IBM Field Engineering

Theory of Operation

2065 Processing Unit, Volume 2

Preface

This manual describes the operation of the 2065 Processing Unit. It is assumed that the reader has a knowledge of processors, of ALD interpretation, and of the basic circuits used in the 2065.

The EC levels of the ALD's and CLD's for the basic 2065, upon which this manual and its companion maintenance diagram manual are based, are:

ALD's:	EC 705369	9/68
CLD's:	EC 705340	3/68
Power:	EC 711576	8/68

The manual consists of two volumes, and is divided into six chapters and three appendices. Volume 1, Form Y27-2036-0, contains:

Chapter 1, Introduction. Discusses system organization and data flow; character codes, instruction formats, and operands; program execution and control; and the CPU functional units and the Universal instruction set.

Chapter 2, Functional Units. Analyzes registers, adders, and counters individually, except for those units that work together to perform a specific function (for example, variable-field-length register and its associated byte counter).

Volume 2, Form Y27-2037-0, contains:

Chapter 3, Principles of Operation. Presents a detailed analysis of instruction fetching, and instruction execution by instruction class.

Chapter 4, Features. Discusses the features available for the 2065 CPU.

Chapter 5, Power Distribution and Control. Describes the power distribution and control within the CPU (making a distinction between 2065's and 2060's that have been converted to 2065's) and within the system.

Chapter 6, Console Controls and Maintenance Features. Discusses the controls on the system control panel and on the CE panel and their application, and the maintenance features available.

Appendix A, Special Circuits. Discusses the special circuits in the 2065.

Appendix B, World Trade Differences. Discusses the major difference between the World Trade version of the Model 65 and the domestic version.

Appendix C, Example of FLT Generation. Discusses FLT generation, using a simple four-block tree as an example.

Volume 2 also contains the index for the complete manual.

Following most paragraph heads are bullets (key statements preceded by ●) which summarize significant points about the subject. The bullets serve two functions: (1) they provide the CE with the key points of the topic, and (2) they provide quick reference for review and recall for the CE who is familiar with the machine. Detailed text follows, providing the non-classroom student with the fill-in material necessary for self-instruction.

The diagrams supporting the text are divided into two groups: (1) purely instructional diagrams and (2) maintenance-oriented diagrams and diagrams that aid recall. Examples of the first group are high-level block diagrams and diagrams that show general data flow and timing considerations. These diagrams are generally not affected by engineering changes, and, if they include AND/OR logic blocks, the blocks are drawn in positive logic convention and do not maintain ALD lines or line names. The instruction diagrams, which are placed in this manual and called "Figures", are numbered consecutively within a chapter. (For example, 1-1 is the first figure in Chapter 1; 3-7 is the seventh figure in Chapter 3.)

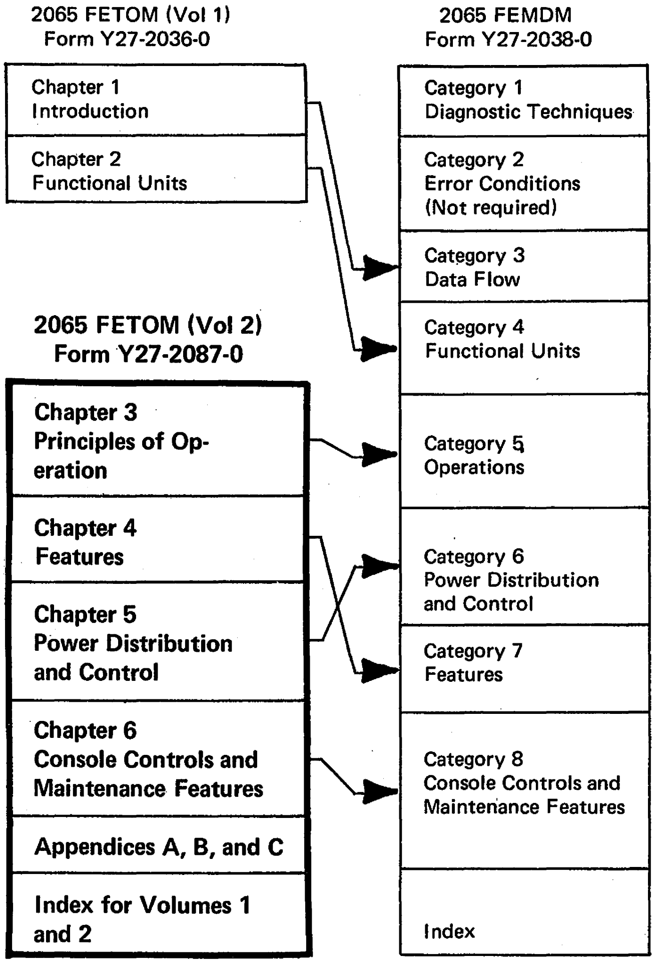
The diagrams of the second group are referenced in this manual (for example, Diagram 5-30, FEMDM) but are located in the companion FE Maintenance Diagrams Manual to allow ready reference during maintenance and to facilitate updating the diagrams to new engineering levels. These diagrams are grouped by categories similar to the chapters of this manual.

Second Edition (May 1974)
This edition, SY27-2037-1, is a reprint of SY27-2037-0, incorporating changes released in the following FE Supplement:
SY27-2259 (dated October 16, 1969)

Changes are periodically made to the information herein; any such changes will be reported in subsequent revisions or Technical Newsletters.

This manual has been prepared by the IBM System Products Division, Product Publications, Dept. B97, PO Box 390, Poughkeepsie, N.Y. 12602. A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be sent to the above address. Comments become the property of IBM.

The relationship of this manual to the FEMDM is shown below. (Arrows indicate cross-referencing between chapters in this manual and categories of diagrams in the FEMDM: for example, most references in Chapter 3 are made to Category 5 diagrams.)



Companion, related, and prerequisite manuals and standards are:

2065 Processing Unit
FEMDM, Form Y27-2038-0
FEMM, Form Y27-2270-0

IBM System/360 Principles of Operation, SRL, Form A22-6821-7.

2065 Processing Unit, 7070/7074 Compatibility Feature
FETOM, Form Y27-2106-0
FEDM, Form Y27-2107-0

2065 Processing Unit, 7080 Compatibility Feature
FETOM, Form Y27-2090-0
FEDM, Form Y27-2091-0

2065/2067 Processing Unit, 709/7040/7044/7090/7094/7094II Compatibility Feature
FETOM, Form Y27-2098-0
FEDM, Form Y27-2099-0

2365 Processor Storage
FETOM, Form Y22-6608-0
FEDM, Form Y22-6601-1
FEMM, Form Y22-6600-1

2361 Core Storage
FETOM, Form Y22-2897-0
FEDM, Form Y22-2895-0
FEMM, Form Y22-2894-0

2860 Selector Channel
FETOM, Form Y27-2220-0
FEMDM, Form Y27-2221-0
FEMM, Form Y22-2893-1

2870 Multiplexer Channel (70,000 Series)
FETOM, Form Y27-2152-0
FEDM, Form Y27-2153-0
FEMM, Y27-2154-0

1052 Adapter and 2150 Console, FETOM, Form Y22-2808

SLT Component Circuits, FEMI, Form Z22-2798 (IBM Confidential)

SLT Power Supplies, FEMI, Form 223-2799

SLT Packaging, FEMI, Form 223-2800

Control Automation System (CAS) Logic Diagram (CLD), IBM Corporate Engineering Standard, CES 0-1046-4

Abbreviations

ABC	AB register byte counter
ac	alternating current
adr	address, addressed, addressing
ALD	automated logic diagram
amp	ampere
ASC	address store compare
ATN	alternate test number
BCD	binary-coded decimal
BCU	bus control unit
C	capacitor
CAW	channel address word
CB	circuit breaker
CC	condition code
CCW	channel command word
CE	customer engineer
charistic	characteristic
CLD	control automation system logic diagram
CPU	central processing unit
CR	diode

CROS	capacitive read-only storage
CSW	channel status word
CT	conditional terminate
dc	direct current
dec div	decimal divide
dec ovflo	decimal overflow
DX	first byte in a series of destination bytes
DX + 1	second byte in a series of destination bytes
DX + 2	third byte in a series of destination bytes
end op	end operation
EPO	emergency power off
ERSLT	expected result
exp ovflo	exponent overflow
exp unflo	exponent underflow
F	fuse
FEMDM	Field Engineering Maintenance Diagrams Manual
FEMI	Field Engineering Manual of Instruction
FEMM	Field Engineering Maintenance Manual
FETOM	Field Engineering Theory of Operation Manual
fix-pt div	fixed-point divide
fix-pt ovflo	fixed-point overflow
FLT	fault locating test
flt-pt div	floating-point divide
FLUT	Fault Locating Utility program
FPR	floating-point register
fract	fraction
GIS	general initialization sequence
GPR	general-purpose register
hex	hexadecimal
HSS	high-speed storage
Hz	Hertz
IC	instruction counter
I-Fetch	instruction fetching
ILC	instruction length code
I/O	input/output
IPL	initial program load
K	kilo
K	relay
kHz	kilohertz
LAL	local storage address latches
LAR	local storage address register
LCS	large capacity storage
LS	local storage
LSWR	local storage working register
MAR	memory address register
max	maximum
MCW	maintenance control word
mHz	megahertz
MMSC	maintenance mode stop clock
MPR	multiplier
ms	millisecond
multisys	multisystem
no op	no operation
ns	nanosecond

op code	operation code
oper	operation
opr	operand
P	parity
PAA	parallel adder A-side
PAB	parallel adder B-side
PAL	parallel adder latch
pf	picofarad
PK	power contactor
PP	partial product
PQ	partial quotient
priv oper	privileged operation
prot	protection
PS	power supply
PSW	program status word
R	resistor
ROS	read-only storage
ROSAR	read-only storage address register
ROSBR	read-only storage backup register
ROSDR	read-only storage data register
ROSPARA	read-only storage previous address register A
ROSPARB	read-only storage previous address register B
SAA	serial adder A-side
SAB	serial adder B-side
SAB	storage address bus
SAL	serial adder latch
SAR	storage address register
SBA	serial adder bus A
SBB	serial adder bus B
SCOPEX	scoping index
SCR	silicon-controlled rectifier
SDBI	storage data bus in
SDBO	storage data bus out
signif	significance
SLT	solid logic technology
SMS	standard modular system
SOROS	scan out read-only storage
spec	specification
SRL	Systems Reference Library
STAT	status trigger
STC	ST register byte counter
stg	storage
SW BD	switch board
sync	synchronizing
T	transformer
T(DX)	table byte specified by DX
T(DX + 1)	table byte specified by DX + 1
TIC	transfer in channel
TN	test number
uf	microfarad
usec	microsecond
UT	unconditional terminate
V	volt
VFL	variable-field length
≥	greater than or equal to
≧	greater than or equal to
≤	less than or equal to
≦	less than or equal to
=	equal to
≠	not equal to

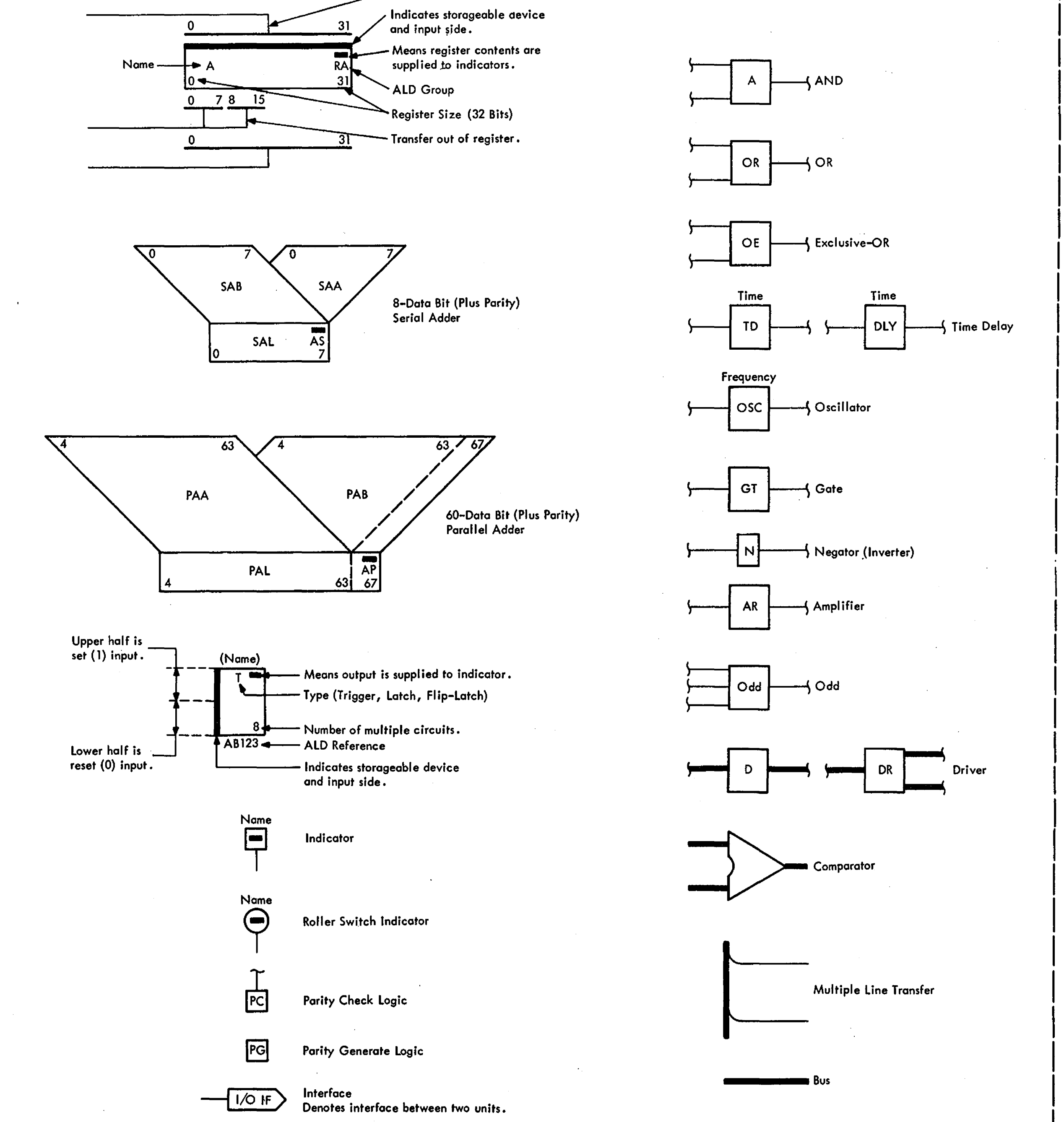
Contents

Illustrations

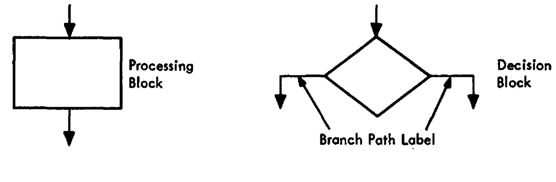
Tables

Legend

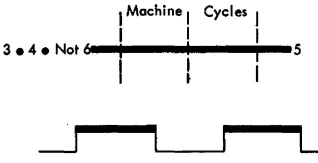
LOGIC DIAGRAMS


Transfer into register.
Indicates storageable device and input side.
Means register contents are supplied to indicators.
Name
A
RA
0
31
ALD Group
0 7 8 15
Register Size (32 Bits)
Transfer out of register.
0
31
AND
OR
Exclusive-OR
Time
TD
Time
DLY
Time Delay
0 7 0 7
SAB
SAA
SAL
AS
8-Data Bit (Plus Parity) Serial Adder
Frequency
OSC
Oscillator
4 63 4 63 67
PAA
PAB
60-Data Bit (Plus Parity) Parallel Adder
PAL
AP
4 63 67
GT
Gate
N
Negator (Inverter)
AR
Amplifier
Upper half is set (1) input.
(Name)
T
Means output is supplied to indicator.
Type (Trigger, Latch, Flip-Latch)
8
Number of multiple circuits.
AB123
ALD Reference
Lower half is reset (0) input.
Indicates storageable device and input side.
Odd
Odd
D
DR
Driver
Name
Indicator
Comparator
Name
Roller Switch Indicator
Multiple Line Transfer
PC
Parity Check Logic
PG
Parity Generate Logic
Bus
I/O IF
Interface
Denotes interface between two units.

FLOWCHARTS



TIMING CHARTS



Heavy bar indicates active state. Number(s) at beginning and end of the bar identify the signal(s) (also on the same chart) that activate and deactivate this line. "Not" preceding a number means that the deactive signal conditions this line.

Waveshape
Heavy bar indicates active state.

Chapter 3. Principles of Operation

This chapter, which discusses the 2065 CPU instructions, is divided into eight sections:

Section 1, Instruction Fetching.
Section 2, Fixed-Point Instructions.
Section 3, Floating-Point Instructions.
Section 4, Decimal Instructions.
Section 5, Logical Instructions.
Section 6, Branching Instructions.
Section 7, Status Switching Instructions.
Section 8, I/O Instructions.

Machine operation during instruction fetching and execution is controlled by ROS microprograms which are represented by CLD's. The discussions in the following sections are based upon simplified versions of the CLD's and upon upper-level, positive-logic diagrams located in the associated FEMDM.

Section 1. Instruction Fetching

Basic control for the instruction fetching (I-Fetch) operation is derived from one of four possible microprograms, depending on the format of the instruction being fetched. Each microprogram performs routines dictated by the instruction format (RR, RX, RS and SI, or SS) and is therefore common to many instructions. (The same microprogram governs the I-Fetch of RS and SI Instructions.) Subsequently, a branch is made to an appropriate microprogram for execution of a specific instruction. These individual execution sequences all terminate with a branch back to the I-Fetch microprogram to continue the sequence.

A typical microprogram sequence is shown in Figure 3-1. The correct I-Fetch microprogram to be entered upon completion of an instruction is dependent on the format of the instruction to be executed next. A test for the format of the upcoming instruction is made on the last cycle of the execution phase. The various actions performed during this last cycle (called the end operation or end-op cycle) must be thoroughly understood before undertaking a detailed analysis of each I-Fetch sequence.

BASIC END-OP CYCLE

- End-op cycle completes execution of instruction and initiates fetching of next instruction.
- End-op cycle is governed by normal end-op or branch end-op ROS word.
- Branch end op is used to speed execution of branch-type operations.

The end-op cycle is the last cycle in the execution phase. During this cycle, actions dictated by the execution phase of the instruction are completed and the fetching of the

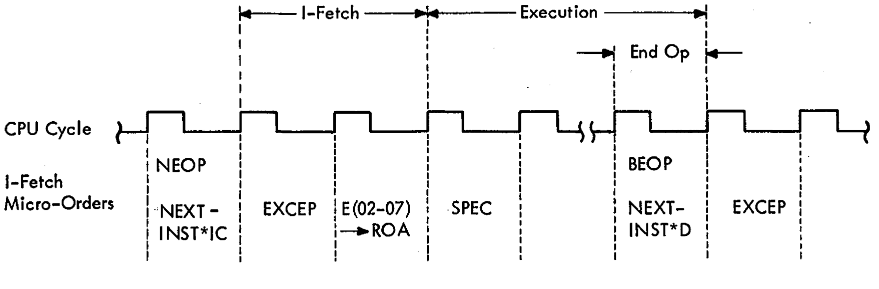


Figure 3-1. Typical Microprogram Sequence

next instruction begins. The execution phase is completed by setting the CC (if specified in the instruction) and by detecting interruptions or exceptional conditions that may have occurred during the execution phase. (The recovery microprograms are discussed after the basic end-op and I-Fetch sequences.)

The instruction fetching begins by:

1. Decoding the format of the upcoming instruction.
2. Initiating the operand fetch required by that format.
3. Establishing the correct I-Fetch sequence which is to follow.
4. Detecting the need for more instructions, and requesting new instructions from main storage when the need exists.

This discussion deals with those end-op actions that affect the subsequent I-Fetch sequence. Although instruction fetching begins during the end-op cycle, the next cycle is defined as the first I-Fetch cycle.

The setting of the CC affects the subsequent I-Fetch only when the upcoming instruction is a Branch on Condition instruction. Depending on the CC, new instructions may be requested from D (condition met) or from the IC (condition not met). The manner in which the CC is set is discussed in the specific execution sequences described in this chapter (Sections 2 through 8).

The actions performed during the end-op cycle are governed by two basic ROS end-op words: normal end-op and branch end-op. (Although they perform different functions during end-op, they perform the same functions for the subsequent I-Fetch sequence.) The normal end-op word is in control of the end operation if the address of the next instruction is specified by the IC. The next instruction is decoded from R. Conversely, the branch end-op word is in control if the address of the next instruction is specified by D. In this case, the next instruction is decoded from the SDBO (the effective R) at the start of the end operation.

The primary function of the branch end-op word is to fulfill specific timing requirements imposed upon execution of some branch instructions (see Section 6 of this chapter). Two conditions lead to a branch end-op micro-order:

1. Sometimes upon execution of a successful branch, end-op takes place before the address of new instructions (in D) has been transferred to the IC. In such cases, the branch end-op word is always in control. To establish the correct I-Fetch microprogram for the next instruction, the branch end-op word samples D(21,22) and the effective-R(0,1) bits; i.e., bits 0 and 1 of the op-code halfword to be transferred to R are sampled directly from the SDBO. Thus, the I-Fetch microprogram for the next instruction is established as soon as the instructions (specified by the branch) arrive from main storage.
2. Except for the Branch on Condition instructions, the CPU assumes that all branches are successful. Accordingly, upon predecoding a branch instruction, the CPU inhibits any IC request to refill Q and, instead, requests instructions per the branch address (in D). If, during execution, the branch proves to be unsuccessful, the instructions accessed by the D-request are not gated into Q, and the CPU must resume processing of the instructions specified by the IC. At this time it may be found that· the unsuccessful branch was the last instruction in Q. Although a request per the IC is immediately generated, at least three cycles (main storage access time) must elapse before the CPU can resume normal processing. Also, because the format of the instruction is usually decoded from R(0,1), additional time would be lost if the first halfword (arriving from main storage) had to be gated to R before the I-Fetch microprogram for the instruction could be established. Under such conditions, use of the branch end-op word increases the speed in establishing the I-Fetch microprogram for the next instruction. The instruction address (in the IC) is temporarily transferred to D. Instead of sampling R(0,1) the branch end-op word samples the effective-R(0,1) to establish the correct I-Fetch microprogram immediately upon arrival of the instructions from main storage.

Prefetching of Operands During End Op

- For RR instructions, one LS register is accessed by R1 field if not a branch instruction; by R2 field if branch instruction.
- For RX, RS, SI, and SS instructions, one LS register is accessed by B-field.

During the end-op cycle, R contains the op-code halfword of the next instruction. The format of the instruction is established by sampling R(0,1), and the operand prefetch dictated by that format is initiated. The end-op cycle is completed with the 'R→E' micro-order, which transfers the op-code halfword to E at the start of the I-Fetch sequence.

The scheme for prefetching operands during end-op time is shown in Diagram 5-1, FEMDM. During this time, an LS register specified in the R or B field of the upcoming instruction is addressed and transferred to T. The desired LS register is addressed by gating the appropriate field of the instruction to LAL. Ingating to LAL is initiated by the 'NEOP' micro-order in the normal end-op word or by the 'BEOP' micro-order in the branch end-op word.

The format of the upcoming instruction is established by decoding R(0,1):

R(0,1)	Instruction Format
00	RR
01	RX
10	RS or SI
11	SS

When an RR format is decoded, a further test is performed to determine whether the upcoming instruction is a branch. If the instruction is not a branch, the R1 field [R(8–11)] is gated to LAL. For an RR branch, however, the R2 field [R(12–15)] is gated to LAL. This action is necessary because, for branch instructions, R2 specifies the LS register containing the branch address. Since in this case a storage request for new instructions must be made as soon as possible, R2 must be gated to LAL first.

When an RX, RS, SI, or SS format is decoded, a test is made to determine which of the four halfword positions in Q contains the second halfword of the upcoming instruction. The B-field of the selected halfword is then always gated to LAL. Selection of the correct halfword in Q depends upon the ROS word (branch or normal) in control of the end-op. The normal end-op word specifies that the address of the upcoming instruction is contained in the IC. In this case, IC(21,22) indicates the Q portion from which the first halfword of the instruction has been transferred to R. Consequently, these bits are decoded to select the second halfword of the instruction in Q. The branch end-op word is in control when the address of the upcoming instructions is in D. Because in this case D(21,22) points to the correct Q position, these bits are used to select the correct B-field in Q.

Prefetching of operands from LS is from locations 0–15 (decimal), unless an RR format, floating-point instruction has been predecoded. In this case, the FPR addressed by R1 is selected by forcing LAL(0) to 1. The contents of the LS register accessed during the end-op cycle are always transferred to T. This action is performed by the '→T' micro-order in the end-op word. Thus, at the start of an I-Fetch sequence, T always contains an operand (per R-field) or the base portion of an operand address (per B-field).

At the completion of an end-op cycle, the halfword containing the op code of the instruction is transferred to E (initiated by the 'R→E' micro-order in the end-op word). Thus, further operand prefetching (by the subsequent I-Fetch sequence) is performed with the op code in E.

Fetching of Instructions by End-Op Micro-Order

A test to establish whether new instructions are required is always performed during end op. If the upcoming instruction is not a branch and Q needs to be refilled, a request for new instructions is generated at end op. If the upcoming instruction is a branch, the storage request is blocked during end op.

Under certain conditions, it is possible to request new instructions from main storage one or two cycles before end-op. This action is initiated by the 'early end-op' (EEOP) micro-order, contained in the execution sequences of some instructions. All execution sequences, including those with the 'EEOP' micro-order, terminate with the end-op word.

A Q-register refill exceptional condition usually follows an end-op request for new instructions. This exceptional condition adds one cycle to the basic RR, RX and RS, and SI I-Fetch routines.

Requests During End Op

During the end-op cycle, a test is made to establish whether Q needs to be refilled with new instructions. The outcome of this test depends upon the format of the upcoming instruction, on its position in Q, and on whether it is a branch or the subject instruction of an Execute instruction.

As shown in Diagram 5-2, FEMDM, a test of the status of Q is initiated by the normal end-op (NEOP) or branch end-op (BEOP) micro-order contained in the normal or branch end-op word, respectively. Upon the decoding of the 'NEOP' micro-order, IC(21,22) is sampled to establish which halfword position in Q has been transferred to R. The same function is performed by the 'BEOP' micro-order when the address of the upcoming instruction is contained in D. In this case, D(21,22) is examined to establish which halfword in Q is to be processed next. Depending on the instruction format decoded from R(0,1), and if the upcoming instruction is neither a branch nor the subject of an Execute instruction, storage requests per the IC may be generated when the first, second, or third halfword position in Q is to be processed next.

Q-Position Transferred to R	Setting of IC(21,22) or D(21,22)	Instruction Format	Type of Request
1st	00	SS	4-cycle
2nd	01	SS	4-cycle
		RX, RS, or SI	3-cycle
3rd	10	All formats	3-cycle
4th	11	All formats	None

Q has already been refilled during the instruction being completed if bits 21 and 22 = 11; therefore, another refilling of Q is not necessary.

Requests During Early End Op

Execution sequences of some instructions contain the 'EEOP' micro-order. The function of this micro-order, which is given 1 or 2 cycles before the 'NEOP' micro-order, is to examine the instruction status in Q and to initiate an early storage request if Q needs refilling. Requests initiated by the 'EEOP' micro-order are blocked if the next instruction to be executed is (1) a branch instruction, (2) an SS instruction, or (3) a subject of an Execute instruction.

Early requests to refill Q are generated according to conditions shown in Diagram 5-3, FEMDM. The normal end-op request is blocked when an early request is in progress. Note that the 'EEOP' micro-order can only initiate a 4-cycle request. The advantage of an early request is that the BCU will address main storage 1 or 2 cycles before end op. When initiated 2 cycles before end op, the refilling of Q does not force the Q-register refill exceptional condition if the instruction being fetched is of the RR or indexed RX format or is a shift instruction.

Selection of I-Fetch Microprogram

- Selection of I-Fetch sequence is controlled by 'NEXT-INST*IC' micro-order during normal end op or by 'NEXT-INST*D' micro-order during branch end op.
- 'NEXT-INST*IC' micro-order specifies functional ROS branch per R(0,1), IC(21,22), B = 0, and X2 = 0.
- 'NEXT-INST*D' micro-order specifies functional ROS branch per effective-R(0,1), D(21,22), B = 0, and X2 = 0.

The correct I-Fetch sequence is entered by establishing the address of the first ROS word in that sequence. This address is then placed into ROSAR so that the desired ROS control word may be obtained on the following cycle.

ROSAR(0–5) is furnished directly by the end-op word as 001000. These bits designate the address of a general I-Fetch operation about to take place. To arrive at the specific I-Fetch sequence (RR, RX, RS and SI, or SS), the bit configuration of ROSAR(6–11) must be established. The manner in which ROSAR(6–11) is established is determined by the ROS word (normal or branch) in control of the end op.

The normal end-op word contains the 'NEXT-INST*IC' micro-order specifying a 64-way functional branch. This micro-order sets ROSAR(6–11) according to the following conditions:

ROSAR Bit	Condition
6	Set if R(0) = 1
7	Set if R(1) = 1
8	Set if instruction X2 field = 0, and RX format
9	Set if instruction B field = 0, and not RR format
10	Set if IC(21) = 1
11	Set if IC(22) = 1

The above actions specify the format of the upcoming instruction, the type of further operand fetch required, and the number of counts by which IC(21,22) must be increased to select the first halfword of the instruction following it in main storage.

The registers affected by the 'NEXT-INST*IC' micro-order are shown in Diagram 5-5, FEMDM. The format of the upcoming instruction is decoded from R(0,1). For non-RR instructions, a test is made to determine whether the B-field of the instruction is equal to zero and, in the case of RX instructions, whether the X2 field is also zero.

The zero test for the B and X2 fields is necessary to establish a correct address computation by the subsequent I-Fetch routine. To increase the speed of operand prefetching, the B-field is always gated to LAL during the end-op cycle. A zero address to LAL accesses LS register 0, the contents of which may not necessarily be zero. However, the condition of B-field being zero requires that the base portion of the operand address be zero. Thus, the subsequent I-Fetch sequence selected must ignore the contents of LS register 0 (accessed by a zero B field). Similarly, in the case of the X2 field being zero, the I-Fetch sequence selected must not address LS per the X2 field. The manner in which the correct I-Fetch sequence is selected is decribed below.

Four 4-way AND's simultaneously sample the four possible B-field locations in Q. Each AND is conditioned if its corresponding four-bit input consists of all zeros. As explained previously, IC(21,22) selects the first halfword of the instruction that has been transferred from Q to R. Therefore, these bits are used as gates to select the second halfword of the instruction in Q. If, for example, the first halfword position of Q has been transferred to R and decoding of R(0,1) shows that the instruction is not of the RR format, Q(16–19) must be selected to obtain the correct B-field. When the first halfword position in Q is transferred to R, IC(21,22) is set to 00. This setting (coupled with the absence of an 'RR block' signal) selects the output of the AND that samples the correct B-field; i.e., Q(16–19). When the B-field of the instruction is found to be zero, ROSAR(9) is set to 1. This action addresses an I-Fetch microprogram that ignores the contents of the LS register accessed by the B-field.

When the upcoming instruction is of an RX format, a similar test is performed to establish whether the X2 field of the instruction, R(12–15), is equal to zero. Upon detecting a zero X2 field, ROSAR(8) is set to 1. This action dictates that the subsequent I-Fetch microprogram does not address LS per X2; i.e., E(12–15).

The ROS branch specified by the 'NEXT-INST*IC' micro-order is completed by forcing IC(21,22) into ROSAR(10,11). This action allows the first I-Fetch microinstruction to correctly update IC(21,22) and R without further testing.

The 'NEXT-INST*D' micro-order in the branch end-op word sets ROSAR(6–11) according to the following conditions:

ROSAR Bit	Condition
6	Set if effective-R(0) = 1
7	Set if effective-R(1) = 1
8	Set if instruction X2 field = 0, and RX format
9	Set if instruction B field = 0, and not RR format
10	Set if D(21) = 1
11	Set if D(22) = 1

Note that ROSAR(6,7) is set from the effective-R rather than from R, and that ROSAR(10,11) is set from D(21,22) rather than from IC(21,22). This is done because R and IC are either still invalid or are just being set by the branch operation in progress.

BASIC RR I-FETCH

- RR format:

Op Code	R1	R2
0 7	8 11	12 15

- Purpose:
  1. For nonbranch instructions, load 1st operand into A, B, and D. Load 2nd operand into S and T.
  2. For branch instructions, load 2nd operand into A, B, and D. Load 1st operand into S and T. Request new instructions, if needed.
  3. Set STC to 100 and ABC to 000.
- Conditions at start of I-Fetch:
  1. Instruction is transferred to E.
  2. If instruction is not a branch, 1st operand is in T; for a branch, 2nd operand is in T.

The following paragraphs describe the basic actions initiated by the ROS microprogram during I-Fetch of RR instructions. It is assumed that no interruptions or exceptional conditions were detected in the preceding end-op cycle.

The RR instructions basically require a 1-cycle I-Fetch. The actions initiated during this cycle are governed by 1 of 4 possible ROS control words selected at end-op time. Selection of the specific ROS word depends on the original position of the RR instruction in Q. This word contains the appropriate micro-orders for incrementing IC(21,22) and for transferring the first halfword of the next instruction to R. Except for these actions, the functions performed by the four ROS words are identical. Diagram 5-6, FEMDM, is a simplified flowchart of an RR I-Fetch; Diagram 5-7 shows the data registers used.

If no interruption or exceptional condition is detected, the entire RR instruction is transferred to E at the start of the I-Fetch cycle (by the 'R→E' micro-order at end op). The operand prefetching, initiated at end op, is then continued. The order in which operands are prefetched depends on whether the instruction is a branch:

1. For nonbranch instructions, the first operand (accessed during end op) is transferred from T via the parallel adder to A, B, and D. The second operand is then addressed by gating E(12–15) to LAL. When the second operand is accessed, it is loaded into S and T.
2. The above order is reversed for branch instructions; i.e., the second operand (accessed during end op) is placed into A, B, and D while the first operand is placed into S and T. A storage request per the branch address is generated subject to the conditions shown in Diagram 5-6.

The correct execution sequence is entered by establishing the address of the first ROS word in that sequence. This address is determined by sampling the instruction op code from E(2–7) by means of the 'E(02–07)→ROA' micro-order. As stated earlier, this description of RR I-Fetch applies only when no exceptional conditions or interruptions are present. The ROS word governing the I-Fetch cycle always contains the 'EXCEP' micro-order, which can override the functional branch per the instruction op code. Therefore, the branch to the first execution cycle occurs only when there are no interruptions or exceptional conditions.

In addition to prefetching the operands, the I-Fetch ROS word contains appropriate micro-orders to increment IC(21,22) and to transfer the first halfword of the next instruction to R. IC(21,22) is set one count higher ('X→IC' micro-order) to point at the next instruction. The first halfword of the next instruction is transferred to R by the 'QXX→R' micro-order.

Included in the first RR I-Fetch word is the 'RESET' micro-order, which causes the following actions:

1. During I-Fetch of branch instructions, initiates the request for new instructions (see Diagram 5-4) and gates E(8–11), instead of E(12–15), to LAL.
2. Resets all STAT's and Edit-instruction controls.
3. Sets STC to 100 and ABC to 000.
4. Forces LAL(0) to 1 for floating-point instructions, causing the FPR's to be addressed.
5. Sets the 'stop' trigger if operating at the instruction-step rate.

Note that IC(21,22) is not advanced, the Q-to-R transfer is not effected, and unsuccessful branch-on-condition requests are not generated if the 'execute in progress' trigger is set. The set state of this trigger indicates that the current I-Fetch is for a subject instruction of an Execute instruction. Therefore, the address of the RR instruction is specified by D and not by the IC.

BASIC RX I-FETCH

- RX format:

Op Code	R1	X2	B2	D2
0 7	8 11	12 15	16 19	20 31

- Purpose:
  1. Compute address of 2nd operand and transfer to D; request 2nd operand from main storage, if necessary.
  2. Transfer 1st operand to S and T.
- Conditions at start of I-Fetch:
  1. 1st halfword of instruction is transferred to E; 2nd halfword is in Q.
  2. Contents of LS register specified by B2 are transferred to T.

The following paragraphs describe the basic actions initiated by the ROS microprogram during I-Fetch of an RX instruction. It is assumed that no interruption or exceptional conditions were detected in the preceding end-op cycle.

The RX instructions basically require a 1- or 2-cycle I-Fetch. The actions initiated during the first I-Fetch cycle are governed by 1 of 16 possible ROS control words selected at end-op time. This selection depends on whether the B2 and/or X2 fields of the instruction are zero, and on the original position of the RX instruction in Q. The zero test establishes four separate cases for the I-Fetch routine: (1) B2 = 0 and X2 = 0, (2) B2 ≠ 0 and X2 = 0, (3) B2 = 0 and X2 ≠ 0, (4) B2 ≠ 0 and X2 ≠ 0. The first two cases require a 1-cycle I-Fetch; the last two, a 2-cycle I-Fetch. Each of the above four routines contains appropriate micro-orders for incrementing IC(21,22) and transferring the first halfword of the next instruction to R. Consequently, a four-way branch is inherent in each routine, depending on the previous IC(21,22) setting; i.e., 00, 01, 10, or 11.

Diagram 5-9, FEMDM, is a simplified flowchart of an RX I-Fetch; Diagram 5-10 shows the data registers used. If no interruptions or exceptional conditions are detected, the op-code halfword of the RX instruction is transferred to E at the start of I-Fetch (initiated by the 'R→E' micro-order at end op). The operand prefetch routine is then continued with the first halfword of the instruction in E and the second halfword in Q. The I-Fetch of non-indexed RX instructions (X2 = 0) is described first.

The 'X→IC' micro-order issued by the first I-Fetch word sets IC(21,22) two counts higher. The first halfword of the next instruction is transferred to R by the 'QXX→R' micro-order if it is now in Q; that is, if IC(21,22) did not equal 10 at the start of I–Fetch. If the instruction is not indexed, the address of the second operand is obtained by adding D2 to the base address. The contents of the LS register per B2 are placed into T at the start of I-Fetch. If B2 was found to be zero during end op, the contents of T are ignored, and the appropriate D2 field in Q is selected and routed to D via the parallel adder. However, if B2 ≠ 0, the contents of T and the D2 field are gated simultaneously to the parallel adder, and the resultant sum is transferred to D. A 3-cycle storage request for the second operand is made from D if the following conditions do not exist:

1. A Q-register refill exceptional condition is in progress.
2. The instruction is Store Halfword, Store Character, or Load Address.
3. The instruction is an unsuccessful Branch on Condition. (A request for new instructions is issued from the IC, if necessary.)

The first operand is obtained from LS per R1 and transferred to S and T. At the completion of the I-Fetch cycle, a branch is made to a specific execution sequence as determined by the 'E(02–07)→ROA' micro-order.

In the case of indexed RX instructions (X2 ≠ 0), two cycles are required to complete the I-Fetch routine (Diagram 5-11, FEMDM). During the first cycle, D2 is added to the contents of T (if B2 ≠ 0) and the result is temporarily stored into B. The LS register specified by X2 is then accessed, and its contents are placed into T. The contents of T and B are added during the second cycle, and the sum (second operand address) is transferred to D. The conditional storage request is now made. After the first operand is obtained from LS and placed into S and T, a branch per the instruction op-code is made to enter the correct execution sequence.

The 'RESET' micro-order:

1. Resets all STAT's and Edit instruction controls.
2. Resets STC and ABC to 000.
3. Initiates any necessary storage requests for branch instructions and for subject instructions of the Execute instruction.
4. Forces LAL(0) to 1 for floating-point instructions, causing the FPR's to be addressed.
5. Sets the 'stop' trigger if operating at the instruction-step rate.

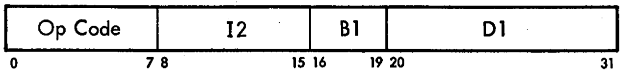
If the instruction being fetched is the subject of an Execute instruction ('execute in progress' trigger is set), the incrementing of IC, the Q-to-R transfer, and the unsuccessful Branch on Condition requests to refill Q are inhibited.

BASIC RS AND SI I-FETCH

- RS format:

Op Code	R1	R3	B2	D2
0 7	8 11	12 15	16 19	20 31

- SI format:



- Purpose:
  1. Add contents of LS register specified by B-field to D-field; place result into D.
  2. Request operand from main storage, if necessary.
  3. For RS instructions, load 1st operand into S and T. (Contents of S and T are ignored for SI instructions.)
- Conditions at start of I-Fetch:
  1. 1st halfword of instruction is transferred to E; 2nd halfword is in Q.
  2. Contents of LS register specified by B-field are transferred to T.

The following paragraphs describe the basic actions initiated by the ROS microprogram during I-Fetch of RS and SI instructions. It is assumed that no interruptions or exceptional conditions were detected in the preceding end-op cycle.

The RS and SI instructions basically require a 1-cycle I-Fetch. The actions initiated during this cycle are governed by 1 of 8 possible ROS control words selected at end-op time. This selection depends on whether the B-field of the instruction is zero, and on the original position of the instruction in Q. The zero test establishes two distinct I-Fetch routines: (1) B = 0 and (2) B ≠ 0. A four-way branch is inherent in each routine, depending on the previous IC(21,22) setting; i.e., 00, 01, 10, or 11.

Diagram 5-13, FEMDM, is a simplified flowchart of RS and SI I-Fetch. If no interruptions or exceptional conditions are detected, the halfword containing the op-code of the RS or SI instruction is transferred to E at the start of I-Fetch (initiated by the 'R→E' micro-order at end op). The operand prefetch routine is then continued with the first halfword of the instruction in E and the second halfword in Q.

The LS register specified by the B-field (B1 or B2) is accessed during end op, and its contents are placed into T at the start of I-Fetch. If the B-field was found to be zero during end op, the I-Fetch routine ignores the contents of T, selects the appropriate D-field (D1 or D2) in Q and routes it to D via the parallel adder. If the B-field is not zero, the contents of T and the D-field are gated simultaneously to the parallel adder and the sum is then transferred to D. A 3-cycle storage request for the second operand is then made from D if the following conditions do not exist:

1. A request to refill Q was generated during the previous execution segment; i.e., IC(21,22) = 01 or 10. For this case, the ROS micro-order is not contained in the I-Fetch word.
2. The E-register contains a shift, Store Multiple, Move (MVI), Test and Set, or I/O instruction. For this case, the 'D sync' latch is prevented from being set.
3. The E-register contains a branch-on-index (BXH, BXLE) instruction. For this case, the 'RESET' micro-order resets the '3-cycle request' trigger, causing a 4-cycle storage request to be made from D regardless of IC(21,22).

Upon loading D with the second operand address, the I-Fetch routine proceeds to set IC(21,22) two counts higher, to transfer the first halfword of the next instruction to R if it is in Q, and to establish the first ROS control word for the execution phase. Fetching of the first operand per E(8–11) is meaningful only for RS instructions. For SI instructions, E(8–11) contains a portion of the immediate operand. Since a common ROS control word governs the I-Fetch of both formats, E(8–11) is always gated to LAL; the contents of the LS register thus accessed are placed into S and T. However, the subsequent execution sequences for SI instructions ignore the contents of S and T.

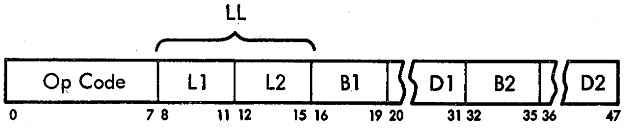
In addition to causing a 4-cycle storage request during the I-Fetch of a branch-on-index instruction, the 'RESET' micro-order:

1. Resets all STAT's and Edit-instruction controls.
2. Resets STC and ABC to 000.
3. Sets the 'stop' trigger if operating at the instruction-step rate.

If the instruction being fetched is the subject of an Execute instruction ('execute in progress' trigger is set), the incrementing of IC and the Q-to-R transfer is inhibited.

BASIC SS I-FETCH

- SS format:



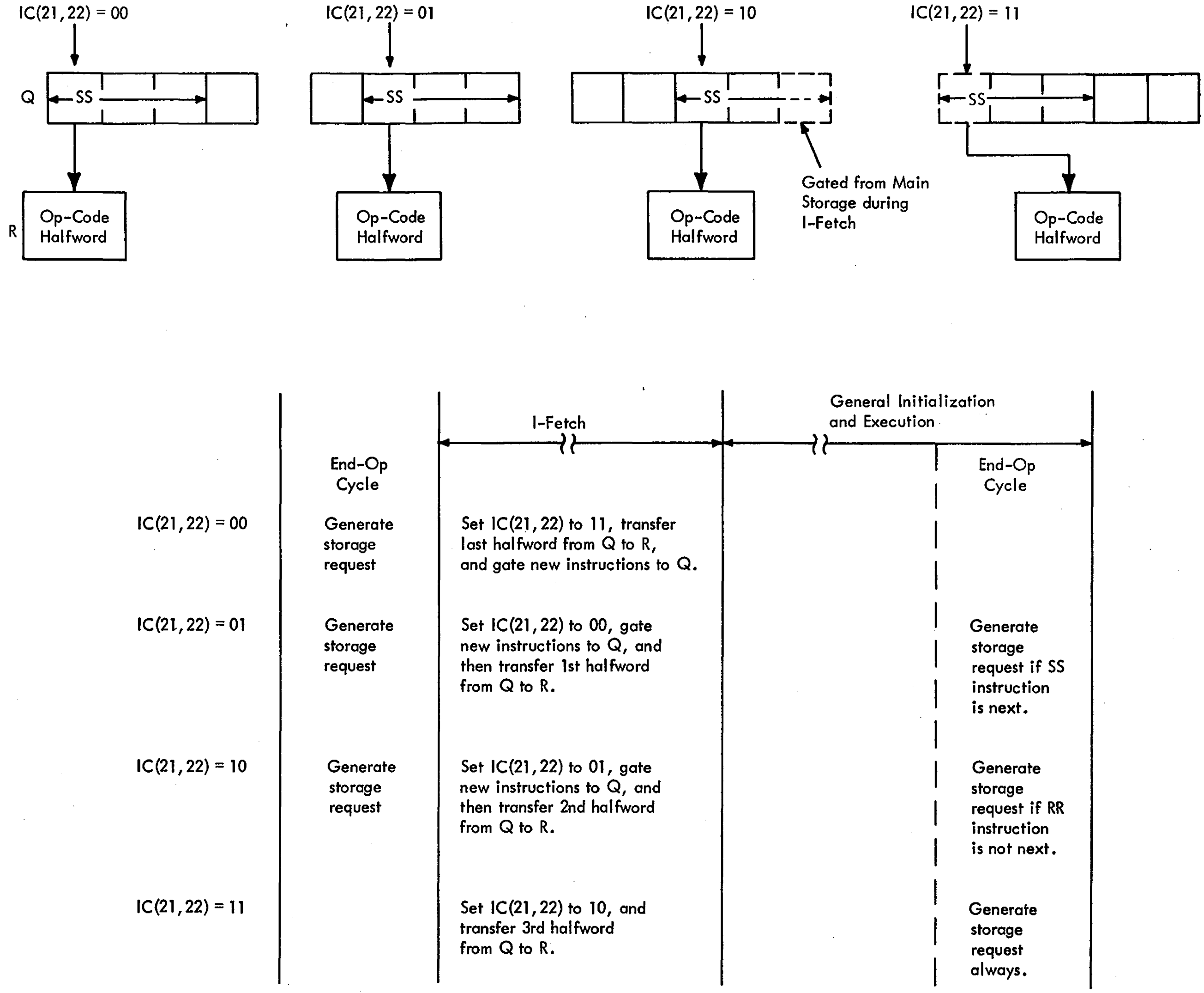
- Purpose:
  1. Transfer op-code halfword of next instruction to R; update IC and place into LSWR.
  2. Transfer computed address of 1st operand (destination) per instruction class to D; request destination operand from main storage (gated into CPU at start of 2nd execution cycle).
     a. Lowest destination address for logical instructions = base address (per B1) + D1.
     b. Highest destination address for decimal instructions = base address (per B1) + D1 + L1.

3. Transfer computed address of 2nd operand (source) to IC and T. Lowest source address = base address (per B2) + D2.
4. Perform ASC test (and invalid instruction address test if complete instruction is in Q).

- Conditions at start of I-Fetch:
  1. 1st halfword of instruction is transferring to E, 2nd halfword is in Q, 3rd halfword is in Q if IC(21,22) ≠ 10 (otherwise 3rd halfword is gated to Q during 4th cycle of I-Fetch).
  2. Base address (per B1) is in T.
  3. Q refill is not in progress if IC(21,22) = 11.

The I-Fetch of SS instructions differs considerably from the I-Fetch routines described thus far (RR, RX, RS, and SI). Differences arise from three characteristics of the SS format: (1) the SS format is three halfwords long, (2) an SS instruction always stores the results into main storage, and (3) two main storage addresses are specified.

As previously stated, requests to refill Q are generated before the CPU runs out of instructions. In describing the I-Fetch microprograms used for RR, RX, RS, and SI formats, it was assumed that the instruction to be executed was contained in Q. Because of the manner in which storage requests for instructions are generated, the assumption is valid for all 1- and 2-halfword instructions. For SS instructions, however, the I-Fetch routine may sometimes begin while the last halfword of the instruction is still in main storage. Figure 3-2 shows all possible locations that the SS instruction may assume in Q and the manner in which storage requests are generated for more instructions. Storage requests for SS instructions are generated (at end-op time) when IC(21,22) is set to 00, 01, or 10. If IC(21,22) = 00 or 01, the entire instruction is in Q at the start of I-Fetch. However, if IC(21,22) = 10, the last halfword of the instruction will arrive from main storage on the fourth cycle of I-Fetch. Consequently, processing of the third halfword of the instruction cannot



	End-Op Cycle	I-Fetch	General Initialization and Execution	End-Op Cycle
IC(21,22) = 00	Generate storage request	Set IC(21,22) to 11, transfer last halfword from Q to R, and gate new instructions to Q.		
IC(21,22) = 01	Generate storage request	Set IC(21,22) to 00, gate new instructions to Q, and then transfer 1st halfword from Q to R.		Generate storage request if SS instruction is next.
IC(21,22) = 10	Generate storage request	Set IC(21,22) to 01, gate new instructions to Q, and then transfer 2nd halfword from Q to R.		Generate storage request if RR instruction is not next.
IC(21,22) = 11		Set IC(21,22) to 10, and transfer 3rd halfword from Q to R.		Generate storage request always.

Figure 3-2. Basic Sequencing for SS Instruction

start until Q is refilled. Finally, if IC(21,22) = 11, at end op, Q has been refilled as a result of a previous end-op cycle and Q(0–31) contains the balance of the upcoming SS instruction.

An SS instruction operates on two operands obtained from main storage and stores the result into the same location from which the first operand was obtained. Therefore, the address of the first operand is also the destination address; the address of the second operand is commonly referred to as the source address. The first and second operand addresses are calculated in a manner similar to that of two-halfword instructions. The address of the first operand is computed first and loaded into D, and a storage request for the operand is made. The partial address of the second operand is then computed while the contents of the IC are transferred to the LSWR. The partial second operand address is loaded into the IC. After completing the I-Fetch routine, a General Initialization Sequence, GIS, is performed, after which control is transferred to the execution phase. (During GIS, the calculation of the second operand address is completed, and a storage request issued, if necessary.) Upon execution of the instruction, results are stored into main storage per the address in D (first operand or destination address).

Address Store Compare (ASC) Test

- Main storage address where data is to be stored is compared with address of current instructions.
- Comparison is made whenever data is stored into main storage.
- If data is stored at instruction address, 'PSC' trigger is set to indicate that instructions in Q must be refetched.
- For SS instructions, ASC test is performed during I-Fetch. Lower and upper limits of destination address are compared with instruction address.

An ASC test must be made each time the CPU stores data into main storage. This test compares the destination address of the data with the current instruction address. If it is found that both addresses are the same, the 'program store compare' (PSC) trigger is set, indicating a need to refetch instructions; i.e., the instructions currently in Q must again be obtained from main storage because the next instruction to be executed may have been modified by the data just stored.† For all *but* SS instructions, the ASC test is made during the execution phase whenever a store operation is performed. Because, in the case of SS instructions, a store operation is always implied, an ASC test has been incorporated in the SS I-Fetch microprogram.

For SS instructions, the ASC test must determine that instructions (currently in Q) were not obtained from a region defined by the upper and lower limits of the destination address for data. This test is made in two steps, as illustrated in Figure 3-3. The first step determines whether the lower limit of the destination address is above the instruction address in the IC. When the lower limit is above, the upper limit must also be above the IC, and the 'PSC' trigger is not set. This condition indicates that current instructions (in Q) cannot be affected by the subsequent store operations. However, if the lower limit of the destination address is found to be below the IC, the 'PSC' trigger is set and a further test must be made to establish that data will not be stored in the instruction path. The last step compares the upper limit of the destination address with the IC. If the IC is found to be above the upper limit, the 'PSC' trigger is reset. In such cases the subsequent store operations will not extend to the IC location. On the other hand, if the IC points below the upper limit, the 'PSC' trigger remains set, indicating that the subsequent store operations may affect the next instruction. Consequently, after execution of the instruction that caused the PSC condition, an exceptional condition microprogram is initiated to refetch the instructions in Q. The details of the refetch microprogram are described under "Program Store Compare Exceptional Condition".

I-Fetch Microprogram

- If request for new instructions has been generated at end op, I-Fetch routine requires 7 cycles; if not, I-Fetch requires 6 cycles.
- Setting of IC(21,22) at end op determines manner in which I-Fetch is performed.

The following paragraphs describe the basic actions initiated by the ROS microprogram during I-Fetch of SS instructions. It is assumed that no interruptions or exceptional conditions were detected in the preceding end-op cycle.

The first halfword of the SS instruction is transferred from R to E at the start of I-Fetch. The operand prefetch (initiated at end op) is then continued with the first halfword in E, the second halfword in Q, and the third halfword in Q [or in main storage if IC(21,22) = 10], as shown in Diagram 5-14, FEMDM.

The SS instructions require a 7- or 6-cycle I-Fetch. The actions initiated during the first I-Fetch cycle are governed by 1 of 8 possible ROS control words selected at end-op time. This selection depends on whether the B1 field of the instruction is zero and on the setting of IC(21,22). The setting of IC(21,22) establishes four distinct cases for the SS I-Fetch microprogram:

1. When IC(21,22) = 00, a 4-cycle storage request to refill Q is generated at end op. Because Q(48–63) contains

†The refetch routine is initiated if the result of a comparison of the destination and IC addresses falls within a 16-byte safety margin: Destination address (in D) = IC address ± 16 bytes. Thus, instructions in Q may not necessarily be modified by the store operation.

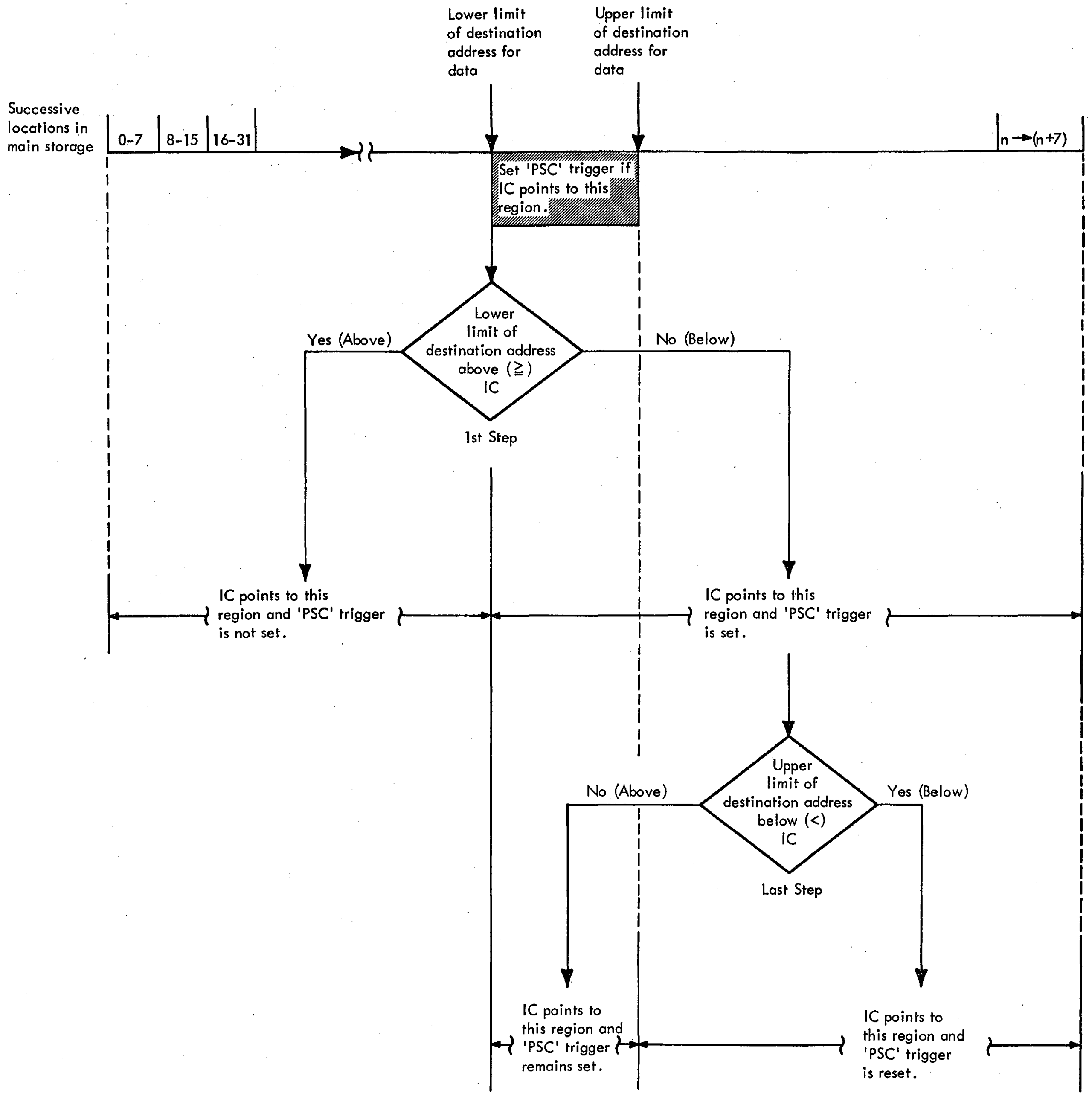

Lower limit of destination address for data
Upper limit of destination address for data
Successive locations in main storage
0-7
8-15
16-31
n→(n+7)
Set 'PSC' trigger if IC points to this region.
Lower limit of destination address above (≧) IC
1st Step
Yes (Above)
No (Below)
IC points to this region and 'PSC' trigger is not set.
IC points to this region and 'PSC' trigger is set.
Upper limit of destination address below (<) IC
Last Step
No (Above)
Yes (Below)
IC points to this region and 'PSC' trigger remains set.
IC points to this region and 'PSC' trigger is reset.

Figure 3-3. ASC Test for SS Instructions

the op-code halfword of the next instruction, the I-Fetch routine must transfer this halfword to R before the next doubleword arrives from main storage. Also, the I-Fetch routine must gate the new instructions to Q at the correct time.

2. When IC(21,22) = 01, a 4-cycle request to refill Q is generated. Because Q contains no new instructions, the I-Fetch routine must wait until Q is refilled and then transfer the first halfword of the next instruction from Q to R. New instructions must be transferred to Q at the correct time.
3. When IC(21,22) = 10, a 3-cycle request to refill Q is generated. Because the third halfword of the SS instruction is in main storage, processing of this halfword is delayed until Q is refilled. New instructions must be transferred to Q at the correct time, after which the next instruction may be gated to R.
4. When IC(21,22) = 11, a storage request is not generated because Q has been refilled as a result of the previous end op. Thus the I-Fetch routine is not concerned with transferring new instructions to Q.

During the first I-Fetch cycle of all SS I-Fetch routines, the lower limit of the destination address is calculated and placed into D, and IC(21,22) is advanced to indicate the first halfword of the next instruction. At the start of I-Fetch, T contains the base portion of the destination address (LS contents per B1). If B1 ≠ 0, the contents of T are added to the appropriate D1 field and the sum is transferred to D. If, however, B1 = 0, the contents of T are ignored and only the D1 field is routed via the parallel adder to D. After the first I-Fetch cycle, D contains the lower limit of the destination address.

The I-Fetch sequence after the first cycle differs for each setting of IC(21,22). The following paragraphs describe the I-Fetch control for each setting.

I-Fetch Control If at End Op IC(21,22) = 00

- STAT D is set to indicate B2 = 0.
- Branch per instruction is made to establish starting destination address.
- If 'execute in progress' trigger is reset, IC(20) is advanced by 1; if trigger is set, IC is not incremented.
- Instruction address is stored into LSWR.
- I-Fetch requires 7 cycles.

If at end op IC(21,22) = 00, Q contains the remaining two halfwords of the SS instruction and Q(48–63) contains the halfword of a new instruction. Q(48–63) is transferred to R by the second ROS word in the I-Fetch sequence. This word also accesses the LS register specified by B2 and initiates the ASC test. The LS register is accessed by gating the B2 field [Q(32–35)] to LAR and transferring the LS contents to S. (STAT D records the condition when B2 = 0 and is interrogated when the source address is computed.) The ASC test is initiated by subtracting the lower limit of the destination address (contained in D) from the IC. If the difference is equal to or greater than zero, the 'PSC' trigger is set to indicate that the destination address may overlap the instruction path.

A branch is made per the instruction op code to the third I-Fetch cycle. This branch establishes the manner in which the upper limit of the destination address must be obtained:

1. For decimal instructions, the L1 field [E(8–11)] is transferred to the parallel adder, where it is added to the contents of D.
2. For logical instructions, the LL field [E(8–15)] is added to the contents of D.

Also, depending on the instruction, either the upper or the lower limit of the destination address becomes the starting point from which operands are to be processed. For all decimal instructions, operand processing starts from the upper limit of the destination address and proceeds toward the lower limit. Conversely, for all logical instructions, operand processing starts from the lower limit and proceeds toward the upper limit. Thus, when a decimal instruction is decoded, the upper limit of the destination address is transferred to D and becomes the starting address from which the first operand will be requested. In the case of logical instructions, the original contents of D (lower limit of destination address) are not changed. Because the upper limit of the destination address is required to complete the ASC test, this address is temporarily stored into T.

The fourth ROS word in the I-Fetch sequence initiates calculation of the source address and gating of new instructions to Q, and requests the first operand from main storage (per the destination address in D). At the start of the fourth cycle, STAT D is tested to establish whether the B2 field of the instruction is zero. If B2 is not zero (STAT D not set), the contents of S (where LS contents per B2 have been placed) and the appropriate D2 field in Q are simultaneously gated to the parallel adder, and the sum is temporarily stored into B. If B2 = 0 (STAT D set), however, the contents of S are ignored and only the D2 field is placed into B. With the completion of the above actions, all halfwords in Q have been processed and Q is refilled with new instructions. The fourth ROS word also initiates a 4-cycle request per D for the destination operand. A ROS micro-order in the first GIS word gates the destination operand into the CPU.

The fifth ROS word in the I-Fetch sequence completes the ASC test. The upper limit of the destination address (contained in T) is subtracted from the IC contents. If the difference is greater than zero, the 'PSC' trigger is reset. This condition indicates that the instruction address is above the highest main storage location into which data is to be stored. However, if the difference (IC minus T) is less than zero, the 'PSC' trigger remains set. The set state of the 'PSC' trigger initiates the program store compare exceptional condition after execution of the current SS instruction.

The sixth ROS word in the I-Fetch sequence usually increments the IC by 8 and then stores the updated address into the LSWR. Following each storage request from the IC, the contents of the IC must be updated by 8 to obtain the address of the next doubleword location from which subsequent instructions will be requested. An exception to this rule occurs if the SS instruction currently being processed is the subject of an Execute instruction. Because, upon execution of the subject instruction, the instructions previously contained in Q must be refetched from the main storage address specified in the IC, the contents of the IC are not updated. Thus, before storing the IC into the LSWR, a test must be made to determine whether an Execute instruction is in progress. This test is accomplished by examining the status of the 'execute in progress' trigger.

6. Main storage requests. Requests for new instructions because of a predecoded branch instruction are inhibited by blocking the decoding of the 'RESET' micro-order. Storage requests for operands are inhibited by blocking the decoding of the 'MS-REQ*D-3' micro-order.
7. Initiation of the invalid instruction address test.

The 'block I-Fetch' trigger is not set by detection of the invalid instruction address test or Q-register refill exceptional conditions. In the first case, because addressing is at fault, the next address must be computed and retained for subsequent evaluation. In the second case, the trigger is not set because this action would inhibit the purpose of the Q-register refill exceptional condition.

The 'block I-Fetch' trigger is reset by the '0→STAT D' micro-order issued by the first word in the recovery microprogram.

Timer Exceptional Condition

- Initiated by 'time clock step' trigger.
- Decrements timer in location 50 (hex) per power-line frequency.
- Sets 'time clock at limit' trigger if timer value is less than zero, thus causing external interruption.

The timer value is stored in permanent main storage location 80, decimal (50, hex). It is stepped at a rate determined by the input power frequency. If the input power frequency is 60 Hz, the timer is stepped each 16.67 ms; if 50 Hz, the timer is stepped each 20 ms. The timer exceptional condition detection scheme is shown in Diagram 5-17, FEMDM.

Each time the input power swings to a positive peak, the 300-ns singleshot† generates a signal to set the 'sample pulse' trigger if the DISABLE INTERVAL TIMER switch is not activated. The output of this trigger sets the 'time clock step' trigger, provided that the CPU is not in the end-op cycle. Once set, the 'time clock step' trigger initiates the timer update sequence.

The need to inhibit setting of the 'time clock step' trigger at end op is twofold: (1) the timer exceptional condition is asynchronous with respect to program execution, and (2) it has the highest priority. Because priority is established at end op, sampling the timer exceptional conditions at this time could result in a priority conflict with a pending interruption or exceptional condition. Thus, use of two triggers ensures that timer priority is present before entry into end op; i.e., if the need to update the timer arises at end op, this event is recorded by the 'sample pulse' trigger, and the timer update sequence is initiated on the next end-op cycle.

Once the 'time clock step' trigger is set, it inhibits the priority circuits of all other interruptions and exceptional conditions. When end op occurs, the trigger output is gated to alter the subsequent I-Fetch by inhibiting the loading of E from R. D is set to 50 (hex) to obtain the timer value from main storage and, if an Execute instruction is not in progress, the 'block I-Fetch' trigger is set. The 'EXCEP' micro-order in the first I-Fetch cycle detects timer priority and forces address 014 (hex) into ROSAR.

The address forced into ROSAR by the timer exceptional condition is used to enter a timer update microprogram. A flowchart of the hardware operations just explained and of the timer update microprogram is shown in Diagram 5-17, FEMDM. The microprogram resets the 'block I-Fetch' trigger and issues a 3-cycle storage request per D to fetch the timer value. While the fetch is in progress, any protection checks from storage are ignored. After the fetch, the 32-bit timer value is loaded into A and then decremented by 5 or 6, depending on 60- or 50-Hz power input frequency, respectively.† The updated timer value is placed into S and T, and then stored per the D-address (location 50, hex). Before storing, however, the timer value is sampled to see if it has been decremented to less than 0. If this condition exists, the 'time clock at limit' latch is set to request an external interruption on the following end op.

CPU Store In Progress Exceptional Condition

Regarding the protection violation, a unique situation occurs if at end op a store operation is also in progress. In this case, detection of an interruption or exceptional condition could cause a failure to detect a "late" storage protect violation if it occurs. To prevent this situation, a special circuit is provided to test for a store-in-progress condition at end op (Diagram 5-18, FEMDM). If an exceptional condition to I-Fetch or a Load PSW instruction has been detected while a store operation is in progress, this circuit forces a microprogram that provides a 2-cycle delay to allow recording of a possible protection check and establishing the correct priority for the subsequent program interruption.

Machine Check Interruption

- Follows log-out microprogram.
- Clears interruption code, resets STAT H, enters common interruption routine.

†The singleshot output is sent to the I/O channel to provide an automatic restart; it is also used by the pulse-mode controls (see Chapter 6, Section 1). Also, if the Multisystem feature is installed, it is used for timing external resets and for detecting system inactivity.

†The variation of decrement value is determined by a pluggable card. This card (type 3388) is inserted into position 01BC2F2, AP811, when 60-Hz input power is used.

A machine check interruption is initiated only by a log-out operation. The machine check interruption detection scheme is shown in Diagram 5-19, FEMDM.

When the log-out operation (initiated by the Diagnose instruction, or a machine check error coupled with the machine check mask-bit being on) is about to be concluded, the '1→MCH-CK-TRP' micro-order causes the 'machine check interrupt' trigger to be set. Once set, the trigger blocks any new machine checks from initiating another log-out operation. This trigger also establishes machine check priority, and the operation then waits for the logout to finish, signified by an end op, before continuing. At end-op time, the set state of the 'machine check interrupt' trigger forces D to 30 (hex) and, if an Execute instruction is not being concluded, sets the 'block I-Fetch' trigger, inhibiting most of the I-Fetch actions.

The 'EXCEP' micro-order in the first I-Fetch cycle detects machine check priority and forces address 00C (hex) into ROSAR. The ROSAR address causes a branch to an interruption microprogram that stores the old PSW per the D-address. Subsequently, the microprogram loads the new PSW per the address in D + 40 (hex). The operation then proceeds as directed by the new PSW.

A flowchart of the hardware operations just explained and of the beginning of the machine check microprogram is shown in Diagram 5-19. The microprogram starts by gating the interruption code to S(16–31) to begin forming the old PSW in ST. This is done by setting S(16–31) to all 0's, because the machine check interruption code, PSW(16–31), is always 0. The 'block I-Fetch' trigger and STAT H are reset. (STAT H is reset to modify the subsequent common interruption routine for specific machine check actions.) At this point, the machine check interruption microprogram enters the common interruption routine.

Program Interruption

- Initiated by 'program interrupt' latch.
- Value in interrupt code triggers is transferred to S(16–31) by 'priority 1' latch.
- Sets STAT H, enters common interruption routine.

A program interruption results from improper conditions arising during the processing of data or instructions. Generally, these improper conditions can be described as errors in programming. When any of these conditions are detected, they cause a value to be placed into the four Interrupt Code triggers. The value inserted reflects the condition responsible for the interruption. The conditions that cause a program interruption and their corresponding Interrupt Code trigger settings are shown in FEMDM Diagrams 5-20, 5-21, and 5-22.

Once any Interrupt Code trigger has been set, the 'program interrupt' latch shown in Diagram 5-22 is set. (The 'INTRP X-branch' micro-order samples this latch to modify the execution sequence of instructions and the I-Fetch of SS format instructions. Once this latch is set, further program violations are lost with the exception of a "late" protection check.) This latch establishes priority for the program interruption by blocking the priority circuits of lower-priority interruptions and exceptional conditions.

At end op, the output of the 'program interrupt' latch is gated to force D to 28 (hex), to set the 'priority 1' latch, and, if an Execute instruction is not in progress, to set the 'block I-Fetch' trigger. The 'EXCEP' micro-order in the first I-Fetch cycle detects program interruption priority and forces address 00A (hex) into ROSAR. This ROSAR address causes a branch to an interruption microprogram to store the old PSW per D. The 'priority 1' latch causes the values in the Interrupt Code triggers (program-interruption code) to be gated to S as part of the old PSW. Subsequently, the microprogram loads the new PSW per the address in D + 40 (hex). CPU operation then proceeds as dictated by the new PSW.

Diagram 5-22 is a flowchart of the hardware operations just described and of the beginning of the program interruption microprogram. The microprogram starts by gating the interruption code from PSW(16–31) to S(16–31). The 'block I-Fetch' trigger and STAT D are reset, and STAT H is set. (STAT H modifies the subsequent common interruption routine for specific program interruption actions.) At this point, the program interruption microprogram enters the common interruption routine.

Supervisor Call Interruption

- Initiated by 'supervisor call' trigger, which is set by preceding Supervisor Call instruction.
- E(8–15), which contains interruption code, is transferred to S(24–31).
- Sets STAT H, enters common interruption routine.

The supervisor call interruption results from execution of the Supervisor Call instruction. Its basic purpose is to initiate a branch to the supervisor program. When the priority of the interruption is established, an address is forced into ROSAR and into D. The address in ROSAR causes the operation to branch to a microprogram which stores the old PSW in the address forced into D and fetches a new PSW from the address in D + 40 (hex). This new PSW places the CPU into the Supervisor state.

Diagram 5-23, FEMDM, shows how the '1→ INTREQ-TGR' micro-order tests for a Supervisor Call instruction and sets the 'supervisor call' trigger. This trigger is reset if the 'interrupt code 4' trigger is set. Because all program interruptions would have been handled before executing the Supervisor Call instruction, the 'interrupt code 4' trigger can be set now only by a "late" protection check. Therefore, performance of the supervisor call interruption is suppressed and a program interruption occurs in its place. If the 'interrupt code 4' trigger is not set, then the 'supervisor call' trigger is not reset.

The 'supervisor call' trigger gates E(8–15) to S(24–31) to begin assembling the old PSW. (The 'supervisor call' trigger also sets the 'priority 1' latch; because the supervisor call and program interruptions cannot be pending at the same time, no conflict results from both setting an interruption priority code of 01.)

Diagram 5-23 is a flowchart of the hardware operations just described and of the beginning of the supervisor call interruption microprogram. The microprogram starts by gating the interruption code from PSW(16–31) to S(16–31). The 'block I-Fetch' trigger and STAT D are reset, and STAT H is set. At this point, the supervisor call interruption microprogram enters the common interruption microprogram routine.

External Interruption

- Remains pending if external mask bit, PSW(7), is not set.
- Initiated by setting of 'time clock at limit' latch, depression of INTERRUPT pushbutton, or recognition of 'external signal in' bus.
- 'Time clock at limit', 'console signal', and 'external signal-2' through '-7' triggers are transferred to S(24–31).
- Sets STAT H, enters common interruption routine.

An external interruption is caused by one of the following if the external bit of the PSW system mask is a 1:

1. The set state of the 'time clock at limit' latch (refer to "Timer Exceptional Condition").
2. The depression of the INTERRUPT pushbutton on the system control panel.
3. The recognition of any signal on the 'external signal in' bus of the Direct Control feature. The external interruption circuits and a flowchart of the initiation of the external interruption microprogram are shown in Diagram 5-24, FEMDM.

Once priority for the external interruption is established during end op, D is forced to 28 (hex), the 'priority 2' trigger is set, and, if an Execute instruction is not in progress, the 'block I-Fetch' trigger is set. The 'EXCEP' micro-order in the first I-Fetch cycle detects external interruption priority and forces address 006 (hex) into ROSAR. The ROSAR address causes a branch to an interruption microprogram to store the old PSW per D. The 'priority 2' trigger causes the contents of the eight signal triggers (the interruption code) to be gated to S(24–31) as part of the old PSW. Subsequently, the microprogram loads the new PSW per the address in D + 40 (hex). CPU operation then proceeds as dictated by the new PSW.

The external interruption microprogram starts by gating status of the 'time clock at limit', 'console signal' and External Signal triggers to S(24–31). S(16–23) is reset to zero, and correct (odd) parity is assigned to S(16–31). The 'block I-Fetch' trigger and STAT D are reset, and STAT H is set. (STAT H is set so that the common interruption routine skips micro-orders pertaining to the machine check interruption.) At this point the microprogram enters the common interruption routine.

I/O Interruption

- Remains pending if associated channel mask bit, in PSW(0–7), is not set.
- Channel 0 has highest interruption priority, followed in order by channels 1–6.
- 3-bit channel address and 8-bit unit address are transferred to S(21–31).
- Sets STAT H, enters common interruption routine.

An I/O interruption results from the reception of a simplexed 'interruption request' signal from a channel if the system mask bit for that channel is a 1. All I/O channels in the system compete for the interruption priority. The multiplexer channel (channel 0) is assigned the highest priority followed in order by selector channels 1 through 6 (maximum). The I/O interruption circuits and a flowchart of the initiation of the I/O interruption microprogram are shown in Diagram 5-25, FEMDM. The 'channel X interrupt request' triggers are set at the start of every non-end-op cycle. At the start of end op, the highest-priority trigger that is set resets all the others.

Once priority for an I/O interruption is established during end op, D is forced to 38 (hex), the 'priority 1' and 'priority 2' triggers are set, and, if an Execute instruction is not in progress, the 'block I-Fetch' trigger is set. The 'EXCEP' micro-order in the first I-Fetch cycle detects I/O interruption priority and forces address 006 (hex) into ROSAR. This ROSAR address causes a branch to an interruption microprogram to store the old PSW per D. The 'priority 1' and 'priority 2' triggers (which have already caused all but the highest-priority interrupt request trigger to be reset) causes the interruption code (the 3-bit channel address and the 8-bit unit address) to be gated to S(21–31) as part of the old PSW. Subsequently, the microprogram loads the new PSW per the address in D + 40 (hex). CPU operation then proceeds as dictated by the new PSW.

The I/O interruption microprogram starts by setting the 'timing gate' trigger, thus gating a response back to the highest-priority channel to reset the interruption request controls. While waiting for the channel to return a 'release' signal, the 'block I-Fetch' trigger is reset and the interruption code is gated to S(16–31). When the microprogram finds the 'release CPU' latch set, signifying that the 'release' signal from channel has been received, the 'timing gate' trigger is reset. STAT H is set so that the common interruption routine, which follows next, skips micro-orders pertaining to the machine check interruption.

Common Interruption Routine

- Program status is assembled in ST and is then stored into old PSW location per interruption cause.
- System is reset if STAT H is reset (machine check interruption).
- Applicable new PSW is fetched per D.
- Processing resumes after new instructions have been fetched and placed into Q.

The common interruption routine (Diagram 5-26, FEMDM) stores the old PSW into main storage and loads a new PSW into the CPU. This routine is entered by all five interruption microprograms.

The IC is reduced by 8 or 16 to reflect the doubleword address of the instruction that caused the interruption. This address is placed into T(40–63) as part of the old PSW. Next, the contents of the PSW register are gated to S(0–15) and T(34–39). E(0,1) is gated to T(32,33) unless the 'instruction length not available' trigger is set, indicating a program interruption is in progress because of a "late" protection check. For that case only, the instruction-length code is set to 0. This action completes the old PSW transfer to ST. The routine inhibits storage protection, sets Marks 0–7, and initiates a 4-cycle storage request to store the old PSW per the D address. An interruption reset clears the CPU of the condition which initiated the access to the interruption microprogram. The 'invalid branch' trigger, the 'invalid instruction address' trigger, and STAT G are reset.

At this point, the common interruption routine checks STAT H to see what class of interruption initiated the operation. If STAT H is reset, the operation is due to a machine check interruption, and the CPU is placed in the scan mode. Three no-op cycles are taken to allow the CPU and main storage to become quiescent. Then a 'system reset' signal clears all control triggers. The 'scan mode' trigger is reset, and the routine prepares to load the new PSW.

STAT H is set if the initiating interruption is other than a machine check. In this case, the CPU is not placed in the scan mode and the system is not reset.

To generate the address for the new PSW, 10 (hex) is placed into B, setting B(59) to 1. Next, the value in B is shifted left twice and gated to PAB as an effective value of 40 (hex). Simultaneously, the old PSW address is gated from D to PAA. The sum, the address of the new PSW, is gated from PAL to D. Storage protection is then inhibited, and a 3-cycle fetch per D is initiated.

The interruption microprogram has now finished the common interruption routine and enters the Load PSW microprogram (Diagram 5-601, FEMDM). When received from main storage, the new PSW is loaded into ST. Because the new PSW will require fetching of instructions from a new storage location, the 'I-Fetch invalid address' trigger is set to enable recording of any invalid address that may result on the subsequent fetch. Portions of the new PSW are loaded into the PSW register and into the IC and D. A 3-cycle storage request per the IC is initiated, and the IC is incremented by 8. At this point, the program shifts to a common branch microprogram. The instruction address in D is incremented by 8 and transferred to the IC in anticipation of a branch instruction. When the first doubleword arrives from main storage it is loaded into Q, and the op-code halfword of the first instruction is transferred to R. D(21,22) is sampled to see if Q needs refilling; if so, a second request per the IC is initiated. If Q does not need refilling, the program generates an end op.

Stop, Wait, and Repeat Exceptional Conditions

The stop exceptional condition is caused by (1)depressing STOP, (2) detecting an address-compare condition when ADDRESS COMPARE STOP is in the stop position, and (3) operating at the instruction-step rate. (If the Multisystem feature is installed, an external-start operation also causes a stop exceptional condition.) The wait exceptional condition is caused by the wait mask bit, PSW(14), being set to a 1. The repeat exceptional condition is caused by activating the REPEAT INSN switch. The scheme for detecting a stop, wait, or repeat instruction exceptional condition is shown in Diagram 5-27, FEMDM.

When any one of these exceptional conditions has priority during end op and an Execute instruction is not in progress, the 'block I-Fetch' trigger is set. During the next cycle, the first I-Fetch cycle, the 'EXCEP' micro-order forces an address into ROSAR: 026, 02A, and 028 (hex) for the stop, wait, and repeat exceptional condition, respectively. This address causes a functional branch to a loop microprogram. Because each of these exceptional conditions may be caused by manual intervention, their microprograms are discussed in Chapter 6, Section 1.

Program Store Compare Exceptional Condition

- Instruction refetch is performed when 'PSC' trigger is set by ASC test or Execute instruction.
- Refetch routine decrements instruction address by 8 or 16 and issues request to refill Q.

An instruction refetch routine is performed if the instructions previously obtained by the CPU must be refetched from main storage. The need to refetch instructions occurs when:

1. The CPU stores data at the main storage address that corresponds to the address currently specified by the IC. In this case, the instructions presently contained in Q may have been modified by the store operation and, therefore, must be refetched.
2. The CPU completes the subject instruction of an Execute instruction and must resume processing the instructions previously contained in Q.

An instruction refetch is initiated by the 'PSC' trigger which is set for either of the above cases. In the first case, the 'PSC' trigger is set as described under "Address Store Compare (ASC) Test." In the second case, the 'PSC' trigger is set by the Execute instruction. The scheme for detecting a program store compare exceptional condition and the instruction refetch microprogram flowchart are shown in Diagram 5-28, FEMDM.

A need to refetch instructions is treated as an exceptional condition by the CPU. When this condition is detected, the 'block I-Fetch' trigger changes the normal I-Fetch routine, and a branch to an instruction refetch microprogram is performed by the first I-Fetch word.

The first ROS word in the refetch microprogram resets the 'block I-Fetch' trigger so that normal I-Fetch can be resumed after Q is refilled. This word also establishes whether the address currently specified in the IC is one or two doublewords ahead of the current instruction. The address in the IC is always at least one doubleword ahead of the address for the instructions in Q. If Q was not refilled before the refetch routine, the IC is one doubleword (8 bytes) ahead of the current instruction; if Q was just refilled, IC is two doublewords or 16 bytes ahead.

IC(21,22) indicates whether Q was refilled before the refetch routine. If IC(21,22) is not set to 11, a request to refill Q (if generated) was blocked by the exceptional condition in progress (i.e., the need for instruction refetch) and the IC is 8 bytes ahead of the current instruction. If, however, IC(21,22) = 11, the need for an instruction refetch occurred after Q was refilled; IC(20) has been incremented, and the IC is 16 bytes ahead. Accordingly, the second ROS word in the refetch microprogram subtracts 8 or 16 from the IC and issues a 3-cycle request per the decremented address. This word also resets the 'PSC' and 'execute in progress' triggers and then causes the Load PSW microprogram to be entered (as shown in Diagram 5-601, FEMDM). Entry corresponds to a point after the new PSW has been loaded but before the successful branch routine. IC is incremented by 8, the next instruction is transferred to R, and Q is refilled, if necessary, before completing the program store compare exceptional condition microprogram with an end op.

Invalid Instruction Address Test Exceptional Condition

- Determines interrupt code triggers to be set if program check was detected while addressing instruction.

When addressing instructions in main storage, three requirements must be met:

1. Because instructions are specified on a 2-byte basis, the least significant bit of the instruction address must always be a zero. Failure to meet this requirement results in a specification program interruption.
2. The instruction address cannot exceed the storage capacity used with a given installation. (The sizes of main storage available for the 2065 CPU are listed in Chapter 1, Section 1.) In addition, the storage unit containing the instruction must be available to the CPU. An attempt by the CPU to execute an instruction from an unavailable or non-existent location results in an addressing program interruption.
3. The instruction address cannot specify an area in main storage that is fetch-protected. An attempt by the CPU to execute instructions from a fetch-protected location results in a protection program interruption.

If any of these three requirements is not met, the CPU hardware forces a new address into ROSAR. The microprogram accessed by this address sets the appropriate program interruption code (specification, addressing, or protection) into the CPU. This microprogram is then followed by the program interruption microprogram previously described. The following paragraphs describe the methods used to detect each violation and to set the appropriate interrupt code triggers.

Specification Detection

All storage requests for instructions do not result in Q being refilled. For example, end-op requests are ignored if the 'block I-Fetch' trigger is set. Also, branch requests made during I-Fetch are ignored if the conditions for a successful branch are not found met during the following execution (non-branch on condition instructions only). For this reason, the least significant address bit of the instruction, IC(23) [or D(23) if preceded by the 'BEOP' micro-order], is detected at the start of I-Fetch. However, because the specification interruption code is not yet set, the program interruption microprogram cannot be immediately entered. Instead, the invalid instruction address test exceptional condition microprogram is entered after processing all interruptions and higher-priority exceptional conditions. If, however, during this forced microprogram, an invalid or fetch-protected address is requested, the specification interruption code is not set because, in either case, the address is outside of "fetchable" storage.

Invalid Address Detection

- 'I-Fetch request' trigger prevents setting of addressing interruption code while refilling Q.
- 'I-Fetch invalid address' trigger indicates IC request is invalid.
- 'Branch invalid address' trigger indicates branch address of successful branch instruction is invalid.

Following a request to refill Q, the IC is incremented by 8 to obtain the instruction address for the next request. The scheme of incrementing the IC ahead of time allows greater speed in requesting instructions from main storage. However, with the IC one doubleword ahead of the instructions in Q, a unique case occurs if the instructions in Q are obtained from the last available location in main storage. In this case, the incremented IC specifies an invalid address; i.e., an address that is in excess of the main storage capacity. Because the Q-register refill routine is initiated before the CPU runs out of instructions, a request per the IC refills Q with instructions from an invalid address.† Even though Q contains invalid instructions, an addressing program interruption must not occur until the CPU attempts to process these instructions. This condition arises because the last valid instruction being processed by the CPU may result in a successful branch to a valid storage location.

A similar situation may occur following an unsuccessful branch instruction that specifies a branch to the last available main storage location. Excluding the Branch on Condition instructions, the CPU assumes that the branch instruction is successful and, accordingly, issues a request per D. Following the request, D is updated by 8 and specifies an invalid address. In this case, an addressing program interruption must not occur because, upon establishing that the branch is unsuccessful, the CPU resumes normal addressing per the IC. For branch instructions, an addressing interruption must occur only when the address specified by a successful branch is above the available main storage capacity. This situation may also exist after any load-PSW operation or after the program store compare exceptional condition.

An invalid-address test is performed each time the CPU issues a request to refill Q. Because a request for invalid instructions will not necessarily cause an interruption, setting of the interrupt code triggers must be blocked while Q is being refilled.† The scheme used for detecting a "true" invalid instruction address error is shown in Figure 3-4.

The 'I-Fetch request' trigger prevents the invalid-address condition from causing an interruption while Q is being refilled. This trigger is set when a need to refill Q is detected; depending on the current instruction status in the CPU, the trigger is set as follows:

1. For non-branch 1- and 2-halfword instructions, the trigger is set by the I-Fetch sequencers.
2. For branch instructions, the trigger is set by the '1→INST-MSREQ' micro-order given at the start of the branch execution. Note that if an unsuccessful Branch on Condition instruction occurs and IC(21,22) = 00, the 'I-Fetch request' trigger is not set because Q will not be refilled.
3. For SS instructions, the trigger is set if IC(21), or D(21) for the 'BEOP' micro-order, is equal to 0 during end op. This condition indicates that the complete SS instruction is already in Q and succeeding instructions are being requested.

The output of the 'I-Fetch request' trigger prevents the interrupt code triggers from being set by the 'invalid address' signal from the BCU. In addition, the output of the 'I-Fetch request' trigger serves as one of the conditioning inputs for the 'I-Fetch invalid address' and 'branch invalid address' triggers. One of these triggers is set whenever the BCU indicates that the address of the storage request exceeds the main storage capacity. The 'invalid address' signal sets the 'I-Fetch invalid address' trigger if the invalid address is due to the CPU fetching ahead. Conversely, this signal sets the 'branch invalid address' trigger when the invalid address is the result of a successful branch instruction.

The 'gate I-Fetch invalid address' trigger dictates whether the 'I-Fetch invalid address' or 'branch invalid address' trigger is to be set. When set, this trigger conditions the 'I-Fetch invalid address' trigger; when reset, the 'branch invalid address' trigger. Depending on the current instruction status in the CPU, the 'gate I-Fetch invalid address' trigger is set as follows:

1. For non-branch 1- and 2-halfword instructions, the trigger is set by the I-Fetch sequencers.
2. For SS instructions, the trigger is set by the 'IF-INV→TGR' micro-order given at the start of the SS I-Fetch routine; i.e., the presence of the 'IF-INV→TGR' micro-order and the absence of the

†The instruction address is considered invalid by the BCU upon detection of a carry from the most significant bit position in the IC. This bit position is defined by the size of the main storage in the particular installation.

†An exception to this rule occurs if the last one or two halfwords of an SS instruction are requested from an invalid address while the first halfword is contained in a valid storage location. In this case, the entire SS instruction is considered to have an invalid address and, because the CPU has attempted to process the instruction, the interrupt code triggers are set as soon as the request is generated.

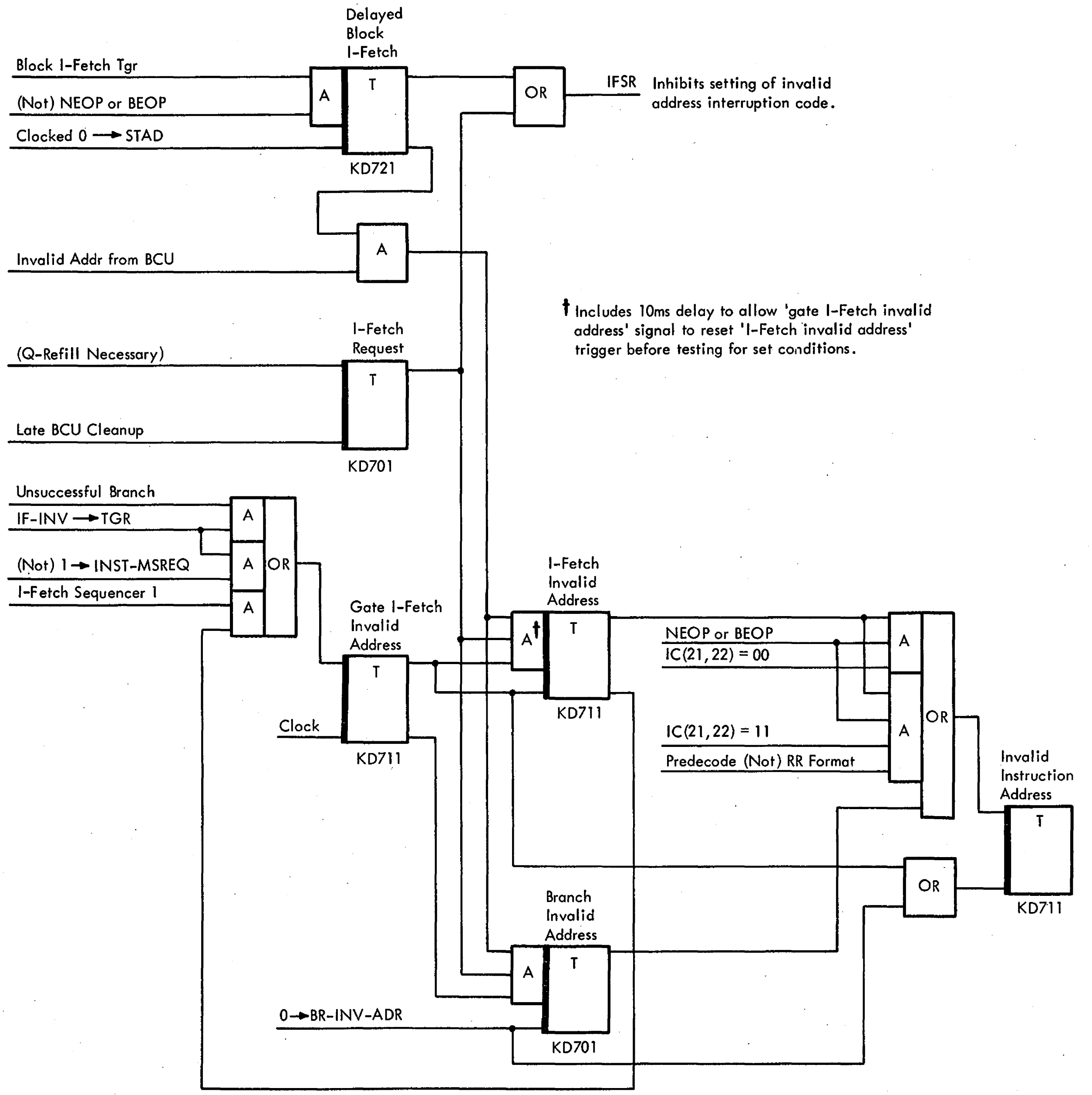


Figure 3-4. Detection of Invalid Instruction Address

'1→INST-MRSEQ' micro-order when the complete SS instruction is in Q.

3. For unsuccessful branch instructions (except Branch on Condition), the trigger is set by the presence of the 'IF-INV→TGR' micro-order and the absence of the '1→INST-MSREQ' micro-order; for an unsuccessful Branch on Condition instruction, the trigger is set by the simultaneous presence of the '1→INST-MSREQ' and 'IF-INV→TGR' micro-orders at the start of execution. (The various branching conditions that may arise are described in Section 6 of this Chapter.)

When the 'gate I-Fetch invalid address' trigger is set, the 'invalid address' signal is allowed to set the 'I-Fetch invalid address' trigger, indicating that Q has been refilled with instructions from an invalid address. However, because R may still contain a valid RR instruction, further testing is required to establish that a true interruption condition exists. The setting of IC(21,22) during end op indicates whether a valid or an invalid instruction is contained in R. If IC(21,22) = 11 and an RR instruction is predecoded, R contains a valid instruction. When IC(21,22) = 11 but the instruction is not of the RR format, the balance of the instruction has been obtained from an invalid location and the 'invalid instruction address' trigger is set. If IC(21,22) = 00, the 'invalid instruction address' trigger is set regardless of the instruction format; this condition indicates that R contains the first halfword of an invalid instruction.

When the 'gate I-Fetch invalid address' trigger is *not* set, the 'invalid address' signal sets the 'branch invalid address' trigger. Because, in this case, the invalid address is the result of a successful branch instruction, the 'invalid instruction address' trigger is set without further testing being necessary.

Fetch Protection Detection

- 'Delayed I-Fetch storage request' trigger prevents setting of protection interruption code while Q is being refilled.
- 'Delayed I-Fetch protect gate' trigger is set if request is due to normal sequencing.
- 'Protected branch address' trigger is set if branch is made to protected location.

The CPU cannot execute instructions from a fetch-protected area in main storage. Because the IC is always one doubleword ahead of the instructions in Q, a unique situation occurs if the instructions in Q are obtained from a main storage location adjacent to a protected area. In this case, the incremented IC specifies a protected address and, because the Q-register refill routine is initiated before the CPU runs out of instructions, the request per the IC refills Q with instructions from a protected address. A protection interruption, however, does not occur until the CPU attempts to execute the protected instructions. This condition arises because the last valid instruction being processed by the CPU may result in a successful branch to a valid storage location.

A protection test is performed each time the CPU issues a request to refill Q. Because a request for protected instructions will not necessarily cause an interruption, the setting of the 'protection check (to CPU)' and 'instruction length not available' triggers is blocked while Q is being refilled. The scheme for detecting a "true" protection violation, shown in simplified form in Figure 3-5, is closely related to the invalid addressing detection scheme previously described. The major difference between the invalid-addressing and fetch-protection schemes is in timing: the 'invalid address' signal arrives at the CPU 1 cycle after the request, while the 'protection check' signal arrives 2 cycles after the request. For this reason a separate circuit is used for detecting a protection violation.

The 'delayed I-Fetch storage request' trigger prevents the 'protection check' signal from causing an interruption while Q is being refilled. This trigger is set one cycle after the 'I-Fetch request' trigger is set by a request to refill Q. The output of the 'delayed I-Fetch storage request' trigger serves as one of the conditioning inputs for the 'delayed I-Fetch protect gate' trigger. This trigger is set if the current storage request is due to a nonbranch or unsuccessful branch request and is set by the same conditions that set the 'gate I-Fetch invalid address' trigger. When the 'delayed I-Fetch storage request' trigger and the 'delayed I-Fetch protect gate' trigger are both set, the 'protection check' signal sets the 'I-Fetch invalid address' trigger. The action at this point is identical to that described for detection of invalid addressing; i.e., a test is made to establish if the CPU has attempted to execute instructions from a protected area, and the 'invalid instruction address' trigger is set if a protect violation has occurred.

If the request to refill Q is a result of a branch instruction (as indicated by the reset state of the 'gate I-Fetch invalid address' trigger), the 'delayed I-Fetch protect gate' trigger is not set. In this case, the 'protection check' signal sets the 'protected branch address' trigger, which in turn sets the 'invalid instruction address' trigger.

Invalid Instruction Address Microprogram

- Issues second request for instruction.
- Interrupt code triggers are set per highest-priority error detected: (1) addressing = 101, (2) protection = 100, (3) specification = 110.

The detection of a specification, addressing, or protection exception and the associated microprogram are shown in Diagram 5-29, FEMDM. Because all three exceptions access the same microprogram, the microprogram must re-establish the nature of the exception to set the proper interruption code. To establish which exception is currently in effect, the microprogram issues a second request for instructions. This time, however, the setting of the interrupt code triggers is not blocked; i.e., the appropriate interruption code is set immediately upon detection of a specification, addressing, or protection exception.

Before generating a second request, the IC must be decremented to the address that caused the exception. The status of IC(21,22) indicates whether the current IC count is one or two doublewords ahead of the required address: if IC(21,22) is not set to 00, the IC is one doubleword ahead; if IC(21,22) = 00, the IC is two doublewords ahead. (The other recovery microprograms test IC(21,22) for a setting of 11; because the invalid instruction address test exceptional condition does not set the 'block I-Fetch' trigger, IC(21,22) is updated and tested for a setting of 00.)

Accordingly, the microprogram subtracts 8 or 16, decimal, from the IC, and loads the decremented address into D. The STC is then set to zero and a 3-cycle request is issued per D. After the request, the STC is incremented

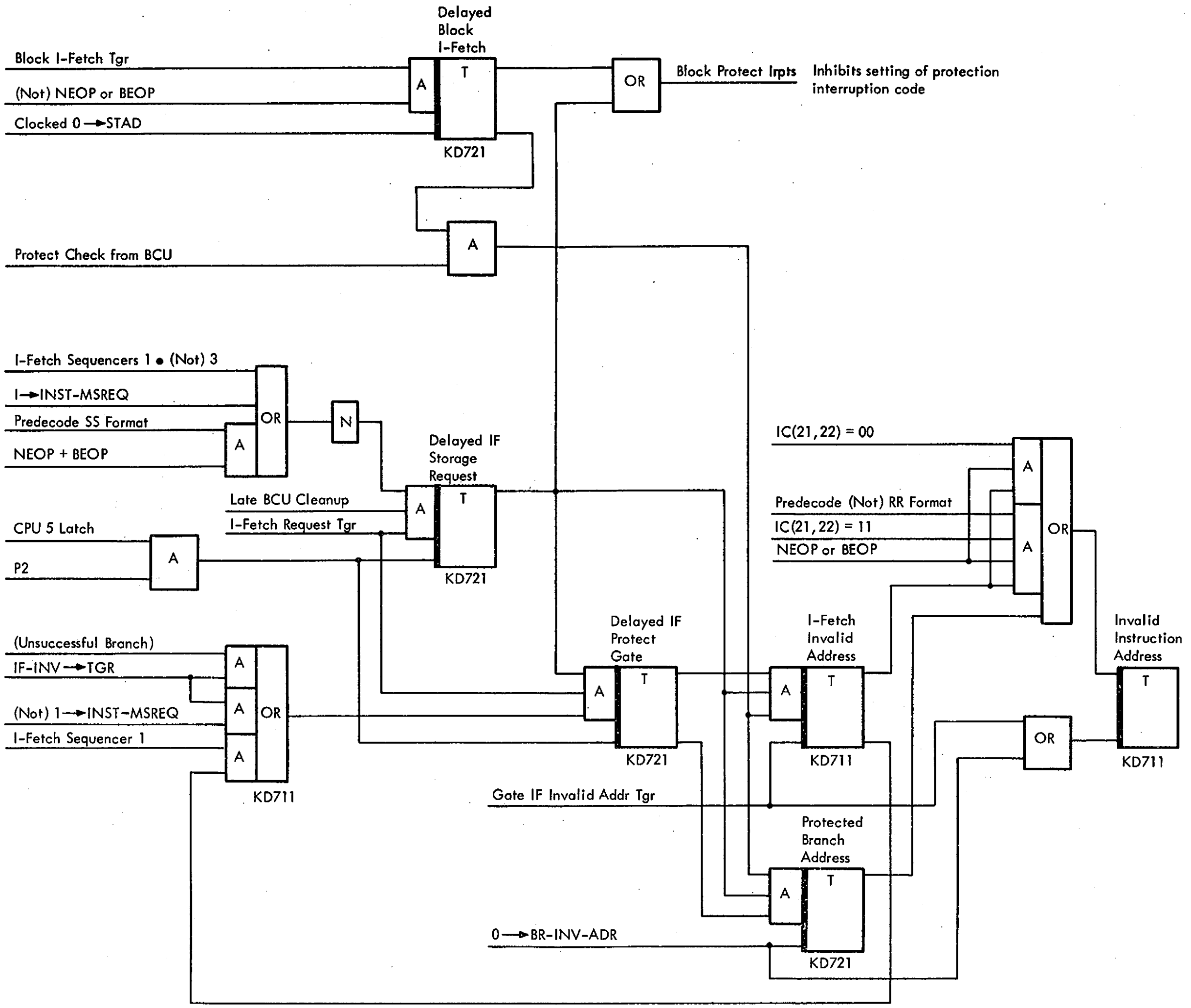


Figure 3-5. Detection of Fetch-Protected Instruction Address

once during each subsequent cycle to provide the required wait interval between the request and what would be the cycle for transferring the instructions to Q. The appropriate interrupt code trigger(s) is then set upon receipt of a specification, addressing, or protection exception. Because the low-order bits of the addressing interruption code equal 101, there is no need to block the protection interruption code (100) from also being set if it is received. However, if either of these conditions is set, the setting of the specification interruption code, 110, is blocked. The specification interruption code need only be set if IC(23) = 1; because the invalid instruction address test exceptional condition does not set the 'block I-Fetch' trigger, IC(23) = 1 if D(23) equalled 1 during the preceding branch end op.

The invalid instruction address test exceptional condition microprogram terminates with a normal end op. If no higher-priority exceptional condition or interruption is detected, the program interruption microprogram is entered next.

Q-Register Refill Exceptional Condition

- One extra I-Fetch cycle is performed to allow refilling of Q without conflicting with execution sequence of next instruction.
- I-Fetch sequencers are always activated.

Following each storage request for instructions, the IC is incremented by 8 to obtain the address from which instructions will be fetched by the next request. This updating is accomplished by gating the contents of the IC to the parallel adder, adding a 1 to IC(20), and gating the incremented address back to the IC. The need to update the IC usually adds another cycle to the I-Fetch of RR, RX, RS, and SI instructions. There are two reasons for this extra cycle:

1. The BCU requires that each main storage address be retained for at least two cycles. Therefore, main storage requests during the first I-Fetch cycle would interfere with end-op requests.

2. Because the parallel adder is used to increment the IC, and the first cycle in the execution phase may also require the use of the parallel adder, the execution phase must be delayed until the IC is incremented.

The case when I-Fetch requires a second cycle (third if an indexed RX instruction) is treated as an exceptional condition in the CPU. The 'EXCEP' micro-order in the first ROS word of the I-Fetch microprogram overrides the functional branch micro-order per the instruction op code [E(02–07)→ROA] and forces a new address into ROSAR. This forced address is determined by the format of the upcoming instruction and the status of the I-Fetch sequencers (Diagram 5-30, FEMDM). Operation of the I-Fetch sequencers is initiated at end op when the need to refill Q exists and the next instruction to be executed is not in the SS format. If the request was generated two cycles before end op, sequencer 2 is latched at the start of I-Fetch; otherwise, sequencer 1 is being set.

At the start of I-Fetch, The 'EXCEP' micro-order samples sequencer 1 to see if it is being set. If it is, a new address is always forced into ROSAR, causing one extra I-Fetch word to be added to the basic I-Fetch.

If sequencer 2 is found latched, the parallel adder is available for use on the next cycle because IC(20) has already been incremented. I-Fetch of RR and shift instructions does not require the fetching of an operand from main storage; also, storage operands for indexed RX instructions are not requested until the second basic I-Fetch cycle. For these reasons, sequencer 2 forces a new ROSAR address only if an RR, indexed RX, or shift instruction is *not* being fetched.

The forced ROSAR addresses as a result of the Q-register refill exceptional condition are shown in Table 3-1. The SS format is included for completeness. However, because the I-Fetch sequencers are not activated, a Q-register refill exceptional condition is never detected during the SS I-Fetch microprogram. Instead, the functions of the sequencers are initiated by micro-orders in that microprogram.

Two-Cycle RR I-Fetch

The actions of the first I-Fetch cycle are unchanged except for the overriding of the 'E(02–07)→ROA' micro-order by the 'EXCEP' micro-order (Diagram 5-6). During

Table 3-1. Q-Register Refill Exceptional Conditions

		Request Issued During Preceding:		Forced ROSAR Address (Hex)
Instruction Being Fetched	IC(21,22) at End Op	EEOP (2 Cycles Early)	NEOP, BEOP, or EEOP (1 Cycle Early)	
RR	00, 01, or 11	Never	Never	None
	10	Yes	Never	None
	10	No	Yes	030
RX, RS, or SI	00 or 11	Never	Never	None
Indexed RX, or shift RS†	01 or 10	Yes	Never	None
Indexed RX	01 or 10	No	Yes	03A
Non-indexed RX	01 or 10	Yes	Never	022
Non-indexed RX	01 or 10	No	Yes	032
Shift RS†	01 or 10	No	Yes	020
Non-shift RS, or SI	01 or 10	Yes	Never	024
Non-shift RS, or SI	01 or 10	No	Yes	034
SS	11	Never	Never	None
	00, 01, or 10	Never	Yes	None

†All shift instructions are of the RS format with an op code of 1000 1XXX.

the second RR I-Fetch cycle, sequencer 2 is set and sequencer 1 is reset. This status increases the IC by 8 and returns the new instruction address to the IC at the start of the next cycle, thus completing the updating of IC(20). During the next cycle (first execution cycle), sequencers 3 and 1 are both set by the 'RASCR' micro-order in the forced word. This condition indicates that new instructions are to be gated to Q at the start of the second execution cycle. The major registers and timing applicable to this sequence are shown in Diagram 5-8, FEMDM.

Forced-Cycle RX I-Fetch

If the request to refill Q was not issued 2 cycles before end op, the actions of the second RX I-Fetch cycle include the same actions as the second RR I-Fetch cycle. Otherwise, IC(20) has already been incremented and sequencers 1 and 3 are automatically set during the second cycle. If the RX instruction is indexed, then a second forced word is now performed; this word contains the same micro-orders as the second word of the basic RX I-Fetch. Otherwise, the first forced word completes the I-Fetch routine after issuing any request inhibited during the first I-Fetch cycle. This action is performed by the 'MS-REQ*D-3' micro-order. D is also transferred to PAL if the request for new instructions was issued 2 cycles before end op. Because in this case sequencer 2 is reset during the first execution cycle, a 'SPEC' micro-order in the first execution word tests the storage address from PAL, not D. The new instructions are gated into Q [and the next op-code word to R from Q(0–15) if IC(21,22) = 10 at end op] at the start of the first execution cycle (second execution cycle if the request was not made two cycles early and the RX instruction is not indexed). The major registers and timing applicable to the non-indexed case are shown in Diagram 5-12, FEMDM.

Two-Cycle RS and SI I-Fetch

The major registers and timing applicable to the 2-cycle RS and SI I-Fetch are the same as that for the 2-cycle, non-indexed RX I-Fetch (Diagram 5-12). If the request to refill Q was not issued two cycles before end op, the actions of the second I-Fetch cycle include the same actions as the second RR I-Fetch cycle. Otherwise, IC(20) has already been incremented and sequencers 1 and 3 are automatically set during this forced cycle. This action results in the refill of Q at the start of the next cycle (first execution cycle). Whichever cycle Q is refilled, Q(0–15) is also transferred to R if IC(21,22) was set to 10 during the preceding end op. This transferring of the op-code halfword is otherwise performed by an appropriate micro-order in the first I-Fetch word. If an MVI, STM, TS, I/O, or shift instruction is being fetched, no further I-Fetch actions are necessary. However, fetching of other RS or SI instructions causes the storage request omitted by the first I-Fetch cycle to be issued now. This request is performed by the 'MS-REQ*D-3' micro-order. Also, D is transferred to PAL if the Q-register refill request was issued two cycles before end op. Because in this case sequencer 2 is reset during the first execution cycle, any 'SPEC' micro-order tests the storage address from PAL, not D.

It was previously stated that the last I-Fetch word always includes the 'E(02–07)→ROA' micro-order. However, there is one exception to this statement: the forced I-Fetch cycle for shift instructions includes the 'E(04–07)→ROA' micro-order, which, in turn, forces the first execution cycle to branch to the second cycle per D rather than per PAL.

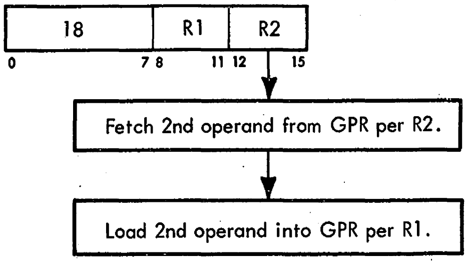
This section discusses the 35 instructions of the fixed-point instruction set. These instructions use the RR, RX, and RS formats. Positive fixed-point numbers are expressed in true binary form, whereas negative numbers are expressed in complement binary form (2's complement form). One operand is always in 1 of the 16 GPR's; the other operand may be in either a GPR or in main storage. For a discussion of number representation, data formats, operand addressing, instruction formats, data flow, program interruptions, and condition codes, see Chapter 1.

LOAD

The fixed-point load instructions provide a means of loading operands into the LS GPR's. The load operation may be register-to-register (RR format) or storage-to-register (RX and RS formats). In any case, the instruction loads the second operand into the first operand location, and the second operand location remains unchanged. In addition, certain load instructions can test the second operand before loading it and can load the second operand in complement, positive, or negative form.

Load, LR (18)

- Load 2nd operand (in GPR per R2) into 1st operand location (in GPR per R1).
- RR format:

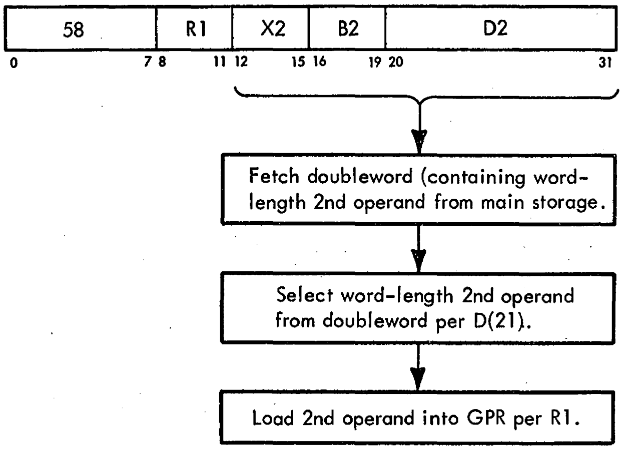

18
R1
R2
0
7 8
11 12
15
Fetch 2nd operand from GPR per R2.
Load 2nd operand into GPR per R1.

- Conditions at start of execution:
  Instruction is in E.
  1st operand is in A, B, and D (not used).
  2nd operand is in S and T.

The Load, LR, instruction loads the second operand from the GPR per R2 into the GPR per R1. At the start of execution, the word-length second operand is in S and T. Because both operands are in GPR's, no specification test is performed. The second operand is loaded into the GPR specified by R1, and an end-op cycle is taken.

Load, L (58)

- Load 2nd operand (in storage) into 1st operand location (in GPR per R1).
- RX format:

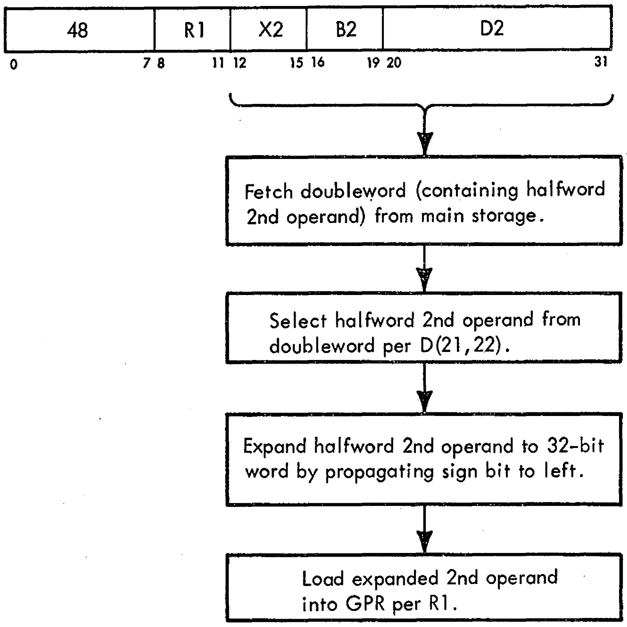

58
R1
X2
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch doubleword (containing word-length 2nd operand from main storage.
Select word-length 2nd operand from doubleword per D(21).
Load 2nd operand into GPR per R1.

- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T (not used).
  2nd operand address is in D.
  Main storage request for 2nd operand has been issued per D.
- D(21) determines which word of doubleword is to be stored: if a 1, right word; if a 0, left word.

The Load, L, instruction loads the second operand from main storage into the GPR per R1. Instruction execution starts with a specification test. If a program specification interruption occurs, an end op is forced and the instruction is suppressed. If no specification check exists, D(21) is tested to determine which word of the doubleword fetched from main storage will be gated from the SDBO to T. If D(21) = 1, the right word is gated; if D(21) = 0, the left word is gated. The contents of T are then loaded into the GPR specified by R1, and an end-op cycle is taken.

Load Halfword, LH (48)

- Load halfword 2nd operand (in storage) into 1st operand location (in GPR per R1).
- RX format:


48
R1
X2
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch doubleword (containing halfword 2nd operand) from main storage.
Select halfword 2nd operand from doubleword per D(21,22).
Expand halfword 2nd operand to 32-bit word by propagating sign bit to left.
Load expanded 2nd operand into GPR per R1.

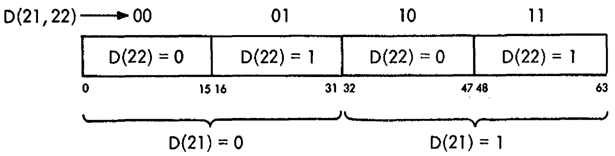
- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T (not used).
  2nd operand address is in D.
  Main storage request for 2nd operand has been issued per D.
- Halfword operand is expanded to word by propagating sign of halfword into 16 high-order bits of word-length register.
- D(21) determines whicn word of doubleword contains halfword 2nd operand: if a 1, right word; if a 0, left word.
- D(22) determines which half of word contains halfword 2nd operand: if a 1, right half; if a 0, left half.

The Load Halfword, LH, instruction loads the halfword second operand (located in main storage) into the GPR specified by R1. The halfword obtained from main storage consists of 16 bits. Before loading the operand into LS, it is expanded to a 32-bit word by propagating the sign bit of the halfword through the 16 high-order bits of a word-length register.

Diagram 5-102, FEMDM, is a flowchart of the Load Halfword instruction. At the start of execution, the first 16 bits of the instruction are in E, the second operand address is in D, and the first operand is in S and T. (The first operand plays no part in the instruction; it is subsequently destroyed when data is loaded into T.) During I-Fetch, a storage request was made to obtain the doubleword containing the halfword second operand.

The instruction first tests for a specification check condition. If a program specification interruption occurs, an end op is forced and the instruction is suppressed. If no specification check exists, the execution continues and may be divided into three general steps:

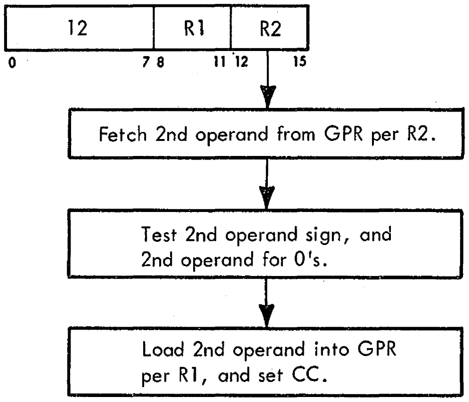
1. Set up A to propagate the sign:
   a. Load 1's into PAL(32–59), shift left four positions, and gate the result to A. ("A" now contains FFFF FF00.) The 1's are generated (ALD AP811) as a result of ROS word 0108 (CLD QB011) containing 100 in bit positions 82, 83, and 84 (CLD QZ011).
   b. Gate the contents of A to PAB, shift left four positions, and gate the result to A. ("A" now contains FFFF F000.)
   c. Gate the contents of A to PAB, shift left four positions, and gate the result to A. ("A" now contains FFFF 0000.)
2. Examine D(21) and D(22) to determine which halfword of the doubleword operand brought out from main storage is to be used as the halfword operand. The specified halfword may be in any one of four possible positions of the doubleword specified by D(21) and D(22):



   a. If D(21) = 0, gate SDBO(0–31) to T; if D(21) = 1, gate SDBO(32–63) to T.
   b. If D(22) = 0, gate T(32–47) to PAA(48–63), and gate a 1 to PAA(47) if T(32) = 0. If D(22) = 1, gate T(48–63) to PAA(48–63), and gate a 1 to PAA(47) if T(48) = 0.
   c. Gate the contents of A to PAB(32–63).
3. Load the selected word:
   a. Gate PAL(32–63) to T.
   b. Gate the contents of T to the GPR specified by the R1 field in E(8–11).
   c. Take an end-op cycle.

Load and Test, LTR (12)

- Load 2nd operand (in GPR per R2) into 1st operand location (in GPR per R1) and set CC according to result.
- RR format:


12
R1
R2
0
7 8
11 12
15
Fetch 2nd operand from GPR per R2.
Test 2nd operand sign, and 2nd operand for 0's.
Load 2nd operand into GPR per R1, and set CC.

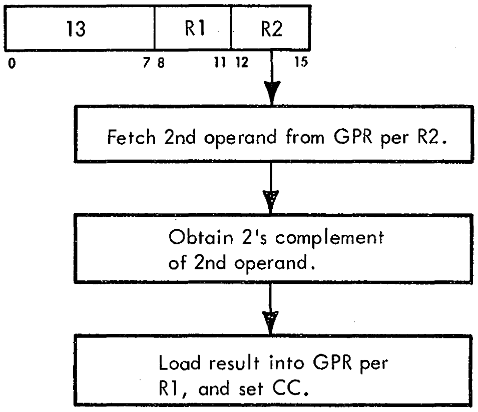
- Conditions at start of execution:
  Instruction is in E.
  1st operand is in A, B, and D (not used).
  2nd operand is in S and T.
- Set STAT A if PAL is all 0's.
- Test T(32) for plus or minus sign.
- STAT A and T(32) determine CC.
- CC setting:
  Result in PAL is zero: CC = 0.
  Result in PAL is less than zero: CC = 1.
  Result in PAL is greater than zero: CC = 2.

The Load and Test, LTR, instruction tests the second operand, from the GPR per R2, for all zeros and loads it into the GPR per R1. (If R1 and R2 specify the same GPR, the operation is equivalent to a test of the data without movement of the data.)

Diagram 5-103, FEMDM, is a flowchart of the instruction. The contents of T (second operand) are gated to PAA(32–63), and STAT A is set if PAL equals zero. The contents of T are then gated to the GPR specified by R1, and the CC is set as follows. If STAT A is set, the CC is set to 0. If STAT A is not set, the sign bit [T(32)] determines the CC: if the sign is minus [T(32) = 1], the CC is set to 1; if the sign is plus, the CC is set to 2. An end op completes instruction execution.

Load Complement, LCR (13)

- Load 2's complement of 2nd operand (in GPR per R2) into 1st operand location (in GPR per R1) and set CC according to result.
- RR format:


13
R1
R2
0
7 8
11 12
15
Fetch 2nd operand from GPR per R2.
Obtain 2's complement of 2nd operand.
Load result into GPR per R1, and set CC.

- Conditions at start of execution:
  Instruction is in E.
  1st operand is in A, B, and D (not used).
  2nd operand is in S and T.
- Set STAT B if overflow occurred.
- Set STAT A if PAL is all 0's.
- Test T(32) for plus or minus sign.
- STAT's A and B, and T(32) determine CC.
- CC setting:
  Result in PAL is zero: CC = 0.
  Result in PAL is less than zero: CC = 1.
  Result in PAL is greater than zero: CC = 2.
  Overflow: CC = 3.

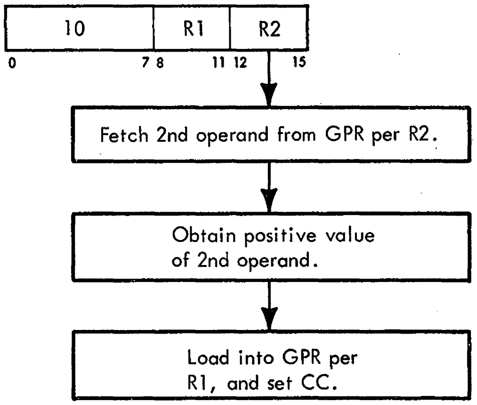
The Load Complement, LCR, instruction loads the 2's complement of the second operand from the GPR per R2 into the GPR per R1.

See Diagram 5-104, FEMDM, a flowchart of the instruction. The contents of T (second operand) are gated in 2's complement form to PAA(32–63). STAT A is set if PAL equals zero, and STAT B is set if a fixed-point overflow occurs. (Overflow occurs if the maximum negative number is 2's complemented.) The contents of PAL are transferred (via T) to the GPR specified by R1, and the CC is set as follows.

If STAT B is set, the CC is set to 3. If STAT A is set, the CC is set to 0. If neither STAT is set, the sign bit [T(32)] determines the CC: if the sign is minus [T(32) = 1], the CC is set to 1; if the sign is plus, the CC is set to 2. An end op completes instruction execution.

Load Positive, LPR (10)

- Load 2nd operand (unchanged if positive, 2's complemented if negative; in GPR per R2) into 1st operand location (in GPR per R1).
- RR format:



10
R1
R2
0
7 8
11 12
15
Fetch 2nd operand from GPR per R2.
Obtain positive value of 2nd operand.
Load into GPR per R1, and set CC.

- Conditions at start of execution:
  Instruction is in E.
  1st operand is in A, B, and D (not used).
  2nd operand is in S and T.
- Set STAT A if PAL is all 0's.
- T(32) determines whether operand loaded is positive.
- If T(32) = 1, 2's complement operand.
- Set STAT B if overflow occurs.
- STAT's A and B and T(32) determine CC.
- CC setting:
  Result in PAL is zero: CC = 0.
  Result in PAL is greater than zero: CC = 2.
  Overflow: CC = 3.

The Load Positive, LPR, instruction loads the absolute value of the contents of the GPR specified by R2 into the GPR specified by R1. The instruction also tests for an all-zero result and for an overflow condition. (Overflow occurs only when the maximum negative number is 2's complemented.) The results of the tests are indicated by the CC.

Diagram 5-105, FEMDM, is a flowchart of the Load Positive instruction. At the start of execution, the instruction is in E, the first operand is in A, B, and D, and the second operand is in S and T. The first cycle of the instruction places the contents of T into PAL(32–63) and tests PAL for all 0's. If PAL equals zero, STAT A is set. The data in T is then loaded into the GPR per E(8–11) (R1).

Because the purpose of the instruction is to load only positive numbers, a test for negative numbers is made by examining T(32). If T(32) = 1, the data loaded in LS was a negative number. In this case, the contents of T must be converted to a positive number (true form) and reloaded into the GPR per E(8–11), thus destroying the negative number in that location.

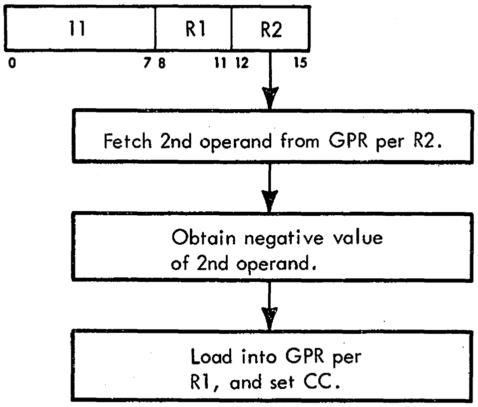
While in PAL, the 2's complement form of the data is tested to see whether overflow occurred when the number was converted. If overflow occurred, STAT B is set.

If T(32) = 0, the data loaded in LS was positive and need not be changed.

STAT's A and B, and T(32) determine the CC as follows. If STAT B is set, the CC is set to 3. If STAT A is set, the CC is set to 0. If neither STAT is set, the sign bit [T(32)] determines the CC; however, the sign can only be plus [T(32) = 0] and the CC is set to 2. An end-op cycle completes instruction execution.

Load Negative, LNR (11)

- Load 2nd operand (unchanged if negative, 2's complemented if positive; in GPR per R2) into 1st operand location (in GPR per R1).
- RR format:



11
R1
R2
0
7 8
11 12
15
Fetch 2nd operand from GPR per R2.
Obtain negative value of 2nd operand.
Load into GPR per R1, and set CC.

- Conditions at start of execution:
  Instruction is in E.
  1st operand is in A, B, and D (not used).
  2nd operand is in S and T.
- Set STAT A if PAL is all 0's.
- T(32) determines whether operand loaded is negative.
- If T(32) = 0, 2's complement operand.
- STAT A and T(32) determine CC.
- CC setting:
  Result in PAL is zero: CC = 0.
  Result in PAL is less than zero: CC = 1.

The Load Negative, LNR, instruction loads the negative value of the contents of the GPR specified by R2 into the GPR specified by R1. The LNR instruction also tests the operand for all zeros and indicates the result in the CC.

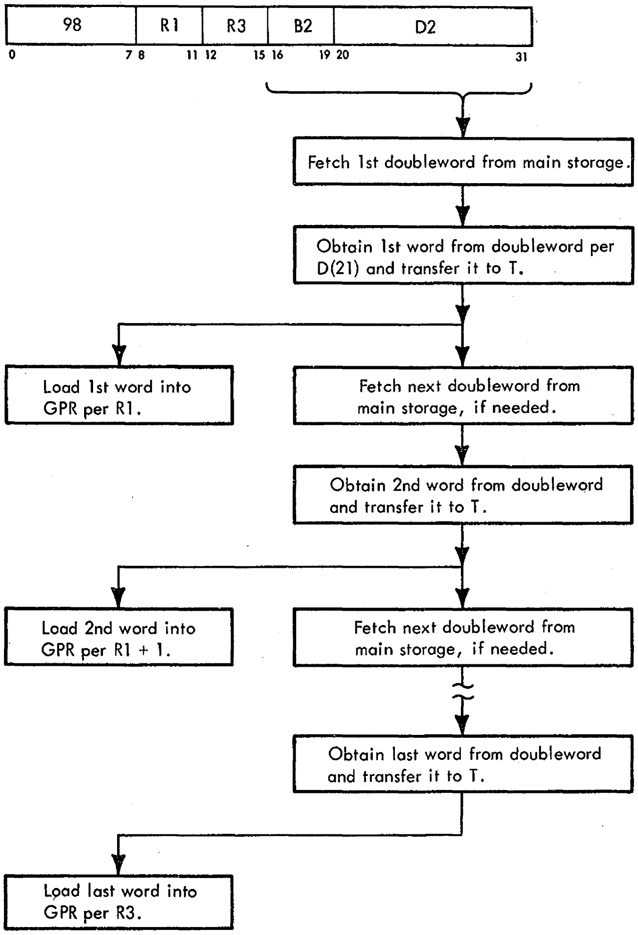
See Diagram 5-106, FEMDM, a flowchart of the instruction. At the start of execution, the instruction is in E, the first operand is in A, B, and D, and the second operand is in S and T. The first cycle of the instruction places the contents of T into PAA(32–63) and tests PAL for all 0's. If PAL contains all 0's, STAT A is set. The data in T, regardless of whether the result equals zero, is loaded into the GPR per E(8–11) (R1).

Because the purpose of the LNR instruction is to load only negative numbers, a test for positive numbers is made by examining T(32). If T(32) = 0, the data loaded into LS was a positive number. In this case, the contents of T must be converted to a negative number (2's complement form) and reloaded into the GPR per E(8–11), thus destroying the positive number in that location. If T(32) = 1, the data loaded into LS was a negative number and need not be changed.

STAT A and T(32) determine the CC as follows. If STAT A is set, the CC is set to 0. If STAT A is not set, the sign bit [T(32)] determines the CC; however, the sign can only be minus [T(32) = 1] and the CC is set to 1. An end-op cycle completes instruction execution.

Load Multiple, LM (98)

- Load 2nd operand (as many words as required; in storage) into GPR's, in ascending order, starting with 1st operand location (per R1) and ending with 3rd operand location (per R3).
- RS format:


98
R1
R3
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch 1st doubleword from main storage.
Obtain 1st word from doubleword per D(21) and transfer it to T.
Load 1st word into GPR per R1.
Fetch next doubleword from main storage, if needed.
Obtain 2nd word from doubleword and transfer it to T.
Load 2nd word into GPR per R1 + 1.
Fetch next doubleword from main storage, if needed.
Obtain last word from doubleword and transfer it to T.
Load last word into GPR per R3.

- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T (not used).
  Starting operand address is in D.
  Main storage request for 2nd operand has been issued per D.
- R1 and R3 are compared to determine whether one or more words are to be loaded.
- If R1 = R3, only one word is to be loaded.
- If R3 is less than R1, addresses wraparound from GPR 15 to GPR 0.
- D(21) determines which word of doubleword is to be loaded into LS: if a 1, right word; if a 0, left word.
- 8 is added to D when more than one word is to be loaded.
- R1 is incremented by 1 each time a new word is to be loaded.

The Load Multiple, LM, instruction loads 32-bit words from main storage into LS. The GPR's are loaded in the ascending order of their addresses, starting with the GPR addressed by R1 and continuing up to and including the GPR addressed by R3. All combinations of GPR addresses specified by R1 and R3 are valid. When R3 is less than R1, the addresses wrap around from GPR 15 to GPR 0.

Diagram 5-107, FEMDM is a flowchart of the Load Multiple instruction. At the start of execution, the first 16 bits of the instruction are in E, the first operand is in S and T, and the starting operand address is in D. During I-Fetch, a storage request was made to fetch the operand addressed by D.

The instruction first tests for a specification check condition. If there is a specification check, a program specification interruption occurs and the operation is suppressed. Assuming there is no specification check, R1 [E(8–11)] and R3 [E(12–15)] are compared to determine whether one or more words are to be loaded. If R1 equals R3, only one word is to be loaded; if unequal, more than one word is to be loaded.

Assume one word is to be loaded (R1 = R3). D(21) is tested to determine which word of the doubleword from main storage is to be loaded. If D(21) is a 0, the left word is selected; if a 1, the right word is selected. The selected word is loaded into the GPR specified by R1, and an end-op cycle is taken, completing the instruction.

If more than one word is to be loaded (R1 and R3 are unequal), the main storage address in D is incremented by 8 to address the next doubleword in main storage, if it should be needed. D(21) is tested to determine which word of the doubleword from SDBO will be loaded first. R3 is decremented by 1, and R1 and R3 are again compared. As a result of these two tests, four possible conditions may exist, resulting in four different branches as follows:

1. If D(21) = 0 and R1 = R3, two words are to be loaded starting with the left word of the doubleword on the

SDBO. In this case, no further main storage request is necessary. SDBO(0–31) is gated to T, SDBO(0–63) is gated to AB, and the contents of T are loaded into the GPR specified by R1. E(8–11) (R1) is incremented by 1 to address the next sequential GPR, and the contents of B are transferred (via T) to the GPR specified by E(8–11) (R1 + 1).

2. If D(21) = 0 and R1 does not equal R3, more than two words are to be loaded starting with the left word. A main storage request per D is initiated to fetch the next doubleword from main storage. The microprogram enters the basic loop, which loads a doubleword, one word at a time, in consecutive GPR's and initiates another main storage request. An exit from the basic loop is made when one or two words remain to be loaded.
3. If D(21) = 1 and R1 = R3, two words are to be loaded starting with the right word. A main storage request per D is initiated because the second word to be loaded is contained in the next doubleword in main storage. SDBO(32–63) is transferred (via T) to the GPR specified by E(8–11) (R1). E(8–11) is incremented by 1, and, when the next doubleword is available, SDBO(0–31) is transferred (via T) to the GPR specified by E(8–11) (R1 + 1).
4. If D(21) = 1 and R1 does not equal R3, more than two words are to be loaded starting with the right word. A main storage request per D is initiated to fetch the next doubleword from main storage, and SDBO(32–63) is transferred (via T) to the GPR specified by E(8–11) (R1). E(8–11) is incremented by 1 to address the next sequential GPR, and D is incremented by 4 to address the next sequential doubleword in main storage. R1 and R3 are again compared; if equal, only two more words are to be loaded. When the requested doubleword is available, it is loaded, one word at a time, into the two sequential GPR's specified. If R1 does not equal R3, the microprogram enters the basic loop.

Note that the last time D was incremented, it was incremented by 4 rather than by 8. At the start of this sequence, D(21) equalled 1, which is equivalent to a value of 4 in D. Adding 4 to D increases the value to 8, which will address the next sequential doubleword in main storage. D(21) has also been changed to a 0, which allows the microprogram to remain in the basic loop as long as is required. Note that a branch on D(21) is performed in the basic loop.

The 4 that is added to D is developed in F. At the start of the instruction, F was set to -64 (1100 0000). F(0) is then set to 0, establishing a value of 0100 0000 in F. F(0–3) and F(4–7) are transposed, giving a value of 0000 0100 (4) in F. When F(4–7) is added to D, D is incremented by 4.

When the last word has been loaded into the GPR specified by R3, an end-op cycle is taken, completing the instruction.

ADD-TYPE INSTRUCTIONS

- Fixed-point add-type instructions use RR and RX formats.
- 2nd operand is algebraically added to 1st operand.
- For subtract and compare instructions, 2nd operand is in 2's complement form.
- Except for compare instructions, result is stored into 1st operand location.
- CC is determined by op code and hardware conditions.

The fixed-point add-type instructions use the RR format with word-length operands, the RX format with word-length operands, and the RX format with a halfword second operand.

At the start of execution of RR format fixed-point instructions, the first operand is in B (also in A and D) and the second operand is in T (also in S).

At the start of execution of RX format fixed-point instructions, the first operand is in S and T, and a main storage request for the second operand has been issued per D. Because the second operand is fetched from main storage, a specification test is performed (Diagram 5-108, Sheet 1, FEMDM), and a program specification interruption is taken if the second operand address does not specify integral boundaries. If a program specification interruption is taken, the instruction is suppressed. If no specification check occurs, the first operand is transferred from T to B, and the specified word of the doubleword requested from main storage is selected per D(21) and is gated to T. The first operand is now in B and the second operand is in T, which is the same condition which would exist after an RR I-Fetch.

If the RX format instruction specifies a halfword second operand, two additional functions must be performed. The desired halfword [selected per D(22)] of the word in T [selected per D(21)] must be loaded into the low-order halfword of T, and the sign bit must be propagated left to fill the high-order halfword of T.

The fixed-point add-type instructions may be divided into three functional groups: add, subtract, and compare. All add-type instructions set a CC, and all except compare instructions store the result into the first operand location.

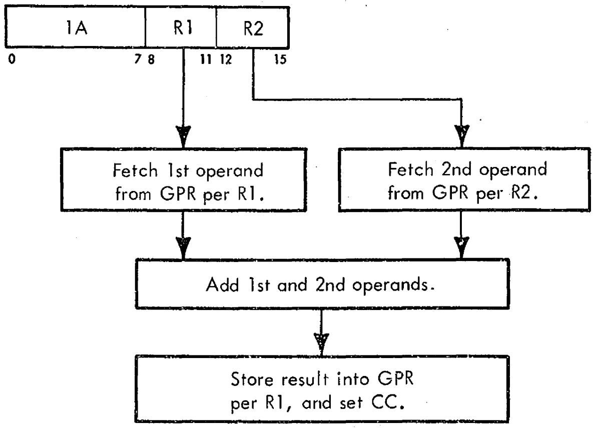
The CPU performs fixed-point add-type instructions as follows (Diagram 5-108, Sheet 2):

1. The second operand is algebraically added to the first operand and the result is stored (except for compare instructions) into the first operand location. For subtract and compare instructions, the second operand (sign bit and integer) is 2's complemented, which, in effect, inverts the sign.
2. Because of the sign notation used, and because positive numbers exist in true binary form and negative numbers exist in 2's complement form, the operand signs are treated as high-order extensions of the integers.

3. Except for Add Logical and Subtract Logical instructions, the sign bit of the result [T(32)] is used as one factor in determining the CC; carry conditions from the high-order digit and from the sign bit are tested for a fixed-point overflow condition (recorded in STAT B).
4. For all fixed-point add-type instructions, a zero result is indicated by setting STAT A.
5. For Add Logical and Subtract Logical instructions, the sign bit of the result is treated as a high-order extension of the integer, and is tested for a carry condition to determine the CC. The result of Add Logical or Subtract Logical instructions is the same as for the corresponding add or subtract instruction, except that the result is not tested for a fixed-point overflow condition and the significance of the CC is different. (See Table in Sheet 2 of Diagram 5-108.)

Add, AR (1A)

- Algebraically add 2nd operand (in GPR, per R2) to 1st operand (in GPR, per R1) and place result into 1st operand location.
- RR format:

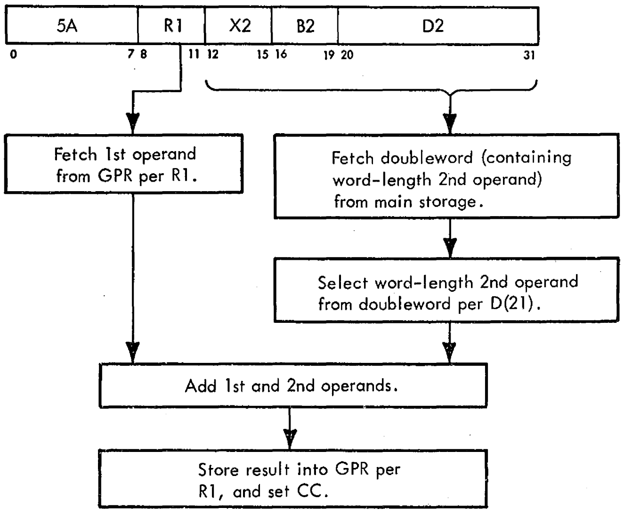

1A
R1
R2
0
7 8
11 12
15
Fetch 1st operand from GPR per R1.
Fetch 2nd operand from GPR per R2.
Add 1st and 2nd operands.
Store result into GPR per R1, and set CC.

- Conditions at start of execution:
  Instruction is in E.
  1st operand is in A, B, and D.
  2nd operand is in S and T.
- CC setting:
  Result is zero (STAT A is set): CC = 0.
  Result is less than zero [T(32) = 1, and STAT's A and B are reset]: CC = 1.
  Result is greater than zero [T(32) = 0, and STAT's A and B are reset]: CC = 2.
  Overflow (STAT B is set): CC = 3.

The Add, AR, instruction algebraically adds the second operand (from the GPR per R2) to the first operand (from the GPR per R1) and stores the result into the first operand location. For the instruction execution, refer to "Add-Type Instructions" and Diagram 5-108.

Add, A (5A)

- Algebraically add 2nd operand (in storage) to 1st operand (in GPR, per R1) and place result into 1st operand location.
- RX format:


5A
R1
X2
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch 1st operand from GPR per R1.
Fetch doubleword (containing word-length 2nd operand) from main storage.
Select word-length 2nd operand from doubleword per D(21).
Add 1st and 2nd operands.
Store result into GPR per R1, and set CC.

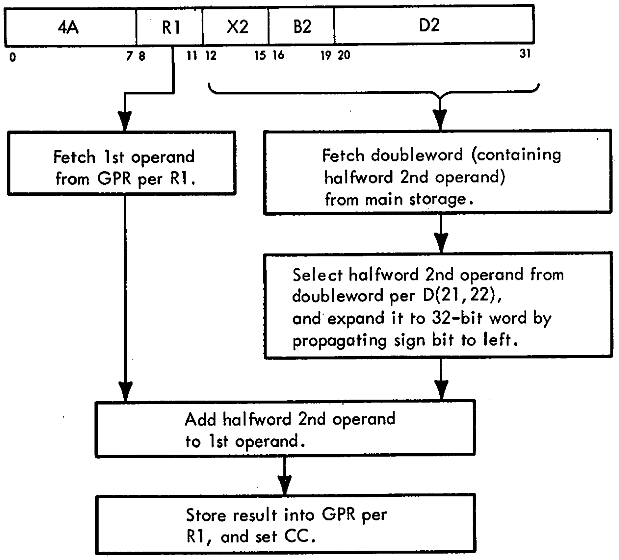
- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  2nd operand address is in D.
  Main storage request for 2nd operand has been issued per D.
- D(21) determines which word of doubleword contains 2nd operand: if a 1, right word; if a 0, left word.
- CC setting:
  Result is zero (STAT A is set): CC = 0.
  Result is less than zero [T(32) = 1, and STAT's A and B are reset]: CC = 1.
  Result is greater than zero [T(32) = 0, and STAT's A and B are reset]: CC = 2.
  Overflow (STAT B is set): CC = 3.

The Add, A, instruction algebraically adds the second operand (from storage) to the first operand (from the GPR per R1) and stores the result into the first operand location. D(21) determines which word of the doubleword fetched from main storage contains the word-length second operand: if D(21) = 1, the right word; if D(21) = 0, the left word. For the instruction execution, refer to "Add-Type Instructions" and Diagram 5-108.

Add Halfword, AH (4A)

- Algebraically add halfword 2nd operand (in storage) to 1st operand (in GPR per R1) and place result into 1st operand location.

- RX format:


4A
R1
X2
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch 1st operand from GPR per R1.
Fetch doubleword (containing halfword 2nd operand) from main storage.
Select halfword 2nd operand from doubleword per D(21,22), and expand it to 32-bit word by propagating sign bit to left.
Add halfword 2nd operand to 1st operand.
Store result into GPR per R1, and set CC.

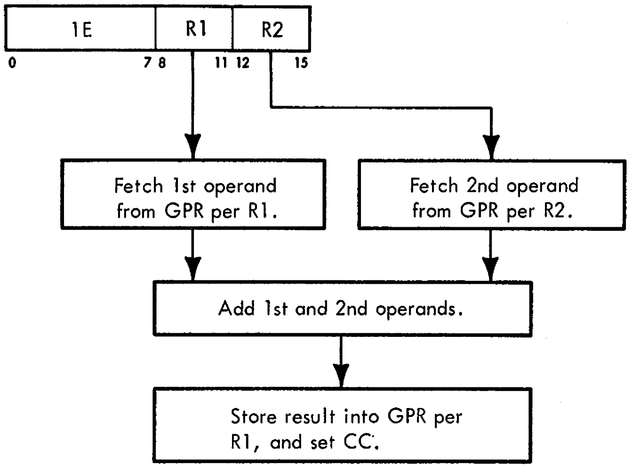
- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  2nd operand address is in D.
  Main storage request for 2nd operand has been issued per D.
- D(21) determines which word of doubleword contains halfword 2nd operand: if a 1, right word; if a 0, left word.
- D(22) determines which half of word contains halfword 2nd operand: if a 1, right half; if a 0, left half.
- CC setting:
  Result is zero (STAT A is set): CC = 0.
  Result is less than zero [T(32) = 1, and STAT's A and B are reset]: CC = 1.
  Result is greater than zero [T(32) = 0 and STAT's A and B are reset]: CC = 2.
  Overflow (STAT B is set): CC = 3.

The Add Halfword, AH, instruction algebraically adds the halfword second operand (from storage) to the first operand (from the GPR per R1) and stores the result into the first operand location. D(21) determines which word of the doubleword fetched from main storage contains the halfword second operand, and D(22) determines which halfword of that word contains the second operand, as follows: D(21) = 0, left word; D(21) = 1, right word; D(22) = 0, left halfword; D(22) = 1, right halfword. When the halfword second operand is selected, it is expanded to a word by propagating the sign bit through the 16 high-order bit positions of T.

For the instruction execution, refer to "Add-Type Instructions" and Diagram 5-108.

Add Logical, ALR (1E)

- Algebraically add 2nd operand (in GPR per R2) to 1st operand (in GPR per R1) and place result into 1st operand location.
- RR format:


1E
R1
R2
0
7 8
11 12
15
Fetch 1st operand from GPR per R1.
Fetch 2nd operand from GPR per R2.
Add 1st and 2nd operands.
Store result into GPR per R1, and set CC.

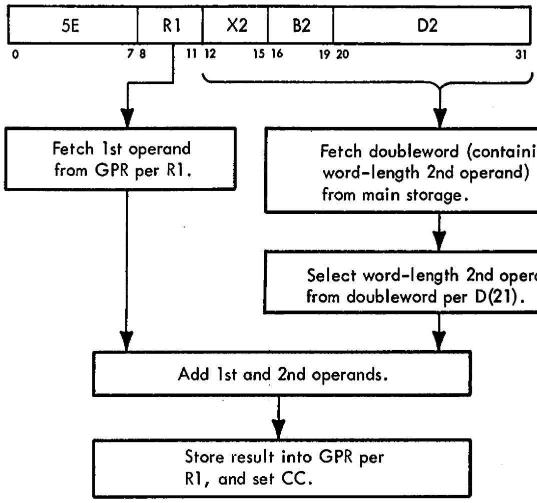
- Conditions at start of execution:
  Instruction is in E.
  1st operand is in A, B, and D.
  2nd operand is in S and T.
- CC setting:
  Result is zero and no carry from PAL(32) [STAT A is set and A(31) = 0]: CC = 0.
  Result is not zero and no carry from PAL(32) [STAT A is reset and A(31) = 0]: CC = 1.
  Result is zero and carry from PAL(32) [STAT A is set and A(31) = 1]: CC = 2.
  Result is not zero and carry from PAL(32) [STAT A is reset and A(31) = 1]: CC = 3.

The Add Logical, ALR, instruction algebraically adds the second operand (from the GPR per R2) to the first operand (from the GPR per R1) and stores the result into the first operand location. The sign bit of the sum is treated as a high-order extension of the integer, and is tested for a carry condition [A(31) = 1] to determine the CC. The sum is the same as for the AR instruction; the only difference in execution is that the sum is not tested for a fixed-point overflow condition, and that the significance of the CC's is different.

For the instruction execution, refer to "Add-Type Instructions" and Diagram 5-108.

Add Logical, AL (5E)

- Algebraically add 2nd operand (in storage) to 1st operand (in GPR per R1) and place result into 1st operand location.
- RX format:


5E
R1
X2
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch 1st operand from GPR per R1.
Fetch doubleword (containing word-length 2nd operand) from main storage.
Select word-length 2nd operand from doubleword per D(21).
Add 1st and 2nd operands.
Store result into GPR per R1, and set CC.

- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  2nd operand address is in D.
  Main storage request for 2nd operand has been issued per D.
- D(21) determines which word of doubleword contains 2nd operand: if a 1, right word; if a 0, left word.
- CC setting:
  Result is zero and no carry from PAL(32) [STAT A is set and A(31) = 0] : CC = 0.
  Result is not zero and no carry from PAL(32) [STAT A is reset and A(31) = 0] : CC = 1.
  Result is zero and carry from PAL(32) [STAT A is set and A(31) = 1] : CC = 2.
  Result is not zero and carry from PAL(32) [STAT A is reset and A(31) = 1] : CC = 3.

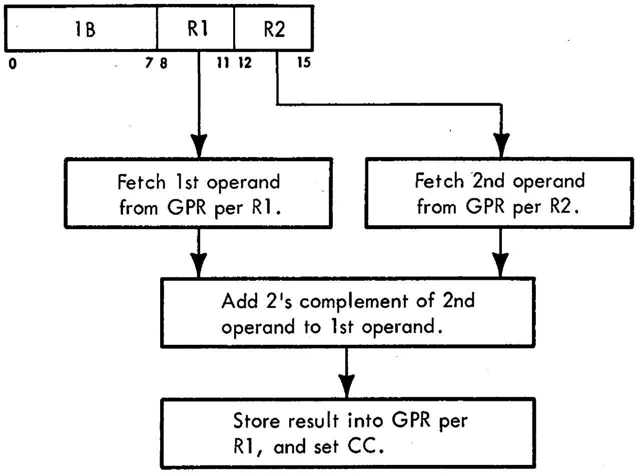
The Add Logical, AL, instruction algebraically adds the second operand (from storage) to the first operand (from the GPR per R1) and stores the result into the first operand location. D(21) determines which word of the doubleword fetched from main storage contains the word-length second operand: if D(21) = 1, the right word; if D(21) = 0, the left word.

The sign bit of the sum is treated as a high-order extension of the integer, and is tested for a carry condition [A(31) = 1] to determine the CC. The sum is the same as for the A instruction; the only difference in execution is that the sum is not tested for a fixed-point overflow condition, and that the significance of the CC's is different.

For the instruction execution, refer to "Add-Type Instructions" and Diagram 5-108.

