IBM Field Engineering

Theory of Operation

2065 Processing Unit, Volume 1

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 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.)

Second Edition (November 1973)
This is a reprint of SY27-2306-0 incorporating changes released in the following Technical Newsletter:

SY27-2258 (dated October 16, 1969).

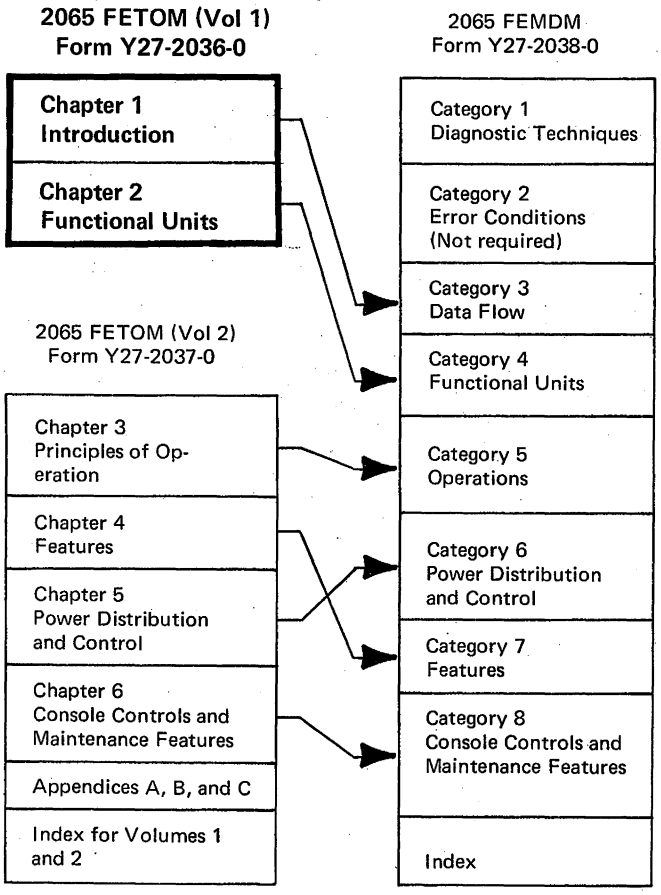
Changes are periodically made to the specifications herein; any such changes will be reported in subsequent revisions or Technical Newsletters.

Text for this manual has been prepared with the IBM SELECTRIC ® Composer.

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 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.

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 2 are made to Category 4 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

(THIS VOLUME)

Illustrations

(THIS VOLUME)

(VOLUME 2)

Tables

(THIS VOLUME)

(VOLUME 2)

Abbreviations

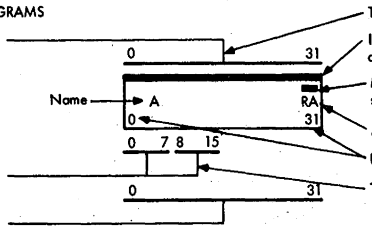
ABC	AB register byte counter
ac	alternating current
adr	address, addressed, addressing
ALD	automated logic diagram
amp	ampere
ASC	address store compare
ATN	alternate test number
BCD	binary-coded decimal
BCU	bus control unit
C	capacitor
CAW	channel address word
CB	circuit breaker
CC	condition code
CCW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

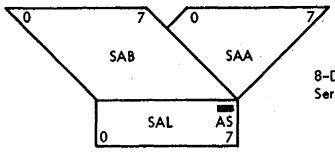
CROS	capacitive read-only storage
CSW	channel status word
CT	conditional terminate
dc	direct current
dec div	decimal divide
dec ovflo	decimal overflow
DX	first byte in a series of destination bytes
DX + 1	second byte in a series of destination bytes
DX + 2	third byte in a series of destination bytes
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
$\geq$	greater than or equal to
$\geqslant$	greater than or equal to
$\leq$	less than or equal to
$\leqslant$	less than or equal to
=	equal to
$\neq$	not equal to

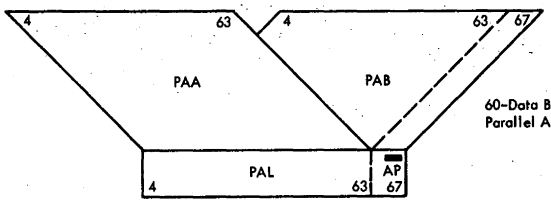
Legend

LOGIC DIAGRAMS

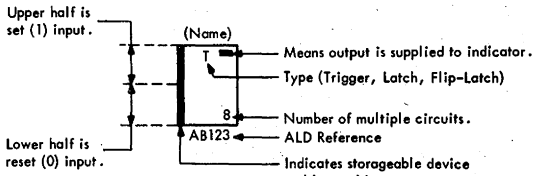




8-Data Bit (Plus Parity) Serial Adder



60-Data Bit (Plus Parity) Parallel Adder

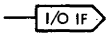


Indicator

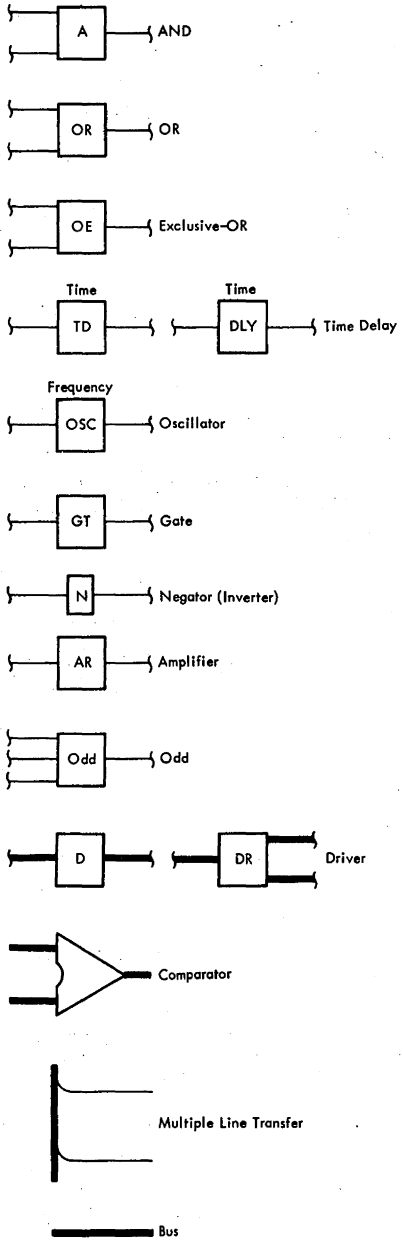
Roller Switch Indicator

Parity Check Logic

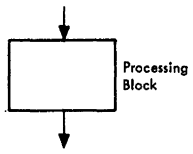
Parity Generate Logic

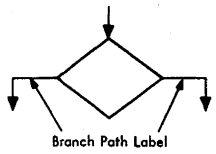


Interface
Denotes interface between two units.

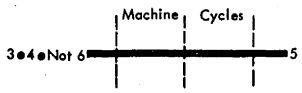


FLOWCHARTS

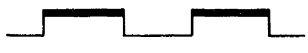




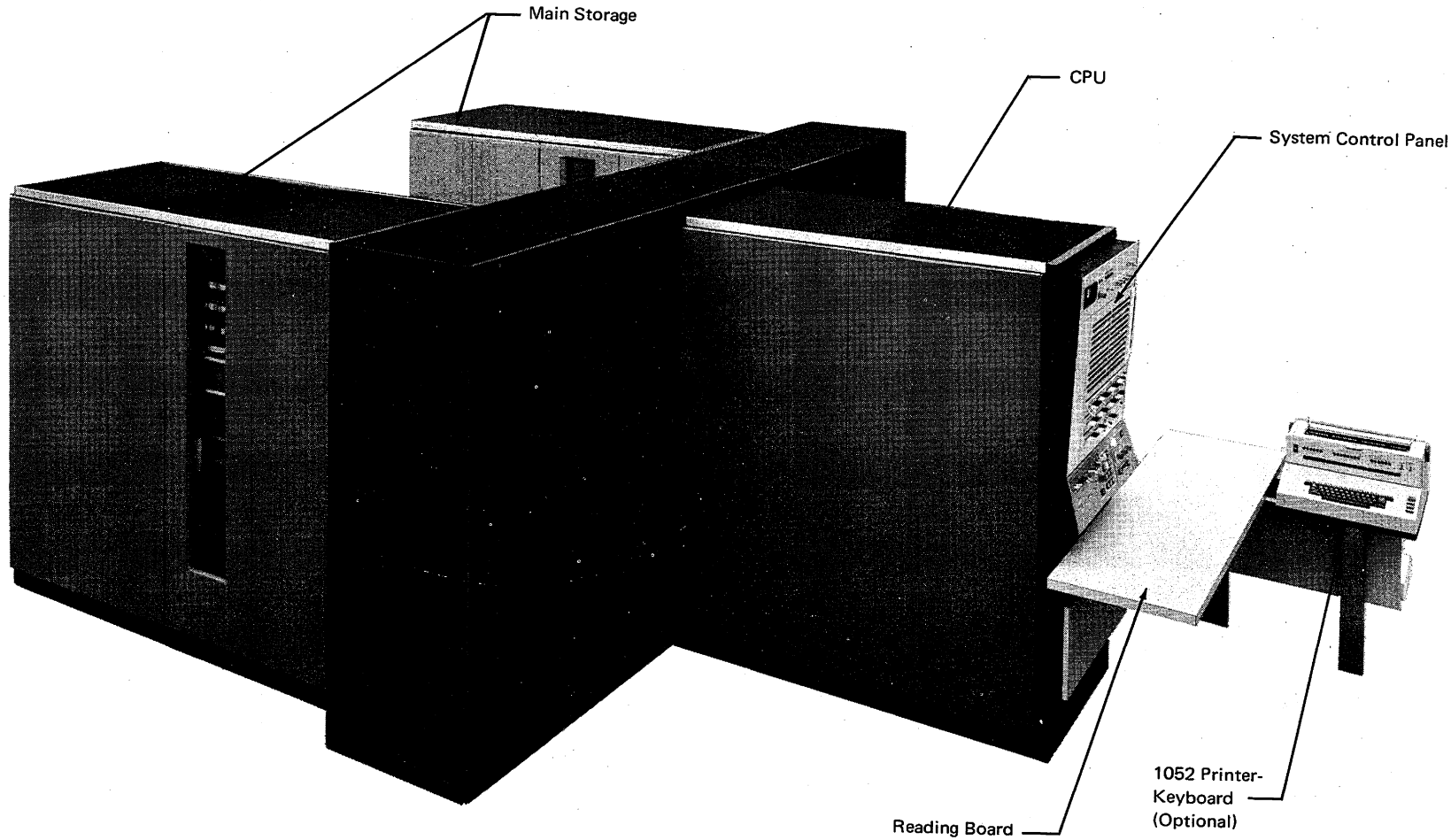
TIMING CHARTS



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.



System/360 Model 165

This chapter introduces the 2065 Processing Unit (Central Processing Unit or CPU) of the System/360 Model 65. It is divided into four sections:

Section 1, System Description, describes the basic system in terms of main storage, CPU, and input/output (I/O), the features available, and the data flow. Also included is a discussion of multisystem configurations and the resources available to achieve a multisystem.

Section 2, System Coding, discusses the hexadecimal number system, character codes, instruction formats, and operands used in the System/360 Model 65.

Section 3, Program Execution and Control, discusses the role of the supervisor program, the eight program states, the make-up of the program status word (PSW), the purpose and implementation of interruptions and exceptional conditions, and the initiation and control of I/O operations.

Section 4, CPU Description, discusses the functional units of the CPU, Instruction Fetching (I-Fetch) and instruction execution, the Universal instruction set by instruction class, and power considerations.

Section 1. System Description

BASIC SYSTEM

A basic System/360 Model 65 is shown in Figure 1-1. This system consists of a 2065 CPU, one 2365 Processor Storage Model 1 or up to four Model 2's, and up to six 2860 Selector Channels and one 2870 Multiplexer channel. (The number and configuration of I/O control units and devices is flexible and is therefore not shown.) Five models are available:

1. Model G65. Uses one 2365 Processor Storage, Model 1 (131,072 bytes).
2. Model H65. Uses one 2365 Processor Storage, Model 2 (262,144 bytes).
3. Model I65. Uses two 2365 Processor Storage, Model 2's (524,288 bytes).
4. Model IH65. Uses three 2365 Processor Storage, Model 2's (786,432 bytes).
5. Model J65. Uses four 2365 Processor Storage, Model 2's (1,048,576 bytes).

As shown in Figure 1-1, the storage address bus (SAB) and the two storage data buses ['storage data bus in' (SDBI) and 'storage data bus out' (SDBO)] are common to the CPU and I/O channels. The Bus Control Unit (BCU), located in the CPU, monitors and controls the availability of these buses and selects the main storage area to be used. Storage requests (stores and fetches) generated by the CPU and I/O channels enter the BCU, which issues a response signal to the requesting unit having the highest priority. The response signal places main storage at the disposal of the unit for one storage cycle. All units must compete for the next available storage cycle in the same manner. I/O channel requests are given priority over CPU requests, and each channel has an assigned priority.

The 2065 CPU and main storage layout is shown in Figure 1-2. The system control panel is attached to the end of the CPU frame. One or two optional 1052 Printer-Keyboards may be placed adjacent to the CPU reading board to serve as operator consoles.

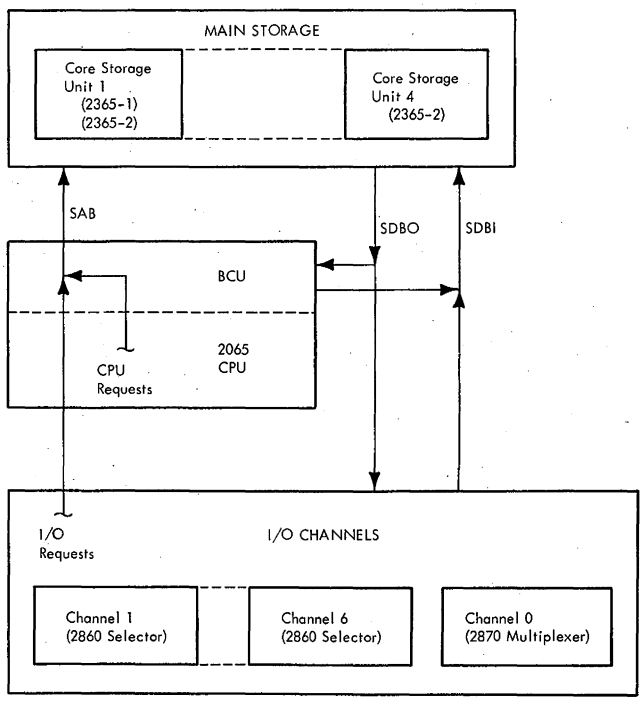


Figure 1-1. Basic System Configuration

FEATURES AVAILABLE

Features available for the 2065 CPU are:

1. Direct Control. Provides two instructions, Read Direct and Write Direct, and six external interruption lines which are independent of I/O channel operations. The instructions provide timing signals and transfer a byte of information between two CPU's or between the CPU and an external device.
2. 2361 Attachment. Provides bulk core storage of from 1,048,576 bytes to 8,388,608 bytes through the attachment of up to four 2361 Core Storage Units. Two models are available: Model 1, which has a storage capacity of 1,048,576 bytes; and Model 2, which has a capacity of 2,097,152 bytes. This feature provides two-way interleaving for pairs of 2361 units.
3. 1052 Adapter. Permits attachment of one 1052 Printer-Keyboard, Model 7, for system console I/O. Also includes a program-controlled alarm. Two 1052 Adapter features can be installed, allowing the attachment of a 1052 on each side of the CPU reading board.
4. Multisystem. Permits two 2065 CPU's to be joined to form a Multiprocessing System/360, Model 65. Both CPU's share a configuration control panel and two to four 2365 Model 13 Processor Storage units (262,144 bytes each). See Additional Storage Attachment feature also.

4A. Additional Storage Attachment (Model J only). Four features, each permitting the attachment of one additional 2365 Model 13 Processor Storage in a Multiprocessing System/360 Model 65.

5. Emergency Power-Off Control. Provides, in effect, a single Emergency Power-Off switch in a "room" or "area".
6. 7070/7074 Compatibility (Models H, I, IH, and J only). Allows the execution of 7070 and 7074 instructions.
7. 7080 Compatibility (Models H, I, IH, and J only). Enables the system to execute 705 and 7080 instructions.
8. 709/7040/7044/7090/7094/7094II Compatibility (Models I, IH, and J only). Enables the system to execute 709, 7040, 7044, 7090, 7094, and 7094II instructions.

