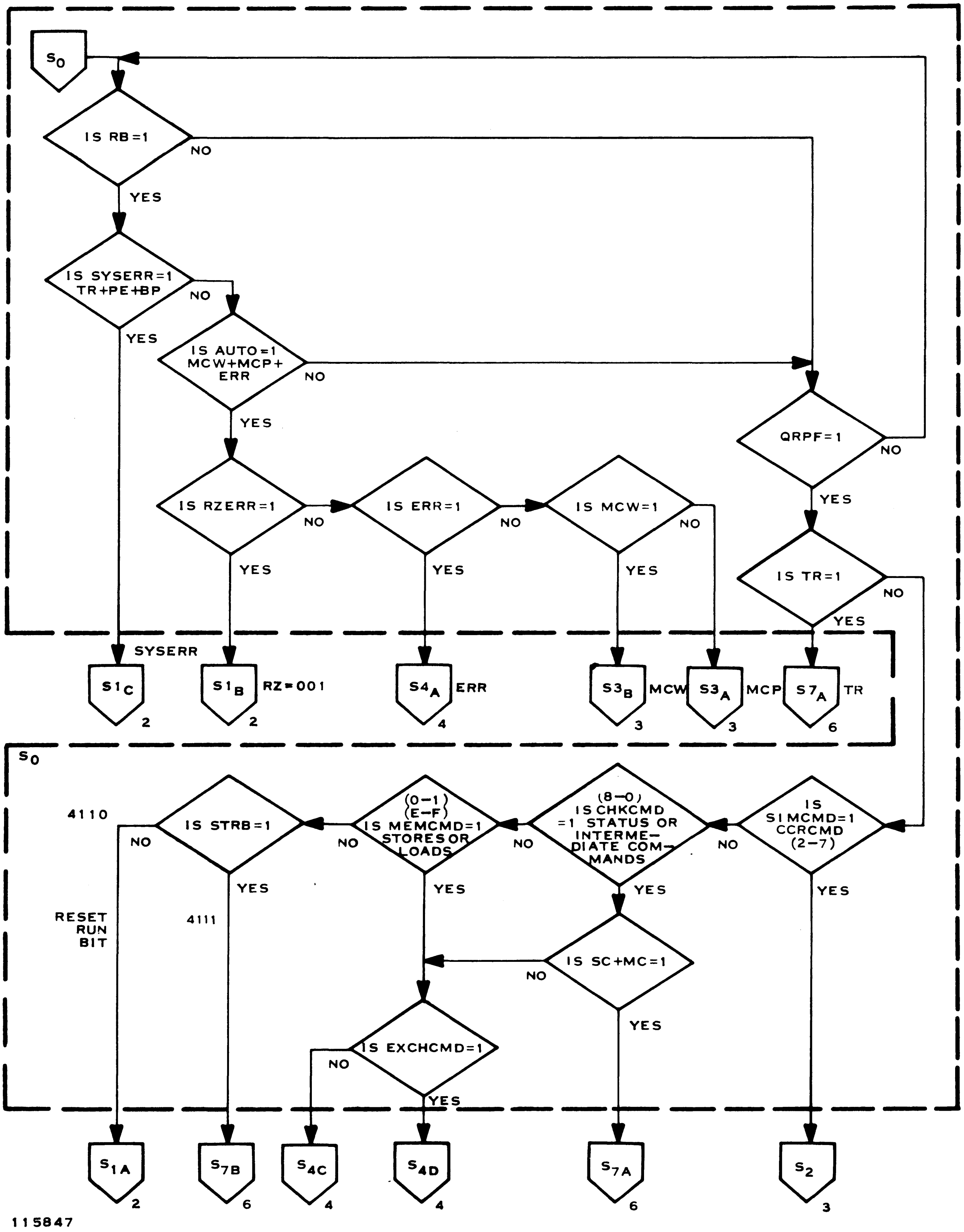
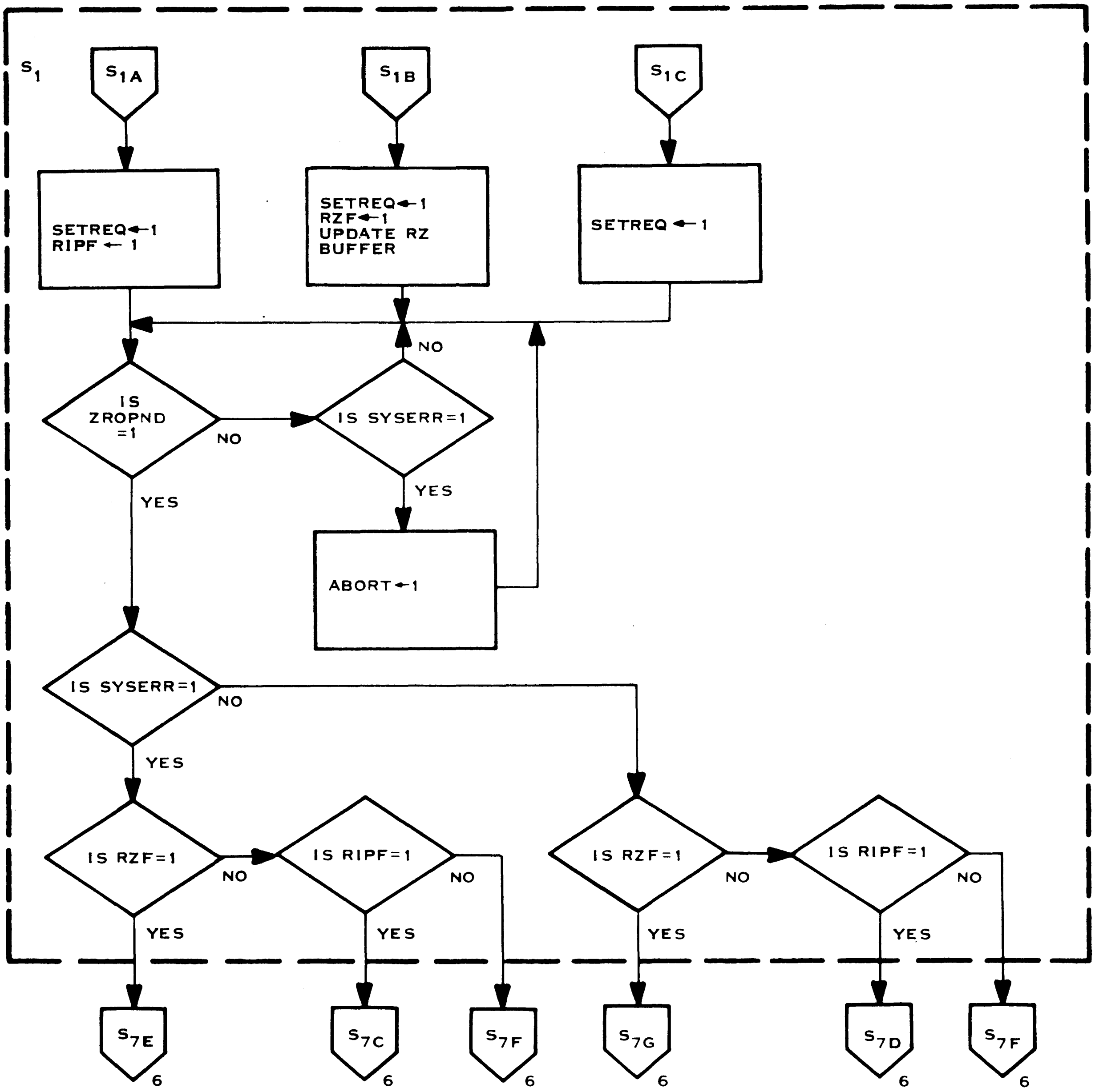


Figure 4-66. Sequence Control Flowchart (Sheet 1 of 7)



115848

Figure 4-66. Sequence Control Flowchart (Sheet 2 of 7)

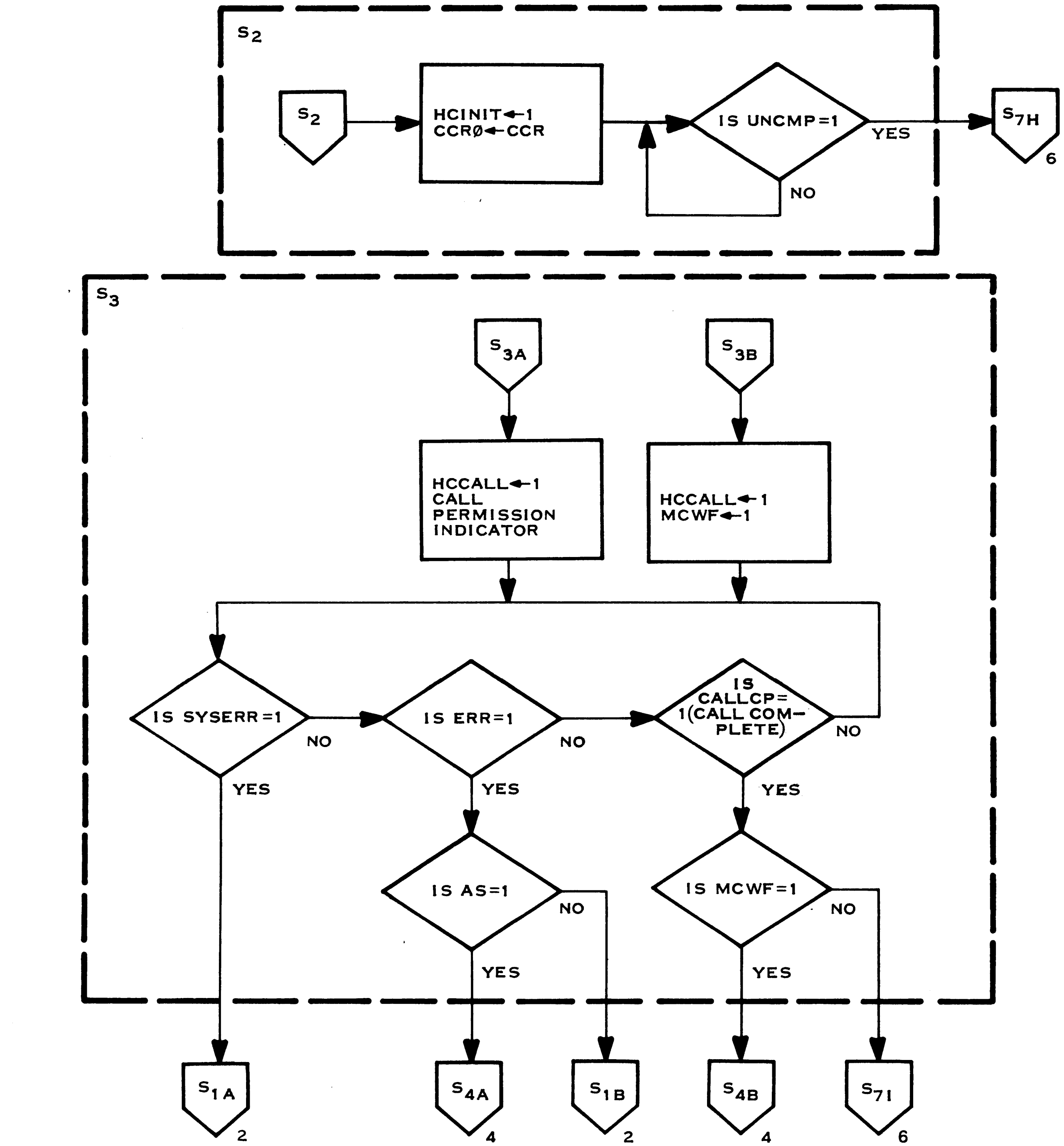
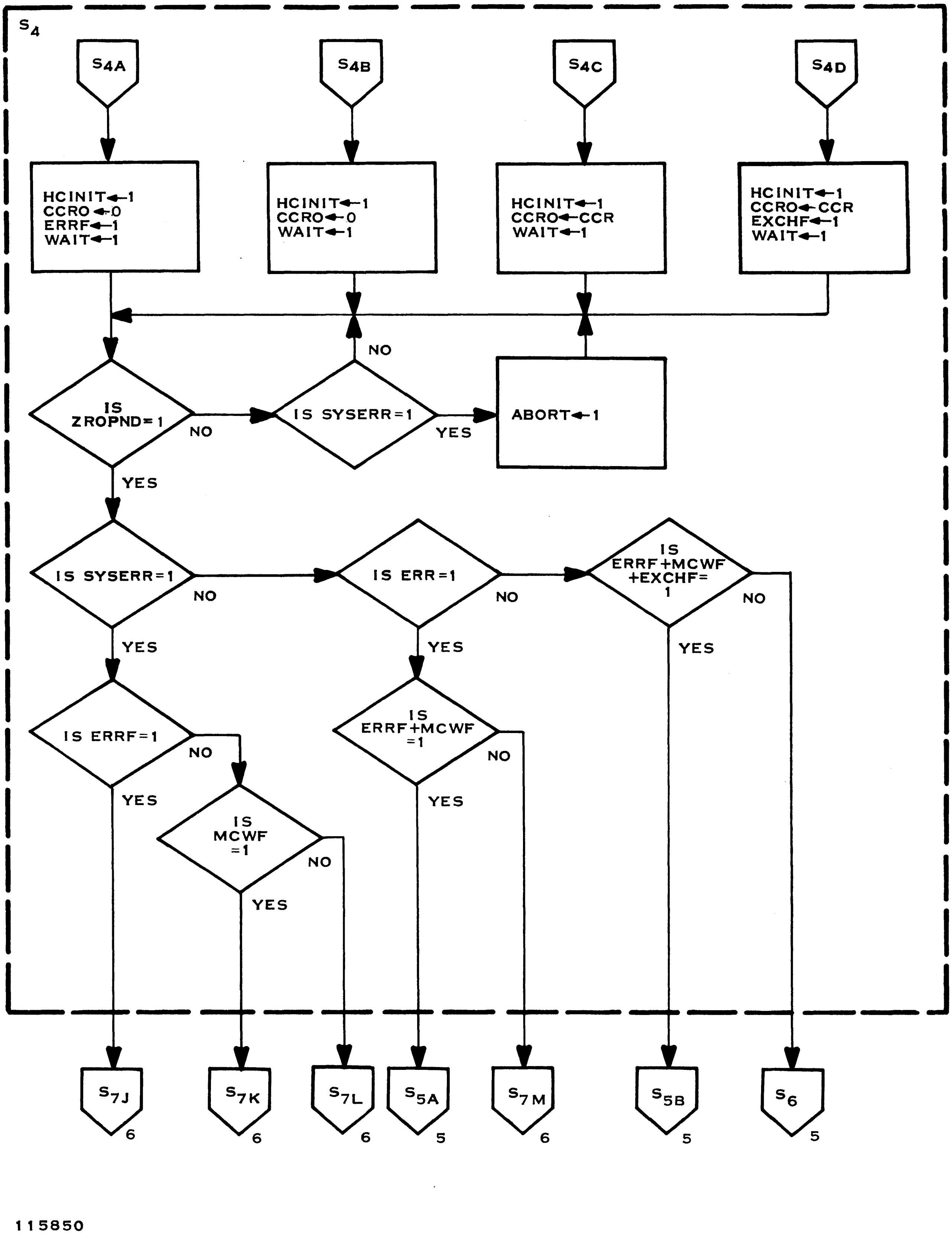
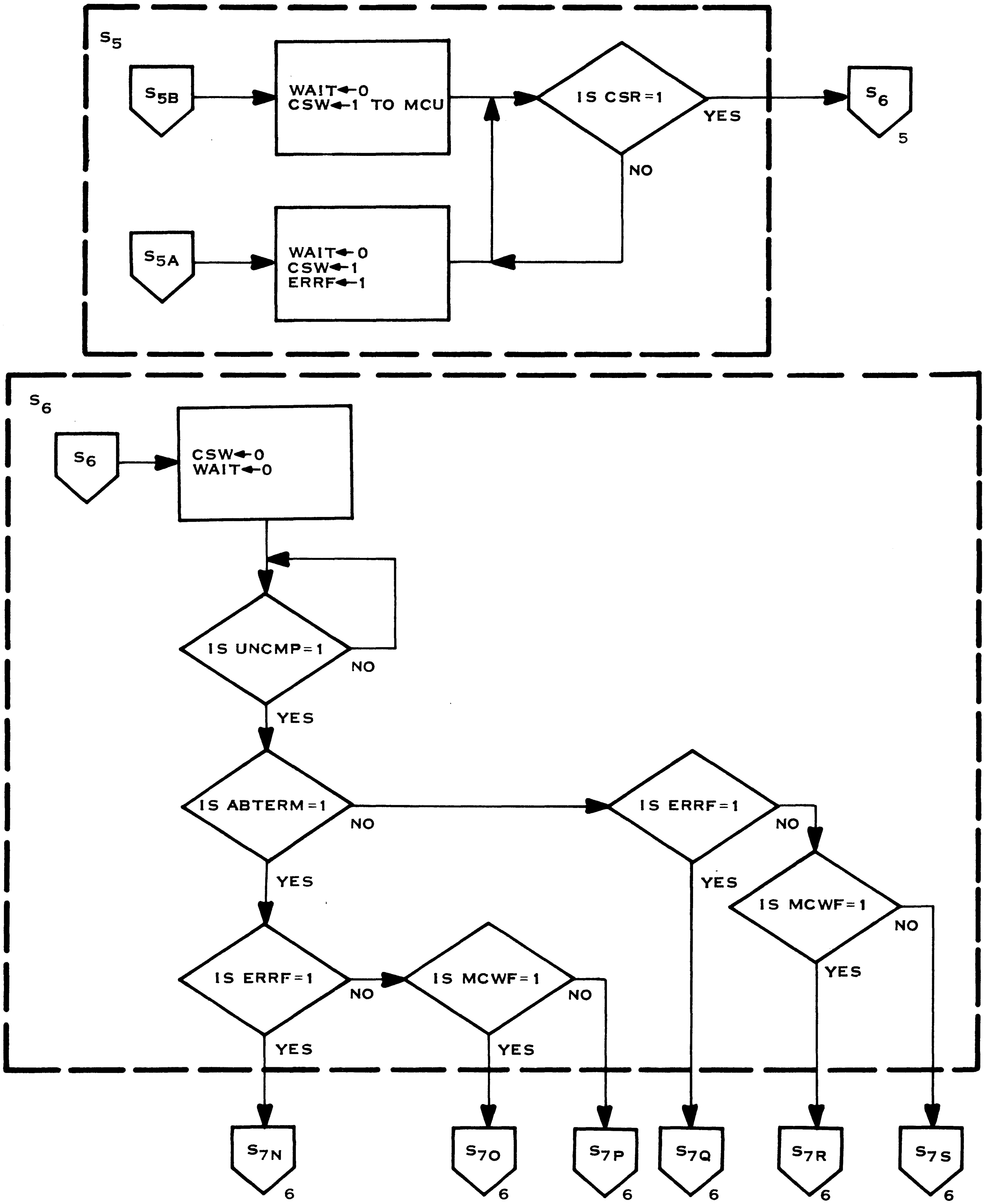


Figure 4-66. Sequence Control Flowchart (Sheet 3 of 7)



115850

Figure 4-66. Sequence Control Flowchart (Sheet 4 of 7)



115851

Figure 4-66. Sequence Control Flowchart (Sheet 5 of 7)

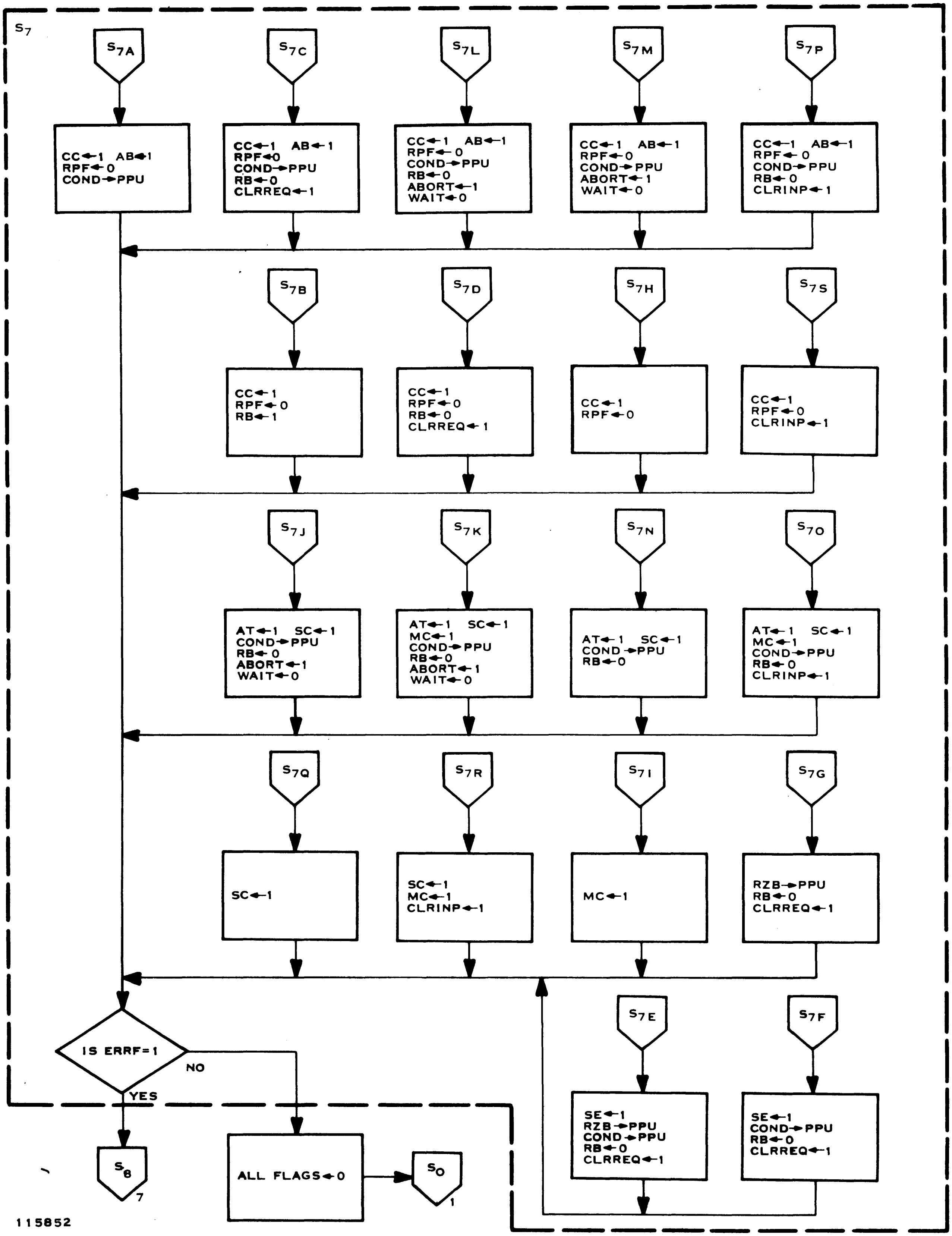
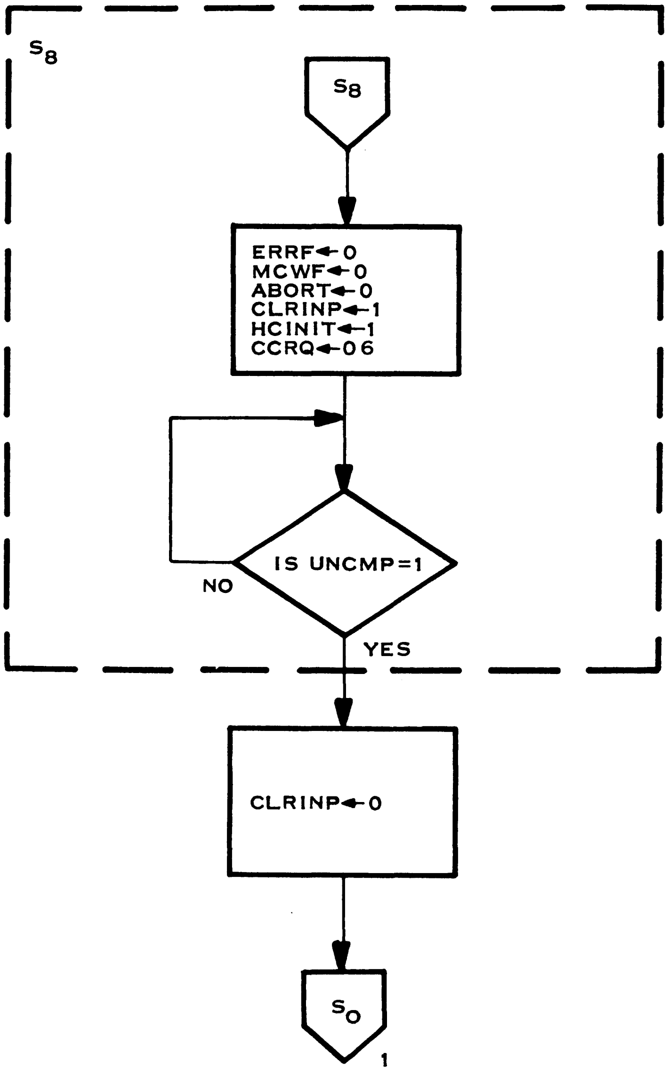
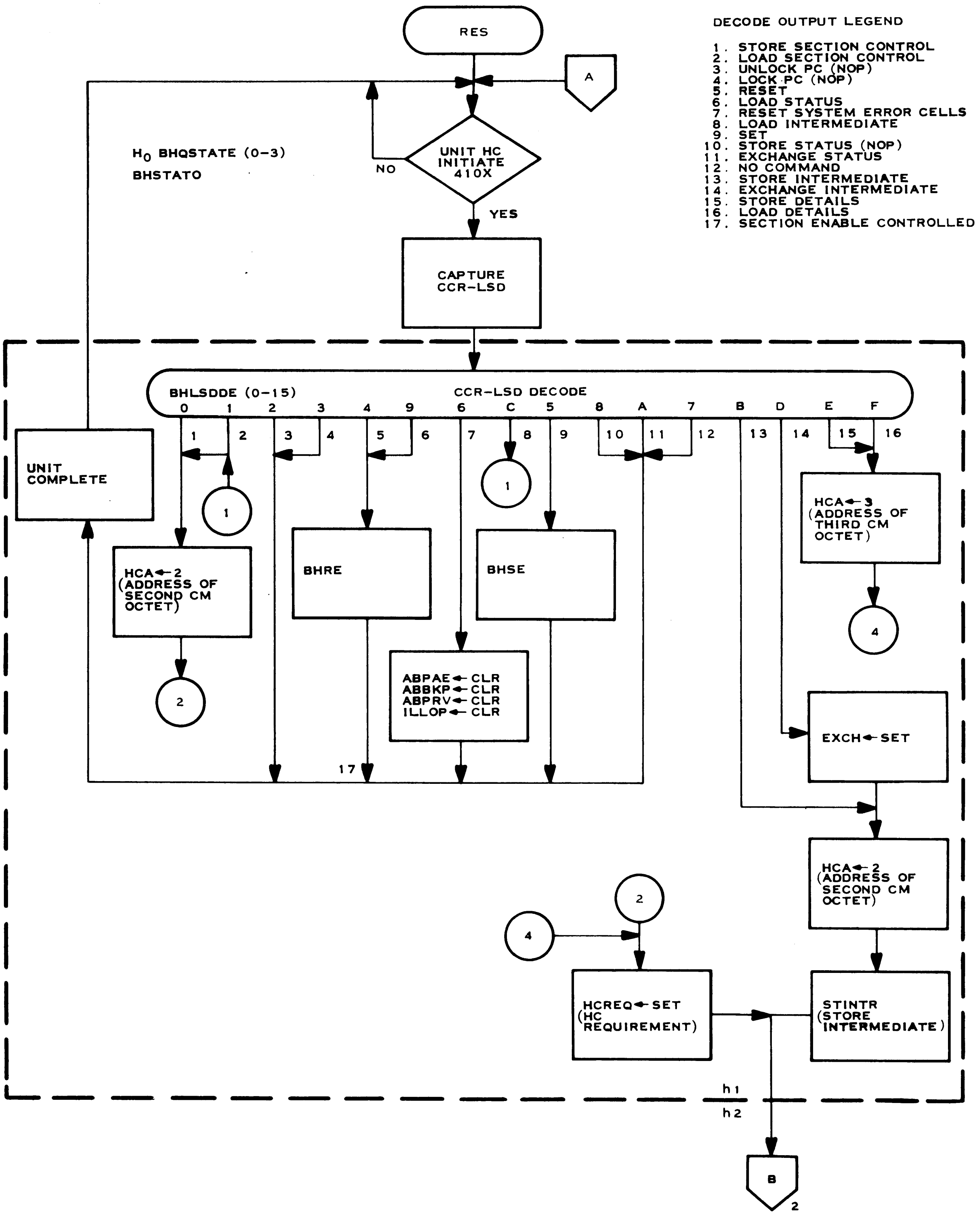


Figure 4-66. Sequence Control Flowchart (Sheet 6 of 7)



115871

Figure 4-66. Sequence Control Flowchart (Sheet 7 of 7)



DECODE OUTPUT LEGEND

1. STORE SECTION CONTROL
2. LOAD SECTION CONTROL
3. UNLOCK PC (NOP)
4. LOCK PC (NOP)
5. RESET
6. LOAD STATUS
7. RESET SYSTEM ERROR CELLS
8. LOAD INTERMEDIATE
9. SET
10. STORE STATUS (NOP)
11. EXCHANGE STATUS
12. NO COMMAND
13. STORE INTERMEDIATE
14. EXCHANGE INTERMEDIATE
15. STORE DETAILS
16. LOAD DETAILS
17. SECTION ENABLE CONTROLLED

(A)115795

Figure 4-67. MBU Unit Hard Core Flowchart (Sheet 1 of 3)

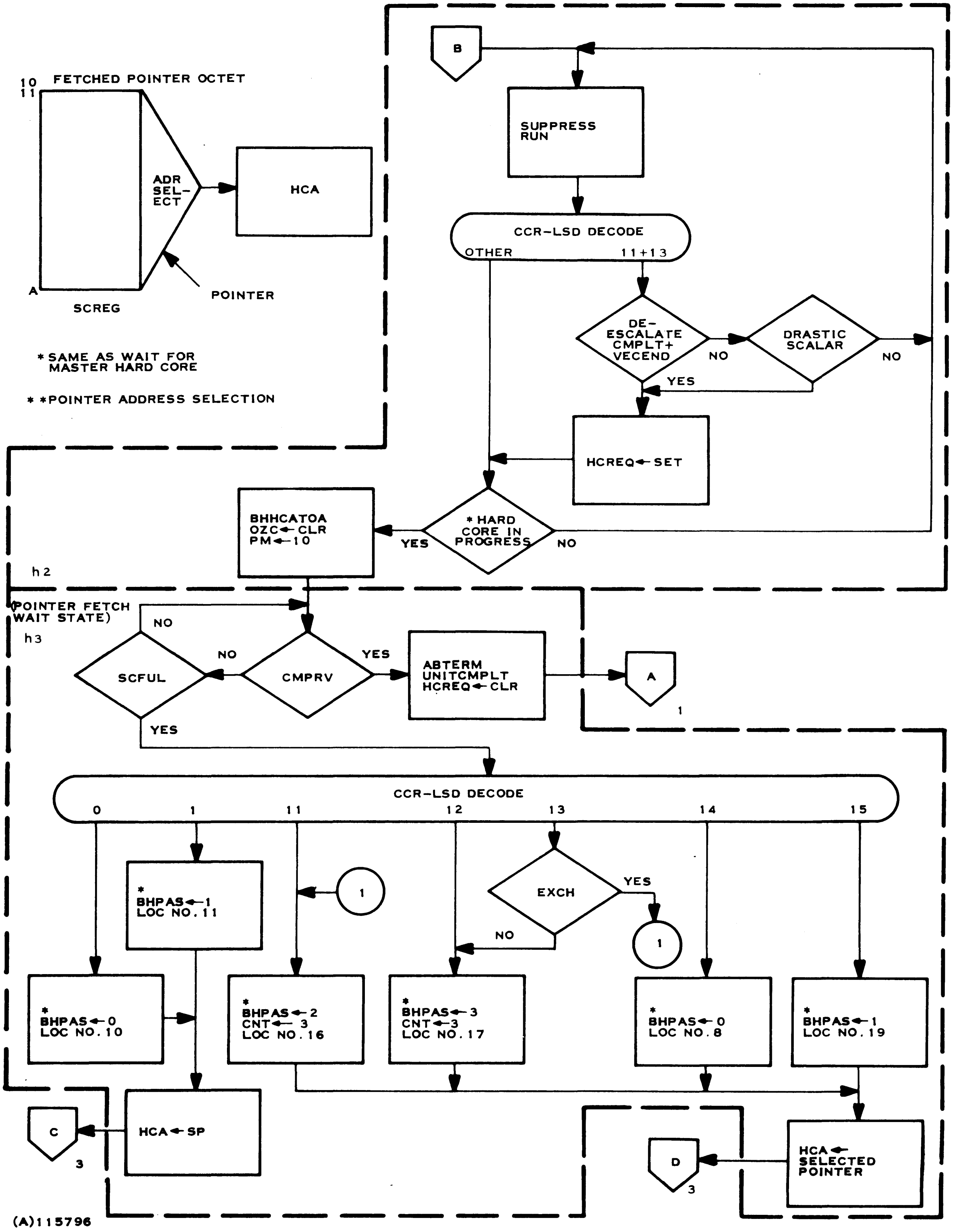


Figure 4-67. MBU Unit Hard Core Flowchart (Sheet 2 of 3)