Subtract, SR (1B)

- Algebraically subtract 2nd operand (in GPR, per R2) from 1st operand (in GPR per R1) and place result into 1st operand location.
- RR format:


1B
R1
R2
0
7 8
11 12
15
Fetch 1st operand from GPR per R1.
Fetch 2nd operand from GPR per R2.
Add 2's complement of 2nd operand to 1st operand.
Store result into GPR per R1, and set CC.

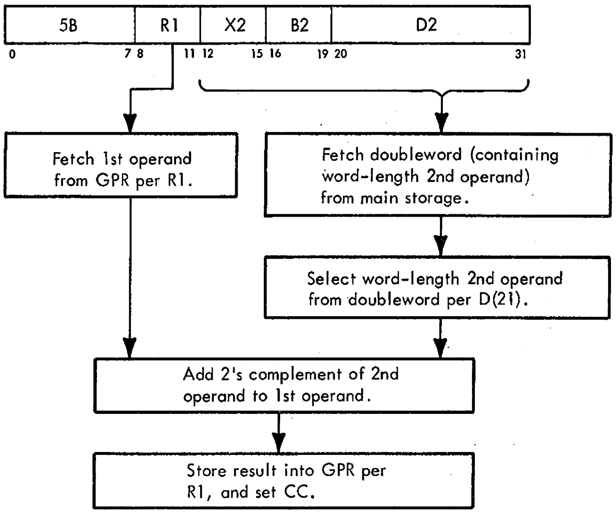
- Conditions at start of execution:
  Instruction is in E.
  1st operand is in A, B, and D.
  2nd operand is in S and T.
- CC setting:
  Result is zero (STAT A is set): CC = 0.
  Result is less than zero [T(32) = 1 and STAT's A and B are reset] : CC = 1.
  Result is greater than zero [T(32) = 0 and STAT's A and B are reset] : CC = 2.
  Overflow (STAT B is set): CC = 3.

The Subtract, SR, instruction adds the 2's complement of the second operand (from the GPR per R2) to the first operand (from the GPR per R1) and stores the result into the first operand location. The only difference between the SR and AR instructions is that the second operand is 2's complemented.

For the instruction execution, refer to "Add-Type Instructions" and Diagram 5-108.

Subtract, S (5B)

- Algebraically subtract 2nd operand (in storage) from 1st operand (in GPR per R1) and place result into 1st operand location.
- RX format:


5B
R1
X2
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch 1st operand from GPR per R1.
Fetch doubleword (containing word-length 2nd operand) from main storage.
Select word-length 2nd operand from doubleword per D(21).
Add 2's complement of 2nd operand to 1st operand.
Store result into GPR per R1, and set CC.

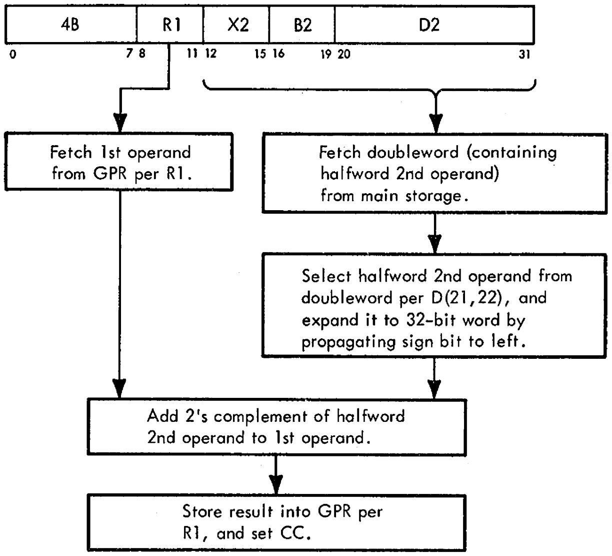
- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  2nd operand address is in D.
  Main storage request for 2nd operand has been issued per D.
- D(21) determines which word of doubleword contains 2nd operand: if a 1, right word; if a 0, left word.
- CC setting:
  Result is zero (STAT A is set): CC = 0.
  Result is less than zero [T(32) = 1 and STAT's A and B are reset]: CC = 1.
  Result is greater than zero [T(32) = 0 and STAT's A and B are reset]: CC = 2.
  Overflow (STAT B is set): CC = 3.

The Subtract, S, instruction adds the 2's complement of the second operand (from storage) to the first operand (from the GPR per R1) and stores the result into the first operand location. The only difference between the S and A instructions is that the second operand is 2's complemented.

For the instruction execution, refer to "Add-Type Instructions" and Diagram 5-108.

Subtract Halfword, SH (4B)

- Algebraically subtract halfword 2nd operand (in storage) from 1st operand (in GPR per R1) and place result into 1st operand location.
- RX format:


4B
R1
X2
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch 1st operand from GPR per R1.
Fetch doubleword (containing halfword 2nd operand) from main storage.
Select halfword 2nd operand from doubleword per D(21,22), and expand it to 32-bit word by propagating sign bit to left.
Add 2's complement of halfword 2nd operand to 1st operand.
Store result into GPR per R1, and set CC.

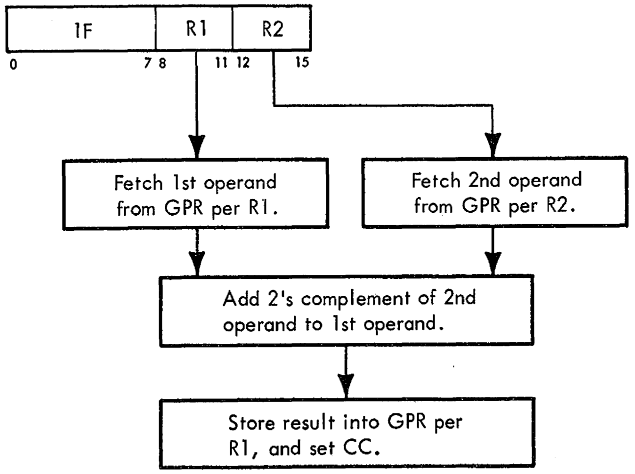
- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  2nd operand address is in D.
  Main storage request for 2nd operand has been issued per D.
- D(21) determines which word of doubleword contains halfword 2nd operand: if a 1, right word; if a 0, left word.
- D(22) determines which half of word contains halfword 2nd operand: if a 1, right half; if a 0, left half.
- CC setting:
  Result is zero (STAT A is set): CC = 0.
  Result is less than zero [T(32) = 1 and STAT's A and B are reset]: CC = 1.
  Result is greater than zero [T(32) = 0 and STAT's A and B are reset]: CC = 2.
  Overflow (STAT B is set): CC = 3.

The Subtract Halfword, SH, instruction adds the 2's complement of the halfword second operand (from storage) to the first operand (from the GPR per R1) and stores the result into the first operand location. The only difference between the SH and AH instructions is that the second operand is 2's complemented.

For the instruction execution, refer to "Add-Type Instructions" and Diagram 5-108.

Subtract Logical, SLR (1F)

- Algebraically subtract 2nd operand (in GPR per R2) from 1st operand (in GPR per R1) and place result into 1st operand location.
- RR format:


1F
R1
R2
0
7 8
11 12
15
Fetch 1st operand from GPR per R1.
Fetch 2nd operand from GPR per R2.
Add 2's complement of 2nd operand to 1st operand.
Store result into GPR per R1, and set CC.

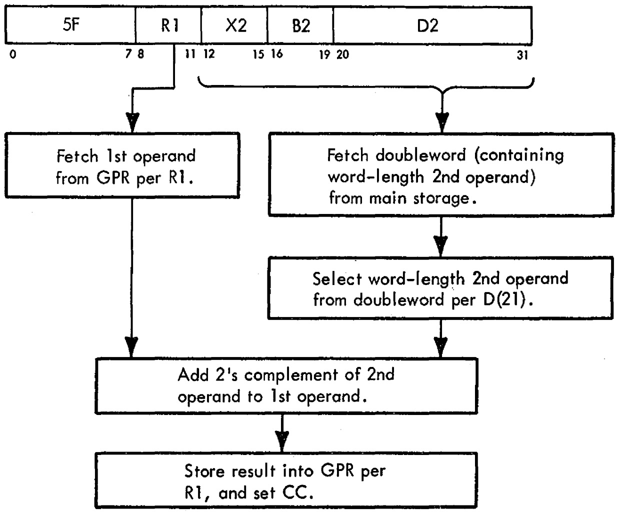
- Conditions at start of execution:
  Instruction is in E.
  1st operand is in A, B, and D.
  2nd operand is in S and T.
- CC setting:
  Result is not zero and no carry from PAL(32) [STAT A is reset and A(31) = 0] : CC = 1.
  Result is zero and carry from PAL(32) [STAT A is set and A(31) = 1] : CC = 2.
  Result is not zero and carry from PAL(32) [STAT A is reset and A(31) = 1] : CC = 3.

The Subtract Logical, SLR, instruction adds the 2's complement of the second operand (from the GPR per R2) to the first operand (from the GPR per R1) and stores the result into the first operand location. The sign bit of the result is treated as a high-order extension of the integer, and is tested for a carry condition [A(31) = 1] to determine the CC. The result is the same as for the ALR instruction; the difference in execution is that the second operand is 2's complemented, the result is not tested for a fixed-point overflow condition, and the significance of the CC's is different.

For the instruction execution, refer to "Add-Type Instructions" and Diagram 5-108.

Subtract Logical, SL (5F)

- Algebraically subtract 2nd operand (in storage) from 1st operand (in GPR per R1) and place result into 1st operand location.
- RX format:


5F
R1
X2
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch 1st operand from GPR per R1.
Fetch doubleword (containing word-length 2nd operand) from main storage.
Select word-length 2nd operand from doubleword per D(21).
Add 2's complement of 2nd operand to 1st operand.
Store result into GPR per R1, and set CC.

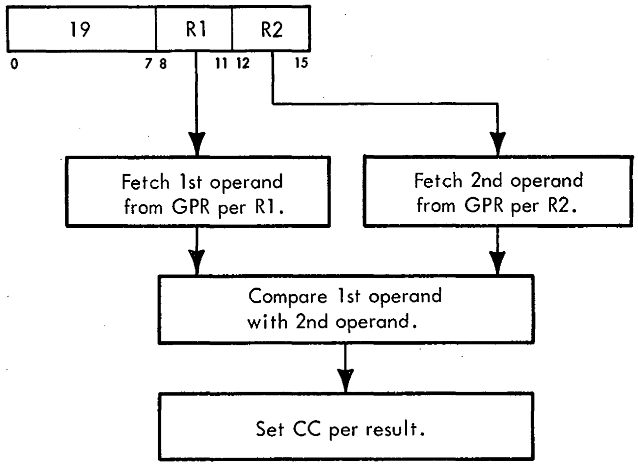
- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  2nd operand address is in D.
  Main storage request for 2nd operand has been issued per D.
- D(21) determines which word of doubleword contains 2nd operand: if a 1, right word; if a 0, left word.
- CC setting:
  Result is not zero and no carry from PAL(32) [STAT A is reset and A(31) = 0] : CC = 1.
  Result is zero and carry from PAL(32) [STAT A is set and A(31) = 1] : CC = 2.
  Result is not zero and carry from PAL(32) [STAT A is reset and A(31) = 1] : CC = 3.

The Subtract Logical, SL, instruction algebraically adds the 2's complement of the second operand (from storage) to the first operand (from the GPR per R1) and stores the result into the first operand location. The sign bit of the result is treated as a high-order extension of the integer, and is tested for a carry condition [A(31) = 1] to determine the CC. The result is the same as for the AL instruction; the difference in execution is that the second operand is 2's complemented, the result is not tested for a fixed-point overflow condition, and the significance of the CC's is different.

For the instruction execution, refer to "Add-Type Instructions" and Diagram 5-108.

Compare, CR (19)

- Algebraically compare 1st operand (in GPR, per R1) with 2nd operand (in GPR, per R2) and set CC according to result.
- RR format:

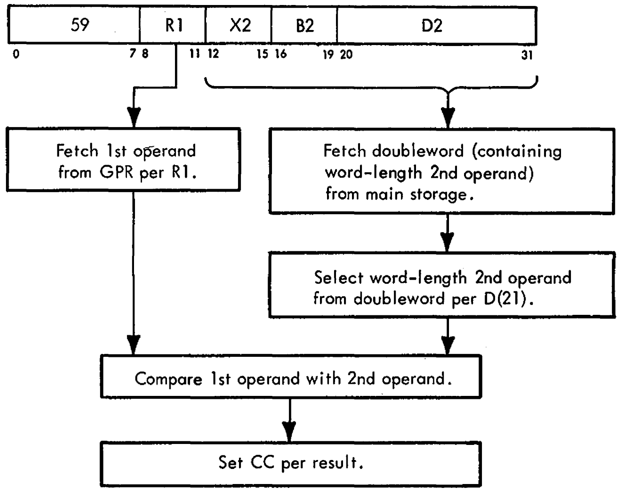

19
R1
R2
0
7 8
11 12
15
Fetch 1st operand from GPR per R1.
Fetch 2nd operand from GPR per R2.
Compare 1st operand with 2nd operand.
Set CC per result.

- Conditions at start of execution:
  Instruction is in E.
  1st operand is in A, B, and D.
  2nd operand is in S and T.
- CC setting:
  Operands are equal (STAT A is set): CC = 0.
  1st operand is less than 2nd operand [STAT B is set or T(32) = 1]: CC = 1.
  1st operand is greater than 2nd operand [STAT B is set and T(32) = 1, or STAT B is reset and T(32) = 0]: CC = 2.

The Compare, CR, instruction algebraically compares the first operand (from the GPR per R1) with the second operand (from the GPR per R2) and sets the CC according to the result. The compare operation is accomplished by adding the 2's complement of the second operand to the first operand and setting the CC according to the result. The result is not stored. For the instruction execution, refer to "Add-Type Instructions" and Diagram 5-108.

Compare, C (59)

- Algebraically compare 1st operand (in GPR per R1) with 2nd operand (in storage) and set CC according to result.
- RX format:


59
R1
X2
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch 1st operand from GPR per R1.
Fetch doubleword (containing word-length 2nd operand) from main storage.
Select word-length 2nd operand from doubleword per D(21).
Compare 1st operand with 2nd operand.
Set CC per result.

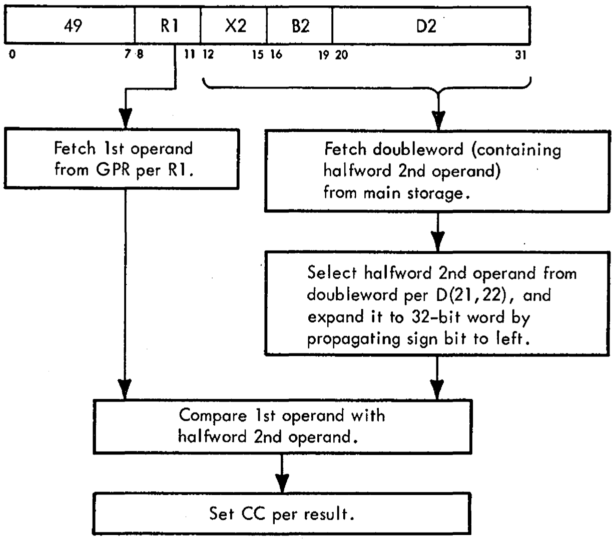
- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  2nd operand address is in D.
  Main storage request for 2nd operand has been issued per D.
- D(21) determines which word of doubleword contains 2nd operand: if a 1, right word; if a 0, left word.
- CC setting:
  Operands are equal (STAT A is set): CC = 0.
  1st operand is less than 2nd operand [STAT B is set or T(32) = 1]: CC = 1.
  1st operand is greater than 2nd operand [STAT B is set and T(32) = 1, or STAT B is reset and T(32) = 0]: CC = 2.

The Compare, C, instruction algebraically compares the first operand (from the GPR per R1) with the second operand (from storage) and sets the CC according to the result.

Because the word-length second operand is in main storage, D(21) determines which word of the doubleword fetched from main storage contains the second operand: if a 1, right word; if a 0, left word. The compare operation is accomplished by adding the 2's complement of the second operand to the first operand and setting the CC according to the result. The result is not stored. For the instruction execution, refer to "Add-Type Instructions" and Diagram 5-108.

Compare Halfword, CH (49)

- Algebraically compare 1st operand (in GPR per R1) with halfword 2nd operand (in storage) and set CC according to result.
- RX format:



49
R1
X2
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch 1st operand from GPR per R1.
Fetch doubleword (containing halfword 2nd operand) from main storage.
Select halfword 2nd operand from doubleword per D(21,22), and expand it to 32-bit word by propagating sign bit to left.
Compare 1st operand with halfword 2nd operand.
Set CC per result.

- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  2nd operand address is in D.
  Main storage request for 2nd operand has been issued per D.
- D(21) determines which word of doubleword contains halfword 2nd operand: if a 1, right word; if a 0, left word.
- D(22) determines which half of word contains halfword 2nd operand: if a 1, right half; if a 0, left half.
- CC setting:
  Operands are equal (STAT A is set): CC = 0.
  1st operand is less than 2nd operand [STAT B is set or T(32) = 1] : CC = 1.
  1st operand is greater than 2nd operand [STAT B is set and T(32) = 1, or STAT B is reset and T(32) = 0] : CC = 2.

The Compare Halfword, CH, instruction algebraically compares the first operand (from the GPR per R1) with the halfword second operand (from storage) and sets the CC according to the result.

Because the halfword second operand is in main storage, D(21) determines which word of the doubleword fetched from main storage contains the halfword second operand: if a 1, right word; if a 0, left word. D(22) determines which half of that word contains the second operand: if a 1, right half; if a 0, left half. The halfword second operand is expanded to word-length by propagating the sign bit through the high-order 16-bit positions of T. The compare operation is accomplished by adding the 2's complement of the halfword second operand to the first operand and setting the CC according to the result. For the instruction execution, refer to "Add-Type Instructions" and Diagram 5-108.

MULTIPLY

There are three fixed-point multiply instructions:

1. Multiply, MR, RR format – which uses a 32-bit multiplier and multiplicand, and produces a 64-bit product.
2. Multiply, M, RX format – which uses a 32-bit multiplier and multiplicand, and produces a 64-bit product.
3. Multiply Halfword, MH, RX format – which uses a 16-bit multiplier and a 16-bit multiplicand, and produces a 32-bit product.

Note: In the Multiply, M, and Multiply Halfword, MH, instructions, the roles of the first and second operands are reversed from the roles defined in the *System/360 Principles of Operation,* SRL, Form A22-6821-6. That is, the second operand is the multiplicand and the first operand is the multiplier. (Interchanging the operand roles does not affect the product.) The result, however, still replaces the first operand.

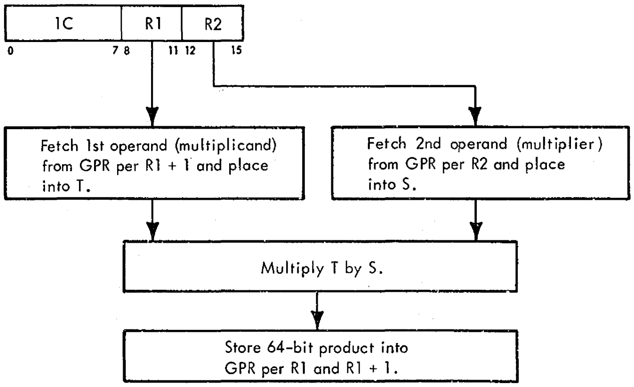
Each of the three fixed-point multiply instructions has a unique initialization routine. The initialization routines (Diagram 5-109, Sheet 1, FEMDM) perform a specification test, set E(12–15) to 15, set the STC to 3, and establish the operands in S and T as follows:

1. Multiply, MR, RR format – Transfers the multiplier (second operand from GPR per R2) to S and the multiplicand (first operand from GPR per R1 + 1) to T.
2. Multiply, M, RX format – Transfers the multiplicand (first operand from GPR per R1 + 1) to S and the multiplier (second operand from main storage) to T.
3. Multiply Halfword, MH, RX format – Transfers the multiplicand (first operand from GPR per R1) to S and the multiplier (halfword second operand from main storage, expanded to word length by propagating the sign bit through the 16 high-order bit positions) to T.

A common multiply microprogram is then entered. Multiples of T are selected per bit-pairs from S and are added to a partial product in B to form a new partial product. Low-order partial product bit-pairs are accumulated in F and SAL. When SAL has accumulated a partial product byte, it is stored into S, replacing the byte of S already used. Sixteen multiply cycles are taken until a word (four bytes) of product is accumulated in S. PAL now contains the high-order word of the product and S contains the low-order word of the product. The product is stored in an even/odd pair of GPR's specified by R1 and R1 + 1, and an end-op cycle is taken to terminate the operation. (For the Multiply Halfword instruction, only the low-order 32 bits of the product are stored into the GPR per R1.)

Multiply, MR (1C)

- Multiply 1st operand (in GPR per R1 + 1) by 2nd operand (in GPR per R2) and place 64-bit product into 1st operand location (in GPR per R1 and R1 + 1).
- RR format:


1C
R1
R2
0
7 8
11 12
15
Fetch 1st operand (multiplicand) from GPR per R1 + 1 and place into T.
Fetch 2nd operand (multiplier) from GPR per R2 and place into S.
Multiply T by S.
Store 64-bit product into GPR per R1 and R1 + 1.

- Conditions at start of execution:
  Instruction is in E.
  Contents of even-address GPR specified by R1 are in A, B, and D (not used).
  Multiplicand (1st operand) is in odd-address GPR specified by R1 + 1.
  2nd operand (multiplier) is in S and T.
- Multiple selection bits (M1,M2) are selected from multiplier (in S) per E(12–15).
- Multiples of multiplicand (in T) are selected by M1,M2 bits and 'TX' trigger.
- Multiples of multiplicand are added to partial product in B.
- Partial product bits from B(66,67) are accumulated in SAL and F per E(14,15).
- SAL contains byte of partial product when filled.
- SAL is transferred to correct byte in S per STC.
- When last (4th) byte is transferred to S, multiplier in S is replaced by low-order half of product; high-order bits are in PAL.

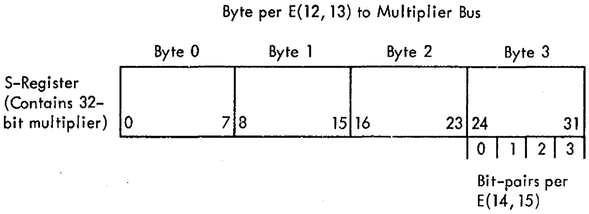
The Multiply, MR, instruction multiplies the contents of T (multiplicand) by the contents of S (multiplier). Because both the multiplier and the multiplicand are 32-bit signed integers, the product is a 64-bit signed integer and must be stored into an even/odd pair of GPR's.

A flowchart of the operation is shown in Diagram 5-109, FEMDM. To correctly specify the even/odd GPR pair, the R1 field of the instruction must refer to an even-address GPR or a program specification interruption occurs. After R1 is tested to see whether it is even, 15 and 3 are placed into E(12–15) and the STC, respectively. The value in E(12–15) selects the correct multiple selection bits (M1,M2) from the multiplier in S, and the value in the STC correctly positions the partial product byte in S. Each value is sequentially reduced during the operation. The value in T is now destroyed, and the multiplicand is transferred from the GPR per R1 + 1 to T. At this point, S contains the multiplier and T contains the multiplicand.

Execution of the MR instruction occurs in three iterative steps.

1. Selection of multiplicand multiples.
2. Addition of multiplicand multiples to partial product to form a new partial product.
3. Extraction of partial product bits to form a product.

Multiple selection bits (M1,M2) are selected from S per E(12–15), which is initially set to 15 (decimal) and is decremented by 1 each time multiple selection bits are selected. E(12,13) determines which byte of S is gated to the multiplier (MPR) bus, and E(14,15) determines which bit-pair of the selected byte is used to set M1,M2:


Byte per E(12, 13) to Multiplier Bus
Byte 0
Byte 1
Byte 2
Byte 3
S-Register (Contains 32-bit multiplier)
0
7
8
15
16
23
24
31
0
1
2
3
Bit-pairs per E(14, 15)

Bits M1,M2, which have the same bit configuration as the bit-pair selected from S by E(12–15), are used with the 'TX' trigger to select a multiple of the multiplicand which will be added to a partial product to develop a new partial product. The multiple selected for all combinations of M1,M2 bits and the state of the 'TX' trigger is listed in Table 3-2.

Four bit configurations of M1,M2 are possible, representing decimal values of 0, 1, 2, and 3. Five multiplicand multiples can be selected: 0 x T, 1 x T, -1 x T, 2 x T and -2 x T. Multiplicand multiples are developed and applied as follows:

1. 0 x T: Zero's are added to the partial product.

2. 1 x T: The multiplicand (in T) is added to the partial product.
3. -1 x T: The multiplicand is added in 2's complement form to the partial product.
4. 2 x T: The multiplicand is gated to PAA, shifted left one bit position (in effect, doubles its value), and addéd to the partial product.
5. -2 x T: The multiplicand, shifted left one bit position, is added in 2's complement form to the partial product.

No provision has been made to develop a multiple of the multiplicand of 3 x T. Therefore, when M1,M2 has a decimal value of 3, a multiple of -1 x T is selected and the 'TX' trigger is set and remains set into the next multiply cycle. Note in Table 3-2 that when the 'TX' trigger is set during the selection of a multiple, it has the effect of increasing the value of the multiple by 1 for the corresponding value of M1,M2. Because the partial product is shifted right two positions before each multiple is added, the value of the multiple is increased by a factor of 4. The effect of the 'TX' trigger's being set is to increase the value of the multiple (defined by M1,M2) by 4. Thus the multiplicand is, in effect, multiplied by -1 and +4 (plus the multiple which would have been selected if the 'TX' trigger were not set).

When the partial product is shifted right two positions (right 4 and left 2) after each addition of the selected multiple of the multiplicand, the low-order bit-pair of the partial product is gated to SAL per E(14,15) and is added to the contents of F. When a byte of the partial product is accumulated in F, the byte is transferred to S per the STC (via SAL), replacing the byte of multiplier which has been used.

Table 3-2. Value of Multiple Determined by Multiple Selection Bits (Fixed-Point)

Multiple Selection Bits		'TX' Trigger	T-Register Times Value Indicated	Set 'TX' Trigger
M1	M2			
0	0	0	0 x T	No
0	1	0	1 x T	No
1	0	0	2 x T	No
1	0	0	-2 x T	No†
1	1	0	-1 x T (2's Complement)	Yes
0	0	1	1 x T	No
0	1	1	2 x T	No
1	0	1	-1 x T (2's Complement)	Yes
1	1	1	0 x T	Yes

† Used on last multiple selection if multiplicand is negative.

When the low-order partial product bit-pair is gated to SAL, E(12–15) has been decremented twice and the low-order bit pair is stored into F(6,7) per the value of 01 in E(14,15) (C of Sheet 3, Diagram 5-109, FEMDM).

As previously mentioned, the partial product bits are transferred via SAL to F. This register accumulates a partial product byte (eight bits). When the last two partial product bits, required to complete a byte, are selected [E(14,15) = 10], the contents of F are transferred to SAL, where the two partial product bits are positioned correctly. This byte is then transferred to S and positioned according to the value of the STC. At this point, the STC is equal to 011, thus placing the partial product byte into S(24–31). The STC and E(12–15) are then reduced by 1, and selection of partial product bits is continued. The microprogram remains in the multiple selection loop until E(12,13) = 00 when tested. At this time, E(12–15) contains 0001 and is decremented to 0000; and the microprogram enters the multiply termination routine.

During the multiply termination routine, five events take place:
1. A multiple is selected for the high-order bit-pair in S.
2. Because there is a 2-cycle lag between the selection of a multiple and the storage of the corresponding partial product bit-pair into F, E(12–15) wraps around to 1110 to control the storage of the last two bit-pairs of the partial product in F (via SAL).
3. The high-order partial product byte is transferred from F to S per the STC. (The last bit-pair, however, does not go to F, because the last byte is gated directly from SAL to S.)
4. The high-order word of the partial product is transferred from PAL to T, and from T to the GPR specified by R1.
5. The low-order word of the partial product in S is transferred to T, and from T to the GPR specified by R1 + 1.

This sequence places the complete product into the even/odd GPR pair in LS. An end-op cycle is then taken, and the operation is finished.

The product sign follows the rules of algebra (except that the sign of a zero product is plus); however, the sign bits are manipulated as though they were high-order extensions of the integer throughout the multiply operation. Two multiply examples follow; the first, in Figure 3-6, uses two positive operands, and the second, in Figure 3-7, uses the same operands with minus signs. Note that the product is the same in both cases with no special handling of the sign bits required.

	Hex	Binary	Decimal
Multiplicand (in T)	001A	0000 0000 0001 1010	+26
Multiplier (in S)	00A3	0000 0000 1010 0011	x +163
Product (to S via F)	108E	0001 0000 1000 1110	+4238

1. M1, M2 bits are derived from S per E(12-15).

2. See Table 3-2 for multiple selection.

1 M1, M2	'TX' Tgr	Multiple	E(12-15)	Remarks	Contents in PAL	Accumulate Bit-Pairs in F; Transfer Bytes to S
	0		1111	Initial partial product	0000 0000 0000 0000	
1 1	0	-1 x T	1111	+ T (2's complement)	1111 1111 1110 0110	
	1		1110	New partial product	1111 1111 1110 0110	
	1		1110	Right 2 positions	1111 1111 1111 1001	10
0 0	1	1 x T	1110	+ T (true)	0000 0000 0001 1010	10
	0		1101	New partial product	0000 0000 0001 0011	
	0		1101	Right 2 positions	0000 0000 0000 0100	11
1 0	0	2 x T	1101	+ 2T (true, left 1)	0000 0000 0011 0100	1110
	0		1100	New partial product	0000 0000 0011 1000	
	0		1100	Right 2 positions	0000 0000 0000 1110	00
1 0	0	2 x T	1100	+ 2T (true, left 1)	0000 0000 0011 0100	00 1110
	0		1011	New partial product	0000 0000 0100 0010	
	0		1011	Right 2 positions	0000 0000 0001 0000	10
0 0	0	0 x T	1011	Add 0's	0000 0000 0000 0000	1000 1110
	0		1010	New partial product	0000 0000 0001 0000	
	0		1010	Right 2 positions	0000 0000 0000 0100	00
0 0	0	0 x T	1010	Add 0's	0000 0000 0000 0000	00
	0		1001	New partial product	0000 0000 0000 0100	
	0		1001	Right 2 positions	0000 0000 0000 0001	00
0 0	0	0 x T	1001	Add 0's	0000 0000 0000 0000	0000
	0		1000	New partial product	0000 0000 0000 0001	
	0		1000	Right 2 positions	0000 0000 0000 0000	01
0 0	0	0 x T	1000	Add 0's	0000 0000 0000 0000	01 0000
	0		0111	New partial product	0000 0000 0000 0000	
	0		0111	Right 2 positions	0000 0000 0000 0000	00
0 0	0	0 x T	0111	Add 0's	0000 0000 0000 0000	0001 0000

and so on

S ← 0001 0000 1000 1110

1 0 8 E

Notes:
1. T = T-register.
2. For RX format instruction. reverse multiplier and multiplicand.

Figure 3-6. Fixed-Point Multiply, Example No. 1 (RR Format)

	Hex	Binary	Decimal
Multiplicand (in T)	FFE6	1111 1111 1110 0110	-26
Multiplier (in S)	FF5D	1111 1111 0101 1101	x -163
Product (to S via F)	108E	0001 0000 1000 1110	+4238

1. M1, M2 bits are derived from S per E(12-15).

2. See Table 3-2 for multiple selection.

M1, M2	'TX' Tgr	Multiple	E(12-15)	Remarks	Contents in PAL	Accumulate Bit-Pairs in F; Transfer Bytes to S
	0		1111	Initial partial product	0000 0000 0000 0000	
0 1	0	1 x T	1111	+ T (true)	1111 1111 1110 0110	
	0		1110	New partial product	1111 1111 1110 0110	
	0		1110	Right 2 positions	1111 1111 1111 1001	10
1 1	0	-1 x T	1110	+ T (2's complement)	0000 0000 0001 1010	10
	1		1101	New partial product	0000 0000 0001 0011	
	1		1101	Right 2 positions	0000 0000 0000 0100	11
0 1	1	2 x T	1101	+ 2T (true, left 1)	1111 1111 1100 1100	1110
	0		1100	New partial product	1111 1111 1101 0000	
	0		1100	Right 2 positions	1111 1111 1111 0100	00
0 1	0	1 x T	1100	+ T (true)	1111 1111 1110 0110	00 1110
	0		1011	New partial product	1111 1111 1101 1010	
	0		1011	Right 2 positions	1111 1111 1111 0110	10
1 1	0	-1 x T	1011	+ T (2's complement)	0000 0000 0001 1010	1000 1110
	1		1010	New partial product	0000 0000 0001 0000	
	1		1010	Right 2 positions	0000 0000 0000 0100	00
1 1	1	0 x T	1010	Add 0's	0000 0000 0000 0000	00
	1		1001	New partial product	0000 0000 0000 0100	
	1		1001	Right 2 positions	0000 0000 0000 0001	00
1 1	1	0 x T	1001	Add 0's	0000 0000 0000 0000	0000
	1		1000	New partial product	0000 0000 0000 0001	
	1		1000	Right 2 positions	0000 0000 0000 0000	01
1 1	1	0 x T	1000	Add 0's	0000 0000 0000 0000	01 0000
	1		0111	New partial product	0000 0000 0000 0000	
	1		0111	Right 2 positions	0000 0000 0000 0000	00
1 1	1	0 x T	0111	Add 0's	0000 0000 0000 0000	0001 0000

and so on

S ← 0001 0000 1000 1110

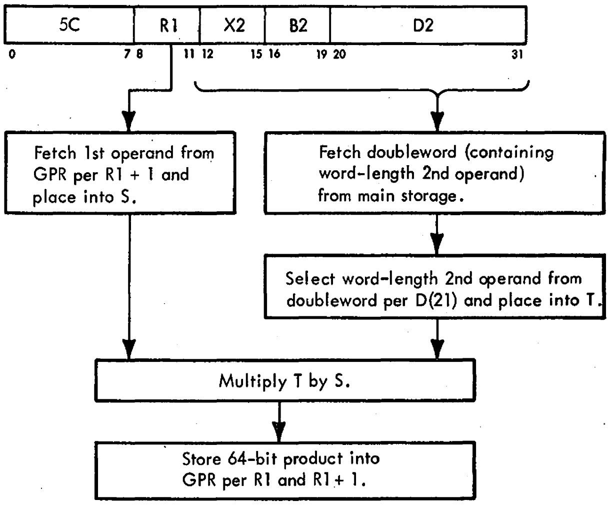
1 0 8 E

Notes:
1. T = T-register.
2. For RX format instructions, reverse multiplier and multiplicand.

Figure 3-7. Fixed-Point Multiply, Example No. 2 (RR Format)

Multiply, M (5C)

- Multiply 1st operand (in GPR per R1 + 1) and 2nd operand (in storage) and place 64-bit result into 1st operand location (in GPR per R1 and R1 + 1).
- See Note under "Multiply."
- RX format:

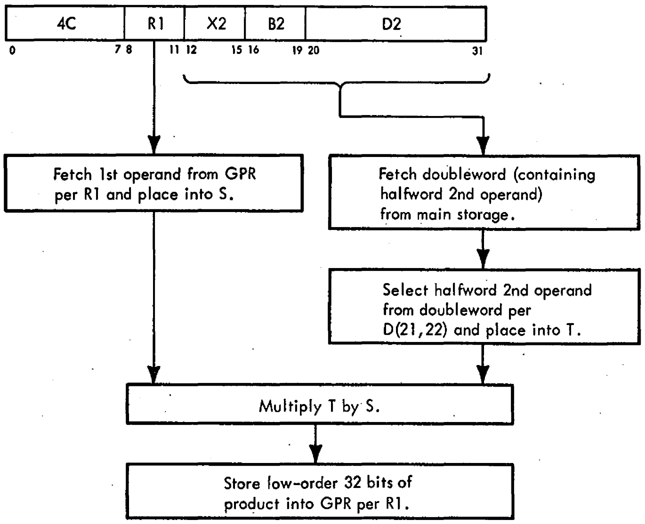

5C
R1
X2
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch 1st operand from GPR per R1 + 1 and place into S.
Fetch doubleword (containing word-length 2nd operand) from main storage.
Select word-length 2nd operand from doubleword per D(21) and place into T.
Multiply T by S.
Store 64-bit product into GPR per R1 and R1 + 1.

- Conditions at start of execution:
  First 16 bits of instruction are in E.
  Contents of even-address GPR per R1 are in S and T (not used).
  1st operand is in odd-address GPR per R1 + 1.
  2nd operand address is in D.
  Main storage request for 2nd operand has been issued per D.
- D(21) determines which word of doubleword contains 2nd operand: if a 1, right word; if a 0, left word.

The Multiply, M, instruction multiplies the contents of T (second operand from storage) and the contents of S (first operand, from GPR per R1 + 1), and stores the 64-bit product into the GPR's per R1 and R1 + 1. Once the operands have been obtained, the operation is identical to that of the Multiply, MR, instruction, except that the roles of the multiplier and multiplicand are reversed. (See Note under "Multiply" and Diagram 5-109.)

Multiply Halfword, MH (4C)

- Multiply 1st operand (in GPR per R1) and halfword 2nd operand (in storage) and place low-order 32 bits of result into 1st operand location.
- See Note under "Multiply".
- RX format:


4C
R1
X2
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch 1st operand from GPR per R1 and place into S.
Fetch doubleword (containing halfword 2nd operand) from main storage.
Select halfword 2nd operand from doubleword per D(21,22) and place into T.
Multiply T by S.
Store low-order 32 bits of product into GPR per R1.

- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  2nd operand address is in D.
  Main storage request for 2nd operand has been issued per D.
- D(21) determines which word of doubleword contains halfword 2nd operand: if a 1, right word; if a 0, left word.
- D(22) determines which half of word contains halfword 2nd operand: if a 1, right half; if a 0, left half.

The Multiply Halfword, MH, instruction multiplies the contents of T (expanded halfword second operand from main storage) and the contents of S (first operand from GPR per R1) and stores the 32 low-order bits of the product into the first operand location. D(21,22) determines the location of the second operand within the doubleword obtained from main storage. The second operand is then expanded to a word-length operand by propagating the sign bit through the high-order 16 bit positions of T.

From this point, the operation is identical to that of the Multiply, MR, instruction, except that the roles of the multiplier and multiplicand are reversed. (See Note under "Multiply.") For a flowchart of the operation, see Diagram 5-109.

DIVIDE

Fixed point division is performed by repetitive reduction of the dividend by multiples of the divisor to obtain a remainder whose value is less than that of the divisor, and to accumulate partial quotient (PQ) bits derived from the partial remainders to form a quotient. The basic nonrestoring method is used, with the dividend in true form. Nonrestoring division means that if a negative remainder is obtained during the reduction cycles (an overdraw has been made), the remainder is not corrected, but instead the next divisor multiples are made positive until the remainder becomes positive again. (In most valid divide operations, the first reduction cycle is an intentional overdraw.)

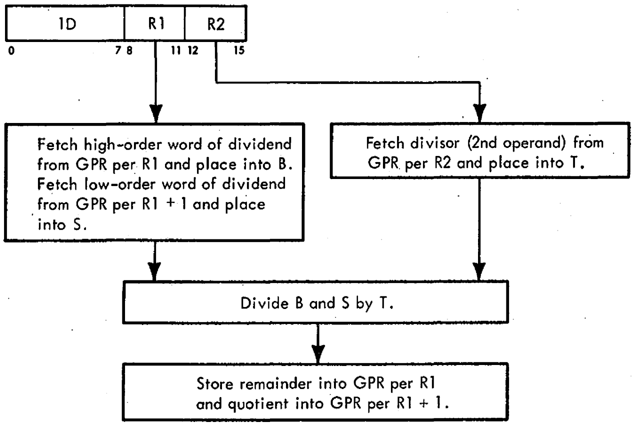
There are two fixed-point divide instructions: Divide, DR, RR format, and Divide, D, RX format. Each has a unique initialization routine. The initialization routines: provide a specification test; place the low-order half of the dividend into S and the high-order half of the dividend into B, in true form regardless of the sign; place the divisor into T; store the signs for later use; and set E(12–15) and the STC to 0.

A common divide microprogram is then entered. Two bits of the low-order dividend are appended to the high-order dividend. A multiple of the divisor is selected to reduce the dividend to a partial remainder. The inverted sign of the partial remainder is stored into F as a partial quotient (PQ) bit. Thirty-two such reduction cycles are taken, accumulating PQ bits in F until a byte of quotient is obtained. The quotient byte is stored into S, replacing the byte of the low-order dividend which has been used. When four quotient bytes have been stored into S, S contains the quotient and B contains the remainder.

The microprogram now enters one of four termination routines, determined by the sign of the divisor and the form (true or 2's complement) of the quotient. The termination routines establish the proper sign and form of the remainder and quotient, according to the convention of fixed-point arithmetic and the rules of algebra. The remainder and quotient are then stored in an even/odd pair of GPR's specified by R1 and R1 + 1, and an end-op cycle is taken to terminate the operation.

Divide, DR (1D)

- Divide 1st operand (in GPR per R1 and R1 + 1) by 2nd operand (in GPR per R2) and place result into 1st operand location (remainder in GPR per R1; quotient in GPR per R1 + 1).
- RR format: (See adjoining column.)
- Conditions at start of execution:
  Instruction is in E.
  High-order half of dividend (1st operand) is in A, B, and D.
  Low-order half of dividend is in GPR per R1 + 1.
  Divisor (2nd operand) is in S and T.


1D
R1
R2
0
7 8
11 12
15
Fetch high-order word of dividend from GPR per R1 and place into B. Fetch low-order word of dividend from GPR per R1 + 1 and place into S.
Fetch divisor (2nd operand) from GPR per R2 and place into T.
Divide B and S by T.
Store remainder into GPR per R1 and quotient into GPR per R1 + 1.

The Divide, DR, instruction divides the contents of B (high-order bits of dividend) and S (low-order bits of dividend) by the contents of T (divisor).

The dividend is a 64-bit signed integer occupying an even/odd pair of GPR's addressed by R1 and R1 + 1, respectively. To correctly specify the even/odd pair of GPR's, R1 must refer to an even-numbered GPR or a program specification interruption occurs. A 32-bit signed remainder and a 32-bit signed quotient replace the dividend in the even-numbered and odd-numbered GPR, respectively, of LS. The divisor is also a 32-bit integer.

Two bits of the low-order half of the dividend are first placed into the high-order half of the dividend. A multiple of the divisor is then selected and subtracted from the high-order half of the dividend to form a partial remainder. The resultant value determines the partial quotient (PQ) bit which is placed into F to accumulate the bits until a PQ byte is available. This PQ byte is transferred to S, which contains the low-order half of the dividend, and replaces those bits that have already been used in the operation. This action continues until a complete quotient and remainder are available, at which time they are stored into LS and an end-op cycle is taken.

The sign of the quotient is determined algebraically; four possible combinations of signs can occur:

1. + dividend, + divisor, + quotient, + remainder.
2. - dividend, - divisor, + quotient, - remainder.
3. + dividend, - divisor, - quotient, + remainder.
4. - dividend, + divisor, - quotient, - remainder.

Note that if the dividend and divisor signs are alike the quotient is positive; if unlike the quotient is negative. Note also that the sign of the remainder is the same as the sign of the dividend, except for a zero result, which is always positive.

When the relative magnitude of the dividend and divisor is such that the quotient cannot be expressed by a 32-bit signed integer, a program fixed-point divide interruption occurs. When this happens, the instruction is suppressed, leaving the dividend unchanged in local storage.

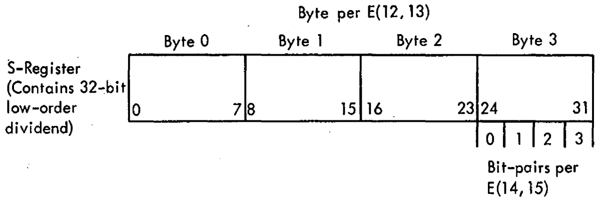
General Discussion

- Multiple selection bits are determined by E(12–15) and S bits.
- Multiples of divisor are determined by 'DVDL1' or 'DVDL0' micro-order, carry from PAL(28), and T(32).
- PQ bits are transferred and accumulated in F per E(14,15), and 'DVDL1' or 'DVDL0' micro-order.
- F contains byte of partial quotient when filled.
- Contents of F are transferred to correct byte in S per STC.

Execution of the DR instruction occurs in three iterative steps:

1. Transfer of bits from low-order half of dividend to high-order half of dividend.
2. Selection of divisor multiple.
3. Determination of quotient bits.

Selection of the two bits from the low-order half of the dividend is shown in B of Sheet 4, Diagram 5-110, FEMDM. Multiple selection bits (M1,M2) are selected from S per E(12–15), which is initially set to 0 and is incremented by 1 after the selection of each pair of multiple selection bits. E(12,13) determines which byte of S is selected, and E(14,15) determines which bit-pair of the selected byte will be used to set M1, M2:



M1,M2 has the same bit configuration as the bit-pair selected from S by E(12–15), and is inserted into B(64,65) [via PAL(64,65)] where it extends the high-order portion of the dividend in B.

Selection of the divisor multiple is shown in A of Sheet 4, Diagram 5-110. The factors which determine the divisor multiple are:

1. The carry condition from PAL(28) from the previous add cycle.
2. The state of T(32), which is the divisor sign bit.
3. The 'DVDL0' or 'DVDL1' micro-order.

Four divisor multiples are developed and applied as follows:

1. TL0 (+1 x T): The divisor in T is added to the partial remainder in AB.
2. TCL0 (-1 x T): The 2's complement of the divisor in T is added to the partial remainder in AB.
3. TL1 (+2 x T): The divisor in T is shifted left one bit position (in effect, doubled in value) and is added to the dividend or partial remainder in AB.
4. TCL1 (-2 x T): The 2's complement of the divisor in T is shifted left one bit position and is added to the dividend or partial remainder in AB.

The first divisor multiple is arbitrarily set to +2 x T if the divisor is negative (STAT G set) or to -2 x T if the divisor is positive (STAT G reset). This selection is done for two reasons:

1. The carry condition from PAL(28), which is normally a factor in selecting a divisor multiple, is meaningless at this time, because no previous reduction cycle has taken place.
2. If the relative magnitude of the dividend and divisor allows, the first reduction must be an overdraw. (Except for one special case, if the first reduction cycle does not result in an overdraw, the quotient and remainder will be invalid.)

Determination of the PQ bit is shown in C of Sheet 4, Diagram 5-110. The result of adding the selected divisor multiple to the dividend or partial remainder is stored into AB as a new partial remainder. The PQ bit is decoded as the inverse of A(28), and is gated to F (via SAL) per E(14,15) and the 'DVDL0' or 'DVDL1' micro-order in effect at that time. When a PQ byte has been accumulated in F, it is gated to S per the STC, where it replaces the byte of S which has already been used. When four PQ bytes have been stored into S, the complete quotient is in S and the remainder is in B.

At this point, STAT G is tested. (Recall that STAT G was set to the sign of the divisor.) If set, the divisor is negative and is in 2's complement form. Because the first reduction cycle is an attempt to overdraw the dividend, the negative divisor is shifted left 1 bit position (in effect, doubled in value) to PAA(31–62). If STAT G is reset, the divisor is positive and must be 2's complemented and shifted left one bit position to reduce the dividend. During the addition, the sign of the divisor is propagated into PAL(24–31), and the result (remainder) is gated to AB(24–67). A(28) is then tested to determine the PQ bit. A remainder in true form [A(28) = 0] causes a 1-bit to be selected for the PQ bit; a 2's complement remainder [A(28) = 1] causes a 0-bit to be selected for the PQ bit.

Two micro-orders, 'DVDL0' and 'DVDL1', are alternately used in the divide algorithm. Each micro-order has two functions: (1) to determine the location of the PQ bit in SAL (and F) from the bit-pair selected by E(14,15), and (2) to determine the shifting of the divisor multiple to PAA. [The carry condition from PAL(28) and the state of T(32) determine whether the multiple will be in true or 2's complement form.] The

'DVDLO' micro-order causes the selected PQ bit to be placed into the odd SAL bit position of the bit-pair selected by E(14,15), thus locating the PQ bit in the PQ byte being accumulated in F. This micro-order also determines that the divisor multiple will be gated to PAA(32–63) (no shift). The 'DVDL1' micro-order causes the selected PQ bit to be placed into the even SAL bit position, and also determines that the divisor multiple will be gated to PAA(31–62) (shifted left 1 bit position).

Detailed Discussion

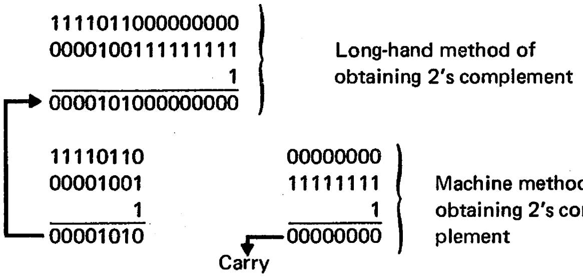
- Select M1,M2 bits from S per E(12–15).
- Insert M1,M2 bit-pair as low-order extension of high-order dividend in B.
- Select divisor multiple per: carry condition from PAL(28), divisor sign [T(32)], and 'DVDLO' or 'DVDL1' micro-order.
- Reduce dividend (or remainder) in AB' by divisor multiple selected.
- Determine PQ bits per A(28).
- Accumulate PQ bits in F to form PQ byte.
- Accumulate PQ bytes in S to form quotient.
- Determine validity of quotient and remainder.

A flowchart of the DR instruction is shown in Diagram 5-110. At the start of execution, the instruction is in E, the high-order half of the dividend is in A, B, and D, and the second operand (divisor) is in S and T. (As previously mentioned, the dividend occupies an even-odd pair of GPR's.) To correctly specify the even-odd pair of GPR's, the R1 field of the instruction must refer to an even-numbered GPR, or a program specification interruption occurs.

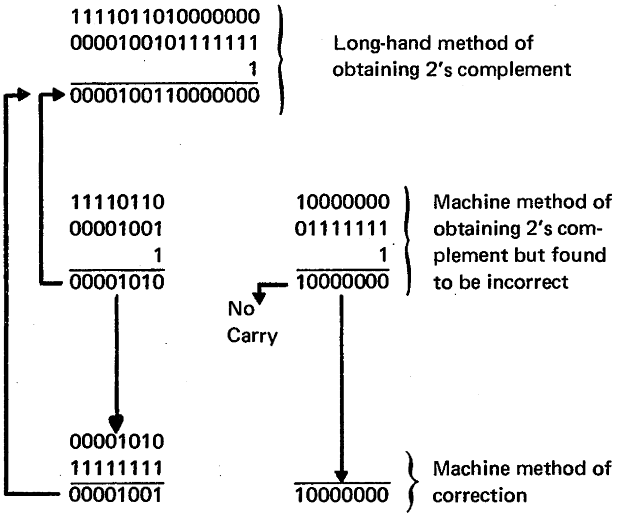
The states of B(32) and T(32) are first tested. B, at this time, contains the high-order half of the dividend, and T contains the divisor. If B(32) = 1, STAT B is set; if T(32) = 1, STAT G is set. These STAT's are used in later operations to determine the correct sign of the quotient and remainder, and to obtain a dividend in true form in B and S.

Recall that the dividend must be in true form. Because the dividend is a 64-bit operand and the maximum operable word length is 32 bits, the high-order half and the low-order half of the dividend must be treated separately when determining the true value of the dividend. To obtain a true value of a negative dividend, the dividend must be 2's complemented. Because the complementation process is always accomplished on a 2's complement basis during fixed-point operations, and because the high-order and low-order halves of the dividend are treated separately, the value of the high-order bits may be incorrect when in the true form. To prevent an incorrect high-order dividend from being operated on, the value of the low-order dividend bits is tested by checking for a carry out of the high-order bit location when 2's complemented. If a carry occurs, the high-order dividend bits are in the correct form. If a carry does not occur, the high-order bits are incorrect and a minus 1 must be added to the bits to obtain the desired dividend value. The following two examples illustrate this method (using only a 16-bit double word):

Example 1: Correct Value Obtained.



Example 2: Incorrect Value Obtained and How It Is Corrected.


1111011010000000
0000100101111111
1
0000100110000000
Long-hand method of obtaining 2's complement
11110110
00001001
1
00001010
10000000
01111111
1
10000000
No Carry
Machine method of obtaining 2's complement but found to be incorrect
00001010
11111111
00001001
10000000
Machine method of correction

Sheet 2 of Diagram 5-110 illustrates the method used by the CPU to obtain a positive dividend value in B and S.

With the correct values of the dividend in B and S and the divisor in T, E(12–15) is set to 0000 to allow selection of the first byte of the low-order half of the dividend. The first partial remainder is obtained by transferring the contents of B (high-order half of dividend) to PAB(32–63) and placing the divide multiple selection bits, M1,M2, into PAL(64,65). The divide multiple selection bits are determined by decoding E(12–15) and the S bits, as previously described.

After selection of the M1,M2 bits, E(12–15) is incremented by 1, setting up conditions for selection of the next M1,M2 bits. The resultant value of the addition of M1,M2 to the high-order half of the dividend is transferred to AB(24–67), from where it is shifted left 2 into the parallel adder, introducing the M1,M2 bits into the partial remainder.

Recall that STAT G was set to the sign value of the divisor. At this point, STAT G is tested (Sheet 3 of Diagram 5-110.) If set, the divisor is negative and is in 2's complement form, and is transferred to the parallel adder, shifting left 1 to PAA(31–62). If STAT G is reset, the divisor is positive and must be 2's complemented before being transferred to the parallel adder so that it can reduce the dividend. During the addition, the sign of the divisor is propagated into PAL(24–31) and is checked after the addition to determine the PQ-bit setting.

At this point in the operation, the 'DVDL0' micro-order is in effect and causes the first PQ bit to be gated to the odd SAL bit position of the high-order bit-pair of SAL [E(14,15) = 01] (C of Sheet 4, Diagram 5-110). The first PQ bit is extraneous, because it reflects the condition of A(28) before the first reduction cycle, and it is replaced by a valid bit 2 cycles later. The result of the first reduction is placed into AB.

A 'DVDL1' micro-order is issued next. This micro-order causes selection of a PQ bit per A(28). (AB presently contains the result of the first reduction.) If the result of the first reduction is negative [A(28) = 1], a 0 is the selected PQ bit; if positive [A(28) = 0], a 1 is the selected PQ bit. Because the 'DVDL1' micro-order is now in effect, the PQ bit is gated to the even SAL bit position of the high-order bit-pair of SAL [E(14,15) = 01], and SAL is gated to F. The contents of AB are transferred to PAB, and the divisor multiple [selected by the carry condition of PAL(28), the state of T(32), and the 'DVDL0' micro-order] is gated to PAA. The result of the addition, together with a new pair of multiple selection bits, is gated to AB as a new partial remainder. The next divisor multiple is selected per the carry condition of PAL(28), the status of T(32) (which remains the same for the entire divide operation), and the 'DVDL1' micro-order. Refer to Table 3-3 for the value of the divisor multiple for all conditions.

Selection of the PQ bits and divisor multiples occurs as just described until a complete byte of the PQ is available: (1) the 'DVDL0' micro-order is issued, and the PQ bit obtained from the last addition is stored into the odd register location determined by E(14,15), (2) the next divisor multiple is selected, placed in the adder, and added to the partial remainder, and (3) a 'DVDL1' micro-order is issued. The 'DVDL1' micro-order accomplishes the same operation as the 'DVDL0' micro-order, except that it places the PQ bit into the even GPR location per E(14,15) and shifts the selected multiple left 1. (Refer to Sheet 3, Diagram 5-110.)

When a byte of the PQ is obtained, it is transferred to S per the STC, replacing those bits of the low-order half of the dividend that have already been used. Operations continue in the same manner; i.e., PQ bits are selected to form a PQ byte, this byte is transferred to S, and the used dividend bits are replaced until a complete quotient is developed.

Thirty-two reduction cycles are provided by the fixed-point divide microprogram. To obtain a valid quotient and remainder, the absolute value of the dividend and divisor must be so related that in 32 reduction cycles the dividend can be reduced to a remainder whose value is less than that of the divisor. If the highest-order significant bit of the dividend is less than 31 bit positions to the left of the highest-order significant bit of the divisor, the quotient is in true form and a valid result is obtained. If the highest-order significant bit of the dividend is more than 30 bit positions to the left of the highest-order significant bit of the divisor, the quotient is in 2's complement form and an invalid result is obtained unless the quotient is the maximum negative number (10000 . . . . . . 000). Valid results are stored, but invalid results cause a program fixed-point divide interruption and the original operands remain unchanged in storage.

When E(14,15) = 11 and the STC = 11 (when tested), E(12–15) is incremented to 0000, the next-to-last PQ bit

Table 3-3. Divide Multiple Values, Fixed-Point

Carry from PAL(28)		T(32)		Divide Multiple Micro-Order	
Yes	No	1	0	'DVDL0'	'DVDL1'
X			X	2's complement of T.	2's complement of T shifted left 1.
X		X		T.	T shifted left 1.
	X	X		2's complement of T.	2's complement of T shifted left 1.
	X		X	T.	T shifted left 1.

is stored into F(6), and the last reduction cycle is taken (Sheet 3, Diagram 5-110). The last PQ bit is taken from the remainder in AB [A(28)] and is gated to SAL(7) [per E(14,15) = 00 and the 'DVDLO' micro-order]. This action completes the last byte in SAL, which is gated to S per the STC.

A branch on divisor sign (STAT G) and quotient form (true or 2's complement, STAT C,) causes the microprogram to enter 1 of 4 termination routines. The purpose of the termination routines is to test for valid results, to establish the quotient and remainder in the proper form according to the proper sign, to store the corrected remainder and quotient, and to end the operation.

Sheet 6 of Diagram 5-110 is a flowchart of the fixed-point divide termination routine if the quotient is in true form. In this case, the quotient and remainder are valid, and it is only necessary to store the results in the proper form. (Positive results must be stored in true form and negative results must be stored in 2's complement form, according to the convention of fixed-point arithmetic and the rules of algebra.) Because the dividend was stored in B and S in true form regardless of sign, the results may or may not be in the correct form. It is therefore necessary to branch on the original signs of the operands and on the form of the results to achieve the proper form for the results. If the remainder is in 2's complement form, it is the result of an overdraw, and must be corrected by adding the divisor in true form before storing or complementing the remainder according to the sign of the dividend.

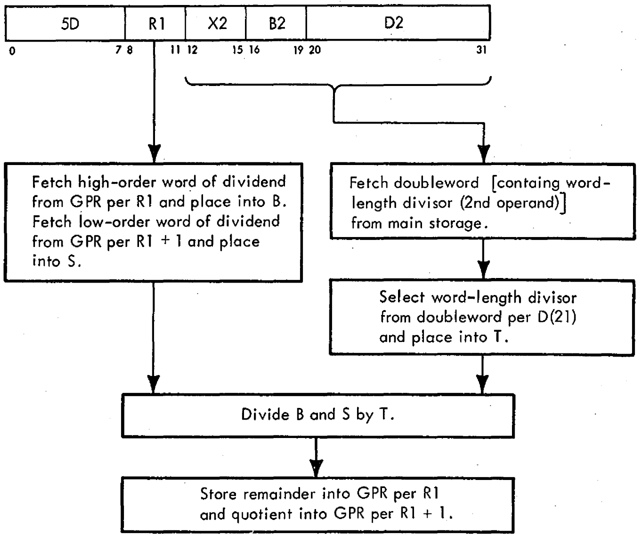
Sheet 5 of Diagram 5-110 is a flowchart of the fixed-point divide termination routine if the quotient is in 2's complement form. If the quotient is in 2's complement form and it is the maximum negative number (10000 . . . . . . 000), the results are valid and the termination routine must accomplish the same tasks as for the true form termination. If the quotient is not the maximum negative number, a program fixed-point divide interruption is taken and the original operands remain unchanged in storage.

When the quotient and remainder are valid and have been converted to the proper form according to their algebraic sign, the remainder is stored into the GPR per E(8–11) and the quotient is stored into the GPR per E(8–11) + 1, replacing the high-order and low-order halves of the dividend respectively, and leaving the divisor unchanged in storage. An end-op cycle is taken, finishing the operation.

Two examples of the fixed-point divide operation are presented in Figures 3-8 and 3-9. These examples are the inverse of the two fixed-point multiply examples (Figures 3-6 and 3-7).

Divide, D (5D)

- Divide 1st operand (in GPR per R1 and R1 + 1) by 2nd operand (in storage) and place result into 1st operand location (remainder in GPR per R1; quotient in GPR per R1 + 1).
- RX format:


5D
R1
X2
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch high-order word of dividend from GPR per R1 and place into B. Fetch low-order word of dividend from GPR per R1 + 1 and place into S.
Fetch doubleword [containg word-length divisor (2nd operand)] from main storage.
Select word-length divisor from doubleword per D(21) and place into T.
Divide B and S by T.
Store remainder into GPR per R1 and quotient into GPR per R1 + 1.

- Conditions at start of execution:
  First 16 bits of instruction are in E.
  High-order half of dividend (1st operand) is in S and T.
  Low-order half of dividend, is in GPR per R1 + 1.
  2nd operand address is in D.
  Main storage request for 2nd operand has been issued per D.
- D(21) determines which word of doubleword contains divisor: if a 1, right word; if a 0, left word.

The Divide, D, instruction divides the contents of B (high-order bits of dividend) and S (low-order bits of dividend) by the contents of T (divisor). D(21) determines which word of the doubleword fetched from main storage contains the divisor: if a 1, right word; if a 0, left word. Once the divisor is obtained from main storage, the operation is identical to the operation of the DR instruction (Diagram 5-110).

CONVERT

There are two fixed-point convert instructions, both in the RX format: Convert to Binary and Convert to Decimal. These instructions convert the radix of an operand from decimal to binary and binary to decimal, the second operand is tested to see whether the proper integral boundary has been specified. If the address is located on an incorrect (non-doubleword) integral boundary, a program specification interruption occurs. The CVB operation is suppressed, and the data in LS and in main storage remains unchanged. If no program specification interruption occurs, the operation continues.

Recall that, at the start of execution, a storage request for the second operand had been issued. At this time, the data (doubleword operand) is present at the SDBO and is gated into ST, destroying the first operand. The STC, which selects the correct byte to be converted from ST, is set to 000, thus selecting S(0–7).

The contents of T (low-order half of the doubleword operand) are transferred to the LSWR. (This data is converted at a later time.) Because the converted data is to be stored into T and D, they are now cleared. The first byte (bits 0–7) of S is transferred to the serial adder B bus. Bits 0–3 are transferred to SAL(0–3) and SAL(4–7) and on to F. F now contains the first decimal digit to be converted.

As the decimal data from S is passed through the serial adder to F, it is tested for invalid digits. If the digits are invalid, STAT E is set and later, when tested, causes a program data interruption, which terminates the operation. If the digits are valid, the first decimal digit is transferred from F(4–7) to PAB(60–63). The contents of D and T are then transferred and shifted left 1 to PAA(7–30) and PAA(31–62), respectively, and added to the decimal digit. At this time, D and T contain zero.

The result of the addition, which is the decimal digit, is transferred from PAL(8–63) to DT and from PAL(32–63) to B(32–63). The contents of DT are then transferred to PAA(8–63). The contents of B are now shifted left 2, placed into PAB(4–65), and added to PAA. This action, in effect, multiplies the original decimal digit by 5. The result of the addition is then transferred from PAL(8–63) to DT. A byte from S is transferred to the serial adder B bus per the STC. At this time, the byte transferred to the serial adder is the first byte in S (STC = 000). This action allows the second decimal digit to be placed into the serial adder. From the serial adder, the data is sent to F. The STC is increased by 1 so that the next byte from S can be transferred when selected.

The second decimal digit, F(4–7), is transferred to PAB(60–63). The contents of DT are then transferred to PAA(7–62). This transfer shifts the converted data left 1, in effect multiplying the original decimal digit by 10 (x5 and x2). PAA and PAB are added, and the result in PAL(8–63) is transferred to DT. The contents of PAL(32–63) are transferred to B(32–63).

The next byte in S (bits 8–15) is now transferred to F via SAL. STAT D is then tested. If STAT D is set, it indicates the low-order word of the doubleword operand is being converted; if reset, the high-order word is being converted. At this time, STAT D is reset. The STC is now tested to see which byte of S is being worked on; the value presently in the STC is 001. Because STAT D is reset and the STC does not equal 011, operations continue in the same manner as previously described; i.e., a decimal digit is brought in and added to the sum of the converted digits, and the result is multiplied by 10. This procedure continues until all digits have been converted.

While the last two decimal digits of the high-order word are being processed, STAT D is set to indicate the low-order word. When the last decimal digit of the high-order word has been transferred to F, the STC is set to 000; the low-order word is transferred from the LSWR (where it was stored at the beginning of the operation) to S and is converted in the same manner as the high-order word.

When the low-order byte of the low-order word is transferred from SAL to F, the sign is tested for validity, setting STAT E if invalid. The state of STAT E is then tested. If STAT E is set, a data-check condition exists and an end-op cycle is taken, leaving the contents of LS unaltered. If STAT E is not set, the sign of the number is determined by [F(4–7)], and the last decimal digit is converted. If F(4–7) is a plus sign, the converted data is transferred from T to the GPR per E(8–11). If F(4–7) is a minus sign, the converted data is first 2's complemented and then transferred from T to the GPR per E(8–11). STAT G is then set if T(32) = 1. PAL(32–63) is tested for all zero's (T = 0), and a branch is made on the result of this test.

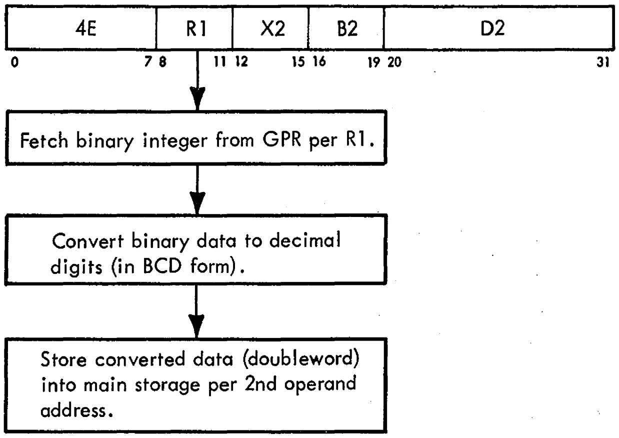
The contents of D (overflow bits) are then transferred to PAL(40–63), and PAL(32–63) is again tested for all zero's (D = 0). Note that D is not 2's complemented when T is 2's complemented for a negative sign, and should always equal zero unless an overflow occurred.

If T(32–63) = 0 and D(0–23) = 0, the result is zero and a normal end-op cycle is taken. In all cases, if D(0–23) does not equal 0, an overflow has occurred, a fixed-point divide check condition exists and an end-op cycle is taken. If D(0–23) = 0 and T(32–63) does not equal 0, a further test is made to determine if the maximum positive or negative number has been exceeded. If the decimal sign [F(4–7)] was positive and T(32) = 1 (STAT G set), the maximum positive number has been exceeded. If the decimal sign [F(4–7)] was negative and T(32) = 0 (STAT G reset), the maximum negative number has been exceeded. (For a negative sign, the contents of T have previously been changed to 2's complement form.) In both of the above cases, a fixed-point divide check condition exists and an end-op cycle is taken. If the maximum number has not been exceeded, an end-op cycle is taken, completing the operation.

Note that even if an overflow condition is detected or if the maximum positive or negative number has been exceeded, the low-order 32 bits of the converted integer are stored into the GPR per E(8–11) and the only indication is the fixed-point divide check condition. If any decimal digit or the sign is invalid, a data check condition exists and the operation is terminated without storing any data.

Convert to Decimal, CVD (4E)

- Convert radix of 1st operand (in GPR per R1) from binary to decimal and place result into 2nd operand location (in storage).
- RX format:


4E
R1
X2
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch binary integer from GPR per R1.
Convert binary data to decimal digits (in BCD form).
Store converted data (doubleword) into main storage per 2nd operand address.

- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  2nd operand address is in D.
- Operand to be converted is 32-bit signed binary integer.
- Converted data is in packed decimal format.
- Positive sign is encoded as 1100 or 1010.
- Minus sign is encoded as 1101 or 1011.

The Convert to Decimal (CVD) instruction converts the radix of the first operand (from GPR per R1) from binary to decimal, storing the result into the second operand location (in main storage). The number to be converted is a 32-bit signed binary integer; the 31 binary bits yield 15 decimal digits in the packed format. The sign bit may be encoded in two forms for both positive and negative numbers. A positive sign may be 1100 or 1010; a minus sign may be 1101 or 1011. The choice between the two sign representations is determined by PSW(12).

A binary number is converted to decimal by parallel decimal correct function, which operates from AB to the parallel adder. The binary number being converted is extracted one bit at a time from the most significant bit to the least significant bit (left to right) and is added to a previous partially converted decimal number. The resultant number is first multiplied by 2, by shifting left 1, and then decimal-corrected. The parallel decimal-correct function extracts 6 or 0 from the partially converted number and adds the 6 or 0 to twice the partially converted number so that the resulting decimal number does not exceed the maximum decimal number of 9. This function is performed for each half-byte of AB(28–63) for every add cycle. Table 3-4 lists the AB bits and parallel adder bits used in the decimal-correct function. The process of conversion is repeated until all bits of the binary number have been examined. At the completion of the conversion, the converted number is shifted left 4, and the correct sign is placed into the low-order bit positions. The example shown in Figure 3-10 illustrates the method of converting from binary to decimal.

Table 3-4. Conversion to Decimal (Excess-6)

AB Bits Set	Set PAB Bus Bits (+6)
A(28)	PAB(29,30)
A(29,30)	PAB(29,30)
A(29) and A(31)	PAB(29,30)
B(32)	PAB(33,34)
B(33,34)	PAB(33,34)
B(33) and B(35)	PAB(33,34)
B(36)	PAB(37,38)
B(37,38)	PAB(37,38)
B(37) and B(39)	PAB(37,38)
B(40)	PAB(41,42)
B(41,42)	PAB(41,42)
B(41) and B(43)	PAB(41,42)
B(44)	PAB(45,46)
B(45,46)	PAB(45,46)
B(45) and B(47)	PAB(45,46)
B(48)	PAB(49,50)
B(49,50)	PAB(49,50)
B(49) and B(51)	PAB(49,50)
B(52)	PAB(53,54)
B(53,54)	PAB(53,54)
B(53) and B(55)	PAB(53,54)
B(56)	PAB(57,58)
B(57,58)	PAB(57,58)
B(57) and B(59)	PAB(57,58)
B(60)	PAB(61,62)
B(61,62)	PAB(61,62)
B(61) and B(63)	PAB(61,62)

Diagram 5-112, FEMDM, is a flowchart of the CVD instruction. At the start of execution, the first 16 bits of the instruction are in E, the first operand is in S and T, and the second operand address is in D. The first portion of the operation is devoted to testing for specification-check and address-store-compare conditions. If a specification check is present, a program specification interruption occurs and the operation is suppressed. If an address-store-compare condition occurs, the 'PSC' trigger is set and the operation continues. The value of S(0) (sign of original binary number) is set into STAT C. At the end of the operation, STAT C is examined to determine the sign of the converted number.

Convert +146 Binary (92 Hex) to +146 BCD

1. SAL = 1001 0010 (shift left 1 after each addition).
2. Gate 6 or 0 to each half-byte of PAB per AB bits (see Table 3-4).

AB												Remarks
52	53	54	55	56	57	58	59	60	61	62	63	
0	0	0	0	0	0	0	0	0	0	0	0	Gate 0 to each half-byte of PAB.
											1	Hot carry to PAA(63) [SAL(0) = 1].
0	0	0	0	0	0	0	0	0	0	0	1	Sum to AB and DT.
0	0	0	0	0	0	0	0	0	0	1	0	Gate DT left 1 to PAA.
0	0	0	0	0	0	0	0	0	0	0	0	Gate 0 to each half-byte of PAB.
0	0	0	0	0	0	0	0	0	0	1	0	Sum to AB and DT.
0	0	0	0	0	0	0	0	0	1	0	0	Gate DT left 1 to PAA.
0	0	0	0	0	0	0	0	0	0	0	0	Gate 0 to each half-byte of PAB.
0	0	0	0	0	0	0	0	0	1	0	0	Sum to AB and DT.
0	0	0	0	0	0	0	0	1	0	0	0	Gate DT left 1 to PAA.
0	0	0	0	0	0	0	0	0	0	0	0	Gate 0 to each half-byte of PAB.
											1	Hot carry to PAA(63) [SAL(0) = 1].
0	0	0	0	0	0	0	0	1	0	0	1	Sum to AB and DT.
0	0	0	0	0	0	0	1	0	0	1	0	Gate DT left 1 to PAA.
0	0	0	0	0	0	0	0	0	1	1	0	Gate 0 or 6 to each half-byte of PAB.
0	0	0	0	0	0	0	1	1	0	0	0	Sum to AB and DT.
0	0	0	0	0	0	1	1	0	0	0	0	Gate DT left 1 to PAA.
0	0	0	0	0	0	0	0	0	1	1	0	Gate 0 or 6 to each half-byte of PAB.
0	0	0	0	0	0	1	1	0	1	1	0	Sum to AB and DT.
0	0	0	0	0	1	1	0	1	1	0	0	Gate DT left 1 to PAA.
0	0	0	0	0	0	0	0	0	1	1	0	Gate 0 or 6 to each half-byte of PAB.
											1	Hot carry to PAA(63) [SAL(0) = 1].
0	0	0	0	0	1	1	1	0	0	1	1	Sum to AB and DT.
0	0	0	0	1	1	1	0	0	1	1	0	Gate DT left 1 to PAA.
0	0	0	0	0	1	1	0	0	0	0	0	Gate 0 or 6 to each half-byte of PAB.
0	0	0	1	0	1	0	0	0	1	1	0	Result [place sign of 1100 into B(64-67) per STAT C].

0001 0100 0110 1100 = +146 in BCD packed format

Figure 3-10. Convert to Decimal Example

Because conversion is done on a positive operand basis, the sign of the operand is determined by testing T(32). If T(32) is a 1, the data to be converted is negative and its 2's complement form must be derived; if a 0, the data is positive. The contents of D are then shifted left 4 and transferred to the LSWR. Because the high-order converted data is stored in D, D is cleared. The first byte of data is now sent from S to SAL per the STC (STC = 0).

The first decimal convert value (0 or 6) is obtained by examining the contents of AB. Because AB is cleared at this time, the first decimal convert value is all 0's. (See Table 3-4 for conversion values.) The decimal convert value is placed into PAB(28–63), and a hot carry is generated if SAL(0) = 1. If SAL(0) = 0, a hot carry is not generated and the value in PAB is transferred directly to PAL. The contents in SAL are shifted left 1, thus placing the next bit from the byte into SAL(0). Next, PAL(8–63) is transferred to AB(8–63), PAL(8–31) is transferred to D(0–23), and PAL(32–63) is transferred to T(32–63). This action places the first converted bit into AB and DT. The value in DT is then shifted left 1 to PAA(7–63). A decimal convert value is again obtained from B and placed into PAB. SAL(0) is now checked. If it is a 1, a hot carry is generated; if a 0, no carry is generated. In either case, the numbers are added. The result of the addition is then transferred to DT and AB. The contents of SAL are shifted left 1, bringing in the next bit for conversion. The ABC is increased by 1.

The contents of SAL (a byte of S per the STC) are shifted left one digit position as follows: F is gated to SAA, the byte of S (per the STC) is gated to SAB, and SAL is gated to F and S (per the STC). This operation is equivalent to adding the contents of SAL to itself, which doubles the value of SAL and, in effect, shifts the bits one digit position to the left. After seven add cycles, every bit in SAL has been presented in sequence to PAA(63) via SAL(0), the original low-order bit of F (and of the byte of S) has been shifted to the high-order bit position of F and of the byte of S, and the STC has been incremented by 1 to present the next byte of S to SAL and F. To assure that the new byte of S presented to SAB will be added to all zeros, F(4–7) is gated to SAA(0–3) and all zeros are gated to SAA(4–7).

The ABC indicates the progress of the microprogram in converting bits of the byte from S (now located in SAL, F, and S). When the ABC = 4, five bits of the byte have been converted, the microprogram has branched (per ABC = 3) to a routine to convert the last three bits of the byte, the ABC is set to 0, and the STC is incremented by 1 to present the next byte of S to SAL and F. One conversion cycle is taken and the microprogram re-enters the conversion loop, converting the next byte of S in the same manner as the first. This routine continues until, at the time the microprogram branches from the conversion loop, the STC = 3. The microprogram then enters a termination routine, during which the last three bits of the low-order byte of S are converted and the sign is set per STAT C. (Recall that STAT C was set to the value of the sign of the binary number.) If STAT C is set, the last decimal convert value (converting bit 7 of the last byte) is obtained and the hot carry is generated according to the value of SAL(0); the result is placed into AB and DT. A minus sign (1101 or 1011) is then placed into B(64–67). AB is shifted left 4 and transferred to ST(0–63). Mark triggers 0 through 7 are set, and the converted data is transferred to main storage. An end-op cycle is taken, completing the operation. If STAT C is reset, a positive sign is placed into B(64–67), shifted into the low-order bits of the converted operand, and stored into main storage.

The CVD instruction is the only instruction which uses the excess-6 gates from B to PAB. (For a discussion of the excess-6 gates, refer to "Parallel Adder" in Chapter 2.) Note in the example in Figure 3-10 that the result of each addition is gated from PAL to AB and to DT; then the contents of DT are gated left 1 to PAA (in effect, doubled in value). By examining the bit configuration of each half-byte of AB, a decision is made whether to add 6 to the corresponding half-byte of DT after being shifted left 1 to PAA. This decision is a prediction whether or not each half-byte of AB, if doubled in value, will exceed 9. Each half-byte of AB which has a value of 5 or greater will, if doubled, exceed 9. Therefore, for decimal correction purposes, 6 must be added to the corresponding half-bytes of DT.

For simplicity, the low-order half-byte of B is taken as an example in Table 3-5; however, each half-byte of AB (28–63) is simultaneously examined. Note in the table that if B(60–63) contains a value of 0 through 4, the value, when doubled, will *not* exceed 9 and a 6 is *not* added. If B(60–63) contains a value of 5 through 15, however, its value, when doubled, *will* exceed 9 and a 6 *must* be added. Note that for values 5 through 15 B(60–63) contains one or more of the following bit combinations: B(60) = 1, B(61,62) = 11, B(61) and B(63) = 1.

STORE

There are three fixed-point store instructions: Store, ST, RX format; Store Halfword, STH, RX format; and Store Multiple, STM, RS format. The function of the store instructions is to store the contents of specified GPR(s) into main storage.

Store, ST (50)

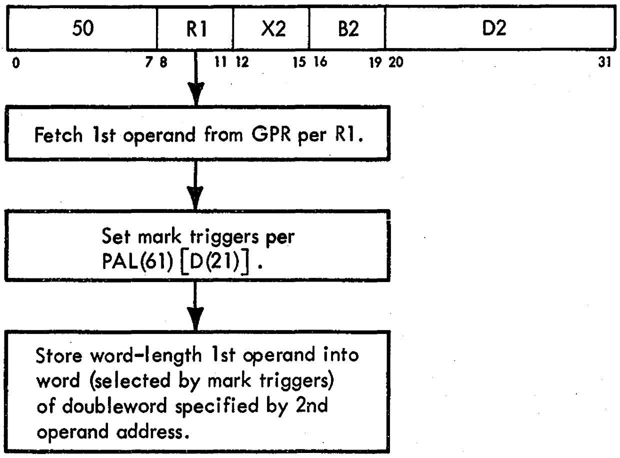
- Store 1st operand (in GPR per R1) into 2nd operand location (in storage).

Table 3-5. Excess-6 Conversion, B(60–63)

Decision Making Factors (ALD RB753)	B-bits				Decimal Value	Decimal Value if Doubled	Add to PAB(60–63)
	60	61	62	63			
None of the below conditions	0	0	0	0	0	0	0
	0	0	0	1	1	2	
	0	0	1	0	2	4	
	0	0	1	1	3	6	
	0	1	0	0	4	8	
B(61) and B(63) = 1	0	1	0	1	5	10	6 [1 to PAB(61,62)]
B(61,62) = 11	0	1	1	0	6	12	
	0	1	1	1	7	14	
B(60) = 1	1	0	0	0	8	16	
	1	0	0	1	9	18	
	1	0	1	0	10	20	
	1	0	1	1	11	22	
	1	1	0	0	12	24	
	1	1	0	1	13	26	
	1	1	1	0	14	28	
	1	1	1	1	15	30	

Refer to Table 3-4 for remaining half-bytes of AB.

- RX format:



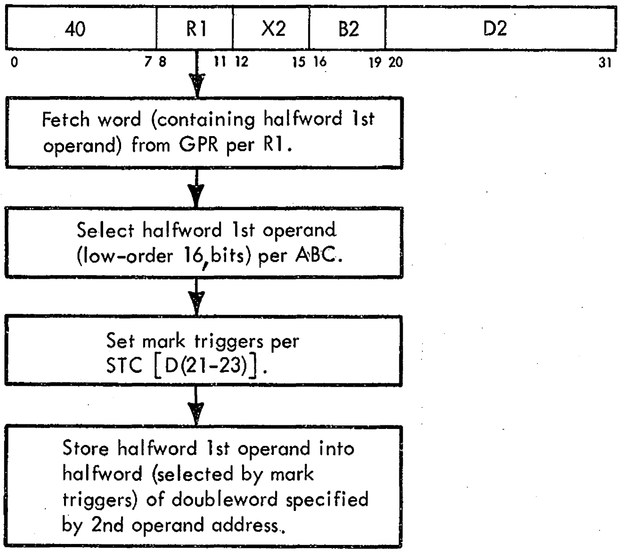
50
R1
X2
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch 1st operand from GPR per R1.
Set mark triggers per PAL(61) [D(21)].
Store word-length 1st operand into word (selected by mark triggers) of doubleword specified by 2nd operand address.

- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  2nd operand address is in D.
- PAL(61) determines into which word of doubleword 1st operand is to be stored: if a 1, right word; if a 0, left word.

The Store, ST, instruction stores the first operand (from the GPR per R1) into the main storage address specified by the second operand address. Diagram 5-113, FEMDM, is a flowchart of the ST instruction. PAL(61) determines which word (right or left) of the doubleword addressed by D will receive the first operand; mark triggers 0–3 or 4–7 are set accordingly.

Store Halfword, STH (40)

- Store halfword 1st operand (in GPR per R1) into 2nd operand location (in storage).
- RX format:



40
R1
X2
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch word (containing halfword 1st operand) from GPR per R1.
Select halfword 1st operand (low-order 16 bits) per ABC.
Set mark triggers per STC [D(21-23)].
Store halfword 1st operand into halfword (selected by mark triggers) of doubleword specified by 2nd operand address.

- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  2nd operand address is in D.
- ABC selects 16 low-order bits of 1st operand for storage; high-order bits are ignored.
- STC [D(21–23)] positions 16 low-order bytes for storage.

The Store Halfword (STH) instruction stores the 16 low-order bits of the GPR specified by R1 into the main storage location specified by the second operand address. The high-order bits are not used.

The selected halfword is stored through the use of mark triggers, which reflect the value of D(21–23). This value, plus 1, signifies into which portion of the doubleword the halfword is to be stored. Diagram 5-114, FEMDM, is a flowchart of the STH instruction. At the start of execution, the first 16 bits of the instruction are in E, the first operand is in S and T, and the second operand address (into which the halfword operand is to be stored) is in D.

The second operand address is first tested to see that it is on an integral boundary; if not, a program specification interruption occurs and the operation is suppressed. Assuming no specification error, D(21–23) is transferred to the STC to select the correct portion of the main storage doubleword into which the 16 low-order bits of the first operand are to be stored. Next, the first operand (located in T) is transferred to B. The ABC is then set to 6 by placing all 1's into the ABC and subtracting 1 from this value. (The ABC selects the two low-order bytes of the operand presently located in B.) The mark trigger, which transfers the selected high-order byte of the halfword from ST to the main storage location, is selected per the STC. (See Table 3-6 for the STC and mark trigger settings and the corresponding operand bits transferred.) The eight high-order bits of the halfword are now transferred from B(48–55) to the correct position via the serial adder and the STC. A 3-cycle storage request is given. (Three cycles later, the data in ST is stored into main storage.) Also at this time, the ABC and STC are increased by 1 to select the next byte of data and to position the byte into ST by the time the 3-cycle storage request has elapsed.

To determine whether this store operation modified the instruction to be executed next, an address-store-compare test is made by comparing the IC with the main storage address used in the store operation. The test is made by transferring the 2's complement of D (address of main storage doubleword into which the halfword operand is to be stored) to PAA(40–63). The contents of the IC are transferred to PAB(40–63); PAA and PAB are added, and the result is shifted right 4 in PAL. The PAL is tested for 0; if 0, an address-store-compare condition exists and the 'PSC' trigger is set. This trigger is tested

Table 3-6. Operand Bits Transferred, STH Instruction

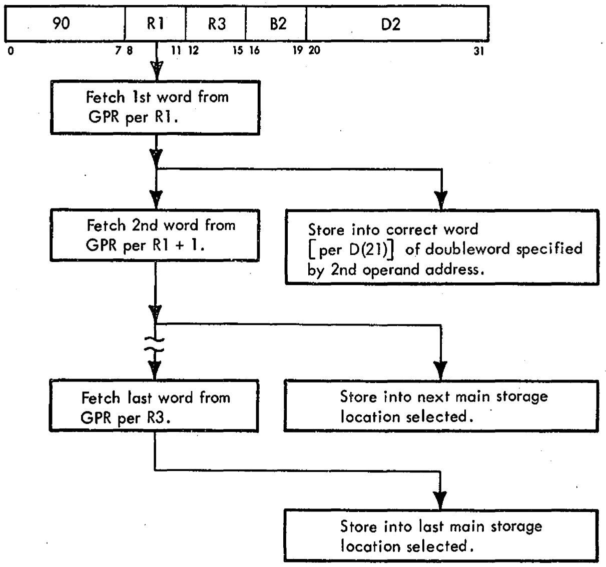
STC			Mark Trigger	Operand Bits Transferred
D(21)	D(22)	D(23)		
0	0	0	0	0–7
0	1	0	2	16–23
1	0	0	4	32–39
1	1	0	6	48–55
STC +1				
0	0	1	1	8–15
0	1	1	3	24–31
1	0	1	5	40–47
1	1	1	7	56–63

during the I-Fetch sequence of the next instruction and, if set, causes the modified instruction to be fetched from main storage and reloaded in Q. (Refer to "ASC Test" in Section 1 of this Chapter for an explanation of the address-store-compare test during I-Fetch.)

At this point, the eight low-order bits of the halfword operand are transferred from B(56–63) to ST via the SAL and STC. The associated mark triggers are also set at this time per the STC. An end-op cycle is taken to complete the operation. If a protection check condition occurs, the operation is suppressed, because no data storage takes place. Instead, the next instruction is fetched and executed, followed by a program interruption due to the "late" protection check.

Store Multiple, STM (90)

- Store into 2nd operand location (as many words as required, in storage) contents of GPR's, in ascending order, starting with 1st operand location (per R1) and ending with 3rd operand location (per R3).
- RS format: (See adjoining column.)
- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  2nd operand address is in D.
- Number of words to be stored is determined by E(8–11) and E(12–15).
- Addressed GPR's wrap around from 15 to 0.
- D(21) determines into which word of doubleword the first word is to be stored: if a 1, right word; if a 0, left word.



90
R1
R3
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch 1st word from GPR per R1.
Fetch 2nd word from GPR per R1 + 1.
Store into correct word [per D(21)] of doubleword specified by 2nd operand address.
Fetch last word from GPR per R3.
Store into next main storage location selected.
Store into last main storage location selected.

The Store Multiple (STM) instruction stores one or more 32-bit words from LS, starting with the GPR specified by R1 and ending with the GPR specified by R3, into main storage. The area in main storage where the contents of the GPR's are placed starts at the location designated by the second operand address and continues through as many words as needed in an ascending order.

The number of words to be stored is determined by E(8–11) and E(12–15). If the contents of these bit locations are equal, only one word is to be stored. If the contents of the bit locations are not equal, storage of words is continued; E(8–11) is updated by 1 for each word stored until the bit locations are equal; then, one more word is stored, and the operation is completed. Once it has been decided whether one or more words are to be stored, it must be determined into which word of the doubleword, in main storage, the first word is to be stored; D(21) serves this function. If D(21) = 0, the first word is to be placed into the left word of the doubleword; if D(21) = 1, into the right word.

See Diagram 5-115, FEMDM, a flowchart of the STM instruction. At the start of execution, the first 16 bits of the instruction are in E, the first operand is in ST, and the second operand address is in D. The instruction first tests for a specification-check condition. If one exists, a program specification interruption occurs and the operation is suppressed. Assuming there is no specification check, the contents of D are transferred to PAA(40–63). A 3-cycle storage request is given. To determine whether one or more words are to be stored, E(8–11), R1, is compared with E(12–15), R3. If E(8–11) equals E(12–15), only one word is to be stored; if it does not, more than one word is to be stored.

Assume one word is to be stored [E(8–11) equals E(12–15)]. D(21) is tested to determine whether the LS operand is to be stored into the left or right word of the main storage doubleword. If D(21) = 0, the left word is selected; if D(21) = 1, the right word is selected. Assume D(21) = 1. Mark triggers 4 through 7 are set to gate T(32–63) to bits 32–63 of the main storage doubleword when data is stored. An address-store-compare test is then made. This test involves transferring the 2's complement of D to PAA(40–63) and 7 to PAA(61–63), and transferring the contents of the IC to PAB(40–63); they are added and shifted right 4 to PAL. The PAL is then tested for zero. If zero, the 'PSC' trigger is set. This trigger is tested during end op and, if set, indicates that the next instruction to be executed has been modified. The modified instruction must then be refetched into Q during I-Fetch. (For information about the address-store-compare test, refer to "ASC Test" in Section 1 of this chapter.)

A protection test is also made by main storage while the next instruction is being fetched. The protection key in the PSW is compared with the storage key for the location. If the keys agree, storage is permitted. If the keys do not agree, storage is not permitted, the instruction is terminated, and a "late" protection interruption is taken after the execution of the next instruction.

Now assume D(21) = 0 and E(8–11) equals E(12–15). Again only one word is to be stored into main storage. In this case, however, the word is to be stored in the left word of the doubleword location in main storage. Accordingly, mark triggers 0 through 3 are set to gate S(0–31) to bits 0–31 in main storage.

Now assume E(8–11) is not equal to E(12–15). If D(21) = 1, the first word is to be stored into the right word of the doubleword (mark triggers 4 through 7 are set). Because more than one word is to be loaded, the next sequentially addressed word from LS is transferred to S. A storage request is then given. The contents of D are increased by 8. E(8–11) and E(12–15) are again compared. If equal, mark triggers 0 through 3 are set and the data is gated into main storage. If they are not equal, more than two words are to be stored into main storage. Because the second operand to be gated from LS is already in S, and because D contains the address where both the second and third operands are to be stored, the third operand is now transferred out of LS and mark triggers 0 through 7 are set. The data is stored into main storage. Address-store-compare and protection-key tests are made for each new main storage address. Only the last storage request issued can cause a "late" protection interruption.

If more than two words are to be stored, the operation continues as just described. E(8–11) is incremented, a word is loaded into ST, the address in D is increased by 8, and storage requests are generated when needed. When E(8–11) and E(12–15) are equal, the last word is loaded into main storage and an end-op cycle is taken.

If D(21) = 0, and if E(8–11) and E(12–15) are not equal, the first word is to be placed into the left word of the doubleword location in main storage and the second word into the right word.

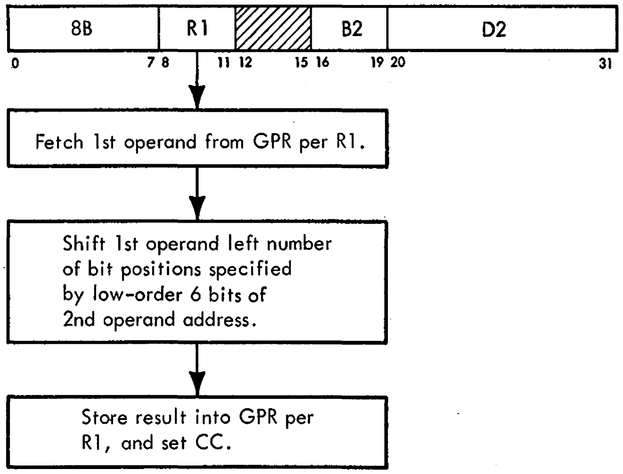
SHIFT

There are four fixed-point shift instructions: Shift Left Single, SLA; Shift Left Double, SLDA; Shift Right Single, SRA; and Shift Right Double, SRDA. Their function is to shift the first operand (right or left) and to store the result into the first operand address. The first operand may be a word or a doubleword in length and is shifted the amount specified by the low-order six bits of the second operand address. The specified amount of shift is accomplished in increments of left 1, left 2, left 4, and right 4 shifts. To expedite the operation, combinations of these increments are used whenever possible, and a maximum number of left 4 and right 4 shifts are used.

The CC is set to indicate the status of the result. A left-shifted result is tested for an overflow condition to determine if significant high-order digits were lost. During a right-shift, significant low-order digits may be lost with no indication.

Shift Left Single, SLA (8B)

- Shift 1st operand (in GPR per R1) left number of bit positions specified by low-order 6 bits of 2nd operand address and place result into 1st operand location.
- RS format:


8B
R1
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch 1st operand from GPR per R1.
Shift 1st operand left number of bit positions specified by low-order 6 bits of 2nd operand address.
Store result into GPR per R1, and set CC.

- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  Amount of shift is in D and PAL.
- D(18–23) indicates total amount of shift.

- Methods of shifting:
  Left 1 from T to PAA.
  Left 2 from AB to PAB.
  Left 4 from PAA or PAB to PAL.
- E(12–15) indicates number of left 4 shifts to be performed.
- E(12–15) is reduced by 1 after each multiple of four shifts occurs.
- 0's are supplied to vacated bit positions.
- Overflow occurs if data is shifted out of bit position 1.
- CC setting:
  Result in PAL is zero: CC = 0.
  Result in PAL is less than zero: CC = 1.
  Result in PAL is greater than zero: CC = 2.
  Overflow: CC = 3.

The Shift Left Single, SLA, instruction shifts the first operand left the number of bit positions specified by the low-order six bits of the second operand address. (The remainder of the second operand address is ignored.) The second operand request per D, normally initiated during RS I-Fetch, is blocked during I-Fetch of a shift instruction because D does not contain a main storage address. Refer to "Basic RS and SI I-Fetch" in Section 1 of this chapter.

The sign of the first operand remains unchanged. All 31 bits of the operand participate in the left shift. Zeros are transferred into the vacated low-order register positions. If a bit unlike the sign bit is shifted out of position 1, an overflow occurs, causing a program fixed-point overflow interruption during end op if the fixed-point overflow mask bit is a 1.

Left-shifting can be accomplished by three methods: (1) shifting left 1 bit position from T to the parallel adder, (2) shifting left 2 bit positions from AB to the parallel adder, and (3) shifting left 4 bit positions from the parallel adder to PAL. In the interest of speed, the desired amount of left shift is accomplished using a maximum number of left 4 shifts and a minimum number of left 1 and left 2 shifts (not more than one of each). Also, whenever possible, a left 1 or a left 2 shift is combined with a left 4 shift because these combinations can be accomplished in one pass through the parallel adder. Table 3-7 shows how left-shifting is accomplished for any amount of shift desired.

See Diagram 5-116, FEMDM, a flowchart of the SLA instruction. At the start of execution, the first 16 bits of the instruction are in E, the first operand is in S and T, and the number of bit positions to be shifted is in D and PAL.

In the SLA instruction, PAL(58–63) is first tested to determine the amount of shift. (PAL contains the same value as D.) If PAL(58–61) = 0, a shift of less than 4 is needed; PAL(62) is then tested. If PAL(62) = 1, a shift of either 3 or 2 is to be performed; if PAL(62) = 0, either a shift of 1 or no shift is to be performed.

As an example, assume that PAL(58–61) does not equal 0 and that PAL(62,63) = 11. A shift of left 7, left 11, or more is indicated. STAT C is set to S(0), the sign of the

Table 3-7. Left Shift Combinations

PAL(58–61)	PAL(62,63)	Total Shift Desired	Incremental Shifting Sequence		
0000	00	None	None		
	01	Left 1	Left 1		
	10	Left 2	Left 2		
	11	Left 3	Left 1	Left 2	
Not 0000	00	Left 4, 8, 12 and so on	Left 4	Left 4 until E(12–15) = 1	
	01	Left 5, 9, 13 and so on	Left 1 and left 4	Left 4 until E(12–15) = 1	
	10	Left 6, 10, 14 and so on	Left 2 and left 4	Left 4 until E(12–15) = 1	
	11	Left 7, 11, 15 and so on	Left 1 and left 4	Left 2 if E(12–15) = 1	
				Left 2 and left 4 if E(12–15) ≠ 1	Left 4 until E(12–15) = 1

first operand that will be transferred to LS at the end of the operation. D(18–21), which is equal to PAL(58–61), is transferred to E(12–15). This value is reduced sequentially by 1 after every left 4 shift until it equals 0001, at which time no more shifting of data is necessary. The contents of T are transferred to PAA(31–62), thus achieving a left 1 shift. T(32) is propagated into PAL(26–31). Because a left 4 shift takes place only when transferring data from PAA or PAB into PAL, PAA(8–67) is shifted left 4 positions into PAL(4–63). PAL(26–32) is now tested to see that no important data has been lost during the shifting. If these bits are not all 0's or all 1's, STAT B is set, indicating overflow occured; if found set during end op, a program fixed-point overflow interruption occurs if the associated overflow mask bit in the PSW is a 1. PAL(24–67) is transferred to AB(24–67). E(12–15) is tested to see whether it contains a value of 0001. If it does, only a left 7 shift is to be performed; otherwise, a shift of left 11 or more is to be made.

Assume E(12–15) = 0001. A 1 is subtracted from E(12–15). The contents of AB(6–67) are transferred to PAB(4–65). This transfer accomplishes a left 2 shift, making a total left shift of 7 positions. PAL(26–32) is tested for overflow, and PAL(32–63) is tested for all 0's. The data is transferred to LS, and an end-op cycle is taken to complete the operation.

Now assume E(12–15) does not equal 0001, indicating a total left shift of at least 11 positions or more is called for by the instruction. Because at this point the data has already been shifted left 5 positions, a left 6 shift is necessary to achieve the minimum left 11 shift. A 1 is subtracted from E(12–15). AB(6–67) is transferred to PAB(4–65), yielding a left 2 shift. A left 4 shift is achieved by shifting the contents of PAB(4–65) to PAL(0–61). PAL(26–32) is tested for overflow, and PAL(32–63) is tested for all 0's. PAL(32–63) is transferred to T(32–63), and T(32–63) is transferred to the GPR per E(8–11). The value of STAT C is placed into the sign position of the GPR. E(12–15) is again checked for a value of 0001. If the bits now contain this value, the data has been shifted the correct number of times. A 1 is subtracted from E(12–15), and an end-op cycle is taken, completing the operation. If E(12–15) does not equal 0001, a left shift of more than 11 is required for this instruction.

If E(12–15) still does not equal 0001, 1 is again subtracted from E(12–15). T(32–63) is transferred to PAA(32–63), and the data is shifted left 4 positions into PAL(28–63). After testing PAL(26–32) and PAL(32–63), the contents of PAL(32–63) are transferred to T, and the data in T(33–63) is transferred into the GPR per E(8–11). The value of STAT C is placed into the GPR sign position per E(8–11).

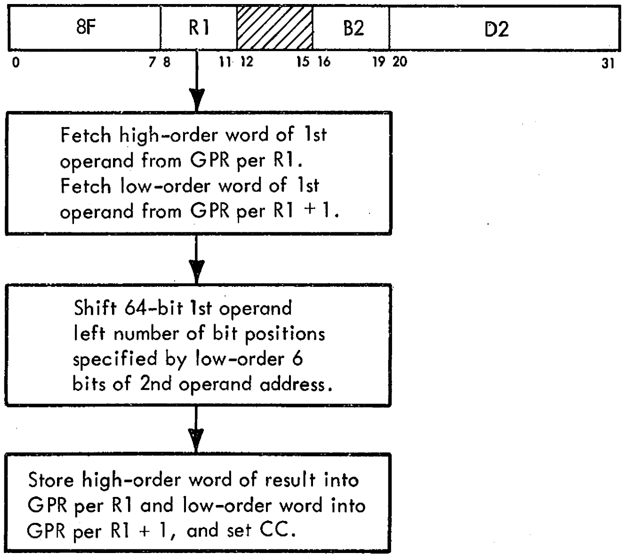
Note: The result of the first 11 shifts is transferred to LS before E(12–15) is tested to see whether a shift of more than 11 places is required. If a total shift greater than 11 is specified, the data is shifted left an additional four places and transferred to LS, where it destroys the 11-place shifted operand stored earlier. E(12–15) is tested; if a shift greater than 15 in specified, the data is again shifted left 4 and transferred to LS. This procedure of testing E(12–15), shifting left 4, and transferring the result to LS continues until E(12–15) equals 0001.

E(12–15) is again checked for 0001. If E(12–15) = 0001 at this time, 1 is subtracted from it and an end-op cycle is taken, completing the operation. If E(12–15) does not equal 0001, the operand continues to be shifted in multiples of 4 until E(12–15) equals 0001. The CC is then set per STAT A, STAT B, and the result sign [T(32)], and an end-op cycle is taken.

Left shifts of other amounts are performed in a similar manner. (Refer to Table 3-7 and Diagram 5-116, FEMDM.)

Shift Left Double, SLDA (8F)

- Shift 1st operand (in GPR per R1 and R1 + 1) left number of bit positions specified by low-order six bits of 2nd operand address and place result into 1st operand location.
- RS format:


8F
R1
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch high-order word of 1st operand from GPR per R1. Fetch low-order word of 1st operand from GPR per R1 + 1.
Shift 64-bit 1st operand left number of bit positions specified by low-order 6 bits of 2nd operand address.
Store high-order word of result into GPR per R1 and low-order word into GPR per R1 + 1, and set CC.

- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  Amount of shift is in D and PAL.
- D(18–23) indicates total amount of shift.
- E(12–15) indicates number of left 4 shifts to be performed.
- E(12–15) is reduced by 1 after each multiple of four shifts occurs.

- High-order bits of low-order word of doubleword operand are saved and placed into high-order word operand.
- 0's are supplied to vacated bit positions.
- Overflow occurs if data is shifted out of bit position 1 of high-order word.
- CC setting:
  Result in PAL is zero: CC = 0.
  Result in PAL is less than zero: CC = 1.
  Result in PAL is greater than zero: CC = 2.
  Overflow: CC = 3.

The Shift Left Double, SLDA, instruction shifts the doubleword first operand left the number of bit positions specified by the low-order six bits of the second operand address. The R1 field of the instruction is the address of the GPR containing the high-order 32 bits of the doubleword operand. The low-order word of the doubleword operand is in the GPR per R1 + 1. The R1 field of the instruction must specify an even register. When R1 is odd, a program specification interruption occurs.

The sign of the doubleword operand is the sign of the even GPR. The high-order bit (sign) of the odd GPR is treated as an integer bit. As the information is shifted, 0's are supplied to the vacated low-order positions. If a bit unlike the sign bit is shifted out of bit position 1 of the high-order word of the doubleword operand, an overflow occurs. The overflow causes a program fixed-point overflow interruption if the associated mask bit in the PSW is a 1.

The SLDA instruction is similar to the SLA instruction in that a left 1 shift occurs when transferring data from T to PAA, a left 2 shift occurs when transferring data from AB to PAB, and a left 4 shift occurs when data is transferred from the inputs of the parallel adder to PAL. Also, the total shift specified is accomplished using a maximum number of left 4 shifts and a minimum number of left 1 and left 2 shifts. The differences are a result of handling a doubleword, one word at a time, in the SLDA instruction.

B(64–67), which contains the overflow bits from the low-order word, may be thought of as a four-bit register inserted between the high-order word and the low-order word. It is therefore necessary to shift these bits four bit positions to the left to position them correctly into the high-order word. This positioning is accomplished in different ways according to the shift being performed; the result, however, is always that of shifting the overflow bits left four bit positions.

Refer to Diagram 5-117, FEMDM, a flowchart of the SLDA instruction. At the start of execution, the first 16 bits of the instruction are in E, the high-order word of the first operand is in S and T, and the number of bit positions to be shifted is in D and PAL.

In the beginning of the operation, the sign of the operand (which will be transferred to LS at the end of the operation) is stored into STAT C. Because the low-order word of the doubleword operand is treated first, that word is now transferred from LS. (Recall that S and T presently contain the high-order word of the doubleword.) Once the low-order word of the doubleword is obtained, the data is shifted as for the SLA instruction. The high-order bits of the low-order word of the doubleword are not lost when shifted out but are transferred to B(64–67). When the high-order word of the doubleword is obtained, these bits are shifted into the high-order word of the operand.

The low-order word is shifted the amount specified by PAL(62,63), and the low-order 32 bits of the result are stored in the GPR specified by R1 + 1 (odd register). A zero test is performed, and STAT A is set if the low-order word is all zeros. The high-order bits that were shifted out of the low-order word are gated from PAL(28–31) to B(64–67) where they are stored, shifted left four positions and appended to the high-order word. The high-order word is transferred from the even GPR specified by R1, is shifted the same amount as the low-order word per PAL(62,63), and is added to the overflow bits from the low-order word. An overflow test is performed to determine if significant bits were lost from the high-order word, and STAT B is set if an overflow occurred. A zero test is performed, and STAT A is reset if the high-order result is not all zeros.

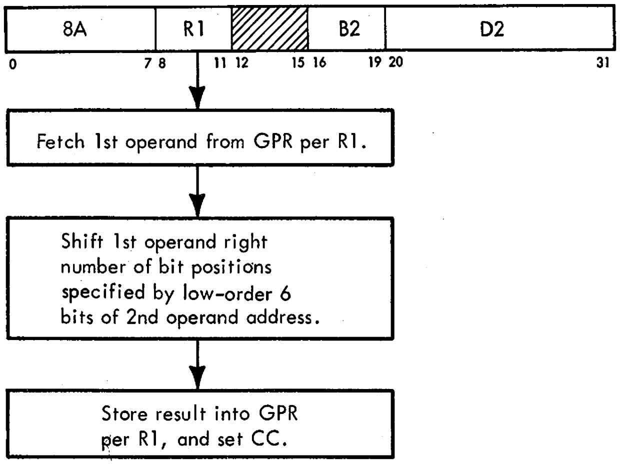
If PAL(58–61) was zero at the end of I-Fetch, the last shift has been performed and the low-order 31 bits of the high-order word, plus the inserted sign bit (per STAT C), are stored into the even GPR specified by R1. The CC is set per hardware conditions, and an end-op cycle is taken.

If PAL(58–61) was not zero, the low-order word has been transferred from the odd GPR to S, and one or more left 4 shifts remain to be performed. The low-order word in S is shifted left 4 and stored into S. The four overflow bits are retained in B(64–67), the high-order word in T is gated to PAA, B(64–67) is gated to PAB, and the result is shifted left 4 to PAL. This sequence shifts the high-order word, plus the overflow bits from the low-order word, left 4. The low-order 31 bits of the high-order word, plus the inserted sign bit (per STAT C), are gated to T and stored into the even GPR per R1. The number of left 4 shifts to be performed is determined by E(12–15), which was set per D(18–21) at the start of execution. E(12–15) is decremented after each left 4 shift is performed. The microprogram remains in the shift left 4 loop until E(12–15) is reduced to 0001, at which time the low-order word is stored into the odd GPR specified by R1 + 1. The CC is set per STAT A, STAT B, and the result sign [A(0)], and an end-op cycle is taken.

When the low-order word is shifted, a zero test is performed and STAT A is set if the low-order result is all zeros. When the high-order word is shifted, a zero test is performed and STAT A is reset if the high-order result is not all zeros. When the high-order word is shifted, an overflow test is performed; STAT B is set if an overflow occurred and a program fixed-point overflow interruption is taken if the associated mask bit in the PSW is a 1.

Shift Right Single, SRA (8A)

- Shift 1st operand (in GPR per R1) right number of bit positions specified by low-order six bits of 2nd operand address and place result into 1st operand location.
- RS format:


8A
R1
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch 1st operand from GPR per R1.
Shift 1st operand right number of bit positions specified by low-order 6 bits of 2nd operand address.
Store result into GPR per R1, and set CC.

- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  Amount of shift is in D and PAL.
- D(18–23) indicates total amount of shift.
- Method of shifting: right 4 from PAA or PAB to PAL.
- Shifts of right 3 or less are obtained by combining left 1, left 2, or left 3 shifts with right 4 shift.
- E(12–15) indicates number of shifts to be performed in multiples of 4.
- E(12–15) is reduced by 1 after each shift of 4.
- CC setting:
  Result in PAL is zero: CC = 0.
  Result in PAL is less than zero: CC = 1.
  Result in PAL is greater than zero: CC = 2.

The Shift Right Single, SRA, instruction shifts the first operand right the number of bit positions specified by the low-order six bits of the second operand address. (The remainder of the address is ignored.)

The sign of the first operand remains unchanged. All 31 bits of the operand participate in the shift. Bits equal to the sign are supplied to the vacated high-order bit positions. Low-order bits are shifted without inspection and are lost.

Right-shifting is accomplished only be shifting right 4 bit positions at a time from the parallel adder to the PAL. Right shifts of less than 4 are obtained by combining left 1, left 2, or left 3 shifts with the right 4 shift. In the interest of speed, the desired amount of right shift is accomplished using a maximum number of right 4 shifts and a minimum number of left 1 and left 2 shifts (not more than one of each). Also, wherever possible, a left 1 or left 2 shift is combined with a right 4 shift because these combinations can be accomplished in one pass through the parallel adder. Table 3-8 shows how right-shifting is accomplished for any amount of shift desired.

Diagram 5-118, FEMDM, is a flowchart of the SRA operation. At the start of execution, the first 16 bits of the instruction are in E, the first operand is in S and T, and the number of bit positions to be shifted is in D and PAL.

In the SRA operation, PAL(58–63) initially determines the amount of shift. PAL(58–61) determines whether a shift of more than right 3 is to occur. If PAL(58–61) = 0, a shift of right 3 or less is to be performed; if PAL(58–61) does not equal 0, then shifts of right 4 or more are to occur.

As an example, assume that PAL(58–61) does not equal 0 and that PAL(62,63) = 01. A shift of right 5, right 9, or more is indicated. D(18–21) is transferred to E(12–15) to determine the number of right 4 shifts to be used if a shift of more than right 5 is to occur. PAL(32–63) is now tested for all 0's. If this condition is present, STAT A is set. The first operand in T(32–63) is transferred to AB, with T(32) propagated into A(26–31).

The first operand in T(32–63) is transferred to PAA(31–62), causing a left 1 shift of the data, and T(32) is propagated into PAL(26–31). PAA(31–62) and PAL(26–31) are transferred right 4 positions to PAL(35–66) and PAL(30–35), respectively, giving, in effect, a shift of right 3 positions. The data in PAL is transferred to AB(24–67), from where it is transferred to PAB(4–65), causing a left 2 shift. The total effective shift at this time is a right 1 shift. PAL(32–63) is now tested for all 0's, and STAT A is set if all 0's are present. PAL(32–63) is transferred to T(32–63), from where the data is transferred into the GPR per E(8–11). E(12–15) is tested for 0001. If this value exists, an end-op cycle is taken to complete the operation. If E(12–15) does not equal 0001, then the data continues to be shifted in multiples of 4, and E(12–15) is reduced by 1 for each right 4 shift until it equals 0001. At this time the CC is set per STAT A and the result sign [T(32)], and an end-op cycle is taken.

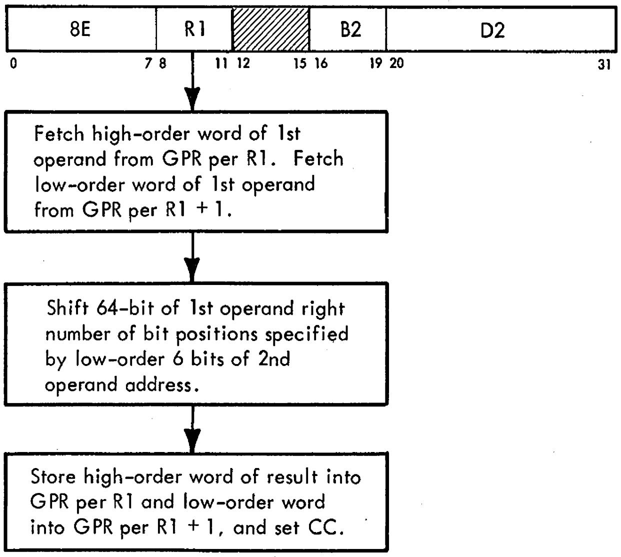
Right shifts of other amounts are performed in a similar manner. (Refer to Table 3-8 and Diagram 5-118, FEMDM.)

Table 3-8. Right Shift Combinations

PAL(58–61)	PAL(62,63)	Total Shift Desired	Incremental Shifting Sequence		
0000	00	None	None		
	01	Right 1	Left 1 and right 4	Left 2	
	10	Right 2	Left 2 and right 4		
	11	Right 3	Left 1 and right 4		
Not 0000	00	Right 4, 8, 12 and so on	Right 4	Right 4 until E(12–15) = 1	
	01	Right 5, 9, 13 and so on	Left 1 and right 4	Left 2 and right 4	Right 4 until E(12–15) = 1
	10	Right 6, 10, 14 and so on	Left 2 and right 4	Right 4	Right 4 until E(12–15) = 1
	11	Right 7, 11, 15 and so on	Left 1 and right 4	Right 4	Right 4 until E(12–15) = 1

Shift Right Double, SRDA (8E)

- Shift 1st operand (in GPR per R1 and R1 + 1) right number of bit positions specified by low-order six bits of 2nd operand address and place result into 1st operand location.
- RS format:



Fetch high-order word of 1st operand from GPR per R1. Fetch low-order word of 1st operand from GPR per R1 + 1.

Shift 64-bit of 1st operand right number of bit positions specified by low-order 6 bits of 2nd operand address.

Store high-order word of result into GPR per R1 and low-order word into GPR per R1 + 1, and set CC.

- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is in S and T.
  Amount of shift is in D and PAL.
- D(18–23) indicates total amount of shift.
- E(12–15) indicates number of shifts to be performed in multiples of 4.
- E(12–15) is reduced by 1 after each shift of 4.
- Low-order bits of high-order word of doubleword operand are saved and placed into low-order word of operand.
- Value of operand sign is supplied to vacated bit positions.
- CC settings:
  Result in PAL equals zero: CC = 0.
  Result in PAL is less than zero: CC = 1.
  Result in PAL is greater than zero: CC = 2.

The Shift Right Double, SRDA, instruction right-shifts the doubleword first operand the number of bit positions specified by the low-order six bits of the second operand address. (The remainder of the address is ignored.) The R1 field of the instruction addresses the high-order 32 bits of the doubleword operand. The low-order word of the operand is in the GPR per R1 + 1. The R1 field must specify an even GPR; a program specification interruption occurs when R1 is odd.

The sign of the doubleword operand is the sign of the even GPR (R1). The sign bit (high-order bit) of the odd GPR (R1 + 1) is treated as an integer bit. Bits equal to the sign of the doubleword operand are supplied to the vacated high-order positions as the information is shifted. Bits shifted out of the low-order positions are lost.

Diagram 5-119, FEMDM, is a flowchart of the SRDA instruction. At the start of execution, the first 16 bits of the instruction are in E, the first operand is in S and T, and the number of bit positions to be shifted is in D and PAL.

The SRDA instruction execution is similar to that of the SRA instruction. The differences are a result of shifting a doubleword, one word at a time, in the SRDA instruction. Because the high-order word is shifted first, it is necessary to retain the underflow bits from the high-order word, to shift them right 4, and to append them to the low-order word.

Any specified amount of right shift involves at least one shift of right 4 for both the high-order word and the low-order word. Therefore, the underflow bits from the high-order word are appended to the low-order word [by transferring B(64–67) to PAB(28–31)] before the right 4 shift. This action results in the underflow bits and the low-order word being shifted right 4 as a unit.

Each time the shifted high-order word is stored, a zero test is performed and STAT A is set if the result is 0. When the low-order word has been shifted the same amount, STAT A is reset if the result is not 0. STAT A and the result sign [A(0)] determine the CC.

This section discusses the 44 instructions making up the floating-point instruction set. Before analyzing the instructions, however, the following paragraphs discuss exponent overflow and underflow and zero results, and list the conditions at the start of execution. (For a discussion of number representation, data formats, normalization, operand addressing, instruction formats, data flow, program interruptions, and condition codes, see Chapter 1.)

EXPONENT OVERFLOW AND UNDERFLOW

- Exponent overflow occurs if two positive characteristics are added, or if positive number is added to positive characteristic, and final result is negative characteristic.
- Exponent underflow occurs if two negative characteristics are added, or if quantity is subtracted (complement added) from negative characteristic, and final result is positive characteristic.
- Exponent underflow program interruption occurs if PSW(38) = 1.

During floating-point operations, values may be chosen that cause the CPU to yield invalid results. For example, the largest positive exponent that can be expressed as a floating-point characteristic is +63, and is represented in excess-64 notation as 111 1111 (7F, hex). Assume that +63 is the characteristic of a floating-point operand and that a 1 is added to it:

```
0  1  2  3  4  5  6  7
S  1  1  1  1  1  1  1   +63 (Excess-64 Notation)
S  0  0  0  0  0  0  1   + 1
S  0  0  0  0  0  0  0
 ↙
Carry
```

Note that the sum in bits 1–7, instead of indicating an exponent of +64, indicates an exponent of -64, 128 less than the true exponent. Thus, *exponent overflow* occurred. The rule for exponent overflow is: if two positive characteristics are added, or if a positive number is added to a positive characteristic, and the final result is a negative characteristic, exponent overflow occurred. This rule does not hold for intermediate result characteristics which may exceed the highest expressible exponent.

If exponent overflow occurs, an interruption is forced and cannot be masked off (refer to Chapter 1). The resulting invalid characteristic is not altered and remains in the result register for examination by the interruption-handling microprogram.

The largest negative exponent that can be expressed as a floating-point characteristic is -64, and is represented in excess-64 notation as a characteristic of all zeros. Assume that -64 is the characteristic of a floating-point operand and that a 1 is subtracted from it:

```
0  1  2  3  4  5  6  7
S  0  0  0  0  0  0  0   –64 (Excess-64 Notation)
S  1  1  1  1  1  1  1   2's Complement of 1
S  1  1  1  1  1  1  1
```

The difference in bits 1–7, instead of indicating an exponent of -65, indicates an exponent of +63, 128 more than the true exponent. This is known as *exponent underflow.* The rule for exponent underflow is: if two negative characteristics are added, or if a quantity is subtracted (complement-added) from a negative characteristic, and the final result is a positive characteristic, exponent underflow occurred. This rule does not hold for intermediate characteristics which may exceed the most negative expressible exponent.

If exponent underflow occurs, an interruption takes place only if the exponent-underflow mask bit [PSW(38)] is a 1 (refer to Chapter 1). If the interruption is taken, the resulting invalid characteristic is not altered and remains in the result register for examination by the interruption-handling microprogram. However, if the underflow interruption is masked off, the entire result (sign, characteristic, and fraction) is converted to a true zero (see "Zero Results"). This value can be a valid result for some calculations because exponent underflow indicates that the result was very small (less than 16^{-64}) and therefore close to zero.

Referring to the two examples given above, the CPU determines if exponent overflow or underflow occurred by examining bit 1 of the final characteristic and testing for a carry out of bit 1. If a carry occurred during a characteristic addition and bit 1 is a 0, exponent overflow occurred; if a carry did *not* occur during a characteristic subtraction and bit 1 is a 1, exponent underflow occurred.

ZERO RESULTS

A zero result is normally stored into the result register as a true zero; that is, a zero characteristic, a zero fraction, and a plus sign. A true zero may occur because of the magnitude of the operands or it may be forced. A true zero is forced if exponent underflow occurs during add, subtract, multiply, or divide instructions and the exponent-underflow mask is off [PSW(38) = 0]. A true zero is also forced when a result fraction is zero and the program interruption for significance is masked off during add or subtract instructions. "Significance" means that an add or subtract instruction resulted in a zero fraction. This condition causes a significance program interruption if the significance mask is on [PSW(39) = 1]. When a significance condition occurs with the mask on, the result characteristic and sign remain unchanged and are stored with the zero fraction. True zero is *never* forced when a zero fraction occurs during a load, store, or halve instruction. Whenever a result has a zero fraction, an exponent overflow or exponent underflow condition is ignored.

CONDITIONS AT START OF EXECUTION

The conditions at the start of execution for the RR and RX instructions, short and long operands, are:

1. RR, Short Operands:
   a. First operand is in A, B, and D (24-bit fraction only).
   b. Second operand is in S and T.
   c. Instruction is in E.
2. RR, Long Operands:
   a. 32 bits of first operand are in A, B, and D (24-bit fraction only).
   b. 32 bits of second operand are in S and T.
   c. Low-order fractions of first and second operands are in LS.
   d. Instruction is in E.
3. RX, Short Operands:
   a. First operand is in S and T.
   b. Main storage request for second operand has been issued per D.
   c. First 16 bits of instruction are in E.
4. RX, Long Operands:
   a. 32 bits of first operand are in S and T.
   b. Low-order fraction of first operand is in LS.
   c. Main storage request for 2nd operand has been issued per D.
   d. First 16 bits of instruction are in E.

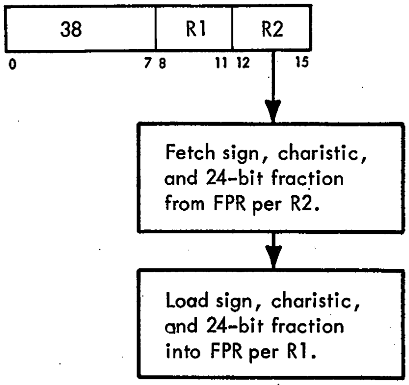
LOAD

The floating-point load instructions provide a means of loading operands into the LS FPR's. The load operation may be register-to-register (RR format) or storage-to-register (RX format) and may use short or long operands. In any case, the instruction loads the second operand into the first operand location, and the second operand location remains unchanged.

In addition, certain floating-point load instructions can test or modify the sign of the second operand before loading it into LS. The second operand may be also complemented before loading.

Load, LER (38) – RR Short Operands

- Load 2nd operand (in FPR, per R2) into 1st operand location (in FPR, per R1).
- RR format:

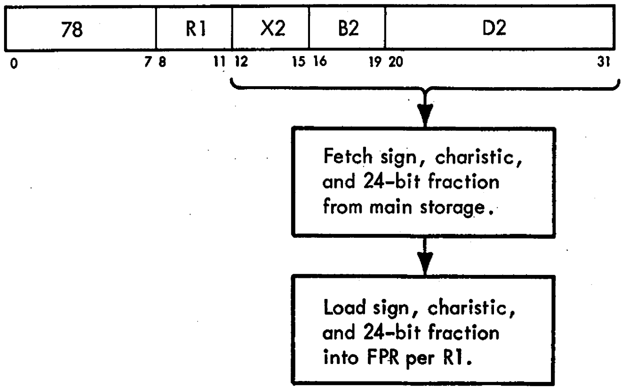

38
R1
R2
0
7 8
11 12
15
Fetch sign, charistic, and 24-bit fraction from FPR per R2.
Load sign, charistic, and 24-bit fraction into FPR per R1.

- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.

The Load, LER, instruction (Diagram 5-202, FEMDM) loads the 32-bit second operand from the LS FPR specified by the R2 field into the first operand location specified by the R1 field. During the RR I-Fetch, the second operand is gated to S and T. At the beginning of the execution phase, a specification test is initiated. If no specification check occurred, the contents of T (second operand) are gated to the LS FPR specified by R1. If a specification check did occur, the operation is suppressed and a specification program interruption occurs.

Load, LE (78) – RX Short Operands

- Load 2nd operand (in storage) into 1st operand location (in FPR, per R1).
- RX format:


78
R1
X2
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch sign, charistic, and 24-bit fraction from main storage.
Load sign, charistic, and 24-bit fraction into FPR per R1.

- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.

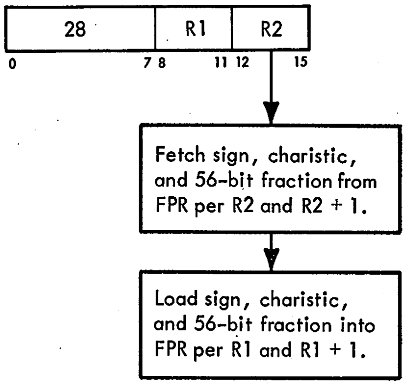
The Load, LE, instruction (Diagram 5-203, FEMDM) loads the 32-bit second operand from the main storage location specified by the effective address into the first operand location specified by the R1 field. The effective address of the second operand must be on a word boundary or a specification program interruption occurs. During the RX I-Fetch, the effective address is computed and placed into D. A main storage request for the second operand is then initiated per D.

At the beginning of the execution phase, a specification test is initiated. If no specification check occurred, either SDBO(0–31) or SDBO(32–63) is gated to T. Because a main storage request is always for a doubleword and only a word operand is desired, D(21) determines which word of the main storage doubleword is used; if D(21) = 1, the right word is gated to T; if D(21) = 0, the left word. From T, the second operand is gated to the LS FPR specified by R1.

If a specification check occured at the start of the execution phase, the operation is suppressed and a specification program interruption occurs.

Load, LDR (28) – RR Long Operands

- Load 2nd operand (in FPR, per R2 and R2 + 1) into 1st operand location (in FPR, per R1 and R1 + 1).
- RR format:


28
R1
R2
0
7 8
11 12
15
Fetch sign, charistic, and 56-bit fraction from FPR per R2 and R2 + 1.
Load sign, charistic, and 56-bit fraction into FPR per R1 and R1 + 1.

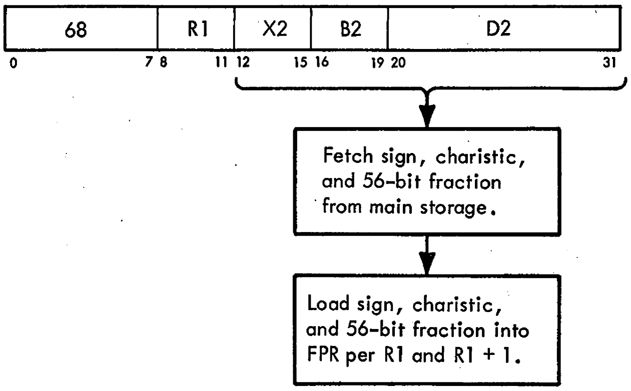
- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Low-order fractions of 1st and 2nd operands are in LS.
  Instruction is in E.

The load, LDR, instruction (Diagram 5-202, FEMDM) loads the doubleword second operand from the LS FPR specified by R2 and R2 + 1 into the first operand location specified by R1 and R1 + 1. During the RR I-Fetch, the high-order 32 bits of the second operand are placed into S and T. At the beginning of the execution phase, a specification test is initiated. If no specification check occurred, the low-order 32 bits of the second operand are fetched from the odd register of the even/odd pair of FPR's specified by the R2 field. From T, the low-order 32 bits of the second operand are loaded into the odd register of the even/odd pair of FPR's specified by the R1 field. The high-order 32 bits of the second operand are then gated from S to T. From T, they are loaded into the even register of the even/odd pair of FPR's specified by the R1 field.

If a specification check occurred at the start of the execution phase, the operation is suppressed and a specification program interruption occurs.

Load, LD (68) – RX Long Operands

- Load 2nd operand (in storage) into 1st operand location (in FPR, per R1 and R1 + 1).
- RX format:


68
R1
X2
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch sign, charistic, and 56-bit fraction from main storage.
Load sign, charistic, and 56-bit fraction into FPR per R1 and R1 + 1.

- Conditions at start of execution:
  32 bits of 1st operand are in S and T.
  Low order fraction of 1st operand is in LS.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.

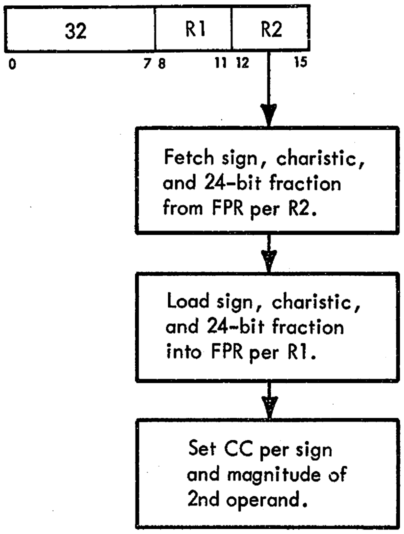
The Load, LD, instruction (Diagram 5-203, FEMDM) loads the doubleword second operand from the main storage location specified by the effective address into the first operand location specified by R1 and R1 + 1. The effective address of the second operand must be on a doubleword boundary or a specification program interruption occurs. During the RX I-Fetch, the effective address is computed and placed into D. A main storage request for the second operand is then initiated per D.

At the beginning of the execution phase, a specification test is initiated. If no specification check occurred, the doubleword second operand is gated from SDBO to ST. The low-order 32 bits are then loaded into the odd register of an even/odd pair of FPR's specified by the R1 field. The high-order 32-bits of the second operand are then gated from S to T. From T, they are loaded into the even register of the even/odd pair of FPR's specified by the R1 field.

If a specification check occurred at the start of the execution phase, the operation is suppressed, and a specification program interruption occurs.

Load and Test, LTER (32) – RR Short Operands

- Load 2nd operand (in FPR, per R2) into 1st operand location (in FPR, per R1).
- RR format:


32
R1
R2
0
7 8
11 12
15
Fetch sign, charistic, and 24-bit fraction from FPR per R2.
Load sign, charistic, and 24-bit fraction into FPR per R1.
Set CC per sign and magnitude of 2nd operand.

- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.
- CC setting:
  2nd operand fraction equals zero: CC = 0.
  2nd operand is less than zero: CC = 1.
  2nd operand is greater than zero: CC = 2.

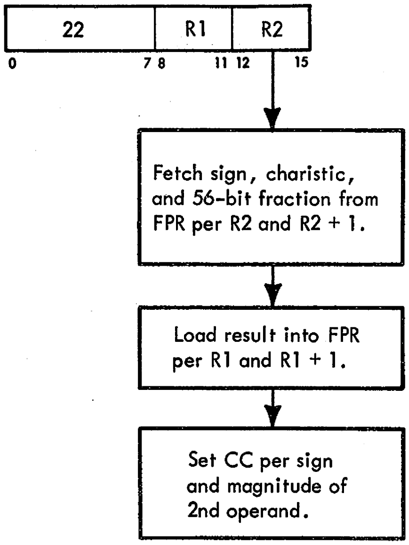
The Load and Test, LTER, instruction (Diagram 5-204, FEMDM) loads the 32-bit second operand from the FPR specified by the R2 field into the first operand location specified by the R1 field. The sign and magnitude of the second operand determine the CC, as follows:

1. If the fraction of the second operand equals zero, the CC is set to 0.
2. If the second operand (sign, characteristic, and fraction) is less than zero, the CC is set to 1.
3. If the second operand (sign, characteristic, and fraction) is greater than zero, the CC is set to 2.

The LTER execution is similar to LER execution. However, the sign of the second operand is saved in STAT C, and STAT A is set if the fraction equals zero. The CC is determined during the normal end-op cycle. If STAT A is set, indicating that the fraction equals zero, the CC is set to 0. The sign and characteristic are not considered when the fraction equals zero. If the second operand fraction is not equal to zero, the sign (STAT C) determines a greater-than-zero or less-than-zero condition. If the sign is minus (STAT C set), the second operand is less than zero and the CC is set to 1. If the sign is plus (STAT C reset), the second operand is greater than zero and the CC is set to 2. Setting the CC depends upon the 'Set-CR' micro-order, the instruction op-code, and the hardware conditions specified above.

Load and Test, LTDR (22) – RR Long Operands

- Load 2nd operand (in FPR, per R2 and R2 + 1) into 1st operand location (in FPR, per R1 and R1 + 1).
- RR format:


22
R1
R2
0
7 8
11 12
15
Fetch sign, charistic, and 56-bit fraction from FPR per R2 and R2 + 1.
Load result into FPR per R1 and R1 + 1.
Set CC per sign and magnitude of 2nd operand.

- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Low-order fractions of 1st and 2nd operands are in LS.
  Instruction is in E.
- CC setting:
  2nd operand fraction equals zero: CC = 0.
  2nd operand is less than zero: CC = 1.
  2nd operand is greater than zero: CC = 2.

The Load and Test, LTDR, instruction (Diagram 5-205, FEMDM) loads the doubleword second operand from the FPR specified by R2 and R2 + 1 into the first operand location specified by R1 and R1 + 1. The sign and magnitude of the second operand determine the CC, as follows:

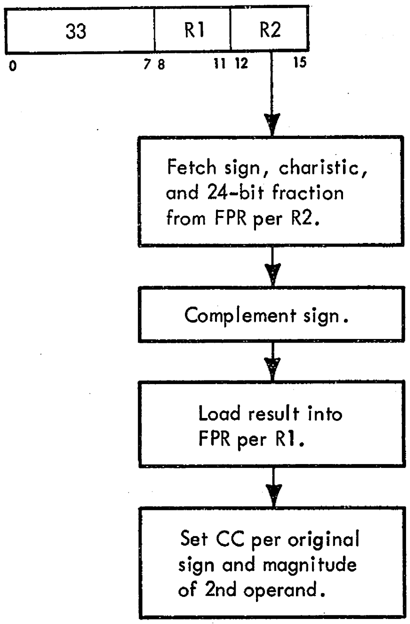
1. If the fraction of the second operand equals zero, the CC is set to 0.

2. If the second operand (sign, characteristic, and fraction) is less than zero, the CC is set to 1.
3. If the second operand (sign, characteristic, and fraction) is greater than zero, the CC is set to 2.

The LTDR execution is similar to LDR execution. However, the sign of the second operand is saved in STAT C, and STAT A is set if the fraction equals zero. The CC is determined during the normal end-op cycle in the same manner as explained for the LTER instruction.

Load Complement, LCER (33) – RR Short Operands

- Load 2nd operand (in FPR, per R2) into 1st operand location (in FPR, per R1) with sign complemented.
- RR format:


33
R1
R2
0
7 8
11 12
15
Fetch sign, charistic, and 24-bit fraction from FPR per R2.
Complement sign.
Load result into FPR per R1.
Set CC per original sign and magnitude of 2nd operand.

- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.
- CC setting:
  2nd operand fraction equals zero: CC = 0.
  Original sign is plus: CC = 1.
  Original sign is minus: CC = 2.

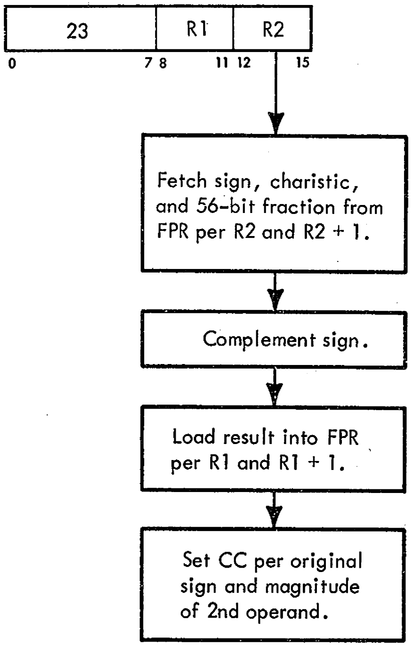
The Load Complement, LCER, instruction (Diagram 5-204) loads the 32-bit second operand from the FPR specified by the R2 field into the first operand location specified by the R1 field. During the loading, the sign is changed to the opposite value (complemented). The original sign and magnitude of the second operand determine the CC, as follows:

1. If the fraction of the second operand equals zero, the CC is set to 0.
2. If the original sign of the second operand is plus, the CC is set to 1.
3. If the original sign of the second operand is minus, the CC is set to 2.

Except for complementing the sign of the second operand, LCER execution is identical to LTER execution.

Load Complement, LCDR (23) – RR Long Operands

- Load 2nd operand (in FPR, per R2 and R2 + 1) into 1st operand location (in FPR, per R1 and R1 + 1) with sign complemented.
- RR format:


23
R1
R2
0
7 8
11 12
15
Fetch sign, charistic, and 56-bit fraction from FPR per R2 and R2 + 1.
Complement sign.
Load result into FPR per R1 and R1 + 1.
Set CC per original sign and magnitude of 2nd operand.

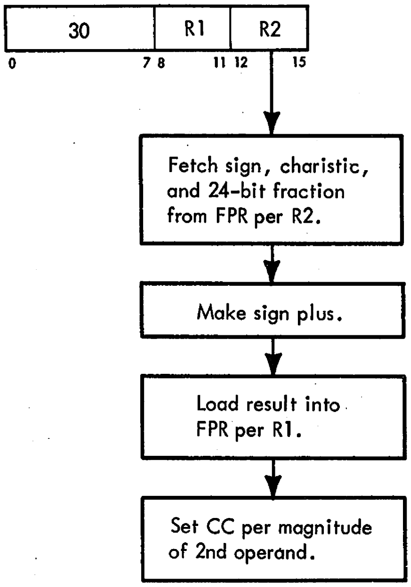
- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Low-order fractions of 1st and 2nd operands are in LS.
  Instruction is in E.
- CC setting:
  2nd operand fraction equals zero: CC = 0.
  Original sign is plus: CC = 1.
  Original sign is minus: CC = 2.

The Load Complement, LCDR, instruction (Diagram 5-205) loads the doubleword second operand from the FPR specified by R2 and R2 + 1 into the first operand location specified by R1 and R1 + 1. During the loading, the sign is complemented. The original sign and magnitude of the second operand determine the CC in the same manner as explained for the LCER instruction.

Except for complementing the sign of the second operand, LCDR execution is identical to LTDR execution.

Load Positive, LPER (30) – RR Short Operands

- Load 2nd operand (in FPR, per R2) into 1st operand location (in FPR, per R1) with sign made plus.
- RR format:


30
R1
R2
0
7 8
11 12
15
Fetch sign, charistic, and 24-bit fraction from FPR per R2.
Make sign plus.
Load result into FPR per R1.
Set CC per magnitude of 2nd operand.

- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.
- CC setting:
  2nd operand fraction equals zero: CC = 0.
  2nd operand is greater than zero: CC = 2.

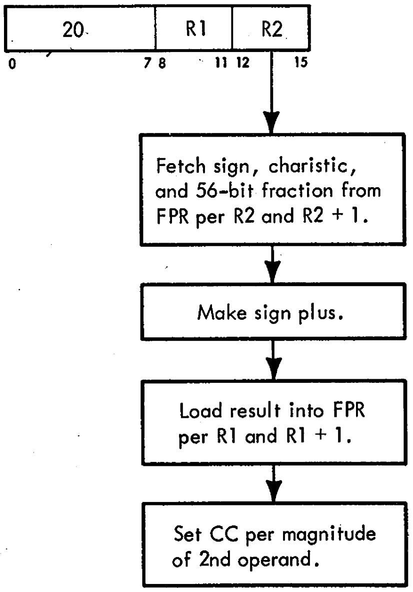
The Load Positive, LPER, instruction (Diagram 5-204) loads the 32-bit second operand from the FPR specified by the R2 field into the first operand location specified by the R1 field. During the loading, the sign is made plus. Thus, the result stored is always zero or greater. STAT A is set if the second operand fraction equals zero. The magnitude of the second operand determines the CC, as follows:

1. If the second operand fraction equals zero, the CC is set to 0.
2. If the second operand is greater than zero, the CC is set to 2.

Except for making the sign of the second operand plus, LPER execution is identical to LTER execution.

Load Positive, LPDR (20) – RR Long Operands

- Load 2nd operand (in FPR, per R2 and R2 + 1) into 1st operand location (in FPR, per R1 and R1 + 1) with sign made plus.
- RR format:


20
R1
R2
0
7 8
11 12
15
Fetch sign, charistic, and 56-bit fraction from FPR per R2 and R2 + 1.
Make sign plus.
Load result into FPR per R1 and R1 + 1.
Set CC per magnitude of 2nd operand.

- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Low-order fractions of 1st and 2nd operands are in LS.
  Instruction is in E.
- CC setting:
  2nd operand fraction equals zero: CC = 0.
  2nd operand is greater than zero: CC = 2.

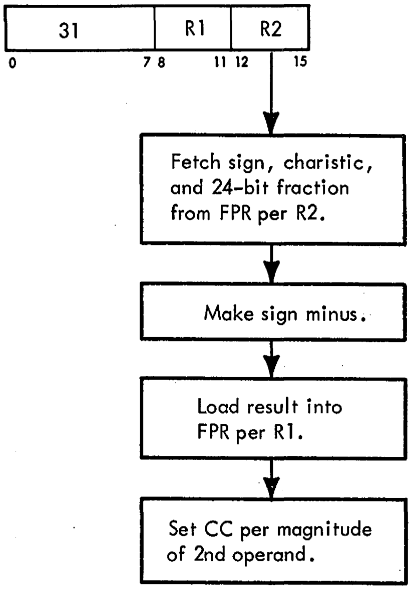
The Load Positive, LPDR, instruction (Diagram 5-205) loads the doubleword second operand from the FPR specified by R2 and R2 + 1 into the first operand location specified by R1 and R1 + 1. During the loading, the sign is made plus. Thus, the result stored is always zero or greater. STAT A is set if the second operand fraction equals zero. The magnitude of the second operand determines the CC in the same manner as explained for the LPER instruction.

Except for making the sign of the second operand plus, LPDR execution is identical to LTDR execution.

Load Negative, LNER (31) – RR Short Operands

- Load 2nd operand (in FPR, per R2) into 1st operand location (in FPR, per R1) with sign made minus.

- RR format:


31
R1
R2
0
7 8
11 12
15
Fetch sign, charistic, and 24-bit fraction from FPR per R2.
Make sign minus.
Load result into FPR per R1.
Set CC per magnitude of 2nd operand.

- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.

- CC setting:
  2nd operand fraction equals zero: CC = 0.
  2nd operand is less than zero: CC = 1.

The Load Negative, LNER, instruction (Diagram 5-204) loads the 32-bit second operand from the FPR specified by the R2 field into the first operand location specified by the R1 field. During the loading, the sign is made minus. Thus, the result stored is always zero or less. STAT A is set if the second operand fraction equals zero. The magnitude of the second operand determines the CC, as follows:

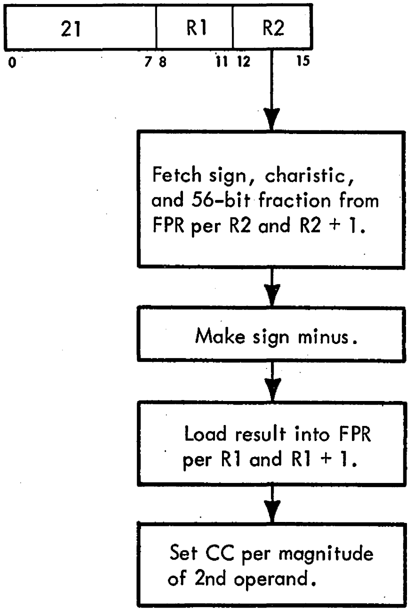
1. If the second operand fraction equals zero, the CC is set to 0.
2. If the second operand is less than zero, the CC is set to 1.

Except for making the sign of the second operand minus, LNER execution is identical to LTER execution.

Load Negative, LNDR (21) – RR Long Operands

- Load 2nd operand (in FPR, per R2 and R2 + 1) into 1st operand location (in FPR, per R1 and R1 + 1) with sign made minus.

- RR format:


21
R1
R2
0
7 8
11 12
15
Fetch sign, charistic, and 56-bit fraction from FPR per R2 and R2 + 1.
Make sign minus.
Load result into FPR per R1 and R1 + 1.
Set CC per magnitude of 2nd operand.

- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Low-order fractions of 1st and 2nd operands are in LS.
  Instruction is in E.

- CC setting:
  2nd operand fraction equals zero: CC = 0.
  2nd operand is less than zero: CC = 1.

The Load Negative, LNDR, instruction (Diagram 5-205) loads the doubleword second operand from the FPR specified by R2 and R2 + 1 into the first operand location specified by R1 and R1 + 1. During the loading, the sign is made minus. Thus, the result stored is always zero or less. STAT A is set if the second operand fraction equals zero. The magnitude of the second operand determines the CC in the same manner as explained for the LNER instruction.

Except for making the sign of the second operand minus, LNDR execution is identical to LTDR execution.

ADD, SUBTRACT, AND COMPARE

- 20 add, subtract, and compare instructions.
- Short and long operands are available in both formats.
- Add and subtract instruction results may be normalized or unnormalized.
- Results are in true form with plus or minus values.

The 20 floating-point add-type instructions are divided into three major groups: add, subtract, and compare. RR and RX formats using short and long operands are available in each group. The results of add and subtract instruction may be normalized or unnormalized, depending upon the instruction being executed. A CC is set on all add-type instructions; the compare instructions cause a CC to be set with no result stored.

The CPU computes the sum of floating-point numbers as follows (assume that $0.004_{16} \times 16^5$ is to be added to $0.502_{16} \times 16^4$.):

1. Equalizes the characteristics.
   a. If the characteristics are unequal, the operand with the smallest characteristic is shifted right the number of hex digits necessary to equalize the characteristics. In the example, $0.502_{16} \times 16^4$ has the smallest characteristic (exponent); it is therefore changed to $0.0502_{16} \times 16^5$, thus making the characteristics equal.
   b. If the number of shifts exceeds the number of hex digits available, the operand with the largest characteristic becomes the intermediate result.
2. When the characteristics are equal, algebraically adds the first and second operand fractions.
   a. If the signs are alike, adds the first operand fraction to the second operand fraction. In the example, the signs are alike; therefore, the fractions are added giving a sum of $0.0542_{16} \times 16^5$.
   b. If the signs are unlike, subtracts the second operand fraction from the first operand fraction (adds the 2's complement of the second operand fraction to the first operand fraction).
3. If the intermediate result fraction is in complement form, recomplements it (takes 2's complement) to obtain the true fraction value.
4. Normalizes the intermediate result, if normalization is called for. Assume a 3-digit machine. If normalization is specified, the final result of the example (2a above) becomes $0.542_{16} \times 16^4$. If normalization is *not* specified, the low-order digit (guard digit) is truncated, and the final result is $0.054_{16} \times 16^5$.
5. Determines the sign and characteristic value.
6. Stores the sign, characteristic, and fraction into LS as specified by the R1 field.
7. Sets the CC per hardware conditions.

For subtraction of floating-point numbers, the algebraic rule applies: to subtract two numbers, change the sign of the subtrahend and proceed as in addition. When subtracting floating-point numbers, the sign of the second operand is complemented. The rules of addition apply as outlined in the previous paragraph. To illustrate, assume the same numbers as used above and that the signs are unlike. Thus, a 2's complement add (step 2b above) is performed:

$$\begin{array}{ll} .0502_{16} & \\ \underline{.FFC0_{16}} & \text{(2's complement of } .0040_{16}\text{)} \\ .04C2_{16} & \end{array}$$

The difference, then, is $0.04C_{16} \times 16^5$ if unnormalized; or $0.4C2_{16} \times 16^4$, if normalized.

The compare instructions are similar to the subtract instructions; the results, however, are not stored. The compare instructions algebraically compare the first operand with the second operand and set the CC accordingly. These objectives are accomplished by complementing the sign, algebraically adding the fractions, determining a high, low, or equal condition, and setting the CC.

The basic objectives of the add-type instructions are shown in Sheet 1 of Diagrams 5-206 and 5-207, FEMDM (short and long operands, respectively). After the RR or RX I-Fetch and the specification test, the remaining operand and/or low-order fraction(s) must be fetched or the low-order fractions reset to zeros. The signs are saved in STAT's. For short operand instructions (Diagram 5-206), zeros are gated to the low-order fractions of the 64-bit operands.

The characteristics are then compared. Preshifting occurs, if necessary, followed by the addition or subtraction of the fractions. Because the characteristics must be equal before algebraically adding the operands, the characteristics are subtracted to determine whether they are equal and whether preshifting is meaningful. For short operands, the characteristic difference must be 7 or less; for long operands, 15 or less.

If the characteristic difference is greater than 7 (short operands) or 15 (long operands) the fraction resulting after right-shifting equals zero. Therefore, preshifting is not performed, and the operand with the largest characteristic becomes the result. If preshifting is meaningless, the value in AB or ST is the result. If the characteristics are within range, the smallest fraction is right-shifted until the characteristics are equal; the fractions are then added algebraically to form an intermediate result. If a high-order carry occurs as a result of the addition (overflow), the intermediate result is right-shifted one hex digit and the characteristic is increased by 1. If this increase causes a characteristic overflow, an exponent-overflow program interruption occurs.

The intermediate result consists of 7 or 15 hex digits and a possible carry. The low-order digit is a guard digit retained from the fraction which is shifted right. Only one guard digit participates in the fraction addition. The guard digit is zero if no shift occurs.

After the addition or subtraction, a test is made for compare instructions, normalized instructions, or unnormalized instructions. Postnormalization, recomplementation, and/or fraction overflow correction is accomplished during this phase. The final result is stored (except on compare instructions), and the CC is set according to the computed results. An end op completes instruction execution.

For normalized instructions the intermediate result fraction is left-shifted as necessary to form a normalized fraction. Vacated low-order digit positions are filled with zeros, and the characteristic is reduced by the amount of the shift. If normalization causes the characteristic to underflow, an exponent-underflow program interruption condition exists; the sign, characteristic, and fraction are made zero if the underflow mask bit [PSW(38)] is a 0. If PSW(38) is a 1, a program interruption occurs, and the characteristic is made 128 larger than the true result; the sign and fraction remain unchanged. If no left shift takes place, the guard digit is removed to obtain the proper fraction length.