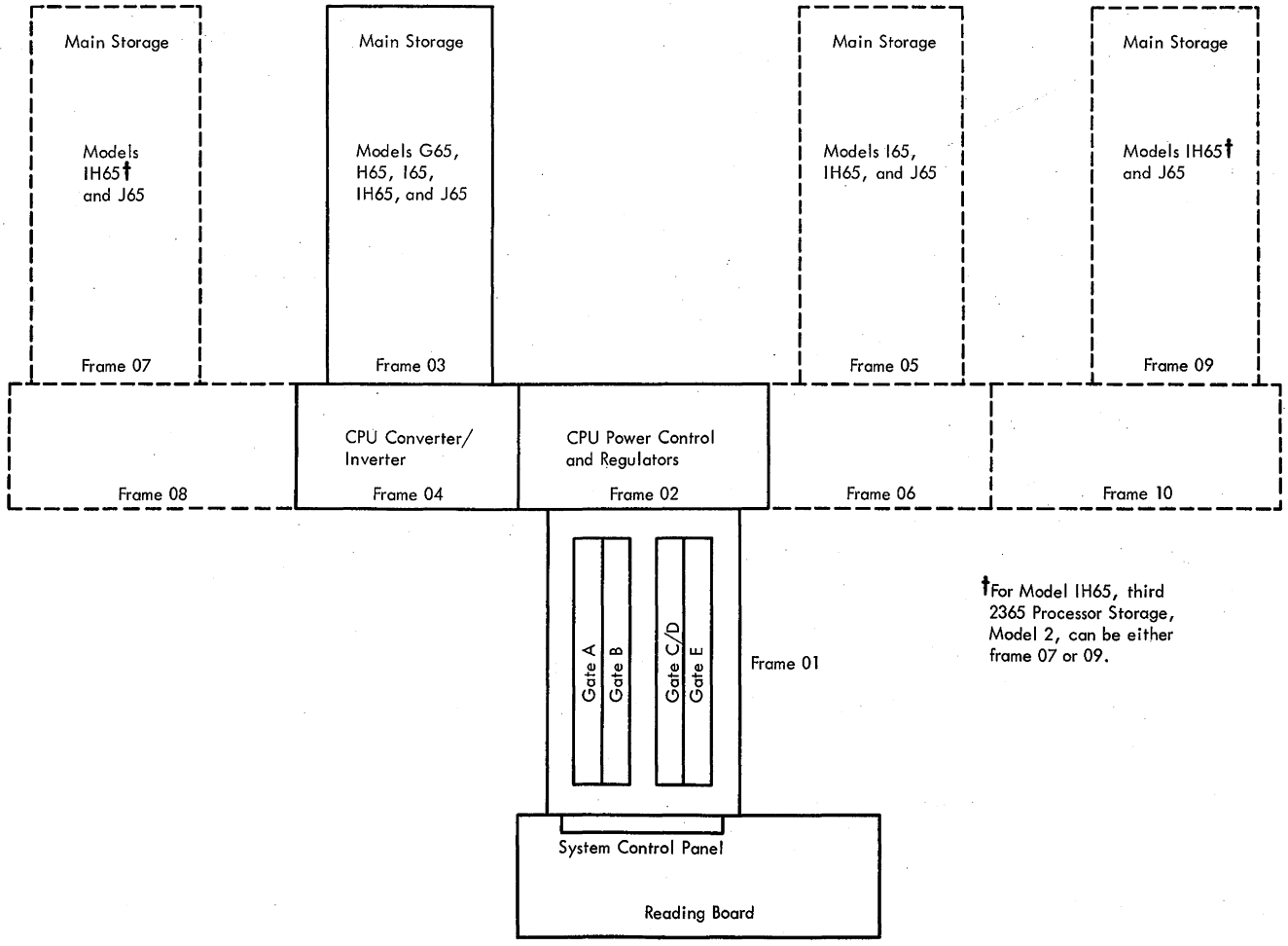


Figure 1-2. System/360 Model 65 Layout

SYSTEM DATA FLOW

The basic data flow in the System/360 Model 65 is shown in Diagram 3-1, FEMDM. The multiplexed SDBO and SDBI are the data paths between main storage and the CPU, and between main storage and the I/O channels. These buses transfer 64 data bits (8 bytes or a doubleword) plus 8 parity bits on each main storage cycle. The BCU determines which unit may access main storage according to a priority scheme. The multiplexed SAB carries 21 address bits plus 3 parity bits from the highest-priority requesting unit to the main storage addressing logic.

When storing data, the eight mark triggers determine which bytes of the data placed on the SDBI are to be stored into the doubleword addressed by SAB. When fetching data, a doubleword of data is transferred, via the SDBO, to the requesting unit from the main storage location addressed by SAB.

The bus-in and bus-out lines between the I/O channels and their attached control units are part of the Standard I/O Interface, and transfer one byte of data at a time.

Each I/O channel has buffer registers which can receive or transmit a doubleword of data between the I/O channel and main storage, and which can receive or transmit a byte of data at a time between the I/O channel and the attached I/O control units.

Parity checking throughout the system is at a byte level.

Main Storage

- Byte, eight bits, is format building block and basic addressable unit of information.
- Halfword is two consecutive bytes.
- Word is four consecutive bytes.
- Doubleword is eight consecutive bytes.
- Byte locations are numbered consecutively, starting with 0.
- Group of bytes is addressed by lowest-numbered (leftmost) byte.
- Data is fetched in doubleword lengths.
- Data is stored on byte basis.
- Store and fetch protection is provided for 2048-byte blocks.

The byte, which consists of eight bits, is the basic building block of all formats and the basic addressable unit of information. A ninth bit, the parity bit, is transmitted with each byte and establishes odd parity for each byte. The parity bit cannot be affected by the program; its only purpose is to cause a machine check interruption when a parity error is detected. References to the size of data fields and registers therefore exclude the associated parity bits. All storage capacities are expressed in number of bytes provided, regardless of the physical word size used.

Bytes may be handled separately or grouped together in fields. A *halfword* is a field of two consecutive bytes; a *word*, a field of four consecutive bytes; a *doubleword*, a field of eight consecutive bytes. The location of any field of bytes in main storage is specified by the address of its lowest-numbered (leftmost) byte.

Byte locations in main storage are numbered consecutively, starting with 0; each number is considered the address of the corresponding byte. A group of bytes in main storage is addressed by the lowest-numbered (leftmost) byte of the group. The CPU can accommodate binary addresses up to 24 bits long, thus providing addresses for up to 16,777,216 bytes. Addresses 0 through 4095 can be generated without a base address or an index. This property is important when the program status word (PSW) and general register contents must be preserved and restored during program switching. Thus this area has special significance to supervisor programs and contains all permanent storage assignments, such as old PSW's, new PSW's, channel address word, channel status word, and the interval timer value (Table 1-1).

The available storage is normally contiguously addressable, starting at address 0. An addressing program interruption occurs when addressing beyond the maximum available capacity of the installation. Except for a few instructions, the interruption occurs only when attempting to use the data and not before.

Each time a storage unit is accessed by the CPU or I/O channels, information is either stored into or fetched from that storage unit. When a storage request is received via the BCU, core storage performs a read/write storage cycle. If a fetch operation is to be performed ('store' latch reset), the addressed doubleword from the core array is gated to the memory data registers (MDR's), a pair of 32-bit registers which serves as a buffer. The output of the MDR's is gated to the SDBO and to the write circuits of the core array to be regenerated into the addressed doubleword location of main storage. The SDBI is blocked so that data is not entered into main storage.

If a store operation is to be performed ('store' latch set), the SDBO is blocked, and the data on the SDBI is accompanied by one or more mark signals on the mark bus. There is a mark line for each of the eight SDBI bytes. Presence of signals on these lines indicates which bytes are to be replaced in core storage. Thus data can be stored selectively by bytes; a single store operation can replace up to eight bytes in storage at the address designated by the SAB. Although the SDBO is blocked, the addressed doubleword from the core array is gated to the MDR's. There, those bytes that do not have a corresponding mark line active (those bytes to be regenerated) are combined with the bytes on the SDBI that have a corresponding mark line active (indicating new data). The contents of the MDR's are then transferred to the write circuits and placed into the addressed doubleword location of the core array.

Instructions that involve fetching and subsequent storing of data do not necessarily take storage accesses consecutively; it is possible for a channel to take one or more intervening accesses. Only the Test and Set instruction takes a combination fetch and store access without permitting any intervening accesses.

Table 1-1. Permanent Main Storage Assignments

Main Storage Address		Length	Information Stored
Dec	Hex		
0	0	Doubleword	Initial program loading PSW
8	8	Doubleword	Initial program loading Channel Command Word 1 (CCW 1)
16	10	Doubleword	Initial program loading CCW 2
24	18	Doubleword	External interruption, old PSW
32	20	Doubleword	Supervisor call interruption, old PSW
40	28	Doubleword	Program interruption, old PSW
48	30	Doubleword	Machine check interruption, old PSW
56	38	Doubleword	I/O interruption, old PSW
64	40	Doubleword	Channel Status Word (CSW)
72	48	Word	Channel Address Word (CAW)
76	4C	Word	Unassigned
80	50	Word	Timer
84	54	Word	Unassigned
88	58	Doubleword	External interruption, new PSW
96	60	Doubleword	Supervisor call interruption, new PSW
104	68	Doubleword	Program interruption, new PSW
112	70	Doubleword	Machine check interruption, new PSW
120	78	Doubleword	I/O interruption, new PSW
128	80	22 doublewords	Diagnostic log-out area

A storage protection capability is provided to protect the contents of certain areas of main storage from destruction (store protection) or misuse (fetch protection). Locations may be protected against store violations or against store and fetch violations but never against fetch violations only. Protection is achieved by dividing main storage into 2048-byte blocks and assigning a five-bit protection key to each block. The keys for the blocks are contained in a separate small core array in the associated storage unit. The low-order bit of the key in storage designates whether the block is protected against fetches. If this bit is a 0, no fetch protection is specified and only store protection is provided. If this bit is a 1, protection applies to both storing and fetching. The key may be assigned by the Set Storage Key instruction and inspected by the Insert Storage Key instruction.

When protection applies to a storage reference, the key in storage is compared with the protection key supplied by the CPU or channel. Access is permitted only when the four high-order bits of the key in storage match the supplied protection key or when the supplied protection key is zero.

2365 Processor Storage

Two models of the 2365 Processor Storage unit (also referred to as the High-Speed Storage or HSS) are available: Model 1, which provides 131,072 bytes; and Model 2, which provides 262,144 bytes. Both models have an internal cycle time of 750 ns.

Each 2365 Model 2 Processor Storage is organized into even- and odd-numbered doubleword storage areas. The access path to these areas is through a shared interface which allows the BCU to interleave even and odd requests; i.e., an odd request can be issued by the BCU halfway through an even cycle, and vice versa. By interleaving references to even-numbered and odd-numbered doublewords in main storage, the effective storage cycle approaches one half the cycle time for the unit. As a maintenance aid, the interleaved mode can be defeated to allow operation of a program solely in the odd or even area of storage.

2361 Core Storage (Optional)

The overall storage capacity of the system may be increased by means of the 2361 Attachment feature. This feature provides bulk, direct-access core storage through the attachment of up to four 2361 Core Storage Units (also referred to as Large Capacity Storage or LCS). Two models are available: Model 1, which has a storage capacity of 1,048,576 bytes; and Model 2, which has a capacity of 2,097,152 bytes. Either one 2361 Model 1 or up to four 2361 Model 2's may be attached if interleaving is *not* utilized; *with* interleaving, either two 2361 Model 1's, or two or four 2361 Model 2's may be attached.

Eight bytes are accessed per storage cycle. Read access time is 3 usec; a read/rewrite storage cycle requires 8 usec. If interleaving is utilized, the LCS units are attached to the system in pairs, and the addressing assignment is split so that one unit contains all odd addresses and the other all even. Thus, these units can be addressed on an interleaved bias to achieve a sequential access rate of 4 usec per doubleword.

2065 Central Processing Unit

The 2065 CPU performs arithmetic, logical, and control instructions specified by a stored program. It contains facilities for addressing main storage, processing fixed-point, floating-point, and decimal arithmetic, operating on logical data, sequencing instructions in the desired order, and initiating I/O operations. Also provided are facilities for character-handling, processing of fixed-length and variable-field-length (VFL) data, indexing, and indirect addressing. Functionally the CPU can be divided into four major sections: control, BCU, instruction fetching, and instruction execution.

Control Section

The basic CPU clock cycle period is 200 ns. A clock signal generator provides a 5-megaHertz (5-mHz) symmetrical (100-ns clock and 100-ns not-clock portions) signal. To provide additional time for CPU logic functions, the symmetrical clock signal is modified to give a 5-mHz unsymmetrical (80-ns clock and 120-ns not-clock portions) signal. Clock distribution logic distributes and synchronizes the clock signals to the logic gates, and stops distribution of clock signals to most of the CPU processing logic upon detection of a machine check during certain operations.

A significant feature of the CPU is the read-only storage (ROS) used to control operations. The ROS contains a permanently recorded microprogram holding information that remains fixed during machine operations. The information is in the form of 100-bit words, each word containing a unique, predetermined bit pattern. When decoded, the bits control gates to route data in the CPU. Word access time is approximately 95 ns. The information can be read out as required, but a physical modification is necessary to change the stored information. In general, a control word is read out from ROS at the end of each machine cycle (200 ns) and controls the CPU during the following machine cycle. Each ROS word contains the address of the ROS word to control the CPU during the following cycle. The number of control words (and machine cycles) required to perform a particular operation may vary because both the individual functions and the address of the next ROS word are modifiable by the (1) operation in progress, (2) data or control bit configuration, and (3) detection of interruptions or exceptional conditions. Used as a control device, ROS eliminates the need for most complex instruction decoders and sequencing networks.

The Program Status Word (PSW), a doubleword, contains the information required for proper program execution. Primarily, the PSW controls instruction sequencing and holds the system status in relation to the program being executed. By storing the PSW, the program can preserve the status of the CPU for subsequent analysis. By loading a new PSW or part of a PSW, the state of the CPU may be changed.

Bus Control Unit

The main storage associated with a system is shared by the CPU and I/O channel. A main storage address bus (SAB) and two main storage data buses (SDBI and SDBO) are also common to the CPU and I/O channel. The BCU, located in the CPU, monitors and controls the availability of these buses and selects the main storage area to be used. Storage requests (stores and fetches) generated by the CPU and I/O enter the BCU, are processed by means of a priority scheme, and are executed when the requesting unit is granted priority. When there is a conflict between the CPU and an I/O channel wanting to use main storage, the BCU gives priority to the channel. Priority between channels is also preassigned, with channel 1 having highest priority, followed by channels 2, 0, 3, 4, 5, and 6. When the CPU cannot access main storage because of a channel, the CPU clock is stopped until main storage is available to the CPU.

Each unit requiring access to main storage issues a request to the BCU via a separate (simplex) control line. The BCU examines the control lines from each unit, establishes which unit should be granted access to main storage for the particular cycle, and generates response signals for that unit. These response signals control the placing and sampling of data on the common bus. Access to main storage for a particular unit is granted by the BCU on a cycle-to-cycle basis, and all units again compete for the subsequent storage cycle.

The CPU operates on a basic cycle of 200 ns while the main storage cycle time is 750 ns. To increase processing speed, the CPU overlaps, whenever possible, storage requests for new data with processing of the existing data within its buffer registers.

Instruction Fetching Section

An instruction buffer, the Q-register, provides buffering for eight instruction bytes (four halfwords), thus reducing the number of storage requests that must be made to fetch instructions from main storage. Associated with the Q-register is the R-register. It holds the halfword (two bytes) containing the op code, received from the Q-register, of the instruction to be executed next. As a result, the R-register allows overlapping of instruction fetching with the refilling

of the Q-register. The E-register holds the halfword containing the op code of the instruction being presently executed. The instruction op code is used to address the ROS, which provides the required microprogram for execution of a specific instruction.

An interruption capability permits the CPU to change state automatically as a result of conditions arising outside the system, in I/O units, or in the CPU. An interruption switches the CPU from one program to another by changing the instruction address; the interruption cause and all essential machine status information are stored for analysis in a program status word. This information is available to the system as required. When a different program is requested, the status information about the current program is stored temporarily and is retrieved when that program is to be continued. This facility allows interrupted programs to resume at the point of interruption without the loss of control conditions.

Before an instruction is executed, it is tested for odd parity. The op-code halfword is tested in the E-register. The remaining halfwords, if any, are tested by the parallel adder half-sum checking circuits as the effective address is calculated.

Instruction Execution Section

The CPU operates on a basic internal cycle time of 200 ns. The data flow (Diagram 3-2, FEMDM) utilizes two major working registers (ST and AB) to give increased speed and simplified implementation of the instruction set. The 60-bit parallel adder is the focal point for most data transfers and facilitates handling the full long fraction in floating-point operations. An eight-bit serial adder provides simultaneous execution of the floating-point exponent as the fraction is operated on in the parallel adder, and has the capability of executing decimal arithmetic and VFL instructions.

The CPU extracts from the doubleword fetched from main storage the bytes on which it will operate. Thus, storage accesses are not required for every byte. As a result, processing speed is increased and system performance is improved. Data may be stored, however, on a byte basis; any number and combination of bytes up to 8 (doubleword) can be stored in one storage cycle. Addresses for data are formulated in the arithmetic section of the CPU and then placed in the D-register for addressing main storage. The D-register is also used for addressing I/O devices on I/O instructions.

The CPU incorporates a local storage of 25 word-length registers: 16 word-length general-purpose registers for fixed-point operations; 4 doubleword length (8 word-length) floating-point registers for floating-point operations; and 1 word-length register, called the working register, for miscellaneous operations. Local storage serves two functions: (1) it is used in generating operand addresses in main storage and (2) it holds operands and intermediate results of data operations, thus eliminating the need for special-purpose registers, such as accumulators. Local storage employs nondestructive readout, eliminating the need for regeneration, and operates on a 200-ns cycle with an 80-ns access time.

In the CPU, checking is facilitated by providing a parity bit for each byte of data. Odd parity is maintained. A parity check is made on data transferred to and from main storage. Most data transfers within the CPU are made via the parallel adder, in which the parity of each operand byte is checked against the half-sums. The parity of the sum, which is not a function of the parity of the operands but is generated logically within the adder, is checked against the latched sum.

As programs are executed, they are checked for correct instructions and data. This monitoring action identifies and separates program errors and machine errors. Because each type of error causes a unique interruption, program errors cannot create machine checks (errors).

Input/Output

- I/O section consists of devices, control units, channels.
- I/O operations are initiated by CPU and thereafter are performed independently of CPU.

Use of the I/O devices by the system is referred to as an I/O operation. The basic I/O operations are reading, which transfers data from an I/O device to main storage, and writing, which records at an I/O device data received from main storage.

The operation of a specific I/O device is governed by a control unit (Diagram 3-1, FEMDM) which provides control unique to its attached devices. The control unit may be a part of the I/O device or may be shared by a number of devices and be a separate logical entity. Besides controlling the operations of specific I/O devices, the control unit makes all I/O devices appear identical to the channel.

All I/O devices are connected, via their control units and the Standard I/O Interface, to the I/O channels. Each channel provides the logic necessary for synchronizing I/O data cycles with those of main storage and exerts programmed control over operations of the I/O devices connected to it. Logically, the channel can be considered as an autonomous entity provided for transferring information between I/O control units and main storage under control of the CPU.

In the System/360 Model 65, the channels interface directly with the CPU, providing a path for the exchange of system control information. This connection is used by the CPU to start and monitor a channel program, and by the channel to alert the CPU of the progress and termination of I/O operations. A BCU-controlled connection between the channel and main storage provides a path for the channel to fetch and store data, as well as channel command words that supplement the direct communication between the channel and the CPU.

Lengthy I/O operations (such as reading and writing) are executed in parallel with the CPU operations. The CPU need only initiate an I/O sequence by issuing an instruction to the channel. Thereafter, the channel establishes the address of the first channel command word for that sequence, establishes connection with the required I/O device, and verifies that the operation designated by the CPU can be executed. Once the device is started and the channel is set up to execute its commands, the CPU is released. Several commands can be chained, creating an I/O program for the device.

Main storage cycles are required during I/O operations to transfer data to or from main storage. These cycles do *not* interfere with CPU operations, except when both the CPU and the channel concurrently attempt to access main storage.

Channel

After the operation with the device is initiated, the channel assembles or disassembles data and synchronizes the transfer of data bytes with main-storage cycles. To transfer data, the channel maintains and updates an address and a count that describe the destination or source of data in main storage. Data is transferred between main storage and the channel eight bytes at a time, and between the channel and the control unit one byte at a time.

Modes of Operation. Data can be transmitted between main storage and an I/O device in two modes: burst mode and multiplex mode.