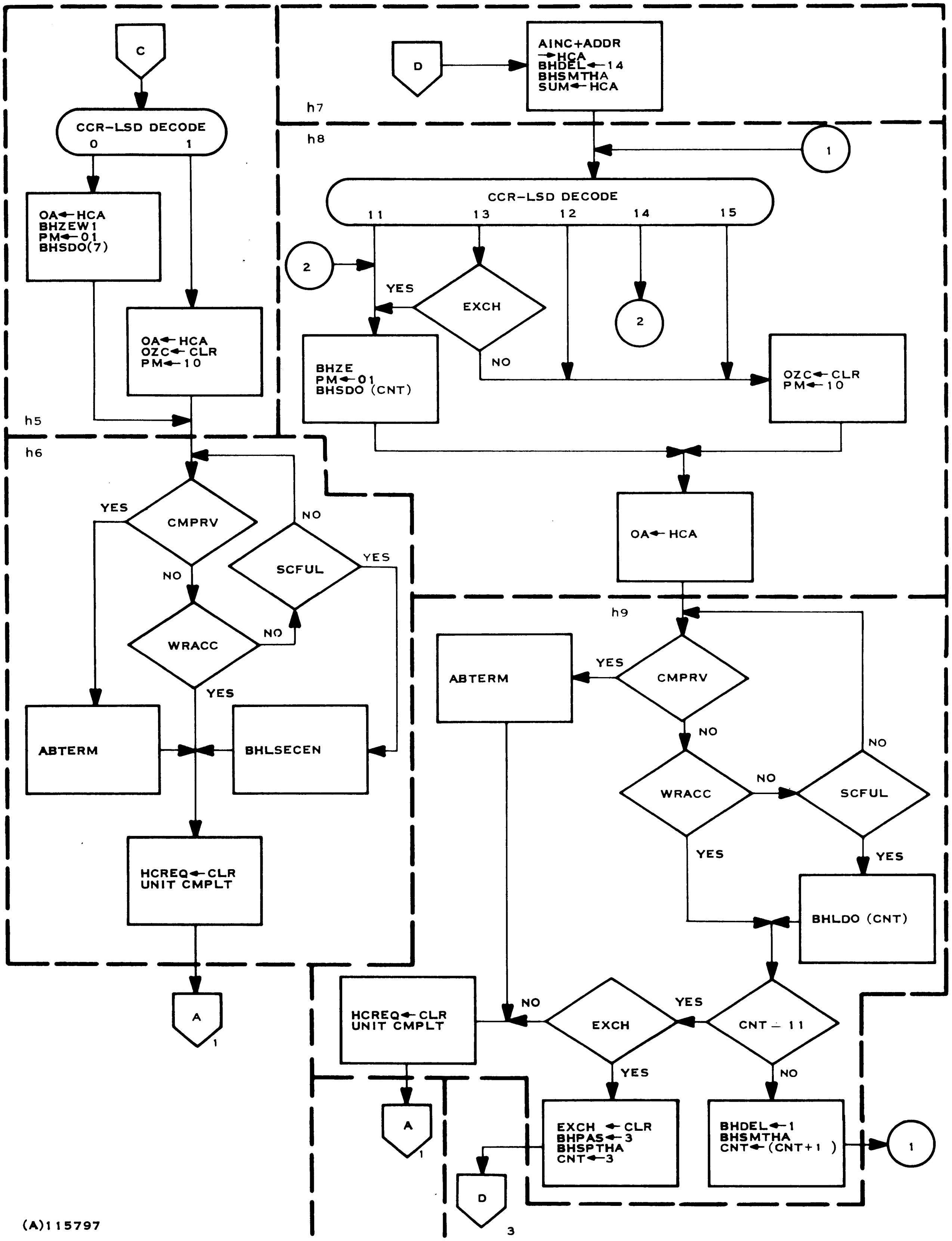
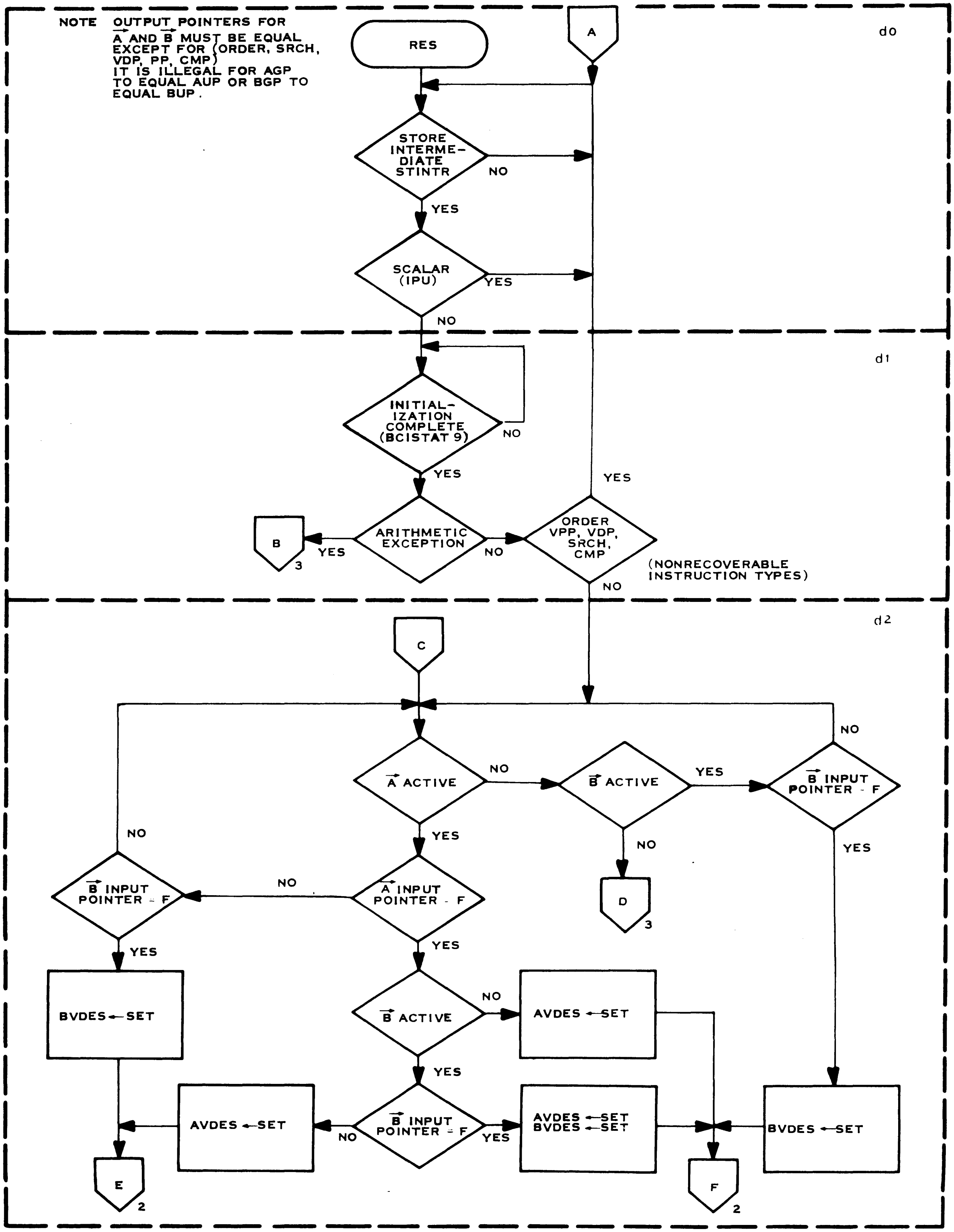
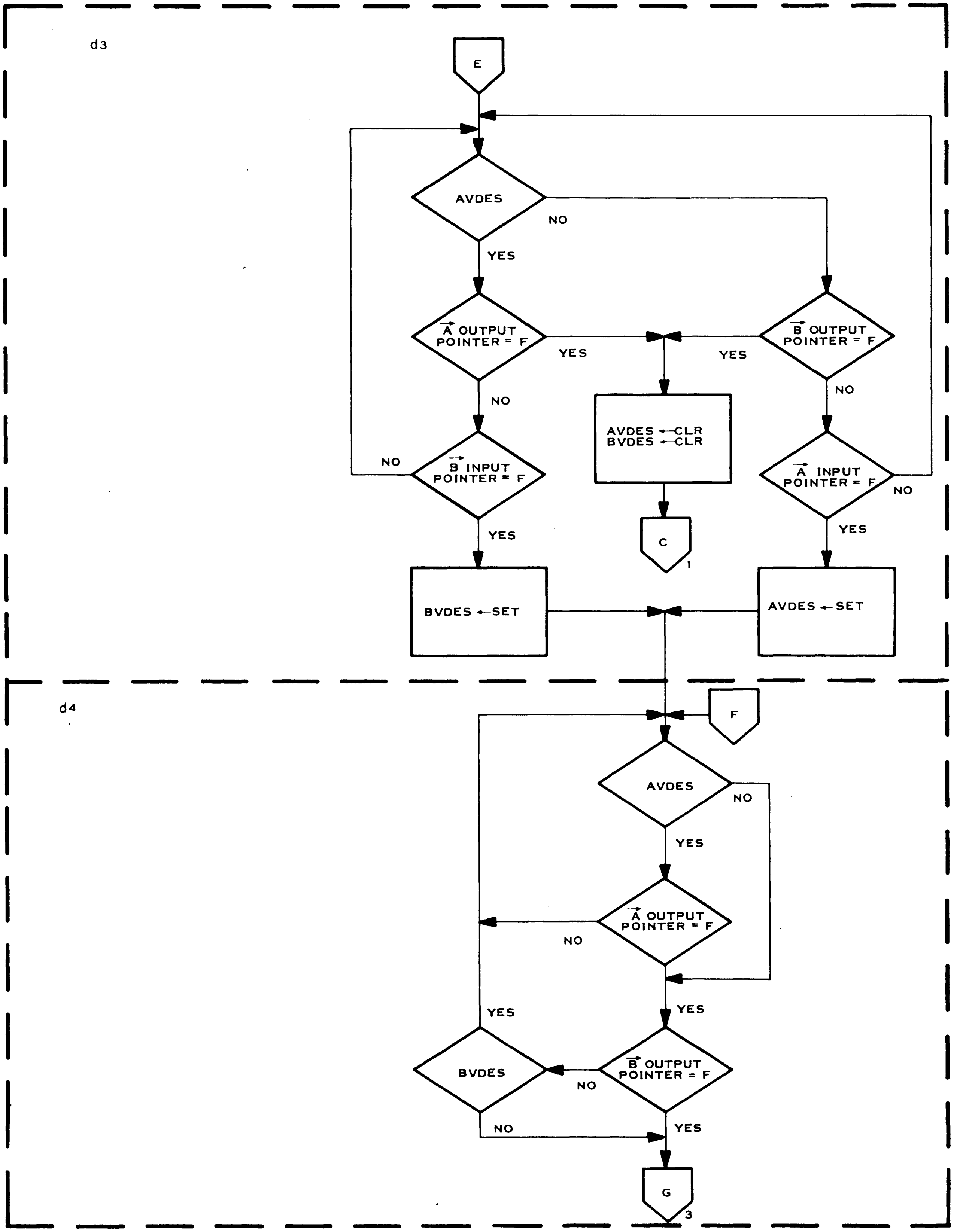


Figure 4-67. MBU Unit Hard Core Flowchart (Sheet 3 of 3)



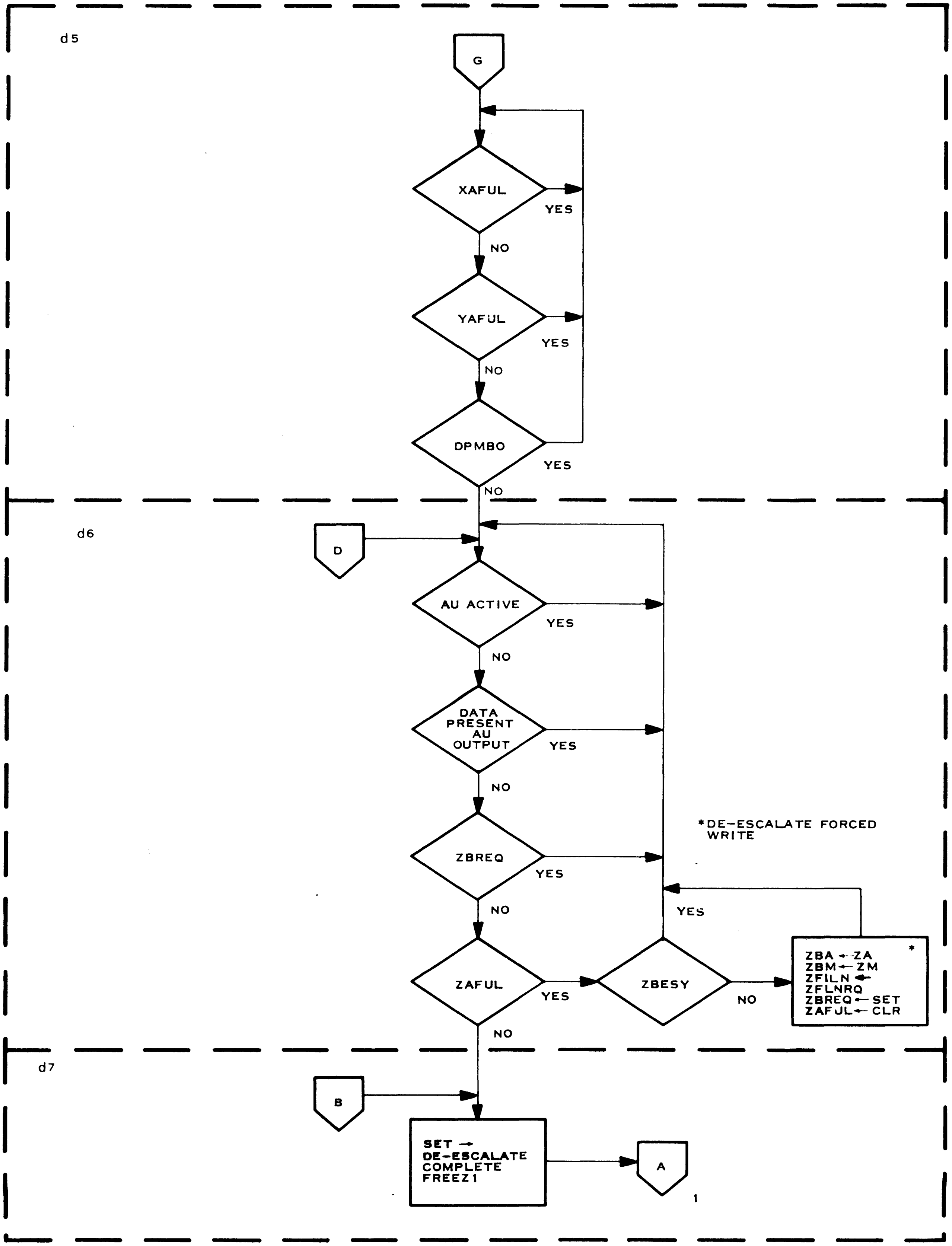
(A)115798

Figure 4-68. MBU Unit Hard Core De-escalate Controller Flowchart (Sheet 1 of 3)



(A)115799

Figure 4-68. MBU Unit Hard Core De-escalate Controller Flowchart (Sheet 2 of 3)



119496

Figure 4-68. MBU Unit Hard Core De-escalate Controller Flowchart (Sheet 3 of 3)

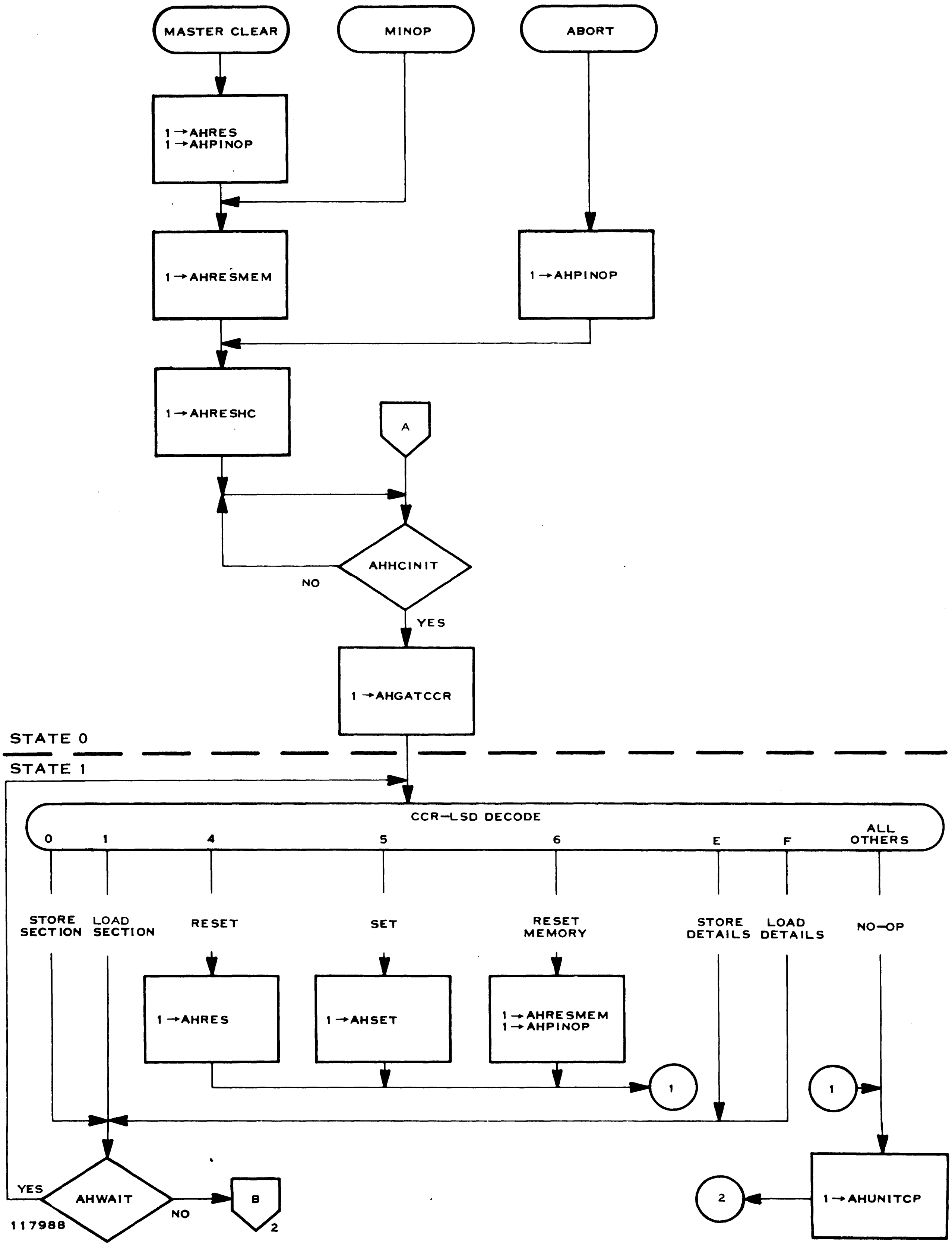
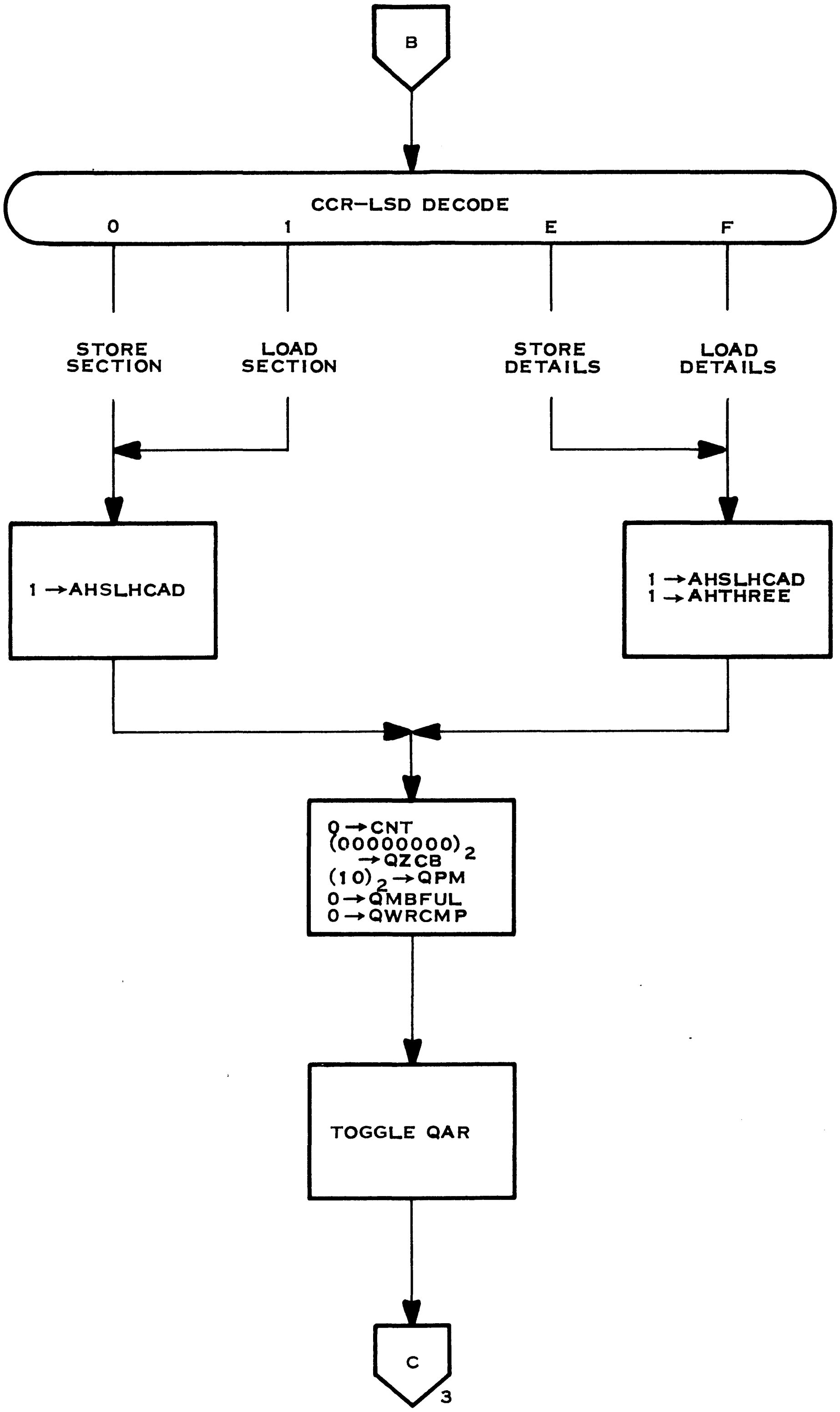


Figure 4-69. AU Unit Hardcore Flowchart (Sheet 1 of 7)



117989

Figure 4-69. AU Unit Hardcore Flowchart (Sheet 2 of 7)

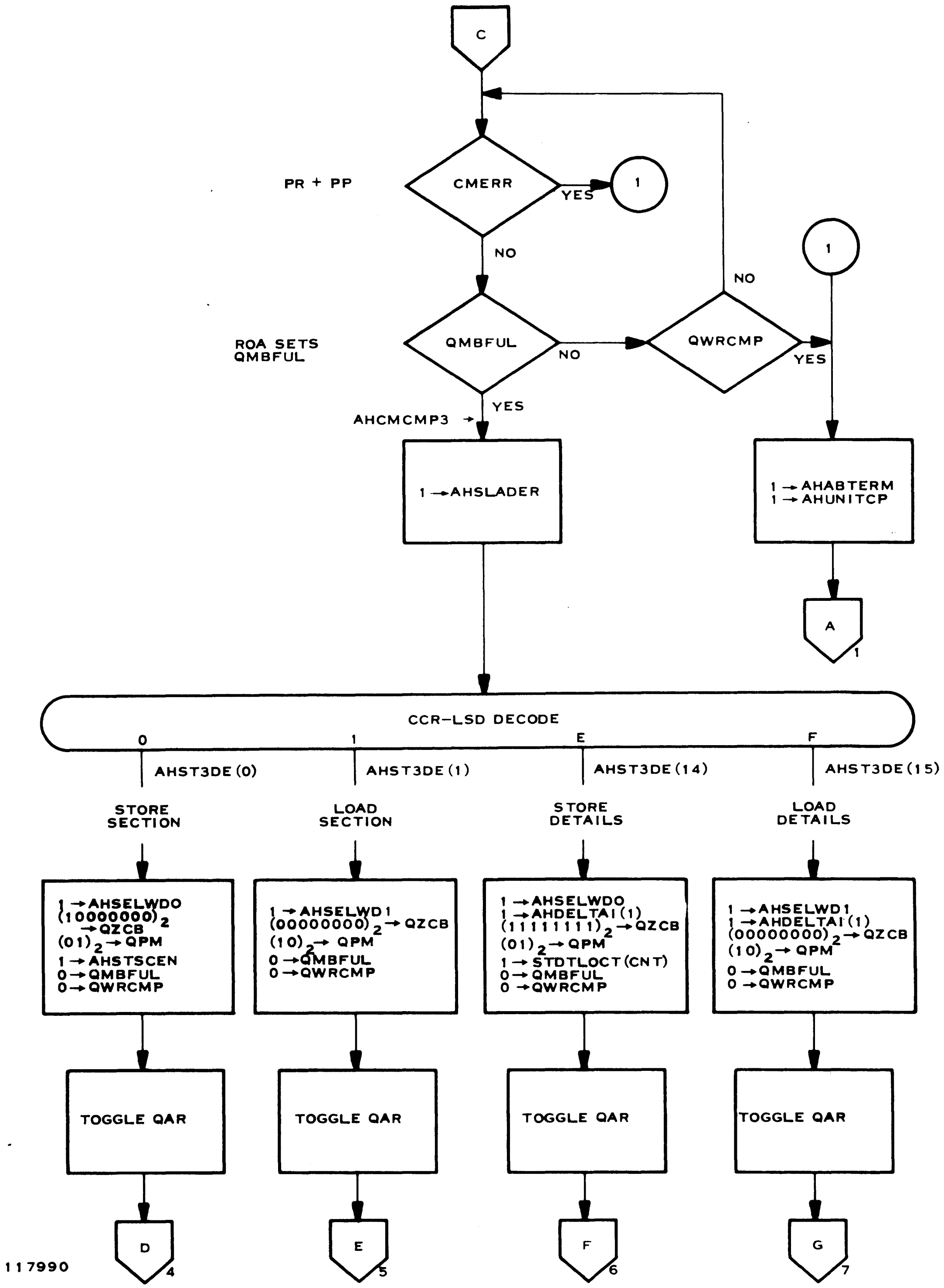
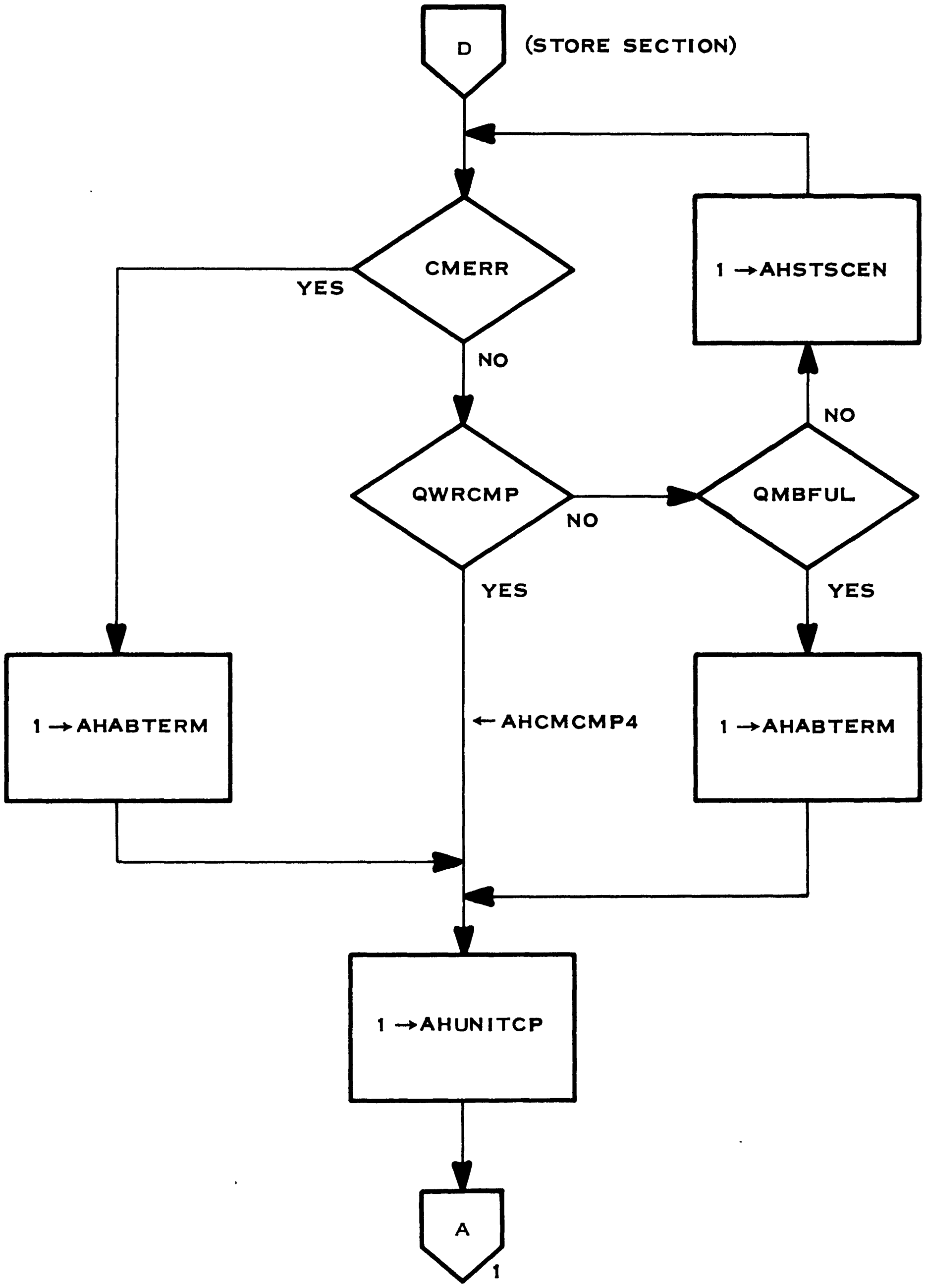
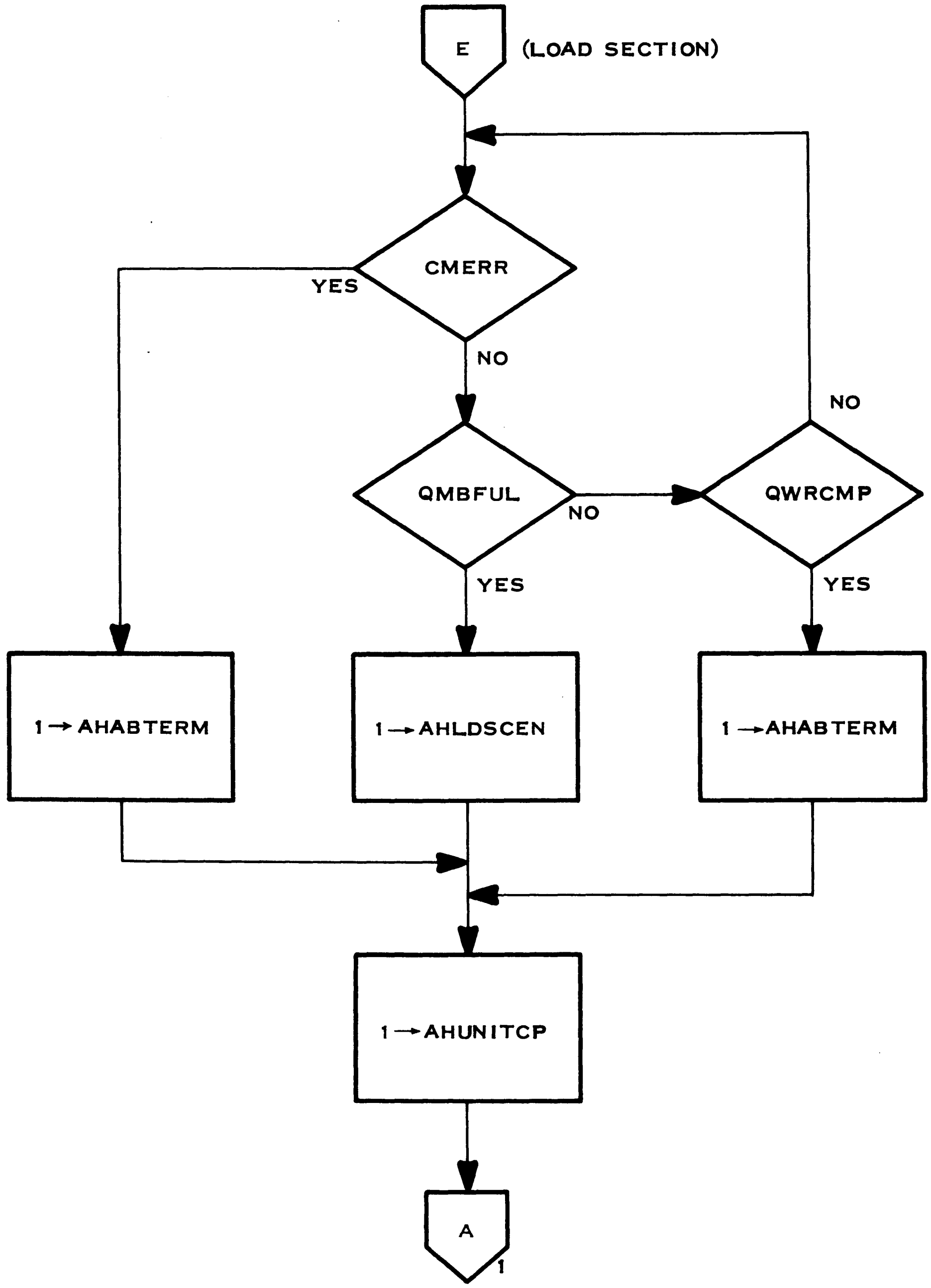


Figure 4-69. AU Unit Hardcore Flowchart (Sheet 3 of 7)



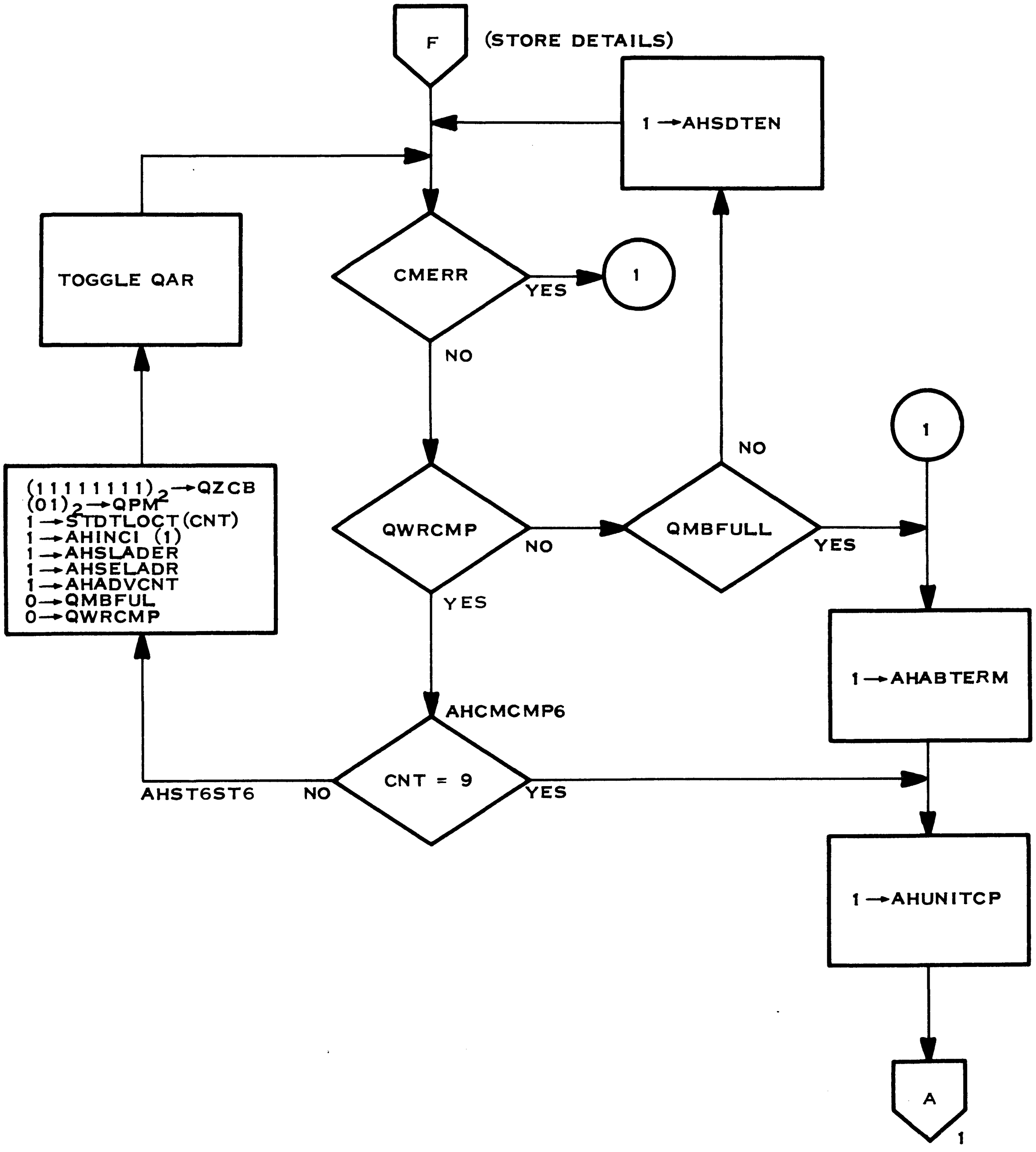
117991

Figure 4-69. AU Unit Hardcore Flowchart (Sheet 4 of 7)



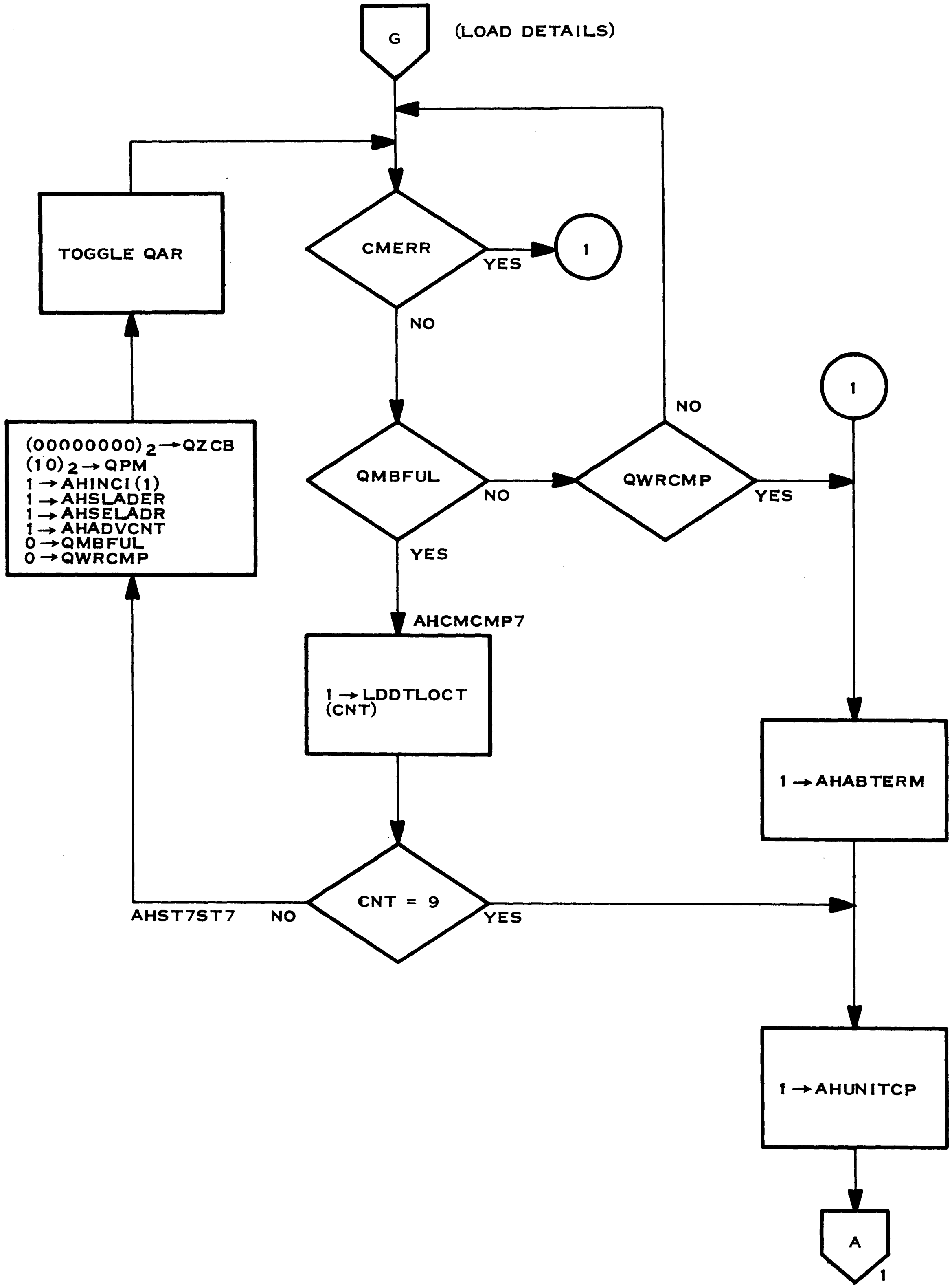
117992

Figure 4-69. AU Unit Hardcore Flowchart (Sheet 5 of 7)