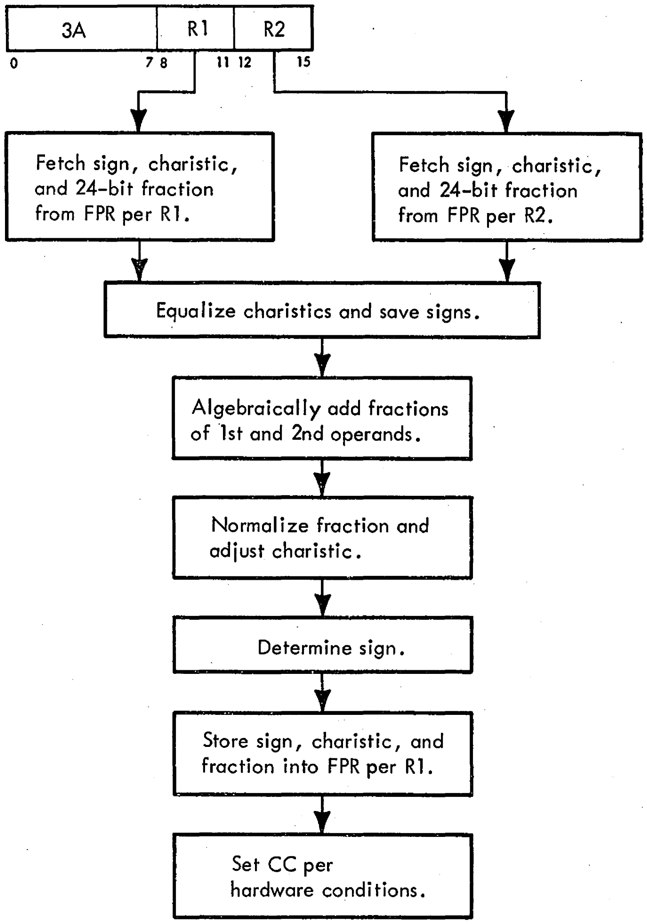
When the intermediate result fraction is zero and the significance mask bit is a 1, a significance program interruption takes place. No normalization occurs, and the intermediate result characteristic remains unchanged. When the intermediate result is zero and the significance mask bit is a 0, a significance program interruption does not occur; rather, the characteristic and the sign are made zero, yielding a true zero result. Exponent underflow does not occur for a zero fraction.

The sign of the result is derived algebraically. However, the sign of a zero result fraction is always positive.

Add Normalized, AER (3A) — RR Short Operands

- Algebraically add 2nd operand (in FPR, per R2) to 1st operand (in FPR, per R1) and place normalized sum into 1st operand location.
- RR format: (See adjoining column.)
- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.

The Add Normalized, AER, instruction (Diagram 5-206) algebraically adds the second operand (specified by R2) to the first operand (specified by R1), and places the normalized sum into the first operand location. The CC is set according to hardware conditions. The AER instruction uses 32-bit operands; therefore, the contents of the low-order halves of the FPR's in LS remain unchanged.


3A
R1
R2
0
7 8
11 12
15
Fetch sign, charistic, and 24-bit fraction from FPR per R1.
Fetch sign, charistic, and 24-bit fraction from FPR per R2.
Equalize charistics and save signs.
Algebraically add fractions of 1st and 2nd operands.
Normalize fraction and adjust charistic.
Determine sign.
Store sign, charistic, and fraction into FPR per R1.
Set CC per hardware conditions.

At the beginning of the execution phase:

1. The first operand is in A, B, and D (24-bit fraction only).
2. The second operand is in S and T.
3. The STC was set to 4 during I-Fetch.
4. The AER instruction is in E.

A specification test is made at the beginning of the execution phase. If a specification check exists, instruction execution is suppressed, and a specification program interruption occurs. Assume that no specification check exists.

Because the AER instruction uses short operands (32 bits) in the RR format, no operand fetch during instruction execution is necessary. The sign of the first operand is saved in STAT F and the sign of the second operand is saved in STAT C. The first operand characteristic is subtracted from the second operand characteristic to determine the characteristic difference. Next, the second operand is gated to D and S. Because the AER instruction operates on short operands, B and T are reset, and the operand is treated as a 56-bit fraction with the low-order bits set to zero. The characteristic difference and the signs determine the next operation to be performed by means of a 10-way 'FLR' micro-order branch.

The 10-way 'FLR' branch on characteristic difference and signs occurs for all add-type instructions. When this branch is encountered, the conditions are as follows:

1. The first operand is in AB (B equals zero for short operands).
2. The second operand is in DT (T equals zero for short operands) and S.
3. The sign of the first operand is in STAT F.
4. The sign of the second operand is in STAT C.
5. The characteristic difference is in SAL and F.

To subtract the first operand characteristic from the second operand characteristic, the 2's complement of the first operand characteristic is added to the second operand characteristic. Because the serial adder consists of 8 binary bit positions, a 1 is forced into bit position 0 on the A-bus (first operand side) of the serial adder, and a 0 is forced into bit position 0 on the B-bus (second operand side) of the serial adder. The characteristic difference is then routed to SAL and F.

The 10-way 'FLR' branch is determined by the result of the characteristic subtraction and by the signs in STAT F and STAT C. The 10-way 'FLR' branch affects bits 8–11 of ROSAR as defined in Sheet 2 of Diagram 5-206. ROSAR(8–11) is set as follows:

1. ROSAR(8) = 1 when a serial adder carry indicates that the second operand is greater than or equal to the first operand.
2. ROSAR(9) = 1 when the signs are alike. If the signs are alike, the fractions are added; if unlike, the fractions are subtracted.
3. ROSAR(10) = 1 when the characteristic difference is within range. ROSAR(10) equaling 1 implies that the characteristic difference is small enough so that equalizing the characteristics is meaningful. (A zero fraction may occur as a result of characteristic equalization.)
4. ROSAR(11) = 1 when the result in SAL is zero. This condition indicates that the characteristics are equal. A serial adder carry also occurs; thus ROSAR(8) will also equal 1.

Assume that the following two characteristics are to be compared to determine the ROSAR value:

2nd operand characteristic: $64_{10} = 1000000_2$
1st operand characteristic: $73_{10} = 1001001_2$

The first operand characteristic is subtracted from the second operand characteristic shown below:

```
2nd operand characteristic:                 0 1000000
Complement of 1st operand characteristic:   1 0110110 ⎫ 2's com-
Add 1:                                      0 0000001 ⎭ plement
                                            ---------
Characteristic difference in SAL(0–7):      1 1110111
                                            ↑ ↑↑↑↑
                          SAL(0) ───────────┘ │││ │
                          SAL(1–3) ───────────┴┴┘ │
                          SAL(4) ─────────────────┘
```

Because no SAL(0) carry occurred, ROSAR(8) = 0. ROSAR(9) depends upon the signs assigned to the fractions of the two operands. The table in Sheet 2 of Diagram 5-206 shows the bit positions tested in SAL to determine the value of ROSAR(10) and whether preshifting is meaningful. In this example, SAL(0–3) = 1111 (binary). Because a short operand instruction is being executed, ROSAR(10) = 0; therefore, the value is in AB because no carry from SAL(0) indicated that the first operand is greater than the second operand. ROSAR(11) = 0 because SAL(0–7) is not all 0's.

If all positions of SAL are 0, ROSAR(11) = 1, indicating that the characteristics of the two operands are equal. ROSAR(9) then determines the next operation (signs alike, add; signs unlike, subtract). In this case [SAL(0–7) = 0], ROSAR(8) also equals a 1 because a SAL(0) carry occurred.

For the next example, assume that the two characteristics in the previous example are interchanged. The characteristics are compared as follows:

2nd operand characteristic: 1001001
1st operand characteristic: 1000000

The first operand characteristic is subtracted from the second operand characteristic shown below:

```
2nd operand characteristic:                 0 1001001
Complement of 1st operand characteristic:   1 0111111 ⎫ 2's com-
Add 1:                                      0 0000001 ⎭ plement
                                            ---------
                     SAL(0) carry ◄──────── 0 0001001
                                            ↑ ↑↑↑↑
                          SAL(0) ───────────┘ │││ │
                          SAL(1–3) ───────────┴┴┘ │
                          SAL(4) ─────────────────┘
```

In this example, ROSAR(8) = 1 because a SAL(0) carry occurred. ROSAR(10) = 0. Because preshifting in this example is meaningless, the operand in ST is the result fraction because the SAL(0) carry indicates that the second operand is the largest.

The two examples just discussed illustrate the determination of the ROSAR(8–11) values. Additional examples are shown in Table 3-9. ROSAR(8–11) determines the ROS branch that performs the next steps in executing the AER instruction.

The examples of determining the ROSAR(8–11) values as shown in Table 3-9 indicate that the fraction of the operand with the smallest characteristic is shifted right when the characteristic difference is seven or less.

Four possible ROSAR(8- '1) values (0010, 0110, 1010, and 1110) cause characteristic equalization and then an algebraic addition or subtraction of fractions. For example, assume that the AER instruction requires characteristic equalization, fraction subtraction, recomplementation (second operand fraction is greater than first operand fraction), and normalization. Further, assume a ROSAR(8–11) value of 0010. The 0010 branch causes one right shift of the second operand to occur, and a 1 is

Table 3-9. Examples of Branching on Characteristic Difference

Description	Example No. 1	Example No. 2	Example No. 3	Example No. 4	Example No. 5
2nd operand characteristic	1000000	1000000	1001000	1000111	1000000
1st operand characteristic	1001001	1001000	1000000	1000000	1000000
2nd operand characteristic	0 1000000	0 1000000	0 1001000	0 1000111	0 1000000
Complement of 1st operand characteristic	1 0110110	1 0110111	1 0111111	1 0111111	1 0111111
Add 1	0 0000001	0 0000001	0 0000001	0 0000001	0 0000001
Difference in SAL(0–7)	1 1110111	1 1111000	← 0 0001000	← 0 0000111	← 0 0000000
			SAL(0) carry	SAL(0) carry	SAL(0) carry
Within Range?* Short	No	Yes	No	Yes	Yes
Long	Yes	Yes	Yes	Yes	Yes
ROSAR(8–11) value	Sub 0000 Add 0100	Sub 0010 Add 0110	Sub 1000 Add 1100	Sub 1010 Add 1110	Sub 1011 Add 1111
Comments	Result in AB.'	Equalize fraction in ST.	Result in ST.	Equalize fraction in AB.	Add or sub. No shift necessary.

Notes:

1. ROSAR(8) = 1 when there is a serial adder carry. A carry indicates that R2 ≧ R1.
2. ROSAR(9) = 1 when the signs are alike.

*3. ROSAR(10) = 1 when the characteristics are within range. ROSAR(10) = 1 on SAL results as follows:
   a. SAL carry and SAL(0–3) = 0 and long operands.
   b. SAL carry and SAL(0–4) = 0.
   c. No SAL carry and SAL(0–3) = 1's and long operands.
   d. No SAL carry and SAL(0–4) = 1's.

4. ROSAR(11) = 1 when the SAL outputs equal 0.

added to F. Note that one guard digit is retained. SAL(4–7) is checked for 1111 (binary). When SAL(4–7) = 1111, the characteristics are equal. Because the test for a branch is made one machine cycle before the ROS branch occurs, the SAL value is one machine cycle behind the actual shift count. For this reason, a test is made for 1111 in SAL(4–7) instead of for 0000.

Once the characteristics are equal, the second operand is subtracted from the first operand (signs unlike). To subtract fractions (signs unlike), the 2's complement of the second operand fraction in DT is added to the first operand fraction in AB with the intermediate fraction result placed into AB and DT. The intermediate fraction result may be in true form or in complement form, or it may be equal to zero. If a zero fraction results, STAT A is set.

Because the larger characteristic is used as the characteristic of the result, it is gated to F(1–7); F(0) is reset.

If the second operand fraction is greater than the first operand fraction, the result is in complement form. Conversely, the result is in true form if the first operand fraction is greater than the second operand fraction. If A(7) = 1, the intermediate result is in complement form. If A(7) = 0, the intermediate result is in true form. When the intermediate result is in complement form, the result must be recomplemented; that is, the 2's complement of the intermediate result is taken after the algebraic addition.

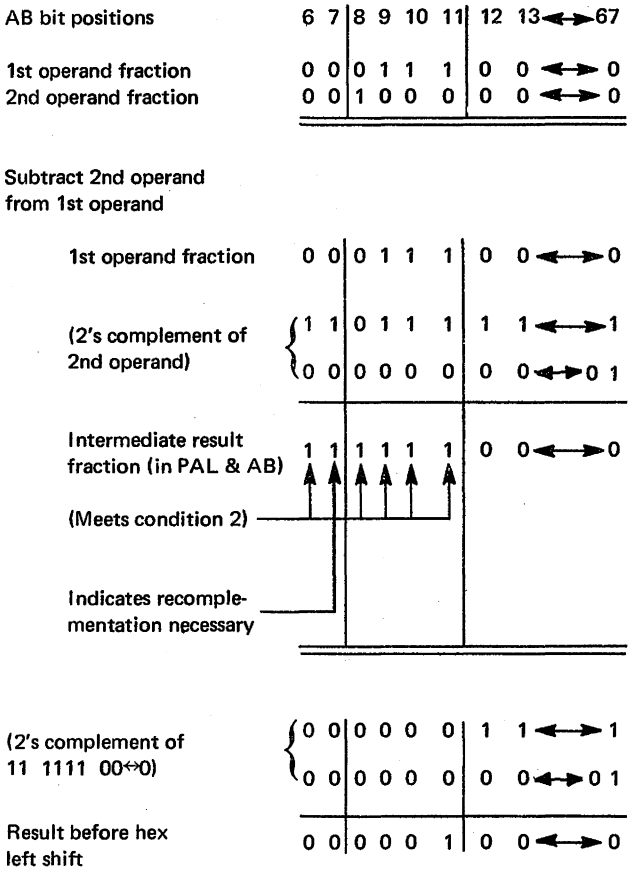
When the result is in true form, and if the fraction is not equal to zero, the fraction must be normalized and stored and the CC set. Because the AER instruction is a normalized instruction, the intermediate result is normalized, if necessary. After normalization, the sign and the characteristic are inserted and stored with the fraction into the first operand location (specified by R1). Assuming no errors or zero fraction, the CC is set. An end-op cycle completes instruction execution.

When in the normalizing loop after subtraction, the intermediate fraction result can be left-shifted out of the high-order hex digit position if the intermediate fraction is 0001. This left shift results in a zero fraction. The zero fraction, in this case, is not a true zero result or a significance condition; therefore, the true value must be restored.

Because the test for ROS branches is made one machine cycle before the ROS branch occurs, a test for normalization is made before the recomplementation is performed. Therefore, the test for normalization is determined by the following conditions:

1. PAL(7–11) is 0's and PAL(7–67) is not 0's.
2. PAL(6,8–11) is 1's and PAL(7–67) is not 0's.

If one of these two conditions is met, at least one normalization cycle is performed after the recomplementation machine cycle. This action is not always necessary, however. For example, if the following fractions are subtracted, the assumed normalization cycle is not necessary:



	6 7	8 9 10 11	12 13◄►67
AB bit positions	6 7	8 9 10 11	12 13◄►67
1st operand fraction	0 0	0 1 1 1	0 0◄►0
2nd operand fraction	0 0	1 0 0 0	0 0◄►0
Subtract 2nd operand from 1st operand			
1st operand fraction	0 0	0 1 1 1	0 0◄►0
(2's complement of 2nd operand)	1 1	0 1 1 1	1 1◄►1
	0 0	0 0 0 0	0 0◄►0 1
Intermediate result fraction (in PAL & AB)	1 1	1 1 1 1	0 0◄►0
(2's complement of 11 1111 00↔0)	0 0	0 0 0 0	1 1◄►1
	0 0	0 0 0 0	0 0◄►0 1
Result before hex left shift	0 0	0 0 0 1	0 0◄►0

The intermediate result fraction above shows that PAL(6,8–11) equals 1's and PAL(7–67) does not equal 0's. This condition causes a branch to the ROS normalization routine. Because A(7) = 1, the intermediate result fraction is in complement form, and the 2's complement of the intermediate fraction must be performed to obtain the true result fraction. As shown in the example, the true result fraction is .0001 0↔0. The one hex left shift that occurs yields a zero fraction result. Therefore, the result fraction located in DT is the true result fraction. The contents of D are transferred to T, and the sign, characteristic, and high-order fraction are stored into the FPR per the R1 field. An end op completes instruction execution.

If the signs were alike and the characteristic difference was within range (i.e., seven or less), the fractions are added [ROSAR(8–11) = 0110, 1110, or 1111]. First, however, the characteristics are equalized, if necessary, by right-shifting the fraction of the operand with the smallest characteristic one hex digit at a time and subtracting one from the characteristic difference. When the characteristic difference goes to zero, the characteristics are equal. Note that, if the first operand characteristic is larger, a 1 is *added* to the characteristic difference because the characteristic difference is in complement form.

The fractions are then added and the results are gated to DT and AB. A guard digit is retained in B(64–67). The characteristic of the larger operand, which is the characteristic of the intermediate result, is placed into F. When adding fractions, the possibility of a fraction overflow exists. A fraction overflow is indicated by a carry out of the high-order position, PA(8). After the two fractions are added, the intermediate result is placed into DT and AB. If A(7) = 1, a fraction overflow occurred. Therefore, the fraction is right-shifted one hex digit, and 1 is added to the intermediate result characteristic. Whether or not a fraction overflow occurred, the microprogram now determines whether the intermediate result fraction is normalized, not normalized, or equal to zero (the guard digit is included in the test for zero). If the fraction is normalized, the contents of STAT C are gated to the sign position of the LS bus, the result characteristic is transferred from F to T(32–39), and the result fraction is gated from D to T(40–63). This result (sign, characteristic, and fraction) is stored into the FPR specified by the R1 field. If the intermediate result fraction is not normalized, it is normalized by shifting left one hex digit at a time and subtracting one from the characteristic until a significant hex digit appears in the high-order position. After normalizing, the operation continues in the same manner as a normalized intermediate fraction. If the intermediate result is zero, a true zero is stored into the FPR specified by the R1 field, and the significance mask bit [PSW(39)] is tested. If the mask bit is a 1, the characteristic is stored with the zero fraction, and a program interruption is initiated. Regardless of the setting of the mask bit, the CC is set to 0, and the instruction is terminated by an end op.

If the intermediate result fraction is shifted right, the possibility of an exponent overflow exists. During normalization of the fraction, the possibility of an exponent underflow exists. When SAL(0) = 1 after a fraction shift (right or left), an exponent overflow or exponent underflow condition exists. The proper sign and fraction are stored. Because bit 0 of the characteristic (characteristic carry) is inhibited from entering LS by the 'RSLT-SIGN→LS' micro-order, the characteristic stored is 128 larger than the true result for underflow and 128 smaller on overflow.

Whenever an exponent overflow or exponent underflow condition exists, SAL(1) and PSW(38) are examined to determine whether a program interruption is to be executed. If SAL(1) = 0, an exponent overflow exists, and an interruption request is unconditionally generated. A program interruption occurs on all exponent overflows. If SAL(1) = 1, an exponent underflow exists; if PSW(38) = 1, an exponent underflow.program interruption request is generated. If PSW(38) = 0, exponent underflow is masked off, and a true zero is stored with no program interruption occurring.

Note that an interruption occurs on all exponent overflows. This overflow indicates that the value of the absolute result exceeds the limits of the machine; therefore, further action is necessary. In some scientific computations, very small numbers may be eliminated from an equation without serious error. An exponent underflow means that the computed result approaches zero. Therefore, the programmer may find that a program interruption is unnecessary, and a true zero result is desirable.

Significance and specification program interruption conditions may also exist during execution of an AER instruction. The action that occurs is shown in Diagram 5-206, and is discussed earlier in this section.

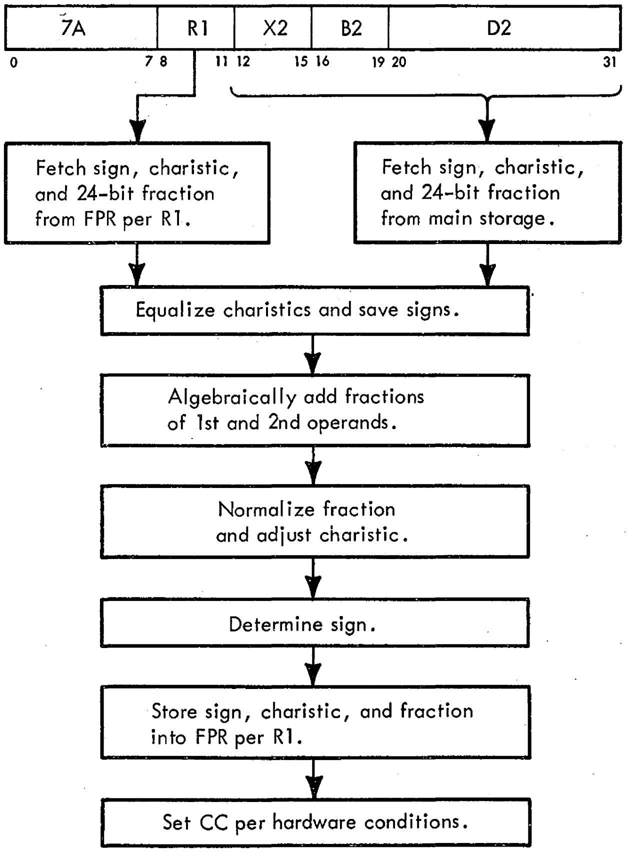
Tests for zero intermediate results are made at several points during instruction execution. If the result is zero, a program interruption occurs if PSW(39) = 1. The positive sign, the result characteristic, and a zero fraction are stored into LS. A program interruption occurs, and the program interruption routine determines the action to be taken. If PSW(39) = 0, a true zero result is stored into LS.

If the characteristics are not within limits when executing the AER instruction [ROSAR(8–11) = 0000, 0100, 1000, or 1100 on 'FLR' branch], the fraction with the largest characteristic is normalized and stored along with the sign into LS per the R1 field.

This discussion of the AER instruction is referred to in the discussions of Add-type instructions that follow. If the AER instruction is understood and the instruction differences noted, any short operand add-type instruction execution path can be followed by referring to Diagram 5-206.

Add Normalized, AE (7A) – RX Short Operands

- Algebraically add 2nd operand (in storage) to 1st operand (in FPR, per R1) and place normalized sum into 1st operand location.
- RX format: (See adjoining column.)
- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.


7A
R1
X2
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch sign, charistic, and 24-bit fraction from FPR per R1.
Fetch sign, charistic, and 24-bit fraction from main storage.
Equalize charistics and save signs.
Algebraically add fractions of 1st and 2nd operands.
Normalize fraction and adjust charistic.
Determine sign.
Store sign, charistic, and fraction into FPR per R1.
Set CC per hardware conditions.

- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.

The Add Normalized, AE, instruction (Diagram 5-206) algebraically adds the second operand (specified by the effective address) to the first operand (specified by R1), and places the normalized sum into the first operand location. The CC is set according to hardware conditions. The AE instruction uses 32-bit operands; therefore the contents of the low-order halves of the FPR's remain unchanged.

The conditions at the beginning of the execution phase are:

1. The first operand is in S and T.
2. A main storage request for the second operand has been issued per the effective address in D.
3. The contents of A and B are unknown.
4. The first 16 bits of the instruction are in E.

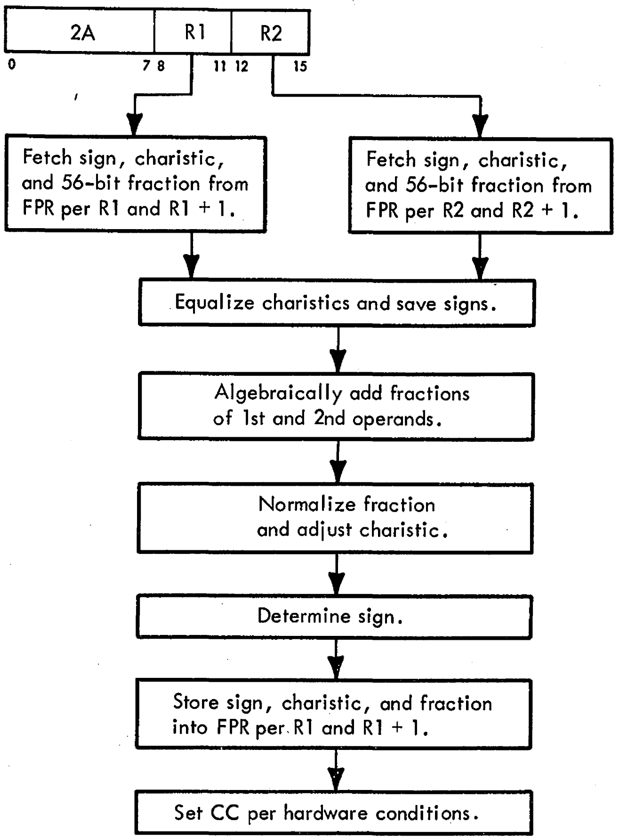
A specification test is made at the beginning of the execution phase. If a specification check exists, instruction execution is suppressed, and a specification program interruption occurs. Assume that no specification check exists.

Because the AE instruction uses short operands (32 bits) in the RX format, no low-order fractions need to be fetched. The first operand is moved from T to A. The second operand arrives from main storage and is placed into T. D(21) determines which word from the SDBO is gated to T. (Note that main storage is addressed on doubleword boundaries.) If D(21) = 0, SDBO(0–31) is gated to T; if D(21) = 1, SDBO(32–63) is gated to T.

The remainder of the AE instruction execution is identical to the execution of the AER instruction.

Add Normalized, ADR (2A) – RR Long Operands

- Algebraically add 2nd operand (in FPR, per R2 and R2 + 1) to 1st operand (in FPR, per R1 and R1 + 1) and place normalized sum into 1st operand location.
- RR format:


2A
R1
R2
0
7 8
11 12
15
Fetch sign, charistic, and 56-bit fraction from FPR per R1 and R1 + 1.
Fetch sign, charistic, and 56-bit fraction from FPR per R2 and R2 + 1.
Equalize charistics and save signs.
Algebraically add fractions of 1st and 2nd operands.
Normalize fraction and adjust charistic.
Determine sign.
Store sign, charistic, and fraction into FPR per R1 and R1 + 1.
Set CC per hardware conditions.

- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Low-order fractions of 1st and 2nd operands are in LS.
  Instruction is in E.
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.

The Add Normalized, ADR, instruction (Diagram 5-207, FEMDM) algebraically adds the second operand (specified by R2 and R2 + 1) to the first operand (specified by R1 and R1 + 1), and places the normalized sum into the first operand location. The CC is set according to hardware conditions. The ADR instruction uses 64-bit operands.

The conditions at the beginning of the execution phase are:

1. 32 bits of the first operand (sign, characteristic, and high-order fraction) are in A and B, and D contains the high-order 24 bits of the fraction.
2. 32 bits of the second operand (sign, characteristic, and high-order fraction) are in S and T.
3. The STC contains a count of 4.
4. The instruction is in E.

Because the ADR instruction uses long operands (64 bits) in the RR format, the low-order fractions of the first and second operands must be fetched from LS. The low-order fraction of the first operand is fetched from LS per E(8–11) + 1 and is placed into B via T and the parallel adder. The low-order fraction of the second operand is fetched from LS per E(12–15) + 1 and placed into T; the high-order fraction is placed into D.

The sign of the first operand is saved in STAT F and the sign of the second operand is saved in STAT C. The first operand characteristic is subtracted from the second operand characteristic, and the characteristic difference and the signs determine the next operation by means of a 10-way 'FLR' branch.

Note that with long operands, the characteristics can be equalized if the characteristic difference is less than or equal to 15. Assume that the following two characteristics are to be compared to determine the ROSAR value:

2nd operand characteristic:	$64_{10} = 1000000_2$
1st operand characteristic:	$73_{10} = 1001001_2$

The first operand characteristic is subtracted from the second operand characteristic as follows:

2nd operand characteristic:	0 1000000	
Complement of 1st operand characteristic:	1 0110110	2's complement
Add 1:	0 0000001	
Characteristic difference in SAL(0–7):	1 1110111	

Because no SAL(0) carry occurred, ROSAR(8) = 0. ROSAR(9) depends upon the signs assigned to the fractions of the two operands. The table in Sheet 2 of Diagram 5-207 shows the bit positions tested in SAL to determine the value of ROSAR(10) and whether preshifting is meaningful. In this example, SAL(0–3) = 1111. Therefore, because ADR is a long operand instruction, ROSAR(10) is set to 1 and preshifting is meaningful. ROSAR(11) = 0 because SAL(0–7) is not all 0's. For other examples of ROSAR setting on an 'FLR' branch, refer to the description of the AER instruction and Table 3-9. The examples shown in Table 3-9 indicate that the fraction of the operand with the smallest characteristic is shifted right when the characteristic difference is 15 or less for long operand instructions.

Four possible ROSAR(8–11) values (0010, 0110, 1010, and 1110) cause characteristic equalization and then an algebraic addition or subtraction of fractions. For example, ROSAR(8–11) is set to 0110, if an ADR instruction is being executed with operands having the following parameters: (1) the characteristic difference is less than 15, (2) the signs of the operands are alike, and (3) the second operand characteristic is the smaller of the two. The 0110 branch right-shifts the second operand in DT one hex digit and adds 1 to the characteristic difference in F until the characteristics are equal [SAL(4–7) = 1111].

After the characteristics are equalized, the fractions of the operands are added and the sum is placed into AB and DT. A plus sign is set into F(0) (in case of a zero fraction result), and the characteristic of the first operand is placed into F(1–7). STAT A is set if the fraction of the result equals zero.

Because ADR is a normalized instruction, the intermediate result is normalized, if necessary. First, however, a test determines whether the fraction overflowed due to the addition. (A carry into bit 7, the low-order bit of the characteristic, is considered a fraction overflow.) If the fraction did overflow, it is shifted right one hex digit, and a 1 is added to the characteristic.

The low-order fraction is then stored into the FPR specified by R1 + 1, and normalization proceeds if the result was not zero and ***not*** normalized. The low-order fraction is stored after each left-shift.

After normalization, the sign and the characteristic are inserted and stored with the high-order fraction into the first operand location (specified by R1). Assuming no error conditions or zero fraction, the CC is set, and an end-op cycle completes the execution.

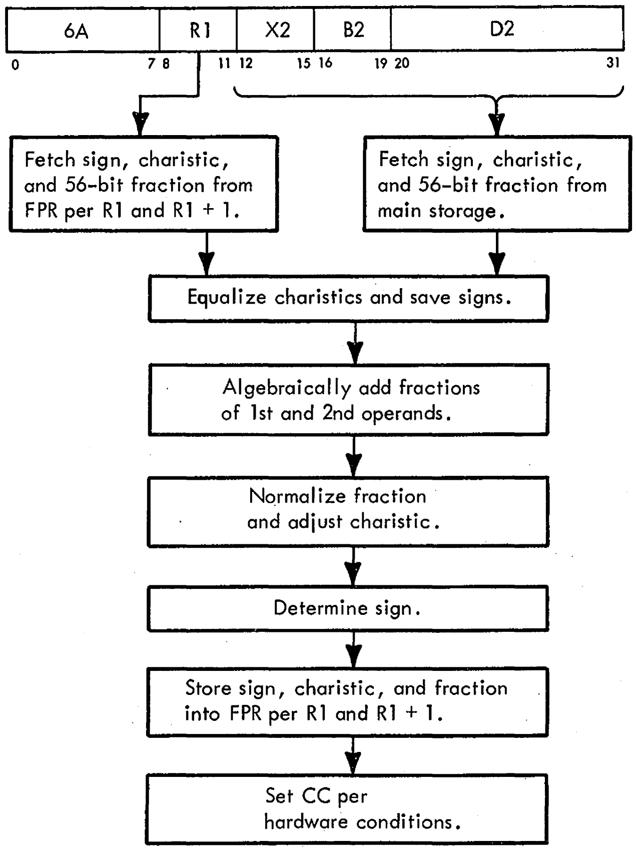
If signs are unlike, when in the normalizing loop, the intermediate result fraction can be left-shifted out of the high-order hex digit position if the intermediate fraction is 0001. Refer to the description of the AER instruction for an explanation of how this problem is handled.

All other operations that may occur during the execution of an ADR instruction are handled as described for an AER instruction, with the exception that the low-order fraction must be considered in all calculations. The major differences are:

1. An additional operand fetch is needed.
2. The low-order halves of the FPR's are used.

Add Normalized, AD (6A) – RX Long Operands

- Algebraically add 2nd operand (in storage) to 1st operand (in FPR, per R1 and R1 + 1) and place normalized sum into 1st operand location.
- RX format:



6A
R1
X2
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch sign, charistic, and 56-bit fraction from FPR per R1 and R1 + 1.
Fetch sign, charistic, and 56-bit fraction from main storage.
Equalize charistics and save signs.
Algebraically add fractions of 1st and 2nd operands.
Normalize fraction and adjust charistic.
Determine sign.
Store sign, charistic, and fraction into FPR per R1 and R1 + 1.
Set CC per hardware conditions.

- Conditions at start of execution:
  32 bits of 1st operand are in S and T.
  Low-order fraction of 1st operand is in LS.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.

The Add Normalized, AD, instruction (Diagram 5-207) algebraically adds the second operand (specified by the effective address) to the first operand (specified by R1 and R1 + 1), and places the normalized sum into the first operand location. The CC is set according to hardware conditions. The AD instruction uses 64-bit operands.

The conditions at the beginning of the execution phase are:

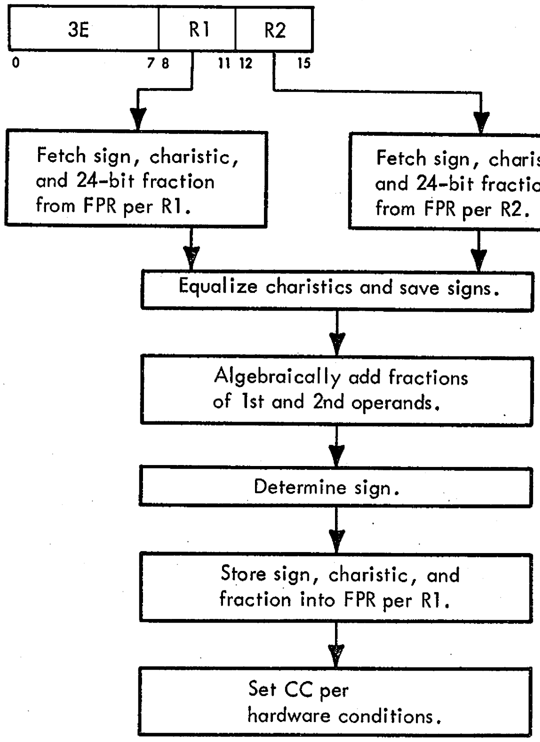
1. 32 bits of the first operand (sign, characteristic, and high-order fraction) are in S and T.
2. A main storage request for the second operand has been issued per the effective address in D.
3. The contents of A and B are unknown.
4. The first 16 bits of the instruction are in E.

Because the AD instruction uses long operands (64 bits) in the RX format, the low-order fraction of the first operand must be fetched from LS. Accordingly, the sign, characteristic, and high-order fraction of the first operand are placed into A. The low-order fraction is then fetched from LS per E(8–11) and routed to B via T and the parallel adder. The 64-bit second operand is fetched from main storage per D and placed into ST; the high-order fraction is also placed into D. The sign of the first operand is saved in STAT F and the sign of the second operand is saved in STAT C.

The first operand characteristic is then subtracted from the second operand characteristic, and the characteristic difference and the signs determine the next operation by means of a 10-way 'FLR' branch. The remainder of the execution of the AD instruction is identical to that of the ADR instruction.

Add Unnormalized, AUR (3E) – RR Short Operands

- Algebraically add 2nd operand (in FPR, per R2) to 1st operand (in FPR, per R1) and place unnormalized sum into 1st operand location.
- RR format:


3E
R1
R2
0
7 8
11 12
15
Fetch sign, charistic, and 24-bit fraction from FPR per R1.
Fetch sign, charistic, and 24-bit fraction from FPR per R2.
Equalize charistics and save signs.
Algebraically add fractions of 1st and 2nd operands.
Determine sign.
Store sign, charistic, and fraction into FPR per R1.
Set CC per hardware conditions.

- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.

The Add Unnormalized, AUR, instruction (Diagram 5-206) algebraically adds the second operand (specified by R2) to the first operand (specified by R1), and places the unnormalized sum into the first operand location. The CC is set according to hardware conditions. Note that the guard digit is not examined to determine the CC setting or checked for a significance condition. Also, because the characteristic is not reduced for normalization, exponent underflow cannot occur.

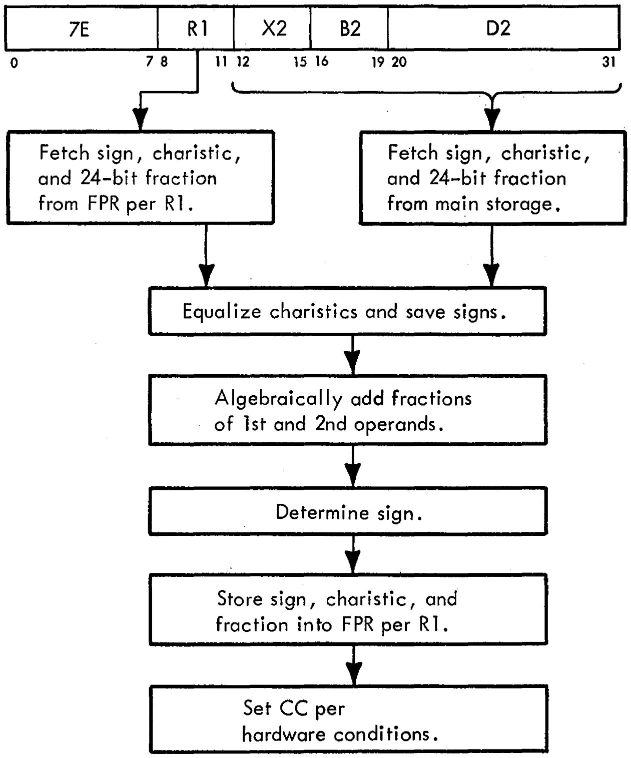
The execution of the AUR instruction is identical to that of the AER instruction until the branch on the type of instruction is performed. After the fraction is tested for overflow and shifted right (if necessary), the result is checked to see if it is zero. If the result is not zero, the result is immediately stored, along with the sign and characteristic, into the FPR specified by R1.

A test then determines whether equalization had occurred (STAT D set). If it did, a possibility exists that the only significant digit may be the guard digit. Therefore, if equalization did occur and no exponent overflow condition exists, the guard digit is removed from the result and the fraction is tested to see if it is zero. If the fraction equals zero, a true zero is stored and a significance program interruption is initiated. Otherwise, the instruction terminates normally.

Add Unnormalized, AU (7E) – RX Short Operands

- Algebraically add 2nd operand (in storage) to 1st operand (in FPR, per R1) and place unnormalized sum into 1st operand location.
- RX format: (See left column of next page.)
- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.

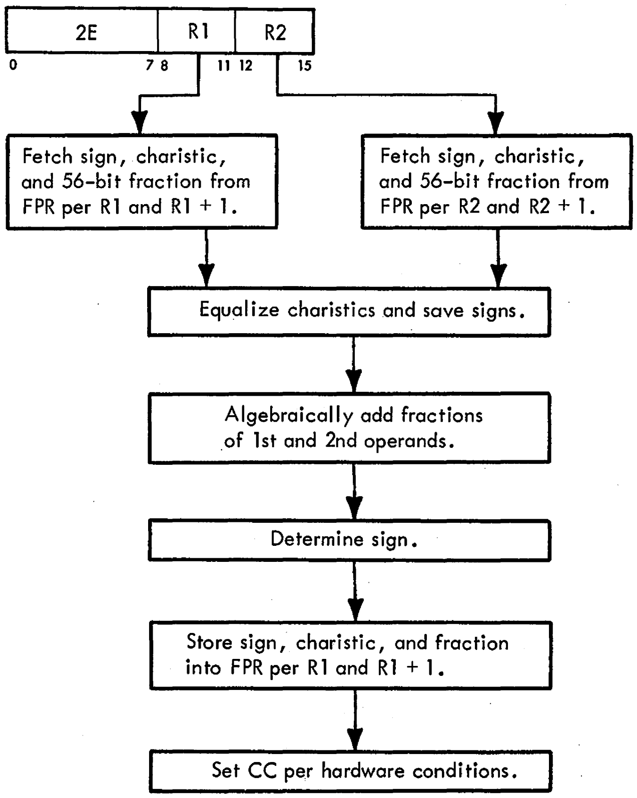
The Add Unnormalized, AU, instruction (Diagram 5-206) algebraically adds the second operand (specified by the effective address) to the first operand (specified by R1), and places the unnormalized sum into the first operand location. Exponent underflow cannot occur. The CC is set


7E
R1
X2
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch sign, charistic, and 24-bit fraction from FPR per R1.
Fetch sign, charistic, and 24-bit fraction from main storage.
Equalize charistics and save signs.
Algebraically add fractions of 1st and 2nd operands.
Determine sign.
Store sign, charistic, and fraction into FPR per R1.
Set CC per hardware conditions.

according to hardware conditions. The execution is identical to that of the AE instruction until the branch to determine an unnormalized, normalized, or compare instruction. From this point, the execution is identical to that of the AUR instruction.

Add Unnormalized, AWR (2E) – RR Long Operands

- Algebraically add 2nd operand (in FPR, per R2 and R2 + 1) to 1st operand (in FPR, per R1 and R1 + 1) and place unnormalized sum into 1st operand location.
- RR format: (See adjoining column.)
- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Low-order fractions of 1st and 2nd operands are in LS.
  Instruction is in E.
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.


2E
R1
R2
0
7 8
11 12
15
Fetch sign, charistic, and 56-bit fraction from FPR per R1 and R1 + 1.
Fetch sign, charistic, and 56-bit fraction from FPR per R2 and R2 + 1.
Equalize charistics and save signs.
Algebraically add fractions of 1st and 2nd operands.
Determine sign.
Store sign, charistic, and fraction into FPR per R1 and R1 + 1.
Set CC per hardware conditions.

The Add Unnormalized, AWR, instruction (Diagram 5-207) algebraically adds the second operand (specified by R2 and R2 + 1) to the first operand (specified by R1 and R1 + 1) and places the unnormalized sum into the first operand location. Exponent underflow cannot occur.

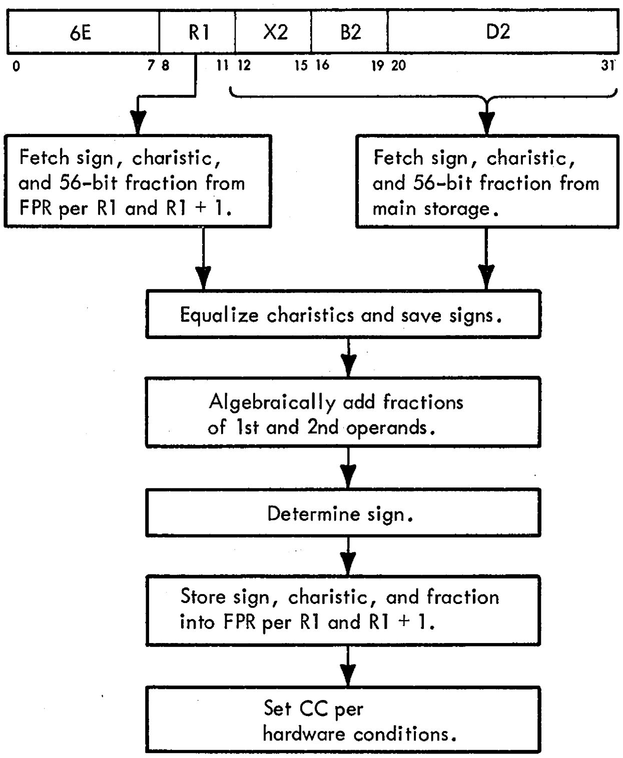
The execution phase is identical to that of the ADR instruction until the branch to determine an unnormalized, normalized, or compare instruction. Assume that the operand signs are alike and that the characteristic difference is less than 15. On Sheet 3 of Diagram 5-207, after the branch, a test determines whether the fraction of the sum overflowed (indicated by a carry into bit 7). If an overflow occurred, the fraction is shifted right one digit, and a 1 is added to the characteristic.

The low-order position of the sum (bits 32–63) is then stored into the FPR designated by R1 + 1. The result fraction is then tested to see if it equals zero. If it does not equal zero, the sign, characteristic, and high-order fraction are stored into the FPR specified by R1. If no exponent overflow occurred, the CC is set and the instruction is terminated by an end op.

Add Unnormalized, AW (6E) – RX Long Operands

- Algebraically add 2nd operand (in storage) to 1st operand (in FPR, per R1 and R1 + 1) and place unnormalized sum into 1st operand location.

- RX format:


6E
R1
X2
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch sign, charistic, and 56-bit fraction from FPR per R1 and R1 + 1.
Fetch sign, charistic, and 56-bit fraction from main storage.
Equalize charistics and save signs.
Algebraically add fractions of 1st and 2nd operands.
Determine sign.
Store sign, charistic, and fraction into FPR per R1 and R1 + 1.
Set CC per hardware conditions.

- Conditions at start of execution:
  32 bits of 1st operand are in S and T.
  Low-order fraction of 1st operand is in LS.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.

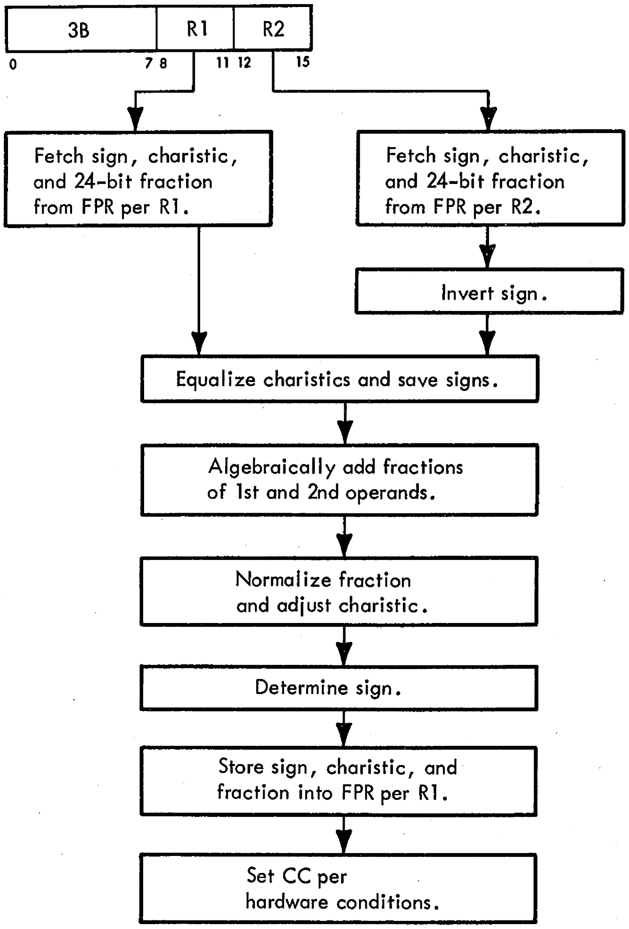
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.

The Add Unnormalized, AW, instruction (Diagram 5-207) algebraically adds the second operand (specified by the effective address) to the first operand (specified by R1 and R1 + 1) and places the unnormalized sum into the first operand location. Exponent underflow cannot occur. The CC is set according to hardware conditions.

The execution phase is identical to that of the AD instruction until the branch to determine an unnormalized, normalized, or compare instruction. From this point, execution is identical to execution of the AWR instruction.

Subtract Normalized, SER (3B) – RR Short Operands

- Algebraically subtract 2nd operand (in FPR per R2) from 1st operand (in FPR, per R1) and place normalized difference into 1st operand location.
- RR format:


3B
R1
R2
0
7 8
11 12
15
Fetch sign, charistic, and 24-bit fraction from FPR per R1.
Fetch sign, charistic, and 24-bit fraction from FPR per R2.
Invert sign.
Equalize charistics and save signs.
Algebraically add fractions of 1st and 2nd operands.
Normalize fraction and adjust charistic.
Determine sign.
Store sign, charistic, and fraction into FPR per R1.
Set CC per hardware conditions.

- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.

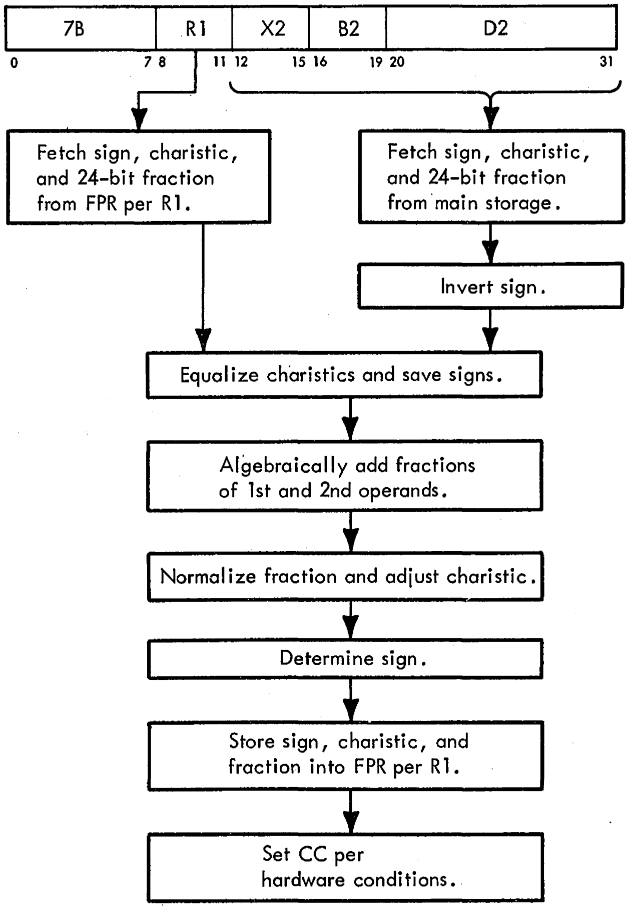
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.

The Subtract Normalized, SER, instruction (Diagram 5-206) algebraically subtracts the second operand (specified by R2) from the first operand (specified by R1), and places the normalized difference into the first operand location. The second operand location remains unchanged. The CC is set according to hardware conditions.

For subtract instructions, the sign of the second operand is complemented, after which the algebraic subtraction is treated as an algebraic addition. Therefore, execution of the SER instruction is identical to that of the AER instruction except that the complement of the second operand sign rather than the true sign is gated to STAT C.

Subtract Normalized, SE (7B) – RX Short Operands

- Algebraically subtract 2nd operand (in storage) from 1st operand (in FPR, per R1) and place normalized difference into 1st operand location.
- RX format:



7B
R1
X2
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch sign, charistic, and 24-bit fraction from FPR per R1.
Fetch sign, charistic, and 24-bit fraction from main storage.
Invert sign.
Equalize charistics and save signs.
Algebraically add fractions of 1st and 2nd operands.
Normalize fraction and adjust charistic.
Determine sign.
Store sign, charistic, and fraction into FPR per R1.
Set CC per hardware conditions.

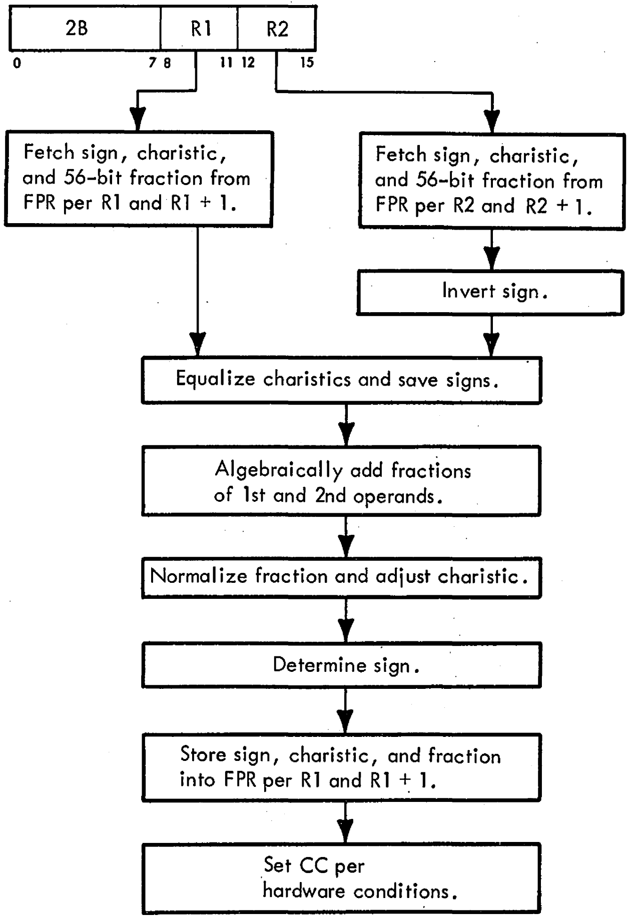
- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.

The Subtract Normalized, SE, instruction (Diagram 5-206) algebraically subtracts the second operand (specified by the effective address) from the first operand (specified by R1) and places the normalized difference into the first operand location. The CC is set according to hardware conditions.

Execution of the SE instruction is identical to that of the AE instruction except that the complement of the second operand sign instead of the true sign is gated to STAT C.

Subtract Normalized, SDR (2B) – RR Long Operands

- Algebraically subtract 2nd operand (in FPR, per R2 and R2 + 1) from 1st operand (in FPR, per R1 and R1 + 1) and place normalized difference into 1st operand location.
- RR format:



2B
R1
R2
0
7 8
11 12
15
Fetch sign, charistic, and 56-bit fraction from FPR per R1 and R1 + 1.
Fetch sign, charistic, and 56-bit fraction from FPR per R2 and R2 + 1.
Invert sign.
Equalize charistics and save signs.
Algebraically add fractions of 1st and 2nd operands.
Normalize fraction and adjust charistic.
Determine sign.
Store sign, charistic, and fraction into FPR per R1 and R1 + 1.
Set CC per hardware conditions.

- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Low-order fractions of 1st and 2nd operands are in LS.
  Instruction is in E.

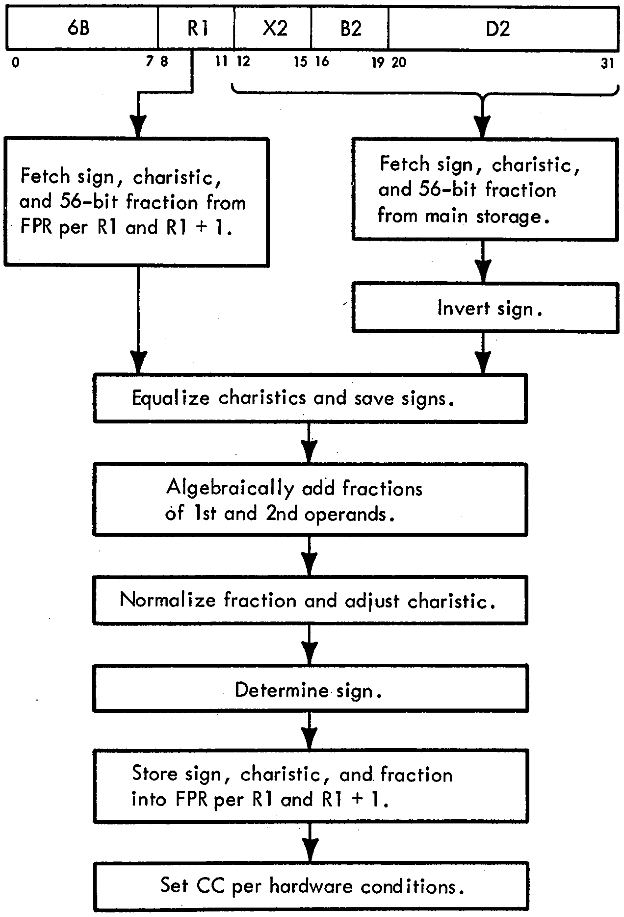
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.

The Subtract Normalized, SDR, instruction (Diagram 5-207) algebraically subtracts the second operand (specified by R2 and R2 + 1) from the first operand (specified by R1 and R1 + 1) and places the normalized difference into the first operand location. The CC is set according to hardware conditions.

Execution of the SDR instruction is identical to that of the ADR instruction except that the complement of the second operand sign instead of the true sign is gated to STAT C.

Subtract Normalized, SD (6B) – RX Long Operands

- Algebraically subtract 2nd operand (in storage) from 1st operand (in FPR, per R1 and R1 + 1) and place normalized difference into 1st operand location.
- RX format:



6B
R1
X2
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch sign, charistic, and 56-bit fraction from FPR per R1 and R1 + 1.
Fetch sign, charistic, and 56-bit fraction from main storage.
Invert sign.
Equalize charistics and save signs.
Algebraically add fractions of 1st and 2nd operands.
Normalize fraction and adjust charistic.
Determine sign.
Store sign, charistic, and fraction into FPR per R1 and R1 + 1.
Set CC per hardware conditions.

- Conditions at start of execution:
  32 bits of 1st operand are in S and T.
  Low-order fraction of 1st operand is in LS.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.

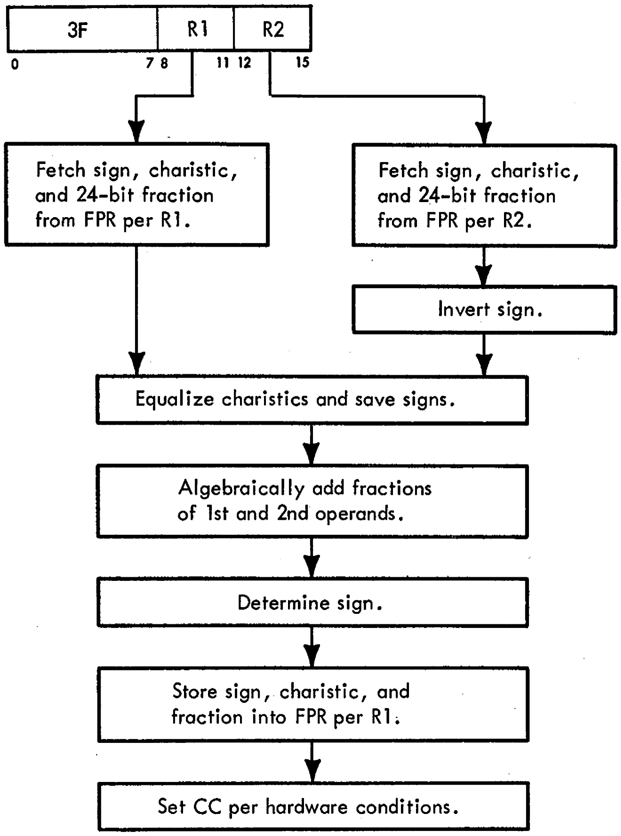
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.

The Subtract Normalized, SD, instruction (Diagram 5-207) algebraically subtracts the second operand (specified by the effective address) from the first operand (specified by R1 and R1 + 1) and places the normalized difference into the first operand location. The CC is set according to hardware conditions.

Execution of the SD instruction is identical to the execution of the AD instruction except that the complement of the second operand sign instead of the true sign is gated to STAT C.

Subtract Unnormalized, SUR (3F) – RR Short Operands

- Algebraically subtract 2nd operand (in FPR, per R2) from 1st operand (in FPR, per R1) and place unnormalized difference into 1st operand location.
- RR format:



3F
R1
R2
0
7 8
11 12
15
Fetch sign, charistic, and 24-bit fraction from FPR per R1.
Fetch sign, charistic, and 24-bit fraction from FPR per R2.
Invert sign.
Equalize charistics and save signs.
Algebraically add fractions of 1st and 2nd operands.
Determine sign.
Store sign, charistic, and fraction into FPR per R1.
Set CC per hardware conditions.

- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.

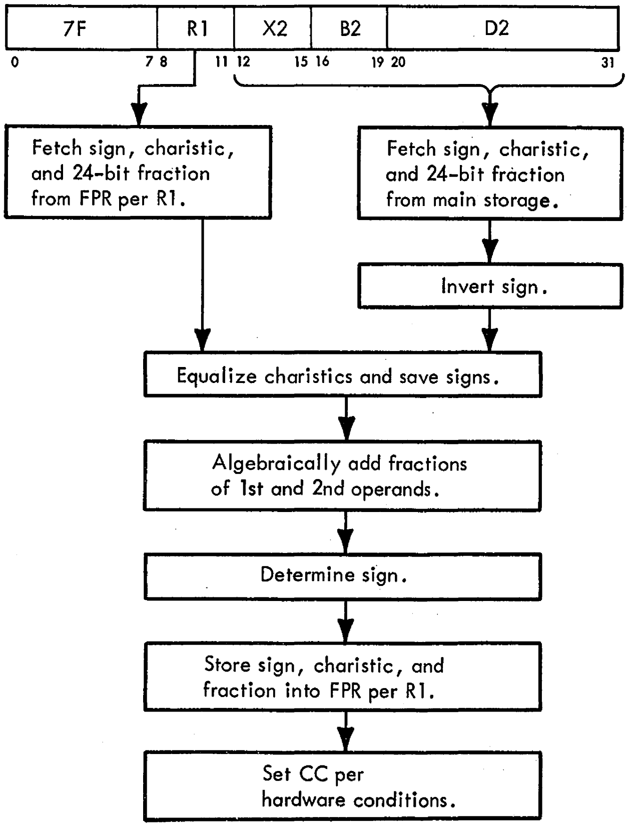
The Subtract Unnormalized, SUR, instruction (Diagram 5-206) algebraically subtracts the second operand (specified by R2) from the first operand (specified by R1) and places the unnormalized difference into the first operand location. The CC is set according to hardware conditions.

Except that the intermediate results of the Subtract Unnormalized instructions are not normalized, the operation is the same as that of the Subtract Normalized instructions. That is, the second operand sign is inverted (and saved in STAT C) and the fractions are algebraically added.

The SUR instruction is executed in the same manner as the AUR instruction except that the complement of the second operand sign instead of the true sign is gated to STAT C. Note that, when executing Subtract Unnormalized instructions, the guard digit is not examined to determine the CC setting or checked for a significance condition. Also, as in Unnormalized Add instructions, exponent underflow cannot occur.

Subtract Unnormalized, SU (7F) — RX Short Operands

- Algebraically subtract 2nd operand (in storage) from 1st operand (in FPR, per R1) and place unnormalized difference into 1st operand location.
- RX format:


7F
R1
X2
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch sign, charistic, and 24-bit fraction from FPR per R1.
Fetch sign, charistic, and 24-bit fraction from main storage.
Invert sign.
Equalize charistics and save signs.
Algebraically add fractions of 1st and 2nd operands.
Determine sign.
Store sign, charistic, and fraction into FPR per R1.
Set CC per hardware conditions.

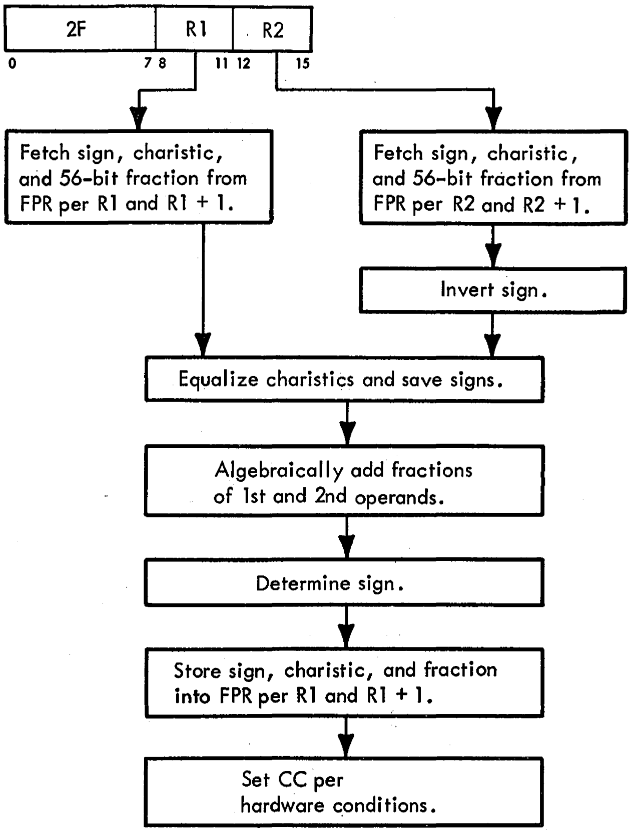
- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.

The Subtract Unnormalized, SU, instruction (Diagram 5-206) algebraically subtracts the second operand (specified by the effective address) from the first operand (specified by R1) and places the unnormalized difference into the first operand location. The CC is set according to hardware conditions.

Execution of the SU instruction is identical to that of the AU instruction except that the complement of the second operand sign instead of the true sign is gated to STAT C.

Subtract Unnormalized, SWR (2F) — RR Long Operands

- Algebraically subtract 2nd operand (in FPR, per R2 and R2 + 1) from 1st operand (in FPR, per R1 and R1 + 1) and place unnormalized difference into 1st operand location.
- RR format:


2F
R1
R2
0
7 8
11 12
15
Fetch sign, charistic, and 56-bit fraction from FPR per R1 and R1 + 1.
Fetch sign, charistic, and 56-bit fraction from FPR per R2 and R2 + 1.
Invert sign.
Equalize charistics and save signs.
Algebraically add fractions of 1st and 2nd operands.
Determine sign.
Store sign, charistic, and fraction into FPR per R1 and R1 + 1.
Set CC per hardware conditions.

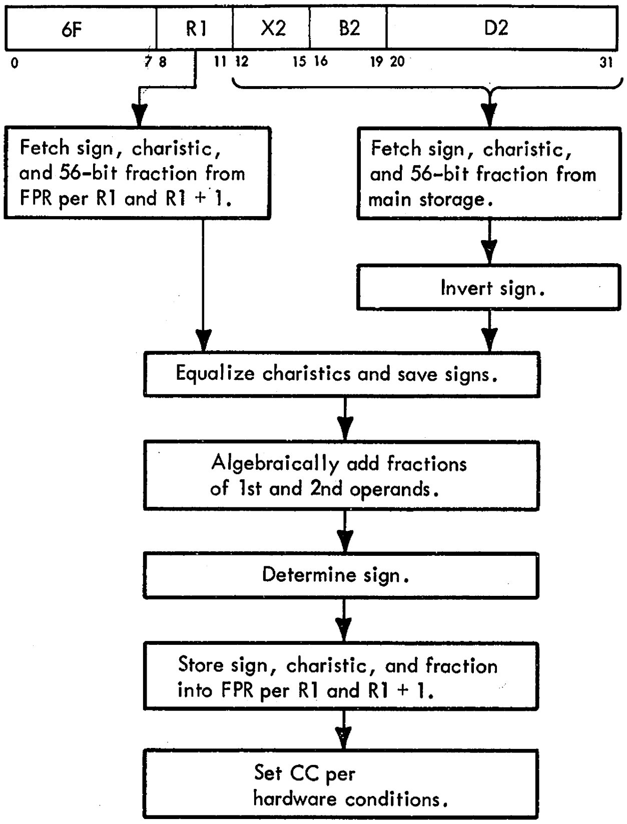
- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Low-order fractions of 1st and 2nd operands are in LS.
  Instruction is in E.
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.

The Subtract Unnormalized, SWR, instruction (Diagram 5-207) algebraically subtracts the second operand (specified by R2 and R2 + 1) from the first operand (specified by R1 and R1 + 1) and places the unnormalized difference into the first operand location. The CC is set according to hardware conditions.

Execution of the SWR instruction is identical to that of the AWR instruction except that the complement of the second operand sign instead of the true sign is gated to STAT C.

Subtract Unnormalized, SW (6F) – RX Long Operands

- Algebraically subtract 2nd operand (in storage) from 1st operand (in FPR, per R1 and R1 + 1) and place unnormalized difference into 1st operand location.
- RX format:


6F
R1
X2
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch sign, charistic, and 56-bit fraction from FPR per R1 and R1 + 1.
Fetch sign, charistic, and 56-bit fraction from main storage.
Invert sign.
Equalize charistics and save signs.
Algebraically add fractions of 1st and 2nd operands.
Determine sign.
Store sign, charistic, and fraction into FPR per R1 and R1 + 1.
Set CC per hardware conditions.

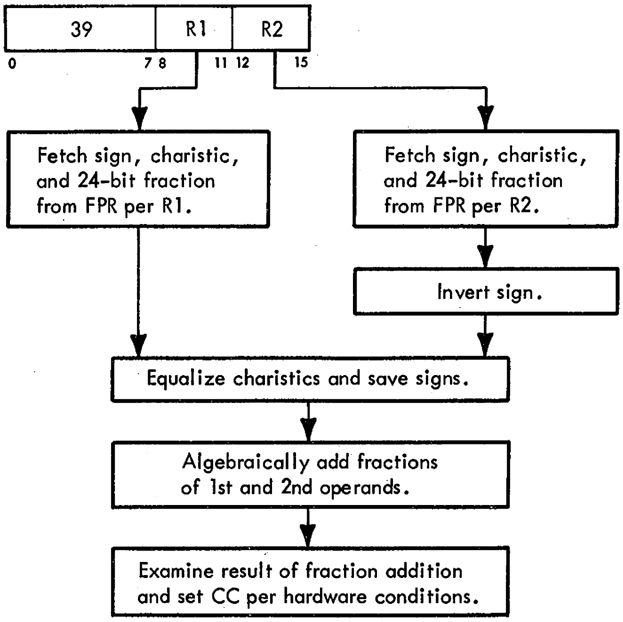
- Conditions at start of execution:
  32 bits of 1st operand are in S and T.
  Low-order fraction of 1st operand is in LS.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction is less than zero: CC = 1.
  Result fraction is greater than zero: CC = 2.

The Subtract Unnormalized, SW, instruction (Diagram 5-207) algebraically subtracts the second operand (specified by the effective address) from the first operand (specified by R1 and R1 + 1) and places the unnormalized difference into the first operand location. The CC is set according to hardware conditions.

Execution of the SW instruction is identical to that of the AW instruction except that the complement of the second operand sign instead of the true sign is gated to STAT C.

Compare, CER (39) – RR Short Operands

- Algebraically compare 1st operand (in FPR, per R1) with 2nd operand (in FPR, per R2); CC indicates result.
- RR format:


39
R1
R2
0
7 8
11 12
15
Fetch sign, charistic, and 24-bit fraction from FPR per R1.
Fetch sign, charistic, and 24-bit fraction from FPR per R2.
Invert sign.
Equalize charistics and save signs.
Algebraically add fractions of 1st and 2nd operands.
Examine result of fraction addition and set CC per hardware conditions.

- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.

- CC setting:
  Operands are equal: CC = 0.
  1st operand is less than 2nd operand: CC = 1.
  1st operand is greater than 2nd operand: CC = 2.

The Compare, CER, instruction (Diagram 5-206) algebraically compares the first operand (specified by R1) with the second operand (specified by R2); the CC indicates that the first operand is equal to, less than, or greater than the second operand.

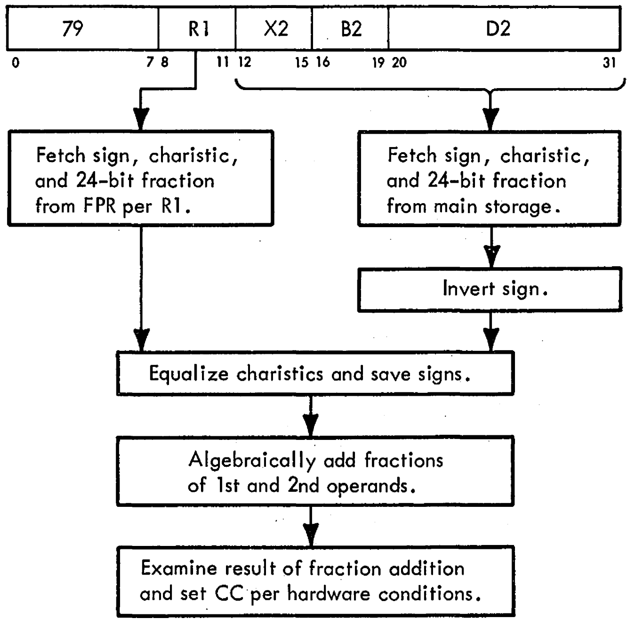
Comparison is algebraic, taking into account the sign, fraction, and characteristic of each operand. An exponent inequality is not decisive for magnitude determination because the fractions may have different numbers of leading zeros. Equality is established by following the rules for floating-point subtraction. The intermediate result is not normalized or stored. When the intermediate result, including a possible guard digit, is zero, the operands are equal. Numbers with zero fractions compare equal even when they differ in sign or characteristic. Exponent overflow, exponent underflow, or significance check cannot occur. The CC is set per hardware conditions at end-op time.

In the CER instruction, the contents of the low-order halves of the FPR's are ignored. (Neither operand location is changed as a result of any compare instructions.)

Execution of the CER instruction is identical to that of the SER instruction until the branch to determine an unnormalized, normalized, or compare instruction. At that point, the CC is set per hardware conditions, and the instruction is terminated by an end op.

Compare, CE (79) – RX Short Operands

- Algebraically compare 1st operand (in FPR, per R1) with 2nd operand (in storage); CC indicates result.
- RX format:


79
R1
X2
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch sign, charistic, and 24-bit fraction from FPR per R1.
Fetch sign, charistic, and 24-bit fraction from main storage.
Invert sign.
Equalize charistics and save signs.
Algebraically add fractions of 1st and 2nd operands.
Examine result of fraction addition and set CC per hardware conditions.

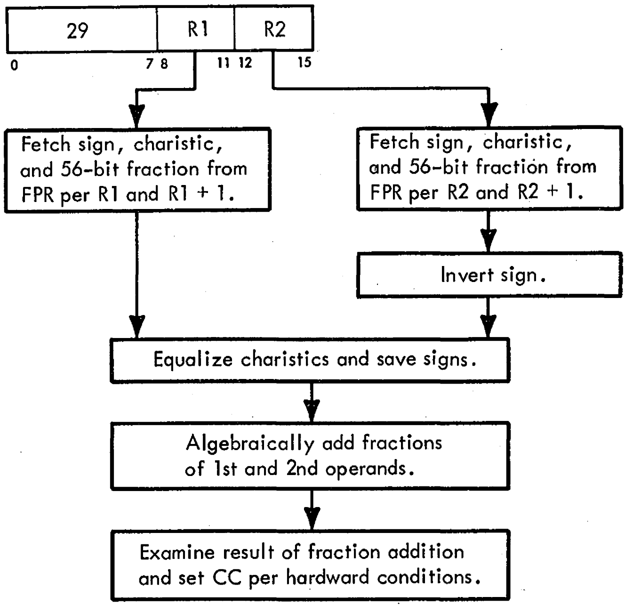
- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request for 2nd operand has been issued per D.
  1st 16 bits of instruction are in E.
- CC setting:
  Operands are equal: CC = 0.
  1st operand is less than 2nd operand: CC = 1.
  1st operand is greater than 2nd operand: CC = 2.

The Compare, CE, instruction (Diagram 5-206) algebraically compares the first operand (specified by R1) with the second operand (specified by the effective address); the CC indicates the result. Exponent underflow, exponent overflow, or significance check cannot occur.

Execution of the CE instruction is identical to that of the SE instruction until the branch to determine an unnormalized, normalized, or compare instruction. At that point, the CC is set per hardware conditions, and the instruction is terminated by an end op.

Compare, CDR (29) – RR Long Operands

- Algebraically compare 1st operand (in FPR, per R1 and R1 + 1) with 2nd operand (in FPR, per R2 and R2 + 1); CC indicates result.
- RR format:


29
R1
R2
0
7 8
11 12
15
Fetch sign, charistic, and 56-bit fraction from FPR per R1 and R1 + 1.
Fetch sign, charistic, and 56-bit fraction from FPR per R2 and R2 + 1.
Invert sign.
Equalize charistics and save signs.
Algebraically add fractions of 1st and 2nd operands.
Examine result of fraction addition and set CC per hardward conditions.

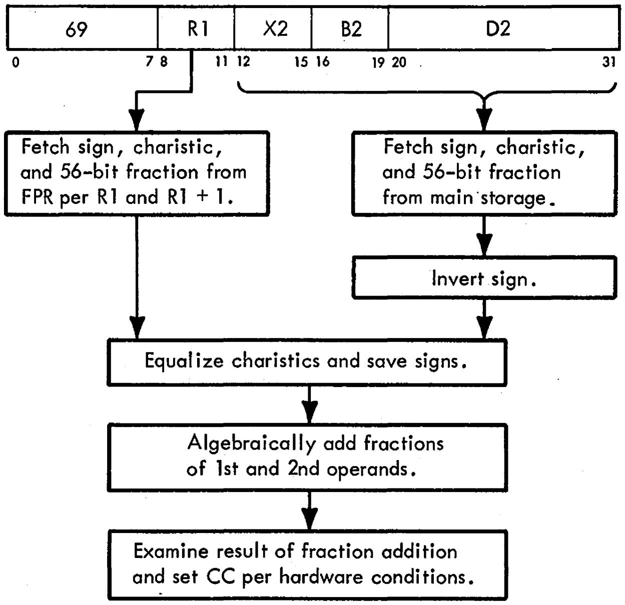
- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Low-order fractions of 1st and 2nd operands are in LS.
  Instruction is in E.
- CC setting:
  Operands are equal: CC = 0.
  1st operand is less than 2nd operand: CC = 1.
  1st operand is greater than 2nd operand: CC = 2.

The Compare, CDR, instruction (Diagram 5-207) algebraically compares the first operand (specified by R1 and R1 + 1) with the second operand (specified by R2 and R2 + 1); the CC indicates the result. Exponent underflow, exponent overflow, or significance check cannot occur.

Execution of the CDR instruction is identical to that of the SDR instruction until the branch to determine an unnormalized, normalized, or compare instruction. At that point, the CC is set per hardware conditions, and the instruction is terminated by an end op.

Compare, CD (69) – RX Long Operands

- Algebraically compare 1st operand (in FPR, per R1 and R1 + 1) with 2nd operand (in storage); CC indicates result.
- RX format:



69
R1
X2
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch sign, charistic, and 56-bit fraction from FPR per R1 and R1 + 1.
Fetch sign, charistic, and 56-bit fraction from main storage.
Invert sign.
Equalize charistics and save signs.
Algebraically add fractions of 1st and 2nd operands.
Examine result of fraction addition and set CC per hardware conditions.

- Conditions at start of execution:
  32 bits of 1st operand are in S and T.
  Low-order fraction of 1st operand is in LS.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.
- CC setting:
  Operands are equal: CC = 0.
  1st operand is less than 2nd operand: CC = 1.
  1st operand is greater than 2nd operand: CC = 2.

The Compare, CD, instruction (Diagram 5-207) algebraically compares the first operand (specified by R1 and R1 + 1) with the second operand (specified by the effective address); the CC indicates the result. Exponent underflow, exponent overflow, or significance check cannot occur.

Execution of the CD instruction is identical to that of the SD instruction until the branch to determine an unnormalized, normalized, or compare instruction. At that point, the CC is set per hardware conditions, and the instruction is terminated by an end op.

HALVE

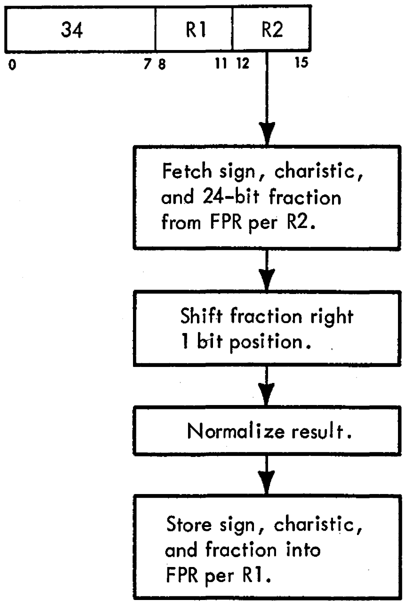
The Halve instructions (HER and HDR) divide the second operand by 2 and place the normalized quotient into the first operand location. The Halve instructions are in the RR format with short and long operand options available. In the HER instruction, the low-order half of the result register remains unchanged.

After the second operand is in ST, the sign and the characteristic are saved in F, and the high-order fraction (24 bits) is placed into D (long operands). Shifting the fraction one bit position to the right divides the operand by 2. Because the data in PAL cannot be shifted right 1 directly, two machine cycles are necessary. First the fraction is shifted left 1 from D to the parallel adder and then shifted right 4 to PAL, thus yielding an effective right 3 shift. Next, the fraction is placed into AB. A left 2 shift occurs when the fraction is routed to DT via the parallel adder, thus resulting in a right 1 shift and thereby dividing the fraction by 2. Guard digits are saved and used if normalization is necessary. After the fraction is normalized, the sign, characteristic, and fraction are stored into LS per R1, completing instruction execution.

The halve operation differs from the divide operation in that 2 is the only divisor.

Halve, HER (34) – RR Short Operands

- Divide 2nd operand (in FPR, per R2) by 2 and place normalized quotient into 1st operand location (in FPR, per R1).
- RR format:



34
R1
R2
0
7 8
11 12
15
Fetch sign, charistic, and 24-bit fraction from FPR per R2.
Shift fraction right 1 bit position.
Normalize result.
Store sign, charistic, and fraction into FPR per R1.

- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.

The Halve, HER, instruction (Diagram 5-208, FEMDM) divides the second operand (specified by R2) by 2 and places the normalized quotient into the FPR specified by R1. To divide by 2, the fraction is shifted right one bit position.

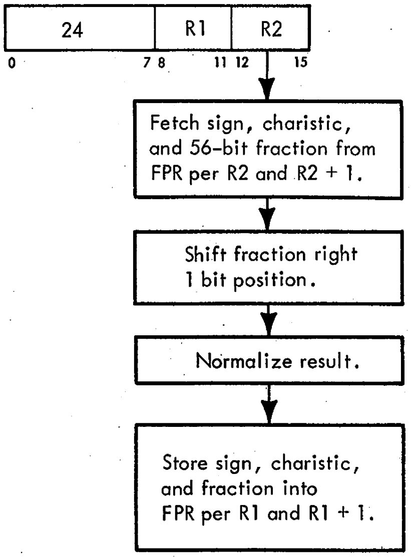
At the start of execution, the second operand is in S and T. After a specification test, the fraction is placed into D, the sign is saved in STAT C, and the characteristic is saved in F. The fraction in D is then shifted right one bit position. A test is initiated to determine whether the fraction is zero and normalized. If the fraction equals zero, the FPR specified by R1 is set to a true zero and the operation terminates with an end op. If the fraction contains leading zero's, it is normalized by shifting left one hex digit at a time until a significant hex digit appears in the high-order position. A 1 is subtracted from the characteristic for each shift.

After normalization, the characteristic and fraction are gated to T from F and D, respectively. The result is then stored into the FPR specified by R1. The sign position is set by forcing the contents of STAT C onto the LS bus as the fraction and characteristic are being stored.

The characteristic is then tested for a possible exponent underflow. If no underflow occurred, the instruction is terminated. If an underflow did occur, the remainder of the operation is determined by the state of the underflow mask bit.

Halve, HDR (24) – RR Long Operands

- Divide 2nd operand (in FPR, per R2 and R2 + 1) by 2 and place normalized quotient into 1st operand location (in FPR, per R1 and R1 + 1).
- RR format:


24
R1
R2
0
7 8
11 12
15
Fetch sign, charistic, and 56-bit fraction from FPR per R2 and R2 + 1.
Shift fraction right 1 bit position.
Normalize result.
Store sign, charistic, and fraction into FPR per R1 and R1 + 1.

- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Low-order fractions of 1st and 2nd operands are in LS.
  Instruction is in E.

The Halve, HDR, instruction (Diagram 5-209, FEMDM) divides the second operand, (specified by R2 and R2 + 1) by 2 and places the normalized quotient into the FPR specified by R1 and R1 + 1. To divide by 2, the fraction is shifted right one bit position.

At the start of the execution phase, the sign, characteristic, and high-order fraction are in S and T. After a specification test, the high-order fraction is placed into D and the low-order fraction is fetched from LS per R1 + 1 and gated to T. The sign is saved in STAT C, and the characteristic is saved in F. The fraction in DT is then shifted right one bit position, and a test determines whether the fraction is equal to zero and normalized. If the fraction equals zero, the FPR specified by R1 and R1 + 1 is set to a true zero and the operation terminates with an end op. If the fraction contains leading zero's, it is normalized by shifting left one hex digit at a time until a significant hex digit appears in the high-order position. A 1 is subtracted from the characteristic for each shift.

After normalization, the low-order fraction is stored into the FPR specified by R1 + 1. The characteristic and high-order fraction are then gated to T from F and D, respectively. The result is stored into the FPR specified by R1. The sign position is set by forcing the contents of STAT C onto the LS bus as the fraction and characteristic are being stored.

The characteristic is then tested for a possible exponent underflow. If no underflow occurred, the instruction is terminated. If an underflow did occur, the remainder of the operation is determined by the state of the underflow mask bit.

MULTIPLY

- Multiplies 1st operand and 2nd operand and places normalized product into 1st operand location.
- *Note:* In 2065 floating-point multiply operations, roles of 1st and 2nd operands are reversed from roles defined in *System/360 Principles of Operation*, SRL, Form A22-6821-6. That is, 2nd operand is multiplicand and 1st operand is multiplier. (Interchanging operand roles does not affect product.) Result, however, still replaces 1st operand.
- Product is 64 bits for both short and long operand instructions.
- Characteristics are added and 64 is subtracted to obtain intermediate characteristic.
- Operands are prenormalized before multiplying.

- Product is normalized before storing.
- Sign of product is determined algebraically.

Two floating-point numbers are multiplied by adding their characteristics and multiplying their fractions. For example, if 1250 is to be multiplied by 5, converting these numbers to hex notation yields the equation

$$(.4E2_{16} \times 16^3) \times (.5 \times 16^1).$$

The product is obtained, as follows:

$$\begin{array}{rr} & .4E2_{16} \times 16^3 \\ \times & .5_{16} \times 16^1 \\ \hline \text{Product of fractions} \longrightarrow & .186A_{16} \times 16^4 \nwarrow \text{Exponents are added} \end{array}$$

The product ($.186A_{16}$ x 16^4) equals 6250 (decimal), which is the product of 1250 x 5.

When two floating-point numbers are multiplied, the characteristics are added to yield the final characteristic value of the product, as shown above. Because excess-64 notation is used, 64 must be subtracted from the characteristic sum because the characteristic value is in excess-128 [(C1 + 64) + (C2 + 64) = C1 + C2 + 128] after characteristic addition. When 64 is subtracted, the result is returned to excess-64 notation (C1 + C2 + 128 - 64 = C1 + C2 + 64). For instance, in floating-point format, the exponents used in the example above yield characteristics of 67 (3 + 64) and 65 (1 + 64). The sum of these characteristics is 132. Subtracting 64 from 132 leaves 68, which is equivalent to an exponent of 4 in excess-64 notation.

If one or both operand fractions contain leading zeros, the unnormalized operand(s) is prenormalized. That is, the operands are normalized before multiplication begins. Prenormalization increases product precision. By prenormalizing the operands, a maximum of one postnormalization cycle is necessary. Postnormalization is the process of normalizing the product after fraction multiplication. Prenormalization and postnormalization are accomplished by shifting the fraction left one hex digit and subtracting 1 from the characteristic value for each left shift until a significant hex digit appears in the high-order position of the fraction.

The product for both short and long operand multiply instructions is 64 bits long. Note that, if the fraction is not prenormalized, dropping the low-order bits of the product in excess of 64 may result in a false zero product without prenormalization. This result would occur often in long operand instructions because 56 low-order bits of the product are lost when executing the multiply algorithm. Thus, to prevent a false zero, the product for long operand instructions would have to be 120 bits long. A false zero is prevented, however, by prenormalizing the operands and postnormalizing the intermediate product. During postnormalization, the intermediate product characteristic is reduced by the number of left shifts. For long operands, the low-order bits of the intermediate product are dropped before left-shifting. For short operands (six-digit fractions), the product fraction has the full 14 hex digits of the long format, and the two low-order hex fraction digits are accordingly always zeros. The two low-order hex fraction digits are zeros in short operand instructions because a maximum of 12 nonzero hex digits is possible when multiplying two six-digit numbers. In multiplication, the number of digits in the product cannot exceed the sum of the available operand digits. Therefore, because 14 product digits are available, the two low-order hex digits of short operand products are always zeros.

The sign of the product is determined algebraically; that is, if the signs of the operands are alike (both plus or both minus), the product is assigned a plus sign; if the signs are unlike, the product is made negative.

Exponent overflow occurs if the final product characteristic exceeds 127. The operation is terminated, and a program interruption occurs. The overflow interruption condition does not occur for a partial product characteristic exceeding 127 when the final characteristic is brought within range through normalization.

When exponent underflow occurs, the final product characteristic is less than zero. The sign, characteristic, and fraction are made zero, and a program interruption occurs if the corresponding mask bit is a 1. Underflow is not signalled when the characteristic of an operand becomes less than zero during prenormalization, and the correct characteristic and fraction value are used in the multiplication.

When all 14 result fraction digits are zero, the product sign and characteristic are made zero, yielding a true zero result, exponent underflow is not signalled, and no interruption is taken. The program interruption for significance is never taken for multiplication.

Data Flow and Algorithm

- See Note under "Multiply".
- Signs are saved in STAT's C and F.
- Characteristics are added in serial adder; carry is saved in STAT D.
- Fraction multiplication is performed by multiply/divide logic.
- E(12–15) selects two multiplier bits from S.
- S bits determine multiple value to be added to partial product.

The data paths of the floating-point operands during multiplication are shown in Diagram 5-210, FEMDM. The characteristic is computed in the serial adder (A of the diagram). The signs are saved in STAT C and STAT F. To add the characteristics, the first operand characteristic is gated to SAA(1–7) from AB per the ABC, and the second is gated to SAB(1–7) from ST per the STC. The characteristic sum is routed to F and the characteristic carry is saved in STAT D and F(0). 64 is subtracted from the characteristic by adding the 2's complement of 64 to the sum in F.

Note: For an RX instruction with a normalized first operand, the first operand characteristic and sign are in S or T and the second operand characteristic and sign are in A.

Fraction multiplication is performed by the multiply/divide logic (B of Diagram 5-210) and is similar to fixed-point multiplication (Section 2 of this Chapter) except that the operands are shorter. After the signs are saved, the characteristic is determined and the operands are prenormalized; the multiplicand is in DT, the multiplier is in S, and a count, representing the number of repetitive operations necessary to perform the multiplication, is in E(12–15). Multiplication is performed two bits at a time; that is, the multiplicand in DT is multiplied by two bits of the multiplier in S using the multiple-selection decoder and the parallel adder. The count in E(12–15), in addition to keeping track of the number of operations, determines which multiplier bits are to be used, starting with the low-order bits and moving two bits to the left for each multiplication. The result of each two-bit multiplication is a multiple of the multiplicand which is then added to the partial product formed by previous two-bit multiplications. Thus, a new partial product is obtained. The two-bit multiplications continue until E(12–15) indicates that all bits of the multiplier have been used. At that time, the intermediate product is contained in AB(4–67).

The steps of the fraction multiplication are:

1. Place the constant F (hex) into E(12–15).
2. Using the value in E(12–15), select two multiplier bits (M1 and M2) from S. E(12,13) selects the byte and E(14,15) selects the two bits within a byte (see B of Diagram 5-210). For example, with the original value in E(12–15), E(12,13) = 11 selecting the third byte in S (bits 24–31), and E(14,15) = 11 selecting bits 6 and 7 within that byte. Thus, the first M1, M2 values used are S(30,31). Table 3-10 shows which S bits are selected for all values of E(12–15).
3. The value of the M1, M2 bits, in conjunction with the 'TX' trigger, gates the correct multiple of the multiplicand to the PAA. Considering the M1, M2 bits as a two-bit multiplier, the multiplicand in DT can be multiplied by 0 (M1, M2 = 00), by 1 (M1, M2 = 01), by 2 (M1, M2 = 10), or by 3 (M1, M2 = 11), as follows:
   a. M1, M2 = 00 and 'TX' Trigger Is Reset: Because zero times the multiplicand is zero, nothing is added to the partial product. However, the partial product is shifted right two bit positions.

Table 3-10. Multiplier Bits Selected, Floating-Point Multiply

	E(14,15)							
	00		01		10		11	
E(12,13)	M1	M2	M1	M2	M1	M2	M1	M2
00	0	1	2	3	4	5	6	7
01	8	9	10	11	12	13	14	15
10	16	17	18	19	20	21	22	23
11	24	25	26	27	28	29	30	31

   b. M1, M2 = 01 and 'TX' Trigger Is Reset: One times the multiplicand equals the multiplicand. Thus, the multiplicand is added to the partial product, and the partial product is shifted right two bit positions.
   c. M1, M2 = 10 and 'TX' Trigger Is Reset: The multiplicand is multiplied by 2 by shifting it left one bit position. The result is then added to the partial product, and the partial product is shifted right two bit positions.
   d. M1, M2 = 11 and 'TX' Trigger Is Reset: Because the facilities for multiplying the multiplicand by 3 are not directly available, an effective multiplication by 3 is accomplished by multiplying by 4 and subtracting one times the multiplicand (4X - 1X = 3X). The minus 1X occurs first by adding the 2's complement of the multiplicand to the partial product, and 4X occurs by adding 1 to the next two multiplier bits. The 'TX' trigger is set to remember that an additional 1 is required in the next cycle. After adding the 2's complement of the multiplicand to the partial product, the partial product is shifted right two bit positions.
   e. M1, M2 = 00 and 'TX' Trigger Is Set: One times the multiplicand is added to the partial product, which is then shifted right two bit positions.
   f. M1, M2 = 01 and 'TX' Trigger Is Set: Two times (left 1 shift) the multiplicand is added to the partial product, which is then shifted right two bit positions.
   g. M1, M2 = 10 and 'TX' Trigger Is Set: The 2's complement of the multiplicand is added to the partial product, and the 'TX' trigger is again set. The resulting partial product is shifted right two bit positions.
   h. M1, M2 = 11 and 'TX' Trigger is Set: Zero is added to the partial product, and the 'TX' trigger is set. The partial product is shifted right two bit positions.

Note that in each case the new partial product is shifted right two bit positions. Thus, each higher-order multiplicand multiple is displaced two positions to the left when added to the partial product. This right 2 shift is accomplished by shifting the partial product right 4 as it enters PAL, and shifting it left 2 when it is gated to PAB for addition to the next multiplicand multiple. The two low-order bits [B(66,67)] are shifted out, and therefore lost on each cycle.

Note: Steps 2 and 3 are performed by the 'SEL-MPL*E3' micro-order and hardware conditions.

4. Reduce E(12–15) by 1.
5. Determine, by the count in E(12–15), whether all bits of the multiplier fraction have been used to select a multiplicand multiple. If not, repeat steps 2–4; if they have all been used, proceed to step 6.
6. After decoding the last S bits, the 'TX' trigger is checked. If the 'TX' trigger is set, one additional termination cycle is necessary to obtain the final intermediate product. If the 'TX' trigger is reset, no extra cycle is necessary. After the fraction intermediate product is obtained, the fraction is normalized (post-normalization), the characteristic is adjusted, the sign is determined, and the final 64-bit product is stored into the FPR specified by R1 and R1 + 1 [located in E(8–11)].

To illustrate the multiply operation, assume that the following fractions are to be multiplied:

$0.24_{10} \times 0.15_{10} = 0.0360_{10}$ or $0.18_{16} \times .F_{16} = 0.168_{16}$

The operands in machine language become:

0 1000000. 0001 1000 0↔0 X 0 1000000. 1111 0000 0↔0

In hex notation, the example becomes:

+40.18 X +40.F = +40.168

Further, assume that a short operand instruction in the RR format is to be executed. For this discussion, assume that 0.15 (decimal) is the multiplier (first operand) and 0.24 (decimal) is the multiplicand (second operand). At the start of execution, the instruction is in E; the first operand is in A, B, and D (this value is not used and is subsequently destroyed), and the second operand is in S and T.

The signs are saved, the characteristics are determined, and the fractions are prenormalized before beginning the multiply algorithm. The value in E(12–15) is set to 15 and sequentially reduced by 1 during the operation. Before fraction multiplication begins, the first operand fraction is transferred to S, the second operand is transferred to D, and B and T are reset.

The first multiple of the multiplicand is determined by checking E(12–15), which initially contains 1111 (binary). Using Table 3-10 to determine the M1 and M2 bits, the first bits selected are S(30,31). At this time, S(30,31) = 00. Thus, the first partial product placed into AB equals zero. The sequential reduction of E(12–15) continues until the value equals 0101, at which time the partial product in AB equals zero.

Referring to Table 3-10, when E(12–15) = 0101, S(10,11) is selected. These selected bits determine the multiple (M1, M2) of the multiplicand to be added to the partial product in AB. Because S(10,11) = 11 (binary), the 2's complement of DT is gated to PAA. The contents of AB are shifted left 2 at this time (AB equals zero) and gated to PAB. The output of the parallel adder is shifted right 4 to the PAL's. The contents of the PAL's are gated to AB, forming a new partial product. AB(4–67) now contains 1111.1111 1110 1000 0↔0. PA(4) is propagated into PAL(4–7) by the 'R→' micro-order. Because S(10,11) = 11, the 'TX' trigger is set (Table 3-11). Because E(12-15) is decremented after each multiple selection, zeros are added to PAA on the next multiple selection (0 X DT), as shown in Table 3-11. During this select multiple, the contents of AB are shifted left 2 to PAB, and the next partial product is shifted right 4 to the PAL's and AB(4–67), thus yielding an effective right 2 shift. The new partial product in AB(4–67) becomes 1111.1111 1111 1010 0↔0.

Table 3-11. Value of Multiple Determined by Multiple Selection Bits (Floating-Point)

Multiple Selection Bits		TX Trigger	DT Register Times Value Indicated (Add to Partial Product in AB)	Set TX Trigger
M1	M2			
0	0	0	0 X DT	No
0	1	0	1 X DT	No
1	0	0	2 X DT	No
1	1	0	-1 x DT (2's Complement)	Yes
0	0	1	1 X DT	No*
0	1	1	2 X DT	No
1	0	1	-1 x DT (2's Complement)	Yes
1	1	1	0 X DT	Yes

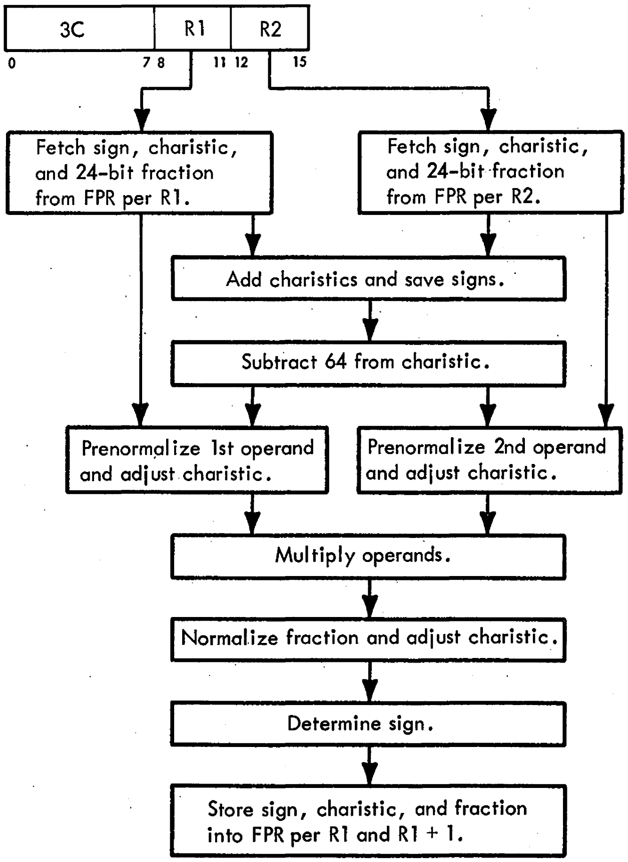
* Used on last multiple selection if 'TX' trigger is set.

At this point, all multiples of the multiplicand have been selected. If the 'TX' trigger is not set, the partial product in AB becomes the intermediate result. In this example, however, the 'TX' trigger was set because the multiplier bits equalled 11 and the 'TX' trigger was previously set (Table 3-11). Therefore, DT must be added to the partial product in AB. The contents of AB are shifted left 2 to PAB and added to DT. No right 4 shift from the parallel adder to the PAL's occurs at this time. The intermediate product is transferred from the PAL's to AB(4–67) and DT. The intermediate product is .0001 0110 1000 0↔0 (0.168_{16}). In this example, normalization is not necessary. The sign, characteristic, and fraction are stored into the FPR specified by R1 and R1 + 1. An end-op cycle completes the operation.

If the integers were preceded by zeros in this example, prenormalization of the operands would occur before executing the multiple algorithm.

Multiply, MER (3C) – RR Short Operands

- Multiply 1st operand (in FPR, per R1) and 2nd operand (in FPR, per R2) and place normalized product into 1st operand location (in FPR, per R1 and R1 + 1).
- See Note under "Multiply".
- RR format:


3C
R1
R2
0
7 8
11 12
15
Fetch sign, charistic, and 24-bit fraction from FPR per R1.
Fetch sign, charistic, and 24-bit fraction from FPR per R2.
Add charistics and save signs.
Subtract 64 from charistic.
Prenormalize 1st operand and adjust charistic.
Prenormalize 2nd operand and adjust charistic.
Multiply operands.
Normalize fraction and adjust charistic.
Determine sign.
Store sign, charistic, and fraction into FPR per R1 and R1 + 1.

- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.

The Multiply, MER, instruction (Diagram 5-211, FEMDM) multiplies the second operand (specified by R1) by the first operand (specified by R2) and places the normalized product into the first operand location (per R1 and R1 + 1).

The conditions at the beginning of the execution phase are:

1. The first operand is in A, B, and D (24-bit fraction only).
2. The second operand is in S and T.
3. The STC contains a value of 4.
4. The instruction is in E.

The second operand fraction (multiplicand) is transferred from T to D. The first operand sign is saved in STAT F and the second is saved in STAT C. The characteristics are added, yielding an excess-128 characteristic, and the sum is placed into F. SA(0) is saved in STAT D and placed into F(0). B and T are reset by transferring zeros from PAL(32–63) to B and T. The first operand (multiplier) is fetched from LS and placed into S to be used for the select multiple function. A constant of 15 is placed into E(12–15) to be used for selecting the two multiple bits from S. The operands are now in position so that multiplying may begin. The ROS microprogram assumes that both operands are normalized. However, the operands are tested, via a four-way branch, to determine whether prenormalization is necessary. The four-way branch tests for the following conditions:

1. First and second operands are normalized.
2. First operand is normalized and the second is unnormalized.
3. First operand is unnormalized and the second is normalized.
4. First and second operands are unnormalized.

Assume that both operands need normalizing. The second operand is normalized by left-shifting the fraction in DT one hex digit and subtracting 1 from the characteristic. Left-shifting continues until the second operand fraction is normalized.

After the fraction of the second operand is normalized, the first operand (multiplier) is transferred from S to B. The contents of T (0's for short operands) are saved in the LSWR. Normalization is accomplished by left-shifting the contents of AB one hex digit and subtracting 1 from the characteristic. (B is reset during the first shift.) Left-shifting continues until the fraction of the first operand is normalized. On each left-shift, the shifted low-order fraction (0's) is stored into the FPR specified by R1

[E(8–11)]. S is then loaded with the short operand multiplier. T is reset, and DT becomes a 56-bit multiplicand (second operand).

Because the characteristic is in excess-128 notation, 64 is subtracted from F so that the excess-64 rule applies. AB is reset, and the multiply algorithm begins. A 'SEL-MPL*E3' micro-order is executed, and 1 is subtracted from E(12–15) for each machine cycle. When E(12–15) = 0100, all 12 pairs of multiples have been selected. Because the 'TX' trigger may have been set on the previous multiple selection, a last multiple selection is necessary to add in the multiplicand to obtain the correct product.

Because the operands were normalized before multiplying, a maximum of one left-shift is necessary to normalize the intermediate product fraction. If A(8–11) = 0, one left shift of the intermediate product fraction is necessary. When the left shift occurs, a 1 is subtracted from the characteristic sum. The characteristic of the final product is located in SAL(1–7) and F(1–7). The sign is determined algebraically, and then the sign, characteristic, and 56-bit fraction are stored into the FPR specified by R1 and R1 + 1.

If SAL(0) = 1, an exponent overflow or exponent underflow condition exists and the product is incorrect. Zeros are stored into the first operand location if an exponent underflow has occurred. A program interruption occurs on all exponent overflows, and on exponent underflows if masked on. If SAL(0) = 0, the stored product is correct. An end-op cycle completes instruction execution.

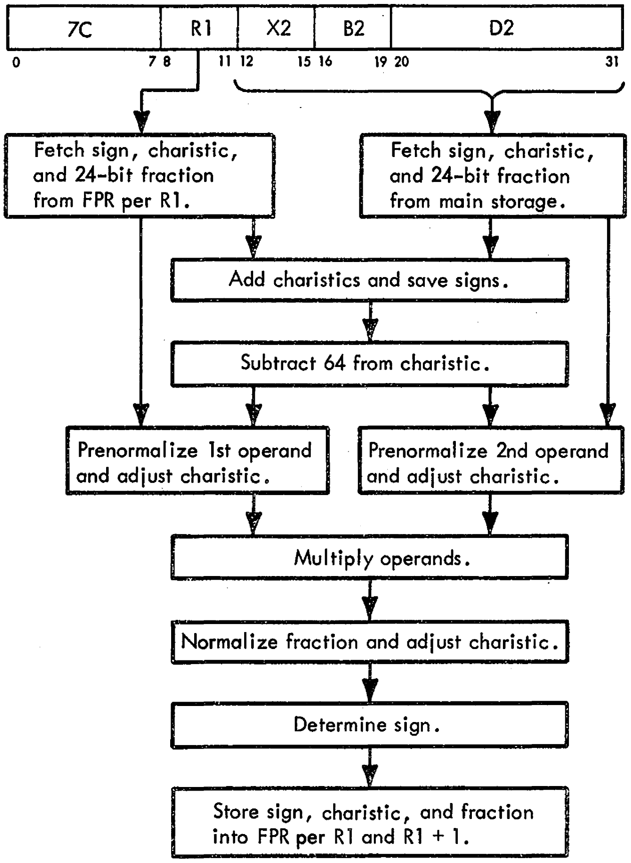
Multiply, ME (7C) – RX Short Operands

- Multiply 1st operand (in FPR, per R1) and 2nd operand (in storage) and place normalized product into 1st operand location (in FPR, per R1 and R1 + 1).
- See Note under "Multiply".
- RX format: (See adjoining column.)
- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.

The Multiply, ME, instruction (Diagram 5-211) multiplies the second operand (specified by the effective address) by the first operand (specified by R1) and places the normalized product into the first operand location (per R1 and R1 + 1)

The conditions at the beginning of the execution phase are:

1. The first operand is in S and T.
2. A main storage request for the second operand has been issued per the effective address in D.
3. The first 16 bits of the instruction are in E.


7C
R1
X2
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch sign, charistic, and 24-bit fraction from FPR per R1.
Fetch sign, charistic, and 24-bit fraction from main storage.
Add charistics and save signs.
Subtract 64 from charistic.
Prenormalize 1st operand and adjust charistic.
Prenormalize 2nd operand and adjust charistic.
Multiply operands.
Normalize fraction and adjust charistic.
Determine sign.
Store sign, charistic, and fraction into FPR per R1 and R1 + 1.

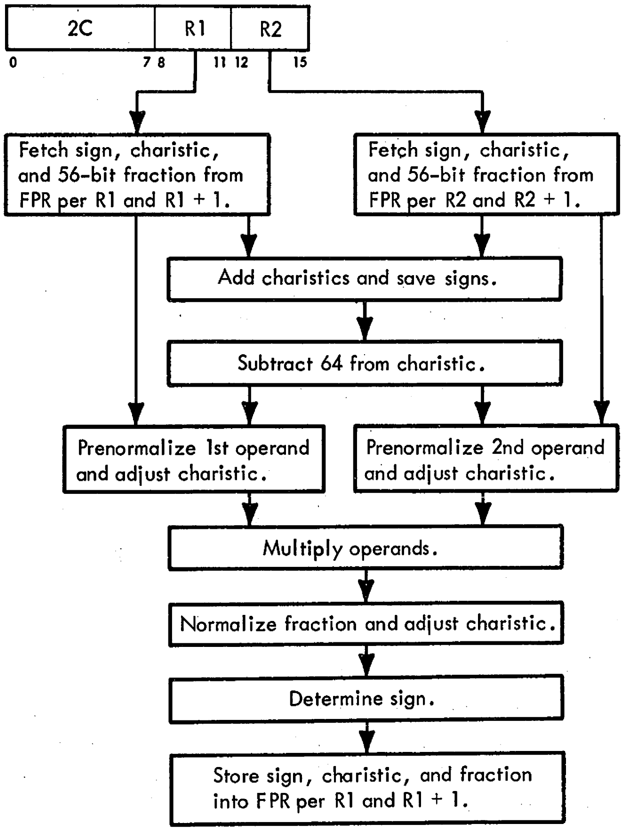
The first operand (multiplier) is placed into A, a constant of 15 is placed into E(12–15), and the STC is set to 4. Assume that the first operand is not normalized and that the second is normalized. The second operand is fetched from main storage (per the effective address in D) and placed into ST. If D(21) = 1, the second operand (multiplicand) is in T; conversely, if D(21) = 0, the second operand is in S and must be placed into T. The sign of the first operand is saved in STAT F and the sign of the second is saved in STAT C. The characteristics are added, and the sum is placed into F. SAL(0) is saved in STAT D and F(0). The fraction of the second operand is placed into D. B and T are reset, the first operand is placed into S, and 15 is placed into E(12–15). A four-way branch determines the next operation. From this point, operation is similar to that of the MER instruction.

If the first operand was normalized, the second operand (multiplicand) from main storage is placed into AB. T and the STC are reset. The transfer of the second operand fraction to D is determined by D(21). If D(21) = 1, the second operand from B is transferred to A and D. If D(21) = 0, the second operand in A is transferred to D. Note that the sign of the first operand is saved in STAT C and that of the second in STAT F. The characteristics are added, and the sum is saved in F. The characteristic carry is saved in STAT D and F(0).

Because the first operand was initially normalized, the ROS microprogram assumes that the second operand is also normalized. Therefore, the first partial product is computed. If the second operand needs normalizing, however, the operands and the constant 15 in E(12–15) are restored, and the second operand is normalized before multiplying. Once both operands are normalized, the operands are multiplied and the results stored.

Multiply, MDR (2C) – RR Long Operands

- Multiply 1st operand (in FPR, per R1 and R1 + 1) and 2nd operand (in FPR, per R2 and R2 + 1) and place normalized product into 1st operand location (in FPR, per R1 and R1 + 1).
- See Note under "Multiply".
- RR format:



2C
R1
R2
0
7 8
11 12
15
Fetch sign, charistic, and 56-bit fraction from FPR per R1 and R1 + 1.
Fetch sign, charistic, and 56-bit fraction from FPR per R2 and R2 + 1.
Add charistics and save signs.
Subtract 64 from charistic.
Prenormalize 1st operand and adjust charistic.
Prenormalize 2nd operand and adjust charistic.
Multiply operands.
Normalize fraction and adjust charistic.
Determine sign.
Store sign, charistic, and fraction into FPR per R1 and R1 + 1.

- Conditions at start of execution:

  32 bits of 1st operand are in A, B, and D (24-bit fraction only).

  32 bits of 2nd operand are in S and T.

  Low-order fractions of 1st and 2nd operands are in LS.

  Instruction is in E.

The Multiply, MDR, instruction (Diagram 5-212, FEMDM) multiplies the second operand (specified by R2 and R2 + 1) by the first operand (specified by R1 and R1 + 1) and places the normalized product into the first operand location (per R1 and R1 + 1).

The conditions at the beginning of the execution phase are:

1. 32 bits of the first operand are in A, B, and D (24-bit fraction only).
2. 32 bits of the second operand are in S and T.
3. The STC contains a value of 4.
4. The instruction is in E.

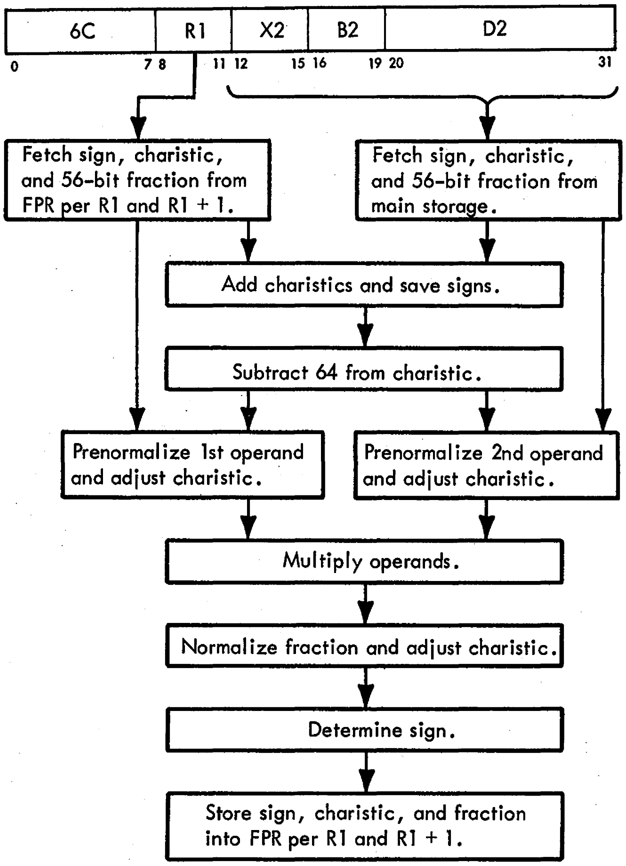
The high-order fraction of the second operand (multiplicand) is transferred from T to D. The low-order fraction of the first operand (multiplier) is placed into S, and that of the second operand is placed into T. DT contains the multiplicand fraction, and S contains the low-order fraction of the multiplier. The signs are saved in STAT C and STAT F. The characteristics are added, and the sum is placed into F. The characteristic carry is saved in STAT D and also placed into F(0). A constant of 15 is placed into E(12–15) to be used for selecting the two multiple bits located in S.

The operands are now in position so that multiplying may begin. The ROS microprogram assumes that both operands are normalized. However, the operands are tested, via a four-way branch, to determine whether prenormalization is necessary. Assume that the first operand is normalized and that the second needs normalizing. The second operand is normalized by left-shifting the contents of DT one hex digit and subtracting 1 from the characteristic on each shift. Left-shifting continues until the fraction is normalized. Because the characteristic sum is in excess-128 notation, 64 is subtracted from the characteristic in F. AB is reset, and the first multiple is selected. The multiples are selected per E(12–15) until E(12–15) = 0001, indicating that the multiples must be selected from the high-order fraction located in LS. Accordingly, the high-order fraction of the first operand (multiplier) is fetched from LS per R1 [E(8–11)], and placed into S. From this point, the multiply execution is the same as that for the MER instruction.

Multiply, MD (6C) – RX Long Operands

- Multiply 1st operand (in FPR, per R1 and R1 + 1) and 2nd operand (in storage) and place normalized product into 1st operand location (in FPR, per R1 and R1 + 1).
- See Note under "Multiply".

- RX format:


6C
R1
X2
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch sign, charistic, and 56-bit fraction from FPR per R1 and R1 + 1.
Fetch sign, charistic, and 56-bit fraction from main storage.
Add charistics and save signs.
Subtract 64 from charistic.
Prenormalize 1st operand and adjust charistic.
Prenormalize 2nd operand and adjust charistic.
Multiply operands.
Normalize fraction and adjust charistic.
Determine sign.
Store sign, charistic, and fraction into FPR per R1 and R1 + 1.

- Conditions at start of execution:
  32 bits of 1st operand are in S and T.
  Low-order fraction of 1st operand is in LS.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.

The Multiply, MD, instruction (Diagram 5-212) multiplies the second operand (in storage) by the first operand (specified by R1 and R1 + 1) and places the normalized product into the first operand location (per R1 and R1 + 1).

The conditions at the beginning of the execution phase are:

1. 32 bits of the first operand are in S and T.
2. The low-order fraction of the first operand is in LS.
3. A main storage request for the second operand has been issued per the effective address in D.
4. The first 16 bits of the instruction are in E.

The first operand (multiplier) sign, characteristic, and high-order fraction are transferred from T to A. The low-order fraction of the first operand is fetched from LS and placed into S. If the first operand is not normalized, the STC is reset, the low-order fraction is transferred from S to B, the second operand (multiplicand) is fetched from main storage and placed into ST and D (high-order fraction in D), the characteristics are added, and the signs are saved (first operand sign in STAT F and second in STAT C).

The low-order fraction of the first operand is again placed into S, and a constant of 15 is placed into E(12–15). The four-way branch determines the next operation. The remainder of the operation is identical to that of the MDR instruction.

If the first operand was normalized, the second operand is fetched from main storage and placed into AB. The second operand fraction (multiplicand) is transferred from AB to DT. The sign of the first operand is saved in STAT C and that of the second in STAT F. The characteristics of the first and second operands are added; the result is placed into F, and the characteristic carry is saved in STAT D. [The carry is also transferred to F(0).] Because the first operand is normalized, the ROS microprogram assumes that the second operand is also normalized; therefore, the first multiple is selected. If the second operand needs to be normalized, the initial conditions are restored and the ROS microprogram proceeds with the normalization of the second operand.

DIVIDE

- Divides 1st operand (dividend) by 2nd operand (divisor) and places normalized quotient into 1st operand location.
- Characteristics are subtracted, and 64 is added to the characteristic difference.
- Operands are prenormalized before dividing.
- Quotient is 32 bits for short operands, 64 bits for long operands.
- Quotient is normalized.
- Sign of quotient is determined algebraically.
- No remainder is retained.

The Divide instruction divides the first operand (dividend) by the second operand (divisor) and places the normalized quotient into the first operand location. In short operand instructions, the low-order halves of the FPR's are ignored and remain unchanged.

A floating-point division consists of a characteristic subtraction and a fraction division. The difference between the dividend and divisor characteristics, plus 64, is used as an intermediate quotient characteristic.

The quotient fraction is normalized by prenormalizing the operands. Postnormalizing the intermediate quotient is never necessary, but a right-shift of one hex digit may be necessary if the normalized dividend fraction is larger than the normalized divisor fraction. The intermediate quotient characteristic is adjusted for the shifts. Low-order digits of the quotient fraction are removed to obtain the desired number of digits.

The sign of the quotient is determined algebraically. That is, if the signs of the operands are alike, a plus sign is assigned to the quotient; if the signs are unlike, a minus sign is set into the quotient.

A program interruption for exponent overflow occurs when the final quotient characteristic exceeds 127; the operation is terminated.

A program interruption for exponent underflow occurs when the final quotient characteristic is less than zero and the corresponding mask bit is a 1. Underflow is not signalled for the intermediate quotient or for the operand characteristics during prenormalization.

If division by a divisor with a zero fraction is attempted, the divide operation is suppressed. The dividend remains unchanged, and a program interruption for floating-point divide occurs. When the dividend fraction is zero, the quotient fraction will be zero. The quotient sign and characteristic are made zero, yielding a true zero result without taking the program interruption for exponent underflow or exponent overflow. The program interruption for significance is never taken for division. The CC remains unchanged.

Characteristic Computation

After the first and second operands are fetched and placed into the proper registers, the characteristics are subtracted. Because the complement gates to the serial adder are on the SAA bus, the first operand characteristic (C1) is subtracted from the second operand characteristic (C2). Therefore, the characteristic computations differ from what might be expected. (Normally, C1 - C2 would be expected.)

The CPU takes the following steps in computing the quotient characteristic:

1. Subtracts C1 from C2 (dividend characteristic from divisor characteristic).
2. Subtracts 64 from the characteristic difference.
3. Normalizes the first operand and adds a 1 to the intermediate characteristic for each digit position that the fraction is shifted.
4. Normalizes the second operand and subtracts a 1 from the intermediate characteristic for each digit position that the fraction is shifted.
5. Takes the 2's complement of the intermediate characteristic.
6. Checks for a divisor fraction greater than a dividend fraction. If the dividend is the larger number, right-shifts the dividend one hex digit and adds 1 to the characteristic.
7. Saves the final characteristic.
8. Checks the final characteristic for exponent overflow or exponent underflow.

As an example of this computation, assume that two hex numbers are to be divided, .004 by .02:

$$\frac{\text{1st operand dividend}}{\text{2nd operand divisor}} \pm \frac{.004 \times 16^5}{.02 \times 16^2} = \pm .2 \times 16^3 = \pm 3.2$$

Fraction

Characteristic

Convert the above characteristics to excess 64 notation:

```
 ┌──────────────── C1
 69.004
 ──────
 66.02
 └──────────────── C2
```

Convert the above characteristics to binary form:

$$\frac{1000101.004}{1000010.02} = \frac{69.004}{66.02} = \pm 67.2 \text{ or } .2 \times 16^3$$

after 64 is subtracted from the characteristic.

Step 1. The machine subtracts the characteristics (C2 - C1):

```
1000010    C2
0111010 }  2's complement of C1
      1 }                                .004
───────                           ──────────
1111101                           1111101.02
```

Step 2. 64 is subtracted from the characteristic to maintain excess-64 notation:

```
1111101
0111111 }  2's complement of 64
      1 }                                .004
───────                           ──────────
0111101                           0111101.02
```

Step 3. Note that the first operand hex fraction requires two left-shifts to prenormalize. Shift left 2 and add 2 to the characteristic:

```
0111101
0000010                                    .4
───────                           ──────────
0111111                           0111111.02
```

Step 4. The second operand hex fraction requires one left-shift. Shift left 1 and subtract 1 from the characteristic:

```
0111111
1111110 }  2's complement of 1
      1 }                                  .4
───────                            ─────────
0111110                            0111110.2
```

Step 5. Take the 2's complement of the characteristic:

```
1000001 }  2's complement of 0111110
      1 }                          1000010.4
───────                            ─────────
1000010                                   .2
```

Step 6. Because the dividend fraction is greater than the divisor fraction, the dividend is shifted right 1 and 1 is added to the characteristic before dividing fractions:

```
1000010
0000001                                          1000011.04
1000011     = 67 = final characteristic                  .2
```

Step 7. Save 67, which is the final characteristic.

Step 8. Divide fractions and store quotient:

```
1000011.04
        .2   = ±  67.2
                  | |
                  | |__________ Result fraction
                  |____________ Result characteristic
```

Subtracting the first operand characteristic from the second effectively makes the characteristic difference part of the divisor (dividend/divisor); to add to the characteristic, therefore, the value must be subtracted. For example, excess-64 notion is used in the CPU. Subtracting (C1 + 64 from C2 + 64) equals C2 - C1 + 0; therefore, 64 must be added to the characteristic difference to maintain excess-64 notation. Because the C2 minus C1 difference is 2's-complemented later in the operation, 64 must be subtracted (2's complement and add) from the characteristic that is part of the divisor. The characteristic must be part of the dividend to obtain the final quotient characteristic.

The intermediate characteristic is 2's-complemented to obtain the correct characteristic of the quotient because the initial characteristic subtraction places the intermediate characteristic in the divisor. The intermediate characteristic is not considered to be in the 2's complement form.

Normalization

In the divide operation, both fractions must be normalized before dividing the fractions. Also, the divisor must be larger than the dividend. If the divisor is less than the dividend, the dividend is divided by 16 by right-shifting the dividend four binary bit positions. Prenormalization and making the divisor larger than the dividend make postnormalization unnecessary.

Fraction Division

- Is performed as follows:
  1. Gate dividend to adder, shifted left 2; gate divisor to adder, shifted left 1.
  2. Add; result is partial dividend.
  3. Develop quotient bit: if partial dividend is in true form, place a 1 into quotient; if partial dividend is in 2's complement form, place a 0 into quotient.
  4. Gate partial dividend and divisor to adder (*no* shift). If partial dividend is in true form, gate divisor in 2's complement form; if partial dividend is in 2's complement form, gate divisor in true form.
  5. Perform steps 2 and 3.
  6. Perform steps 1–5.
  7. Continue until count signals end of algorithm.

The basic algorithm for the floating-point fraction divide operation is similar to the algorithm used in fixed-point divide. The characteristics of the two operands are subtracted, and 64 is added to maintain excess-64 notation. The divisor fraction is subtracted from the dividend fraction. A carry indicates that the dividend is greater than the divisor. The dividend must be less than the divisor; if not, a right 4 shift of the dividend is required. Division is accomplished by successive subtractions and storing of quotient bits as determined by the carry. Successive subtractions are performed, and the divisor is, in effect, shifted right one position with respect to the dividend for each subtraction (actually, the dividend is shifted left as explained below).

In binary arithmetic, to divide one number (dividend) by another (divisor), the dividend is repeatedly reduced by subtracting the divisor. The number of times this reduction can be done is the solution (quotient). There are two methods of performing binary division: restore and nonrestore (Figure 3-11).

In restore division, the result of a reduction of the dividend by the divisor is retained only if the result is the true difference (as opposed to a 2's complement difference); a carry indicates that the result is in true form. This result, called the partial dividend, is used in the next reduction. However, if the result is the 2's complement of the difference (no carry), the result is discarded and the old partial dividend is doubled in relation to the divisor to participate in the next reduction. A 1 is inserted into the quotient when the result is true (carry), and a 0 is inserted when the result is in 2's complement form (no carry).

In the nonrestoring method of division, the result of a reduction is retained as the new partial dividend whether it is in true or 2's complement form. When a partial dividend is in true form, the 2's complement of the divisor is added to it; when the partial dividend is in 2's complement form, the true divisor is added to it. In each reduction, the partial dividend is shifted left one bit in relation to the divisor; also, a 1 is inserted into the quotient when the result is true (carry) and a 0 is inserted into the quotient when the result is in 2's complement form (no carry).

Shifting the dividend left one position doubles its value and is equivalent to halving the divisor. Similarly, shifting the dividend left two positions quadruples it and is equivalent to reducing the divisor to ¼ of its value. Note

```
Problem:  45 ÷ 7 = 6 3/7

Dividend:  0 0 1 0  1 1 0 1 = (2 x 16) + (13 x 1) = 45          n/c = no carry
Divisor:            0 1 1 1 =            ( 7 x 1) =  7            c = carry
Quotient:           0 1 1 0 =            ( 6 x 1) =  6
Remainder:          0 0 1 1 =            ( 3 x 1) =  3

A.  Restore

                                                         0 0 1 1 0
                                             0 1 1 1 |0 0 1 0 1 1 0 1
2's Complement Divisor:                                1 0 0 1                 1st Reduction
2's Complement Result (Discarded):               n/c   1 0 1 1

True Partial Dividend:                                   0 1 0 1
2's Complement Divisor:                                  1 0 0 1               2nd Reduction
2's Complement Result (Discarded):                 n/c   1 1 1 0

True Partial Dividend:                                     1 0 1 1
2's Complement Divisor:                                    1 0 0 1             3rd Reduction
True Result:                                            c  0 1 0 0

True Partial Dividend:                                       1 0 0 0
2's Complement Divisor:                                      1 0 0 1           4th Reduction
True Result:                                              c  0 0 0 1

True Partial Dividend (Remainder):                             0 0 1 1
2's Complement Divisor:                                        1 0 0 1         5th Reduction
2's Complement Result (Discarded):                        n/c  1 1 0 0


B.  Non-Restore

                                                         0 0 1 1 0
                                             0 1 1 1 |0 0 1 0 1 1 0 1
2's Complement Divisor:                                  1 0 0 1               1st Reduction
2's Complement Result:                              n/c  1 0 1 1

2's Complement Partial Dividend:                           0 1 1 1
True Divisor:                                              0 1 1 1             2nd Reduction
2's Complement Result:                                n/c  1 1 1 0

2's Complement Partial Dividend:                             1 1 0 1
True Divisor:                                                0 1 1 1           3rd Reduction
True Result:                                              c  0 1 0 0

True Partial Dividend:                                         1 0 0 0
2's Complement Divisor:                                        1 0 0 1         4th Reduction
True Result:                                                c  0 0 0 1

True Partial Dividend (Remainder):                               0 0 1 1
2's Complement Divisor:                                          1 0 0 1       5th Reduction
2's Complement Result:                                      n/c  1 1 0 0

2's Complement Partial Dividend (Remainder-Divisor):             1 1 0 0
True Divisor:                                                    0 1 1 1       Correction Cycle
True Result (Remainder):                                      c  0 0 1 1
```

Figure 3-11. Restore and Non-Restore Division

the similarity between the restore and nonrestore methods of division (the first successful reduction of the dividend is made by one-quarter of the divisor):

(Restore) Dividend minus ¼ Divisor, is equivalent to (Non-Restore) Dividend minus Divisor + ½ Divisor + ¼ Divisor.

In the 2065, the nonrestoring method of division is used, and the dividend is shifted left rather than shifting the divisor right. The development of the quotient and the left-shifting of the dividend is performed as follows:

1. The dividend is gated to the adder shifted left 2. The divisor is gated to the adder shifted left 1, thus yielding an equivalent right 1 displacement with respect to the dividend. (The divisor may be in true or 2's complement form, depending on the partial dividend.)
2. The two numbers are added, and the result (partial dividend) is placed back into the dividend register.

3. A quotient bit is developed from the partial dividend obtained in step 2. If the partial dividend is in true form, a high-order carry occurred and the high-order bit is a 0; a 1 is therefore placed into the quotient. Conversely, if the partial dividend is in 2's complement form, a high-order carry did *not* occur and the high-order bit is a 1; a 0 is therefore placed into the quotient.
4. The partial dividend and the divisor are gated to the adder. Because the partial dividend was shifted left 2 in step 1, the correct displacement between them exists, and they are not shifted now. If the partial dividend obtained in step 2 is in true form, the divisor is gated in 2's complement form; if the partial dividend is in 2's complement form, the divisor is gated in true form.
5. Steps 2 and 3 are repeated.
6. Steps 1 through 5 are repeated.

The operation continues developing quotient bits, developing new partial dividends, gating the partial dividend to the adder (shifted left 2 every other time), and gating the divisor to the adder in true or 2's complement form (shifted left 1 every other time) until a count, which is reduced every time the partial dividend is shifted, signals the end of the algorithm.

Figure 3-12 is an example of the divide operation as it is performed in the 2065 CPU. Note that the first subtraction does *not* result in a quotient bit but is used to decide whether the divisor is to be true- or 2's complement-added on the next cycle.

Data Flow and Algorithm

- Signs are saved in STAT's C and F.
- Characteristic computation is performed in serial adder.
- Characteristic difference is saved in F.
- 'DVDL0' and 'DVDL1' micro-orders are unique to divide algorithm.

Problem: $.124_{10} \div .4_{10} = .31_{10}$

Dividend = .1FB
Divisor = .666
Quotient = .4F5

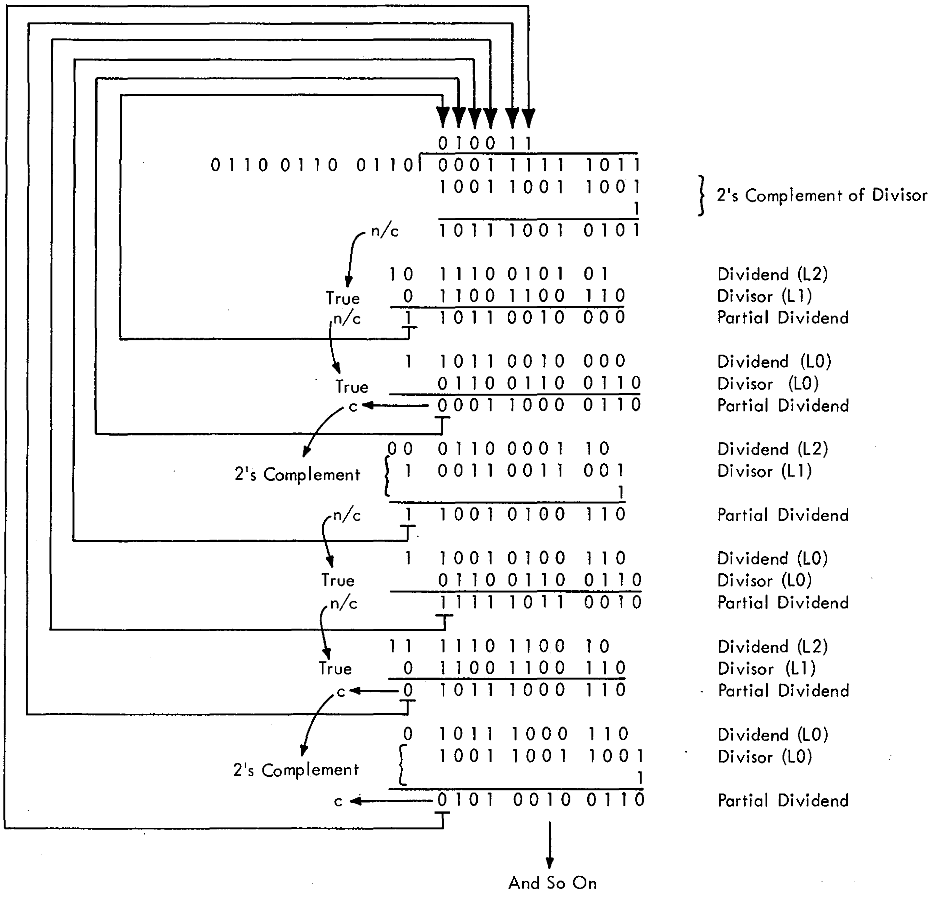

0 1 0 0 1 1
0 1 1 0 0 1 1 0 0 1 1 0 0 0 0 1 1 1 1 1 1 0 1 1
1 0 0 1 1 0 0 1 1 0 0 1
1
2's Complement of Divisor
n/c 1 0 1 1 1 0 0 1 0 1 0 1
1 0 1 1 1 0 0 1 0 1 0 1 Dividend (L2)
True 0 1 1 0 0 1 1 0 0 1 1 0 Divisor (L1)
n/c 1 1 0 1 1 0 0 1 0 0 0 0 Partial Dividend
1 1 0 1 1 0 0 1 0 0 0 0 Dividend (L0)
True 0 1 1 0 0 1 1 0 0 1 1 0 Divisor (L0)
c 0 0 0 1 1 0 0 0 0 1 1 0 Partial Dividend
0 0 0 1 1 0 0 0 0 1 1 0 Dividend (L2)
2's Complement 1 0 0 1 1 0 0 1 1 0 0 1 Divisor (L1)
1
n/c 1 1 0 0 1 0 1 0 0 1 1 0 Partial Dividend
1 1 0 0 1 0 1 0 0 1 1 0 Dividend (L0)
True 0 1 1 0 0 1 1 0 0 1 1 0 Divisor (L0)
n/c 1 1 1 1 1 0 1 1 0 0 1 0 Partial Dividend
1 1 1 1 1 0 1 1 0 0 1 0 Dividend (L2)
True 0 1 1 0 0 1 1 0 0 1 1 0 Divisor (L1)
c 0 1 0 1 1 1 0 0 0 1 1 0 Partial Dividend
0 1 0 1 1 1 0 0 0 1 1 0 Dividend (L0)
2's Complement 1 0 0 1 1 0 0 1 1 0 0 1 Divisor (L0)
1
c 0 1 0 1 0 0 1 0 0 1 1 0 Partial Dividend
And So On

Figure 3-12. Fraction Divide Example

- Fraction division uses parallel adder.
- Divisor is in DT; dividend is in AB; quotient is developed in S.

The first operation that occurs is the computation of the final characteristic (For an example, see "Characteristic Computation"). The data paths for the signs and characteristics are shown in A of Diagram 5-213, FEMDM. The signs are saved in STAT C and STAT F. The first and second operand characteristics are gated to the SAA and SAB per the ABC and STC, respectively. To subtract the characteristics, the 2's complement of the first operand characteristic is added to the second operand characteristic. The characteristic difference is stored into F(0–7), and the characteristic carry [SA(0)] is saved in STAT D. Other inputs to the SAB bus allow subtracting 64, subtracting 1, gating the 2's complement of F, or adding 1 to the value in F. After the final characteristic is computed, the result is stored into S(0–7) per the STC.

The data path for the derivation of the divide multiple is shown in B of Diagram 5-213. When the divide algorithm begins, the divisor (first operand) is in DT and the dividend is in AB.

Two micro-orders ('DVDL0' and 'DVDL1') are used specifically during the divide algorithm. These micro-orders have three functions: (1) to gate the true or 2's complement of DT (divisor) to the PAA; (2) to determine the amount of shift (L0 = no shift, L1 = left 1 shift) of the divisor (contents of DT) to the PAA; and (3) to determine the partial quotient (PQ) bit and the PQ bit location after the subtraction of the divisor and the partial dividend has taken place.

Whether the true or 2's complement form of the divisor is sent to the PAA is determined by the PA(4) carry from the previous algebraic subtraction of the divisor and partial dividend, and the amount of shift (L0 or L1) as determined by the 'DVDL0' or the 'DVDL1' micro-order. If a PA(4) carry occurred, the 2's complement is gated (L0 or L1) to the parallel adder. If a PA(4) carry did not occur, DT is gated (L0 or L1) in true form to the parallel adder. The data in AB is gated to PAB with no shift or a left 2 shift under micro-order control ('AB' and 'ABL2').

As previously noted, the 'DVDL0' and the 'DVDL1' micro-orders determine the PQ bit and the location of the bit. The PQ bit is determined by testing AB(4) for a 0 or a 1. If AB(4) = 1, the partial dividend is in 2's complement form and a 0 is placed into the selected PQ SAL location. If AB(4) = 0, the partial dividend is in true form, and a 1 is placed into the selected PQ SAL location.

As shown in B of Diagram 5-213, the PQ location in SAL is determined by E(14,15) and by the 'DVDL0' or 'DVDL1' micro-order. E(14,15) selects the pair of SAL bits into which the PQ bit is to be placed. The 'DVDL0' micro-order selects the odd bit of the selected pair; the 'DVDL1' micro-order selects the even bit. At the same time that the PQ bit is gated into SAL, the contents of F are added to the PQ bit and saved in F. After a PQ byte (eight bits) is available, the contents of F(0–7) are gated to S per the STC. After S is filled with the quotient (or PQ), the contents of S are stored into LS per E(8–11).

For a discussion of the divide algorithm, assume that the final characteristic is in S(0–7) and that the normalized fractions are in DT (divisor) and AB (dividend). By definition, the CPU requires that floating-point numbers consist of a sign, a characteristic, and a fraction. Because no provisions are made in the CPU to handle integers in floating-point instructions, the divisor must be larger than the dividend to retain a fraction quotient. After both fractions are normalized, therefore, the contents of DT are subtracted from the contents of AB. A carry from PA(4) indicates that the dividend is larger than the divisor. Whenever the dividend is larger than the divisor, the contents of AB must be restored and shifted right 4 (divided by 16) before proceeding with the divide algorithm, and a 1 must be added to the characteristic. If no carry occurred from PA(4), the dividend is less than the divisor, and the CPU proceeds with the divide algorithm.

When the divisor (d) is subtracted from the dividend (D), the difference is placed into AB (D - d in AB). If a right 4 shift was necessary, the divisor (d) is restored and divided by 16 (d in DT). AB now contains the dividend (D). At the beginning of the divide algorithm, the 2's complement of DT is shifted left 1 and added to the contents of AB shifted left 2; the result is placed into AB, thus yielding an effective left 1 shift of AB (dividend). The contents of AB may be expressed by the equation 4D - 2d = contents of AB. The value 4D - 2d is in AB after the first machine cycle of the divide algorithm.

If the dividend was less than the divisor, D - d is in AB. The CPU proceeds to add the contents of DT shifted left 1 to the contents of AB shifted left 2, with the result placed into AB. This addition results in the equation 4(D - d) + 2d = contents of AB. Simplifying the equation yields 4D - 4d + 2d = 4D - 2d. At the end of the first cycle of the divide algorithm, the same result (4D - 2d) is obtained as when the dividend was larger than the divisor. The CPU continues with the divide algorithm.

During the first machine cycle of the divide algorithm, the 'DVDL0' micro-order also selects the DT gating to the parallel adder per the PA(4) carry. The subtraction resulting from the 'DVDL0' micro-order is accomplished during the next machine cycle. The PQ bit, however, is determined by the A(4) value that was computed during the previous machine cycle. On the first cycle of the divide algorithm, the contents of AB (dividend) are shifted left 2 by a micro-order and added to the contents of DT (divisor) shifted left 1 per a micro-order. Thus the divisor is shifted right 1 with respect to the dividend, but the resulting partial dividend is displaced left 2 in AB. On the next divide select multiple subtraction, the dividend and the divisor are subtracted without shifting, yielding the correct right 1 displacement. The following cycle causes AB and DT to shift again. Note that, as the dividend is shifted left, the low-order bit positions of AB are filled with 0's.

As previously noted, the PQ bit is gated to SAL per E(14,15) and the 'DVDL0' or 'DVDL1' micro-order. A 1 is added to the ABC after each pair of PQ bits is gated to F via SAL. When the ABC equals 3, F contains eight PQ bits (one per byte). The PQ byte is gated to S per the STC.

After each byte is gated to S, a 1 is added to the STC. When the STC equals 3, S contains the characteristic and fraction (or high-order fraction). The contents of S are stored into the LSWR.

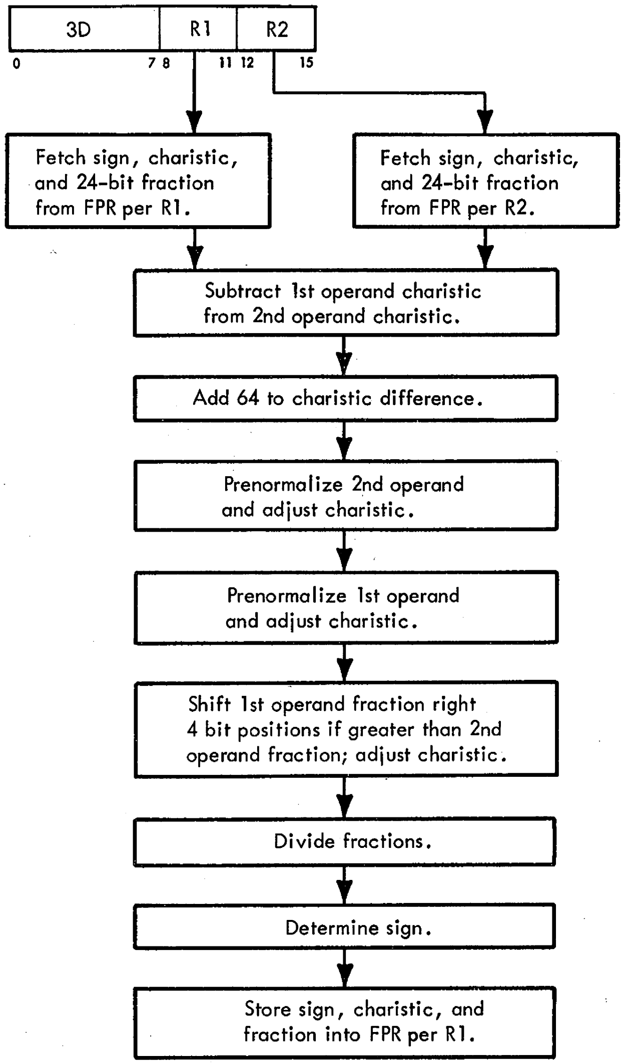
Before initiating the divide algorithm, STAT D was reset to indicate the first pass at loading PQ bytes into S. After the sign, characteristic, and high-order fraction are stored into the LSWR, the instruction and STAT D determine the next operation. If a short operand instruction is being executed, the sign is inserted and stored with the characteristic and fraction into the FPR per E(8–11). An end-op cycle completes instruction execution.

If the instruction was a long operand instruction, the sign, characteristic, and high-order fraction are stored into the FPR per E(8–11). STAT D is set, and the divide algorithm continues. The contents of the LSWR are returned to T. The same operations as described above are performed to obtain the remaining low-order fraction part of the quotient, and the same three-way branch is encountered. This time the divide algorithm is completed, and the low-order fraction is stored into the FPR per E(8–11) + 1. An end-op cycle completes instruction execution. The remainder in AB is not stored.

Divide, DER (3D) – RR Short Operands

- Divide 1st operand (in FPR, per R1) by 2nd operand (in FPR, per R2) and place normalized quotient into 1st operand location.
- RR format: (See adjoining column.)
- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.

The Divide, DER, instruction (Diagram 5-214, FEMDM) divides the first operand (specified by R1) by the second operand (specified by R2) and places the normalized quotient into the first operand location. No remainder is retained.


3D
R1
R2
0
7 8
11 12
15
Fetch sign, charistic, and 24-bit fraction from FPR per R1.
Fetch sign, charistic, and 24-bit fraction from FPR per R2.
Subtract 1st operand charistic from 2nd operand charistic.
Add 64 to charistic difference.
Prenormalize 2nd operand and adjust charistic.
Prenormalize 1st operand and adjust charistic.
Shift 1st operand fraction right 4 bit positions if greater than 2nd operand fraction; adjust charistic.
Divide fractions.
Determine sign.
Store sign, charistic, and fraction into FPR per R1.

The conditions at the beginning of the execution phase are:

1. The first operand is in A, B, and D (24-bit fraction only).
2. The second operand is in S and T.
3. The STC contains a value of 4.
4. The instruction is in E.

If no specification check occurred, the second operand fraction is transferred from T to D. The characteristics are subtracted, and 64 is algebraically added to the characteristic difference to maintain excess 64-notation. The sign of the first operand is saved in STAT F and the sign of the second operand is saved in STAT C. B and T are reset, and the contents of AB and ST are treated as 56-bit fractions.

In the divide instructions, both operands are prenormalized before the divide algorithm begins. A four-way branch determines the prenormalization path by testing A(8–11), the dividend, and PAL(40–43), the divisor, for the normalized conditions:

1. The first and second operands are normalized.

2. The first operand is normalized, and the second is unnormalized.
3. The first operand is unnormalized, and the second is normalized.
4. The first and second operands are unnormalized.

Assume that both operands are unnormalized. The second operand (divisor in DT) is shifted left 4 until the operand is normalized. A 1 is subtracted from the intermediate quotient characteristic for each shift.

After the second operand is normalized, the first operand is normalized by left-shifting until the fraction contains a hex digit [A(8–11) not equal to zero]. On each left-shift, a 1 is added to the intermediate quotient characteristic in F.

After the operands are normalized, the second operand fraction (divisor) is subtracted (take 2's complement of second operand and add) from the first operand fraction. Before branching on the PAL(4) carry, the 2's complement of the intermediate characteristic is computed and placed into F. Also, the constant 5 is placed into E(12–15) for controlling the divide algorithm. A carry from PAL(4) indicates that the dividend is larger than the divisor. If the dividend is larger than the divisor, the dividend is restored and is divided by 16 by a right-shift of one hex digit. A 1 is added to the characteristic value, which is the final characteristic of the quotient. The final characteristic is placed into S(0–7).

No carry from PAL(4) indicates that the dividend is less than the divisor, at which time the first machine cycle of the divide algorithm is executed. A test is made to determine an overflow or underflow condition. Assume that no overflow or underflow condition exists.

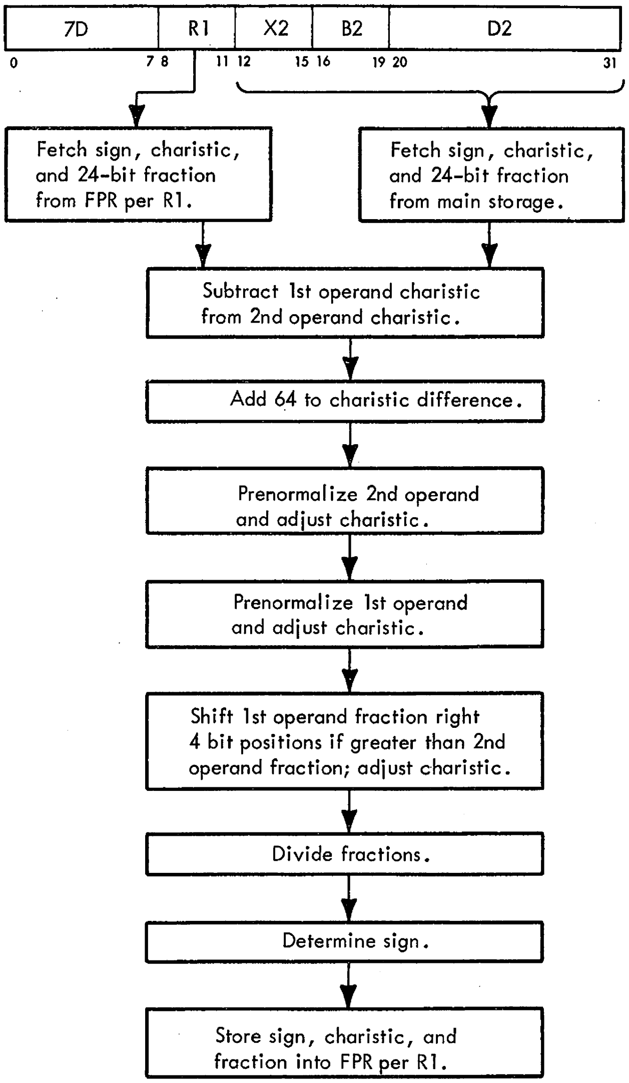
Fraction division begins as shown in Sheet 4 of Diagram 5-214. Figure 3-13 is an example of the action that occurs in parallel adder bits 4–11 (bits 12–31 being considered to equal 0's).

During the normalization routine, tests for zero fractions are made. If the second operand fraction (divisor) equals zero, the divide operation is suppressed and a floating-point divide program interruption occurs. If the first operand fraction (dividend) equals zero, a true zero quotient results. A true zero is stored into the first operand location, and an end-op cycle completes instruction execution.

Divide, DE (7D) – RX Short Operands

- Divide 1st operand (in FPR, per R1) by 2nd operand (in storage) and place normalized quotient into 1st operand location.