In the burst mode of operation, a device monopolizes the channel and stays logically connected to the channel for transmission of a burst of information. The burst can consist of a number of bytes, a block of data, or a sequence of blocks with associated chained commands. Only one I/O device can communicate with the channel during the time a burst is transmitted.

In the multiplex mode of operation, the channel facilities are shared by a number of concurrent I/O operations. All I/O operations are split into short intervals of time during which only a segment of information is transmitted over the interface. The intervals associated with different operations vary in response to demands from the I/O devices. The channel controls are occupied with any one operation only for the time required to transmit a segment of information. The segment can consist of a byte of data, a few bytes of data, or a control sequence such as the initiation of a new operation.

Operation in burst and multiplex modes is distinguished by the way the channels respond to I/O instructions. A channel operating in the burst mode appears busy to new I/O instructions. A channel operating in the multiplex mode appears available to new I/O instructions.

Types of Channels. Highlights:

- Model 65 can accommodate two types of channels: selector channel, which operates in burst mode; multiplexer channel, which can operate in either burst or multiplex mode.
- Model 65 can utilize up to six selector channels (addresses 1 through 6) and one multiplexer channel (address 0).

A model 65 can be equipped with two types of channels: 2860 Selector and 2870 Multiplexer. The channels are classified according to the modes of operation they can sustain.

The channel facilities required for sustaining an I/O operation are termed a *subchannel*. The subchannel consists of the channel storage used for recording the addresses, count, and any status and control information associated with the I/O operation. The mode in which a channel can operate depends upon whether it has one or multiple subchannels.

The 2860 Selector channel has only one subchannel and operates only in the burst mode. The bursts extend over the whole block of data, or, when command chaining is specified, over the whole sequence of blocks. The selector channel cannot perform any multiplexing and, therefore, can be involved in only one data transmission operation at a time. Once a data transfer is initiated, a logical connection is established between the selected I/O device and the channel. This connection persists for the duration of the operation, and other I/O devices cannot communicate with the channel. In the meantime, however, other I/O devices attached to the channel can execute operations that do not involve communication with the channel. The selector channel keeps scanning I/O devices for interruption conditions when it is not executing an operation or a chain of operations. A maximum of six selector channels may be attached to a 2065; their addresses are 1 through 6. A 2860 Model 1 contains one selector channel; a 2860 Model 2 contains two; a 2860 Model 3 contains three.

The 2870 Multiplexer channel is intended primarily for operation with lower-speed devices. It contains multiple subchannels and can operate in either multiplex or burst mode. It can switch between the two modes of operations at any time, and an operation on any one subchannel can occur partially in the multiplex and partially in the burst mode.

When the multiplexer channel operates in the multiplex mode, it can concurrently sustain one I/O operation per subchannel, provided that the aggregate data rate does not exceed the capacity of the channel. Each subchannel appears to the program as an independent selector channel. When the multiplexer channel is not servicing an I/O device, it keeps scanning its devices for data and for interruption conditions.

When the multiplexer channel operates in the burst mode, the subchannel associated with the burst operation monopolizes all channel facilities and appears to the program as a single selector channel for the duration of the burst. The remaining subchannels on the multiplexer channel remain dormant and resume communicating with I/O devices when the burst is completed.

One 2870 Multiplexer channel may be attached to a 2065, and its address must be 0.

Control Units

The control unit is the buffer between the I/O device and the channel. It adapts the characteristics of the device to the standard form of control provided by the channel. A control unit may be housed separately or it may be physically and logically an integral part of the device. The control unit accepts control signals from the channel, controls the timing of data transfer to the channel, and provides indications of device status.

Except for the signal used to establish priority among control units, all communication to and from the channel occurs over a common bus, and any bus signals provided by the channel are available to all control units. At any instant, only one control unit is logically connected to the channel. Selection of a control unit for communication with the channel is controlled by a signal that passes serially through all control units and permits each control unit to respond sequentially to the signals provided by the channel. A control unit remains logically connected to the channel until it has transferred the information it needs or has, or until the channel signals it to disconnect.

Up to eight control units (limited by electrical considerations) can be connected to one channel.

I/O Devices

The I/O devices consist of auxiliary storage as well as equipment used to communicate with the system users. The auxiliary storage includes direct-access devices such as disks and drums, as well as sequential-access devices such as magnetic tape units and punched card equipment. The equipment used to communicate between the system and its users includes a wide range of devices: printers, visual display units, keyboards, and communication terminals. Up to 256 directly addressable devices (limited by addressing facilities) can be attached to one channel.

Multisystem Configurations

- Multisystem consists of two or more interconnecting systems, each with a CPU.
- Multisystem can be achieved by:
  1. Channel-to-Channel Adapter
  2. Transmission Control Unit
  3. Two-Channel Switch Feature
  4. Device Switching Unit
  5. Shared LCS Feature
  6. Direct Control Feature
  7. Multisystem Feature

A multisystem consists of two or more systems, each with a CPU, that can communicate with each other without manual intervention. Thus, in addition to their specialized jobs, the individual systems may pool resources to perform a common job.

A multisystem configuration consists of two CPU's with a complement of storage and I/O equipment. This equipment is shared by the CPU's, but is considered to be logically independent if the CPU's are interconnected by well-defined interfaces so that they can, upon reconfiguration, operate without communicating with each other. Examples of logically independent system components are storage units, CPU's, I/O control units, and I/O devices.

Although logically independent, these system components may still be physically dependent because they share common equipment. Communication between the CPU's of a multisystem may be achieved by transmitting information from one CPU to another through a connecting link or by giving them access to a shared storage medium.

Figure 1-3 shows, in simplified form, the major components of a single, or simplex, system. A multisystem configuration can be achieved by the following:

1. Channel-to-Channel Adapter. Allows connecting the I/O interfaces of two channels (Figure 1-4). The main purpose of the channel-to-channel adapter is to make each channel appear as a control unit to the other channel. Transmission of data between the two channels

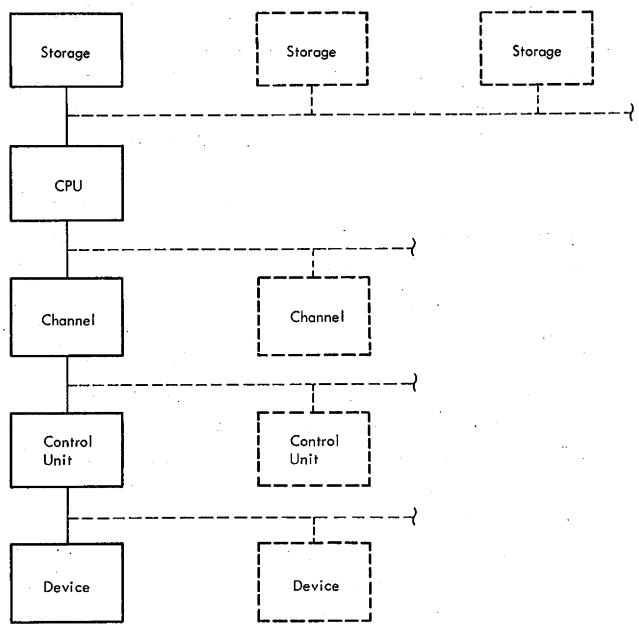


Figure 1-3. Functional Structure of a Simplex System

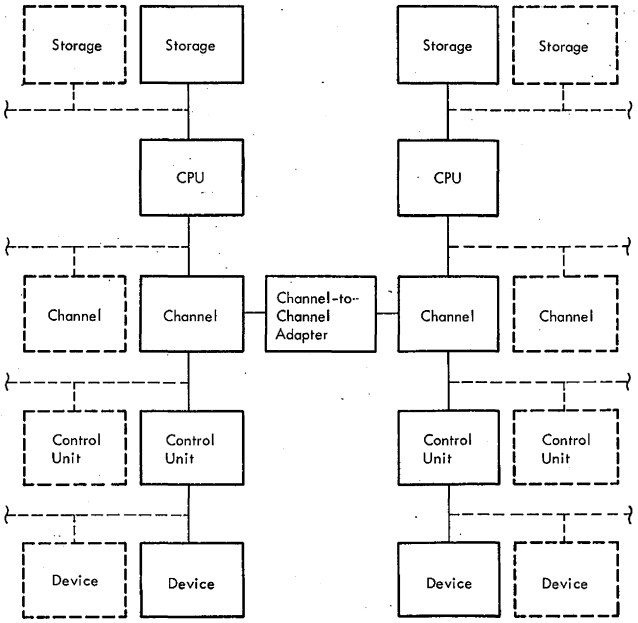


Figure 1-4. Channel-to-Channel Adapter as Multisystem Connector

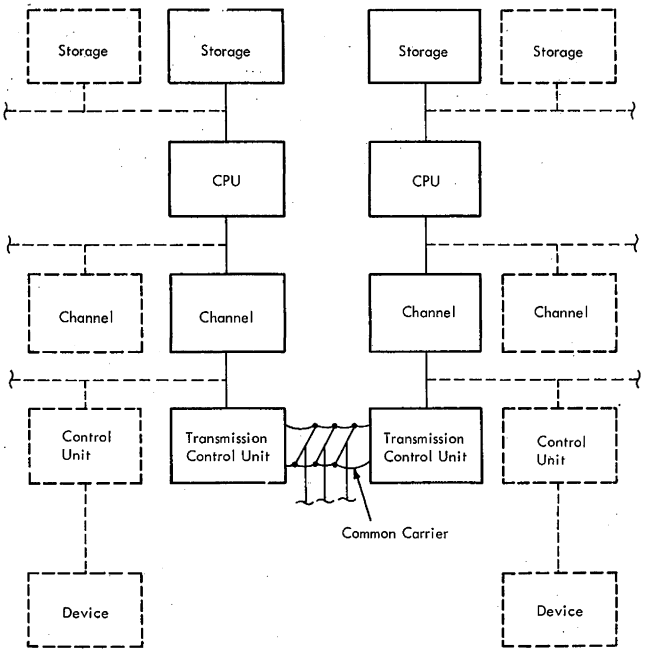


Figure 1-5. Transmission Control Unit as Multisystem Connector

is by byte at a rate established by the two channels. Because of the standardization of the I/O interface, this adapter may connect any model of the System/360 to any other model, and may use any type of channel on a given model. Any number of channel-to-channel adapters may be used in a multisystem, but their main function is in a multisystem emphasizing medium reconfiguration time or equipment specialization.

2. Transmission Control Unit. Permits communication by private line or common carrier. As indicated in Figure 1-5, communication is established by a specialized device interface rather than via the channel interface. The rate of data transmission is determined mainly by the line capacity. Any two models of System/360, as well as those of any other system, can be connected. The major multisystem application of the transmission control unit is for geographically separated computers.
3. Two-Channel Switch Feature. Permits sharing of control unit. When two or more CPU's have access to a common file, information placed into the common file by one CPU can be read by another. In contrast to transmission, sending and receiving are not simultaneous, and a one-to-one relation between recording and retrieval is not necessary. The choice of the shared media is determined by access time, capacity, and cost per bit. Shared devices are useful for program restarting information for job recovery upon reconfiguration. Disks, drums, and tape units may be pooled for storage of system programs as well as a means of communication between specialized CPU's to achieve improved turnaround time. Because a control unit normally controls several disk files, drums, or tape units, a switch between a channel and a control unit allows efficient sharing of a control unit by two CPU's (Figure 1-6).
4. Device Switching Unit. Tape units or other I/O devices are shared between control units (Figure 1-7), rather than control units being shared between channels as in Figure 1-6. This choice permits pooling of tape units between control units and permits simultaneous operation of any combination of tape units. This logical

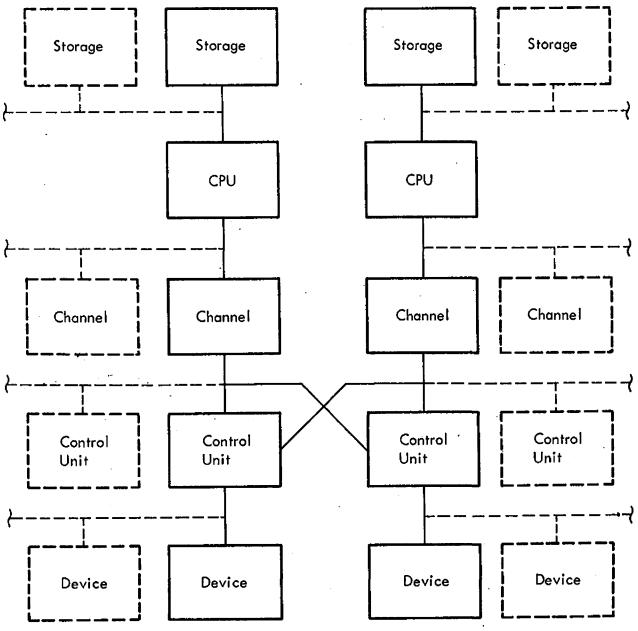


Figure 1-6. 2-Channel Switch Feature as Multisystem Connector

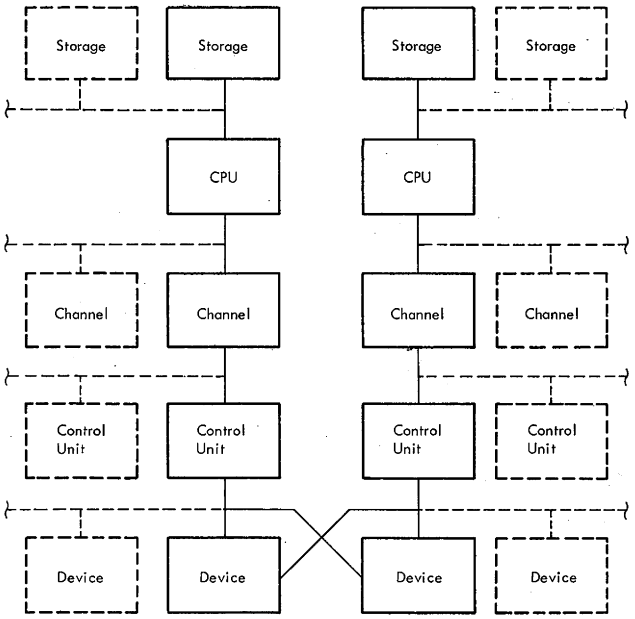


Figure 1-7. Device Switching Unit as Multisystem Connector

ability increases the thruput of a multisystem, as well as providing a common file. For example, the sharing of any pair of tape drives by two control units improves the sorting time significantly.

5. Shared LCS Feature. LCS can be shared by two CPU's (Figure 1-8). When one program is executed by different CPU's, it is desirable to have the locations of instructions and data located in identical addresses in every CPU. This addressing convention is adopted in System/360 for multisystem operation. This application of shared storage is mainly used in multisystems requiring a short reconfiguration time.
6. Direct Control Feature. Communications between CPU's may be direct (Figure 1-9). Using the Direct Control feature, control signals and one byte of control information are transferred without waiting for recognition. (The F- and G-registers in the 2065 are used by the Read Direct and Write Direct instructions, respectively.) This multisystem connection, unlike those previously described, does not lend itself to the transfer of large amounts of data.
7. Multisystem Feature. Any one or any combination of the foregoing interconnection methods results in a multisystem. However, the most sophisticated method of interconnecting two Model 65's utilizes the Multisystem feature to form the Multiprocessing System/360 Model 65 (Figure 1-10). 2365 Processor Storage Model 13 units are used, the CPU's are equipped with the Multisystem feature (with the Direct Control feature as a prerequisite), and most of the control units are equipped with the Two-Channel Switch feature. Any of the other interconnection methods, (except the Shared LCS feature) may also be incorporated. The Multiprocessing System/360 Model 65 and the Multisystem feature are described in Chapter 4, Section 2.

The interconnection methods described above are not always used to form a multisystem. A single system can also effectively make use of:

1. Channel-to-Channel Adapter feature.
2. Transmission control unit.
3. Two-Channel Switch feature.
4. Device-switching unit.
5. Direct Control feature (connected to an external device).

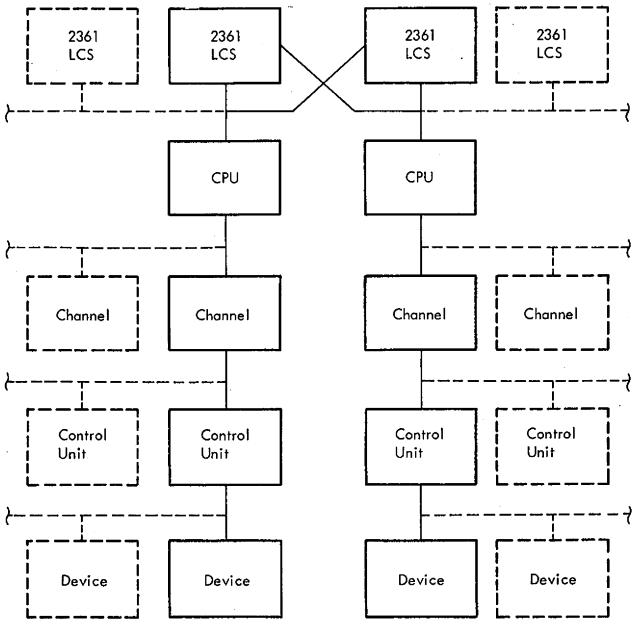


Figure 1-8. Shared LCS Feature as Multisystem Connector

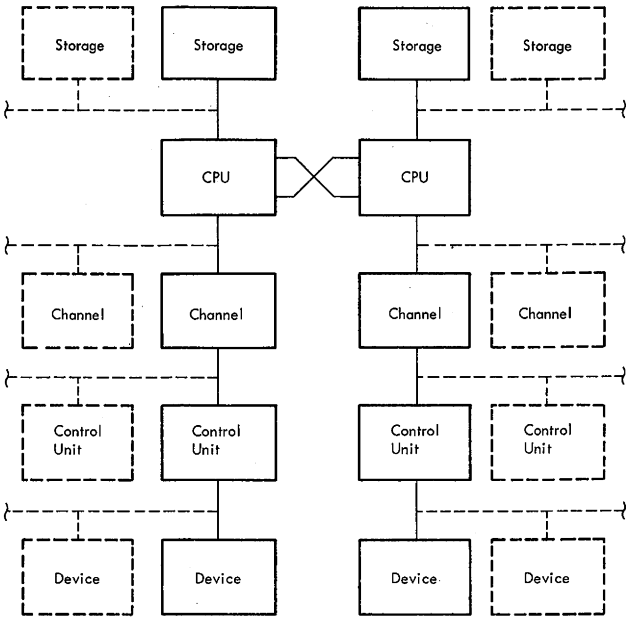


Figure 1-9. Direct Control Feature as Multisystem Connector

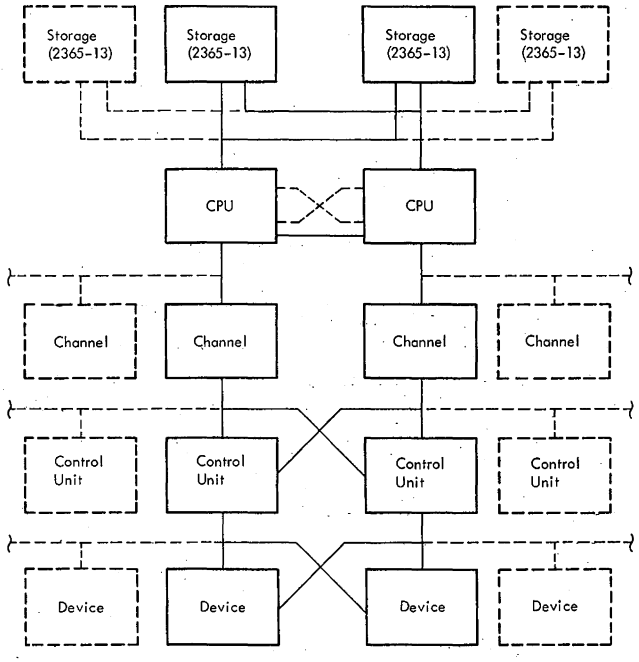


Figure 1-10. Multiprocessing System/360 Model 65

To understand the operation of the CPU, it is necessary to become familiar with the system coding. Accordingly, this section discusses: (1) the hexadecimal (hex) number system, (2) the 8-bit zoned character codes, (3) the instruction formats and operand designations, and (4) the various data formats.