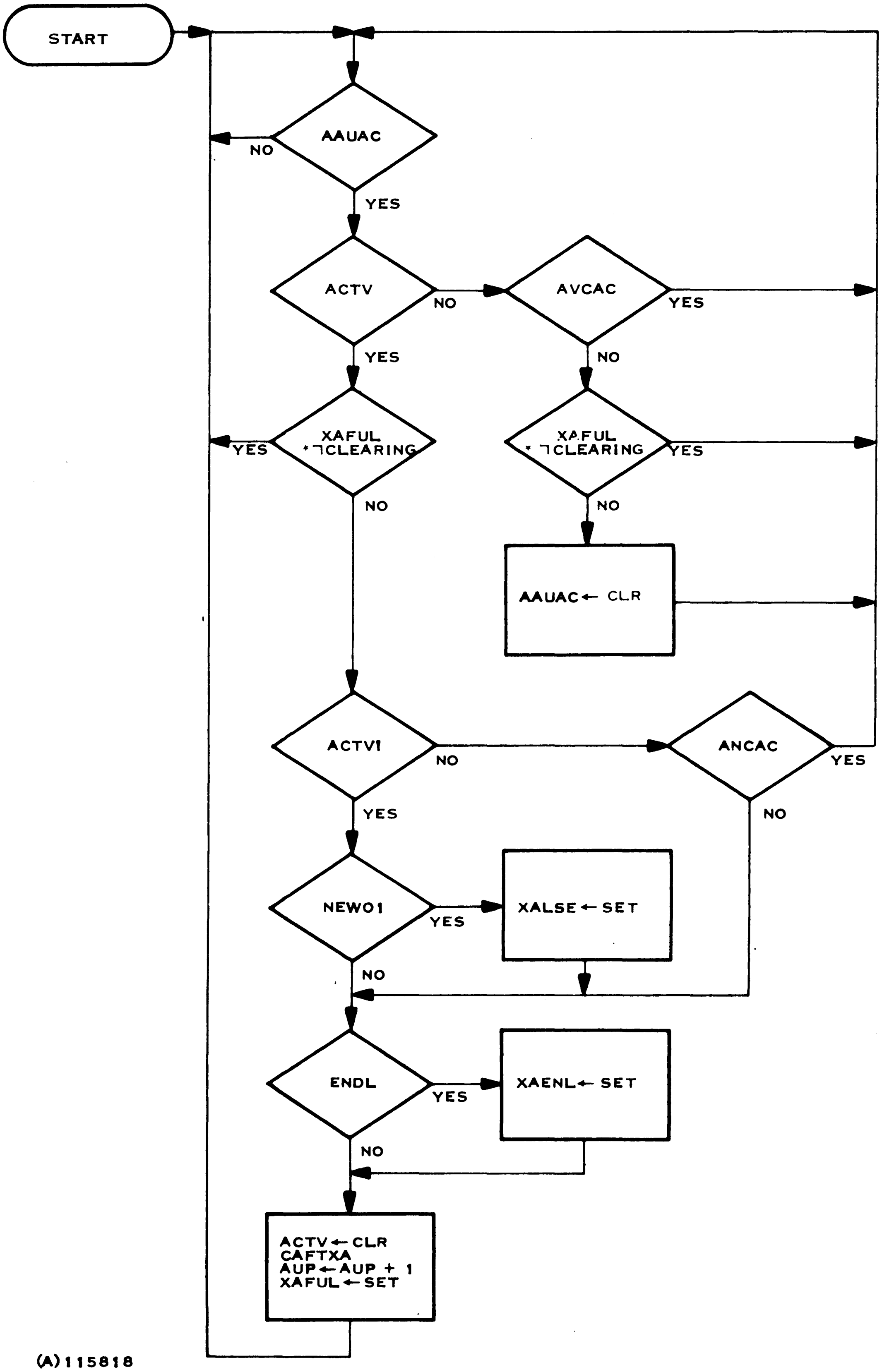
117993

Figure 4-69. AU Unit Hardcore Flowchart (Sheet 6 of 7)



117994

Figure 4-69. AU Unit Hardcore Flowchart (Sheet 7 of 7)



(A)115818

Figure 4-70. CAF Output Control Flowchart

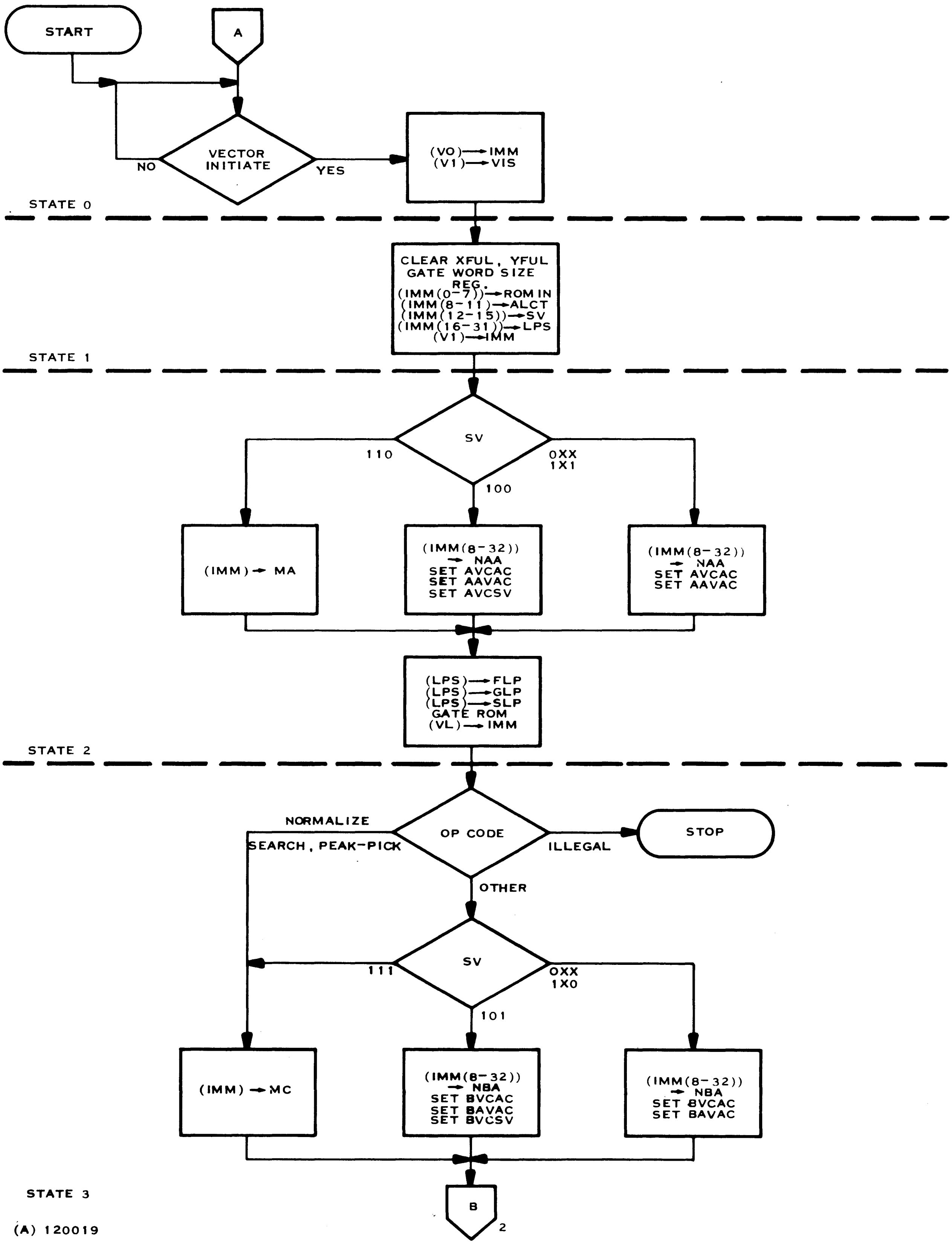
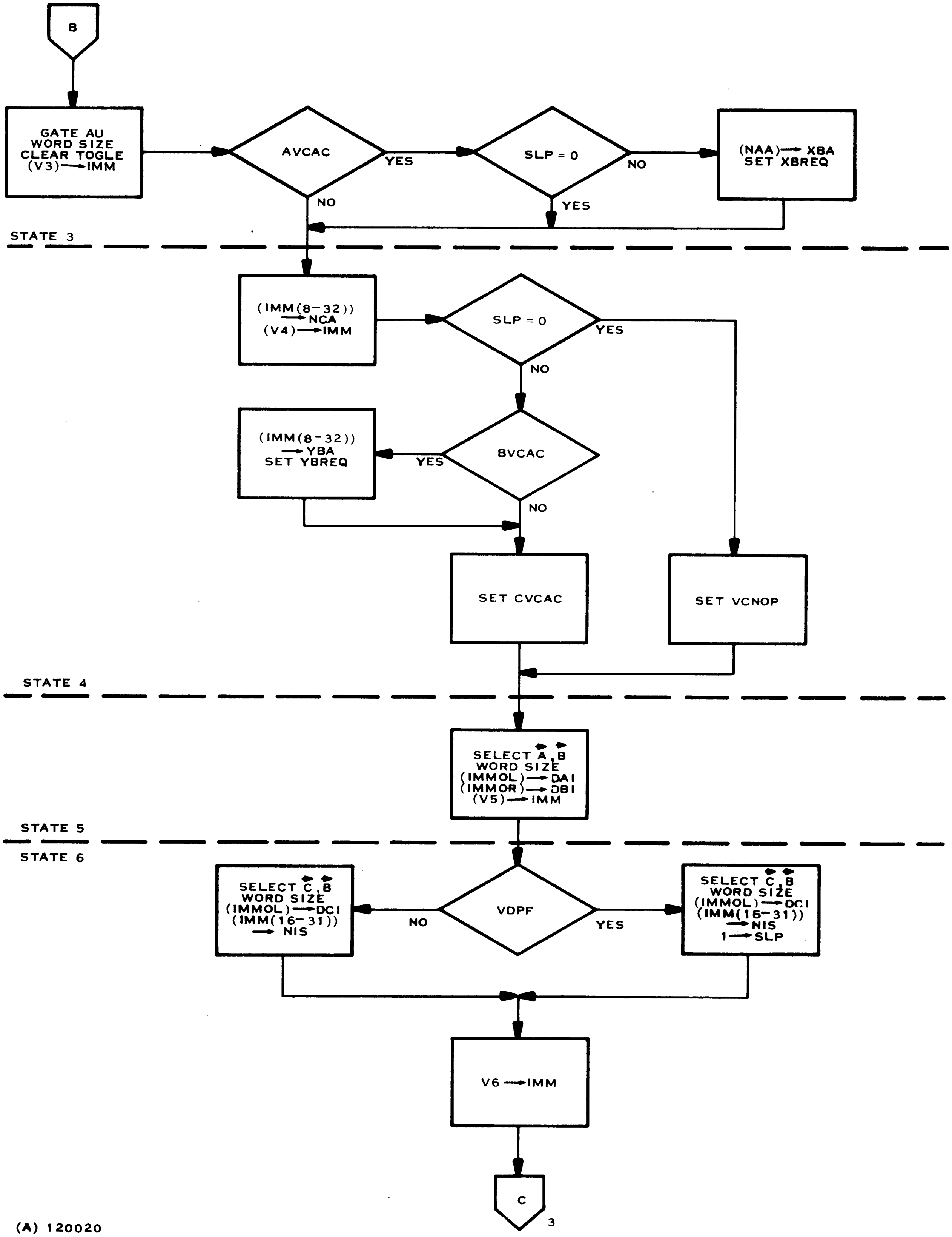


Figure 4-71. Vector Initialization Control Flowchart (Sheet 1 of 4)



(A) 120020

Figure 4-71. Vector Initialization Control Flowchart (Sheet 2 of 4)

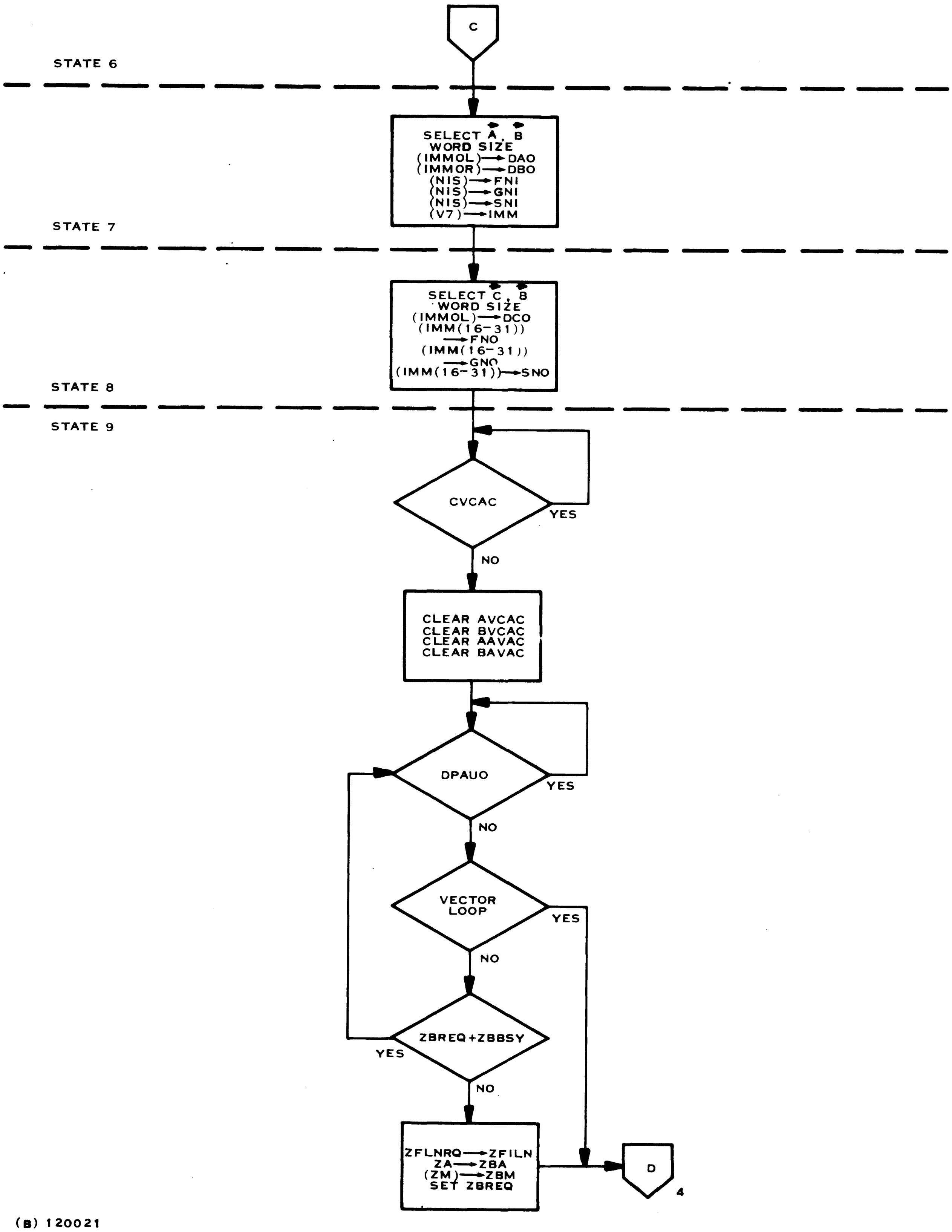
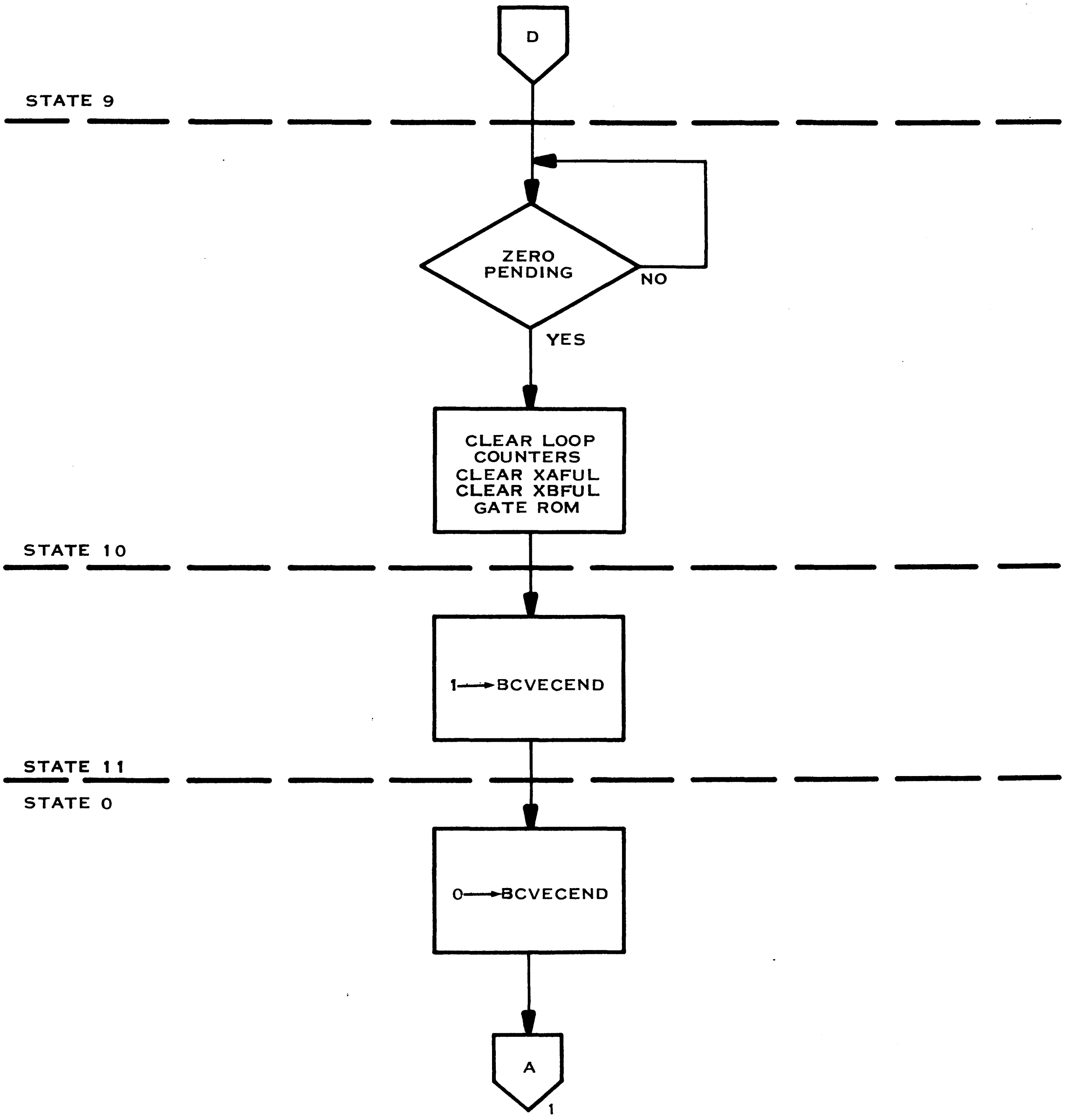
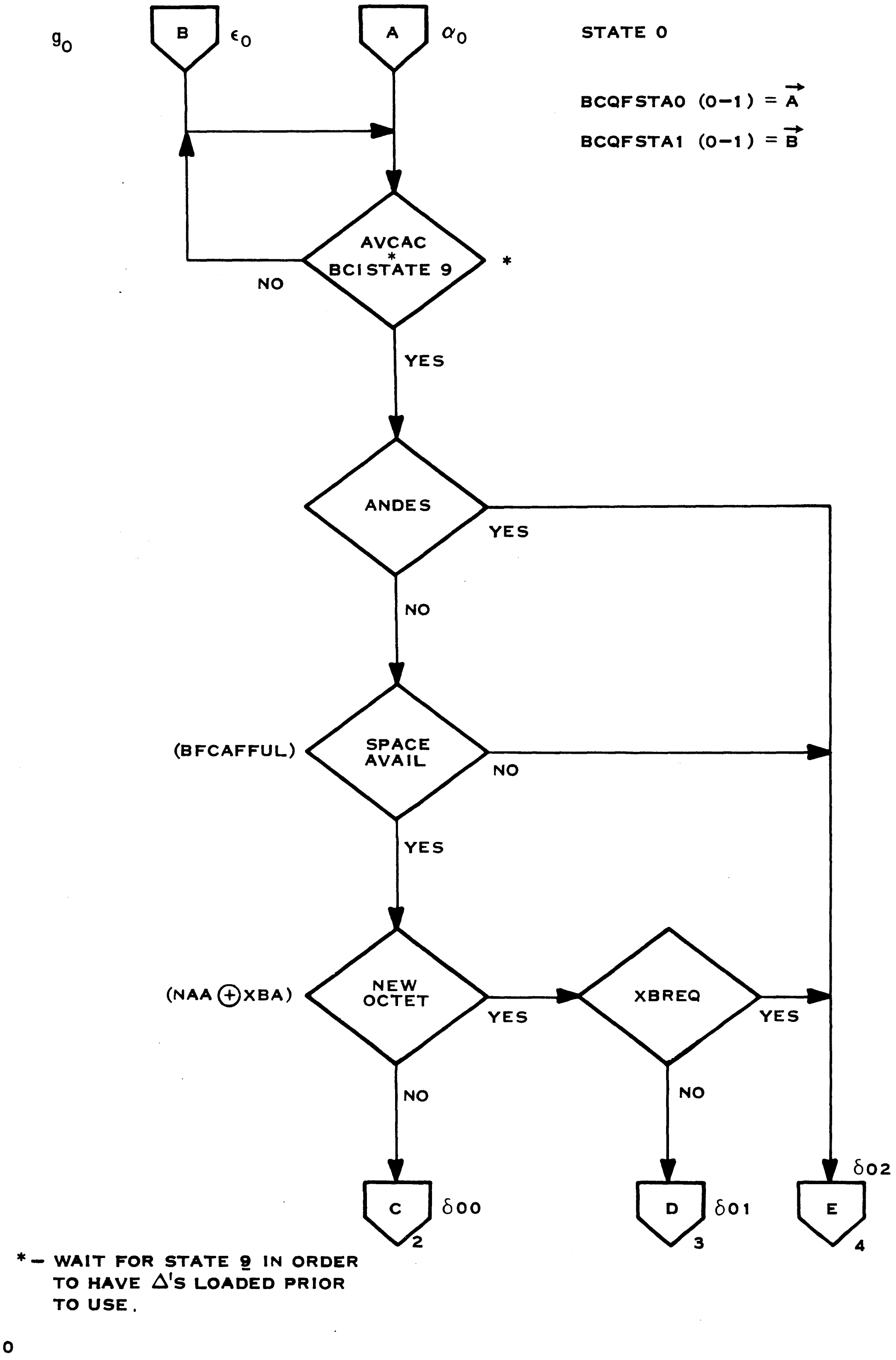


Figure 4-71. Vector Initialization Control Flowchart (Sheet 3 of 4)



(A) 120022

Figure 4-71. Vector Initialization Control Flowchart (Sheet 4 of 4)



(A) 115800

Figure 4-72. A/B Vector Address Generation Flowchart (Sheet 1 of 17)

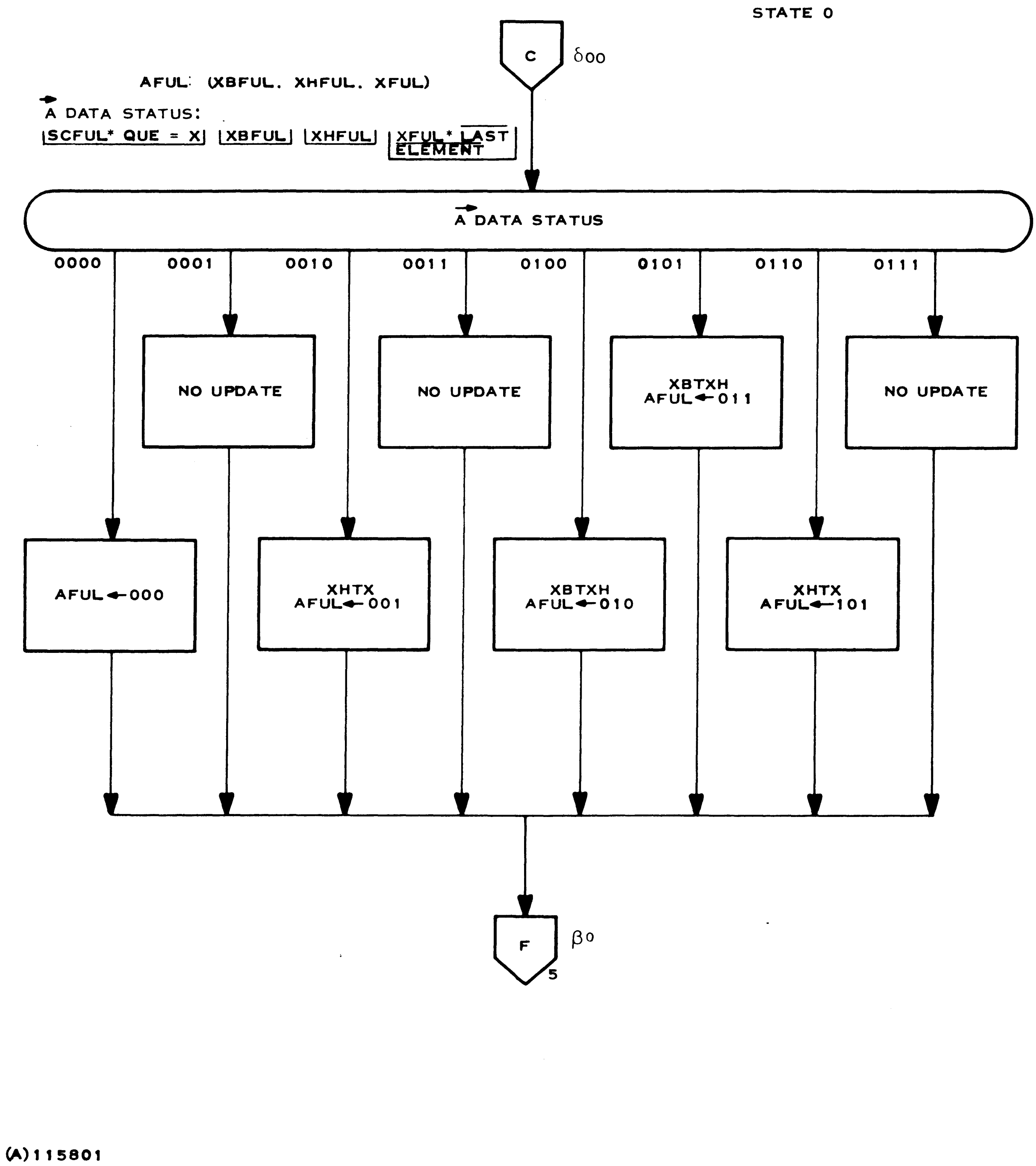
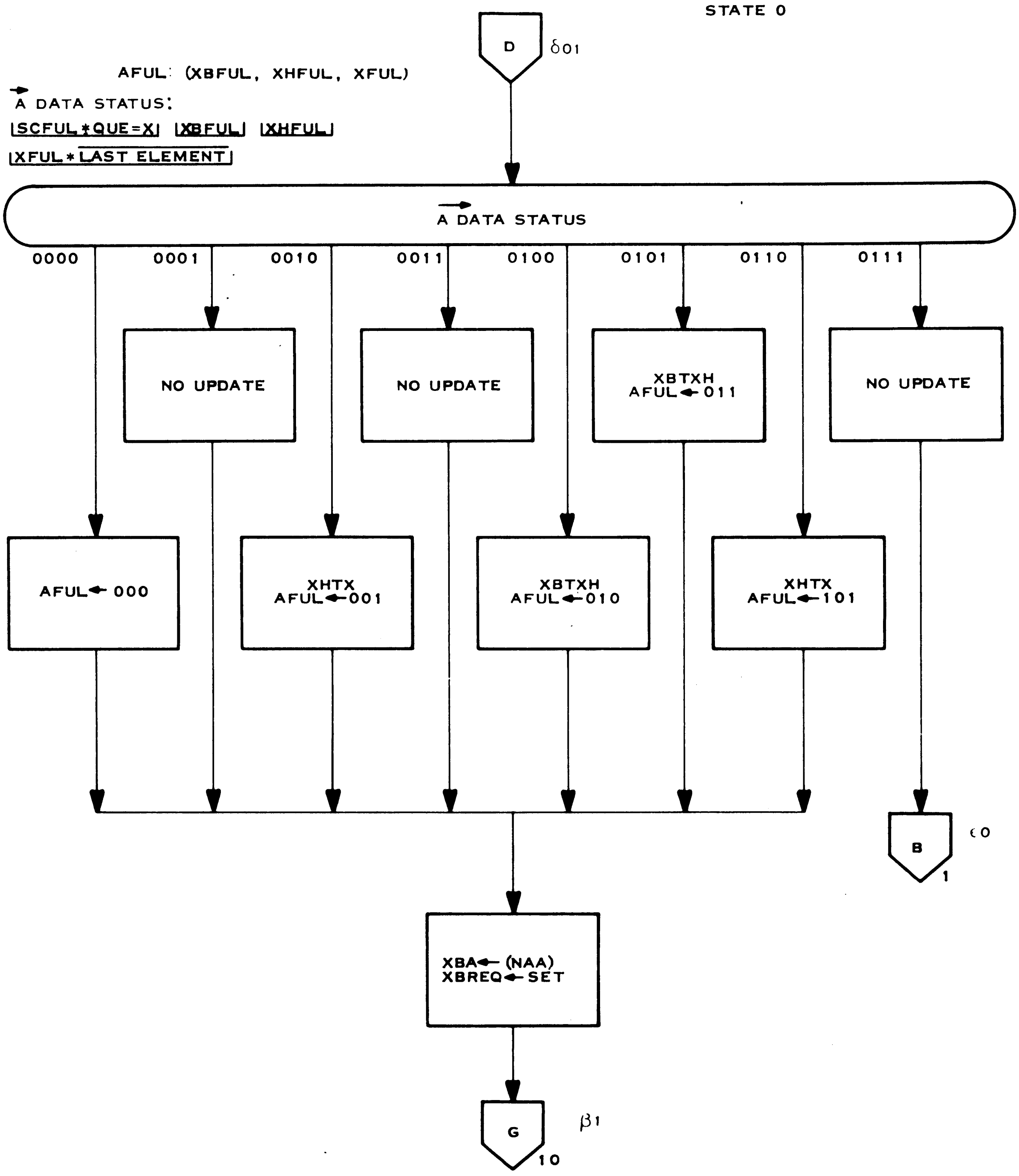


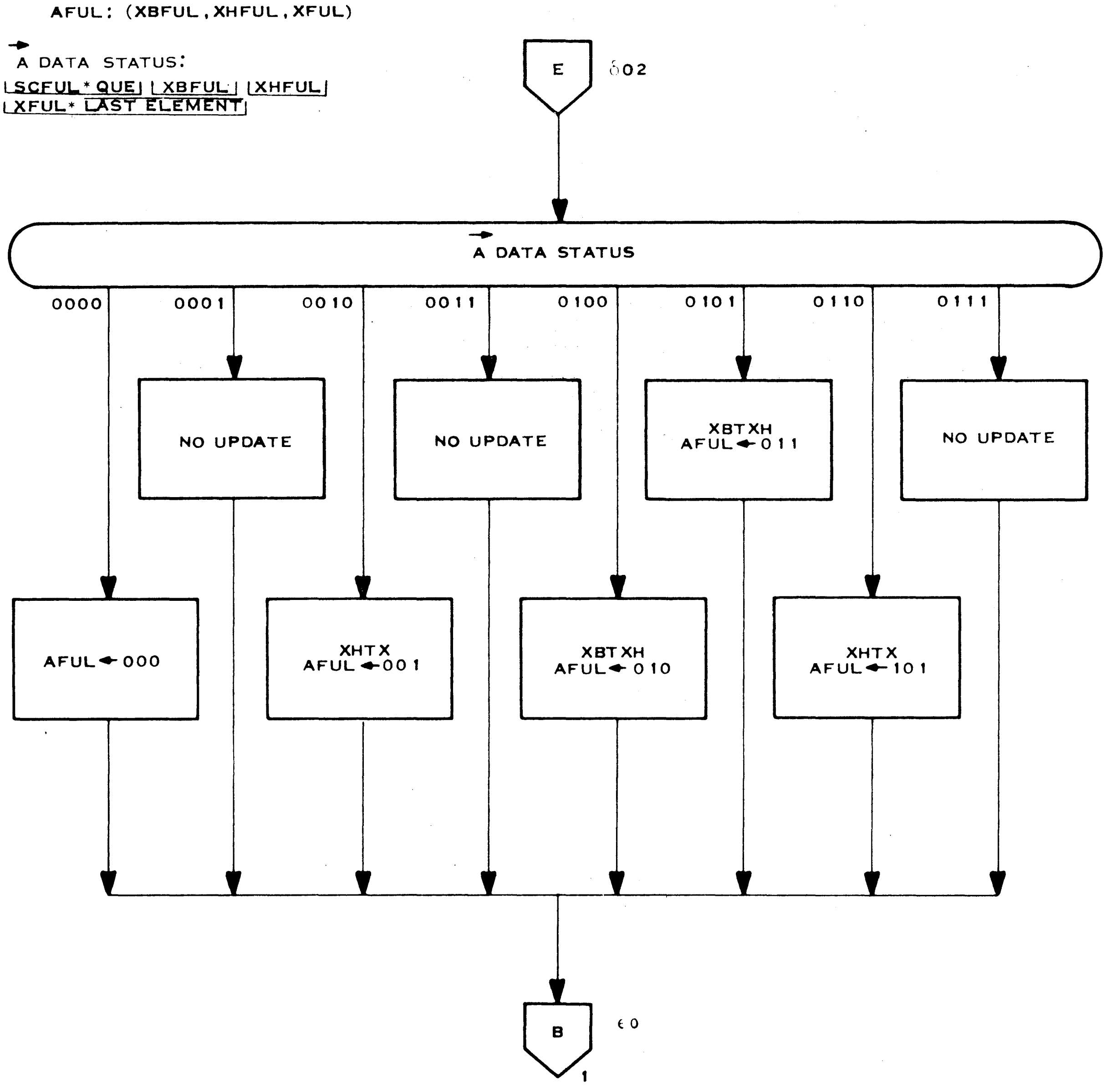
Figure 4-72. A/B Vector Address Generation Flowchart (Sheet 2 of 17)



(A)115802

Figure 4-72. A/B Vector Address Generation Flowchart (Sheet 3 of 17)

STATE 0



(A) 115803

Figure 4-72. A/B Vector Address Generation Flowchart (Sheet 4 of 17)

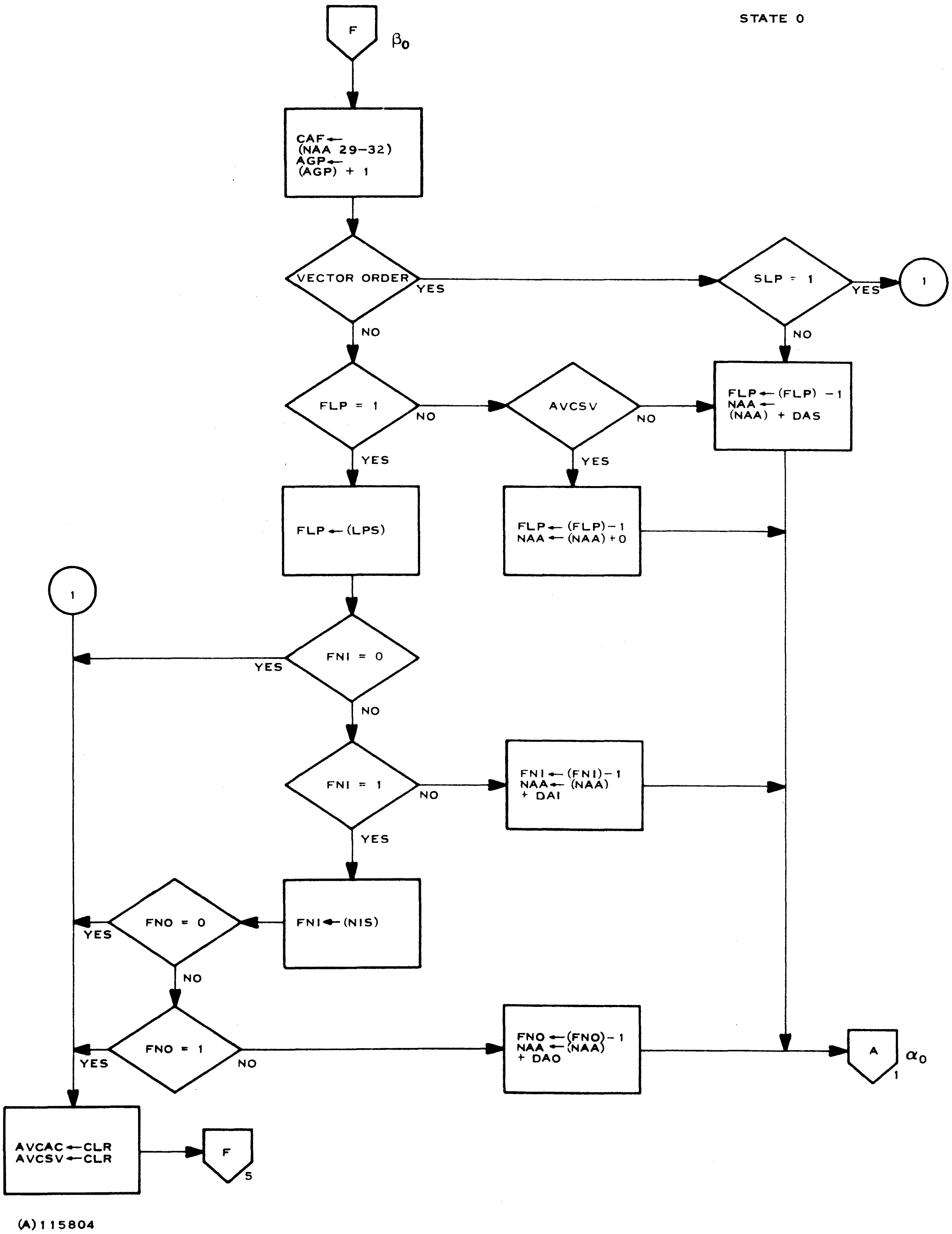


Figure 4-72. A/B Vector Address Generat'on Flowchart (Sheet 5 of 17)