- RX format:

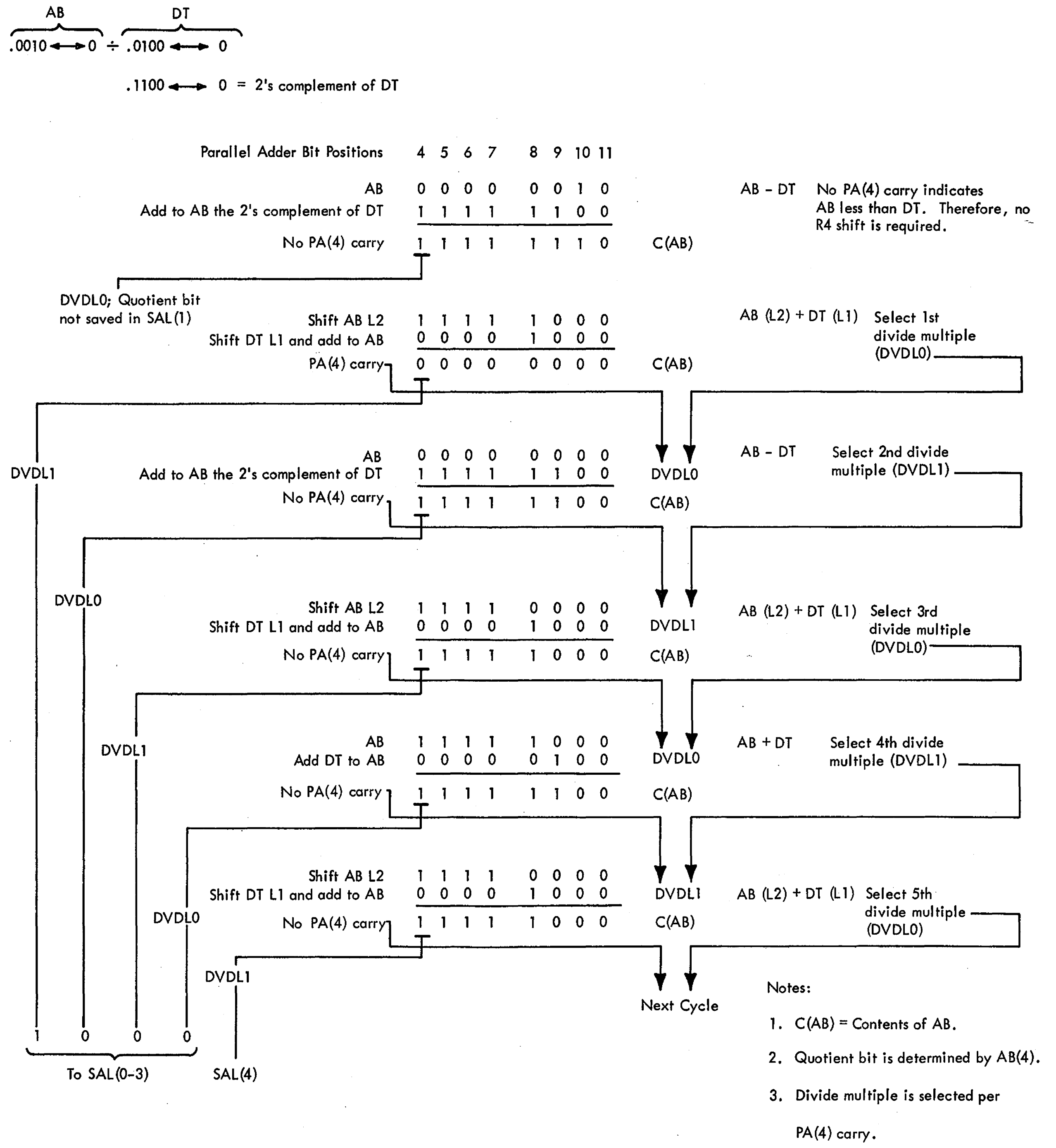

7D
R1
X2
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch sign, charistic, and 24-bit fraction from FPR per R1.
Fetch sign, charistic, and 24-bit fraction from main storage.
Subtract 1st operand charistic from 2nd operand charistic.
Add 64 to charistic difference.
Prenormalize 2nd operand and adjust charistic.
Prenormalize 1st operand and adjust charistic.
Shift 1st operand fraction right 4 bit positions if greater than 2nd operand fraction; adjust charistic.
Divide fractions.
Determine sign.
Store sign, charistic, and fraction into FPR per R1.

- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.

The Divide, DE, instruction (Diagram 5-214) divides the first operand (specified by R1) by the second operand (from main storage) and places the normalized quotient into the first operand location. No remainder is retained.

The conditions at the beginning of the execution phase are:

1. The first operand is in S and T.


AB
DT
.0010 ◄—► 0 ÷ .0100 ◄—► 0
.1100 ◄—► 0 = 2's complement of DT
Parallel Adder Bit Positions 4 5 6 7 8 9 10 11
AB 0 0 0 0 0 0 1 0
Add to AB the 2's complement of DT 1 1 1 1 1 1 0 0
No PA(4) carry 1 1 1 1 1 1 1 0
C(AB)
AB - DT
No PA(4) carry indicates AB less than DT. Therefore, no R4 shift is required.
DVDL0; Quotient bit not saved in SAL(1)
Shift AB L2 1 1 1 1 1 0 0 0
Shift DT L1 and add to AB 0 0 0 0 1 0 0 0
PA(4) carry 0 0 0 0 0 0 0 0
C(AB)
AB (L2) + DT (L1)
Select 1st divide multiple (DVDL0)
DVDL1
AB 0 0 0 0 0 0 0 0
Add to AB the 2's complement of DT 1 1 1 1 1 1 0 0
No PA(4) carry 1 1 1 1 1 1 0 0
DVDL0
C(AB)
AB - DT
Select 2nd divide multiple (DVDL1)
DVDL0
Shift AB L2 1 1 1 1 0 0 0 0
Shift DT L1 and add to AB 0 0 0 0 1 0 0 0
No PA(4) carry 1 1 1 1 1 0 0 0
DVDL1
C(AB)
AB (L2) + DT (L1)
Select 3rd divide multiple (DVDL0)
DVDL1
AB 1 1 1 1 1 0 0 0
Add DT to AB 0 0 0 0 0 1 0 0
No PA(4) carry 1 1 1 1 1 1 0 0
DVDL0
C(AB)
AB + DT
Select 4th divide multiple (DVDL1)
Shift AB L2 1 1 1 1 0 0 0 0
Shift DT L1 and add to AB 0 0 0 0 1 0 0 0
No PA(4) carry 1 1 1 1 1 0 0 0
DVDL0
DVDL1
C(AB)
AB (L2) + DT (L1)
Select 5th divide multiple (DVDL0)
DVDL1
Next Cycle
1 0 0 0
To SAL(0-3)
SAL(4)

Notes:

1. C(AB) = Contents of AB.
2. Quotient bit is determined by AB(4).
3. Divide multiple is selected per PA(4) carry.

Figure 3-13. Floating-Point Divide Example

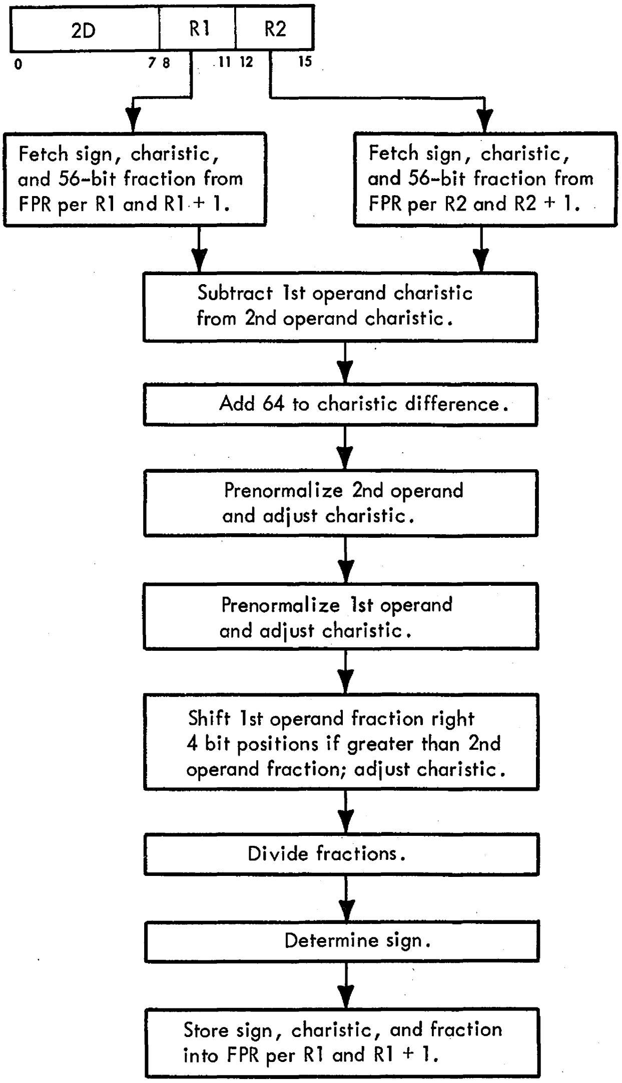
2. A main storage request for the second operand has been issued per the effective address in D.
3. The first 16 bits of the instruction are in E.

The first operand is transferred from T to A, and the STC is set to 4. The second operand is fetched from main storage per D. D(21) determines which 32 bits of the 64-bit doubleword are gated to T (Sheet 2 of Diagram 5-214). From this point, instruction execution is the same as that of the DER instruction.

Divide, DDR (2D) — RR Long Operands

- Divide 1st operand (in FPR, per R1 and R1 + 1) by 2nd operand (in FPR, per R2 and R2 + 1) and place normalized quotient into 1st operand location.

- RR format:



2D
R1
R2
0
7 8
11 12
15
Fetch sign, charistic, and 56-bit fraction from FPR per R1 and R1 + 1.
Fetch sign, charistic, and 56-bit fraction from FPR per R2 and R2 + 1.
Subtract 1st operand charistic from 2nd operand charistic.
Add 64 to charistic difference.
Prenormalize 2nd operand and adjust charistic.
Prenormalize 1st operand and adjust charistic.
Shift 1st operand fraction right 4 bit positions if greater than 2nd operand fraction; adjust charistic.
Divide fractions.
Determine sign.
Store sign, charistic, and fraction into FPR per R1 and R1 + 1.

- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Low-order fractions of 1st and 2nd operands are in LS.
  Instruction is in E.

The Divide, DDR, instruction (Diagram 5-215, FEMDM) divides the first operand (specified by R1 and R1 + 1) by the second operand (specified by R2 and R2 + 1) and places the normalized quotient into the first operand location. No remainder is retained.

The conditions at the beginning of the execution phase are:

1. 32 bits of the first operand are in A, B, and D (24-bit fraction only).
2. 32 bits of the second operand are in S and T.
3. The STC contains a value of 4.
4. The instruction is in E.

If no specification check occurred, the high-order fraction of the divisor is gated from T to D. The low-order fractions of the first and second operands are placed into B and T, respectively. As a result, the dividend fraction is in AB and the divisor fraction is in DT.

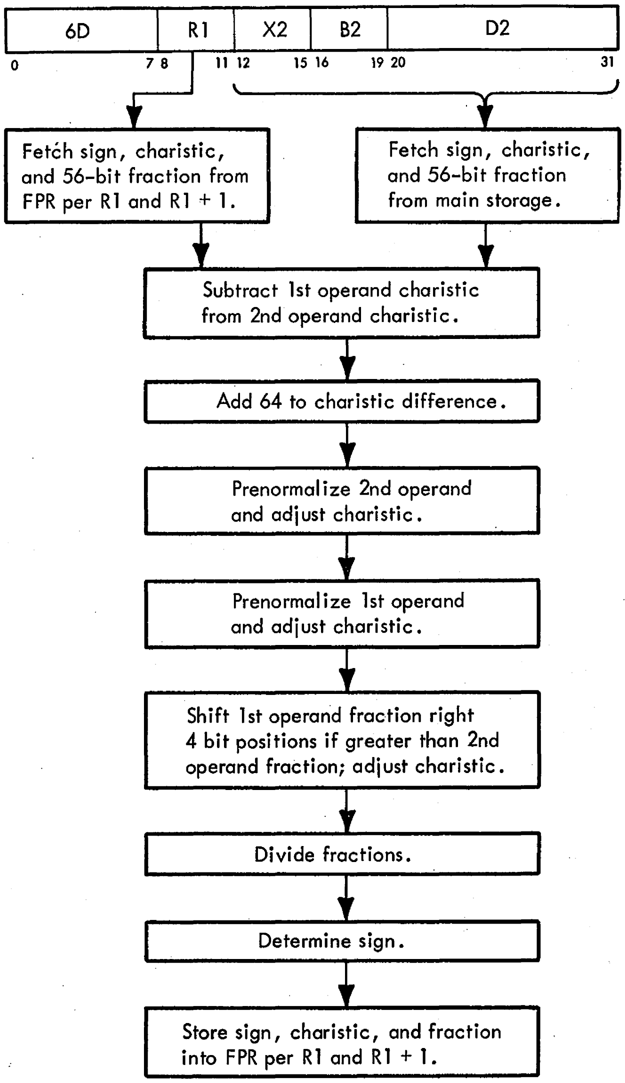
The signs are saved in STAT C and STAT F. The characteristics are subtracted, and excess-64 notation is maintained. The ABC is reset.

The four-way branch (Sheet 3 of Diagram 5-215) determines the next operation. The normalization routine, divide algorithm, and end ops are explained in the DER instruction discussion.

Divide, DD (6D) – RX Long Operands

- Divide 1st operand (in FPR, per R1 and R1 + 1) by 2nd operand (in storage) and place normalized quotient into 1st operand location.

- RX format:



6D
R1
X2
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch sign, charistic, and 56-bit fraction from FPR per R1 and R1 + 1.
Fetch sign, charistic, and 56-bit fraction from main storage.
Subtract 1st operand charistic from 2nd operand charistic.
Add 64 to charistic difference.
Prenormalize 2nd operand and adjust charistic.
Prenormalize 1st operand and adjust charistic.
Shift 1st operand fraction right 4 bit positions if greater than 2nd operand fraction; adjust charistic.
Divide fractions.
Determine sign.
Store sign, charistic, and fraction into FPR per R1 and R1 + 1.

- Conditions at start of execution:
  32 bits of 1st operand are in S and T.
  Low-order fraction of 1st operand is in LS.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.

The Divide, DD, instruction (Diagram 5-215) divides the first operand (specified by R1 and R1 + 1) by the second operand (from main storage) and places the normalized quotient into the first operand location. No remainder is retained.

The conditions at the beginning of the execution phase are:

1. 32 bits of the first operand are in S and T.
2. The low-order fraction of the first operand is in LS.
3. A main storage request for the second operand has been issued per the effective address in D.
4. The first 16 bits of the instruction are in E.

The sign, characteristic, and high-order fraction of the first operand are transferred from T to A. The low-order fraction of the first operand is fetched from LS and placed into B via T and the parallel adder. The second operand (64 bits) is fetched from main storage and placed into ST. The high-order fraction is transferred from S to D, and the contents of DT become the 56-bit fraction divisor. The signs are saved in STAT C and STAT F. The characteristics are subtracted, and 64 is added to the characteristic difference to maintain excess-64 notation.

The first operand (dividend) is in AB, and the second operand (divisor) is in DT. The next step is to check for the prenormalization of the dividend and divisor fractions (Sheet 3 of Diagram 5-215). The remainder of the operation is identical to that of the DDR instruction.

STORE

The Store instructions (STE and STD) store the first operand from an FPR in LS into the second operand location in main storage. The Store instructions are in the RX format with short and long operand options available. In the STE instruction, the low-order half of the first operand register is ignored. The first operand location remains unchanged.

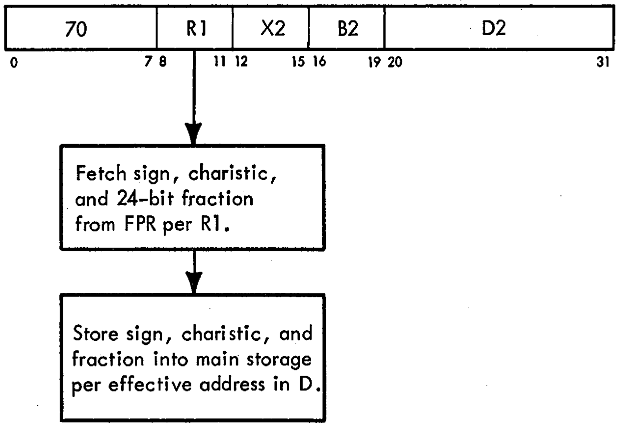
Storing must be on word boundaries for the STE instruction and on doubleword boundaries for the STD instruction.

For all Store instructions, an address store compare test is made because the instructions that are in Q may be modified in main storage by the Store instruction. If an instruction is modified in main storage and is not corrected in Q, the program may not be properly executed; therefore, Q must be reloaded. The address store compare test compares the main storage address, where data is to be stored, with the effective address indicated by the Store instruction. The comparison is made by subtracting the contents of D (effective address) from the contents of the IC, shifting the difference right 4, and testing for a zero result. If the difference equals zero, the difference is less than 16; therefore, the 'program store compare' trigger is set to indicate that the instructions in Q must be refetched. The address store compare test is discussed in Section 1 of this Chapter.

Store, STE (70) – RX Short Operands

- Store 1st operand (in FPR, per R1) into 2nd operand location (in storage).

- RX format:

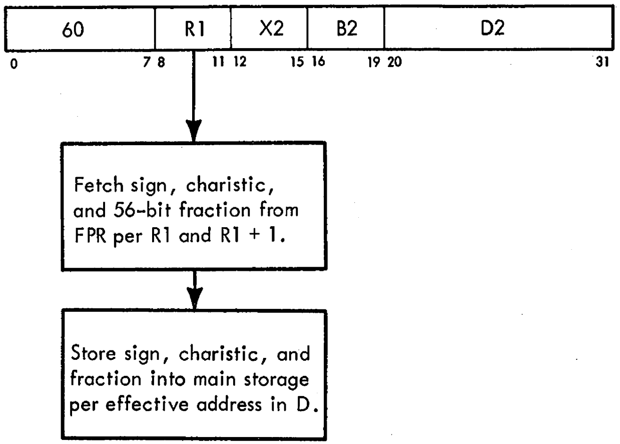

70
R1
X2
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch sign, charistic, and 24-bit fraction from FPR per R1.
Store sign, charistic, and fraction into main storage per effective address in D.

- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request has been issued per effective address in D.
  First 16 bits of instruction are in E.

The Store, STE, instruction (Diagram 5-216, FEMDM) stores the first operand from the FPR specified by R1 into the main storage location specified by the effective address (second operand location). During the RX I-Fetch, the effective address is computed and placed into D, and a main storage request is initiated. The effective address must be on a word boundary or a specification program interruption is taken. Bit 21 of the effective address [PAL(61)] is tested to determine which mark triggers are set. If PAL(61) = 1, mark triggers 4–7 are set; if PAL(61) = 0, Mark triggers 0–3 are set. An address store compare test is then performed, and the instruction is terminated by an end-op cycle.

Store, STD (60) – RX Long Operands

- Store 1st operand (in FPR, per R1 and R1 + 1) into 2nd operand location (in storage).
- RX format:


60
R1
X2
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch sign, charistic, and 56-bit fraction from FPR per R1 and R1 + 1.
Store sign, charistic, and fraction into main storage per effective address in D.

- Conditions at start of execution:
  32 bits of 1st operand are in S and T.
  Low-order fraction of 1st operand is in LS.
  Main storage request has been issued per effective address in D.
  First 16 bits of instruction are in E.

The Store, STD, instruction (Diagram 5-216) stores the first operand from the even/odd pair of FPR's specified by R1 and R1 + 1 into the main storage location specified by the effective address (second operand location). During the RX I-Fetch, the effective address is computed and placed into D and a main storage request is initiated. The effective address must be on a doubleword boundary or a specification program interruption is taken. The low-order half of the first operand is gated to T from the FPR specified by R1 + 1, and mark triggers 0–7 are set. After an address store compare test is performed, the 64-bit operand is stored per the effective address, and the instruction is terminated by an end-op cycle.

Section 4. Decimal Instructions

This section discusses the nine instructions that operate on decimal data. The instructions use the SS format and assume packed operands and results except for Pack, which has a zoned operand, and Unpack, which has a zoned result. For a discussion of the number representation, data formats, excess-6 arithmetic, operand addressing, instruction formats, data flow, program interruptions, and condition codes, refer to Chapter 1.

INSTRUCTION HANDLING

- Depending on instruction, processing of 1 or 2 operands may be specified.
- Pack, Unpack, Move with Offset, and Zero and Add instructions operate on 1 operand.
- Add, Subtract, Compare, Multiply, and Divide instructions operate on 2 operands.
- All add-type instructions set CC.
- Major serial adder functions used by decimal instructions are:
  Excess-6 translation.
  Decimal correction.
  Complement gating.
  Cross-gating.
  Zone or sign insertion.
  Invalid digit and sign detection.
  Zero detection.

Decimal instructions may be classified into the general categories of 1- and 2-operand instructions. The 1-operand instructions are Pack, Unpack, Move with Offset, and Zero and Add. The 2-operand instructions are Add, Subtract, Compare, Multiply, and Divide.

In the 1-operand instructions, the first operand is not processed but its address is used as the destination address; the second operand is processed, and the results are placed into the first operand location. The 1-operand instructions are handled by fetching the second operand to AB. Successive AB bytes are selected per the ABC and are processed in the serial adder, and the resultant bytes are entered into ST per the STC. After all second operand bytes have been processed, the contents of ST are stored into main storage at the first operand address.

The 2-operand Add, Subtract, and Compare instructions are executed by fetching the first operand to ST and the second operand to AB. A true add or complement add operation is then performed in the serial adder one byte at a time, with the resultant bytes replacing the first operand bytes in ST as they are processed. For the Add and Subtract instructions, the results are stored into main storage at the first operand address. The Compare instruction does not store the result, but performs a test to determine the high, low, or equal relationship of the first operand to the second operand and sets the CC accordingly.

For the 2-operand Multiply and Divide instructions, the operands must be properly aligned in the registers prior to entering execution. This function is performed by the appropriate right- and left-adjust sequences incorporated in the individual microprogram of the instruction.

Basically, the multiply operation is performed by repetitive addition. The product bytes are developed one byte at a time, starting with the low-order byte. Each time one byte of product is developed, it is stored into main storage under control of the corresponding mark trigger. The instruction then proceeds to develop the next higher-order product byte. Upon execution of the instruction, the first operand is completely replaced by the product.

The Divide instruction is performed by repetitive subtraction. It is the only decimal instruction that processes the operands starting with the high-order bytes. The full divisor and a sufficient number of high-order dividend bytes are fetched to perform the first successful subtraction. Then by repeatedly subtracting the divisor from the dividend and counting the number of successful subtractions, the high-order quotient byte is developed. This byte is stored into main storage, and the instruction proceeds to develop the next lower-order quotient byte. Upon execution of the instruction, the first operand is completely replaced by the quotient and the remainder. The remainder occupies the low-order portion of the destination field.

The results of the Add, Subtract, and Compare instructions are used to set the CC. All other decimal instructions leave the code unchanged. The Add and Subtract instructions set the CC to 0, 1, or 2 to indicate a zero, less-than-zero, or greater-than-zero result; the CC is set to 3 if the result of the operation overflows. The Compare instruction sets the CC to 0, 1, or 2 to indicate that the first operand was equal to, less than, or greater than the second operand.

The serial adder performs many functions on its input data. The functions of excess-6 translation, decimal correction, and complement gating are discussed in "Data

Handling" in Chapter 1, Section 4. Additional serial adder functions used by the decimal operations are:

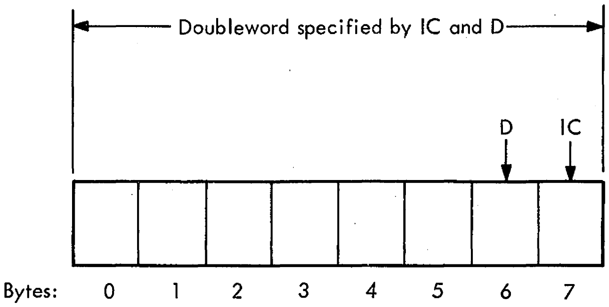
1. Cross-gating. The two-digit input to the A-side of the adder is swapped upon gating to SAL; the digit at adder A-side (0–3) is gated to SAL(4–7), and A-side (4–7) is gated to SAL(0–3). This function is used mainly by the Pack and Unpack instructions to interchange the sign and digit positions.
2. Zone or sign insertion. The correct zone or sign code (USASCII-8 or EBCDIC) is applied from the ROSDR to the adder A-side. The zone or sign may be merged with the digit in any combination.
3. Invalid digit and sign detection. The inputs to the A and B sides of the adder are tested for invalid digits or signs. An appropriate interrupt trigger is set upon detection of an invalid code.
4. Zero detection. This function is used to sense overflow conditions and also to detect all-zero results. An all-zero result placed into main storage must carry a positive sign. Consequently, arithmetic instructions such as Add and Subtract specify testing of each SAL byte for zeros. If upon execution of the instruction it is found that an all-zero result has been stored, the instruction forces storing of a plus sign at the low-order byte address.

WORD OVERLAP CONDITION

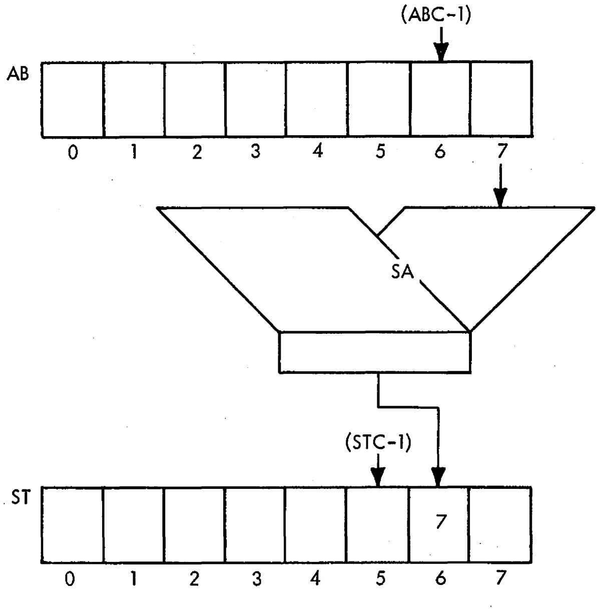
- Word overlap condition exists when: IC(0–20) = D(0–20) and IC(21–23) > D(21–23).
- Test for word overlap is performed by GIS of all one-operand instructions.
- Execution of one-operand instructions provides separate microprogram to handle word overlap conditions.
- No special action is taken to detect word overlap in two-operand instructions.
- Word overlap in two-operand instructions causes data program interruption.

Data is fetched from and placed into main storage one doubleword at a time. However, program compatibility of the 2065 CPU with smaller models in the System/360 requires that all results placed into main storage must be considered to be stored one byte at a time as they are processed. There are some cases where this compatibility would not be maintained unless special actions were taken. The condition that requires special handling is called "word overlap" and occurs when the fields of the first and second operands specified by the instruction overlap.

The operand addresses and field lengths may be such that one or more bytes in main storage are specified as part of both the first and the second operands. For example, consider the case in which the IC and D specify the same doubleword in storage; the IC specifies byte 7 as the starting second operand byte to be processed in this doubleword, and D specifies byte 6 as the starting first operand address. At least two operand bytes are to be processed.



This doubleword is fetched from main storage and placed into AB and ST. A one-operand instruction, such as Move with Offset, will process the first AB byte (byte 7) in the serial adder and place the result into the designated first operand byte; i.e., byte 6 in ST. Then, ABC and STC are reduced one count to designate the next AB byte to be processed and the ST location into which the results must be placed.

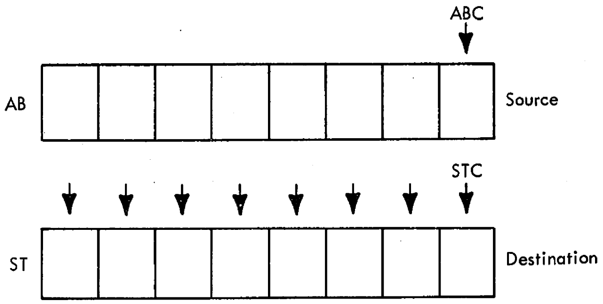


Note, however, that the preceding move operation has replaced the original contents of byte 6 in ST with the contents of byte 7. Thus, the next AB byte to be processed (original byte 6) is no longer valid and must be updated; i.e., the equivalent of storing ST byte 6 and then refetching this byte to AB must be performed.

As seen from the preceding example, the word-overlap condition may require special handling of data. Execution of all one-operand decimal instructions (Pack, Unpack, Move with Offset, and Zero and Add) provides for two alternate microprograms. One microprogram is for the normal, or not-word-overlap, case; the other handles the word-overlap condition. Selection of the appropriate microprogram is dependent on the outcome of the word-overlap test, which is performed in the General Initialization Sequence (GIS) of all one-operand instructions.

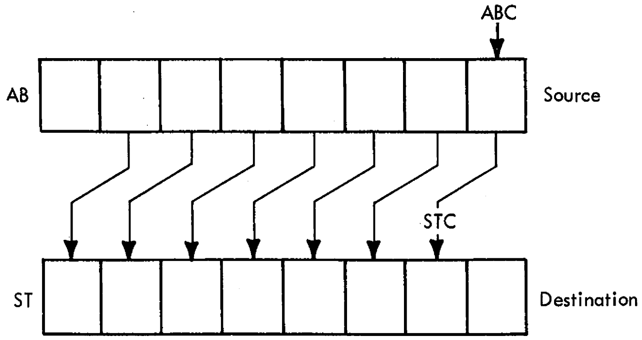
A word-overlap condition exists when both operands have the same doubleword address. The manner in which the first and second operand bytes are specified within this doubleword determines whether special data handling is required. When the word-overlap condition exists, three cases of byte specification may be distinguished:

1. The first and second operand bytes are the same; no special data handling is required.†



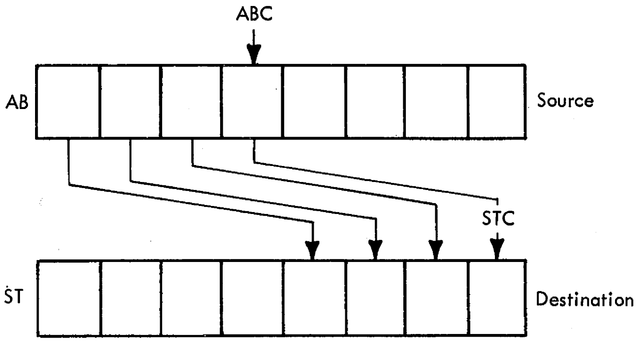
In this case, the destination bytes are placed into the same locations from which the source bytes are obtained. Because processing of any source byte does not affect the contents of the next source byte, no updating of source bytes is necessary.

2. The first operand bytes are specified "ahead" of the second operand bytes; special data handling is required.



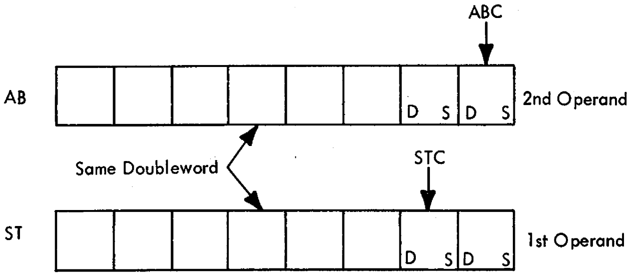
In this case, the destination bytes are placed into the locations from which the next source bytes will be processed. The data in AB becomes "obsolete" after processing of one or more source bytes. (The crossover point at which data becomes obsolete depends on the amount of skew between ABC and STC.) This word overlap case requires special data-handling techniques.

3. The first operand bytes are specified "behind" the second operand bytes; no special data handling is required.



In this case, the destination bytes are placed behind the source bytes as they are processed. Thus processing of any source byte cannot affect the contents of the next source byte, and no updating is necessary.

In two-operand arithmetic instructions, no special action is taken to detect word overlap. Word overlap is ignored during execution of a Compare instruction, because this instruction does not store the results. The operand fields specified for Add, Subtract, Multiply, and Divide instructions either should not overlap at all or should have coincident rightmost bytes. The GIS for these instructions does not perform the word overlap test, because improper overlap of the operands causes an invalid data condition to be detected in the execution phase. In two-operand instructions, the operand fields are correctly specified when the rightmost byte of each operand contains the operand sign; all bytes to the left of the sign byte must contain only digits. This requirement cannot be fulfilled when both operands in the instruction specify the same doubleword with different byte addresses. The following example shows that the sign byte of the first operand is also the "digit" byte of the second operand.



During execution of the instruction, all operand digits are checked for validity. Detection of a sign code in the digit position forces a data program interruption.

† Except for Unpack instruction. This instruction generates two bytes of destination for each byte of source and requires special data handling in all cases of word overlap.

GENERAL INITIALIZATION SEQUENCE

At the completion of the SS I-Fetch, the CPU is in the following status:

1. A main storage request for the doubleword containing the low-order byte of the first operand has been issued per D.
2. D contains the low-order byte address (contents of GPR addressed by B1, + D1, + L1) of the first operand.
3. The IC contains the high-order byte address (contents of GPR addressed by B2, + D2) of the second operand.
4. The next instruction address has been transferred from the IC to the LSWR.
5. E(0–7) contains the instruction op code.
6. The 'PSC' trigger has been set, if appropriate.

Following the SS I-Fetch, a branch is made per the instruction op code to the appropriate General Initialization Sequence (GIS). The general functions of the GIS for decimal instructions are described below. Functions peculiar to a specific instruction are covered in subsequent paragraphs dealing with the execution of that instruction.

The function of the GIS microprogram is to set up initial conditions for the execution phase. These include:

1. Gating the first operand from the SDBO to ST. A D request for the doubleword containing the low-order byte of the first operand was issued during SS I-Fetch.
2. Adding of L2 field to IC. At the end of SS I-Fetch, the IC contains the address of the high-order byte of the second operand. To address the low-order byte of the second operand, the IC must be incremented by the L2 field during GIS.
3. Initiating a storage request per the IC for the second operand. GIS initiates a storage request for the doubleword containing the low-order byte of the second operand.
4. Setting of STC and ABC. The STC is set to the rightmost first operand byte in ST, the byte to be processed first. Because the address of the rightmost byte is specified by D(21–23), the STC is set per these bits. Similarly, the rightmost second operand byte is selected in AB by gating IC(21–23) to the ABC.
5. Gating the second operand from the SDBO to AB. An IC request for the second operand is issued during GIS. Subsequently, the GIS gates the second operand from the SDBO to AB.
6. Performing a sign handling function. For add-type instructions, the sign of the result is tentatively set to agree with the sign of the first operand before the execution phase. The GIS examines the signs in the rightmost bytes of both operands and establishes whether to perform a true add or a complement add operation. For multiply and divide operations, the sign of the result is determined during the execution phase by examining the appropriate STAT's, which have been previously set according to the signs of the two operands. Both operands are tested for an invalid sign.
7. Performing the word overlap test. A word overlap test is performed during the GIS for Pack, Unpack, Move with Offset, and Zero and Add instructions.

ADD, SUBTRACT, AND COMPARE

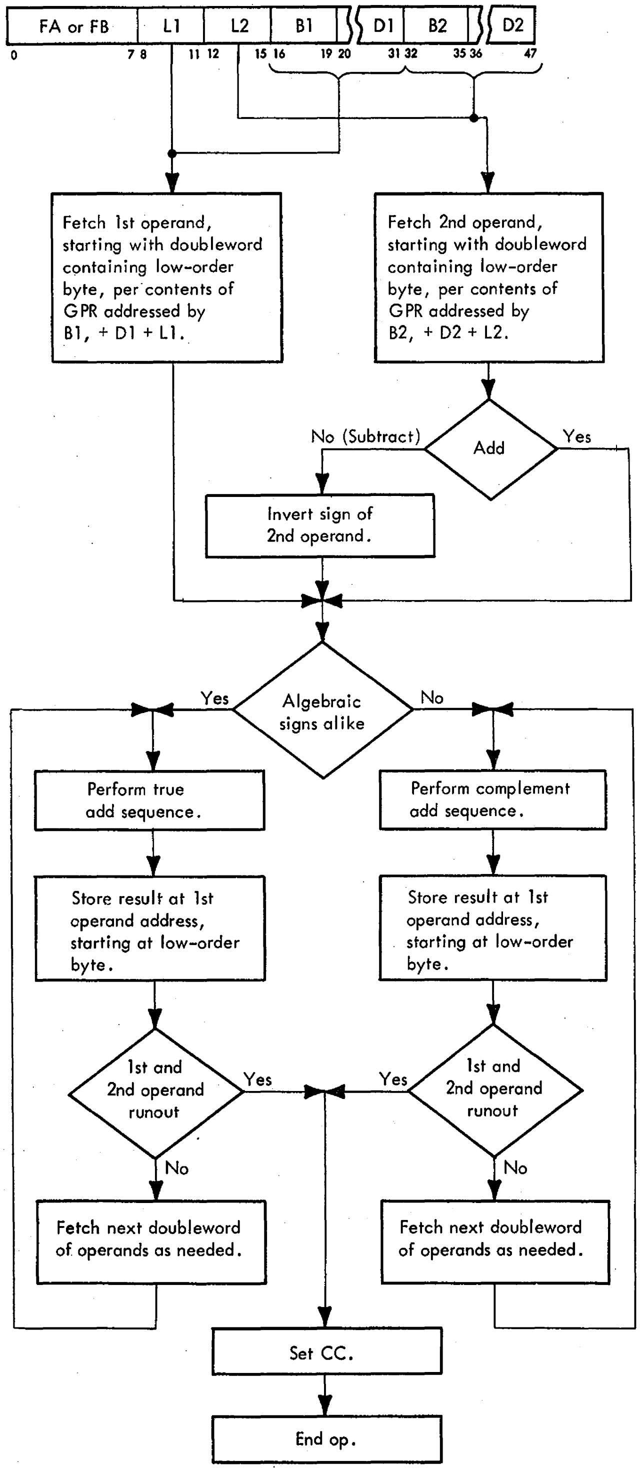
During the GIS, separate sign handling is performed for the Add, Subtract, and Compare instructions; the sign of the second operand is, in effect, inverted for Subtract and Compare instructions. After exit from the GIS, the three instructions share a common true add or complement add routine, depending on the operand signs and the instruction. Because the result of a Compare instruction is not stored into main storage, the setting of the mark triggers and overflow detection are inhibited in hardware during execution of this instruction.

Add, AP (FA) and Subtract, SP (FB)

- Algebraically add (subtract) 2nd operand (in storage) to (from) 1st operand (in storage) and place result into 1st operand location.
- SS format: (See following page.)
- CC setting:
  Result is zero: CC = 0.
  Result is less than zero: CC = 1.
  Result is greater than zero: CC = 2.
  Overflow: CC = 3.

The Decimal Add and Subtract instructions specify an algebraic addition with the sign of the second operand inverted for the subtract instruction. The sign of the result is tentatively set to agree with the sign of the first operand. Then, if the algebraic signs of the two operands are alike, a true add sequence is performed; if unlike, a complement add sequence is performed. The tentative result sign is correct in all cases except for a complement add operation where the magnitude of the second operand is equal to or greater than the magnitude of the first operand. If the magnitudes of the two operands are equal, the result is zero and a positive sign is stored into the result field. If the magnitude of the second operand is greater than that of the first, a complement result is formed and the sign of the result is inverted by setting it to the algebraic sign of the second operand when the result is recomplemented. All signs and digits are tested for validity. The operand fields may overlap when their low-order bytes coincide; therefore, it is possible to add a number to itself.

The result is stored at the first operand address. If the first operand field is too short to contain all significant digits of the result, a decimal overflow occurs.


FA or FB
L1
L2
B1
D1
B2
D2
0
7 8
11 12
15 16
19 20
31 32
35 36
47
Fetch 1st operand, starting with doubleword containing low-order byte, per contents of GPR addressed by B1, + D1 + L1.
Fetch 2nd operand, starting with doubleword containing low-order byte, per contents of GPR addressed by B2, + D2 + L2.
No (Subtract)
Add
Yes
Invert sign of 2nd operand.
Yes
Algebraic signs alike
No
Perform true add sequence.
Perform complement add sequence.
Store result at 1st operand address, starting at low-order byte.
Store result at 1st operand address, starting at low-order byte.
1st and 2nd operand runout
Yes
Yes
1st and 2nd operand runout
No
No
Fetch next doubleword of operands as needed.
Fetch next doubleword of operands as needed.
Set CC.
End op.

GIS for Add and Subtract

The GIS microprogram for the Add and Subtract instructions is shown in Diagram 5-301, FEMDM. It performs the following functions:

1. Loads into ST the doubleword containing the low-order byte of the first operand.
2. Adds L2 to the IC, after which it requests, per the IC, the doubleword containing the low-order byte of the second operand. When this doubleword arrives, loads it into AB.
3. Transfers the L1 count to F(0–3).
4. Assigns a result sign that agrees with the sign of the first operand.
5. Performs a branch on the algebraic signs of the operands (contained in STAT's C and F) to enter the true add or complement add sequence.

At the start of GIS, the IC contains the address of the high-order byte of the second operand (contents of GPR addressed by B2, + D2). To obtain the address of the low-order byte, L2 is added to the IC, and a main storage request is issued per the result.

The L1 count in E(8–11) is destroyed during subsequent execution and must be preserved in F(0–3). This action is necessary because, upon execution of the instruction, it may be found that results were placed in main storage in complement form. Because the final result must be true, the destination field is refetched and recomplemented, and the sign is inverted. In such cases, the L1 count in F(0–3) is used to refetch the correct number of destination bytes.

The result is arbitrarily assigned the sign of the first operand by performing a branch of STAT's F and C. STAT F is set if the first operand is minus. Note, however, that the sign of the second operand is not known at this time and STAT C will always be in the reset state. Thus, when STAT's F and C are alike, it indicates that STAT F is not set (first operand plus); when the STAT's are not alike, it indicates that STAT F has been set and the first operand is negative.

A second branch on STAT's F and C is performed after the sign of the second operand has been sensed, and STAT C set accordingly. STAT C is set for a minus sign for an Add instruction and for a plus sign for a Subtract instruction. This is the only difference in the execution of an Add or Subtract instruction. If the STAT's are alike, a true add sequence is entered. Upon entry into this sequence, the result always carries the correct sign. If the STAT's are not alike, a complement add sequence is entered. In this case, the algebraic sign of the result cannot be known at the start of the operation because it is dependent on the relative magnitude of the operands. If the sign has been assigned incorrectly, the result of the complement add operation will be in complement form. This condition will be detected at the completion of the instruction, in which case the result will be recomplemented and the sign inverted by setting the result sign to the algebraic sign of the second operand (per STAT C).

True Add Sequence

- True +6 add operation exits on one or more of the following conditions:
  L1 or STC = 0.
  L2 = 0.
  ABC = 0.

- STAT A is set if result is not zero.
- STAT B is set if overflow occurs.
- STAT E is set if operand digit or sign is invalid.
- STAT G is set if Compare instruction.
- STAT H is set if carry to next byte occurs.

An overall flowchart of the true add sequence and the data path used for its execution are shown in Sheet 1 of Diagram 5-302, FEMDM. The flowchart outlines the major functional steps and sequences used in the Add, Subtract, and Compare microprogram.

Upon entry into the true add sequence, the signs of both operands have been examined (by the GIS) and the correct sign has been entered into the low-order destination byte in ST. The first step in the microprogram is to true-add the digits contained in the sign bytes of the operands. The result is then placed into the digit portion of the destination byte. At this point, one complete byte of the result has been developed. The operand length codes (L1 and L2) and the status of the byte counters (STC and ABC) are examined for one or more of the following exit conditions from the true add loop:

1. STC and L1 ≠ 0, L2 = 0.
   The second operand field has run out.
2. STC and L1 ≠ 0, L2 ≠ 0, ABC = 0.
   More second operand bytes are needed.
3. STC or L1 = 0, L2 ≠ 0, ABC = 0.
   The first operand field has run out, or ST is full and more first operand bytes are needed. In either case, ST must be stored into the destination address per D. More second operand bytes are needed.
4. STC or L1 = 0, L2 ≠ 0, ABC ≠ 0.
   The same conditions exist as in item 3 except that more second operand bytes are not needed.
5. STC or L1 = 0, L2 = 0.
   The first operand field has run out, or ST is full and more first operand bytes are needed. In either case, ST must be stored into the destination address per D. The second operand field has run out.

If none of the above exit conditions exist, the microprogram re-enters the true add loop to generate the next destination byte. L1, L2, STC, and ABC are decremented one count, the selected AB and ST bytes are added in the serial adder, and the result replaces the selected ST byte. After this, the status of all counters is again sensed for exit conditions.

Upon establishing one or more exit conditions, the operations dictated by the conditions are performed, and, if L1 is not zero, the true add loop is re-entered. When L1 is zero, all destination bytes have been processed. The microprogram then performs an overflow test and a test for all-zero result.† If an all-zero result has been obtained, the address of the low-order destination byte is restored in D and a plus is stored at this address. Restoration is necessary, because D is decremented by 8 for each doubleword of first operand that is fetched. If, for example, two doublewords of first operand have been fetched, the address of the low-order destination byte is obtained by adding 16 to D.

A detailed flowchart of the true add sequence is shown in Sheet 2 of Diagram 5-302. It is an expanded version of the overall flowchart, showing the data handling used in the various subroutines of the true add operation. This data handling is straightforward for the most part and requires no explanation. Those areas in need of clarification are discussed in the following subparagraphs.

True Add Operation. The selected AB byte is gated (true +6) to the serial adder and added as a binary number to the selected ST byte. The adder output is decimal-corrected at the input to SAL and gated back to the selected ST byte. For the first (or sign) byte, only bits 0–3 of the selected AB byte are gated to the adder. The decimal correction involves examining the carry from each digit and logically subtracting 6 from each result digit that did not have a carry. As each byte is processed, ABC, STC, L1 and L2 are decremented by 1 and the selected mark trigger is set (except for a Compare instruction). The carry from each byte is saved in STAT H and, if set, results in a carry to bit 7 of the next byte processed. STAT A is set if any nonzero result digit is detected. STAT E is set if any invalid digit is detected at the inputs to the serial adder.††

Exit Conditions. An exit is made when one or more of the following conditions exist as determined by a functional branch micro-order ('Decimal' micro-order):

1. L1 or STC = 0.
2. L2 = 0.
3. ABC = 0.

Five possible exit conditions exist:

1. STC and L1 ≠ 0, L2 = 0.
   The second operand field has been completely processed. AB and ABC are cleared, and the high-order source extend routine is started to process the remaining destination field.

† The Add, Subtract, ,and Compare instructions have the same microprogram. The Compare instruction does not store the result into main storage and, upon exit from the true add loop, enters the end-op sequence.

†† When an invalid digit is detected at the serial adder in-buses, the adder forces 1's into its sum and parity output latches. This action insures that valid parity is always gated to ST from the serial adder.

2. STC and L1 ≠ 0, L2 ≠ 0, ABC = 0.
   A second operand fetch sequence is started to fetch the next doubleword of second operand to AB.
3. STC or L1 = 0, L2 ≠ 0, ABC = 0.
   The first operand has either run out, or ST is full and the next doubleword of the first operand is needed. In either case, ST must be stored if an Add or Subtract instruction is being executed. (If a Compare instruction is being executed, no mark triggers are set and ST is not stored.) The next doubleword of the second operand is needed.

   After the destination store cycle, L1 is tested for all 1's to determine whether the L1 field has run out. If L1 equals all 1's, a second operand fetch is initiated before entering the first operand runout sequence; if L1 does not equal all 1's, the first operand fetch sequence is performed, followed by the second operand fetch sequence and resumption of the true add loop.
4. STC or L1 = 0, L2 ≠ 0, ABC ≠ 0.
   The same conditions exist as in item 3 except that a second operand fetch is not needed and is not performed.
5. STC or L1 = 0, L2 = 0.
   A storage request per D is issued to store ST into the destination field (unless a Compare instruction is being executed). AB and ABC are cleared to start the high-order source extend routine. A further test is required to determine whether L1 was zero.

   If L1 is now all 1's, all destination bytes have been processed. A carry from the last destination byte indicates an overflow condition, and STAT B is set.

   If L1 is not all 1's, a first operand fetch sequence is started, after which the high-order source extend routine is resumed.

First Operand Fetch. A separate entry is made into this routine, per STAT G, for a Compare instruction. In a compare operation, a D request for the next doubleword of first operand has already been given. D is decremented by 8, and the doubleword arriving at the SDBO is gated to ST.

For Add or Subtract instructions, D is decremented by 8, and a D request is made for the next doubleword of first operand. F is incremented by 1 to record the number of fetches made. This information will be required to restore the low-order address in D in case an all-zero result is obtained. If ABC equals 111, a second operand fetch routine is started. If ABC does not equal 111, the appropriate addition or high-order source extend is started.

Second Operand Fetch. The IC is decremented by 8, and the next doubleword of second operand is fetched to AB. After this, the appropriate addition or overflow routine is started as determined by the L1 count.

Second Operand Runout. An all-zeros AB byte is gated true +6 to the serial adder and added to the selected ST byte. The result is decimal-corrected and gated back to the selected ST byte. STAT's A, E, and H and the mark triggers are set as previously explained.

L1 and STC are decremented by 1 as each byte is processed; ABC and L2 are not stepped. The sequence is repeated until L1 is stepped to zero, with an exit to the destination store and first operand fetch sequences whenever STC equals 7. A carry from the last destination byte is an overflow condition and sets STAT B. The end-op sequence is started when L1 equals zero.

First Operand Runout and Overflow Test. An overflow condition exists whenever a carry results as the last destination byte is processed or whenever a nonzero digit is detected in the source field after the destination field has been processed. STAT B is set if STAT H is set when entering this routine. Next, the remaining second operand bytes are gated true +6 to the serial adder with STAT B being set if any nonzero bytes are detected.

ABC and STC are decremented by 1 as each byte is processed. The next source doubleword is fetched to AB whenever ABC is stepped to zero unless L2 equals zero. When L2 is stepped to zero, the end-op sequence is started.

Zero Result. If at the completion of the true add operation STAT A is not set, an all-zero result has been obtained. In this case, the Add and Subtract instructions always force a positive sign into the low-order byte of the destination field. (If STAT G is set, an exit is made to the end-op sequence because no correction of the result is required for a Compare instruction.)

The low-order destination address is regenerated by adding 8 to D the number of times indicated in F(4–7). STC is set per D(21–23), and the selected ST byte is cleared.

STAT B is examined to determine an overflow condition. For a zero result and no overflow, a plus sign is inserted via the serial adder into the low-order destination byte with the selected mark trigger being set. A storage request is given to store the sign into the destination field, and the end-op sequence is started. For a zero result and overflow, the destination sign is not corrected. The end-op sequence is started immediately.

End-Op Sequence. The instruction address (original content of the IC) is restored from the LSWR to the IC, and STAT G is reset. A data program interruption occurs if STAT E is set. A decimal overflow program interruption occurs if STAT B is set and STAT E is not set. The CC is set per hardware conditions as shown in Table 3-12.

Table 3-12. Condition Code Setting Per Hardware Conditions, Decimal Instructions

Hardware Conditions	Setting
Not STAT B ● Not STAT A ● (Add, or Subtract, or Zero and Add)	0
Not STAT A ● Not STAT H ● Compare	0
Not STAT A ● STAT H ● STAT F ● Not STAT C ● Compare	0
Not STAT A ● STAT H ● Not STAT F ● STAT C ● Compare	0
Not STAT B ● STAT A ● STAT F ● (Add, or Subtract, or Zero and Add)	1
STAT A ● STAT F ● STAT H ● Compare	1
STAT A ● STAT C ● Not STAT H ● Compare	1
STAT F ● STAT C ● STAT H ● Compare	1
Not STAT B ● STAT A ● Not STAT F ● (Add, or Subtract, or Zero and Add)	2
STAT A ● Not STAT F ● STAT H ● Compare	2
STAT A ● Not STAT C ● Not STAT H ● Compare	2
STAT H ● Not STAT F ● Not STAT C ● Compare	2
STAT B ● (Add, or Subtract, or Zero and Add)	3

Note: ● Designates logical AND connective.

Complement Add Sequence

- Complement add operation with exits on one or more of the following conditions:
  1. L1 or STC = 0.
  2. L2 = 0.
  3. ABC = 0.
- STAT A is set if result is not zero.
- STAT B is set if overflow occurs.
- STAT D is set if result must be recomplemented.
- STAT E is set if operand digit or sign is invalid.
- STAT G is set if Compare instruction.
- STAT H is set if carry to next byte occurs.
- Carry out of last destination byte indicates result is in true form; no carry condition indicates result is in complement form.

Diagram 5-303, Sheet 1, FEMDM, is an overall flowchart of the complement add sequence. This flowchart outlines the major functional steps and sequences used in the Add, Subtract, and Compare microprogram.

Upon entry into this sequence, the signs of both operands have been examined (by the GIS) and the sign of the first operand has been inserted as the sign of the result. This sign may or may not turn out to be the correct sign: if the first operand is larger than the second, the result carries the correct sign; if the reverse is true, the sign of the result must be inverted.

Basically, the complement add microprogram is similar to the true add sequence previously described. The first step in the microprogram is to complement add the digits contained in the sign bytes of the operands. The result is then placed into the digit portion of the destination sign byte. At this point, one complete byte of result has been developed. The operand length codes (L1 and L2) and the status of the byte counters (STC and ABC) are examined for one or more of the following conditions:

1. The result byte is contained in the last byte of ST and must be stored (STC = 0).
2. Additional first operand bytes must be fetched from main storage (STC = 0 and L1 ≠ 0).
3. Additional second operand bytes must be fetched from main storage (ABC = 0 and L2 ≠ 0).
4. The second operand has run out; i.e., all second operand bytes have been processed, and zeros must be added to the first operand bytes (L2 = 0 but L1 ≠ 0).
5. The first operand has run out; i.e., the destination field has been completely processed (L1 = 0).

If none of the above conditions exist, the microprogram enters the complement add loop to generate the next destination byte. L1, L2, STC, and ABC are decremented one count, the selected AB byte is complement-added to the selected ST byte, and the result replaces the selected ST byte. After this, the status of all counters is again sensed for exit conditions.

Upon establishing one or more exit conditions, the operations dictated by the conditions are performed and, if L1 is not zero, the complement add loop is re-entered. When L1 is zero, all destination bytes have been processed. The microprogram then performs an overflow test, a zero-result test, and a complement result test.† If an

† The Compare instruction does not store the result into main storage and, upon exit from the subtract loop, enters the end-op sequence.

all-zero or complement result has been generated, the address of the low-order destination byte is restored to D. Then, the result is either recomplemented and the sign inverted or a plus is stored into the low-order destination byte.

Sheet 2 of Diagram 5-303 is a detailed flowchart of the complement add sequence. It is an expanded version of the overall flowchart, showing the data handling in the various subroutines of the complement add operation. The complement add operation is similar to the true add sequence described previously. For this reason, only the differences are discussed below.

Complement Add Operation. The selected AB byte is converted to 2's complement form at the input to the serial adder, and is then added to the selected ST byte. For the first (or sign) byte, bits 0–3 only of the selected AB byte are gated in inverted binary form to the adder, with a hot carry supplied to bit 3 to convert to 2's complement.

Second Operand Runout. An all-zeros AB byte is gated in complement form to the serial adder and added to the selected ST byte. Thus the second operand is extended with high-order binary 1's (decimal 9's + 6).

Overflow Test. Generally, an overflow condition exists, if, upon processing all destination bytes, a non-zero source byte is detected. One exception occurs when the first source byte sensed, after the first operand has run out, equals 1 and a "carry" condition exists. A carry condition is determined by STAT's A and H being set; i.e., a nonzero result and a carry from the previous byte.

When L2 has been stepped to zero, a carry from the last destination byte is examined. A carry condition indicates a true result, and the end-op sequence is started. No carry indicates a complement result, and a recomplement sequence is started for Add and Subtract instructions. For a Compare instruction (STAT G set), the end-op sequence is started immediately.

Zero Result and Recomplement Setup. STAT D is set when an entry is made to this routine because of the result's being in complement form. STAT D is not set when an entry is made because of a zero result. If STAT G is set when entering this routine, an exit is made to the end-op sequence because no corrections of the result are required for the Compare instruction.

The low-order destination address is regenerated by adding 8 to D the number of times indicated in F(4–7). STC is set per D(21–23), and the selected ST byte is cleared. If an overflow condition exists (STAT B set), the results need not be corrected and the end-op sequence is started immediately.

Recomplement Sequence. The original L1 count was saved in F(0–3) by the GIS. This count is now placed into the L2 location in E; i.e., E(12–15). The L1 location in E(8–11) is set to zero and then decremented one count to provide an exit from the first operand fetch routine to the recomplement sequence. The complement result is gated from the SDBO to AB, and the recomplement sequence is started.

The sign byte is processed by gating bits 0–3 of the selected AB byte in inverted binary form to the serial adder, with a hot carry supplied to bit 3. The sign is inverted by inserting the algebraic sign of the second operand into serial adder bits 4–7, as determined by STAT C. Bits 0–3 are decimal-corrected, and the adder output is gated to ST.

All bytes following the sign byte are processed by gating the selected AB byte complement to the serial adder, where it is added to an all-zero ST byte. The adder output is decimal-corrected and gated back to the selected ST byte.

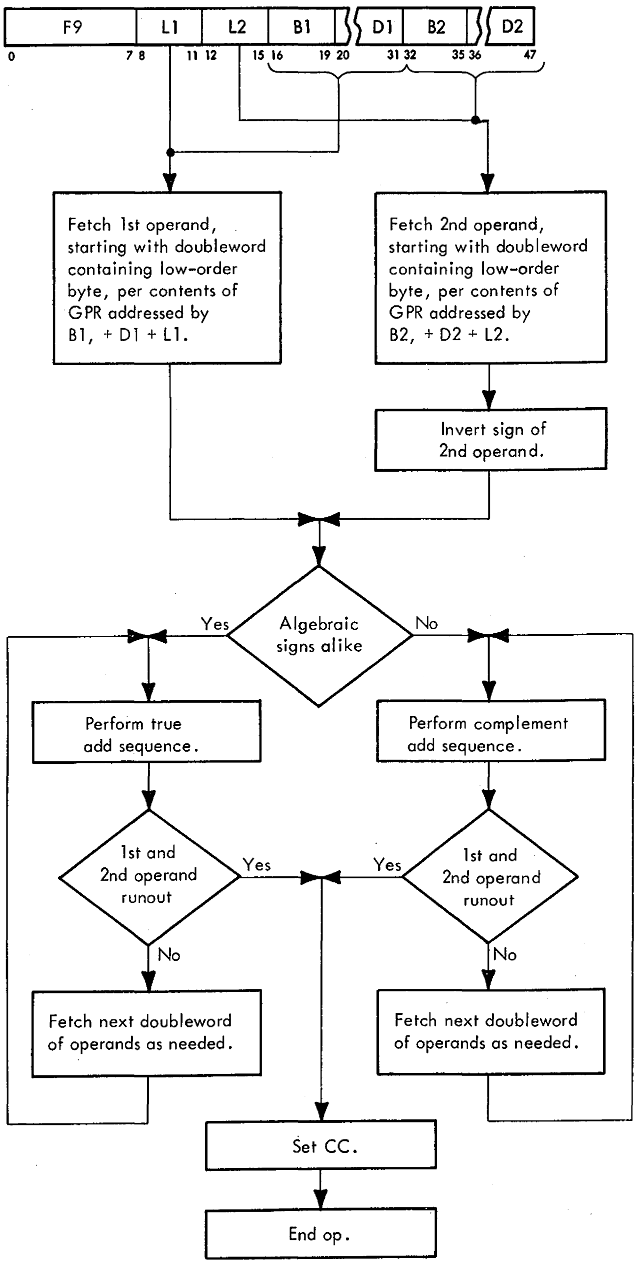
As each byte (including the sign byte) is processed, the ABC, STC, and L2 counts are decremented. The mark trigger selected by the STC is set. The serial adder carry is saved in STAT H. STAT A is set on nonzero digits, and STAT E is set on invalid digits.

Recomplementation is continued until the L1 in E(12–15) is stepped to zero. If ABC steps to zero and L1 is not zero, ST is stored and the next doubleword of destination is fetched to AB. When L1 steps to zero, ST is stored into the destination field, AB is cleared, and STAT F is set or reset per STAT C. The CC is set per hardware conditions (see Table 3-12), and the instruction is ended.

Compare, CP (F9)

- Algebraically compare 1st operand (in storage) with 2nd operand (in storage) and set CC according to result.
- SS format: (See left column of next page.)
- CC setting:
  Operands are equal: CC = 0.
  1st operand is less than 2nd operand: CC = 1.
  1st operand is greater than 2nd operand: CC = 2.

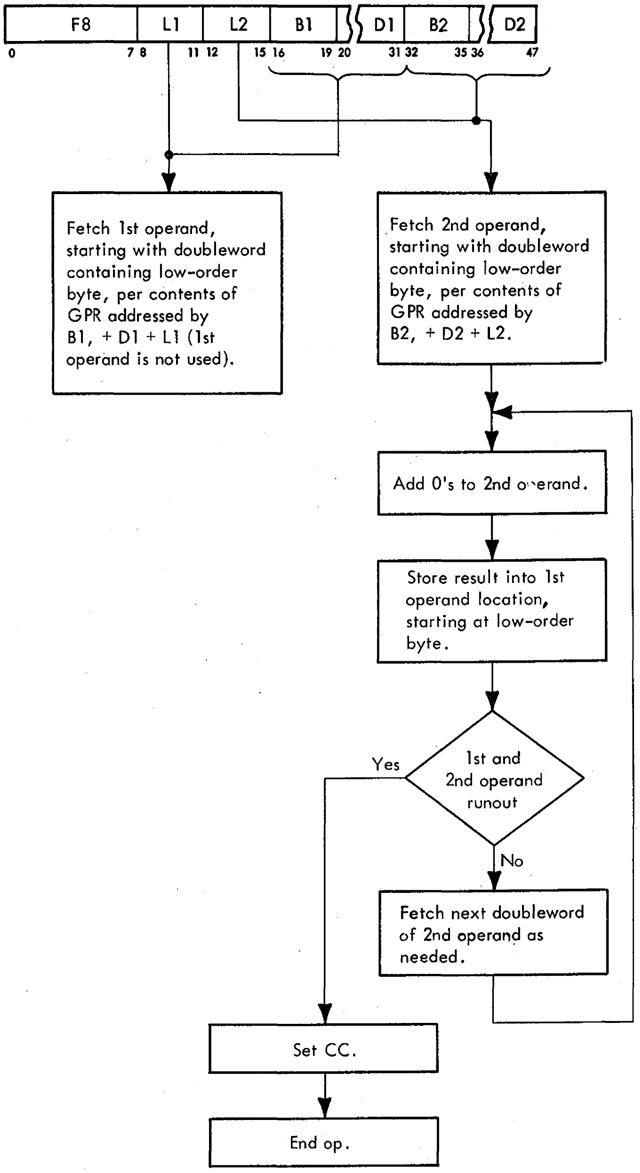
The CP instruction shares the same true add and complement add routines used by the Add and Subtract instructions. The GIS microprogram for the Compare instruction is shown in Diagram 5-301. As in the Subtract instruction, this microprogram effectively inverts the sign of the second operand by setting STAT C on a positive sign. The result of the compare operation is not placed


F9
L1
L2
B1
D1
B2
D2
0
7 8
11 12
15 16
19 20
31 32
35 36
47
Fetch 1st operand, starting with doubleword containing low-order byte, per contents of GPR addressed by B1, + D1 + L1.
Fetch 2nd operand, starting with doubleword containing low-order byte, per contents of GPR addressed by B2, + D2 + L2.
Invert sign of 2nd operand.
Algebraic signs alike
Yes
No
Perform true add sequence.
Perform complement add sequence.
1st and 2nd operand runout
Yes
Yes
1st and 2nd operand runout
No
No
Fetch next doubleword of operands as needed.
Fetch next doubleword of operands as needed.
Set CC.
End op.

into main storage. STAT G is set to provide a means of taking special action, where required for the Compare instruction, during execution of the common true add or complement add sequences.

ZERO AND ADD, ZAP (F8)

- Place 2nd operand (in storage) into 1st operand location (in storage).
- SS format:


F8
L1
L2
B1
D1
B2
D2
0
7 8
11 12
15 16
19 20
31 32
35 36
47
Fetch 1st operand, starting with doubleword containing low-order byte, per contents of GPR addressed by B1, + D1 + L1 (1st operand is not used).
Fetch 2nd operand, starting with doubleword containing low-order byte, per contents of GPR addressed by B2, + D2 + L2.
Add 0's to 2nd operand.
Store result into 1st operand location, starting at low-order byte.
Yes
1st and 2nd operand runout
No
Fetch next doubleword of 2nd operand as needed.
Set CC.
End op.

- CC setting:
  2nd operand is zero: CC = 0.
  2nd operand is less than zero: CC = 1.
  2nd operand is greater than zero: CC = 2.
  2nd operand cannot fit into destination field: C = 3.

The operation specified by the ZAP instruction is equivalent to addition to zero. A zero result is always made positive. When high-order digits are lost because of overflow, a zero result has the sign of the second operand.

Only the second operand is checked for valid sign and digit codes. Extra high-order zeros are supplied if needed. When the first operand field is too short to contain all significant digits of the second operand, a decimal overflow occurs and results in a decimal overflow program interruption, provided that the decimal overflow mask bit, PSW(37), is 1. The first and second operand fields may overlap when the rightmost byte of the first operand field is coincident with or to the right of the rightmost byte of the second operand. A flowchart of the GIS and execution of the Zero and Add instruction is shown in Diagram 5-304, FEMDM.

At the start of the GIS, the following actions have been performed by SS I-Fetch: (1) a D request has been issued for the doubleword containing the low-order byte of the first operand; (2) the low-order byte address of the first operand (contents of GPR addressed by B1, + D1 + L1) is in D; and (3) the high-order byte address of the second operand (contents of GPR addressed by B2, +D2) is in the IC. During the GIS, the doubleword containing the low-order byte of the first operand is gated from the SDBO to ST, the IC is incremented by the L2 count to address the low-order byte of the second operand, the STC is set to the low-order destination byte, an IC request for the second operand is issued, and a word overlap test is performed. If a word-overlap condition is predicted by this test, the instruction address is restored to the IC, the 'invalid data interrupt' trigger is set, and the instruction is ended. If no word-overlap condition is detected, the doubleword containing the low-order byte of the second operand is gated from the SDBO to AB.

The sign byte is processed by gating bits 0–3 of the selected AB byte to the serial adder. Bits 4–7 of the AB byte are decoded for a positive, negative, or invalid sign. The approved plus or minus sign is inserted into SAL(4–7) and gated, with the digit, to the selected ST byte.

All bytes following the sign byte are processed by gating the selected AB byte true +6 to the serial adder. The selected ST byte is not gated to the adder, and the validity check at the adder B-side is inhibited in hardware. The adder output is decimal-corrected and gated to the selected ST byte.

As each byte is processed, including the first byte, the ABC, STC, L1, and L2 are decremented, the selected mark trigger is set, STAT E is set for invalid data, and STAT A is set for a nonzero digit.

The byte-by-byte transfer from AB to ST is continued until one or more of the following exit conditions are detected via a ROS branch ('Decimal' micro-order):

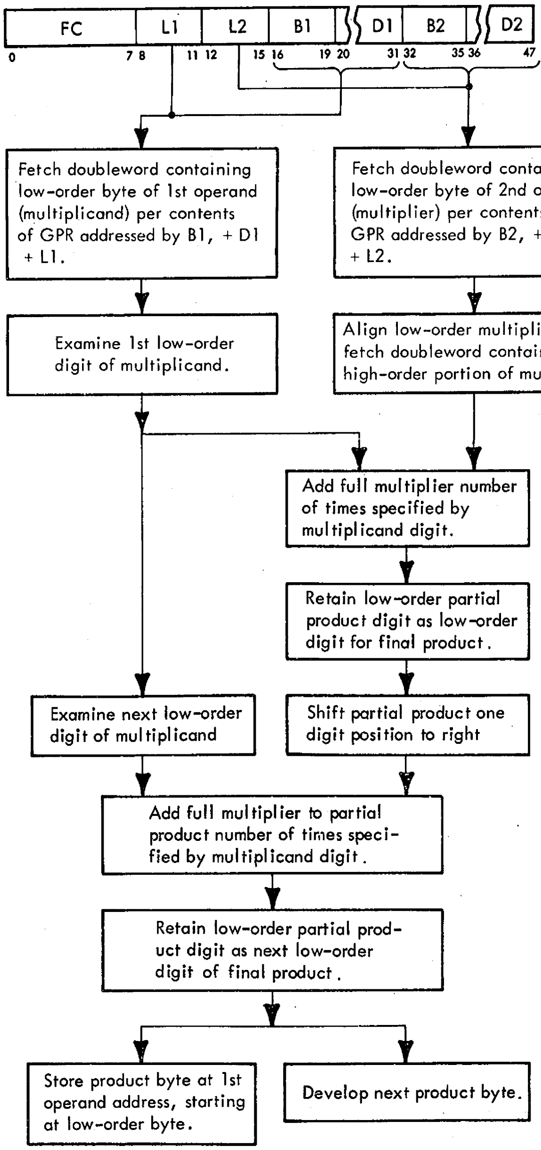
1. STC and L1 ≠ 0, L2 = 0.
   The second operand field has run out. AB and ABC are cleared and zeros are gated to ST per the STC until L1 or STC equals zero. If the STC is reduced to zero before L1, ST is stored per the D address, and zeros are gated to ST per the STC until L1 is reduced to zero.
2. STC and L1 ≠ 0, L2 ≠ 0, ABC = 0.
   More second operand bytes are needed. The next doubleword of the second operand is gated to AB and the zero and add loop is resumed.
3. STC or L1 = 0, L2 ≠ 0, ABC = 0.
   The destination field has run out or ST is full, requiring a destination store cycle. More second operand bytes are needed. ST is stored into the destination address per D, and the next doubleword of the second operand is fetched per the IC and gated to AB. If L1 equals all 1's (the destination field has run out), AB is tested for nonzero digits to determine if a decimal overflow condition exists. If L1 does not equal all 1's, the zero and add loop is resumed.
4. STC or L1 = 0, L2 ≠ 0, ABC ≠ 0.
   The same conditions exist as in item 3, except that a second operand fetch is not needed and is not performed.
5. STC or L1 = 0, L2 = 0.
   The destination field has run out or ST is full, requiring a destination store cycle. The second operand field has run out. ST is stored in the destination address per D. If L1 equals all 1's, both operands have run out. If L1 does not equal all 1's, the second operand runout sequence is performed.

MULTIPLY, MP (FC)

- Multiply 1st operand (in storage) by 2nd operand (in storage) and place result into 1st operand location.
- SS format: (See following page.)
- Maximum multiplicand field (1st operand) is 16 bytes.
- Maximum multiplier field (2nd operand) is 8 bytes.
- Multiplicand field initially contains high-order zero field equal in length to multiplier field.
- L2 > 7 or L2 ≧ L1 causes specification program interruption.
- Multiplication accomplished by repetitive addition or subtraction:

Multiplicand Digit	Sequence Selected
0	–
1–4	Addition
5–9	Subtraction

The Decimal Multiply instruction replaces the multiplicand (1st operand) with the product of the multiplicand and the multiplier (2nd operand). To be able


FC
L1
L2
B1
D1
B2
D2
0
7 8
11 12
15 16
19 20
31 32
35 36
47
Fetch doubleword containing low-order byte of 1st operand (multiplicand) per contents of GPR addressed by B1, + D1 + L1.
Fetch doubleword containing low-order byte of 2nd operand (multiplier) per contents of GPR addressed by B2, + D2 + L2.
Examine 1st low-order digit of multiplicand.
Align low-order multiplier and fetch doubleword containing high-order portion of multiplier.
Add full multiplier number of times specified by multiplicand digit.
Retain low-order partial product digit as low-order digit for final product.
Examine next low-order digit of multiplicand
Shift partial product one digit position to right
Add full multiplier to partial product number of times specified by multiplicand digit.
Retain low-order partial product digit as next low-order digit of final product.
Store product byte at 1st operand address, starting at low-order byte.
Develop next product byte.

to store the product in the multiplicand field at all times, several restrictions are imposed on both the multiplicand and the multiplier:

1. In any multiply operation, the maximum number of product digits that can be obtained is equal to the sum of the digits in the two operands. Because the product is stored in the multiplicand field, this field must initially contain high-order zero digits for at least a field size equal to that of the multiplier. Thus the multiplicand field is initially split into two parts; the high-order zero field of length equal to the multiplier, and the low-order field containing the effective multiplicand digits. This arrangement of the multiplicand ensures that product overflow will not occur (Figure 3-14).
2. By definition, the multiplier field must be at least one digit less than the multiplicand. Because the multiplicand must initially contain a zero field equal in size to the multiplier digits, the multiplier size is limited to 8 bytes (15 digits and sign). A specification program interruption occurs if the multiplier length code designates more than 8 bytes (L2 is greater than 7), or if L2 is greater than or equal to L1.
3. The maximum product size is 31 digits and sign (16 multiplicand digits plus 15 multiplier digits). The sign is determined algebraically from the multiplier and the multiplicand signs, even if one or both operands are zero. Because during sign resolution two sign positions are merged into one, at least one high-order digit of the product field is zero.

The multiply operation is executed in much the same manner as in manual arithmetic.† The multiplicand is examined one digit at a time, starting with the low-order digit, and the entire multiplier is added the number of times specified by the multiplicand digit. After the first multiplicand digit has been processed, the low-order digit of the resulting partial product (PP) is saved as the low-order product digit. The PP is then shifted one digit position to the right and brought into computation of the next product digit (one order higher than before). This time, the multiplier is added to the PP the number of times specified by the next digit of the multiplicand, and the low-order digit of the new PP thus formed becomes the next product digit. The PP is again shifted to the right, and the sequence is continued until all digits of the multiplicand have been processed. The PP resulting after the last multiplicand digit has been processed becomes the high-order product.

Figure 3-15 illustrates a typical repetitive addition sequence used for multiplication. As each multiplicand byte is processed, the multiplicand length code (L1) is reduced by one count and compared with the multiplier length code (L2). When L1 = L2, all effective multiplicand digits have been processed and the operation is completed.

To reduce the number of computations in the multiply operation, either a repetitive add or a repetitive subtract sequence may be performed. Selection of the sequence is dependent on the magnitude of the multiplicand digit under consideration. An add sequence is selected if the magnitude of the digit is in the range of 1 through 4. For multiplicand digits of magnitude 5 or greater, a subtract sequence is selected. This sequence deducts the multiplier from the PP the number of times specified by the 10's complement of the multiplicand digit and then adds 1 to the next digit of the multiplicand; increasing the next high-order digit of the multiplicand has the effect of

† The major difference is that the roles of the multiplicand and the multiplier are reversed. Because of its size (up to 16 bytes), the entire multiplicand cannot be held in the CPU at one time. For this reason, the full multiplier (up to 8 bytes) is fetched to the CPU and multiplied by the individual digits of the multiplicand, which is fetched from main storage 1 byte at a time.

adding the multiplier 10 times. For example, the equivalent of a multiplication by 7 is subtracting the operand 3 times to obtain a negative PP and then effectively adding the operand to the PP 10 times.

An example of a typical subtract sequence used for multiplication is shown in Figure 3-16. Note that the PP resulting from a subtract operation is in 10's complement form. When the 10's complement PP is shifted right, its high-order digit position must be extended with a 9.

Following are general and detailed descriptions of the multiply microprogram. The general description outlines the overall structure of the microprogram, enumerates its major functional steps and sequences, and explains their relationship to the overall operation. The detailed description analyzes each sequence individually, making specific references to the register-to-register data transfer in the CPU.

General Description

Upon entering the multiply microprogram, the following actions have been performed by SS I-Fetch:

1. A D request has been issued for the doubleword containing the low-order multiplicand byte.
2. The low-order multiplicand address has been placed into D.
3. The IC contents have been transferred to the LSWR, and the high-order multiplier address has been placed into the IC.

An overall flowchart of the multiply microprogram and the general data path used for its execution are shown in Sheet 1, of Diagram 5-305, FEMDM. The major subroutines and functional steps, shown in the figure, are explained below. Additional simplified diagrams are provided as an aid in visualizing the data handling performed. For the most part, these diagrams do *not* show the gates and data paths used in the CPU, but are intended solely to convey how the multiply algorithm is implemented. For purposes of illustration, a seven-byte multiplicand and a four-byte multiplier are assumed in these diagrams.

General Initialization Sequence

This sequence gates the multiplicand from the SDBO to ST and sets the STC to the low-order multiplicand byte. It increments the address in the IC to the address of the low-order byte of the multiplier by adding L2 to the IC contents. An IC request is issued for the multiplier (second operand), starting at the low-order address. The contents of D are transferred to the STC.

The GIS gates the multiplier from the SDBO to AB and sets the ABC to the low-order multiplier byte. It also performs several actions relating to the subsequent left-adjust sequence of the multiplier:

1. The low-order digit of the multiplicand (in ST) is transferred to F(0–3).
2. STAT F is set if the sign of the multiplicand is negative and STAT E is set if the sign is invalid.
3. The multiplier length code, L2, is transferred to F(4–7).

The functions performed by the GIS are illustrated in Figure 3-17.

Specification Test

This test verifies that the length codes for both operands in the instruction are correctly specified; i.e., L2 specifies eight bytes or less and is smaller than L1.

Incorrect Specification

Detection of an invalid specification forces a specification program interruption. The instruction address is restored from the IC to the LSWR, and the instruction is ended.

Multiplier Left-Adjust Sequence

The multiplier bytes are transferred from AB to ST in such a manner that the high-order multiplier byte occupies the leftmost byte in ST. STAT C is set if the multiplier sign is negative and STAT E is set if the multiplier sign is invalid.

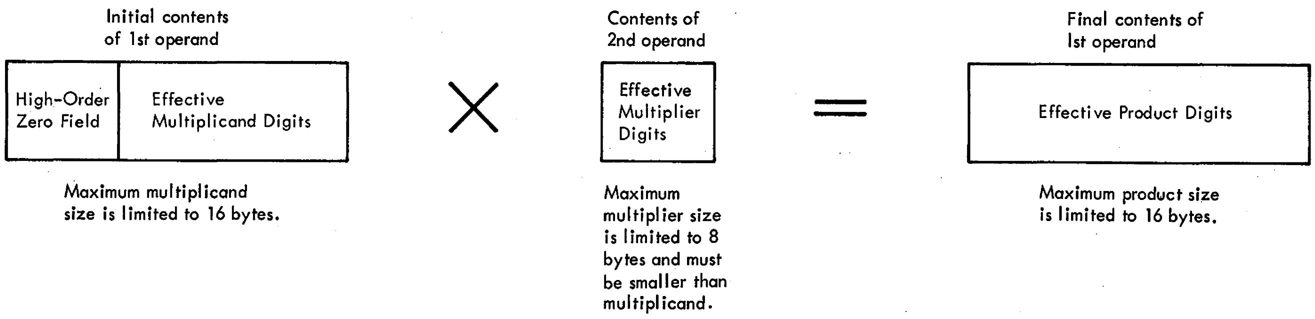

Initial contents of 1st operand
High-Order Zero Field
Effective Multiplicand Digits
Maximum multiplicand size is limited to 16 bytes.
×
Contents of 2nd operand
Effective Multiplier Digits
Maximum multiplier size is limited to 8 bytes and must be smaller than multiplicand.
=
Final contents of 1st operand
Effective Product Digits
Maximum product size is limited to 16 bytes.

Figure 3-14. Operand Specifications for Decimal Multiply Instruction

Multiply (+204) by (-32) to obtain a product of (-6,528).

Execution:

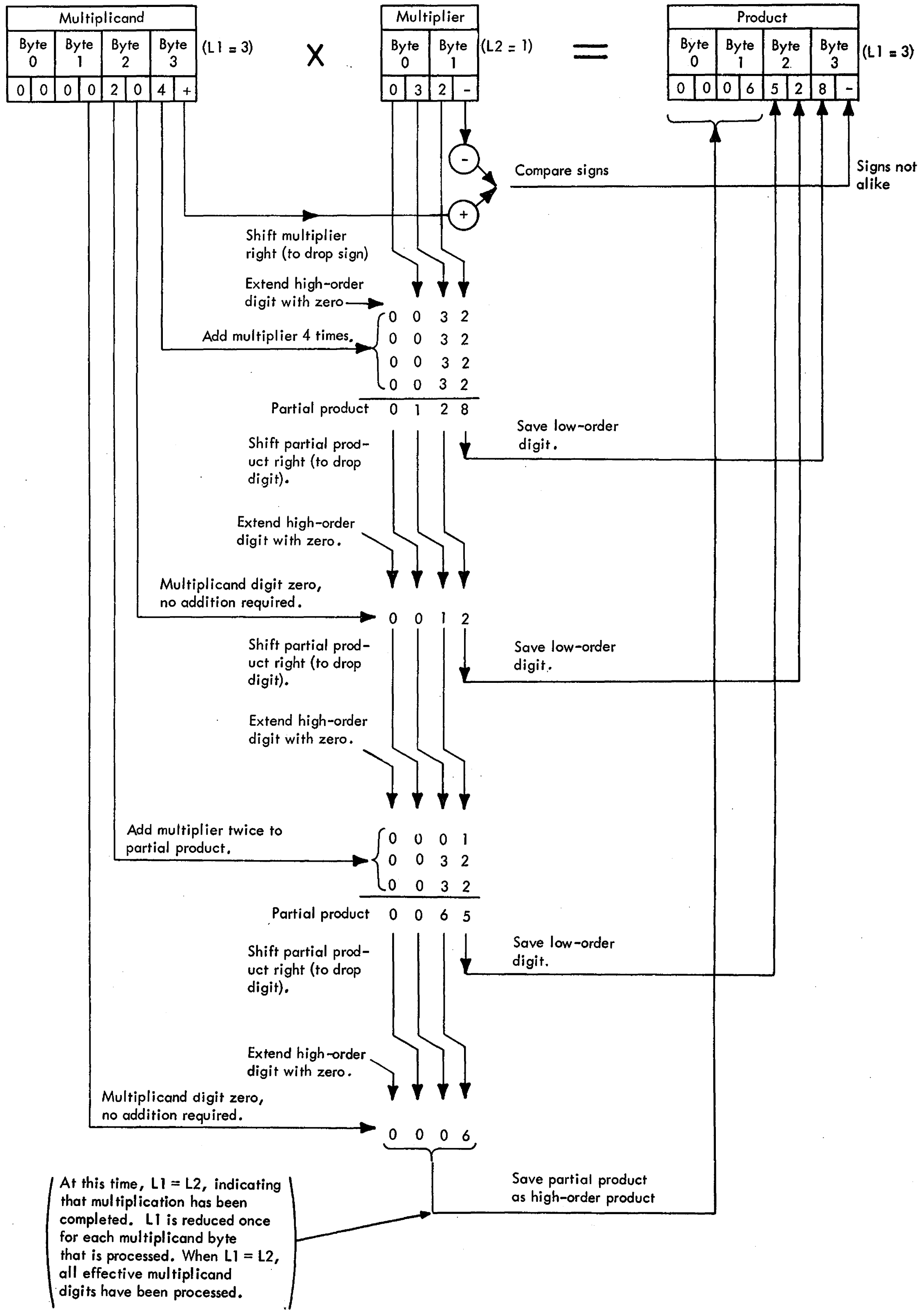
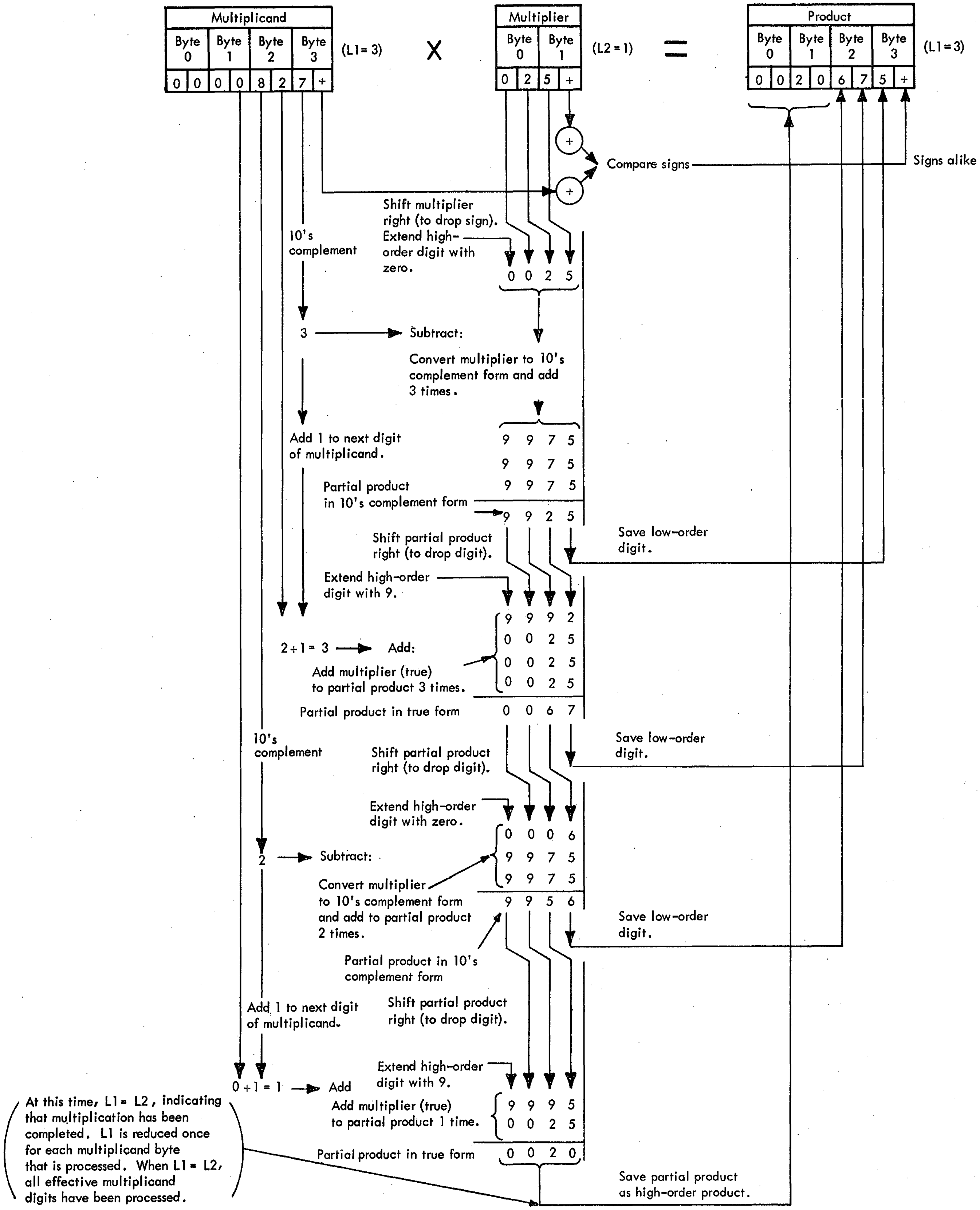

Multiplicand
Byte 0
Byte 1
Byte 2
Byte 3
0 0 0 0 2 0 4 +
(L1 = 3)
X
Multiplier
Byte 0
Byte 1
0 3 2 -
(L2 = 1)
Product
Byte 0
Byte 1
Byte 2
Byte 3
0 0 0 6 5 2 8 -
(L1 = 3)
-
+
Compare signs
Signs not alike
Shift multiplier right (to drop sign)
Extend high-order digit with zero
Add multiplier 4 times.
0 0 3 2
0 0 3 2
0 0 3 2
0 0 3 2
Partial product 0 1 2 8
Save low-order digit.
Shift partial product right (to drop digit).
Extend high-order digit with zero.
Multiplicand digit zero, no addition required.
0 0 1 2
Save low-order digit.
Shift partial product right (to drop digit).
Extend high-order digit with zero.
Add multiplier twice to partial product.
0 0 0 1
0 0 3 2
0 0 3 2
Partial product 0 0 6 5
Save low-order digit.
Shift partial product right (to drop digit).
Extend high-order digit with zero.
Multiplicand digit zero, no addition required.
0 0 0 6
Save partial product as high-order product
At this time, L1 = L2, indicating that multiplication has been completed. L1 is reduced once for each multiplicand byte that is processed. When L1 = L2, all effective multiplicand digits have been processed.

Figure 3-15. Typical Multiply Add Sequence

Multiply (+827) by (+25) to obtain a product of (+20,675).

Execution:



Multiplicand
Byte 0
Byte 1
Byte 2
Byte 3
0 0 0 0 8 2 7 +
(L1= 3)
X
Multiplier
Byte 0
Byte 1
0 2 5 +
(L2 = 1)
=
Product
Byte 0
Byte 1
Byte 2
Byte 3
0 0 2 0 6 7 5 +
(L1=3)
+
+
Compare signs
Signs alike
Shift multiplier right (to drop sign).
Extend high-order digit with zero.
10's complement
0 0 2 5
3
Subtract:
Convert multiplier to 10's complement form and add 3 times.
Add 1 to next digit of multiplicand.
9 9 7 5
9 9 7 5
9 9 7 5
Partial product in 10's complement form
9 9 2 5
Shift partial product right (to drop digit).
Save low-order digit.
Extend high-order digit with 9.
9 9 9 2
0 0 2 5
0 0 2 5
0 0 2 5
2+1= 3
Add:
Add multiplier (true) to partial product 3 times.
Partial product in true form
0 0 6 7
10's complement
Shift partial product right (to drop digit).
Save low-order digit.
Extend high-order digit with zero.
0 0 0 6
9 9 7 5
9 9 7 5
2
Subtract:
Convert multiplier to 10's complement form and add to partial product 2 times.
9 9 5 6
Save low-order digit.
Partial product in 10's complement form
Add 1 to next digit of multiplicand.
Shift partial product right (to drop digit).
Extend high-order digit with 9.
0 + 1 = 1
Add
Add multiplier (true) to partial product 1 time.
9 9 9 5
0 0 2 5
Partial product in true form
0 0 2 0
Save partial product as high-order product.
At this time, L1 = L2, indicating that multiplication has been completed. L1 is reduced once for each multiplicand byte that is processed. When L1 = L2, all effective multiplicand digits have been processed.

Figure 3-16. Typical Multiply Subtract Sequence

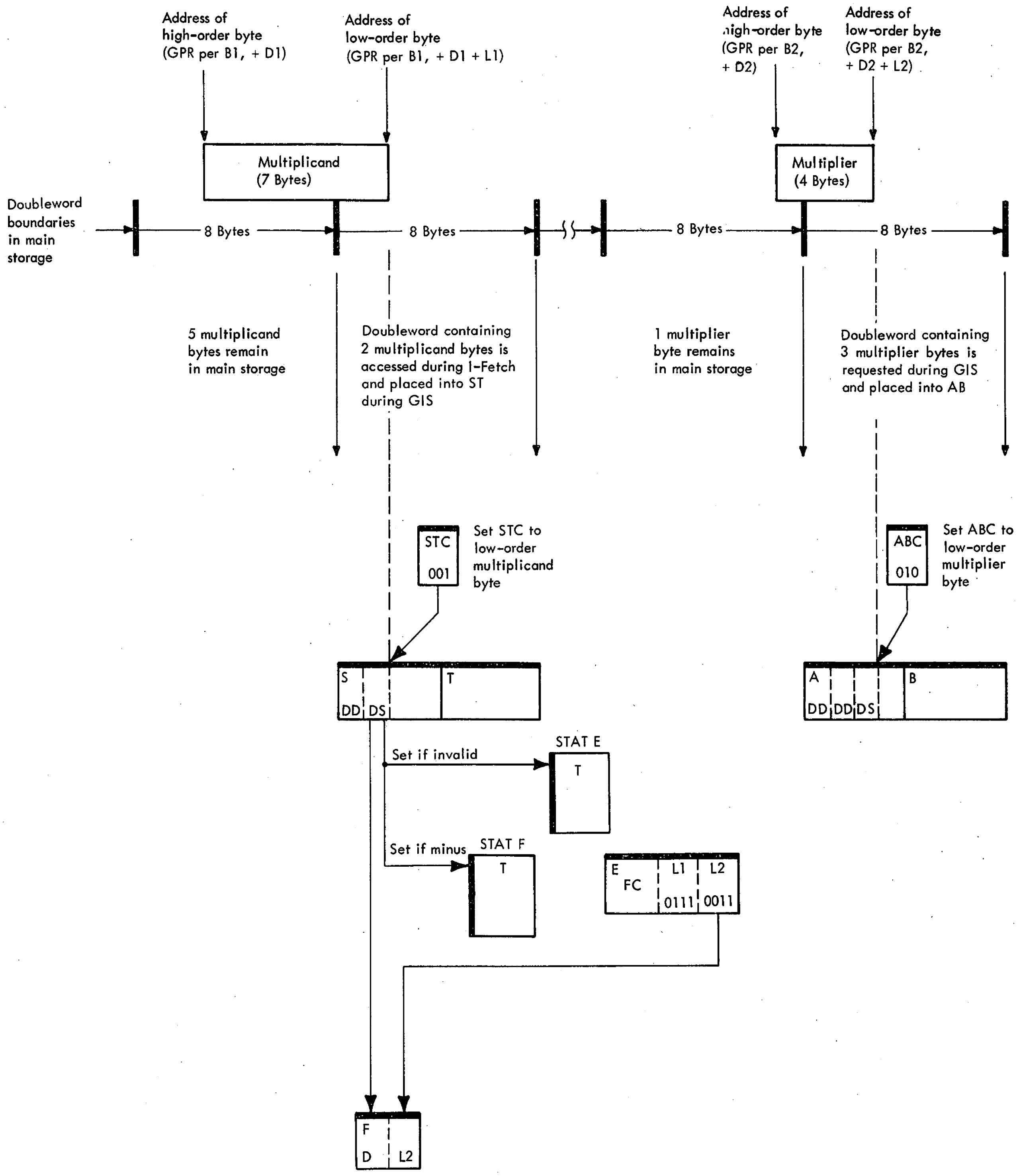

Address of high-order byte (GPR per B1, + D1)
Address of low-order byte (GPR per B1, + D1 + L1)
Address of high-order byte (GPR per B2, + D2)
Address of low-order byte (GPR per B2, + D2 + L2)
Multiplicand (7 Bytes)
Multiplier (4 Bytes)
Doubleword boundaries in main storage
8 Bytes
8 Bytes
8 Bytes
8 Bytes
5 multiplicand bytes remain in main storage
Doubleword containing 2 multiplicand bytes is accessed during I-Fetch and placed into ST during GIS
1 multiplier byte remains in main storage
Doubleword containing 3 multiplier bytes is requested during GIS and placed into AB
STC
001
Set STC to low-order multiplicand byte
ABC
010
Set ABC to low-order multiplier byte
S
T
DD
DS
A
B
DD
DD
DS
Set if invalid
STAT E
T
Set if minus
STAT F
T
E
FC
L1
L2
0111
0011
F
D
L2

Figure 3-17. Data Handling During GIS of Decimal Multiply

The left-adjust transfer is initiated by setting the STC per L2 (Figure 3-18). Because the maximum multiplier length is limited to eight bytes, only three of the four bit positions in L2 are needed to specify the length code; i.e., the count in L2 may range from a minimum of 0000 (for 1 byte) to a maximum of 0111 (for eight bytes). Setting the STC per L2 automatically selects, according to the multiplier size, the correct ST position for the low-order byte of the multiplier; the number of bytes to the left of the selected ST position corresponds to the length field of the full multiplier.

The transfer is performed one byte at a time, through the serial adder, starting with the low-order multiplier byte. ABC, STC, and L2 are decremented by 1 for each byte transferred. The multiplier is completely transferred when the L2 count is decremented to zero. Because the first IC request (during I-Fetch) does not necessarily gate the full multiplier to AB, it may be necessary to fetch the balance of the multiplier from main storage. (This fetch occurs if the ABC steps to zero before L2 steps to zero.)

After exit from the left-adjust sequence, the full multiplier has been fetched and left-adjusted to ST. Note that the original ST contents (multiplicand) have been destroyed except for the low-order multiplicand byte, which is saved during the GIS; i.e., digit placed into F(0–3) and sign-recorded by STAT F. The destroyed multiplicand bytes are later refetched from main storage, one byte at a time, as required by the multiply operation.

L2 Restoration

During transfer of the multiplier bytes from AB to ST, the L2 count in E(12–15) is decremented by 1 for each byte transferred. At the completion of the left-adjust sequence, L2 has been decremented to zero. The initial L2 count, saved in F(4–7) during the GIS, is now restored to E(12–15). The L2 count will be required by the subsequent multiply sequence.

Multiplier Right-4 Shift to Drop Sign

In the multiply operation to follow, product bytes are developed by adding the entire multiplier the number of times specified by successive digits of the multiplicand. The sign of the multiplier does not enter into the

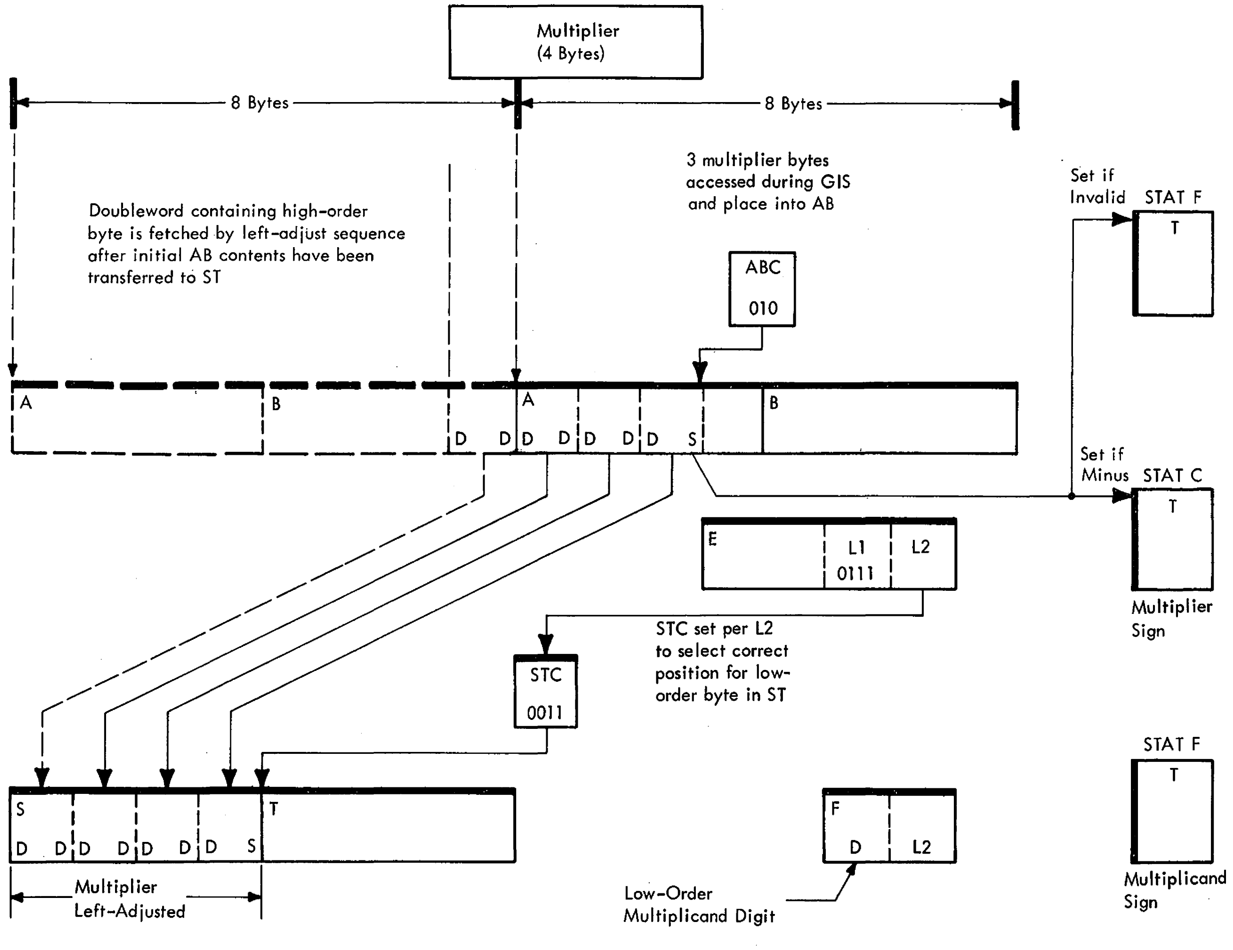
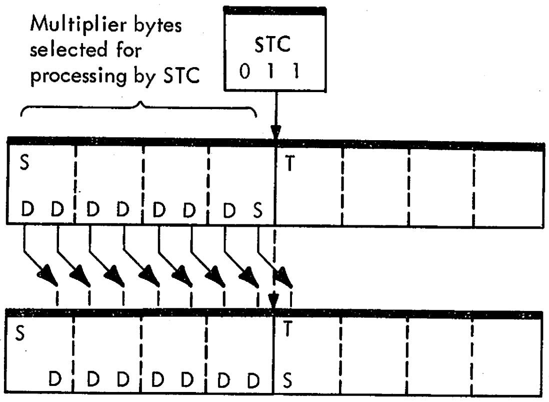

Multiplier
(4 Bytes)
8 Bytes
8 Bytes
3 multiplier bytes accessed during GIS and place into AB
Doubleword containing high-order byte is fetched by left-adjust sequence after initial AB contents have been transferred to ST
ABC
010
Set if Invalid
STAT F
T
A
B
D D D D D D D S
A
B
Set if Minus
STAT C
T
E
L1
0111
L2
Multiplier Sign
STC set per L2 to select correct position for low-order byte in ST
STC
0011
STAT F
T
S
D D D D D D D S
T
F
D
L2
Multiplier Left-Adjusted
Low-Order Multiplicand Digit
Multiplicand Sign

Figure 3-18. Data Handling During Multiplier Left-Adjust Sequence

repetitive addition sequence and must be discarded. The sign is discarded by shifting the multiplier in ST 4 bit positions (one digit) to the right as illustrated below. This action places the sign beyond the rightmost multiplier byte selected by the STC for subsequent computation.



Sign Handling

A test of STAT F and STAT C is made to establish the product sign algebraically:

1. Signs alike (both STATS set or reset) – set sign plus.
2. Signs not alike (one STAT set and the other reset) – set sign minus.

Upon establishing the correct product sign, it is placed into F(4–7).

Basic Multiply Add or Subtract Sequence for Left Digit

This sequence processes the digit in the left portion of the multiplicand byte. (The low-order byte of the multiplicand always contains the digit in the left portion and the sign in the right portion.) The entire multiplier (in ST) is added or subtracted the number of times specified by the left digit of the multiplicand saved in F(0–3). An add sequence is performed if the digit in F(0–3) is 4 or less; a subtract sequence, if 5 or greater. A data program interruption occurs prior to a storage cycle if an invalid multiplicand or multiplier digit or sign is detected. The PP resulting from the add or subtract sequence replaces the multiplicand in ST.

Product Byte Store

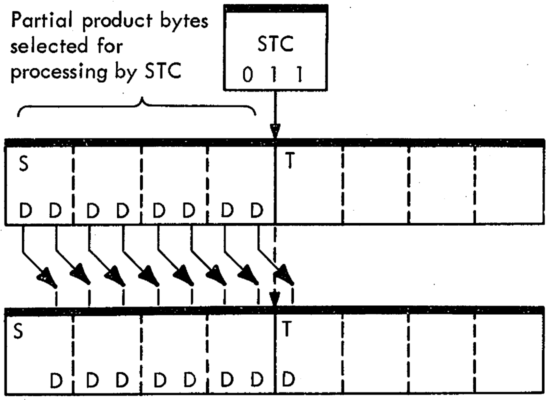
The product is stored into main storage one byte at a time. After exit from the left-digit sequence, one complete byte of product has been developed and must be stored. If the exit is made for the first time, this byte consists of the product sign (in F) and the low-order digit of PP (in ST). All product bytes generated thereafter consist of two digits: one (in F) has been saved from a previous PP developed in the right digit sequence, and the second is the low-order digit of a new PP (in ST) obtained in the left-digit sequence.

Multiplicand Request

A request from D is issued for the next byte of the multiplicand in main storage.

Partial Product Right-4 Shift to Drop Digit

The low-order digit of PP has been stored as a product digit and must not enter into subsequent computation. The digit is discarded by shifting the PP in ST four bit positions to the right. This action places the digit beyond the rightmost PP byte selected by the STC for computation of the next product digit.



L1 = L2

This test establishes whether all digits of the multiplicand have been processed. At the start of the multiply operation, the total field length specified by L1 includes a zero field equal in size to the multiplier plus the effective field of the multiplicand:

L1 = L2 + number of effective multiplicand bytes.

Because L1 is decremented once after each effective multiplicand byte is processed, all the effective multiplicand bytes have been processed when L1 equals L2.

Complete Multiplicand Byte Fetch

If L1 is not equal to L2, the multiply sequence is continued. The next byte of the multiplicand (requested earlier) is selected from the SDBO and placed into F. Control is then transferred to the add or subtract sequence for the right digit of the multiplicand.

Basic Multiply Add or Subtract Sequence for Right Digit, and Shift Right-4 Sequence

This sequence processes the digit in the right portion of the multiplicand byte. The entire multiplier is added to (or subtracted from) the PP in ST. The number of add or subtract operations is controlled by the right digit of the multiplicand contained in F(4–7). After a new PP has been developed in ST, its low-order digit replaces the right digit of the multiplicand in F(4–7). The PP is then shifted four bit positions to the right to drop the low-order digit, and an entry is made to the left-digit sequence to process the next multiplicand digit contained in F(0–3).

Multiplicand Zero Test and Partial Product Store

When L1 equals L2, all the effective digits of the multiplicand have been processed. The remaining multiplicand bytes are fetched from main storage and tested for zero. Detection of a nonzero digit results in an interruption. After the zero test is completed, the PP is stored as the high-order product into main storage and the instruction is ended.

Detailed Description

- STAT E is set if digit or sign is invalid.
- STAT A is set if digit is not zero.
- STAT G is set if multiplier is zero.
- STAT H is set to generate hot carry.
- STAT D is set to add 1 to next digit.

Sheet 2 of Diagram 5-305 is a detailed flowchart of the multiply microprogram. This flowchart is an expanded version of the overall flowchart, showing the data handling used in the various subroutines of the Multiply instruction. For the most part, this data handling is straightforward and requires no explanation. Those areas in need of clarification are discussed in the following subparagraphs.

General Initialization Sequence

This sequence shares a common microprogram with the Divide instruction. An appropriate branch is taken to enter either the divide or the multiply sequence.

Multiplier Left-Adjust Sequence

The ABC has been set to select the low-order multiplier byte in AB. The STC is now set per L2, E(12–15). The transfer is performed one byte at a time via the serial adder. As each byte is gated to the serial adder, it is tested for nonzero value and for invalid digits. STAT E is set upon detection of an invalid digit or sign, and STAT A is set upon detection of a nonzero digit. If upon completion of the left-adjust transfer STAT A remains reset, the multiplier value is zero.

Multiplier Right-4 Shift and L2 Restoration

The multiplier is shifted four bit positions to the right and transferred from ST to AB. L2 is transferred from F(4–7) to E(12–15). Both actions are performed in parallel. As the high-order multiplier bytes are gated from S to PAA and the right-4 shift is initiated, the L2 count is transferred from F(4–7), via the serial adder, to S(28–31). After the right-4 shift has been performed through the parallel adder, the L2 count is gated from S to PAA and the zero count in E(12–15) is gated to PAB. The net result (original L2 count) is gated from PAL to E(12–15).

Sign Handling

STAT G is set if STAT A has not been set during the preceding sequence. This step is taken to indicate a zero multiplier condition which requires special action.

Basic Multiply Add or Subtract Sequence

To perform a branch on the value of the multiplicand digit, the digit must be in SAL(4–7). This requirement is dictated by the 'W=(1–15)' micro-order which samples SAL(4–7). For this reason, the contents of F are cross-gated through the serial adder and placed back into F. SAL(4–7) is then examined for the following values:

1. SAL(4–7) = 0
   No addition cycles are required.
2. SAL(4–7) = 1 through 4
   The multiplier in AB is added to the PP in ST the number of times specified by the digit value.
3. SAL(4–7) = 5 through 9
   The multiplier in AB is subtracted from the PP in ST the number of times specified by the 10's complement of the digit value (10 minus the digit value). STAT H is set to supply a hot carry for the subtract sequence. STAT D is set to add a 1 to the next digit of the multiplicand (equivalent to adding the multiplicand 10 times).
4. SAL(4–7) = invalid digit
   The definition of an invalid digit is dependent on whether the digit to be processed is the first digit of the multiplicand; i.e., the digit immediately following the sign. If it is the first digit, then any value in the range of 10 through 15 is considered invalid and sets the interrupt trigger. After the first digit has been processed, a value of 10 is permissible in SAL(4–7), provided that it was formed by an original value of 9 to which a 1 has been added because STAT D was set. Under these conditions, the value of 10 does not set the interrupt trigger, no addition cycles are required, and a carry is propagated to the next digit by setting STAT D.

The multiplier-to-PP addition or subtraction is done one byte at a time in the serial adder, with the AB byte gated true +6 if adding and complement if subtracting. The ABC and STC are both initially set to the L2 value and are decremented by 1 each time a byte is processed. When the ABC count is stepped to 000, F(4–7) is examined to determine whether further additions or subtractions are necessary. If so, the STC and ABC are again set to the L2 value, F(4–7) is incremented if subtracting or decremented if adding, and the multiplier is again added to or subtracted from the PP. The micro-order which steps the digit in F(4–7) is executed after the digit has been examined to determine whether further add or subtract cycles are required. For this reason, when a branch on F(4–7) is being made, a value of 1 when adding or of 9 when subtracting indicates that the multiplicand digit is completely processed. The low-order digit of the PP in ST is the product digit developed.

Product Byte Store, PP Right-4 Shift to Drop Digit, Multiplicand Request

These three functions are accomplished in parallel fashion. After initiating the store operation, control is transferred to the shift-right-4 sequence. When the ST contents have been temporarily transferred, the store operation is resumed; the product byte is cross-gated, transferred to ST, and stored into main storage per the D-address. Thereafter, the microprogram requests the next multiplicand byte from main storage and simultaneously completes the right-4 shift.

As illustrated in Figure 3-19, the PP is shifted right-4 via the parallel adder. This shifting is done in several steps, with the LSWR being used as temporary storage for the operand. Upon completion of the right-4 shift, B(64–67) is normally inserted as the high-order S digit. B(64–67) was previously set to 0 if the value in F(4–7) was less than 5, or to 9 if F(4–7) was 5 or greater, for then the PP was in 10's complement form. An exception is made when STAT G is set, indicating an all-zero multiplier. In this case, B(64–67) is not inserted into high-order PP because it should always be zero.

Complete Multiplicand Byte Fetch

If there are more multiplicand digits to be processed (L1 ≠ L2), the contents of T are temporarily transferred to the LSWR and either the left or the right half of the operand is gated from the SDBO to T. The next byte of the multiplicand is then selected per the STC and transferred to F. (Note that if the left half word has been gated to T, the high-order STC bit is forced to 1, because, otherwise, the STC would select a byte from S.)

Basic Multiply Add or Subtract Sequence for Right Digit, and Shift Right-4 Sequence

The next digit of the multiplicand in F(4–7) is examined, STAT's D and H are set or reset as required, a 0 or 9 is placed into B(64–67), and the appropriate add or subtract loop is entered. After exit from the add or subtract loop, the low-order digit of the resulting PP is saved in F(4–7). A right-4 shift is then performed on the PP in ST so that the low-order digit is dropped. At the completion of the right-4 shift, the left-digit sequence is resumed. As explained previously,' the contents of F are cross-gated and the next digit of the multiplicand is sampled from SAL(4–7).

Multiplicand Zero Test and Partial Product Store

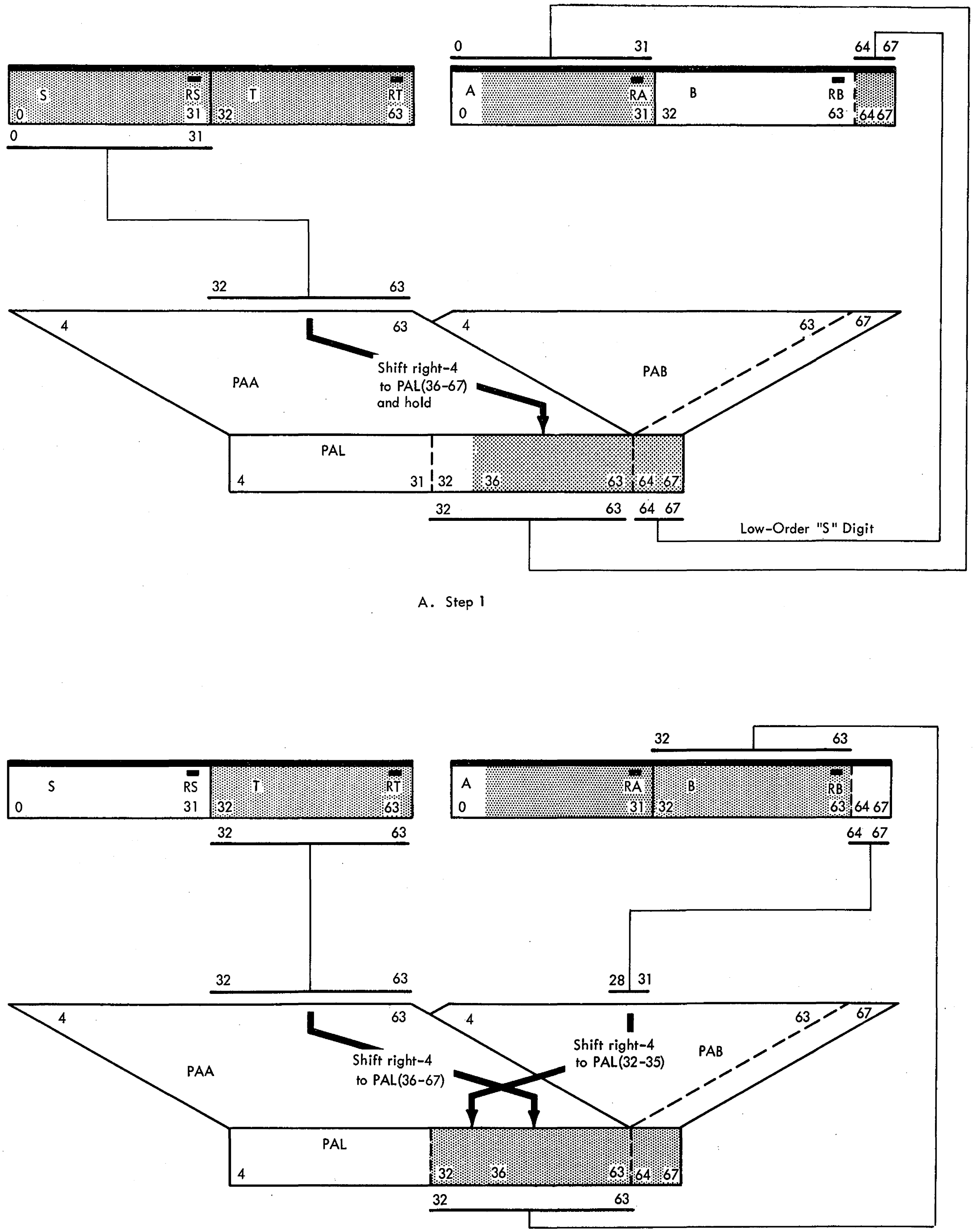
An entry to this routine is made from the left-digit sequence. By operand definition, the remaining high-order multiplicand bytes should all be zeros. The PP in ST is the high-order product and must be stored into the high-order portion of the initial multiplicand field. However, if STAT D is set at this time, the multiplier must be added to the PP once more. After this has been done, the contents of ST are transferred to AB and the high-order multiplicand bytes are fetched to ST (per the D-address). The STC is set per D(21–23) to designate the first high-order multiplicand byte to be tested for zero. The ABC is set per L2 to designate the first high-order PP byte in AB.

The selected multiplicand byte in ST is tested for zero; then the selected PP byte in AB is transferred via the serial adder to ST, and the corresponding mark trigger is set. ABC, STC, and L1 are decremented by 1 for each byte transferred. If a nonzero byte is detected in the high-order field of the multiplicand, the interrupt trigger is set and the instruction is ended.

The AB-to-ST byte transfer is continued until L1 or STC is stepped to zero, at which time the ST contents are stored into main storage. If the STC has been stepped to zero (L1 ≠ 1111), the next high-order bytes of the multiplicand are fetched to ST and the sequence is resumed. If L1 has been stepped to zero (L1 = 1111), the instruction is ended.

DIVIDE, DP (FD)

- Divide 1st operand (in storage) by 2nd operand (in storage) and place result into 1st operand location (quotient is leftmost in 1st operand location; remainder, rightmost).

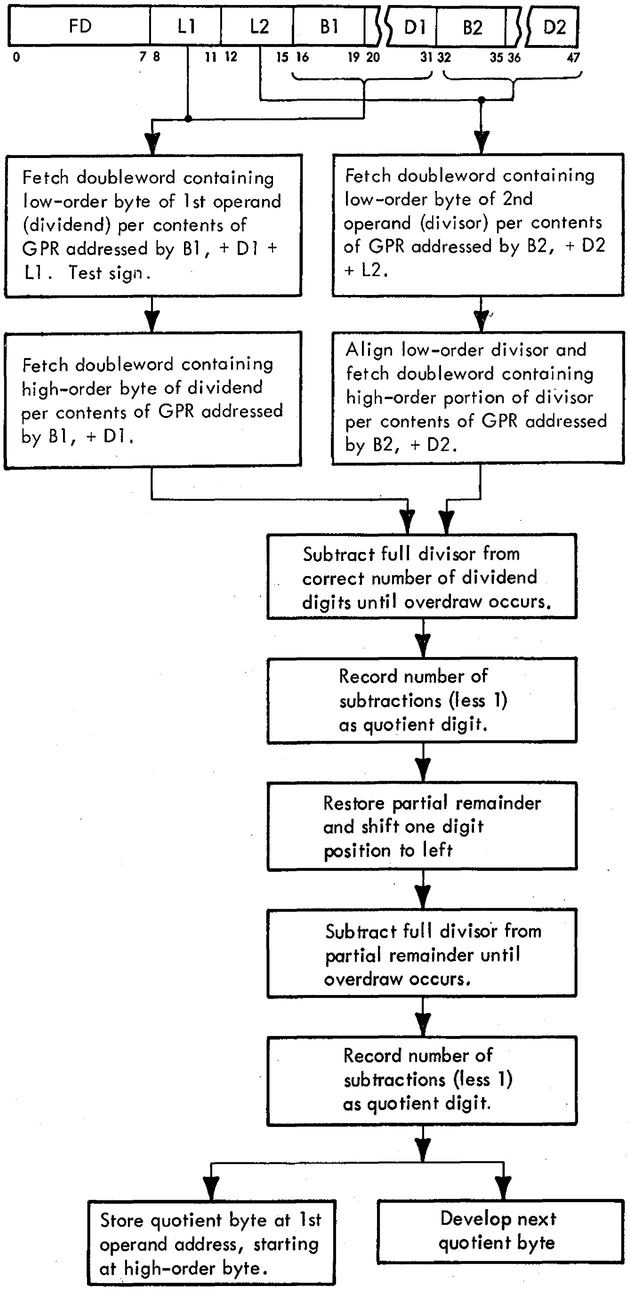

S
RS
T
RT
A
RA
B
RB
PAA
PAB
PAL
Shift right-4 to PAL(36-67) and hold
Low-Order "S" Digit
Shift right-4 to PAL(36-67)
Shift right-4 to PAL(32-35)
0
31
32
63
64
67
4
36
28

A. Step 1

B. Step 2

Figure 3-19. Data Flow for Right-4 Shift of ST to AB, Decimal Multiply

- SS format:


FD
L1
L2
B1
D1
B2
D2
0
7 8
11 12
15 16
19 20
31 32
35 36
47
Fetch doubleword containing low-order byte of 1st operand (dividend) per contents of GPR addressed by B1, + D1 + L1. Test sign.
Fetch doubleword containing low-order byte of 2nd operand (divisor) per contents of GPR addressed by B2, + D2 + L2.
Fetch doubleword containing high-order byte of dividend per contents of GPR addressed by B1, + D1.
Align low-order divisor and fetch doubleword containing high-order portion of divisor per contents of GPR addressed by B2, + D2.
Subtract full divisor from correct number of dividend digits until overdraw occurs.
Record number of subtractions (less 1) as quotient digit.
Restore partial remainder and shift one digit position to left
Subtract full divisor from partial remainder until overdraw occurs.
Record number of subtractions (less 1) as quotient digit.
Store quotient byte at 1st operand address, starting at high-order byte.
Develop next quotient byte

- Maximum dividend field (1st operand) is 16 bytes.
- Maximum divisor field (2nd operand) is 8 bytes.
- L2 specifies byte length for divisor and remainder.
- L2 > 7 or L2 ≥ L1 causes specification program interruption.
- Division accomplished by repetitive subtraction.
- Dividend field must initially contain sufficient number of high-order zeros to make possible storing of quotient and remainder.

The Decimal Divide instruction replaces the dividend (1st operand) with the quotient and the remainder. To be able to store the quotient and the remainder into the dividend field at all times, several restrictions are imposed on the initial size of the dividend and the divisor (Figure 3–20).

The maximum dividend field is 16 bytes long. It is eventually replaced by the quotient, which is stored leftmost in the field, and by the remainder, which is stored rightmost. The size of the remainder is equal to the initial divisor size and is therefore predefined by length code L2. Because the minimum remainder size is 1 byte (L2 = 0), the maximum quotient size is limited to 15 bytes. By definition, the size of the divisor (and remainder) cannot exceed 8 bytes. A divisor greater than 8 bytes, or in excess of the dividend, is recognized as a specification error; the instruction is suppressed and a specification program interruption occurs.

The operand signs are tested for validity before instruction execution, and, if either sign is invalid, a data program interruption is taken before the contents of main storage are altered.

To make sure that the quotient and remainder will fit into the destination field, the magnitudes of the dividend and the divisor are compared before entering the divide sequence. This comparison, called "divide check" or "trial subtraction," yields the number of quotient digits that will result if division is carried out. If the predicted quotient is larger than that allowed, the instruction is suppressed and a decimal divide program interruption occurs. For this reason, an overflow condition cannot exist upon execution of a Divide instruction.

The dividend, divisor, quotient, and remainder are all signed integers, right-aligned in their fields. The sign of the quotient is determined algebraically from the dividend and divisor signs. The sign of the remainder is the same as the sign of the dividend. These rules hold true even when the quotient or the remainder is zero.

The divide operation is executed in much the same manner as in manual arithmetic. First, the divisor is properly aligned with the high-order dividend; then, by repeatedly subtracting the divisor from the dividend and counting the number of successful subtractions, the high-order quotient digit is determined. The partial remainder resulting from the last successful subtraction is shifted one digit position to the left, and the next lower-order dividend digit is inserted at the low-order end of the partial remainder. To obtain the next quotient digit, the divisor is again subtracted from the partial remainder. This sequence is repeated until all dividend

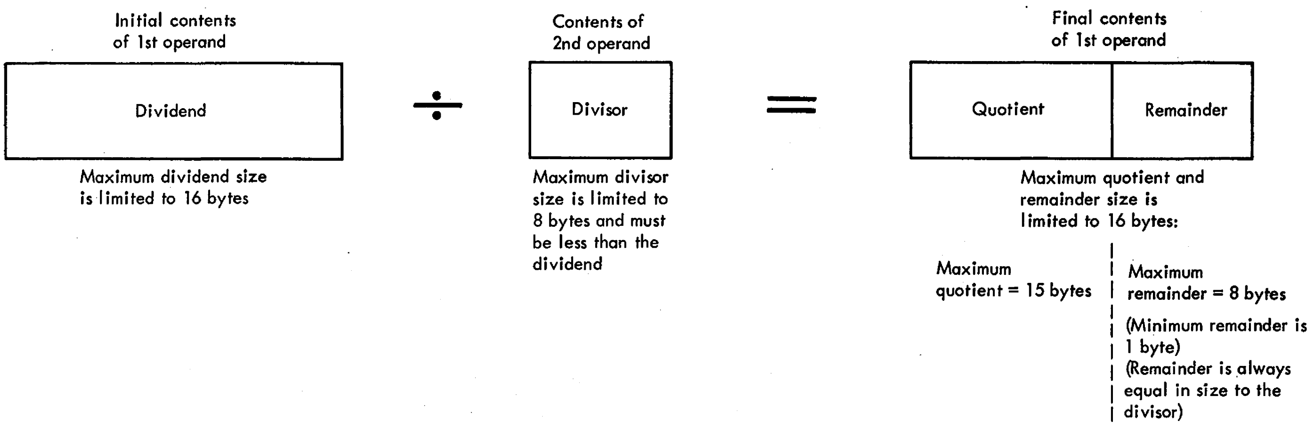

Initial contents of 1st operand
Dividend
Maximum dividend size is limited to 16 bytes
Contents of 2nd operand
Divisor
Maximum divisor size is limited to 8 bytes and must be less than the dividend
Final contents of 1st operand
Quotient
Remainder
Maximum quotient and remainder size is limited to 16 bytes:
Maximum quotient = 15 bytes
Maximum remainder = 8 bytes
(Minimum remainder is 1 byte)
(Remainder is always equal in size to the divisor)

Figure 3-20. Operand Specifications for Decimal Divide

digits have been processed. The remainder resulting from the final successful subtraction is given the sign of the dividend and stored into the low-order end of the dividend field.

Figure 3-21 illustrates a typical repetitive subtract sequence used to accomplish division. Initially, a sufficient number of high-order dividend digits must be selected to perform the first successful subtraction. Successful subtractions of the divisor from the dividend occur until the partial remainder is overdrawn. The divisor is then added back once to restore the correct partial remainder. At the same time, the quotient digit is decremented by 1 to compensate for the overdraw. As each dividend byte is processed, the length code of the dividend (L1) is reduced by 1 and compared with the length code of the divisor (L2). Because the size of the remainder is also defined by length code L2, the condition of L1 equal to L2 indicates that all the effective bytes of the dividend have been processed, and the remainder is to be stored into the rest of the destination field. Note that, to be able to fit the quotient and the remainder into the destination field, this field must initially contain high-order zeros. A data program interruption occurs if the dividend does not have at least one leading zero.

Following are general and detailed descriptions of the divide microprogram. The general description outlines the overall structure of the microprogram, enumerates its major functional steps and subroutines, and explains their relationship to the overall operation. The detailed description analyzes each sequence individually, making specific references to the register-to-register data transfer in the CPU.

General Description

Upon entering the divide microprogram, the following actions have been performed by SS I-Fetch:

1. A D-request has been issued for the doubleword containing the low-order dividend byte.
2. The low-order dividend address has been placed into D.
3. The contents of the IC have been transferred to the LSWR, and the high-order divisor address has been placed into the IC.

An overall flowchart of the divide microprogram and the general data path used for its execution are shown in Sheet 1 of Diagram 5-306, FEMDM. The major subroutines and functional steps, shown in the figure, are explained in the following subparagraphs. Additional simplified diagrams are provided as an aid in visualizing the data handling performed. For the most part, these diagrams do not show the gates and data paths used in the CPU, but are intended solely to convey how the divide algorithm is implemented. For purposes of illustration, a nine-byte dividend and a three-byte divisor are assumed in these diagrams.

General Initialization Sequence

This sequence shares a common microprogram with the Multiply instruction. An appropriate branch is taken to enter either the divide or the multiply sequence. GIS gates the low-order dividend from the SDBO to ST, increments the IC by L2 to address the low-order byte of the divisor, issues an IC request for the divisor, and gates the divisor to AB. The ABC is set to the low-order divisor byte in AB (Figure 3–22), and the STC is set per L2. STAT E is set if the dividend sign is invalid, and D is decremented by L1 to the address of the high-order byte of the dividend.

Specification Test

This test verifies that the length codes for both operands in the instruction are correctly specified; i.e., L2 specifies eight bytes or less and is smaller than L1.

Incorrect Specification

Detection of an invalid specification forces a specification program interruption. The instruction address is restored to the LSWR, and the instruction is ended.

Divide (+1315) by (-57) to obtain a quotient of (-23) and a remainder of (+4).

Execution:

Low-order byte is addressed to check for valid sign, then D is decremented by L1 to address high-order byte.

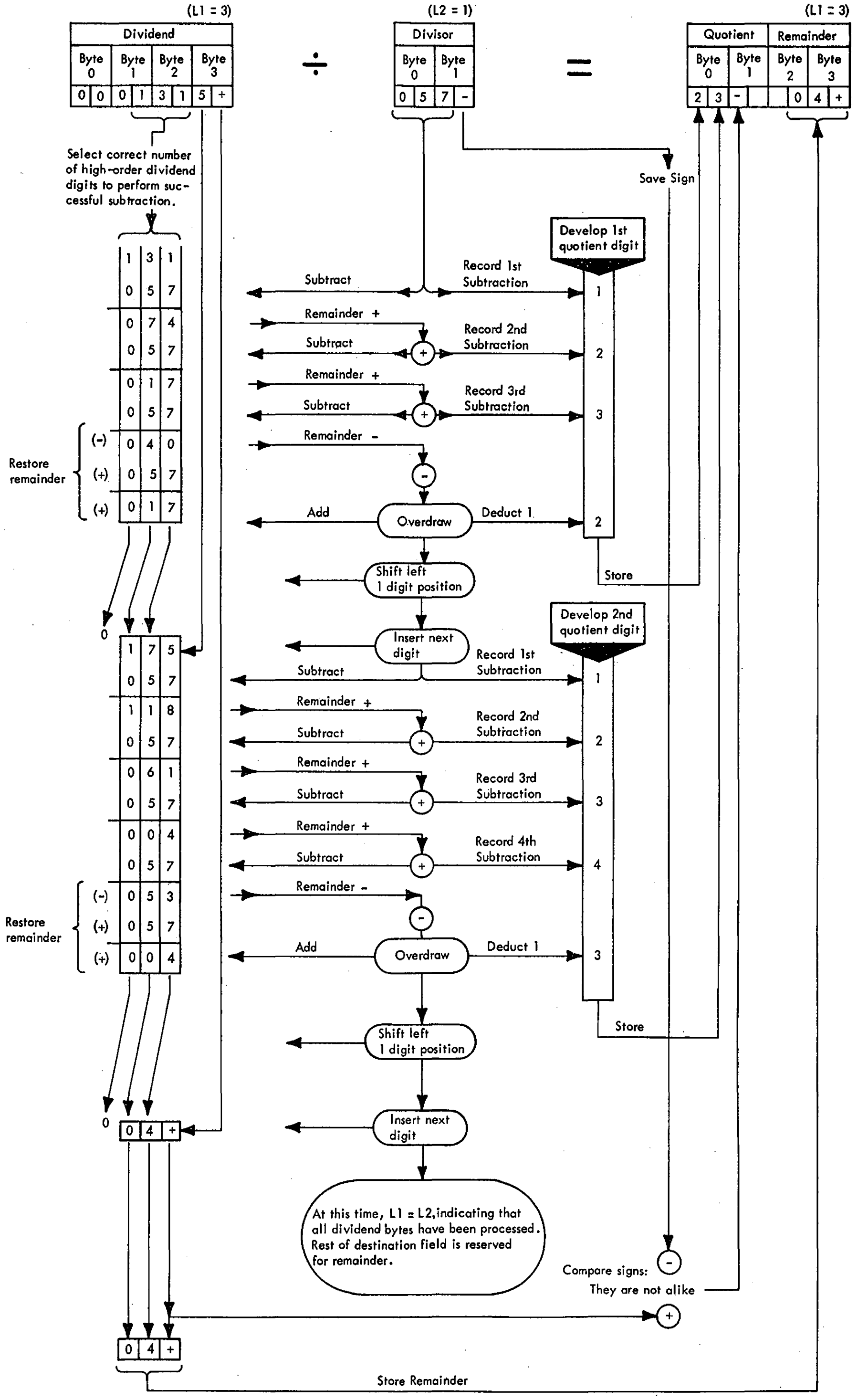


Figure 3-21. Example of a Typical Divide Sequence

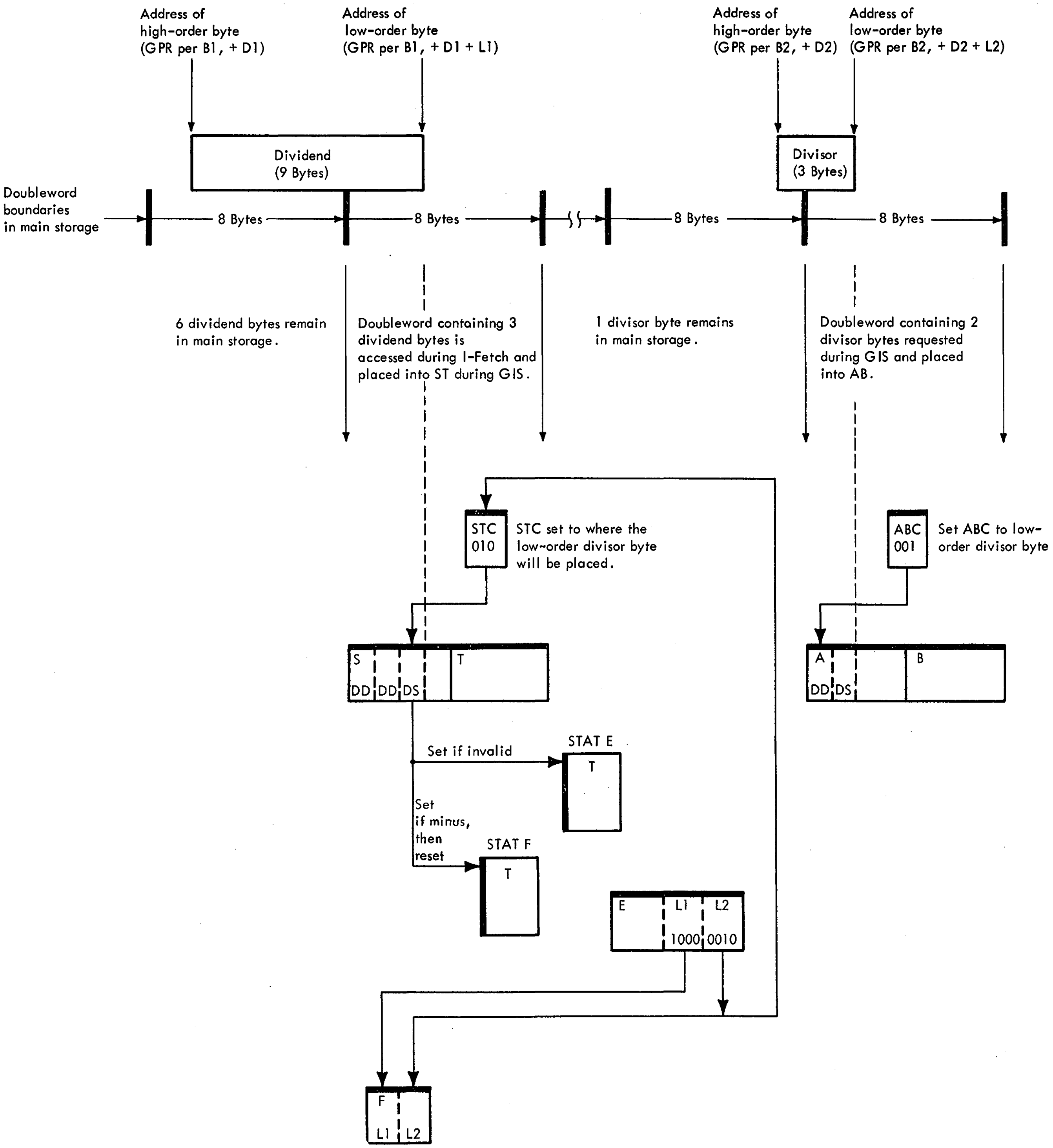

Address of high-order byte (GPR per B1, + D1)
Address of low-order byte (GPR per B1, + D1 + L1)
Address of high-order byte (GPR per B2, + D2)
Address of low-order byte (GPR per B2, + D2 + L2)
Dividend (9 Bytes)
Divisor (3 Bytes)
Doubleword boundaries in main storage
8 Bytes
8 Bytes
8 Bytes
8 Bytes
6 dividend bytes remain in main storage.
Doubleword containing 3 dividend bytes is accessed during I-Fetch and placed into ST during GIS.
1 divisor byte remains in main storage.
Doubleword containing 2 divisor bytes requested during GIS and placed into AB.
STC 010
STC set to where the low-order divisor byte will be placed.
ABC 001
Set ABC to low-order divisor byte
S
DD DD DS
T
A
DD DS
B
Set if invalid
STAT E
T
Set if minus, then reset
STAT F
T
E
L1 L2
1000 0010
F
L1 L2

Figure 3-22. Data Handling During GIS of Decimal Divide

Divisor Left-Adjust Sequence

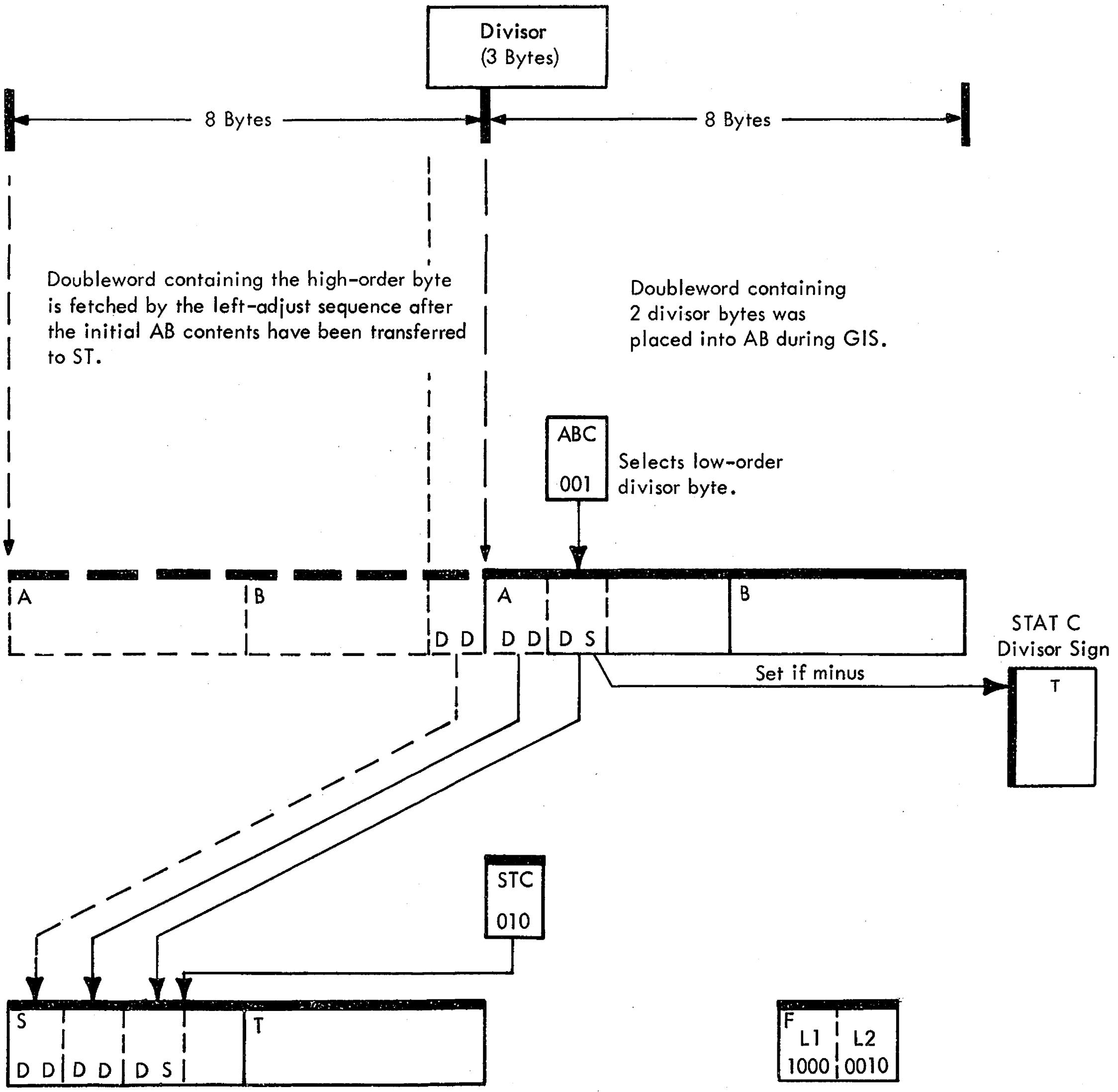
The divisor bytes are transferred from AB to ST in such a manner that the high-order divisor byte occupies the leftmost byte in ST. STAT C is set if the divisor sign is negative.

The left-adjust transfer is initiated by setting the STC per L2 (a function performed during the GIS). Because the maximum divisor length is limited to eight bytes, only three of the four bit positions in L2 are needed to effectively specify the length code; i.e., the count in L2 may range from a minimum of 0000 (for one byte) to a maximum of 0111 (for eight bytes). Setting the STC per L2 automatically selects, according to the divisor size, the correct ST position for the low-order byte of the divisor; the number of bytes to the left of the selected ST position corresponds to the length field of the full divisor (Figure 3–23). The actual transfer is performed one byte at a time, through the serial adder, starting with the low-order divisor byte. ABC, STC, and L2 are decremented by 1 for each byte transferred. The divisor is completely transferred when L2 is decremented to zero. Because the first IC request (during I-Fetch) does not necessarily access the full divisor to AB, it may be necessary to fetch the balance of the divisor from main storage. (This fetch occurs if the ABC steps to zero before L2 steps to zero.)

After exit from the Divisor Left-Adjust sequence, the full divisor has been fetched and left-adjusted to ST.

Dividend Fetch and'Left-Adjust Sequence

This sequence fetches a sufficient number of high-order dividend bytes to perform a trial subtraction of the divisor from the dividend. The full divisor is subtracted once from the high-order dividend. Because the maximum divisor size is eight bytes, eight high-order dividend bytes are required for trial subtraction. If the dividend is eight bytes or less, it is completely fetched during this



Divisor
(3 Bytes)
8 Bytes
8 Bytes
Doubleword containing the high-order byte is fetched by the left-adjust sequence after the initial AB contents have been transferred to ST.
Doubleword containing 2 divisor bytes was placed into AB during GIS.
ABC
001
Selects low-order divisor byte.
A
B
D D
A
D D
D S
B
STAT C
Divisor Sign
Set if minus
T
STC
010
S
D D
D D
D S
T
F
L1
1000
L2
0010

Figure 3-23. Data Handling During Divisor Left-Adjust Sequence

sequence; if greater than eight bytes, only the first eight high-order bytes are fetched. The dividend is fetched to AB and then transferred to ST in such a manner that the high-order byte occupies the leftmost byte in ST.

Upon entry into this sequence, ST is assumed to be completely occupied by the divisor. (If the divisor is four bytes or less, it is confined solely to S; if greater than four bytes, the divisor extends into T.) Because left-adjustment of the dividend requires the use of ST, the divisor must be transferred from ST; S is gated to the parallel adder and held in PAL, and T is stored into the LSWR (Figure 3–24).†

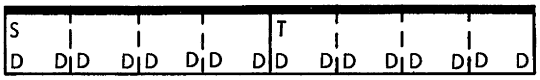
A request per the high-order dividend address is issued from D. Upon arrival of the dividend from main storage, the SDBO is gated to AB. The ABC is set per D(21–23) to select the high-order dividend byte. The left-adjust transfer is initiated by setting STC to 000, thus selecting the leftmost byte in ST. The dividend bytes are then transferred to ST, starting with the high-order byte. (The actual transfer is performed one byte at a time through the serial adder.) The ABC and STC are incremented, and L1 is decremented by 1 for each byte transferred. If L1 steps to zero before the ABC or STC steps to 7, the full dividend has been fetched and left-adjusted to ST. Because the first request does not necessarily access eight bytes of dividend to AB, it may be necessary to fetch additional dividend bytes from main storage. This fetch occurs if the ABC steps to 7 before the STC steps to 7 or L1 steps to zero.

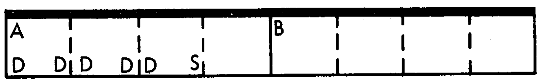
Restore L1 and L2 to E

Left-adjustment of the divisor and dividend has decremented L2 and L1 to zero. The initial L2 and L1 counts, saved in F during GIS, are now restored to E(8–15). These counts are required by the subsequent divide sequence.

Assemble Divisor in AB and Dividend in ST

The divisor in PAL and LSWR is restored to AB. Upon completion of this function, both operands are left-aligned: the dividend is in ST, and the divisor is in AB.



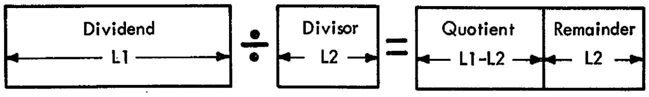


Trial Subtraction

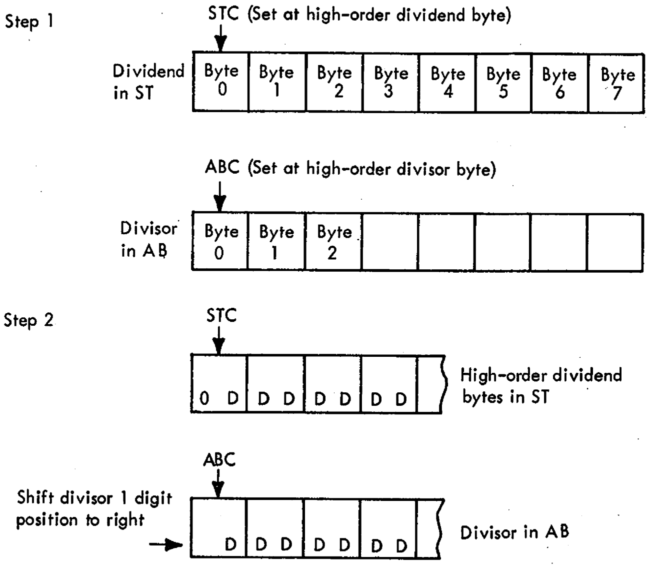
The divisor bytes in AB are subtracted from an equivalent number of high-order dividend bytes in ST. The remainder is then examined to establish whether the divide operation to follow will generate a result (quotient plus remainder) that will fit into the destination field. A negative remainder indicates that the destination field specified in the instruction is sufficiently large to accommodate the result. A positive remainder, however, indicates that the result cannot fit into the destination field, and a decimal divide program interruption occurs.

How prediction by trial subtraction is possible may be understood from the following considerations:

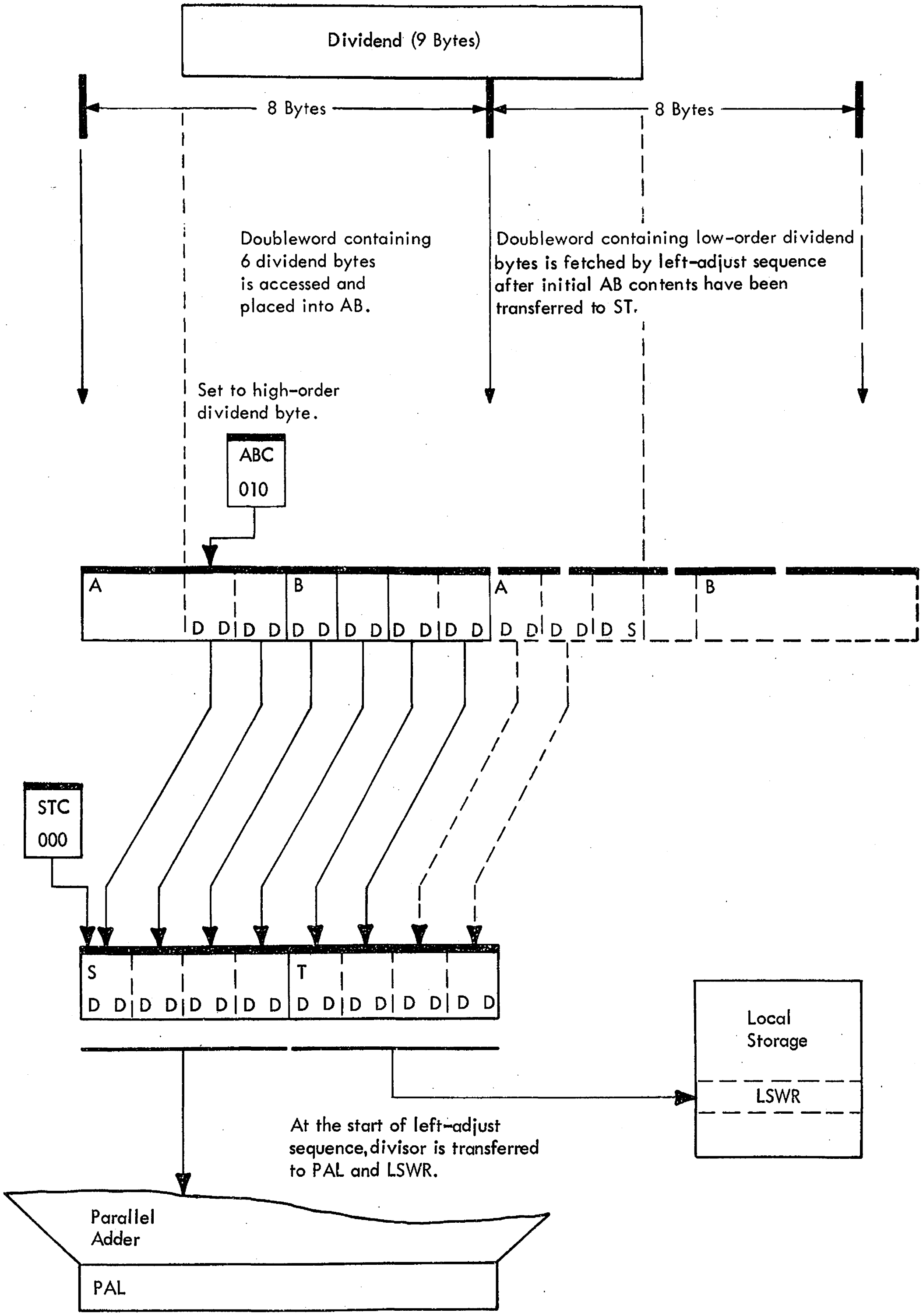
1. By definition, the dividend is at least one order higher than the divisor. The high-order digit position in the dividend is always zero.
2. The length code of the divisor (L2) is also the length code for the remainder. Consequently, the maximum number of quotient bytes that will fit into the destination field is equal to L1 minus L2. By operand definition, the difference of L1 minus L2 may range from a minimum of one byte to a maximum of eight bytes.



3. To perform the trial subtraction, the high-order divisor digit is aligned with the high-order digit of the dividend. This is performed in two steps: (1) the high-order divisor byte is aligned with the high-order byte of the dividend, and (2) because by definition the high-order digit position in the dividend is always zero, the divisor is shifted right one digit position to align the significant digits in both operands.



† The '→ HOLD' micro-order is issued on each cycle of the left-adjust sequence to hold the S contents in PAL.