HEXADECIMAL NUMBER SYSTEM

- System uses 16 symbols: 0–9, A–F.
- Base of system is 16.
- System is shorthand notation for binary numbers.
- Four binary bits are represented by one hex symbol.
- Byte is represented by two hex symbols.

Binary numbers have approximately 3.3 times as many terms as their decimal counterparts. This increased length presents a problem when talking or writing about binary numbers. A long string of 1's and 0's cannot be effectively spoken or read. A shorthand system is necessary, one that has a simple relationship to the binary system and that is compatible with the basic eight-bit byte used in the CPU. The hexadecimal (hex) number system meets these requirements.

The hex system has 16 symbols: 0–9, A–F. Counting is performed as in the decimal and binary systems. When the last unique symbol (F) is reached, a 1 is placed in the next position to the left and counting resumes with a 0 in the original position, as follows:

0	10	20	A0
1	11	21	A1
2	12	22	A2
3	13	23	↓
4	14	↓	
5	15		
6	16		and so on
7	17		
8	18		
9	19		
A	1A	9A	
B	1B	9B	
C	1C	9C	
D	1D	9D	
E	1E	9E	
F	1F	9F	

One hex symbol can represent four binary bits. Thus the 8-bit binary byte, in turn, can be represented by two hex symbols. The relationship between the hex, binary, and decimal systems is as follows:

Hex	Binary	Decimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

The important relationship to remember is that four binary positions are equivalent to one hex position.

Hex numbers are represented in the same manner as decimal and binary numbers, except that the base is 16. The terms of the number represent the coefficients of the ascending powers of 16. For example, consider the hex number 257 (decimal equivalent equals 599):

$$257 = (2 \times 16^2) + (5 \times 16^1) + (7 \times 16^0)$$

$$= (2 \times 256) + (5 \times 16) + (7 \times 1)$$

$$= 512 + 80 + 7$$

$$= 599_{10}$$

EIGHT-BIT ZONED CHARACTER CODES

All I/O devices requiring a zoned character code use either the Extended Binary-Coded-Decimal Interchange Code (EBCDIC) or the USA Standard Code for Information Interchange extended to eight bits (USASCII-8). The EBCDIC and USASCII-8 codes for the hex characters 0–F are listed below. For charts showing the complete EBCDIC and USASCII-8 codes with associated graphic characters, refer to the SRL, *IBM System/360 Principles of Operation*, Form A22-6821-6. The codes do not have a symbol defined for all 256 eight-bit codes. To represent codes that do not

have a defined symbol, two hex terms (representing four bits each) may be used instead of the eight-bit code.

Hex Code	Printed Graphic	EBCDIC Code	USASCII-8 Code
0000	0	1111 0000	0101 0000
0001	1	1111 0001	0101 0001
0010	2	1111 0010	0101 0010
0011	3	1111 0011	0101 0011
0100	4	1111 0100	0101 0100
0101	5	1111 0101	0101 0101
0110	6	1111 0110	0101 0110
0111	7	1111 0111	0101 0111
1000	8	1111 1000	0101 1000
1001	9	1111 1001	0101 1001
1010	A	1100 0001	1010 0001
1011	B	1100 0010	1010 0010
1100	C	1100 0011	1010 0011
1101	D	1100 0100	1010 0100
1110	E	1100 0101	1010 0101
1111	F	1100 0110	1010 0110

INSTRUCTION CODING

The Model 65 uses the Universal instruction set, which enables the CPU to execute fixed-point, floating-point, decimal, logical, branching, status switching, and I/O instructions. Two more status switching instructions, Read Direct and Write Direct, may be implemented by installing the Direct Control feature.

The Universal instruction set uses five instruction formats: RR, RX, RS, SI, and SS. Operands are designated as first, second, or third operands. For addressing purposes, the operands are grouped into three classes:

1. Effectively addressed operands in main storage.
2. Immediate operands in the instruction format.
3. Operands in local storage (LS).

Instruction Formats

- Five instruction formats are available: RR, RX, RS, SI, and SS.
- Halfword is basic building block for instruction.
- Instructions are made up of 1, 2, or 3 halfwords.
- First halfword contains 8-bit op code and, depending on format:
  4-bit LS address for operand, or operand address component.
  4-bit mask.
  8-bit immediate operand.
  4-bit or 8-bit length fields.
- Second and third halfwords contain:
  4-bit LS address for operand address component.
  12-bit displacement.

Five instruction formats are available, denoted by the format codes RR, RX, RS, SI, and SS. The format codes express, in general terms, the operation to be performed. RR denotes a register-to-register operation; RX, a register-to-indexed-storage operation; RS, a register-to-storage operation; SI, a storage and immediate-operand operation; SS, a storage-to-storage operation. The Universal instruction set for the Model 65 may be divided into seven classes; the breakdown by format is as follows:

Instruction Class	Format
Fixed-Point	RR, RX, RS
Floating-Point	RR, RX
Decimal	SS
Logical	RR, RX, RS, SI, SS
Branching	RR, RX, RS
Status Switching	RR, SI
I/O	SI

The basic unit of length for instructions is the halfword, consisting of two bytes. The length of an instruction format can be 1, 2, or 3 halfwords. It is related to the number of initial main storage references necessary for the operation. An instruction making no reference to main storage (RR) consists of one halfword; an instruction making one reference (RX, RS, or SI) consists of two halfwords; an instruction making two references (SS) consists of three halfwords. All instructions must be located in main storage on an integral boundary for halfwords. The five formats are shown in Figure 1-11.

For purposes of describing the execution of instructions, operands are designated as first, second, or third operands, referring to the manner in which the operands participate in the operation. The operand to which a field in an instruction format pertains is denoted by the number following the letter designation of the field; for example, the R1 field is the address of an LS register containing the first operand; R2, the second operand.

As shown in Figure 1-11, the first halfword of each format consists of two parts. The first byte contains the operation code (op code), which identifies the operation to be performed. Bits 0 and 1 specify the format, bits 2 and 3 specify the class of instruction, and bits 4 through 7 identify the instruction within the class. The second byte of the first halfword is used as either two four-bit fields or a single eight-bit field. The fields and the information contained within the fields are as follows:

1. R1, R2, and R3: four-bit address of an LS register containing the first, second, and third operands, respectively.

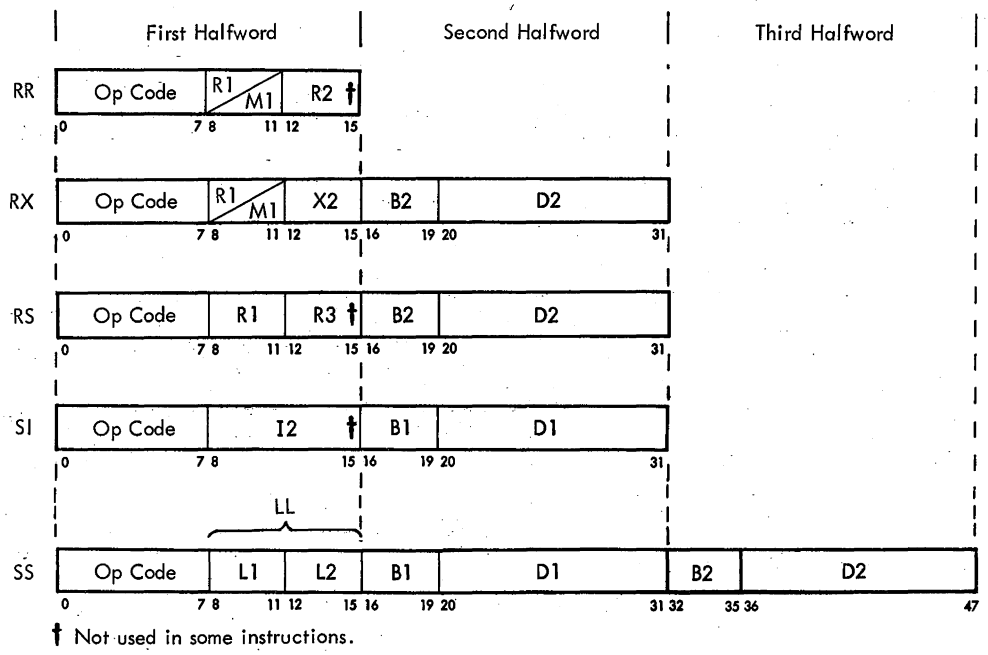


Figure 1-11. Instruction Formats

2. M1: four-bit mask used in some branching instructions.
3. X2: four-bit address of an LS register containing the index value used in generating the effective second operand address.
4. I2: eight-bit byte of immediate data (second operand).
5. L1 and L2: four-bit length (up to 16 bytes) of first and second decimal VFL operands, respectively.
6. LL: eight-bit length field (up to 256 bytes) for logical VFL operands.

The second and third halfwords always contain the same information: a four-bit address of an LS register containing the base value to be added to the following 12-bit displacement field.

Operand Addressing

For addressing purposes, operands are grouped into three classes: (1) effectively addressed operands in main storage, (2) immediate operands in the instruction format, and (3) operands in LS.

Effectively Addressed Operands

- Operands are either fixed-length or VFL.
- Fixed-length operands are located on integral boundary.
- Length of VFL operand is specified by L or LL field.
- L and LL fields denote number of bytes to right of addressed byte.
- Effective operand address is sum of 24-bit base address, 12-bit displacement, and 24-bit index value.
- Base address and index value are located in LS.
- Displacement is located in instruction format.

An effectively addressed operand is selected from a main storage location not related to the location of the instruction referring to it. It is specified by means of a main storage address. When the operand consists of more than one byte, the address gives the location of the first byte; subsequent bytes are located in the next-higher addressed byte locations. Both the first and second operands of an instruction can be effectively addressed.

Effectively addressed operands can be of either fixed length or variable field length. The length of VFL operands, in terms of the number of bytes to the right of the addressed byte, is specified by the L or LL field of the instruction. The LL field can be eight bits long and thus can specify a maximum operand field length of 256 bytes.

In the instruction format, an effectively addressed operand is specified by a base address, a displacement, and, in some cases, an index value. The base address and the index value are contained in LS general-purpose registers addressed by the B and X fields, respectively, of the instruction. The registers contain 32 bits, the low-order 24 of which constitute an unsigned address component (base address or index value). The high-order eight bits of the register are ignored. The 24-bit base address is included in every address computation. The 24-bit index value is included in the address computation as specified by the RX instruction format.

The displacement value is a 12-bit number contained in the D-field of the instruction. It is included in every address computation. The displacement provides for relative addressing up to 4095 bytes beyond the base address.

In computing the effective operand address, the base address and the index value are treated as 24-bit positive binary integers having no sign position. The displacement is

similarly treated as a 12-bit positive binary integer. The three numbers are added. Because every operand address includes a base address, the sum is always 24 bits long. Any overflow above the 24 low-order bits of the sum is ignored, causing a lower address to be generated. If this lower address is *above* the maximum available storage, an addressing program interruption occurs. If the lower address is available the CPU accesses that location.

An instruction may contain zeros in the B, X, or D field. In the case of the B and X fields, a zero does not denote the address of LS general-purpose register 0, but indicates that base and index values of zero are to be used in generating the effective operand address. Similarly, a D field of zero indicates a displacement of zero.

Fixed-length fields, halfwords, words, and doublewords, must be located in main storage on an integral boundary for that length field. A boundary is called "integral" for a field when its storage address is a multiple of the length of the field in bytes (Figure 1-12). For example, words (four bytes) must be located in main storage so that their address is a multiple of the number 4. A halfword (two bytes) must have an address that is a multiple of 2, and doublewords (eight bytes) must have an address that is a multiple of 8.

Main storage addresses are expressed in binary form. Therefore, integral boundaries for halfwords, words, and doublewords can be specified only by binary addresses in which 1, 2, or 3 of the low-order bits, respectively, are zero (Figure 1-12). Thus, integral boundaries for words are binary addresses in which the two low-order bit positions are zero; for example, 00000, 00100, 01000, and 01100.

VFL fields are not limited to integral boundaries, but may start on any byte location.

Immediate Operands

Immediate operands are contained in SI instructions for logical operations. They are one byte (bits 8–15) long, serve as the second operand, and are designated I2.

Operands in Local Storage

- LS registers are addressed by four-bit R-field in instruction format.
- LS GPR's are addressed 0–15.
- LS FPR's are addressed 0, 2, 4, and 6.
- For fixed-point doubleword operands, the register address implies use of a pair of adjacent registers.

Fixed-point and floating-point operands may be located in the 16 general-purpose registers (GPR's) and the 4 floating-point registers (FPR's), respectively, of LS. The registers are addressed by a four-bit field in the instruction, designated the R-field. The GPR's are designated by addresses 0–15, whereas the FPR's are identified by addresses 0, 2, 4, and 6. (When an FPR is designated by any other address, a specification program interruption occurs.) The op code of the instruction implies whether the GPR's or the FPR's are to be used.

The GPR's are 32 bits (one word) in length. Fixed-point operands normally have an implied length of one word. In some operations, one register address implies the use of a pair of adjacent GPR's, thus providing a doubleword. For these instructions, the addressed register (say R1) contains the high-order operand bits and must have an even address, and the implied register (R1 + 1) contains the low-order operand bits and has the next higher address.

The FPR's are 64 bits or a doubleword in length, and can contain either a short (one word) or a long (doubleword) floating-point operand. A short operand occupies the high-order bits of an FPR; the low-order bits are ignored.

DATA FORMATS

Data can be numeric, alphabetic, or logical, and fixed or variable in length. Numeric data is distinguished as fixed-

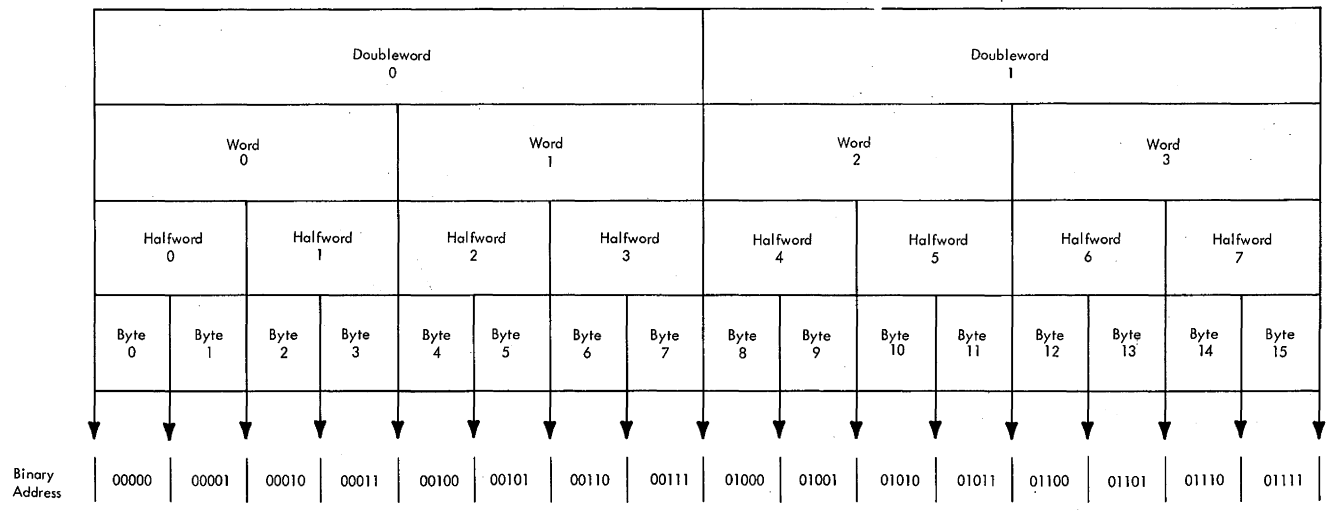


Figure 1-12. Main Storage Integral Boundaries

point, floating-point, or decimal. The data may be divided into four classifications:

1. Fixed-point numbers, having a binary radix and a fixed length, usually a word or a halfword.
2. Floating-point numbers, represented by a 7-bit characteristic and a signed hex fraction, occupying either a word or a doubleword.
3. Decimal numbers, represented by four-bit binary-coded-decimal (BCD) digits, usually packed two digits to a byte, and variable in length.
4. Logical information, represented by eight-bit zoned character codes, and fixed or variable in length.

Fixed-Point Data

Fixed-point instructions are available for loading, adding, subtracting, comparing, multiplying, and dividing. A pair of conversion instructions, Convert to Binary and Convert to Decimal, provides transition between decimal and binary radix without the use of tables.

The basic fixed-point operand is the 32-bit binary word. To improve performance and storage utilization, 16-bit halfword operands may be specified in most operations. In both lengths, bit position 0 holds the sign of the number, with the remaining bit positions designating the magnitude of the number. To preserve precision, some products and all dividends are 64 bits long.

Because a 24-bit address can be accommodated in the 32-bit word, fixed-point instructions can be used both for integer operand arithmetic and for address computation. This combined usage provides economy and permits the entire fixed-point instruction set to be used for address computation. Thus, multiplication, shifting, and logical manipulation of address components are possible.

Number Representation

- Positive numbers are represented in true binary form with sign of 0.
- Negative numbers are represented in 2's complement notation with sign of 1.