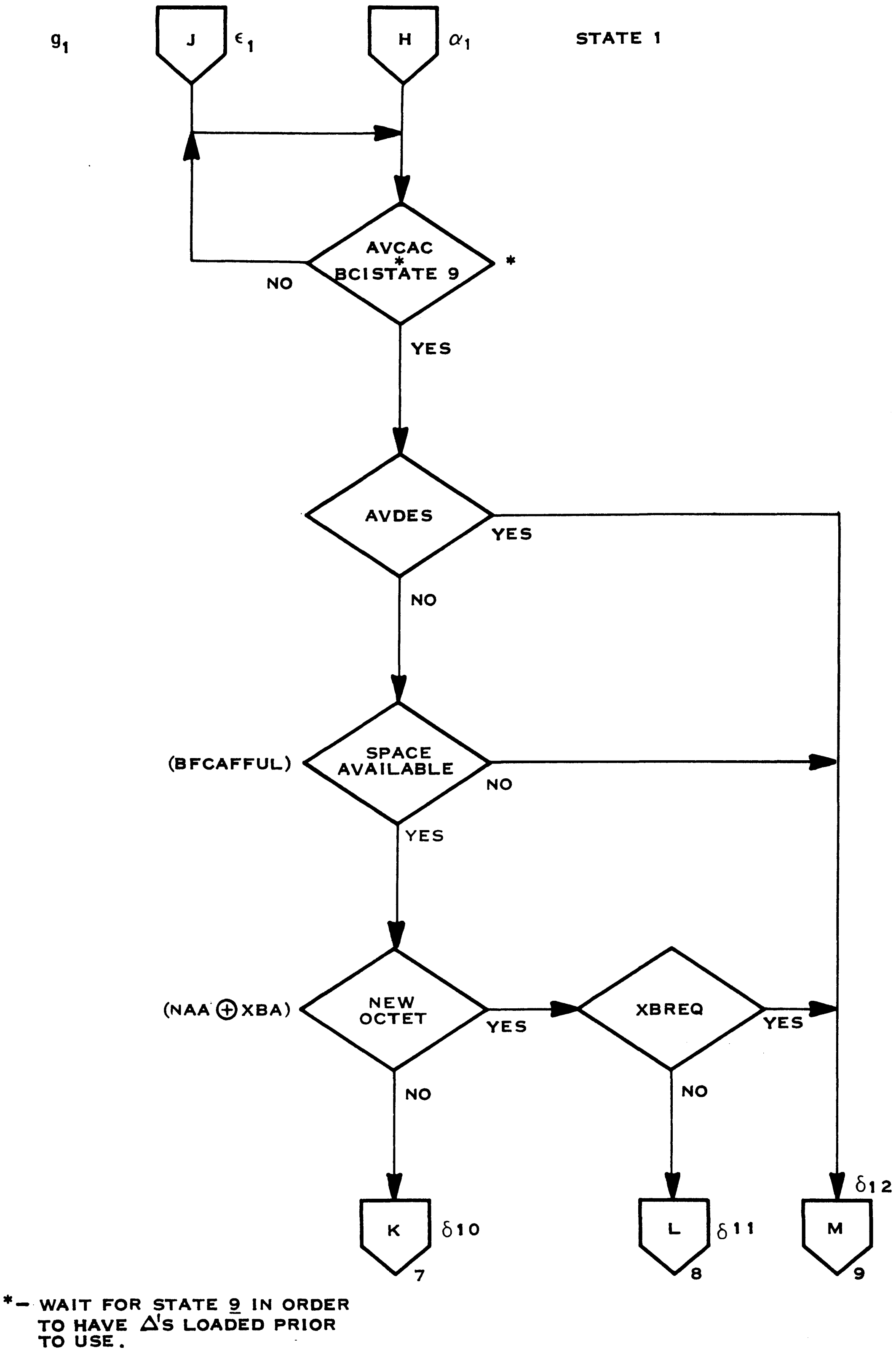
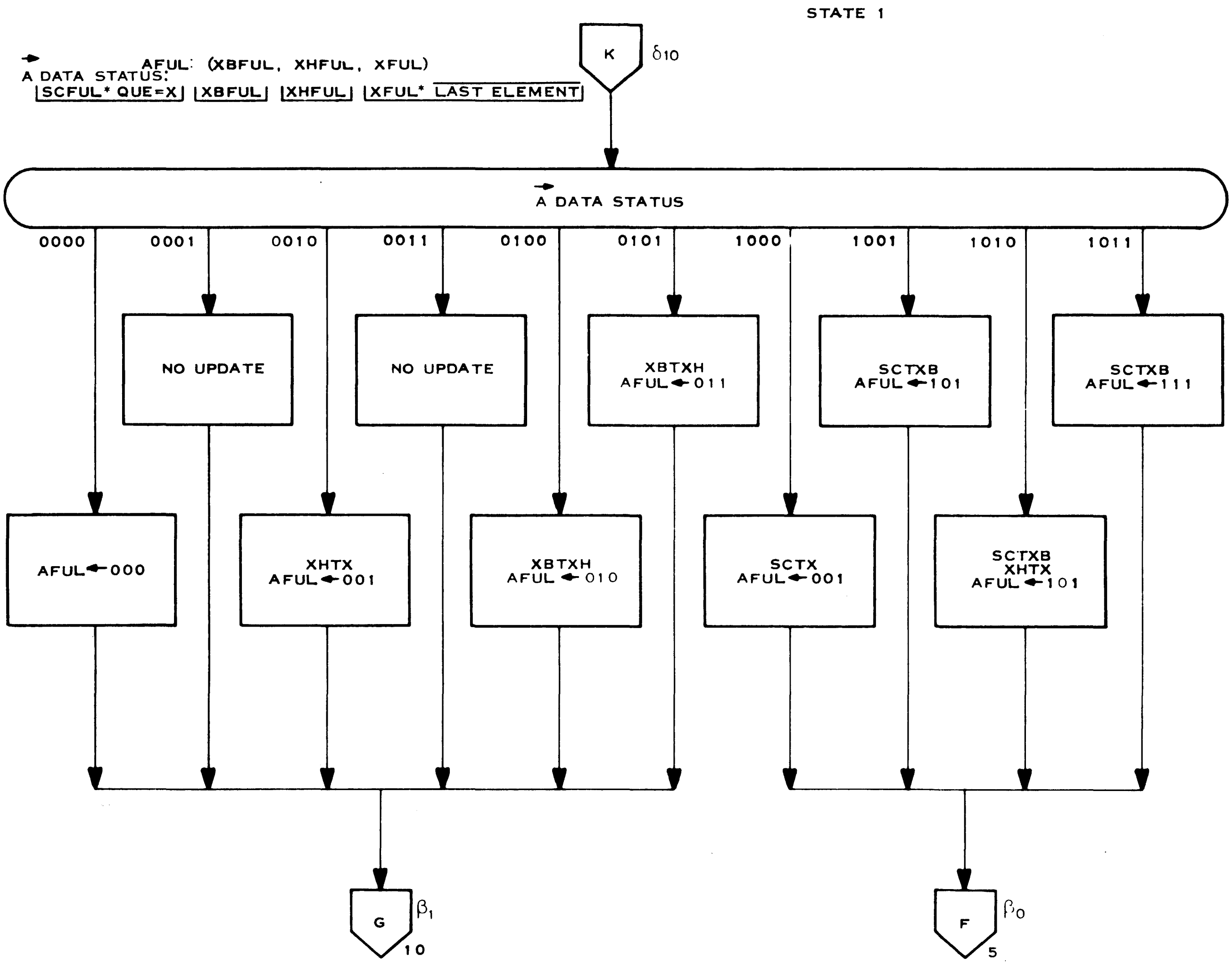
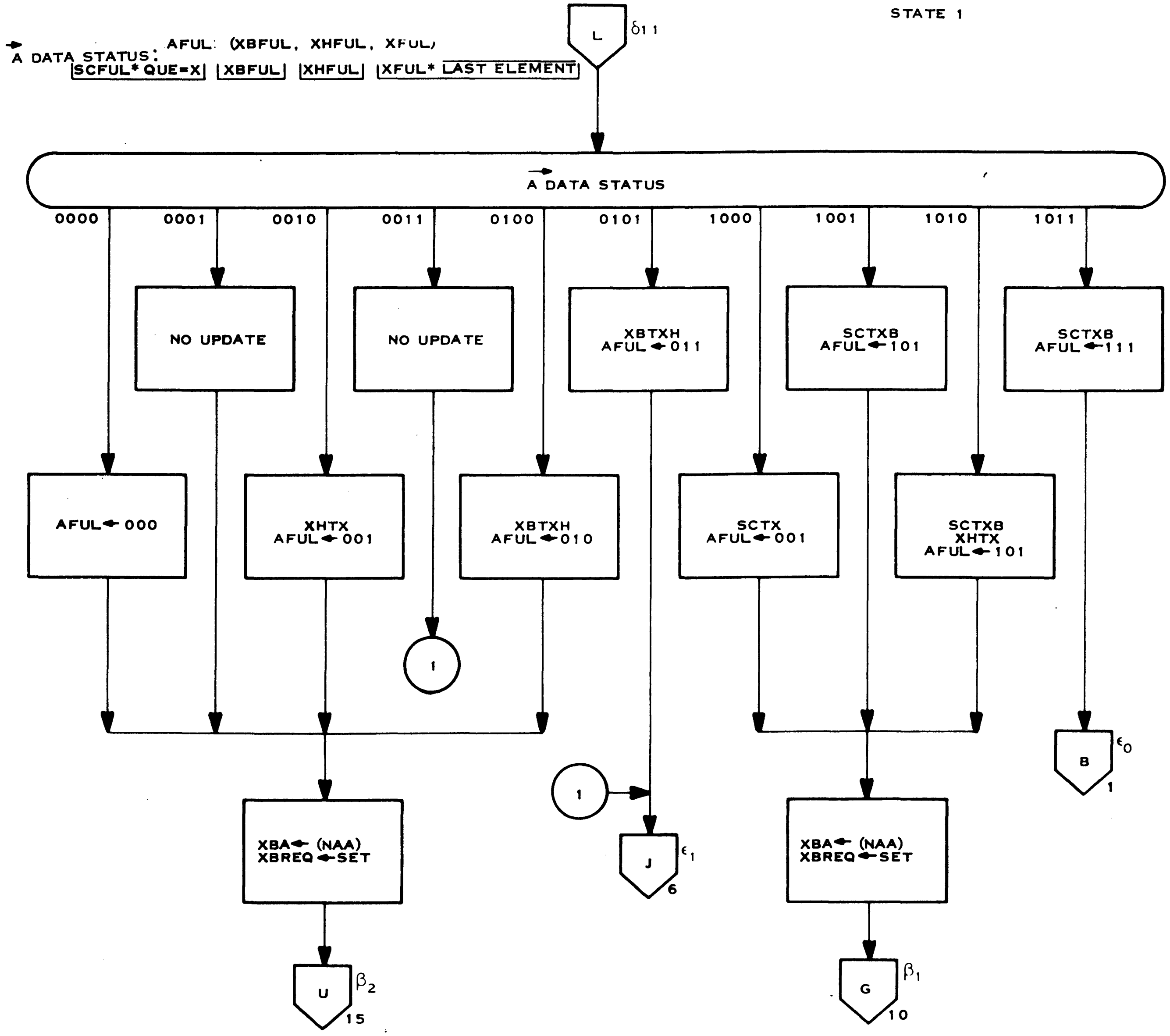


Figure 4-72. A/B Vector Address Generation Flowchart (Sheet 6 of 17)



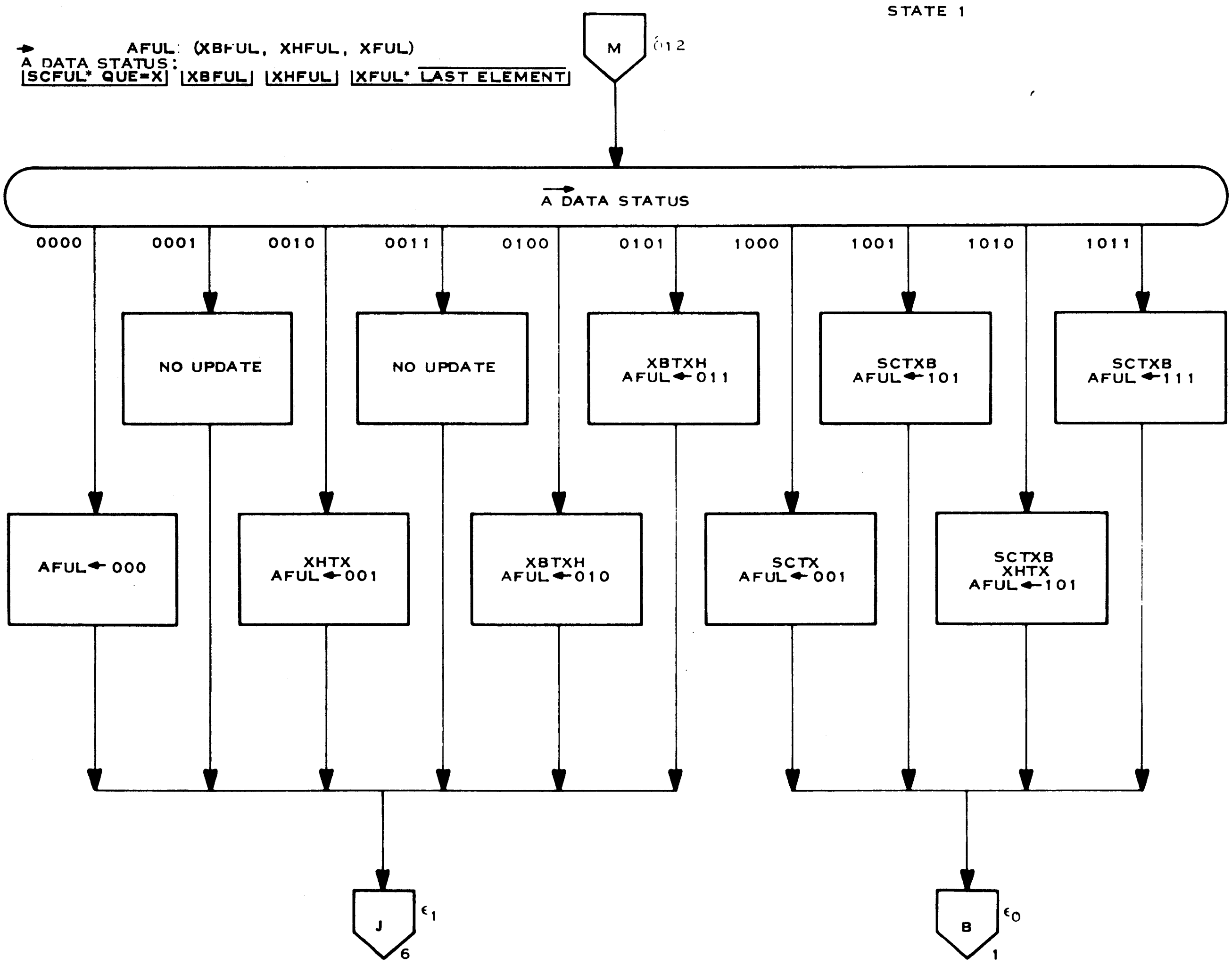
(A)115806

Figure 4-72. A/B Vector Address Generation Flowchart (Sheet 7 of 17)



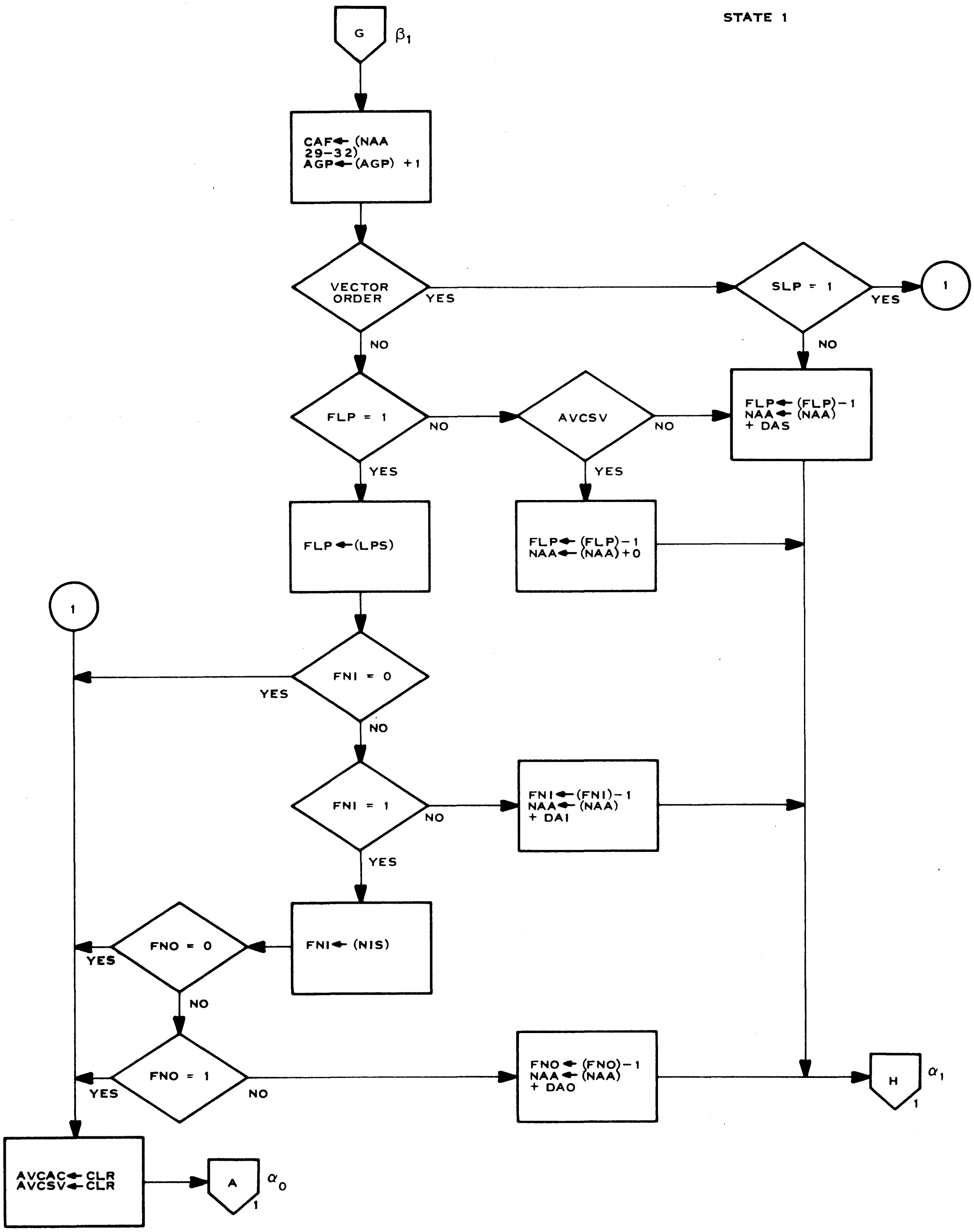
(A)115807

Figure 4-72. A/B Vector Address Generation Flowchart (Sheet 8 of 17)



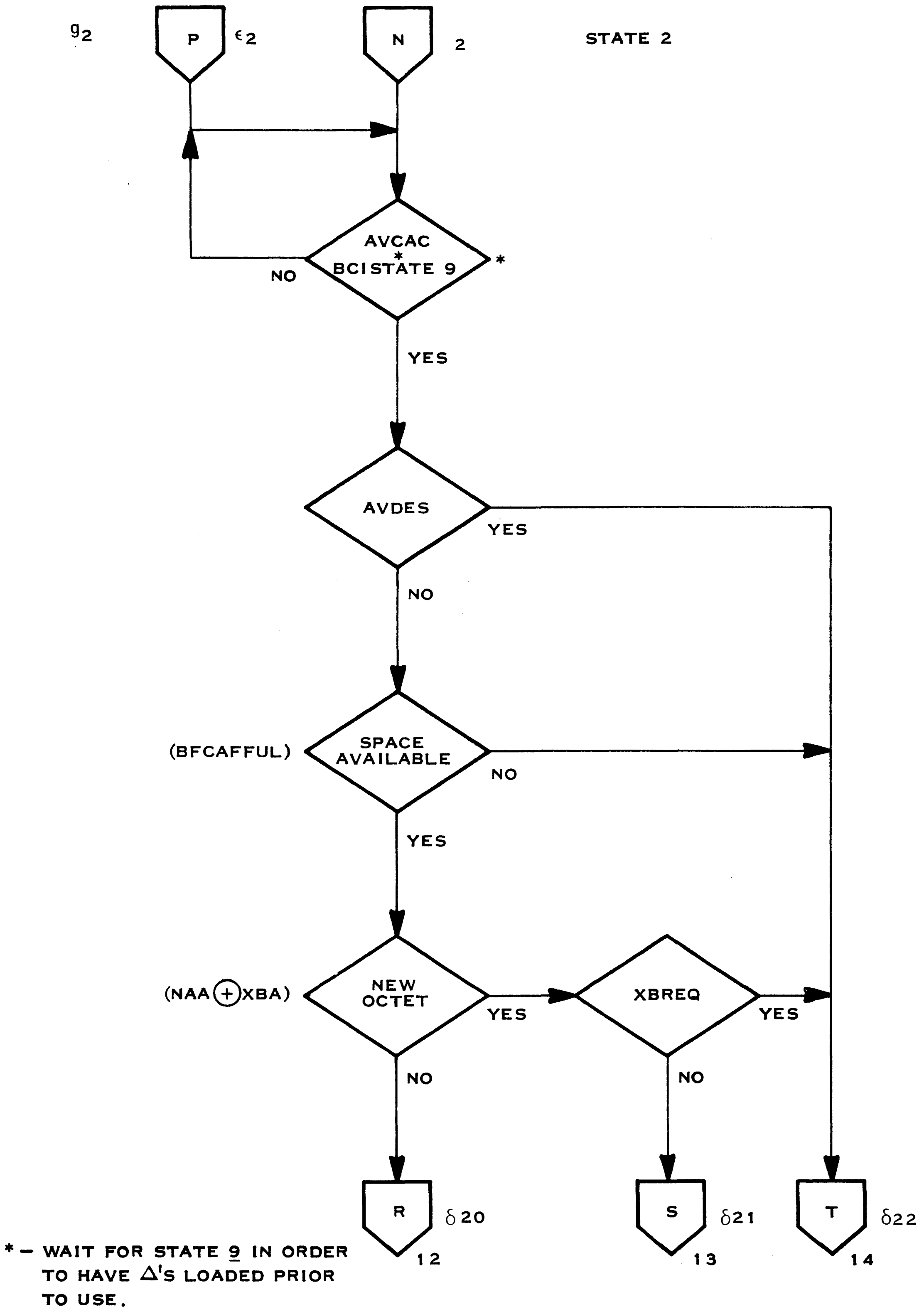
(A)115808

Figure 4-72. A/B Vector Address Generation Flowchart (Sheet 9 of 17)



(A)115809

Figure 4-72. A/B Vector Address Generation Flowchart (Sheet 10 of 17)



(A) 115810

Figure 4-72. A/B Vector Address Generation Flowchart (Sheet 11 of 17)

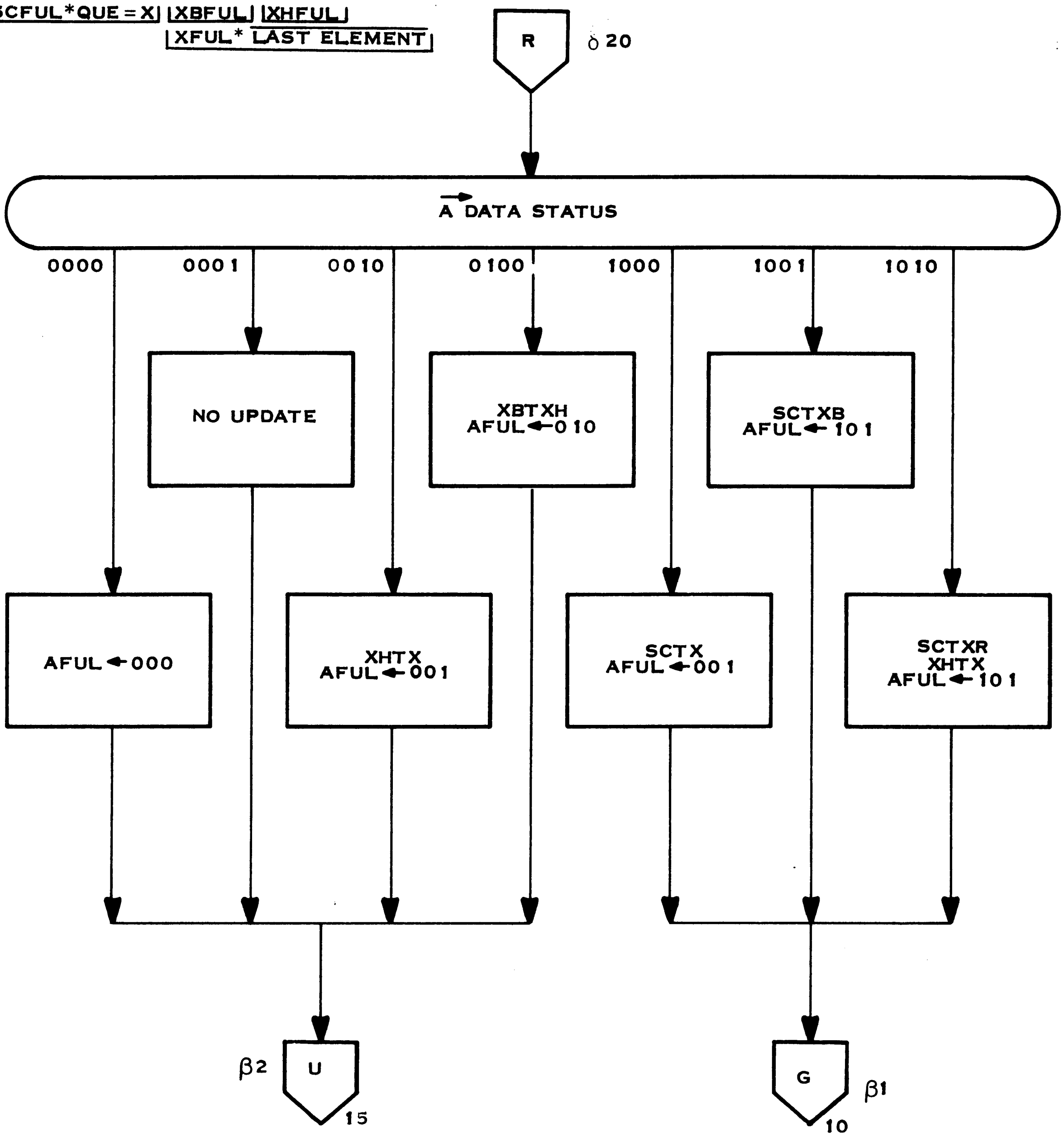
STATE 2

AFUL:(XBFUL,XHFUL,XFUL)

$\vec{A}$ DATA STATUS:

$\lfloor \text{SCFUL*QUE = X} \rfloor \lfloor \text{XBFUL} \rfloor \lfloor \text{XHFUL} \rfloor$

$\lfloor \text{XFUL* } \overline{\text{LAST ELEMENT}} \rfloor$



(A) 115811

Figure 4-72. A/B Vector Address Generation Flowchart (Sheet 12 of 17)

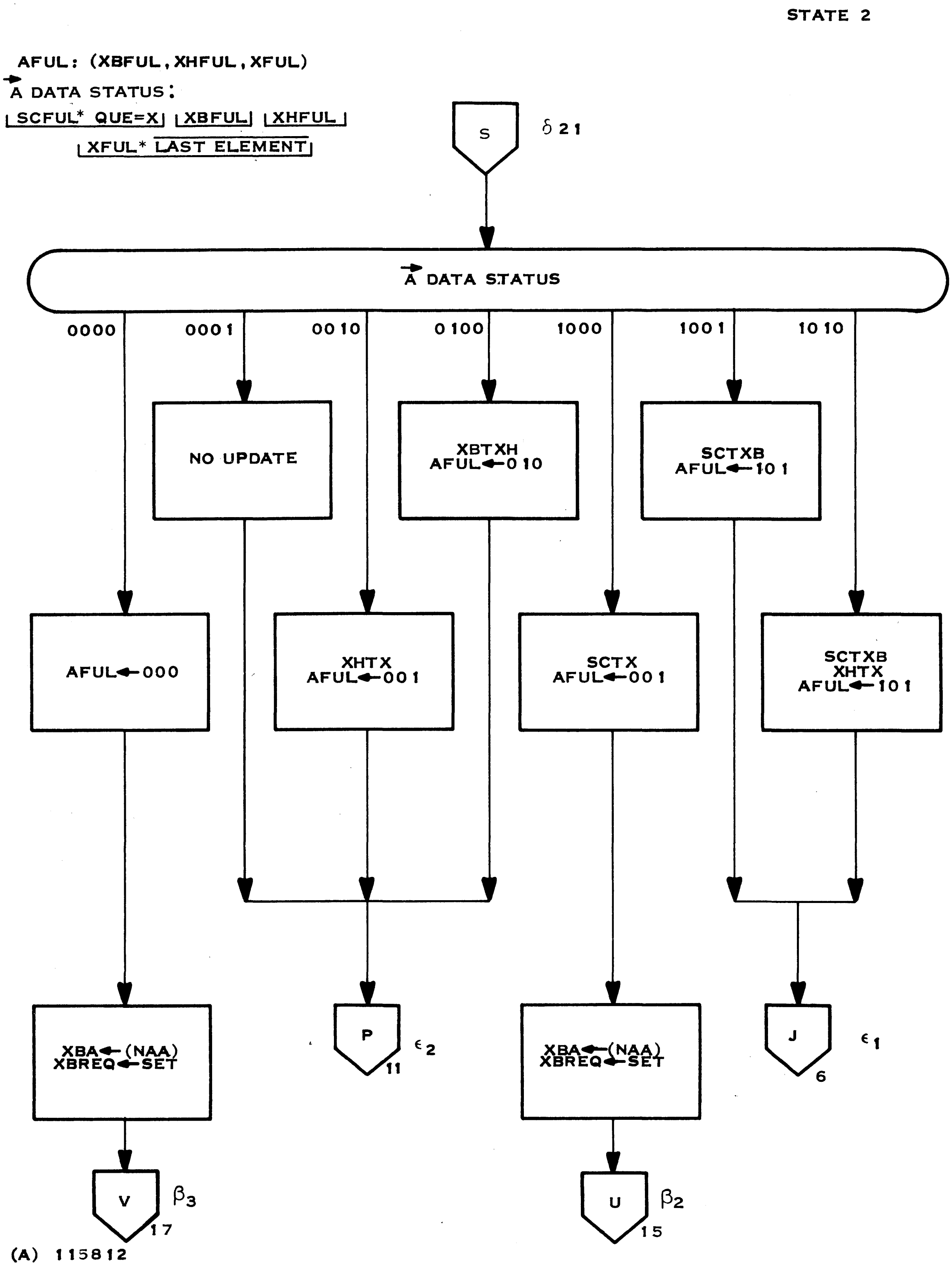
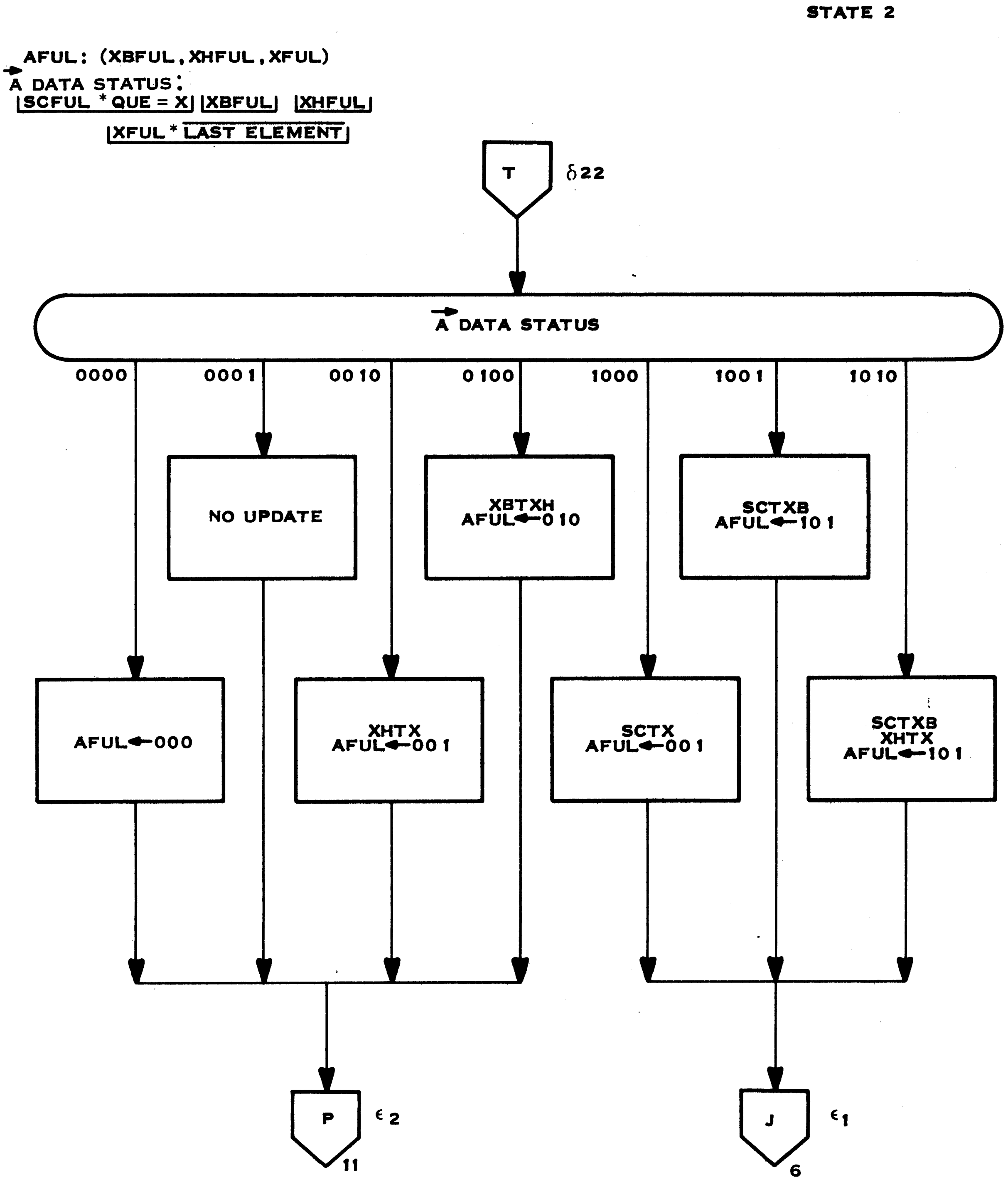
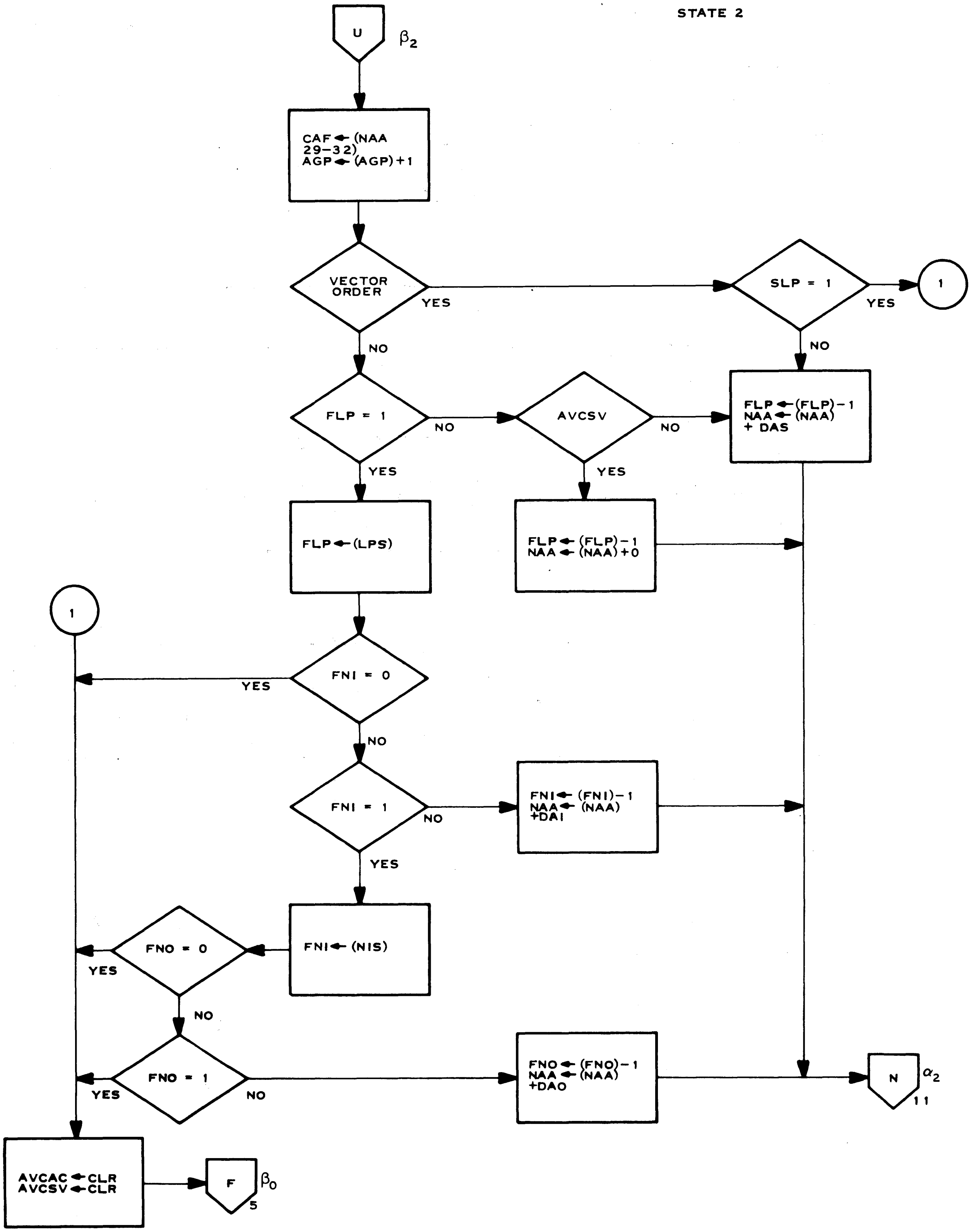


Figure 4-72. A/B Vector Address Generation Flowchart (Sheet 13 of 17)