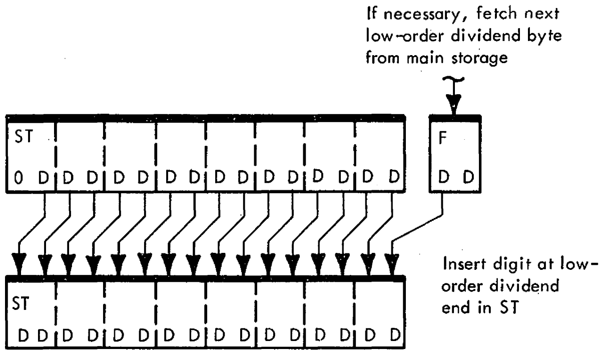
Dividend (9 Bytes)
8 Bytes
8 Bytes
Doubleword containing 6 dividend bytes is accessed and placed into AB.
Doubleword containing low-order dividend bytes is fetched by left-adjust sequence after initial AB contents have been transferred to ST.
Set to high-order dividend byte.
ABC
010
A
B
D D D D D D D D D D D D
A
D D D D D S
B
STC
000
S
D D D D D D D D
T
D D D D D D D D
Local Storage
LSWR
At the start of left-adjust sequence, divisor is transferred to PAL and LSWR.
Parallel Adder
PAL

Figure 3-24. Data Handling During Dividend Fetch and Left-Adjust Sequence

4. Because the dividend is at least one order higher than the divisor, alignment of the high-order divisor digit with that of the dividend is equivalent to multiplying the divisor at least 10 times; if the dividend is one order higher, the divisor is multiplied 10 times; if two orders higher, 100 times; if three orders higher, 1000 times; and so on. Thus, during the trial subtraction, a quantity at least 10 times that of the divisor is subtracted from the dividend.
5. Because the maximum number of quotient digits allowed (L1 minus L2) corresponds to the difference between the orders of magnitude in the two operands, the result of the trial subtraction must always yield a negative remainder; otherwise, the number of quotient digits that would be generated would not fit into the destination field.

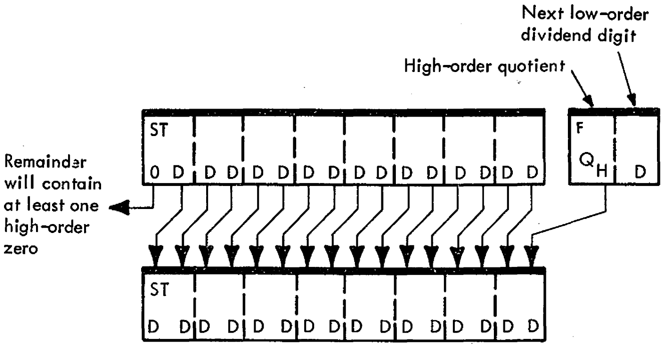
Shift Dividend One Digit to Left

The dividend is shifted one digit to the left to allow a successful subtraction of the divisor from the dividend. (To develop the quotient digit, the divisor must be repeatedly subtracted from the dividend until a negative remainder occurs.) Upon initiating the left-4 shift, a test is made to establish whether an additional low-order dividend digit is required for generation of the first quotient digit. If required, the next low-order dividend byte is fetched from main storage and placed into F. The digit is selected from F(0–3) and inserted at the low-order end of the dividend in ST.


If necessary, fetch next low-order dividend byte from main storage
ST
0 D D D D D D D D D D D D D D D
F
D D
Insert digit at low-order dividend end in ST
ST
D D D D D D D D D D D D D D D D

Generate Quotient and Left-Digit Sequence

The divisor is repeatedly subtracted from the dividend until an overdraw occurs; i.e., a negative remainder is obtained. The number of successful subtractions is recorded and, after the last successful subtraction, becomes the high-order quotient digit. This digit is tested for validity and then inserted into F(0–3) by the left-digit sequence. This sequence also shifts the partial remainder (in ST) one digit to the left and inserts the next low-order dividend digit into the low-order end of ST.


Next low-order dividend digit
High-order quotient
ST
0 D D D D D D D D D D D D D D D
F
Q_H D
Remainder will contain at least one high-order zero
ST
D D D D D D D D D D D D D D D D

Correct Low-Order Remainder Byte

In certain cases, the low-order remainder byte in ST must be corrected. The need for correction will become apparent when the Divide instruction is analyzed in detail. (See "Detailed Description" below.)

Generate Next Quotient Digit and Right Digit Sequence

After exit from the left-digit sequence, the operand length codes (L1 and L2) are compared to establish whether the last byte of the quotient is being processed. If L1 equals L2, the correct quotient sign is inserted into F(4–7). The last quotient byte (in F) is stored; then, the partial remainder (in ST) is stored into the low-order destination field as the final remainder.

If L1 does not equal L2, the next quotient digit is generated and placed into F(4–7) by the right-digit sequence. At the completion of this sequence, one complete byte of quotient is contained in F. This byte is stored into main storage, and the sequence for the left digit of the next quotient byte is started.

Detailed Description

- STAT A is set to indicate nonzero divisor.
- STAT C is set if divisor is negative.
- STAT D is set if dividend is less than eight bytes.
- STAT E is set if digit or sign is invalid.
- STAT F is set if dividend is negative.
- STAT G is first set if divisor is five bytes or greater. STAT G is then set again to enter left-digit sequence.
- STAT H is set to generate hot carry for subtract sequence.

Sheet 2 of Diagram 5-306 is a detailed flowchart of the divide microprogram. This figure is an expanded version of the overall flowchart, showing the data handling used in the various subroutines of the Divide instruction. The major subroutines and those areas in need of clarification are explained in the following subparagraphs.

General Initialization Sequence

This sequence shares a common microprogram with the Multiply instruction. An appropriate branch is taken to enter either the divide or the multiply sequence.

To test for an invalid dividend sign before the dividend is altered in main storage, the low-order dividend is accessed by a D request during SS I-Fetch. At the start of GIS, the doubleword containing the low-order byte of the dividend is gated to ST. The IC is incremented by L2 to address the low-order divisor byte, and the low-order divisor is accessed by an IC request. The contents of D are transferred to the STC, length codes L1 and L2 are transferred to F, the ABC is set to 7, and the IC is decremented by 8 to address the next doubleword of the divisor.

The dividend sign is tested, and STAT E is set if the sign is invalid. STAT F is set if the dividend sign is negative, but is immediately reset. D contains the address of the low-order byte of the dividend (contents of GPR addressed by B1, + D1 + L1). Because the divide operation begins at the high-order byte of the dividend, D is decremented by L1 to address the high-order byte of the dividend (contents of GPR addressed by B1, + D1). The STC is set per L2.

A test of L2 (contained in the STC) is performed to establish the byte size of the divisor. STAT G is set if the divisor is equal to or greater than five bytes. This function increases the execution speed when assembling the divisor in AB (see "Assemble Divisor in AB and Dividend in ST"); i.e., if the divisor is four bytes or smaller, the LSWR need not be restored to B.

Divisor Left-Adjust Sequence

1. The initial STC setting selects the rightmost ST byte that contains the low-order divisor byte.
2. STAT C is set if the divisor sign is negative.
3. If the ABC steps to zero before L2 steps to zero, the remaining low-order divisor bytes are fetched from main storage.
4. The divisor digits are checked for validity, and STAT E is set if an invalid digit is detected. STAT A is set to indicate a nonzero divisor. Division by zero results in a decimal divide program interruption during trial subtraction.
5. Upon fetching the full divisor, the divisor address is no longer needed, and the instruction address is restored to the IC.

Dividend Fetch and Left Adjust Sequence

1. The divisor is shifted one digit position to the right so that its high-order digit will be aligned with that of the dividend. (The dividend is not yet available at this time.)
2. The low-order divisor word is transferred from T to the LSWR. The high-order divisor word is gated to the parallel adder and held in PAL by the '- → HOLD' micro-order.
3. The STC is set to zero to select the high-order ST byte (where the high-order dividend byte will be placed).
4. A test of the high-order L1 bit is performed to establish the byte size of the dividend. STAT D is set if the dividend is less than eight bytes. This function increased the execution speed upon exit from the trial subtraction. A branch per STAT D is made to determine whether the complete dividend has been fetched. (If STAT D is set, the full dividend has been fetched, because at least eight dividend bytes are fetched to perform the trial subtraction.)
5. If the ABC steps to zero before L1 or STC steps to zero, the D-address is incremented by 8 and a fetch of the next doubleword of the dividend is made. The destination address for the subsequent quotient bytes is then restored by subtracting 8 from D.

Assemble Divisor in AB and Dividend in ST

This function is performed in parallel with the L1 and L2 restoration sequence. The high-order divisor word is transferred from PAL to A, after which restoration of the L1 and L2 counts is started. During the restoration sequence, STAT G is tested to establish whether the full divisor has been assembled in AB. If STAT G is not set, the divisor is four bytes or less. Therefore, the full divisor was contained in PAL and has been placed into AB. In this case, the restoration sequence is completed and an exit is made to the trial subtraction routine.

If STAT G is set, the low-order portion of the divisor is contained in the LSWR and must be transferred to B before entering the trial subtraction. The LSWR contents must be transferred to B via ST, which contains the left-aligned dividend. Several execution cycles are used to transfer the LSWR contents to B without destroying the ST contents.

Trial Subtraction

The divisor is subtracted from the dividend one byte at a time. After the last subtract cycle, a branch is made on a carry condition from SAL(0). Presence of a carry indicates that the remainder is positive, and the instruction is ended. Absence of a carry indicates a negative remainder, and that the result of the divide operation will fit into the destination field.

Dividend (or Partial Remainder) Left-4 Shift

The dividend is shifted left one digit position to perform the first successful subtraction of the divisor from the dividend. Upon initiating the left-4 shift, a test (per STAT D) is made to establish whether an additional dividend digit must be inserted into the low-order end of ST. If STAT D is set (see "Dividend Fetch and Left-Adjust Sequence," step 4), all dividend bytes have been fetched from main storage and the left-4 shift is completed. If STAT D is not set, the following actions take place:

1. The D-address is incremented by 8, and the next doubleword of the dividend is requested from main storage.
2. The contents of T are temporarily transferred to the LSWR. (Upon arrival of the dividend doubleword from main storage, T is loaded with the dividend word containing the next digit to be inserted.)
3. The STC is set per D(21–23) to select the correct dividend byte in the requested doubleword.
4. The left-4 shift of the dividend is completed. The high-order dividend word is in S, and the low-order word is in the LSWR.
5. A branch per D(21) is made to establish which word in the SDBO contains the next dividend byte. The correct word is then gated from SDBO to T. [Note that, if the left SDBO word is gated to T, STC(0) is forced to 1 to select the correct byte in T.]
6. The selected dividend byte is transferred from T to F. The shifted low-order dividend word is then restored from the LSWR to T.
7. The destination address is restored by subtracting 8 from D.
8. The high-order L1 bit is tested to establish the byte size of the dividend, and STAT D is set if the dividend is less than eight bytes. This function increases the execution speed upon exit from the right-digit sequence.

Generate Quotient Sequence

1. The ABC and STC are set per L2 to select the low-order operand bytes. STAT H is set to provide a hot carry to the serial adder.
2. The selected AB byte is subtracted from the selected ST byte via the serial adder. The result is gated back to the selected ST byte, with the carry being saved in STAT H. Any invalid digit detected in the serial adder sets STAT E.
3. The ABC and STC are decremented as each byte is processed. When the ABC is stepped to zero, a 1 is added to F(4–7) and the ABC and STC are again set per L2.
4. If a serial adder carry results upon processing the high-order byte, the partial remainder in ST is positive and the divisor is again subtracted from the dividend. F(4–7) is incremented by 1 each time a complete subtraction is made.
5. If there is no carry upon processing the high-order byte, an exit is made to the appropriate left- or right-digit sequence, as determined per STAT G.
6. Note that, before starting each subtract sequence, the partial remainder resulting from the previous subtraction is saved in the LSWR and PAL. The saving is done because, upon exit on a no-carry condition, an overdraw has occurred and the remainder in ST cannot be used for computation of the next quotient digit. Instead, the partial remainder resulting from the last successful subtraction is used for subsequent computation.

Left-Digit Sequence

1. The quotient digit in F(4–7) is the left digit of a quotient byte. This digit is reduced one count, to compensate for the overdraw, and then cross-gated via the serial adder to F(0–3).
2. The partial remainder resulting from the last successful subtraction and saved in the LSWR and PAL is shifted one digit to the left and restored to ST. The next low-order dividend digit, in B(64–67), is inserted into the low-order end of ST.
3. A test on STC equal to or greater than 4 is made to establish whether the low-order byte of the partial remainder in the LSWR has been overdrawn. In the generate quotient sequence, the contents of T are stored into the LSWR at the same time that the first subtract cycle is performed. Thus, if the partial remainder extends into T (which occurs if the STC is 4 or greater), the low-order divisor byte is subtracted from the low-order partial remainder byte once too often. In such cases, the low-order byte of the partial remainder is corrected by adding it to the low-order divisor byte. After performing the correction, the left-digit sequence is re-entered.
4. If L1 equals L2, the quotient sign byte is processed. Otherwise, the quotient digit generation routine is resumed to develop the next digit.

Correct Low-Order Remainder Byte

This routine is entered from the left- or right-digit sequence if the low-order divisor byte has been subtracted once too often from the low-order byte of the partial remainder. Correction is performed as follows:

1. STAT H is reset to initiate a true add cycle.
2. The low-order partial remainder word is placed into T. The STC is set per L2 to select the low-order byte in T.
3. The low-order divisor byte (per the ABC) is added once to the low-order partial remainder byte (per the STC), and the result is gated to T per the STC.
4. The left- or right-digit sequence is re-entered, as applicable.

Right-Digit Sequence

This sequence is entered when two quotient digits have been generated and placed into F. The following actions are performed:

1. The STC is set per D(21–23), and F is transferred to the selected ST byte. The corresponding mark trigger is set per the STC.
2. A storage request is issued to store the quotient byte per the D address.
3. A left-4 shift of the partial remainder (in PAL and LSWR) is initiated.
4. If STAT D is set, indicating that a dividend byte fetch is not required, the left-4 shift is completed and the partial remainder is restored to ST.
5. If STAT D is not set, the dividend byte fetch sequence is entered.
6. D is decremented by 1 to obtain the destination address for the next quotient byte.
7. F is cleared and STAT G is set to enter the left-digit sequence, after the first quotient digit is generated.
8. If STAT E is set, the 'invalid data interrupt' trigger is set and the instruction is ended.
9. If STAT E is not set, the generate-quotient sequence is entered.

Process Quotient Sign Byte

This routine is entered from the left-digit sequence when L1 equals L2. At this time, all dividend digits have been processed: the low-order quotient digit is in F(0–3), the byte selected by the STC is the dividend sign byte, and the remaining high-order contents of ST are the final remainder. The following actions take place:

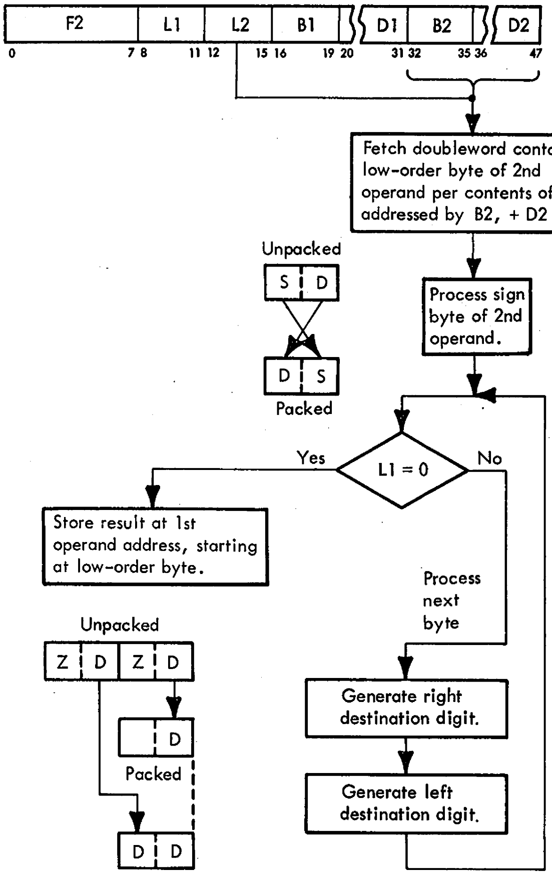
1. The STC and ABC are set per the L2 count. STAT F is set if bits 4–7 of the selected ST byte indicate a negative sign. STAT E is set if the sign is invalid; however, if an invalid sign existed, STAT E would have been set and a data program interruption would have occurred earlier.
2. The correct negative or positive sign is put into F(4–7) as determined by a comparison of STAT's C and F.
3. The ST contents are transferred to AB via the parallel adder.
4. The STC is set per D(21–23), and F is gated to the selected ST byte. The corresponding mark trigger is set, and the selected ST byte is stored into main storage.

Store Remainder Routine

1. The byte selected by the ABC, which is the low-order remainder byte, is saved in F. If necessary, the remainder sign is corrected in the serial adder before gating to F.
2. The remainder is transferred from AB to ST one byte at a time. As each byte is transferred, the corresponding mark trigger is set, the ABC and STC are incremented by 1, and L2 is decremented by 1.
3. When the STC steps to 7, ST contents are stored per the D-address. D is then incremented by 8, and the byte transfer is resumed.
4. When L2 steps to 0, the STC is decremented by 1, and the remainder sign byte is gated from F to ST. The contents of ST are then stored into the low-order destination field, and the instruction is ended.
5. If STAT E is set, an exit is made to the program interruption microprogram.

PACK, PACK (F2)

- Convert format of 2nd operand (in storage) from zoned to packed and place result into 1st operand location (in storage).
- SS format:


F2
L1
L2
B1
D1
B2
D2
0
7 8
11 12
15 16
19 20
31 32
35 36
47
Fetch doubleword containing low-order byte of 2nd operand per contents of GPR addressed by B2, + D2 + L2.
Unpacked
S D
D S
Packed
Process sign byte of 2nd operand.
Yes
L1 = 0
No
Store result at 1st operand address, starting at low-order byte.
Process next byte
Unpacked
Z D Z D
D
Packed
D D
Generate right destination digit.
Generate left destination digit.

- Separate microprogram is used during word overlap.

The Pack instruction assumes source data in the unpacked format. The low-order source byte consists of a sign (bits 0–3) and a digit (bits 4–7). These two characters are swapped as they are gated to the low-order destination byte. All other source bytes consist of a zone (bits 0–3) and a digit (bits 4–7). Only the digits are gated to the destination field, with two bytes of source being processed for each byte of destination.

The sign and digits of the second operand are moved unchanged to the first operand field and are not checked for valid codes. A separate microprogram is provided for byte processing when a word-overlap condition exists. A test for word overlap is performed in the GIS of the instruction and also each time that a new doubleword of source is fetched from main storage.

The GIS microprogram for the Pack instruction is shown in Diagram 5-307, FEMDM. This microprogram gates the low-order first operand from the SDBO to ST, increments the IC by L2 to address the low-order byte of the second operand, gates the low-order second operand from the SDBO to AB, and performs the word-overlap test.

The word-overlap test is performed in two steps. First, the doubleword addresses for the destination and source are compared by subtracting D from the IC. The difference is then shifted four bit positions to the right and gated to PAL, and PAL(40–64) is sensed for an all-zero result to detect a possible word overlap. [The right-4 shift is made to avoid comparison of byte addresses within the doubleword; i.e., the difference for the byte addresses is shifted to PAL(65–67), which is not sensed by the branch.] If the addresses for the doublewords of source and destination are different, no word-overlap condition exists. Thus, if PAL(40–64) is not zero, a branch is made to the appropriate not-word-overlap execution sequence of the instruction.

If PAL(40–63) equals zero, indicating that the same doubleword address has been specified for the source and destination, a second test must be made to verify whether special data handling is required. The contents of D are again subtracted from the IC, but this time a right-4 shift on the difference is not performed and the byte addresses within the same doubleword are compared. If PAL(40–63) equals zero, an identical address has been specified for both source and destination. Because this case of word overlap does not require special data handling, a branch is made to the not-word-overlap microprogram. If, however, PAL(40–63) is not zero, the source and destination bytes are skewed; special data handling is required in the execution phase and, accordingly, a branch is made to the appropriate program.

Instruction Execution, Not Word Overlap

- Basic execution is as follows:
  1. Process sign byte and test for exit conditions.
  2. If no exit conditions, process right destination digit.
  3. Process left destination digit and test for exit conditions.

A flowchart of the execution of the Pack instruction without word overlap is shown in Diagram 5-308, FEMDM. The major functional steps in the microprogram are described in the following subparagraphs.

Process Sign Byte

1. The selected AB byte is gated via the serial adder cross-gates to the selected ST byte. The mark trigger selected by the STC is set.
2. ABC, STC, L1, and L2 are decremented by 1.
3. An exit is made to the appropriate routine if one or more of the counters (ABC, STC, L1, L2) was equal to zero before being stepped.
4. If no exit is made, the next source byte is processed to obtain the right destination digit.

Generate Right Destination Digit

1. Bits 4–7 of the selected AB byte are gated to SAA(4–7); no data is gated to SAA(0–3). The serial adder output is gated from SAL(0–7) to the selected ST byte.
2. The ABC and L2 are decremented by one count.
3. If L2 equals zero before stepping, the remaining source bytes are extended with high-order zeros. (See "Extension of Source Bytes with High-Order Zeros.")
4. If the ABC equals zero before stepping, an exit is made to the source fetch routine. STAT G is set to cause a return to the Generate Left Destination Digit routine after the source fetch.

Generate Left Destination Digit

1. Bits 4–7 of the selected AB byte are gated to SAA(0–3). Bits 4–7 of the selected ST byte are gated to SAB(4–7). The serial adder output is gated back to the selected ST byte, and the mark trigger selected by the STC is set.
2. ABC, STC, L1, and L2 are decremented by 1. If none of these counters equalled zero before stepping, the right digit for the next destination byte is generated. (Generate Right Destination Digit sequence is entered.)

Exit Conditions

An exit is made from the sign byte routine or from the left-digit routine when one or more of the following conditions are detected by the 'DECIMAL' (functional branch) micro-order:

1. L1 or STC = 0.
2. L2 = 0.
3. ABC = 0.

When the exit is on L1 or STC equals zero, a second test on L1-equal-all-1's is required to determine whether an end-op condition exists.

Extension of Source Bytes with High-Order Zeros

This routine is entered when L2 has stepped to zero before L1 has stepped to zero.

The serial adder output (zeros) is gated to the selected ST byte with the selected mark trigger being set. L1 and STC are decremented as each byte is processed. When L1 equals zero, the contents of ST are stored per the

D-address and the common end-op routine is started. When the STC equals zero, the contents of ST are stored, and D is decremented by 8. STAT H is set to cause a return to this routine after storing the contents of ST.

Source Fetch Routine

This routine is shared with the Move with Offset instruction. STAT D is set to cause a return to the pack microprogram.

The second operand is requested from main storage, and the IC is decremented by 8. A word-overlap test is performed. If no word-overlap condition exists, the next doubleword of the second operand is gated from the SDBO to AB. Processing of the left or right destination digit is resumed as determined by STAT G.

Instruction Execution, Word Overlap

- Basic execution is as follows:
  1. Process sign byte. Update AB and test for exit conditions.
  2. If no exit conditions, process right destination digit.
  3. Process left destination digit, update AB, and test for exit conditions.

A flowchart of the Pack instruction execution under word-overlap conditions is shown in Diagram 5-309, FEMDM. This microprogram is entered when a word-overlap condition is detected in the GIS or during a source fetch. The major functional steps in the microprogram are described in the following subparagraphs:

Process Sign Byte

The sign byte of the second operand in AB is processed in the same manner as in the not-word-overlap microprogram.

Update AB from ST

The data in AB is updated by transferring the contents of S to A or the contents of T to B, depending on the STC setting. ABC, STC, L1, and L2 are decremented by 1, and the mark trigger selected by the STC is set.

If any counter equalled zero before decrementing, an exit is made to the proper store, fetch, or extend-with-zeros routine as explained for the not-word-overlap sequence. If no exit conditions exist, processing of the right destination digit is started.

Generate Right Destination Digit

This routine is the same as in the not-word-overlap sequence.

Generate Left Destination Digit

This routine is the same as in the not-word-overlap sequence and is always followed by the update routine.

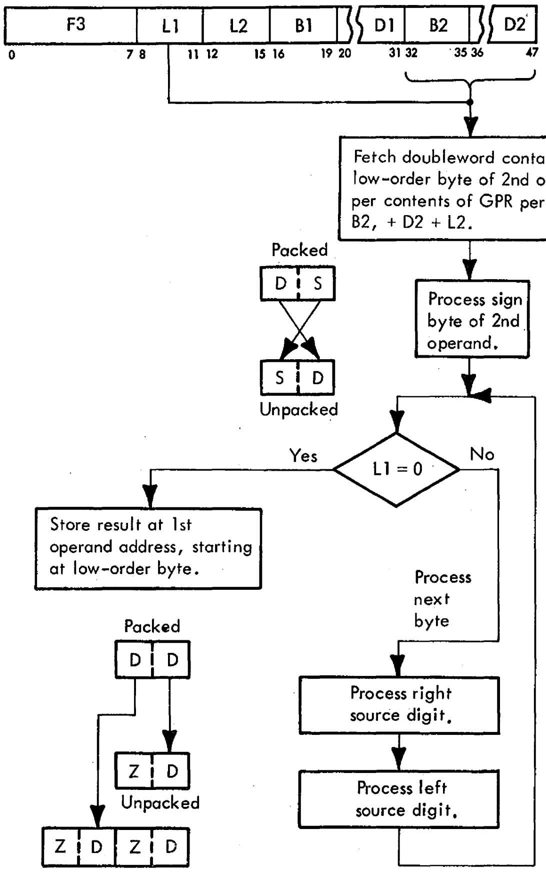
Source Fetch Routine

The next doubleword of source is requested from main storage, after which the IC is decremented by 8. Upon detection of a word-overlap condition, however, this doubleword is not used, because AB must be updated from ST. If, upon entering the source fetch routine, only the right destination digit has been placed into the selected ST byte, this byte is not transferred to AB. Instead, the following action takes place:

1. The portion of ST that has been processed (as determined by the mark triggers) is stored into the destination field, refetched from storage, and gated to both AB and ST.
2. If STAT G is set, indicating that only the right digit of the selected ST byte has been processed, the selected ST byte is transferred to F before the SDBO is gated to ST. After the SDBO is gated to ST, F is reinserted into the selected ST byte, and processing of the left digit is started.
3. If STAT G is not set, indicating that a complete ST byte has been processed, it is not necessary to save the selected ST byte. Processing of the right digit is started immediately.

UNPACK, UNPK (F3)

- Convert format of 2nd operand (in storage) from packed to zoned and place result into 1st operand location (in storage).
- SS format:


F3
L1
L2
B1
D1
B2
D2
0
7 8
11 12
15 16
19 20
31 32
35 36
47
Fetch doubleword containing low-order byte of 2nd operand per contents of GPR per B2, + D2 + L2.
Packed
D S
S D
Unpacked
Process sign byte of 2nd operand.
Yes
L1 = 0
No
Store result at 1st operand address, starting at low-order byte.
Process next byte
Packed
D D
Z D
Unpacked
Z D Z D
Process right source digit.
Process left source digit.

- Separate microprogram is used during word overlap.
- Word-overlap test is performed during GIS and in destination store and source fetch routines.

The Unpack instruction assumes data in the packed format. The low-order source byte consists of a sign (bits 4–7) and a digit (bits 0–3). These two characters are swapped as they are gated to the low-order destination byte. All other source bytes contain a pair of binary-coded-decimal digits. Each digit is transferred to the low-order portion (bits 4–7) of the corresponding destination byte, and a zone character is inserted into the high-order portion byte (bits 0–3). During this transfer, the digits are not tested for validity.

A separate microprogram is provided for byte processing when a word-overlap condition exists. A test for a word-overlap condition is performed in the GIS of the instruction and also each time that a doubleword of data is fetched from or stored into main storage.

The Unpack instruction generates two bytes of destination for each byte of source. Therefore, the condition when the destination bytes are processed "ahead" of the source always exists if the operand fields overlap. When the same doubleword address is specified, special data handling is required regardless of how the operand bytes are arranged in this doubleword. Special handling is necessary each time that source data is fetched from main storage; also, upon storing unpacked data into the destination field, a word-overlap test must be made to determine whether the source data in the CPU must be updated from storage.

The GIS microprogram for the Unpack instruction is shown in Diagram 5-307. When the first overlap indication occurs, the byte addresses are not checked. Instead, a branch is forced into the word-overlap sequence by supplying a hot carry to PAA(60), so that a test of PAL(40–63) always yields a nonzero result.

Instruction Execution, Not Word Overlap

- Basic execution is as follows:
  1. Process sign byte and test for exit conditions.
  2. If no exit conditions, process right source digit.
  3. Process left source digit, and test exit conditions.

A flowchart of Unpack instruction execution without word overlap is shown in Diagram 5-310, FEMDM. The major functional steps in the microprogram are described in the following paragraphs.

Process Sign Byte

The sign byte, selected by the ABC, is gated via the serial adder cross-gates to the selected ST byte, and the corresponding mark trigger is set. ABC, STC, L1, and L2 are decremented by 1, and an exit is made if any counter equalled zero before stepping.

Process Right Source Digit

1. Bits 4–7 of the selected AB byte are gated to SAA(4–7). The approved zone character is inserted into SAA(0–3). The serial adder output is gated to the selected ST byte, and the selected mark trigger is set.
2. L1 and STC are decremented by 1.
3. If L1 equalled zero before stepping, the contents of ST are stored and the common end-op sequence is started.
4. If STC equalled zero before stepping (and L1 was not zero), the destination store routine is started. STAT G is set to record an exit from the right digit routine.

Process Left Source Digit

1. Bits 0–3 of the selected AB byte are gated to SAA(4–7), and the zone character is inserted into SAA(0–3).
2. The adder output is gated to the selected ST byte, and the selected mark trigger is set.
3. ABC, STC, L1, and L2 are decremented by 1. An exit is made to the appropriate routine if any of the above counters equalled zero before stepping. If no exit condition exists, the right source digit in the next source byte is processed.

Exit Conditions

An exit is made from the byte processing routine whenever it is detected that L1, L2, ABC, or STC is equal to zero. Although a separate exit is provided for each possible combination of these conditions, they may be considered to be examined in the following order of priority:

1. L1 = 0
   The contents of ST are stored per the D-address, and the common end-op routine is started.
2. L2 = 0
   AB is cleared, the ABC is set per L2 (which is 7), and STAT H is set to record the end of the source field. If the STC was also zero, the destination store routine is started. If the STC was not zero, the high-order zeros routine is entered per STAT H.
3. STC = 0
   The destination store routine is started. If the ABC was also zero, STAT D is set to cause a source fetch after the destination store.
4. ABC = 0.
   The source fetch routine is started.

Extension of Source Bytes with High-Order Zeros

AB is cleared, and bits 4–7 of the selected AB byte (zeros) are gated to SAA(4–7); the approved zone character is inserted into SAA(0–3). The adder output is gated to the selected ST byte, and the corresponding mark trigger is set. L1 and STC are decremented by 1 for each byte that is processed. An exit is made to the destination store routine when the STC steps to zero, and to end-op when L1 steps to zero.

Source Fetch Routine

A request is made per the IC address, after which the IC is decremented by 8. A word-overlap test is made. If there is no word-overlap condition, the next source word is gated to AB, and the right digit of the next source byte is processed.

Destination Store Routine

1. The contents of ST are stored into the destination field per the D-address, and D is decremented by 8.
2. An exit is made to the source fetch routine if STAT D is set.
3. An exit is made to the high-order zeros routine if STAT H is set.
4. If neither STAT D nor STAT H is set, a word-overlap test is made by comparing the IC and D addresses. If no word overlap exists, the left or right digit is processed as determined by STAT G.

Instruction Execution, Word Overlap

- Basic execution is as follows:
  1. Process sign byte. Update AB and test for exit conditions.
  2. If no exit conditions, process right source digit.
  3. Process left source digit, and test for exit conditions.
- Word-overlap test is performed during source fetch and destination store routines.

A flowchart of Unpack instruction execution under word-overlap conditions is shown in Diagram 5-311, FEMDM. The steps in which this microprogram differs from that for not-word-overlap are explained in the following paragraphs.

Process Sign Byte

This step is the same as in the not-word-overlap sequence except that it is always followed by the update routine.

Update AB from ST

If the STC is less than 4, the contents of S are transferred to A; the contents of T are always transferred to B. The mark trigger selected by the STC is set. ABC, STC, L1, and L2 are decremented by 1. An exit is made to the appropriate routine if any of the above counters equalled zero before their being stepped. If no exit conditions exist, the right digit in the next source byte is processed.

Process Right Source Digit

The right source digit is processed in the same manner as for not-word-overlap. If upon processing the right digit an exit is made on STC equal zero, and ABC is not zero, the contents of S are transferred to A. In this manner, the source is correctly updated before storing the contents of ST.

Process Left Source Digit

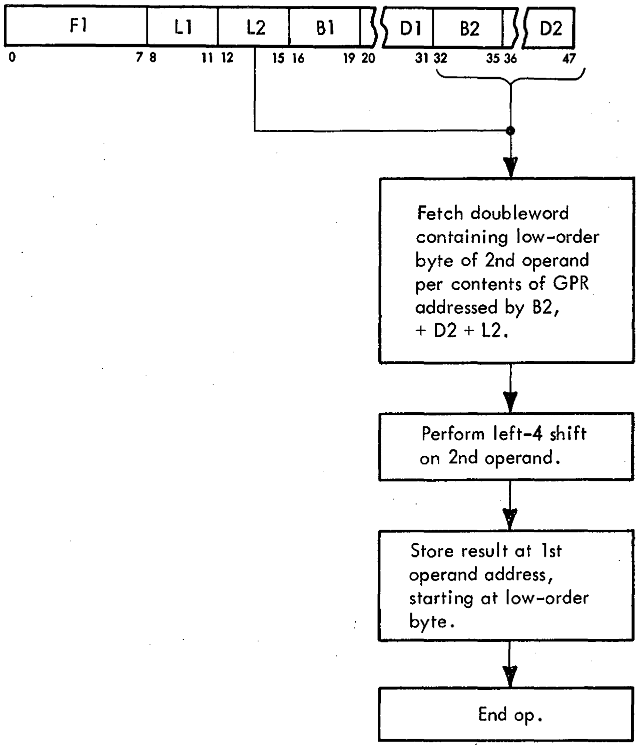
This step is the same as in the not-word-overlap sequence except that it is always followed by the update routine.

Source Fetch Routine

1. The source is requested per the IC address, after which the IC is decremented by 8.
2. The contents of D are subtracted from the IC to prepare for the word-overlap test; also, a test on STC equals 7 is made to establish how the source is to be updated in case of an overlap condition.
3. The condition when STC equals 7 indicates that the STC was zero before entering the source fetch routine. In this case, the destination has been stored into main storage. Thus, to update the source, the doubleword at the SDBO is gated to AB and ST, and processing of the left source digit is started.
4. If the STC is not 7, AB must be updated from ST. After transfer of the contents of ST to AB, processing of the right source digit is started.

MOVE WITH OFFSET, MVO (F1)

- Store 2nd operand (in storage) to left of and adjacent to low-order 4 bits of 1st operand (in storage).
- SS format:


F1
L1
L2
B1
D1
B2
D2
0
7 8
11 12
15 16
19 20
31 32
35 36
47
Fetch doubleword containing low-order byte of 2nd operand per contents of GPR addressed by B2, + D2 + L2.
Perform left-4 shift on 2nd operand.
Store result at 1st operand address, starting at low-order byte.
End op.

- Separate microprogram is used during word overlap.

The MVO instruction performs a left-4 shift on the second operand and transfers the result to the first operand location. Thus, the four low-order bits of the first operand are preserved as the lowest-order character of the second operand. During execution of the instruction, the operand signs and digits are not tested for valid codes.

No decimal shift instruction is provided, because the equivalent of a shift can be obtained by programming. Programs for right or left shift, and for an even or an odd shift amount, are written with Move with Offset instruction and the logical move instructions described in Section 5 of this Chapter.

A separate microprogram is provided for byte processing when a word-overlap condition exists. A test for word overlap is performed in the GIS of the instruction, and also each time that a new doubleword of source is fetched from main storage.

The GIS for the Move with Offset instruction is shown in Diagram 5-307. This microprogram is identical with the GIS microprogram of the Pack instruction.

Instruction Execution, Not Word Overlap

- Basic execution is as follows:
  1. Transfer bits 4–7 of selected AB byte to bits 0–3 of selected ST byte. Decrement counters.
  2. Transfer bits 0–3 of selected AB byte to bits 4–7 of selected ST byte. Repeat first step.
  3. Exit on L1 or STC = 0, L2 = 0, or ABC = 0.

A flowchart of the execution of the Move with Offset instruction when no word-overlap condition exists is shown in Diagram 5-312, FEMDM. Basically, this microprogram specifies a 2-cycle loop with appropriate exits to source fetch, destination store, high-order-zero extend, and end-op routines.

Cycle 1

1. Bits 4–7 of the selected AB byte are gated to SAA(0–3).
2. Bits 4–7 of the selected ST byte are gated to SAB(4–7).
3. The serial adder output is gated back to the selected ST byte, and the corresponding mark trigger is set.
4. L1 and STC are decremented by 1. An exit is made to the destination store routine if L1 or STC equalled zero before stepping.

Cycle 2

1. Bits 0–3 of the selected AB byte are gated to SAA(4–7). No data is gated to serial adder bits 0–3.
2. The serial adder output is gated to the selected ST byte.
3. L2 and ABC are decremented by 1. If L2 was zero before stepping, an exit is made to the high-order zero extend routine. If L2 was not zero but ABC equalled zero, an exit is made to the source fetch routine.
4. If L2 or ABC is not equal to zero, cycle 1 is repeated.

High-Order Zero Extend Routine

An entry is made into this routine when the last source byte has been processed. The selected ST byte contains the high-order source digit in bits 4–7; bits 0–3 are zeros.

The following actions are performed upon entry into the routine:

1. STAT H is set.
2. The selected mark trigger is set.
3. L1 and STC are decremented by 1.
4. If L1 or STC equals zero before stepping, an exit is made to the destination store routine.

If L1 or STC is not zero, a 1-cycle loop is started, which:

1. Gates the serial adder output (zeros) to the selected ST byte.
2. Sets the mark trigger selected by the STC.
3. Decrements L1 and STC by 1.
4. Exits to the destination store routine when L1 or STC equals zero. (STAT H is set to cause re-entry into the high-order zeros routine after the destination is stored.)

Destination Store Routine

1. The contents of ST are stored into the destination field per the D-address.
2. A test is made for the end of the destination field. If the L1 count now equals all 1's, an exit is made to the common end-op sequence.
3. If L1 is not all 1's, D is decremented by 8.
4. If STAT H is set, the high-order zeros routine is resumed. If STAT H is not set, the byte processing loop is started at cycle 2.

Source Fetch Routine†

1. The source is requested from storage, and the IC is decremented by 8.
2. A word-overlap test is made by comparing the IC and D addresses.
3. If no word-overlap condition exists, the doubleword arriving from storage is gated to AB, and byte processing is resumed.

Instruction Execution, Word Overlap

- Basic execution is as follows:
  1. Transfer bits 4–7 of selected AB byte to bits 0–3 of selected ST byte.

† This routine is shared with the Pack instruction. Return to the appropriate microprogram is effected per STAT D.

2. Transfer bits 0–3 of selected AB byte to bits 4–7 of selected ST byte.
3. Update AB from ST, and repeat first step.
4. Exit on L1 or STC = 0, L2 = 0, or ABC = 0.

A flowchart of the execution of the Move with Offset instruction when a word-overlap condition exists is shown in Diagram 5-313, FEMDM. Basically, this microprogram specifies a 3-cycle loop with appropriate exits to source fetch, destination store, high-order-zero extend, and end-op routines.

Cycle 1

This cycle is identical with cycle 1 in the not-word-overlap microprogram.

Cycle 2

1. Bits 0–3 of the selected AB byte are gated to SAA(4–7).
2. The serial adder output is gated to the selected ST byte.

Cycle 3

1. If the STC is less than 4, the contents of S are transferred to A.
2. The contents of T are transferred to B via the parallel adder.
3. L2 and ABC are decremented by 1.
4. An exit is made to the high-order zeros routine if L2 was equal to zero before stepping. An exit is made to the source fetch routine if the ABC was equal to zero and L2 was not zero.
5. If no exit conditions exist, cycle 1 is repeated for the next byte.

High-Order Zero, Destination Store, and Source Fetch Routines

The high-order zero and destination store routines are the same as in the not-word-overlap sequence. The source fetch routine, however, is different.

Upon detecting a word-overlap condition, the source from main storage is not used. Instead, AB is updated from ST: if the STC is equal to 7, the contents of T are transferred to B; if the STC is not 7, the contents of S are transferred to A and the contents of T to B.

Section 5. Logical Instructions

This section discusses the 32 logical instructions. The instructions use all five formats and operate on fixed- and variable-field length data. For a discussion of data formats, operand addressing, instruction formats, data flow, program interruptions, and condition codes, see Chapter 1.

GENERAL INITIALIZATION SEQUENCE

Before execution of SS logical instructions, a General Initialization Sequence (GIS) is performed (Diagram 5-401, FEMDM). The general function of the GIS is to set up initial conditions for the execution phase. These include:

1. Setting of STC and ABC. The STC is set to the rightmost first operand byte in ST, the byte to be processed first. Because the address of the rightmost byte is specified by D(21–23), the STC is set per these bits. Similarly, the rightmost second operand byte is selected in AB by transferring IC(21–23) to the ABC.
2. Transferring the first operand to ST during the first cycle of GIS.
3. Transferring the second operand to AB. An IC request for the second operand is issued on the first cycle of GIS; subsequently, GIS transfers the operand from the SDBO to AB.
4. Performing a word overlap test. For the purpose of this test, refer to "Word Overlap Condition", Section 4 of this Chapter.

At the completion of SS I-Fetch, a branch is made per the instruction op-code to the appropriate GIS microprogram. Note that, because of similarities in the GIS microprograms, the SS logical instructions are divided into two groups. One group consists of the Translate and Translate and Test instructions; the remaining SS instructions form the second group.

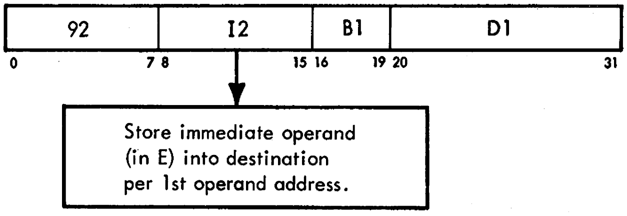
MOVE

Four logical move instructions are available:

1. Move, MVI, SI format. Places an immediate operand into the first operand location.
2. Move, MVC, SS format. Places the second operand into the first operand location.
3. Move Numerics, MVN, SS format. Places the numerics of the second operand bytes into the corresponding positions of the first operand bytes.
4. Move Zones, MVZ, SS format. Places the zones of the second operand bytes into the corresponding positions of the first operand bytes.

Move, MVI (92)

- Place immediate operand (I2 of instruction) into 1st operand location (in storage).
- SI format:

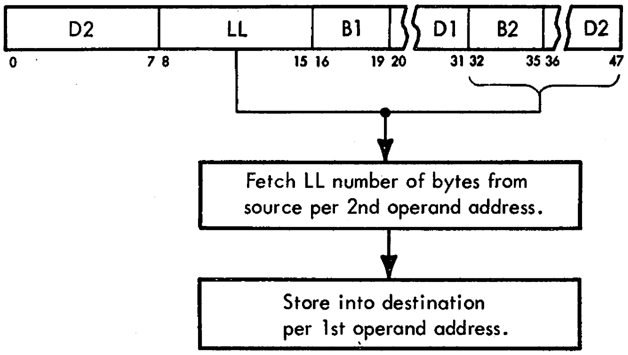

92
I2
B1
D1
0
7 8
15 16
19 20
31
Store immediate operand (in E) into destination per 1st operand address.

- Conditions at start of execution:
  First 16 bits of instruction, containing immediate operand, are in E.
  Main storage request for 1st operand has been issued per D.

The Move, MVI, instruction places the immediate operand into the first operand location. The immediate operand (I2 of instruction) is in E.

Move, MVC (D2)

- Place 2nd operand (in storage) into 1st operand location (in storage).
- SS format:


D2
LL
B1
D1
B2
D2
0
7 8
15 16
19 20
31 32
35 36
47
Fetch LL number of bytes from source per 2nd operand address.
Store into destination per 1st operand address.

- Conditions at end of GIS:
  1st operand is in ST.
  2nd operand is in AB.
  ABC is set to select byte to be routed through serial adder.
  First 16 bits of instruction are in E.

- Move operation can be high or low speed.
- Three separate microprograms are provided:
  High-speed move.
  Word overlap.
  Low-speed move.

Three separate sequences are provided for the MVC instruction. The high-speed move sequence is used when it is possible to transfer a doubleword of data at a time. This condition exists when the high-order bytes of the source and destination are specified on doubleword boundaries and a full doubleword of data remains to be processed; i.e., both the ABC and STC are equal to zero, and the LL count is greater than 6. The word-overlap sequence is used when a word-overlap condition exists. The second operand in AB is updated after each AB byte is processed. The low-speed move sequence is used when the high-speed or word-overlap condition does not exist. (The high-speed and word-overlap conditions are detected in the GIS of the instruction.)

1. Low-Speed Move Sequence
   This sequence is basically a 1-cycle operation in which the AB byte selected by the ABC is transferred through the serial adder to the ST byte selected by the STC, and the mark trigger selected by the STC is set.
   The STC and ABC are incremented, the LL count in E(8–15) is decremented, and the cycle is repeated for the next byte, unless an exit condition exists.
2. Word-Overlap Move Sequence
   This sequence is a 2-cycle sequence in which the first cycle transfers the AB byte, selected by the ABC, to the ST byte selected by the STC. The second cycle updates the source operand in AB by transferring S to A, or T to B, as determined by the value of the STC. The mark trigger selected by the STC is set. The STC and ABC are incremented, the LL count is decremented, and the sequence is repeated for the next byte, unless an exit condition exists.
3. High-Speed Move Sequence
   This routine is entered from the GIS or from the low-speed move routine.
   a. When the entrance is made from the GIS, the source operand has been transferred to ST. The contents of ST are stored by setting mark triggers 0–7 and issuing a storage request per D.
   b. The LL count in E(8–15) is decremented by 8 via the parallel adder and is then tested for all 1's. If this condition exists, an end-op sequence is started. If no end-op condition exists, the IC is incremented by 8 via the parallel adder and a source fetch request is given.
   c. When the entrance is made from the low-speed routine, D is incremented by 8 and the source doubleword from main storage is gated to both AB and ST. If at least 8 bytes remain to be processed, as determined by a ROS branch on LL count being greater than 6, the high-speed move sequence is repeated (starting at step a). If fewer than 8 bytes remain to be processed, the low-speed move sequence is started to process the remaining data.

Exit is made from the low-speed or word-overlap move routines is one of the following conditions exists: (1) LL = 0, or STC = 7 and ABC ≠ 7; (2) LL = 0, or STC = 7 and ABC = 7; (3) only ABC = 7. A separate sequence is entered for each of these conditions, as explained below:

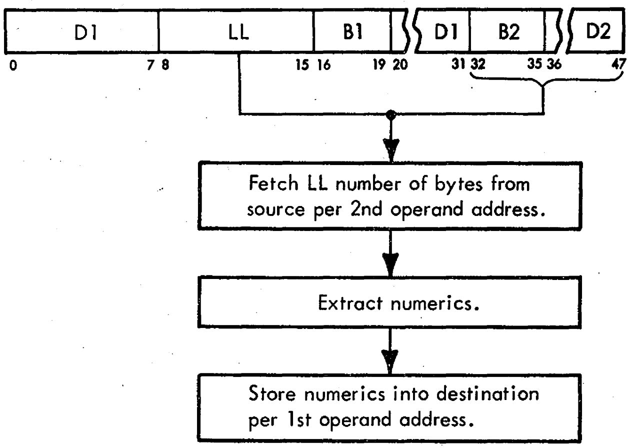
1. LL = 0, or STC = 7 and ABC ≠ 7
   A destination store is initiated, and a test for an end-op condition is made. If the LL count now equals all 1's, an entry is made into a common end-op sequence. If an end-op condition does not exist, D is incremented by 8 via the parallel adder and the low-speed move sequence is continued.
2. LL = 0, or STC = 7 and ABC = 7
   A destination store is initiated, and a test for end-op is made (LL = all 1's). A further test for a high-speed move condition is made. If at this time the LL count is 7 or greater, the IC and D are incremented by 8, a source fetch is initiated, and an entry is made into the high-speed move sequence. If neither an end-op nor a high-speed move condition exists, D is incremented by 8 and a common source fetch routine is entered which increments the IC by 8, fetches the next doubleword of source to AB, and tests for a word-overlap condition. Because there is no word-overlap at this time (ABC = STC), the low-speed move sequence is continued.
3. ABC = 7
   The IC is incremented by 8 through the parallel adder, and a fetch request is given to fetch the next doubleword of source operand. The common source fetch sequence is entered, which tests for word overlap. In this case, word overlap may exist: if it is detected, the source operand from main storage is not gated to AB, but instead ST is gated to AB and a branch is made to the move-word-overlap sequence. If no word overlap exists, the low-speed move sequence is continued after the source operand from main storage is gated to AB.

The common end-op routine is entered when the LL field has been decremented to zero. This routine restores the instruction address from the LSWR to the IC and resets STAT G (because it may have been used during the GIS).

Move Numerics, MVN (D1)

- Place numeric portion (low-order 4 bits) of each byte of 2nd operand (in storage) into low-order 4 bits of corresponding byte of 1st operand (in storage).

- SS format:


D1
LL
B1
D1
B2
D2
0
7 8
15 16
19 20
31 32
35 36
47
Fetch LL number of bytes from source per 2nd operand address.
Extract numerics.
Store numerics into destination per 1st operand address.

- Conditions at end of GIS:
  1st operand is in ST.
  2nd operand is in AB.
  ABC and STC are set to select byte(s) to be routed through serial adder.
  First 16 bits of instruction are in E.
- Separate microprogram is used for word overlap.

The MVN instruction is executed as follows:

1. Bits 4–7 of the selected AB byte are gated to SAA(4–7).
2. Bits 0–3 of the selected ST byte are gated to SAB(0–3).
3. Adder output is gated back to the selected ST byte.

Data is processed one byte at a time, and the fields may overlap in any way. Separate sequences are used for the not-word-overlap and the word-overlap conditions:

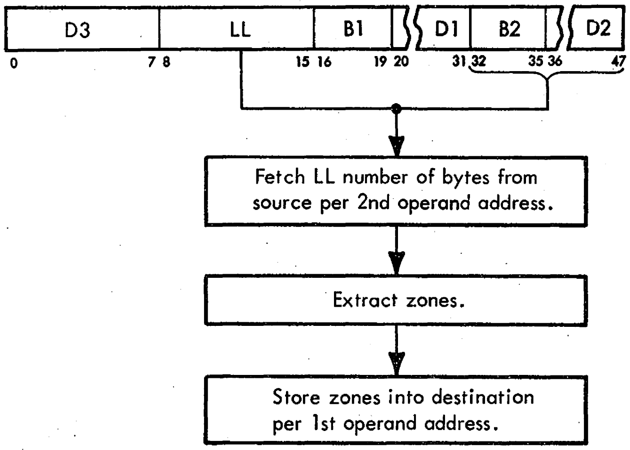
1. Not-Word-Overlap Sequence
   This sequence consists of a 1-cycle loop with an exit when LL = 0, STC = 7, or ABC = 7. As each byte is processed, the corresponding mark trigger is set per the STC; ABC and STC are incremented by 1 and LL is decremented by 1.
2. Word-Overlap Sequence
   This sequence consists of a 2-cycle loop with an exit when ABC or STC = 7, or when LL = 0.
   a. Cycle 1
      Numeric (bits 4–7) is moved from AB to ST.
   b. Cycle 2
      The contents of S are transferred to A, or the contents of T are transferred to B as determined by the STC value. The mark trigger selected by the STC is set; STC and ABC are incremented by 1, and LL is decremented by 1.

An exit from the byte processing sequence is made when LL = 0, STC = 7, or ABC = 7. A separate sequence is entered for each of these conditions, as explained below:

1. LL = 0
   The contents of ST are stored per D into the destination field. The common end-op sequence is started.
2. STC = 7
   The common destination store-fetch routine is started. If the ABC also equals 7, STAT D is set to cause a source fetch before resuming the byte processing loop.
3. ABC = 7
   The common source fetch routine is started, which includes a word-overlap test, which causes the appropriate instruction word-overlap or not-word-overlap loop to be continued.

Move Zones, MVZ (D3)

- Place zone portion (high-order 4 bits) of each byte of 2nd operand (in storage) into high-order 4 bits of corresponding byte of 1st operand (in storage).
- SS format:


D3
LL
B1
D1
B2
D2
0
7 8
15 16
19 20
31 32
35 36
47
Fetch LL number of bytes from source per 2nd operand address.
Extract zones.
Store zones into destination per 1st operand address.

- Conditions at end of GIS:
  1st operand is in ST.
  2nd operand is in AB.
  STC and ABC are set to select byte(s) to be routed through serial adder.
  First 16 bits of instruction are in E.
- Separate microprogram is used for word overlap.

The MVZ instruction specifies the following actions:

1. Bits 0–3 of the selected AB byte are gated to SAA(0–3).
2. Bits 4–7 of the selected ST byte are gated to SAB(4–7).
3. The adder output is gated back to the selected ST byte.

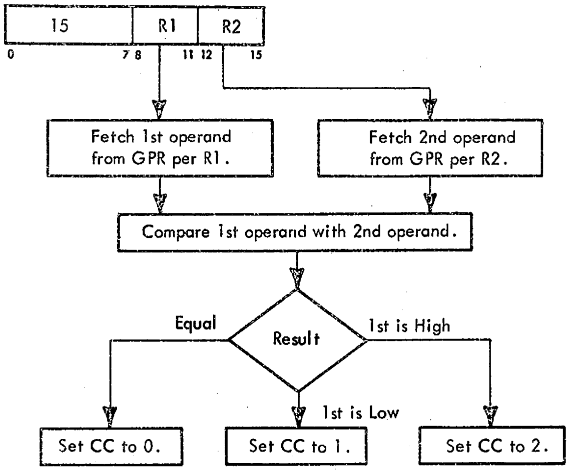
Except for the above actions, the byte processing sequence is the same as that for the MVN instruction.

COMPARE

Four Compare Logical instructions are provided, in the RR, RX, SI, and SS formats. Comparison is binary, and all codes are valid. Operation is terminated when an inequality is found.

Compare Logical, CLR (15)

- Binarily compare 1st operand (in GPR, per R1) with 2nd operand (in GPR, per R2) and set CC according to result.
- RR format:

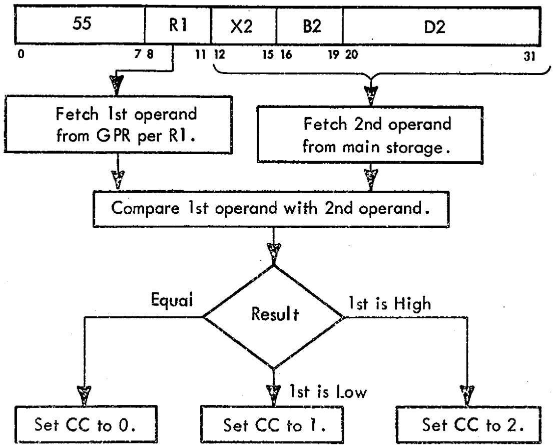

15
R1
R2
0
7 8
11 12
15
Fetch 1st operand from GPR per R1.
Fetch 2nd operand from GPR per R2.
Compare 1st operand with 2nd operand.
Equal
Result
1st is High
1st is Low
Set CC to 0.
Set CC to 1.
Set CC to 2.

- Conditions at start of execution:
  1st operand is in S and T.
  2nd operand is in A and B.
  Instruction is in E.
- CC setting:
  Operands are equal: CC = 0.
  1st operand is less than 2nd operand: CC = 1.
  1st operand is greater than 2nd operand: CC = 2.

The Compare Logical, CLR, instruction, which is in the RR format, compares the first operand with the second operand. Comparison in binary, and is performed left to right, byte by byte. The CC is set according to the result.

Compare Logical, CL (55)

- Binarily compare 1st operand (in GPR, per R1) with 2nd operand (in storage) and set CC according to result.
- RX format:

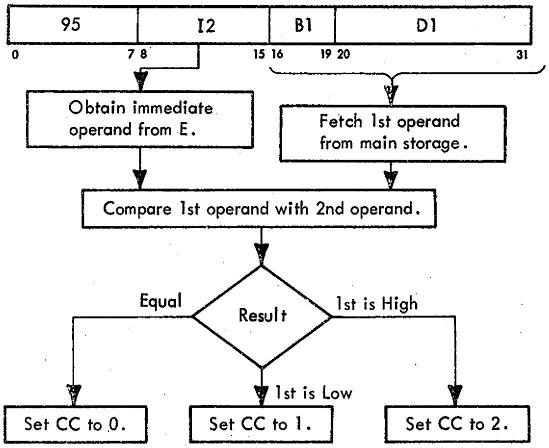

55
R1
X2
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch 1st operand from GPR per R1.
Fetch 2nd operand from main storage.
Compare 1st operand with 2nd operand.
Equal
Result
1st is High
1st is Low
Set CC to 0.
Set CC to 1.
Set CC to 2.

- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.
- CC setting:
  Operands are equal: CC = 0.
  1st operand is less than 2nd operand: CC = 1.
  1st operand is greater than 2nd operand: CC = 2.

The Compare Logical, CL, instruction, which is in the RX format, compares the first operand with the second operand. Comparison is binary, and is performed left to right, byte by byte. The CC is set according to the result.

Compare Logical, CLI (95)

- Binarily compare 1st operand (in storage) with immediate operand (I2 of instruction) and set CC according to result.
- SI format:

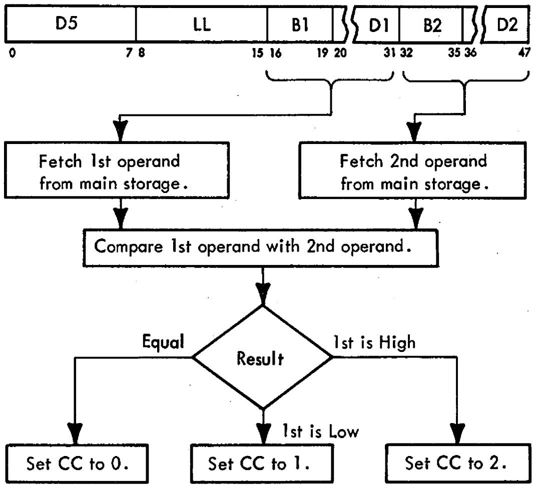

95
I2
B1
D1
0
7 8
15 16
19 20
31
Obtain immediate operand from E.
Fetch 1st operand from main storage.
Compare 1st operand with 2nd operand.
Equal
Result
1st is High
1st is Low
Set CC to 0.
Set CC to 1.
Set CC to 2.

- Conditions at start of execution:
  Main storage request for 1st operand has been issued per D.
  1st 16 bits of instruction, containing immediate operand, are in E.
- CC setting:
  Operands are equal: CC = 0.
  1st operand is less than 2nd operand: CC = 1.
  1st operand is greater than 2nd operand: CC = 2.

The Compare Logical, CLI, instruction, which is in the SI format, compares the first operand with the immediate second operand. Comparison is binary, and is performed left to right. The CC is set according to the result.

Compare Logical, CLC (D5)

- Binarily compare 1st operand (in storage) with 2nd operand (in storage) and set CC according to result.
- SS format:



D5
LL
B1
D1
B2
D2
0
7 8
15 16
19 20
31 32
35 36
47
Fetch 1st operand from main storage.
Fetch 2nd operand from main storage.
Compare 1st operand with 2nd operand.
Equal
Result
1st is High
1st is Low
Set CC to 0.
Set CC to 1.
Set CC to 2.

- Conditions at end of GIS:
  1st operand is in ST.
  2nd operand is in AB.
  ABC and STC are set to select byte(s) to be routed through serial adder.
  First 16 bits of instruction are in E.
- CC setting:
  Operands are equal: CC = 0.
  1st operand is less than 2nd operand: CC = 1.
  1st operand is greater than 2nd operand: CC = 2.
- Because results of operation are not stored into main storage, no special action is required during word overlap.

The CLC instruction is sequenced as follows:

1. The selected AB byte is gated complement to the serial adder with a hot carry to bit 7.
2. The selected ST byte is gated true to the serial adder.
3. The serial adder carry is saved in STAT H.
4. STAT A is set if a nonzero result byte is detected.
5. As each byte is processed, the LL count is decremented and the ABC and STC are incremented.
6. The above routine is continued until a nonzero result is detected in the serial adder, or until the LL count is stepped to zero, with exits for operand fetches when the STC or ABC is stepped to 7.
7. If an exit is made because a nonzero byte is detected, one additional byte will have been gated to the serial adder before the exit is made via the ROS branch. Therefore, STAT H will reflect the carry of the nonzero result byte plus 1. Because STAT H is used to determine the setting of the CC, it is set or reset per the carry of the first nonzero byte encountered.
8. The common end-op routine is used, which sets the CC per the following hardware conditions:

Hardware Conditions	CC Setting
STAT A is reset and equal compare	0
STAT A is set and STAT H is reset	1
STAT A and STAT H are set	2

AND

The AND instruction mixes two operands on a logical AND basis. An AND operation is defined as follows: if *both* operand bits are 1's, the resulting bit is 1; otherwise, the result is a 0. The following example illustrates the AND'ing of two bytes:

Bit positions	0	1	2	3	4	5	6	7
1st operand	1	0	1	0	1	0	1	0
2nd operand	1	0	0	1	1	1	0	0
Result	1	0	0	0	1	0	0	0

Note that only in bit positions 0 and 4 are both operand bits set to 1. Therefore, only bits 0 and 4 of the result are set to 1.

A simplified data flow path for AND, OR, and Exclusive-OR instructions is shown in Figure 3-25. All logical AND's, OR's, and Exclusive-OR's are performed in the serial adder, a byte at a time. The first operand is placed into ST and the second operand is placed into AB. Upon completion of the instruction, the result is placed into ST.

The sequencing of individual bytes from ST and AB through the serial adder is controlled by the STC and the ABC. In the fixed-format RR and RX instructions, the STC and ABC are preset to 4 and incremented to 7 as the four data bytes from ST and AB are routed through the serial adder. For SS instructions, the STC and ABC are

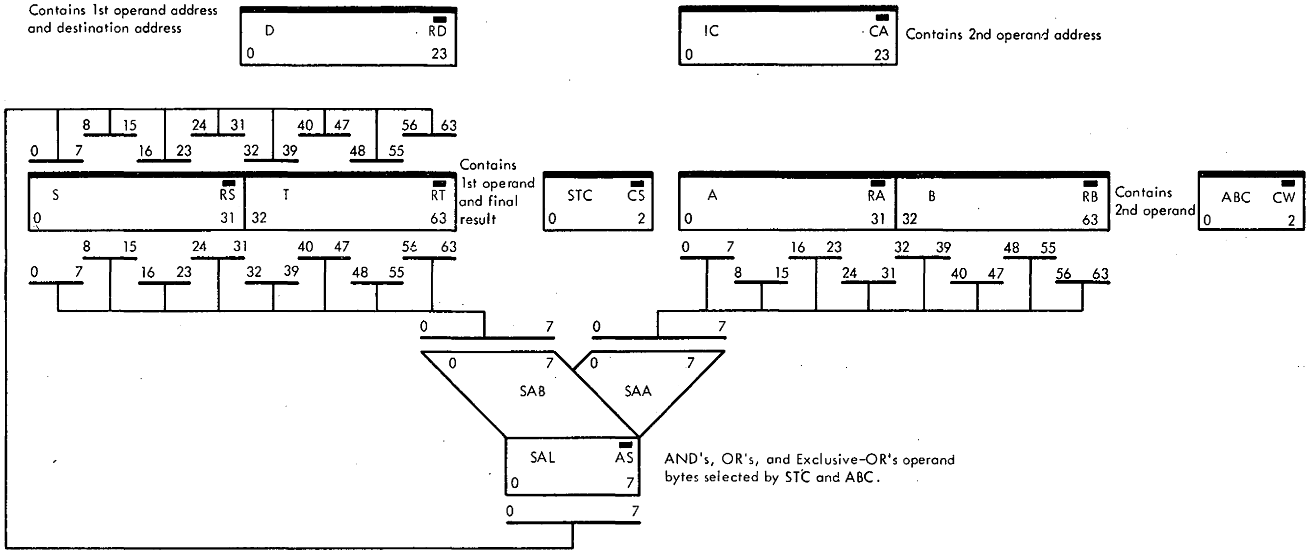
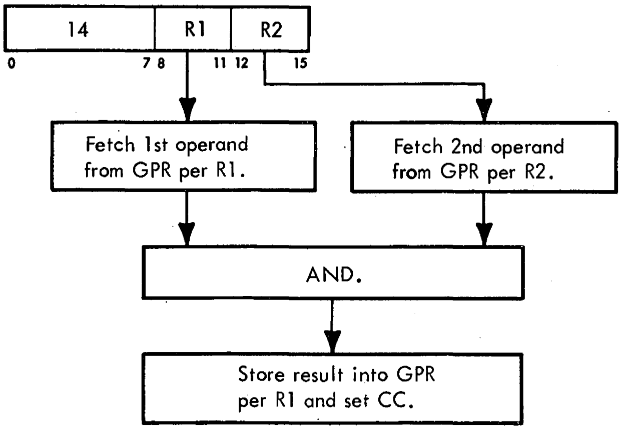

Contains 1st operand address and destination address
D
RD
0
23
IC
CA
0
23
Contains 2nd operand address
S
RS
31
32
T
RT
63
Contains 1st operand and final result
STC
CS
0
2
A
RA
31
32
B
RB
63
Contains 2nd operand
ABC
CW
0
2
SAB
SAA
SAL
AS
0
7
AND's, OR's, and Exclusive-OR's operand bytes selected by STC and ABC.

Figure 3-25. Simplified Data Flow for AND, OR, and Exclusive-OR Instructions

preset to values that point to the bytes where the data starts in their respective registers. For SI instructions, in which the second operand is one byte long (I2 field), the ABC points to the second operand location in AB, and the STC points to the first operand in ST.

AND, NR (14)

- AND 1st operand (in GPR, per R1) with 2nd operand (in GPR, per R2) and place result into 1st operand location.
- RR format:


14
R1
R2
0
7 8
11 12
15
Fetch 1st operand from GPR per R1.
Fetch 2nd operand from GPR per R2.
AND.
Store result into GPR per R1 and set CC.

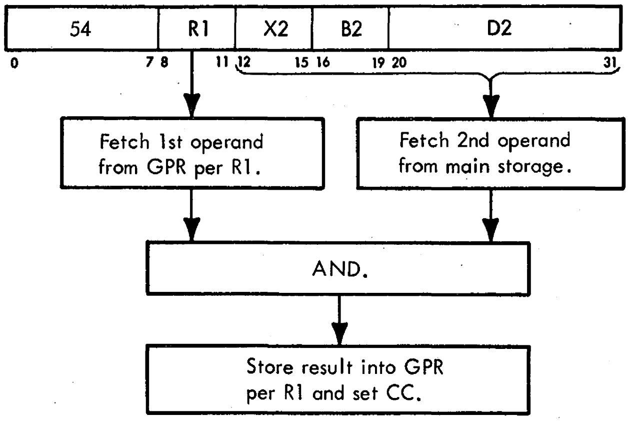
- Conditions at start of execution:
  1st operand is in S and T.
  2nd operand is in A and B.
  Instruction is in E.

- CC setting:
  Result is zero: CC = 0.
  Result is not zero: CC = 1.

The AND, NR, instruction, which is in the RR format, AND's the first operand with the second operand. The AND function is applied left to right, byte by byte.

AND, N (54)

- AND 1st operand (in GPR, per R1) with 2nd operand (in storage) and place result into 1st operand location.
- RX format:


54
R1
X2
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch 1st operand from GPR per R1.
Fetch 2nd operand from main storage.
AND.
Store result into GPR per R1 and set CC.

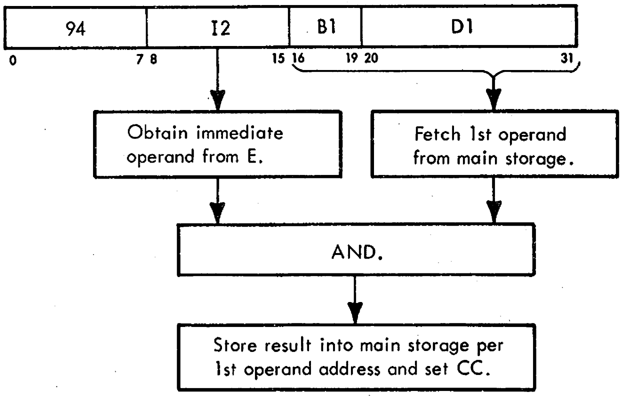
- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.

- CC setting:
  Result is zero: CC = 0.
  Result is not zero: CC = 1.

The AND, N, instruction, which is in the RX format, AND's the first operand with the second operand from main storage. The AND function is applied left to right, byte by byte.

AND, NI (94)

- AND immediate operand (I2 of instruction) with 1st operand (in storage) and place result into 1st operand location.
- SI format:

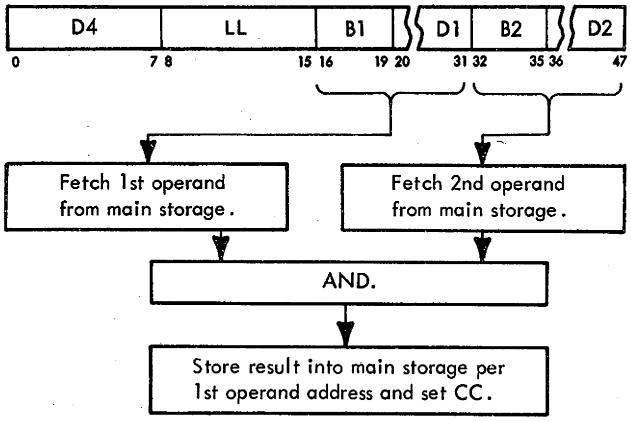

94
I2
B1
D1
0
7 8
15 16
19 20
31
Obtain immediate operand from E.
Fetch 1st operand from main storage.
AND.
Store result into main storage per 1st operand address and set CC.

- Conditions at start of execution:
  Main storage request for 1st operand has been issued per D.
  First 16 bits of instruction, containing immediate operand, are in E.
- CC setting:
  Result is zero: CC = 0.
  Result is not zero: CC = 1.

The AND, NI, instruction, which is in the SI format, AND's the first operand with the immediate second operand. The AND function is applied left to right.

AND, NC (D4)

- AND 1st operand (in storage) with 2nd operand (in storage) and place result into 1st operand location.
- SS format:


D4
LL
B1
D1
B2
D2
0
7 8
15 16
19 20
31 32
35 36
47
Fetch 1st operand from main storage.
Fetch 2nd operand from main storage.
AND.
Store result into main storage per 1st operand address and set CC.

- Conditions at end of GIS:
  1st operand is in ST.
  2nd operand is in AB.
  ABC and STC are set to select byte(s) to be routed through serial adder.
  First 16 bits of instruction are in E.
- CC setting:
  Result is zero: CC = 0.
  Result is not zero: CC = 1.
- Maximum number of bytes is 256.

The NC instruction specifies the following actions:
1. The selected AB byte is gated to SAA(0–7).
2. The selected ST byte is gated to SAB(0–7).
3. Each AB and ST bit is combined using the serial adder AND function.
4. The adder output is gated back to ST. (STAT A is set if the result byte is not zero.)

Except for the above actions, the byte processing sequence is the same as for the Move Numerics instruction.

OR

The OR instruction mixes two operands on a logical OR basis. An OR operation is defined as follows: if *either* operand bit is a 1, the resulting bit is a 1: otherwise, the result is a 0. The following example illustrates the OR'ing of two bytes.

Bit positions	0	1	2	3	4	5	6	7
1st operand	1	0	1	0	1	0	1	0
2nd operand	1	0	0	1	1	1	0	0
Result	1	0	1	1	1	1	1	0

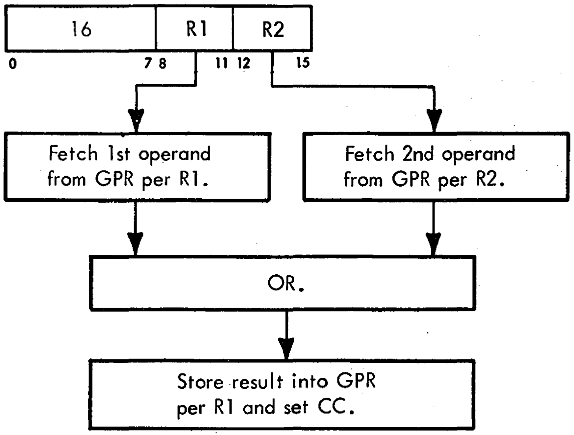
Note that only in bit positions 1 and 7 is neither bit a 1. Thus, only bits 1 and 7 of the result are set to 0, and the remaining bits are set to 1.

The sequencing of operands through the serial adder is similar to the sequencing of the AND instructions. The major difference is that the serial adder applies the OR function.

The OR operation may be executed by an instruction in the RR, RX, SI, or SS format.

OR, OR (16)

- OR 1st operand (in GPR, per R1) with 2nd operand (in GPR, per R2) and place result into 1st operand location.
- RR format:

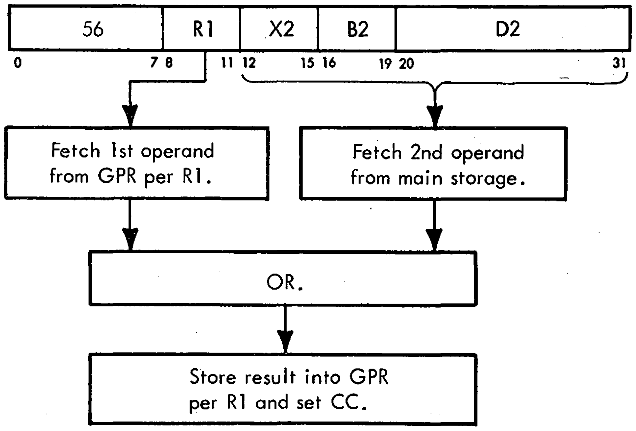

16
R1
R2
0
7 8
11 12
15
Fetch 1st operand from GPR per R1.
Fetch 2nd operand from GPR per R2.
OR.
Store result into GPR per R1 and set CC.

- Conditions at start of execution:
  1st operand is in S and T.
  2nd operand is in A and B.
  Instruction is in E.
- CC setting:
  Result is zero: CC = 0.
  Result is not zero: CC = 1.

The OR, OR, instruction, which is in the RR format, OR's the first operand with the second operand. The OR function is applied left to right, byte by byte.

OR, O (56)

- OR 1st operand (in GPR, per R1) with 2nd operand (in storage) and place result into 1st operand location.
- RX format:

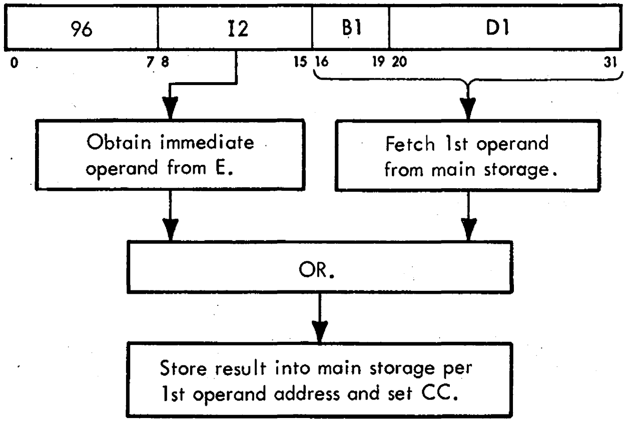

56
R1
X2
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch 1st operand from GPR per R1.
Fetch 2nd operand from main storage.
OR.
Store result into GPR per R1 and set CC.

- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.
- CC setting:
  Result is zero: CC = 0.
  Result is not zero: CC = 1.

The OR, O, instruction, which is in the RX format, OR's the first operand with the second operand from main storage. The OR function is applied left to right, byte by byte.

OR, OI (96)

- OR immediate operand (I2 of instruction) with 1st operand (in storage) and place result into 1st operand location.
- SI format:

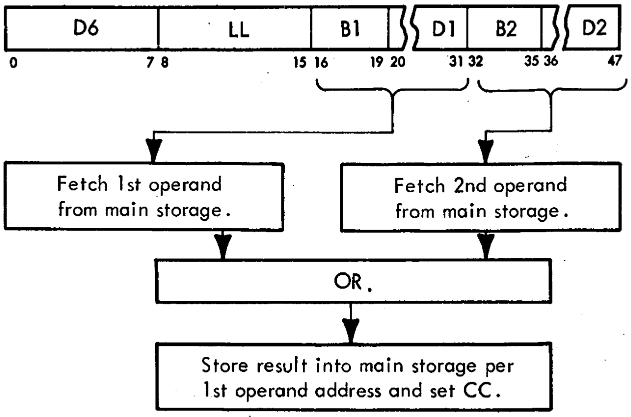

96
I2
B1
D1
0
7 8
15 16
19 20
31
Obtain immediate operand from E.
Fetch 1st operand from main storage.
OR.
Store result into main storage per 1st operand address and set CC.

- Conditions at start of execution:
  Main storage request for 1st operand has been issued per D.
  First 16 bits of instruction, containing immediate operand, are in E.
- CC setting:
  Result is zero: CC = 0.
  Result is not zero: CC = 1.

The OR, OI, instruction, which is in the SI format, OR's the first operand with the immediate second operand. The OR function is applied left to right.

OR, OC (D6)

- OR 1st operand (in storage) with 2nd operand (in storage) and place result into 1st operand location.
- SS format:


D6
LL
B1
D1
B2
D2
0
7 8
15 16
19 20
31 32
35 36
47
Fetch 1st operand from main storage.
Fetch 2nd operand from main storage.
OR.
Store result into main storage per 1st operand address and set CC.

- Conditions at end of GIS:
  1st operand is in ST.
  2nd operand is in AB.
  ABC and STC are set to select byte(s) to be routed through serial adder.
  First 16 bits of instruction are in E.
- CC setting:
  Result is zero: CC = 0.
  Result is not zero: CC = 1.
- Maximum number of bytes is 256.

The OC instruction specifies the following actions:

1. The selected AB byte and the selected ST byte are gated to the serial adder, where they are combined per the serial adder OR function.
2. The adder output is gated back to the selected ST byte, and the selected mark trigger is set per the STC.
3. STAT A is set if the result is not zero.

Except for the above actions, the byte processing sequence is the same as that for the Move Numerics instruction.

EXCLUSIVE-OR

The Exclusive-OR instruction mixes two operands on a logical Exclusive-OR basis. An Exclusive-OR operation is defined as follows: *if one and only one* of the operand bits is a 1, the resulting bit is a 1; otherwise, the result is a 0. The following example illustrates the Exclusive-OR'ing of two bytes.

Bit position	0	1	2	3	4	5	6	7
1st operand	1	0	1	0	1	0	1	0
2nd operand	1	0	0	1	1	1	0	0
Result	0	0	1	1	0	1	1	0

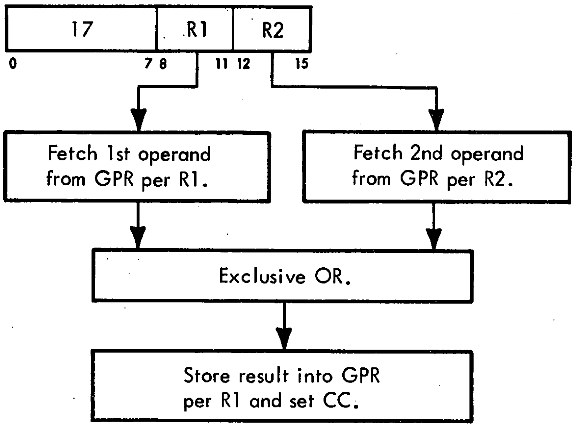
Note that in bit positions 2, 3, 5, and 6 *one and only one* of the operand bits is a 1, and that the corresponding bit positions of the result are set to 1. In bit position 0, both operand bits are 1 and the corresponding result bit is 0. In bit position 1, both bits are 0 and the result is 0.

The sequencing of operands through the serial adder is similar to the sequencing of the AND instructions. The major difference is that the serial adder applies the Exclusive-OR function.

The Exclusive-OR operation may be executed by an instruction in the RR, RX, SI, or SS format.

Exclusive-OR, XR (17)

- Exclusive-OR 1st operand (in GPR, per R1) with 2nd operand (in GPR, per R2) and place result into 1st operand location.
- RR format:


17
R1
R2
0
7 8
11 12
15
Fetch 1st operand from GPR per R1.
Fetch 2nd operand from GPR per R2.
Exclusive OR.
Store result into GPR per R1 and set CC.

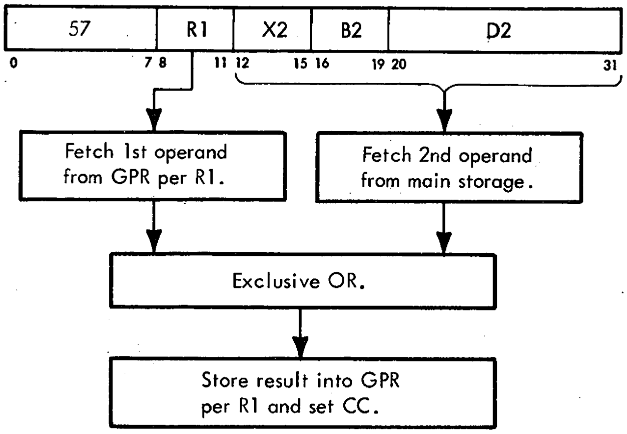
- Conditions at start of execution:
  1st operand is in S and T.
  2nd operand is in A and B.
  Instruction is in E.

- CC setting:
  Result is zero: CC = 0.
  Result is not zero: CC = 1.

The Exclusive-OR, XR, instruction, which is in the RR format, exclusive-OR's the first operand with the second operand. The exclusive-OR function is applied left to right, byte by byte.

Exclusive-OR, X (57)

- Exclusive-OR 1st operand (in GPR, per R1) with 2nd operand (in storage) and place result into 1st operand location.
- RX format:

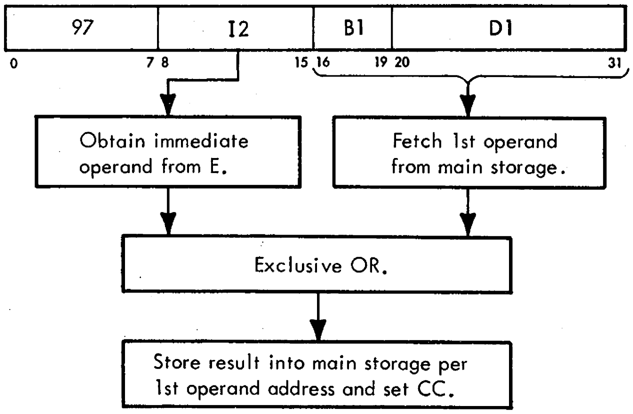

57
R1
X2
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch 1st operand from GPR per R1.
Fetch 2nd operand from main storage.
Exclusive OR.
Store result into GPR per R1 and set CC.

- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.
- CC setting:
  Result is zero: CC = 0.
  Result is not zero: CC = 1.

The Exclusive-OR, X, instruction, which is in the RX format, exclusive-OR's the first operand with the second operand from main storage. The exclusive-OR function is applied left to right, byte by byte.

Exclusive-OR, XI (97)

- Exclusive-OR immediate operand (I2 of instruction) with 1st operand (in storage) and place result into 1st operand location.
- SI format:

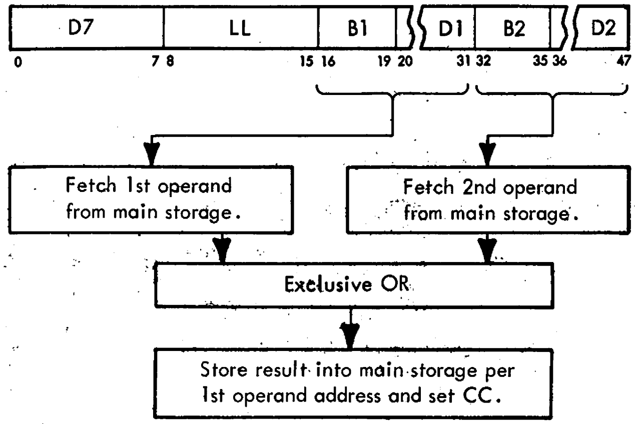

97
I2
B1
D1
0
7 8
15 16
19 20
31
Obtain immediate operand from E.
Fetch 1st operand from main storage.
Exclusive OR.
Store result into main storage per 1st operand address and set CC.

- Conditions at start of execution:
  Main storage request for 1st operand has been issued per D.
  First 16 bits of instruction, containing immediate operand, are in E.
- CC setting:
  Result is zero: CC = 0.
  Result is not zero: CC = 1.

The Exclusive-OR, XI, instruction, which is in the SI format, exclusive-OR's the first operand with the immediate second operand. The exclusive-OR function is applied left to right.

Exclusive-OR, XC (D7)

- Exclusive-OR 1st operand (in storage) with 2nd operand (in storage) and place result into 1st operand location.
- SS format:


D7
LL
B1
D1
B2
D2
0
7 8
15 16
19 20
31 32
35 36
47
Fetch 1st operand from main storage.
Fetch 2nd operand from main storage.
Exclusive OR
Store result into main storage per 1st operand address and set CC.

- Conditions at end of GIS:
  1st operand is in ST.
  2nd operand is in AB.
  ABC and STC are set to select byte(s) to be routed through serial adder.
  First 16 bits of instruction are in E.

- CC setting:
  Result is zero: CC = 0.
  Result is not zero: CC = 1.

- Maximum number of bytes is 256.

The XC instruction specifies the following actions:

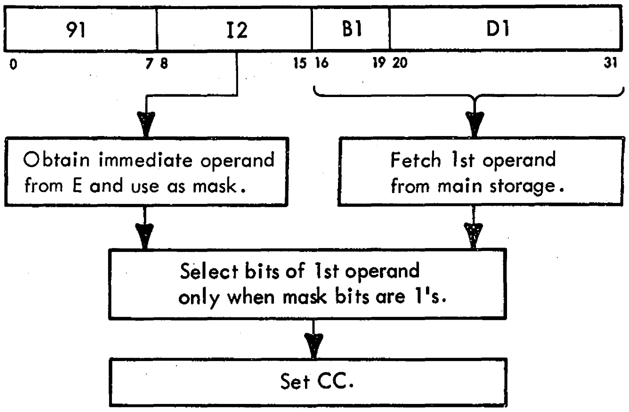
1. The selected AB byte and the selected ST byte are gated to the serial adder, where they are combined per the serial adder Exclusive-OR function.
2. The adder output is gated back to the selected ST byte, and the selected mark trigger is set per the STC.
3. STAT A is set if the result is not zero.

Except for the above actions, the byte processing sequence is the same as that for the Move Numerics instruction.

TEST UNDER MASK, TM (91)

- Set CC according to state of 1st operand bits (in storage) selected by mask bits (I2 of instruction).

- SI format:


91
I2
B1
D1
0
7 8
15 16
19 20
31
Obtain immediate operand from E and use as mask.
Fetch 1st operand from main storage.
Select bits of 1st operand only when mask bits are 1's.
Set CC.

- Conditions at start of execution:
  Main storage request for 1st operand has been issued per D.
  First 16 bits of instruction, containing immediate operand, are in E.

- CC setting:
  Selected bits all zero; mask is all zero: CC = 0.
  Selected bits mixed zero and 1: CC = 1.
  Selected bits all 1: CC = 3.

- Storage contents are not changed.

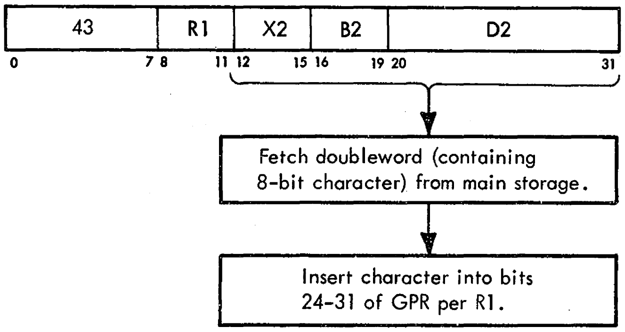
The byte of immediate data, I2, is used as an eight-bit mask. The bits of the mask are made to correspond one for one with the bits of the character in main storage specified by the first operand address.

A mask bit of 1 indicates that the storage bit is selected. When the mask bit is 0, the storage bit is ignored. When all storage bits thus selected are zero, the CC is made 0. The CC is also made 0 when the mask is all-zero. When the selected bits are all-1, the CC is made 3; otherwise, the CC is made 1. The character in storage is not changed.

INSERT CHARACTER, IC (43)

- Insert 2nd operand (byte; in storage) into bits 24–31 of 1st operand location (in GPR, per R1).

- RX format:


43
R1
X2
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch doubleword (containing 8-bit character) from main storage.
Insert character into bits 24-31 of GPR per R1.

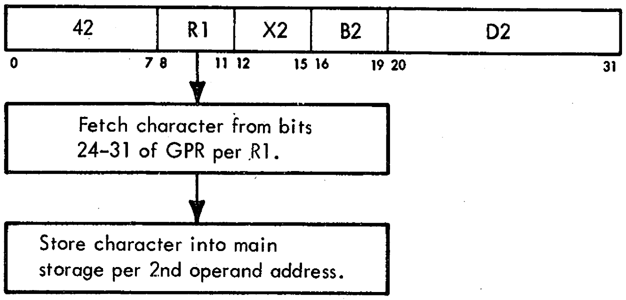
- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.

The Insert Character instruction, which is in the RX format, inserts the byte of second operand into bits 24–31 of the GPR specified by R1. The remaining bits in the GPR are unchanged.

STORE CHARACTER, STC (42)

- Store bits 24–31 of 1st operand (in GPR, per R1) into 2nd operand location (in storage).

- RX format:

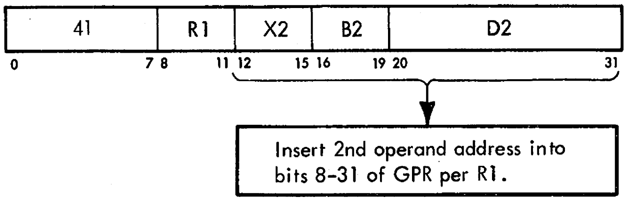

42
R1
X2
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch character from bits 24-31 of GPR per R1.
Store character into main storage per 2nd operand address.

- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.

The Store Character instruction, which is in the RX format, stores the byte (bits 24–31) of first operand into main storage per the second operand address.

LOAD ADDRESS, LA (41)

- Insert 2nd operand address into bits 8–31 of GPR specified by R1.
- RX format:



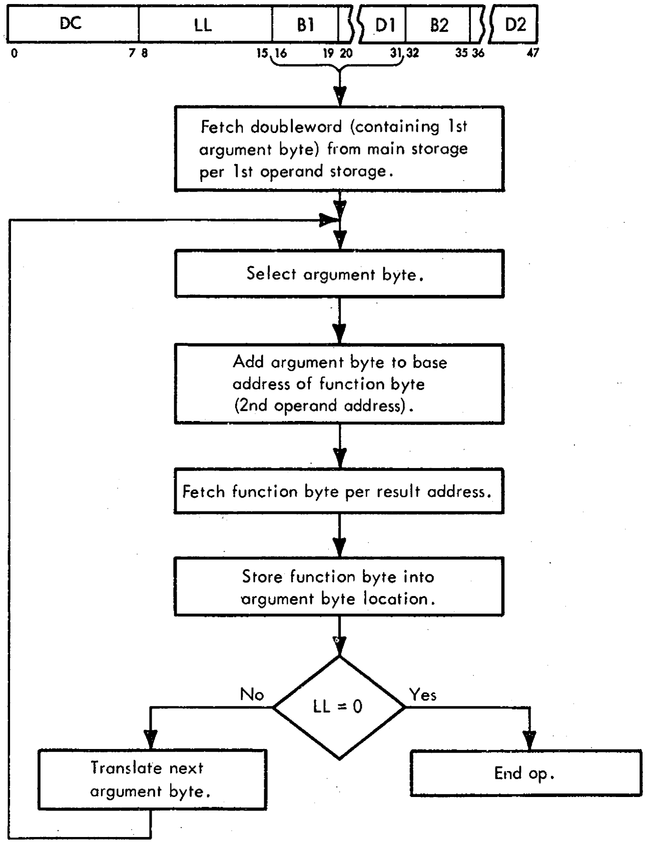
- Conditions at start of execution:
  1st operand is in S and T.
  Main storage request for 2nd operand has been issued per D.
  First 16 bits of instruction are in E.

The address specified by the X2, B2 and D2 fields is inserted into bits 8–31 of the GPR specified by R1; bits 0–7 are made 0's. The address is not inspected for availability, protection, or resolution.

The address computation follows the rules for address arithmetic. Any carries beyond the 24th bit are ignored. The same GPR may be specified by the R1, X2, and B2 instruction field, except that GPR0 can be specified only by the R1 field. In this manner, it is possible to increment the low-order 24 bits of a GPR, other than 0, by the contents of the D2 field of the instruction. The GPR to be incremented should be specified by R1 and by either X2 (with B2 set to zero) or B2 (with X2 set to zero).

TRANSLATE TR (DC)

- Add 1st operand byte (argument; in storage) to effective 2nd operand address, use result as storage address, and place function byte from resulting storage address into corresponding 1st operand byte location.
- SS format:


DC
LL
B1
D1
B2
D2
0
7 8
15 16
19 20
31 32
35 36
47
Fetch doubleword (containing 1st argument byte) from main storage per 1st operand storage.
Select argument byte.
Add argument byte to base address of function byte (2nd operand address).
Fetch function byte per result address.
Store function byte into argument byte location.
LL = 0
No
Yes
Translate next argument byte.
End op.

- Conditions at end of GIS:
  1st operand (destination) is in ST.
  Destination address is in D.
  Source address (contents of GPR per B2, + D2) is in IC.

The Translate instruction selects the first operand bytes for translation one byte at a time, proceeding from left to right. Each argument byte is added to the entire initial address, the second operand address, in the low-order bit positions. The sum is used as the address of the function byte, which then replaces the original argument byte. All data is valid. The operation proceeds until the first operand field is exhausted. The table is not altered unless an overlap occurs.

At the start of the execution sequence, the first operand has been fetched to ST. A request per the IC has been made for the second operand, but this doubleword from main storage is not used.

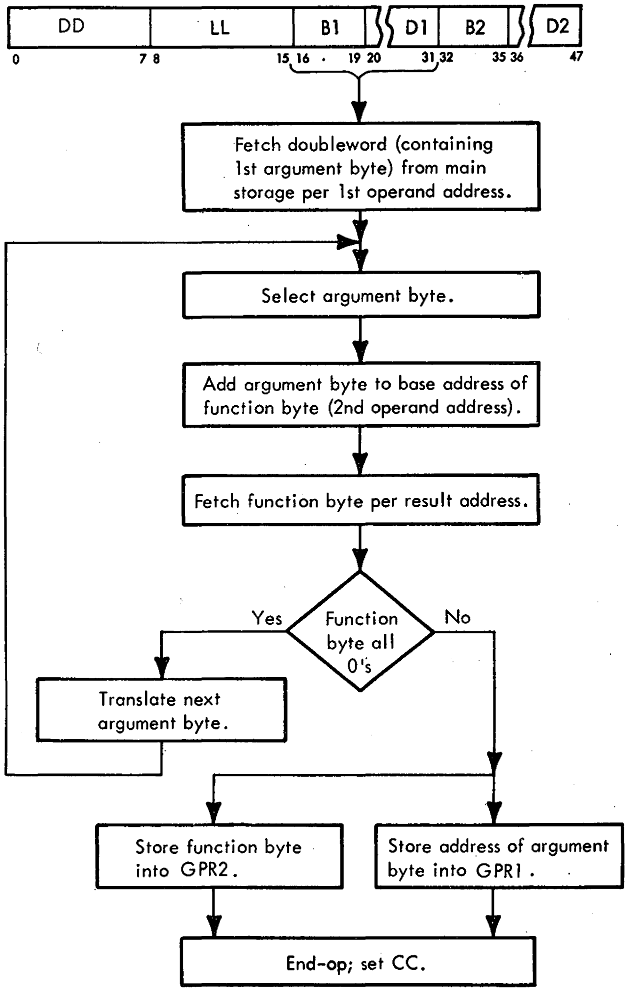
The execution sequence is as follows:

1. The selected ST byte is saved in F. The contents of T are saved in B. (The contents of the IC are saved in A.)
2. T is cleared, the STC is set to 111, and the contents of F (selected destination byte) are placed into T(56–63) via the serial adder.
3. The contents of T are added to the contents of the IC in the parallel adder, and the result is gated back to the IC.
4. A request for the second operand is issued per the IC.
5. The ABC is set per IC(21–23), and the STC is set per D(21–23).
6. The original source address is restored to the IC from A. The destination word is restored to T from B.
7. A word-overlap test was made before the source address was restored to the IC. If no word overlap exists, the table doubleword fetched from main storage is gated to AB. If word overlap is detected, the contents of S are transferred to A (B is already identical with T) and the doubleword from main storage is not used.
8. The selected AB byte is gated via the serial adder to the selected ST byte, and the selected mark trigger is set. The STC and D are incremented by 1. The LL count is decremented by 1.
9. Unless the STC was 7 or LL was zero before stepping, the sequence is repeated for the next destination byte.
10. If LL was zero, the contents of ST are stored and the common end-op sequence is started.
11. If the STC was 7 and LL not equal to zero, the contents of ST are stored and the next destination word is fetched by the common destination fetch sequence, after which the translate sequence is resumed.

TRANSLATE AND TEST, TRT (DD)

- Add 1st operand byte (argument; in storage) to effective 2nd operand address, use result as storage address, and test function byte from resulting storage address. If 0, translate and test next argument byte; if non-0, complete operation by inserting related argument address into GPR1 and function byte into GPR2.

- SS format:


DD
LL
B1
D1
B2
D2
0
7 8
15 16
19 20
31 32
35 36
47
Fetch doubleword (containing 1st argument byte) from main storage per 1st operand address.
Select argument byte.
Add argument byte to base address of function byte (2nd operand address).
Fetch function byte per result address.
Function byte all 0's
Yes
No
Translate next argument byte.
Store function byte into GPR2.
Store address of argument byte into GPR1 .
End-op; set CC.

- Conditions at end of GIS:
  1st operand (destination) is in ST.
  Destination address is in D.
  Source address (contents of GPR per B2, + D2) is in IC.

- CC setting:
  All function bytes are zero: CC = 0.
  Nonzero function byte encountered before operand is exhausted: CC = 1.
  Last function byte is nonzero: CC = 2.

The Translate and Test instruction fetches the function bytes in the same manner as the Translate instruction. Each function byte retrieved from the table is inspected for an all-zero combination.

When the function byte is zero, the operation proceeds with the next operand byte. When the first operand field is exhausted before a nonzero function byte is encountered, the operation is completed by setting the CC to 0. The contents of GPR's 1 and 2 remain unchanged.

When the function byte is nonzero, the related argument address is inserted into the low-order 24 bits of GPR1. This address indicates the argument last translated. The high-order eight bits of GPR1 remain unchanged. The function byte is inserted into the low-order eight bits of GPR2. Bits 0–23 of GPR2 remain unchanged. The CC is set to 1 when one or more argument bytes have not been translated. The CC is set to 2 if the last function byte is nonzero.

The following abbreviations are used in this discussion of the Translate and Test execution sequence:

DX: first byte in series of destination bytes.
T(DX): table byte specified by DX.
DX + 1: second byte in series of destination bytes.
T(DX + 1): table byte specified by DX + 1.'
DX + 2: third byte in series of destination bytes.

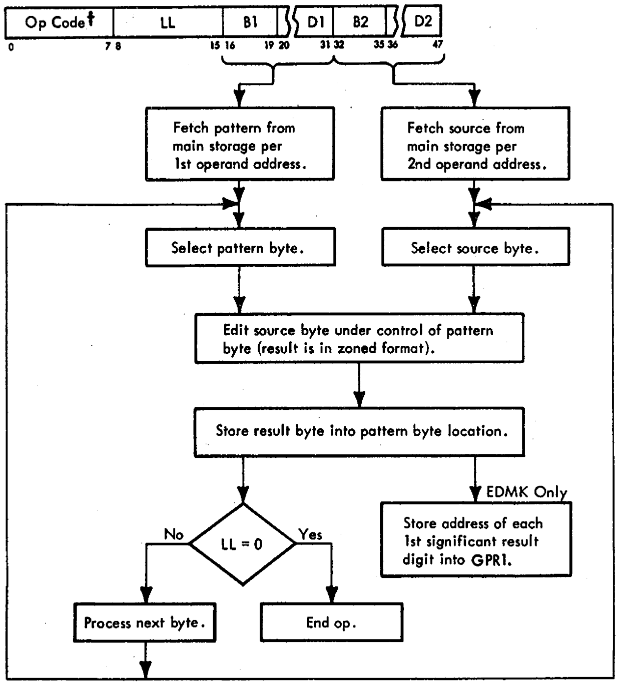
The Translate and Test instruction uses the following execution sequence:

1. First Byte Sequence
   a. The selected ST byte is saved in F.
   b. The contents of ST are transferred to AB.
   c. The STC is set to 3, and the contents of F (DX) are gated, via the serial adder, to byte 3 in S.
   d. Bytes 0, 1, and 2 in S are cleared by gating the contents of SAL to ST and successively decrementing the STC by 1.
   e. The ABC is set per D(21–23), and the STC is set to 3.
   f. The DX in S is added to the contents of the IC, and an IC request is made for T(DX).
   g. A branch per STAT G is made to the T(DX + 1) address generation routine. (STAT G is used to indicate that a table byte has been fetched and is ready for test.)
2. T(DX + 1) Address Generation
   a. The ABC is incremented by 1.
   b. DX is transferred from S to T.
   c. STAT G is set.
   d. The selected AB byte (DX + 1) is gated via the serial adder to byte 3 in S.
   e. The STC is set per IC(21–23).
   f. The T(DX) ingate and T(DX + 1) fetch sequence is started.
3. T(DX) Ingate and T(DX + 1) Fetch Sequence
   a. The table word which contains byte T(DX) is available from main storage, and either the left- or right-half word is gated to T as determined by IC(21). STC(0) is set to 1 to select the correct byte in T. Simultaneously, the contents of T(DX) are subtracted from the contents of IC to restore the table base address.
   b. If LL equals zero, an exit is made to the T(DX) test sequence.
   c. If LL is not zero, DX + 1 (in S) is added to the contents of IC and a fetch request is made for T(DX + 1).
   d. A branch per STAT G starts the T(DX) test sequence.
4. T(DX) Test Sequence and T(DX + 2) Address Generation
   a. The selected byte in T, T(DX), is gated to the serial adder for zero detection and is saved in F.
   b. STAT H is set if the ABC equals zero.
   c. DX + 1 is transferred from S to T.
   d. The STC is set to 3, and the ABC is incremented by 1 (selecting byte to DX + 2).
   e. An exit is made to the LS mark sequence (step 6) if a nonzero result is detected in the serial adder.
   f. An exit is made to the common end-op sequence if the serial adder result is zero and the LL count is zero.
   g. The LL count is decremented and the address in D is incremented by 1.
   h. If STAT H is set, an exit is made to the destination fetch routine.
   i. If no exit conditions are detected, the selected AB byte (DX + 2) is gated via the serial adder to S, and the STC is set per IC(21–23).
   j. The T(DX) ingating and T(DX + 1) fetch sequence is started. The table byte previously referred to as T(DX) has been tested. The table byte previously referred to as T(DX + 1) is now considered T(DX), and the processing loop is resumed.
5. Destination Fetch Routine
   a. Before entering this routine, the ABC has been stepped from 7 to 0 and a fetch request was made for a table byte using byte 0 of the present destination word to generate the table byte address. Because this was an erroneous address, the resulting word from main storage is not used.
   b. STAT's G and H are reset, and the IC is restored to the table base address by subtracting DX + 1.
   c. A fetch request is made per the D-address. The requested doubleword is gated to AB.
   d. The ABC was previously stepped from 0 to 1. It is now decremented to select byte 0 of the new destination doubleword (considered byte DX).
   e. The selected AB byte (DX) is gated via the serial adder to byte 3 in S.
   f. The DX (in S) is added to the contents of IC, and a fetch request is made for T(DX).
   g. Because STAT G is reset, the T(DX + 1) address generation routine is started.

6. LS Mark Sequence
   a. This routine is entered when a nonzero table byte is detected in the serial adder or when the LL count equals zero. (The last table byte tested is in F.)
   b. If the table byte was nonzero, STAT G is reset.
   c. E(8–15) is cleared and used for LAR addressing.
   d. GPR1 is accessed per E(8–15) + 1 and its contents transferred to T.
   e. The STC is incremented to 4, and byte 4 of ST is gated via the serial adder back to ST. Simultaneously, the contents of D are gated to T via the parallel adder.
   f. The contents of T are stored into GPR1; E(8–11) is incremented twice, and the STC is set to 7.
   g. The contents of GPR2 are transferred to T. The contents of F are gated via the serial adder to T(56–63), and the contents of T are stored into GPR2.
   h. STAT A is set if the byte in F was not zero.
   i. The common end-op sequence is started, which sets the CC per STAT's A and G.

EDIT AND EDIT AND MARK, ED AND EDMK (DE AND DF)

- Edit: change format of source (2nd operand; in storage) from packed to zoned, edit source under control of pattern (1st operand; in storage), and place result into 1st operand location.
- Edit and Mark: same as Edit, but in addition place location of each 1st significant digit into GPR1.
- SS format:


Op Code†
LL
B1
D1
B2
D2
0
7 8
15 16
19 20
31 32
35 36
47
Fetch pattern from main storage per 1st operand address.
Fetch source from main storage per 2nd operand address.
Select pattern byte.
Select source byte.
Edit source byte under control of pattern byte (result is in zoned format).
Store result byte into pattern byte location.
EDMK Only
Store address of each 1st significant result digit into GPR1.
No
LL = 0
Yes
Process next byte.
End op.

† DE for Edit
DF for Edit and Mark

- Conditions at end of GIS:
  Pattern (destination operand) is in ST.
  Pattern address is in D.
  Source address (contents of GPR per B2, + D2) is in IC.
  First 16 bits of instruction are in E.
- CC setting:
  Result is zero: CC = 0.
  Result is less than zero: CC = 1.
  Result is greater than zero: CC = 2.

The Edit instruction changes the format of the source (second operand) from packed to zoned, edits the source under control of a pattern (first operand), and places the result into the first operand location. The Edit and Mark instruction performs the same functions and, in addition, places the location of each first significant digit into GPR1. Both instructions are in the SS format, and share a common ROS microprogram with an exit to a separate mark routine for the Edit and Mark instruction. Because the results of a word-overlap condition are unpredictable, no special action is taken when this condition occurs.

Introduction to Edit Operation

- Edit instruction is used to:
  Eliminate high-order zeros.
  Provide asterisk protection.
  Handle sign control (CR).
  Provide punctuation.
  Blank out an all-zero field.
  Protect decimal point by use of significance start character. (This character can also be used to retain high-order zeros when desired.)
  Edit multiple adjacent fields via field separator character.

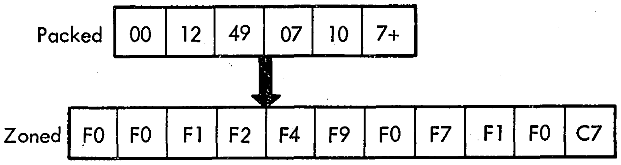
The edit operation is used to produce easy-to-read documents by inserting proper punctuation into a data record. The data to be edited (second operand) is called the "source" and must be in the packed BCD format. Consider the following source field:

00	12	49	07	10	7+

For the above field to be printed in a document, it must first be converted into the zoned format (USASCII-8 or EBCDIC). One function of the edit operation is to change the source field from packed to zoned format.† If

† Each time the digit from the source field replaces a digit select character, the four-bit digit has the proper EBCDIC or USASCII-8 zone bits inserted. PSW(12) determines whether the EBCDIC or USASCII-8 zone is inserted. For this discussion, it is assumed that the system is in EBCDIC mode.

changing from packed to zoned format were all that was necessary to produce a legible report, the Edit instruction would not be necessary, because the Unpack instruction would be sufficient. For instance, if the above packed BCD operand were changed to the EBCDIC zoned format, it would look like this:



If the above zoned BCD field were printed, it would look like this:

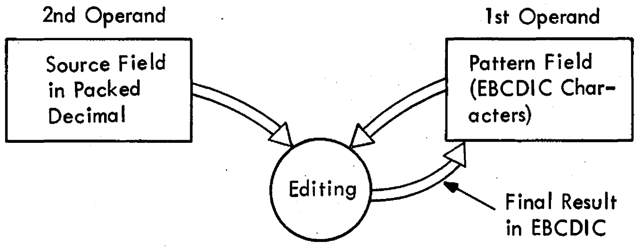
0 0 1 2 4 9 0 7 1 0 7 +

By examining the printed document, one could tell that it was a positive number with a low-order digit of 7. However, the printed document is still not legible. If, for instance, the number represents money, it would be desirable to obtain the following printed result:

$1,249,071.07

This result would require insertion of the commas and decimal points in the right place, as well as other editing. This is the main function of the edit operation.

The edit operation involves moving the source field (second operand) into the pattern field (first operand). The pattern field is initially made up of EBCDIC characters that control the editing. The final edited result replaces the pattern field:


2nd Operand
1st Operand
Source Field in Packed Decimal
Pattern Field (EBCDIC Characters)
Editing
Final Result in EBCDIC

As a rule, the second operand is shorter than the first because one source byte yields two result bytes.

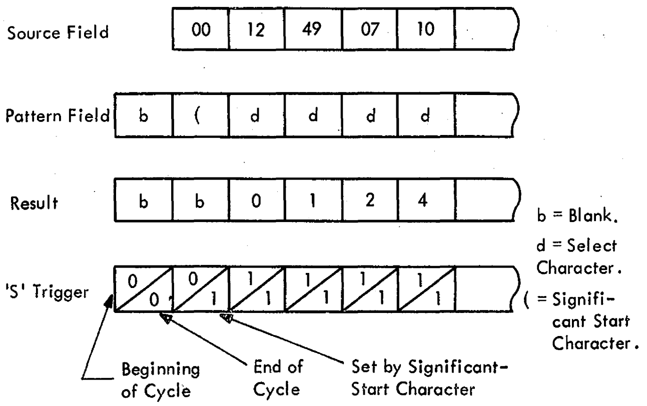
The characters in the pattern field determine the editing that takes place. The high-order (leftmost) character in the pattern field is known as the "fill" character. Any of the 256 possible EBCDIC combinations can be used as the fill character. In many edit operations, however, the fill character consists of an EBCDIC blank (0100 0000). The blank character (represented by "b" in the discussion that follows) is not printed out and facilitates programmed blanking of high-order zero fields.

Besides the fill character, three more control characters in the pattern field have special meaning: the digit select character, the significant start character, and the field separator character. These characters can appear anywhere in the pattern field.

For purposes of discussion, the digit select character is represented by "d." (The binary code for the digit select character is 0010 0000, or a hex 20.) When a digit select character is encountered in a pattern field, it is usually replaced with a digit from the source field. If the digit in the source field is a high-order zero, however, the digit select character is replaced by the fill character. By using a blank as the fill character, high-order zeros can be blanked out. If an asterisk is used as the fill character, asterisk protection for paychecks can be achieved.

Because the digit select character may be replaced by either a source digit or the fill character, the system needs some way of knowing which of the two to choose. This function is provided by a special control trigger, known as the 'S' trigger. When the 'S' trigger is set, it indicates that significant source digits are being processed. Consequently, the digit select characters in the pattern field are replaced with the digits from the source field. At the beginning of the edit operation, the 'S' trigger is always reset. As long as the 'S' trigger is reset, the digit select characters in the pattern field are replaced with the fill character.

As stated previously, the 'S' trigger is set when a nonzero digit is detected in the source field. The 'S' trigger is also set if a significant start character is detected in the pattern field. The significant start character has a bit code of 0010 0001 (hex 21). In this discussion, the symbol for the left parenthesis is used to represent the significant start character. When a significant start character is detected in the pattern field, it is replaced by either a digit from the source field or the fill character. A typical edit operation using the b, d, and (characters is illustrated and explained below.



The edit operation begins by examining the fill character (which is b in the above case). If it is not a digit select or a significant start character, it is left in place in the pattern field. Then, the next pattern character is examined. Because this is a significant start character, the next high-order source digit is examined. Because this source digit is zero and the 'S' trigger is reset (at this time), the significant start character is replaced with the fill character. However, the significant start character sets the 'S' trigger so that all subsequent source digits are significant. The remaining pattern characters in the above example are digit select characters, which are replaced with source digits.

Once significance is started, the 'S' trigger remains set until the sign of the source operand is examined. If a plus sign is detected, the 'S' trigger is reset; if the source has a negative sign, the 'S' trigger remains set because the usual method of indicating a negative quantity in a printed report is with the letters "CR". The following example illustrates how the state of the 'S' trigger identifies the number as a positive or negative quantity:

Source Field (6 Bytes)	00	12	49	07	10	7+								
Pattern Field (14 Bytes)	b	d	d	d	d	d	d	d	d	d	d	d	C	R
Result	b	b	b	1	2	4	9	0	7	1	0	7	b	b

When a pattern character is not one of the three special control characters and the 'S' trigger is set, the character is not changed. If the 'S' trigger is reset, the character is replaced by the fill character. Because detection of a positive sign resets the 'S' trigger, the remaining pattern characters (CR) are replaced by the fill character. If the sign of the source field had been minus, the 'S' trigger would have remained set and characters CR would have been left in the pattern field.

As stated previously, the 'S' trigger is reset when a plus sign is detected in the source field. The 'S' trigger is also reset if a field separator character is detected in the pattern field. The field separator character has a bit code of 0010 0010 (hex 22). In this discussion, the symbol for the right parenthesis represents the field separator character.

The field separator character is used when two or more packed BCD source fields are to be edited with one instruction into a single pattern field. The following edit example illustrates the use of the field separator character.

Source Field (6 Bytes)	01	77	6+	00	00	0+														
Pattern Field (20 Bytes)	b	d	d	(	·	d	d	b	C	R	)	d	d	d	·	d	d	b	C	R
Result	b	b	1	7	·	7	6	b	b	b	b	b	b	b	b	b	b	b	b	b
Prints Out	17.76																			

Note that after the field separator character resets the 'S' trigger, the source field does not contain any significant digits. As a result, the pattern characters are replaced by the fill character (blank).

Introduction to Edit and Mark Operation

The Edit and Mark operation is identical with that of the Edit instruction, except for the additional function of inserting a byte address into bits 8–31 of GPR1. The byte address is inserted each time the 'S' trigger is reset and a nonzero digit is inserted in the result field. The address is not inserted when significance is forced by the significant start character of the pattern. Bits 0–7 of GPR1 are not changed. The Edit and Mark instruction facilitates the programming of floating currency-symbol insertion. The character address inserted into GPR1 is 1 more than the address where a floating currency-sign would be inserted. (The Branch on Count instruction, with zero in the R2 field, may be used to reduce the inserted address by 1.)

The character address is not stored when significance is forced. Therefore, the address of the character following the significant start character should be placed into GPR1 before the Edit and Mark instruction is executed.

When a single instruction is used to edit several numbers, the address of the first significant digit of each number is inserted into GPR1. Only the last address will be available after the instruction is completed.

General Data Handling

Special circuits are packaged in the serial adder for use in the Edit and Edit and Mark instructions. These circuits consist of:

1. A decoder of serial adder bus B (SBB) to detect a digit select, significant start, or field separator character in the selected ST byte.
2. 'Right digit' trigger for AB digit selection.
3. Controls for stepping the ABC.
4. Controls for determining which data (i.e., ST byte, F, or AB digit with zone) is to be used as the result byte, and controls for gating this data to the serial adder.
5. Zero detection of the selected AB digit.
6. Sign detection of the low-order digit of the selected AB byte.
7. Detection of a mark condition.
8. The 'S' trigger with associated set-reset controls.
9. Controls for setting or resetting STAT's.

The destination field is considered a pattern field and is processed one byte at a time, from left to right, under control of the STC. Each ST byte is gated to SBB for decoding and is replaced by a byte of data which, depending on decoded conditions, may be:

1. Original data of ST byte.
2. A selected digit of AB with a zone inserted into the high-order four bits.
3. A fill character, which is contained in F.

The source field is processed, one digit at a time, from left to right, under control of the ABC and 'S' trigger, which selects which digit of a byte is to be used. The selected AB digit is examined only if a digit select or significant start character appears in the selected ST byte. The selected AB digit is not necessarily used as part of the result byte, but the next digit to be processed is selected after the digit has been examined.

Microprogram Description

The flowchart for the Edit and Edit and Mark microprogram is shown in Diagram 5-411, FEMDM. At the start of the execution sequence, the fill character is gated from ST (per the STC) through the serial adder to F. A 2-cycle data-processing sequence is then started and is repeated until all destination operand bytes have been processed. Exits from this sequence are made when required for operand fetching or marking, after which this sequence is continued. The microprogram may be divided into three parts: (1) first cycle, (2) second cycle, and (3) exit conditions.

First Cycle

The first cycle is a decode cycle; no data is transferred. The selected ST byte is gated to SBB, and the selected AB byte is gated to serial adder bus A (SBA) with the digit to be examined determined by the 'right digit' trigger. The decode circuits are activated by ROS. Decoding of SBB, SBA, and the 'S' trigger governs the selection of appropriate inputs to the serial adder, and also whether the 'S' trigger is set or reset. STAT A is set if the selected source digit (in AB) is a nonzero digit. However, if a field separator character is decoded at SBB, STAT A is reset.

STAT E is set if an invalid digit is decoded in SBA(0–3). A 1 is added to D (except for the first entry from another sequence) to keep the byte address in D at the same value as the STC for use in the marking sequence. A mark condition is detected and latched for a branch condition of the Edit and Mark instruction.

Second Cycle

At the start of this cycle, data is gated to the serial adder by hardware controls as explained in the first cycle. The second cycle performs the following control functions:

1. The serial adder output is gated back to the selected ST byte, and the appropriate mark trigger is set.
2. The STC is incremented, and the LL count in E(8–15) is decremented by ROS control.
3. The ABC is incremented by hardware controls.
4. If required, the 'digit select' trigger is complemented. This action is conditional on the following:
   a. The digit selection of AB is changed only if a significant start or digit select character was decoded during the first cycle.
   b. When a sign code is decoded in SBA(4–7) at the time bits 0–3 are selected for examination, the low-order digit (sign) is skipped by stepping the ABC and leaving the 'right digit' trigger reset.
5. If required, exit to a separate routine is made via an eight-way ROS branch, for end-op, operand fetching, or marking. If no exit conditions exist, the execution sequence is repeated.

Exit Conditions

Exits from the data processing sequence are made when one or more of the following conditions exist:

1. Edit and Mark instruction is being executed and a mark condition is detected.
2. LL = 0 or STC = 7.
3. ABC = 7.

Where more than one of the above conditions exists, a branch is made to the proper sequence in the order they are listed above. An explanation of each sequence is given below:

1. Exit on Detection of Mark Condition
   Exit to the mark sequence is made regardless of other branch conditions. Special action is taken to return counter values to what they were before entering the mark sequence, so they can be retested. STAT H is set if the ABC has just stepped from 7 to 0, to record this condition. The contents of AB are destroyed by gating E(8–15) + 1 via the parallel adder to A, and the contents of T via the parallel adder to B. E(8–15) is cleared, and GPR1 is transferred to T using E(12–15) + 1 as the LAR address. The contents of D, the byte address of the last byte processed, are placed into T(40–63). T(32–39) is retained by gating it through the serial adder and back to T at the same time the D-PAL-T transfer occurs. The contents of T are now transferred back to GPR1. Registers and counters are restored to their original contents. The source operand is replaced in AB by refetching it from main storage, and a test is made, via a ROS branch, for any other exit condition which may have been present at the time the mark sequence was started. If no other exit condition exists, the data-processing sequence is resumed.
2. Exit on LL = 0 or STC = 7 (End-Op or Destination Fetch)
   STAT D is set if the ABC also equals 7, and a destination store is started and a test is made for invalid data. If STAT E has been set, an interruption code trigger is set and an end-op sequence is started. If STAT E is not set, a test is made for an end-op condition via a ROS branch. If the LL count has been stepped to all 1's, an end-op sequence is started which sets the CC, restores the instruction address to the IC, and resets STAT G. If an end-op condition does not exist, D is incremented and a fetch request is initiated for the next doubleword of destination operand. A test

is made to see whether a source fetch is also required (ABC = 0 and STAT D set). If not, the data-processing sequence is resumed.

3. Exit on ABC = 7 (Possible Source Fetch)

   A further test must be made to determine whether the last byte of AB has been processed. This is determined by testing the ABC for an all-zero count (i.e., the ABC was stepped from 7 to 0 in the previous cycle). If the ABC is not zero, the data-processing routine is restarted; otherwise, the IC is incremented by 8, and a fetch is initiated for the next doubleword of source operand. This source fetch sequence is common to all VFL logical instructions and incorporates the word-overlap test. However, this test does not affect the edit operation. The source doubleword from main storage is gated from the SDBO to AB, and the data-processing sequence is resumed.

SHIFT

Four logical shift instructions are available:

1. Shift Left Single. Shifts a 32-bit operand left.
2. Shift Left Double. Shifts a 64-bit operand left.
3. Shift Right Single. Shifts a 32-bit operand right.
4. Shift Right Double. Shifts a 64-bit operand right.

The second operand address is not used to address data. Rather, its low-order six bits indicate the number of bit positions to be shifted; the rest of the address is ignored.

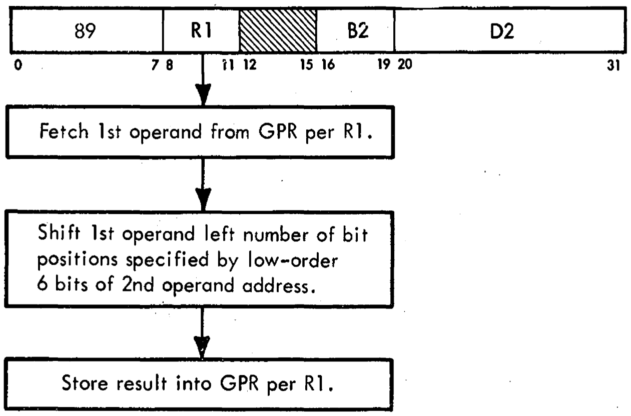
Shifting is accomplished as follows:

1. Left 1 from T to PAA.
2. Left 2 from AB to PAB.
3. Left 4 from PAA or PAB to PAL.
4. Right 4 from PAA or PAB to PAL.

Shifts of right 3 or less are obtained by combining left 1, left 2, or left 3 shifts with a right 4 shift.

Shift Left Single, SLL (89)

- Shift 1st operand (in GPR, per R1) left number of bit positions specified by low-order 6 bits of 2nd operand address.
- RS format:

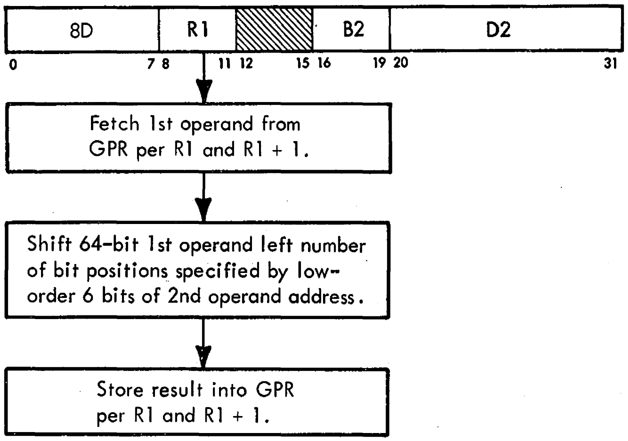

89
R1
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch 1st operand from GPR per R1.
Shift 1st operand left number of bit positions specified by low-order 6 bits of 2nd operand address.
Store result into GPR per R1.

- Conditions at start of execution:
  1st operand is in S and T.
  D(18–23) specifies amount of shift.
  First 16 bits of instruction are in E.

The Shift Left Single, SLL, instruction shifts the 32-bit first operand left the number of bit positions specified by the low-order six bits of the second operand address. All 32 bits of the GPR participate in the shift. High-order bits are shifted out without inspection and are lost. Zeros are supplied to vacated low-order GPR positions. This instruction shares the same microprogram as the fixed-point Shift Left Single, SLA, instruction (Section 2 of this chapter).

Shift Left Double, SLDL (8D)

- Shift 1st operand (in GPR, per R1 and R1 + 1) left number of bit positions specified by low-order 6 bits of 2nd operand address.
- RS format:

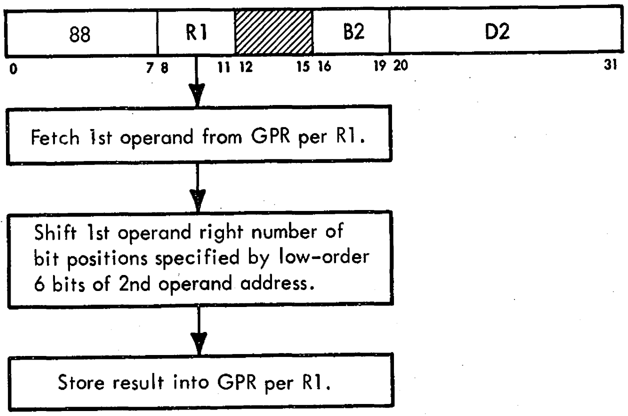

8D
R1
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch 1st operand from GPR per R1 and R1 + 1.
Shift 64-bit 1st operand left number of bit positions specified by low-order 6 bits of 2nd operand address.
Store result into GPR per R1 and R1 + 1.

- Conditions at start of execution:
  1st operand is in ST.
  D(18–23) specifies amount of shift.
  First 16 bits of instruction are in E.

The Shift Left Double, SLDL, instruction shifts the 64-bit first operand left the number of bit positions specified by the low-order six bits of the second operand address. The R1 field of the instruction specifies an even/odd pair of GPR's and must contain an even GPR address. An odd value for R1 is a specification exception and causes a specification program interruption. All 64 bits of the even/odd GPR pair participate in the shift. High-order bits are shifted out of the even-numbered GPR without inspection and are lost. Zeros are supplied to vacated low-order positions of the odd-numbered GPR. This instruction shares the same microprogram as the fixed-point Shift Left Double, SLDA, instruction (Section 2 of this chapter).

Shift Right Single, SRL (88)

- Shift 1st operand (in GPR, per R1) right number of bit positions specified by low-order 6 bits of 2nd operand address.
- RS format:

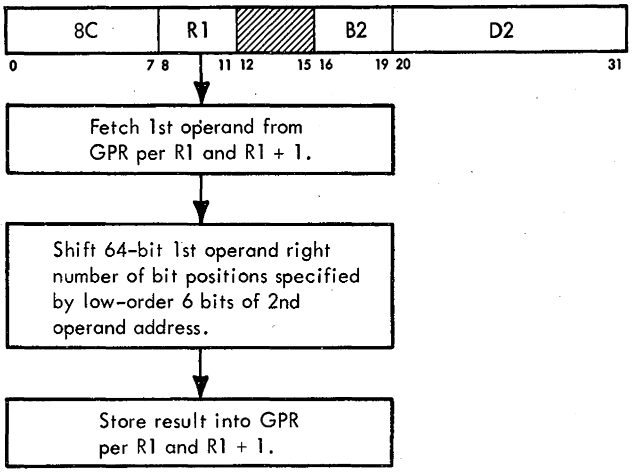

88
R1
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch 1st operand from GPR per R1.
Shift 1st operand right number of bit positions specified by low-order 6 bits of 2nd operand address.
Store result into GPR per R1.

- Conditions at start of execution:
  1st operand is in S and T.
  D(18–23) specifies amount of shift.
  First 16 bits of instruction are in E.

The Shift Right Single, SRL, instruction shifts the 32-bit first operand right the number of bit positions specified by the low-order six bits of the second operand address. All 32 bits of the GPR participate in the shift. Low-order bits are shifted out without inspection and are lost. Zeros are supplied to vacated high-order GPR positions. This instruction shares the same microprogram as the fixed-point Shift Right Single, SRA, instruction (Section 2 of this chapter).

Shift Right Double, SRDL (8C)

- Shift 1st operand (in GPR, per R1 and R1 + 1) right number of bit positions specified by low-order 6 bits of 2nd operand address.
- RS format:


8C
R1
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch 1st operand from GPR per R1 and R1 + 1.
Shift 64-bit 1st operand right number of bit positions specified by low-order 6 bits of 2nd operand address.
Store result into GPR per R1 and R1 + 1.

- Conditions at start of execution:
  1st operand is in ST.
  D(18–23) specifies amount of shift.
  First 16 bits of instruction are in E.

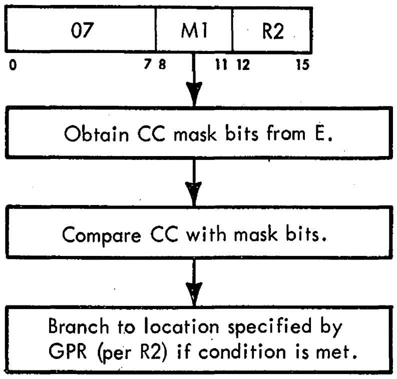
The Shift Right Double, SRDL, instruction shifts the 64-bit first operand right the number of bit positions specified by the low-order six bits of the second operand address. The R1 field of the instruction specifies an even/odd pair of GPR's and must contain an even GPR address. An odd value for R1 is a specification exception and causes a specification program interruption. All 64 bits of the even/odd GPR pair participate in the shift. Low-order bits are shifted out of the odd-numbered GPR without inspection and are lost. Zeros are supplied to vacated high-order positions of the even-numbered GPR. This instruction shares the same microprogram as the fixed-point Shift Right Double, SRDA, instruction (Section 2 of this chapter).

Section 6. Branching Instructions

This section discusses the nine branching instructions. The instructions use the RR, RX, and RS formats. For a discussion of branching, operand addressing, instruction formats, data flow, and program interruptions, see Chapter 1.

BRANCH ON CONDITION, BCR (07)

- Branch to location specified by GPR (addressed by R2) if state of CC is as specified by M1.
- RR format:


07
M1
R2
0
7 8
11 12
15
Obtain CC mask bits from E.
Compare CC with mask bits.
Branch to location specified by GPR (per R2) if condition is met.

- Conditions at start of execution:
  Branch address is in D.
  3-cycle storage request for branch-to instruction has been issued per D if branch is successful.
  Instruction is in E.
- If branch is unsuccessful, 3-cycle storage request to refill Q will be issued per IC, if required.
- Branch is unsuccessful if R2 = 0 or if condition is not met.

The Branch on Condition, BCR, instruction, which has an RR format with an op code of 07, replaces the next sequential instruction address with the branch address located in the GPR specified by R2 if the CC agrees with the corresponding mask bit(s) in the M1 field. The M1 field is used as a four-bit mask. The four bits of the mask correspond, left to right, with the four CC's (0, 1, 2, and 3) as follows:

M1 Field	Mask Position Value	CC
8	8	0
9	4	1
10	2	2
11	1	3

The branch is successful whenever the CC has a corresponding mask bit(s) of 1.

When a branch is to be made on more than one CC, the pertinent CC's are specified in the mask as the sum of their mask position values. A mask of 12, for example, specifies that a branch is to be made on CC's 0 and 1.

When all four mask bits are 1's, that is, the mask position value is 15, the branch is unconditional. When all four mask bits are 0 or when R2 = 0, the branch instruction is equivalent to a No-Operation.

At the start of execution, the instruction is in E and the branch address is in D. For a BCR instruction, the storage request can be generated from two possible places, depending upon whether the branch is successful (Diagram 5-501, FEMDM). If the branch is successful, the storage request is generated per D. If the branch is unsuccessful, the storage request is generated per the IC if Q needs to be refilled. Because the CC's, which have to be compared with the mask bits, are set during the execution phase of a previous instruction and are tested during I-Fetch of the branch instruction, success of the branch can be determined beforehand. Therefore, the BCR instruction knows whether it is successful or unsuccessful before instruction execution.

The branch-to address is always placed in D by the I-Fetch sequence. If the branch is successful, a request is made per D by means of the 'I-Fetch reset' micro-order. The correct halfword within the doubleword from main storage is then gated into Q, and from Q to R per D(21,22). The contents of D are updated by 8 and placed into the IC to address the next sequential doubleword from main storage. If the branch is unsuccessful, the storage request is issued per the IC, if Q needs to be refilled, during I-Fetch (by means of the 'I-Fetch reset' micro-order), and the data from main storage is gated to Q during the execution of the unsuccessful branch.

Successful Branch

The 'execute' and 'PSC' triggers are reset if the branch is successful. The triggers are set if the branch instruction is the subject instruction of an Execute instruction.

The contents of D are transferred to PAA(40–63). Then 8 is added to PAA, and the result (address of next doubleword to be operated on) is transferred to the IC. D(21,22) is now tested. If D(21,22) = 11, it signifies that the next instruction to be executed, when the data is gated into Q from the SDBO, occupies the last halfword of Q, and a request must be made to obtain additional instructions for Q. If D(21,22) equals a value other than

11, then the next instruction to be executed is in some halfword other than the last halfword of Q.

Assume that D(21,22) = 11. In this case, a storage request per the updated instruction address in the IC is given to refill Q. At this time, the data (branch-to instruction) that was requested during I-Fetch of the branch instruction is present at the SDBO and is gated into Q. From Q, the data is gated to R per D(21,22), thus placing the last halfword of Q into R.

The contents of the IC are transferred to the parallel adder, where they are updated by 8 and replaced into the IC to address the next doubleword from main storage. After the IC has been updated, the next sequential doubleword (requested during execution of the branch instruction) is gated from the SDBO into Q. A normal end-op cycle completes the operation.

Now assume that D(21,22) equals a value other than 11 on a successful branch. This condition means that the next instruction to be executed is contained in either the first, second, or third halfword of Q when the data from storage is gated into Q. The data which was requested during I-Fetch of the branch instruction is now present at the SDBO and is gated into Q. The halfword that contains the next instruction to be executed is then gated into R per D(21,22). Format decoding is normally accomplished from R and instruction address decoding from the IC. Because the data to be decoded has just been placed into R and the IC, it is not yet stable and therefore cannot be decoded during this cycle. Rather than delaying a cycle until the information is stable, a branch end-op cycle is taken. This cycle allows decoding of the halfword (containing the next instruction) from the SDBO as the data is transferred from the SDBO to Q and decoding of the instruction address from D.

Unsuccessful Branch

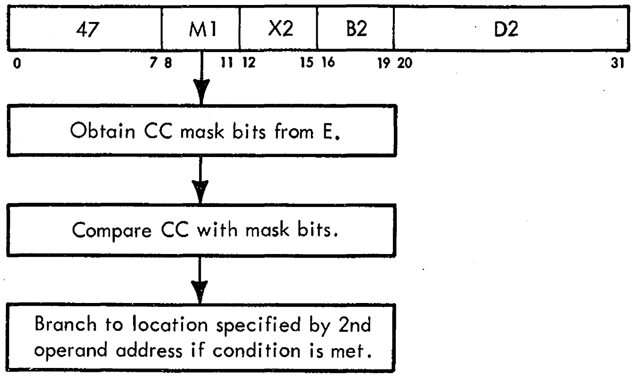
Assume, now, that the branch had been found to be unsuccessful (point B, Diagram 5-501, Sheet 2). Because the storage request was generated per the IC, IC(21,22) is now tested. Assume that IC(21,22) = 11. This value means that the next instruction to be executed is located in the last halfword of the doubleword from main storage that contains the branch instruction. This instruction is in R. Because a request to refill Q per the IC was made during I-Fetch, the IC must be updated. Accordingly, the contents of the IC are now transferred to PAB(40–63). Then 8 is added to PAB, and the result is transferred to the IC and D. At this time, the data that was requested during I-Fetch of the branch instruction is present at the SDBO and is gated into Q. The 'execute' trigger is now tested. If the trigger is set, the IC is reduced by 8 (because 8 had been added to it and the I-Fetch request was blocked by STAT G being set by the Execute instruction), a normal end-op cycle is taken, and an address-store-compare refill of Q will be performed. If the 'execute' trigger is reset, a branch end-op cycle is taken.

Now assume that IC(21,22) = 00, 01, or 10 and that the branch instruction was unsuccessful (point C, Diagram 5-501, Sheet 2). In this case, the next instruction to be executed is in either the first, second, or third halfword of Q, and Q does not need to be refilled. Therefore, a normal end-op cycle is taken, the next instruction format is decoded from R, and the instruction address is decoded from the IC.

A unique situation occurs for the BCR instruction when the 'PSC' trigger is set by an Execute instruction. This situation causes a Q-register refill following the Execute instruction. A branch is taken to the address-store-compare ROS microprogram, 8 or 16 is subtracted from the IC to select the doubleword that contains the instruction following the Execute instruction, and a storage request per the IC is made for that doubleword.

BRANCH ON CONDITION, BC (47)

- Branch to location specified by 2nd operand address if state of CC is as specified by M1.
- RX format:


47
M1
X2
B2
D2
0
7 8
11 12
15 16
19 20
31
Obtain CC mask bits from E.
Compare CC with mask bits.
Branch to location specified by 2nd operand address if condition is met.

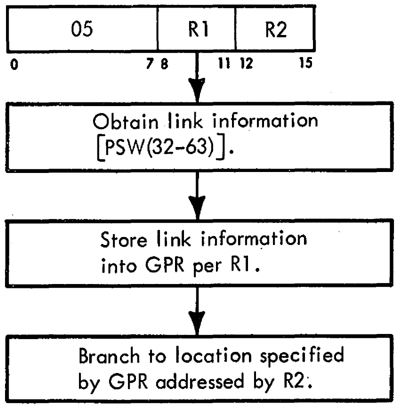
- Conditions at start of execution:
  Branch address is in D.
  3-cycle storage request for branch-to instruction has been issued per D, if branch is successful.
  First 16 bits of instruction are in E.
- If branch is unsuccessful, 3-cycle storage request to refill Q will be issued per IC, if required.

The BC instruction is similar to the BCR instruction, differing only when IC(21,22) = 00 when the branch is unsuccessful (point B, Diagram 5-501, Sheet 2). This value means that the next instruction to be executed is to come from the first halfword of the doubleword which has been requested during I-Fetch. The contents of the IC are updated by 8 and placed into the IC and D. The doubleword from main storage requested during I-Fetch is now gated from the SDBO to Q. The first halfword from Q is then transferred to R.

The 'execute' trigger is now tested. If reset, the branch instruction ends in a branch end op, and format decoding of the next instruction takes place off the SDBO because R is not stable. If the 'execute' trigger is set, the branch instruction was the subject instruction of an Execute instruction, and the present contents of the IC are incorrect because the IC was increased by 8 during this instruction. The contents of the IC are thus reduced by 8 and replaced in the IC. A normal end op completes the operation. Because the BC instruction was the subject of an Execute instruction, the return to the proper next instruction occurs during I-Fetch, as discussed in the BCR instruction.

BRANCH AND LINK, BALR (05)

- Store PSW(32–63), link information, into GPR (addressed by R1) and branch to location specified by GPR (addressed by R2).
- RR format:


05
R1
R2
0
7 8
11 12
15
Obtain link information [PSW(32-63)].
Store link information into GPR per R1.
Branch to location specified by GPR addressed by R2.

- Conditions at start of execution:
  2nd operand is in A, B, and D.
  3-cycle storage request has been issued per D for branch-to instruction.
  Instruction is in E.
- Link information consists of:
  Instruction length code.
  CC.
  Program mask bits.
  Address of next sequential instruction (link address).
- Link information is stored whether branch is successful or unsuccessful.
- If 'execute' trigger is set, link address is address of instruction following Execute instruction.
- Branch is unsuccessful if R2 = 0.

The Branch and Link, BALR, instruction, which has an RR format with an op code of 05, stores the address of the next sequential instruction. Stored with the address is link information containing the instruction length code, the CC, and the program mask bits. The instruction length code stored will be either 1 or 2. If the instruction length code stored is 2, the BALR instruction is the subject of an Execute instruction. If during a BALR operation the R2 field is equal to zero, the branch is considered unsuccessful.

The purpose of the BALR instruction is to branch to a subroutine and provide a means of returning from the subroutine to the main flow of instructions in a program. How this is accomplished is shown in Figure 3-26. When processing the main instruction flow and a BALR instruction is encountered, the address of the instruction which sequentially follows the BALR in the main instruction flow is stored in LS. For the example illustrated in Figure 3-26, the address of the next sequential instruction is 14 and is stored in GPR9. (If the BALR instruction was in address 14 of the main program, then the address stored would be 16.) Once the instruction address is stored, the branch to the subroutine occurs. The subroutine is performed and, when completed, a branch instruction could be issued using the address that was stored during the BALR instruction as the branch address to return to the main flow of instructions. After returning to the main flow of instructions, the program will continue in its normal manner, processing the remaining instructions.

In determining the address which is to be stored as the link address, IC(21,22) and the 'execute' trigger must be tested (Diagram 5-502, Sheet 2, FEMDM). If IC(21,22) = 11 and the 'execute' trigger is set (indicating the BALR is the subject of the Execute instruction and the Execute instruction is located in the second and third halfwords of its doubleword), 16 is subtracted from the IC and placed into T. Thus, if it is necessary to return to the main flow of instructions, the instruction which will be performed next is that instruction sequentially following the Execute instruction. If IC(21,22) does not equal 11 or the 'execute' trigger is reset, 8 is subtracted from the IC and the value is placed into T.

Unsuccessful Branch

E(12–15), which contains the address of the GPR which has the branch address, is examined. If E(12–15) = 0, branching is not to take place and a No-Operation occurs. If E(12–15) equals anything other than zero, branching occurs unconditionally. First assume that the branch is unsuccessful [E(12–15) equals zero]. IC(21,22) is tested. If IC(21,22) = 11, it indicates that the next instruction to be executed is in R (this instruction is the last halfword of the doubleword in Q) and a new doubleword must be placed into Q. If IC(21,22) equals any other value, the

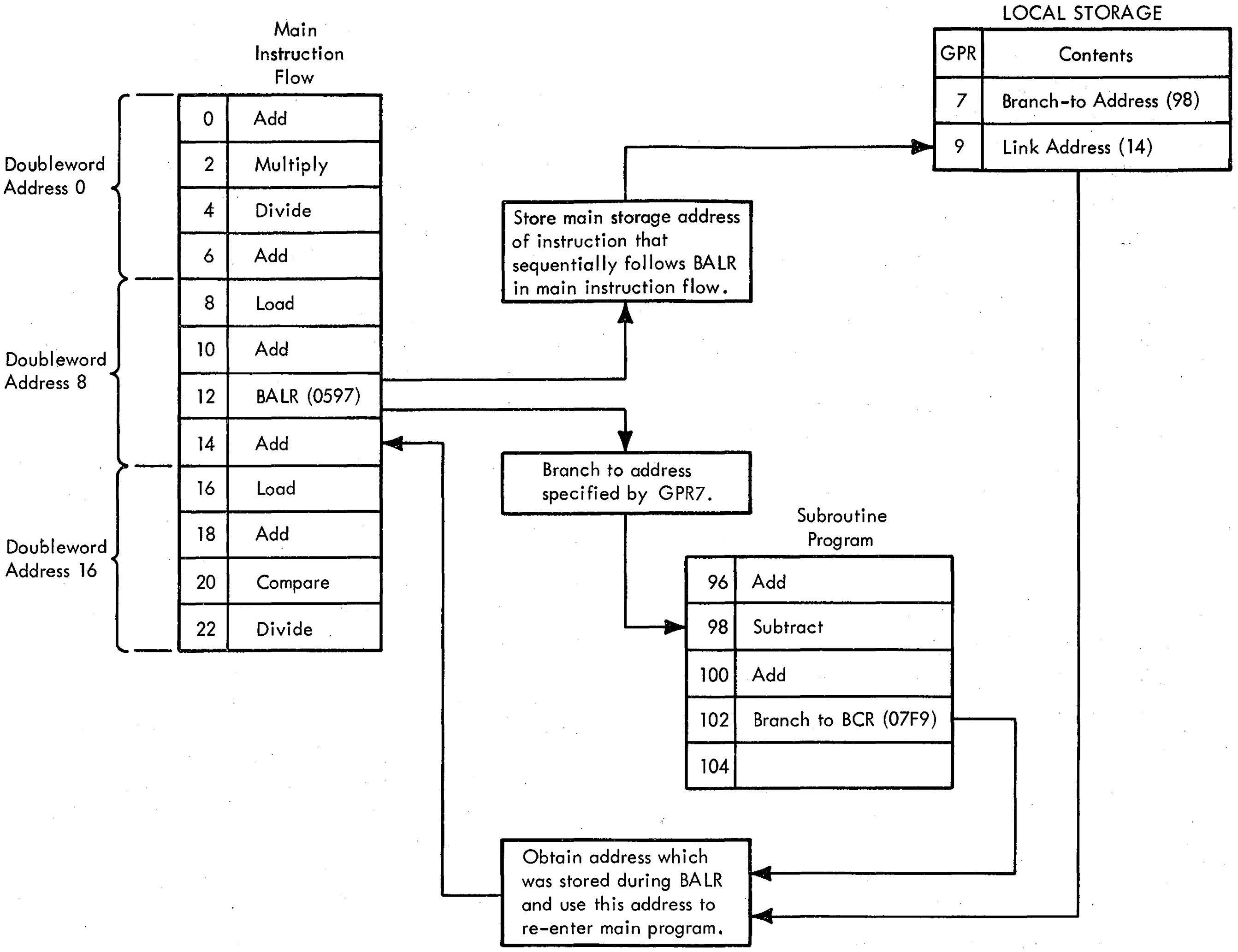


Figure 3-26. Example of Use of Branch and Link Instruction

next sequential instruction is also located in R but Q contains data which is still correct and may be operated on. In either case, the remainder of the link data [PSW(32–39)] is placed into T(32–39) and from there transferred to the GPR per E(8–11). If IC(21,22) equals a value other than 11, a normal end op is taken after storing the data and the next instruction is decoded from R. If IC(21,22) = 11, after storing the link information, a 3-cycle storage request is issued per the IC to obtain the next sequential doubleword to refill Q. The 'execute' trigger is again tested. If set, a normal end op is immediately taken and the next instruction to be executed is the instruction which sequentially follows the Execute instruction (of which the BALR instruction was the subject instruction). (This action is accomplished during the next I-Fetch by means of the ASC microprogram branch which will refetch the instruction following the Execute instruction.) If the 'execute' trigger is reset, the next instruction to be executed is in R. The IC is updated by 8 to select the next sequential doubleword after the one just requested. IC(21,22) is again tested; if it equals 11, the data on the SDBO is gated to Q and a branch end-op cycle is taken.

Successful Branch

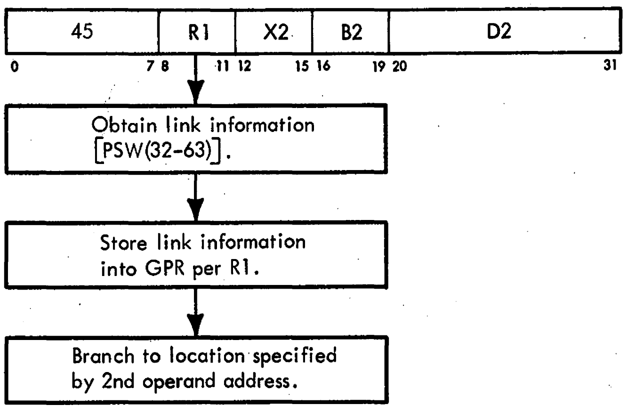
Now assume that E(12–15) does not equal zero (indicating a successful branch). The 'PSC' and 'execute' triggers are reset by a combination of the 'T6' and 'M4' micro-orders. PSW(32–39) is transferred to T(32–39). The doubleword containing the branch-to instruction (requested during I-Fetch of the branch instruction) is now present at the SDBO and is gated to Q.

From Q, the correct halfword is transferred to R per D(21,22). Then 8 is added to D, and the result is placed into the IC to address the next sequential doubleword from main storage. The data in T is now transferred to the GPR per E(8–11).

D(21,22) is tested for a value of 11. If it does not equal 11, the data in R and the IC is not yet stable, thus preventing decoding of the next instruction from R or decoding of the instruction address from the IC. A branch end-op cycle is taken, and the next instruction is decoded off the SDBO which, at this time, is stable. If D(21,22) = 11, the next instruction is located in the last halfword of the doubleword requested during I-Fetch. From Q, the last halfword is transferred to R per D(21,22). Because the halfword that contains the next instruction to be executed is the last halfword of the doubleword, Q must be filled with a new doubleword to allow continuous operation. Then 8 is added to D, and the result is transferred to the IC. A 3-cycle storage request is issued per the IC to obtain the next doubleword. The address of the doubleword is tested for validity; if the address is invalid, the 'I-Fetch invalid address' trigger is set. The IC is updated by 8. By this time, the requested doubleword is present at the SDBO and is gated to Q. A normal end-op cycle is taken, and the next instruction to be executed is decoded off R.

BRANCH AND LINK, BAL (45)

- Store PSW(32–63), link information, into GPR (addressed by R1) and branch to location specified by 2nd operand address.
- RX format:


45
R1
X2
B2
D2
0
7 8
11 12
15 16
19 20
31
Obtain link information [PSW(32-63)].
Store link information into GPR per R1.
Branch to location specified by 2nd operand address.

- Conditions at start of execution:
  Branch address is in D.
  3-cycle storage request has been issued per D for branch-to instruction.
  First 16 bits of instruction are in E.
- Link information consists of:
  Instruction length code.
  CC.
  Program mask bits.
  Address of next sequential instruction (link address).
- Link information is stored whether branch is successful or unsuccessful.
- BAL is unconditional branch.
- If 'execute' trigger is reset and ABC = 0, IC reflects correct address and is stored as link address.
- If 'execute' trigger is reset and ABC does not equal 0, IC is reduced by 8 and then stored as link address.
- If 'execute' trigger is set, link address is address of instruction following Execute instruction.