All fixed-point operands are treated as signed integers. Positive numbers are represented in true binary form with a sign bit of 0. Negative numbers are represented in 2's complement notation with a sign bit of 1. In all cases, the bits between the sign bit and the leftmost significant bit of the integer are the same as the sign bit; i.e., all 0's for positive numbers, all 1's for negative numbers. Therefore, when an operand must be extended with high-order bits, each nonsignificant bit is made equal to the sign bit.

Negative fixed-point numbers are formed in 2's complement notation by complementing the true binary representation of the number and adding 1. For example, to convert the decimal number +26 to 2's complement form (−26), proceed as follows:

	S 1 ← Integer → 31
Decimal +26 to true binary form:	0 0000000 00011010
Complement the binary number:	1 1111111 11100101
Add 1:	1
Result is -26 (2's complement form):	1 1111111 11100110

The result is equivalent to subtracting the number 00000000 00011010 from 1 00000000 00000000.

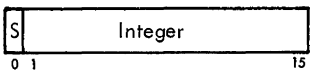
The largest positive number consists of a sign bit of 0 with all 1's in the integer field, whereas the largest negative number consists of a sign bit of 1 with all 0's in the integer field:

	S 1 ← Integer → 31	Decimal Number
Largest positive number:	0 1111111 11111111	= +2,147,483,647
Smallest positive number:	0 0000000 00000000	= 0
Smallest negative number:	1 1111111 11111111	= -1
Largest negative number:	1 0000000 00000000	= -2,147,483,648

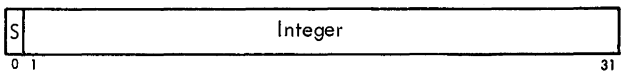
Formats

Fixed-point numbers occur in 16-bit halfword, 32-bit word, or 64-bit doubleword operands. Bit 0 is the sign bit, and the remaining bits make up the integer:

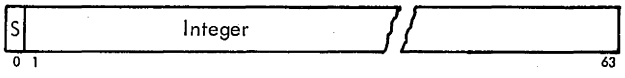
Halfword operand



Word operand



Doubleword operand



In LS, fixed-point operands are a word long. In some operations, such as multiply, divide, and shift, the operand may be a doubleword. The doubleword operands are located in a pair of adjacent 32-bit GPR's and are addressed by an even address that refers to the leftmost (lower-

addressed) register of the pair. In this case, the sign-bit position of the rightmost register *may* contain an operand bit instead of a sign bit. The sign-bit position of the leftmost register must *always* contain a sign bit.

In main storage, fixed-point data may be a halfword, a word, or a doubleword. This data must be located on integral storage boundaries for these units of information. When a halfword operand is fetched from main storage, it is extended to a full word. The original signed integer occupies bits 16 through 31, and is operated on as a full word. When the operand is extended to a full word, bits 0 through 15 assume the state of the original sign bit, now in bit 16.

Floating-Point Data

- Scientific and engineering calculations require that very small and very large numbers be represented.
- Scientific notation uses powers of 10 to simplify calculations with high and low magnitude numbers.
- Floating-point instructions operate upon data that uses powers of 16 to represent numerical quantities.
- Quantity expressed by floating-point number is product of signed hex fraction and 16 raised to power designated by exponent.

When performing calculations for scientific and engineering work, very small or very large numbers must sometimes be represented. Consider the problem of a nuclear physicist who wants to write an equation that contains the value for the mass of a subatomic particle. If he wrote the number as a decimal fraction, he would have to put down a decimal point followed by more than 20 zeros and a few numerals. At the opposite extreme, an astronomer may be calculating distances between objects that may be millions of millions of miles apart.

To overcome the pointless effort of having to write many zeros when working with such numbers, a mathematical method using powers of 10 is used. This method is called scientific notation. For example, in scientific notation, the mass of an electron can be written as 9.107×10^{-28} grams; and the astronomical distance of one light year can be written as 5.878×10^{12} miles.

Consider the parts of a number represented in scientific notation. The example of the mass of an electron, for instance, can be divided into three distinct parts: (1) a signed mixed number (+9.107) multiplied by, (2) 10 raised to the power designated by, (3) a signed exponent (−28). By changing the position of the decimal point and adjusting the exponent to compensate for the change, the number can also be written as a fraction times a power of 10 ($+.9107 \times 10^{-27}$), or an integer times a power of 10 ($+9107 \times 10^{-31}$).

For scientific and engineering applications where quantities may be of the magnitudes mentioned above, the CPU has instructions for handling them. These instructions, called floating-point instructions, manipulate data in a manner similar to scientific notation. However, because the CPU is primarily a binary machine in which numbers can be easily worked upon in hex (four-bit) units, a quantity is represented as a hex number times a power of 16 rather than as a decimal number times a power of 10. Except for this difference in base, floating-point notation is similar to scientific notation; the same rules of algebra apply to powers of 16 as to powers of 10.

Number Representation

- Fraction represents number expressed in hex digits.
- Characteristic specifies exponent to which 16 is raised.
- Characteristic is expressed in excess 64 notation; range is −64 to +63.
- Radix point is to left of high-order hex digit.
- True zero result yields positive sign.

A floating-point number contains the same components as a number written in scientific notation. However, due to the nature of the computer, the format is different and certain rules are imposed upon the way a floating-point number may be represented. The number to be multiplied by a power of 16 is a hex fraction with a fixed length, and the 16 is understood rather than shown. Therefore, as represented in the CPU, a floating-point number consists of a sign, which is the sign of the fraction, a signed exponent, called a characteristic, and a hex fraction. The quantity expressed by this number is the product of the fraction and 16 raised to the power designated by the characteristic (exponent).

The fraction of a floating-point number is expressed in hex digits. The radix point (representing the base 16) of the fraction is assumed to be immediately to the left of the high-order fraction digit. To provide the proper magnitude for a floating-point number, the fraction is considered to be multiplied by a power of 16 (fraction x 16^n power). The characteristic (bits 1–7) indicates the power (exponent). Bit 0 designates the sign of the fraction; it is a 0 if the fraction is a positive number and a 1 if the fraction is negative. Both positive and negative quantities are in true form, with the difference indicated by the sign.

The exponent may also be either a positive or negative number. For example, $-.A8 \times 16^{-2}$ is an example of a floating-point number with a negative fraction and a negative exponent. Therefore, to represent both positive and negative exponents, *excess 64 notation* is used. Excess 64 notation simply means that *+64 (+40 hex) is added to the true exponent* and the value obtained is used as the

characteristic. Therefore, the characteristic varies around a base of 64; i.e., an exponent of 0 is represented by a characteristic of 64, a positive exponent is represented by a characteristic greater than 64, and a negative exponent is represented by a characteristic less than 64. In the example just given, for instance, −2 is the exponent. Adding +64 to −2 yields +62, which is the value of the characteristic in excess 64 notation.

Performing the same calculation in binary gives the following results:

```
Bits  0  1  2  3  4  5  6  7

      S  1  1  1  1  1  1  0     2's complement of 2 (-2)
      S  1  0  0  0  0  0  0     +64 (+40 hex)
      S  0  1  1  1  1  1  0  = +62 (+3E hex)
   Carry
```

If the exponent were +2 (positive exponent), adding +64 yields +66, the characteristic value placed into bits 1–7. In binary, the addition is as follows:

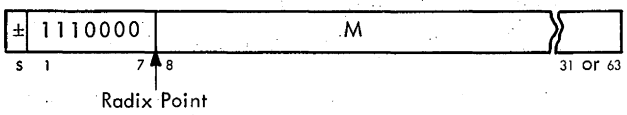
```
Bits  0  1  2  3  4  5  6  7

      S  0  0  0  0  0  1  0     +2 exponent
      S  1  0  0  0  0  0  0     +64 (40 hex)
      S  1  0  0  0  0  1  0  = +66 (42 hex)
```

Note in these examples that a negative exponent in excess 64 notation caused bit 1 (high-order bit of the characteristic) to be a 0, and a positive exponent caused bit 1 to be a 1. This rule holds true for the range of positive and negative exponents, as can be seen in Table 1-2. The table also shows that because only seven binary bits (1–7) are available to represent the characteristic in floating-point format, the most negative exponent that can be expressed is −64 and is represented by an all-zero characteristic. The most positive exponent is +63, and is represented by all 1's (7F hex). Midpoint between these two extremes is a 0 exponent, which is represented by a +64 (+40 hex) characteristic.

Table 1-2. Characteristic Notation

Excess 64 Notation			
Binary	Decimal	Hex	Exponent
0000000	0	0	-64
0000001	1	1	-63
↓	↓	↓	↓
0111111	63	3F	-1
1000000	64	40	0
1000001	65	41	+1
↓	↓	↓	↓
1111110	126	7E	62
1111111	127	7F	63

For another example of converting an exponent to an excess 64 characteristic, assume the value of $\pm M \times 16^{48}$ must be stated in excess 64 notation. The characteristic of the fraction then becomes 48 + 64 = 112 (0110000 + 1000000 = 1110000). The floating-point number thus takes the following form:



To illustrate how numbers are represented in floating-point format, assume that the decimal number 149.25 is to be converted to a floating-point short operand. This conversion is accomplished as follows:

1. The number is separated into a decimal integer and a decimal fraction:

   149.25 = 149 plus 0.25

2. The decimal integer is converted to its hex representation:

   $149._{10} = 95_{16}$

3. The decimal fraction is converted to its hex representation:

   $0.25_{10} = 0.4_{16}$

4. The integral and fractional parts are combined and expressed as a fraction times a power of 16 (exponent):

   $95.4_{16} = (0.954 \times 16^{2})_{16}$

5. The characteristic is developed from the exponent and converted to binary:

   Excess 64 + exponent = characteristic
   64 + 2 = 66 = 1000010

6. The fraction is converted to binary and grouped hexadecimally:

   $.954_{16} = 1001\ 0101\ 0100$

7. The characteristic and the fraction are placed in the short precision format; the sign position contains the sign of the fraction:

S	Characteristic	Fraction
0	1000010	1001 0101 0100 0000 0000 0000

Other examples follow:

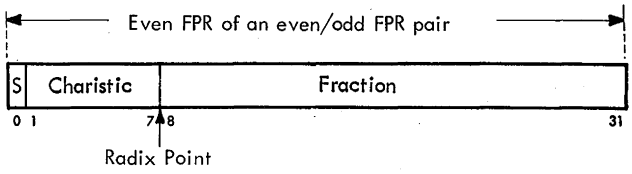
Number	Powers of 16	S	Charistic	Fraction
1.0	$= +1/16 \times 16^{1}$	= 0	1000001	0001 0000 ≀≀ 0000
0.5	$= +8/16 \times 16^{0}$	= 0	1000000	1000 0000 ≀≀ 0000
1/64	$= +4/16 \times 16^{-1}$	= 0	0111111	0100 0000 ≀≀ 0000
0.0	$= +0 \times 16^{-64}$	= 0	0000000	0000 0000 ≀≀ 0000
-15.0	$= -15/16 \times 16^{1}$	= 1	1000001	1111 0000 ≀≀ 0000
2×10^{-78}	$= +1/16 \times 16^{-64}$	= 0	0000000	0001 0000 ≀≀ 0000
7×10^{75}	$= (1-16^{-6}) \times 16^{63}$	= 0	1111111	1111 1111 ≀≀ 1111

Formats

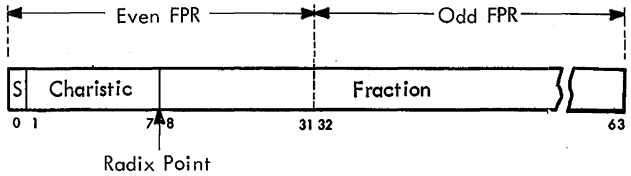
- Data format consists of 1-bit sign, 7-bit characteristic, and 24- or 56-bit fraction.
- Results are 32 bits (short operand) or 64 bits (long operand) long.
- Multiply product is always 64 bits.
- Guard digit is retained.

Floating-point data is represented in the CPU in one of two fixed-length formats, depending upon whether a full-word short operand or a doubleword long operand is desired. Both formats may be used in main storage and in the eight LS FPR's used exclusively by floating-point instructions. The data formats for short and long operands are:

Short Operand



Long Operand



For both formats, the first bit position is the sign bit and the subsequent seven bit positions constitute the characteristic. The following 24 or 56 bits represent the fraction; short operand fractions are 24 bits or 6 hex digits; long operand fractions are 56 bits or 14 hex digits.

When short operands are specified, the results are usually 32-bit floating-point words; the odd FPR of the even/odd pair of FPR's does not participate in the operation and remains unchanged. However, in multiply instructions, the product occupies two FPR's (64 bits).

When long operands are specified, all operands and results are 64-bit floating-point doublewords.

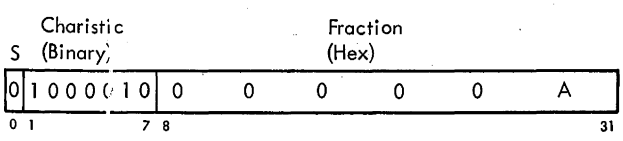
Although final results have 6 or 14 hex fraction digits, intermediate results in addition, subtraction, and compare operations may extend to 7 or 15 fraction digits. This extra digit, called the *guard digit*, occurs when one of the fractions is shifted right (as part of the characteristic equalization process that occurs during execution of add-type floating-point instructions; see Chapter 3, Section 3, *Add, Subtract, and Compare*). The guard digit increases the accuracy of the final result if normalization occurs. In normalization, the fraction is shifted left until a significant digit appears in the high-order digit position of the fraction; thus the guard digit becomes part of the 6 or 14 hex digits of the final result. This saving of the guard digit becomes especially significant where the high-order 6 or 14 digits of the intermediate result are all zeros.

Normalization

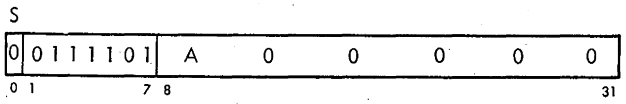
- Normalized fraction has nonzero, high-order hex digit; unnormalized fraction has one or more leading hex zeros.
- Characteristic is adjusted on normalization cycles.
- Postnormalization is normalization of final result.
- Prenormalization is normalization before result computation.
- Results are shifted right if fraction overflow occurs.

A quantity can be represented with the greatest precision by a floating-point number of a given fraction length when that number is normalized. A normalized floating-point number has a nonzero high-order hex fraction digit. If one or more high-order fraction digits are zero, the number is said to be unnormalized. Normalization consists of shifting the fraction left until the high-order hex digit is nonzero and reducing the characteristic by the number of hex digits shifted. A zero fraction cannot be normalized, and its associated characteristic therefore remains unchanged when normalization is called for.

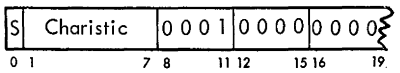
An example of an unnormalized floating-point number in numerical terms is $.00000A_{16} \times 16^{2}$. To convert this number to its normalized form, the number must be shifted five hex digits to the left and, because five shifts are necessary, five is subtracted from the exponent. The result is $.A00000_{16} \times 16^{-3}$. In the CPU, the original number would have the following format:



After normalization, the format would be:



Because normalization applies to hex digits, the three high-order bits of a normalized number may be zero. For example, if the high-order digit of a fraction is a hex 1, normalization will not occur although normalization is specified and bits 8–10 = 000:



Floating-point operations are performed with or without normalization. Addition and subtraction may be specified either way depending upon the instruction op code. The multiply, divide, and halve instructions always specify normalization. The load, compare, and store instructions specify unnormalized results. Normalization usually occurs when the intermediate arithmetic result is changed to the final result. This function is called *postnormalization*. In multiplication and division, the operands are normalized before the arithmetic process. This function is called *prenormalization*.

When an operation is performed without normalization, high-order zeros in the result fraction are not eliminated. In both normalized and unnormalized operations, the initial operands need not be in normalized form.

Decimal Data

- Operands and results are located in main storage.
- VFL is 1–16 bytes.
- Four-bit BCD digits are packed two to a byte for arithmetic.
- Unpacked (zoned) format is used for transmitting data to I/O devices.
- Pack and Unpack instructions are provided.

Decimal instructions are designed for operations requiring few computational steps between the source input and the documented output. Processing of this type is frequently found in commercial applications. Because of the limited number of arithmetic operations performed on each item of data, radix conversion from decimal to binary and back to decimal is not justified, and the use of registers for intermediate results yields no advantage over storage-to-storage processing. Hence, in the Model 65, decimal instructions are provided and both operands and results are located in main storage. Decimal instructions include addition, subtraction, multiplication, division, and comparison.

Decimal arithmetic operates on data in the packed format, in which two four-bit BCD digits are packed two to a byte. They appear in fields of variable length (from 1 to 16 bytes) and are accompanied by a sign in the rightmost four bits of the low-order byte. The use of packed digits within a byte and of variable-length fields within storage results in efficient use of storage and in increased arithmetic performance.

Decimal numbers may also appear in a zoned format for use with I/O devices operating in that format. The zoned format is not used in decimal arithmetic operations, but only for transmitting data to the I/O device. Instructions are provided for packing and unpacking decimal numbers so that they may be changed from the zoned (unpacked) to the packed format and vice versa.

Processing takes place right to left between main storage locations, except in the divide operation which is processed left to right. All decimal instructions use the two-address SS format. Each address specifies the leftmost byte of an operand. Associated with this address is a length field, indicating the number of additional bytes that the operand extends beyond the first byte.

Number Representation

Numbers are represented as right-aligned true integers with a plus or minus sign. Decimal digits 0–9 are represented in the four-bit BCD form by 0000 through 1001, respectively. Codes 1010–1111 (10–15) are not valid as digits and are reserved for sign codes: 1010, 1100, 1110 and 1111 represent a plus; 1011 and 1101 represent a minus.

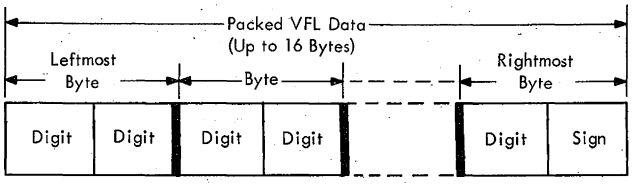
Digit	Code	Sign	Code
0	0000	+	1010
1	0001	–	1011
2	0010	+	1100
3	0011	–	1101
4	0100	+	1110
5	0101	+	1111
6	0110		
7	0111		
8	1000		
9	1001		

All valid sign codes are recognized in decimal operations; however, the appropriate sign codes (and zone codes for the Unpack instruction) generated during the operation depend on the character set specified by PSW(12). If PSW(12) = 0, EBCDIC is selected, and code 1100 is generated for a plus sign, code 1101 is generated for a minus sign, and code 1111 is generated for a zone. If PSW(12) = 1, USASCII-8 is selected, and code 1010 is generated for a plus sign, code 1011 is generated for a minus sign, and code 0101 is generated for a zone.