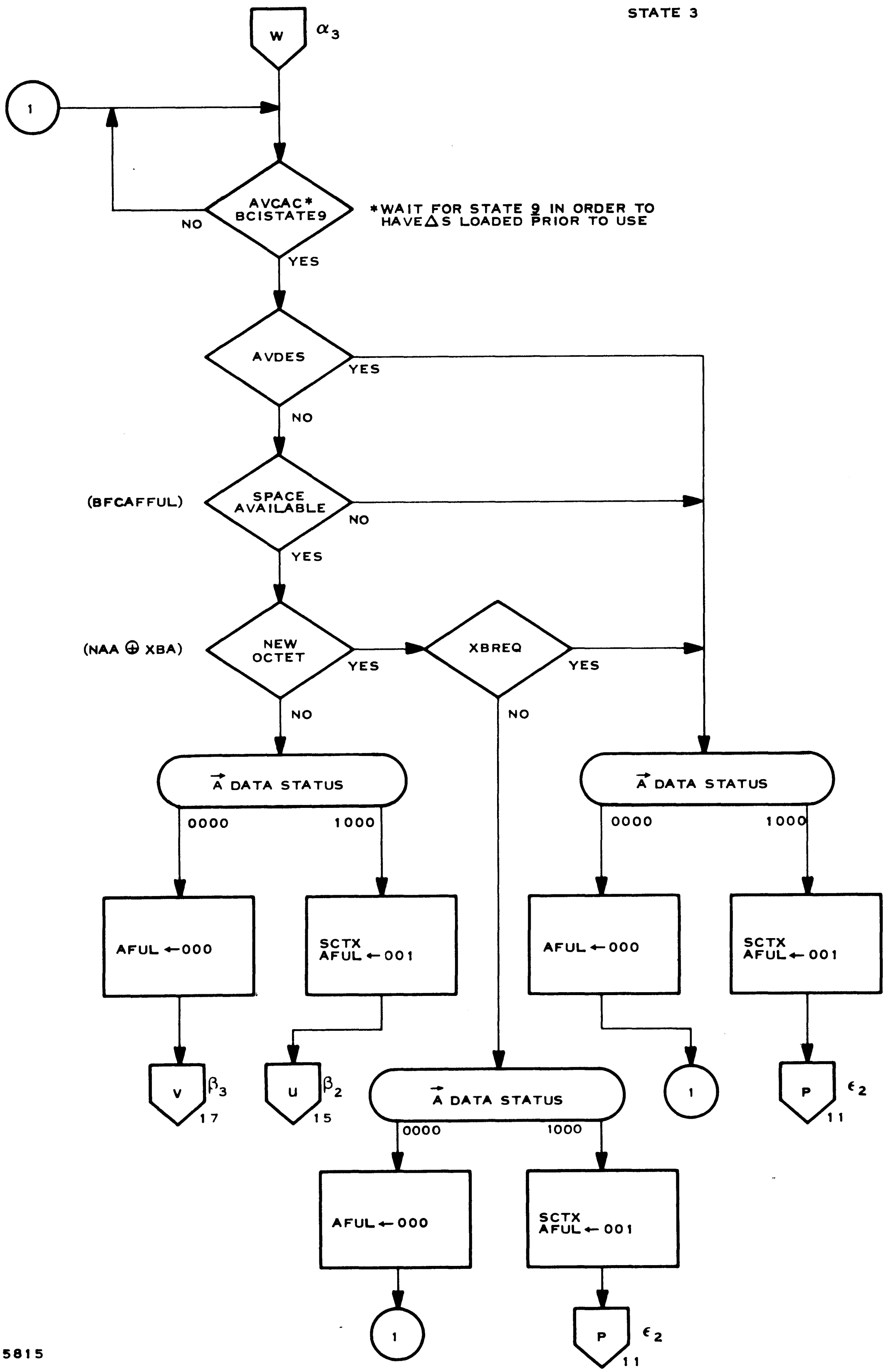
(A) 115813

Figure 4-72. A/B Vector Address Generation Flowchart (Sheet 14 of 17)



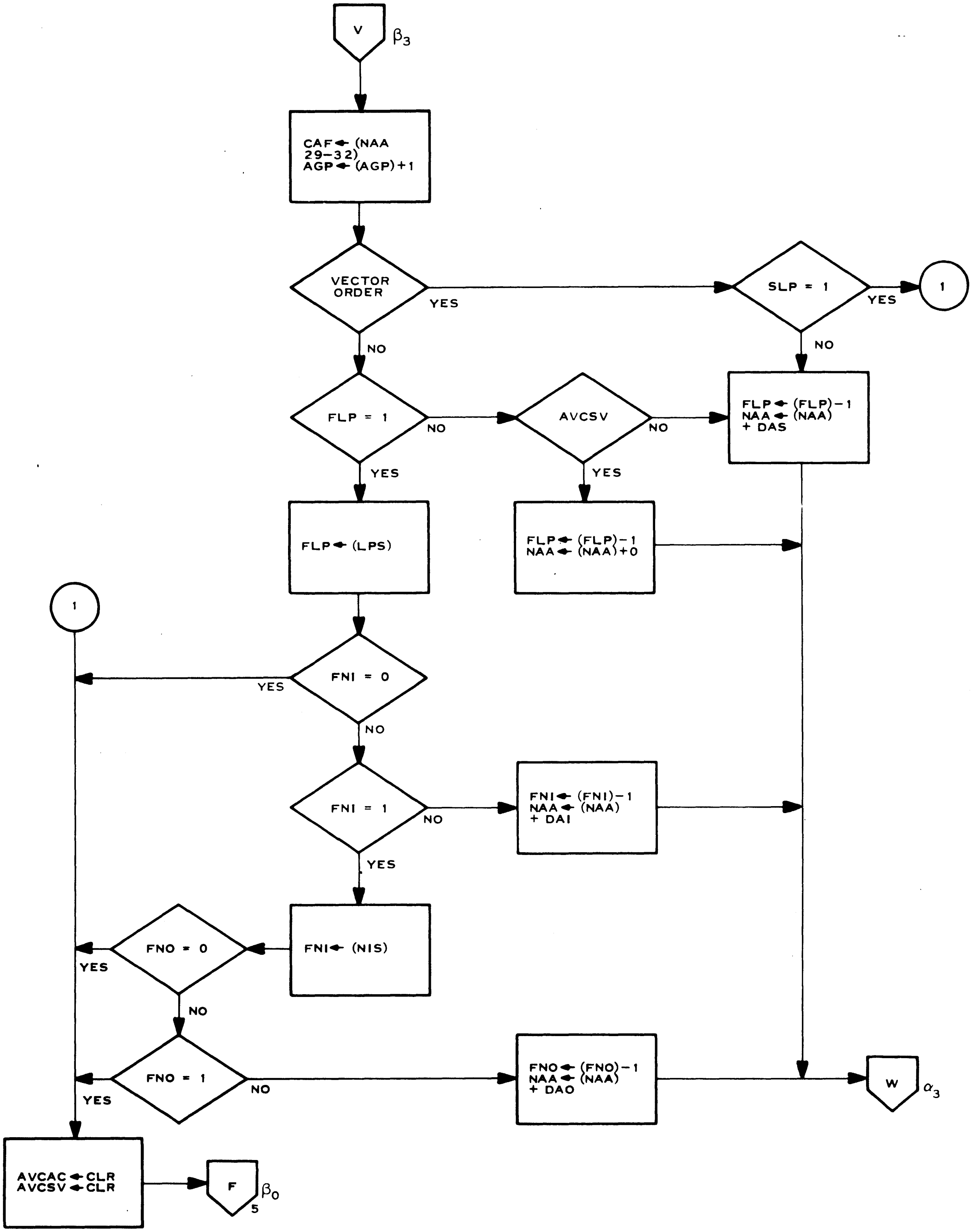
(A)115814

Figure 4-72. A/B Vector Address Generation Flowchart (Sheet 15 of 17)



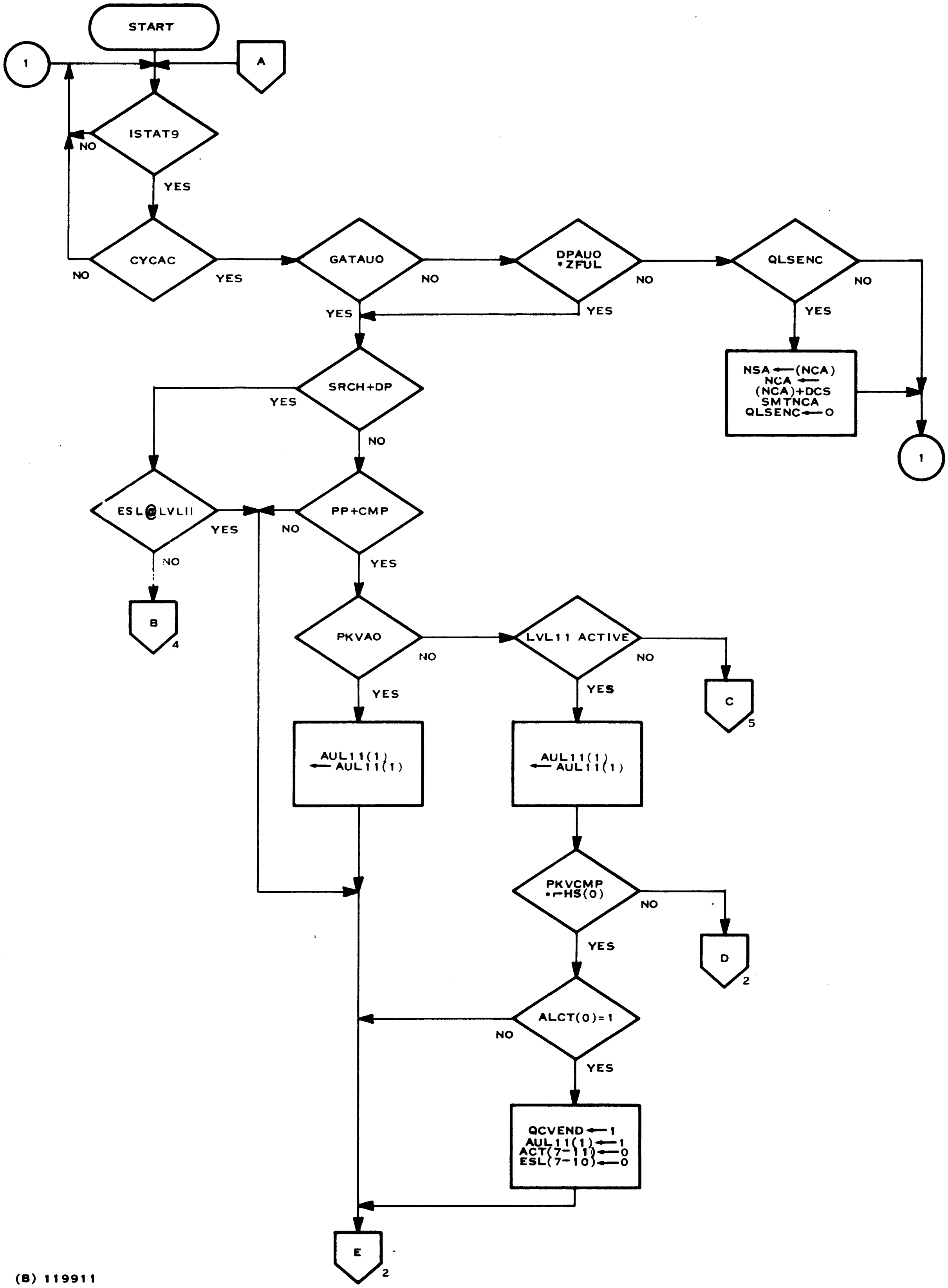
115815

Figure 4-72. A/B Vector Address Generation Flowchart (Sheet 16 of 17)



(A)115816

Figure 4-72. A/B Vector Address Generation Flowchart (Sheet 17 of 17)



(B) 119911

Figure 4-73. C Vector Address Generation Flowchart (Sheet 1 of 6)

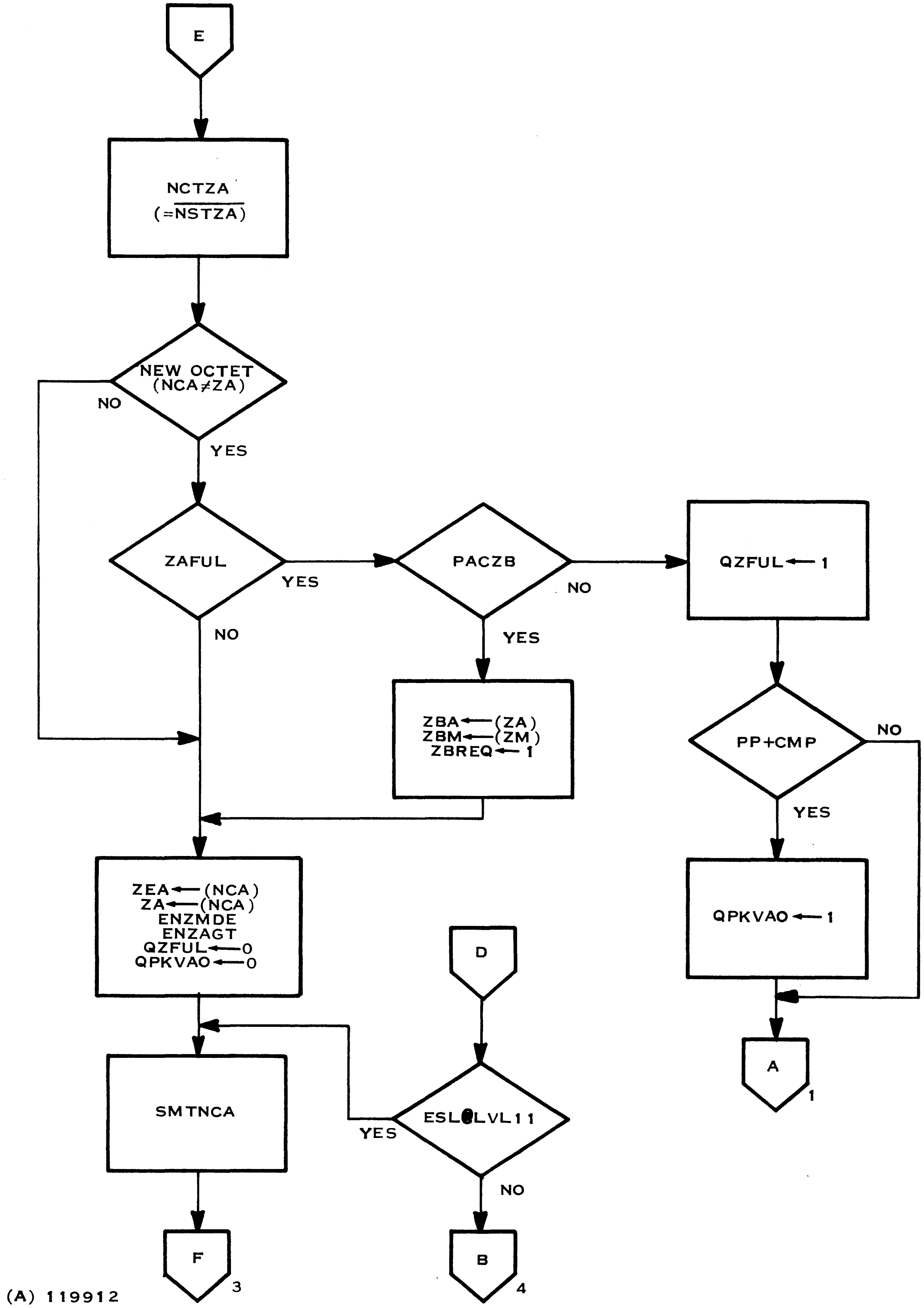


Figure 4-73. C Vector Address Generation Flowchart (Sheet 2 of 6)

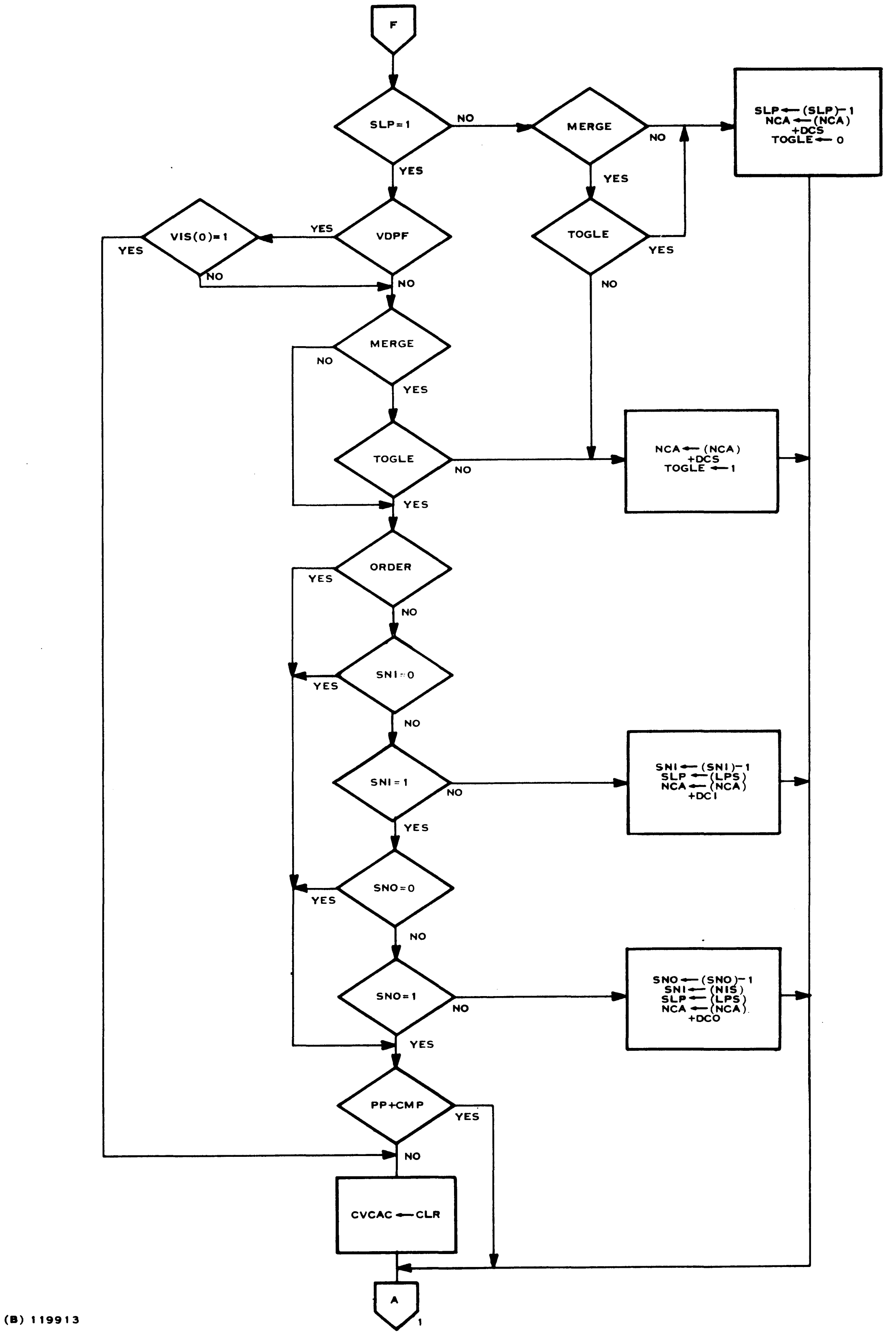
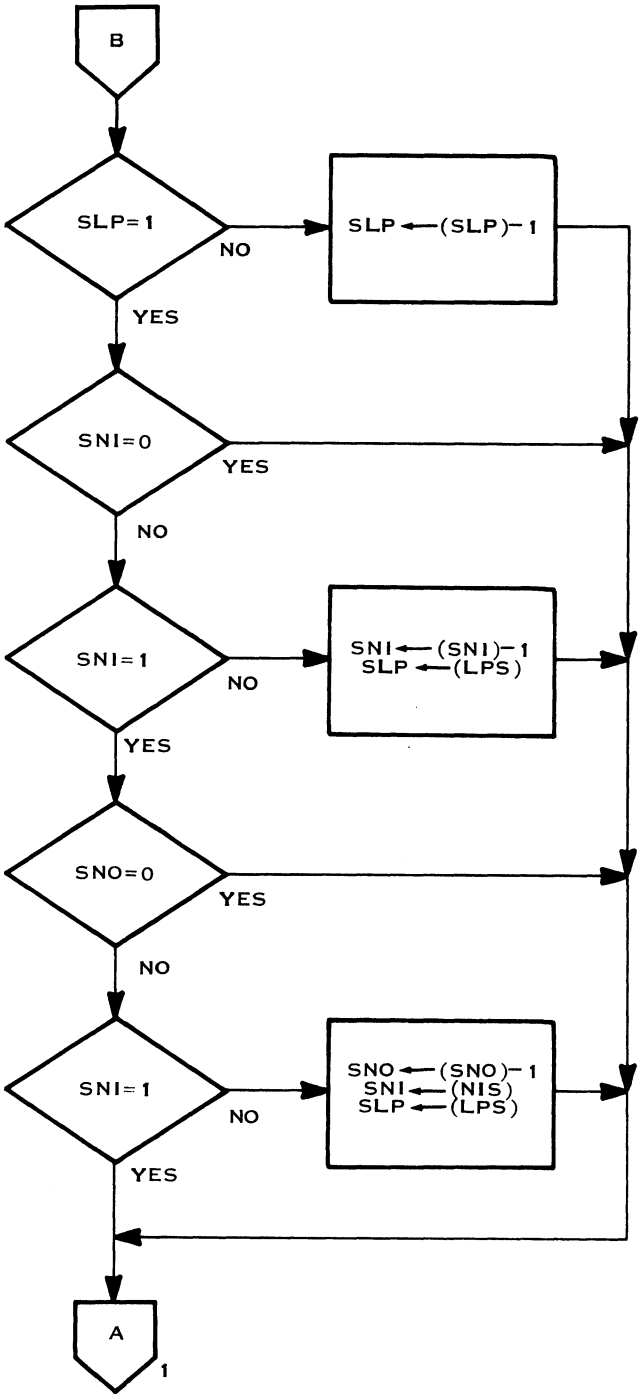
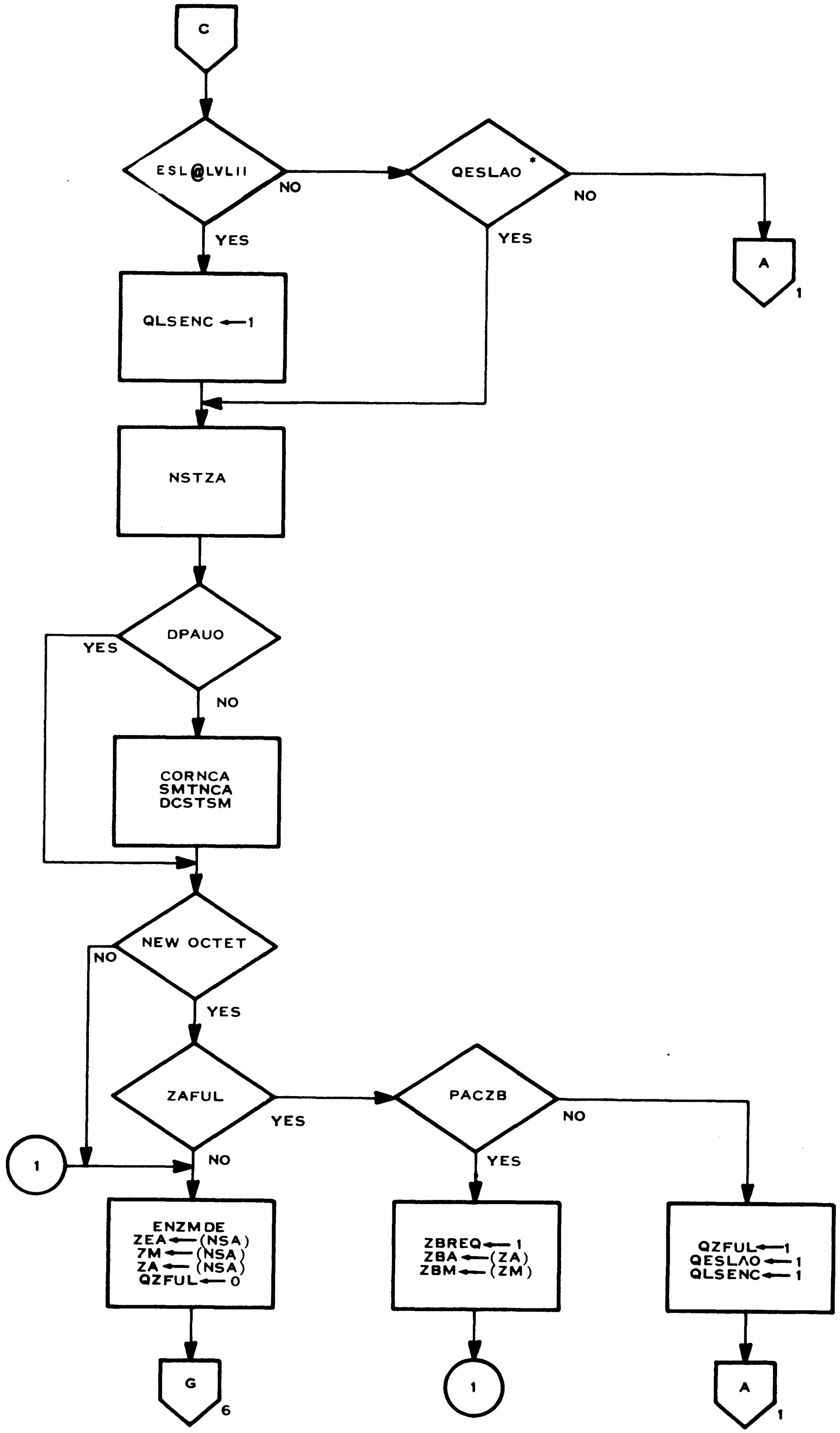


Figure 4-73. C Vector Address Generation Flowchart (Sheet 3 of 6)



(A) 119914

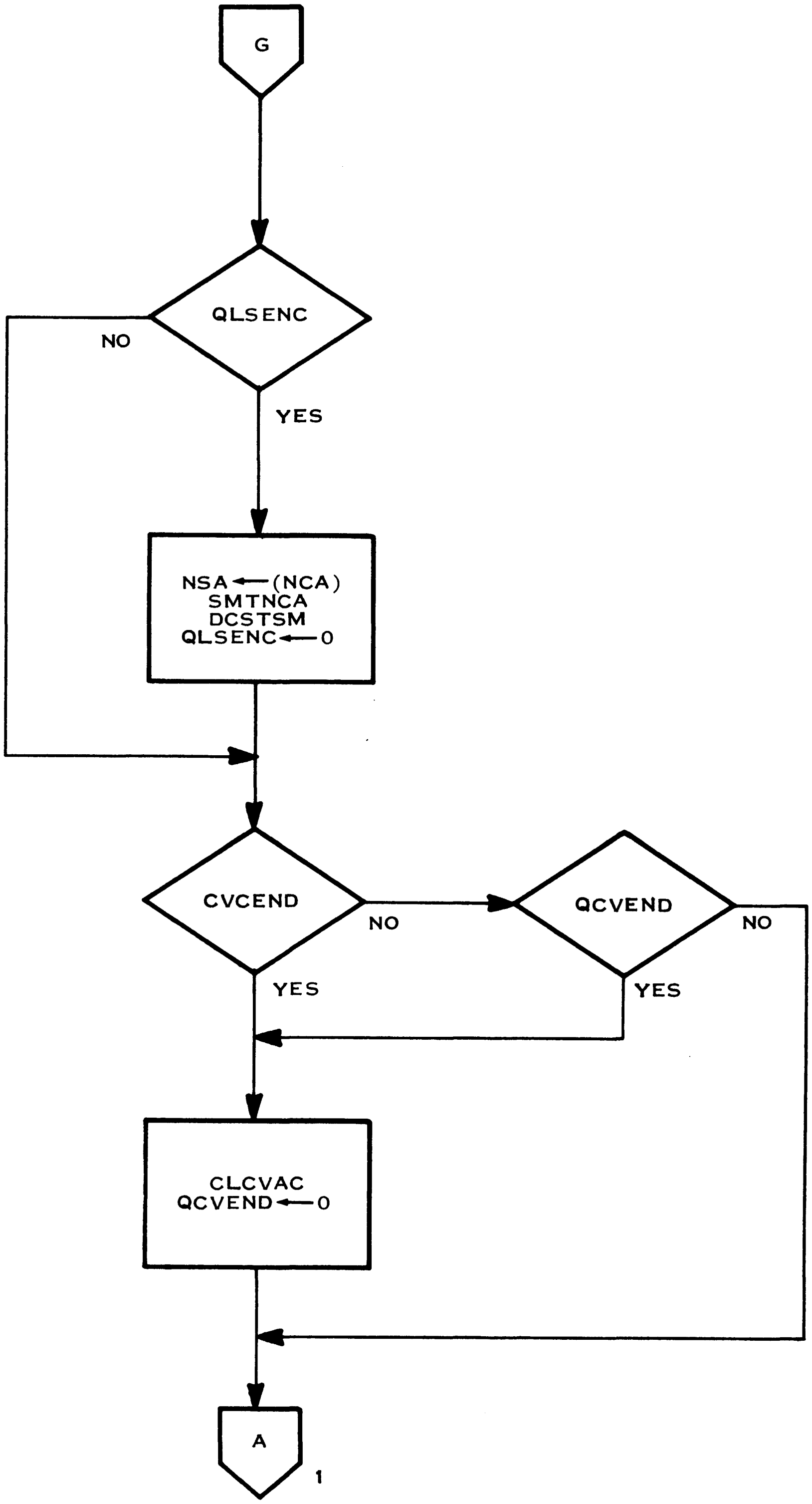
Figure 4-73. C Vector Address Generation Flowchart (Sheet 4 of 6)



* CLEARED BY BCAUTZ

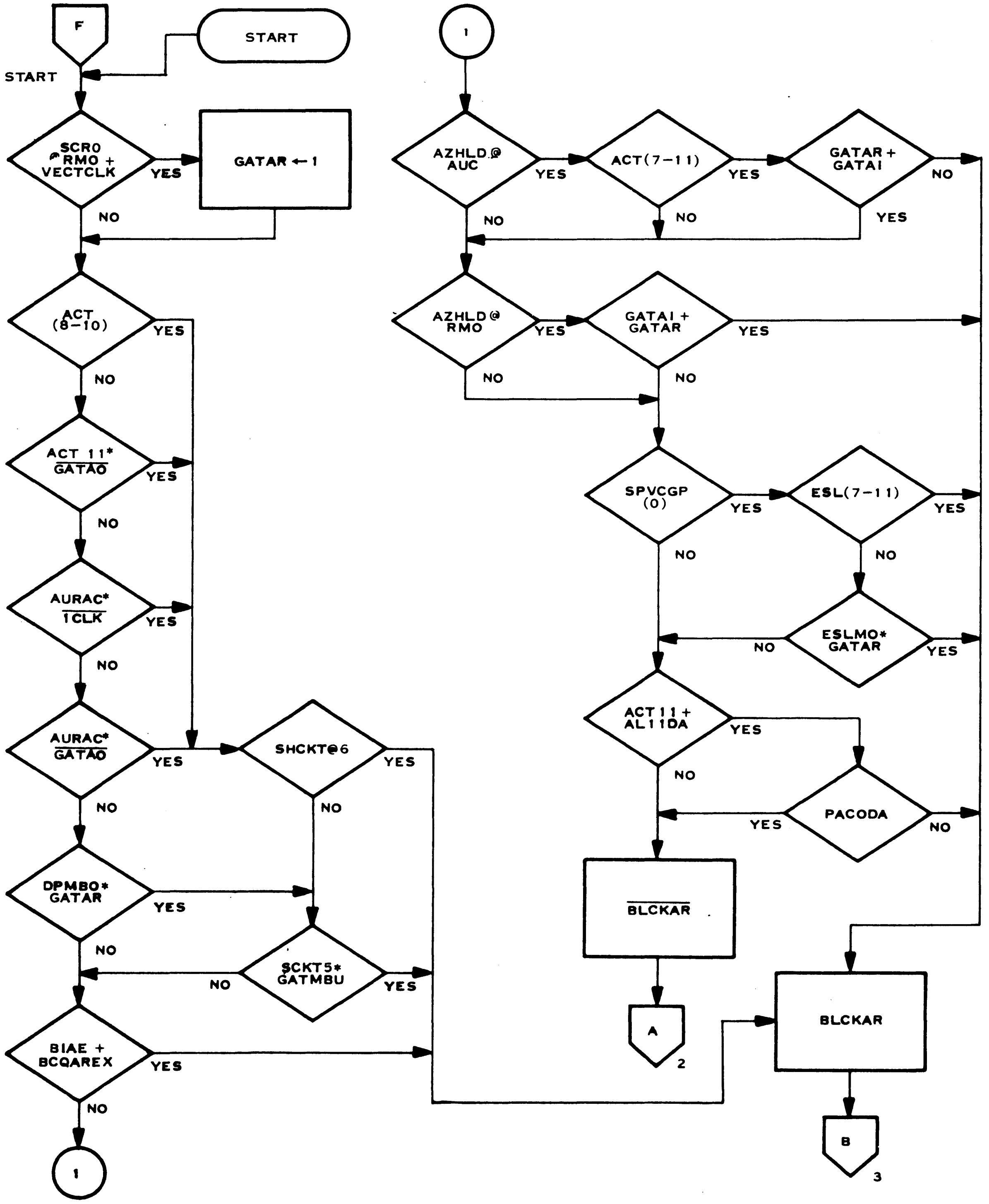
(B) 119915

Figure 4-73. C Vector Address Generation Flowchart (Sheet 5 of 6)



(A) 119916

Figure 4-73. C Vector Address Generation Flowchart (Sheet 6 of 6)



117977

Figure 4-74. AU Control (Sheet 1 of 5)

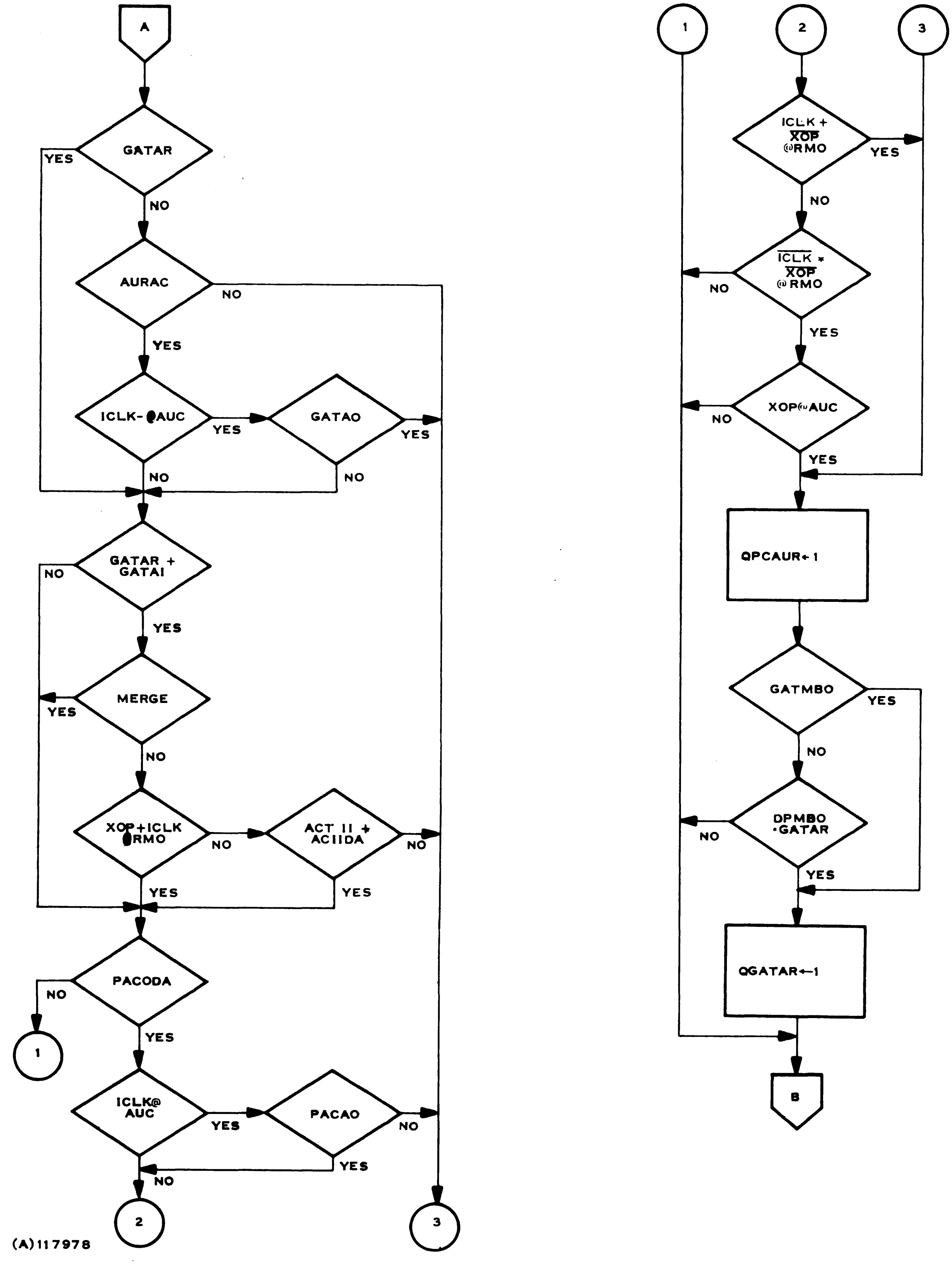


Figure 4-74. AU Control (Sheet 2 of 5)