The Branch and Link, BAL, instruction stores the address of the instruction which, if the branch were unsuccessful, would be the next sequential instruction address. Stored with the address is link information consisting of the instruction length code, the CC, and the program mask bits. The instruction length code stored is 2. The BAL instruction is an unconditional branch with an RX format and an op code of 45.

At the start of execution, the first 16 bits of the instruction are in E, the branch address is in D, and a 3-cycle storage request has been generated per D for the branch-to instruction (Diagram 5-503, FEMDM). At the beginning of the operation, the last three bits of the IC are transferred to the ABC. This value will be used to determine the correct value of the IC before it is stored into LS as link address information. The contents of the IC are transferred to the parallel adder and reduced by 8. This value is then transferred to T, from where it and the remainder of the link data [PSW(32–39)] will be transferred to LS. The 'execute' trigger is then tested; if it is set, the branch operation is the subject instruction of an Execute instruction.

First assume the 'execute' trigger is reset. The ABC is now checked for all zeros. If equal to zero, it indicates that the branch instruction now being executed was located in the third and fourth halfwords of Q. Normally, when an instruction with an RX format occupies the last two halfwords of Q, a storage request is generated during I-Fetch per the IC and the IC is updated by 8. Because this is a branch instruction, however, the storage request from the IC is prevented and the IC is not increased by 8. Therefore, the address presently in T (after being reduced by 8) is incorrect and 8 must be added to it before it is stored into LS.

Assume that the ABC did not equal zero. PSW(32–39) is transferred to T(32–39). Because the ABC was not equal to zero, the link address is correct and can be stored into the GPR per E(8–11). The contents of D are then transferred to the parallel adder, increased by 8, and transferred to the IC. The IC now contains the address of the doubleword in main storage which follows the doubleword containing the branch-to instruction. At this time, a storage request for the next doubleword is issued if D(21,22) = 11. The data requested per D during I-Fetch is at the SDBO and is gated to Q. From Q, the correct halfword is transferred to R per D(21,22).

D(21,22) is now checked for a value of 11. If it is equal to 11, the next instruction to be executed is in the last halfword of Q. The IC is then updated by 8 to address the next doubleword in main storage. By this time, the doubleword that was requested during the branch instruction is present at the SDBO and can be gated into Q. A normal end-op cycle is then taken to complete the operation. If D(21,22) did not equal 11, a branch end-op cycle is taken and the next instruction is decoded off the SDBO.

Now assume that the 'execute' trigger is set, indicating that the branch instruction is the subject instruction of an Execute instruction. IC(21,22) is tested for a value of 11. If it equals 11, the Execute instruction was located in the second and third halfword of its doubleword and, when in Q, I-Fetch issued a storage request and increased the IC by 8. This increase results in an address in the IC that is 16 bytes higher than the doubleword address containing the Execute instruction. Because the address that is stored as link information is the address of the doubleword containing the instruction immediately following the Execute instruction, the address has to be reduced by 16. The address in T, however, has already been reduced by 8; therefore, only 8 must be subtracted from it. The remainder of the link information is now transferred from PSW(32–39) to T(32–39).

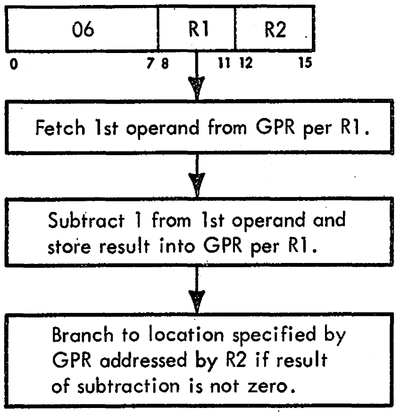
The contents of D are transferred to the parallel adder, updated by 8, and then transferred to the IC to address the next doubleword. The 'execute' trigger is now reset. A storage request for that doubleword whose address was just placed into the IC is issued if D(21,22) = 11. At this time, the data requested during I-Fetch of the branch instruction is present on the SDBO and is gated to Q. The correct halfword in Q is then transferred to R per D(21,22). The link address located in T is transferred to the parallel adder, where it is decreased by 8. This value is now equal to the address of the doubleword that contains the Execute instruction and is transferred to T. From T, the link information is transferred to the GPR per E(8–11).

D(21,22) is now checked for a value of 11. If it equals 11, the next instruction to be executed is in the last halfword of Q. The IC is then updated by 8 to address the next doubleword in main storage. By this time, the doubleword that was requested during the branch instruction is present at the SDBO and can be gated into Q. A normal end-op cycle is then taken to complete the operation. If D(21,22) did not equal 11, a branch end-op cycle is taken and the next instruction is decoded off the SDBO.

Now assume that IC(21,22) did not equal 11. This condition indicates that the Execute instruction was not in the second halfword and a storage request was not automatically generated. The link information presently in T is therefore correct and can be stored into the GPR per E(8–11). The contents of D are increased by 8 and placed into the IC. Again a storage request is generated if D(21,22) = 11. At this time, the doubleword containing the branch-to instruction is located in the SDBO and is gated to Q. The correct halfword in Q is then transferred to R per D(21,22). D(21,22) is tested for a value of 11, and operations continue as previously described, ending with a branch end op or a normal end op.

BRANCH ON COUNT, BCTR (06)

- Subtract 1 from 1st operand (in GPR, per R1) and, if result is not 0, branch to address specified by GPR (addressed by R2).
- RR format:


06
R1
R2
0
7 8
11 12
15
Fetch 1st operand from GPR per R1.
Subtract 1 from 1st operand and store result into GPR per R1.
Branch to location specified by GPR addressed by R2 if result of subtraction is not zero.

- Conditions at start of execution:
  Branch address is in D.
  3-cycle storage request has been issued per D for branch-to instruction.
  First operand is in S and T.
  Instruction is in E.

The Branch on Count, BCTR, instruction subtracts 1 from the first operand (contents of the GPR specified by R1) and, if the result does not equal zero or R2 does not equal zero, branches to the address specified by the contents of the GPR designated by R2. The result of the subtraction is stored into the first operand location. If the result of the subtraction equals zero, the next sequential instruction is executed. If E(12–15) = 0, the branch is automatically unsuccessful. The BCTR instruction has an RR format with an op code of 06.

At the start of execution, the instruction is in E, the first operand is in S and T, the branch address is in D, and a 3-cycle storage request has been issued per D for the branch-to instruction (Diagram 5-504, FEMDM). The first operand is transferred from T to B and from B to the parallel adder, where 1 is subtracted from the operand to determine whether the branch is successful. Before subtracting 1, E(12–15) is tested for zeros. As previously stated, if E(12–15) = 0, the branch is unsuccessful. Assume that E(12–15) does not equal zero. The contents of B are transferred to the parallel adder, where 1 is subtracted from the operand; the result is transferred via T into LS. The result of the subtraction is tested for all zeros; if zero, the branch is unsuccessful; if not zero, the branch is successful.

Successful Branch

First assume that the branch is successful. The 'PSC' and 'execute' triggers are reset. Because D(21,22) indicates in which halfword the branch-to instruction is located, it is examined. If D(21,22) = 11, the instruction is located in the last halfword of the doubleword requested during I-Fetch of the branch instruction. The contents of D, therefore, are updated by 8 and transferred to the IC. By this time, the data requested during I-Fetch is present at the SDBO and can be gated into Q. The last halfword of Q is then transferred to R per D(21,22). A 3-cycle storage request for the next doubleword is now issued per the IC. The contents of the IC are then transferred to the parallel adder, updated by 8, and transferred back to the IC to select the next doubleword from main storage. At this time, the doubleword which sequentially follows the doubleword containing the branch-to instruction is present at the SDBO and is gated into Q. A normal end-op cycle is taken, and the next instruction to be executed is decoded from R.

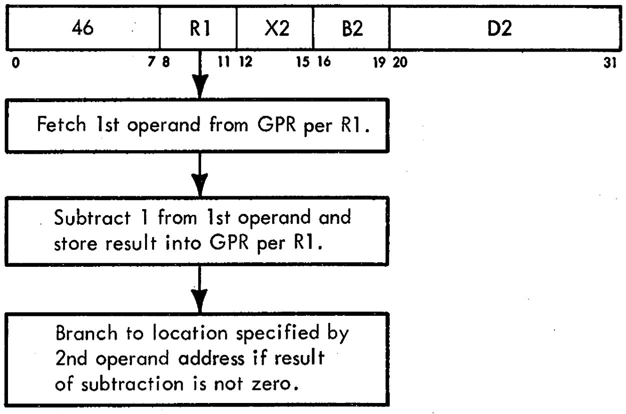
If D(21,22) did not equal 11, the branch-to instruction is located in some halfword other than the last. In this case, the contents of D are transferred to the parallel adder, updated by 8, and then transferred to the IC. At this time, the doubleword containing the branch-to instruction is present at the SDBO and can be gated into Q. From Q, the correct halfword is transferred to R per D(21,22). A branch end-op cycle is taken, and the next instruction is decoded off the SDBO.

Unsuccessful Branch

Now assume that the branch is unsuccessful. If IC(21,22) = 11 or 00, a storage request per the IC must be given during the branch execution phase to obtain the next sequential doubleword from main storage (because this action was inhibited during the end-op cycle by the branch decoder). Once the storage request is issued, the 'execute' trigger is tested. If set, it indicates that the branch instruction is the subject instruction of an Execute instruction. Therefore, a normal end-op cycle is taken to complete the operation. The data requested in this case is not gated into Q. If the 'execute' trigger is reset, IC(21,22) is tested to see whether it contains 11. If the value is 11, the IC is updated by 8 (to select another doubleword) and placed into the IC and D. By this time, the data requested by the storage request given during the execution phase of the branch instruction is present at the SDBO and can be gated into Q. A normal end-op cycle is then taken, and the next instruction to be executed is decoded off R. If IC(21,22) does not equal 11, then it equals 00, and the next instruction is located in the first halfword of the doubleword requested during the execution phase of the branch instruction. The IC is updated by 8, and the result is placed into D and the IC. At this time, the data requested during the execution phase is present at the SDBO. This doubleword is gated to Q; the correct halfword from Q(0–15) is transferred to R. Because the format is normally decoded off R and the data just placed in R is not yet stable, a branch end-op cycle is taken. This cycle allows the next instruction to be decoded off the SDBO, which at this time is stable.

BRANCH ON COUNT, BCT (46)

- Subtract 1 from 1st operand (in GPR, per R1) and, if result is not 0, branch to location specified by 2nd operand address.
- RX format:


46
R1
X2
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch 1st operand from GPR per R1.
Subtract 1 from 1st operand and store result into GPR per R1.
Branch to location specified by 2nd operand address if result of subtraction is not zero.

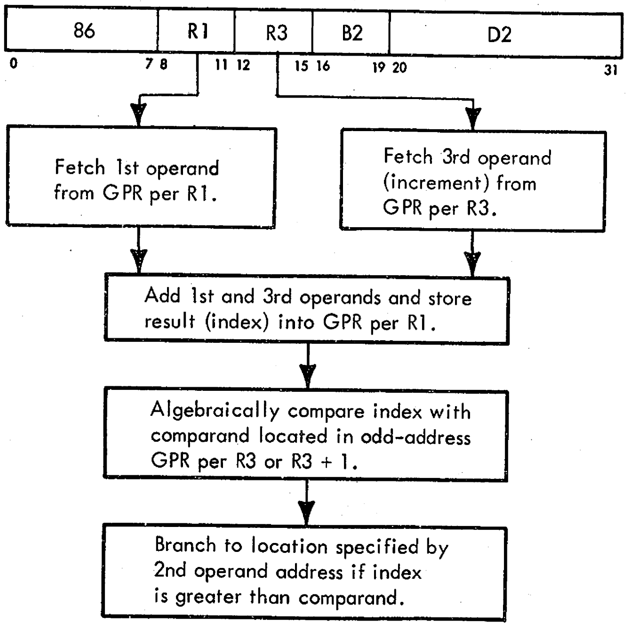
- Conditions at start of execution:
  Branch address is in D.
  3-cycle storage request has been issued per D for branch-to instruction.
  1st operand is in S and T.
  First 16 bits of instruction are in E.

The Branch On Count, BCT, instruction is similar to the BCTR instruction except that E(12–15) is not tested for zero. Refer to Diagram 5-504 for a flowchart of the BCT instruction.

BRANCH ON INDEX HIGH, BXH (86)

- Add increment (3rd operand; in GPR, per R3) to 1st operand (in GPR, per R1), algebraically compare result (index) with comparand (in odd-address GPR specified by R3 or R3 + 1), and, if index is greater than comparand, branch to location specified by 2nd operand address.

- RS format:


86
R1
R3
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch 1st operand from GPR per R1.
Fetch 3rd operand (increment) from GPR per R3.
Add 1st and 3rd operands and store result (index) into GPR per R1.
Algebraically compare index with comparand located in odd-address GPR per R3 or R3 + 1.
Branch to location specified by 2nd operand address if index is greater than comparand.

- Conditions at start of execution:
  Branch address is in D.
  3-cycle storage request has been issued per D for branch-to instruction.
  1st operand is in S and T.
  First 16 bits of instruction are in E.
- Sum of 1st and 3rd operands is always stored whether branch is successful or not.
- Comparand address (R3 or R3 + 1) must be odd.

The Branch on Index High, BXH, instruction, which has an RS format with an op code of 86:

1. Adds the third operand to the first operand.
2. Stores the result (index) into the GPR addressed by R1.
3. Compares the index with a *comparand* obtained from a GPR addressed by R3 or R3 + 1.
4. Branches if the sum is greater than the comparand.

At the start of execution, the first 16 bits of the instruction are in E, the first operand is in S and T, the branch address is in D, and a 3-cycle storage request has been issued per D for the branch-to instruction (Diagram 5-505, FEMDM). To allow the third operand to be placed into T without destroying the first operand, the first operand is transferred to B. The third operand is then transferred from the GPR per E(12–15) and placed into T. The two operands are added, and the result is transferred to B. The comparand, the value that the sum of the two operands (index) is compared with, is now transferred from LS and placed into T. The contents of B and the 2's-complement of T are transferred to the parallel adder and added to determine whether the branch is successful. IC(21,22) is tested for either 11 or 00. If the branch is unsuccessful, this test sets up conditions to issue a storage request per the IC to obtain the next sequential doubleword after the doubleword containing the branch instruction.

Assume that IC(21,22) = 11 or 00. D(21,22) is now tested to determine where in the doubleword the next instruction is located. If D(21,22) = 11, the next instruction to be executed is contained in the last halfword of the doubleword requested during I-Fetch of the branch instruction. The PAL's and E(7) are now tested (by means of the 'J47 ≠ 0' micro-order) to determine whether the branch is successful. First assume the branch is successful; that is, the PAL's are positive (index greater than the comparand) and E(7) = 0. The contents of B (index) are transferred to T, and from there to the GPR per E(8–11). The 'PSC' and 'execute' triggers are reset by means of the 'TIF' micro-order. At this time, the doubleword requested during I-Fetch is present at the SDBO and is gated into Q. The halfword containing the next instruction to be executed is then transferred to R per D(21,22). A test is made to re-establish that D(21,22) = 11. Assuming the original conditions still exist, the contents of D are now updated by 8 and placed into the IC. This action allows the selection of the next doubleword in main storage. A 3-cycle storage request is then issued per the new value in the IC. The IC is then updated by 8 to allow selection of the next doubleword from main storage when needed. By this time, the data requested during the execution phase of the branch instruction is available at the SDBO and can be gated into Q. A normal end-op cycle is then taken to complete the operation. During the end-op cycle, the next instruction executed is decoded off R.

Now assume the branch is unsuccessful. That is, the PAL's are not positive and E(7) = 0. The contents of B (index) are transferred to T, and from there to the GPR per E(8–11). Because the branch is unsuccessful and the contents of R have not been changed, R still contains the instruction that is located in the halfword following the last halfword of the branch instruction. A normal end-op cycle, therefore, can be taken and the instruction decoded off R. If IC(21,22) = 11 or 00, a 3-cycle storage request is issued per the IC, which at this time contains the address of the doubleword that sequentially follows the doubleword containing the branch instruction. The 'execute' trigger is now tested. If set, it indicates that the branch instruction was the subject instruction of an Execute instruction and a normal end-op cycle is taken, completing the operation. If the 'execute' trigger is reset, IC(21,22) is tested for a value of 11. Recall that, when IC(21,22) was previously tested, it was checked for a value of either 11 or 00. To proceed sequentially in the program without any delay, it is now necessary to determine which value the IC contains. First, assume that IC(21,22) = 11. This value indicates that the next instruction occupies the last halfword of the doubleword in which the branch instruction is located and is presently in R. Recall that R is where format decoding of an instruction occurs. This situation being the case, the IC is updated by 8 to address the doubleword that is 16 bytes from the doubleword address containing the branch instruction. The data that was requested when it was found that the branch was unsuccessful is now present at the SDBO (Diagram 5-505, Sheet 3), and can be gated into Q. A branch end-op cycle is taken, completing the operation. The next instruction is decoded off the SDBO when the data is transferred to Q.

Now assume that IC(21,22) = 00. In this case, the data to be executed is located in the first halfword of the doubleword requested during the execution of the branch when the branch instruction was found to be unsuccessful. The IC is updated by 8. At this time, the data is present at the SDBO and can be gated into Q. The first halfword in Q is transferred to R. Recall that the format for the next instruction is normally decoded off R. Because the next instruction to be executed has just been transferred into R, this data is not yet stable and cannot be decoded. A branch end-op cycle is therefore taken. This cycle allows the next instruction format to be decoded off the SDBO. The SDBO is stable at this time and therefore can be used.

Now assume that when D(21,22) was tested for 11 some other value was found. Again, conditions are tested to see whether the branch is successful (Diagram 5-505, Sheet 2). If successful, the data in B is transferred to T and from there to the GPR per E(8–11). Also, the data that was requested during I-Fetch of the branch instruction is present at the SDBO and can now be gated into Q. From Q, the halfword containing the next instruction is transferred to R per D(21,22). If D(21,22) = 11, the contents of D are now updated by 8 and placed into the IC. This action allows selection of the next doubleword in main storage. A 3-cycle storage request is then issued per the new value in the IC. The IC is then updated by 8 to allow selection of the next doubleword from main storage when needed. By this time, the data requested during the execution phase of the branch instruction is available at the SDBO and can be gated into Q. A normal end-op cycle is then taken to complete the operation. During the end-op cycle, the next instruction executed is decoded off R.

If D(21,22) = 11 and the branch is unsuccessful, the contents of B are transferred to T, from where they are transferred to the GPR per E(8–11). If IC(21,22) = 11 or 00, a 3-cycle storage request is issued per the IC. [If IC(21,22) equals a value other than 11 or 00, a normal end cycle is taken.] This request is for the doubleword that sequentially follows the doubleword in main storage containing the branch instruction. The 'execute' trigger is tested next. From this point on, the operation is the same as that previously discussed for an unsuccessful branch.

Assume, now, that when IC(21,22) was initially tested for either 11 or 00, neither of these values was present (Diagram 5-505, Sheet 2). Again D(21,22), the PAL's, and E(7) are tested to determine whether the operation is a successful branch and where the next instruction is located in the doubleword address being branched to. Assume that D(21,22) = 11 and the branch is successful (Diagram 5-505, Sheet 3). Operation from this point on is identical with a successful branch, as previously described, when D(21,22) = 11.

Now assume that D(21,22) = 11 and that the branch is unsuccessful. The contents of B are transferred to T, from where they are transferred to the GPR per E(8–11). Because the branch is unsuccessful and the contents of R have not been changed, R still contains the instruction that is located in the halfword following the last halfword of the branch instruction. A normal end-op cycle, therefore, can be taken and the instruction decoded off R.

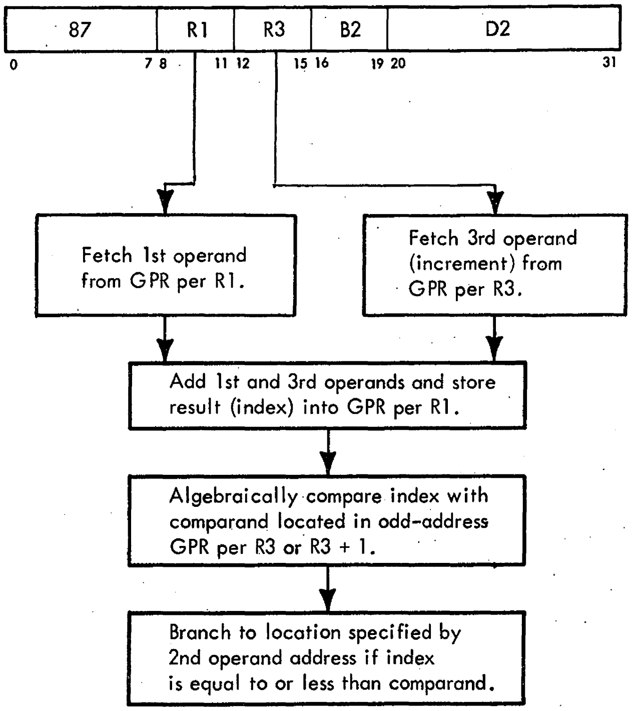
Now assume that D(21,22) did not equal 11 and the branch is successful (Diagram 5-505, Sheet 3). The contents of B are transferred to T, from where they are transferred to the GPR per E(8–11). At this time, the data requested during I-Fetch of the branch instruction is present at the SDBO and is gated to Q. From Q, the correct halfword containing the branch-to instruction is gated into R per D(21,22). The contents of D are then updated by 8 and transferred to the IC to select the next doubleword from main storage. Because the instruction to be executed has just been placed into R and is not yet stable, a branch end-op cycle is taken and the instruction format is decoded off the SDBO.

If the branch is unsuccessful and D(21,22) does not equal 11, the operation is identical with the case where D(21,22) = 11 and the branch is unsuccessful. A normal end-op cycle is taken.

BRANCH ON INDEX LOW OR EQUAL, BXLE (87)

- Add increment (3rd operand; in GPR, per R3) to 1st operand (in GPR, per R1), algebraically compare result (index) with comparand (in odd-address GPR specified by R3 or R3 + 1), and, if index is equal to or is less than comparand, branch to location specified by 2nd operand address.

- RS format:

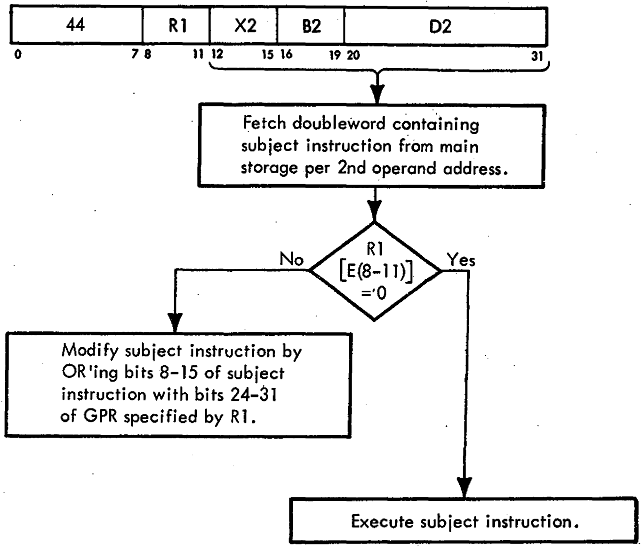

87
R1
R3
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch 1st operand from GPR per R1.
Fetch 3rd operand (increment) from GPR per R3.
Add 1st and 3rd operands and store result (index) into GPR per R1.
Algebraically compare index with comparand located in odd-address GPR per R3 or R3 + 1.
Branch to location specified by 2nd operand address if index is equal to or less than comparand.

- Conditions at start of execution:
  Branch address is in D.
  3-cycle storage request has been issued per D for branch-to instruction.
  1st operand is in S and T.
  First 16 bits of instruction are in E.
- Sum of 1st and 3rd operands is always stored whether branch is successful or not.
- Comparand address (R3 or R3 + 1) must be odd.

The Branch on Index Low or Equal, BXLE, instruction is similar to the BXH instruction except that branching occurs on a low or equal result (Diagram 5-505).

EXECUTE, EX (44)

- Execute subject instruction at location specified by 2nd operand address. Subject instruction may be modified by 1st operand (in GPR, per R1) if E(8–11) is not equal to zero.
- RX format: (See adjoining column.)
- Conditions at start of execution:
  Address of subject instruction is in D.
  3-cycle storage request for subject instruction has been issued per D.
  1st operand is in S and T.
  First 16 bits of instruction are in E.


44
R1
X2
B2
D2
0
7 8
11 12
15 16
19 20
31
Fetch doubleword containing subject instruction from main storage per 2nd operand address.
R1 [E(8-11)] ='0
No
Yes
Modify subject instruction by OR'ing bits 8-15 of subject instruction with bits 24-31 of GPR specified by R1.
Execute subject instruction.

- Modification of subject instruction is accomplished by OR'ing bits 8–15 of subject instruction with bits 24–31 of 1st operand.
- If subject instruction is an Execute instruction, a program execute interruption occurs.
- If effective address of Execute instruction is odd, a program specification interruption occurs.
- 'Execute' trigger is set to indicate next instruction is subject of Execute instruction.
- 'PSC' trigger is set to indicate that Q data is not valid and needs to be refilled.
- If program interruptions are pending, normal end-op cycle is taken; if not, branch end-op cycle is taken.

The Execute, EX, instruction, which has an RX format with an op code of 44, executes a designated instruction whose address in main storage is the second operand address. This designated instruction is referred to as a subject instruction and can be modified by the contents of the first operand located in the GPR register specified by R1. Modification of the subject instruction is accomplished by OR'ing bits 8–15 of the subject instruction with bits 24–31 of the first operand. If R1 = 0, no modification takes place. The subject instruction may be 16, 32, or 48 bits long. If the subject instruction is another Execute instruction, a program execute interruption occurs and operation is suppressed. If the effective address of the EX is odd, a program specification interruption occurs.

At the start of execution, the first 16 bits of the instruction are in E, the first operand is in S and T, the address of the subject instruction is in D, and a 3-cycle storage request for the subject instruction has been issued per D (Diagram 5-506, FEMDM). At the beginning of execution, a test for program specification and execute interruptions is made. If the program specification interruption is present (effective address of Execute instruction is odd), a program interruption occurs and the operation is suppressed. If a program execute interruption is present (the Execute instruction is the subject instruction of an Execute instruction), a program execute interruption occurs and the operation is suppressed. If no interruptions are present, the operation continues. The STC is loaded to 111, allowing the transfer of T(56–63) to the serial adder for modification of the subject instruction if modification is to be accomplished. The contents of D are now transferred to the parallel adder and updated by 8 to address the doubleword that follows the doubleword containing the subject instruction of the Execute instruction. This value is then transferred to D. PAL(61–63) is now transferred to the ABC to select the correct byte for modification of the subject instruction.

D(21,22) is tested to determine whether the subject instruction is contained in the last halfword of the doubleword that was requested during I-Fetch, or in some halfword other than the last. If D(21,22) = 11, the subject instruction is in the last halfword; if any other value, the subject instruction is in some other halfword. First assume that D(21,22) = 11. Because the subject instruction is located in the last halfword of the doubleword addressed during I-Fetch, there is a possibility that part of the instruction is contained in the next doubleword to be addressed. This possibility exists if the subject instruction has a format other than RR. Therefore, the next few operations determine the format of the subject instruction. By doing these tests, an extra request can be prevented, one that can cause an invalid address or protection check if the instruction has an RR format.

At this time, the data requested during I-Fetch is present at the SDBO and can be gated into Q and AB. From Q, the last halfword is transferred to R. The sixth byte in AB is then transferred to the serial adder per the ABC. Minus 64 (1100 0000) is sent to the serial adder, where it is logically AND'ed with the op code of the subject instruction. If the op code denotes an RR format, SAL should now equal zero. 1 is then added to the ABC to transfer the last halfword of AB to the serial adder if the instruction is to be modified.

E(8–11) is now tested to see whether the subject instruction is to be modified. If E(8–11) = 0000, the subject instruction is not to be modified; if any other value, the instruction is to be modified. First assume that E(8–11) = 0000. SAL(0–7) is now tested. Recall that the value in SAL indicates whether the subject instruction has an RR format, and recall that it has already been determined that the subject instruction is contained in the last halfword of Q. Therefore, assume that SAL(0–7) = 0. Because this value indicates that the subject instruction is an RR instruction, there is no need to issue a storage request for the next instruction because there is no information in that doubleword that will affect the operation of the RR instruction. The request for the next doubleword occurs during I-Fetch of the RR instruction.

The last byte in AB is then transferred to T via the serial adder per the ABC and STC. From T, the data is transferred to R. The 'PSC' and 'execute' triggers are set. A test is made for a pending program interruption. If one exists, a program interruption cycle is taken; if there is no interruption, the contents of R are transferred to E and STAT G is set. The setting of STAT G prevents the occurrence of interruptions and the premature prefetching of the next instruction from interfering with the execution of the subject instruction. A branch end-op cycle completes the operation.

Now assume that SAL(0–7) does not equal zero, indicating that the subject instruction has some format other than RR. So that the complete word may be decoded before instruction execution, the doubleword that sequentially follows the subject instruction must be requested. The last byte in AB is transferred to T via the serial adder per the ABC and STC. The 'PSC' trigger is set (insuring return to the main instruction flow upon execution of the subject instruction). This action causes an ASC exceptional condition branch to occur during the I-Fetch following execution of the subject instruction. The ROS microprogram subtracts the correct amount from the IC to select the doubleword containing the instruction following the Execute instruction, and issues a request for that doubleword. A 3-cycle storage request is issued for the next sequential doubleword. The microprogram waits (two storage cycles) until the data requested is present at the SDBO, at which time the data is gated to Q. The 'execute' trigger is set, and a test is made for a pending interruption. If an interruption is present, a program interruption cycle is taken; if not, the data in R is transferred to E and STAT G is set. A branch end-op cycle completes the operation.

If when E(8–11) is tested some value other than 0000 is found, the subject instruction is to be modified. SAL(0–7) is tested to determine whether the subject instruction has an RR format. Assume an RR format. The last byte in AB is then transferred to the serial adder per the ABC. This byte is then OR'ed with the last byte of ST which was transferred to the serial adder per the STC. The results of this OR'ing are then transferred to ST per the STC. The data in T is transferred to R. The 'execute' trigger is set, and the operation continues in the same manner described previously.

Now assume that SAL(0–7) contains some value other than zero. In this case, after the subject instruction is modified, a storage request for the next doubleword must be made. The microprogram waits (two storage cycles) until the data is present at the SDBO, after which the data is transferred to Q. From this point on, the operation is identical with that described for an RR instruction.

Now return to the point where D(21,22) is tested to see whether the subject instruction occupies the last halfword of the doubleword requested during I-Fetch. If D(21,22) equals some value other than 11, a storage request is not issued during execution of the Execute instruction. E(8–11) is now tested to determine whether the subject instruction is to be modified. If E(8–11) does not equal 0000, the instruction must be modified. After the data is placed into Q and AB, and the correct data is placed into the serial adder per the ABC, operation is identical with that of an RR instruction that is to be modified.

If E(8–11) = 0000, however, the subject instruction is not to be modified. Therefore, the correct halfword in Q need only be transferred to R per D(21,22). The 'PSC' and 'execute' triggers are set, and the test for a pending interruption is made. The operation continues in the same manner described previously.

Section 7. Status Switching Instructions

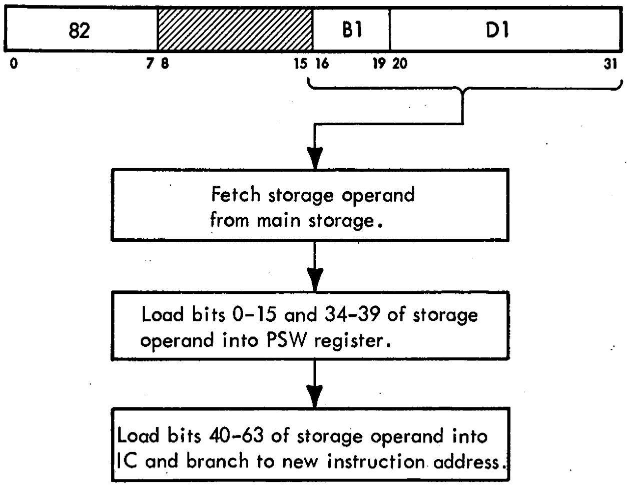
The 10 status switching instructions can change the status of the CPU, the channels, the external units, and the data in main storage:

1. The Load PSW, Set Program Mask, Set System Mask, and Supervisor Call instructions control the status of the CPU.
2. The Set Storage Key, Insert Storage Key, and Test and Set instructions control the status of the data in main storage.
3. The Write Direct and Read Direct instructions control the status of external units (and also transfer data bytes).
4. The Diagnose instruction controls the status of the CPU and channels.

For a discussion of operand addressing, program states, PSW, status switching, instruction formats, data flow, program interruptions, and CC's, see Chapter 1.

LOAD PSW, LPSW (82)

- Load doubleword storage operand (designated by storage operand address) into CPU, thus replacing current PSW, and branch to new instruction sequence.
- SI format:


82
B1
D1
0
7 8
15 16
19 20
31
Fetch storage operand from main storage.
Load bits 0-15 and 34-39 of storage operand into PSW register.
Load bits 40-63 of storage operand into IC and branch to new instruction address.

- Use same microprogram as IPL, PSW RESTART, and interruption operations.
- Conditions at start of execution:
  First 16 bits of instruction are in E.
  Storage operand address is in D.
  Storage request has been issued per D.

The Load PSW, LPSW, instruction loads into the CPU the doubleword from main storage designated by the storage operand address. The doubleword (a new PSW) becomes the current PSW for the next instruction. Bits 40–63 of the doubleword become the address of the next instruction. The new PSW will not allow interruptions until after the LPSW instruction is executed. When the doubleword being loaded has a 1 in position 14 or 15, the CPU enters the Wait state or the Problem state, respectively; this is the only instruction available for entering these states.

Diagram 5-601, FEMDM, is a flowchart of the LPSW instruction. At the start of execution, the first 16 bits of the instruction are in E, the storage operand address is in D, and a storage request for the storage operand has been issued per D.

Tests for a privileged operation check and a specification check are made at the beginning of the execution. The CPU must be in the Supervisor state, and the address of the LPSW instruction must have three low-order 0's; otherwise, a program interruption results.

When the data (new PSW) requested during I-Fetch is available on the SDBO, it is gated to ST. The new PSW is then assembled by transferring S(0–15) and T(34–39) to the PSW register, transferring T(40–63) to the IC, and resetting the interruption request triggers. The ILC remains unchanged until an interruption occurs.

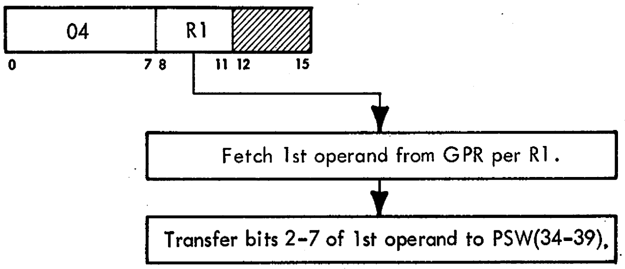
The instruction next makes a 3-cycle storage request per the IC to fill Q with the next instruction. Although the I-Fetch checking circuits have been activated previously, interruptions are inhibited by the setting of the 'I-Fetch request' trigger. The contents of T are now transferred to D so that, when the data is available, D will be able to select the correct halfword in Q to transfer to R. The IC is incremented by 8 to select the next doubleword to be fetched to refill Q. The 'PSC' and 'execute' triggers are reset. These triggers are set if the LPSW instruction is the subject instruction of an Execute instruction. Because the next instruction to be performed is determined by the new PSW being loaded, the triggers must be reset to prevent the instruction following the Execute instruction from being performed after the LPSW instruction.

From this point on, the LPSW instruction execution is the same as for the Branch on Count instruction following a successful branch. When the requested data is available, it is gated to Q, and then transferred to R per D. A decision is now made to refill Q under control of this instruction if D(21,22) = 11. If Q does not need refilling, the instruction terminates with a branch end op. If, however, Q is refilled, a branch end op is not necessary and the instruction terminates with a normal end op.

The LPSW microprogram used by the LPSW instruction is also entered by the interruption, PSW RESTART, and IPL microprograms. The storage operand address is forced to zero by the IPL microprogram and by the depression of the PSW RESTART pushbutton. One of the five new PSW addresses in permanent storage is generated by the interruption microprogram.

SET PROGRAM MASK, SPM (04)

- Replace CC and program mask (bits 34–39) of current PSW with bits 2–7 of 1st operand (in GPR per R1).
- RR format:

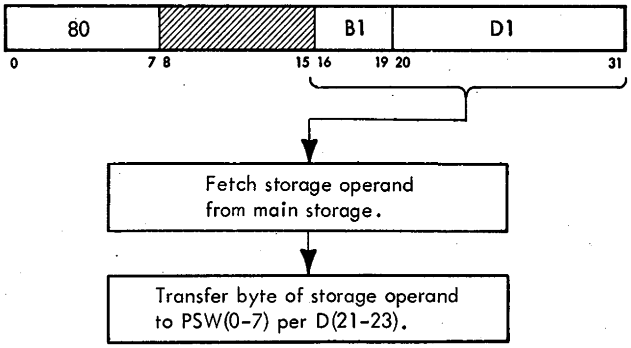

04
R1
0
7 8
11 12
15
Fetch 1st operand from GPR per R1.
Transfer bits 2-7 of 1st operand to PSW(34-39).

- Conditions at start of execution:
  Instruction is in E.
  1st operand is in A, B, and D.
  2nd operand is not used.

The Set Program Mask, SPM, instruction replaces the CC and the program mask of the current PSW with the contents of the GPR addressed by R1 (Diagram 5-602, FEMDM). Bits 2 and 3 of the GPR become the new CC in the current PSW, and bits 4–7 become the new program mask. Bits 2–7 of the first operand may have been loaded from the PSW register by a previous Branch and Link instruction.

SET SYSTEM MASK, SSM (80)

- Replace system mask (bits 0–7) of current PSW with byte from location designated by storage operand address.
- SI format:

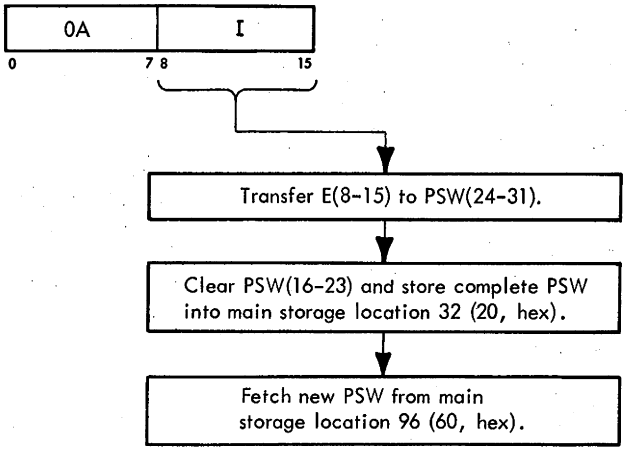

80
B1
D1
0
7 8
15 16
19 20
31
Fetch storage operand from main storage.
Transfer byte of storage operand to PSW(0-7) per D(21-23).

- Conditions at start of execution:
  First 16 bits of instruction are in E.
  Storage operand address is in D.
  Storage request has been issued per D.

The Set System Mask, SSM, instruction replaces the system mask of the current PSW with the addressed storage operand byte (Diagram 5-603, FEMDM). The addressed storage operand byte is fetched from main storage and placed into PSW(0–7) via AB, the serial adder, and S(0–7). The ABC is set per D(21–23) to select the correct byte in AB to gate to the serial adder.

SUPERVISOR CALL, SVC (0A)

- Cause supervisor call interruption; replace old PSW(24–31) with I-field (bits 8–15) of instruction, providing interruption code.
- RR format:

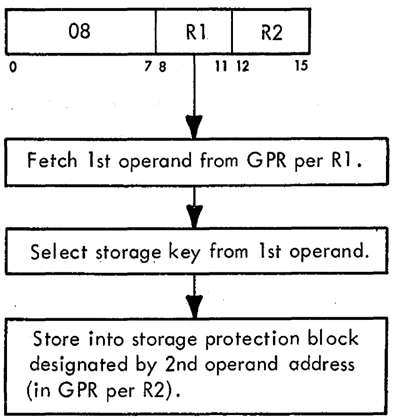

0A
I
0
7 8
15
Transfer E(8-15) to PSW(24-31).
Clear PSW(16-23) and store complete PSW into main storage location 32 (20, hex).
Fetch new PSW from main storage location 96 (60, hex).

- Conditions at start of execution:
  Instruction is in E.
  E(8–15) is interruption code.

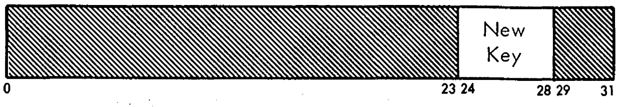
The Supervisor Call, SVC, instruction causes a supervisor call interruption at end-op (Diagram 5-604, FEMDM). The 'supervisor call' trigger is set and, if a timer exceptional condition, a machine check interruption, or a program protection interruption is not pending, the CPU takes a supervisor call interruption. During that operation, bits 8–15 of the instruction, still in E(8–15), are stored as the interruption code into the supervisor call old PSW. The new PSW usually switches the CPU to the Supervisor state. Refer to Section 1 of this Chapter for a discussion of the supervisor call interruption.

SET STORAGE KEY, SSK (08)

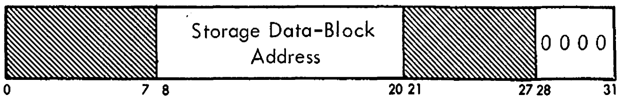
- Set storage key (bits 24–28 of 1st operand; in GPR per R1) for 2048-byte storage block (addressed by bits 8–20 of 2nd operand; in GPR per R2) into storage protection logic in main storage.
- RR format:


08
R1
R2
0
7 8
11 12
15
Fetch 1st operand from GPR per R1.
Select storage key from 1st operand.
Store into storage protection block designated by 2nd operand address (in GPR per R2).

- Conditions at start of execution:
  Instruction is in E.
  1st operand is in A, B, and D.
  2nd operand is in S and T.
- Format of LS word addressed by R1:



- Format of LS word addressed by R2:



- New key is set twice because of two-way interleaving.

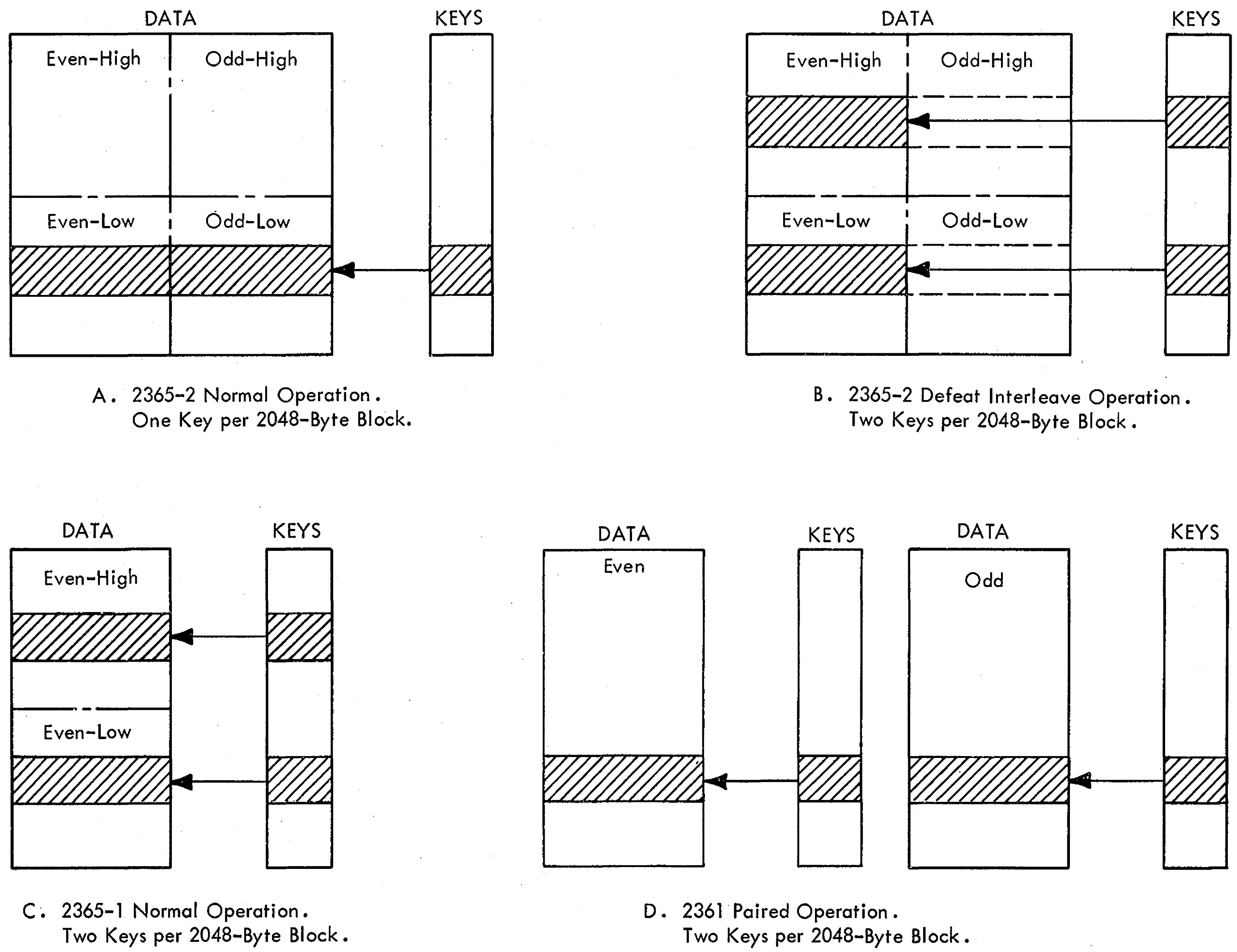
The Set Storage Key, SSK, instruction sets the key of the storage block addressed by the second operand according to the key in the GPR designated by R1. Bits 8–20 of the second operand address a block of 2048 storage bytes. (It is not necessary for the second operand to address the first byte in the block.) During the SSK instruction, bits 21–27 of the second operand, which address doublewords in the storage block, are ignored. Bits 28–31 of the second operand, however, must be 0's, or a program specification interruption occurs. The new storage key is obtained from bits 24–28 of the first operand; the rest of the operand is ignored.

Diagram 5-605, FEMDM, is a flowchart of the SSK instruction. At the start of execution, the instruction is in E, the first operand is in A, B, and D, and the second operand is in S and T. The ABC is set to 7 (111) to allow selection of the new storage key byte from B. The STC is set to 3 (011) to select the low-order byte of the second operand in S, the byte to be tested for a specification check. D is set to the address of the storage block minus 8; this step allows construction of a loop later in the microprogram. If the system is in the Problem state, a program privileged operation interruption occurs. If the system is in the Supervisor state and the address of the storage block does not specify an even doubleword boundary [S(28–31) = 0], a program specification interruption occurs. Otherwise, the execution of the instruction continues by transferring the new key from B to F, via the serial adder.

The new key is now in position to be set into the storage unit. D is incremented by 8, and a 4-cycle storage request is issued per the block address in D. The 'set key' trigger is set, causing a 'set key' signal to be sent to main storage with the 'storage select' signal, and all of the mark triggers are set. As a result, the selected storage unit recognizes its selection as being a request to change one of the keys stored in its storage protection area. The new key is gated from F to the 'key in' bus for the use of the selected storage unit.

Execution of this instruction may change the protection status of unprocessed instructions already in the CPU. Therefore, the 'PSC' trigger is set to force a program store compare exceptional condition during the next I-Fetch to refill Q.

At this point, the setting of one key into main storage has taken place. When operating normally with a 2365 Processor Storage, Model 2, one setting of a new key for 2048 contiguous bytes of data is sufficient, as shown in Figure 3-27(A). However, the SSK instruction in the 2065 always sets the key twice, first for an even doubleword address and then again for the succeeding odd doubleword address. This duplicated setting of the key is done because the System/360 Model 65 has two-way interleaving of



A. 2365-2 Normal Operation.
One Key per 2048-Byte Block.

B. 2365-2 Defeat Interleave Operation.
Two Keys per 2048-Byte Block.

C. 2365-1 Normal Operation.
Two Keys per 2048-Byte Block.

D. 2361 Paired Operation.
Two Keys per 2048-Byte Block.

Figure 3-27. Storage Protection Key Assignments

'storage select' signals based on even and odd storage addresses. There are three cases when contiguous even and odd storage addresses are not in the same storage-protection block in main storage:

1. When the defeat interleave maintenance aid is used, contiguous even and odd storage addresses in the 2365-2 are in two different protected blocks within the same unit, as shown in Figure 3-27(B).
2. When operating normally with a 2365-1, interleaving is always defeated because the 2365-1 has only half the core size of the 2365-2, as shown in Figure 3-27(C).
3. When pairs of 2361 Core Storage (LCS) are attached, even and odd addresses reside in two different storage units, as shown in Figure 3-27(D).

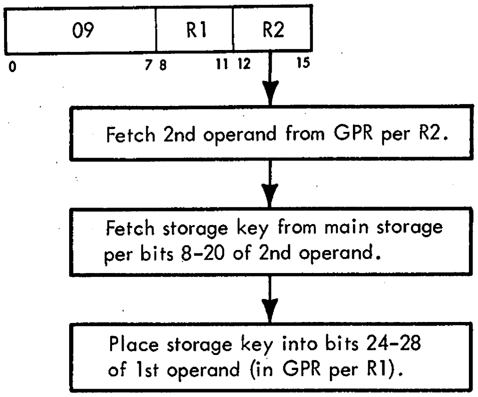
Therefore, after setting the first key, the SSK instruction again increments D by 8 and another 'set key' signal is issued to storage. For the case shown in Figure 3-27(A), the same key location is set again; otherwise, a different key location is set.

Although the System/360 Model 65 always uses two-way interleaving, there is provision in the microprogram for future four-way interleaving which would necessitate setting the key four times. After the key is processed twice, STAT D would be tested. If reset, it would be set, and the new key would be processed twice again into succeeding even and odd storage addresses. However, STAT D now is found set when tested; it is set when the 'set key' trigger is first set and is not reset until after the SSK instruction is completed.

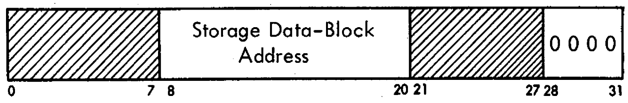
INSERT STORAGE KEY, ISK (09)

- Insert storage protection key for 2048-byte storage block, addressed by bits 8–20 of 2nd operand (in GPR per R2), into bits 24–28 of 1st operand (in GPR per R1).

- RR format:


09
R1
R2
0
7 8
11 12
15
Fetch 2nd operand from GPR per R2.
Fetch storage key from main storage per bits 8-20 of 2nd operand.
Place storage key into bits 24-28 of 1st operand (in GPR per R1).

- Conditions at start of execution:
  Instruction is in E.
  1st operand is in A, B, and D.
  2nd operand is in S and T.
- Format of LS word addressed by R2:



- Storage key is inserted into bits 24–28 of 1st operand.
- Bits 0–23 of 1st operand remain unchanged; bits 29–31 are cleared.
- Key is fetched twice because of two-way interleaving.

The Insert Storage Key, ISK, instruction inserts the storage key addressed by the second operand into bits 24–28 of the first operand (in GPR, per R1). Bits 8–20 of the second operand address a block of 2048 bytes in main storage. Bits 0–7 and 21–27 of the second operand are ignored, whereas bits 28–31 must be 0's or a program specification interruption occurs. The five-bit storage key is set into bits 24–28 of the first operand; bits 29–31 are cleared.

Diagram 5-606, FEMDM, is a flowchart of the ISK instruction. At the start of execution, the instruction is in E, the first operand is in A, B, and D, and the second operand is in S and T. The STC is set to 3 (011) in preparation of setting D to the address of the storage block minus 8, thus allowing the construction of a loop later in the microprogram. The first operand is transferred from B to T via the parallel adder, and the STC is set to 7 (111). The contents of T, with the fetched key inserted into byte 7, will later be transferred back into the GPR designated by R1. Before fetching the key, however, program tests are made. If the system is in the Problem state, a program privileged-operation interruption occurs. If the system is in the Supervisor state and the address of the storage block does not specify an even doubleword boundary [S(28–31) = 0], a program specification interruption occurs. Otherwise, the execution continues. F and the last byte of T are set to 0's. The fetching of the key can now begin.

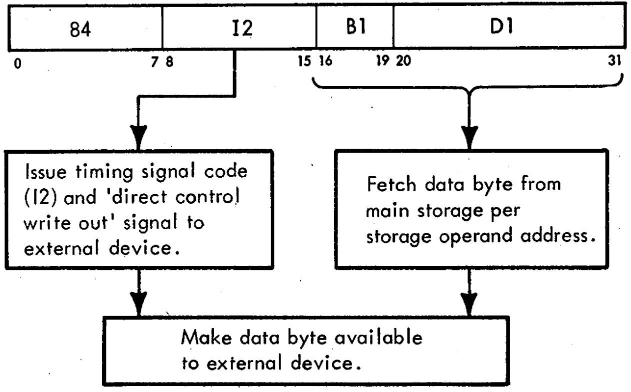
The contents of F are logically OR'ed into the last byte of T via the serial adder, again setting 0's. (This step would be important if the 2065 used four-way interleaving of 'storage select' signals; as with the SSK instruction, the ISK instruction has an unused loop that would result in the fetching of the key four times.) D is incremented by 8 and a 3-cycle storage request is issued per the block address in D. The 'insert key' trigger is set, causing an 'insert key' signal to be sent with the 'storage select' signal. As a result, the selected storage unit recognizes its selection as being a fetch request for one of the keys stored in its storage protection area. The CPU waits until the 'key advance' signal from storage is received and then gates the key from the 'key out' bus into F.

Because the key can be in two different locations, as explained for the Set Storage Key instruction, it is always fetched twice. This scheme ensures that odd addresses can be successfully accessed. After the fetched key is logically OR'ed into T from F, the key for the next doubleword is fetched exactly as before.

When the key is again available, this time from an odd doubleword address, it is logically OR'ed with the key first fetched. The thus modified first operand is then returned to the GPR from which it came. A normal end op completes the execution of the instruction.

WRITE DIRECT, WRD (84)

- Send 'direct control write out' signal and timing signal code (I2 in instruction) to external device for about 0.8usec; make 1 data byte from storage (per storage operand address) available to external device until next WRD is executed.
- SI format:


84
I2
B1
D1
0
7 8
15 16
19 20
31
Issue timing signal code (I2) and 'direct control write out' signal to external device.
Fetch data byte from main storage per storage operand address.
Make data byte available to external device.

- Conditions at start of execution:
  First 16 bits of instruction are in E.
  Storage operand address is in D.
  Storage request has been issued per D.
- 8 data bits remain available in G until another WRD instruction is issued.
- 'Direct control write out' signal and timing signal code may be considered instruction for external device.

Note: The operation of the Write Direct instruction is modified if the Multisystem Feature is installed and if the CPU is not in Model 65 Mode. Refer to Chapter 4, Section 2, Multisystem Feature, "Write Direct Instruction (Not Model 65 Mode)".

The Write Direct, WRD, instruction sends the eight bits of the immediate operand plus a 'direct control write out' signal to an external device for about 0.8usec. During that time, the storage operand becomes available to the external device and remains available until the next WRD instruction is executed. No parity is presented with either the immediate or storage operands when sent to the external device.

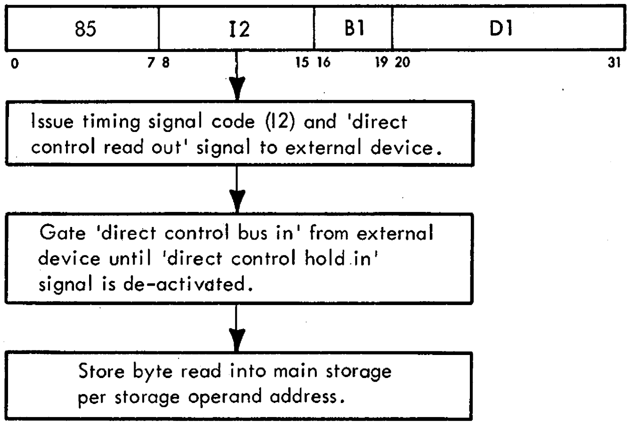
Diagram 5-607, Sheet 1, A, FEMDM, is a flowchart of the WRD instruction. At the start of execution, the first 16 bits of the instruction are in E, the storage operand address is in D, and a storage request for the storage operand has been issued per D. A privileged operation test is first made. If the system is in the Problem state, a program privileged-operation interruption occurs. Also, the validity of the instruction is tested. If the Direct Control Feature is not installed or if the DISABLE DIRECT CONTROL switch is in the active (down) position, the instruction is invalid and a program operation interruption occurs. Otherwise, instruction execution continues.

The 'timing gate' trigger is set for about 0.8usec. During this time, the immediate operand, containing the timing signal code, is sent from E to the external device via the 'timing signal bus out' lines; a 'direct control write out' timing signal is also sent to the external device. These nine timing signals may be considered an instruction for the external device.

While the nine timing signals are being sent, the storage operand is transferred from the SDBO to G via ST and the serial adder. Note that the 'direct control write out' signal is sent when the data in G is being changed. There is no outgating from G to the external device; the contents of G are always available on the 'direct control bus out' lines. G is reset only by the next WRD instruction. After the 'timing gate' trigger is reset, the instruction ends with a normal end op.

READ DIRECT, RDD (85)

- Send 'direct control read out' signal and timing signal code (I2; in instruction) to external device for about 0.6 usec; store 1 data byte from external device into storage (per storage operand address) when 'direct control hold in' signal is absent.
- SI format:


85
I2
B1
D1
0
7 8
15 16
19 20
31
Issue timing signal code (I2) and 'direct control read out' signal to external device.
Gate 'direct control bus in' from external device until 'direct control hold in' signal is de-activated.
Store byte read into main storage per storage operand address.

- Conditions at start of execution:
  First 16 bits of instruction are in E.
  Storage operand address is in D.
  Storage request has been issued per D.
- Timing signal code and 'direct control read out' signal may be considered instruction for external device.

Note: The operation of the Read Direct instruction is modified if the Multisystem Feature is installed and if the CPU is not in Model 65 Mode. Refer to Chapter 4, Section 2, Multisystem Feature, "Read Direct Instruction (Not Model 65 Mode)".

During the Read Direct, RDD, instruction, nine timing signals are sent to an external device. Next, a byte of data is read from an external device until the 'direct control hold in' signal is de-activated; the byte of data is then stored into main storage.

Diagram 5-607, Sheet 1, B, FEMDM, is a flowchart of the RDD instruction. At the start of execution, the first 16 bits of the instruction are in E, the storage operand address is in D, and a storage request has been issued per D. A privileged operation test is first made. If the system is in the Problem state, a program privileged-operation interruption occurs. Also, the validity of the instruction is tested. If the Direct Control Feature is not installed or if the DISABLE DIRECT CONTROL switch is in the active (down) position, the instruction is invalid and a program operation interruption occurs. Otherwise, instruction execution continues. The STC is set per D(21–23), indicating the location in ST that the byte of read-direct data will occupy.

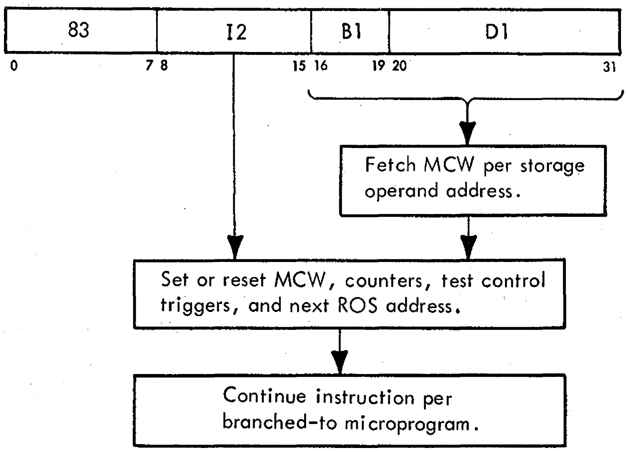
The 'timing gate' trigger is set for about 0.6 usec. During this time, the immediate operand, containing the timing signal code, is sent from E to the external device via the 'timing signal bus out' lines; a 'direct control read out' signal is also sent to the external device. These timing signals may be considered an instruction for the external device. While the 'timing gate' trigger is set, an address store compare test is made; its purpose is to cause Q to be refilled during the next I-Fetch if the byte being changed in main storage is also in Q.

The 'direct control bus in' lines, containing the byte of data to be read from the external device, are gated to F, and a mark trigger is set per the STC. These two actions continuously cycle as long as the 'direct control hold in' signal is being received from the external device. Otherwise, instruction execution continues by storing the contents of F into main storage (via the serial adder, ST per STC, and the SDBI), and finishes with a normal end op.

Neither the 'direct control bus in' lines nor the 'external signal in' lines include a parity bit. However, the serial adder generates odd parity for the data byte before it is stored.

DIAGNOSE (83)

- Load word designated by storage operand address into MCW, set or reset certain control triggers, and branch to ROS address specified by MCW.
- SI format:


83
I2
B1
D1
0
7 8
15 16
19 20
31
Fetch MCW per storage operand address.
Set or reset MCW, counters, test control triggers, and next ROS address.
Continue instruction per branched-to microprogram.

- Conditions at start of execution:
  First 16 bits of instruction are in E.
  Storage operand address is in D.
  Storage request has been issued per D.
- Enable and disable various system maintenance aids.
- Available for entering and leaving emulator mode.

The Diagnose instruction has two purposes: it is available to the diagnostic programmer as a maintenance aid and to the system programmer for emulator mode operations. Whichever instruction application is made, the beginning of its execution is basically the same. The immediate operand is used to set or reset control triggers. The storage operand (a doubleword termed the maintenance control word, MCW) is used to set the MCW register, counters, and the address of the next ROS word. The remainder of the execution is determined by the ROS microprogram branched to and the control triggers.

Diagram 5-608, FEMDM, is a flowchart of the Diagnose instruction. For a description of the MCW format, see Section 2 of Chapter 6. For a description of the branches to emulator microprograms, see the applicable Compatibility Feature FETOM.

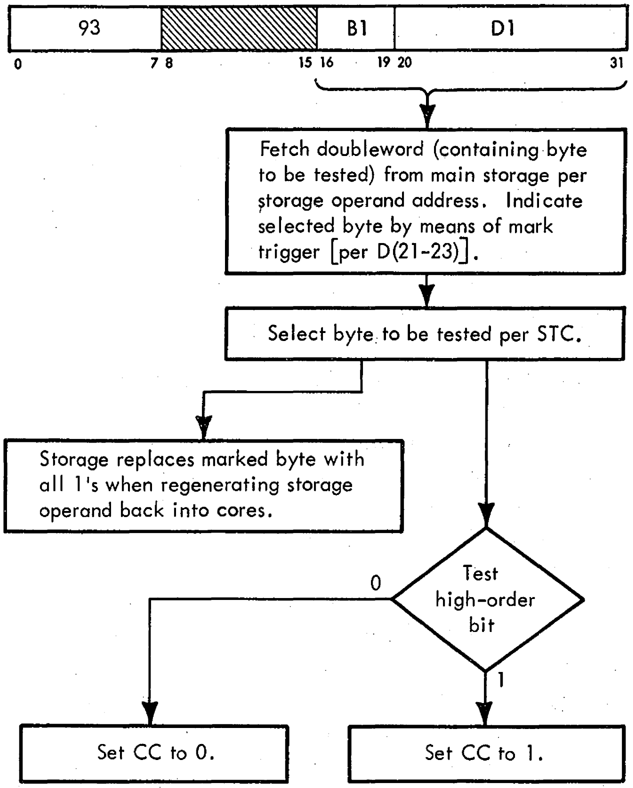
At the start of execution, the first 16 bits of the instruction are in E, the address of the storage operand is in D, and a storage request for the storage operand has been issued per D. If the system is in the Problem state, a program privileged-operation interruption occurs. If the system is in the Supervisor state and the address of the storage operand does not specify a doubleword boundary, a program specification interruption occurs. Otherwise, instruction execution continues by making an address store compare test similar to the test performed during the SS I-Fetch operation. Here it is important for 2067 operations but not for 2065 operations. At this point, the address of the storage operand is no longer needed and is incremented by 8.

The CPU is placed into the scan mode, and the MCW is gated from the SDBO to T (and also to A). The control triggers are now set or reset: T(32–39, 52) is transferred to the MCW register, T(53–57) to the address sequencer, T(58–61) to the FLT counter, T(62, 63) to the FLT clock, E(8, 9) to the two interleave control triggers, and E(10) to the 'diagnose FLT' trigger. The 'scan counter control' and 'diagnose' triggers are set. They remain set depending on the application. The CPU is now taken out of the scan mode and a ROS branch is taken.

The ROS branch address is transferred from T(40–51) to ROSAR(0–11). If the instruction is being used to reset a previously set trigger, the address of a normal end op, 010 (hex), is usually specified and the execution is completed. However, any ROS address may be specified, depending on the application.

TEST AND SET, TS (93)

- Test high-order bit (bit 0) of storage operand byte (in storage), set CC according to state of tested bit, and set addressed byte back into storage as all 1's.
- SI format:


93
B1
D1
0
7 8
15 16
19 20
31
Fetch doubleword (containing byte to be tested) from main storage per storage operand address. Indicate selected byte by means of mark trigger [per D(21-23)].
Select byte to be tested per STC.
Storage replaces marked byte with all 1's when regenerating storage operand back into cores.
Test high-order bit
0
1
Set CC to 0.
Set CC to 1.

- Conditions at start of execution:
  First 16 bits of instruction are in E.
  Storage operand address is in D.
- CC setting:
  High-order bit = 0: CC = 0.
  High-order bit = 1: CC = 1.

The Test and Set, TS, instruction tests the high-order bit of a single byte in main storage and then sets the byte tested to all 1's. The byte to be tested and set to 1's is specified in the storage operand address. The result of the test of the high-order bit is recorded in the CC: if the high-order bit is a 0, the CC is set to 0; if a 1, the CC is set to 1.

Diagram 5-609, FEMDM, is a flowchart of the execution of the Test and Set instruction. The first 16 bits of the instruction are in E, and the storage operand address is in D. The immediate operand and the storage operand requested during I-Fetch are not used. Instead, a storage request per D is made during execution accompanied by a 'test and set' signal. This action allows no other access to this storage location between the fetching and regeneration of the byte. Before issuing the storage request, a mark trigger is set per D(21–23), via the STC, for use later by the storage unit.

The storage unit performs a unique regeneration operation for the Test and Set instruction. The addressed storage doubleword is fetched and set unaltered onto the SDBO exactly as during a fetch operation. Unlike a normal fetch operation, however, the storage unit uses the mark bit supplied by the CPU to designate the byte to be changed. When regenerating the 72-bit word, the storage unit sets the designated byte in core storage to all 1's. Thus, the storage unit does a combination store and fetch operation. The storage protection facility operates as normal for a store operation.

While generating the request for the storage operand, an address store compare test is started, similar to the test performed during the SS I-Fetch operation. Its purpose is to detect if the storage operand is buffered in Q. If it is, the 'PSC' trigger is set, forcing an exceptional condition during the next I-Fetch, thus refilling Q.

The bit to be tested must be placed into T(32). Therefore, the second operand is gated from the SDBO to AB where the proper byte is selected and transferred to T(32–29) via the serial adder and F. An early end op is taken, overlapping the setting of the CC.

The four I/O instructions are executed by the same ROS sequence (CLD QK021). An I/O instruction may be executed only when the CPU is in the Supervisor state. The first operation of an I/O instruction, therefore, is to determine the state of the CPU. If the CPU is not in the Supervisor state, a privileged-operation check occurs, causing a program privileged-operation interruption. If the CPU is in the Supervisor state, execution of an I/O instruction begins by setting the 'timing gate' trigger (Diagram 3-8, FEMDM). The 'timing gate' trigger gates a 'channel select' signal to select 1 of 7 channels, as determined by bits 16–23 of the operand address (base plus displacement). As long as the 'timing gate' trigger remains set, the CPU waits for a 'release' signal from the channel before initiating the I-Fetch of the next instruction.

At the start of execution, the first 16 bits of the instruction are in E; the first operand is not applicable. The base plus displacement produces a 32-bit result, the high-order 8 bits of which are not significant for storage addressing; hence only the low-order 24 bits are stored into D. Because this is an I/O instruction, the address in D is the address of the channel and I/O unit and is not to be interpreted as a main storage address.

For an I/O instruction, only the low-order 11 bits are significant and valid. Although an 8-bit byte has been reserved for the channel address, D(8–15), only seven channels can be attached to a System/360, Model 65. Hence only D(13–15) is necessary to address all possible channels, and all channel addresses above 6 are considered invalid. Therefore, a 1 in any position of D(8–12) indicates an invalid channel address, the 'channel select' signal is blocked, and the operation is terminated with a condition code of 3.

A properly addressed channel may also be unavailable because the channel:

1. Is physically disconnected from the CPU.
2. Power is down.
3. Meter is disabled.
4. Is in the test mode (the CE AUTO/TEST switch is in the TEST position).

If the addressed channel is unavailable for any of the above reasons, the CPU generates a CC of 3 and a 'release' signal so that the CPU may proceed.

If the channel is available, the I/O unit address, D(16–23), is gated to the unit address register in the channel. (The I/O unit address is ignored by the channel for the Test Channel instruction.)

At this point, the CPU is awaiting a 'release' signal from the channel, and the channel is initiating the operation determined by the I/O instruction. When the channel operation has been initiated, a 'release' signal is generated for the CPU. The 'release' signal allows the microprogram to advance and reset the 'timing gate' trigger, thus dropping the 'channel select' signal. The CPU is now free to initiate the next instruction, while the channel may continue to execute the specified operation. The CPU and the channel may continue to share main storage under control of the priority function of the BCU. Priority is, from highest to lowest, channels 1, 2, 0, 3, 4, 5, 6, and CPU.

When the I/O operation is completed after the CPU is released, the channel initiates an I/O interruption to store the status and condition code. If the CPU accepts the interruption, the channel is freed for further I/O operations initiated by the CPU; otherwise, the channel is unavailable because of the pending I/O interruption.

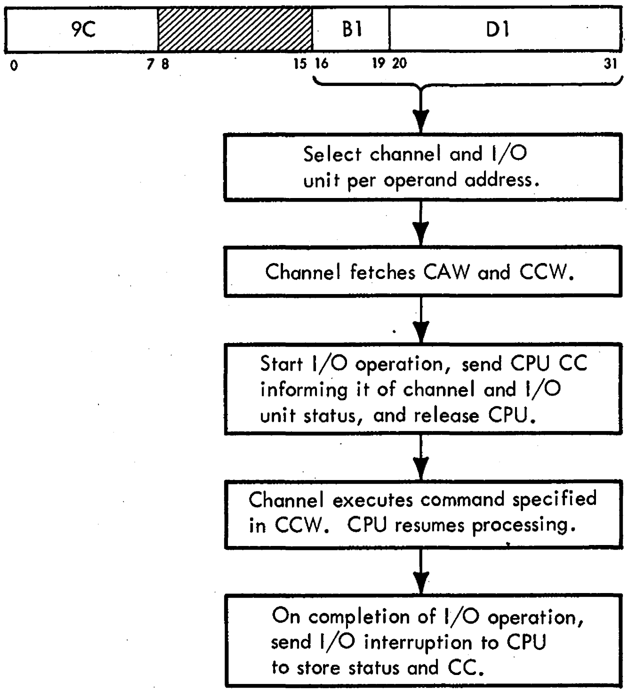
When the 'release' signal enters the CPU, it becomes the 'release CPU' signal. For normal I/O operations, when the 'timing gate' trigger has been set, the ROS microprogram is looping (CLD QK021), and the 'channel select' line is active. The 'release CPU' signal sets the 'release CPU' latch (KX411), the output of which sets ROSAR(11) (DS151). As a result, the microprogram branches and resets the 'timing gate' trigger and proceeds to an end op. The reset 'timing gate' trigger drops the 'channel select' signal and resets the 'release CPU' latch.

When selection of an unavailable channel is attempted, a 'release' signal is not available from the channel. The 'release CPU' latch and a CC of 3 are set by the inactive 'channel available' line. The 'release CPU' latch sets ROSAR(11) as before.

START I/O, SIO (9C)

- Select specified I/O unit and initiate channel command to that unit.

- SI format:


9C
B1
D1
0
7 8
15 16
19 20
31
Select channel and I/O unit per operand address.
Channel fetches CAW and CCW.
Start I/O operation, send CPU CC informing it of channel and I/O unit status, and release CPU.
Channel executes command specified in CCW. CPU resumes processing.
On completion of I/O operation, send I/O interruption to CPU to store status and CC.

- Conditions at start of execution:
  1st 16 bits of instruction are in E.
  1st operand is not applicable.
  Operand address (address of channel and I/O unit) is in D.
- D(13–15) is channel address.
- D(16–23) is I/O unit address.
- If available, channel fetches CAW, specifying CCW location.
- Channel stores status byte if errors in CAW or unit address.
- CC's specify status of channel and I/O unit.
- CC setting:
  Available: CC = 0.
  CSW stored: CC = 1.
  Working: CC = 2.
  Unavailable: CC = 3.

The Start I/O, SIO, instruction selects a specified I/O unit and initiates a channel command to that unit. The channel commands associated with the SIO instruction are: read, read backward, write, sense, control, and transfer in channel.

Diagram 5-701, FEMDM, is a flowchart of the SIO instruction. At the start of execution, the first 16 bits of the instruction are in E and the operand address is in D; the first operand is not applicable. If the channel is available, the unit address is gated to the unit address register in the channel. The channel then fetches the CAW from main storage address 72 (48, hex). The CAW specifies the address of the first CCW and the storage protection key for all channel commands associated with the SIO instruction.

Two operations, fetching the CCW and selecting the I/O unit per the unit address, are now started simultaneously. Fetching of the CCW is initiated by issuing a storage request from the channel to main storage. When the BCU response is received by the channel, the channel places the command address on the SAB and waits for the 'advance' signal. The 'advance' signal is used to place the command information (command code, data address, flags, and counts) into the proper registers in the channel. The 'CCW valid' trigger is set, if there were no errors, to show that the CCW has been received. The CCW information is then checked for correct parity. During the storage operation, the command address was incremented by 1; the updated quantity is now gated back to the command address register in the channel, and may be used to fetch another CCW if chaining was indicated in the first CCW.

The second operation, selection of the proper I/O unit, is started at the same time as fetching of the CCW. Selecting the proper I/O unit is accomplished by placing the unit address on the bus-out to the control unit and issuing an 'address out' signal followed approximately 400ns later by a 'select out' signal. The control unit responds with an 'operation in' signal, which causes the channel to deactivate the 'address out' signal. When the control unit senses the deactivation of the 'address out' signal, it places the address of the selected I/O unit on the bus-in and activates its 'address in' signal. The channel then compares the address it received from the control unit with the address it sent to the control unit to determine that proper selection has been made.

If the addresses are identical, the 'CCW valid' trigger is set, and the operation continues. The CC is placed on the bus-out, and the 'command out' signal is sent to the control unit. The control unit responds with zero status if it can accept the command. The channel then sends a CC of 0 and a 'release' signal to the CPU. When the CPU receives the 'release' signal and the CC is set, the 'timing gate' trigger is reset, dropping the 'channel select' signal. At this point, the CPU and the channel may resume independent operation.

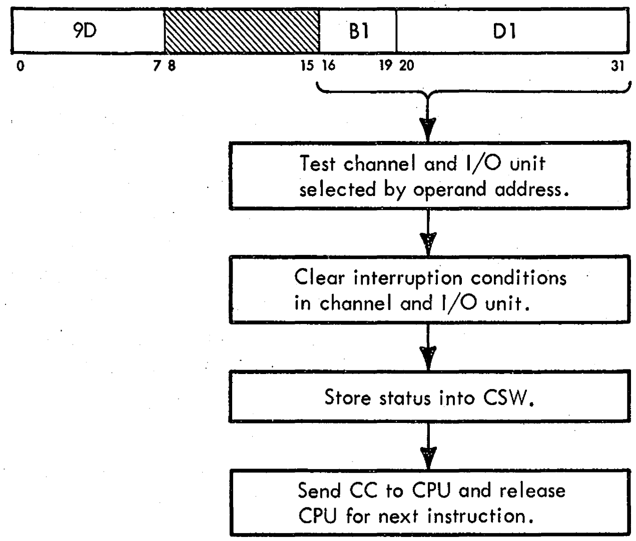
If any errors occurred up to the point where the channel and control unit compare addresses or if the control unit responded with anything other than zero status, a hardware-generated test-I/O code would be placed on the bus-out to the control unit instead of the command code, and the device status would be cleared. The channel would then disconnect from the I/O unit and request a storage cycle. When the BCU response is received, the status information is stored into the CSW, and a CC of 1 and a 'release' signal are sent to the CPU. This action leaves the channel clear and ready to receive another instruction.

If, when the channel is selected, the selected channel is working or unavailable, a CC of 2 or 3, respectively, is sent to the CPU. A 'release' signal is also sent to the CPU, releasing it for execution of other instructions. If any errors are discovered in the CAW or in the unit address, a status byte is stored into the CSW and a CC of 1 is sent to the CPU. A 'release' signal is also sent to the CPU, releasing it for execution of other instructions.

If, when the control unit and the I/O unit are selected, either the control unit or the I/O unit is unavailable, a CC of 3 is sent to the CPU, together with a 'release' signal, thus releasing the CPU for execution of other instructions.

TEST I/O, TIO (9D)

- Clear interruption condition in addressed channel or associated I/O units, and set CC according to status of addressed channel and I/O units.
- SI format:


9D
B1
D1
0
7 8
15 16
19 20
31
Test channel and I/O unit selected by operand address.
Clear interruption conditions in channel and I/O unit.
Store status into CSW.
Send CC to CPU and release CPU for next instruction.

- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is not applicable.
  Operand address (address of channel and I/O unit) is in D.
- D(13–15) is channel address.
- D(16–23) is I/O unit address.
- CC setting:
  Available: CC = 0.
  CSW stored: CC = 1.
  Working: CC = 2.
  Unavailable: CC = 3.

The Test I/O, TIO, instruction clears interruption conditions existing in the addressed channel or addressed I/O unit and stores a CSW in main storage location 64 (40, hex). The CSW is also stored when the channel or I/O unit detects an error during execution of the TIO instruction. The status bits of the CSW identify the error conditions that occurred in the channel or I/O unit. The contents of the CSW pertain to the I/O unit which is addressed by the operand address (base plus displacement) determined during I-Fetch of the TIO instruction.

Diagram 5-702, FEMDM, is a flowchart of the TIO instruction. At the start of execution, the first 16 bits of the instruction are in E and the operand address is in D; the first operand is not applicable.

If the CPU is in the Supervisor state, execution of the TIO instruction begins by setting the 'timing gate' trigger. This trigger sends a 'select' signal to the proper channel as determined by D(13–15). At this time, the unit address [D(16–23)] is sent to the channel. If an interruption is pending, the channel compares this unit address with the unit address it is holding in its unit address register. If they are equal, the interruption condition in the channel is stored into the CSW in main storage location 64 (40, hex), and a CC of 1 is sent to the CPU together with a 'release' signal, releasing the CPU for execution of other instructions. If the addresses are not equal, a CC of 2 is sent to the CPU along with a 'release' signal, allowing the CPU to start processing the next instruction. When the CPU receives the 'release' signal, it resets the 'timing gate' trigger and takes an end-op cycle, completing the operation.

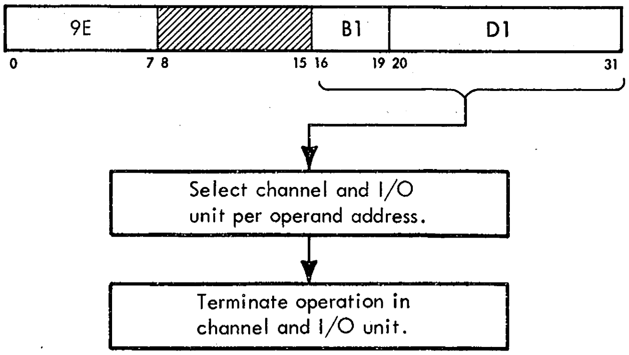
If an interruption is not pending, the unit address sent from the CPU is placed into the unit address register in the channel to select the specified I/O unit. Because this is the TIO instruction, only status is required from the selected I/O unit. If the status of the I/O unit is other than all 0's, a CSW is stored into main storage location 64 (40, hex) and a CC of 1 is sent to the CPU. If the I/O unit status is all 0's, a CC of 0 is sent to the CPU.

A 'release' signal is then sent to the CPU. When the CPU receives the 'release' signal, the 'timing gate' trigger is reset and an end-op cycle is taken, completing the operation. If the status returned to the channel when the test I/O command is issued is all 0's, a CC of 0 is sent to the CPU. This value indicates that the I/O unit is available. The CPU is then released for further instruction execution.

HALT I/O, HIO (9E)

- Terminate current I/O operation at selected channel and I/O unit.

- SI format:


9E
B1
D1
0
7 8
15 16
19 20
31
Select channel and I/O unit per operand address.
Terminate operation in channel and I/O unit.

- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is not applicable.
  Operand address (address of channel and I/O unit) is in D.
- Channel to be halted is determined by D(13–15).
- I/O unit to be halted is determined by D(16–23).
- Channel sends status to CPU via CC and by storing status byte into CSW.
- CC setting:
  Interruption in channel: CC = 0.
  CSW stored: CC = 1.
  Halted: CC = 2.
  Unavailable: CC = 3.

The Halt I/O, HIO, instruction terminates the current I/O operation at the selected channel. The status of the channel is sent, and the status of the I/O unit may be sent, to the CPU by setting the CC and by storing the CSW status byte. The operand address (base plus displacement) determined during I-Fetch of the HIO instruction is used to address the channel and I/O unit.

Diagram 5-703, FEMDM, is a flowchart of the HIO instruction. At the start of execution, the first 16 bits of the instruction are in E and the operand address is in D; the first operand is not applicable. If the CPU is in the Supervisor state, execution of the HIO instruction begins by setting the 'timing gate' trigger. This trigger sends a 'select' signal to the proper channel as determined by D(13–15). At this time, the unit address, D(16–23), is also gated to the channel.

When the specified channel receives the 'channel select' signal, the channel is tested for a pending interruption. If an interruption is pending, a CC of 0 is sent to the CPU together with a 'release' signal. (The interruption is not cleared and remains pending.)

If an interruption is not pending, the channel is tested to determine if it is free (not performing an operation). If the channel is not free ('command out' signal), the I/O unit is disconnected from the channel with an incorrect-length indication, the 'interrupt request' trigger is set, and a CC of 2 is sent to the CPU together with a 'release' signal. (The requested interruption remains pending in the channel.)

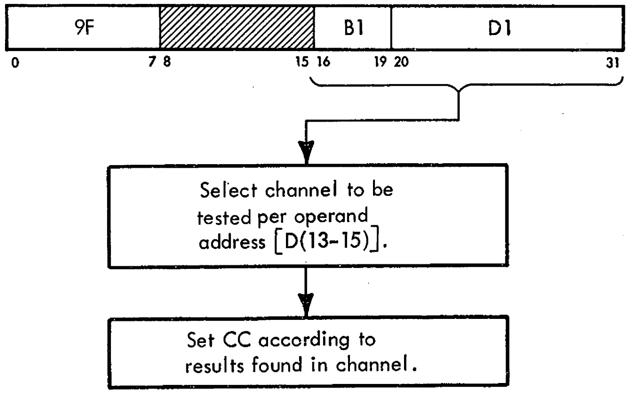
If the channel is free, an attempt is made to select the specified I/O unit. If an I/O unit is not selected (control unit or I/O unit is unavailable), a CC of 3 is sent to the CPU together with a 'release' signal.

If an I/O unit is selected, an address compare test is performed to determine if the correct I/O unit was selected. If the specified I/O unit was selected, it is disconnected from the channel; if some other I/O unit was selected, the 'interface control check' latch is set and an interface reset operation occurs. In either case, an interruption is initiated to store the status bits into the CSW, and a CC of 1 is sent to the CPU together with a 'release' signal.

When a 'release' signal is received by the CPU, the 'timing gate' trigger is reset, dropping the 'channel select' signal. An end-op cycle is taken, completing the operation, and the CPU is free to resume processing.

TEST CHANNEL, TCH (9F)

- Test state of selected channel and set CC accordingly.
- SI format:


9F
B1
D1
0
7 8
15 16
19 20
31
Select channel to be tested per operand address [D(13-15)].
Set CC according to results found in channel.

- Conditions at start of execution:
  First 16 bits of instruction are in E.
  1st operand is not applicable.
  Operand address (address of channel and I/O unit) is in D.
- Channel to be tested is determined by D(13–15).
- D(16–23), which normally selects I/O unit, is ignored.
- CC setting:
  Available: CC = 0.
  CSW ready: CC = 1.
  Working: CC = 2.
  Unavailable: CC = 3.

The Test Channel, TCH, instruction is used to obtain a CC describing the current state of the channel addressed by D(13–15). The channel state is not affected by the TCH instruction. The appropriate CC is generated and set into the CPU CC register, and the CPU is released to perform further instructions (Diagram 5-704, FEMDM).

If an interruption is pending, a CC of 1 is generated; if an interruption is not pending, the channel is tested to determine if it is working. If the channel is working, a CC of 2 is generated; if not working, a CC of 0 is generated. The generated CC is sent to the CPU, together with a 'release' signal. When the CC is set into the CPU CC register, the 'timing gate' trigger is reset and the CPU takes an end-op cycle, after which it is free to initiate the next instruction.

Chapter 4. Features

Section 1. Feature Index

The features available for the 2065 are listed in Table 4-1. Those features discussed in this manual are indicated by a page or chapter and section reference; only the major references are listed. Some of the features are discussed in separate Field Engineering manuals; the title and form number of these manuals are referenced.

Table 4-1. Feature Index

Name and Number	Reference
Additional Storage Attachment 1305 (5th Storage Unit) 1306 (6th Storage Unit) 1307 (7th Storage Unit) 1308 (8th Storage Unit)	Chapter 4, Section 2: Multiprocessing Features
Direct Control 3274	DISABLE DIRECT CONTROL switch: p6-17 Discussion: p1-2 G-register: p2-58 Operation: p3-180, 3-181, 4-4 Power: p5-12
Emergency Power-Off Control 3621 (2 Systems) 3622 (Up to 12 Systems)	Discussion: p5-8
Multisystem 4951 (CPU 1) 4952 (CPU 2)	Chapter 4, Section 2: Multiprocessing Features
1052 Adapter 7920 (1st Adapter) 7921 (2nd Adapter) 7922 (Dual Adapter)	Power: p5-12 1052 Adapter and 2150 Console FETOM: Form Y22-2808
2361 Attachment 8080	Discussion: p2-4, 2-31, 2-37
7070/7074 Compatibility 7117	2065 Processing Unit, 7070/7074 Compatibility Feature FETOM: Form Y27-2106 FEDM: Form Y27-2107
7080 Compatibility 7118	2065 Processing Unit, 7080 Compatibility Feature FETOM: Form Y27-2090 FEDM: Form Y27-2091
709/7040/7044/7090/7094/7094 II Compatibility 7119	2065/2067 Processing Unit, 709/7040/7044/7090/7094/7094 II Compatibility Feature FETOM: Form Y27-2098 FEDM: Form Y27-2099

MULTIPROCESSING SYSTEM/360 MODEL 65

The Multiprocessing System/360 Model 65 (Diagram 7-2, FEMDM) is a particular multiprocessor configuration. It contains two System/360 Model 65's with the Multisystem feature added to each CPU. The Multisystem feature can only be understood when described in the context of the Multiprocessing System/360 Model 65.

The Multiprocessing System/360 Model 65, also called here the multiprocessing system, can operate in one of three ways. Firstly, it can operate as one system, both CPU's controlled by the same supervisor program stored in the completely shared main storage units; because most I/O units are also shared by the CPU's, almost any I/O task can be assigned by the supervisor program to either CPU. Operating two CPU's as resources of one system is termed multiprocessor operation or simply multiprocessing. Secondly, it can be operated as two separate systems, each with its own supervisor program, main storage, CPU, and I/O units. Thirdly, it can be operated with only some of the storage units shared; this case is of limited interest and is not considered here.

The multiprocessing system has a configuration control panel (Diagram 7-11, FEMDM) mounted on the frame section connecting the two CPU's. The following discussion of each of the three major machine areas (main storage, I/O, and CPU):

1. Describes the concepts of multiprocessing.
2. Lists features added to the basic System/360 Model 65's to achieve the multiprocessing system.
3. Describes how the multiprocessing system is tailored by the configuration control panel switches.

Main Storage

In general, a multiprocessing system is characterized by multiple CPU's, each having access to all of main storage. In this fully shared main storage resides a supervisor program that regards the CPU's as a pool of processing resources. The supervisor program can then use the CPU's alternately or simultaneously on one task or separately on two tasks to achieve maximum thruput. Although the supervisor program is necessarily long and complex, it need be placed in only one area of main storage to operate the CPU's; thus, given a fixed number of main storage units and CPU's, more of main storage is available for use by problem programs.

In the Multiprocessing System/360 Model 65, main storage consists of at least two and up to eight 2365-13 Processor Storage units. Each storage unit added after the fourth requires an Additional Wall Section feature and, installed in each CPU, an Additional Storage Attachment feature. Each storage unit has a double BCU-storage interface. Priority circuits in the interface determine which BCU last had access to storage and, in the event of simultaneous requests, give access to the other BCU. This scheme ensures that neither CPU is allowed two successive storage references to a particular storage unit if the other CPU is waiting. Each storage unit has a CE panel allowing off-line maintenance.

Storage Allocation

Each of the main storage units can be allocated to both, either, or none of the CPU's; Storage Allocation switches on the configuration control panel determine which BCU-storage interfaces are enabled. Therefore, the multiprocessing system need not always be operated in a fully shared multiprocessing environment. For example, if multiprocessing is inappropriate for a certain workload, one group of storage units can be allocated to CPU 1 and the remaining storage units to CPU 2. The multiprocessing system can then be operated as two conventional System/360 Model 65's, each CPU run by a different supervisor program.

When allocated to neither CPU, a faulty storage unit can usually be repaired off-line using its CE panel.

Floating Addressing

Because any group of storage units can be allocated to the CPU's, a flexible means of arranging the main storage addresses must be provided. This arrangement is done by Floating Address rotary switches on the configuration control panel, one switch per storage unit. Any of the storage units in a group can contain any of the possible address intervals. This is what is meant by ***floating addressing***.

In round decimal numbers, 262 kilobytes of data can be stored in one 2365-13 Processor Storage, each byte individually addressable; it contains, then, an address interval of 262 kilobytes. The assignment of address intervals to the allocated storage units is therefore in multiples of 262, (i.e., 0 to 262; 262 to 524; and so on). Once the assignments for a group are made, they remain unchanged at least until the next IPL. Exact address ranges are listed in Table 4-2.

Whether the group of storage units is assigned to one CPU or shared by both, the assignment of floating addresses has one restriction: the first address interval of 0 to 262 *must* be assigned. For example, assume storage units 1, 3, and 4 have been allocated to CPU 1. The

Table 4-2. Floating Address Intervals

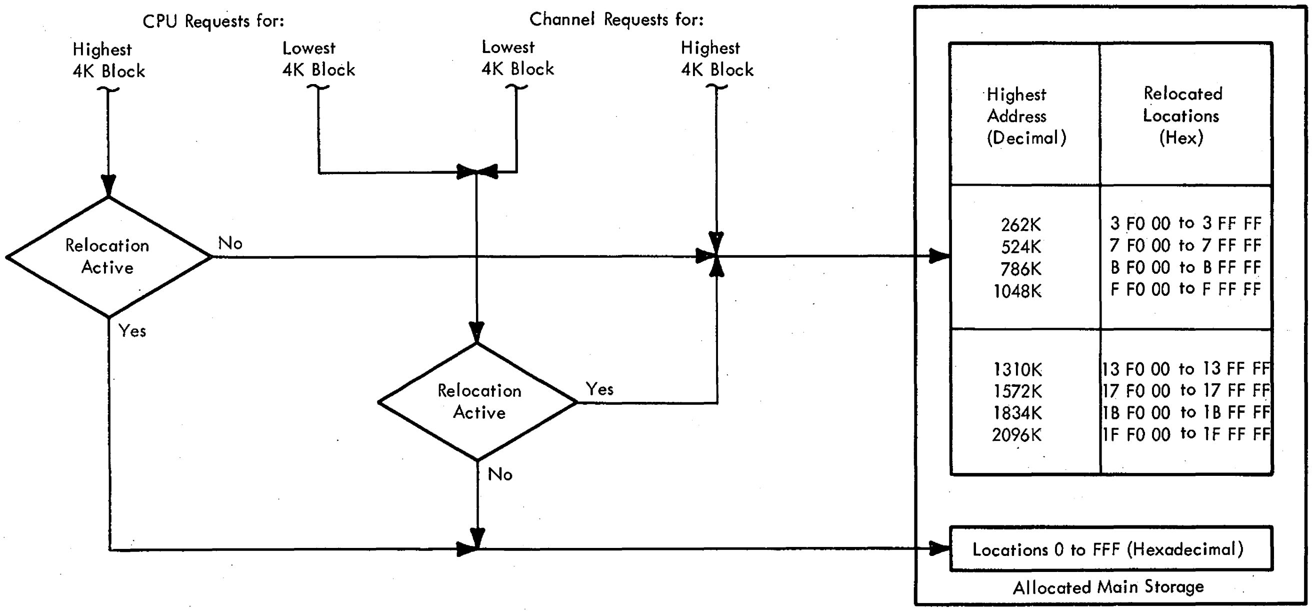
Rotary Switch Label	Decimal	Hex
0 TO 262K	0 to 262,143	0 to 3 FF FF
262K TO 524K	262,144 to 524,287	4 00 00 to 7 FF FF
524K TO 786K	524,288 to 786,431	8 00 00 to B FF FF
786K TO 1048K	786,432 to 1,048,575	C 00 00 to F FF FF
1048K TO 1310K	1,048,576 to 1,310,719	10 00 00 to 13 FF FF
1310K TO 1572K	1,310,720 to 1,572,863	14 00 00 to 17 FF FF
1572K TO 1834K	1,572,864 to 1,835,007	18 00 00 to 1B FF FF
1834K TO 2096K	1,835,008 to 2,097,151	1C 00 00 to 1F FF FF

highest possible address is 786K because 262K X 3 = 786K. Then the contiguous address intervals to be assigned are 0 to 262, 262 to 524, and 524 to 786. They can be assigned to the three storage units in any combination such as storage 3 = 0 to 262, storage 1 = 262 to 524, and storage 4 = 524 to 786. Thus in hex notation, storage 1 contains locations 4 00 00 – 7 FF FF, storage 3 contains locations 0 – 3 FF FF, and storage 4 contains locations 8 00 00 – B FF FF. Presumably, storage 2 has been allocated to CPU 2. In that case, storage 2 also has the address interval of 0 to 262 assigned to it.

Direct Address Relocation

Addresses 0–4095 (decimal) are generated without a base address or index and so are called the *direct addresses*. Direct address relocation causes CPU and attached-channel accesses to the lowest 4096 data bytes in allocated main storage to be made to the highest 4096 bytes; in addition, CPU accesses to the highest 4096 bytes are made to the lowest 4096 bytes. Only one CPU operating in the fully shared multiprocessing environment requires this ability. The assignment is made by a PREFIX switch on the configuration control panel, one switch per CPU. Refer to Figure 4-1 for the relocated addresses. The highest area is relocated to by inserting a 12-bit prefix into each address which has the high order 12 bits set to zero, SAB(0–11) = 0, and hence pertains to locations 0–4095.

The need for direct address relocation by one of the two CPU's sharing all of main storage is understood when the permanent storage assignment addresses are considered. Main storage bytes 0–383 (decimal) constitute the *permanent storage area* of System/360. It is these locations that are referenced by the programs, CPU, and channels for such data as the old and new PSW's, the interval timer value, the CAW, the CSW, the CPU logout, and the channel logout. It is impossible for multiple CPU's to operate normally sharing the same physical locations in main storage for this purpose. For example, old PSW's stored by CPU 1 would be destroyed by old PSW's stored by CPU 2; CPU 1, therefore, would re-enter the interrupted programs of CPU 2. Clearly, each CPU requires a separate permanent storage area. When operating separately with unshared storage allocated to each CPU, no conflict results. When operating together in the fully shared multiprocessing environment, however, the conflict would result if not for direct-address-relocation logic



Highest Address (Decimal)	Relocated Locations (Hex)
262K 524K 786K 1048K	3 F0 00 to 3 FF FF 7 F0 00 to 7 FF FF B F0 00 to B FF FF F F0 00 to F FF FF
1310K 1572K 1834K 2096K	13 F0 00 to 13 FF FF 17 F0 00 to 17 FF FF 1B F0 00 to 1B FF FF 1F F0 00 to 1F FF FF

Figure 4-1. Direct Address Relocation

added to the BCU by the Multisystem feature. Just as the CPU uses main storage addresses up to 383 (decimal) for its operations, the supervisor program uses up to address 4095, generally, for its operations. The tables, pointers, and control blocks associated with only one CPU are stored in that area or conflicts result. Therefore, the first 4096 bytes (0 to FFF, hex) are relocated.

Notice in Figure 4-1 that both permanent storage areas can be made accessible to both CPU's. This function is important for the supervisor program and error recovery routines. Notice also that the channels attached to the *relocating* CPU do not have access to the permanent storage area of the other CPU. This hardware-imposed restriction is inconsequential because the channels never need access to the other CPU's permanent storage area.

Input/Output

Not every I/O device in a multiprocessing system is connected to both CPU's. The 1052 Printer-Keyboard is an example. However, most of the devices are connected to both CPU's. The path of connection is from separate channels, through a double channel-control unit interface, to the control unit of the device. The double channel-control unit interface might be a separate standalone unit or a special feature (Two-Channel Switch feature) of the control unit, depending on the device. In either case, the Remote Switch Attachment feature is added to this two-channel switch interface placing I/O Allocation switches on the configuration control panel, one switch associated with each interface. When operating as a multiprocessing system, all the I/O Allocation switches are enabled. The supervisor program can then issue I/O commands from either CPU to most of the I/O devices. If one CPU is busy, the other can perform the I/O instruction. The CPU that starts an I/O operation is generally the CPU that is interrupted by the I/O when the operation finishes. The two-channel switch interface in the selected path directs the interruption request to the correct CPU.

When operating the two CPU's independently, the I/O devices are usually allocated to only one of the two CPU's. Remember that with or without the Multisystem feature, a System/360 Model 65 can operate as part of a multisystem by a shared-control-unit connection. Other I/O multisystem connections can exist through Channel-to-Channel Adapter feature, shared-device, or communication-line connections. The channels, control units, and devices (and their features) attachable to a basic System/360 Model 65 are attachable to a Multiprocessing System/360 Model 65.

Processing Units

In some multiprocessing systems, one of the CPU's is designed primarily, or even solely, for I/O processing, thus allowing economies in that CPU's capabilities. For instance, a floating-point instruction set is not needed in that CPU. However, when this assignment is done in a multiprocessing system of two CPU's, versatility is lost. In the Multiprocessing System/360 Model 65, the CPU's have equal capabilities. If both CPU's are busy processing, the next task – I/O, logical, or arithmetic – can be processed by the first CPU to become idle.

There are two three-position rotary switches on the configuration control panel, one per CPU. These rotary switches control the mode of system operation of the two CPU's. When multiprocessing together (duplexed), both rotary switches are placed in the MS position (Multisystem mode), enabling use of six special-purpose, inter-CPU signals, the multisystem signals. Also only in this mode, the Set System Mask instruction is modified. When operating independently (simplexed), the CPU is placed in either Model 65 (65 position) or Partition (PTN position) mode. Operation of the Direct Control feature (a prerequisite for installation of the Multisystem feature) is modified in Multisystem and Partition modes. The mode of operation does not affect storage allocation, floating addressing, direct address relocation or I/O allocation.

One Emergency Power-Off Control feature is required in the system.

Multisystem Mode

When multiprocessing in a fully shared environment, both CPU's are manually placed into Multisystem mode. To enhance multiprocessor operations, five multisystem signals are enabled between the CPU's in this mode. One of the signals, 'malfunction alert', is machine-generated. Another, 'gated load pb', is manually generated when LOAD is depressed. The remaining four signals are issued by execution of the Write Direct instruction as shown below:

84	I2	B1	D1
0 7	8 15	16 19	20 31

I2 Field		Multisystem Signal
Bits 8 and 9	= 01	'system reset'
	= 10	'external start'
	= 11	'log I/O interrupt'
Bit 10	= 1	(reserved)
Bit 11	= 1	'system call'

The Write Direct instruction is performed and the multisystem signals are issued regardless of the position of the DISABLE DIRECT CONTROL switch on the CPU system control panel. 'System reset', 'external start', and 'log I/O interrupt' signals are recognized by the receiving

CPU even if its direct control is disabled. Only the 'system call' and 'malfunction alert' signals can cause an external interruption of the receiving CPU.

Direct control might be interconnected between the CPU's or might be connected to external devices, depending on the system configuration. This can be done because the multisystem signals are separate from the direct control interface. However, direct control operations are modified when operating in Multisystem mode. Because the four high-order bits of the I2 field are assigned for special functions, their use in the Read Direct instruction, as well as external signals 0–3, are ignored. Use of bits 8–11 in the Write Direct instruction is restricted to the multiprocessing functions. Also, although direct control operations are still controlled by the DISABLE DIRECT CONTROL switch, no operation program interruption occurs if the switch is in the active (depressed) position. All other direct control operations, including external signals 4–7, are unchanged.

The Set System Mask instruction is modified in Multisystem mode only. The current system mask bits can not be changed by execution of the Set System Mask instruction when operating in Multisystem mode. If operating in the Supervisor state and Multisystem mode, the Set System Mask instruction is suppressed and causes a multisystem program interruption. The interruption code of 12 (hex) stored in the old PSW is used by the supervisor program to identify this source of interruption.

Model 65 Mode

Usually, both CPU's are switched to Model 65 mode when independent operation is desired. The Model 65 mode allows each 2065 to function as if the Multisystem features were not installed. Storage allocation, floating addressing, direct address relocation, and I/O allocation are active regardless of the operating mode. However, the multisystem signals and the multisystem program interruption are not needed when a non-multiprocessing supervisor program is operating one CPU. Therefore, the Direct Control feature and the Set System Mask instruction are as described for a System/360 Model 65 without the Multisystem feature.

The supervisor programs applicable to the conventional System/360 Model 65 are run in Model 65 mode. The emulator program for a Compatibility feature, for example, is run in Model 65 mode. (Any of the three Compatibility features might be installed in the two 2065's, depending on the system requirements.) Before initially loading the emulator program, a simplex system must be manually selected from the multiprocessing system. This selection is done by switching the CPU with the desired Compatibility feature to Model 65 mode and allocating to it the required storage and I/O units; the Floating Address switches are dialed so that storage addressing is continuous from location 0. Direct address relocation is not enabled because the IBM-supplied emulator program cannot be shared by the other CPU.

Partition Mode

Partition mode allows a multiprocessing supervisor program to run with one CPU if the other CPU in the multiprocessing system is unavailable. The other CPU may be unavailable to the multiprocessing program because of a changing workload or because of a machine malfunction. The configuration control panel is used to partition a simplex system from the multiprocessing system to perform other work or system maintenance.

Multiprocessing is simplified in Partition mode. Because the multiprocessing program handles I/O interruptions from only one CPU when running in Partition mode, the Set System Mask instruction operates as described for the 2065 without the Multisystem feature. Direct control operations in Partition mode, however, remain modified as in Multisystem mode; the multiprocessing program can then perform direct control operations the same way in both modes. Because the CPU is operating independently, the multisystem signals are neither issued nor recognized by a CPU in Partition mode.

Multisystem Signals

The six multisystem signals are issued and recognized only in Multisystem mode. Therefore, they are meaningful only if both CPU's are in Multisystem mode:

1. Malfunction Alert. The 'malfunction alert' signal is issued when CPU power is turned off or when a CPU machine check requires a logout operation. (As in a 2065 without the Multisystem feature, a CPU machine check requires a logout followed by a machine check interruption if the CPU CHECK switch is in the PROC position and the machine check mask bit is set to 1.) The receiving CPU is externally interrupted if the external mask bit is set to 1; bit 26 is set to 1 in the external-interruption old PSW. Thus, even if the malfunction of the sending CPU prevents it from performing a reliable logout and machine check interruption, the supervisor program can be notified via the external interruption of the receiving CPU that a malfunction has occurred. An additional CPU machine check condition can be detected in Multisystem mode to enhance the programmed recovery from "hang"-type malfunctions; a system hang machine check is caused if the enabled interval timer is not updated within eight cycles of the line frequency (116.7 to 133.3 ms in 60-Hz installations, 140 to 160 ms in 50-Hz installations). A multisystem timer, added by the Multisystem feature, counts the line-frequency cycles. The 'malfunction alert' signal remains pending in the receiving CPU if the external mask bit is set to 0.

2. Gated Load Pushbutton. The 'gated load pb' signal is issued when LOAD is depressed. All the units available to the receiving CPU, as well as the receiving CPU, are immediately reset: storage units, channels, control units, and devices. After being reset, the receiving CPU is placed in manual mode.
3. System Call. The 'system call' signal is issued by the Write Direct instruction if bit 11 of the I2 field is set to 1. Like the 'malfunction alert' signal, the 'system call' signal causes an external interruption of the receiving CPU if its external mask is set to 1. Bit 27 is set in the external-interruption old PSW. Unless direct control is interconnected between the CPU's, this is the only programmable means of externally interrupting the receiving CPU. The 'system call' signal remains pending in the receiving CPU if the external mask bit is set to 0.

The remaining three signals are mutually exclusive for any one Write Direct instruction because bits 8 and 9 of the I2 field can be coded for only one at a time. The signals do not externally interrupt the receiving CPU. Rather, they cause specific operations used by the multiprocessing program for recovery from program and machine malfunctions. These operations cannot be disabled by PSW mask bits or by the DISABLE DIRECT CONTROL switch.

4. Log I/O Interrupt. The 'log I/O interrupt' signal is issued by the Write Direct instruction if bits 8 and 9 of the I2 field are set to 11. If an I/O interruption is pending in the receiving CPU, the signal causes the CSW and interruption code of the waiting channel to be stored, regardless of its system mask bit. The multiprocessing program can thus recover I/O interruptions left pending at a malfunctioning CPU. Because the status of the I/O operation is not lost, the I/O operation does not have to be repeated.

   The microprogram in process when the 'log I/O interrupt' signal is received is immediately discontinued and the receiving CPU is forced to the manual mode. The operation of the system remains consistent if the CPU is in the Wait state and has all channels masked off when the 'log I/O interrupt' signal is received. If multiple I/O interruptions are pending, only the channel with highest priority is serviced. CPU machine checks are caused and ignored by the log I/O interrupt operation. After log I/O interrupt operations, the receiving CPU must be reset and restarted to continue normal processing.
5. System Reset. The 'system reset' signal is issued by the Write Direct instruction if bits 8 and 9 of the I2 field are set to 01. All the units available to the receiving CPU, as well as the receiving CPU, are reset: storage units, channels, control units, and devices. After being reset, the receiving CPU is placed in manual mode. Unlike the actions started by the 'gated load pb' signal, the system reset operation is delayed for two counts of the line frequency, allowing any storage of data in progress to finish. The multisystem timer is used for this counting function also, regardless of the position of the DISABLE INTERVAL TIMER switch.
6. External Start. The 'external start' signal is issued by the Write Direct instruction if bits 8 and 9 of the I2 field are set to 10. The receiving CPU is started (or restarted) by the 'external start' signal as if the PSW RESTART pushbutton were depressed. That is, the reception of the 'external start' signal causes processing to resume per the PSW in main storage address 0. The external-start operation begins immediately if the receiving CPU is in the Wait or Stopped state; otherwise it begins at the next end op. (If an external system-reset operation is in progress, the external-start operation begins when the reset is finished.) No system reset is performed, no I/O operation is initiated, and no old PSW is stored during the external-start operation.

Summary of Multiprocessing System Advantages

The key advantages of the multiprocessing system may be summarized as follows:

Increased Utilization: In installations having two independent CPU's, certain jobs possibly cannot be processed on one system because sufficient main storage or I/O devices are lacking. The other system, at the same time, may have main storage or I/O devices not being used. Resource sharing by the multiprocessing system minimizes these shortcomings.

Increased Availability: Resource sharing leads to another advantage. The multiprocessing system can continue to process in the event of most single machine malfunctions, even CPU malfunctions. This increases system availability. Concurrent repair of the faulty unit is possible in most cases.

Simplified Management: In an installation with two independent systems, manual balancing of the daily workload between two job queues is needed. Also, the installation data base might be split between the two systems, requiring duplication of records. The multiprocessing system alleviates these two problems when operated as a single system. The need for manually balancing the workload is minimized, and efficient operation is possible for applications needing large online data bases and higher file access rates.

Flexibility: The multiprocessing system can also be reconfigured into two independent systems when required by the workload.

FUNCTIONAL UNITS

Two functional units, the configuration control panel and the multisystem timer, are added; one functional unit, the BCU, is modified by the Multisystem feature.

Configuration Control Panel

The configuration control panel (Diagram 7-11, FEMDM) is mounted on the frame section connecting the two CPU's. The manual switches on this panel, the 'system X available' latches in the storage units, and the 'enable prefix' and 'multisystem mode' triggers in the CPU control the operation of the multiprocessing system. These controls are shown simplified in Diagram 7-3, FEMDM, and are described in the following paragraphs.

Storage Allocation Control

There is one double-pole, two-position toggle switch for each CPU-storage interface. When in the enabled position, CPU dc ground through one pole and -3V dc through the other pole are connected to the floating address switch of the same storage unit; this gates the setting of the floating address switch to the CPU to which that storage unit is allocated. The enabled dc ground is also sent to the designated CPU-storage interface in the storage unit, setting the 'system X available' latch when a 'not clock out' level is received from the same CPU. A 'power on storage frame X' level is then returned to the CPU if the storage unit is ready; that is, if it has power on and is not in test mode. The 'system X available' latch is reset when the CPU or storage unit power is turned off or when the 'not clock out' level from the CPU detects the absence of the enabled dc ground. This interlocking prevents selecting an unavailable storage unit and manually disabling a busy CPU-storage interface.

Floating Address Control

- One switch per storage unit.
- Logical address of each storage unit is encoded into three binary bits.
- 'Floating prefix bit' signals are encoded from allocated Floating Address switch set in highest position.
- If set storage addresses allocated to one CPU are duplicated or are higher than 1048K (without Additional Storage Attachment feature installed), then storage address decoding is blocked and VALID ADDRESS indicator is not lit.

There is one seven-deck, eight-position rotary switch for each storage unit. Decks A, B, and C of each switch encode the position into a three-bit binary value. For example, position 1 = 000, position 2 = 001, position 3 = 010, and so on. Both CPU's receive the encoded value regardless of the storage unit's allocation.

Deck E is used by CPU 1 (deck D by CPU 2) for encoding a three-bit *floating prefix*. The 'floating prefix bit' signals define the storage unit containing the highest 4K block of storage. They correspond to SAB(3–5) and are important during direct address relocation operations. The highest address range set into the allocated Floating Address switches is encoded into a three-bit binary code.

Deck G is used by CPU 1 (deck F by CPU 2) to detect when more than one of the storage units allocated to CPU 1 are assigned the same interval of addresses. The CPU detecting that conflict is blocked from decoding the storage address until the Floating Address switches are correctly positioned. Storage address decoding is also disabled when the rotary switch for any allocated storage unit is assigned an address interval above 1048K (without an Additional Storage Attachment feature installed). If the floating addresses of the allocated storage units are not duplicated and within the correct storage bounds, the VALID ADDRESS indicator lights.

The scheme for detecting duplicate addressing is shown in Figure 4-2, using CPU 1, storage 1, and storage 2 as examples. When the input to the receiver is greater than 1.9V dc, as when no duplicate addresses are assigned, a positive logic level is recognized and inverted to a negative logic level. However, both storage units are shown assigned the address interval of 000 to 262K. Because both storage units are allocated to CPU 1, two 270-ohm resistors are in parallel, lowering the resistance in series with the 200-ohm resistor. This action, in turn, lowers the receiver's input voltage to near ground potential (logical 0 level) instead of +2.4V dc (logical 1 level). Because the input to the receiver is now less than 1.0V dc, a negative logic level is recognized and inverted to a positive voltage level. A positive output voltage from the receiver, therefore, is indicative of a duplicate address assignment. Any further duplication of addresses lowers the input voltage even more; the receiver's output, therefore, remains at the positive logic level.

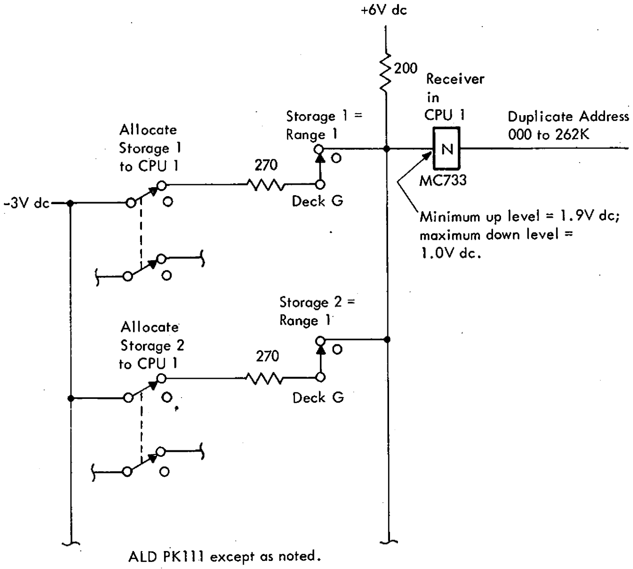

+6V dc
200
Receiver in CPU 1
Duplicate Address 000 to 262K
N
MC733
Minimum up level = 1.9V dc; maximum down level = 1.0V dc.
Storage 1 = Range 1
Allocate Storage 1 to CPU 1
270
Deck G
-3V dc
Storage 2 = Range 1
Allocate Storage 2 to CPU 1
270
Deck G
ALD PK111 except as noted.

Figure 4-2. Duplicate Storage Addressing Detection

Direct Address Relocation Control

The relocation of direct addresses (prefixing) may be enabled manually for either or both CPU's, or may be disabled for both CPU's. There is a two-position toggle switch, labelled PREFIX, for each CPU. The position of the switch is sampled when initially loading programs into the system; that is, when the LOAD pushbutton is depressed or when an external restart operation is performed. The switch position is also sampled when the CPU's power is turned on. If the PREFIX switch is in the ENABLE position, the 'enable prefix' trigger for that CPU is set. Otherwise, the CPU dc ground is returned to the CPU through the PREFIX switch and loading resets the 'enable prefix' trigger. Only when the 'enable prefix' trigger is set will storage requests for direct addresses be relocated.

Multiprocessing System Mode Control

The operating modes of the CPU's are separately controlled; either CPU can be placed in Multisystem, Partition, or Model 65 mode by its own Mode switch. Both of these switches are three-position, normally closed rotary switches, with the common terminal connected to the dc ground of the associate CPU. The Multisystem mode is entered and left only when the CPU's usage meter is stopped, such as when in the Stopped state. For this reason, the 'CPU clock stopped' signal samples the level returned from the MS position of the rotary switch. If the level is floating, a logical 1 is sensed and the 'multisystem mode' trigger is set; if CPU dc ground is returned, a logical 0 is sensed and the 'multisystem mode' trigger is reset. Switching between Partition and Model 65 modes is not synchronized with the operation of the CPU.

I/O Allocation Control

There are two switches on the configuration control panel for each Two-Channel Switch feature. Unlike all the other switches, they do not interface with the CPU logic. Refer to the FETOM of the applicable control unit for the operational description of the Remote Switch Attachment feature.

BCU Modifications

The following discussion explains the modifications to the BCU to accommodate up to 8 shared main storage units.

Storage Address Decoding with Prefixing Disabled

In a 2065 without the Multisystem feature, SAB(4,5) is decoded to select one of up to four 2365 Processor Storage units. Because Additional Storage Attachment features might be installed, the Multiprocessing system must provide for selecting up to eight storage units. Therefore, storage address decoding is concerned with SAB(3–5). Because of floating addressing, SAB(3–5) is not directly decoded but rather compared to the three binary-encoded lines returned from the Floating Address switches of each storage unit. If an equal comparison is made and the 262K block of addresses is available to the CPU, then the 'select' signal is issued to that storage unit.

Diagram 7-4, FEMDM, shows the storage address decoding. The circuits shown in the lower part of the diagram relocate direct addresses; if the CPU is not enabled for prefixing, these circuits are inactive. When the CPU has priority to access main storage, the address of the desired storage location is gated from D, IC, or PAL to SAB via a 24-bit OR in the BCU. Bits 3, 4, and 5 are gated from the OR by the 'not channel priority' signal to floating address comparison circuits, one for each storage unit. The desired storage unit is decoded if the storage unit with the equal comparison is available to the CPU and if the Floating Address switches are properly set. As in the 2065 without the Multisystem feature, bits 6 and 20 of the storage address are decoded in parallel to the decoding of bits 3–5; this decoding causes an even or odd interleaved storage selection to be made by the 'select timing pulse' signal. Unlike storage requests made by the CPU, the storage address decoding is done directly from the SAB when the storage request is made by an attached channel. Otherwise, the same decoding circuits are used for both CPU- and channel-initiated storage requests.

Storage Address Decoding with Prefixing Enabled

Three storage address detectors, in parallel with the decoding of the storage address and activated by the 'enable prefix' trigger, determine when prefixing is needed (Diagram 7-4, FEMDM). One detector determines if the CPU is attempting to access a location within its first 4K block of main storage addresses (prefix compare CPU). Another detector determines if an attached channel is attempting to access that first 4K block (prefix compare channel). The third detector determines if the CPU is attempting to access its last 4K block (reverse prefix compare). If such is not the case, prefixing is not needed and the 'select timing pulse' signal is issued to the decoded storage unit as a selection signal, just as if prefixing were not enabled. If, however, one of the three detectors determines that prefixing is needed, the 'select timing pulse' signal is blocked from being sent to main storage as a selection signal. Instead, the 'select timing pulse' signal sets the 'normal prefix' latch if relocation to its highest 4K block is needed (prefix compare CPU or channel) or sets the 'reverse prefix' latch if relocation to its lowest 4K block is needed (reverse prefix compare).

With either the 'normal prefix' or 'reverse prefix' latch set, the SAB is changed and a different storage unit is decoded to reflect this change. The 'normal prefix' latch changes SAB(3–5) to the 'floating prefix bit' signals and SAB(6–11) to all 1's. Thus, instead of the storage unit with the lowest addresses being decoded, the storage unit with the highest addresses is now decoded. The 'reverse prefix' latch changes SAB(0–11) to all 0's by degating the IC and D inputs to the SAB OR in the BCU. Because the BCU cannot degate bits put on the SAB by the channel, reverse prefixing is limited to CPU storage requests for the highest 4K block. The 'normal prefix' or 'reverse prefix' latch sets the 'any prefix' trigger, which deactivates the three storage address detectors; this action allows the second 'select timing pulse' signal, generated because a selection has been unsuccessfully tried, to be issued as an even or odd selection signal to the correct storage unit.

Invalid Storage Address

Addressing is invalid if:

1. More than one allocated storage unit is assigned the same addresses, whether or not requested.
2. Any storage unit, whether or not requested, is manually assigned an address interval greater than 1048K and an additional Storage Attachment feature is not installed.
3. The storage unit containing the requested address is not allocated to the CPU, is in test mode, or does not have dc power on.
4. No storage unit is assigned the requested address.
5. The requested address is higher than 2,097,151, decimal; that is, SAB(0, 1, or 2) = 1.

Specific circuits in the BCU are needed to detect all invalid addresses except for the fourth case listed: no storage unit assigned the requested address. In this case, a successful comparison of the three 'combined SAB' lines to the Floating Address switches does not occur. Therefore, none of the 'decode frame X select' lines are activated at the time of the first 'select timing pulse' signal. No plug card is needed to specify the highest valid address as in the 2065 without the Multisystem feature.

Just as in the 2065 without the Multisystem feature, a second 'select timing pulse' signal is generated when a selection of an invalid address is tried but a 'select' signal is not sent. The second signal is issued to the first available storage unit as a selection signal together with a 'cancel' signal. The first available storage unit is identified by the lowest-numbered physical frame returning a 'power on frame' level to the CPU. Although prefixing also blocks the first 'select timing pulse' signal and causes a second one to be generated, the requested address is not considered invalid and a 'cancel' signal is not sent.

BCU-Storage Operations

The BCU-storage operations differ from the 2065 without the Multisystem feature in two major respects: (1) the storage selection signal is issued even if the decoded storage unit is busy, and (2) an extra decoding cycle is added by the prefixing operation.

Because the selection signal is issued even if the decoded storage unit is busy, the BCU-cleanup operation is delayed until the 'accept' signal is returned from the storage unit. A non-busy storage unit immediately returns an 'accept' signal to the BCU. Even though not busy, the response time from different storage units varies because of different cable lengths to the CPU. A busy storage unit records the 'select' signal until all higher-priority requests are serviced. For 'select' signals arriving at a busy storage unit, the time necessary to establish priority is overlapped with the storage cycle in progress.

A CPU-initiated storage request is shown in Diagram 7-5, FEMDM. The 'accept' signal is received 300 ns after the earliest expected time. Because the 'accept' signal is not detected at the start of the cycle following the issuance of the selection signal, an 'inhibit oscillator' signal is generated (line 25 in the diagram). The BCU and CPU clocks are inhibited at not-clock time by turning off the gated oscillator pulses. The gated oscillator is similar to that used in the Model J65; see Chapter 2, Section 1. The 'accept' signal forms the not-clock-time pulses needed to complete the stopped cycle and turns on the gated oscillator that forms the succeeding timing pulses (lines 2 and 3). The 'BCU cleanup' signal (line 30) is generated by the setting of the 'HSS accept' latch (line 29). Therefore, not until then are the CPU priority circuits reset. However, to speed up the servicing of pending channel requests, the 'accept' signal is delayed 25 ns to form the 'timed accept pulse' signal (not shown). The 'timed accept pulse' signal scans the channel sync latch outputs, setting the channel priority trigger of the highest-priority request regardless of the BCU cleanup.

The extra cycle needed for the normal prefixing of a CPU storage request is shown in Diagram 7-6, FEMDM. The BCU timing of a channel request needing normal prefixing or a CPU request needing reverse prefixing is the same as that shown. Notice that the CPU clock is stopped for one cycle, even though in the example shown there is no delay receiving the 'accept' signal from the selected storage unit. This ensures that the data is gated at the correct time. When prefixing a channel request, the CPU clock is not stopped.

Multisystem Timer

The multisystem timer is a four-bit counter made up of polarity-hold circuits. It is added by the Multisystem feature to detect CPU inactivity and to delay the external system reset operation. Counting occurs only in Multisystem mode. The interval timer must be enabled for CPU inactivity to be detected and identified as a machine malfunction.

Operation

Diagram 7-7, FEMDM, shows the logic, timing, and counting scheme of the multisystem timer. 'Time clock step' pulses are counted if the CPU is in the Multisystem mode and either the interval timer is enabled or an external system reset operation is in progress. These 300-ns 'time clock step' pulses arrive at input-power line frequency (every 16 2/3 ms in 60-Hz systems; 20 ms in 50-Hz systems). A 100-ns time delay forms two 100-ns pulses; the first pulse, 'update overflow counter', is followed 300 ns later by the second pulse, 'transfer counter to latch'. Until the conditions for a counter reset are met, the counter triggers are set per the latches by the 'update overflow counter' pulse; following this, the triggers are set directly into the latches by the 'transfer counter to latch' pulse. The counter then waits for the next 'time clock step' pulse.

The counter is reset to zero by either an end op or an 'external system reset' signal. Both of these conditions degate the inputs to the triggers, update the triggers to zero, and then transfer the zero state of the triggers to the latches. The 'end op' trigger is set for 200 ns; therefore, in this case the second 100-ns pulse follows the first by 200-ns. The 'external system reset' signal is about 80 ns long, so in this case the two pulses are about 70 ns long and the second follows the first by 100 ns.

System Hang Timing

The timer, when in the Multisystem mode, counts the number of input-power frequency cycles occurring from the latest end-op cycle. While the CPU is operating normally, the counter may count one cycle, two at the most, before the next end op occurs. When the high-order bit of the counter, the 'system hang' trigger, is set, eight cycles are counted and it is assumed the CPU has ceased to operate, or is "hung", due to a machine malfunction. This assumption need not always be true. For example, a slow external device can cause the Read Direct instruction to be excessively long. Performing a manual storage ripple operation (as when clearing storage) with the interval timer enabled also sets the 'system hang' trigger; therefore, the interval timer should be manually disabled when clearing storage in Multisystem mode. The malfunction need not occur in the CPU either. For example, as previously discussed, the CPU and BCU clocks are inhibited until an 'accept' signal is received from the selected storage unit. If a machine malfunction in storage prevents the 'accept' signal from being issued, the CPU is "hung" and the 'system hang' trigger is set.

The eight 'time clock step' pulses are counted within 116.7 to 133.3 ms because the end op is independent from the counting and the period of a 60-Hz pulse is 16-2/3 ms. (140 to 160 ms are needed if the input-power frequency is 50 Hz.) The 'system hang' trigger sets the 'system hang error' latch and, if set, resets the gated oscillator control latches: 'select sent' and 'inhibit'. The 'system hang error' latch sets the 'error' trigger in the machine check interruption logic. The resetting of the oscillator control latches ensures that the BCU and CPU clocks are running; without these timing pulses the logout and machine check interruption microprograms cannot be performed.

External System Reset Timing

The timer delays the resetting of the system by the 'external system reset' signal for two counts of the 'time clock step' pulse. The 'external system reset' signal sets the 'external system reset' trigger and resets the counter. The 'external system reset' trigger then allows 'time clock step' pulses to be counted. The second 'time clock step' pulse updates the counter, setting the 'overflow 1' trigger. This action requires 16.7–33.3 ms for 60-Hz power (20–40 ms for 50-Hz power). At that time, then, the 'external reset delayed' trigger is set.

MULTISYSTEM OPERATIONS

The Multisystem feature modifies three instructions (Set System Mask, Write Direct, and Read Direct), and adds six operations (one per multisystem signal).

Set System Mask Instruction (Multisystem Mode)

The Set System Mask instruction operates as described in Chapter 3, Section 7, if the CPU is not in Multisystem mode. In Multisystem mode, the instruction is always suppressed and a program interruption occurs: in the Problem state, an interruption code of 2 (privileged operation) is stored into the old PSW; in the Supervisor state, an interruption code of 18, decimal (multisystem), is stored into the old PSW.

To set an interruption code of 18, the 'interrupt code 16' trigger is added by the Multisystem feature (Diagram 5-22, FEMDM). The 'SPEC' micro-order in the first ROS word sets 'interrupt code 2' and '16' triggers and forces an end op if the 'multisystem mode' trigger is set in the Supervisor state.

Write Direct Instruction (Not Model 65 Mode)

The Multisystem feature modifies the direct control circuits for the Write Direct instruction (Diagram 7-8, FEMDM) but does not change the write-direct microprogram. If the CPU is in Model 65 mode, the Write Direct instruction operates as described in Chapter 3, Section 7.

In the Partition mode, the instruction begins with a privileged operation test. If PSW(15) = 1, 'interrupt code 2' trigger is set and the instruction is suppressed by forcing an end op and is followed by a privileged operation program interruption. If PSW(15) = 0, the CPU is in the Supervisor state and execution continues. The 'partition mode' signal disables the invalid op code detector in case the DISABLE DIRECT CONTROL switch is in the active (down) position, preventing an operation program interruption. The 'partition mode' signal also prevents the issuing of E(8–11) to the external device when the 'timing gate' trigger is set. If the DISABLE DIRECT CONTROL switch is active, the 'direct control write out' signal and E(8–15) are blocked when the 'timing gate' trigger is set. The data byte is fetched and placed into the G-register as in Model 65 mode, and the instruction is completed with an end op regardless of the position of the DISABLE DIRECT CONTROL switch.

In Multisystem mode, the instruction, beginning with the same privileged operation test, continues if PSW(15) = 0. The 'multisystem mode' trigger disables the invalid op code detector, preventing an operation program interruption if the DISABLE DIRECT CONTROL switch is active. The 'multisystem mode' trigger also prevents the issuing of E(8–11) to the external device when the 'timing gate' trigger is set. If the DISABLE DIRECT CONTROL switch is active, the 'direct control write out' signal and E(8–15) are blocked when the 'timing gate' trigger is set. Regardless of the position of the DISABLE DIRECT CONTROL switch, the multisystem signals, E(8–11), are decoded and issued to the other CPU when the 'timing gate' trigger is set; 'system reset', 'external start', 'log I/O interrupt', and 'system call' signals can be issued by the Write Direct instruction only when executed in Multisystem mode. The data byte is fetched and placed into G as in Model 65 mode, and the instruction is completed with an end op.

Read Direct Instruction (Not Model 65 Mode)

The Multisystem feature modifies the direct control circuits but does not change the read-direct microprogram. The Read Direct instruction (Diagram 7-8, FEMDM) operates as described in Chapter 3, Section 7, if the CPU is in the Model 65 mode.

In Partition or Multisystem mode, the instruction begins by testing whether PSW(15) = 0. If it does not equal 0, 'interrupt code 2' trigger is set and the instruction is suppressed by forcing an end op and is followed by a privileged operation program interruption. If PSW(15) = 0, the CPU is in the Supervisor state and the execution continues. The 'partition mode' signal or the 'multisystem mode' trigger disables the invalid op code detector in case the DISABLE DIRECT CONTROL switch is in the active (down) position, preventing an operation program interruption. The 'partition mode' signal or 'multisystem mode' trigger also prevents the issuing of E(8–11) to the external device when the 'timing gate' trigger is set. If the DISABLE DIRECT CONTROL switch is active, the 'direct control read out' signal and E(8–15) are blocked when the 'timing gate' trigger is set. The data byte received from the external device is stored as in Model 65 mode, and the instruction is completed with an end op.

Malfunction Alert

The 'malfunction alert' signal, generated only in Multisystem mode, results from either a CPU machine check or a loss of CPU power. A CPU machine check is detected when the 'error' trigger is set and the CPU CHECK switch is in the PROC (process) position. A loss of CPU power is detected by the dropping of the 'enable direct control' relay, K50; K50 is one of the first relays to drop during a manual or automatic power-off operation. Either of these conditions triggers a singleshot, generating a 825-ns pulse (Diagram 7-9, FEMDM). This pulse is issued to the other CPU via a gated driver. All the direct control signals use a gated driver to prevent the issuing of spurious signals during normal power-up and power-down operations. However, the gating of the 'MS malfunction alert' signal is handled slightly differently to assure its issuance when the CPU power is removed, whether accidentally or purposely. A reed relay provides an electromechanical delay of the 'power off' signal. The reed relay is on a standard SLT card and is not a special circuit. The 5-second pick delay of the 'power on' relay, K47, prevents the 'MS malfunction alert' signal from being issued during the power-on sequence.

The 'MS malfunction alert' signal sets the 'external signal 2' latch in the receiving CPU if that CPU also is in Multisystem mode. If no higher-priority exceptional conditions or interruptions are pending at the next end op and the external mask bit is set, the receiving CPU is then externally interrupted.

Gated Load

Depressing the LOAD pushbutton causes a normal initial program load operation and, if the CPU Mode switch is in the MS position, the issuance of an 'MS gated load pb' signal (Diagram 7-9, FEMDM) to the other CPU. The driver for this multisystem signal is gated by the same electromechanically delayed 'power off' signal as the 'malfunction alert' signal.

The receiving CPU forms the 'MS gated load pb' signal into a 30-usec pulse if it also has its CPU Mode switch in the MS position. This pulse resets the receiving CPU and its attached units at the same time the sending CPU and its attached units are being reset. The receiving system is reset as if its SYSTEM RESET pushbutton were depressed, as shown in Diagram 8-7, FEMDM. At the end of the reset operation, the CPU is in the stop loop.

System Call

An 'MS system call' signal is issued when a Write Direct instruction with bit 11 specified is executed in Multisystem mode; it sets the 'external signal 3' latch in the receiving CPU if that CPU also is in Multisystem mode. If no higher-priority exceptional conditions or interruptions are pending at the next end op and the external mask bit is set, the receiving CPU is then externally interrupted.

Log I/O Interrupt

- Three latches control receiving CPU, not ROS.
- D is forced to address of I/O old PSW, and 'priority 1' and 'priority 2' triggers are set.
- 'Timing gate' trigger gates 'interrupt response' signal to highest-priority channel requesting I/O interruption.
- After channel stores CSW, CPU issues D-request to store interruption code.
- CPU is reset before and after log I/O interrupt operation.

A 'log I/O interrupt' signal is issued when a Write Direct instruction with bits 8 and 9 of the I2 field set to 11 is executed in Multisystem mode. If the receiving CPU is also in Multisystem mode, it is reset and then its 'manual' trigger and 'log I/O preparation' latch are set, initiating the log I/O interrupt operation (Diagram 7-10, FEMDM). Setting the 'manual' trigger lights the MANUAL indicator on the system control panel, disables the interval and multisystem timers, and stops the selected usage meter on the CPU; because the 'stop' trigger is not set, the stop loop is not entered. Setting of the 'log I/O preparation' latch prevents further ROS control of the CPU by holding the ROS sense latches reset. The receiving CPU is now controlled by the three sequence latches: 'log I/O preparation', 'log I/O execute', and 'log I/O terminate'. The operation of the channels, main storage, and the BCU is not affected. Direct address relocation remains active if the CPU's PREFIX switch is set to ENABLE.

Once the 'log I/O preparation' latch is set, up to five machine cycles might elapse before FLT-2 time when the 'log I/O execute' latch is set. With both of these latches set, the interrupt request trigger of the highest-priority channel requesting an I/O interruption is set, regardless of the channel mask bits in PSW(0–6). Next, at FLT-1 time plus 80 ns, a 'branch or normal end op' signal is generated that (1) forces D to 38 (hex), the location of the I/O old PSW, and (2) sets the 'priority 1' and 'priority 2' triggers, preventing any further change of the 'channel X interrupt request' triggers and setting the 'timing gate' trigger. If any 'channel X interrupt request' trigger is set, the setting of the 'priority 1' and 'priority 2' triggers is ensured by either the resetting or the inhibiting of all higher-priority exceptional conditions and interruptions.

The 'timing gate' trigger gates the 'interrupt response' signal to the channel determined by the 'channel X interrupt request' trigger set. The channel then proceeds to store a CSW into direct-address bytes 40–47. The channel's interruption operation is unchanged by the log I/O interrupt operation. When the channel finishes storing the CSW, it sets the 'release CPU' latch in the CPU. Waiting until FLT-2 time, the 'log I/O terminate' latch is set. At FLT-3 time, the 'log I/O store interrupt' signal is generated, causing:

1. The 'log I/O preparation' latch to be reset.
2. The I/O interruption code to be transferred to S(21–31).
3. The 'D sync' trigger to be set.
4. The 'mark 2' and 'mark 3' triggers to be set.

During the next machine cycle a 'log I/O terminate and 0 time' signal is generated, resetting the CPU. The CPU is now ready to receive another 'log I/O interrupt' signal from the other CPU. Meanwhile, the BCU, if not busy with a channel storage request, immediately services the D-request. D still contains the direct address of the I/O old PSW, 38 (hex). Because only 'mark 2' and '3' signals are issued, only the interruption code field is altered.

The log I/O interrupt operation ends with the 'log I/O terminate' latch set. The latch is reset while another log I/O interrupt operation is executed or if a system reset is performed. Until a system reset is performed, either the 'log I/O preparation' or 'log I/O terminate' latch is set, holding the ROS sense latches reset.

If there is no I/O interruption request pending when a 'log I/O interrupt' signal is received, the CPU waits with both the 'log I/O preparation' and 'log I/O execute' latches set. Once every four cycles the forced 'branch or normal end op' signal tests whether any 'channel X interrupt request' trigger is set.

External System Reset

A 'system reset' signal is issued when a Write Direct instruction with bits 8 and 9 set to 01 is executed in Multisystem mode. The 'system reset' signal sets the 'external system reset' trigger in the receiving CPU if that CPU's Mode switch is in the MS position. When set, this trigger performs five functions:

1. Inhibits further setting of the channel sync latches, preventing the honoring of channel storage requests.
2. Blocks the setting of the 'storage 2' trigger, preventing selection of main storage.
3. Blocks the issuance of any 'malfunction alert' signals.
4. Enables the multisystem timer.
5. Keeps external starts pending until the system is reset.

The multisystem timer delays the system reset operation until all processing has stopped. When the multisystem timer counts two line frequency cycles, the receiving system is reset as if the SYSTEM RESET pushbutton on its system control panel were depressed. At the end of the reset operation, the CPU is in the stop loop.

External Start

An 'external start' signal is issued when a Write Direct instruction with bits 8 and 9 set to 10 is executed in Multisystem mode; it sets the 'external start' trigger in the receiving CPU if that CPU also is in the Multisystem mode.

This trigger sets the 'external start' latch, waiting first for any end-op or external system reset operations in progress to finish. The 'external start' latch puts the CPU into the stop loop by setting the 'stop' trigger. When the stop loop tests whether the PSW RESTART pushbutton has been depressed (ROS micro-order J30), the 'external start' latch forces ROSAR(11) to set, causing a branch to the load PSW routine. Also at this time the 'enable prefix' trigger is set per the PREFIX switch on the configuration control panel.

POWER DISTRIBUTION AND CONTROL

The Multisystem and Additional Storage Attachment features do not change the power distribution of the CPU except that in CPU 2 the DC return bus is not connected to the frame bond (logic YA023).

With the installation of the Multisystem feature, relay K6 in the CPU is no longer held picked by the last active 'storage driver on' signal (logic YA021). Therefore, when the POWER OFF pushbutton is depressed on the system control panel, K6 immediately drops regardless of the status of the storage units. The dropping of K6 does not allow the system power-off sequence to wait until the core driver power of the last storage unit is off but immediately turns off CPU power, channel power, and (only if the POWER OFF pushbutton is being depressed on the last CPU with power on) main storage power. To protect the contents of main storage, each 2365 Model 13 contains gated line drivers similar to those used in the Direct Control feature in the CPU. The gated line drivers are the first circuits to be turned off during the storage power-off sequence and the last circuits to be turned on during the storage power-on sequence. Refer to the FETOM of the 2065 Model 13 storage for a complete discussion of its power controls.

Several of the power controls are modified by both the Multisystem feature and the first Additional Storage Attachment feature. The changes are discussed next under "Console Controls and Maintenance Features".

CONSOLE CONTROLS AND MAINTENANCE FEATURES

The following paragraphs describe the configuration control panel and the modifications made by the Multisystem feature to the system control panel and to the logout and scan-in operations.

Configuration Control Panel

The configuration control panel is added by the Multisystem feature and is shared by the two CPU's. For the following discussion of the configuration control panel, refer to Diagram 7-11, FEMDM.

Storage Allocation Switches

A Storage Allocation toggle switch for each CPU is associated with each storage unit. Only with the switch set to ENABLE is the storage unit available to that CPU. At least one storage unit is allocated to each operating CPU. The same storage unit may be allocated to both CPU's. A change in storage allocation becomes effective when the associated CPU enters the Stopped or Wait state.

Floating Address Switches

One Floating Address switch is associated with each storage unit. These rotary switches control the address range of each storage unit in 262 kilobyte intervals. The first four positions are: 0 TO 262K, 262K TO 524K, 524K TO 786K, and 786K TO 1048K. If the first Additional Storage Attachment feature is installed, these switches may also be placed in one of the next four positions: 1048K TO 1310K, 1310 TO 1572K, 1572K TO 1834K, and 1834K TO 2096K.

Contiguous addresses are usually assigned for each CPU, starting with address 0. Two storage units may not be assigned the same address interval if either is allocated to both CPU's. To provide maximum thruput, the direct-address main storage unit(s) should be as close to the CPU as possible; refer to Diagram 7-2, FEMDM. The Floating Address switches are always active.

PREFIX Switches

One PREFIX toggle switch is associated with each CPU. With the switch set to ENABLE, CPU references to the low-order 4096 (decimal) bytes and the high-order 4096 bytes of main storage are swapped, or relocated; also, channel references to the low-order 4096 bytes are relocated to the high-order 4096 bytes. Relocation does not occur when the PREFIX siwtch is set to DISABLE. Normally, neither of these two switches is set to ENABLE when no storage units are shared; only one is enabled when all storage units are shared. A change in position of a PREFIX switch becomes effective when the associated CPU (1) performs a power-on sequence, (2) has its LOAD pushbutton depressed, or (3) performs an external start operation.

CPU Mode Switches

One CPU Mode switch is associated with each CPU and determines the operating mode of that CPU. Multisystem mode (MS) is entered or exited when the CPU clock is stopped, according to the position of the switch. Switching between Partition (PTN) and Model 65 (65) modes is immediately effective.

I/O Allocation Switches

These switches enable or disable communications between the associated CPU and the control unit attached to two channels. Switches for up to 24 control units can be accommodated on the configuration control panel. These switches are labeled as a group, *I/O CONTROL*, and individually according to the control unit type. Refer to the applicable control unit FETOM for a further description of these switches.

VALID ADDRESS Indicators

One VALID ADDRESS indicator is associated with each CPU. When not lit, an incorrect manual assignment of an allocated storage unit has been made: either the same floating addresses have been assigned more than once or a Floating Address switch is in one of the last four positions without an Additional Storage Attachment feature being installed. The indicator lights after the needed correction is made. Not until then can the associated CPU be operated.

System Control Panel Modifications, Multisystem Feature

EMERGENCY PULL Switch

The pulling of an EMERGENCY PULL, EPO, switch on either CPU removes power beyond the entry point from all units in the multiprocessing system. The two EPO switches of the CPU's are directly interconnected by Emergency Power-Off Control feature No. 3621 or, if there are other EPO switches in the multiprocessing system, by Emergency Power-Off Control feature No. 3622.

POWER ON Pushbutton

The POWER ON pushbutton on the system control panel of each CPU turns on the power of that CPU and all the units, shared and nonshared, controlled by that CPU. Power is applied to all storage units. The POWER OFF pushbutton removes power from that CPU and all the nonshared units controlled by that CPU; however, if the power of the other CPU has previously been turned off, the power of the shared units and all storage units is also turned off. The power of units having their LOCAL/REMOTE switch in LOCAL is not controlled.

Marginal Voltage Control

The +6VM power supplies of the channels attached to the CPU can be varied from the system control panel. Only the storage units with the 6VMC switch (on each 2365's power control panel) in the proper position can be margined from the system control panel: SYSA for CPU 1 and SYSB for CPU 2.

DISABLE INTERVAL TIMER Switch

The activation of the DISABLE INTERVAL TIMER switch (N13) prevents decrementing of the interval timer and incrementing of the multisystem timer.

DISABLE DIRECT CONTROL Switch

The activation (depression) of the DISABLE DIRECT CONTROL toggle switch (N27) still disables the 'direct control timing signal out' bus and the external signals received from the external device. However, only in Model 65 mode does a Read Direct or Write Direct instruction cause a program interruption when the switch is active. In Multisystem and Partition modes, the instructions are executed but no direct control timing signals are issued to the external device. The DISABLE DIRECT CONTROL switch does not affect the issuing or receiving of the six multisystem signals.

Storage Switches

The storage ADDRESS switches must be set to the relocated storage address, when applicable, for scoping or stop-on-storage-address-compare functions. Store and Display manual operations are not affected.

When the CPU's share main storage, the DEFEAT INTERLEAVING switches (N9) on both system control panels must be in the same position (unless specified otherwise during a diagnostic program).

When a storage unit available to both CPU's detects an error while the requesting CPU's STOP ON STORAGE CHECK switch (N11) is active, the operation of that storage unit immediately stops. Therefore, if the requesting CPU has not already received the 'accept' signal, its operation is also stopped. Eventually, when both CPU's are waiting for the 'accept' signal from the stopped storage unit, the entire multiprocessing system enters the Stopped state. The CPU's do not automatically enter the stop loop.

Indicators

The indicators added by the Multisystem feature are shown in Diagram 7-12, FEMDM. The STORAGE FRAME X ENAB indicator shows the storage unit's status with respect to the CPU; regardless of that status, the STORAGE FRAME X 1048K, 524K, and 262K indicators show the binary-encoded position of the Floating Address switch and, hence, the storage unit's lowest address. The CPU identification indicators 1 and 2 are wired into the respective CPU's during system installation. The remaining indicators are self-explanatory.

System Control Panel Modifications, Additional Storage Attachment Features

The installation of the first Additional Storage Attachment feature adds a section to the configuration control panel (as shown in Diagram 7-11, FEMDM) and further modifies the system control panel as next described.

Marginal Voltage Control

Marginal voltage control and metering for storage units 1 through 4 can only be done from CPU1. For storage units 5 through 8, marginal checking can only be done from CPU2.

POWER CHECK Indicators

POWER CHECK indicators for storage units 1 through 4 are on CPU1; those for storage units 5 through 8 are on CPU2.

STORAGE INDICATE Switch and Indicators

A four-position rotary switch is added to panel D, replacing the two-position STORAGE INDICATE lever switch on panel F. The four positions are 1–4, 5–8, 9–12, and 13–16 for the basic storage modules (BSM's) in storage units 1 and 2, 3 and 4, 5 and 6, and 7 and 8, respectively. The display, shown in Diagram 7-123, FEMDM, is made regardless of storage allocations.

Logout and Scan In

The indicators added by the Multisystem feature are included in the logout data. Only in Multisystem and Partition modes are the 16 GPR's also included in the logout data (Diagram 8-115, FEMDM). Three feature triggers can be set by a scan-in operation: 'enable prefix', 'external start', and 'external system reset'.

Chapter 5. Power Distribution and Control

When the 2065 CPU was introduced, the 2060 CPU's already produced were converted to 2065's. Although the functions of the two resulting power systems are the same, it is occasionally necessary to differentiate between the converted units and those originally manufactured as 2065's. Converted and original units may be identified by comparing the CE panel with Diagram 8-29, FEMDM. Additional variations in the power system are described for 50-Hz and 60-Hz line frequency and the installed optional features.

AC POWER DISTRIBUTION

The primary ac power distribution is shown in Figure 5-1 for 60-Hz units and in Figure 5-2 for 50-Hz units.

60-Hz Units

Main power from circuit breaker 1 (CB1) is applied to transformer T1 via fuses F1 and F2 (Fig. 5-1). T1 provides 28V ac for the channel's and 2365's remote margin power, the alarm circuit, and a full-wave bridge rectifier. This bridge rectifier supplies 24V dc to the Emergency Power-Off (EPO) loop and, via PK1 contacts, to the 24V dc bus. (The 24V dc bus energizes the thermal protection circuits and the relay gate.)

Main power is applied by PK1 contacts to:

1. T2 via CB12. T2 provides 115V ac to convenience outlets in the CPU and the 2365's via CB11, and to the 1052 Printer-Keyboard via F4 and PK2 contacts.
2. T3 via F3 and CB2. T3 provides 40V ac to the usage meters, 12.6V ac to the interval timer logic and the 1052 adapter logic, and low-voltage ac to another full-wave bridge rectifier, which in turn provides ±20V dc to the undervoltage (and overvoltage in the converted units) protection circuits.
3. T4 via F3 and CB7. T4 provides 28V ac to the converter/inverter.
4. PK2 via CB9.

Main power is applied by PK2 contacts to:

1. Gate and power supply blowers via CB8 (and CB3 through CB6 in the converted units).
2. Converter/inverter.

50-Hz Units

Main power is taken across the three phases or across each phase and neutral, depending on the voltages at the location (Figure 5-2). Transformer taps and switches are adjusted accordingly.

Main power from CB1 is applied to T1 via F1 and the phase/neutral terminals (Switchboard 1, SW BD1).

T1 provides 28V ac for the channel's and 2365's remote margin power, the alarm circuit, and a full-wave bridge rectifier. This bridge rectifier supplies 24V dc to the EPO loop, and, via PK1 contacts, to the 24V dc bus.

Main power is applied by PK1 contacts to:

1. Convenience outlets in the CPU and 2365's via CB5 and the phase/neutral terminals (SW BD2), and to 1052 Printer-Keyboard via CB5, PK2 contacts, and the phase/neutral terminals (SW BD2).
2. T3 via F2, CB2, and the phase/neutral terminals (SW BD2). T3 provides 40V ac to the usage meters, 12.6V ac to the interval timer logic and the 1052 adapter logic, and low-voltage ac to another full-wave bridge rectifier, which in turn provides ±20V dc to the undervoltage protection circuits.
3. T4 via F2, CB3, and phase/neutral terminals (SW BD2). T4 provides 28V ac to the converter/inverter.
4. PK2.