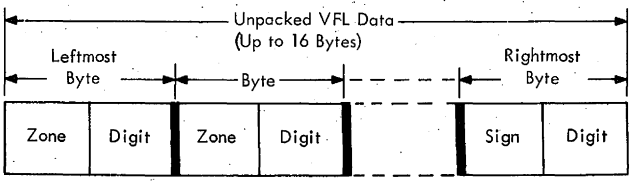
Formats

Decimal operands reside in main storage only. The operand field length may range from a minimum of one byte to a maximum of 16 bytes. The operands need not occupy the entire field length but are always right-aligned in the field; i.e., the sign of the operand is always in the rightmost byte of the specified field. This rightmost byte contains the lowest-order operand digit and the operand sign. All decimal instructions (except Divide) process the operands from low order to high order, or from right to left between main storage locations.

Data may be in the packed or unpacked (zoned) format. In the packed format, two four-bit BCD digits are placed adjacently in an eight-bit byte, except for the rightmost (low-order) byte of the field. In the low-order byte, a four-bit sign (sign of the decimal number) is placed to the right of the decimal digit.



In the unpacked or zoned format, a decimal digit normally occupies the four low-order bits of a byte, the *numeric*. The four high-order bits of a byte are called the *zone*. An exception is the rightmost byte in the field, where the sign of the decimal number occupies the zone position.



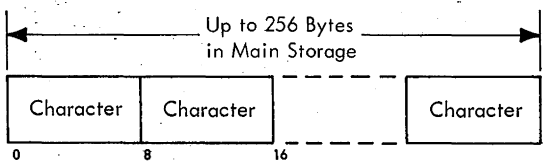
Logical Data

- Data is fixed-length or VFL.
- One byte of immediate data is held in some instruction formats.

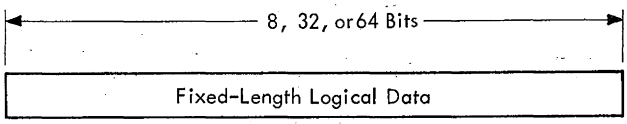
The logical instructions provide for moving, comparing, bit testing, bit connecting, translating, editing, and shifting operations. Except for the editing instructions, data is not treated as numbers. Editing converts packed decimal digits into alphanumeric characters; the digits, signs, and zones are recognized and generated as for decimal instructions.

Data resides in main storage or in LS, or is contained in the instruction format. The data may be a single byte, a word, a doubleword, or variable in length. When two operands participate in the operation, they have equal length, except in the editing instructions. The data format depends on the type of operation performed:

1. In storage-to-storage operations, data has a VFL format, starting at any byte address and continuing for a maximum of 256 bytes; it is processed left to right.



2. In storage-to-register operations, the main storage data may be either a word or a byte. The word must be located on a word boundary; that is, the low-order two bits of its address must be 0's. Data in GPR's normally occupies all 32 bits. Bits are treated uniformly, and no distinction is made between sign and numeric bits. In a few operations, only the low-order eight bits of the register participate, leaving the remaining 24 bits unchanged. In some shift operations, 64 bits of an even/odd pair of GPR's participate.



3. In operations which introduce data directly from the SI format instruction as an immediate operand, data is restricted to an eight-bit byte. Only one byte may be introduced per instruction, and only one byte from main storage takes part in the operation.

This section discusses the supervisor program, the eight program states, the program status word, interruptions and exceptional conditions, and the initiation and control of I/O operations.

CONTROL PROGRAM

Because internal processing speeds of data processing systems have increased without a corresponding reduction in the time required by the operator to load programs and to manually insert data, the setup time has become a more significant factor in system operation. To reduce this setup time, during which the system is idle, control programs were devised to control execution of problem programs. The simplest control program, which shares main storage with the problem programs, controls the loading of problem programs; the problem programs handle their own I/O operations (Figure 1-13, A). Operation is as follows:

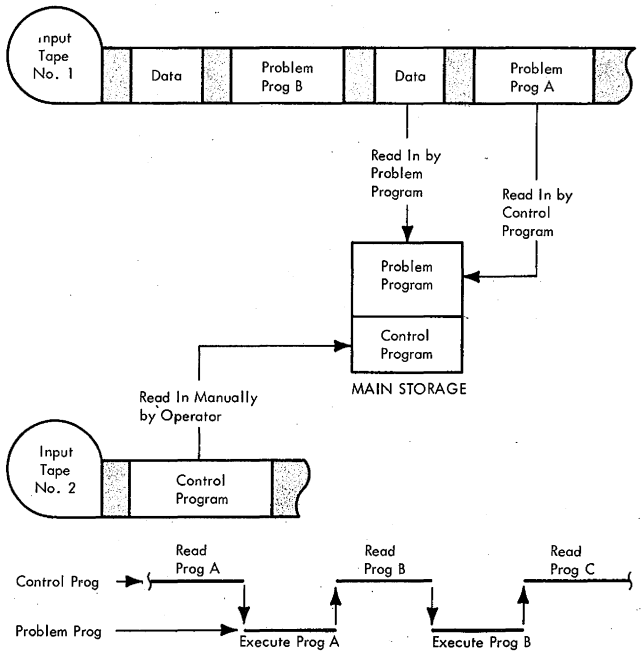
1. An input device is prepared containing the problem programs and their associated data.
2. The operator loads the control program into main storage.
3. The control program loads in the first problem program and then passes control to the problem program.
4. The problem program reads in its data and performs its assigned task.
5. When the problem program is finished, it passes control back to the control program.
6. The control program then loads in the next problem program and passes control to it.
7. This operation continues until all programs are executed.

Notice three things about the use of a control program in the preceding example:

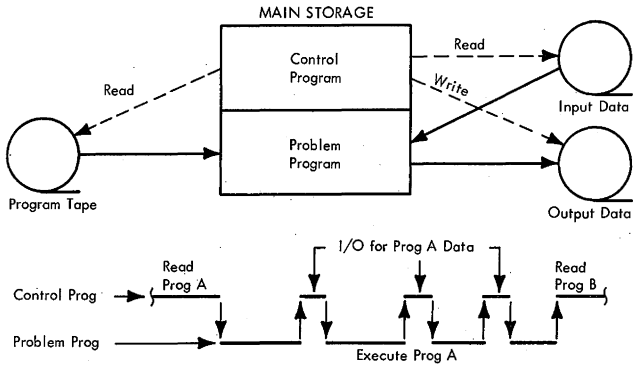
1. The system never halts between jobs.
2. The control program remains in main storage as the problem programs are executed.
3. The control program serves only as a link between jobs; its only function is to bring in a new problem program as each job is finished.

The simple control program discussed above has limited functions. Other functions can be included, such as initiation and control of I/O operations. Because the problem program is designed mainly to process data, the read and write operations necessary to transfer data between the I/O devices and main storage can be handled by the control program. Each operation to be handled by the control program may consist of many instructions. Besides telling the I/O device to start, for example, the instructions check for error conditions and I/O device status.

In this function, control passes back and forth between the problem and control programs *during* execution of the problem program (Figure 1-13, B). The control program not only reads in the problem programs but also handles



A. Simple Control Program - Loads Problem Programs



B. Expanded Control Program - Loads Problem Programs, and Handles I/O Operations for Problem Program Data

Figure 1-13. Examples of Control Program Functions

the I/O data operation during execution of the problem program. The *problem* program transfers control to the *control* program whenever an I/O operation is necessary.

The control program can be given other functions as well. However, the more functions a control program has, the more main storage space it requires. This problem is solved by placing into main storage only those key parts of the control program that ensure continuous, coordinated operation of the system. This portion is called the *supervisor*. The remaining parts of the control program are placed on a high-speed, direct-access I/O device, such as a disk storage unit, and are brought into main storage as they are required to perform a specific function.

In the System/360, the supervisor is the control center of the operating system. Its primary function is to perform a variety of services for other parts of the system including problem programs. It coordinates and controls the performance of these services to yield efficient and coordinated use of the physical and programming facilities of the system. The supervisor prevents programs and routines that are run on the system from interfering with one another and with operation of the control program. This control is accomplished, in part, through its use of *privileged* instructions, such as storage protection and I/O instructions, which can be executed only by the control program.

A service performed by the supervisor may be specifically requested by a program, such as a request for storage space, or it may be a service that is automatically provided when a contingency occurs, such as attempting to recover from an error condition. Among the services the supervisor may provide are:

1. Allocating main storage space required by programs during their execution.
2. Sharing areas of main storage among routines that need not be in main storage at the same time.
3. Loading programs into main storage.
4. Controlling the concurrent execution of programs and routines.
5. Scheduling and controlling I/O operations.
6. Providing standard procedures that assist in diagnosing exceptional conditions, such as underflow in floating-point arithmetic operations.
7. Keeping a running log of machine check and I/O errors for CE diagnostic use.

One of the reasons why a control program is used in the System/360 is to eliminate machine idle time. Realizing this, the designers of System/360 did not incorporate a halt instruction. Therefore, a problem program cannot issue a halt instruction when it is finished but passes control to the supervisor by means of a Supervisor Call instruction. Machine idle time is further reduced in another way. A machine check (such as an even number of bits in a byte) or a program check (such as locating a halfword operand on an odd-byte address) does *not* cause an error stop but causes an automatic branch to the supervisor, unless instructed by the programmer to be ignored. These automatic branches to the supervisor are called *interruptions*. That is, the current sequence of problem instructions is interrupted and an automatic branch (interruption) is taken to a new sequence of control instructions.

When an interruption occurs, the interruption-handling routine of the supervisor stores the status of the CPU and fetches information with which to control the CPU while it handles the interruption. The status of the CPU is contained in a doubleword called the *old* program status word (PSW). Bits of this old PSW show the cause of the interruption and the program state of the CPU at the time of the interruption. There are eight paired states, Problem/Supervisor, Operating/Stopped, Running/Wait, and Interruptable/Masked, all of which, except Stopped, are defined in the PSW. The following paragraphs define the program states and discuss the PSW before the discussion of interruptions is continued.

PROGRAM STATES

The eight program states which determine the overall CPU status differ in the way they affect the CPU functions and in the way their status is indicated. Refer to Table 1-3 for pertinent information about the program states.

Problem/Supervisor

In the Problem state, all I/O, protection, and direct control instructions are invalid as well as the Load PSW, Set System Mask, and Diagnose instructions. These instructions are called *privileged* instructions. A privileged instruction encountered in the Problem state constitutes a privileged-operation interruption and interrupts the operation. In the Supervisor state, all instructions are valid.

The CPU is switched between the Problem and Supervisor states by changing PSW(15). When PSW(15) is a 1, the CPU is in the Problem state; when a 0, the CPU is in the Supervisor state. This bit can be changed only by introducing a new PSW. Thus, the status switching for Problem/Supervisor state may be performed by an interruption operation or by a Load PSW instruction containing a new PSW with the desired value in bit 15. Because the Load PSW instruction is a privileged instruction, the CPU must be in the Supervisor state before the switch. The CPU status can also be changed between Problem and Supervisor states by issuing a Supervisor Call instruction or an initial program load (IPL). The Supervisor Call instruction causes an interruption which will load new PSW data. This new PSW data may change the state of the CPU. Similarly, the IPL introduces a new PSW. The new PSW may introduce the Problem or Supervisor state, regardless of the preceding CPU state.

Table 1-3. Program States

State	Comments
Problem	Load PSW, Set System Mask, and Diagnose instructions, and all I/O, storage protection, and direct control instructions are invalid. These instructions are termed privileged instructions.
Supervisor	All instructions are valid.
Operating	CPU processes instructions (if not in Wait state) and interruptions (if not masked off). Entered by: 1. Depressing START on system control panel. 2. Starting an IPL operation.
Stopped	Instructions and interruptions are not processed; interruptions remain pending. Execution of program is not affected by stopping CPU. Entered by: 1. Depressing STOP on system control panel. 2. Detecting equality on an address-compare-stop operation. 3. Completing one instruction when in instruction-step mode. 4. Turning power on or following a system reset.
Running	Instruction processing proceeds in normal manner.
Wait	No instructions are processed, and main storage is not addressed. The CPU waits for an I/O or external interruption to occur before executing further instructions, or for an IPL operation.
Interruptable	Interruptions are accepted.
Masked	System and machine-check interruptions remain pending, and program interruptions are ignored.

Operating/Stopped

When the CPU is in the Stopped state, instructions and interruptions are not executed. When the CPU is in the Operating state, instructions are executed as long as the CPU is not also in the Wait state. Interruptions are taken if they are not masked off. Manual operations, such as load PSW, can be used *only* when in the Stopped state. A change in the Stopped/Operating states can occur only by manual intervention or by machine malfunction. No instruction or interruption can start or stop the CPU. The CPU is placed in the Stopped state when STOP on the system control panel is depressed, detecting equality on an address-compare-stop operation, completing one instruction when in instruction-step mode, and after power is turned on or following a system reset, except during IPL. The CPU is placed in the Operating state when START on the system control panel is depressed and when an IPL is started.

Changing from the Operating state to the Stopped state occurs at the end of instruction execution and before the start of the next instruction execution. When the CPU is in the Wait state, the change from Operating to Stopped occurs immediately. All interruptions pending and not masked off are taken while the CPU is still in the Operating state. The interruptions cause an old PSW to be stored and a new PSW to be fetched before entering the Stopped state. Once the CPU is in the Stopped state, interruptions are no longer taken but remain pending.

Running/Wait

In the Wait state, no instructions are processed and main storage is not addressed. In the Running state, instruction fetching and execution proceed in the normal manner. The CPU status is switched between the Wait and Running states by PSW(14). When PSW(14) is a 1, the CPU is in the Wait state; when a 0, in the Running state. This bit can only be changed by introducing a new PSW. Thus, switching from the Running to the Wait state may be achieved by the privileged instruction Load PSW, by an interruption such as given by a Supervisor Call instruction, or by an IPL. Switching from the Wait to the Running state may be achieved by an I/O or external interruption or by an IPL. The new PSW may introduce the Wait or Running state regardless of the preceding CPU state.

Interruptable/Masked

The Masked/Interruptable state of the CPU is determined by the system mask bits [PSW(0–7)], the machine-check mask bit [PSW(13)], and the program mask bits [PSW(36–39)]. If a mask bit is a 1, the associated interruption is accepted; if it is a 0, system and machine-check interruptions remain pending and program interruptions are ignored. The PSW bits and interruptions that will occur if the bit is active are listed in Table 1-4.

The Masked/Interruptable state of the CPU is switched by changing the mask bits in the PSW. The program mask may be changed separately by the Set Program Mask instruction, and the system mask may be changed separately by the Set System Mask instruction. The machine-check mask bit can be changed only by introducing an entirely new PSW, as in the Problem/Supervisor and Wait/Running states. Thus, a change in the entire masked status may be achieved by the privileged instruction Load PSW, by an interruption such as for the Supervisor Call instruction, or by an IPL. Regardless of the preceding program state, the new PSW may introduce a new mask status.

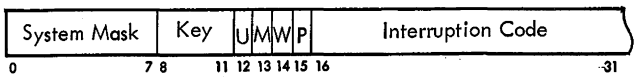
Table 1-4. PSW Interruption Mask Bit Designation

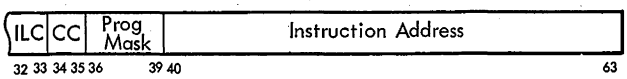
PSW Bit	Interruption
System mask	
0	Multiplexer channel (0)
1	Selector channel 1
2	Selector channel 2
3	Selector channel 3
4	Selector channel 4
5	Selector channel 5
6	Selector channel 6
7	Timer, INTERRUPT pushbutton, or external signals 2–7
Machine-check mask	
13	Machine check
Program mask	
36	Fixed-point overflow
37	Decimal overflow
38	Exponent underflow (floating-point)
39	Significance (floating-point)

PROGRAM STATUS WORD

- PSW provides program status and controls system operation.
- Active PSW is termed *current* PSW.
- Interruption causes current PSW to be stored into old PSW location.
- Load PSW instruction, interruption, PSW restart, and IPL introduce new PSW into CPU.
- Set Program Mask and Set System Mask instructions load new PSW mask fields into CPU.

A doubleword, the program status word (PSW), contains the information required for proper program execution. In general, the PSW controls instruction sequencing, and holds and indicates the status of the system in relation to the program being executed. The PSW has the following format:





Bits 0–7, System Mask. Associated with I/O and external interruptions as follows:

System Mask Bit	Interruption Source
0	Multiplexer channel (0)
1	Selector channel 1
2	Selector channel 2
3	Selector channel 3
4	Selector channel 4
5	Selector channel 5
6	Selector channel 6
7	Timer, INTERRUPT pushbutton, or external signals 2–7

When a system mask bit is a 1, the associated source can interrupt the CPU; when a 0, the interruption remains pending.

Bits 8–11, Key. Contain the CPU storage protection key. The key is matched with the storage key whenever data is stored, or whenever data is fetched from a location that is fetch-protected.

Bit 12, U (USASCII-8). Affects decimal operations only. When a 1 and in unpacked format, decimal digits are represented by USASCII-8. When a 0, EBCDIC is specified.

Bit 13, M (Machine-Check Mask). When a 1, a machine check causes the CPU to log out its status to main storage and to take a machine-check interruption. If the machine-check mask is a 0, the machine check remains pending. As a maintenance aid, the CPU CHECK switch on the system control panel can modify the machine-check mask bit functions.

Bit 14, W (Wait State). When a 1, CPU is in the Wait state; instructions are not executed until an external or I/O interruption or an IPL occurs. When a 0, the CPU is in the Running state.

Bit 15, P (Problem State). When a 1, the CPU is in the Problem state. When a 0, the CPU is in the Supervisor state.

Bits 16–31, Interruption Code. Identify the cause, purpose, or source of the interruption. (Has no meaning in a new PSW.)

Bits 32 and 33, ILC (Instruction Length Code). Indicate the length, in halfwords, of the last processed instruction. This code is predictable only for most program and supervisor call interruptions. For I/O and external interruptions, the interruption is not caused by the last interpreted instruction, and the code is not predictable for these instructions. For machine-check interruptions, the setting of the code may be affected by the

malfunction and, therefore, is unpredictable. A code of 0, used only for program interruptions, indicates that the instruction address in the PSW is *not* the location of the instruction following the instruction that caused the program interruption.

Bits 34 and 35, CC (Condition Code). Contain the condition code that reflects the result of most arithmetic, logical, or I/O instructions. Each of these operations can set the code to any one of four states, and the conditional-branch instructions can specify the state to be used as the criterion for branching. For example, the CC may reflect such conditions as nonzero, overflow, and underflow. Once set, the CC remains unchanged until modified by an instruction that reflects a different code. The two bits of the CC provide for four possible binary settings: 00, 01, 10, and 11. This manual refers to the CC's as 0, 1, 2, and 3. (The CC has no meaning in a new PSW.)