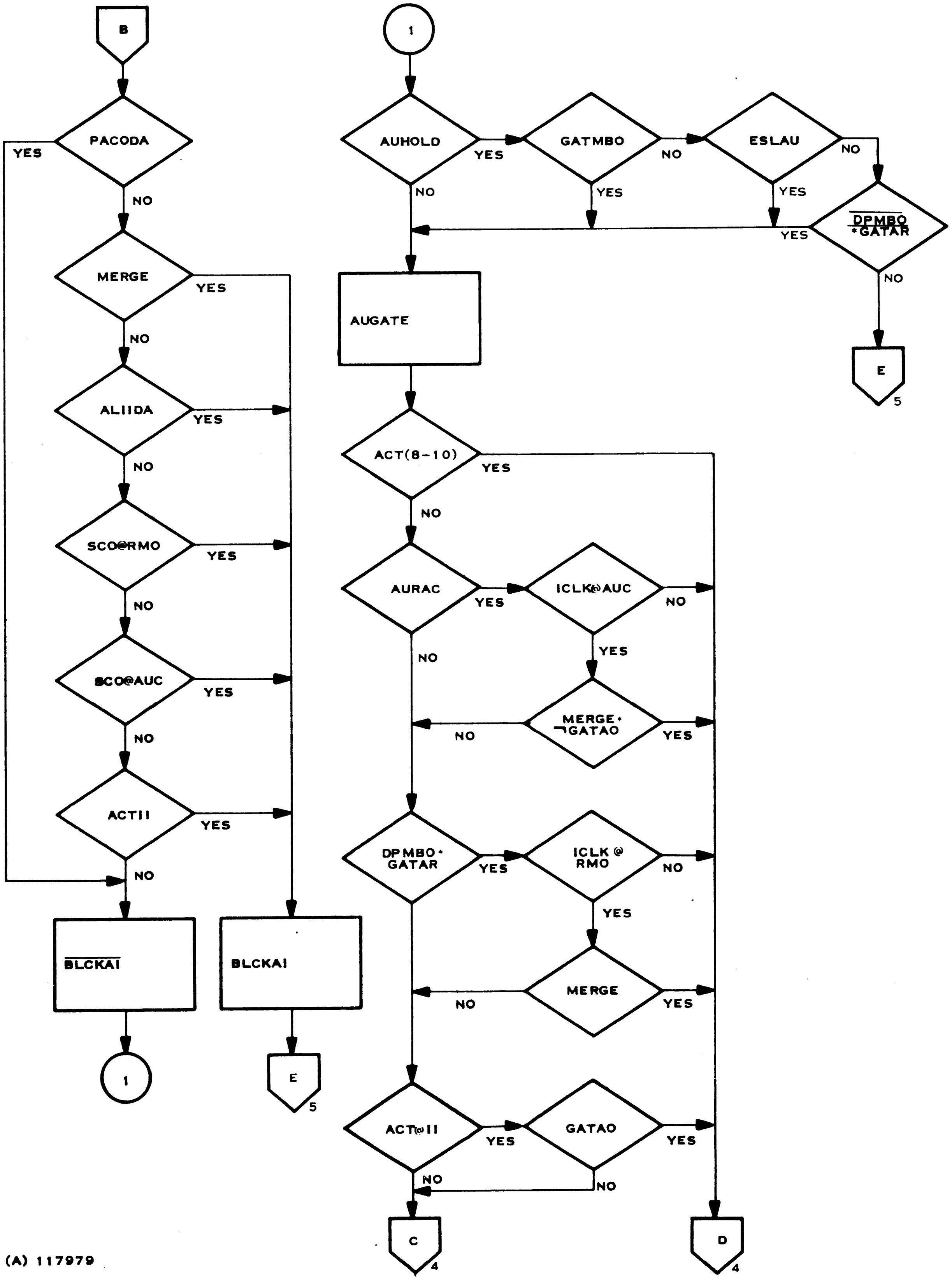
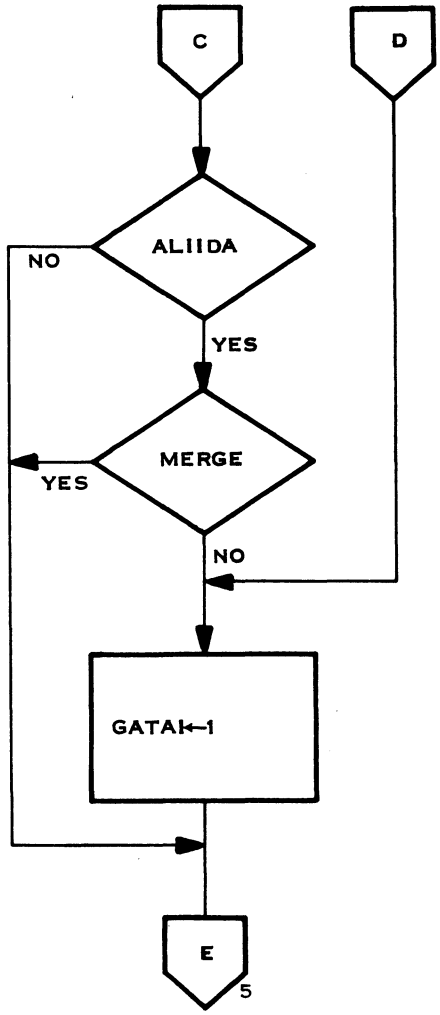
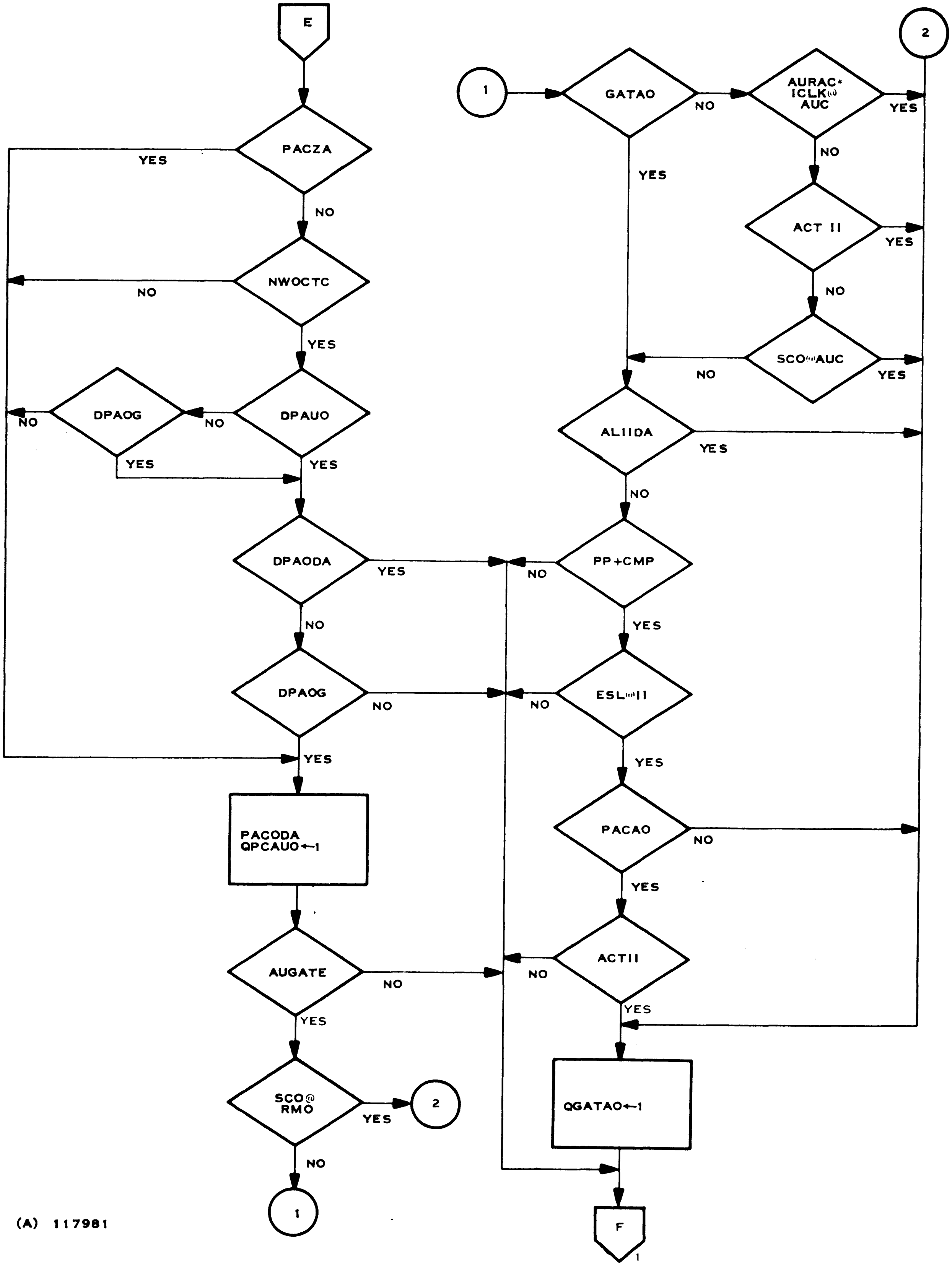


Figure 4-74. AU Control (Sheet 3 of 5)



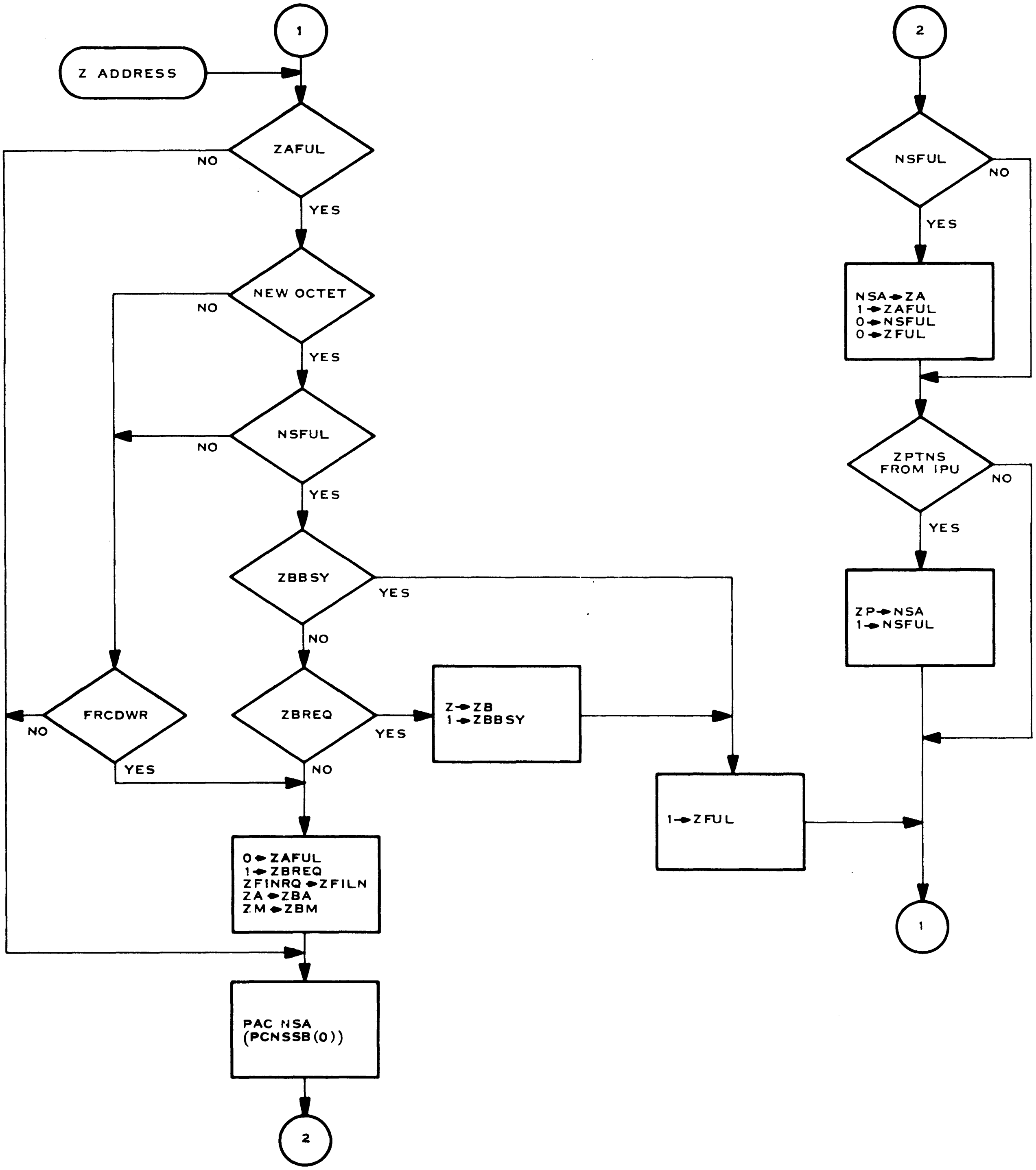
(A) 117980

Figure 4-74. AU Control (Sheet 4 of 5)



(A) 117981

Figure 4-74. AU Control (Sheet 5 of 5)



115785

Figure 4-75. Z Address Flow

SECTION V

MAINTENANCE

Maintenance data consists of a series of diagnostic tests for isolating faulty areas in the IPU, plus use of block diagrams and flowcharts to further isolate malfunctions. Refer to the ASC Mainframe Diagnostics Manual, part number 930064-1, for description of the diagnostic procedures.

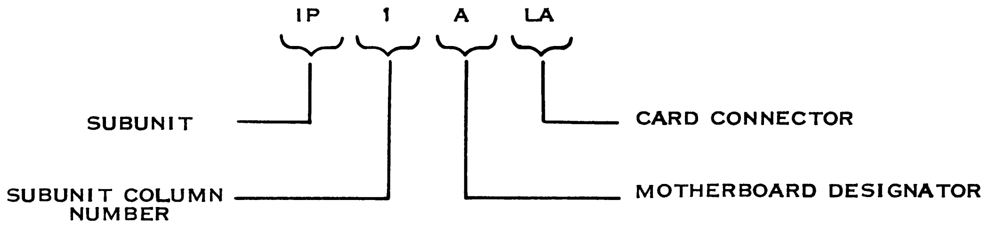
SECTION VI
PARTS LISTING

6-1 INTRODUCTION

This section provides a listing of replaceable parts for the ASC Central Processor and the part numbers associated with each part. These lists are intended as a guide for ordering new replacement parts and installing them in the CP. Minor items, such as screws, washers etc., are not included.

6-2 LOGIC CARDS

Logic cards for the Central Processor are contained in three chassis: the IPU, the MBU and the AU. Each chassis contains three motherboards that hold the logic cards. These motherboards are designated with the letters A, B, and C from top to bottom, respectively. Each card slot in a motherboard is designated with a two letter label, lettered from LA to LV. These designators are used to identify the particular card location in the CP. Figure 6-1 illustrates the information contained in a card location designator. The first two letters refer to the chassis. The next character is always a one in the CP. The fourth character designates the motherboard within that chassis, and the last two letters identify the card slot on that motherboard. Table 6-1 lists all logic circuits used in the CP arranged by card location.



(A) 115137

Figure 6-1. Card Location Information

Table 6-1. Central Processor Logic Cards

Card Location	Function	Part Number	Card Location	Function	Part Number
IP1 A LA	DUMMY	695011-1	IP1 C LB	DUMMY	695011-1
IP1 A LB	DLOGCLK (0)	650356-1	IP1 C LC	DUMMY	695011-1
IP1 A LC	DUMMY	695011-1	IP1 C LD	TERMCRD	650296-1
IP1 A LD	IUADDR (0)	686517-1	IP1 C LE	DUMMY	695011-1
IP1 A LE	IUADDR (1)	686517-1	IP1 C LF	IUPIPEA (0)	911793-1
IP1 A LF	IUFILE (0)	686484-1	IP1 C LG	IUPIPEA (1)	911793-1
IP1 A LG	IUFILE (1)	686484-1	IP1 C LH	IUPIPEA (2)	911793-1
IP1 A LH	IUFILE (2)	686484-1	IP1 C LI	IUPIPEA (3)	911793-1
IP1 A LI	IUFILE (3)	686484-1	IP1 C LJ	DROMCRDA (0)	650299-37
IP1 A LJ	IUFILE (4)	686484-1	IP1 C LK	BUHOOA	710297-1
IP1 A LK	IUFILE (5)	686484-1	IP1 C LL	DROMCRDB (0)	650299-38
IP1 A LL	IUFILE (6)	686484-1	IP1 C LM	IUPIPEA (4)	911793-1
IP1 A LM	IUFILE (7)	686484-1	IP1 C LN	IUPIPEA (5)	911793-1
IP1 A LN	IUFILE (8)	686484-1	IP1 C LO	IUPIPEA (6)	911793-1
IP1 A LO	IUFILE (9)	686484-1	IP1 C LP	IUPIPEA (7)	911793-1
IP1 A LP	IUFILE (10)	686484-1	IP1 C LQ	DUMMY	695011-1
IP1 A LQ	IUFILE (11)	686484-1	IP1 C LR	TERMCRD	650296-1
IP1 A LR	IUFILE (12)	686484-1	IP1 C LS	DUMMY	695011-1
IP1 A LS	IUFILE (13)	686484-1	IP1 C LT	DUMMY	695011-1
IP1 A LT	IUFILE (14)	686484-1	IP1 C LU	DUMMY	695011-1
IP1 A LU	IUFILE (15)	686484-1	IP1 C LV	DUMMY	695011-1
IP1 A LV	TERMCRD	650296-1	MB1 A LA	DROMCRDA (0)	650299-17
IP1 B LA	DUMMY	695011-1	MB1 A LB	DROMCRDB (0)	650299-18
IP1 B LB	DUMMY	695011-1	MB1 A LC	BUROM (0)	686490-1
IP1 B LC	TERMCRD	650296-1	MB1 A LD	DROMCRDB (1)	650299-19
IP1 B LD	IUCONTA (0)	911835-1	MB1 A LE	DROMCRDA (1)	650299-20
IP1 B LE	DUMMY	695011-1	MB1 A LF	DROMCRDA (2)	650299-21
IP1 B LF	IUHAZ (3)	686514-1	MB1 A LG	DROMCRDB (2)	650299-22
IP1 B LG	IUHAZ (2)	686514-1	MB1 A LH	BUROM (1)	686490-1
IP1 B LH	IUHAZ (1)	686514-1	MB1 A LI	DROMCRDB (3)	650299-23
IP1 B LI	IUHAZ (0)	686514-1	MB1 A LJ	DROMCRDA (3)	650299-24
IP1 B LJ	DUMMY	695011-1	MB1 A LK	BUHOOA	710297-1
IP1 B LK	BUHOOA	710297-1	MB1 A LL	DROMCRDA (4)	650299-25
IP1 B LL	IUCTLA (0)	911832-1	MB1 A LM	DROMCRDB (4)	650299-26
IP1 B LM	IUNEWA (0)	911820-1	MB1 A LN	BUROM (2)	686490-1
IP1 B LN	DUMMY	695011-1	MB1 A LO	DROMCRDB (5)	650299-27
IP1 B LO	IUPACB (0)	911829-1	MB1 A LP	DROMCRDA (5)	650299-28
IP1 B LP	IUINFACA (0)	911826-1	MB1 A LQ	DROMCRDA (6)	650299-29
IP1 B LQ	DUMMY	695011-1	MB1 A LR	DROMCRDB (6)	650299-30
IP1 B LR	IULASTA (0)	911787-1	MB1 A LS	BUROM (3)	686490-1
IP1 B LS	IUMISCA (0)	911814-1	MB1 A LT	DROMCRDB (7)	650299-31
IP1 B LT	TERMCRD	650296-1	MB1 A LU	DROMCRDA (7)	65029 F32
IP1 B LU	DUMMY	695011-1	MB1 A LV	TERMCRD	650296-1
IP1 B LV	DUMMY	695011-1	MB1 B LA	DUMMY	695011-1
IP1 C LA	DUMMY	695011-1	MB1 B LB	DUMMY	695011-1

Table 6-1. Central Processor Logic Cards (Continued)

Card Location	Function	Part Number	Card Location	Function	Part Number
MB1 B LC	DUMMY	695011-1	AU1 A LD	AUADD (0)	650368-1
MB1 B LD	BUZAG (0)	650362-1	AU1 A LE	AUADD (1)	650368-1
MB1 B LE	BUDATA (0)	650365-1	AU1 A LF	AUADD (2)	650368-1
MB1 B LF	BUDATA (1)	650365-1	AU1 A LG	AUADD (3)	650368-1
MB1 B LG	BUDATA (2)	650365-1	AU1 A LH	AUADD (4)	650368-1
MB1 B LH	BUDATA (3)	650365-1	AU1 A LI	AUADD (5)	650368-1
MB1 B LI	BUDATA (4)	650365-1	AU1 A LJ	AUADD (6)	650368-1
MB1 B LJ	BUDATA (5)	650365-1	AU1 A LK	AUADD (7)	650368-1
MB1 B LK	BUDATA (6)	650365-1	AU1 A LL	AUCTLI (0)	650371-1
MB1 B LL	BUDATA (7)	650365-1	AU1 A LM	AUADD (8)	650368-1
MB1 B LM	BUDATA (8)	650365-1	AU1 A LN	AUADD (9)	650368-1
MB1 B LN	BUDATA (9)	650365-1	AU1 A LO	AUADD (10)	650368-1
MB1 B LO	BUDATA (10)	650365-1	AU1 A LP	AUADD (11)	650368-1
MB1 B LP	BUDATA (11)	650365-1	AU1 A LQ	AUADD (12)	650368-1
MB1 B LQ	BUDATA (12)	650365-1	AU1 A LR	AUADD (13)	650368-1
MB1 B LR	BUDATA (13)	650365-1	AU1 A LS	AUADD (14)	650368-1
MB1 B LS	BUDATA (14)	650365-1	AU1 A LT	AUADD (15)	650368-1
MB1 B LT	BUDATA (15)	650365-1	AU1 A LU	DLOGCLK	650356-1
MB1 B LU	TERMCRD	650296-1	AU1 A LV	TERMCRD	650296-1
MB1 B LV	DUMMY	695011-1	AU1 B LA	TERMCRD	650296-1
MB1 C LA	TERMCARD	650296-1	AU1 B LB	AUOUTA (0)	911805-1
MB1 C LB	BUHOOA	710297-1	AU1 B LC	AUOUTA (1)	911805-1
MB1 C LC	BUCAF (0)	686511-1	AU1 B LD	AUOUTA (2)	911805-1
MB1 C LD	BUCAF (1)	686511-1	AU1 B LE	AUOUTA (3)	911805-1
MB1 C LE	BUCTLB	911838-1	AU1 B LF	AUOUTA (4)	911805-1
MB1 C LF	BUACTLB	911817-1	AU1 B LG	AUOUTA (5)	911805-1
MB1 C LG	BULOOP (0)	686478-1	AU1 B LH	AUOUTA (6)	911805-1
MB1 C LH	BULOOP (1)	686478-1	AU1 B LI	AUOUTA (7)	911805-1
MB1 C LI	BULOOP (2)	686478-1	AU1 B LJ	AUCTL3A	911808-1
MB1 C LJ	BULOOP (3)	686478-1	AU1 B LK	AUROMFFA	911811-1
MB1 C LK	BUADDR (0)	686475-1	AU1 B LL	AUCTL2A	911823-1
MB1 C LL	BUADDR (1)	686475-1	AU1 B LM	AUCTL4	686508-1
MB1 C LM	BUADDR (2)	686475-1	AU1 B LN	AUNORM (0)	686493-1
MB1 C LN	BUADDR (3)	686475-1	AU1 B LO	AUNORM (1)	686493-1
MB1 C LO	BUADDR (4)	686475-1	AU1 B LP	AUNORM (2)	686493-1
MB1 C LP	LOGCLK	650356-1	AU1 B LQ	AUNORM (3)	686493-1
MB1 C LQ	LOGCLK	650356-1	AU1 B LR	AUNORM (4)	686493-1
MB1 C LR	DUMMY	695011-1	AU1 B LS	AUNORM (5)	686493-1
MB1 C LS	TERMCRD	650296-1	AU1 B LT	AUNORM (6)	686493-1
MB1 C LT	BUMHCI	710258-1	AU1 B LU	AUNORM (7)	686493-1
MB1 C LU	DUMMY	695011-1	AU1 B LV	TERMCRD	650296-1
MB1 C LV	DUMMY	695011-1	AU1 C LA	TERMCRD	650296-1
AU1 A LA	AUXSEL (0)	911802-1	AU1 C LB	AUMULT (0)	686520-1
AU1 A LB	AUXSEL (1)	911802-1	AU1 C LC	AUSUMD (10)	686487-1
AU1 A LC	TERMCARD	650926-1	AU1 C LD	AUMULT (1)	686520-1

Table 6-1. Central Processor Logic Cards (Continued)

Card Location	Function	Part Number	Card Location	Function	Part Number
AU1 C LE	AUSUMD (8)	686487-1	AU1 C LN	AUSUMD (1)	686487-1
AU1 C LF	AUSUMD (4)	686487-1	AU1 C LO	AUMULT (5)	686520-1
AU1 C LG	AUMULT (2)	686520-1	AU1 C LP	AUSUMD (9)	686487-1
AU1 C LH	AUSUMD (2)	686487-1	AU1 C LQ	AUSUMD (3)	686487-1
AU1 C LI	AUSUMD (7)	686487-1	AU1 C LR	AUMULT (6)	686520-1
AU1 C LJ	AUMULT (3)	686520-1	AU1 C LS	AUSUMD (0)	686487-1
AU1 C LK	AUSUMD (6)	686487-1	AU1 C LT	AUMULT (7)	686520-1
AU1 C LL	AUSUMD (5)	686487-1	AU1 C LU	AUCTL5A (0)	911799-1
AU1 C LM	AUMULT (4)	686520-1	AU1 C LV	TERMCRD	650296-1

SECTION VII
DIAGRAMS

(To be supplied on site.)

APPENDIX A
LEVEL 2 ROM CONTENTS

LEVEL 2 ROM CONTENTS (Sheet 1 of 16)
OP CODE 0X

X = Least Significant Hex of Op Code

OUTPUT BIT NUMBER ↓	F E D C	B A 9 8	7 6 5 4	NOP 3 2 1 0	OUTPUT SIGNATURE IPQRM___ ↓
0 1 2 3	0	0	0	0	RS0 RS1 RS6 BR
4 5 6 7	0	0	0	0	RS8 RS9 RDT DW
8 9 10 11	0	0	0	0	CHZ AEH LAE EXN
12 13 14 15	0	0	0	0	LAM RHZ CAR HW
16 17 18 19	0	0	0	0	M AU4 AU5 IOP
20 21 22 23	0	0	0	0	AU6 AU7 GP1 PP
24 25 26 27	0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1	GP2 GP3 GP4 SW
28 29 30 31	0	0	0	0	EXM LF ORG VCT

LEVEL 2 ROM CONTENTS (Sheet 2 of 16)
OP CODE 1X

X = Least Significant Hex of Op Code

OUTPUT BIT NUMBER ↓	LFM LO LR L	LF XCH LL L	LD LLA LH L	LAE· LAM	OUTPUT SIGNATURE IPQRM___ ↓
	F E D C	B A 9 8	7 6 5 4	3 2 1 0	
0	0 0 0 1	0 0 0 0	0 0 0 0	1 1 0 0	RS0
1	0 1 1 0	0 1 1 0	1 0 1 1	1 1 0 0	RS1
2	0 0 1 0	0 0 0 0	0 0 0 0	0 0 0 0	RS6
3	1 1 1 1	1 1 1 1	1 0 1 1	1 1 0 0	BR
4	0 0 1 0	0 0 1 0	0 0 1 0	0	RS8
5	0 0 0 0	0 0 0 0	1 0 0 0		RS9
6	0 1 1 1	0 1 1 1	1 0 1 1		RDT
7	0 0 0 0	0 0 0 0	1 0 0 0		DW
8	0	0	0	0 0 0 0	CHZ
9				1 1 0 0	AEH
10				1 0 0 0	LAE
11				0 0 0 0	EXN
12	0 0 0 0	0 0 0 0	0 0 0 0	0 1 0 0	LAM
13	0 1 1 1	0 0 1 1	1 0 1 1	0 0 0 0	RHZ
14	0 0 1 0	0 0 1 0	0 0 0 0	0 0 0 0	CAR
15	0 0 1 0	0 0 1 0	0 0 1 0	0 0 0 0	HW
16	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	M
17	0 1 0 0	0 1 0 0	0 0 0 0	0 0 0 0	AU4
18	0 1 1 1	0 0 1 1	1 0 1 1	1 1 0 0	AU5
19	0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 1	IOP
20	0 1 1 1	0 1 1 1	1 0 1 1	1 1 0 0	AU6
21	0 0 1 1	0 0 1 1	1 0 1 1	1 1 0 0	AU7
22	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	GP1
23	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	PP
24	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	GP2
25	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	GP3
26	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	GP4
27	1 1 0 1	1 1 0 1	0 0 0 1	1 1 0 0	SW
28	0 0 0 0	0 0 0 0	0	0	EXM
29	1 0 0 0	1 0 0 0			LF
30	1 0 0 0	1 0 0 0			ORG
31	0 0 0 0	0 0 0 0			VCT

LEVEL 2 ROM CONTENTS (Sheet 3 of 16)

OP CODE 2X

X = Least Significant Hex of Op Code

OUTPUT BIT NUMBER ↓	STFM	STO	STR	ST	STF	STOH	STL	ST	STD		STH	ST	STZD	SPS	STZH	STZ	OUTPUT SIGNATURE IPQRM—— ↓
	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	1	1	RS0
1	0	1	1	0	0	1	1	0	1	0	1	1	1	1	1	1	RS1
2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	RS6
3	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	BR
4	0	0	1	0	0	1	1	0	0	0	1	0	0	0	1	0	RS8
5	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	RS9
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	RDT
7	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	DW
8	0				0				0				0				CHZ
9																	AEH
10																	LAE
11																	EXN
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	LAM
13	0	1	1	1	0	1	1	1	1	0	1	1	1	0	1	1	RHZ
14	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	CAR
15	0	0	1	0	0	1	1	0	0	0	1	0	0	0	1	0	HW
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	M
17	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	AU4
18	0	1	1	1	0	1	1	1	1	0	1	1	0	1	0	0	AU5
19	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	IOP
20	0	1	1	1	0	1	1	1	1	0	1	1	0	1	0	0	AU6
21	0	0	1	1	0	0	1	1	1	0	1	1	0	1	0	0	AU7
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	GP1
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	PP
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	GP2
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	GP3
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	GP4
27	1	1	0	1	1	0	0	1	0	0	0	1	0	1	0	1	SW
28	0	0	0	0	0	0	0	0	0				0				EXM
29	0	0	0	0	0	0	0	0									LF
30	1	0	0	0	1	0	0	0									ORG
31	0	0	0	0	0	0	0	0									VCT

LEVEL 2 ROM CONTENTS (Sheet 4 of 16)

OP CODE 3X

X = Least Significant Hex of Op Code

OUTPUT BIT NUMBER	LMD LMF LMH LM	LNMD LNMF LNMH LNM	STND STNF STNH STN	LND LNF LNH LN	OUTPUT SIGNATURE IPQRM___
↓	F E D C	B A 9 8	7 6 5 4	3 2 1 0	↓
0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	RS0
1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	RS1
2	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	RS6
3	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	BR
4	0 0 1 0	0 0 1 0	0 0 1 0	0 0 1 0	RS8
5	1 0 0 0	1 0 0 0	1 0 0 0	1 0 0 0	RS9
6	1 1 1 1	1 1 1 1	0 0 0 0	1 1 1 1	RDT
7	1 0 0 0	1 0 0 0	1 0 0 0	1 0 0 0	DW
8	0 0 0 0	0	0 0 0 0	0 0 0 0	CHZ
9	0 0 1 1		0 0 1 1	0 0 1 1	AEH
10	0 0 0 0		0 0 0 0	0 0 0 0	LAE
11	0 0 0 0		0 0 0 0	0 0 0 0	EXN
12	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	LAM
13	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	RHZ
14	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	CAR
15	0 0 1 0	0 0 1 0	0 0 1 0	0 0 1 0	HW
16	0 0 0 0	0 0 0 0	0 0 0 0	0	M
17	1 1 1 1	1 1 1 1	0 0 0 0		AU4
18	1 1 1 1	0 0 0 0	1 1 1 1		AU5
19	0 0 0 0	0 0 0 0	0 0 0 0		IOP
20	1 1 0 0	1 1 0 0	1 1 0 0	1 1 0 0	AU6
21	1 1 0 0	1 1 0 0	1 1 0 0	1 1 0 0	AU7
22	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	GP1
23	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	PP
24	0 0 1 1	0 0 1 1	0 0 1 1	0 0 1 1	GP2
25	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	GP3
26	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	GP4
27	0 1 0 1	0 1 0 1	0 1 0 1	0 1 0 1	SW
28	0	0	0	0	EXM
29					LF
30					ORG
31					VCT

LEVEL 2 ROM CONTENTS (Sheet 5 of 16)

OP CODE 4X

X = Least Significant Hex of Op Code

OUTPUT BIT NUMBER	SMFD SMF SMH SM	SFD SF SH S	AMFD AMF AMH AM	AFD AF AH A	OUTPUT SIGNATURE IPQRM
↓	F E D C	B A 9 8	7 6 5 4	3 2 1 0	↓
0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	RS0
1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	RS1
2	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	RS6
3	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	BR
4	0 0 1 0	0 0 1 0	0 0 1 0	0 0 1 0	RS8
5	1 0 0 0	1 0 0 0	1 0 0 0	1 0 0 0	RS9
6	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	RDT
7	1 0 0 0	1 0 0 0	1 0 0 0	1 0 0 0	DW
8	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	CHZ
9	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	AEH
10	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	LAE
11	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	EXN
12	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	LAM
13	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	RHZ
14	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	CAR
15	0 0 1 0	0 0 1 0	0 0 1 0	0 0 1 0	HW
16	0 0 0 0	0 0 0 0	0 0 0 0	0	M
17	1 1 1 1	1 1 1 1	0 0 0 0		AU4
18	1 1 1 1	0 0 0 0	1 1 1 1		AU5
19	0 0 0 0	0 0 0 0	0 0 0 0		IOP
20	1 1 0 0	1 1 0 0	1 1 0 0	1 1 0 0	AU6
21	1 1 0 0	1 1 0 0	1 1 0 0	1 1 0 0	AU7
22	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	GP1
23	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	PP
24	0 0 1 1	0 0 1 1	0 0 1 1	0 0 1 1	GP2
25	1 1 0 0	1 1 0 0	1 1 0 0	1 1 0 0	GP3
26	1 1 0 0	1 1 0 0	1 1 0 0	1 1 0 0	GP4
27	0 1 0 1	0 1 0 1	0 1 0 1	0 1 0 1	SW
28	0	0	0	0	EXM
29					LF
30					ORG
31					VCT

LEVEL 2 ROM CONTENTS (Sheet 6 of 16)
OP CODE 5X

X = Least Significant Hex of Op Code

OUTPUT BIT NUMBER ↓	LI F E D C	SIH SI B A 9 8	LEA LIH LI 7 6 5 4	LEA AIH AI 3 2 1 0	OUTPUT SIGNATURE IPQRM___ ↓
0	0 0 0 1	0 0 0 0	0 1 0 0	0 0 0 0	RS0
1	0 0 0 0	0 0 1 1	0 0 1 1	0 0 1 1	RS1
2	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	RS6
3	0 0 0 0	0 0 0 0	0 1 0 0	0 1 0 0	BR
4	0 0 0 0	0 0 1 0	0 0 1 0	0 0 1 0	RS8
5	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	RS9
6	0 0 0 1	0 0 1 1	0 1 1 1	0 1 1 1	RDT
7	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	DW
8	0	0 0 0 0	0	0 0 0 0	CHZ
9		0 0 1 1		0 0 1 1	AEH
10		0 0 0 0		0 0 0 0	LAE
11		0 0 0 0		0 0 0 0	EXN
12	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	LAM
13	0 0 0 1	0 0 1 1	0 0 1 1	0 0 1 1	RHZ
14	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	CAR
15	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	HW
16	0 0 0 1	0 0 1 1	0 0 1 1	0 0 1 1	M
17	0 0 0 0	0 0 1 1	0 0 0 0	0 0 0 0	AU4
18	0 0 0 1	0 0 0 0	0 1 1 1	0 1 0 0	AU5
19	1 1 1 0	1 1 0 0	1 0 0 0	1 0 0 0	IOP
20	0	0	0	0	AU6
21					AU7
22					GP1
23					PP
24	0 0 0 0	0 0 1 1	0 0 0 0	0 0 1 1	GP2
25	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	GP3
26	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	GP4
27	0 0 0 1	0 0 1 1	0 1 1 1	0 1 1 1	SW
28	0 0 0 1	0 0 0 1	0 0 0 1	0 0 0 1	EXM
29	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	LF
30	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	ORG
31	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	VCT

LEVEL 2 ROM CONTENTS (Sheet 7 of 16)
OP CODE 6X

X = Least Significant Hex of Op Code

OUTPUT BIT NUMBER	MFD MF MH M	M M	DFD DF DH D	A A	OUTPUT SIGNATURE IPQRM
↓	F E D C	B A 9 8	7 6 5 4	3 2 1 0	↓
0	0 0 0 0	0 1 0 0	0 0 0 0	0 1 0 0	RS0
1	1 1 1 1	0 0 0 0	1 1 1 1	0 0 0 0	RS1
2	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	RS6
3	1 1 1 1	0 1 0 1	1 1 1 1	0 1 0 1	BR
4	0 0 0 0	0 0 0 0	0 0 1 0	0 0 0 0	RS8
5	1 0 0 1	0 0 0 0	1 0 0 0	0 0 0 0	RS9
6	1 1 1 1	0 1 0 1	1 1 1 1	0 1 0 1	RDT
7	1 0 0 0	0 0 0 0	1 0 0 0	0 0 0 0	DW
8	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	CHZ
9	1 1 0 1	0 1 0 1	1 1 1 1	0 1 0 1	AEH
10	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	LAE
11	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	EXN
12	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	LAM
13	1 1 1 1	0 1 0 1	1 1 1 1	0 1 0 1	RHZ
14	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	CAR
15	0 0 1 0	0 0 0 0	0 0 1 0	0 0 0 0	HW
16	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	M
17	1 1 1 1	0 1 0 1	0 0 0 0	0 0 0 0	AU4
18	1 1 1 1	0 0 0 0	1 1 1 1	0 0 0 0	AU5
19	0 0 0 0	1 0 1 0	0 0 0 0	1 0 1 0	IOP
20	1 1 0 0	0	1 1 0 0	0	AU6
21	1 0 1 0		1 0 1 0		AU7
22	0 0 0 1		1 1 1 0		GP1
23	0 0 0 0		0 0 0 0		PP
24	1 1 0 0	0 0 0 0	0 0 0 1	0 1 0 1	GP2
25	1 0 1 1	0 0 0 0	1 0 0 1	0 0 0 0	GP3
26	0 1 0 1	0 1 0 1	0 1 0 1	0 0 0 0	GP4
27	0 1 0 1	0 1 0 1	0 1 0 1	0 1 0 1	SW
28	0	0	0	0	EXM
29					LF
30					ORG
31					VCT