Main power is applied by PK2 contacts to:

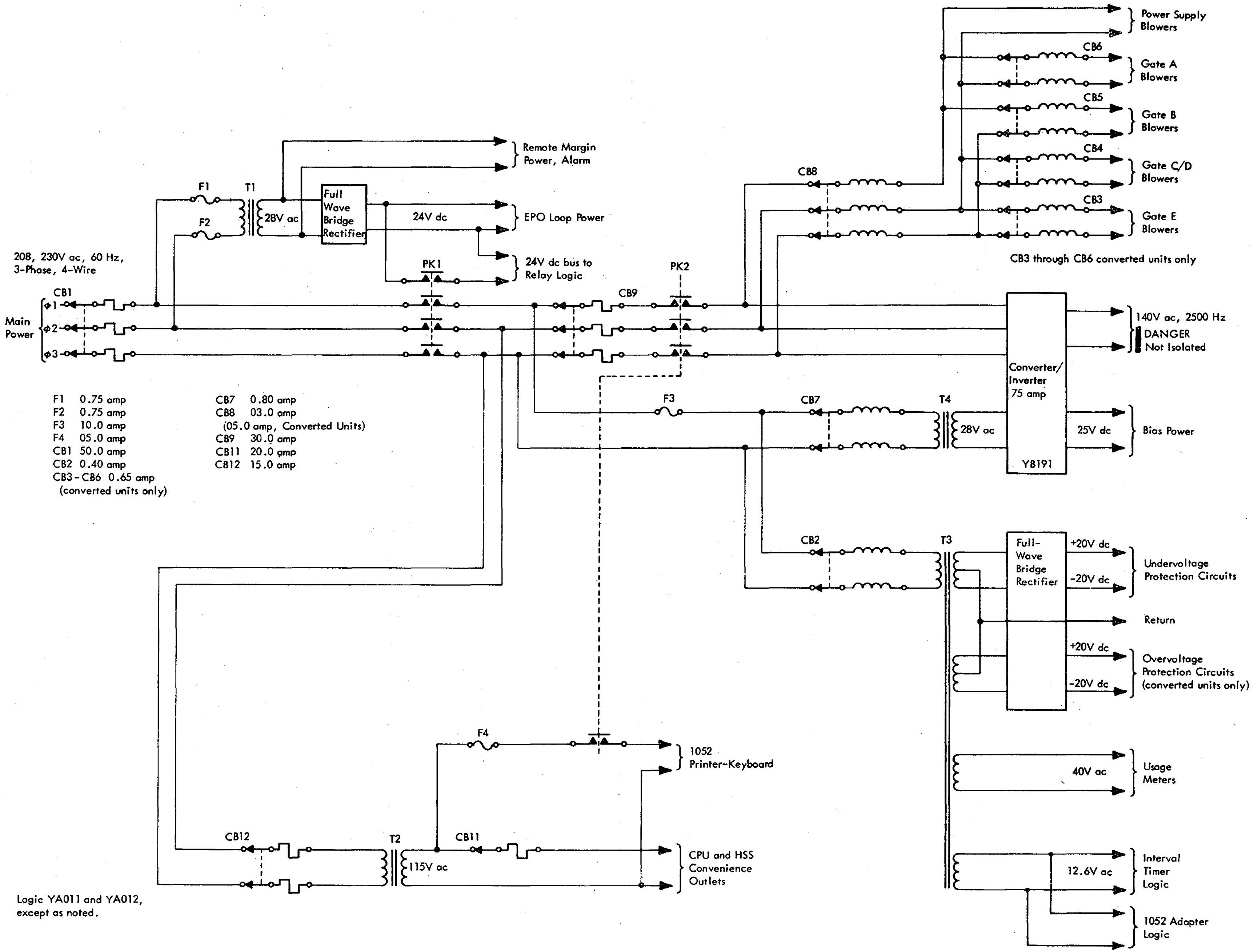
1. Gate and power supply blowers via CB4, T2, and phase/neutral terminals (SW BD 2).
2. Converter/inverter via a three-phase autotransformer.

Converter/Inverter

The converter/inverter converts the main 50-Hz or 60-Hz, 3-phase ac to dc and inverts the dc to 140V, 2500-Hz, 1-phase ac for the regulators. A detailed description of the operation is given in the *SLT Power Supply,* FEMI, Form 223-2799. Diagram 6-1, FEMDM, is a simplified schematic.

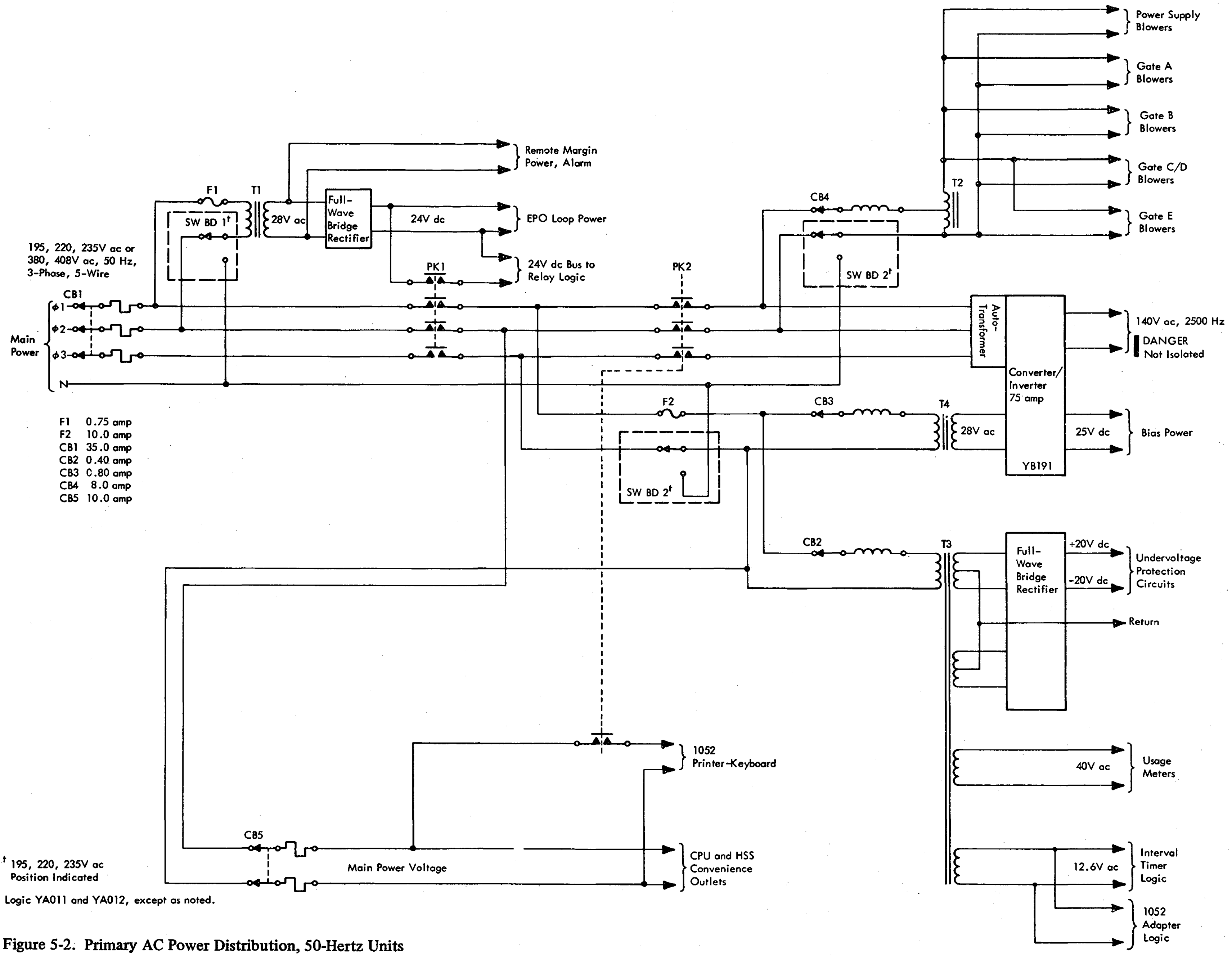
The 3-phase wall power is converted to dc by the 3-phase bridge rectifier. The output of the rectifier is filtered by capacitors C0, C1, and C2. In addition to filtering, C1 and C2 provide a split source for the inverter. Resistors R1 and R2 help balance this split source and provide bleeder loading to discharge the capacitors when power is turned off.

The inverter is basically two silicon-controlled rectifiers (SCR's) that alternately switch the load across Edc at a 2500-Hz rate. A 2500-Hz square wave is formed at load points A and B. Switching is performed by the two load SCR's, SCR1 and SCR2, and the two commutation SCR's, SCR3 and SCR4. Assume SCR1 and SCR3 are gated on. Load current I1 enters the load at point A. Capacitor C3 charges to Edc. When C3 reaches full Edc, SCR3 turns off. At this point, SCR4 is gated on. C3 discharges against



Logic YA011 and YA012, except as noted.

Figure 5-1. Primary AC Power Distribution, 60-Hertz Units



F1	0.75 amp
F2	10.0 amp
CB1	35.0 amp
CB2	0.40 amp
CB3	0.80 amp
CB4	8.0 amp
CB5	10.0 amp

† 195, 220, 235V ac Position Indicated

Logic YA011 and YA012, except as noted.

Figure 5-2. Primary AC Power Distribution, 50-Hertz Units

SCR1, turning it off. At this point, SCR2 is gated on. Load current I2 enters the load at point B; C3 charges to Edc. When C3 reaches full Edc, SCR4 turns off. At this point, SCR3 is gated on. C3 discharges against SCR2, turning it off. The cycle then repeats with SCR1 gated on.

DANGER
The output to the regulators is not isolated from the wall power, presenting a lethal potential to ground.

DC POWER DISTRIBUTION

The various dc voltages required by the CPU logic are provided by 15 regulators. A 16th regulator, the 48V power supply (PS9), is not installed in the CPU when the 2150 Console option is used. The 2150 Console has a multivoltage power supply which provides the 48V dc. Table 5-1 lists the output voltage levels and the major load for each regulator. Figure 5-3 is representative of their distribution.

High-Frequency Regulator Modules

- CPU regulators:

Volts, dc	Amps, dc	Type	Qty
+6	25	SCR Control	1
+6	40		4
+3	40		7
-3	40		2
-18	11	Magnetic amplifier control	1

Table 5-1. High-Frequency Regulator Modules

Power Wall Location	Nominal Output	Logic	Load	
			CPU Gate	Other
PS1	+6V 25 amp	YA061	C and D upper	—
PS2	+3V 40 amp	YA062	B lower	—
PS3	+3V 40 amp	YA061	E lower	—
PS4	-3V 40 amp	YA062	C and D	—
PS5	+3V 40 amp	YA061	B and E upper	—
PS6	+3V 40 amp	YA062	C and D	—
PS7*	+6V 40 amp	YA061	E	—
PS8*	+6V 40 amp	YA062	C and D lower	—
PS9†	+48V 2 amp	YA141	—	Keyboard, printer, and audible alarm of 1052
PS10	-3V 40 amp	YA072	A, B, and E	—
PS11*	-18V 11 amp	YA071	C and D upper	—
PS12	+3V 40 amp	YA072	A lower	Upper roller switch indicators
PS13	+3V 40 amp	YA071	B upper	—
PS14	+3V 40 amp	YA072	A upper	Lower roller switch indicators, usage meters
PS15*	+6V 40 amp	YA071	B	—
PS16*	+6V 40 amp	YA072	A	—

* Marginable regulator.
† PS9 is added by 1052 Adapter Feature if 2150 Console is not included in system.

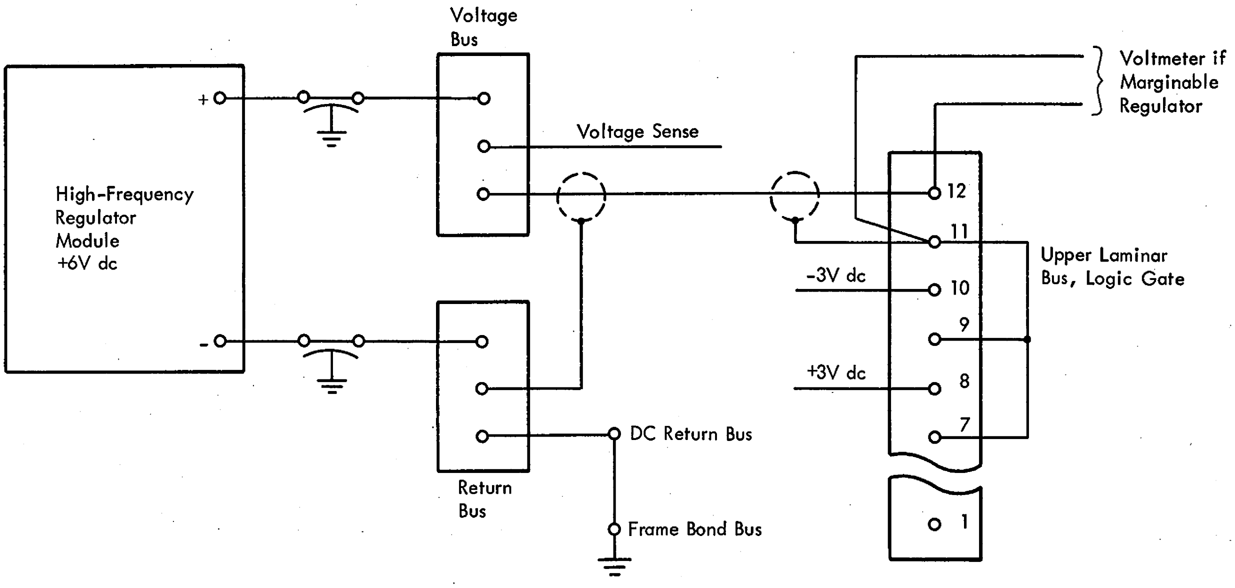

Voltage Bus
+
-
High-Frequency Regulator Module +6V dc
Voltage Sense
Voltmeter if Marginable Regulator
12
11
-3V dc
10
9
+3V dc
8
7
1
Upper Laminar Bus, Logic Gate
DC Return Bus
Return Bus
Frame Bond Bus

Figure 5-3. Representative DC Distribution

The high-frequency regulators rectify, control, and filter the 2500-Hz ac from the converter/inverter to the necessary dc voltage levels. There is transformer isolation from the wall power. A detailed description of the operation is given in the *SLT Power Supply,* FEMI, Form 223-2799.

DANGER
The input to the regulators from the converter/inverter is not isolated from the wall power, presenting a lethal potential to ground.

Two types of regulators are used in the CPU: the SCR-controlled output type and the magnetic-amplifier-controlled output type. The 18V dc ROS regulator (PS11) is of the first type, the others are of the second type.

The SCR-controlled output regulator utilizes the SCR's ability not to conduct until it is gated on to control the output voltage level. The duty cycle (time on versus time off in each half of the input ac cycle) of the SCR is determined by an adjustable voltage-sentitive feedback loop from the output terminals. The time that the SCR's are gated on determines the output voltage level (after averaging by the filters) at the immediate load current.

The magnetic-amplifier-controlled output regulator uses a bridge magnetic amplifier to control the output voltage level. The magnetic amplifier is a "square loop" toroidal core with three windings: gate, bias, and control (Diagram 6-2, FEMDM). The gate winding passes power to the output relative to the degree of saturation in the core. The core is set at a point on the slope of the saturation curve below saturation by the relative sum of the currents in the bias and control windings. The fixed current in the bias winding is counteracted by the variable out-of-phase current in the control winding. The amount of counteraction (and, therefore, the degree of saturation) determined by the two windings is adjusted according to the amount of error detected by the voltage feedback sensed from the output terminals. The remote sensing feature of these regulators is not used.

At the start of the gate cycle (the half of the input ac cycle determined by the diodes in the particular gate winding current path), the inductance of the gate winding is high (because the core is not saturated), causing a slow current rise. At some point in the gate cycle, the current in the gate winding rises (because the saturation level is rising with the additional current) to a point where the relative total of the currents in all three windings is sufficient to saturate the core. At saturation, the gate-winding current jumps to full current. When the current cycle passes through zero into the reset cycle, the saturation level returns to the point determined by the bias and control-winding currents.

The dc return levels of the regulators (logic or system ground) are made common with the wall power ground, the convenience outlet ground, the relay ground, and the frame at one point (frame bond bus). Voltage transients of high potential can exist in these other circuits at any time; usually these transients are caused by the switching of relays or contactors within the system. No ground circuit is completely without resistance. Therefore, if more than one low-resistance path were allowed between the logic ground circuits and the other ground circuits, transient currents could circulate in the "ground loop" thus formed, possibly high enough to induce erroneous switching of the semi-conductor components.

Marginal Adjustments

Several power supplies in the CPU and in the attached storage units and channels may have their output level varied from the nominal output. This feature allows critical circuits to be tested with nonstandard voltages as an aid in predicting failures.

When a power supply or attached unit is margined, an 'active margin' 24V dc level is generated. Each 24V dc level is fed to an OR to light the ACTIVE MARGIN indicator, showing the system has an active marginal adjustment. The MARGIN/METER SEL rotary switch may be turned to successive positions to locate the margined supply or unit. The LOCATE MARGIN indicator lights when the switch is at the position of a margined supply or unit. The voltage level is shown on the meter. (See logic YA081, YA082, and YA121 for the circuits.)

Within the CPU, four of the 6V power supplies (PS7, PS8, PS15, and PS16) and the 18V ROS supply (PS11) have their output levels adjusted by individual panel controls. These marginal controls have a cam and switch for the 24V dc 'active margin' level.

In the channels and 2365's, the margined supplies are adjusted by a motor drive controlled by the MARGIN CHANNEL/STOR lever switch and selected by the MARGIN/METER SEL switch. 28V ac through the raise and lower motor in each unit are returned through the rotary switch to select the unit and the lever switch to actuate the motor. The RAISE position causes the motor to adjust the marginal voltages to a higher level. The LOWER position adjusts the marginal voltages to a lower level. The 'active margin' 24V dc level is sent to the CPU by each unit with a margined supply.

POWER-ON SEQUENCE

- Close all CPU CB's.
- Move CPU READY/OFF switch to READY position.
- Close service CB.
- Depress CPU ON or POWER ON pushbutton after waiting 5 seconds.

DC power in the CPU may be brought up in two ways. It may be initiated at the CE panel if only CPU power-on is desired, or it may be initiated at the system control panel or at the 2150 Console if full system power is desired. Diagram 6-3, FEMDM, shows the primary power-on sequence; Diagram 6-4, FEMDM, shows the secondary power-on sequence; and Diagram 6-5, FEMDM, is a simplified diagram of the CPU power control circuits.

The primary power is sequenced on as follows:

1. Close all CB's: CB1, CB2, CB7, CB9 (60-Hz units only), CB11, and CB12 (also CB3–CB6 in converted units).
2. Move CPU READY/OFF switch on CE panel to READY.
3. Close service CB to apply main power to CB1. As a result, the following occurs:
   a. K4 and K28, the EPO relays, are picked by 24V dc through the EPO switch(es).
   b. PK1, the EPO contactor, is picked by 24V dc through K4 contacts.
   c. The 24V dc bus, source of all other 24V dc used in the CPU, is energized through PK1 contacts.
   d. If the stepping switch is not at the start position (position 26) when the 24V dc bus is energized, it is advanced to the start position by 24V dc through K32 and the start-interlock contacts. These contacts remain closed until the stepping switch reaches the start position. Refer to *SLT Power Supply,* FEMI, Form 223-2799, for a description of the stepping switch.
   e. Main power is applied to PK2, the CPU power contactor, through PK1 contacts and CB9. (CB9 is in 60-Hz units only.)
   f. K47 (K25 in converted units) is picked by 24V dc through the power-off pushbuttons, K12 contacts, CPU READY/OFF switch, and protection-relay contacts. K47 (K25) has a 5-second RC time delay before it transfers. Secondary power is usually sequenced on by depressing a POWER ON pushbutton (step 4) but is also sequenced on when the CPU ON pushbutton is depressed (step 5). Because of the delaying action of K47 (K25), at least 5 seconds must elapse before either pushbutton has any effect. (The 5-second delay ensures the delay of the RC turn-on circuits in the regulators. Otherwise, rapid alternations between automatic power-offs and attempted power-ons could cause overcurrents in the regulators.)
4. Depress POWER ON on system control panel or 2150 Console.
   a. The stepping switch drive coil is pulsed by 24V dc through stepping switch contacts A-26 to A-COM and the interrupter contacts; the stepping switch advances to position 1.
   b. K7 and K32 are picked through CR9; K6 through CR9 and CR8.
   c. K6, K7, and K32 are held by 24V dc through the power-off pushbuttons and K7 contacts.
   d. K12 is picked by 24V dc through K6 contacts.
   e. K5 is picked by 24V dc through the POWER ON pushbutton, stepping switch contacts C-1 to B-1, and K47 (K25) and K46 (K26) contacts.
   f. K2 is picked through CR11.
   g. K2 is held by 24V dc through K6 contacts, CPU READY/OFF switch, thermal and overcurrent sense relay contacts, CR7, and K2 contacts.

h. K5 is momentarily held by 24V dc through the same protection-relay contacts, CR10, and K5 and K46 (K26) contacts.

i. K46 (K26) is energized by 24V dc through K5 contacts and transfers after a 5-second RC time delay.

j. PK2 is picked by 24V dc through K5 contacts.

k. Main power is applied to the converter/inverter and the blowers through PK2 contacts.

l. The converter/inverter supplies 140V, 2500-Hz ac to the regulators.

m. As the regulators develop the dc power for the CPU logic, undervoltage sense relays K10 and K11 are picked.

n. K5 is now held by 24V dc through the same protection-relay contacts as before, CR10, and K5, K10, and K11 contacts.

Note: If K10 and K11 are not picked by the time K46 (K26) transfers, K5 is not held, thus dropping PK2 and K46 (K26).

o. K47 (K25) drops as K10 and K11 pick.

p. If PS9 is in the CPU, it supplies 48V dc to bus II BB 9 through relay II-K1 contacts. II-K1 is picked by 24V dc through K5, K10, and K11 contacts.

q. Momentary continuity through K35 contacts starts the power-on reset operation described in Chapter 6, Section 1. K35, which is shown in Diagram 6-6, A, FEMDM, is in series with CR15, a 100-uf capacitor, and a set of normally open contacts of K46 (K26) to the 24V dc bus. The initial charging current of the capacitor is sufficient to pick K35. As the charge on the capacitor builds up, the current becomes insufficient to hold K35, causing it to drop.

r. The stepping switch is advanced to position 2 by 24V dc through K10, K11, and stepping switch contacts A-1 to A-COM.

Full system power is achieved in a sequential and interlocked manner by the power control interface circuitry. After initiating power on in the CPU, the CPU stepping switch waits for CPU power-up to be confirmed, then advances to an attached stand-alone unit, directs it to bring its power up, waits for power-up to be confirmed, and advances to the next unit, repeating this procedure until power is up in all attached units. Each stand-alone unit has its own internal power-sequencing control. Diagram 6-7, FEMDM, shows a typical power control interface connection, using channel 1 as an example.

With the stepping switch at position 2, the system power-on sequence continues automatically.

s. 24V dc (supplied by channel 1) through K4, K6, and stepping switch contacts C-2 to B-2 provide channel 1 with a 'power pick' level. This level initiates internal power-on sequencing of channel 1. When channel 1 has completed power-on sequencing, 24V dc (from the CPU) are returned as a 'power complete' signal through stepping switch contacts A-2 to A-COM to advance stepping switch to position 3. K19 is picked and held at the same time.

t. Channel 2 is picked using position 3, K4, and K6 contacts. The 'power complete' signal advances the stepping switch to position 4 and picks and holds K21.

u. Channel 3 is picked using position 4, K4, and K6 contacts. The 'power complete' signal advances the stepping switch to position 5 and picks and holds K31.

v. The stepping switch is advanced to position 11 by 24V dc at stepping switch contacts A-5 through A-10 to A-COM.

w. Storage unit 1 is picked using position 11, K4, and K7 contacts. The 'power complete' signal advances the stepping switch to position 12 and picks and holds K29.

x. Storage unit 2 is picked using position 12, K4, and K7 contacts. The 'power complete' signal advances the stepping switch to position 13 and picks and holds K30.

y. Storage unit 3 is picked using position 13, K28, and K7 contacts. The 'power complete' signal advances the stepping switch to position 14 and picks and holds K20.

z. Storage unit 4 is picked using position 14, K28, and K7 contacts. The 'power complete' signal advances the stepping switch to position 15 and picks and holds K22.

aa. LCS unit 1 is picked using position 15, K28, and K32 contacts. The 'power complete' signal advances the stepping switch to position 16.

ab. LCS unit 2 is picked using position 16, K28, and K32 contacts. The 'power complete' signal advances the stepping switch to position 17.

ac. LCS unit 3 is picked using position 17, K28, and K32 contacts. The 'power complete' signal advances the stepping switch to position 18.

ad. LCS unit 4 is picked using position 18, K28, and K32 contacts. The 'power complete' signal advances the stepping switch to position 19.

Note: If any of these units are not attached to the CPU, a jumper must be installed to provide a simulated 'power complete' level return to advance the stepping switch to the next position. If any unit is attached to the CPU but is not being used in the system, a LOCAL/REMOTE switch in the unit allows it to be bypassed in the sequencing. In LOCAL, power control is at the unit, except for EPO. In REMOTE, the unit is sequenced on by the stepping switch.

ae. In converted units, the stepping switch is advanced to position 26, the start position, by 24V dc at stepping switch contacts A-19 through A-25 to A-COM. This action completes the system power-on sequence of the converted units.

af. In original units, the stepping switch is advanced to position 25 by 24V dc at stepping switch contacts A-19 through A-24 to A-COM.

ag. K35 is momentarily picked through stepping switch contacts B-25 and C-25, CR16, and another 100-uf capacitor. In large system configurations, the 'power-on reset' signal follows the one issued when K46 was picked. This action completes the system power-on sequence of the original units.

5. Depress CPU ON pushbutton on CE panel.
   a. K5 is picked by 24V dc through K12 contacts, CPU ON pushbutton, and K47 contacts.
   b. K2 is picked through CR11.
   c. K2 is held by 24V dc through K12 contacts, CPU READY switch, thermal and overcurrent sense relay contacts, CR7, and K2 contacts.
   d. K5 is momentarily held by 24V dc through the same protection-relay contacts, CR10, and K5 and K46 contacts.
   e. The sequencing continues as described in steps 4i through 4q. (The holding 24V dc, however, is through K12 contacts rather than K6 contacts.)

POWER-OFF SEQUENCE

CPU power-off can be initiated manually or automatically. Secondary power in the CPU is dropped if the CPU READY/OFF switch is placed in OFF, if an automatic power-off sequence is initiated, or if a system POWER OFF pushbutton is depressed (as well as all other remotely controlled units). Primary and secondary power of all units is dropped if an EPO switch is pulled.

Normal Power-Off

- Depress POWER OFF to drop system secondary power.
- Move CPU READY/OFF switch to OFF to drop only CPU secondary power.

System secondary power-off is initiated at the system control panel or at the 2150 Console; the CPU and channel power goes off after the storage units to protect the stored data. CPU secondary power can be separately controlled at the CE panel; however, if any attached storage unit is on, data in storage may be lost. Diagram 6-8, FEMDM, shows the power-off sequence, which is as follows:

1. Depress POWER OFF on either system control panel or 2150 Console. As a result:
   a. The 24V dc holding line to K7 and K32 is opened, causing them to drop.
   b. The stepping switch is advanced to position 26 when the step relay driver is energized through the normally closed contacts of K32. (In converted units, the stepping switch is already in position 26.)
   c. The 'power hold' line to the storage units, through K7 and K32 contacts, is opened, causing them to cycle down unless they are shared with another CPU that is still on.
   d. As the last storage unit goes down, the 24V dc holding K6 drops, causing K6 to drop.
   e. The 24V dc holding line to K12, K5, and K2, through K6 contacts, is opened, causing them to drop.
   f. The 24V dc holding line to PK2, through K5 contacts, is opened, causing it to drop.
   g. Main power to the converter/inverter, through PK2 contacts, is opened, causing CPU dc to go off.
   h. AC power through PK2 contacts is opened to the blowers and to the 1052 Printer.
   i. The 'power hold' line to the channels, through K6 contacts, is opened, causing them to cycle down. (They do not cycle down if in local-control mode or if equipped with a Two-Channel Switch feature and the other interface has its power on.)
   j. K47 (K25 in converted units) is again picked through the path described in the power-on sequence, step 3f, allowing dc power to be turned on again.
2. Move CPU READY/OFF switch on CE panel to OFF.

   Note: Under these conditions, data may be lost in the storage units as the CPU dc goes down.

   a. The 24V dc holding line to K5 and K2 is opened, causing them to drop.
   b. Same as steps 1f through 1h of the normal power-off sequence.
   c. If system power was up, it remains up because K6, K7, and K32 did not drop.
   d. The CPU dc power cannot be turned on again until K47 (K25) is again picked by returning the CPU READY/OFF switch to CPU READY and waiting 5 seconds. Both the system and CPU power-on pick paths are open when K47 (K25) is dropped.

Emergency Power-Off

- Pull any EMERGENCY PULL (EPO) switch.

Emergency power-off in the system is initiated by pulling the EMERGENCY PULL (EPO) switch at either the system control panel or at the 2150 Console, or by losing continuity through the Emergency Power-Off Control Feature (Diagram 6-7, FEMDM).

Note: All EMERGENCY PULL switches latch mechanically and must be reset by the CE.

For safety, the Emergency Power-Off Control Feature may be installed whenever more than one system is in an area or room; the feature is *always* installed when the systems are interconnected by other than communication lines. Feature No. 3621 is shared by two systems; feature No. 3622 can be shared by up to 12 systems. If the feature is added to the CPU, it is located in frame 4. Any EPO switch can initiate an emergency power-off in all systems sharing the feature. While correcting the power fault in one system, the power of the remaining system(s) can be returned by means of the NORMAL/BY-PASS switch located on the feature. The switch should be placed in BY-PASS by a CE only after the cause of the EPO event has been identified; the switch should be returned to NORMAL as soon as the cause has been corrected.

Diagram 6-7 shows the EPO circuits in the stand-alone units using the CPU and channel 1 as an example. Within the CPU, the following takes place during an emergency power-off:

1. The 24V holding line to K4, through the EPO switches and the Emergency Power-Off Control Feature, is opened, causing K4 to drop.
2. The 24V dc holding line to PK1, through K4 contacts, is opened, causing PK1 to drop.
3. Main power to the CPU, except to T1, through PK1 contacts, is opened, causing a complete power-off. T1 provides the low-voltage ac to a bridge rectifier for the dc to establish the EPO loop.
4. The EPO-hold line to the stand-alone units, through K4 or K28 contacts, is opened, causing them to drop complete power, even if they are in the local-control mode.

Automatic Power-Off

Secondary CPU power is sequenced down if an overcurrent (K8), overvoltage (also K8), undervoltage (K10 or K11), or thermal (K9, K14–K17, or K24) condition is detected. K8, K9, K14–K17, and K24 contacts are in series with the CPU READY/OFF switch; transfer of any of these relays causes a CPU power-off sequence similar to the sequence when the CPU READY/OFF switch is moved to OFF. Refer to the discussion of normal power-off, steps 2a through 2c. Because K47 (K25 in converted units) remains dropped, the cause of the power-off must first be corrected before CPU dc power can again be turned on.

The dropping of K10 or K11 results in a similar power-off sequence, except K2 remains held and K47 picks after a 5-second delay. Because K47 is picked, the CPU dc power may be turned on again; however, unless the faulty regulator voltage to the undervoltage sense circuit is corrected or turned off, the CPU power is again automatically removed.

Overcurrent Protection

The overcurrent sensing circuits are internal to each regulator. (Refer to *SLT Power Supply,* FEMI, Form 223-2799, for circuit details.) Any fault that draws excessive current from any regulator, except PS9, causes the CPU to drop power.

Figure 5-4, A, shows the overcurrent sense loop for original units. When an overcurrent condition exists, continuity is provided between terminals 8 and 9. 24V dc, through auxiliary switch contacts on CB2, CB7, or CB8 in 60-Hz units (or CB2, CB3, and CB4 in 50-Hz units) or through terminals 8 and 9 of the regulators except PS9, pick K8, the overcurrent sense relay.

Figure 5-4, B, shows the overcurrent sense loop for converted units. When an overcurrent condition exists, continuity between terminals 8 and 9 is broken. 24V dc, through auxiliary switch contacts on CB2 through CB8 and through terminals 8 and 9 of the regulators except PS9, drop K8, the overcurrent sense relay.

Terminals 8 and 9 of PS9 are not in the overcurrent sense loop because PS9 is used only by the 1052 Adapter feature. 24V dc, through terminals 8 and 9 of PS9, pick K18, the 48V power-check relay. (K18 is dropped in converted units.)

Overvoltage Protection

- Excessively high regulator output voltage causes CPU dc power to drop via overcurrent sense loop.
- Original units have overcurrent/overvoltage SMS card in each regulator.
- Converted units have positive and negative overcurrent circuits external to regulators.

Any fault that raises the output voltage level of any regulator except PS9 above a preset level causes the CPU to drop power via the overcurrent sense loop. (Because PS9 is not required for CPU operation, an overvoltage condition sensed in that regulator only lights power fault indicators.) An SCR, in series with a low resistance, is connected across the output terminals of each regulator except PS9. When gated on, the SCR shorts the regulator output through the low resistance, causing an overcurrent condition within the regulator. The internal overcurrent protection circuit initiates a CPU power-off sequence.

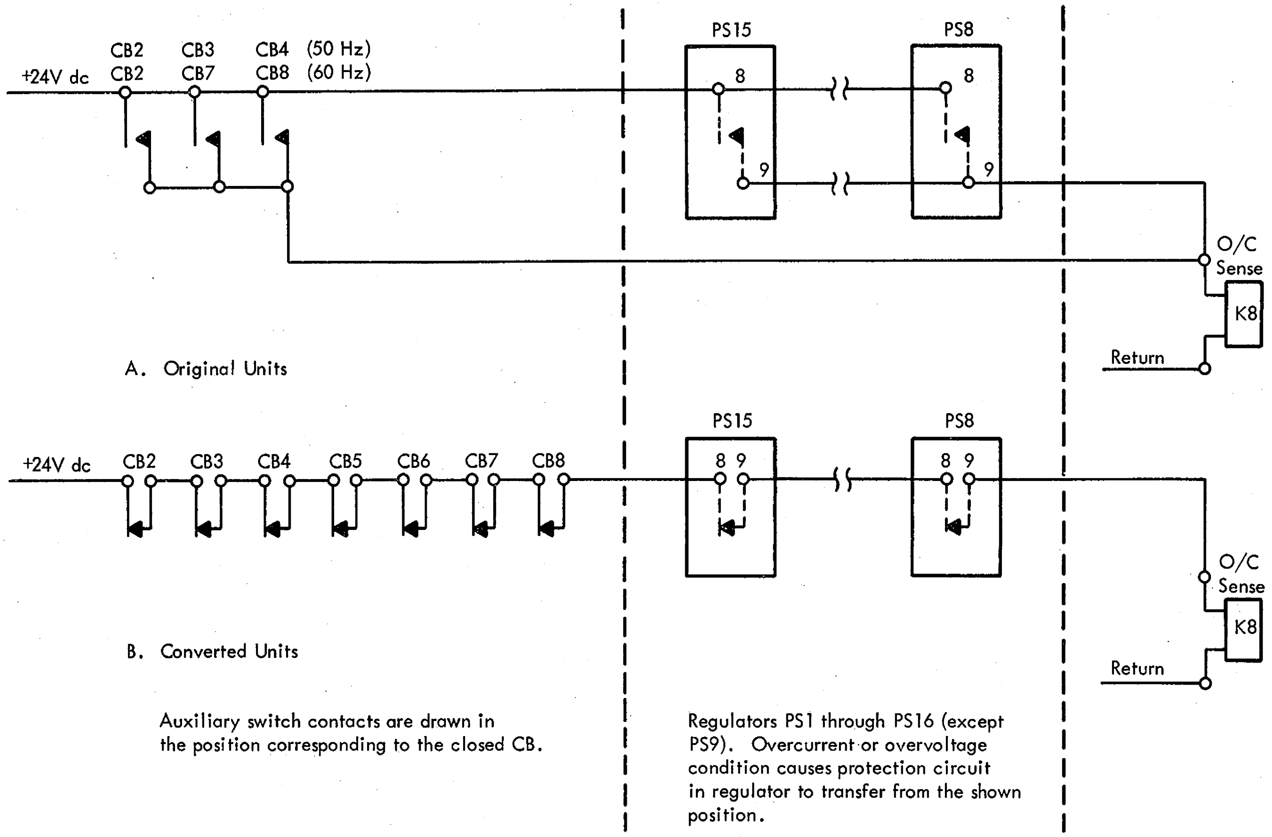

CB2
CB3
CB4 (50 Hz)
+24V dc
CB2
CB7
CB8 (60 Hz)
PS15
PS8
8
9
O/C
Sense
K8
Return
A. Original Units
+24V dc
CB2
CB3
CB4
CB5
CB6
CB7
CB8
8 9
B. Converted Units
Auxiliary switch contacts are drawn in the position corresponding to the closed CB.
Regulators PS1 through PS16 (except PS9). Overcurrent or overvoltage condition causes protection circuit in regulator to transfer from the shown position.

Figure 5-4. Overcurrent Protection Loop

The original units have separate overvoltage protection circuits for each regulator. An external assembly mounted on the output bus senses the output voltage and provides the low-resistance path through an SCR; the internal overcurrent/overvoltage SMS card provides the gating voltage to turn on the SCR. The overvoltage protection circuits in converted units have some logic in common with all the regulators (see logic YA091). Also, the overvoltage protection circuits in converted units are completely external to the regulators. Their operation is discussed in the following paragraphs.

Positive Regulators, Converted Units. A line from each positive regulator except PS9 is fed through a metering jack to the emitter of a sensing transistor. The base of this transistor is held at a preset upper limit voltage by Zener diode Z1 and potentiometer R4 (3V supplies) or R9 (6V supplies). The collector series load resistors are connected to ground or 0V. When its emitter is held negative with respect to its base by the regulator, the transistor is biased off, holding the collector circuit and the SCR gate at 0V. When the regulator output rises above the upper limit, causing the emitter to become positive with respect to the base, the transistor starts to conduct. The current drawn through the collector load resistors raises the collector circuit and the SCR gate to a positive level with respect to the regulator output level, gating the SCR on.

Negative Regulators, Converted Units. A line from each negative regulator except PS11 provides, through a metering jack, the return path for the collector circuit of an SCR gating transistor and the emitter of a sensing transistor. The base of the sensing transistor is held at a preset negative voltage limit by Z2 and R7. The collector load resistor is connected to ground or 0V. The base of the SCR gating transistor is fed by a resistor from the collector of the sensing transistor. The emitter of the SCR gating transistor is connected to ground or 0V. The sensing transistor, when its emitter is held positive with respect to its base by the regulator, is biased off, and its collector circuit and the base of the SCR gating transistor are held at 0V. As a result, the SCR gating transistor is biased off, and its collector circuit and the SCR gate are held at the regulator output level. When the regulator output level increases beyond the negative voltage limit, causing the emitter of the sensing transistor to become negative with respect to its base, the transistor starts to conduct. The current drawn through the collector load resistor lowers its collector circuit and the base of the SCR gating transistor to a negative level. With its base made negative with respect to its emitter, the SCR gating transistor starts to conduct. The current drawn through the collector load resistor raises the collector circuit and the SCR gate to a positive level with respect to the regulator output level; the SCR is gated on.

Regulator PS11 is sensed in the same way as the other negative supplies, except that the emitter voltage of the sensing transistor is established through an adjustable voltage-dividing network in the line from the regulator. This dividing network permits the sensing transistor to use the same voltage reference (Z2 and R7) as the other negative supplies.

Undervoltage Protection

The logic voltage sensing circuits provide an indication that all CPU dc is above minimum voltage levels (2.4V). Any fault that lowers the output voltage level of any regulator except PS9 below the minimum causes CPU power to drop. (See logic YA111 for the sensing circuits.)

A line from each positive regulator except PS9 is fed through isolation switches, labeled UNDER VOLTAGE CHECK, to an AND. The output level of the AND is determined by the lowest input level; that is, the one nearest zero or ground. Thus, if any supply not isolated is off, the output level of the AND is near zero. However, if all the supplies are on, the output level of the AND is about 3V. The output of the AND is transferred directly to the base of transistor Q1, which shares an emitter load resistor with Q2. The base of Q2 is held at 2.4V by Zener diode CR59 and resistor R8. Q2 now carries the full emitter load current and holds the common emitter circuit to 2.4V. With its emitter held positive with respect to its base, Q1 is biased off. As the output level of the AND raises the base of Q1 to 2.4V, Q1 begins to conduct and starts to share with Q2 the current drawn by the emitter load resistor. As the output level of the AND raises the base of Q1 above 2.4V, Q1 raises the emitter circuit above 2.4V. With its emitter raised to a level positive with respect to its base, Q2 is biased off. Q1 now carries the full emitter load current, which is sufficient to pick K10.

A similar circuit, of opposite polarity, checks the negative regulators and picks K11.

The UNDER VOLTAGE CHECK switches permit CPU dc power to be brought up for servicing purposes, with a low output-voltage level from a regulator.

> CAUTION
> Under these conditions, CPU logic may be damaged by the nonstandard voltages. Refer to the *2065 Processing Unit* FEMM, Form Y27-2039-2, Chapter 5, for undervoltage troubleshooting.

Thermal Protection

- Thermal relays are picked during power-on sequence.
- Overtemperature condition drops thermal relay, causing CPU power to drop and indicators to light.
- THERMAL RESET pushbutton on CE panel must be depressed before CPU power can be restored.

Thermal protection is provided by sensing elements (thermostats) in the return path of each thermal relay. If an overtemperature condition exists, the return path is opened, dropping the associated relay (causing CPU dc to drop) and lighting an indicator. The thermostats are at the top of the following areas: logic gate A (K9), gate B (K14), gates C and D (K15), gate E (K16), high-frequency regulator modules (K17), and the converter/inverter (K24). Logic YA022 shows the relay and indicator circuits.

The thermal relays are automatically reset when the 24V dc bus is energized by PK1 contacts, and are manually reset when the THERMAL RESET pushbutton on the CE panel is depressed.

Relay K3, which provides the 24V dc pulse to reset the thermal relays, is in series with a 1500-uf capacitor and a set of normally closed contacts of K46 (K26 in converted units) and the 24V dc bus. The initial charging current of the capacitor is sufficient to pick K3. As the charge on the capacitor builds up, the current becomes insufficient to hold K3, causing it to drop. [When K46 (K26) transfers, the capacitor is discharged through a 12-ohm resistor. Relay K46 (K26) transfers about 5 seconds after CPU ON or POWER ON are depressed.] The manual path for picking K3 is through the THERMAL RESET pushbutton to the 24V dc bus.

24V dc, through the transferred contacts of K3 and through diodes CR1–CR6, cause the thermal relays to pick, providing normal temperature conditions exist (all thermostats closed). Each thermal relay holds through its own contacts as K3 drops. An open thermostat causes the relay to drop, and its contacts light the associated indicator on the CE panel.

INDICATORS

Indicators on the system control panel show an incomplete power-up status in the CPU or in the attached HSS units and channels. The CPU and the units are individually indicated, as is the system status. (See logic YA026 and YA083, and Diagram 8-1, FEMDM.) The system status is shown by the POWER ON pushbutton. This pushbutton is normally white but glows red when system power is incomplete. This condition is also indicated on the 2150 Console.

System Power-On Indicator

24V dc, through the CPU voltage-sense relay contacts (K10 and K11) and the HSS units and channels power-complete relay contacts (K19–K22 and K29–K31), light the System Power-On indicator (clear or white lamp backlighting the POWER ON pushbutton). If any of these relays is not picked, the indicator remains off and the dual lighted POWER ON pushbutton glows red as described below.

Power Check Indicators

There are nine power check indicators:

1. CPU POWER CHECK. 24V dc through K2 contacts are fed through parallel-connected contacts on voltage-sense relays K10 and K11 to the CPU POWER CHECK indicator. If either of these two relays is not picked after K2 is picked, this indicator lights. 24V dc are also fed through parallel-connected contacts on overcurrent relay K8 and thermal-sense relays K9, K14–17, and K24 to the CPU POWER CHECK indicator. 24V dc, through 48V power check relay contacts K18 or through the CPU READY/OFF switch in OFF, also light this indicator.
2. STOR FRAME 1 POWER CHECK. 24V dc, via K32 contacts, are fed through K29 contacts to this indicator. If K29 is not picked after K32 is picked, this indicator lights.
3. STOR FRAME 2 POWER CHECK. Similarly, through K30 contacts, this indicator lights.
4. STOR FRAME 3 POWER CHECK. Similarly, through K20 contacts, this indicator lights.
5. STOR FRAME 4 POWER CHECK. Similarly, through K22 contacts, this indicator lights.
6. CHAN FRAME 1 POWER CHECK. Similarly, through K19 contacts, this indicator lights.
7. CHAN FRAME 2 POWER CHECK. Similarly, through K21 contacts, this indicator lights.
8. CHAN FRAME 3 POWER CHECK. Similarly, through K31 contacts, this indicator lights.
9. System Power Check. A line from the CPU POWER CHECK indicator and a line from each unit power check indicator are combined at an OR. The OR turns on the System Power Check indicator. The System Power Check indicator also lights when any of the unit power check indicators is on. Because the bulb of the System Power Check indicator is red and backlights the translucent POWER ON pushbutton, the pushbutton glows red.

1052 PRINTER-KEYBOARD POWER

The CPU contains the +48V dc power supply (PS9) and the audible alarm used by the 1052 Adapter feature if a 2150 Console is not attached to the system. The power circuits added to the CPU for the 1052 Printer-Keyboard and the audible alarm are discussed here and are shown in Diagram 6-9, FEMDM. The manner of sequencing on the +48V dc has been described in step 4p of the power-on sequence.

Audible Alarm

The audible alarm provides the operator with a signal when directed by the program. It may be used, for example, to signal a program hangup or the end of a program.

The 48V dc through relay A-K2 have a return path through the 1052 Adapter logic. When logic grounds the return path, A-K2 is picked. A-K2 is then held by A-K1 and A-K2 contacts. 28V ac, through A-K1 and A-K2 contacts, starts the bell ringing. The 24V dc, through A-K2 contacts, picks A-K1. A-K1 has a short delay before it transfers; A-K2 drops and the bell stops ringing as A-K1 transfers.

Dual 1052 Power Interface

The CPU provides +48V dc for the keyboard of the 1052 via one connector and +48V dc and ac power to the printer via another connector. When a second 1052 is added, a Dual 1052 Adapter junction box is plugged into this second connector. The adapter box has two sets of connectors, one set for the 1052 on the right side of the system console, the other set for the 1052 on the left side. Four switches on the junction box allow the power in any cable to be turned off so that a faulty 1052 may be removed for maintenance without removing CPU power.

DIRECT CONTROL POWER

- Spurious direct control signals are avoided during normal CPU power transitions.

Two precautions are taken to ensure that spurious signals are not issued via the direct control feature during normal CPU power transitions: (1) the stepping switch is prevented from generating a 'power-on-reset' signal until K46 (K26 in converted units) has picked, and (2) signal line drivers with gated power are used. Diagram 6-6, B, FEMDM, shows the relay logic.

24V dc go through K46 normally-open contacts to position 25 contacts of the stepping switch, ensuring that the direct control circuits are reset after all regulators have reached their nominal voltage levels. Also, K47 (K25 in converted units) is again energized by 24V dc through the same K46 and stepping-switch position 25 contacts, CR17, K5 contacts, and CR18. After the 5-second transfer delay of K47, the power and logic of the system is quiescent. K50 is then picked through K47 contacts, turning on the +6V dc to the G-register and timing-signal line drivers. Until then, no direct control signals may be issued to the external device. (If only CPU power is turned on, the stepping switch remains in position 26. In this case, only one power-on reset occurs; K46 is picked by 24V dc through K46, K7, and K5 contacts and CR18.)

Although K50 picks last during a power-on sequence, it drops first during a normal power-off sequence. If a system POWER OFF pushbutton is depressed, the stepping switch is immediately advanced to position 26, removing the 24V dc holding K50; if the CPU READY/OFF switch is placed in OFF, K5 immediately drops, removing the 24V dc holding K50. With K50 de-energized, the direct control interface line drivers are grounded, disabling them from transmitting a spurious signal.

USAGE METERS AND KEY SWITCH

- Process (left) meter records time customer used system.
- System maintenance time is recorded on CE (right) meter by key-switch control.

Two meters on the system control panel show the system running time while it is processing customer data (process meter) and while it is being operated by the CE (CE meter). (The process meter is also known as the cluster, central processing complex, system, and customer meter.) A key-operated switch selects the meter to be driven. The normal position of this switch allows power to be applied to the process meter. The CE, using a key, switches to the CE meter. The key cannot be removed when the switch is in the CE position.

The meter selected by the key switch records time as long as the 'pass pulse' trigger is set if (1) not in the Wait state and the 'manual' trigger is not set or (2) a 'meter in' signal is received from the channels. The selected meter also runs momentarily when any pushbutton on the system control panel is depressed (except STOP while already in the manual mode). The 'pass pulse' trigger is set by a start, load, or reset operation and, when in single-cycle-mode operation, is reset after gating one pulse (refer to "Clock Control and Signal Distribution", Chapter 2, Section 1). In general, the recording of time stops while a CPU machine check condition exists; however, the recording continues if, during an IPL operation, the CPU CHECK switch is in the DSBL position when the check condition occurs or if, during any other operation, the STOP ON STORAGE CHECK switch is depressed (activated) when the check condition occurs.

The multiplexed 'meter in' signal from the channels is received whenever an I/O operation initiated by the CPU is still in progress and requires recording. The CPU sends two signals to the channels. The 'CPU clock not stopped' signal is sent unless the CPU is in the Wait state or manual mode or is not recording time due to a CPU machine check. The 'meter out' signal is sent whenever time is being recorded on the process meter. These two signals are used by the channels and their attached units in running their own usage meters and, in turn, in generating the 'meter in' signal.

Diagram 8-28, FEMDM, shows the CPU metering circuits. The selected meter runs for a minimum of 400 ms. Relay RR1 on the usage meter card is picked by a signal from the CPU logic. 40V ac through RR1 contacts drive the meter.

Chapter 6. Console Controls and Maintenance Features

Maintenance aids available to the CE for the 2065 CPU fall into two categories: (1) those used for error detection during normal operation, such as error detection logic and interruptions (discussed in Chapter 1, Section 3), and (2) those aids used for diagnosing the cause of failures and for preventive maintenance. This second category includes:

1. System control panel: Contains the controls necessary for initiating any operation, for manual testing, and for performing various maintenance tasks. In addition, indicators allow monitoring of the CPU operation by displaying the status of important registers and control triggers.
2. CE panel: Contains the controls necessary for initiating power on and off sequences. In addition, indicators show the power status.
3. Diagnose instruction: Allows certain diagnostic functions to be performed on the CPU or the channel. It is used in conjunction with the maintenance control word (MCW) to allow such diagnostic functions as reversing parity and suppressing data checks. In addition, it can be used to initiate the logout and FLT functions.
4. MCW: Used in conjunction with the Diagnose instruction, FLT's, and ROS tests.
5. Logout, ROS tests, and FLT's: Logout stores the status of the console indicators into fixed positions of main storage when a trouble symptom occurs; the data logged out may be subsequently recalled for analysis although the status of the indicated logic is changed from what it was when the symptom appeared. ROS tests check each bit position of every ROS word. FLT's check the CPU at the logic-block level.
6. Ripple tests: Provide the capability of (1) storing data from the DATA switches into all addresses in LS or main storage and inserting 1's with correct parity into the storage-protect keys, and (2) reading out all locations of LS or main storage and displaying the data.
7. Diagnostic programs: Check the CPU on a functional basis by programming the CPU to perform one or more instructions or sets of instructions.
8. Marginal checking: Allows operation of critical circuits with nonstandard voltages to detect if any are approaching failure.

The maintenance aids provide the CE with a wide choice of troubleshooting techniques when isolating a fault. Thus, the CE can tailor his troubleshooting procedure to suit the particular problem and his own experience.

The maintenance aids have interrelated functions, dependent upon the troubleshooting technique used. For example, scan logic, which provides the control necessary to perform ROS tests and FLT's, is also used in logout, a system control panel function. On the other hand, switches on the system control panel are used to initiate all maintenance programs (i.e., FLT's, ROS tests, and diagnostics).

This chapter is divided into two sections. Section 1, Console Controls, discusses the manual controls and indicators on the system control panel and on the CE panel, and their application. Section 2, Maintenance Features, discusses the Diagnose instruction, MCW's, logout, ROS tests, FLT's, ripple tests, diagnostic programs, and marginal checking.

Section 1. Console Controls

SYSTEM CONTROL PANEL

The system control panel, in addition to its main function as the operating and monitoring center of the system, is one of the prime maintenance aids available to the CE. Using this panel, a CE can duplicate many program operations or portions of operations manually and can repeatedly exercise portions of the machine logic at a normal or a reduced rate of operation.

Some of the maintenance routines that can be performed from the system control panel include storage ripple, marginal checking, and frequency bias. Other controls allow the CE to stop the CPU at the end of the current instruction, to display main storage or LS, to store into main storage or LS, and to log out indicator status to fixed positions in main storage. (The latter is a function of the scan logic and is described in Section 2 of this chapter.)

The system control panel is divided into seven separate panels (A to G), as shown in Diagram 8-1, FEMDM. The operator control section (panel G) is identical to those used on the other models of the System/360, thus providing the operators with compatibility between models. If the 2150 Console or the 2250-1 Display Unit is included in the Model 65 System, the operator control section and the EMERGENCY PULL switch may be duplicated on one of these units to provide monitor control operation.

The functions of the controls and indicators, by panel, on the system control panel are defined briefly in the following listing:

PANEL A

1. DC voltmeter. Indicates the voltage levels of the marginable supplies. The particular supply indicated is determined by the MARGIN/METER SEL switch.
2. MARGIN/METER SEL switch. Has 12 positions to select the power supply to be indicated by the meter and to determine which of the attached stand-alone units may be marginally checked:
   a. STORE FRAME 1: Selects HSS unit 1.
   b. STORE FRAME 2: Selects HSS unit 2.
   c. STORE FRAME 3: Selects HSS unit 3.
   d. STORE FRAME 4: Selects HSS unit 4.
   e. CHAN FRAME 1: Selects channel 1.
   f. CHAN FRAME 2: Selects channel 2.
   g. CHAN FRAME 3: Selects channel 3.
   h. ROS LOCATE: Selects the 18V ROS bias power supply (gate D in the CPU).
   i. CPU A: Selects gate A in the CPU.
   j. CPU B: Selects gate B in the CPU.
   k. CPU C: Selects gate C in the CPU.
   l. CPU E: Selects gate E in the CPU.

   Note: When an Additional Storage Attachment feature is installed in CPU2, STORE FRAME 1, 2, 3, and 4 select HSS units 5, 6, 7, and 8, respectively.

3. MARGIN indicators:
   a. ACTIVE: Indicates that an internal power supply or an attached HSS unit or channel is being marginally checked.
   b. LOCATE: Indicates when the MARGIN/METER SEL switch is set to the position corresponding to the margined power supply, attached HSS unit, or channel.
4. POWER CHECK indicators. These eight indicators (CPU, STOR FRAME 1, 2, 3, and 4, and CHAN FRAME 1, 2, and 3) indicate an incomplete power-up status in the CPU, HSS units 1, 2, 3, and 4, and channels 1, 2, and 3, respectively.

   Note: When an Additional Storage Attachment feature is installed in CPU2, STOR FRAME 1, 2, 3 and 4 indicate an incomplete powerup status in HSS units 5, 6, 7, and 8, respectively.

5. MARGIN CHANNEL/STOR switch. Applies power to a motor in the channel or HSS unit selected by the MARGIN/METER SEL switch to lower or raise the output voltage levels from the marginable supplies in that unit or channel. The channel or storage unit must be in remote operation.

PANEL B

The five potentiometers on this panel (ROS, +6M A GT, +6M B GT, +6M C GT, and +6M E GT) raise or lower the output voltage levels from the 18V ROS and the 6V gate A, B, C, and E supplies, respectively.

PANEL C

The pull switch on this panel, EMERGENCY PULL, when pulled, initiates emergency power-off in the system. The switch latches in the out position and must be manually restored by the CE.

PANEL D

This panel is blank in the basic 2065. When an Additional Storage Attachment feature is installed, a four-position STORAGE INDICATE rotary switch is added:

1. STOR 1–4: Allows the contents of the storage address registers for HSS units 1 and 2 to be displayed on the roller switches.
2. STOR 5–8: Indicates HSS units 3 and 4.
3. STOR 9–12: Indicates HSS units 5 and 6.
4. STOR 13–16: Indicates HSS units 7 and 8.

PANEL E

1. Roller switches and indicators. This section of the panel contains six six-position roller switches, with 36 indicators (implicitly numbered 0–35) associated with each roller switch. A roll chart associated with each roller switch rotates with the roller switch to correspond with the roller position, thus showing the information being displayed for each indicator. Diagram 8-2, FEMDM, identifies the indicators for the six positions of each roller switch. These indicators are tested between positions of the switch. Position 6 of roller 6 is used to test the remaining indicators on the system control panel and on the 2150 Console.
2. DATA 0–31 and DATA 32–63 switches. These 64 switches, in hexadecimal groups, permit data to be entered manually. Correct parity is automatically generated by contacts on the switches.
3. ADDRESS switches. These 24 switches, in hexadecimal groups, select an addressable location in ROS, LS, or main storage. Correct parity is automatically generated by contacts on the switches.
4. STOR CHK (storage check) indicator: Indicates an error in the storage units.
5. PROC CHK (processor check) indicator: Indicates an error in the CPU.

PANEL F

1. TEST MODE, REPEAT switch: Repeats the ROS test or FLT in main storage continuously.

2. TEST MODE, ROS/PROC/FLT switch. This switch has three positions:
   a. ROS: Places the CPU into ROS test mode and removes program control.
   b. PROC (process) – normal position for CPU operation.
   c. FLT: Places the CPU in the FLT mode and removes program control.
3. STORAGE INDICATE switch. This switch has two positions:
   a. STOR 1–4 - normal position: Allows the contents of the storage address registers for HSS units 1 and 2 to be displayed on the roller switches.
   b. STOR 5–8: When in this position, the operation is the same as in the normal position for all Model 65 configurations except Models IH and J. For these models, the contents of the storage address registers for HSS units 3 and 4 are displayed on the roller switches.

Note: This switch is spare if an Additional Storage Attachment feature is installed.

4. FREQUENCY ALTERATION switch: Decreases the CPU clock period from 200 ns to 195 ns. Operates only with the CE key switch in the CE position.
5. DEFEAT INTERLEAVING switch. This switch has three positions:
   a. PROC (process) – normal position: Addressing is interleaved with no even/odd storage area reversal, unless changed by the Diagnose instruction.
   b. REV (reverse): Interleaving is disabled and the even and odd storage areas are reversed. No reversal occurs on Model G65.
   c. NO REV (no reverse): Interleaving is disabled and no even/odd storage area reversal takes place.
6. STOP ON STORAGE CHECK switch: Inhibits operation of and maintains the environment of the main storage basic storage module in which an error was detected. Other basic storage modules are not affected.
7. DISABLE INTERVAL TIMER switch: Prevents the interval timer from being decremented.

   Note: When the Multisystem feature is installed, disabling the interval timer when operating in the multisystem mode also disables the detection of CPU inactivity.

8. STORAGE SELECT switch. This switch has three positions:
   a. MAIN – normal position: Selects main storage for manually storing or displaying data.
   b. LOCAL: Selects LS for manually storing or displaying data.
   c. MAIN BYTE: Same as MAIN except that the byte selected by ADDRESS switches 21–23 is the only byte affected by a manual store operation.
9. ADDRESS COMPARE STOP switch: Stops processing if the main storage address being requested agrees with bits 2 through 20 of the ADDRESS switches.
10. CPU CHECK switch. This switch has three positions:
    a. PROC (process) – normal position: If the PSW machine check mask [PSW(13)] is a 1, the CPU stops on detection of a CPU check and the status is logged into main storage. A machine check interruption then takes place. If the mask is a 0, the result is the same as if the switch is in the DSBL position.
    b. DSBL (disable): The CPU does not stop on detection of a machine check, but the check trigger is set. No logout or interruption takes place.
    c. STOP: The CPU stops on detection of a machine check, but there is no logout of data. The check trigger is set.
11. PULSE MODE switch. This switch has three positions:
    a. PROC (process) – normal position: Does not affect CPU operation.
    b. COUNT: Provides a means of looping through a selected number of machine cycles (maximum of 2047). The number of cycles is entered into DATA switches 53–63. Each loop starts at the address contained in main storage location 0. The RATE switch must be in the PROCESS position.
    c. TIME: Provides looping when the interval timer is decremented. Each loop starts at the address contained in main storage location 0. The RATE switch must be in the PROCESS position.
12. REPEAT INSN (instruction) switch. This switch has three positions:
    a. PROC (process) – normal position: Does not affect CPU operation.
    b. SINGLE: Allows the first instruction in the DATA switches to be repeated continuously.
    c. MPLE (multiple): Allows continuous looping through the four instruction halfwords in the DATA switches.
13. REPEAT ROS ADDRESS switch: Continuously reads out the ROS address specified by ADDRESS switches 0–11. ROS TRANSFER must be depressed to start this loop.
14. DISABLE DIRECT CONTROL switch. Causes Write Direct and Read Direct instructions to become invalid instructions.

Note: The DISABLE DIRECT CONTROL switch is added by the Direct Control feature. Refer to Chapter 4 if the Multisystem feature is also installed.

15. RATE switch. This switch has four positions:
    a. INSN STEP (instruction step): CPU executes one machine instruction for each time START is depressed.
    b. PROCESS: Does not affect CPU operation; CPU operates at normal clock speed.
    c. SINGLE CYCLE: CPU advances by its minimum clock amount for each depression of START; all CPU operations are the same as for the PROCESS position.
    d. SINGLE CYCLE STORAGE INHIBIT: Same as SINGLE CYCLE without storage references.
16. SYSTEM RESET pushbutton: Resets main storage check triggers, on-line channels and control units, and CPU controls and check triggers to their initial state.
17. CHECK RESET pushbutton: Resets all CPU check triggers and latches to the non-error state.
18. PSW RESTART pushbutton: Loads a new PSW from main storage location 0 and starts processing if the RATE switch is in PROCESS.
19. ROS TRANSFER pushbutton: Serves to display the contents of any ROS location or to begin processing from any ROS address.
20. SET IC (instruction counter) pushbutton: Enters an address from the ADDRESS switches into bits 40–63 of the active (current) PSW. The data referenced will be loaded into Q, the first instruction will be transferred to R, and the IC will be updated by +8 bytes.
21. STORE pushbutton: Enters data from the DATA switches into the storage location specified by the STORAGE SELECT and ADDRESS switches.
22. DISPLAY pushbutton: Displays data from LS or main storage specified by the STORAGE SELECT and ADDRESS switches.

    Note: If the Multisystem feature is installed and prefixing is active when the SET IC, STORE, or DISPLAY pushbutton is depressed, the main storage address set into the ADDRESS switches is relocated.

23. START pushbutton: Starts the CPU operating in the mode selected by the RATE switch.
24. STOP pushbutton: Terminates CPU operation without changing the environment.
25. RESTART FLT I/O pushbutton: Backspaces one tape record and starts reading during ROS test or FLT operations.
26. LOG OUT pushbutton: Stores CPU status into fixed locations in main storage when the CPU is in the Stopped state.
27. Usage meters and CE Key switch: The usage meters indicate elapsed CPU running time: the Process (left) meter shows elapsed customer usage time; the CE (right) meter shows elapsed system maintenance time. The CE Key switch, when turned to the right, disables the Process meter and enables the CE meter and the FREQUENCY ALTERATION switch.

PANEL G

1. POWER ON pushbutton: Initiates power-on sequence in the CPU and the system units. The pushbutton is back-lighted by two indicator lights. When system power is on, the pushbutton glows white; when system power is off or not correctly on, the pushbutton glows red.
2. POWER OFF pushbutton: Initiates power-off sequence in the CPU and the system units.
3. LOAD UNIT switches: These three switches select the I/O unit to be used by a load operation.
4. INTERRUPT pushbutton: Causes an external interruption in the system and sets bit 25 of the interruption code to a 1.
5. SYSTEM indicator: Indicates that a CPU usage meter is running.
6. MANUAL indicator: Indicates the CPU is in the Stopped state.
7. WAIT indicator: Indicates the CPU is in the Wait state.
8. TEST indicator: Indicates that a switch on panel F is not in the normal operating position or that a channel is in the test mode.
9. LOAD indicator: Indicates a CPU load operation. A successful load turns this indicator off.
10. LOAD pushbutton: Resets the system and starts a load operation.

In the following paragraphs, the system control panel is presented in two parts: (1) manual controls, which discusses the function of each control on panels E, F, and G, and describes any interrelationship between the controls; and (2) indicators, which lists the conditions under which each indicator is turned on. Each control and indicator is located by an alphameric designation, referenced to the system control panel co-ordinates. For example, the SYSTEM RESET pushbutton is located at P30; the roller switches are located at D–J, 13–49.

Manual Controls

Manual operations are initiated and controlled by switches and pushbuttons. Pushbutton operation is synchronized with CPU clock signals as shown in Diagram 8-3, FEMDM. Depressing any of the pushbuttons (or performing a power-on reset) generates 'switch SS', 'delayed SS', and 'short SS' signals, which are used during manual control functions.

Functionally, manual controls consist of special logic and ROS microprograms. The CPU must be in the Stopped state before most manual operations can be initiated. The CPU is placed in the Stopped state by:

1. Performing a power-on-reset operation.
2. Depressing SYSTEM RESET (P30).
3. Depressing STOP (S30).
4. Detecting an address-compare condition when the ADDRESS COMPARE STOP switch (N17) is in the stop position.
5. Initiating an instruction-step-mode operation.
6. Initiating a scan operation.

When the CPU is in the Stopped state, it is in a ROS microprogram loop (stop loop) and the MANUAL indicator (S46) is lit.

Note: When the Multisystem feature is installed, the external reset operation generates the pushbutton single-shot signals and simulates depression of SYSTEM RESET, thus placing the CPU into the Stopped state. The external start operation also momentarily places the CPU into the Stopped state.

Stop Loop

- ROS microprogram loop.
- 'Stop' and 'manual' triggers control stop loop.
- Address of next instruction is in D.
- Interruptions and I/O operations are completed before entering stop loop.
- Six pushbuttons are sampled by stop loop: STORE, DISPLAY, SET IC, START, ROS TRANSFER, PSW RESTART.

When the CPU is in the Stopped state, a ROS microprogram, called the "stop loop", is continuously executed. The stop loop is shown by the heavy lines in Diagram 8-4, FEMDM; Stop loop pushbutton gating is shown in Diagram 8-5, FEMDM.

When in the stop loop, no program instructions are executed, the interval timer [location 80, decimal (50, hex), of main storage] is not stepped, and the usage meters are stopped unless a channel is running. Also, the MANUAL indicator (S46) is lit, and the stop loop determines the starting location of the next instruction to be executed and transfers the result to D (contents of IC minus 8 or 16). The contents of D are displayed in roller switch 1, position 2; the contents of the IC are displayed in roller switch 6, position 3.

Two triggers control the stop loop: the 'stop' trigger and the 'manual' trigger; Diagram 8-6, FEMDM. The 'stop' trigger, which forces the CPU to the stop loop, is set by:

1. Depressing STOP (S30).
2. Setting the 'instruction step' trigger in conjunction with an 'I-Fetch reset' micro-order or a 'reset interrupt triggers' micro-order.
3. Detecting an address-compare condition when the ADDRESS COMPARE STOP switch (N17) is in the stop position.
4. Being in scan operation.

Note: When the Multisystem feature is installed, the 'stop' trigger is also set at the initiation of an external start operation.

At end op, the 'stop' trigger's being set indicates an exceptional condition. A branch is forced to a count delay microprogram, provided all pending interruptions and I/O operations are completed. (All interruptions not masked off and all I/O operations are completed before entering the count delay and stop loop microprograms.) The 'stop' trigger forces address 026 (hex) into ROSAR to enter the count delay microprogram, which allows time to recognize that a pushbutton has been depressed before the stop loop is entered.

After the count delay microprogram is executed, the 'manual' trigger is set by the 'set stop loop trigger' micro-order (Diagram 8-6, FEMDM), and the stop loop is entered. The 'manual' trigger allows manual operations only when the CPU is in the Stopped state; however, the SYSTEM RESET (P30), CHECK RESET (P35), and LOAD (S51) pushbuttons operate in all modes. The 'manual' trigger also makes the RATE switch (Q25) inoperative. Whenever the CPU is in the stop loop or in one of the microprogram routines entered from the stop loop, the CPU operates at normal machine speed regardless of the position of the RATE switch.

The 'stop' trigger is not set during a system reset or power-on reset; therefore, the stop loop is entered directly (Diagram 8-4).

The stop loop continually samples six pushbuttons: STORE (R32), DISPLAY (R35), SET IC (R30), START (S25), ROS TRANSFER (Q35), and PSW RESTART (Q30). When the stop loop senses that one of these pushbuttons has been depressed, a branch is made to a new microprogram to perform the pushbutton function (Diagram 8-5). The microprograms for the six pushbuttons are shown in Diagram 8-4. The microprogram just executed either branches to the count delay microprogram and re-enters the stop loop or continues other preselected operations.

Note: When the Multisystem feature is installed, the stop loop also senses the state of the 'external start' latch, branching to the PSW-restart microprogram when it is set.

Power-On Reset

Depressing POWER ON (N43) initiates the power-on sequence. After power is applied to the system, a system-reset occurs. The 'power-on reset' signal resets the system to an initial state and generates the 'switch SS', 'delayed SS', and 'short SS' signals. A ROS address of 00B (hex) is forced to execute the power-on reset microprogram to clear all LS locations (Diagram 8-7); main storage locations remain unchanged. The stop loop microprogram is then entered.

The 'system reset' signal performs the same reset functions as the SYSTEM RESET pushbutton (P30) (Diagram 8-7). After the system is reset, the ROS microprogram is entered by setting ROSAR(8,10,11) (address 00B, hex). LS is reset, and the stop loop is then entered.

SYSTEM RESET Pushbutton

Activating the SYSTEM RESET pushbutton (P30) resets all main storage check triggers, on-line channels and control units, and CPU controls and check triggers to their initial state. Data-flow registers remain unchanged. A system reset does not affect equipment in off-line channel operation. The SYSTEM RESET pushbutton is active in all modes of operation. When it is depressed, the system is reset to an initial state and ROSAR (10,11) is set (address 003). The CPU enters the stop loop. Because a system reset may occur in the middle of an operation, the contents of the PSW and of result registers or storage locations are unpredictable. If the CPU is in the pulse mode, the manual controls are not reset when a system reset is initiated (Diagram 8-7).

The lines used during a system reset are also activated for a power-on reset and IPL reset. Depressing LOAD (S51) initiates a system reset before entering the IPL microprogram. This system reset, however, is not propagated to the channels. During the IPL, the channel latches the 'initial program load' signal from the CPU, and thus initiates a channel reset and a control unit and I/O device reset.

A system reset micro-order is provided to reset the CPU when in the scan mode. The micro-order resets all check conditions but does not reset the channels or storage units and does not force a ROS address.

CHECK RESET Pushbutton

The CHECK RESET pushbutton (P35) serves to reset all CPU check triggers and latches to the nonerror state (Diagram 8-7). All logic check indicators on the system control panel are turned off. The CHECK RESET pushbutton is active in all modes of operation; depressing it does not change the mode of operation. The operation continues as though no error conditions existed; the results, however, may be unpredictable.

STOP Pushbutton

The STOP pushbutton (S30) terminates machine operations while retaining the machine environment. The CPU proceeds to the end of the instruction being executed at the time STOP is depressed; all I/O operations in process and all pending interruptions not masked off are completed before entering the stop loop. The operator may continue normal program operation by depressing START (S25), or he may execute certain manual operations (e.g., instruction-step operation).

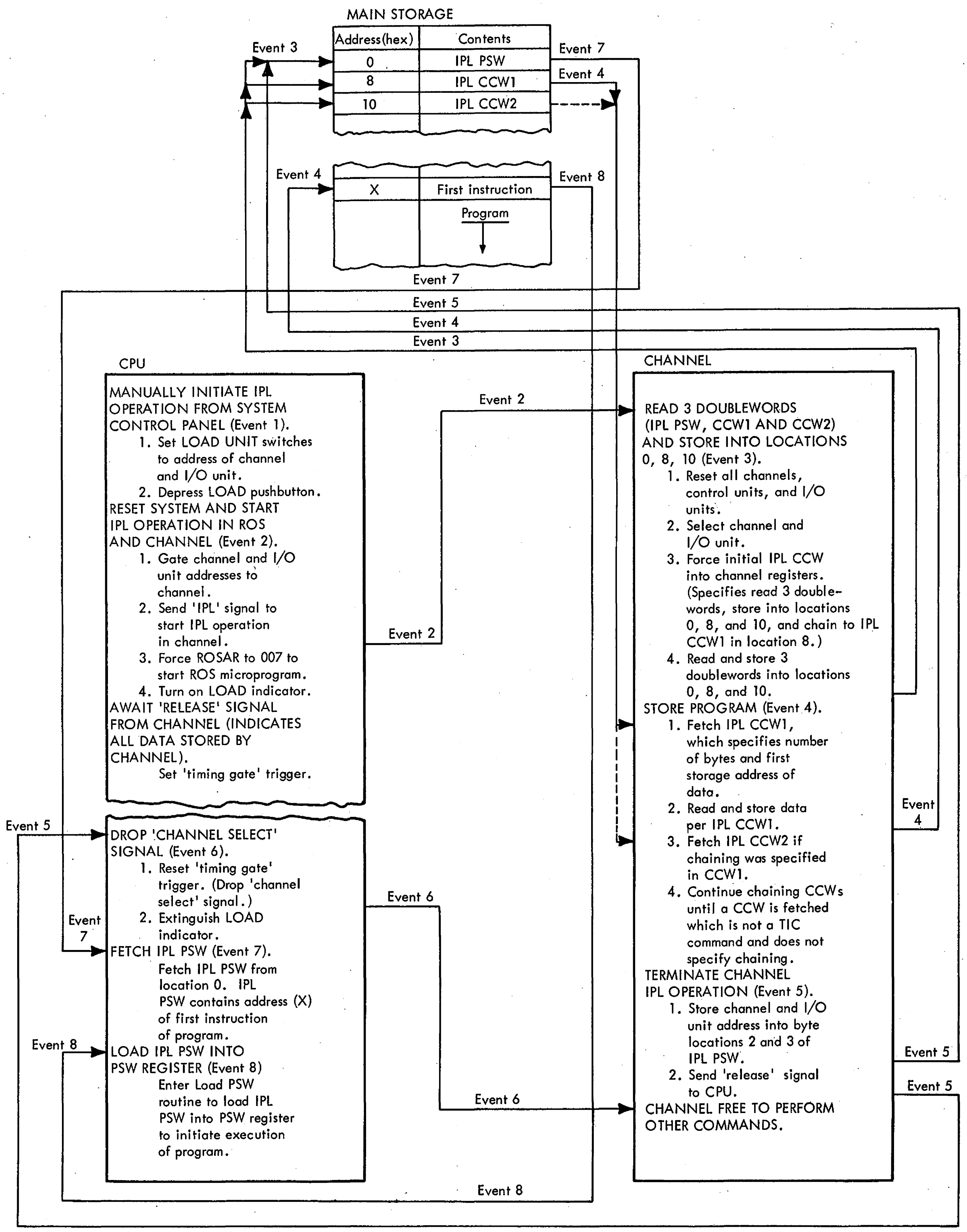
If the ROS microprogram is not performing an end-op micro-order, depressing STOP sets the 'stop' trigger (Diagram 8-6). At end-op, an exceptional condition is signaled to I-Fetch. The 'EXCEP' micro-order and the 'stop' trigger set ROSAR(6,9,10) (address 026 hex); the count-delay microprogram is executed and the stop loop is entered.

LOAD Pushbutton (IPL)

- LOAD pushbutton is active in all modes of operation.
- Loads program from preselected I/O unit per LOAD UNIT switches.
- 24 bytes of data are read automatically into main storage locations 0, 8, 10 (hex).
- CPU initiates channel operation, which loads program and passes control back to CPU for execution of program.
- Channel does not release CPU until program is loaded and channel and I/O unit addresses have been stored into bits 21–31 of IPL PSW in main storage location 0.
- CPU starts execution of program by loading doubleword from main storage location 0 as current PSW.

The purpose of the initial program load (IPL) operation is to allow the channel to load a new program, or a series of programs, into main storage from a preselected I/O unit and to initiate the execution of the program by the CPU. The basic function of the IPL operation is to read in from a specified I/O unit and to store into main storage 24 bytes of data (three doublewords) which will control the operation while the desired programs are being loaded into main storage. (The IPL procedure is common to all models of the IBM System/360.) The first doubleword, starting at address 0, is the IPL PSW, which is used by the CPU to initiate the program once it has been loaded (Figure 6-1). The next two doublewords, starting at addresses 8 and 10 (hex), are the IPL CCW1 and IPL CCW2, respectively, which control the channel during the IPL operation.

The IPL operation is initiated manually from the system control panel. The input channel and I/O unit are selected by the LOAD UNIT switches (Q, 43–51). The left switch, numbered 0 through 7, selects 1 of 7 allowable channels (channels 0–6). The other two switches, which are numbered 0 through F (hex), are set to the unit address of the selected I/O unit. When the selected I/O unit is ready, depression of LOAD (S51)



MANUALLY INITIATE IPL OPERATION FROM SYSTEM CONTROL PANEL (Event 1).

1. Set LOAD UNIT switches to address of channel and I/O unit.
2. Depress LOAD pushbutton.

RESET SYSTEM AND START IPL OPERATION IN ROS AND CHANNEL (Event 2).

1. Gate channel and I/O unit addresses to channel.
2. Send 'IPL' signal to start IPL operation in channel.
3. Force ROSAR to 007 to start ROS microprogram.
4. Turn on LOAD indicator.

AWAIT 'RELEASE' SIGNAL FROM CHANNEL (INDICATES ALL DATA STORED BY CHANNEL).

Set 'timing gate' trigger.

DROP 'CHANNEL SELECT' SIGNAL (Event 6).

1. Reset 'timing gate' trigger. (Drop 'channel select' signal.)
2. Extinguish LOAD indicator.

FETCH IPL PSW (Event 7).

Fetch IPL PSW from location 0. IPL PSW contains address (X) of first instruction of program.

LOAD IPL PSW INTO PSW REGISTER (Event 8)

Enter Load PSW routine to load IPL PSW into PSW register to initiate execution of program.

READ 3 DOUBLEWORDS (IPL PSW, CCW1 AND CCW2) AND STORE INTO LOCATIONS 0, 8, 10 (Event 3).

1. Reset all channels, control units, and I/O units.
2. Select channel and I/O unit.
3. Force initial IPL CCW into channel registers. (Specifies read 3 doublewords, store into locations 0, 8, and 10, and chain to IPL CCW1 in location 8.)
4. Read and store 3 doublewords into locations 0, 8, and 10.

STORE PROGRAM (Event 4).

1. Fetch IPL CCW1, which specifies number of bytes and first storage address of data.
2. Read and store data per IPL CCW1.
3. Fetch IPL CCW2 if chaining was specified in CCW1.
4. Continue chaining CCWs until a CCW is fetched which is not a TIC command and does not specify chaining.

TERMINATE CHANNEL IPL OPERATION (Event 5).

1. Store channel and I/O unit address into byte locations 2 and 3 of IPL PSW.
2. Send 'release' signal to CPU.

CHANNEL FREE TO PERFORM OTHER COMMANDS.

Figure 6-1. Normal IPL Operation

initiates the IPL operation. The LOAD pushbutton causes a system reset, turns on the LOAD indicator (S49), turns off the MANUAL indicator (S46) (if on), selects the channel, sends an eight-bit (+ P) unit address to the channel, sends an 'IPL' signal to the channel, and causes the CPU to enter the IPL microprogram. Diagram 8-8, FEMDM, is a flow chart of the IPL operation.

The 3-usec IPL 'system reset' signal resets the CPU by suspending all instruction processing, interruptions, and timer updating. The contents of LS remain unchanged. The IPL 'system reset' signal is not propagated to the channel. Instead, the channel receives the 'IPL' signal from the CPU, thus initiating a channel reset and dropping the 'operational out' signal to all on-line nonshared control units for 6 usec. Thus, all control units and their I/O units are reset.

After the system reset, ROSAR(9–11) is set, thus forcing a ROS address of 007 (Diagram 8-9, FEMDM). ROSAR(9,10) is set directly from the LOAD pushbutton. ROSAR(11) is set by the 'force address' trigger, which is set by the LOAD pushbutton. The 'timing gate' trigger is then set and the channel is tested for a 'release' signal. The LOAD pushbutton also sends an 'IPL' signal to the channel. The 'IPL status' trigger and the 'hold maintenance console IPL' latch gate the channel address and the I/O unit address to the channel. The selected channel gates the unit address to its unit address register and selects the specified I/O unit. The channel forces a hardware-generated initial CCW for a Read command, with a byte count of 24, and sets a 'chain command' bit in its flag register. The 'CCW valid' trigger is set, a command address of 8 (the address of IPL CCW1 in main storage) is set, and the data address is set to 0. This initial CCW indicates that 24 bytes of data are to be read into locations 0, 8, and 10 (hex) of main storage and that chaining will take place to IPL CCW1.

After the I/O unit has been selected, the IPL-forced CCW controls the transfer of the first three doublewords of data into main storage locations 0, 8, and 10 (hex); it then initiates a chain operation to IPL CCW1 at location 8. The data just stored governs further operation. IPL CCW1 may specify any valid operation for the selected I/O unit, and may be chained to IPL CCW2. Normally, IPL CCW1 initiates the read-in of another series of CCW's, then chains to IPL CCW2, which is a 'transfer in channel' (TIC) command, placing the channel under control of the new CCW's just read in under control of IPL CCW1. When the load-routine command chain is completed, the entire program has been loaded from the I/O unit and stored into main storage; the CPU can thus start executing the program just loaded.

The CPU, which is awaiting a 'release' signal from the channel, requires the identity of the channel and I/O unit involved in the IPL operation. When the first three doublewords of data were stored, the IPL PSW was stored into location 0. Because the interruption code of the IPL PSW (bits 21–31) has no other function, the channel stores the channel and I/O unit address into the IPL PSW interruption code, and sets bits 16–20 to all 0's as part of the IPL termination. (Bits 40–63 contain the starting instruction address of the program.) A 'release' signal is then sent to the CPU as the last function of the channel IPL routine.

During the channel operation, the 'select channel' signal is active. Normally, the channel time-out circuits detect the absence of a 'release' signal to the CPU after about 100 ms, and the channel and I/O interface are reset. During an IPL operation, however, the time-out circuits are deactivated because the channel must hold the CPU long enough to load the complete program; channel detection of interruptions is also blocked. (For a detailed analysis of the channel portion of an IPL operation, refer to the applicable channel FETOM.)

When the 'release' signal is sent to the CPU, the 'timing gate' trigger is reset and the 'channel select' signal is deactivated. The CPU starts an "IPL load PSW" routine, which loads the CPU PSW-register with the doubleword stored at main storage location 0 (IPL PSW). The CPU and the channel now resume normal operation. The CPU initiates the program just loaded; the instruction address in the current PSW is the address of the first instruction of the new program.

When the IPL operation was started by depressing LOAD, the LOAD indicator was turned on. When the IPL operation is successfully completed, the LOAD indicator is turned off. If a check condition or a selection failure occurs during the IPL operation, or if the I/O unit specified is not ready or available, the CPU continues to await a 'release' signal from the channel and the LOAD indicator remains on.

If the program to be loaded is a ROS test, the TEST MODE, ROS/PROC/FLT switch (N3) is set to ROS, the CPU CHECK switch (N19) is set to DSBL (disable), and SYSTEM RESET (P30) is depressed before LOAD is depressed. This procedure causes the ROS test to be loaded similar to a normal IPL, except that logic hardware circuits, rather than the normal ROS microprogram, control the CPU operation. A ROS test sets the 'maintenance mode stop clock' trigger, which stops the CPU clock and prevents a ROS cycle (Diagram 8-8). Thus, the ROS test is loaded even though ROS may be failing. The operating procedure for the ROS test is given in ALD M8005.

If an FLT is to be performed, the TEST MODE, ROS/PROC/FLT switch is set to FLT, the CPU CHECK switch is set to DSBL, and SYSTEM RESET is depressed before LOAD is depressed. This procedure causes the FLT to be loaded similar to a normal IPL, except that the setup operation for execution of the FLT is different (Diagram 8-8). The operating procedure for the FLT is given in ALD M8005.

DATA Switches

The 64 DATA switches (K, L, 14–49) (Diagram 8-1, Sheet 2 of 2) allow the operator to enter data manually into the system. They are alternately colored black and white in groups of four to facilitate entering hex data into the CPU. Each switch is a two-position toggle switch, with the up position equalling a 0 and the down position equalling a 1. When using the DATA switches, the selected microprogram first places 1's into ST. Then if a DATA switch equals a 1, the corresponding bit in ST is unchanged (remains set); if a DATA switch equals a 0, however, the corresponding bit is reset to 0.

The DATA switches are used during the following manual operations (DATA switch gating is shown in Figure 6-2):

1. Store.
2. Storage ripple store.
3. Repeat instruction.
4. Pulse mode count function (switches 53–63).

The DATA switches are used during the following programmed operations:

1. Diagnose, with MCW(8–19) = 5B7.
2. FLT test number search after a stop.

When data is entered into the CPU, correct parity is automatically generated. If the switches are altered during an operation, such as repeat instruction or storage ripple, an error will probably occur.

ADDRESS Switches

The 24 ADDRESS switches (M, 14–40) allow the operator to manually select any address in ROS, LS, or main storage. They are alternately colored black and white in groups of four to facilitate entering hex addresses into the CPU. Each is a two-position toggle switch, with the up position equalling a 0 and the down position equalling a 1. During manual operations, the selected manual microprogram places 1's into D. If an ADDRESS switch equals a 1, the corresponding bit in D is unchanged. If an ADDRESS switch equals a 0, however, the corresponding bit in D is reset.

The ADDRESS switches are used during the following manual operations (ADDRESS switch gating to D, ROSAR, and comparison circuits is shown in Figure 6-3):

1. Store.
2. Display.
3. Set IC.
4. ROS transfer.
5. ROS repeat address.
6. Main storage address-compare stop or sync.
7. ROS address-compare sync.

To address main storage or LS, the ADDRESS switches are used with the STORAGE SELECT switch (N15). ADDRESS switches 0–20 select a doubleword in main storage; ADDRESS switches 2–20 may be used in conjunction with the ADDRESS COMPARE STOP switch (N17) when selecting an address for an address-compare stop or sync. ADDRESS switches 19–23 select an LS address.

ADDRESS switches 0–11 select a ROS address to be used to obtain a ROS address sync (a sync pulse is generated when the ROS address and the ADDRESS switches agree), or contain a ROS address for use with the ROS TRANSFER pushbutton (Q35).

When an address is entered, correct parity is automatically generated when gated to the CPU.

ADDRESS COMPARE STOP Switch

The ADDRESS COMPARE STOP switch (N17) enables the operator to stop at a predetermined address. To do so, he enters the address into the ADDRESS switches and places the ADDRESS COMPARE STOP switch in the stop (down) position. When ADDRESS switches 2–20 match the address sent to the BCU, the 'address compare' trigger is set (address comparison is performed on doubleword boundaries). The CPU will stop at the end of the instruction in progress. The 'stop' trigger is set, and the count delay microprogram is entered; entry to the stop loop is identical with depressing STOP. The TEST indicator (S48) is lit whenever this switch is in the down position.

In the center or normal position, a sync pulse is generated whenever SAB(2–20) matches ADDRESS switches 2–20 (Figure 6-3).

STORAGE SELECT Switch

The STORAGE SELECT switch (N15) is a three-position toggle switch that selects LS or main storage. It is used in conjunction with the ADDRESS switches and the STORE (R32) or DISPLAY (R35) pushbutton. The positions and corresponding functions are:

1. MAIN position – Selects the doubleword main storage location specified by ADDRESS switches 0–20 for both storing and displaying data.
2. MAIN BYTE position – For storing, selects the doubleword location (per ADDRESS switches 0–20) and byte (per ADDRESS switches 21–23) within the doubleword. For displaying, selects the doubleword in main storage to be displayed (same as MAIN).

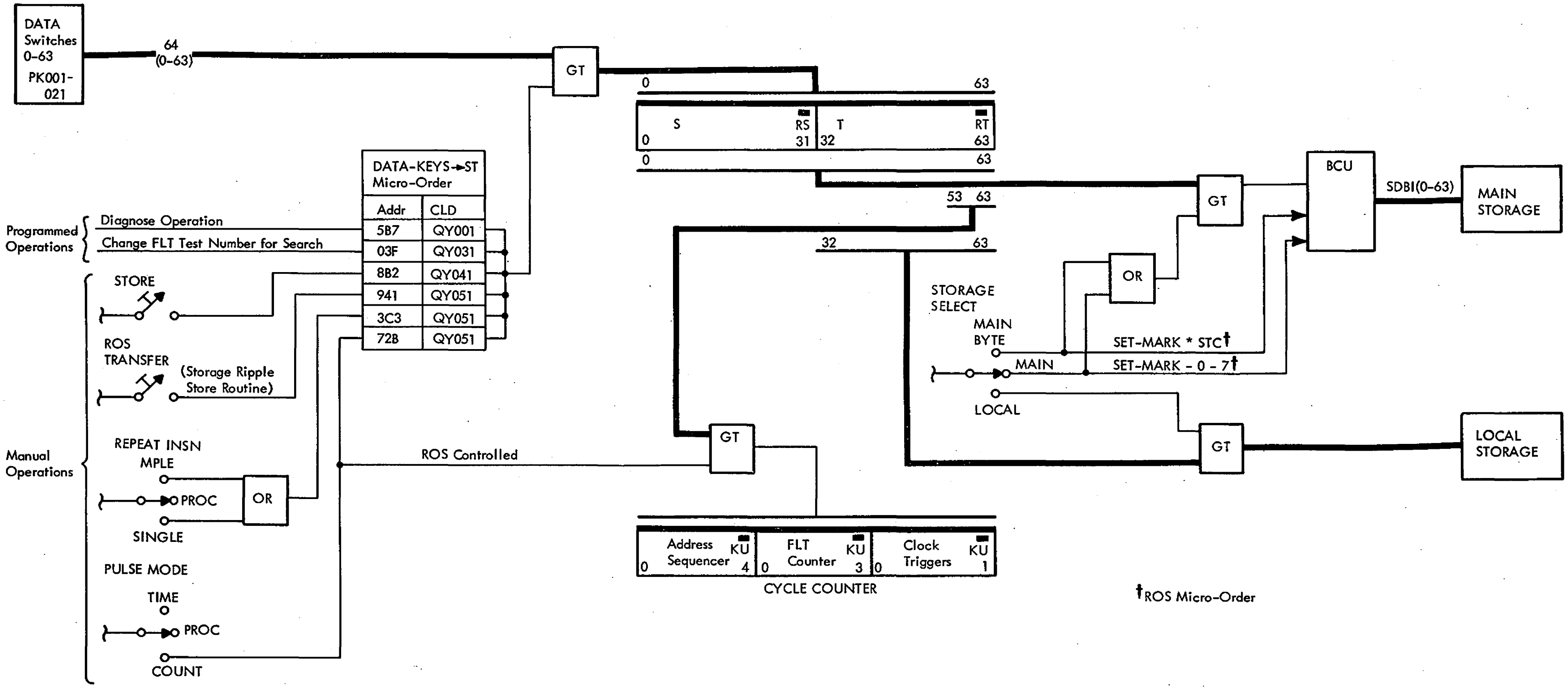
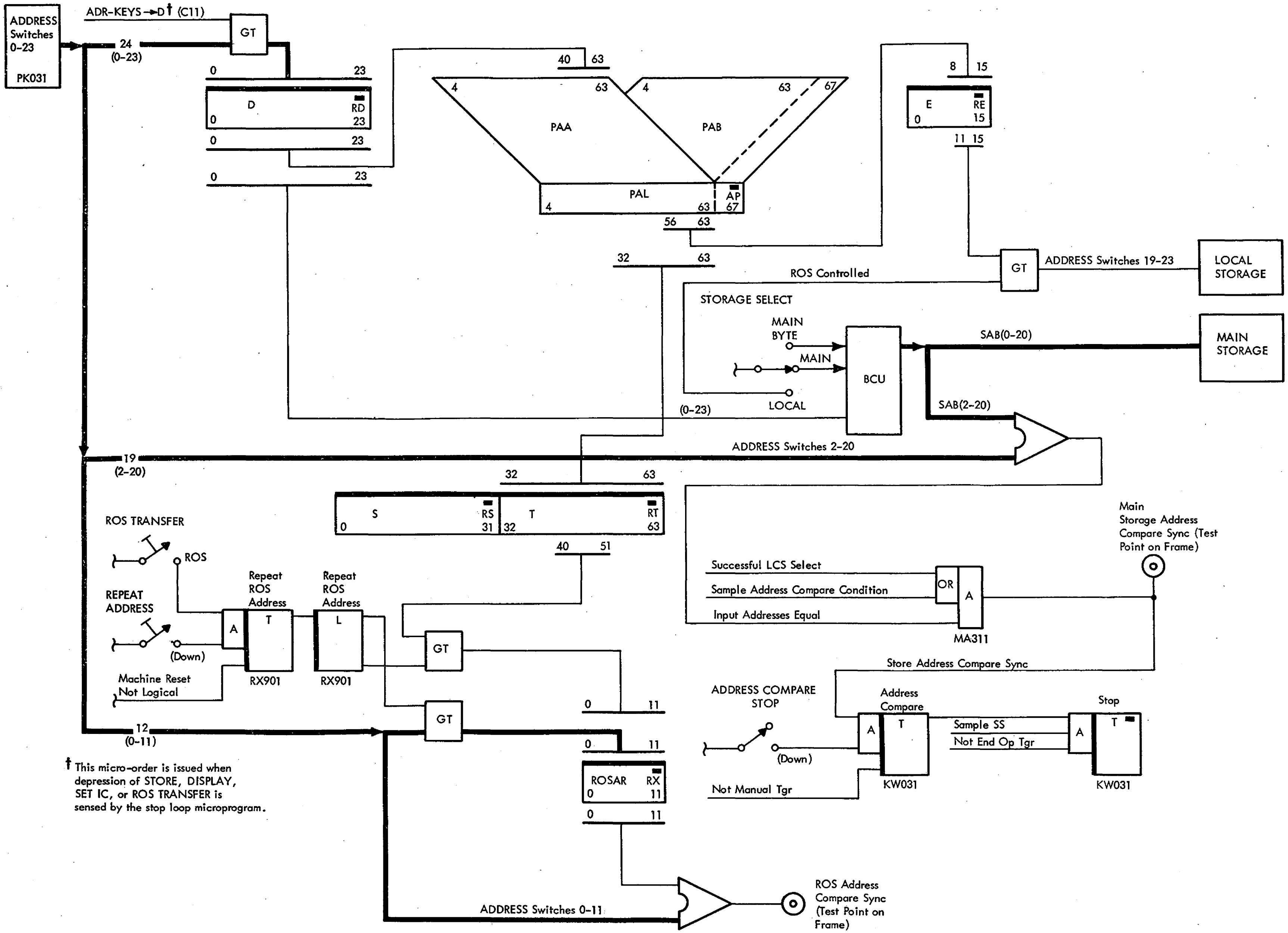


Figure 6-2. Data Switch Gating



† This micro-order is issued when depression of STORE, DISPLAY, SET IC, or ROS TRANSFER is sensed by the stop loop microprogram.

Figure 6-3. Address Switch Gating

3. LOCAL position – Selects an LS location (per ADDRESS switches 19–23) for both storing and displaying data.

The STORAGE SELECT switch conditions hardware so that a ROS branch may occur by setting or inhibiting ROSAR(11). The 'SB-PB' and 'LS-PB' micro-orders allow the stop loop microprogram to perform the storage selection specified by the STORAGE SELECT switch. STORAGE SELECT switch gating is shown in Diagram 8-10, FEMDM.

The use of the STORAGE SELECT switch is discussed in the following paragraphs whenever it is involved in a specific manual operation.

DEFEAT INTERLEAVING Switch

The DEFEAT INTERLEAVING switch (N10) permits the CE to choose which halves of main storage are the high-order and low-order portions for maintenance use. It is a three-position switch that performs the following functions (Diagram 8-11, FEMDM):

1. NO REV (up) position – Interleaving of main storage addresses is disabled, and locations in each storage unit are addressed consecutively.
2. REV (down) position – Interleaving of main storage addresses is disabled, and locations in each storage unit are addressed consecutively. Also, the high and low halves of each storage unit are reversed. No reversal occurs on Model G65.
3. PROC (center) position – Normal position of the switch. Addresses are interleaved in the normal manner with no reversal of storage addresses, unless changed by the Diagnose instruction.

By using this switch, the CE may address the first 16,383 doublewords consecutively rather than in an interleaved manner. When used for this purpose and the switch is in NO REV, the first 16,383 doubleword addresses to be selected will be even and the next 16,383 doubleword addresses will be odd. In REV, the first 16,383 doubleword addresses will be odd and the next 16,383 doubleword addresses will be even.

The signals generated by this switch are sent to the BCU and to main storage. When in NO REV or REV, the TEST indicator (S48) is lit.

STORAGE INDICATE Switch

The STORAGE INDICATE switch (N5) selects one of two pairs of storage units so that the contents of their storage address registers may be displayed on the roller switches. This switch has two positions:

1. STOR 1–4 (normal position). Storage address register contents for storage units 1 and 2 are displayed on position 5 of roller switches 1, 2, 5, and 6.
2. STOR 5–8. The operation is the same as for the normal position for all Model 65 configurations except Models IH and J. For these models, Storage address register contents for storage units 3 and 4 are displayed on position 5 of roller switches 1, 2, 5, and 6.

STOP ON STORAGE CHECK Switch

The STOP ON STORAGE CHECK switch (N11) inhibits further storage accesses in the main storage unit in which a storage error occurred so that the indicators will not be changed. The STOR CHK indicator (K52) lights to show that a storage error occurred and requires attention. The roller switches must be examined to determine the error and the address of the failing main storage word. The STOP ON STORAGE CHECK switch provides a storage stop upon encountering a storage check; this machine stop must not be confused with the Stopped state or the stop loop. The STOP ON STORAGE CHECK switch may be used in conjunction with the CPU CHECK switch (N19).

The 'stop on storage check switch' signal is sent to main storage, where the check signals are generated and sent back to the CPU. Storage checks caused by accesses by channels and by other CPU's also cause a stop. The TEST indicator (S48) is lit when this switch is not in the normal position.

SET IC Pushbutton

The SET IC pushbutton (R30) provides a means of entering an address into the current PSW. The pushbutton sets the instruction-address portion (bits 40–63) of the PSW [D(0–23)] to the value specified by the ADDRESS switches. The CPU is reset to the start of an I-Fetch at that address. Four instruction halfwords are loaded into Q per the address in D (contents of the ADDRESS switches); 8 is then added to D, and the sum is placed into the IC.

The first addressed instruction halfword is loaded into R per D(21,22). If D(21,22) = 11, Q is loaded with the next group of four instruction halfwords per the IC, and 8 is added to the IC (Diagram 8-4). If D(21,22) ≠ 11, Q is loaded only once.

After the instruction halfwords are fetched and loaded into Q and R, the count delay microprogram is entered. After the count delay, the stop loop is re-entered. Further manual intervention is required to start program execution. [Normally, START (S25) is depressed.] Note that the CPU must be in the Stopped state (stop loop) for this pushbutton to function. Note, too, that when the CPU is in the Stopped state, the instruction address is contained in D. After depressing the SET IC pushbutton, the new address contained in the IC is one or two doublewords more than the address contained in the ADDRESS switches.

RATE Switch

- Controls rate of instruction execution.
- CPU must be in stop loop before switch is activated.

The RATE switch (Q25) selects the rate at which instructions are to be executed. This rotary switch has four positions: PROCESS, INSN (instruction) STEP, SINGLE CYCLE, and SINGLE CYCLE STORAGE INHIBIT.

Two latches and two triggers control the RATE switch operation: 'instruction step' and 'single cycle' latches, and 'pass pulse' and 'block' triggers (Diagram 8-12, FEMDM).

The 'instruction step' latch performs two functions: (1) it allows setting the 'stop' trigger so that only one instruction is executed with each depression of START (S25); (2) it disables the stepping of the interval timer.

The 'single cycle' latch allows single-cycle operation. One machine cycle is allowed with each depression of START, unless the CPU requests additional machine cycles. The interval timer is disabled by the 'single cycle' latch.

When the 'pass pulse' trigger is set, CPU machine cycles are allowed. This trigger blocks the CPU machine cycles when in the single-cycle mode (RATE switch in SINGLE CYCLE or SINGLE CYCLE STORAGE INHIBIT position).

The 'block' trigger is used in single-cycle operation to allow one clock signal to be gated to the CPU each time START is depressed. The clock signal is blocked by resetting the 'pass pulse' trigger.

The TEST indicator is lit when the RATE switch is in any position other then PROCESS.

PROCESS Position. When the RATE switch is in PROCESS, the system operates at the normal clock speed of 200 ns; this is the position for normal program execution.

INSN STEP Position. The INSN STEP position allows the execution of one machine instruction for each depression of START; any instruction may be executed. All I/O operations and interruptions (not masked off) are executed after the instruction is completed. The CPU then enters the stop loop at the same point as the STOP pushbutton. The TEST indicator (S48) is lit when the RATE switch is in INSN STEP.

Instruction-step operation is shown in Diagram 8-13, FEMDM. The CPU must be in the stop loop before entering or leaving the instruction-step mode; the interval timer is disabled during the instruction-step mode.

When the RATE switch is moved to INSN STEP, the 'instruction step' latch is set if the CPU is in the stop loop or if the 'pass pulse' trigger is reset (Diagram 8-12). The interval timer is disabled after the 'instruction step' latch is set. No further action occurs until START (S25) is depressed, at which time one instruction is executed. At end-op, the 'stop' trigger is set by an 'I-Fetch reset' micro-order or a 'reset interrupt triggers' micro-order, thus forcing the CPU to enter the stop loop. The CPU remains in the stop loop until further action is taken by the operator.

SINGLE CYCLE Position. The SINGLE CYCLE position allows the CPU to advance one machine cycle (200 ns) each time START (S25) is depressed (Diagram 8-14, FEMDM). The CPU must be in the stop loop before entering or leaving the single-cycle mode and remains in the stop loop until START is depressed. The CPU begins executing instructions one machine cycle at a time for each depression of START. In Diagram 8-14, it is assumed that no CPU requests are generated; if an asynchronous device is used or if a storage request is given, however, more than one machine cycle is required. The single-cycle mode continues through all CPU functions of the instruction to the point of initiation of the asynchronous operation. The asynchronous operation begins on the next depression of START and runs to the completion point in a normal manner.

If the asynchronous device initiates an interruption request during single-cycle operation, the interruption is broken into single machine cycles. More than one depression of START is therefore required. The CPU runs at normal machine speed in the stop loop.

When the RATE switch is in SINGLE CYCLE, the 'single cycle' latch is set (Diagram 8-12). The CPU remains in the stop loop until START is depressed, at which time the CPU advances one machine cycle. The 'block' trigger is set as shown in Diagram 8-12. If there is no 'BCU hold on clock' signal, the 'pass pulse' trigger is reset after the 'block' trigger is set, thus inhibiting CPU clock signal distribution. START must be depressed for each CPU machine cycle advance. The TEST indicator (S48) is lit when the RATE switch is in SINGLE CYCLE.

SINGLE CYCLE STORAGE INHIBIT Position. The SINGLE CYCLE STORAGE INHIBIT position allows the CPU to advance one machine cycle (200 ns) each time START is depressed. All CPU requests are ignored, and asynchronous operations are suppressed.

Except for the inhibit signals sent to the BCU and inhibiting the 'stop 1' and 'stop 2' triggers from affecting the CPU clock, the single-cycle-storage-inhibit function is the same as the single-cycle function, Diagrams 8-12 and 8-14.

REPEAT INSN Switch

The REPEAT INSN (instruction) switch (N23) provides a means of repeating a single instruction or of repeating up to four instruction halfwords. The REPEAT INSN switch has three positions (Diagram 8-15, FEMDM): PROC, normal CPU operation; SINGLE; and MPLE (multiple).

A trigger-latch combination controls the repeat-instruction functions: 'repeat instruction adjust' trigger and 'repeat instruction initialization' latch.

The 'repeat instruction adjust' trigger forces a branch to the manual control repeat exceptional condition microprogram (ROS address 028, hex) at end-op of the start microprogram and sets the 'repeat instruction initialization' latch. STAT G is set to block re-entering the repeat-instruction microprogram at microprogram end-op. The trigger is set when the CPU is in the Stopped state and the REPEAT INSN switch is in SINGLE or MPLE; it is reset when the CPU is in the Stopped state and the REPEAT INSN switch is placed in PROC.

The 'repeat instruction initialization' latch blocks transfer to Q and stepping of the interval timer when in repeat-single-instruction mode. The latch is reset when in the stop loop and the RATE switch (Q25) is placed in PROCESS. The CPU must be in the Stopped state before entering or leaving the repeat-instruction mode. See Diagram 8-16, FEMDM, for the repeat-instruction operations.

The TEST indicator (S48) is lit when the REPEAT INSN switch is in any position other then PROC.

Repeat Single Instruction. When the REPEAT INSN switch is in SINGLE, one instruction is continuously executed. The instruction to be repeated is entered into the DATA switches, beginning with byte 0. If the CPU is in the stop loop, the 'repeat instruction adjust' trigger is set when the REPEAT INSN switch is placed in either MPLE or SINGLE. To begin the instruction, depress START (S25).

In the repeat-single-instruction mode, the repeat-instruction microprogram is executed to set up initial conditions before entering I-Fetch of the instruction to be executed (Diagram 8-16). The objectives of the microprogram are to load the contents of the DATA switches into Q, set IC(21,22) to 00, inhibit updating of IC(20) or above, gate the first instruction halfword from Q to R, set STAT G, and set the 'repeat instruction initialization' latch.

When the CPU is in the stop loop and the REPEAT INSN switch is in SINGLE, the 'repeat instruction adjust' trigger is set. The START pushbutton initiates the repeat-instruction function. A ROS address of 028 (hex) is forced into ROSAR to enter the repeat-instruction microprogram (Diagram 8-15). During this microprogram, STAT G is set which, in turn, sets the 'repeat instruction initialization' latch. This action prevents the loading of new instructions into Q. Note that STAT G is reset by the 'reset' micro-order that performs all the resets necessary before the next I-Fetch. The instruction that was loaded into Q from the DATA switches is executed.

Because the 'repeat instruction adjust' trigger was not reset during the initial setup routine, ROS address 028 (hex) is forced at end-op of the instruction to enter the repeat-instruction microprogram. Re-entering the repeat-instruction microprogram on each instruction resets IC(21,22) to 00, thus causing the first instruction to be repeated. Because the 'repeat instruction initialization' latch is set, Q is not loaded after the initial loading. The instruction is continuously executed until the CPU is manually stopped. The TEST indicator (S48) is lit while the REPEAT INSN switch is in SINGLE.

Repeat Multiple Instructions. When the REPEAT INSN switch is placed in MPLE, the four instruction halfwords loaded into Q are continuously executed per IC(21,22). The 'repeat instruction initialization' latch inhibits data from being transferred from the SDBO to Q. Once instruction execution begins, the repeat-instruction microprogram is not entered because the 'repeat instruction adjust' trigger is reset (Diagram 8-16).

The repeat-multiple-instructions function is similar in operation to the repeat-single-instruction function except for the following:

1. The interval timer is allowed to step.
2. The 'repeat instruction adjust' trigger is reset.
3. Interruptions are executed.

Resetting the 'repeat instruction adjust' trigger prevents re-entry to the repeat-instruction microprogram. Four instruction halfwords are continuously executed per IC(21,22), until the CPU is manually stopped. The TEST indicator (S48) is lit when the REPEAT INSN switch is in MPLE.

STORE Pushbutton

- Allows storing data into main storage or LS from DATA switches per STORAGE SELECT switch and ADDRESS switches.

The STORE pushbutton (R32) provides a means of storing information into any address of LS or main storage. The CPU must be in the Stopped state for this pushbutton to function.

The contents of the DATA switches are placed into the location specified by the ADDRESS switches and the STORAGE SELECT switch (N15) (Diagram 8-4). If the STORAGE SELECT switch is in LOCAL, the five low-order ADDRESS switches (19–23) specify the LS location into which the contents (32 bits plus 4 parity bits) of the right-half of the DATA switches (32–63) are to be stored. ADDRESS switch 19 in the 0 position permits storing into the general-purpose registers and in the 1 (down) position permits storing into the floating-point registers. ADDRESS switches 19 and 20, when set to 1's, address the working register.

If the STORAGE SELECT switch is in MAIN, the contents (64 data bits plus 8 parity bits) of the DATA switches are stored into main storage on a doubleword boundary per ADDRESS switches 0–20.

If the STORAGE SELECT switch is in MAIN BYTE, one byte is stored into main storage per ADDRESS switches 21–23. ADDRESS switches 0–20 specify the doubleword boundary in main storage. The value contained in the ADDRESS switches is placed into D for storing in main storage. The contents of the ADDRESS switches are placed into E (via D) for storing in LS.

For all store operations, the original contents of D, S, and T are destroyed. Correct parity is automatically generated before storing into either main storage or LS. After the data is stored, the microprogram enters the count delay routine and the CPU re-enters the stop loop (Diagram 8-4).

When STORE is depressed, the pushbutton signal is AND'ed with the 'STO-PB' micro-order. When the stop loop microprogram senses that STORE has been depressed, ROSAR(11) is set, causing entry into the store microprogram routine. The stop loop is re-entered after the store operation is executed.

DISPLAY Pushbutton

- Allows displaying of data from main storage into ST and AB or from LS into T per STORAGE SELECT switch and ADDRESS switches.

The DISPLAY pushbutton (R35) serves to display the contents of any location in LS or main storage. The CPU must be in the Stopped state for this pushbutton to function. The address and the storage to be used are determined by the ADDRESS switches and the position of the STORAGE SELECT switch (N15), respectively (Diagram 8-4). Data from main storage (64 data bits plus 8 parity bits) is displayed in ST and AB. (Set roller switches 1 and 2 to position 3 to display contents of ST, and set roller switches 3 and 4 to position 3 to display the contents of AB.) Data from LS (32 data bits plus 4 parity bits) is displayed in T. (Set roller switch 2 to position 3 to display contents of T.)

When DISPLAY is depressed, the pushbutton signal is AND'ed with the 'DIS-PB' micro-order. When the stop loop microprogram senses that DISPLAY has been depressed, ROSAR (11) is set, causing entry into the display microprogram. The original contents of S, T, and D are destroyed. After the selected data has been displayed, the count delay microprogram is executed and the stop loop is re-entered.

START Pushbutton

- Starts CPU processing.
- Initiates selected manual functions.

The START pushbutton (S25) serves to start the CPU in the process, instruction step, single-cycle, or single-cycle storage inhibit mode, depending on the position of the RATE switch (Q25). In Diagram 8-4, it is assumed that the CPU is in the process mode.

When START is depressed, a 'reset stop and manual triggers' micro-order resets the 'stop' and 'manual' triggers. An end-op occurs, and processing begins with I-Fetch of the next instruction. The 'interrupts' latch blocks interruptions at end-op of the start microprogram (Diagram 8-17, FEMDM). Therefore, interruptions are blocked until end-op of the first instruction. This latch does not block interruptions when the CPU is in the Wait state.

When START is depressed, the pushbutton signal is AND'ed with the 'STT-PB' micro-order. When the stop loop microprogram senses that START has been depressed, ROSAR(11) is set, causing entry to the start microprogram. Normal program execution continues.

If START is depressed after a normal halt, instruction processing continues as if no halt had occurred; pending interruptions are taken after execution of the first instruction. If START is depressed after an abnormal stop-loop entry or after a system reset, the results are unpredictable.

ROS TRANSFER Pushbutton

- ROS TRANSFER pushbutton allows ROS microprogram branch to any ROS location.

The ROS TRANSFER pushbutton (Q35) allows entry into a ROS word. Depressing ROS TRANSFER places the contents of the 12 high-order ADDRESS switches into ROSAR (Diagram 8-18, FEMDM). The next microinstruction is taken from ROS and placed into the ROS sense latches. Further action now depends upon the position of the RATE switch.

If the RATE switch (Q25) is in PROCESS, the CPU continues executing ROS words from the entry point. If the RATE switch is in INSN STEP, the CPU continues until an end-op is reached.

If the RATE switch is in SINGLE CYCLE or SINGLE CYCLE STORAGE INHIBIT, the CPU stops with the ROS word specified by the ADDRESS switches contained in the sense latches. (This ROS word may be displayed by means of the appropriate indicators.) Depressing START (S25) advances ROS one cycle, and the contents of the ROS data register may then be displayed. Further depressions of START advances ROS as in the single-cycle mode.

Regardless of the position of the RATE switch, checks may occur as a result of storage data bus transfer to registers and from the registers through the parallel adder. To prevent the CPU from stopping on these checks, place the CPU CHECK switch (N19) in DSBL. (See "CPU CHECK Switch".)

When ROS TRANSFER is depressed, the pushbutton signal is AND'ed with the 'ROS-PB' micro-order. When the stop loop microprogram senses that ROS TRANSFER has been depressed, ROSAR(11) is set, causing entry into the ROS transfer microprogram. Instruction execution continues from the ROS address entered into the 12 high-order ADDRESS switches.

Storage-Ripple Microprogram

The storage-ripple microprogram (Diagram 8-19, FEMDM) is capable of (1) storing data from the DATA switches in all addresses in LS or main storage and putting correct parity into the storage-protect keys, or (2) reading all locations of LS or main storage and displaying the data. Main storage or LS is chosen by means of the STORAGE SELECT switch (N15).

If main storage is selected, the storage-ripple microprogram begins at address 0 and continues until an invalid address is detected, at which point a restart beginning at address 0 occurs. Diagram 8-19 shows the restart on an interruption request that resulted from detecting an invalid address. If LS is selected, the storage-ripple microprogram begins at address 0 and loops through all addresses in LS. Manual intervention (e.g., system reset or IPL) is required to exit from the storage-ripple microprogram.

The storage-ripple microprogram may also be used in troubleshooting by loading all storage locations with a predetermined value and then reading back the data. This microprogram may be used in conjunction with the STOP ON STORAGE CHECK switch (N11).

Storage-Ripple-Store Function. This function allows storing data into all locations of LS or main storage. To store data into LS or main storage, the CPU must be in the Stopped state. Enter the data to be stored into the DATA switches, and position the STORAGE SELECT switch (N15) to select LS or main storage. Enter 800006 (hex) into ADDRESS switches 0–23, and depress ROS TRANSFER. The data previously entered into the DATA switches is continually stored into all locations in the selected storage. Also, correct parity is placed into the storage protect keys on main storage ripple. Incorrect data may be stored if the DATA switches are changed when in the storage-ripple-store routine. To terminate the routine, depress SYSTEM RESET (P30). The ROS microprogram for the storage-ripple-store function is shown in Diagram 8-19.

Because there is no automatic means of clearing main storage and LS, the storage-ripple-store microprogram may be used for this purpose.

Storage-Ripple-Display Function. This function allows reading and displaying data from all locations in LS or main storage. The storage-ripple-display microprogram continuously reads all locations in LS or main storage as determined by the setting of the STORAGE SELECT switch (N15). The CPU must be in the Stopped state before ROS TRANSFER is depressed. To execute the storage-ripple-display routine, enter 800000 (hex) into the ADDRESS switches, select main storage or LS, and depress ROS TRANSFER (Diagram 8-19). If LS is selected by means of the STORAGE SELECT switch, the data is contained in S and PAL(32–63). If main storage is selected, the data is contained in AB and ST. The data may then be displayed using roller switches. Data is checked in PAL for correct parity.

To terminate the routine, depress SYSTEM RESET (P30).

REPEAT ROS ADDRESS Switch

The REPEAT ROS ADDRESS switch (N25) provides a means of continuously reading out a specified ROS address (Diagram 8-18). To do so, enter the address of the ROS word into the ADDRESS switches, activate the REPEAT ROS ADDRESS switch, and depress ROS TRANSFER (Q35). The same ROS address is accessed on each machine cycle. All storage requests are blocked. Changing the ADDRESS switches while in the repeat ROS loop may result in ROS parity checks.

The TEST indicator (S48) is lit when the REPEAT ROS ADDRESS switch is in the down position.

PSW RESTART Pushbutton and Wait State

- Allows loading new PSW from main storage location 0.
- If PSW(14) = 1, enter Wait state; if 0, enter Running state.

The PSW RESTART pushbutton (Q30) allows the operator to load a new PSW from main storage location 0 into the CPU. After the new PSW is fetched, the CPU continues processing if the RATE switch (Q25) is in PROCESS.

The 'stop' and 'manual' triggers are reset at the beginning of the PSW-restart microprogram (Diagram 8-4). The PSW RESTART pushbutton causes entry to the normal load PSW instruction routine, which refills Q, R, and E.

At every end-op, PSW(14) is tested. If PSW(14) = 1, the Wait state is entered (Diagram 8-20, FEMDM). [If PSW(14) = 0, the CPU is placed in the Running state.] The wait-state routine is depicted in Diagram 8-20, B, FEMDM. If the interval timer is to be stepped, the wait loop is re-entered after stepping the timer. If STOP (S30) is depressed, the stop loop is entered. When a restart from the stop loop is executed, the Wait state is re-entered if PSW(14) = 1.

An interruption causes a new PSW to be loaded into the CPU. PSW(14) is again tested, and the Wait state is re-entered if PSW(14) = 1; otherwise, the CPU is placed in the Running state. The usage meter is stopped when in the Wait state or stop loop unless a channel is operating.

When PSW RESTART is depressed, the pushbutton signal is AND'ed with the 'PSW-PB' micro-order. When the stop loop microprogram senses that PSW RESTART has been depressed, ROSAR(11) is set, causing entry to the PSW restart microprogram. A new PSW is loaded, and program execution continues.

DISABLE DIRECT CONTROL Switch

The DISABLE DIRECT CONTROL switch (N27) is added by the Direct Control feature. It is used to manually inhibit (partition) communications between the CPU and the external unit. When the switch is placed in the down position:

1. Write Direct and Read Direct instructions become invalid instructions, causing an operation program interruption after their I-Fetch.
2. Generation of 'direct control write out' and 'direct control read out' timing signals is inhibited.
3. The setting of the 'external interruption' latches for signals 2–7 is inhibited.

Note: When the Multisystem feature is installed, the functions of the DISABLE DIRECT CONTROL switch are altered when operating in the partitioned or multisystem mode. Refer to the discussion of this switch in the Multisystem feature write-up in Chapter 4.

DISABLE INTERVAL TIMER Switch

The DISABLE INTERVAL TIMER switch (N13), when placed in the down position, prevents the interval timer [main storage location 80, decimal (50, hex)] from being advanced (Diagram 8-21, FEMDM). In the center position, the timer is stepped at regular predetermined intervals.

In addition to the switch, the timer is disabled when the Diagnose instruction sets MCW(0) (disable timer bit) or when operating in the:

1. Stop-loop routine.
2. Single-cycle mode.
3. Instruction-step mode.
4. Repeat-instruction mode.
5. Scan mode.

The DISABLE INTERVAL TIMER switch is inactive when the PULSE MODE switch (N21) is in the TIME position and the 'pulse mode initialization' trigger is set. When this switch is in the down position, the TEST indicator (S48) is lit.

INTERRUPT Pushbutton

The INTERRUPT pushbutton (S43) initiates an external interruption by setting the 'console signal' trigger. If PSW(7) = 1, an interruption is taken after the current instruction, and interruptions of higher priority are completed. If PSW(7) = 0, the interruption request remains pending.

During the interruption, PSW(25) is made a 1, indicating that the INTERRUPT pushbutton is the source of the interruption. This pushbutton is effective while power is up for the system.

CPU CHECK Switch

The CPU CHECK switch (N19), a three-position toggle switch, is used to control the system when a machine check is encountered. The machine checks that set the error trigger and the logic controlled by the CPU CHECK switch are shown in Diagram 8-22, FEMDM. The TEST indicator (S48) is lit when this switch is in any position other than PROC.

When the CPU CHECK switch is in PROC and a machine check is detected, and PSW(13) = 1 (machine check mask bit), the CPU is stopped and the machine status is logged out to main storage. A machine check interruption is then executed. If PSW(13) = 0, the check is ignored, except that the error trigger is set, and no logout or interruption occurs.

When CPU CHECK is in STOP and a machine check is detected, the CPU clock is stopped and no logout occurs. The error trigger is set; the type of error may be determined by examining the roller switches. If CHECK RESET (P35) is depressed, operation is resumed, but the results may be unpredictable.

When CPU CHECK is in DSBL and a machine check is detected, the error trigger is set. Logout and interruptions do not occur, and the operation is not terminated. Program execution continues, ignoring machine checks. The error trigger may be reset by depressing CHECK RESET or SYSTEM RESET (P30).

PULSE MODE Switch

The PULSE MODE switch (N21) provides a means of looping through a number of machine cycles, starting at a selected address, or of looping each time the interval timer is advanced. The PULSE MODE switch has three positions: PROC (process), normal CPU operation; TIME; and COUNT.

The CPU must be in the stop loop before entering or leaving pulse mode operation. The PULSE MODE switch is inoperative during repeat-instruction mode.

Two triggers control pulse mode operation: 'pulse mode adjust' trigger and 'pulse mode initialization' trigger (Diagram 8-23, FEMDM). The 'pulse mode adjust' trigger determines when to force an overriding branch to the pulse-mode-initialization microprogram and when to reset the system. The 'pulse mode initialization' trigger is set by depressing START (S25) with the 'pulse mode adjust' trigger set. As a result, pulse mode operation begins.

The TEST indicator (S48) is lit when the PULSE MODE switch is in any position other then PROC.

PROC Position. The PROC position is used during normal program execution.

TIME Position. Highlights:

- Load program into main storage.
- Place starting address of program into main storage byte locations 5–7 by means of ADDRESS switches.
- Enter stop loop.
- Place PULSE MODE in TIME position.
- Depress START.

When the PULSE MODE switch is in the TIME position, instruction execution begins at the address specified in the address portion of the doubleword located in location 0 of main storage. Therefore, the starting address must be loaded (by means of the ADDRESS switches) into bits 40–63 of the doubleword located at location 0 before depressing START (S25). Entering data manually into main storage (from the DATA switches) must be done with the CPU in the stop loop and with STORE (R32) depressed. The RATE switch (Q25) must be in the PROCESS position.

The initial setup conditions are:

1. The program is in main storage.
2. The starting address is in main storage byte locations 5 through 7.
3. The CPU is in the stop loop.
4. The PULSE MODE switch is in the TIME position.

A flowchart of the pulse mode operation is shown in Diagram 8-24, FEMDM. When the CPU is in the stop loop and the PULSE MODE switch is set to TIME, the 'pulse mode adjust' trigger is set (Diagram 8-23). Depressing START sets the 'pulse mode setup' latch. This action fires a 350-ns singleshot that initiates the pulse-mode function. The 'pulse mode initialization' and the 'force address' triggers are set. This action, coupled with the set 'pulse mode adjust' trigger, sets ROSAR(9,11) to force an address of 005 into ROSAR. A system reset is generated. Note that the 'reset manual control' signal is inhibited (Diagram 8-23). The CPU then enters the pulse-mode microprogram. The 'pulse mode adjust' trigger is reset by the pulse-mode microprogram. The loading of the count into the MCW is meaningless. Instructions are then executed until the interval timer is stepped, at which time the 'pulse mode adjust' trigger is set. Because the 'pulse mode initialization' trigger is set, a system reset occurs, an address of 005 is forced into ROSAR, and the pulse-mode microprogram is re-entered. This action results in executing the program from clock step to clock step. Looping through the pulse-mode microprogram and the main storage program continues until manually stopped.

COUNT Position. Highlights:

- Load program into main storage.
- Place starting address of program into main storage byte locations 5–7 by means of ADDRESS switches.
- Enter stop loop.
- Place PULSE MODE in COUNT position.
- Enter number of machine cycles (up to 2047) to be executed into DATA switches 53–63.
- Depress START.

When the PULSE MODE switch is in the COUNT position, instruction execution begins at the address specified in the address portion of the doubleword located in location 0 of main storage, and proceeds through the number of machine cycles entered into DATA switches 53–63. Each time the cycle counter is reduced to 0, a machine reset occurs and the program is re-entered.

Except for PULSE MODE being in the COUNT position and the count entered into DATA switches 53–63, the initial setup conditions are similar to the time-mode routine. As shown in Diagram 8-24, the same microprogram is executed for both COUNT and TIME positions. The program in main storage is entered at end-op of the microprogram. The cycle counter is reduced by 1 on each machine cycle. When the counter equals 0, a machine reset occurs, and the pulse-mode microprogram is again executed. The looping through the microprogram and the storage program continues until manually stopped.

LOG OUT Pushbutton

The LOG OUT pushbutton (S35) serves to log the machine status into fixed locations of main storage. This pushbutton is inactive during normal processing. The logic associated with the LOG OUT pushbutton is shown in Diagram 8-25, FEMDM.

TEST MODE, ROS/PROC/FLT Switch

The TEST MODE, ROS/PROC/FLT switch (N3) serves to take the CPU out of program control for FLT's or ROS tests. It has three positions:

1. ROS: When in this position and LOAD (S51) is depressed, ROS tests are read into storage from the I/O unit selected by the LOAD UNIT switches. Signals from the selected channels cause the tests to be executed in the CPU.
2. PROC – normal position: Normal CPU processing takes place.
3. FLT: When in this position and LOAD is depressed, FLT's are read into storage from the I/O unit selected by the LOAD UNIT switches. Signals from the selected channels cause the tests to be executed in the CPU.

Logic associated with this switch is shown in Diagram 8-26, FEMDM. The interval timer is disabled and the TEST indicator (S48) is lit when this switch is in the ROS or FLT position.

TEST MODE, REPEAT Switch

The TEST MODE, REPEAT switch (N1) is used to continuously repeat the test in main storage or the new test being sought. When this switch is in the Repeat (down) position, depressing START (S25) causes the ROS test or FLT in main storage to be executed repeatedly. If RESTART FLT I/O (S32) is depressed, a new FLT is sought, according to the alternate test number in T, and executed repeatedly.

RESTART FLT I/O Pushbutton

The RESTART FLT I/O pushbutton (S32) provides a means of backspacing tape one record and of starting a read operation during an FLT or a ROS test. When the TEST MODE, ROS/PROC/FLT switch (N3) is in the ROS or FLT position, depressing this pushbutton signals the channel to start with the backspace-CCW at byte location 128, decimal (80, hex). The associated logic is shown in Diagram 8-27, FEMDM.

CE Key Switch and Usage Meters

- Selected usage meter records time CPU or channel is operating.
- 'Meter in' signal indicates channel is operating.
- 'Meter out' signal indicates Process meter is recording.
- 'Not clock out' signal indicates CPU is not processing.

Two meters on the system control panel show the system running time while it is processing customer data (Process meter, R5) and while it is being operated by the CE (CE meter, R13). (The Process meter is also known as the cluster, central processing complex, system, and customer meter.) A key-operated switch (R9) selects the meter to be activated. The normal position of this switch allows power to be applied to the Process meter. The CE, using a key, switches to the CE meter. When in the CE position, the key cannot be removed and the TEST indicator (S48) is lit.

The meter selected by the key switch records time as long as the 'pass pulse' trigger is set if: (1) not in the Wait state and the 'manual' trigger is not set or (2) a 'meter in' signal is received from the channels. The selected meter also runs momentarily when any pushbutton on the system control panel is depressed (except the STOP pushbutton while already in the manual state). The 'pass pulse' trigger is set by a start, load, or reset operation and, when in single-cycle-mode operation, is reset after gating one pulse (refer to Chapter 2, Section 1). The recording of time is inhibited as soon as a stop-on-CPU-check condition is detected, without waiting for the 'manual' trigger to be set during the ensuing stop-loop operation.

The 'meter in' signal, multiplexed from the channels, is received whenever an I/O operation initiated by the CPU still is in progress and requires recording. The CPU sends two signals to the channels. The 'meter out' signal is sent whenever time is being recorded on the Process meter. The 'not clock out' signal is sent when the CPU is in the Wait or manual states or is not recording time due to a CPU machine check. These two signals are used by the channels and their attached units in running their own usage meters and, in turn, generating the 'meter in' signal.

Diagram 8-28, FEMDM, shows the CPU metering circuits. The selected meter runs for a minimum of 400 ms. Relay RR1 on the use meter card is picked by a signal from the CPU logic. 40V ac through RR1 contacts drive the meter.

FREQUENCY ALTERATION Switch

The FREQUENCY ALTERATION switch (N7) is used to increase the CPU clock frequency. When this switch is in the center position, each machine cycle is 200 ns (normal speed). With the CE Key switch in the CE position and the FREQUENCY ALTERATION switch in the down position, the CPU clock cycle is decreased to 195 ns ±½ ns (Diagram 8-28).

Indicators

On panel E of the system control panel, there are six rows of indicators, with 36 indicators (implicitly numbered 0–35) in each row (Diagram 8-1). Associated with each row of indicators is a six-position roller switch. The operator may display the contents of a register or the status of a trigger or latch by placing the proper roller switch in the correct position. A roll chart above each row of indicators shows the information being displayed for each indicator. As the roller position is changed, the roll chart rotates to correspond with the roller position. The roller switch, the positions, and the information displayed for each row of indicators are shown in Diagram 8-2.

A lamp test for the roller switches is provided by positioning the roller switches between detent positions. The sixth position of roller 6 is a lamp test for the panel E and G indicators on the system control panel and on the remote operator's console.

Other indicators on the system control panel indicate machine status, check conditions, and power status, as follows:

1. SYSTEM (S45) – Lights when the Process meter or CE meter is running.
2. MANUAL (S46) – Lights when the CPU is in the Stopped state. The CPU is executing the stop loop microprogram.

3. WAIT (S47) – Lights when the CPU is in the Wait state. The CPU is executing the wait-loop microprogram, if the MANUAL indicator is not lit.
4. TEST (S48) – Lights when a manual control is not in a position for normal processing or a maintenance function is being applied, as follows:
   a. The RATE switch (Q25) is in a position other than PROCESS.
   b. The CPU CHECK switch (N19) is in STOP or DSBL.
   c. The DISABLE INTERVAL TIMER switch (N13) is down.
   d. The ADDRESS COMPARE STOP switch (N17) is down.
   e. The PULSE MODE switch (N21) is in TIME or COUNT.
   f. The TEST MODE, ROS/PROC/FLT switch (N3) is in ROS or FLT.
   g. The STOP ON STORAGE CHECK switch (N11) is down.
   h. The REPEAT INSN switch (N23) is in MPLE or SINGLE.
   i. The DEFEAT INTERLEAVING switch (N9) is in NO REV or REV.
   j. The REPEAT ROS ADDRESS switch (N25) is down.
   k. A channel is in test mode.
   l. The Diagnose instruction is active.
   m. The "lamp test" position (position 6 of roller 6) is selected.
   n. The 'FLT TON test light' signal is active.
   o. The CE Key switch is in CE.
5. LOAD (S49) – Lights when the CPU is in a load state (IPL microprogram). The LOAD indicator is turned off after a successful load.
6. STOR CHK (K51) – Lights on all storage errors associated with the CPU or channel. The roller switches must be examined to determine the specific error.
7. PROC CHK (M51) – Lights on all CPU errors. The roller switches must be examined to determine the specific error.
8. MARGIN indicators:
   a. ACTIVE (A15): Lights when an internal power supply or an attached storage unit or channel is being marginally checked.
   b. LOCATE (A17): Lights when the MARGIN/METER SEL switch is set to the position corresponding to the margined power supply, attached storage unit, or channel.
9. Power Check indicators – Indicators on the system control panel show an incomplete power-up status in the CPU or in the attached storage units and channels. The CPU and the units are individually indicated, as is the system status. There are nine power check indicators:
   a. CPU POWER CHECK (A19): 24V dc through relay K2 contacts are fed through parallel-connected contacts on the two voltage-sense relays to the CPU POWER CHECK indicator. If either of these relays is not picked after K2 is picked, this indicator lights (ALD YA021). 24V dc are also fed through parallel-connected contacts on the overcurrent and thermal-sense relays to the CPU POWER CHECK indicator (ALD YA022). 24V dc through 48V power-check relay contacts, or through the CPU READY/OFF switch (on the CE panel) in the OFF position, also light this indicator (ALD YA021).
   b. STOR FRAME 1 POWER CHECK (A21): 24V dc, via relay K32 contacts are fed through relay K29 contacts to this indicator. If K29 is not picked after K32 is picked, this indicator lights (ALD YA026).
   c. STOR FRAME 2 POWER CHECK (A22): Similarly, through relay K30 contacts, this indicator lights.
   d. STOR FRAME 3 POWER CHECK (A23): Similarly, through relay K20 contacts, this indicator lights.
   e. STOR FRAME 4 POWER CHECK (A24): Similarly, through relay K22 contacts, this indicator lights.
   f. CHAN FRAME 1 POWER CHECK (A26): Similarly, through relay K19 contacts, this indicator lights.
   g. CHAN FRAME 2 POWER CHECK (A27): Similarly, through relay K21 contacts, this indicator lights.
   h. CHAN FRAME 3 POWER CHECK (A28): Similarly, through relay K31 contacts, this indicator lights.
   i. System Power Check: A line from the CPU POWER CHECK indicator and a line from each unit Power Check indicator are combined at an OR. The OR turns on the System Power Check indicator, which is a red lamp backlighting the POWER ON pushbutton (N43). If a Power Check indicator is on, the System Power Check indicator also lights.
10. System Power-On Indicator: An indicator on the system control panel shows complete power-up status in the CPU and in the attached storage units and channels (ALD's YA026 and YA083). This System Power On indicator consists of a clear or white lamp

backlighting the POWER ON pushbutton (N43). This condition is also indicated on the remote operator's console. The operation of the System Power On indicator is described in Chapter 5.

CE PANEL

The CE panel is shown in Diagram 8-29, FEMDM. The controls and indicators function as follows (see Chapter 5 for details):

1. THERMAL RESET pushbutton: Resets the thermal sense relays in the CPU.
2. CPU READY/OFF switch. This switch has two positions:
   a. READY – normal position: Allows CPU power-up sequencing to continue if the thermal and overcurrent conditions are normal.
   b. OFF: Drops CPU power without affecting system power. The CPU is bypassed on a system power-on sequence.
3. CPU ON pushbutton: Starts CPU power-on sequencing if the CPU READY/OFF switch is in the READY position. Does not affect system power.
4. THERMAL TRIP indicators: These six indicators show the location of the overtemperature condition that dropped CPU power. The temperature sensors are located in gates A, B, C/D, and E, the converter/inverter, and the power supply tubs.
5. UNDER VOLTAGE CHECK switches (located on the relay gate below the CE panel in the converted units): These 15 switches isolate the power supplies from the undervoltage sensing circuits.

This section discusses the 2065 maintenance features available to the CE: (1) Diagnose instruction and associated MCW's; (2) Logout, ROS tests, and FLT's; (3) ripple tests; (4) diagnostic programs; and (5) marginal checking. For a discussion of the system control panel and the CE panel, refer to Section 1 of this chapter.

DIAGNOSE INSTRUCTION AND MCW'S

The Diagnose instruction (Section 7 of Chapter 3) has two purposes: it is available to the diagnostic programmer as a maintenance aid and to the system programmer for emulator mode operations. The Diagnose instruction has an SI format with an op code of 83:

83	I2	B1	D1
0 7	8 15	16 19	20 31

The operations that the Diagnose instruction can do are selected by the I2 field of the instruction and by the bit configuration of the doubleword addressed by the storage operand address (contents of GPR addressed by B1, + D1). The right half of this doubleword is the MCW, and bears the same relation to the Diagnose instruction as the CCW does to an I/O instruction. That is, the Diagnose instruction addresses the location of the MCW, and the MCW specifies the machine function. Note that an exception to this analogy is the I2 field, which can also designate certain diagnostic functions.

The bits of the I2 field have the following meaning:

Defeat Intlv		Diag FLT	▨	Set Em Mode	▨
No Rev	Rev				
8	9	10	11 13	14	15

1. Bit 8, Defeat Interleaving and No Reversal of Storage Addresses. Interleaving is disabled and no even/odd storage area reversal takes place.
2. Bit 9, Defeat Interleaving and Reverse Storage Addresses. Interleaving is disabled and the even and odd storage areas are reversed.
3. Bit 10, Diagnose FLT. Allows portions of FLT's to be executed under control of the Diagnose instruction. While the FLT's are being executed, special CPU functions are generated, and storage requests and clock are inhibited. This bit, which is used in conjunction with the setting of address 6B0 in ROSAR, simulates the FLT position of the TEST MODE, ROS/PROC/FLT switch on the system control panel.
4. Bits 11–13. Spares.
5. Bit 14, Set Emulator Mode. Allows the CPU to enter the emulator mode. (See Compatibility Feature FETOM's for a discussion of emulator mode operations.)
6. Bit 15. Spare.

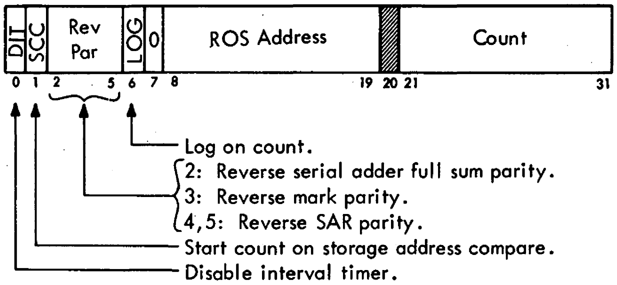
Besides the functions that can be selected by the I2 field, the Diagnose instruction can:

1. Perform channel diagnostic functions. When a channel is selected by the MCW, it is disconnected from its control units and connected to an internal (to the channel) interface simulator which consists of a one-byte register and associated controls. This simulator allows the diagnostic program to test the channel regardless of the type of attached control units that may be available.
2. Reverse the state of certain parity bits. Incorrect parity can thus be placed into the CPU or into the channel to test the error detection logic.
3. Cause a logout after a specified number of cycles (CPU only).
4. Branch into the ROS microprogram at any point.
5. Perform portions of the LS ripple tests.
6. Branch to an end-op micro-order and thus terminate normally.
7. Force PAL full-sum errors.
8. Perform scan-in operations. (Refer to *2065 Processing Unit* FEMM, Form Y27-2039-2 for the procedure.)
9. Stop the CPU clock.
10. Read the contents of the system control panel DATA switches into storage.

The MCW is a 32-bit word used to control Diagnose-instruction and scan operations. When the Diagnose instruction is executed, the MCW is in the main storage location specified by the storage operand address and designates the diagnostic function(s) to be performed. For ROS tests and FLT's, the MCW is contained in word 1 of each test on the ROS or FLT test tape and specifies the control conditions necessary for the test, such as the expected result and the ROS word to be used for gating control. A different format is used for each MCW.

Diagnose Instruction MCW for CPU

When executing the Diagnose instruction and MCW(7) = 0, the MCW applies to the CPU, as follows:

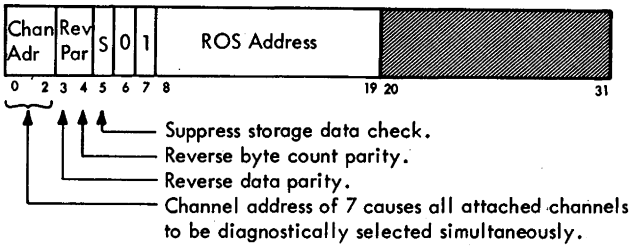


1. Bit 0, Disable Interval Timer. Disables the interval timer.
2. Bit 1, Start Count on Storage Address Compare. When used with bit 6, the FLT counter does not begin decrementing until the BCU addresses the same location as set into the MAIN STOR ADDRESS COMPARE (ADDRESS switches 2–20) switches.
3. Bit 2, Reverse Serial Adder Full-Sum Parity. Reverses the parity bit in the full-sum latch of the serial adder, thus allowing testing of the parity-checking circuits.
4. Bit 3, Reverse Mark Parity. Reverses the parity of the mark bits being sent from the BCU to main storage, thus allowing testing of the mark parity-checking circuits in main storage.
5. Bits 4 and 5, Reverse SAR Parity. Cause the parity bits which are sent to the storage address register to be reversed as follows:
   00 – No parity reversal.
   01 – Reverse low-order parity bit.
   10 – Reverse next-higher parity bit.
   11 – Reverse high-order parity bit.
6. Bit 6, Log On Count. Causes a logout to main storage when the FLT counter reaches 0. At the conclusion of the logging operation, the CPU performs a machine check interruption.
7. Bit 7. Must be a 0.
8. Bits 8–19, ROS Address. When the Diagnose instruction has completed its execution phase, these address bits are placed into ROSAR, and the operation branches to this location. The address placed into ROSAR can specify any location in ROS. Refer to Diagram 5-608, Sheet 2, FEMDM, for the most frequently used ROS addresses.
9. Bit 20. Spare.
10. Bits 21–31, Count. Specify the number of cycles (200 ns) that are to occur before the CPU enters a logout routine. The count field is made up as follows: MCW(21–25) is loaded into the address sequencer; MCW(26–29) is loaded into the FLT counter; MCW(30,31) is loaded into the FLT clock. By combining the address sequencer, the FLT counter, and the FLT clock, a maximum count of 2047 (11 bits) can be obtained. (These three counters are combined only when the MCW is used with the Diagnose instruction.) The FLT clock controls the decrementing of the FLT counter and address sequencer by 1; when the three counters combined equal 0, the logout routine is started.

Diagnose Instruction MCW for Channel

When executing the Diagnose instruction and MCW(7) = 1, the MCW applies to the channel. When a channel is selected by the MCW, the channel is disconnected from its control units and is connected to an internal interface simulator. The interface simulator consists of a one-byte register and associated controls. This simulator allows the diagnostic program to test the channel regardless of the type of control units that may be attached and available.

The MCW format for the channel diagnostic function is:

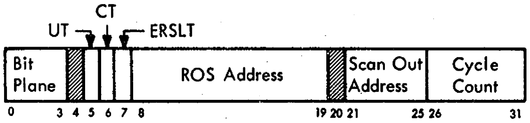

Chan Adr
Rev Par
S
0
1
ROS Address
0 2 3 4 5 6 7 8 19 20 31
Suppress storage data check.
Reverse byte count parity.
Reverse data parity.
Channel address of 7 causes all attached channels to be diagnostically selected simultaneously.

1. Bits 0–2, Channel Address. Select the channel to be diagnosed according to the following bit configuration:
   000 – Select channel 0 (multiplexer channel).
   001 – Select channel 1.
   010 – Select channel 2.
   011 – Select channel 3.
   100 – Select channel 4.
   101 – Select channel 5.
   110 – Select channel 6.
   111 – Select all channels simultaneously.
2. Bit 3, Reverse Data Parity. Causes all bytes that are read from the interface simulator to have reversed parity. This action allows testing of the storage bus-in parity check circuit.
3. Bit 4, Reverse Byte Count Parity. Provides a means of testing the byte control check circuits.
4. Bit 5, Suppress Storage Data Check. Prevents a storage data check from causing a channel data check and prevents a channel control check on a CCW fetch operation. Preventing the channel control check allows invalid CCW's to be brought into the channel to test sections of the channel check circuits.
5. Bit 6. Must be a 0.
6. Bit 7. Must be a 1.

7. Bits 8–19, ROS Address. When the Diagnose instruction has completed its execution phase, these address bits are placed into ROSAR and the operation branches to this location. The address placed into ROSAR can specify any location in ROS. Refer to Diagram 5-608, Sheet 2, FEMDM, for the most frequently used ROS addresses.
8. Bits 20–31. Spares.

ROS Test MCW

The MCW for ROS tests, the right half of word 1 of each test, contains the following control information about the test:

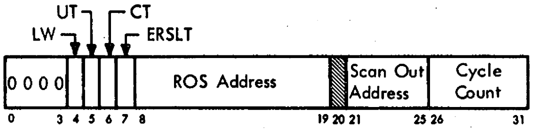


1. Bits 0–3, Bit Plane. Contain the number of the ROS bit plane tested. These bits are for display only; the ROS word is selected by the ROS address in bits 8–19, and the bit to be tested is selected by the mask.
2. Bit 4. Spare.
3. Bit 5, Unconditional Terminate (UT). If this bit equals 1, the test always causes a stop.
4. Bit 6, Conditional Terminate (CT). If this bit equals 1 and an error is encountered, testing is terminated. If, however, it equals 1 and the test does not detect an error, the CPU continues with the next ROS test. If this bit equals 0, termination is dependent upon the status of MCW(5).
5. Bit 7, Expected Result (ERSLT). Defines what the status of the ROS bit being tested should be. Comparing the ROS bit with the 1 or 0 in this bit determines whether the test passed or failed.
6. Bits 8–19, ROS Address. Contain the ROS address of the micro-instruction that contains the bit to be tested.
7. Bit 20. Spare.
8. Bits 21–25, Scan Out Address. Specify the scan-out address (roller switch position) of the portion of the ROSDR that contains the bit to be tested. This address will be either 15, 16, or 17. (See ALD M3051.)
9. Bits 26–31, Cycle Count. Determine the number of CPU clock cycles ROS must cycle to fetch the desired ROS word into ROSDR.

FLT MCW

The MCW in the right half of word 1 of every FLT contains control data about the test being performed. Pertinent bits of the MCW are retained in the MCW register during the running of an FLT. Other bits set the address sequencer, determine the ROS starting address, and fix the number of clock cycles the CPU will take following scan in.

Following is the format of an MCW during an FLT:



1. Bits 0–3. Contain 0's.
2. Bit 4, LW. Defines whether the left half or right half of the word to be scanned out contains the exit trigger status. If bit 4 equals 1, the left half is the desired word. If bit 4 equals 0, the right half word contains the trigger status.
3. Bit 5, UT. Always causes a stop when it equals 1.
4. Bit 6, CT. May cause a stop, depending on the outcome of the test. If this bit equals 1 and an error is encountered, testing is terminated. This bit is always set in zero-cycle and one-cycle FLT's currently in use.
5. Bit 7, ERSLT. Defines what the status of the exit trigger should be. Comparing the exit trigger state with the 1 or 0 in this bit determines whether the test passed or failed.
6. Bits 8–19, ROS Address. Contain the ROS address of the ROS word to be used for gating control.
7. Bit 20. Spare.
8. Bits 21–25, Scan Out Adaress. Specify the scan-out address of the scan-out word containing the status of the exit trigger.
9. Bits 26–31, Cycle Count. Fix the number of times the CPU is to cycle following scan in.

LOGOUT, ROS TESTS, AND FLT'S

Logout stores the status of the system control panel indicators into fixed positions of main storage when a trouble symptom occurs; the data logged out may be subsequently recalled for analysis although the status of the indicated logic is changed from what it was when the symptom appeared. ROS tests check each bit position of every ROS word. FLT's check the CPU at the logic block level.

Introduction

Logout, ROS tests, and FLT's are implemented by special hardware called scan logic. Therefore, when employing these maintenance aids, the CPU is said to be in the scan mode.

The scan logic performs the following functions:

1. Controls the operation of FLT's.
2. Records the state of the CPU when a machine malfunction is detected (logout).
3. Executes the Diagnose instruction.
4. Controls the operation of the ROS tests.

In the discussion that follows, certain terms are used which refer only to scan operations. These terms are defined as follows:

1. Scan mode. The CPU is said to be in "scan mode" during those operations that use the scan logic; i.e., FLT, ROS tests, and logout.
2. Scan in. The process of loading CPU register positions and control triggers with a predetermined bit pattern from storage.
3. Scan out. The sampling of the status of certain triggers via a special data path from the indicator logic to PAL.
4. Logout. The function that transfers status indicators and register contents to storage when signaled by the system.

Logout

The logout function of the scan logic stores the status of various triggers and registers, reflecting the state of the CPU, into predesignated locations of main storage. The 22 doublewords logged out (Table 6-1) are stored into main storage locations 80 through 128 (hex). (For a detailed list of the 22 doublewords, refer to ALD's M3021–M3061.)

Note: The Multisystem feature adds 16 log words (22–37, in main storage locations 180–1F8); see Chapter 4, Section 2.

The status of each trigger logged out is represented by a 1 if it is set, or by a 0 if it is reset. Thus a record of the machine state, at the time that a CPU or storage error occurs, is stored unchanged in predetermined locations of main storage with a fixed format. This record can then be accessed by a program or by manual controls for analysis, printout, or display.

A logout operation can be initiated by:

1. Manually depressing the LOG OUT pushbutton on the system control panel. The system must be in manual mode.
2. Executing the Diagnose instruction when a log-on-count function is specified. Logout occurs after a predetermined number of CPU clock cycles (preset by the programmer).
3. Detecting a machine check during normal CPU operation if the CPU CHECK switch is in the PROC position and the PSW machine check mask bit is on. When a storage error is detected and the CPU is enabled for machine checks, an additional word (log word 20) is logged out. This word, from the roller switch associated with the storage in error (position 5 on rollers 1, 2, 5, or 6), contains the information, excluding storage protect indicators, that is displayed in the roller switch indicators.

Table 6-1. Logout Format

Main Storage Byte Address (Hex)	Log Word	Left Half	Right Half
080	0	Parity for D-, F-, and Q-registers	Parity for ST-register
088	1	Miscellaneous controls	Parity for AB-, E-, and R-registers
090	2	G-register, LCS, and segmented clock	Storage checks
098	3	-----	Interruption, I-Fetch, and STAT's
0A0	4	PSW	BCU, FLT, and manual controls
0A8	5	-----	FLT controls, MCW, and counters
0B0	6	-----	ABC, STC, LAR, B(64–67), mark triggers, and PAL(64–67)
0B8	7	A-register	B-register
0C0	8	-----	Features
0C8	9	-----	IC and SAL
0D0	10	R- and E-registers	D- and F-registers
0D8	11	Q(0–31)	Q(32–63)
0E0	12	-----	CPU checks
0E8	13	Parity for LSWR, PSW, and IC; LSWR, PSW, IC, and checks	LSWR
0F0	14	-----	Gate control triggers
0F8	15	-----	ROSDR(70–99) and edit triggers
100	16	-----	ROSDR(37–69)
108	17	-----	ROSDR(1–35)
110	18	-----	ROSAR (3 registers)
118	19	-----	ROSAR, parity for ROSDR, LS address register, and gate control triggers
120	20	-----	Storage address, cycle, and data check(s)*
128	21	S-register	T-register

Note: PAL(64–67) and SAL(0–7) should always = 0 on logout, and the parity bits should always = 1.

*Present only when a storage check occurs and machine checks are enabled.

ROS Tests

- Test each bit of each ROS word.
- Tests are on tapes or disks.

ROS tests are the principal means of testing the validity of the ROS bit planes. These tests, generated by a computer program from the tapes used in the manufacture of the ROS bit planes, are stored on magnetic tape or disks. When testing ROS, the tests are read into two main storage buffer areas, starting at locations 8000 and 8080 (hex). Under scan logic control, the ROS tests compare the value of a particular bit in a selected ROS word with its expected value as specified on the ROS test tape. Each ROS test is read into the CPU and checks one bit of a ROS word. At the same time that the test from one buffer area is being executed in the CPU, the other buffer area is being filled from the test tape via the channel.