Bits 36–39, Program Mask. Each bit is associated with a program interruption as follows:

Program Mask Bit	Program Interruption
36	Fixed-point overflow
37	Decimal overflow
38	Exponent underflow (floating-point)
39	Significance (floating-point)

When a program mask bit is a 1, the associated program interruption results in an interruption; when 0, the interruption is lost.

Bits 40–63, Instruction Address. Specify the leftmost byte address of the next instruction.

The active or controlling PSW is called the *current* PSW. The information making up the current PSW is held in triggers and registers in the CPU and is constantly being updated as instructions are executed. (The instruction-address field of the PSW, for example, is held in the instruction counter and is updated to give the address of the next instruction to be executed.) When an interruption is taken, the PSW is assembled in the ST register and is transferred to a fixed location in main storage corresponding to the interruption. (The stored PSW is called the *old* PSW.) In this way, the program can preserve for subsequent analysis the status of the CPU at the time of interruption. To complete the interruption, a new PSW is introduced into the CPU from another unique main storage location, and the instruction at the specified location is fetched.

In certain circumstances, the entire PSW is loaded into the CPU; in others, only part of it. As explained in the preceding paragraph, when the CPU is interrupted, the entire current PSW is stored and an entire new PSW is loaded. The state of the CPU may be changed by executing the Load PSW instruction, which introduces a new PSW. New program mask and system mask bits may be specified by altering the corresponding fields in the PSW through the Set Program Mask and Set System Mask instructions, respectively. The Set Program Mask instruction also changes the condition code.

INTERRUPTIONS AND EXCEPTIONAL CONDITIONS

- Five classes of interruptions are recognized:
  Machine check
  Program
  Supervisor call
  External
  I/O
- Eight exceptional conditions are recognized:
  Timer
  CPU store in progress
  Manual control stop
  Manual control wait
  Manual control repeat
  Program store compare
  Invalid instruction address test
  Q-register refill
- Program interruption suppresses, terminates, or allows completion of instruction being processed.
- Machine-check interruption terminates instruction being processed.
- Other interruptions and exceptional conditions allow completion of instruction being executed.

In general terms, the purpose of the interrupt area in the CPU is twofold: (1) to recognize defined interruptions and exceptional conditions that may arise while the system is in operation, and (2) to control the action that is subsequently initiated. The distinction between interruptions and exceptional conditions is that interruptions always allow a new instruction flow to be entered whereas exceptional conditions do not, unless an associated interruption condition is also present.

Five classes of interruptions have been defined for the System/360:

1. Machine Check. Caused by the machine-checking circuits detecting a machine malfunction. The program automatically enters a diagnostic routine if the machine-check mask bit is a 1. The malfunction is indicated in the logout data.
2. Program. Caused by unusual conditions (such as incorrect operands or programming errors) encountered in a program. The exact error is indicated in the Interrupt Code triggers.
3. Supervisor Call. Caused by the program issuing an instruction to turn control over to the supervisor program. The reason for the call is shown in E(8–15).

4. External. Caused by the interval timer going from positive to negative, or by depressing the INTERRUPT pushbutton on the system control panel, or, if the Direct Control feature is installed, by an external device requiring attention.
5. I/O. Caused by an I/O device ending an operation or otherwise needing attention, or by operator intervention at an I/O device. Identification of the device and channel causing the interruption is signalled to the CPU. The status of the device and channel is stored in a fixed main storage location.

Table 1-5 lists for each interruption the interruption code, mask bits, ILC, and how the instruction execution is finished.

Unlike interruptions, exceptional conditions vary between models of the System/360. In the Model 65, eight exceptional conditions are recognized:

1. Timer. Caused by positive swing of line frequency.
2. CPU Store In Progress. Caused by a store operation in progress while an interruption or other exceptional condition is to be serviced or a Load PSW instruction is to be performed.
3. Manual Control Stop. Caused by a need to switch the program from Operating to Stopped state.
4. Manual Control Wait. Caused by the program switching from Running to Wait state.
5. Manual Control Repeat. Caused by the repeat instruction operation, a maintenance aid using manual controls.
6. Program Store Compare. Caused by possibly altering a main storage location already prefetched as an instruction, or by returning to the instruction flow following an Execute instruction.
7. Invalid Instruction Address Test. Caused when the next instruction to be performed is at an invalid, protected, or incorrectly specified main storage location.
8. Q-Register Refill. Caused by the need to delay processing of the next instruction when the Q-register is being refilled.

Interruptions and exceptional conditions are always processed after ending the current instruction and before starting the next instruction. The finishing of the current instruction is influenced by the cause of the interruption; the instruction may be completed, terminated, or suppressed.

In instruction completion, the results of the operation are stored and the condition code (CC) is set as for normal instruction operation, although the result may be altered by the interruption condition. In instruction termination, execution of the instruction has started and some of the data in the registers has changed. The results may therefore be incorrect. The CC may also be incorrect. The instruction may or may not be allowed to continue to completion; the final results should not be used for further computation. In instruction suppression, execution of the instruction is halted by forcing an end-op cycle. Results are not stored, and the CC is not changed.

In some cases, the instruction is almost finished before the interruption condition is detected. For these cases, the CPU blocks any change to the CC and prevents storing of the unpredictable final result by changing the store operation to a fetch operation.

A program interruption may suppress, terminate, or allow completion of the instruction being processed; the particular effect depends on the instruction being performed and the cause of the condition. A machine-check interruption terminates the instruction (or any other action being performed) immediately upon detection of the malfunction. The other interruptions and exceptional conditions detected during instruction execution allow the instruction to be completed.

As an instruction ends and before the next instruction starts, the CPU interrupt logic examines triggers to determine whether an interruption or exceptional condition has arisen. If an interruption operation is to be performed, an exit is made from the current instruction flow and, following the interruption operation, a new instruction flow can be entered. If an exceptional condition operation is to be performed, the current instruction flow is usually not left but only delayed for the time it takes to perform the operation.

During execution of an instruction, several interruptive events may occur. For example, the instruction may give rise to a program interruption, an external interruption signal may be received, a timer exceptional condition may occur, a machine malfunction may occur, and an I/O interruption request may be made. Instead of the program interruption, a supervisor call interruption might occur; these two interruptions cannot occur together, because they are mutually exclusive. When simultaneous interruptions occur, the competing interruptions are serviced in a fixed order of priority.

The priority of interruptions and exceptional conditions is:

1. Timer exceptional condition.
2. CPU store in progress exceptional condition.
3. Machine-check interruption.
4. Program interruption.
5. Supervisor call interruption.
6. External interruption.
7. I/O interruption.
8. Manual control stop exceptional condition.
9. Manual control wait exceptional condition.
10. Manual control repeat exceptional condition.
11. Program store compare exceptional condition.
12. Invalid instruction address test exceptional condition.
13. Q-register refill exceptional condition.

Table 1-5. Interruptions

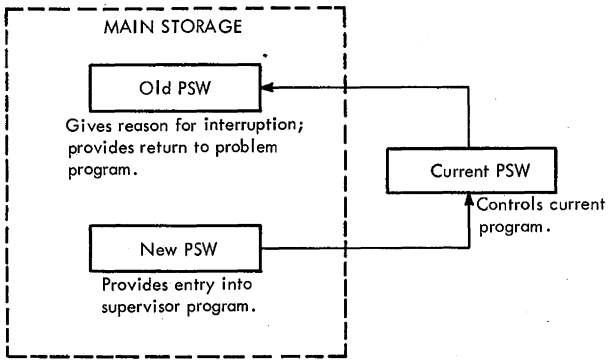
Interruption	Interruption Code (Old PSW Bits 16–31)	PSW Mask Bit	ILC	How Instruction Execution Is Finished
Machine Check	00000000 00000000	13	u	Terminated
Program				
Operation	00000000 00000001	–	1, 2, 3	Suppressed
Privileged operation	00000000 00000010	–	1, 2	Suppressed
Execute	00000000 00000011	–	2	Suppressed
Protection	00000000 00000100	–	0, 2, 3	Suppressed or Terminated
Addressing	00000000 00000101	–	0, 1, 2, 3	Suppressed or Terminated
Specification	00000000 00000110	–	1, 2, 3	Suppressed
Data	00000000 00000111	–	2, 3	Terminated
Fixed-point overflow	00000000 00001000	36	1, 2	Completed
Fixed-point divide	00000000 00001001	–	1, 2	Suppressed or Completed
Decimal overflow	00000000 00001010	37	3	Completed
Decimal divide	00000000 00001011	–	3	Suppressed
Exponent overflow	00000000 00001100	–	1, 2	Terminated
Exponent underflow	00000000 00001101	38	1, 2	Completed
Significance	00000000 00001110	39	1, 2	Completed
Floating-point divide	00000000 00001111	–	1, 2	Suppressed
Multisystem	00000000 00010010	–	2	Suppressed
Supervisor Call	00000000 rrrrrrrr	–	1	Completed
External				
External signal 7	00000000 ppppppp1	7	u	Completed
External signal 6	00000000 pppppp1 p	7	u	Completed
External signal 5	00000000 ppppp1 pp	7	u	Completed
External signal 4	00000000 pppp1 ppp	7	u	Completed
External signal 3	00000000 ppp1 pppp	7	u	Completed
External signal 2	00000000 pp1 ppppp	7	u	Completed
INTERRUPT pushbutton	00000000 p1 pppppp	7	u	Completed
Timer	00000000 1ppppppp	7	u	Completed
I/O				
Multiplexer channel (0)	00000000 aaaaaaaa	0	u	Completed
Selector channel 1	00000001 aaaaaaaa	1	u	Completed
Selector channel 2	00000010 aaaaaaaa	2	u	Completed
Selector channel 3	00000011 aaaaaaaa	3	u	Completed
Selector channel 4	00000100 aaaaaaaa	4	u	Completed
Selector channel 5	00000101 aaaaaaaa	5	u	Completed
Selector channel 6	00000110 aaaaaaaa	6	u	Completed

Notes:
u: Unpredictable; E(0,1)
r: R1 and R2 fields of Supervisor Call instruction
p: Set if pending before PSW(7) is set to a 1.
a: I/O device address

Interruptions

- Each class of interruption has two related PSW's: old and new.
- Old and new PSW's have fixed, unique main storage addresses.
- Current PSW is stored into old address.
- 64 is added to old PSW address to get new PSW address.
- New PSW becomes current PSW.
- There are 12 unconditional interruptions.
- 20 interruptions have associated PSW mask bits. If mask bit is 1, take interruption; if 0, ignore interruption.

An interruption replaces the entire *current* PSW. It is placed into a fixed location in main storage, and becomes the *old* PSW (Figure 1-14). This old PSW gives the reason for the interruption and also provides a return to the interrupted program. A *new* PSW is then fetched from a fixed location in main storage and becomes the current PSW. This new PSW provides an entry into the correct interruption-handling routine in the supervisor program:



Each of the five classes of interruptions has its own distinct locations for new and old PSW's, as follows:

	Decimal Address		Hex Address	
Interruption	Old PSW	New PSW	Old PSW	New PSW
External	24	88	18	58
Supervisor Call	32	96	20	60
Program	40	104	28	68
Machine Check	48	112	30	70
I/O	56	120	38	78

Thus, for example, a machine-check interruption causes the current PSW to be placed into location 48 and a new PSW to be brought out from location 112. Note that these locations are all divisible by 8 because they contain doublewords, and that the location of any new PSW is 64 higher than its corresponding old PSW location.

The five classes of interruptions tell the supervisor only the *general* reason for the interruption. For instance, the fact that the new PSW was brought out of location 104 tells the supervisor that the interruption was caused by a program check. The supervisor still needs to know what type of program check occurred. This is the function of the interruption code, which is set into the current PSW automatically by the CPU logic before the PSW is stored. By examining the interruption code in bits 16–31 of the old PSW, the program-check routine in the supervisor program can tell specifically whether it was a specification, addressing, or some other type of error. In the case of I/O interruptions, the interruption code tells the supervisor which channel and I/O unit caused the I/O interruption. In the case of a machine-check interruption, the supervisor must inspect the logout data to learn the specific malfunction.

After the interruption has been processed by the supervisor, the last instruction can be a Load PSW. This instruction causes the old PSW to once again become the current PSW, and the CPU is back in the problem program.

The load PSW instruction may also be used to: (1) allow the supervisor to change the current PSW, and (2) load the PSW for a new problem program after the program has been read into main storage by the supervisor.

Interruption Masking

Sometimes it is not desirable to allow an interruption. This condition becomes apparent when I/O interruptions are considered (Figure 1-15). Assume an I/O operation is completed, resulting in an I/O interruption. The current PSW is stored as the old PSW to give the supervisor the reason for (or the source of) the interruption. This old PSW also enables the supervisor to return to the interrupted problem program. A new PSW is then brought out of storage and becomes the current PSW which indicates the first instruction of the I/O interruption-handling routine. If at this time a second I/O interruption, perhaps caused by operator intervention at an I/O device of another channel, were allowed, the old PSW stored as a result of the first I/O interruption would be lost. The supervisor can prevent this second I/O interruption from being accepted until it has processed the first I/O interruption by means of mask bits in the new PSW.

Twenty of the 32 interruptions for the five classes can be masked off by associated mask bits in the PSW. (The

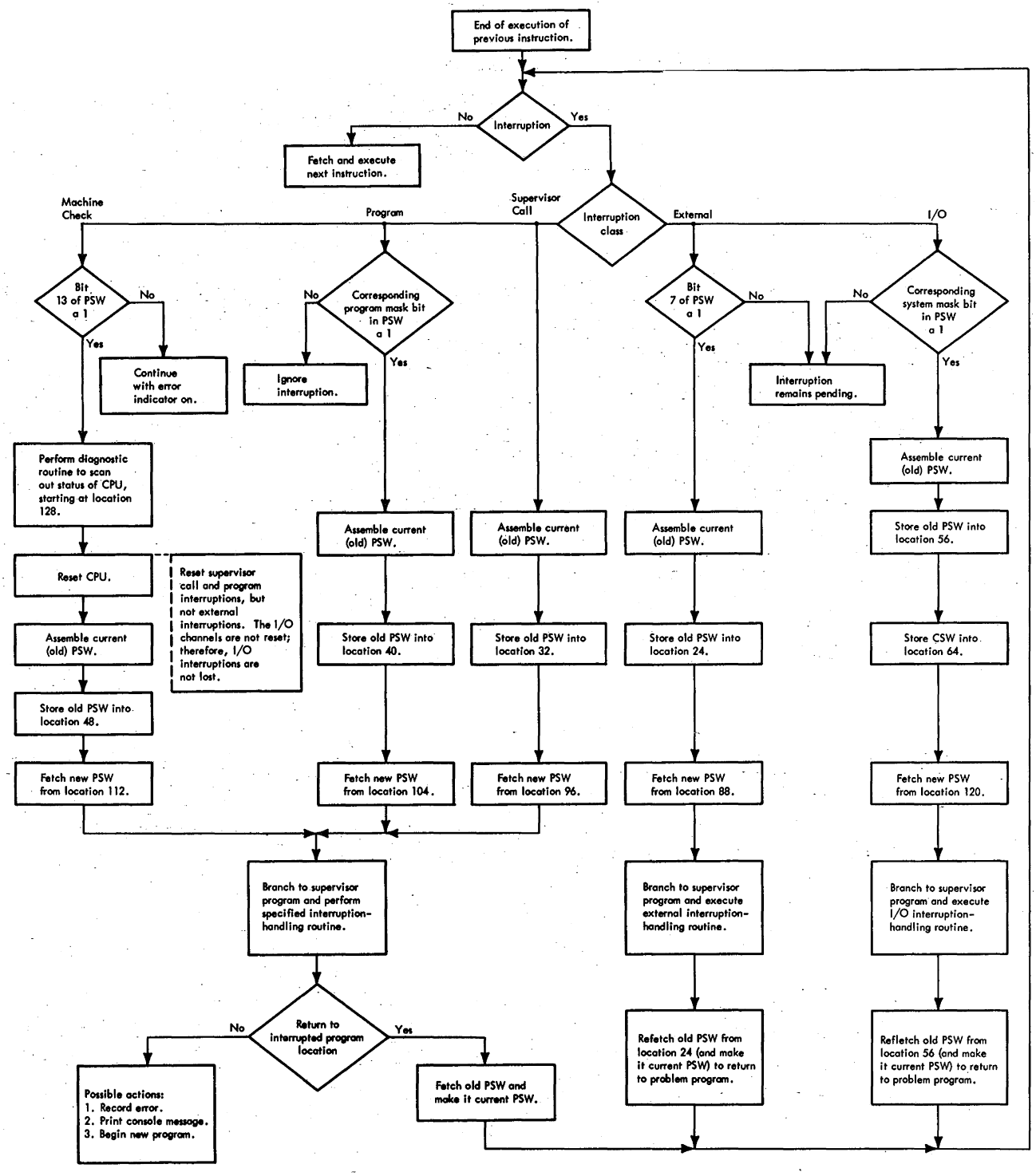


Figure 1-14. Action Taken When Single Interruption Occurs

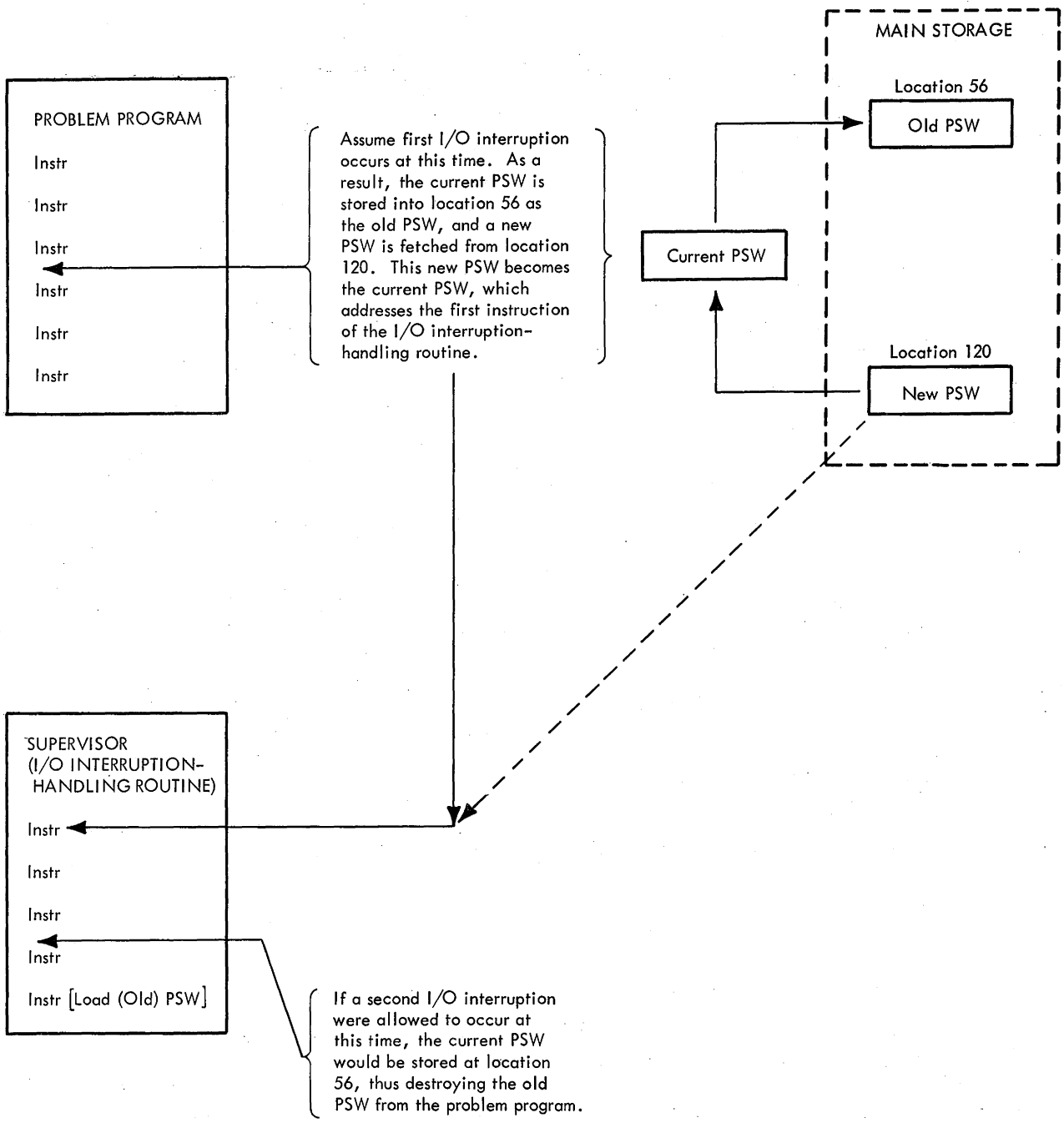


Figure 1-15. Example of Need for Interruption Masking

remaining 12 interruptions are unconditional; they are always taken.) If the corresponding mask bit is a 1, the interruption is taken; if a 0, the interruption is ignored or remains pending. External and I/O interruptions may be masked off by the system mask field of the PSW; machine-check interruptions may be masked off by the machine-check mask bit; 4 of the 15 program interruptions may be masked off by the program mask field.