LEVEL 2 ROM CONTENTS (Sheet 8 of 16)

OP CODE 7X

X = Least Significant Bit of Hex Code

OUTPUT BIT NUMBER ↓	MIH MI F E D C	MI MI B A 9 8	DIH DI 7 6 5 4	AI AI 3 2 1 0	OUTPUT SIGNATURE IPQRM___ ↓
0	0 0 0 0	0 1 0 0	0 0 0 0	0 1 0 0	RS0
1	0 0 1 1	0 0 0 0	0 0 1 1	0 0 0 0	RS1
2	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	RS6
3	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	BR
4	0 0 0 0	0 0 0 0	0 0 1 0	0 0 0 0	RS8
5	0 0 0 1	0 0 0 0	0 0 0 0	0 0 0 0	RS9
6	0 0 1 1	0 1 0 1	0 0 1 1	0 1 0 1	RDT
7	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	DW
8	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	CHZ
9	0 0 0 1	0 1 0 1	0 0 1 1	0 1 0 1	AEH
10	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	LAE
11	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	EXN
12	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	LAM
13	0 0 1 1	0 1 0 1	0 0 1 1	0 1 0 1	RHZ
14	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	CAR
15	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	HW
16	0 0 1 1	0 1 0 1	0 0 1 1	0 1 0 1	M
17	0 0 1 1	0 1 0 1	0 0 0 0	0 0 0 0	AU4
18	0 0 1 1	0 0 0 0	0 0 1 1	0 0 0 0	AU5
19	1 1 0 0	1 0 1 0	1 1 0 0	1 0 1 0	IOP
20	0 0 0 0	0	0 0 0 0	0	AU6
21	0 0 1 0		0 0 1 0		AU7
22	0 0 0 1		0 0 1 0		GP1
23	0 0 0 0		0 0 0 0		PP
24	0 0 0 0	0 0 0 0	0 0 0 1	0 1 0 1	GP2
25	0 0 1 1	0 0 0 0	0 0 0 1	0 0 0 0	GP3
26	0 0 0 1	0 1 0 1	0 0 0 1	0 0 0 0	GP4
27	0 0 1 1	0 1 0 1	0 0 1 1	0 1 0 1	SW
28	0 0 0 1	0 1 0 1	0 0 0 1	0 1 0 1	EXM
29	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	LF
30	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	ORG
31	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	VCT

LEVEL 2 ROM CONTENTS (Sheet 9 of 16)

OP CODE 8X

X = Least Significant Hex of Op Code

OUTPUT BIT NUMBER ↓	DBNZ DBZ IBNZ IBZ F E D C	DBNZ DBZ IBNZ IBZ B A 9 8	BCG BCLE BCG BCLE 7 6 5 4	DSNE DSE ISNE ISE 3 2 1 0	OUTPUT SIGNATURE IPQRM___ ↓
0	1 1 1 1	0 0 0 0	1 1 0 0	0 0 0 0	RS0
1	0 0 0 0	1 1 1 1	0 0 1 1	1 1 1 1	RS1
2	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	RS6
3	0 0 0 0	0 0 0 0	0 0 0 0	1 1 1 1	BR
4	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	RS8
5	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	RS9
6	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	RDT
7	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	DW
8	0 0 0 0	0 0 0 0	0 0 0 0	0	CHZ
9	0 0 0 0	0 0 0 0	0 0 0 0		AEH
10	0 0 0 0	0 0 0 0	0 0 0 0		LAE
11	1 1 1 1	1 1 1 1	1 1 1 1		EXN
12	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	LAM
13	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	RHZ
14	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	CAR
15	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	HW
16	0 0 0 0	0 0 0 0	0 0 0 0	0	M
17	1 1 1 1	1 1 1 1	0 0 0 0		AU4
18	0 0 0 0	0 0 0 0	1 1 1 1		AU5
19	0 0 0 0	0 0 0 0	0 0 0 0		IOP
20	1 1 0 0	1 1 0 0	0	1 1 0 0	AU6
21	0 0 0 0	0 0 0 0		0 0 0 0	AU7
22	0 0 0 0	0 0 0 0		0 0 0 0	GP1
23	0 0 0 0	0 0 0 0		0 0 0 0	PP
24	1 1 1 1	1 1 1 1	0 0 0 0	0 0 0 0	GP2
25	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	GP3
26	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	GP4
27	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	SW
28	0	0	0	0	EXM
29					LF
30					ORG
31					VCT

LEVEL 2 ROM CONTENTS (Sheet 10 of 16)
OP CODE 9X

X = Least Significant Hex of Op Code

OUTPUT BIT NUMBER	MOD BAE BXEC	BLX BLB	PUL XEC BRC MCW	PSH INT BCC MCP	OUTPUT SIGNATURE IPQRM___
↓	F E D C	B A 9 8	7 6 5 4	3 2 1 0	↓
0	0 0 0 0	0 0 1 0	0 0 0 0	0 0 0 0	RS0
1	1 0 0 0	0 0 0 0	1 0 0 0	1 1 0 0	RS1
2	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	RS6
3	1 0 0 0	0 0 0 0	1 1 0 0	1 1 0 0	BR
4	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	RS8
5	1 0 0 0	0 0 0 0	1 0 0 0	1 1 0 0	RS9
6	0 0 0 0	0 0 1 1	1 0 0 0	0 1 0 0	RDT
7	1 0 0 0	0 0 0 0	1 0 0 0	1 0 0 0	DW
8	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	CHZ
9	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	AEH
10	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	LAE
11	0 0 1 1	0 0 1 1	0 0 1 0	0 0 1 0	EXN
12	0	0	0	0	LAM
13					RHZ
14					CAR
15					HW
16	0 0 0 0	0 0 0 0	0 0 0 1	0 0 0 1	M
17	1 0 0 0	0 0 1 1	0 0 0 0	0 0 0 0	AU4
18	1 0 0 0	0 0 0 0	1 0 0 0	0 0 0 0	AU5
19	0 1 0 0	1 1 0 0	0 0 0 0	0 0 0 0	IOP
20	1 0 0 0	0	1 0 0 0	1 1 0 0	AU6
21	1 0 0 0		1 0 0 0	1 0 0 0	AU7
22	0 0 0 0		0 0 0 0	0 0 0 0	GP1
23	0 0 0 0		1 0 0 0	1 0 0 0	PP
24	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	GP2
25	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	GP3
26	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	GP4
27	0 0 1 1	0 0 1 1	0 1 1 1	0 1 1 1	SW
28	0	0	0	0	EXM
29					LF
30					ORG
31					VCT

LEVEL 2 ROM CONTENTS (Sheet 11 of 16)
OP CODE AX

X = Least Significant Hex of Op Code

	NFH NFX	FHFD FXFD FHFL FXFL		FDFX FLFH FLFX	
OUTPUT BIT NUMBER ↓	F E D C	B A 9 8	7 6 5 4	3 2 1 0	OUTPUT SIGNATURE IPQRM___ ↓
0	0 0 0 0	0 0 0 0	0	0 0 0 0	RS0
1	0 0 1 1	1 1 1 1		0 1 1 1	RS1
2	0 0 0 0	0 0 0 0		0 0 0 0	RS6
3	0 0 1 1	1 1 1 1		0 1 1 1	BR
4	0 0 0 0	0 0 0 0	0	0 0 1 0	RS8
5	0 0 0 1	1 1 0 0		0 0 0 0	RS9
6	0 0 1 1	1 1 1 1		0 1 1 1	RDT
7	0 0 0 0	0 0 0 0		0 0 0 0	DW
8	0	0 0 0 0	0	0 0 0 0	CHZ
9		1 1 1 1		0 1 1 1	AEH
10		0 0 0 0		0 0 0 0	LAE
11		0 0 0 0		0 0 0 0	EXN
12	0 0 0 0	0 0 0 0	0	0 0 0 0	LAM
13	0 0 1 1	1 1 1 1		0 1 1 1	RHZ
14	0 0 0 0	0 0 0 0		0 0 0 0	CAR
15	0 0 1 0	1 1 1 1		0 1 1 1	HW
16	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	M
17	0 0 1 1	1 1 1 1	0 0 0 0	0 0 0 0	AU4
18	0 0 1 1	0 0 0 0	0 0 0 0	0 0 0 0	AU5
19	1 1 0 0	0 0 0 0	1 1 1 1	1 0 0 0	IOP
20	0	0	0	0	AU6
21					AU7
22					GP1
23					PP
24	0 0 0 0	0	0	0	GP2
25	0 0 0 0				GP3
26	0 0 0 0				GP4
27	0 0 0 1				SW
28	0	0	0	0	EXM
29					LF
30					ORG
31					VCT

LEVEL 2 ROM CONTENTS (Sheet 12 of 16)
OP CODE BX

X = Least Significant Hex of Op Code

OUTPUT BIT NUMBER ↓	F E D C	B A 9 8	7 6 5 4	VECT 3 2 1 0	OUTPUT SIGNATURE IPQRM___ ↓
0	0	0	1 0 1 0	0 0 0 0	RS0
1			0 1 1 0	0 0 0 0	RS1
2			1 0 1 0	0 0 0 0	RS6
3			0 1 1 0	0 0 0 1	BR
4	0	0	0	0	RS8
5					RS9
6					RDT
7					DW
8	0	0	0	0	CHZ
9					AEH
10					LAE
11					EXN
12	0	0	0	0	LAM
13					RHZ
14					CAR
15					HW
16	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	M
17	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	AU4
18	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	AU5
19	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 0	IOP
20	0	0	0	0	AU6
21					AU7
22					GP1
23					PP
24	0	0	0	0 0 0 0	GP2
25				0 0 0 0	GP3
26				0 0 0 0	GP4
27				0 0 0 1	SW
28	0	0	0	0 0 0 0	EXM
29				0 0 0 0	LF
30				0 0 0 0	ORG
31				0 0 0 1	VCT

LEVEL 2 ROM CONTENTS (Sheet 13 of 16)
OP CODE CX

X = Least Significant Hex of Op Code

OUTPUT BIT NUMBER ↓	SCD C SCH SC	CFD CF CH C	SLD RVS SLH SL	SAD SAH SA	OUTPUT SIGNATURE IPQRM___ ↓
	F E D C	B A 9 8	7 6 5 4	3 2 1 0	
0	0 1 0 0	0 0 0 0	0 0 0 0	0 0 0 0	RS0
1	1 0 1 1	1 1 1 1	1 1 1 1	1 0 1 1	RS1
2	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	RS6
3	0 1 0 0	1 1 1 1	0 0 0 0	0 0 0 0	BR
4	0 0 1 0	0 0 1 0	0 0 1 0	0 0 1 0	RS8
5	1 0 0 0	1 0 0 0	1 0 0 0	1 0 0 0	RS9
6	1 0 1 1	0 0 0 0	1 1 1 1	1 0 1 1	RDT
7	0 0 0 0	1 0 0 0	0 0 0 0	0 0 0 0	DW
8	0 1 0 0	1 1 1 1	0	0 0 0 0	CHZ
9	0 0 0 0	0 0 0 0		1 0 1 1	AEH
10	0 0 0 0	0 0 0 0		0 0 0 0	LAE
11	0 0 0 0	0 0 0 0		0 0 0 0	EXN
12	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	LAM
13	1 0 1 1	0 0 0 0	1 1 1 1	1 0 1 1	RHZ
14	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	CAR
15	0 0 0 0	0 0 1 0	0 0 0 0	0 0 0 0	HW
16	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	M
17	1 1 1 1	1 1 1 1	0 0 0 0	0 0 0 0	AU4
18	1 0 1 1	0 0 0 0	1 1 1 1	0 0 0 0	AU5
19	0 0 0 0	0 0 0 0	0 0 0 0	0 1 0 0	IOP
20	0	1 1 0 0	0 1 0 0	0	AU6
21		1 1 0 0	0 0 0 0		AU7
22		0 0 0 0	0 0 0 0		GP1
23		0 0 0 0	0 0 0 0		PP
24	0 1 0 0	1 1 1 1	0 0 0 0	0 0 0 0	GP2
25	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	GP3
26	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	GP4
27	1 1 1 1	0 1 0 1	1 1 1 1	1 0 1 1	SW
28	0	0	0	0	EXM
29					LF
30					ORG
31					VCT

LEVEL 2 ROM CONTENTS (Sheet 14 of 16)
OP CODE DX

X = Least Significant Hex of Op Code

OUTPUT BIT NUMBER ↓	CI F E D C	CIH CI B A 9 8	7 6 5 4	3 2 1 0	OUTPUT SIGNATURE IPQRM___ ↓
0	0 1 0 0	0 0 0 0	0	0	RS0
1	0 0 0 0	0 0 1 1			RS1
2	0 0 0 0	0 0 0 0			RS6
3	0 0 0 0	0 0 0 0			BR
4	0	0 0 1 0	0	0	RS8
5		0 0 0 0			RS9
6		0 0 0 0			RDT
7		0 0 0 0			DW
8	0 1 0 0	0 0 1 1	0	0	CHZ
9	0 0 0 0	0 0 0 0			AEH
10	0 0 0 0	0 0 0 0			LAE
11	0 0 0 0	0 0 0 0			EXN
12	0	0	0	0	LAM
13					RHZ
14					CAR
15					HW
16	0 1 0 0	0 0 1 1	0 0 0 0	0 0 0 0	M
17	0 1 0 0	0 0 1 1	0 0 0 0	0 0 0 0	AU4
18	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	AU5
19	1 0 1 1	1 1 0 0	1 1 1 1	1 1 1 1	IOP
20	0	0	0	0	AU6
21					AU7
22					GP1
23					PP
24	0 1 0 0	0 0 1 1	0	0	GP2
25	0 0 0 0	0 0 0 0			GP3
26	0 0 0 0	0 0 0 0			GP4
27	0 1 0 0	0 0 1 1			SW
28	0 1 0 0	0 0 0 1	0	0	EXM
29	0 0 0 0	0 0 0 0			LF
30	0 0 0 0	0 0 0 0			ORG
31	0 0 0 0	0 0 0 0			VCT

LEVEL 2 ROM CONTENTS (Sheet 15 of 16)
OP CODE EX

X = Least Significant Hex of Op Code

	EQCD EQC	XORD XOR	CORD COR ORD OR	CANDD CAND ANDD AND	
OUTPUT BIT NUMBER ↓	F E D C	B A 9 8	7 6 5 4	3 2 1 0	OUTPUT SIGNATURE IPQRM___ ↓
0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	RS0
1	0 0 1 1	0 0 1 1	1 1 1 1	1 1 1 1	RS1
2	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	RS6
3	0 0 1 1	0 0 1 1	1 1 1 1	1 1 1 1	BR
4	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	RS8
5	0 0 1 0	0 0 1 0	1 0 1 0	1 0 1 0	RS9
6	0 0 1 1	0 0 1 1	0 0 1 1	0 0 1 1	RDT
7	0 0 1 0	0 0 1 0	1 0 1 0	1 0 1 0	DW
8	0	0	1 1 0 0	1 1 0 0	CHZ
9			0 0 0 0	0 0 0 0	AEH
10			0 0 0 0	0 0 0 0	LAE
11			0 0 0 0	0 0 0 0	EXN
12	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	LAM
13	0 0 1 1	0 0 1 1	0 0 1 1	0 0 1 1	RHZ
14	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	CAR
15	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	HW
16	0 0 0 0	0 0 0 0	0 0 0 0	0	M
17	0 0 1 1	0 0 1 1	0 0 0 0		AU4
18	0 0 1 1	0 0 0 0	1 1 1 1		AU5
19	1 1 0 0	1 1 0 0	0 0 0 0		IOP
20	0 0 0 0	0 0 0 0	1 1 0 0	1 1 0 0	AU6
21	0 0 1 1	0 0 1 1	1 1 1 1	1 1 1 1	AU7
22	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	GP1
23	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	PP
24	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	GP2
25	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	GP3
26	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	GP4
27	0 0 0 1	0 0 0 1	0 1 0 1	0 1 0 1	SW
28	0	0	0	0	EXM
29					LF
30					ORG
31					VCT

LEVEL 2 ROM CONTENTS (Sheet 16 of 16)
OP CODE FX

X = Least Significant Hex of Op Code

	EQCI	XORI	CORI ORI	CANDI ANDI	
OUTPUT BIT NUMBER ↓	F E D C	B A 9 8	7 6 5 4	3 2 1 0	OUTPUT SIGNATURE IPQRM___ ↓
0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	RS0
1	0 0 0 1	0 0 0 1	0 1 0 1	0 1 0 1	RS1
2	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	RS6
3	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	BR
4	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	RS8
5	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	RS9
6	0 0 0 1	0 0 0 1	0 0 0 1	0 0 0 1	RDT
7	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	DW
8	0	0	0 1 0 0	0 1 0 0	CHZ
9			0 0 0 0	0 0 0 0	AEH
10			0 0 0 0	0 0 0 0	LAE
11			0 0 0 0	0 0 0 0	EXN
12	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	LAM
13	0 0 0 1	0 0 0 1	0 0 0 1	0 0 0 1	RHZ
14	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	CAR
15	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	HW
16	0 0 0 1	0 0 0 1	0 1 0 1	0 1 0 1	M
17	0 0 0 1	0 0 0 1	0 0 0 0	0 0 0 0	AU4
18	0 0 0 1	0 0 0 0	0 1 0 1	0 0 0 0	AU5
19	1 1 1 0	1 1 1 0	1 0 1 0	1 0 1 0	IOP
20	0 0 0 0	0 0 0 0	0 1 0 0	1 1 0 0	AU6
21	0 0 0 1	0 0 0 1	0 1 0 1	1 1 0 1	AU7
22	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	GP1
23	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	PP
24	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	GP2
25	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	GP3
26	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	GP4
27	0 0 0 1	0 0 0 1	0 1 0 1	0 1 0 1	SW
28	0	0	0	0	EXM
29					LF
30					ORG
31					VCT

APPENDIX B
LEVEL 3 ROM CONTENTS

LEVEL 3 ROM CONTENTS (Sheet 1 of 16)

OP CODE 0X

X = Least Significant Hex of Op Code

OUTPUT BIT NUMBER				NOP	OUTPUT SIGNATURE IRQRM___
↓	F E D C	B A 9 8	7 6 5 4	3 2 1 0	↓
0	0	0	0	0	ADW
1					AHW
2					BAE
3					BBX
4	0	0	0	0	BCC
5					BCG
6					BCL
7					BRC
8	0	0	0	0	BRH
9					DCO
10					BXC
11					IDN
12	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	IDZ
13	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	DC1
14	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	LLA
15	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	MCP
16	0	0	0	0	DC2
17					NSP
18					SHW
19					PSH
20	0	0	0	0	RSE
21					SDW
22					SGT
23					SKE
24	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	OCK
25	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	SPS
26	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	RGS
27	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	MDV
28	0	0	0	0	XCH
29					STR
30					CAR
31					VST

LEVEL 3 ROM CONTENTS (Sheet 2 of 16)
OP CODE 1X
X = Least Significant Hex of Op Code

OUTPUT BIT NUMBER	LFM LO LR L	LF XCH LL L	LD LLA LH L	LAE LAM	OUTPUT SIGNATURE IRQRM___
↓	F E D C	B A 9 8	7 6 5 4	3 2 1 0	↓
0	0 0 0 0	0 0 0 0	1 0 0 0	0	ADW
1	0 0 1 0	0 0 1 0	0 0 1 0		AHW
2	0 0 0 0	0 0 0 0	0 0 0 0		BAE
3	0 0 0 0	0 0 0 0	0 0 0 0		BBX
4	0	0	0	0	BCC
5					BCG
6					BCL
7					BRC
8	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	BRH
9	0 1 1 1	0 0 1 1	1 0 1 1	1 1 0 0	DCO
10	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	BXC
11	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	IDN
12	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	IDZ
13	1 1 1 1	0 1 1 1	1 1 1 1	1 1 0 0	DC1
14	0 0 0 0	0 0 0 0	0 1 0 0	0 0 0 0	LLA
15	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	MCP
16	0	0	0	0	DC2
17					NSP
18					SHW
19					PSH
20	0	0	1 0 0 0	0	RSE
21			0 0 0 0		SDW
22			0 0 0 0		SGT
23			0 0 0 0		SKE
24	0 1 1 1	0 1 1 1	1 0 1 1	1 1 0 0	OCK
25	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	SPS
26	0 0 0 0	0 1 0 0	0 0 0 0	0 0 0 0	RGS
27	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	MDV
28	0 0 0 0	0 1 0 0	0	0	XCH
29	0 0 0 0	0 1 0 0			STR
30	0 0 1 0	0 0 1 0			CAR
31	0 0 0 0	0 0 0 0			VST

LEVEL 3 ROM CONTENTS (Sheet 3 of 16)

OP CODE 2X

X = Least Significant Hex of Op Code

OUTPUT BIT NUMBER ↓	STFM STO STR ST	STF STOH STL ST	STD STH ST	STZD SPS STZH STZ	OUTPUT SIGNATURE IRQRM___ ↓
	F E D C	B A 9 8	7 6 5 4	3 2 1 0	
0	0 0 0 0	0 0 0 0	1 0 0 0	1 0 0 0	ADW
1	0 0 1 0	0 1 1 0	0 0 1 0	0 0 1 0	AHW
2	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	BAE
3	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	BBX
4	0	0	0	0	BCC
5					BCG
6					BCL
7					BRC
8	0	0	0	0	BRH
9					DCO
10					BXC
11					IDN
12	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	IDZ
13	1 1 1 1	0 1 1 1	1 0 1 1	1 1 1 1	DC1
14	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	LLA
15	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	MCP
16	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	DC2
17	0 1 1 1	0 1 1 1	1 0 1 1	1 0 1 1	NSP
18	0 0 1 0	0 1 1 0	0 0 1 0	0 0 1 0	SHW
19	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	PSH
20	0	0	1 0 0 0	1 0 0 0	RSE
21			1 0 0 0	1 0 0 0	SDW
22			0 0 0 0	0 0 0 0	SGT
23			0 0 0 0	0 0 0 0	SKE
24	0 1 1 1	0 1 1 1	1 0 1 1	1 1 1 1	OCK
25	0 0 0 0	0 0 0 0	0 0 0 0	0 1 0 0	SPS
26	0 1 1 1	0 1 1 1	1 0 1 1	1 1 1 1	RGS
27	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	MDV
28	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	XCH
29	0 1 1 1	0 1 1 1	1 0 1 1	1 1 1 1	STR
30	0 0 1 0	0 0 1 0	0 0 0 0	0 0 0 0	CAR
31	0 0 1 1	0 0 1 1	1 0 1 1	0 0 0 0	VST

LEVEL 3 ROM CONTENTS (Sheet 4 of 16)

OP CODE 3X

X = Least Significant Hex of Op Code

OUTPUT BIT NUMBER	LMD LMF LMH LM	LNMD LNMF LNMH LNM	STND STNF STNH STN	LND LNF LNH LN	OUTPUT SIGNATURE IRQRM___
↓	F E D C	B A 9 8	7 6 5 4	3 2 1 0	↓
0	1 0 0 0	1 0 0 0	1 0 0 0	1 0 0 0	ADW
1	0 0 1 0	0 0 1 0	0 0 1 0	0 0 1 0	AHW
2	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	BAE
3	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	BBX
4	0	0	0	0	BCC
5					BCG
6					BCL
7					BRC
8	0 0 0 0	0 0 0 0	0	0 0 0 0	BRH
9	1 1 1 1	1 1 1 1		1 1 1 1	DCO
10	0 0 0 0	0 0 0 0		0 0 0 0	BXC
11	0 0 0 0	0 0 0 0		0 0 0 0	IDN
12	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	IDZ
13	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	DC1
14	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	LLA
15	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	MCP
16	1 1 1 1	1 1 1 1	0 0 0 0	0 0 0 0	DC2
17	0 0 0 0	0 0 0 0	1 1 1 1	0 0 0 0	NSP
18	0 0 1 0	0 0 1 0	0 0 1 0	0 0 1 0	SHW
19	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	PSH
20	1 0 0 0	1 0 0 0	1 0 0 0	1 0 0 0	RSE
21	0 0 0 0	0 0 0 0	1 0 0 0	0 0 0 0	SDW
22	0 0 1 1	0 0 1 1	0 0 1 1	0 0 1 1	SGT
23	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	SKE
24	1 1 0 0	1 1 0 0	1 1 0 0	1 1 0 0	OCK
25	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	SPS
26	0 0 1 1	0 0 1 1	1 1 1 1	0 0 1 1	RGS
27	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	MDV
28	0	0	0 0 0 0	0	XCH
29			1 1 1 1		STR
30			0 0 0 0		CAR
31			0 0 0 0		VST

LEVEL 3 ROM CONTENTS (Sheet 5 of 16)

OP CODE 4X

X = Least Significant Hex of Op Code

OUTPUT BIT NUMBER ↓	SMFD SMF SMH SM F E D C	SFD SF SH S B A 9 8	AMFD AMF AMH AM 7 6 5 4	AFD AF AH A 3 2 1 0	OUTPUT SIGNATURE IRQRM___ ↓
0	1 0 0 0	1 0 0 0	1 0 0 0	1 0 0 0	ADW
1	0 0 1 0	0 0 1 0	0 0 1 0	0 0 1 0	AHW
2	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	BAE
3	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	BBX
4	0	0	0	0	BCC
5					BCG
6					BCL
7					BRC
8	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	BRH
9	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	DCO
10	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	BXC
11	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	IDN
12	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	IDZ
13	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	DC1
14	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	LLA
15	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	MCP
16	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	DC2
17	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	NSP
18	0 0 1 0	0 0 1 0	0 0 1 0	0 0 1 0	SHW
19	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	PSH
20	1 0 0 0	1 0 0 0	1 0 0 0	1 0 0 0	RSE
21	1 0 0 0	1 0 0 0	1 0 0 0	1 0 0 0	SDW
22	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	SGT
23	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	SKE
24	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	OCK
25	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	SPS
26	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	RGS
27	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	MDV
28	0	0	0	0	XCH
29					STR
30					CAR
31					VST

LEVEL 3 ROM CONTENTS (Sheet 6 of 16)

OP CODE 5X

X = Least Significant Hex of Op Code

OUTPUT BIT NUMBER	LI	SIH SI	LEA LIH LI	LEA AIH AI	OUTPUT SIGNATURE IRQRM
↓	F E D C	B A 9 8	7 6 5 4	3 2 1 0	↓
0	0	0 0 0 0	0 0 0 0	0 0 0 0	ADW
1		0 0 1 0	0 0 1 0	0 0 1 0	AHW
2		0 0 0 0	0 0 0 0	0 0 0 0	BAE
3		0 0 0 0	0 0 0 0	0 0 0 0	BBX
4	0	0	0	0	BCC
5					BCG
6					BCL
7					BRC
8	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	BRH
9	0 0 0 1	0 0 1 1	0 1 1 1	0 1 1 1	DCO
10	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	BXC
11	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	IDN
12	0	0	0	0	IDZ
13					DC1
14					LLA
15					MCP
16	0 0 0 1	0 0 0	0 1 1 1	0 1 0 0	DC2
17	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	NSP
18	0 0 0 0	0 0 1 0	0 0 0 0	0 0 1 0	SHW
19	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	PSH
20	0	0 0 0 0	0	0 0 0 0	RSE
21		0 0 0 0		0 0 0 0	SDW
22		0 0 1 1		0 0 1 1	SGT
23		0 0 0 0		0 0 0 0	SKE
24	0 0 0 1	0 0 0 0	0 1 1 1	0 1 0 0	OCK
25	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	SPS
26	0 0 0 0	0 0 1 1	0 0 0 0	0 0 1 1	RGS
27	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	MDV
28	0	0	0	0	XCH
29					STR
30					CAR
31					VST

LEVEL 3 ROM CONTENTS (Sheet 7 of 16)

OP CODE 6X

X = Least Significant Hex of Op Code

OUTPUT BIT NUMBER	MFD MF MH M	M M	DFD DF DH D	A A	OUTPUT SIGNATURE IRQRM___
↓	F E D C	B A 9 8	7 6 5 4	3 2 1 0	↓
0	1 0 0 0	0	1 0 0 0	0	ADW
1	0 0 1 0		0 0 1 0		AHW
2	0 0 0 0		0 0 0 0		BAE
3	0 0 0 0		0 0 0 0		BBX
4	0	0	0	0	BCC
5					BCG
6					BCL
7					BRC
8	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	BRH
9	1 1 1 1	0 1 0 1	1 1 1 1	0 1 0 1	DCO
10	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	BXC
11	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	IDN
12	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	IDZ
13	1 1 1 1	0 1 0 1	1 1 1 1	0 1 0 1	DC1
14	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	LLA
15	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	MCP
16	1 1 1 1	0 1 0 1	1 1 1 1	0 1 0 1	DC2
17	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	NSP
18	0 0 1 0	0 0 0 0	0 0 0 0	0 0 0 0	SHW
19	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	PSH
20	1 0 0 0	0 0 0 0	1 0 0 0	0 0 0 0	RSE
21	1 0 0 0	0 0 0 0	1 0 0 1	0 0 0 0	SDW
22	0 1 1 0	0 1 0 1	0 0 0 0	0 1 0 1	SGT
23	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	SKE
24	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	OCK
25	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	SPS
26	1 1 1 1	0 1 0 1	1 1 1 1	0 1 0 1	RGS
27	0 0 0 1	0 0 0 0	0 0 0 1	0 0 0 0	MDV
28	0	0	0	0	XCH
29					STR
30					CAR
31					VST

LEVEL 3 ROM CONTENTS (Sheet 8 of 16)
OP CODE 7X
X = Least Significant Hex of Op Code

OUTPUT BIT NUMBER	MIH MI	MI MI	DIH DI	AI AI	OUTPUT SIGNATURE IRQRM___
↓	F E D C	B A 9 8	7 6 5 4	3 2 1 0	↓
0	0 0 0 0	0	0 0 0 0	0	ADW
1	0 0 1 0		0 0 1 0		AHW
2	0 0 0 0		0 0 0 0		BAE
3	0 0 0 0		0 0 0 0		BBX
4	0	0	0	0	BCC
5					BCG
6					BCL
7					BRC
8	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	BRH
9	0 0 1 1	0 1 0 1	0 0 1 1	0 1 0 1	DCO
10	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	BXC
11	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	IDN
12	0	0	0	0	IDZ
13					DC1
14					LLA
15					MCP
16	0 0 0 0	0	0	0	DC2
17	0 0 0 0				NSP
18	0 0 1 0				SHW
19	0 0 0 0				PSH
20	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	RSE
21	0 0 0 0	0 0 0 0	0 0 0 1	0 0 0 0	SDW
22	0 0 1 0	0 1 0 1	0 0 0 0	0 1 0 1	SGT
23	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	SKE
24	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	OCK
25	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	SPS
26	0 0 1 1	0 1 0 1	0 0 1 1	0 1 0 1	RGS
27	0 0 0 1	0 0 0 0	0 0 0 1	0 0 0 0	MDV
28	0	0	0	0	XCH
29					STR
30					CAR
31					VST

LEVEL 3 ROM CONTENTS (Sheet 9 of 16)

OP CODE 8X

X = Least Significant Hex of Op Code

OUTPUT BIT NUMBER	DBNZ DBZ IBNZ IBZ	DBNZ DBZ IBNZ IBZ	BCG BCLE BCG BCLE	DSNE DSE ISNE ISE	OUTPUT SIGNATURE IRQRM___
	F E D C	B A 9 8	7 6 5 4	3 2 1 0	
0	0	0	0	0	ADW
1					AHW
2					BAE
3					BBX
4	0	0	0 0 0 0	0	BCC
5			1 0 1 0		BCG
6			0 1 0 1		BCL
7			0 0 0 0		BRC
8	1 1 1 1	1 1 1 1	1 1 1 1	0	BRH
9	0 0 0 0	0 0 0 0	0 0 0 0		DCO
10	0 0 0 0	0 0 0 0	0 0 0 0		BXC
11	1 0 1 0	1 0 1 0	0 0 0 0		IDN
12	0 1 0 1	0 1 0 1	0 0 0 0	0 0 0 0	IDZ
13	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	DC1
14	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	LLA
15	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	MCP
16	0	1 1 1 1	0	0	DC2
17		0 0 0 0			NSP
18		0 0 0 0			SHW
19		0 0 0 0			PSH
20	0 0 0 0	0 0 0 0	0	0 0 0 0	RSE
21	0 0 0 0	0 0 0 0		0 0 0 0	SDW
22	1 1 1 1	1 1 1 1		0 0 0 0	SGT
23	0 0 0 0	0 0 0 0		0 1 0 1	SKE
24	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	OCK
25	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	SPS
26	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	RGS
27	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	MDV
28	0	0	0	0	XCH
29					STR
30					CAR
31					VST

LEVEL 3 ROM CONTENTS (Sheet 10 of 16)

OP CODE 9X

X = Least Significant Hex of Op Code

OUTPUT BIT NUMBER	MOD BAE BXEC	BLX BLB	PUL XEC BRC MCW	PSH INT BCC MCP	OUTPUT SIGNATURE IRQRM___
↓	F E D C	B A 9 8	7 6 5 4	3 2 1 0	↓
0	1 0 0 0	0 0 0 0	1 0 0 0	1 0 0 0	ADW
1	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	AHW
2	0 0 1 0	0 0 0 0	0 0 0 0	0 0 0 0	BAE
3	0 0 0 0	0 0 1 1	0 0 0 0	0 0 0 0	BBX
4	0	0	0 0 0 0	0 0 1 0	BCC
5			0 0 0 0	0 0 0 0	BCG
6			0 0 0 0	0 0 0 0	BCL
7			0 0 1 0	0 0 0 0	BRC
8	0 0 1 1	0 0 1 1	0 0 1 0	0 0 1 0	BRH
9	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	DCO
10	0 0 0 1	0 0 0 0	0 0 0 0	0 0 0 0	BXC
11	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	IDN
12	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	IDZ
13	0 0 1 1	0 0 1 1	0 1 1 0	0 1 1 0	DC1
14	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	LLA
15	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 1	MCP
16	1 0 0 0	0	1 0 0 0	1 0 0 0	DC2
17	0 0 0 0		0 0 0 0	0 0 0 0	NSP
18	0 0 0 0		0 0 0 0	0 0 0 0	SHW
19	0 0 0 0		0 0 0 0	1 0 0 0	PSH
20	0	0	0	0 1 0 0	RSE
21				0 0 0 0	SDW
22				0 0 0 0	SGT
23				0 0 0 0	SKE
24	0 0 0 0	0 0 0 0	0 0 0 1	0 1 0 1	OCK
25	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	SPS
26	1 0 0 0	0 0 1 1	0 0 0 0	1 0 0 0	RGS
27	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	MDV
28	0 0 0 0	0	0 0 0 0	0 0 0 0	XCH
29	1 0 0 0		1 0 0 0	1 0 0 0	STR
30	0 0 0 0		0 0 0 0	0 0 0 0	CAR
31	0 0 0 0		0 0 0 0	0 0 0 0	VST

LEVEL 3 ROM CONTENTS (Sheet 11 of 16)

OP CODE AX

X = Least Significant Hex of Op Code

OUTPUT BIT NUMBER ↓	NFH NFX F E D C	FHFD FXFD FHFL FXFL B A 9 8	7 6 5 4	FDFX FLFH FLFX 3 2 1 0	OUTPUT SIGNATURE IRQRM___ ↓
0	0 0 0 0	0 0 0 0	0	0 0 0 0	ADW
1	0 0 1 0	1 1 1 1		0 1 1 1	AHW
2	0 0 0 0	0 0 0 0		0 0 0 0	BAE
3	0 0 0 0	0 0 0 0		0 0 0 0	BBX
4	0	0	0	0	BCC
5					BCG
6					BCL
7					BRC
8	0 0 0 0	0 0 0 0	0	0 0 0 0	BRH
9	0 0 1 1	1 1 1 1		0 1 1 1	DCO
10	0 0 0 0	0 0 0 0		0 0 0 0	BXC
11	0 0 0 0	0 0 0 0		0 0 0 0	IDN
12	0 0 0 0	0 0 0 0	0	0 0 0 0	IDZ
13	0 0 1 1	1 1 1 1		0 1 1 1	DC1
14	0 0 0 0	0 0 0 0		0 0 0 0	LLA
15	0 0 0 0	0 0 0 0		0 0 0 0	MCP
16	0	1 1 1 1	0	0 1 1 1	DC2
17		0 0 0 0		0 0 0 0	NSP
18		1 0 1 0		0 0 0 0	SHW
19		0 0 0 0		0 0 0 0	PSH
20	0 0 0 1	1 1 0 0	0	0 1 0 0	RSE
21	0 0 0 0	0 0 0 0		0 1 0 0	SDW
22	0 0 0 0	0 0 0 0		0 0 0 0	SGT
23	0 0 0 0	0 0 0 0		0 0 0 0	SKE
24	0	0 0 0 0	0	0 0 0 0	OCK
25		0 0 0 0		0 0 0 0	SPS
26		1 1 1 1		0 1 1 1	RGS
27		0 0 0 0		0 0 0 0	MDV
28	0	0	0	0	XCH
29					STR
30					CAR
31					VST

LEVEL 3 ROM CONTENTS (Sheet 12 of 16)
OP CODE BX
X = Least Significant Hex of Op Code