The ROS test format consists of two doublewords: word 0 and word 1. Word 1 contains the mask and MCW. The mask, a 32-bit field, in conjunction with the scan-out address field of the MCW, selects the ROS bit to be tested from the word read out of ROS. A particular bit is tested by making all mask bits 1's except the bit that corresponds to the test bit. Word 0 contains the test number (TN) and alternate test number (ATN). The TN, a four-byte field, contains two two-byte numbers that identify the test pattern. The lower-order two bytes are the complement of the TN, and the high-order two bytes are the TN. The ATN, which refers to another ROS test, is also represented in true and complement form, with the complement and true numbers reversed from that of the TN format. The TN refers to the test being executed. The ATN refers to the test that will be executed if the tests are restarted after a failure stop (generally, it refers to the next test).

The first part of each ROS test tape contains hardcore tests to establish that the CPU is able to run ROS tests. Testing should not proceed beyond the hardcore tests if failures are encountered.

Following the hardcore tests are the actual ROS tests. During these tests, the ROS word to be tested is selected by the ROS address in the MCW. The CPU clock is allowed to generate clock signals to cycle ROS so that the bit under test is placed into the ROSDR. The word containing this bit is defined by the scan-out address in the MCW and is transferred (scanned out) via the indicator driver logic and PAL to the T-register.

The mask is transferred from main storage to the S-register, and then the status of the bit under test is determined by comparing S with T. The result of this comparison is compared with the ERSLT bit to determine whether the test passed or failed. Pass, fail, and intermittent-fail are the three possible results of the comparison. The CT and UT bits in the MCW then determine whether to proceed with testing or to terminate.

If testing is terminated on a failure, the last two hex digits of the ROS address of the failing word are displayed in S(0–15) (roller 1, position 3), and the bit plane is displayed as the CPU TEST ADDRESS on roller 5, position 2, bits 0–3. To further define the S-register contents, S(0–7) indicates the failing word in hex, and S(8–15) indicates the failing bit in decimal.

The test is repeated until a transfer in channel (TIC) pulse notifies the CPU that the alternate buffer is filled and new test data is available.

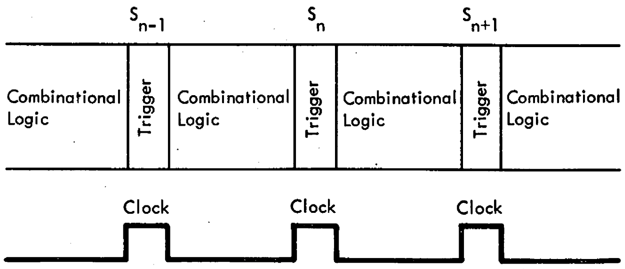
Because FLT's cannot be run if malfunctions exist in ROS, the ROS tests should be run first, followed by the FLT's.

FLT's

- FLT's check CPU logic at block level.
- Exit trigger determines whether test passed or failed.
- FLT's have three basic phases: scan in, clock advance, scan out.
- Three categories of FLT's are hardcore tests, zero-cycle tests, and one-cycle tests.

FLT's are a unique maintenance concept in that CPU logic is checked without executing CPU instructions (i.e., without executing a program in the ordinary sense). In an FLT, fault detection is performed at the logic level. That is, FLT's are concerned with the logical function (OR, AND, INVERT) of a block rather than with its operational function (e.g., as an adder, counter, control) in the CPU.

The tests are automatically produced by an IBM 7090 System which uses as its source the logic description of the CPU. Because the FLT's are not generated from an actual machine but are made from a mathematical model stored in 7090 storage, the FLT generator program treats the CPU as a set of triggers which can take on a new state (S) every time a clock signal is generated:



Thus, FLT's are generated for groups of logic blocks. A group consists of an exit trigger (a temporary storage element taking on new data at clock time and holding the information during not-clock time) and associated logic that can affect the trigger during an advance of a predetermined number of cycles.

The FLT generating programs analyze the group of logic blocks and determine, from a set of input values, an expected response at the output of the exit trigger. (Note that, although the exit trigger determines whether the test passed or failed, it is not being tested per se; the FLT tests the logic that precedes it.) Using the information gained by the analysis, the program generates a test pattern which it places on tape. Each test pattern is an FLT. At the same time, the program generates the documentation necessary to troubleshoot a failure detected by the FLT.

The application of an FLT can be broken into three phases:

1. Scan in. Triggers are set to the desired state by a scan-in microprogram. Triggers that cannot be set are assumed to be at a predetermined value.
2. Clock advance. The number of clock signals specified by the FLT MCW are allowed to be generated. These signals advance the CPU from one trigger level to the next. The exit trigger assumes a value that is a function of the state of the CPU during the previous cycle.
3. Scan out. The new value of the exit trigger is compared with the expected result. If the machine is operating properly, the two values should agree. If the new value and the expected result differ, a fault has been located, and testing terminates† with reference to the FLT documentation.

FLT's can be divided into three categories:

1. Hardcore tests. Check the scan and normal CPU logic necessary to run FLT's.
2. Zero-cycle tests. Determine whether a trigger value can be changed by scan in and also whether the new value can be sensed. Zero-cycle tests establish the machine capability to scan in and scan out before running one-cycle FLT's.
3. One-cycle tests. During these FLT's, data is scanned into the CPU, the clock is allowed to run, and the exit trigger is scanned out and compared with a known value. One-cycle tests check combinational†† logic within the CPU.

Note that the terms "zero-cycle" and "one-cycle" do not refer to the number of clock cycles allowed after scan in. For example, if during a zero-cycle test the exit trigger requires a clock signal to set it, the clock must run for one cycle. These terms refer to test techniques rather than to any time element.

FLT Tapes. Highlights:

- FLT's are stored on magnetic tape or disks in the following order: hardcore tests, zero-cycle tests, one-cycle tests.

The FLT's are stored on magnetic tape or disks. Each FLT tape consists of thousands of tests, each concerned with a single sensitive path. Also contained on the tapes are tests preliminary to FLT's; i.e., tests designed to ensure that the CPU is capable of performing FLT's and that the triggers to be tested can be set or reset and the change sensed.

Each tape is divided into three sections: hardcore tests, zero-cycle tests, and one-cycle tests. The one-cycle tests are considered to be the true FLT's because these test combinational logic. A brief description of the three kinds of tests on the FLT tape follows:

1. Hardcore tests. The test tape begins with hardcore tests to check out the CPU hardware necessary for operating the FLT's. Hardcore tests determine that the S- and T-registers are functioning properly, that their bit content can be correctly sensed at PAL, and that the CPU can make decisions based on the outcome of a test and then act on that decision.
2. Zero-cycle tests. Next in the testing sequence are the zero-cycle tests. These tests verify that the exit trigger status can be changed and that the change can then be observed or sensed. Zero-cycle tests set the trigger, using either a special scan input or a normal machine path, and then verify the change in status of the trigger being tested. Because clock signals are needed to set most triggers, the CPU clock is allowed to cycle just enough times to set the trigger. Upon completion of the zero-cycle tests, the CPU's ability to run FLT's has been verified.
3. One-cycle tests. These tests make up the bulk of the FLT tape. They vary in the amount of logic checked, and run in sequence until a failing test is encountered, whereupon testing is terminated and the failing test number is displayed in the S-register. One-cycle tests require two or three clock cycles to set a trigger.

Tape Generation. Highlights:

- FLT tape is computer-generated from ALD data.
- Program develops sensitive tree with entry points and terminating in an exit trigger.
- Tests are printed out in two formats: test, including entry pattern, and listing of sensitive points (SCOPEX) which is used in troubleshooting.
- New tests are added to existing tape using FLUT.

The FLT tape is computer-generated from ALD data. Using this data, a special program develops tests that, when executed, affect triggers indicatable on the system control panel (designated exit triggers). The program works back from all entries into each of these triggers,

† Termination is conditional upon the contents of the MCW.

†† Combinational logic is all the logic required to pass the state of one trigger to the next by executing a specified number of clock cycles.

searching the logic path for "sensitive" points; i.e., points at which an error or a failure would propagate to the exit trigger. Only one fault is assumed for each sensitive point.

The logic feeding these sensitive points makes up a sensitive net, and, as the number of nets grows, the combinational logic involved resembles a tree; the exit trigger forms the tip of the tree, and the sensitive nets make up the body of the tree. The search continues until the program encounters another trigger that can be used as an entry point into the sensitive tree. Usually, several entry points into a particular tree are available. However, entry points are selected on the basis that data injected at the entry points will propagate through to the exit trigger. The program also selects micro-orders that will move data from the entry points to the exit trigger. Upon completion of the search, the program has developed a sensitive tree with entry points and terminating in an exit trigger. The program also has developed an entry pattern that results in a predictable status of the sensitive nets and a predictable change at the exit trigger. Once this pattern has been developed, an evaluator program verifies its correctness.

Upon verification, the test is printed out in two formats: the test itself, including the entry pattern, and a listing of the sensitive points within the tree. The first of these becomes the taped FLT, and the latter is the scoping documentation (scoping index, or SCOPEX), used in troubleshooting a failing test.

To put the FLT's in CPU language, the entry pattern is scanned into the CPU, micro-instructions are selected to move the data through the sensitive tree during a given number of clock cycles, and the exit trigger is then observed (scanned out) to determine whether it is at the predicted value for the test. If the trigger is not at the predicted value, the test has failed, and the CE may then repeat the failing test continuously and scope the sensitive points on the tree to find the net with the failure in it. Certain sensitive points appear in more than one test, and the same exit trigger may be the observation point for more than a single sensitive path. Thus, newly tested points are indicated as being newly tested on SCOPEX to indicate to the CE that this is the first time these sensitive nets have been encountered and tested.

By verifying that AND's, OR's, and INVERT's are functioning as designed, FLT's verify the operating capability of the CPU although specific computer functions are not performed. FLT's ascertain that the CPU is operating according to design specifications and, therefore, should be capable of functioning as a CPU.

After the FLT has been evaluated as a valid test, it becomes a part of the FLT tape and is distributed for field use. New FLT's are continually being generated, some to test new logic that results from engineering changes and other to test areas of the CPU that are not now being checked. New tests are incorporated into existing tapes with the Fault Locating Utility (FLUT) program. Normally, however, a complete tape is sent to the field incorporating both old and new tests. An example of how one FLT is generated appears in Appendix C.

FLT Hardcore Tests. The first eight records on the FLT tape contain tests that ensure that the hardcore logic necessary to run zero-cycle and one-cycle tests is operational. Hardcore logic is defined as all the logic, scan and normal, necessary to load the FLT's into storage, to scan in, to scan out, to make decisions based on the outcome of a test, and to act on those decisions.

The CPU hardcore hardware testing is subject to the following limitations:

1. Scan hardware is not tested directly; it is, however, exercised in the hardcore portion with tests designed to isolate the trouble.
2. The BCU is not tested in the FLT's, but the BCU must be functioning properly to run FLT's. Also, main storage and storage buses must be operating. Main storage may be checked using ripple tests.
3. The channel and tape drive (or disk) used to read in the FLT's are not tested. These units must be tested using manual controls on the channel.
4. Local storage is not needed to run FLT's, and, therefore, is not tested. Local storage is checked by ripple tests.
5. The ROS microprogram and ROS must be fault-free to run FLT's. Therefore, because hardcore tests do *not* check ROS, ROS tests should be run before FLT's.

Zero-Cycle Tests. To further check the operation of the scan hardware, zero-cycle tests determine whether the scan-in and/or scan-out paths are operative. Zero-cycle tests check only those triggers displayed on the system control panel. In these tests, a pattern is scanned into the machine, the clock is not advanced, and the exit trigger is observed. If the trigger has a scan-in path, three tests are performed: one for the reset state, one for the set state, and one for the reset state again. If the trigger has no scan-in path, only one test is performed for the reset state. While one trigger is being tested, other triggers can assume various states. Whenever possible, random states are used to simulate the combinations that may be used in a normal test and to reveal interaction between triggers.

The functions checked with zero-cycle tests include:

1. Reset to triggers (this is, in effect, a scan-in-zero operation).
2. Scan-in path to triggers.
3. Ability of a trigger to hold its value in the absence of a clock signal.
4. That scan-out bus gating signals can be generated.

One-Cycle Tests. One-cycle tests are the true FLT's. The input, produced by the FLT generating system, is a test that detects and locates at least one fault. The input pattern is scanned into selected triggers, and the CPU is allowed to advance a given number of cycles. The result, which is in the exit trigger, is compared with the value expected for a correctly operating machine.

If the value in the trigger does not agree with the expected result, testing is terminated and the failing TN is displayed in S(0–15) for reference to the SCOPEX.

The SCOPEX is a series of lists, one for each test. Each list is headed by the test number in hex, followed by a row of asterisks, and consists of several lines, each line referring to a pin in the machine. If a pin in the machine is contained in a list, the net which feeds that pin is sensitive for the test pattern applied; a failure on the card, which can be observed with an oscilloscope at that pin, would cause the test to fail.

FLT Format. Each FLT on the test tape consists of nine doublewords, numbered 0 through 8 (10 doublewords for machines with the 7080 Compatibility Feature or the 709/7040/7044/7090/7094/7094II Compatibility Feature). This is one complete test. Each test occupies one of two areas in storage. Buffer 1 begins at main storage location 8000 (hex) and contains nine doublewords. Buffer 2 begins at main storage location 8080 (hex) and also contains nine doublewords.

While one test is being executed from one buffer, the other buffer is being filled from the channel. Filling the buffers is a sequential process, and the processing of the FLT's is in the same sequence, without further addressing.

The contents of the nine or ten doublewords is as follows:

1. Word 0 and words 2 through 7 or 8 contain bit patterns that are scanned into the CPU.
2. Word 1 contains a mask and the FLT MCW. The mask defines the exit trigger; the MCW, in the right half, contains control information about the test.
3. Word 8 (or 9) contains the TN in both true and complement form and the ATN in the same format. Word 8 (or 9) is not read into the CPU during test sequencing unless a fault is encountered or an ATN search is performed.

At the satisfactory completion of a test, the 'pass' trigger is set, the address sequencer is set to 7 (or 8), and word 7 (or 8) of the test is the first word scanned in again. When a test fails, the 'fail' trigger is set. After a 'TIC' pulse is received from the channel, the last thing the CPU does is to fetch word 8 (or 9) from storage and leave it in the S- and T-registers. If the CE desires to skip this test and continue with the remainder of the tests on the tape, he may depress the RESTART FLT I/O pushbutton. The signal generated by depressing this pushbutton causes the I/O device to backspace to the beginning of the record and to start reading the tests into storage again. As each buffer is filled, the TN in the left half of word 8 (or 9) is put into the S-register, and a comparison is made between the new TN in the S-register and the ATN in the T-register by using the scan out S and T facility. The result is 0 for the test immediately following that test in which a failure occurred. Upon detection of this 0 result in PAL, the CPU again begins testing and continues until the next failure. This procedure is the only way of getting past a failing test and continuing the test tape sequence.

Scan Logic Functional Units

The scan logic functional units (Diagram 8-101, FEMDM), in conjunction with the CPU operational hardware and the ROS microprogram, control the CPU during FLT, ROS tests, logout, and Diagnose instruction execution. During these operations, the scan logic supplements and sometimes overrides the operational CPU logic. In addition, the scan logic provides direct data flow paths (scan in) into triggers not ordinarily having direct entry paths and from indicatable storage devices to PAL (scan out).

During scan operations, CPU timing is controlled by two clocks (scan clock and FLT clock), and sequential operation is determined by four counters: address sequencer, check counter, FLT counter, and ROS test sequencer. These counters time-share the scan counter latches and decrementer. That is, although each counter is used for a specific function at a specific time, their contents are all decremented by the scan counter decrementer. For certain operations, the address sequencer and the FLT counter can be logically joined to triggers in the FLT clock to form an 11-bit cycle counter.

The five-bit address sequencer contains the low-order bits of storage word addresses, and its decoded value controls the gating signals to the scan-in or scan-out logic. The address sequencer also forms the high-order five bits of an 11-bit counter (as mentioned previously) for the Diagnose instruction and pulse-mode functions. The six-bit FLT counter counts CPU clock signals when required; the low-order two bits of the FLT counter, which are contained in the FLT clock, count oscillator signals reaching FLT controls. The check counter permits a tape record to be read a maximum of 32 times in case of tape errors or storage errors before determining that a test cannot be performed. The three-bit ROS test sequencer sequences the scan hardware during ROS tests.

Each ROS test or FLT has an associated MCW which allows the test to be controlled by a test pattern on tape. MCW's, including the Diagnose instruction MCW, are stored in the scan functional units and decoded by scan hardware.

Scan Timing

Scan timing is controlled by two clocks, the scan clock and the FLT clock. The scan clock is synchronized to the CPU clock and produces clock and not-clock signals similar to the CPU clock. Normally, the scan clock runs continuously whether the CPU is in scan mode or functional operation, and can be made to run with the CPU clock off. The output of the scan clock steps the FLT clock.

The FLT clock is used in conjunction with the scan clock to provide scan control timing. It is synchronized to the scan and CPU clocks, and provides four output signals, FLT-time 0–3. The two FLT clock triggers can be logically combined with the address sequencer and the FLT counter to form an 11-bit cycle counter.

Scan Clock. Highlights:

- Consists of inverters which delay 'gated oscillator D' signal from CPU clock.
- Distributes clock and not-clock signals with delay levels of P0-3 to P2.
- Runs continuously unless blocked by BCU.
- During scan operations, CPU clock is turned off, leaving scan clock in control.

The scan clock (Diagram 8-102, FEMDM) consists of a number of inverters, each of which delays a 'gated oscillator D' signal from the CPU clock approximately 10 ns. The output of each delaying inverter is distributed as a clock or not-clock signal with a delay level of from P0-3 to P2. (See Chapter 2, Section 1, for an explanation of clock, not-clock, and delay levels.) The scan clock runs continuously unless one of the following actions stops it:

1. The 'pass pulse' trigger is reset (which also stops the CPU clock). This trigger is set when any operation is initiated from the system control panel, and normally remains set throughout most CPU operations. However, it can be reset by one of the following conditions, if the BCU is not holding the CPU clock on:
   a. 32 attempts to read in an FLT or ROS test were unsuccessful because of channel or storage errors.
   b. The UT bit was on at the completion of an FLT or ROS test.
   c. The CT bit was on and the 'fail' trigger was set during an FLT or ROS test.

   Note: The above three conditions reset the 'pass pulse' trigger at FLT-clock-2 time if the ROS test sequencer is at 0.

   d. The 'block' trigger is set. This trigger is set if one of the pushbuttons listed in Diagram 8-102 is depressed in single-cycle mode.
2. The 'stop clock' trigger is set with the RATE switch *not* in the SINGLE CYCLE STORAGE INHIBIT position and with the TEST MODE, REPEAT switch not set. The 'stop clock' trigger is set by the 'CPU 2' latch, the 'insert key' trigger, or the 'LCS interleave' latch.
3. The 'STOP 1' or 'STOP 2' micro-orders are executed.

Because the scan clock is driven by a 'gated oscillator D' signal (basic clock signal delayed; Diagram 4-3, FEMDM), it is always in synchronization with the CPU clock. The scan clock signals, P0-3, P0-2, P0-1, P0, P1, and P2 (Diagram 8-102) are developed by passing the 'gated oscillator D' signal through inverters. Each inverter delays the signal approximately 10 ns. The P0-3 signal is the first scan clock output and is developed after being delayed by an adjustable time delay. (The time delay should be adjusted so that P0 from the scan clock coincides with P0 from the CPU clock; See ALD M8001.) The P0-3 signal is then inverted and delayed 10 ns to produce the P0-2 signal. The P0-2 signal, in turn, is inverted and delayed to generate P0-1. The remaining scan clock signals are generated in the same manner as P0-2 and P0-1. These clock signals are distributed throughout the scan logic to time scan operations. Clock and not-clock signals have the same relation to scan triggers and latches as CPU clock and not-clock signals have to functional logic (see Chapter 2, Section 1).

The advantage of having a separate scan clock for scan logic is that during scan operations the CPU clock can be turned off, leaving the scan clock in control. This function is under control of the 'maintenance mode stop clock' (MMSC) trigger. (The operation of the MMSC trigger is discussed in "Scan Stop-CPU-Clock Logic".)

With both the scan and CPU clocks running, scan hardware is controlled by the scan clock, and normal CPU functions are controlled by the CPU clock.

FLT Clock. Highlights:

- Provides scan control timing and counts machine cycles.
- Consists of 'FLT clock-0' and '-1' triggers and 'FLT time-0' to '-3' latches.
- Stepped by scan clock.

The FLT clock (Diagram 8-103, FEMDM) is the prime scan operation sequencer which, in conjunction with the scan clock, provides scan control timing and counts machine cycles. It consists of two triggers, 'FLT clock-0' and '-1', and four latches, 'FLT time-0' to '-3'. Clock and not-clock signals from the scan clock step the FLT clock to provide the outputs shown in the timing chart. Six signals are generated, 'clock-0' and '-1', and 'FLT time-0' to '-3'. The clock steps once for each machine cycle. Thus, in four machine cycles, the FLT clock steps from 'FLT time-0' to '-3' and then recycles.

Two triggers, 'FLT clock 0' and 'FLT clock 1', are stepped by scan clock signals and by the conditions of the four latches. 'FLT time-0', '-1', '-2', and '-3'. The FLT clock triggers function as a reverse binary counter of scan clock cycles. The FLT time latches record the count indicated in the FLT clock triggers.

Assume that the 'FLT time 0' latch is set and both FLT clock triggers are reset (00). Because 'FLT time-1' to '-3' latches are reset, the rise of scan clock P0 sets both FLT clock triggers (11). At not-clock P0 time, the 'FLT time 0' latch is reset, and at not-clock P1 time, the 'FLT time 1' latch is set. Then, at clock P0-1 time, both FLT clock triggers are reset.

Because the 'FLT time 1' latch is set, only the 'FLT clock 0' trigger is set at clock P0 time, at which time the triggers equal 10. This operation continues with the triggers counting down binarily from 11 to 00 during clock time and the latches stepping up to 11 during not-clock time. When the trigger count is 00, the next not-clock P1 signal sets the 'FLT time 0' latch, and the cycle is repeated.

The FLT clock triggers are also used as the low-order two bits of the FLT counter and the cycle counter. As shown in Diagram 8-103, T(62,63), containing MCW(30,31), can be transferred to these triggers during cycle counter operation. Operation of the triggers as part of the FLT counter and the cycle counter is discussed in "FLT Counter" and "Cycle Counter", respectively.

Scan Counter Latches and Decrementer

Three counters and two sequencers (Figure 6-4) control the sequential operation of scan associated functions: the address sequencer, the check counter, the FLT counter, the ROS test sequencer, and the cycle counter. (The latter is an 11-bit counter composed of the address sequencer, the FLT counter, and the two FLT clock triggers.) Each sequencer or counter is decremented by routing the contents of the sequencer or counter through the scan counter latches and back to the source via the scan counter decrementer. The scan counter decrementer subtracts 1 from the value gated through it. However, although all counters are decremented by the scan counter decrementer, only one counter may be stepped during any one clock cycle.

Input and Output. Inputs to the scan counter latches are from the address sequencer, the check counter, the FLT counter, and the ROS test sequencer. Input is accomplished at not-clock time. The scan counter latch output automatically conditions the scan counter decrementer and, at the following clock time, the decrementer output is transferred into the source counter or sequencer. Inputs to the scan counter latches and to the source counter or sequencer are under control of the same 'not-clock P1' signal as shown in Figure 6-4. This signal resets the counter or sequencer before the decremented value is entered.

Scan Counter Decrementer. The scan counter decrementer (Diagram 8-104, FEMDM) is a logic network that subtracts 1 from any value routed through it. For example, assume that the address sequencer contains 21 (decimal) or 10101. When the address sequencer contents are to be decremented, they are transferred to the scan counter latches. The value is then routed through the decrementer, where it is reduced to 20 (10100) and sent back to the address sequencer.

Address Sequencer

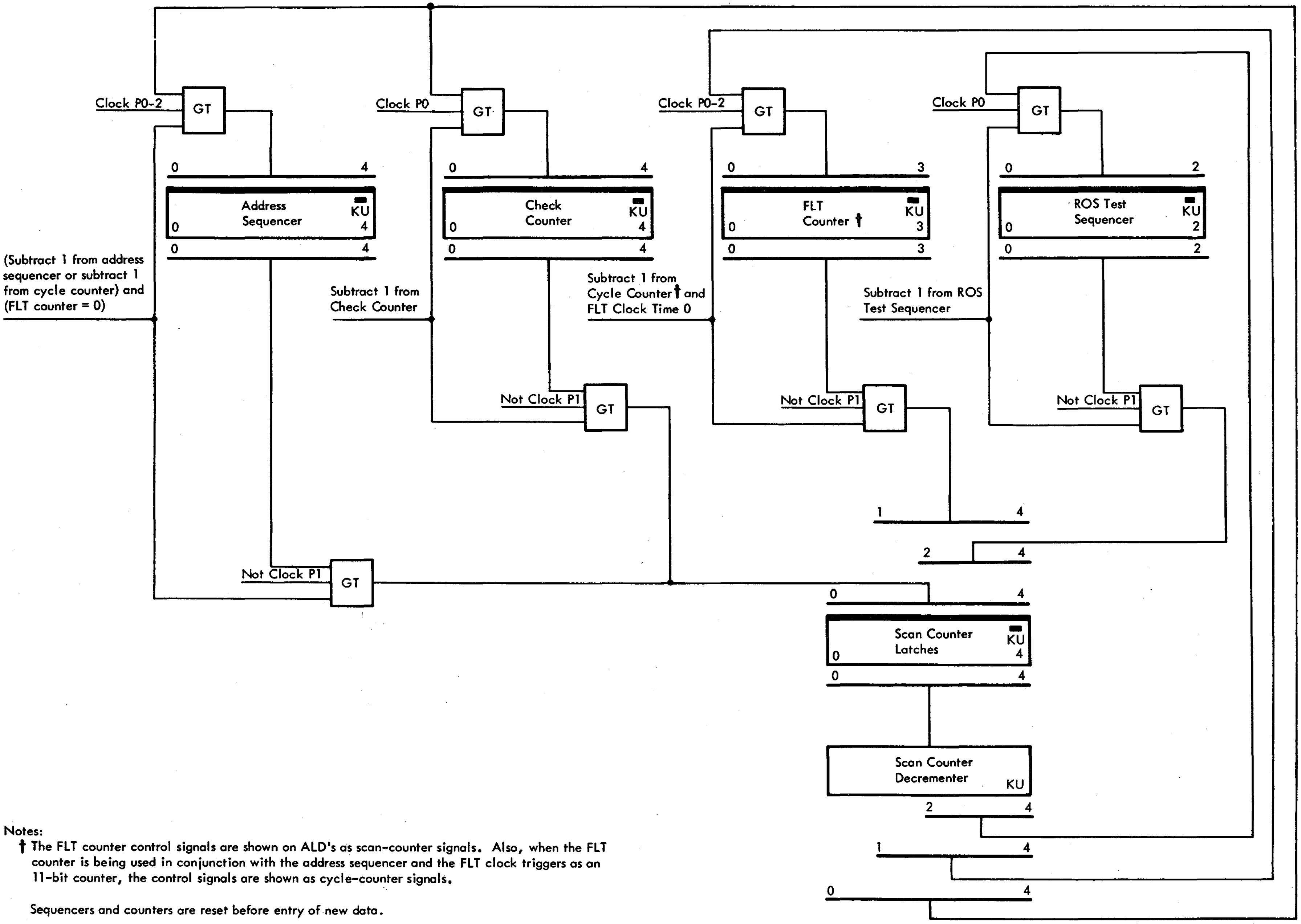
- Provides data used to generate main storage addresses; generates signals to select scannable triggers and latches on scan operations; functions as five high-order positions of cycle counter.
- Inputs are from T(53–57), hardware (set to 21 or 23), ROS micro-orders (set to 7, 8, or 21).

The address sequencer, a five-position trigger register, has two main functions: to sequence through a predetermined number of main storage addresses, and to select the scan address of the scannable triggers and latches to be scanned out to PAL. When the address sequencer is performing the first function, either a fixed address or a portion of the MCW is placed in it, and its output is transferred to a storage address generator which adds the necessary bits to make up the main storage address. The address sequencer contents are then decremented and transferred to the storage address generator to select the next address, and so forth. For the second function, the sequence is generally the same, except that the contents are transferred to the address sequencer decoder, which brings up 1 of 21 gating signals to scan out the proper scannable triggers and latches. Decrementing is accomplished by transferring the address sequencer contents to the scan counter latches and back again via the scan counter decrementer.

The address sequencer is also used as the five high-order positions of the 11-bit cycle counter during log-on-count or pulse-mode operation. This function of the address sequencer is discussed in "Cycle Counter".

The address sequencer (Figure 6-5) can be loaded with preassigned values by micro-orders or hardware, or T(53–57) can be transferred to it depending upon the operation. Its contents can be transferred to the address sequencer decoder for selection of scannable triggers and latches, to the storage address generator for addressing storage, or to the scan counter latches and decrementer.

During the scan-in portion of FLT's, the address sequencer is set to a count of 8 and then decremented by 1 for each test word fetched from main storage. The eighth word fetched (address sequencer equals 1) is the MCW part of the FLT test pattern and contains a new setting for the address sequencer. This new setting selects the scannable triggers and latches to be scanned out for the test comparison. The scan-out word is selected by decoding the address sequencer contents and the LW bit of the MCW. This action selects the roller switch indicating the exit trigger.

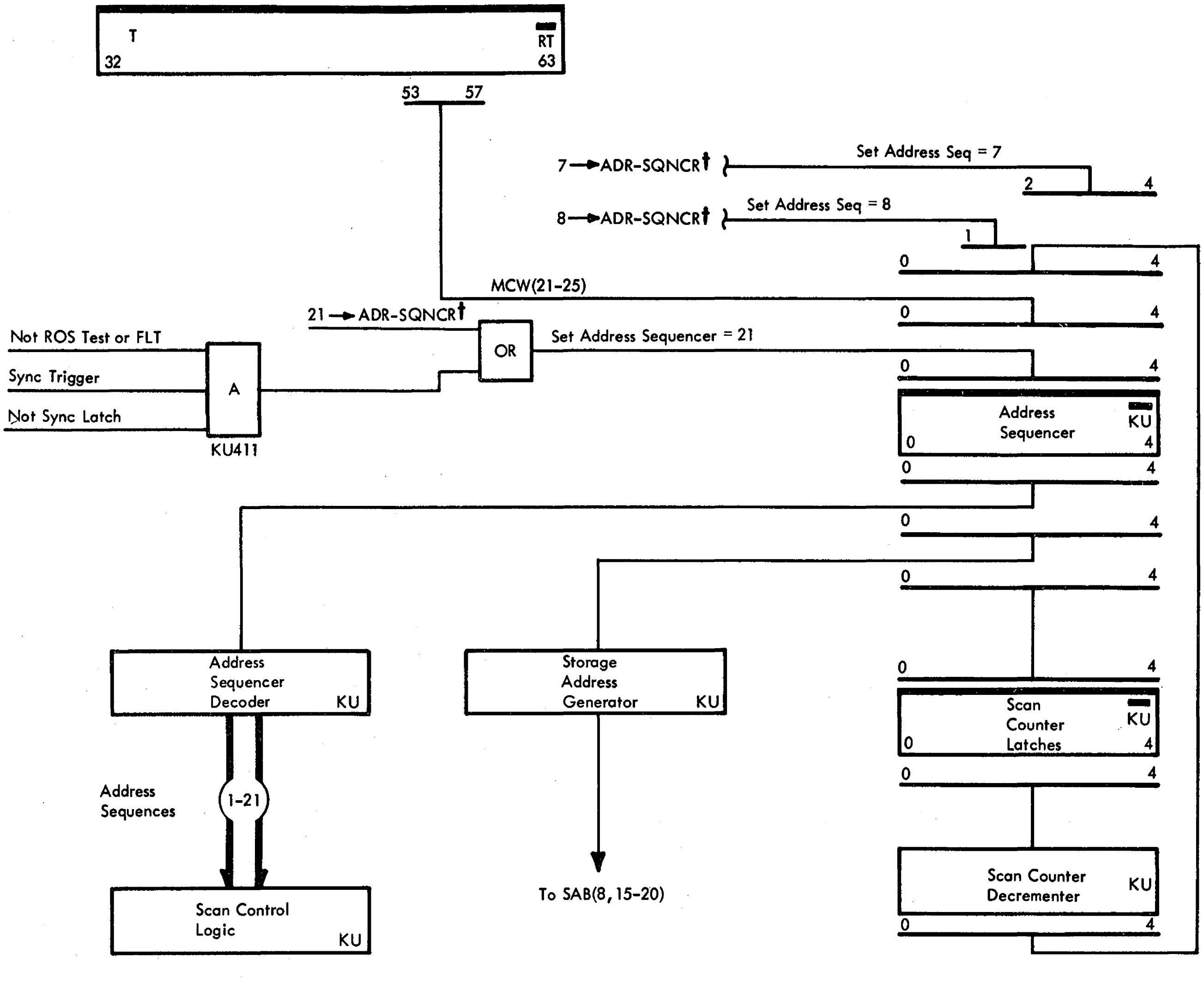


Notes:

† The FLT counter control signals are shown on ALD's as scan-counter signals. Also, when the FLT counter is being used in conjunction with the address sequencer and the FLT clock triggers as an 11-bit counter, the control signals are shown as cycle-counter signals.

Sequencers and counters are reset before entry of new data.

Figure 6-4. Scan Counter Latches and Decrementer Data Flow



†ROS micro-order

Figure 6-5. Address Sequencer Data Flow

The action taking place while the address sequencer steps from 8 to 0 is under control of the microprogram. When the count in the address sequencer is 0, the scan-in is complete for FLT's.

During a logout operation, the address sequencer is initially set to a count of 21 or 23 (decimal) and then decremented by 1 for each doubleword logged out. Again, the count in the address sequencer is decoded by the address sequencer decoder, and the logout word is selected using the indicator drivers as the source point. The address sequencer count is also decoded by the storage address generator to yield a main storage address for the logout word. This address is then put on the SAB before storing the logout word into main storage. While the sequencer steps from 21 (or 23) to 14, the CPU clock is turned off to prevent register operation. At a count of 13 (decimal), the CPU clock is again allowed to run, and the logout operation continues under ROS control. At count 0, logout is complete.

During logout, the address sequencer goes to 0 twice. The first time the count goes to 0, the sequencer is forced to a count of 21. This action recalls the first word logged out (word 21, ST contents) to correct possible incorrect parity. The word is then restored, and the sequencer is set to a count of 7 and again allowed to count down to 0, at which time the original parity bits are stored and the operation is completed. (Because no micro-order exists to set the address sequencer to 0, it is set to 7 and allowed to decrement to 0.)

Address Sequencer Decoder

The address sequencer decoder (Figure 6-6) interprets the output of the address sequencer during scan operations. The decoder consists of a high-order and a low-order bit section. The high-order bit section decodes the two high-order bits to yield, for example, 10XXX. The three low-order bits are then decoded to give, for example, XX101. The combination of the two, in this instance,

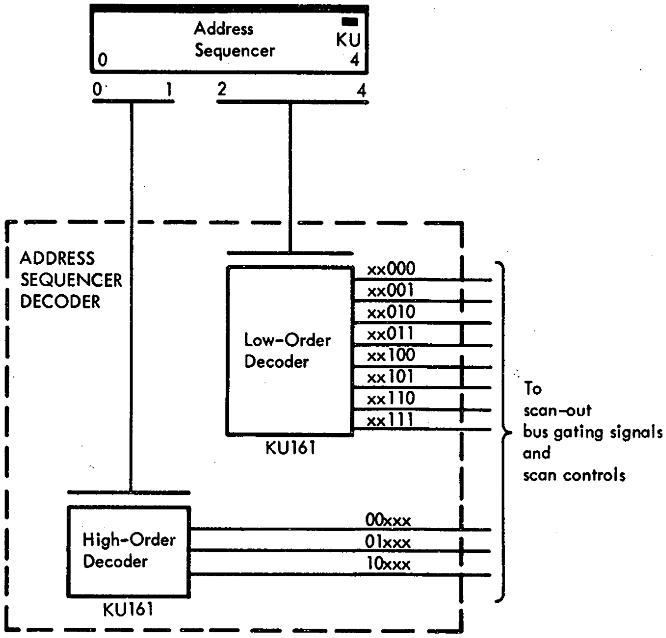


Figure 6-6. Address Sequencer Decoder

shows the address sequencer count to be 21 (10101). Thus, the word logged out or scanned out would be word 21, the contents of S and T.

The scan logic uses the indicator drivers, normally used for system control panel display, as scan-out source points. Therefore, the address sequencer selects the roller switch that contains the desired information. In the case where the count of the address sequencer is 21, the desired information is found in rollers 1 and 2, position 3 (S and T) (see "Scan-Out Bus").

Storage Address Generator

- Scan operations use three areas of main storage: logout area (80–128, hex), buffer 1 (8000–8048, hex), buffer 2 (8080–80C8, hex).
- SAB values to address location within the three areas are derived from address sequencer contents.

Three areas of main storage are used for scan operations: locations 80 through 128 (hex) is the logout area, which contains the log words after a log operation; and locations 8000 through 8048 (hex) and 8080 through 80C8 (hex) are the two areas that contain the FLT's or ROS tests. The SAB values necessary to address any location within these three areas are basically derived from the contents of the address sequencer. Forming a 24-bit address from the five bits of the address sequencer is accomplished by the storage address generator (Diagram 8-105, FEMDM).

The chart in Diagram 8-105 shows how the storage address is encoded. For all scan operations, bits 0–7, 9–14, and 21–23 of the storage address are set to 0's (bits 21–23 are 0's because logout words, FLT words, and ROS-test words are doublewords which are always on doubleword boundaries). Address sequencer bits 0 and 1–4 are transferred to SAB(15,17–20) to select the location within a scan area of storage.

In logout operations, bit 16 of the address is set to the complement of bit 0 of the address sequencer. Thus, for logout area addresses 80 through F8 (hex), SAB(15) is set to 0 and SAB(16) to 1; for addresses 100 through 128 (hex), SAB(15) is set to 1 and SAB(16) to 0. SAB(8) is always set to 0.

FLT's or ROS tests are always contained in two areas of storage called buffers. Buffer 1 is from 8000 through 8048 (hex), and buffer 2 is from 8080 through 80C8 (hex). When ROS tests or FLT's are being performed, each test, as it is read in, goes to the opposite buffer. For example, if the first test read from tape is placed into buffer 1, the next test is placed into buffer 2, the third into buffer 1, and the fourth into buffer 2, and so on. This scheme allows one test to be performed while the next test is being read into storage.

SAB(16) controls which buffer is addressed. When SAB(16) = 0, buffer 1 is addressed; when SAB(16) = 1, buffer 2 is addressed. During FLT's and ROS tests, SAB(16) is under control of the 'buffer 1' trigger. When this trigger is set, indicating that buffer 1 is to be addressed, SAB(16) is forced to 0. When the 'buffer 1' trigger is reset, indicating that buffer 2 is to be addressed, SAB(16) is set to 1.

During ROS tests, the 'buffer 1' trigger is complemented every time the ROS test sequencer equals 5. During FLT's, the 'buffer 1' trigger is under control of the 'ROS 31→CHK-CNTR' micro-order. If this micro-order is decoded and a test is in storage ('TIC' pulse received), the 'buffer 1' trigger is complemented.

For FLT's and ROS tests, SAB(8) is always set to 1 to address the portion of main storage starting at 8000 (hex).

SAB gating signals during scan operations are controlled by the 'scan sync' trigger. When this trigger is set, the contents of the address sequencer are transferred onto SAB. During FLT's and logout operations, the trigger is set by the 'ROS MS-REQ*SCAN4' micro-order. During ROS tests, it is set when the ROS test sequencer equals 1, 3, and 6.

Check Counter

The check counter, a five-position polarity-hold register, records errors encountered while reading FLT or ROS test tapes into storage and during scan in. At the start of the read-in operation, the check counter is set to 31 (decimal) (maximum count for a 5-bit register). It is then decremented by 1 each time an error is encountered. At the same time that the check counter is decremented, another attempt is made to read in the test that produced the error. After 32 errors (attempts at reading in the test), the check counter is stepped to 0, and the operation is terminated.

Decrementing is accomplished by transferring the check counter contents to the scan counter latches and back again via the scan decrementer.

Input and Output. The check counter (Figure 6-7) can only be set to 31 (decimal) or decremented via the scan counter decrementer; it has no other data inputs or outputs. It is set to 31 (decimal) if any of the following conditions occur:

1. Execution of the '31→CHK-CNTR' micro-order with the 'gap' trigger set.
2. ROS test sequencer equals 0 or 6.
3. ROS test sequencer equals 1 and the 'repeat' latch is set.
4. ROS test sequencer equals 4 and the 'pass' trigger is set, unless one of the following occurs:
   a. An input error is detected.
   b. The UT bit is set.

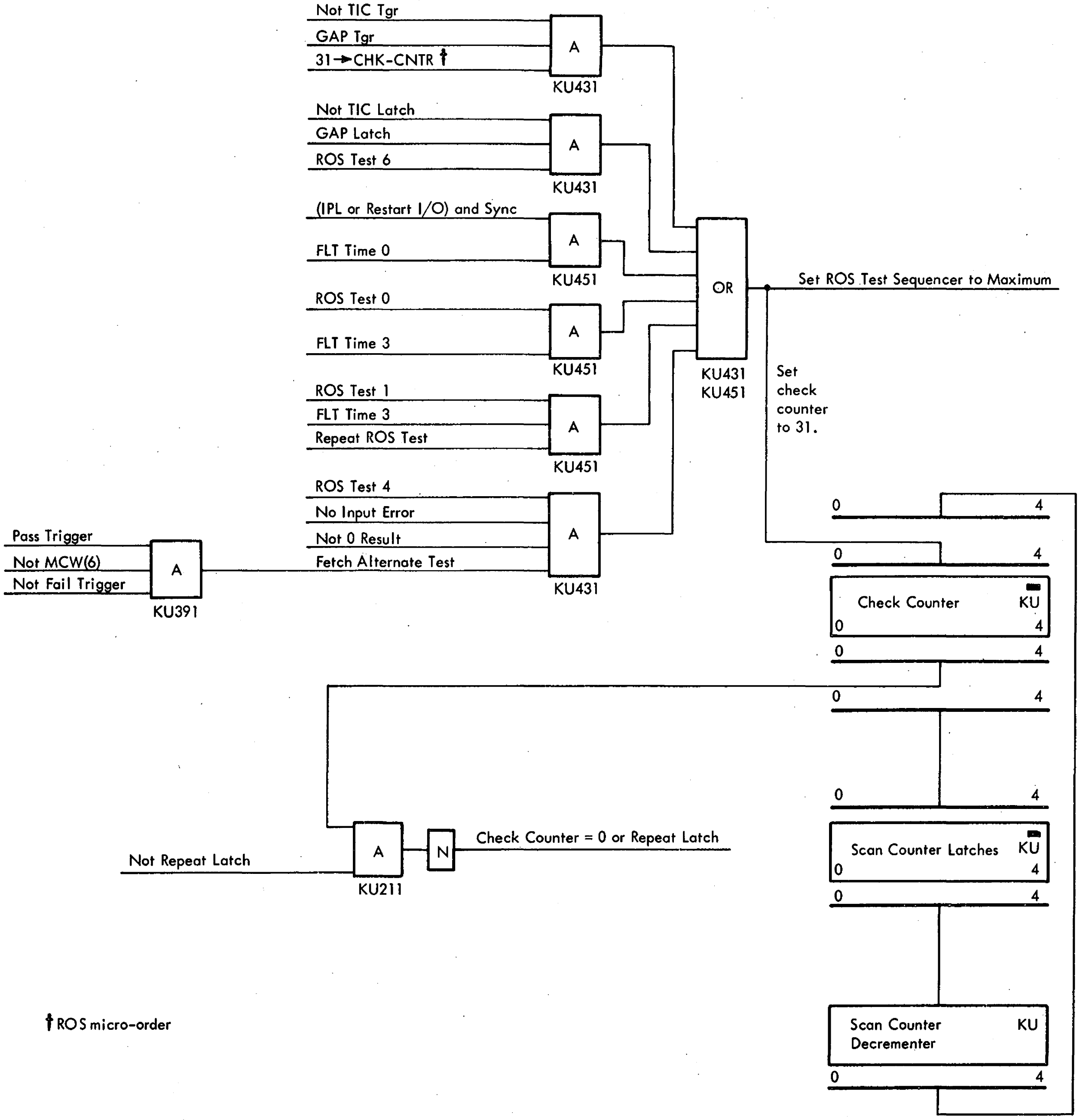


Figure 6-7. Check Counter Data Flow

c. The 'fail' trigger is set.

d. The test comparison in PAL resulted in 0.

5. An IPL is initiated or the test is restarted.

Check Counter Decrementing. The check counter decrementing controls, when activated, transfer the contents of the check counter to the scan counter latches and decrementer and back to the check counter; thus, the check counter is reduced by 1. During FLT's, the check counter is decremented by the 'ROS CHK CNTR-1' micro-order. During ROS tests, the check counter is decremented whenever an input error is detected.

FLT Counter

The FLT counter is a four-latch register/counter which is coupled with the two triggers of the FLT clock to make a six-position counter. Its main purpose is to count the number of cycles that the CPU is allowed to advance after scan in and before scan out. During FLT's and ROS tests, the FLT counter is set to the desired count by the cycle count field [MCW(26–31)]. During log-on-count and pulse-mode operations, the FLT counter (four bits) is combined with the address sequencer (five bits) and the two triggers of the FLT clock to make up an 11-bit cycle counter.

The two parts of the FLT counter are decremented differently. The low-order bits, FLT-clock 1 and 2, are a reverse binary counter that counts down from 3 to 0. Decrementing is accomplished by signals from the scan clock. Each time the low-order bits reach 0, the high-order bits, FLT-counter 0–3, are decremented via the scan counter decrementer in the same manner as the address sequencer and the check counter. The following scan clock signal resets the FLT-clock 1 and 2 to 3.

Input. The FLT counter (Figure 6-8) is loaded from T(58–61) or forced to the maximum count. Except during certain portions of the ROS tests, the MCW cycle

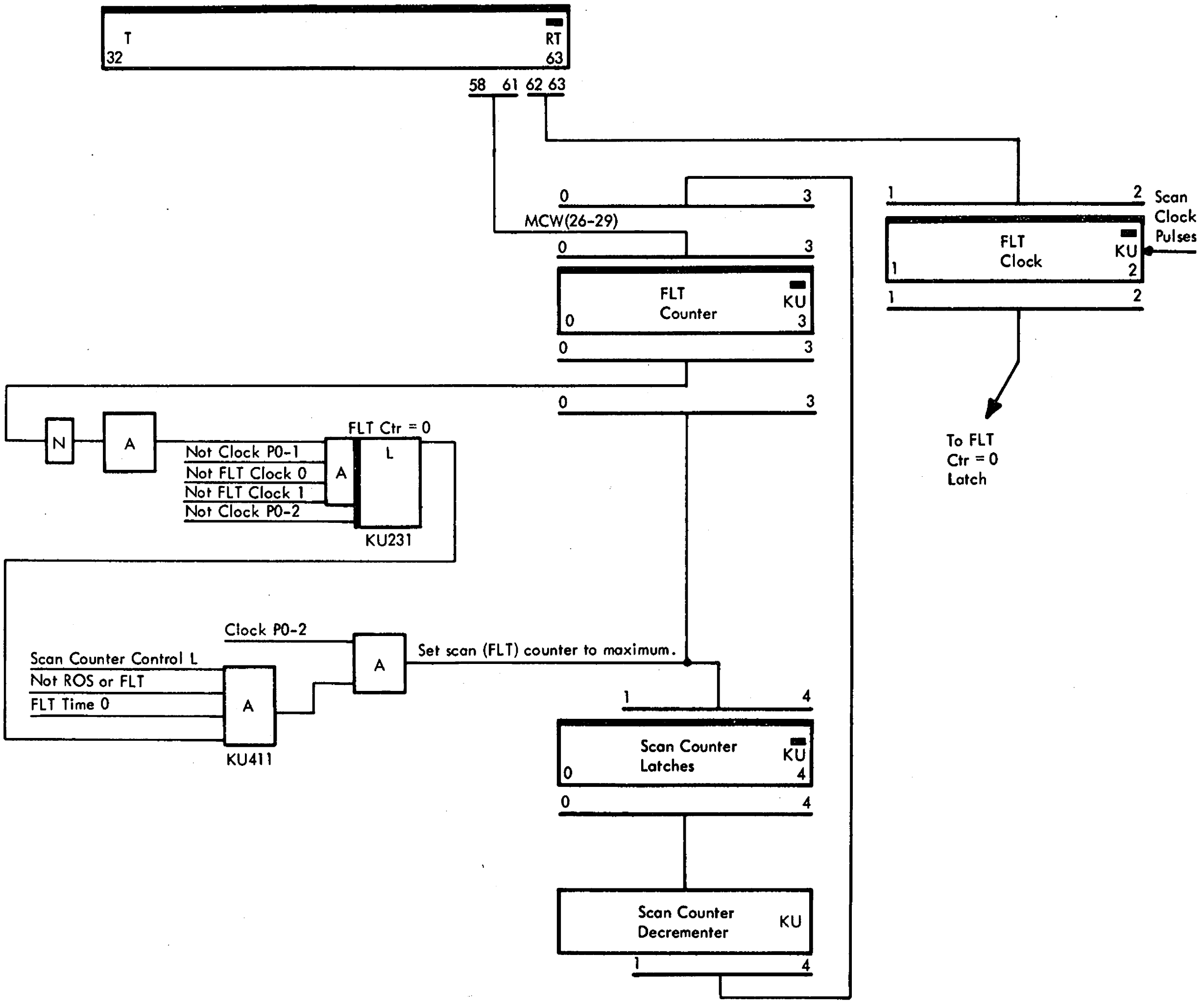


Figure 6-8. FLT Counter Data Flow

count field is gated from T(58–63) to the FLT counter and FLT clock by the 'ROS T→MCW' micro-order. During ROS tests, the cycle count field is gated from T to the FLT counter when the ROS test sequencer equals 2.

FLT Counter Decrementing. The FLT counter (Diagram 8-106, FEMDM) is decremented under control of two latches: the 'scan counter control' latch and the 'FLT counter equal 0' latch. Initially, the 'FLT counter equal 0' latch is reset, conditioning one leg of AND 1, the 'FLT clock-0' and '-1' triggers are set to some value (normally 3; see "FLT Clock"), and the 'FLT counter-0' to '-3' triggers are set to some value (either to maximum or to the value contained in the MCW cycle count field). When the FLT counter is to be decremented, the 'scan counter control' trigger is set which, in turn, sets the 'scan counter control' latch to bring up a second leg of AND 1.

For FLT operations, the 'scan counter control' trigger is set by the 'ROS 1→CTR CTL TGR' micro-order. For ROS tests, the trigger is set when the ROS test sequencer equals 2. A log-on-count-with-address-compare operation [Diagnose instruction with MCW(1) set] also sets the trigger if the storage address agrees with the ROS address field of the MCW.

With three conditions on AND 1 met (i.e., 'scan counter control' latch set, not SOROS and sync latch, and FLT counter not equal to 0), as soon as the FLT clock is decremented to 0 (decremented once per cycle), the FLT counter contents are decremented by transferring them to the scan counter decrementer and back again. At the same time, the FLT clock is reset to 3. When the FLT clock is again decremented to 0, the FLT counter is again decremented. This operation continues until the FLT counter, including the 'FLT clock-0' and '-1' triggers, are at 0 (reset). At that time, the 'FLT counter equal 0' latch is set, which inhibits further decrementing of the FLT counter.

Cycle Counter

During pulse-mode and log-on-count operations, the address sequencer is joined with the FLT counter and the two triggers of the FLT clock to form an 11-bit cycle counter (Figure 6-9). The low-order positions of the cycle counter are decremented as described in "FLT Counter Decrementing". When the FLT counter is decremented to 0 ('FLT counter equal 0' latch set), the address sequencer contents are decremented (gated to scan counter decre-

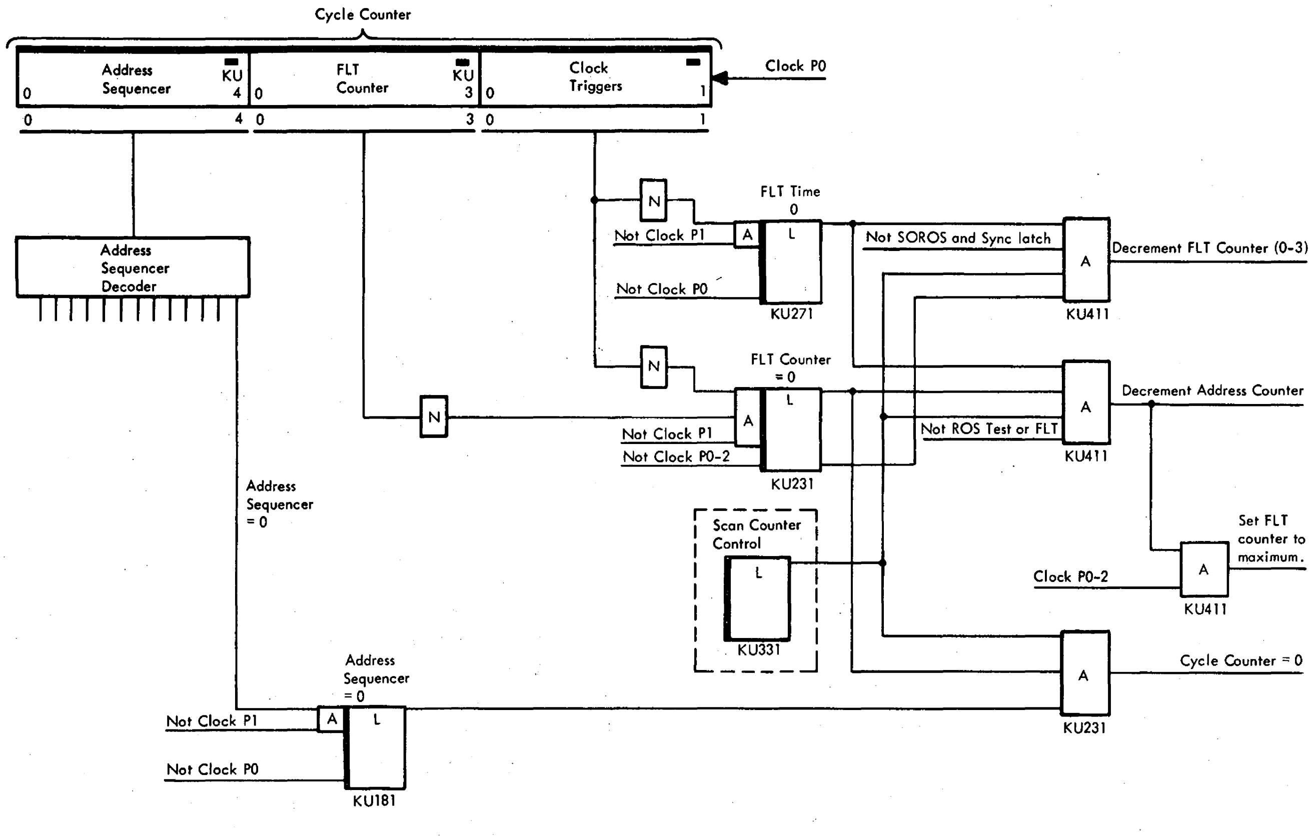


Figure 6-9. Cycle Counter Data Flow

menter and returned to address sequencer). At the same time, the 'FLT clock-0' and '-1' triggers are set, and the FLT counter is set to the maximum (all 1's). These counters are decremented as before until they reach 0 again; the address sequencer is then decremented, and the operation is repeated.

When all three counters equal 0, the 'cycle counter equals zero' signal is generated to stop the clock (pulse-mode) or to perform a logout operation.

ROS Test Sequencer

The ROS test sequencer (Figure 6-10), three polarity-hold circuits, controls the scan logic during a ROS test. At the start of the ROS test routine, the ROS test sequencer is set to a count of 7; it is decremented at FLT-time 3 by having its contents transferred to the scan counter latches and decrementer. The output is decoded by the ROS test decoder, which generates one of seven signals, depending upon the present value of the sequencer. These signals control the sequential operation of a ROS test.

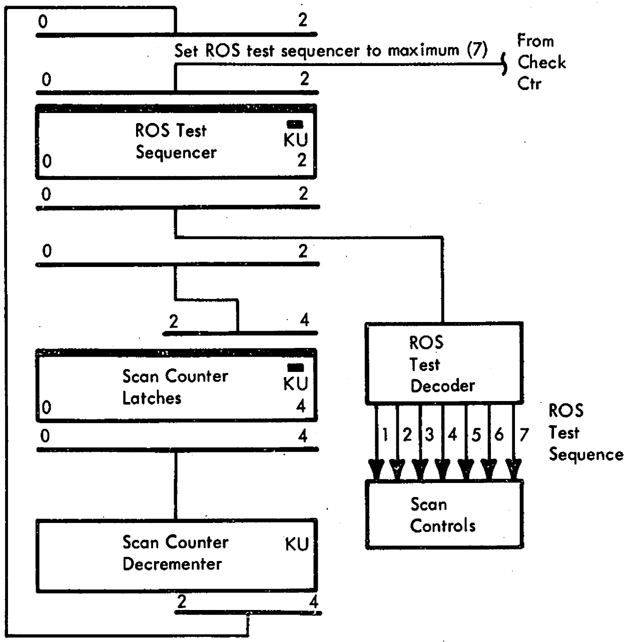


Figure 6-10. ROS Test Sequencer Data Flow

Scan-Out Bus

- Scan-out bus is data path from indicator logic to PAL.
- Status of 64 indicatable storage devices (scan-out word) is scanned out to PAL by one scan-out address.
- One half of scan-out word is transferred to PAL during one cycle.

The scan-out bus (Diagram 8-107, FEMDM) is a special data path used in scan operations that allows direct transfer into PAL from storage devices (triggers, latches, polarity-holds) which ordinarily have no direct input into PAL. The operation that performs this function is referred to as "scan out". Scan out makes use of the indicator roller switch position logic to perform this transfer. Thus, the first requirement necessary to scan out a storage device (except for the LSWR) is that the device is indicatable; that is, that is has an indicator on the system control panel roller switches.

During a scan out, the status of 64 indicatable storage devices are scanned out using one scan address (address sequencer contents); the 64 bits transferred to PAL are referred to as a scan-out word. The 22 scan-out words are listed on ALD's M3021–M3061 as logout words, because in a logout operation, the scan-out words are logged out (placed into the logout locations of main storage). Note that there is no direct correlation between the scan-out address and the roller switch position of the indicator. For example, in Diagram 8-107, scan-out address 21 is shown as selecting the same storage device as the 'roller 1 position 3' signal from the system control panel.

The indicatable storage devices to be scanned out at any one time are selected by the output of the address sequencer decoder and by a signal from the scan controls. Only one half of the scan word is transferred to PAL during one cycle; therefore the signal from the scan controls will be either 'scan out left' (selects bits 0–31 of a scan word) or 'scan out right' (selects bits 32–63). The output of the address sequencer decoder and the 'scan out left' or 'scan out right' signal places a signal on the same line as does the system control panel roller switch (Diagram 8-107). This signal allows the storage device output signal to light an indicator on the system control panel, or to be transferred to PAL via the scan logic.

At the same time, to prevent the roller switch setting from affecting the scan out, the 'not block indicator switches' signal is activated. Thus, the only indicatable storage devices whose outputs are sent to the indicator drivers are the ones selected by the address sequencer decoder output and the scan-out signal.

Gating signals route the indicator signals from the storage devices to two places: to indicators on the system control panel and to the scan-out bus. From the scan-out bus, they are transferred by the 'enable bus' signal to PAL(32–63).

Logout Controls. The controls for a logout operation are shown in Diagram 8-108, FEMDM. A logout is started by an 'error log required' signal, by depressing the LOG OUT pushbutton (in manual mode only), or by executing the Diagnose instruction. The 'SOROS' trigger sets the address sequencer to 21. The 'SOROS' trigger and the 'sync' latch request a storage cycle to store the logged out word. The FLT clock provides timing signals to control the logout operation until the address sequencer reaches 14; the CPU clock is then started and ROSAR is forced to 019 (hex) to control the rest of the logout operation.

Diagram 8-109, FEMDM, illustrates the scan-out path to PAL(54) on a logout operation. Assume that the address sequencer equals 10 (decimal). A scan out is initiated to transfer the contents of D(22) to PAL(54) if the 'scan out right' signal is generated. Normally, the 'scan out right' signal is generated by the 'SCAN OUT-RTWD' micro-order. However, on a scan-out ROS operation, it is generated by the scan controls. At the same time that the 'scan out right' signal is generated, the 'enable scan bypass' signal is produced. This signal is generated by the 'SCAN BYPASS' micro-order or, in the case of a scan-out ROS operation, by the scan controls. In addition to allowing transfer into PAL, the 'enable scan bypass' signal inhibits transfer to the system control panel indicators by the switchable indicator logic.

With the 'scan out right' signal active and the address sequencer equal to 10 (decimal), D(22) is transferred to indicator 25 of the row of indicators normally selected by roller 1 and to the scan out bus. Because the 'scan out S and T' signal is *not* up at this time, D(22) is transferred to PAL(54) by the 'enable scan bypass' signal. At the same time, the remainder of the right half of scan word 10 is transferred to the appropriate bits of PAL.

Because there are only four storage indicator roller areas and one logout area for a possible eight storage units, special provisions are made to log out the roller indicators of the failing unit. Diagram 8-110, FEMDM, shows the selection of the roller indicators of the failing unit. When an error signal is sent from storage 1, AND 1 is satisified, AND 2 is not satisfied, and the 'gate storage (1–4) indicators' signal is generated. Storage units 1–4 are conditioned to be indicated, but the storage unit check (in this case the 'storage 1 check' signal) generates the scan-out signal for the failing unit only. The storage frame indicators (always logged out) identify the particular failing unit and type of error (data or address).

Scan Out S and T. The scan out S and T operation is the method by which the mask in a ROS test or FLT selects the particular bit of the scan word to be tested. During a ROS test or FLT, the mask, which is set to all 1's except for the bit corresponding to the bit of the scan word to be tested, is placed into S(0–31), and the scan word containing the bit to be tested is placed into T(32–63). Using the scan-out bus (Diagram 8-109), the S and T contents are transferred by the 'scan out S and T' signal to a network composed of two AND's and an OR with an inverted output for each bit of S and T.

As can be seen from Diagram 8-109, the PAL bit positions remain 0's except for the position where both the input from S and the input from T are 0's. For example, if T(54) = 0 and S(22) = 0, PAL(54) is set. For any other combination of T(54) and S(22), PAL(54) remains reset. Because S contains a mask in which all bits are set to 1's except the bit that corresponds to the bit to be tested in T, all bits of PAL are always set to 0's except the bit to be tested, and that bit is also a 0 unless the scan word in T contains a 0 in the tested position.

The 'scan out S and T' signal is a result of the 'SCAN OUT S-REG' and 'SCAN OUT T-REG' micro-orders or the ROS test sequencer equalling 1 or 4.

Scan Stop-CPU-Clock Logic

During certain scan operations, the CPU clock must be stopped while the scan clock is allowed to run. This operation is controlled by the 'MMSC' trigger (Diagram 8-111, FEMDM). When this trigger is set, the unsymmetrical clock signal that controls CPU trigger and latch setting and resetting is inhibited; thus all CPU functions controlled by CPU clock signals are stopped, except as allowed by scan clock signals.

The 'MMSC' trigger is set by the following conditions:

1. Depressing LOAD with the TEST MODE, ROS/PROC/FLT switch in the ROS position.
2. Depressing RESTART FLT I/O.
3. Depressing LOAD. The system reset initiated by the IPL function resets the 'MMSC' trigger; thus, the CPU clock is restarted after the system has been reset.
4. Depressing LOG OUT.
5. Certain errors occur with the CPU CHECK switch in the STOP position.
6. Cycle counter equals 0 on a pulse-mode or log-on-count operation.
7. FLT counter steps to 0 during a ROS test or FLT.

Control Triggers

The scan logic contains many triggers which are used for status and control. The most important of these, with their function, are:

1. 'Restart I/O' (Diagram 8-112, FEMDM). Set whenever RESTART FLT I/O is depressed with the TEST MODE, ROS/PROC/FLT switch in either the ROS or FLT position. When set, this trigger causes the ROS or FLT tape to be backspaced to the beginning of the record. The buffers are then refilled by reading in each test until the test specified by the ATN is scanned into the CPU.
2. 'Scan mode'. Set when the scan controls are under microprogram control. The ROS fields used by the scan microprogram are shared by the non-scan functions of the CPU. They are interpreted as scan micro-orders if the 'scan mode' trigger is set.
3. 'Sync' (Diagram 8-112). Starts a ROS test or an FLT after an IPL or restart-I/O operation by synchronizing these operations with the scan controls.
4. 'Repeat'. Set by the RESTART FLT I/O pushbutton. When set, the ROS test or FLT currently running is repeated continuously.

5. 'FLT test' (Diagram 8-112). Set by the TEST MODE, ROS/PROC/FLT switch being in the FLT position. It puts the CPU into FLT mode.
6. 'Scan out ROS' (SOROS) (Diagram 8-108). Set by a machine error when in log-on-error mode, by the LOG OUT pushbutton, or by the 'cycle counter = 0' signal when performing a log-on-count operation. Initiates the scan-out portion of a logout operation.
7. 'Pass', 'fail', 'intermittent' (Diagram 8-112). Store the results of a ROS test or FLT. MCW(7) is the ERSLT bit, and PAL equals 0 if the bit to be tested in the scan word is a 1. Thus, if MCW(7) equals 1 and PAL equals 0, or if MCW(7) equals 0 and PAL is not equal to 0, the 'pass' trigger is set. If MCW(7) equals 1 and PAL is not equal to 0, or if MCW(7) equals 0 and PAL is equal to 0, the 'fail' trigger is set. For FLT's, the test is performed under microprogram control. For ROS tests, the test is performed when the ROS test sequencer equals 1. In either case, if both the 'pass' and 'fail' triggers are set, the 'intermittent' trigger is set.

Scan Mode Control of ROS

Scan mode operations affect three fields of ROSDR: field D (bits 17–19), field F (bits 25–30), and field G (bits 31–35). These fields serve dual functions. In the normal mode, they are decoded from the ROSDR latches as standard CPU control lines. In scan mode, they are decoded as special scan control lines and are referred to as field S.

The scan mode is controlled by the 'scan mode' trigger. When the 'scan mode' trigger is reset, the standard decode path is used. When the 'scan mode' trigger is set, however, the standard control lines are blocked and scan control lines (using common CPU control line codes) are activated.

The 'scan mode' trigger can only be set in normal CPU mode and reset only in scan mode. The scan control logic generates blocking signals to inhibit register gating signals at the ROSDR decode logic and to allow scan control use of ROS in sequencing through its test operations.

Scan logic also affects ROS microbranching (Diagram 8-113, FEMDM). The J-field micro-orders shown in the diagram are listed on ALD M7021.

Scan/Channel Interface

Because ROS tests and FLT's are read in from an I/O device (tape or disk), special interface lines (Diagram 8-114, FEMDM) are necessary for communication between the channel and the scan logic. A brief description of these lines follows:

1. 'TIC pulse'. A multiplex line from the channels to the scan controls that carries a 250 ±30-ns signal when a 'TIC' command is encountered after the 'IPL' latch is reset. This signal indicates a buffer has been filled.
2. 'Gap pulse'. A multiplex line from the channels to the scan controls that carries a minimum 1-usec signal whenever an end of record is passed. This signal does not occur if an error was detected in the record.
3. 'Unit data error'. A multiplex line from the channels to the scan controls that signals a data error from the time of discovery until a restart is initiated.
4. 'Channel control error'. A multiplex line from the channels to the scan controls that rises on a control error, command reject, or exceptional condition, and signals that the channel is unable to proceed.
5. 'Stop scan'. A multiplex line from the scan controls to the channels. This line carries a 250 ±50-ns signal that commands the operating channel to stop transmission to storage and wait. The tape or disk proceeds to the end of record and is deselected.
6. 'Start scan'. A multiplex line from the scan controls to the channels. This line carries a 250 ±50-ns signal that commands the selected channel to restart transmission of data to storage. The channel fetches the command at location 80 (hex) and continues. This action causes the restart to begin always at the front of the block it was transmitting at the time the 'stop scan' signal was received. If the 'stop scan' signal is received during an error backspace, the backspace is concluded and the ensuing 'start scan' signal initiates the read command that loads buffer 1.
7. 'Scan mode'. A multiplex line that indicates to the channel that the CPU scan controls are operative. It causes the channel to go into FLT mode when it receives the 'IPL' signal.

Operational Analysis

The following paragraphs present a detailed operational analysis of logout, ROS tests, and FLT's.

Logout

- Logout stores 22 doublewords, which reflect CPU status, into main storage locations 80–128 (hex).
- Operation is controlled by scan logic and ROS microprogram.
- Address sequencer is set to 21 and decremented by 1 for each log word stored.

The logout function, whether initiated by depressing the LOG OUT pushbutton, by executing the Diagnose instruction, or by detecting a machine check, stores 22 doublewords (log words), which reflect the CPU status, into main storage locations 80–128 (hex). The information presented in these log words is determined by the scan logic and is therefore fixed at a specified engineering level for any particular 2065 CPU. A list of the log-word locations and their contents is presented in ALD's M3021–M3061.

The words are logged out in a fixed sequence, as defined by the scan logic, with log word 21 being stored into location 128 (hex), log word 20 into location 120, and so on in reverse order until log word 0 is stored into location 80 (hex).

The logout operation is both hardware- and ROS-controlled. When a logout operation is initiated, for example by depressing the LOG OUT pushbutton†, the address sequencer is set to 21 (23 on a storage logout) and is decremented by 1 for each log word stored. From the instant that LOG OUT is depressed until the address sequencer is reduced to 14, the scan hardware controls the operation. When the address sequencer equals 13, control is switched to a ROS microprogram which completes the logout.

Although the CPU is under ROS control for the last 14 words logged out, the address sequencer is still decremented for each log word. This decrementing occurs because the address sequencer, in addition to controlling the logout sequence, defines the storage address for each log word and the indicatable storage devices to be logged out.

The log words compose the status of most indicatable storage devices (i.e., triggers, latches, and registers that have an indicator on the roller switches). For this reason, the scan-out bus, which is a data path from the indicator drivers to PAL, is used to transfer the status of these devices to PAL (the path used is identical to that used in an FLT or ROS test during a scan out; see "Scan-Out Bus"). From PAL, the log words are gated to ST, and then to main storage via the SDBI. Correct parity is assigned in PAL.

Because cycling of the CPU clock could change the contents of some of the storage devices logged out, the CPU clock is turned off (blocked) while log words 21 through 14 are being logged out, and timing is controlled by the scan and FLT clocks. Scan/FLT clock signals are distributed throughout the scan logic to control the logout function during the period that the CPU clock is turned off. For example, the address sequencer is decremented at FLT-time 3, a latched output of 190-ns duration.

At the end of a logout, the CPU performs an end op, and the 'machine check interrupt' trigger is set.

If the logout is caused by a storage error, all storage units are disabled because the CPU cannot determine which storage had the error. Therefore, no storage unit can be recycled until the contents of its indicators have been saved. As the first step in saving the indicators, the address sequencer is set to 23. The two extra counts of the address sequencer provide a delay which is necessary because of the cable delay and of the relatively slow indicator drivers in storage and the CPU. When the sequencer reaches 21, the storage units are again made available, and the contents of the indicators displaying information pertaining to the faulty main storage are transferred to PAL and held in PAL(32–63). The remainder of the logout sequence is the same as for a machine check logout.

Hardware-Controlled Sequence. Highlights:

- Logout is initiated by Diagnose instruction, LOG OUT pushbutton, or machine error.
- 'SOROS' trigger initiates hardware-controlled portion of logout sequence.
- For each word logged out: (1) address sequencer is decremented, (2) scan-out logic places right half of log word into T, and (3) log word is stored into main storage location addressed by storage address generator.
- Only right half of log words 20–14 contain data.

As stated previously, a logout (Diagram 8-115, FEMDM) may be initiated by executing the Diagnose instruction, by depressing LOG OUT, or by detecting a machine error when in log-on-error mode. The operational differences between the three methods occur before the actual logout sequence and are as follows:

1. The Diagnose instruction initiates a logout operation if MCW(6) = 1, thus specifying a log-on-count operation. In this case, MCW(21–31) is sent to the cycle counter, which is decremented by 1 each machine cycle. When the cycle counter equals 0, the 'SOROS' trigger is set, initiating a logout sequence.
2. If the logout operation is initiated by depressing LOG OUT with the CPU in manual mode, the 'console logout' latch and the 'pass pulse' trigger are set. Normally, the 'pass pulse' trigger will already be set. However, if the RATE switch is in the SINGLE CYCLE position, the 'pass pulse' trigger remains set only for the duration of one CPU clock signal (Diagram 4-3, FEMDM). This action allows the logout operation to be stepped through one cycle at a time by depressing START for each cycle. With the 'pass pulse' trigger and 'console logout' latch both set, the 'SOROS' trigger is set.
3. When the CPU CHECK switch is in the PROC position and the PSW machine check mask bit is a 1, a machine check ('error' trigger set) initiates a logout. Any of the following errors sets the 'error' trigger (which, in turn, sets the 'SOROS' trigger):
   a. CPU/storage error.
   b. Serial adder half-sum or full-sum error.
   c. Parallel adder half-sum or full-sum error.
   d. ROS error.
   e. E(0–15) parity error.
   f. MPR bus parity error.

† Note that the LOG OUT pushbutton can initiate a logout only when the CPU is in the manual mode.

When a storage check is detected by the processor, the following sequences occur:

a. The BCU resets any CPU request triggers that might have just been set. If the request triggers were set early enough to have sent a 'select' signal to storage, they are reset in a normal fashion and are, therefore, not reset by the storage check sequence.
b. All storages are made to look busy by blocking the set input to the 'not busy' triggers. This action prevents any channel from selecting the faulty storage unit.
c. The CPU 'stop clock' trigger is held reset. Because the storage check is asynchronous to any other BCU activity, this trigger might have been set by some other CPU 'select' signal.

All of these conditions persist until the scan logic signals release to the BCU. At the same time, the address sequencer is set to 23 instead of the normal 21. This setting allows a delay of approximately 1.6 usec until the count is reduced to 21. A 'scan-out storage X' line is activated to transfer the contents of the storage indicators to PAL(32–63). During this delay, the address or data check indicator stabilizes (worst case is 1 usec) and provides the proper gating signal level for its respective roller switch; e.g., storage 1 is roller 1, position 5; storage 4 is roller 6, position 5. When the address sequencer equals 21, the contents of the proper roller switch is transferred to PAL. Just before dropping the gating signal level, a 'hold paddl 32–63' line is activated to allow the input gates to be deactivated without losing the storage indicator information. At this point, the BCU is released by the scan logic. Any channel requests that were held pending in the BCU start being serviced in a normal manner, and a scan storage request for storing ST is issued. The remainder of the logout sequence is identical to a machine check logout sequence, except for the gate PAL to T and the set marks functions when the address sequencer equals 20. (Normally, these signals are blocked.)

Note that, in each case, the 'SOROS' trigger is set, thus starting the hardware portion of the logout sequence (Diagram 8-115). After the 'SOROS' trigger is set, the 'sync' trigger is set at FLT-time 3 to synchronize the logout operation to the FLT clock. The output of the 'sync' trigger sets the address sequencer to 21 (first log word), and sets the 'sync' latch at not-clock time. (The 'sync' latch being set causes the address sequencer to be decremented every FLT-time 3.)

With the 'sync' latch set, a storage request to the BCU is initiated at FLT-time 0. This request gates the address generated by the storage address generator to SAB. Because the first word to be logged out reflects the contents of ST (log word 21), the contents of ST are now stored (gated to SDBI).

When the 'sync' trigger was set, the 'MMSC' trigger was also set. This action keeps the CPU clock off (the 'error' trigger inhibited CPU clock signals) but allows the scan clock to run. Therefore, the status of all operational registers, except the registers used for the logout (S, T, and PAL), is preserved during the logout sequence.

Because the contents of ST have been stored, the address sequencer is decremented to 20 at FLT-time 3. A scan-out operation is performed, which gates the right half of log word 20 (selected by the address sequencer decoder) to PAL(32–63) from the indicatable storage devices via the scan-out bus. From PAL, log word 20 is gated to T for transfer to main storage. At FLT-time 0, another storage request is issued to store the contents of ST. This operation continues for log words 19 through 14; that is, for each word logged out: (1) the address sequencer is decremented, (2) a scan out places the right half of the log word into PAL, which is subsequently gated to T, and (3) the log word is stored into the main storage location addressed by the storage address generator. Note that only the right half of log words 20 through 14 is stored; the left half in storage remains unchanged and could contain anything.

When the address sequencer has been decremented to 14 and the log word has been stored, the 'MMSC' trigger is reset to start the CPU clock, address 19 is forced into ROSAR, and the address sequencer is reduced to 13. (Note that the ROSAR contents have been already logged out.) At this point, control is transferred to a ROS microprogram.

ROS-Controlled Sequence. Highlights:

- ROS microprogram directs logout from time address sequencer equals 13 until end of logout sequence.
- Log word 13 is formed by: (1) scanning out to T, (2) transferring T to LSWR, (3) transferring S to T, (4) transferring LSWR to S.
- Remaining words are formed by: (1) decrementing address sequencer by 1, (2) scanning out to T, (3) transferring T to LSWR, (4) transferring S to T, (5) transferring LSWR to S, (6) storing log word into storage.
- When address sequencer equals 0, parity is corrected on log word 21 (ST).

The ROS microprogram directs the logout operation from the time the address sequencer equals 13 until the end of the logout sequence (Diagram 8-115). The first microorder in the logout microprogram transfers the contents of the LSWR to S, thus preserving the LSWR contents in a log word because the LSWR is used during the remainder of the operation. S now contains half of log word 13. The complete log word is formed by: (1) scanning to T (scan out to PAL whose contents are transferred to T), (2) transferring the data in T to the LSWR, (3) transferring the contents of S to T, and (4) transferring the contents of the LSWR to S. As a result, ST contains the 13th log word. The operation continues in the following cycle:

1. Decrement address sequencer by 1.
2. Scan out to T (left half).
3. Transfer contents of T to LSWR.
4. Scan out to T (right half).
5. Transfer contents of LSWR to S.
6. Store word into main storage.

This cycle is repeated until the address sequencer equals 0, at which time 22 log words have been stored into main storage locations 80–128 (hex).

When the logout operation was started, the first word stored into main storage (log word 21) was composed of the contents of ST. The parity associated with this data is not always correct. Therefore, to insure that correct parity is stored with log word 21, the address sequencer is set to 21, and a scan storage request is initiated to fetch the original contents of ST in log word 21. When available on the SDBO, the word is gated to AB. The address sequencer is then set to 7, and AB (scan address 7) is scanned out. When AB is scanned out, correct parity is inserted for the data (original ST contents). The address sequencer is again set to 21, and a scan storage request is initiated to put log word 21 back into main storage with correct parity. Next, the address sequencer is reset to 7 and repeatedly decremented by 1 until it equals 0, at which time the original ST parity bits are stored. The scan system is then reset, the 'machine check interrupt' trigger is set, and the operation concludes at end op.

ROS Tests

- ROS tests consist of two doublewords: word 0 has TN and ATN; word 1 has mask and MCW.
- All bits of ROS bit planes are checked for 1 or 0.
- Each ROS test is repeated until receipt of 'TIC' pulse.

ROS tests are scan-controlled tests of the ROS microprogram. Each ROS test consists of two doublewords, designated words 0 and 1, which are read into the CPU from the ROS test tape. Word 0 contains the TN and ATN, and word 1 contains a mask and the MCW. A single ROS test checks one bit position of one ROS word and the cycling of data from ROS to the ROSDR.

Each ROS test tape begins with hardcore tests to check out the hardware that controls subsequent ROS tests. Upon successful completion of these hardcore tests, the true ROS tests are begun and continue until all bits of the ROS bit planes have been checked for a 1 or 0, or until an error is encountered. If an error occurs, the CPU stops and the failing bit number is displayed in S. Thus, no documentation is required.

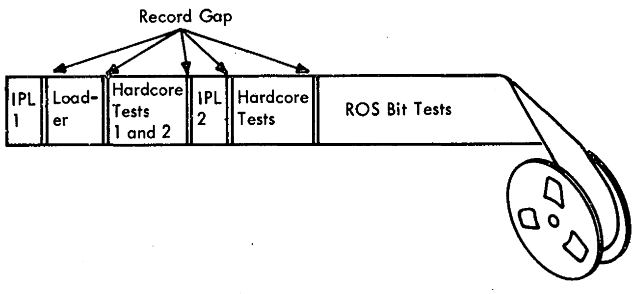
The tests are loaded into buffer areas 1 and 2, and 'TIC' pulses are generated as the buffers are filled. Each ROS test is continuously repeated until the CPU receives the 'TIC' pulse.

A single test consists of fetching the MCW from storage, permitting the CPU clock to advance a given number of cycles, and comparing one of the ROSDR triggers with an expected result. If the state of the trigger matches its predicted value, the 'pass' trigger is set and testing proceeds to the next sequential test. If the actual and predicted values disagree, the 'fail' trigger is set and testing is terminated.

The ROS tests are controlled by the ROS test sequencer, which is stepped by the scan clock. At the start of the ROS test, the ROS test sequencer is set to 7 and is decremented to 0 as the test progresses to conclusion. For each count of the test sequencer, a certain part of the ROS test is performed.

The number of clock signals required to move the selected ROS word into the ROSDR is specified in the cycle count field of the MCW. This count is set into the FLT counter; when the counter equals 0, the CPU clock is stopped and the bit comparison begins.

ROS Test Tape. The ROS test tape contains the following records:



1. Record 1, IPL 1. Contains the 24-byte "bootstrap" program necessary for any IPL operation. When LOAD is depressed, the three doublewords of this record are read into storage locations, as follows:

Storage Location (Hex)	Word	Contents
0	PSW	Backspace (not used).
8	CCW 1	Read command to read 40 bytes to location 80, and chain command to location 10.
10	CCW 2	TIC command to location 88.

2. Record 2, Loader. Contains the "loader" program that reads in the ROS tests. The IPL program in record 1 reads this record into storage locations as follows:

Storage Location (Hex)	Word	Contents
80	CCW	Backspace command.
88	CCW	Read command to read 16 (decimal) bytes to buffer 1 (location 8000, hex), and a chain data tag to location 90.
90	CCW	TIC command to location 98.
98	CCW	Read command to read 16 bytes to buffer 2 (location 8080), and a chain data tag to location A0.
A0	CCW	TIC command to location 88.

3. Record 3, Hardcore Test 1 and 2. Contains the first ROS hardcore test, two doublewords. Hardcore tests check the scan and CPU hardware required to do the actual ROS testing. Any failure encountered during the hardcore tests must be corrected before the actual tests have validity.
4. Record 4, IPL 2. The first hardcore test causes a stop. When LOAD is depressed, the following IPL program replaces the IPL 1 program:

Storage Location (Hex)	Word	Contents
0	PSW	Backspace record.
8	CCW	TIC command to location 88 (loader program).

5. Records 5–8, Hardcore Tests. Contain the remaining hardcore tests.
6. ROS Bit Tests. The remaining records on the tape contain the true ROS tests. Each test pattern (two doublewords) tests one bit of one ROS word.

ROS Test Setup. Several controls on the system control panel must be operated to initiate the ROS tests. The procedure to run a ROS test appears in *2065 Processing Unit FEMM*, Form Y27-2039-2 and in ALD M8005. However, a short discussion of the setup is included here because it affects the operation.

Diagram 8-116, Sheet 2, FEMDM, shows the start of a ROS test. The ROS test tape is mounted first. The LOAD UNIT switches are set to the address of the tape unit holding the test tape, the TEST MODE, ROS/PROC/FLT switch is set to the ROS position, and the CPU CHECK switch is set to DSBL. Going into ROS test mode causes the 'ROS test initiated' latch to be set and a 'scan mode' signal to be sent to the channel to prepare for a read operation. The manual control operations necessary to get started are concluded by depressing SYSTEM RESET, loading all DATA switches to 1's, depressing STORE to transfer the DATA switches to ST, and depressing LOAD.

IPL 1. Highlights:

- IPL is under hardware control.
- Channel operations are same as normal IPL.
- CPU clock is stopped until release is received from channel.
- 40 bytes are read into storage, starting at location 80 (hex).

Depressing LOAD with the TEST MODE, ROS/PROC/FLT switch set to ROS initiates an IPL operation that is different from the normal program load in that the operation is under hardware control (Diagram 8-8, FEMDM). Channel operations are identical to a normal IPL operation; 24 bytes are read from the selected device into the first three doubleword locations of main storage. However, when the 'IPL status' trigger is set, the 'MMSC' trigger is also set, which stops the CPU clock. Because the CPU clock is stopped, the IPL microprogram is not initiated and the remainder of the IPL is under scan and channel control.

At this point, the CPU is idle, waiting for a 'release CPU' signal from the channel; only the scan clock is running. Meanwhile, the channel IPL operation reads in record 1 of the ROS test tape, and executes the channel program specified by record 1. As a result, 40 bytes (record 2) are read into storage starting at location 80 (hex). Record 2 contains the loader program that reads each ROS test into the proper buffer area in storage.

After record 2 has been read in, command chaining causes CCW 2 in record 1 to be executed. CCW 2 is a TIC command to location 88, which now contains a read CCW. When the channel has finished executing the TIC command, it sends a 'release' signal to the CPU.

Because the CPU is in the ROS test condition, receipt of the 'release CPU' signal from the channel causes the 'timing gate' trigger to be reset. With the 'timing gate' trigger reset, the 'release CPU' latch is reset and the 'IPL status' latch is set. The 'sync' trigger and 'sync' latch are set to synchronize the operation to the FLT clock. In addition, the 'sync' trigger output resets the ROS sense latches and inhibits register ingating. At this point, channel selection is dropped and a scan reset is initiated. The scan controls are set up for operation by resetting the 'MMSC' trigger, setting the 'pass' trigger, resetting the 'fail' trigger, and setting the check counter to 31 and the ROS test sequencer to 7. The CPU is now ready to run the first hardcore tests.

Loader. While the scan controls are being set up to run the first hardcore test, the TIC command in CCW 2 of the IPL 1 program causes the channel program in record 2 of the ROS test tape [now in the 40 byte locations of storage beginning at 80 (hex)] to be executed. The read CCW's in locations 88 and 98 cause the channel to read the hardcore tests in record 3 into buffers 1 and 2 (8000 and 8080, respectively).

Meanwhile, scan control operations in the CPU begin when the ROS test sequencer is set to maximum, thus placing the CPU into ROS test state 7. In state 7, the 'buffer 1' trigger is reset, the 'ROS test' trigger is set, and a scan system reset clears the scan IPL controls. The ROS test sequencer is decremented by 1. During ROS test state 6, the 'sync' latch is reset, an address is forced to storage controls, and a scan storage request for the ROS test word containing the TN and the ATN word is initiated. The operation then waits for a 'TIC' pulse before progressing. This 'TIC' pulse results when the first hardcore test in record 3 has been read into storage from the channel (execution of TIC CCW in location 90). After the 'TIC' pulse, the ROS test sequencer is decremented by 1.

At this time, an ATN search is made. The test number, all 1's, from the test buffer is transferred to the S-register, and a successful search should result.

Hardcore Tests and IPL 2. Highlights:

- Eight records of hardcore tests at beginning of FLT and ROS test tapes determine that CPU hardware used to run tests is functioning.
- Failing test stops, with failing bit pattern in S.
- If hardcore tests run successfully, testing terminates on correct stop; when testing is resumed, CPU enters true ROS tests or FLT zero-cycle tests.

Every FLT and ROS test tape begins with hardcore tests (see ALD M8006) to determine that the CPU hardware used in running these tests is functioning properly. These tests are almost identical for ROS tests and FLT's; differences will be pointed out. Because the CPU hardware most involved in running FLT's or ROS tests is S, T, PAL, and the connecting paths between these points, these are the logic areas checked by the hardcore tests.

Eight records of tests are involved with hardcore testing. Records 1, 2, and 4 load the CPU for testing; the others check the CPU for various functions or capabilities. These functions include sensing for 1's or 0's in S and T, verifying the CPU's ability to stop when the stop conditions are met, and verifying the ability of the CPU to conduct an ATN search and sequential testing. A failing test stops testing, with the failing test bit identification pattern displayed in S and T. If all hardcore tests are run successfully, testing terminates on a hardcore stop (correct stop); when testing is restarted, the CPU enters the true ROS tests or, in the case of FLT's, the zero-cycle tests.

Because the ROS hardcore tests are sequenced by the same controls that sequence the ROS tests and because the FLT hardcore tests are executed in the same manner as the regular FLT's, the hardcore tests have a different format for ROS tests and FLT's. ROS hardcore tests have the following format:

		Buffer 1	Buffer 2
TN	ATN	8000	8080
Mask	MCW	8008	8088

Each FLT hardcore test contains the nine or ten words used in an FLT; however, only two of the words are significant to the test. These are words 8 or 9, and 1, the TN/ATN and the MCW, respectively. The TN/ATN causes the CPU to progress through the tape records. The MCW causes the CPU to make decisions. The other seven words read in during a hardcore test contain 0's and contribute nothing to the test.

Hardcore tests check S, T, and PAL by performing the TN/ATN comparison. The TN is in S, and the ATN from the previously executed test is in T. The corresponding bits in each register are compared at the scan-out-bus OR. If either bit is a 1, the inverted output of the OR is sensed as a 0.

For the result comparison, the mask is brought to S and compared with the value scanned into T during execution of the test. In hardcore tests, the mask is all 1's or all 0's. This condition forces a pass or fail condition regardless of the value scanned into T during execution. A 0 output is sensed if the mask is all 1's; a positive output is sensed if the mask is all 0's. Using the mask in this manner, in conjunction with the MCW, causes the CPU to decide whether to take the next test, an alternate test, or to terminate testing.

The significant bits of the MCW used during hardcore testing are 5, 6, and 7. MCW(5) is the UT bit, and, if set, causes the CPU to stop after the test, regardless of the outcome of the test. MCW(6) is the CT bit, and, if set, causes the CPU to stop if the test fails or to take the next test if the test passes. MCW(7) is the ERSLT bit, which specifies whether a 0 or a 1 should be sensed at PAL following the result-comparison portion of the test. The combinations of the ERSLT and the output at PAL that determine the setting of the 'pass' or 'fail' trigger are:

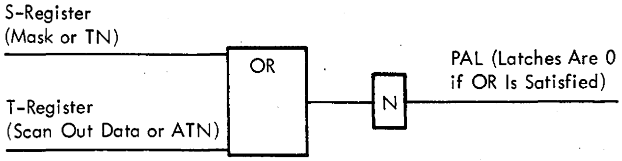
ERSLT Bit	PAL = 0	Set 'Pass' Tgr	Set 'Fail' Tgr
0	Yes	–	Yes
0	No	Yes	–
1	Yes	Yes	
1	No	–	Yes

The setting of the 'pass' or 'fail' trigger is then compared with the setting of MCW(5, 6) to determine the next operation, as follows:

Trigger Output		'Pass'	'Fail'	
MCW(5) (UT)	MCW(6) (CT)	Tgr	Tgr	Action
0	0	1	0	Continue – alternate test
0	1	1	0	Continue – next test
0	0	0	1	Continue – next test
0	1	0	1	Stop – gate alternate test on restart
1	0	1	0	Stop – gate alternate test on restart
1	0	0	1	Stop – gate alternate test on restart

Whenever the CPU stops during hardcore tests (whether an error stop or an unconditional stop), the ATN is left in T. When the test is restarted, the tape backspaces to the beginning of the test record and begins reading into storage. As each test is brought in, the CPU brings the TN of each sequential test into S. S and T are then compared, and, when PAL equals 0, the CPU performs the test that is in the buffer at that time.

One of the preliminary steps in preparing the CPU to perform ROS tests or FLT's is to set S and T to all 1's by setting the DATA switches to the down position and depressing STORE (see ALD M8005-1). This step is required because it is not known initially that these registers are functioning. By forcing 1's into all positions, the CPU is forced to take the first test, which is test 1 on record 3. The rationale is that a failing bit in one register is compensated for by the bit in the other register, so that when S and T are compared, the result is 0 and the test is taken. Shown below is a single position of the scan-out bus; a similar position exists for each bit in S and T.



Note that a 1 input to either side of the OR satisfies the OR condition, and the inverted output is sensed as a 0. If neither input is a 1, the OR is not satisfied, and the inverted output is sensed as a 1.

Test 1, Record 3. Highlights:

- Determines that 1's in S can be sensed as 0's at PAL.
- If test passes, CPU stops with 'pass' trigger set.
- If test fails, CPU stops with 'fail' trigger set or does not stop.
- Depressing RESTART FLT I/O on passing test repeats test. If restart is initiated on failing test, "runaway" tape may occur.

Note: This test must be taken because of setting S and T manually (assuming data failures only).

This test is the first of the hardcore series and has the following format:

Mask (S)	MCW (T)	TN (S)	ATN (T)
FF FF FF FF	05 00 00 00	FF FF FF FF	00 00 00 00

The MCW defines the conditions of the test, and in this instance the MCW has the configuration, in the first eight bits, of 0000 0101. With bits 5 and 7 set to 1's, the conditions of the test are that the test must terminate unconditionally (bit 5) and the ERSLT bit (bit 7) is a 1; therefore, the output of PAL, following the result-comparison, should be sensed as 0. This result means that the mask of all 1's in S should be sensed as all 0's if S and the paths through the PAL are functioning properly. This is the purpose of this first test – to determine that 1's in S can be sensed as 0's at PAL, and that an unconditional termination stops testing.

If this first test passes, the CPU stops with the 'pass' trigger set. Should the test fail, the CPU stops with the 'fail' trigger set. If the CPU does not stop, a malfunction exists in some other portion of the machine, an ATN was not performed successfully (could not sense 1's in S *or* T), or the CPU failed to act on the UT bit.

The correct indications for the successful passing of test 1 are:

1. S all 1's (roller 1, position 3).
2. T all 0's (roller 2, position 3).
3. 'Pass' trigger set (roller 5, position 2, bit 29).
4. UT bit set (roller 5, position 2, bit 5).
5. ERSLT bit set (roller 5, position 2, bit 7).
6. 'Buffer 1' trigger set (roller 5, position 2, bit 35).

When RESTART FLT I/O is depressed after a passing test, the test is repeated and again stops because the TN is all 1's. If a restart is initiated after a failing test, a "runaway" tape may occur. (A runaway tape condition occurs because the TN search routine keeps searching the tape for a nonexistent TN.) The indication of this condition is that S continues incrementing as test numbers are brought in for comparison. The failure is in S, in that one bit is being sensed as a 1 (0 in S). This runaway condition should result anytime the restart sequence is initiated following a failure indication on test 1.

On a failure indication or tape runaway on restart, T can be used to compensate for the failing bit in S by performing the following procedure:

Note: This procedure applies to the FLT tape only.

1. Reset (place in up position) DATA switches 32–63 (T), one at a time, keeping all other switches set (down). Depress STORE and RESTART FLT I/O in that order.
2. Repeat step 1 until resetting one of the DATA switches causes the tape to run away. The bit in S corresponding to the up switch in T is the failing bit, and is not being recognized as a 0 in PAL.

Analysis of the above procedure reveals that the DATA switches, by setting T to 1's, is compensating for the bit in S that is not being sensed as a 0 at PAL. The test fails, but the CPU stops until the bit in T, corresponding to the failing bit in S, is 0, at which time both inputs to the OR are 0 and the tape runs away. Assume the test fails when DATA switch 37 is up (reset). This condition means that bit 5 of S is failing.

Because the TN/ATN comparison and the result-comparison tests in this hardcore test are identical, a failing test may be repeated for scoping during the result-comparison portion of the test. The recommended procedure after the failing S-bit has been determined follows:

1. Restart from the beginning with all DATA switches set. At the first stop, set the TEST MODE, REPEAT switch and depress START. Scope the suspected S-bit at the time it is used for the result-comparison test (not TN/ATN compare). S should contain an all 1's mask for the result-comparison.
2. While synchronizing the scope on the 'scan out S and T' signal, scope the OR's shown on the left side of ALD AP691 for the suspected S-bit.

A tape runaway can continue because of failure of the UT bit. If the tape runaway continues when the above sequence is performed, the single-cycle routine detailed in *2065 Processing Unit FEMM*, form Y27-2039-2, should be followed.

If test 1 passes, test 2 must be taken on the second IPL.

IPL 2. Depressing LOAD after the stop at the end of the hardcore tests in record 3 reads in record 4, which contains the IPL 2 channel program. (See "ROS Test Tape".) The IPL 2 channel program replaces IPL 1 and contains a TIC CCW at location 8. Executing this TIC causes a branch to the read CCW in location 88 (loader), which, in turn, refills the buffer 1 area of storage from the next tape record.

The next CCW is a TIC to location 98, which contains a read CCW that fills buffer 2. At the same time that buffer 2 is being filled, the hardcore test (test 2, record 5) in buffer 1 is executed.

Test 2, Record 5. Test number 2 in the hardcore sequence determines whether 1's in T can be sensed as 0's at PAL. This test also leads into record 6. The format of test 2 is:

Mask (S)	MCW (T)	TN (S)	ATN (T)
FF FF FF FF	01 00 00 00	FF FF FF FF	FF FF FF FF

The test is taken because the TN is all 1's (S equals 0 at PAL). The MCW sets up the condition that the ERSLT bit is a 1, and, because there is no termination bit, an ATN search is performed. The mask of all 1's causes the test to pass unconditionally. (Test 1 determined that S is functioning properly.)

Following the test, the CPU immediately begins execution of the first test of record 6, as determined by the TN/ATN search (all 1's in T and all 0's in S). Recall that on a TN search, the 32-bit TN of each test in sequence is put into S to be compared with the 32 bits left in T by the previously executed test.

If the next test to be executed is not test 1 in record 6, a 1-bit in T is not being sensed at PAL. Should this occur, the failing bit results in a configuration in S that causes the CPU to do a successful TN search. The CPU stops at the test in record 6 bearing the bit configuration that agrees with the failing bit in T. The possible configurations are listed in ALD M8006-1.

To summarize, test 2 should cause the CPU to execute the first test in record 6. A failing bit in T, however, causes the CPU to execute an alternate test in record 6 and then stop, displaying a bit in S. The corresponding bit in T is the bit that is failing. The recommended procedure following a hardcore stop at this point is discussed next.

Record 6. The CPU stops in record 6 only if the ATN from test 2 cannot be sensed as all 0's. If no failures occurred in T and in the scan-out path from T to PAL, the first test in record 6 is executed and has the following format:

Mask (S)	MCW (T)	TN (S)	ATN (T)
FF FF FF FF	01 00 00 00	00 00 00 00	80 00 00 01

This test passes unconditionally because of 1's in the mask. The CPU does a TN search because the 'pass' trigger is set and no termination bit exists. The search leads the CPU to the first test in record 7 because the TN of test 1 in record 7 has the configuration 7F FF FF FE, which is the complement of the ATN of the first test in record 6.

If the ATN from test 2 cannot be sensed as all 0's, a successful test number comparison is *not* performed for test 1 of record 6. The test is *not* executed, and record 6 is searched until a test is found whose test number contains a bit corresponding to the failing bit in T. This test is executed and the CPU stops because the UT bit is set. For example, if the CPU executes the test with the TN configuration of 00 00 80 00, bit 48 in T is being sensed as a 1 at PAL. Scoping this error is difficult because the CPU does not refetch the TN during repeat. To pin-point the malfunction to a test that can be repeated and scoped, use the following procedure (FLT's only):

1. Put all 0's into DATA switches 0–63 on the system control panel. Depress STORE and RESTART FLT I/O. This procedure should bring the CPU to the second correct hardcore stop in record 8 (test 3).
2. Depress RESTART FLT I/O again. This action should cause the CPU to begin the zero-cycle tests. A failing test should be encountered in the zero-cycle test sequence that tests the T-bit that failed when running record 6. This zero-cycle test may be repeated and the failing points scoped to isolate the malfunction.

Note: If tape runaway is encountered during this routine, bit 63 or 55 in T is in error, and the TN search (which involves these bit positions) to start the zero-cycle tests is unsuccessful. Should this event happen, return the CPU to the second hardcore stop and repeat the procedure, changing the ATN of the second hardcore stop to 102 or 201.

If no error is encountered, the first test of record 6 will lead into the first test of record 7.

Test 1, Record 7. The first test in record 7 normally follows the successful testing of test 1 in record 6. The record 7 test places a 0 into each bit position in T and verifies that the 0 can be sensed as a 1 at PAL; the procedure is then repeated for S. Each test checks one bit position only. Error indications are the same for either register. Error stops are listed in ALD M8006-2. If all tests in record 7 are run successfully, the CPU begins the tests on record 8.

Test 1, Record 8. Test 1 in record 8 determines the ability of the CPU to take the next test rather than do a TN search, as was done in the previous records. The format of test 1 is:

Mask (S)	MCW (T)	TN (S)	ATN (T)
FF FF FF FF	03 00 00 00	22 22 22 22	44 44 44 44

The MCW, with a configuration of 0000 0011 (bits 6 and 7 set), determines the ability of the CPU to take the next test in sequence rather than search for the alternate test. The 'pass' trigger is set by the mask being all 1's and the ERSLT bit equal to 1. With the 'pass' trigger set and the CT bit equal to 1, the next test is executed without a TN search. If the CPU cannot force the next test under these conditions, it stops when it reaches test 5 of record 8. Assuming that test 1 passes, the CPU executes the next test, which is test 2 in record 8.

Test 2, Record 8. Test 2 in record 8 tests the ability of the CPU to take the next test on a failing condition. The format of test 2 is:

Mask (S)	MCW (T)	TN (S)	ATN (T)
00 00 00 00	01 00 00 00	55 55 55 55	33 33 33 33

This test must fail because the mask is all 0's and the ERSLT bit equals 1. With no termination bit set, the conditions dictate that the CPU is to execute the next test. If an error is encountered in this test, the CPU executes test 6, and stops. If test 2 is translated correctly, the CPU executes test 3, the second correct hardcore stop.

Test 3, Record 8. Test 3 in record 8 tests the ability of the CPU to stop on a failure. The format of test 3 is:

Mask (S)	MCW (T)	TN (S)	ATN (T)
FF FF FF FF	02 00 00 00	DD DD DD DD	FE FE 01 01

The MCW in this test has a format of 0000 0010, with the CT bit set to a 1. Because the ERSLT bit equals 0 and all 1's are in the mask, this test must fail. With the CT bit set and a failing test, the CPU should stop with the ATN in T. The complement of this ATN is the first true ROS test or, if the FLT tape is being run, the first test in the zero-cycle series. Therefore, when RESTART FLT I/O is depressed, testing should start with the first ROS test or the first zero-cycle test.

The correct indications for the successful passing of test 3 of record 8 are:

1. S: all D's (roller 1, position 3).
2. T: FE FE 01 01 (FFFF 0000 for ROS tests) (roller 2, position 3).
3. 'Fail' trigger set (roller 5, position 2, bit 30).
4. CT bit set (roller 5, position 2, bit 6).
5. UT bit set (set by hardware) (roller 5, position 2, bit 5).
6. ERSLT bit reset (roller 5, position 2, bit 7).
7. 'Buffer 1' trigger set (roller 5, position 2, bit 35).

If the CPU does not stop on this test, it will take the next test, 4, which has the UT bit set. The CPU must then stop because this bit has already been successfully tested.

Summary of Hardcore Tests. The hardcore tests have checked the CPU for the following:

1. Ability to sense 1's in S and T as 0's at PAL.
2. Ability to sense 0's in S and T as 1's at PAL.

3. Ability to take next test.
4. Ability to perform a TN search.
5. Ability to stop on a failing test.
6. Ability to stop on an UT signal.
7. Ability to make a result-comparison and decide on next step.
8. That all data paths connected with the above functions are operating properly.

ROS Bit Tests. Highlights:

- ROS tests, two doublewords, are read from tape one at a time, alternately into buffers 1 and 2.
- Test sequencing is controlled by ROS test sequencer.
- A test consists of: (1) fetching MCW from storage, (2) advancing CPU clock, (3) comparing state of one ROSDR trigger with predetermined result.
- If 'pass' trigger is set, operation proceeds to next test; if 'fail' trigger is set, testing is terminated.

The remaining records on the ROS test tape contain the test patterns used to check the ROS bit planes. Each ROS test pattern consists of two doublewords in the same configuration as the hardcore tests. (See "Hardcore Tests and IPL 2".) These tests are read from the tape one at a time, alternately into buffer 1 and buffer 2. At the completion of each read-in cycle, a 'TIC' pulse is generated (by the channel) which initiates the ROS test sequence.

Test sequencing is controlled by the ROS test sequencer. When the 'TIC' pulse is received, the ROS test sequencer is set to maximum (7). Then, as each portion of the test is executed, it is decremented by 1. For each count of the sequencer, the CPU is said to be in a certain "ROS state"; for example, if the ROS test sequencer equals 5, the CPU is in ROS test state 5. When the count reaches 0, the test is complete and the CPU enters the Wait state until another 'TIC' pulse is received.

A test consists of fetching the MCW from storage, permitting the CPU clock to advance a given number of cycles, and comparing the state of one of the ROSDR triggers with a predetermined result. If the state of the trigger matches its predicted value, the 'pass' trigger is set and the operation proceeds to the next test. If the actual and predicted values disagree, the 'fail' trigger is set and testing is terminated.

When the 'TIC' pulse is received, a TN comparison takes place. If the TN comparison is successful, the operation proceeds to scan in the MCW; otherwise, the operation reverts to the Wait state. CPU clock signals are distributed as long as the FLT counter does not equal 0. When the FLT counter equals 0, the CPU clock is stopped and the expected result comparison is started. The address sequencer governs the loading of a portion of the ROSDR word (32 bits) into T. The mask is in S (the other half of the word that was loaded into the MCW), and the result comparison takes place. If a 'TIC' or 'gap' pulse is not received, the operation returns to scan in and repeats the test.

For the following discussion of the ROS test, refer to Diagram 8-116, FEMDM.

ROS Test State 7. A ROS test is started when the ROS test sequencer and the check counter are set to maximum. This action occurs because one of the following conditions is present:

1. A 'gap' pulse is received from the channel, indicating an end of record. This condition is tested for during ROS test state 6.
2. The ROS test sequencer stepped to 0. This condition is the successful end of one ROS test; therefore the next test is initiated automatically.
3. The TN comparison failed during ROS test state 4; therefore the next test is brought in.
4. LOAD or RESTART FLT I/O was depressed.

During ROS test state 7, the only operation is the decrementation of the ROS test sequencer.

ROS Test State 6. During ROS test state 6, the 'sync' latch is reset, the TN/ATN address is forced to storage controls, and a scan storage request for the TN/ATN word is initiated. The operation then waits for a 'TIC' pulse before progressing. When the 'TIC' pulse is received, ensuring that a test is in storage, the ROS test sequencer is decremented by 1.

ROS Test State 5. During ROS test state 5, the state of the 'buffer 1' trigger is inverted so that the next ROS test is read into the other buffer. The TN of the incoming word is gated to S, and the 'TIC' latch is reset. The ROS test sequencer is then decremented by 1.

ROS Test State 4. During ROS test state 4, a 'scan out S and T' signal determines whether this is the test to be executed. This determination is accomplished by comparing the contents of S and T, via the negative-OR inputs to PAL (scan-out bus). S contains the TN of the current test obtained from the ROS test just read into storage during ROS test state 5. For the first test (hardcore test in record 3), T contains all 1's loaded from the DATA switches. For all subsequent tests, T contains the ATN of the previous test. This number should be the complement of the current TN. Thus, a successful comparison results in all 0's being sent to PAL.

PAL is then checked for an all 0 result. If PAL does *not* contain all 0's, indicating that the test currently in storage is not the one searched for, the ROS test sequencer is again set to maximum and decremented by 1. This action causes the CPU to be again in ROS test state 6, and the next sequential test on the tape is initiated. This operation continues until either the entire tape has been searched or the correct TN has been found. If a valid TN cannot be found, this condition is known as a tape runaway.

If PAL does contain all 0's, indicating a successful comparison, the 'pass' and 'fail' triggers are reset and the ROS test sequencer is decremented by 1.

ROS Test State 3. During ROS test state 3, a storage request is made for the mask/MCW word, and the ROS test sequencer is decremented by 1.

ROS Test State 2. Highlights:

- MCW fetched from storage is transferred to ST and subsequently distributed to address sequencer (bits 21–25), MCW-register (bits 0–7,20), ROSAR (bits 8–19), FLT counter (bits 26–31).
- CPU clock signals cycle ROS until FLT counter is reduced to 0.
- When FLT counter equals 0, CPU clock is stopped and result in ROSDR is scanned out to T.
- ROS test sequencer is decremented by 1.

During ROS test state 2, the doubleword fetched from storage is transferred to ST, and the address sequencer, the FLT counter, and the FLT clock are reset. T now contains the MCW which is subsequently distributed as follows:

1. T(32–39,52), which contains the ROS plane number [MCW(0–3)], the UT bit [MCW(5)], the CT bit [MCW(6)], and the ERSLT bit [MCW(7)], is transferred to the MCW register. Note that MCW(4) and MCW(20) are not used and therefore contain 0's. The ROS plane number is not used for the test but is displayed in case of a failure as a guide to the CE. For most ROS tests, except the hardcore tests, the UT bit is a 0 and the CT bit is a 1. The ERSLT bit is a 1 or a 0, depending upon the design of the ROS plane being tested.
2. T(40–51), which contains the ROS address of the plane to be tested, is set into ROSAR.
3. T(53–57), which contains the scan word address of the ROSDR bit to be tested, is transferred to the address sequencer.
4. T(58–63), which contains a count of the number of clock cycles needed to read out one ROS word, is transferred to the FLT counter and FLT clock.

At the same time, the 'scan counter control' trigger is set. The trigger output deactivates the ROS sense latch reset (up until this time the ROS sense latches have been held reset; therefore no microprogram operations have been taking place) and sets the 'scan counter control' latch. This latch causes the ROS test sequencer output to be blocked, thus taking the operation out of ROS test sequencer control, and inhibits stepping the ROS test sequencer.

At this time, the bit test begins ROS functions using the address in ROSAR. The CPU clock cycles ROS until the FLT counter (loaded from the MCW word and decremented by 1 in synchronism with the CPU clock cycles) is reduced to 0. When the FLT counter equals 0, the 'cycle counter equals zero' latch is set and the 'MMSC' trigger is set to stop the CPU clock and the test. The scan-out bus is now used to scan out the results. The 'SOROS' and 'sync' triggers are set, and the 'sync' latch is set to complete synchronization of the controls with the FLT clock. The ROS sense latches are reset, and register ingating is inhibited. The 'MMSC' trigger is then reset. The output of the scan-out bus is transferred to PAL and subsequently to T. The 'scan counter control' latch is reset, thus activating the 'reset ROS sense latch' signal and restoring control to ROS test state 2. A scan system reset resets the latches and triggers used to control scan out. The ROS test sequencer is decremented by 1.

ROS Test State 1. During ROS test state 1, the results in T and the mask in S are negative-OR'ed into PAL by the 'scan out S and T' signal. A scan storage request for the TN/ATN word is initiated and, depending on the ERSLT bit value and the PAL zero-result check, the 'pass' or 'fail' trigger is set as follows:

PAL	ERSLT Bit	'Pass' Tgr	'Fail' Tgr
= 0	1	set	–
≠0	0	set	–
= 0	0	–	set
≠0	1	–	set

If a 'TIC' signal has not arrived to indicate that the alternate buffer is full, and a scan input error did not occur, the operation returns to ROS test state 3 to repeat the test. If an error or a 'TIC' or 'gap' signal occurred, the ROS test sequencer is decremented by 1, thereby going to ROS test state 0.

ROS Test State 0. During ROS test state 0, the word fetched in ROS test state 1 is loaded into ST, and a stop or continue decision is made. If the test is to be continued, a TN or an ATN is specified, and the ROS test sequencer is set to 7 and decremented by 1. If the test is to be stopped, the 'pass pulse' trigger is reset to stop the FLT and scan clocks, the UT bit, MCW(5), is set, and a 'stop scan' signal stops the tape at the end of the current record. The ROS test sequencer is set to 7, and the scan controls stop. To continue testing ROS, either RESTART FLT I/O or LOAD must be depressed.

If the ROS tests are restarted by depressing RESTART FLT I/O, the same conditions are set up as for an IPL, except that instead of an 'IPL' signal, a 'start scan' signal is raised to the channel. This signal causes the backspace CCW in location 80 to be performed.

When the ROS test stops because of a failure, the following information is displayed in the system control panel indicators:

1. Failing plane (first digit of ROS address), in hex [roller 5, position 2, bits 0–3 (CPU TEST ADDRESS)].
2. Failing ROS address (last two digits of ROS address), in hex [roller 1, position 3, S(0–7)].
3. Failing bit in decimal [roller 1, position 3, S(8–15)].

Fault Locating Tests

- Each FLT checks small portion of logic by setting up conditions that affect exit trigger which can then be sensed for proper output.
- If error occurs, test is terminated and CE uses SCOPEX card with failing TN identification to determine test points to scope.
- FLT tapes consist of hardcore, zero-cycle, and one-cycle tests.
- FLT test consists of 9 or 10 doublewords.

As discussed earlier, each FLT checks a small portion of the logic by setting up certain conditions that affect a specific trigger (known as an exit trigger) which can then be sensed for the proper output. Each test on the tape, therefore, sets up the CPU to the proper status (certain triggers set) that results in a particular output of the logic under test. After the CPU has been set up for a test (scan in), the CPU clock is advanced a sufficient number of cycles to change the status of the exit trigger. The indicator associated with the exit trigger is then selected (scan out) and compared with an ERSLT bit in the FLT MCW, set as part of the scan-in routine. If the values are equal, the test passes; if unequal, the test fails. Because each test is repeated a number of times, both the pass and fail indicators may be on, which is interpreted as an intermittent failure.

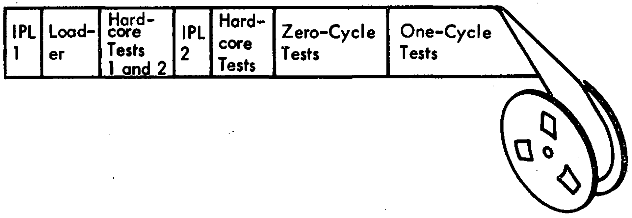
A summary of an FLT includes:

1. Load test into storage.
2. Scan into CPU triggers.
3. Advance CPU clock.
4. Stop CPU clock.
5. Scan out to T (scans desired indicator to T).
6. Load mask into S.
7. Scan out S and T (compares exit trigger with expected result).
8. Go to next test, or terminate and display failing TN.

CE action following this sequence (assuming an error or fault) requires locating the SCOPEX card with the failing TN identification, scoping the test points listed, and replacing the failing card(s).

FLT's are rarely run singly. So little time is involved that it is easier to run the entire series on a tape or until a failing test is encountered. The failing test may then be repeated until the fault is isolated.

FLT Tape. The make-up of the FLT tape is similar to that of the ROS test tape and contains the following records:



1. Record 1, IPL 1. This record contains the 24-byte "bootstrap" program necessary for any IPL operation. When LOAD is depressed, the three doublewords of this record are read into storage locations, as follows:

Storage Location (Hex)	Word	Contents
0	PSW	Backspace (not used).
8	CCW 1	Read command to read 40 bytes to location 80, and chain command to location 10.
10	CCW 2	TIC command to location 88.

2. Record 2, Loader. This record contains the loader program that reads in the FLT's. The IPL program in record 1 reads this record into storage locations, as follows:

Storage Location (Hex)	Word	Contents
80	CCW	Backspace command.
88	CCW	Read command to read 72 bytes to buffer 1 (location 8000), and a chain data tag to location 90.
90	CCW	TIC command to location 98.
98	CCW	Read command to read 72 bytes to buffer 2 (location 8080), and a chain data tag to location A0.
A0	CCW	TIC command to location 88.

3. Record 3, Hardcore Tests 1 and 2. This record contains the first FLT hardcore test. The hardcore tests for FLT's are identical to the ROS test hardcore tests except for their format. The FLT hardcore tests have the following format:

Storage Location (Hex)	FLT Word	Left Word	Right Word
8000	0	0	0
8008	1	Mask	MCW
8010	2	0	0
8018	3	0	0
8020	4	0	0
8028	5	0	0
8030	6	0	0
8038	7	0	0
8040	8	TN	ATN

Note that all locations except those containing FLT words 1 and 8 are blank. These blank locations are not used in hardcore tests.

4. Record 4, IPL 2. The first hardcore test causes a stop. When LOAD is depressed, the following IPL program replaces the IPL 1 program:

Storage Location (Hex)	Word	Contents
0	PSW	Backspace record.
8	CCW	TIC command to location 88 (loader program).

5. Records 5–8, Hardcore Tests. These records contain the remaining hardcore tests.
6. Zero-Cycle Tests. The next series of records on the FLT tape are the zero-cycle tests. These tests further check the scan logic necessary to run the main FLT's (one-cycle tests). They determine whether the scan-in and scan-out paths are operative. Scan-in and scan-out tests can exist only for triggers which have a scan-out path. In these tests, a pattern is scanned into the machine, the CPU clock is allowed to advance, and the output trigger is observed. If the trigger has a scan-in path, three tests are performed: for a reset state, for a set state, and again for a reset state. If the trigger has no scan-in path, only one test is performed, and this test ascertains that the trigger is at its reset state. While one trigger is being tested, other triggers can take on various states. Whenever possible, random states simulate the combinations which may be used in a one-cycle test and reveal interaction between triggers.
7. One-Cycle Tests. The remainder of the tape contains the one-cycle tests which are the true FLT's. The input has been generated by the FLT generating system and found to be a test that will detect and locate at least a single fault. An input pattern is scanned into the predetermined triggers, and the CPU clock is allowed to advance; the result, which is in the exit trigger, is compared with the value expected for a machine operating correctly.

The last test in each record is a dummy test, which keeps the channel from reading across the interrecord gap before the result of the last one-cycle test is known. Thus, if the last one-cycle test fails, a stop-scan sequence can be executed before the interrecord gap is reached. The dummy test is also used by the FLUT program. When used with FLUT, the dummy test is identified by a TN of all 1's and is forced to pass by having a mask of all 1's and an expected result bit of 1. The dummy test also provides a successful restart if the last test in a segment should fail.

Table 6-2. FLT Format

Storage Location (Hex)	Scan-In Word*	Left Word	Right Word
8000 and 8080	0	S	T
8008 and 8088	1	Mask	MCW
8010 and 8090	2	Q	Q
8018 and 8098	3	A	Miscellaneous triggers
8020 and 80A0	4	B	Miscellaneous triggers
8028 and 80A8	5	D	Miscellaneous triggers
8030 and 80B0	6	PSW	IC
8038 and 80B8	7	E, R	LAR
8040 and 80C0	8	TN	ATN

*Refer to ALD's M3071 and M3081 for bit assignments.

FLT Format. Zero-cycle and one-cycle FLT's have the same format (Table 6-2), consisting of 9 doublewords (10 if a 7080 Compatibility Feature or a 709/7040/7044/7090/7094/7094II Compatibility Feature is attached). The doublewords (scan-in words) are used as follows (see ALD's M3071 and M3081 for the FLT format if a Compatibility Feature is attached):

1. Scan-In Words 0 and 2–7. Contain the scan-in test pattern, which is distributed to registers and triggers as shown in Table 6-2. The data establishes a CPU state before performing the FLT.
2. Scan-In Word 1, Left Half (Mask). Specifies the exit trigger to be sensed. This is accomplished in the scan-out bus by OR'ing the scan-out word containing the exit trigger status with the mask bits. The mask contains all 1's except for a 0 in the position corresponding to the trigger being sensed. The output of the OR is inverted so that if the exit trigger is set, the value sent to PAL is 0.
3. Scan-In Word 1, Right Half (MCW). Contains the control information necessary to run the test:

MCW Bits	Scan-In Word 1 Position	Contents
0–3	32–35	0's
4	36	LW bit
5	37	UT bit
6	38	CT bit
7	39	ERSLT bit
8–19	40–51	ROS address (bits 73–84 of input pattern)
20	52	0
21–25	53–57	Scan out address
26–31	58–63	Cycle count

4. Scan-In Word 8. Contains the TN and ATN as follows: bits 0–15, TN; bits 16–31, complement of TN; bits 32–47, complement of ATN; bits 48–63, ATN.

FLT Test Setup. Several controls on the system control panel must be operated to initiate FLT's. The procedure for running FLT's appears in *2065 Processing Unit FEMM*, form Y27-2039-2 and ALD M8005. However, a short description of the setup is included here because it affects the operation.

Diagram 8-117, Sheet 2, FEMDM, shows the start of an FLT. The FLT tape is mounted first. The LOAD UNIT switches are set to the address of the tape unit holding the test tape, the TEST MODE, ROS/PROC/FLT switch is set to FLT, and the CPU CHECK switch is set to DSBL. Going into FLT test mode causes the 'FLT test' trigger to be set and a 'scan mode' signal to be sent to the channel. SYSTEM RESET is now depressed to place the CPU in a stop loop with the 'manual' trigger set. The DATA switches are set to all 1's, and STORE is depressed. This action places all 1's into ST so that T now contains an ATN for the first hardcore test. Lastly, LOAD is depressed, initiating a normal IPL operation which reads in IPL 1.

IPL 1. Depressing LOAD with the CPU in FLT test mode initiates operations that are identical to a normal IPL until a 'release' signal is received from the channel; i.e., 24 bytes are read from the selected device into the first three doubleword locations of main storage under ROS microprogram and channel control. At this point the CPU is idle, waiting for a 'release' signal from the channel. Meanwhile, the channel IPL operation reads in record 1 of the FLT tape and executes the channel program specified by record 1. (See "FLT Tape".) As a result, 40 bytes (record 2) are read into storage, starting at location 80 (hex). Record 2 contains the loader program that will read each FLT into the proper buffer area in storage.

After record 2 has been read in, command chaining causes CCW 2 in record 1 to be executed. This CCW is a TIC command to location 88, which now contains a read CCW. When the channel has finished executing the TIC command, it sends a 'release CPU' signal to the CPU. As soon as the 'release CPU' signal is received, the IPL microprogram is continued and the CPU proceeds as though performing a normal IPL. However, when the 'timing gate' trigger is reset (Diagram 8-117, Sheet 3), the 'IPL status' latch is set, which in turn sets the 'MMSC' trigger. With the 'MMSC' trigger set, the CPU clock is stopped, thus halting all ROS operations.

The 'sync' trigger and 'sync' latch are set to synchronize the operation to the FLT clock. In addition, the 'sync' trigger output resets the ROS sense latches and inhibits register input. Address 001 is forced into ROSAR, and the 'MMSC' trigger is reset to release the CPU clock. Thus, the FLT microprogram is initiated. ROS control now places the CPU in scan mode and sets the address sequencer to 7. The CPU is now ready to run the first hardcore tests.

Loader. Highlights:

- FLT's are read into two buffer areas (8000 and 8080, hex) in main storage.
- Each time a buffer is filled, 'TIC' pulse informs CPU that test can be performed.
- Read-in operation continuously checks for errors.
- After 32 errors, test is halted.
- Test is repeated until 'TIC' pulse is received.

The loader channel program read in by IPL 1 is used to transfer all FLT's, whether hardcore, zero-cycle, or one-cycle tests, to main storage. The test data is read into two buffer areas in main storage [starting at locations 8000 (hex), designated buffer 1, and 8080 (hex), designated buffer 2]. Each time a buffer is filled with a single test, a 'TIC' pulse informs the CPU that the buffer is filled and the test can be used by the CPU.

The tests are read into the buffer areas on a sequential basis. For example, test 1 is read into buffer 1. As test 1 is being processed, buffer 2 is being filled with test 2. As test 2 is being performed, buffer 1 is being refilled with test 3, and so on.

A continuing check for errors is made during read in and during transfer of data to the CPU. If an error is encountered, the tape is backspaced and another attempt is made to read the data into storage. The check counter, set to 31 by a micro-order, keeps track of the number of errors encountered and allows 32 attempts to read in before the test is halted. Detection of an error decrements the check counter by 1 and causes a backspace of the tape so that the read in is retried. Manual intervention is required following a stop under these conditions. Channel-control checks may also occur which halt the test procedure immediately.

Once a test is begun, it is repeated until a 'TIC' pulse is received, indicating the next test is ready for processing. No count is made of the number of times a test is run as this is a function of the data rate of the I/O device involved.

During normal sequential processing of FLT's (no faults encountered), the address sequencer is set to 7 and the eight doublewords composing a single FLT are scanned into the CPU. The following is a summary of the read-in and test sequence:

1. While the FLT is being scanned into the CPU, a continuing check for storage and/or input errors is made. Storage errors might occur between storage and the CPU. Input errors might occur between channel and storage while the next FLT is being read into the alternate buffer. For each error sensed, the check counter is decremented by 1. Detection of an error results in a backspace of the tape and a retry.
2. When scan in is complete, the 'scan mode' trigger is reset, and an unconditional branch to the ROS word addressed by MCW(8–19) is performed.

3. The 'scan counter control' trigger is set, and the FLT counter keeps track of the number of CPU cycles specified by MCW(26–31). The CPU clock is stopped when the FLT counter equals 0.
4. Following a successful scan in and clock advance, a scan-out operation places the test result into T. The combination of MCW(4) and MCW(21–25) determines the scan-out word that is transferred to T.
5. At this point, the mask is fetched from storage and transferred to S. A result comparison between the mask and the scanned-out word is made, and the 'pass' or 'fail' trigger is set. The next operation to be performed is determined by pass, fail, unconditional-, or conditional-terminate conditions.
6. Execution of this FLT is repeated until a 'TIC' or 'end of record' ('gap') signal is received. When either signal is received, the decision to stop or continue is made, and appropriate controls are set.

Transmission Checks During FLT Read In. When the channel discovers a data check, it sends a 'unit data error' signal to the FLT controls. The channel then automatically backspaces one record and starts reading into buffer 1 again.

The FLT controls, upon receipt of the 'unit data error' signal, decrement the check counter, reset the CPU, and wait for a 'TIC' pulse from the channel. When the 'TIC' pulse is received, the FLT controls start the testing from storage buffer 1, repeating the tests from the record that had the data check. If the check persists, the record is retried up to 32 times. If 32 attempts to complete the record are unsuccessful, the machine stops with the UDC (Unit Data Check) indicator (roller 5, position 2, bit 32) on.

If a control check occurs in the channel, the channel stops transmission and sends the 'channel control error' signal to the FLT controls. The signal stops the testing immediately and turns on the CCC (Channel Control Check) indicator (roller 5, position 2, bit 33).

If the BCU recognizes an address check, data check, or invalid address check, this information is sent to the FLT controls. On any of these checks, a 'stop scan' signal backspaces the record and starts reading into buffer 1 again. The CPU and FLT controls are reset, and the check counter is decremented. The FLT controls wait for a 'TIC' pulse from the channel. When this pulse is received, the FLT controls start testing from buffer 1, repeating the tests from the record involved. After 32 unsuccessful attempts to complete the test record, automatic testing stops with the error indicator on.

Hardcore Tests. Functionally, the hardcore tests at the beginning of the FLT tape are identical to the ROS hardcore tests. The differences are in the format and control of the tests. FLT hardcore tests have the same format as zero-cycle and one-cycle tests except that all scan-in words except words 1 and 8 are 0's. Word 1 contains the mask and MCW, and word 8 contains the TN/ATN. These words are used in the same manner as in the ROS hardcore tests.

FLT hardcore tests are controlled by the FLT ROS microprogram instead of by hardware as in the ROS hardcore tests. Successful testing terminates on a hardcore stop (a correct stop), and, upon restart, the CPU enters zero-cycle tests. The CE may attempt isolation and repair on an error stop encountered during hardcore tests according to the procedure in ALD M8006.

Zero-Cycle and One-Cycle Tests. Highlights:

- Scan in loads CPU with test data.
- Test cycle allows CPU to act on scan-in data.
- Scan out collects data after clock advance.
- Mask is compared with scan-out data.
- Testing ends or continues, depending on result comparison.

Each FLT follows a similar 6-step routine under ROS control (with the exception of scan out, which may be either under ROS or hardware control). The FLT sequence may be summarized as follows:

1. Scan In. When a storage buffer has been filled with an FLT and the CPU receives a 'TIC' pulse, scan in begins. The address sequencer is set to 7 by a micro-order, and words 0 through 7 are read into the CPU. These words enter the CPU via Q or ST, and are scanned into various registers and triggers throughout the CPU, using normal data paths, to set up the machine environment for a particular test. During scan in, certain micro-orders are interpreted for scan control rather than for functional operations. From T, the MCW is transferred to the address sequencer, the FLT counter, and the MCW register when the address sequencer equals 1. When the address sequencer equals 0, scan in is completed except for ST, and the test cycle phase is entered. A micro-order transfers T to ROSAR, and on the next cycle ST is scanned in.
2. Test Cycles. The count in the FLT counter, set by the MCW, determines the number of cycles taken by the CPU. A micro-order, specified by MCW(8–19), controls this portion of the test. When the FLT counter equals 0, the CPU clock is stopped and scan out begins.
3. Scan Out. The status of the exit trigger is placed into T. The scan-out word containing the status of the exit trigger is specified by the count in the address sequencer. MCW(4) determines whether the right or left word contains the exit trigger status. If the count in the address sequencer is 14 or greater, scan out is controlled by hardware. If the count is 13 or less, scan out is controlled by ROS.

4. Result Comparison. The exit trigger is compared with a known value, one that is predicted on the basis of the information scanned in. This comparison is accomplished by fetching the mask from storage, placing it into S, then OR'ing the scan-out word and the mask in the scan-out bus. The mask contains all 1's except for the position corresponding to the exit trigger, which is a 0. Assuming the exit trigger is set when it is OR'ed with the 0 in the mask, the output is transferred to PAL as a 0. Next, the CPU compares the output of the OR with MCW(7), the ERSLT bit. With all 0's in PAL and MCW(7) = 1, the CPU determines that the test has passed and sets the 'pass' trigger. If the comparison is unfavorable, the 'fail' trigger is set. Because the test is continually repeated until the next 'TIC' pulse is received, both the 'pass' and the 'fail' triggers can be set, indicating an intermittent failure, in which case the 'intermittent' trigger is also set. After the result-comparison, the CPU must decide whether to terminate testing or to continue.
5. Terminate or Continue. This decision is a major point in the FLT sequence. Four triggers determine what the CPU will do next: 'pass', 'fail', 'UT' [MCW(5)], and 'CT' [MCW(6)]. The CT bit is always set in current FLT's (except for certain hardcore tests). Depending upon the setting of these triggers, the decision that will be made is as follows:

Trigger Output				
'UT'	'CT'	'Pass'	'Fail'	Action
0	0	1	0	Continue – alternate test
0	1	1	0	Continue – next test
0	0	0	1	Continue – next test
0	1	0	1	Stop – gate alternate test on restart
1	0	1	0	Stop – gate alternate test on restart
1	0	0	1	Stop – gate alternate test on restart

The CPU repeats the current test until the 'TIC' pulse is received from the channel. At that time, the decision to stop or continue is made. If the CPU is to continue, the test in the opposite buffer is scanned in. No count is made of the number of times the test is repeated.
6. TN/ATN Comparison. This comparison is accomplished in hardcore and whenever RESTART FLT I/O is depressed. The TN located in word 8 (word 9 for a 7080 Compatibility Feature or a 709/7040/7044/7090/7094/7094II Compatibility Feature) is compared with the ATN left in T by the last test to be executed. Also, a specific TN may be entered via the DATA switches, and the CPU searches for this number to the exclusion of all other tests. As FLT's are now set up, the ATN in each test is the number of the next FLT on the tape (with the exception of hardcore tests).

Scan-In. Highlights:

- Test pattern establishes trigger status before test.
- Test words are read into S, T, or Q, then distributed throughout CPU under microprogram control.

The scan-in portion of an FLT consists of fetching a test pattern to establish a trigger status before a test. Scan-in test words are read into S, T, or Q from the FLT buffers. From these registers, the data is transferred throughout the CPU, under microprogram control, via special scan circuits and normal CPU paths. Transfer paths are determined by the microprogram, which is repeated without variation for each FLT. At the end of scan in, the CPU clock is allowed to cycle the number of times required to condition the exit trigger. Scan in ordinarily fetches eight words from storage into the CPU but provision has been made for a ninth word to be used with the 7080 Compatibility Feature or the 709/7040/7044/7090/7094/7094II Compatibility Feature.

A scan in is started with the address sequencer set to 7 (Diagram 8-117, Sheet 3, FEMDM). A scan storage request fetches scan-in word 7 of the record. This word is transferred to ST, the address sequencer is reduced by 1, and a check is made for a scan input error. If an error is encountered, the check counter is reduced by 1, a 'start scan' signal causes the tape to backspace, and the operation begins again where scan-mode is set into the CPU. If no error occurred, another scan storage request is made for word 6, which is held in the CPU pending a check for a 'TIC' or 'gap' pulse. A 'TIC' or 'gap' pulse indicates the alternate storage buffer is full. Because the scan in was just started and the CPU is much faster than a tape drive, it can be assumed that no 'TIC' or 'gap' pulse is received. The address sequencer is reduced by 1, and a scan input error sample is made. If no scan input error occurred, another scan storage request is made. Anytime a scan input error occurs, the check counter is reduced by 1, the 'start scan' signal causes the tape drive to backspace one record, and the operation repeats from the time scan-mode is set.

Scan in continues to operate for words 5, 4, 3, and 2 exactly as it did for word 6, except that no check for a 'TIC' or 'gap' pulse is made when the address sequencer equals 2. Word 1 (mask and MCW) is loaded into ST, and a check is made for a 'TIC' or 'gap' pulse. Before word 0 is transferred into the CPU, the mask and MCW (word 1) are transferred out of ST. Diagram 8-117, Sheet 4, shows that T(32–39) is transferred to the MCW, T(53–57) is transferred to the address sequencer, T(58–61) is transferred to the FLT counter and FLT clock, and the 'scan counter control' trigger is set. If no scan input error occurred, word 0 is loaded into ST after T(40–51) is loaded into ROSAR. The 'scan counter control' latch is then set to allow the test to be performed.

Test Cycles. The CPU clock is allowed to advance a number of cycles as specified by MCW(26–31) (Diagram 8-117, Sheet 5). A portion of the microprogram starting at the address specified by MCW(8–19) is performed. Note that at this point the CPU seems to be running a program. However, this is not the case. The micro-orders being performed merely allow the scan-in test pattern to be transferred through the logic being tested to change the state of the exit trigger. If setting these triggers at scattered points causes three legs of an AND to be tested, then 1's are directed to those triggers when testing the AND. If the three triggers happen to be those normally set during, for example, I-Fetch, this fact is incidental to FLT testing.

As soon as the CPU clock cycles begin, the FLT counter is stepped in synchronism with the clock. When the proper number of cycles has been taken, the FLT counter steps to 0, which causes the 'cycle counter equals zero' latch to be set. The 'SOROS' and 'MMSC' triggers and the 'sync' latch are then set. The 'MMSC' trigger stops the CPU clock. As soon as the 'sync' latch is set and the operation is synchronized to the scan and FLT clocks, the 'MMSC' trigger is reset. The operation proceeds under ROS or scan logic control, depending on the value in the address sequencer.

Scan Out. During scan out (Diagram 8-117, Sheet 5), the exit trigger status is transferred from the indicator driver circuits, through PAL, to T. This path is the same path used for logout operations.

The word scanned out is determined by the scan-out address in the address sequencer. The setting of the address sequencer determines which roller switch on the system control panel contains the desired information, and MCW(4) determines whether the right or left half of the word should be scanned out. Scan out is first controlled by scan hardware and then by the microprogram. The hardware-controlled portion of scan out is always performed first regardless of the value in the address sequencer, although only those fields addressed by a value of 14 or greater are scanned out. If the exit trigger is in a scan-out word whose address is 13 or less, the field is scanned out under microprogram control.

ROS control is resumed by forcing one address if the address sequencer is greater than 13, and another address if it is less than 14. The microprogram started by the former is ready to perform a result comparison because the exit trigger value is in T. The latter program first branches on MCW(4) to determine if the left or the right half of the doubleword addressed by the address sequencer should be scanned out. The LSWR is gated to T by a normal CPU gating signal; therefore a branch must be made on address sequencer equal 13. Because the LSWR is contained in the right half of word 13, a branch on address sequencer equal 13 is unnecessary if MCW(4) equals 1. By means of these branches, the microprogram transfers the field to be tested to T, and thus begins the result comparison.

Result Comparison. FLT's check logic by moving data through a group of logic to an exit trigger and causing a change in the state of the exit trigger. To determine the success or failure of a test, the change in the exit trigger must be sensed on the basis of the data entered into the logic. This change is predictable, and the data on the test tape designates whether the exit trigger is to be a 1 or a 0 at the conclusion of each test. The expected state of the trigger is contained in the MCW as the ERSLT bit.

The scan out S-and-T function (Diagram 8-117, Sheet 5) determines the setting of the exit trigger by comparing a mask with the scan-out word containing the status of the exit trigger. Scan out places the selected scan-out word into T. A mask word, fetched from storage, is put into S. The mask contains all 1's except for the bit position corresponding to the exit trigger in T. This bit position is a 0, and therefore the exit trigger determines the setting of PAL.

If the exit trigger is set, a 0 is transferred to the corresponding PAL bit; if reset, a 1 is transferred to PAL. Thus, PAL equals 0 only if the exit trigger equals 1. If the ERSLT bit [MCW(7)] also equals 1, or if PAL does not equal 0 (indicating that the exit trigger is reset) and the ERSLT bit equals 0, the 'pass' trigger is set. If MCW(7) equals 1 and PAL does not equal 0, or if MCW(7) equals 0 and PAL equals 0, the 'fail' trigger is set. In any case, if both the 'pass' and 'fail' triggers are set, the 'intermittent' trigger is also set.

Terminate or Continue. When the mask was loaded into S, the address sequencer was set to 8 and a scan storage request was initiated to fetch scan-in word 8 (TN/ATN). At the same time that the mask is compared with the scan-out word, scan-in word 8 arrives from storage and is gated to ST. The result in PAL and the ERSLT bit are used to set the 'pass' or 'fail' trigger, and thus record the result. The address sequencer is then decremented, resulting in a repeat of the same test just executed.

Assuming the test was successful and no errors were found, the operation described is repeated until the buffer is filled with the next test or until an end-of-record gap is signalled. These conditions are repeatedly sampled for during scan in. If a 'TIC' or 'gap' pulse is received, scan-in execution is abandoned. The address sequencer is set to 8 and the TN/ATN of the test in progress is fetched and loaded into ST. After a system reset which clears the CPU registers, a stop or continue check is then made (Diagram 8-117, Sheet 5). If a continue condition results, the operation proceeds to the setting of scan mode to start the next or alternate test as determined by the stop or continue controls. In case of a stop conclusion, the 'stop scan' signal goes to the channel and the UT bit [MCW(5)] is set. The clocks are then stopped by resetting the 'pass pulse' trigger, and the CPU enters a microprogram loop.

After a stop condition, manual intervention is necessary. The operation then depends upon the pushbutton depressed, as follows:

1. START (with TEST MODE, REPEAT switch off). Runs the test that causes the stop one time and causes another stop.
2. START [with TEST MODE, REPEAT switch in the Repeat (down) position]. Continues to run the test that caused the stop until TEST MODE, REPEAT is turned off.
3. STORE. May be used to set an ATN into T. (May be used in conjunction with RESTART FLT I/O to perform an ATN search for a known test.)
4. RESTART FLT I/O. Sends a 'start FLT' signal to channel and resets the UT and CT bits and the 'fail' and 'buffer 1' triggers. The 'pass' trigger is set. (This combination represents an ATN search condition.)
5. LOAD. Initiates a system reset and begins at the start of the FLT test tape.

TN/ATN Comparison. Every FLT contains the TN and the ATN of the test which the CPU is to execute upon completion of the current test. This information is contained in word 8 (word 9 for the 7080 Compatibility Feature or the 709/7040/7044/7090/7094/7094II Compatibility Feature) and consists of the TN and ATN in both true and complement form.

The hardcore tests verify the ability of the CPU to conduct a TN search by comparing the TN (in S) with the ATN (in T) using the 'scan out S and T' signal. A 0 output indicates a favorable comparison, and the CPU then executes the test. If the inverted-OR output is not sensed as all 0's, the CPU rejects the test and continues searching. The bit configuration in hardcore tests cannot strictly be called a TN, but the functions of comparison, acceptance, or rejection of the test are valid.

During zero-cycle and one-cycle tests, the TN search is rarely used. If the CPU does not encounter a failing test once it has entered the FLT sequence, word 8 is not even used. The address sequencer is set to 7, and scan in begins immediately.

However, if a failing test is encountered, the last action the CPU takes is to put the ATN into T. The test immediately following the failing test has a TN that is the complement of the ATN. When this TN is brought into S and compared with the ATN, a 0 output results, and the CPU begins testing again with this test. This action allows the CE to get past a failing test when he desires to do so. This situation could happen, for example, when an FLT is failing because of an engineering change and the CE is aware that failure is not due to a malfunction. Each FLT has the TN of the next sequential FLT as its ATN.

For example, assume the computer has just run TN 01 09 and the test has failed. The next test is TN 01 0A. RESTART FLT I/O is depressed to get past the failing test. Upon restart the tape is backspaced to the beginning of the record containing the failing test. The TN of each test in the record is brought into S for comparison with the ATN left in T by the failing test. The comparison is favorable when the test immediately following the failing test is encountered, and the CPU resumes testing at that point. TN 01 09 leaves the following configuration in T (right half of word 8):

ATN Complement	ATN
FE F5	01 0A

TN 01 0A, the next test, has the following configuration in the left half of word 8 which was brought into S:

TN	TN Complement
01 0A	FE F5

The following binary bit configuration is scanned out:

```
T:  1111  1110  1111  0101  0000  0001  0000  1010

S:  0000  0001  0000  1010  1111  1110  1111  0101
```

The CPU resumes testing with TN 01 0A and continues until another failure is encountered or to the end of the testing sequence.

RIPPLE TESTS

The ripple tests may be used to exercise either main storage or local storage and several functional units within the CPU. The tests use only the internal hardware of the CPU, including several ROS words, and storage. The procedure stores the contents of the DATA switches into all locations of storage or displays the contents of all storage locations.

The ripple tests may be used as a quick confidence test of CPU and storage operation or as a means of identifying, with the parity check indicators, a failing major functional unit within the CPU. The overall ripple test routine is contained in *2065 Processing Unit*, FEMM, form Y27-2039-2.

DIAGNOSTIC PROGRAMS

For a list and discussion of the diagnostic programs available for the Model 65, refer to the *2065 Processing Unit*, FEMM, form Y27-2039-2.

MARGINAL CHECKING

The marginal checking facility enables several units in the system to be operated with nonstandard voltage conditions, thus providing a means of detecting critical circuits that are deteriorating to a critical voltage-sensitive operating point. In addition, the CPU may have its clock period decreased from 200 ns to 195 ns, thus providing a means of detecting circuits that have developed a slower switching speed.

Several of the power supplies in the CPU, and several power supplies in the storage units and channels whose power control is interconnected with the CPU, may be varied from the nominal output voltage by controls at the CPU. The CPU power supplies are varied directly from the system control panel. (The adjustment procedure for these controls is contained in *2065 Processing Unit*, FEMM, form Y27-2039-2.) The storage units and channels are varied by motor drives controlled from the system control panel. (The adjustments of the remote control assemblies are contained in their respective FEMM's.)

The marginal checks are performed by running the ROS tests and FLT's with all 6V marginable power supplies in the CPU reduced to 5.5V dc and the ROS power supply reduced to 80 percent of nominal. If the tests passed, rerun them with normal voltages, but with the FREQUENCY ALTERATION switch set. After a successful run of the ROS tests and FLT's, run selected diagnostic programs with the marginable power supplies in the CPU, storage units, and channels set from nominal to higher or lower output.

Appendix A. Special Circuits

The only special circuits in the 2065 are in ROS for the differential amplifiers, latches, and driver circuits; they are not, however, field repairable. For a discussion of the standard SLT circuits, refer to the *SLT Component Circuits,* FEMI, Form Z22-2798 *(IBM Confidential, for release only to authorized persons).*

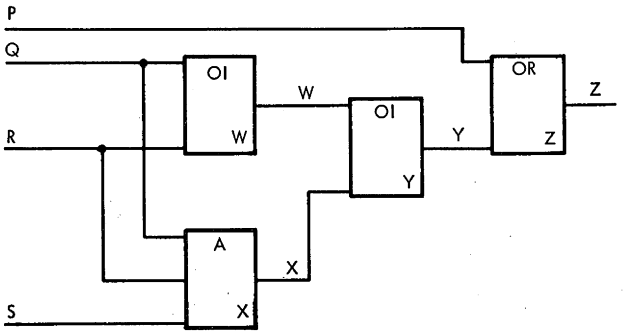
Appendix B. World Trade Differences

The major difference between World Trade versions of the Model 65 and the domestic version is that 50-Hz power is used in World Trade machines. This change primarily affects: (1) power distribution and control (Chapter 5), (2) decrementing of the interval timer (Chapter 3, Section 1: "Timer Exceptional Condition"), and (3) decrementing of the Multisystem timer, if the Multisystem feature is installed (Chapter 4, "Multisystem Timer").

Appendix C. Example of FLT Generation

Consider the simple four-block tree shown in Figure C-1. Assume that output Z is observable and that inputs P, Q, R, and S can be set directly. The FLT generating programs try to develop tests for each block to show that each input can, by itself, control the output and that the output can take both values. For example, the tests for a three-input OR would be 100, 101, 001, which check the inputs, and 000, which checks for a 0 output.

If tests are to be generated for the logic shown in Figure C-1, the first test pattern generated would put 10 as the inputs to block Z, which then yields a 1 output and is sensitive. The 1 input (P) is also sensitive, but the 0 input is not. To make the output of Y 0, a 1 is required on at least one input. Assume the program chooses W, which requires Q and R to be 0. This action forces X to 0; S can be any value, and is arbitrarily set to 0. Thus, the first test pattern is 1000 and tests only two nets. (See line 1 of the table in Figure C-1.)



Test #	P	Q	R	S	W	X	Y	Z
1	1$	0	0	0	1	0	0	1$
2	0	1$	0	0	0$	0$	1$	1$
3	0$	0$	0$	0	1$	0	0$	0$
4	0$	1$	1$	1$	0	1$	0$	0$
5	0	0	1$	0	0$	0$	1$	1$
6	0	0$	1	1	0$	0$	1$	1$
7	0	1	0$	1	0$	0$	1$	1$
8	0	1	1	0$	0$	0$	1$	1$

Legend: $ means that this net is sensitive.

Figure C-1. Four-Block Tree and FLT Pattern Generated

To test the other input of Z, a pattern 01 is needed. For the Y output to be 1, W and X must both be 0 and, because Y is now sensitive, so are W and X. To make the output of X 0, at least one input must be 0, and R is arbitrarily chosen. Now to make the output of W 0, Q must be 1 and is sensitive. Again, S is set to 0.

The next test pattern generated puts 00 as inputs to Z and checks the output for 0. The same procedure that set Y to 0 in test 1 is followed. However, Y is now sensitive and, as shown in line 3 of the table, so are Q, R, and W.

Block Z has now been fully tested and so the program steps back a level to check block Y. The first test for this sequence already exists in test 3 and so is not redeveloped. The second test requires 01 as the Y inputs. For the X output to be a 1 means that Q, R, and S must all be 1's. This condition automatically causes the output of W to be 0, which is correct. Because Y cannot be observed directly, its output must be propagated to Z as a sensitive path. This propagation is done by making P 0. Now Z, Y, X, and all of the inputs to X are sensitive. Because the output test for block Y is performed by the second test, testing of block Y is now complete.

The first input test for block W is already included in test 2. The second test makes Q 0 and R 1, which causes the output W to be 0. To propagate this condition sensitively through Y requires that X be 0, which it is. Again, to propagate through Z, P must be 0. S is arbitrarily set to 0. This condition is line 5 of the table in Figure C-1. The output test for block W is included in test 3.

The input test patterns for AND X should be 011, 101, and 110. None of these has been set up so far. Because these are all primary inputs, no real test development is needed. The only other important input is P, which must be 0 for Y to be sensitive. These three patterns are shown in lines 6, 7, and 8 of the table in Figure C-1. The output test for block X is already in pattern 4.

A test now exists for every possible logic failure in the network, and the process ends. The test patterns are formatted for the machine, termination bits are added, and index numbers are assigned. These numbers are those in the table in Figure C-1, with a constant added to define the segment.

The failure data from the table is now used to produce a SCOPEX. In a SCOPEX, all sensitive points are listed for each test, together with indications showing which are newly tested. Table C-1 is an abbreviated version of SCOPEX for the first five tests. Note that the "Fed By" column refers only to those sensitive items feeding the given net. Thus, in test 2, although the output of block X is sensitive, no inputs are sensitive and so no input is listed. The list for test 4 shows an example of the G/FI notation. In this test, all three inputs to X are 1 and are sensitive. However, entry Q has been tested at 1 through block W before this. Therefore, the entry is now newly tested, but is newly tested as an input to X. For this reason, the G/FI is put on the line of X.

Table C-1. SCOPEX Example

	V	Net	Ref	Fed By
	***1			
G/F	1	Z	AA	A1
G/F	1	P	A1	Entry
	***2			
	1	Z	AA	AB
G/F	1	Y	AB	AC, AD
G/F	0	X	AC	–
G/F	0	W	AD	A1
G/F	1	Q	A1	Entry
	***3			
G/F	0	A	AA	AB, A1
G/F	0	Y	AB	AC
G/F	1	W	AC	A2, A3
G/F	0	P	A1	Entry
G/F	0	Q	A2	Entry
G/F	0	R	A3	Entry
	***4			
	0	Z	AA	AB, A1
	0	Y	AB	AC
G/FI	1	X	AC	A2, A3, A4
	0	P	A1	Entry
	1	Q	A2	Entry
G/F	1	R	A3	Entry
G/F	1	S	A4	Entry
	***5			
	1	Z	AA	AB
	1	Y	AB	AC, AD
	0	W	AC	–
	0	X	AD	A1
G/F	1	R	A1	Entry

Index

Note: All index entries for Chapters 1 and 2 are in Volume 1; all other entries are in Volume 2.