OUTPUT BIT NUMBER ↓	F E D C	B A 9 8	7 6 5 4	VECT 3 2 1 0	OUTPUT SIGNATURE IRQRM___ ↓
0	0	0	1 0 1 0	0	ADW
1			0 1 1 0		AHW
2			1 0 1 0		BAE
3			0 1 1 0		BBX
4	0	0	0	0	BCC
5					BCG
6					BCL
7					BRC
8	0	0	0	0	BRH
9					DCO
10					BXC
11					IDN
12	0	0	0	0	IDZ
13					DC1
14					LLA
15					MCP
16	0	0	0	0 0 0 1	DC2
17				0 0 0 0	NSP
18				0 0 0 0	SHW
19				0 0 0 0	PSH
20	0	0	0	0	RSE
21					SDW
22					SGT
23					SKE
24	0	0	0	0	OCK
25					SPS
26					RGS
27					MDV
28	0	0	0	0	XCH
29					STR
30					CAR
31					VST

LEVEL 3 ROM CONTENTS (Sheet 13 of 16)

OP CODE CX

X = Least Significant Hex of Op Code

OUTPUT BIT NUMBER	SCD C SCH SC	CFD CF CH C	SLD RVS SLH SL	SAD SAH SA	OUTPUT SIGNATURE IRQRM___
	F E D C	B A 9 8	7 6 5 4	3 2 1 0	
0	0	1 0 0 0	0	0	ADW
1		0 0 1 0			AHW
2		0 0 0 0			BAE
3		0 0 0 0			BBX
4	0	0	0	0	BCC
5					BCG
6					BCL
7					BRC
8	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	BRH
9	1 1 1 1	1 1 1 1	1 1 1 1	1 0 1 1	DCO
10	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	BXC
11	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	IDN
12	0 0 0 0	0 0 0 0	0	0	IDZ
13	0 1 0 0	1 1 1 1			DC1
14	0 0 0 0	0 0 0 0			LLA
15	0 0 0 0	0 0 0 0			MCP
16	0 1 0 0	1 1 1 1	0 0 0 0	0 0 0 0	DC2
17	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	NSP
18	0 0 1 0	0 0 1 0	0 0 1 0	0 0 1 0	SHW
19	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	PSH
20	1 0 0 0	1 0 0 0	1 0 0 0	1 0 0 0	RSE
21	1 0 0 0	1 0 0 0	1 0 0 0	1 0 0 0	SDW
22	0 1 0 0	1 1 1 1	0 0 0 0	0 0 0 0	SGT
23	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	SKE
24	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	OCK
25	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	SPS
26	1 1 1 1	1 1 1 1	1 1 1 1	1 0 1 1	RGS
27	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	MDV
28	0	0	0	0	XCH
29					STR
30					CAR
31					VST

LEVEL 3 ROM CONTENTS (Sheet 14 of 16)

OP CODE DX

X = Least Significant Hex of Op Code

OUTPUT BIT NUMBER	CI	CIH CI			OUTPUT SIGNATURE IRQRM___
↓	F E D C	B A 9 8	7 6 5 4	3 2 1 0	↓
0	0	0 0 0 0	0	0	ADW
1		0 0 1 0			AHW
2		0 0 0 0			BAE
3		0 0 0 0			BBX
4	0	0	0	0	BCC
5					BCG
6					BCL
7					BRC
8	0 0 0 0	0 0 0 0	0	0	BRH
9	0 1 0 0	0 0 1 1			DCO
10	0 0 0 0	0 0 0 0			BXC
11	0 0 0 0	0 0 0 0			IDN
12	0	0	0	0	IDZ
13					DC1
14					LLA
15					MCP
16	0	0 0 0 0	0	0	DC2
17		0 0 0 0			NSP
18		0 0 1 0			SHW
19		0 0 0 0			PSH
20	0 0 0 0	0 0 0 0	0	0	RSE
21	0 0 0 0	0 0 0 0			SDW
22	0 1 0 0	0 0 1 1			SGT
23	0 0 0 0	0 0 0 0			SKE
24	0 0 0 0	0 0 0 0	0	0	OCK
25	0 0 0 0	0 0 0 0			SPS
26	0 1 0 0	0 0 1 1			RGS
27	0 0 0 0	0 0 0 0			MDV
28	0	0	0	0	XCH
29					STR
30					CAR
31					VST

LEVEL 3 ROM CONTENTS (Sheet 15 of 16)

OP CODE EX

X = Least Significant Hex of Op Code

OUTPUT BIT NUMBER ↓	EQCD EQC	XORD XOR	CORD COR ORD OR	CANDD CAND ANDD AND	OUTPUT SIGNATURE IRQRM ↓
	F E D C	B A 9 8	7 6 5 4	3 2 1 0	
0	0 0 1 0	0 0 1 0	1 0 1 0	1 0 1 0	ADW
1	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	AHW
2	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	BAE
3	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	BBX
4	0	0	0	0	BCC
5					BCG
6					BCL
7					BRC
8	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	BRH
9	0 0 1 1	0 0 1 1	1 1 1 1	1 1 1 1	DCO
10	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	BXC
11	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	IDN
12	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	IDZ
13	0 0 1 1	0 0 1 1	1 1 1 1	1 1 1 1	DC1
14	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	LLA
15	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	MCP
16	0 0 1 1	0 0 1 1	1 1 1 1	1 1 1 1	DC2
17	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	NSP
18	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	SHW
19	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	PSH
20	0 0 1 0	0 0 1 0	1 0 1 0	1 0 1 0	RSE
21	0 0 1 0	0 0 1 0	1 0 1 0	1 0 1 0	SDW
22	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	SGT
23	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	SKE
24	0 0 1 1	0 0 1 1	1 1 1 1	1 1 1 1	OCK
25	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	SPS
26	0 0 1 1	0 0 1 1	1 1 1 1	1 1 1 1	RGS
27	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	MDV
28	0	0	0	0	XCH
29					STR
30					CAR
31					VST

LEVEL 3 ROM CONTENTS (Sheet 16 of 16)

OP CODE FX

X = Least Significant Hex of Op Code

OUTPUT BIT NUMBER	EQCI	XORI	CORI ORI	CANDI ANDI	OUTPUT SIGNATURE IRQRM___
↓	F E D C	B A 9 8	7 6 5 4	3 2 1 0	↓
0	0	0	0	0	ADW
1					AHW
2					BAE
3					BBX
4	0	0	0	0	BCC
5					BCG
6					BCL
7					BRC
8	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	BRH
9	0 0 0 1	0 0 0 1	0 1 0 1	0 1 0 1	DCO
10	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	BXC
11	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	IDN
12	0	0	0	0	IDZ
13					DC1
14					LLA
15					MCP
16	0	0	0	0	DC2
17					NSP
18					SHW
19					PSH
20	0	0	0	0	RSE
21					SDW
22					SGT
23					SKE
24	0 0 0 1	0 0 0 1	0 1 0 1	0 1 0 1	OCK
25	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	SPS
26	0 0 0 1	0 0 0 1	0 1 0 1	0 1 0 1	RGS
27	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	MDV
28	0	0	0	0	XCH
29					STR
30					CAR
31					VST

APPENDIX C
AU DETAILS MAP

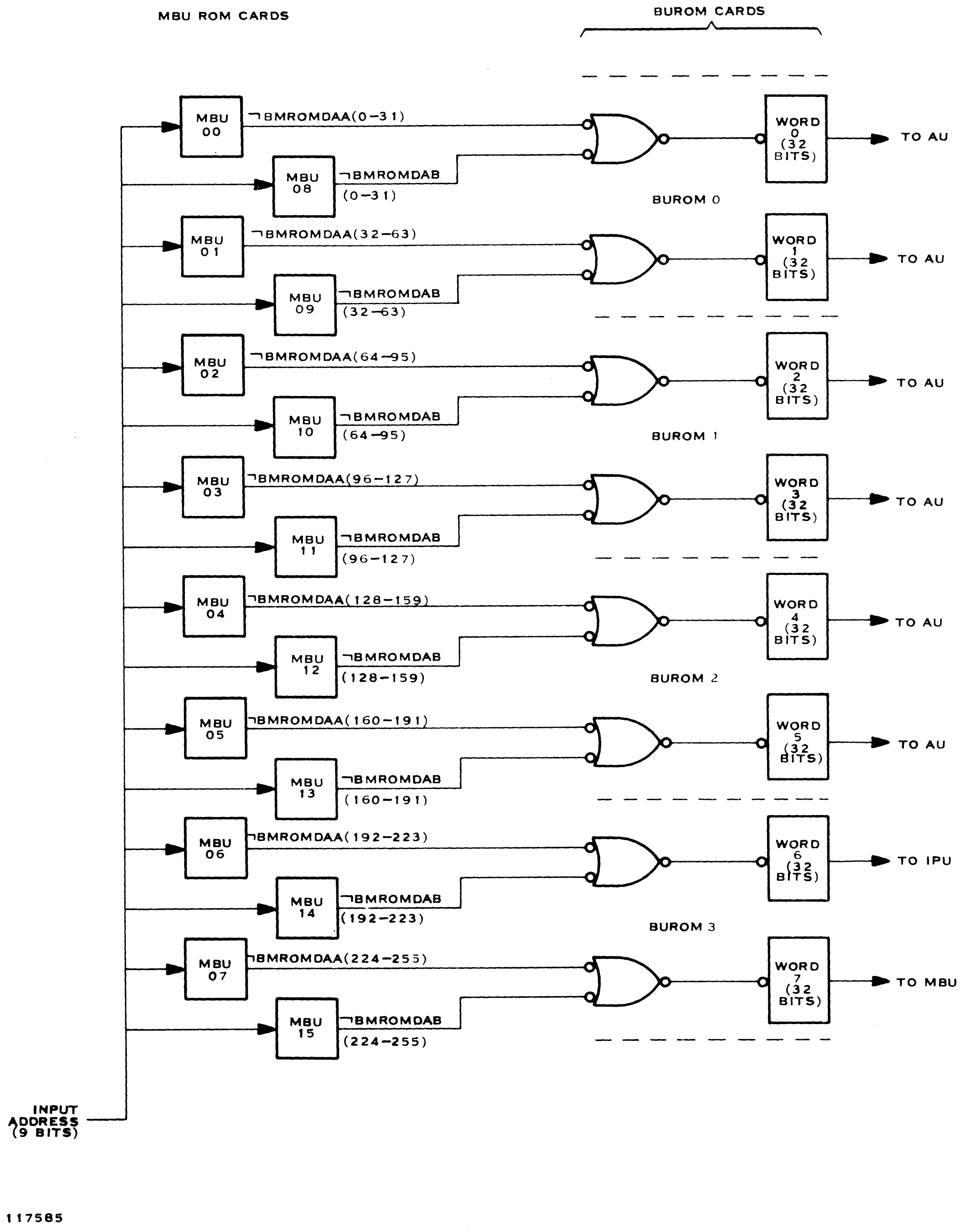


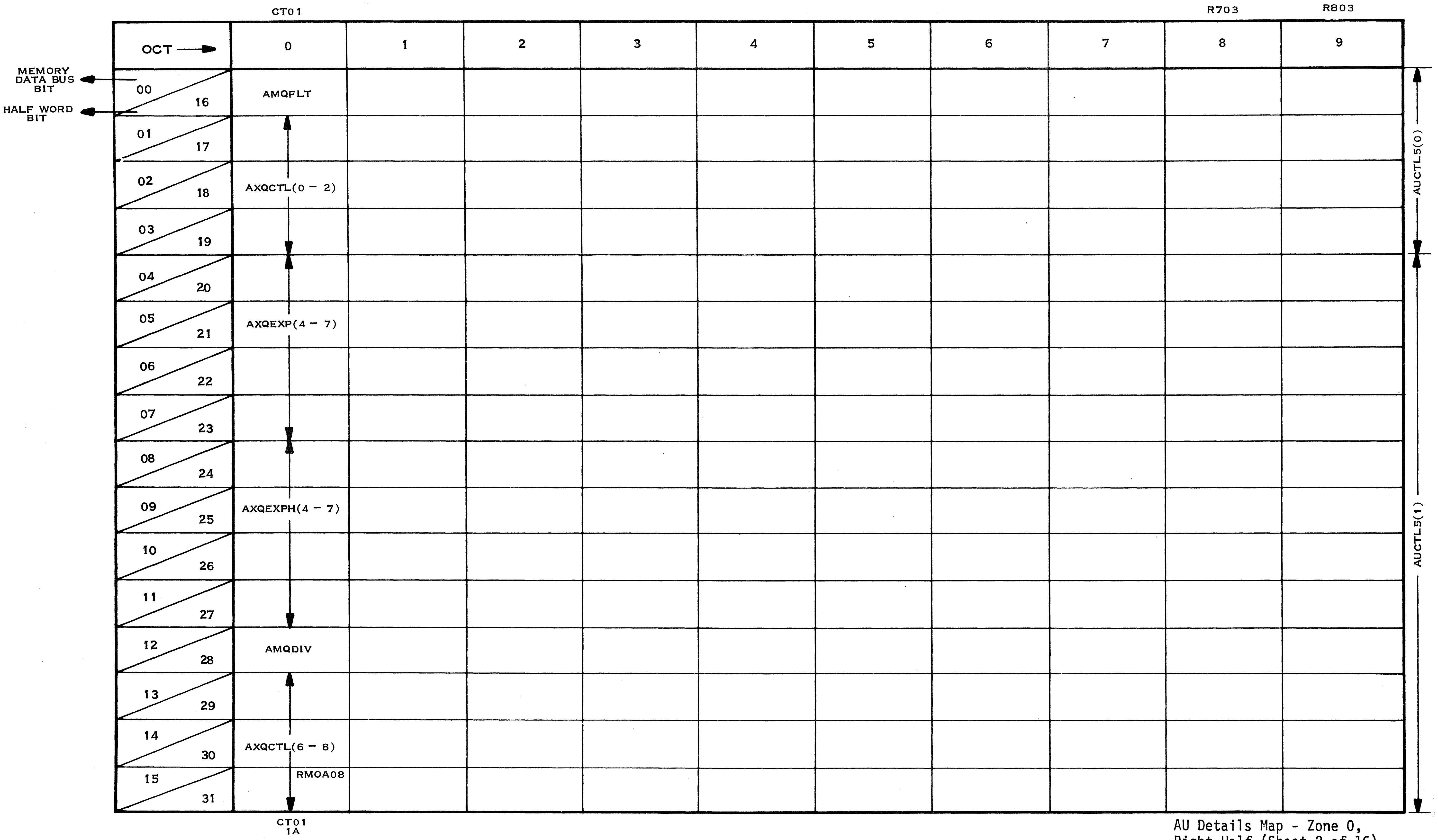
Figure C-1. MBU ROM Data Distribution

MEMORY DATA BUS BIT

HALF WORD BIT

OCT →		0	1	2	3	4	5	6	7	8	9	
00	00	ALQSHENC(0 − 2) 5	ALQSHENC(3 − 4) 5	ALQFLUFN 5	AXQUFFLT 5	☒	☒	☒	☒	☒	☒	AUCTL2
01	01	5	5	☒	☒	☒	☒	☒	☒	☒	☒	
02	02	5	ALQADDCR(8) 4	ALQFLOFN 5	☒	☒	☒	☒	☒	☒	☒	
03	03	A^QPPOFX 4	AOQSGNCV(1) 5	AOQGINF(0 − 4) 7	AOQGIND(0 − 4) 7	ASQACCEX 7	AOQOFFX 8	AOQRG 8	AOQPPZER 7	☒	☒	AUCTL3
04	04	ASQIDXCT(0 − 3) 7	AOQSGNCV(2) 5	7	7	AQQRCDT 7	AOQOFFL 8	AOQRE 8	ASQESL 7	☒	☒	
05	05	7	AOQCVSGN 5	7	7	ASQFXOF 7	AOQUFFL 8	AOQCL 8	ASQRFODD 7	☒	☒	
06	06	7	AOQLSHOF 5	7	7	AOQDVCHK 8	AOQGZER 8	AOQGC 8	AOQPPINL 7	☒	☒	
07	07	7	AOQFLFXOF 5	7	7	ASQITMCT 8	AOQRL 8	AOQCE 8	AOQXOFSF 7	☒	☒	
08	08	AXQEXP(0 − 3)										AUCTL5(0)
09	09											
10	10											
11	11											
12	12	AXQEXPH(0 − 3)										
13	13											
14	14											
15	15											

AU Details Map - Zone 0,
Left Half (Sheet 1 of 16)



	CT01								R703	R803	
OCT →	0	1	2	3	4	5	6	7	8	9	
00 / 16	AMQFLT										AUCTL5(0)
01 / 17	AXQCTL(0 − 2)										
02 / 18											
03 / 19											
04 / 20	AXQEXP(4 − 7)										AUCTL5(1)
05 / 21											
06 / 22											
07 / 23											
08 / 24	AXQEXPH(4 − 7)										
09 / 25											
10 / 26											
11 / 27											
12 / 28	AMQDIV										
13 / 29	AXQCTL(6 − 8)										
14 / 30											
15 / 31	RMOA08										
	CT01 1A										

AU Details Map - Zone 0,
Right Half (Sheet 2 of 16)

MEMORY DATA BUS BIT

HALF WORD BIT

	CT02								R704	R804	
OCT →	0	1	2	3	4	5	6	7	8	9	
00 / 32											UNUSED
01 / 33											
02 / 34	-	U	N	U	S	E	D	-			
03 / 35											
04 / 36		NOT CONNECTED TO MEMORY									
05 / 37											
06 / 38											
07 / 39											
08 / 40	AXQCTL(3 – 5)										AUCTL5(0)
09 / 41											
10 / 42											
11 / 43	AXQSGN(0)										
12 / 44	AHQSGNH(0)										
13 / 45											
14 / 46											
15 / 47											
	1A										

AU Details Map - Zone 1, Left Half (Sheet 3 of 16)

MEMORY DATA BUS BIT

HALF WORD BIT

	CT03								R705	R805	
OCT →	0	1	2	3	4	5	6	7	8	9	
00 48											AUCTL5(0)
01 49											
02 50											
03 51											
04 52	AXQCTL(9 – 11)										AUCTL5(1)
05 53											
06 54											
07 55											
08 56											
09 57											
10 58											
11 59											
12 60											
13 61											
14 62											
15 63											
	1A										

AU Details Map - Zone 1,
Right Half (Sheet 4 of 16)

MEMORY DATA BUS BIT
HALF WORD BIT

	RN00	AB00	CD00	S000	L000	RS00	NS00	AA00	R706	R806	
OCT →	0	1	2	3	4	5	6	7	8	9	
00 / 64	AMQPKPK RM06(15)	↑	↑	↑	↑	↑	↑	↑	☒	☒	AUADD(1)
01 / 65	AMQORD RM09(00)								☒	☒	
02 / 66	AMQBITR RM08(00)								☒	☒	
03 / 67	☒								☒	☒	
04 / 68	AMQHL CD05(10)								☒	☒	AUADD(1)
05 / 69	AMQDL CD05(11)								☒	☒	
06 / 70	AMQFLSBM RM05(13)								☒	☒	
07 / 71	☒								☒	☒	
08 / 72	1	AIQAB(00 − 15) 1	AIQCD(00 − 15) × 1	AEQSOR(00 − 15) × 2	AEQLOR(00 − 15) × 2	ARQSH(00 − 15) × 3	ARQNS(00 − 15) × 3	AAQADD(00 − 15) 4	☒	☒	AUADD(2)
09 / 73	AMQFLFXC RM05(10)								☒	☒	
10 / 74	AMQKG8EN RM08(07)								☒	☒	
11 / 75	☒								☒	☒	
12 / 76									☒	☒	AUADD(3)
13 / 77	AMQFLCTL RM08(03)								☒	☒	
14 / 78	AMQSTNFX RM09(10)								☒	☒	
15 / 79	☒	↓	↓	↓	↓	↓	↓	↓	☒	☒	
	22	23	1C	1D	1E	1F	20	21			

AU Details Map - Zone 2, Left Half (Sheet 5 of 16)

MEMORY DATA BUS BIT

HALF WORD BIT

	RN01	AB01	CD01	S001	LO01	RS01	NS01	AA01	R707	R807	
OCT →	0	1	2	3	4	5	6	7	8	9	
00 80	0								☒	☒	AUADD(4)
01 81	AMQFXFLC RM05(15) 0								☒	☒	
02 82	AMQFL RM05(03) 0								☒	☒	
03 83	☒								☒	☒	
04 84	0								☒	☒	AUADD(5)
05 85	AMQODINL RM08(15) 0								☒	☒	
06 86	AMQFLSB RM08(04) 0								☒	☒	
07 87	☒								☒	☒	
08 88	AMQSRCH RM09(09) 0	AIQAB(16 – 31)	AIQCD(16 – 31)	AEQSOR(16 – 31)	AEQLOR(16 – 31)	ARQSH(16 – 31)	ARQNS(16 – 31)	AAQADD(16 – 31)	☒	☒	AUADD(6)
09 89	AMQSRCK2 RM07(06) 0								☒	☒	
10 90	AMQFLADM RM05(05) 0								☒	☒	
11 91	1 ☒								☒	☒	
12 92									☒	☒	AUADD(7)
13 93	AMQMYCK2 RM08(13) 0								☒	☒	
14 94	AMQMYCK4 RM08(14) 0								☒	☒	
15 95	☒								☒	☒	
	22	23	1C	1D	1E	1F	20	21	1A	1B	

AU Details Map - Zone 2, Right Half (Sheet 6 of 16)

MEMORY DATA BUS BIT → 00

HALF WORD BIT → 96

	RN02	AB02	CD02	CO02	LO02	RS02	NS02	AA02	R708	R808	
OCT →	0	1	2	3	4	5	6	7	8	9	
00 96	AMQSHIF RM07(04) 0								☒	☒	AUADD(8)
01 97	AMQFIRS RM05(02) 0								☒	☒	
02 98	AMQFLAD RM05(04) 0								☒	☒	
03 99	☒								☒	☒	
04 100	AMQCIRC RM08(01) 0								☒	☒	AUADD(9)
05 101	AMQARSH RM04(10) 0								☒	☒	
06 102	0								☒	☒	
07 103	☒								☒	☒	
08 104	AMQVDP RM03(03) 0	AIQAB(32 – 47) 1	AIQCD(32 – 47) 1	AEQSOR(32 – 47) 2	AEQLOR(32 – 47) 2	ARQSH(32 – 47) 3	ARQNS(32 – 47) 3	AAQADD(32 – 47) 4	☒	☒	AUADD(10)
09 105									☒	☒	
10 106	AMQPSOPL RM0(01)								☒	☒	
11 107	☒								☒	☒	
12 108	AMQLARFX RM08(09)								☒	☒	AUADD(11)
13 109	AMQLARFL RM08(08)								☒	☒	
14 110	AMQSARFX RM09(05)								☒	☒	
15 111	☒								☒	☒	
	22	23	1C	1D	1E	1F	20	21	1A	1B	

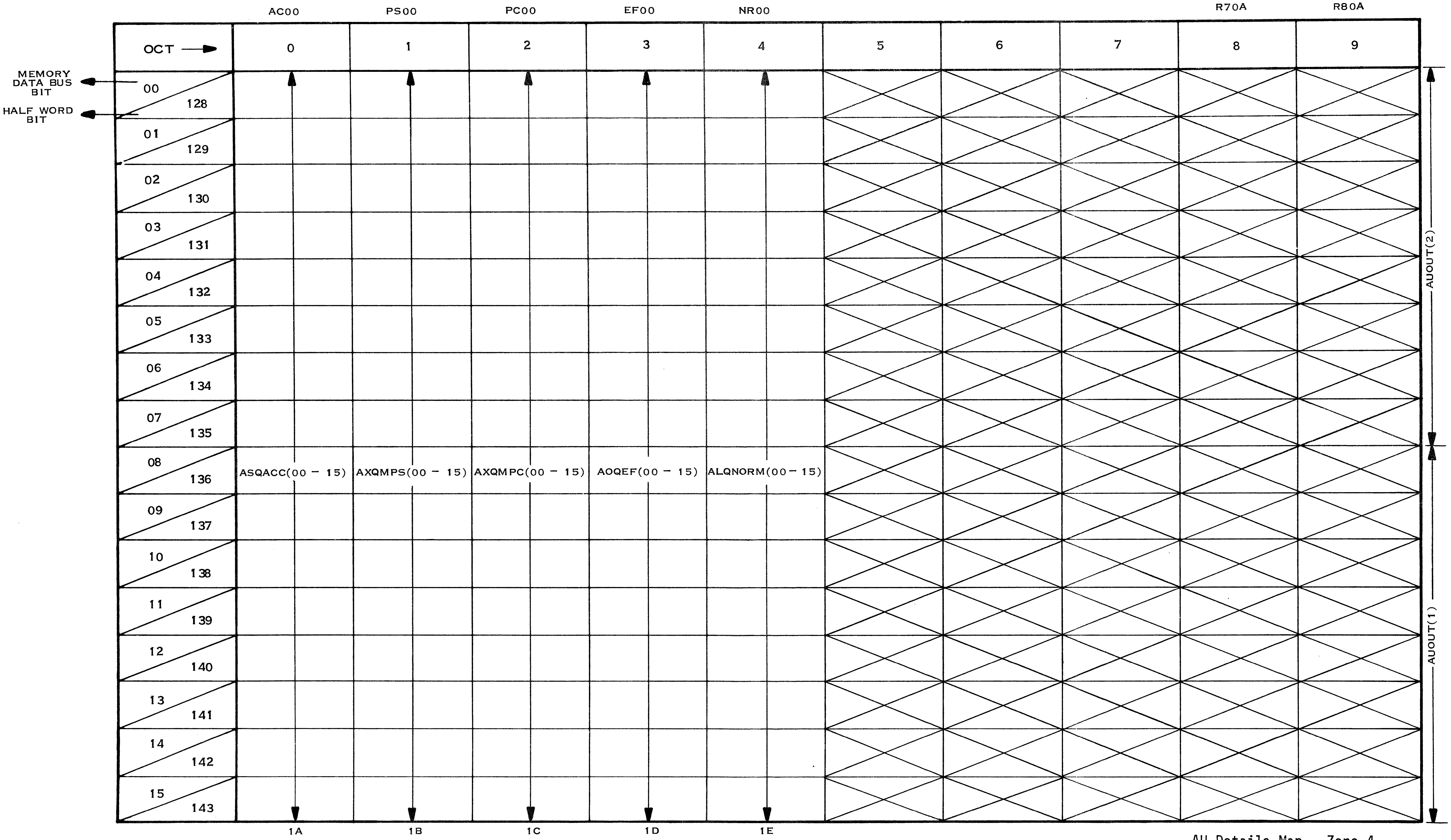
AU Details Map - Zone 3, Left Half (Sheet 7 of 16)

MEMORY DATA BUS BIT ←

HALF WORD BIT ←

	RN03	AB03	CD03	SO03	LO03	RS03	NS03	AA03	R709	R809	
OCT →	0	1	2	3	4	5	6	7	8	9	
00 112	AMQSARFL RM09(04) 1										AUADD(12)
01 113	AMQLMGFL RM08(11) 0										
02 114	AMQLMGFL RM08(10) 0										
03 115											
04 116	AMQSMGFX RM09(08) 0										AUADD(13)
05 117	AMQSMGFL RM09(07) 0										
06 118	AMQ1 RM09(11) 0										
07 119											
08 220	AMQ2 RM09(12) 0	ATQAB(48 – 63) 1	AIQCD(48 – 63) 1	AEQSOR(48 – 63) 2	AEQLOR(48 – 63) 2	ARQSH(48 – 63) 3	ARQNS(48 – 63) 3	AAQADD(48 – 63) 4			AUADD(14)
09 221	AMQ3 RM09(13) 0										
10 222	AMQ4 RM09(14) 0										
11 223											
12 224	AMQSCHXM RM09(06) 0										AUADD(15)
13 225	AMQFXAR RM08(05) 0										
14 226	AMQFLAR RM05(06) 0										
15 227											
	22	23	1C	1D	1E	1F	20	21	1A	1B	

AU Details Map - Zone 3, Right Half (Sheet 8 of 16)



AU Details Map - Zone 4,
Left Half (Sheet 9 of 16)

MEMORY DATA BUS BIT

HALF WORD BIT

	AC01	PS01	PC01	EO01	NR01				R70B	R70B	
OCT →	0	1	2	3	4	5	6	7	8	9	
00 144				16							AUOUT(2)
01 145				17							
02 146				18							
03 147				19							
04 148				20							
05 149				21							
06 150				22							
07 151				23							
08 152	ASQACC(16 − 31)	AXQMPS(16 − 31)	AXQMPC(16 − 31)	AOQEF(16 − 31) 24	ALQNORM(16 − 31)						AUOUT(3)
09 153				25							
10 154				26							
11 155				27							
12 156				28							
13 157				29							
14 158				30							
15 159				31							
	1A	1B	1C	1D	1E						

AU Details Map - Zone 4, Right Half (Sheet 10 of 16)

MEMORY DATA BUS BIT →

HALF WORD BIT →

	AC02	PS02	PL02	EF02	NR02				R70	R80	
OCT →	0	1	2	3	4	5	6	7	8	9	
00 160	ASQACC(32 − 47)	AXQMPS(32 − 47)	AXQMPC(32 − 47)	AOQEF(32 − 47)	ALQNORM(32 − 47)						AUOUT(4)
01 161											
02 162											
03 163											
04 164											
05 165											
06 166											
07 167											
08 168											AUOUT(5)
09 169											
10 170											
11 171											
12 172											
13 173											
14 174											
15 175											
	1A	1B	1C	1D	1E						

AU Details Map - Zone 5, Left Half (Sheet 11 of 16)

MEMORY DATA BUS BIT

HALF WORD BIT

	AUOUT					AUCTL4					
OCT →	0	1	2	3	4	5	6	7	8	9	
00 176							AMQLNTCK BLANK AT MB			☒	AUOUT(6)
01 177										☒	
02 178								ALQGDGT(0 − 3)	ALQCFXSF(0 − 4)	☒	
03 179										☒	
04 180										☒	
05 181									☒	☒	
06 182									☒	☒	
07 183									☒	☒	
08 184	ASQACC(48 − 63)	AXQMPS(48 − 63)	AXQMPC(48 − 63)	AOQEF(48 − 63)	ALQNORM(48 − 63)	BLANK AT MB ANQOGT(1 − 15)	BLANK AT MB AMQNGT(1 − 15)	BLANK AT MB AQQAE(0 − 8)	☒	☒	AUOUT(7)
09 185									☒	☒	
10 186									☒	☒	
11 187									☒	☒	
12 188									☒	☒	
13 189								AAQAPXOF	☒	☒	
14 190								AOQPPZER	☒	☒	
15 191								AOQPPINL	☒	☒	

AU Details Map - Zone 5, Right Half (Sheet 12 of 16)

MEMORY DATA BUS BIT

HALF WORD BIT

	XA00	XB00	XC00	XD00	VM00	VL00	NM00	NL00	DM00	DL00	
OCT →	0	1	2	3	4	5	6	7	8	9	
00 192											AUMULT(0)
01 193											
02 194											
03 195											
04 196											AUMULT(1)
05 197											
06 198											
07 199											
08 200	AXQABM(00 − 15) 1	AXQABL(32 − 47) 1	AXQCDM(00 − 15) 1	AXQCDL(32 − 47) 1	AXQDVRM(00− 15) 9	AXQDVRL(32− 47) 9	AXQDVNDM(00−15) 9	AXQDVNDL(32−47) 9	AXQMODFM(00−15) 9	AXQMODFL(32−47) 9	AUMULT(2)
09 201											
10 202											
11 203											
12 204											AUMULT(3)
13 205											
14 206											
15 207											
	1A	1B	1C	1D	1E	1F	20	21	22	23	

AU Details Map - Zone 6, Left Half (Sheet 13 of 16)

MEMORY DATA BUS BIT

HALF WORD BIT

	XA01	XB01	XC01	XD01	VM01	VL01	NM01	NL01	DM01	DL01	
OCT →	0	1	2	3	4	5	6	7	8	9	
00 208											AUMULT(4)
01 209											
02 210											
03 211											
04 212											AUMULT(5)
05 213											
06 214											
07 215											
08 216	AXQABM(16 − 31) 1	AXQABL(48 − 63) 1	AXQCDM(16 − 31) 1	AXQCDL(48 − 63) 1	AXQDVRM(16− 31) 9	AXQDVRL(48 − 63) 9	AXQDVNDM(16−31) 9	AXQDVNDL(48−63) 9	AXQMODFM(16−32) 9	AXQMODFL(48 − 63) 9	AUMULT(6)
09 217											
10 218											
11 219											
12 220											AUMULT(7)
13 221											
14 222											
15 223											

AU Details Map - Zone 6,
Right Half (Sheet 14 of 16)

MEMORY DATA BUS BIT

HALF WORD BIT

	RD00	R100	R110	X002 R3	R400	R500	R600	R700	R800	R900	
OCT →	0	1	2	3	4	5	6	7	8	9	

00
224
	AMQACC(1)
	AMQROMFF(0)
	0
		AMQBTNRC
		AMQROMFF(10)
			AMQFLDIV
			AMQROMFF(20)
				AMQXOFCD(1)
				AMQROMFF(30)
					AMQPKPK
					AMQROMFF(40)
								AMQADTYP	AMQNGT(9 – 10)		AUROMFF

01
225
	AMQACC(6)
	AQROMFF(1)
	0
		AMQCRCSH
		FF(11)
			AMQFLDNM
			FF(21)
				AMQXOFCD(2)
				FF(31)
					AMQPPCMP
					FF(41)
							AMQSOFSL(1 – 3)	AMQDL			

02
226
	AMQACC(8)
	FF(2)
	0
		AMQDIV
		FF(12)
			AMQFLMDF
			FF(22)
				AMQLGARS
				FF(32)
					AMQPPCPS
					FF(42)
						AMQRCEN(15 – 18)		AMQFL	AMQNGT(13)	ANQOGT(6 – 10)	

03 227	AMQACTPS FF(3) 0	AMQDIVDP FF(13)	AMQFLMUL FF(23)	FF(33)	AMQPPINL FF(43)		AMQSRCK2	AMQFLFXC			

04
228
	FF(4)
	0
		AMQDLMUL
		FF(14)
			AMQFLSBM
			FF(24)
				AMQLOGCD(0 – 2)
				FF(34)
					AMQRCEN(5)
					FF(44)
								AMQFXFLC	AMQNGT(16 – 17)		

05 229	FF(5) 0	AMQFIRS FF(15)	AMQFLSUB FF(25)	FF(35)	FF(45)	AMQRCEN(20 – 21) ANQRCEN (17)	ANQXOFCD(0)	AMQHL	ANQXOFCD (1)	ANQOGT(13)	

06
230
	AMQAOFSL(0 – 3)
	FF(6)
	0
		AMQFLAD
		FF(16)
			AMQFXNOR
			FF(26)
				AMQLOGCP
				FF(36)
					AMQRCEN(7 – 9)
					FF(46)
						AMQRSACC	AMQSCCD	AMQHWQOT	ANQXOFCD
									(2)		

07 231	FF(7) 0	AMQFLADM FF(17)	AMQFXSH FF(27)	FF(37)	AMQRCEN (2)	AMQSHIF	AMQVDP	AMQDIVFX		ANQOGT(16 – 17)	

08
232
	AMQARCP
	FF(8)
		AMQFLAR
		FF(18)
			AMQG2DIS
			FF(28)
				AMQNORM(0 – 2)
				FF(38)
					ANQRCENC
					(1)
						AMQSLMUL	AMQVECT	AMQCONT(11)	ANQOGT(1 – 2)	ANQOGT(25)	

09 233	AMQARSH FF(9)	AMQFLDAB FF(19)	AMQG8DIS FF(29)	FF(39)	AMQRCEN(13 – 14) ANQRCENC (0)	ANQRCEN (18)	AMQXCT(0)	ANQARCP	AMQNGT(3)	AMQNGT(19)	
10 234	ANQOGT(0)	ANQOGT(14)				AMQNGT(2)	AMQNGT(2)			ANQRCEN (5)	
11 235	ANQOGT(11)	ANQOGT(15)	ANQOGT(19 – 21)	ANQOGT(22 – 24)	ANQOGT(26 – 27)			AMQNGT(6 – 8)	ANQOGT(3 – 5)	ANQRCEN (7)	
12 236	ANQOGT(12)	ANQOGT(18)			AMQNGT(21)	AMQNGT(1)	AMQNGT(4 – 5)			ANQXOFCD (0)	
13 237			AEQED(7)	AEQET	AEQFLCAR	ARQNSEXT	¬AAQFPS(0)				AUCONT
14 238	AEQED(1 – 3)	AEQED(4 – 6)	AMQINC	AEQAGTC	AEQEDEX	AOQSGDEL	¬AAQFPS(2)				
15 239				AEQEDARX	¬AEQSCHIE	AAQADDEX	¬ARQDCD(6)				
	1A	1B	1C	1D	1E	1F	20	21	22	23	

AU Details Map - Zone 7,
Left Half (Sheet 15 of 16)

MEMORY DATA BUS BIT

HALF WORD BIT

								R701	R801	
OCT →	0	1	2	3	4	5	6	7	8	9
00 240										
01 241										
02 242		-	U	N	U	S	E	D	-	
03 243										
04 244				NOT	CONNECTED	TO	MEMORY			
05 245										
06 246										
07 247										
08 248										
09 249										
10 250										
11 251										
12 252										
13 253										
14 254										
15 255										

AU Details Map - Zone 7,
Right Half (Sheet 16 of 16)

TEXAS INSTRUMENTS

INCORPORATED
EQUIPMENT GROUP
AUSTIN, TEXAS

PUBLICATION UPDATE

PUBLICATION

PROGRAM ASC PUBLICATION NO. 930020-2

TITLE Central Processor OMI

DATE July 1973 JOB NO. 1886A

TYPE OF CHANGE

□ IMMEDIATE
(MAY CAUSE PERSONAL INJURY OR
EQUIPMENT DAMAGE/FAILURE)

□ ROUTINE
(BATCH PROCESSED)

SUBMITTED BY

NAME ______________ PHONE ______________

ADDRESS ______________

MAIL STATION ______________ DATE ______________

LIST PAGE AND PARAGRAPH OR FIGURE NUMBERS AND DESCRIBE RECOMMENDED CHANGES.

FORWARD CHANGES BY FOLDING THIS SHEET AND STAPLING. RETURN ADDRESS IS ON BACK OF SHEET.

TEXAS INSTRUMENTS INCORPORATED
EQUIPMENT GROUP
P.O. BOX 2909
AUSTIN, TEXAS 78767

ATTENTION: TECHNICAL DATA BRANCH
MAIL STATION 2146