System Mask Field. The system mask field consists of eight bits [PSW(0–7)] which can be used selectively or collectively to mask all I/O and external interruptions:

System Mask Bit	Interruption Source
0	Multiplexer channel (0)
1	Selector channel 1
2	Selector channel 2
3	Selector channel 3
4	Selector channel 4
5	Selector channel 5
6	Selector channel 6
7	Timer, INTERRUPT pushbutton, or external signals 2–7

To prevent an I/O or external interruption before the first interruption has been processed, the system mask of the new PSW should contain zeros. When a system mask bit is a 0, the associated I/O or external interruption remains pending.

The system mask field may be changed by introducing a new PSW, or it may be changed separately by the Set System Mask instruction.

Machine-Check Mask Bit. The machine-check mask bit [PSW(13)] controls the acceptance of a machine-check interruption. If this bit is a 0, machine-check interruptions are ignored and remain pending. If this bit is a 1, machine-check interruptions are taken, depending on the position of the CPU CHECK switch on the system control panel. If this switch is in the PROC (normal) position, the CPU stops and the status is logged into main storage; a machine-check interruption then takes place. If the CPU CHECK switch is in the DSBL (disable) position, the CPU does not stop upon detection of a machine check and no logout or interruption takes place. If the switch is in the STOP position, the CPU stops upon detection of a machine check, but there is no logout of data and no interruption takes place.

The usual mode of operation is to have the CPU CHECK switch set to the PROC position and PSW(13) set to a 1.

The machine check mask bit can be changed only by introducing a new PSW.

Program Mask Field. The program mask field consists of four bits [PSW(36–39)], each of which is associated with a program check:

Program Mask Bit	Program Interruption
36	Fixed-point overflow
37	Decimal overflow
38	Exponent underflow (floating-point)
39	Significance (floating-point)

When a program mask bit is a 1, the associated program check results in an interruption; when a 0, no interruption occurs and the condition does not remain pending.

The program mask field may be changed by introducing a new PSW, or it may be changed separately by the Set Program Mask instruction.

Instruction Address Determination

- PSW holds address of instruction to be executed next.
- Interruption, if any, occurs during instruction execution.
- For program and supervisor call interruptions, instruction address less ILC gives address of preceding instruction during which interruption occurred.

As stated earlier, the instruction address portion of the current PSW is used by the interruption operation to fetch an instruction. Once the instruction has been fetched, the instruction address portion of the PSW is updated to address the next instruction. Interruptions are serviced only *after* an instruction is finished. Therefore, the instruction address portion of the old PSW does not contain the address of the last instruction executed, but rather contains the address of the next instruction that *would* have been executed if the interruption had not occurred. For some interruptions, it is desirable to locate the instruction during which the interruption occurred. To obtain this location, the instruction address portion of the old PSW must be decremented by the supervisor. To do so, the supervisor must know the length of the last instruction executed. This

length, in halfwords, is given by the instruction length code (ILC) of the old PSW (bits 32 and 33). The ILC for the five formats is as follows:

ILC	PSW Bits 32 and 33	Op Code Positions 0 and 1	Instruction Length (in Halfwords)	Format
0	00	–	Not available	–
1	01	00	1	RR
2	10	01	2	RX
2	10	10	2	RS and SI
3	11	11	3	SS

The ILC is predictable only for program and supervisor-call interruptions. For I/O and external interruptions, the interruption is not caused by the preceding instruction, and the ILC is therefore not predictable for these instructions. For machine-check interruptions, the ILC setting may be affected by the malfunction and, therefore, may be incorrect. Therefore, the instruction causing these interruptions, if any, must be located by other means.

For supervisor-call interruptions, the ILC is 1, indicating the halfword length of the Supervisor Call instruction. For program interruptions, an ILC of 1, 2, or 3 indicates the length of the instruction in halfwords. The ILC of 0 is reserved for a storage protection condition detected after completion of the instruction that caused the violation; this condition is called a *late* storage protection check and results is an *indefinite* program interruption.

Note that for a program interruption caused by an incorrect branch address, the address determined from the instruction address and ILC is the branch address and not the address of the branch instruction.

Machine-Check Interruption

The machine-check interruption provides a means for recognizing a machine malfunction. The following malfunctions cause a machine check:

1. ROS word parity check.
2. Parallel adder full-sum parity check.
3. Parallel adder half-sum parity check.
4. Serial adder full-sum parity check.
5. Serial adder half-sum parity check.
6. E-register parity check.
7. Multiply/divide logic error.
8. Storage address check to CPU.
9. Storage data check to CPU.
10. System hang (Multisystem feature only).

Each malfunction sets a specific trigger which, in turn, sets the 'error' trigger and lights the PROC CHK indicator on the system control panel and the CHK SUMM indicator on roller switch 3 (position 1, bit 20).

Acceptance of a machine-check interruption depends upon the state of the machine-check mask bit, PSW(13), and upon the position of the CPU CHECK switch on the system control panel.

If the machine-check mask bit is a 1 and the CPU CHECK switch is in the PROC position, machine-check interruptions are taken. The current instruction is terminated, and a diagnostic routine called *logout* is initiated. The status of the CPU is logged out into the permanent main storage area starting at location 128 (decimal) and extending through location 295, a total of 22 doublewords. A machine-check interruption then takes place; the old PSW is stored at main storage location 48 (decimal) with an interruption code of 0 and a new PSW is fetched from main storage location 112.

If the machine-check mask bit is a 0 and the CPU CHECK switch is in the PROC position, the interruption remains pending and the CPU attempts to complete the current instruction and to proceed with the next instruction.

If the CPU CHECK switch is in the DSBL (disable) position, the interruption is ignored regardless of the state of the machine-check mask bit.

If the switch is in the STOP position, the CPU stops upon detection of the machine check regardless of the state of the mask bit.

Following an emergency power-off, power-on, or system-reset operation, incorrect parity may exist in storage and registers. Unless new information is loaded, a machine-check condition may occur erroneously. Once storage and registers are cleared, a machine-check interruption can be caused only by a machine malfunction and not by invalid data or instructions.

Program Interruptions

Program interruptions result from improper specifications or unusual conditions arising during the processing of data or instructions. There are 15 program interruptions, 4 of which may be masked off by associated bits in the PSW program mask field; the remaining 11 are unconditionally taken. If the associated mask bit is a 0, the interruption is ignored and does not remain pending.

The program interruption causes the old PSW to be stored into main storage location 40 (decimal) and a new PSW to be fetched from location 104. Interruption code bits 28–31 identify the cause of the interruption. Bits 16–27, the remainder of the interruption code, are made zeros.

Four Interrupt Code triggers determine the code to be set into bits 28–31 of the interruption code. The outputs of the four triggers are encoded to give the 15 possible codes. The specific trigger(s) to be set is determined by a combination of a micro-order, the op code, and data conditions. The 'program interrupt' latch is set by the

Interrupt Code triggers. A brief description of the 15 program interruptions follows:

1. Operation. An invalid op code is detected. The instruction is suppressed; the ILC is 1, 2, or 3.
2. Privileged Operation. A privileged instruction is encountered in the Problem state. The instruction is suppressed; the ILC is 1 or 2.
3. Execute. The subject instruction of an Execute instruction is another Execute instruction. The instruction is suppressed; the ILC is 2.
4. Protection. The storage key of a main storage location does not match the storage protection key in the PSW. For a store protection violation, the instruction is suppressed, except for Store Multiple, Read Direct, and Test and Set instructions and VFL operations, which are terminated. The ILC is 0, 2, or 3. For a fetch-protection violation, the instruction is terminated except for the Execute instruction which is suppressed; the protected information is not loaded into an addressable register or moved to another storage location. The ILC is 0, 2, or 3. In the case of a violation caused by an I/O operation, data transmission is terminated in such a way that the protected information is not recorded on an output medium. The violation is indicated in the channel status word stored as a result of the operation. The ILC is 2 or 3.
5. Addressing. An address specifies any part of data, an instruction, or a control word outside the available main storage. In most cases, the instruction is terminated. Data in main storage remains unchanged, except when designated by valid addresses. The instruction is suppressed for the Convert to Decimal, Diagnose, and Execute instructions, for certain SI-format instructions, and for certain store instructions. The ILC is 0, 1, 2, or 3.
6. Specification. A specification program interruption is caused by any one of six conditions:
   a. A data, instruction, or control-word address does not specify an integral boundary for the unit of information.
   b. The R1 field of an instruction specifies an odd LS register address instead of an even register address for a pair of GPR's that contains a doubleword operand.
   c. An FPR address other than 0, 2, 4, or 6 is specified.
   d. The multiplier or divisor in decimal arithmetic operations exceeds 15 digits and sign.
   e. The divisor in decimal division is equal to or greater than the dividend, or the multiplier in decimal multiplication is equal to or greater than the multiplicand.
   f. The block address specified in Set Storage Key or Insert Storage Key instructions does not have the four low-order bits set to zero.

   The instruction is suppressed; the ILC is 1, 2, or 3.
7. Data. A data interruption is caused by any one of three conditions:
   a. The sign or digit codes of operands in decimal arithmetic, editing, or convert-to-binary operations are incorrect.
   b. Fields in decimal arithmetic operations overlap incorrectly.
   c. The decimal multiplicand has too many high-order significant digits. (The number of high-order zeros in the multiplicand must at least equal the multiplier field.)

   The instruction is terminated; the ILC is 2 or 3.
8. Fixed-Point Overflow. A high-order carry occurs or high-order significant bits are lost in fixed-point add, subtract, shift, or load operations. The instruction is completed by ignoring the overflow; the ILC is 1 or 2. The interruption may be masked off by making the fixed-point overflow mask bit [PSW(36)] a 0; the interruption is then ignored.
9. Fixed-Point Divide. The quotient exceeds the register size in a fixed-point divide instruction, or the result of a Convert to Binary instruction exceeds 31 bits. The divide instruction is suppressed, and the Convert to Binary instruction is completed by ignoring the extra bits. The ILC is 1 or 2.
10. Decimal Overflow. The destination field is too small to contain the result field in a decimal arithmetic operation. The instruction is completed by ignoring the overflow information; the ILC is 3. The interruption may be masked off by making the decimal-overflow mask bit [PSW(37)] a 0; the interruption is then ignored.
11. Decimal Divide. The quotient exceeds the specified data field size in a decimal division. The instruction is suppressed; the ILC is 3.
12. Exponent Overflow. The result exponent (characteristic) of a floating-point addition, subtraction, multiplication, or division overflows, and the result fraction is not zero. The operation is completed by making the characteristic 128 smaller than the true result; the sign and fraction remain unchanged. The ILC is 1 or 2.
13. Exponent Underflow. The result of a floating-point addition, subtraction, multiplication, or division underflows, and the result fraction is not zero. A program interruption occurs if the exponent-underflow mask bit [PSW(38)] is a 1. The operation is completed by replacing the result with a true zero if the mask bit is 0. If the mask bit is 1, the characteristic is made 128 larger than the true result, and the sign and fraction remain unchanged. The ILC is 1 or 2.
14. Significance. The result of a floating-point addition or subtraction has an all-zero fraction. The instruction is completed; the ILC is 1 or 2. The interruption may be masked off by making the significance mask bit [PSW(39)] a 0; the interruption is then ignored.
15. Floating-Point Divide. A division is attempted by a

floating-point divisor with a zero fraction. The instruction is suppressed; the ILC is 1 or 2.

Note: If the Multisystem feature is installed, a fifth Interrupt Code trigger is added for interruption code bit 27. A multisystem program interruption code of 18 (decimal) occurs if the Set System Mask instruction is encountered when in Multisystem mode. The instruction is suppressed; the ILC is 2.

Supervisor-Call Interruption

The supervisor-call interruption is used by the problem program to pass control to the supervisor program. To do so, the problem program executes the Supervisor Call instruction which, in turn, causes a supervisor-call interruption. The interruption is unconditionally taken; there is no associated mask bit in the PSW. The old PSW is stored at main storage location 32 (decimal), and the new PSW is fetched from location 96. The R1 and R2 fields (bit positions 8–15) of the Supervisor Call instruction become the low-order half (bits 24–31) of the interruption code of the old PSW. These bits may be used to convey a message (for example, the reason for the call) from the calling program to the supervisor program. Bits 16–23 of the interruption code are made zero. The ILC is 1, indicating the one-halfword length of the Supervisor Call instruction.

External Interruptions

The external interruption enables the CPU to respond to signals from external units, from the INTERRUPT pushbutton on the system control panel, and from the interval timer. An external interruption may occur at any time, and interruptions from the different sources may occur simultaneously. Interruptions are kept pending until accepted by the CPU. When several interruptions from one or more sources are pending, only one interruption is taken servicing all pending external interruptions.

An external interruption can occur only when system mask bit 7 is a 1 and after the current instruction is executed. The interruption causes the old PSW to be stored at main storage location 24 (decimal) and a new PSW to be fetched from location 88. The source of the interruption is identified by bit positions 24–31 of the interruption code; bits 16–23 are made zero.

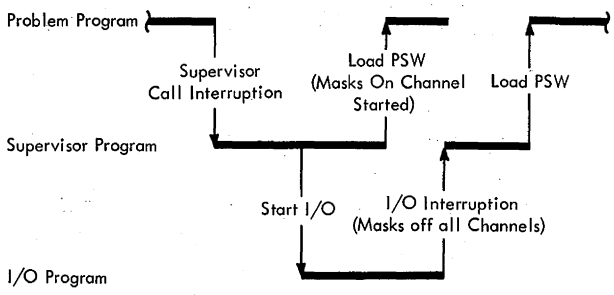
A brief discussion of the three external interruptions follows:

1. External Signals. Available only if the Direct Control feature is installed. May be masked off by depressing DISABLE DIRECT CONTROL on the system control panel or by setting bit 7 of the system mask field to a 0. Six signal-in lines, representing external signals 2–7, are connected to the CPU to cause an external interruption. The specific external signal causing the interruption is identified by a unique bit position in positions 26–31 of the interruption code. For example, external signal 7 is identified by a 1 in bit position 31, external signal 6 by a 1 in bit position 30, and so on. The Multisystem feature, if installed, assigns specific meanings to external signals 2 and 3. (Refer to Chapter 4, Section 2.)
2. INTERRUPT Pushbutton. If bit 7 of the system mask field is a 1, depressing INTERRUPT on the system control panel causes an external interruption. Bit position 25 of the interruption code is set to a 1.
3. Timer. If bit 7 of the system mask field is a 1 and the interval timer value changes from positive to negative, an external interruption occurs. Bit position 24 of the interruption code is set to a 1. The interruption is initiated as the timer count proceeds from a positive number, including zero, to a negative number. The interval timer is updated (decremented) 60 times a second or 50 times a second, depending on the line frequency. It is possible that, after an interruption is initiated, the timer may have been updated several times before the CPU is actually interrupted, depending on the instruction being executed and on the state of the mask bit. The operation of the timer is controlled by the DISABLE INTERVAL TIMER switch on the system control panel.

I/O Interruptions

The I/O interruption enables the CPU to respond to signals from I/O devices. A request for an I/O interruption may occur at any time, and more than one request may occur at the same time. The requests remain pending in the I/O area of the system until accepted by the CPU. Priority is established among requests so that only one interruption request is honored at a time. The order of priority is channel 0, then channels 1–6. Note that this priority is different from the priority the BCU establishes for servicing I/O storage requests.

I/O interruptions generally occur at the end of an I/O operation. Most I/O operations are overlapped with processing; an I/O interruption, therefore, is an efficient way of signalling the supervisor that the I/O operation is finished. After the I/O interruption-handling routine in the supervisor is finished, control is passed to the problem program:



An I/O interruption can occur only if the associated mask bit in the system mask field of the PSW is a 1. The

I/O interruption causes the old PSW to be stored at main storage location 56 (decimal) and the channel status word associated with the interruption to be stored at location 64. The new PSW is fetched from location 120. Bit positions 21–23 and 24–31 of the interruption code identify the channel and the I/O device, respectively, causing the interruption. Bit positions 16–20 are made zero.

Exceptional Conditions

Exceptional conditions are processed by the CPU after finishing the instruction in progress. After processing the exceptional condition, the instruction flow might be continued, stopped, repeated, or left, depending on the specific exceptional condition. An exceptional condition operation does not change the current PSW, nor does the PSW contain mask bits for exceptional conditions.

Timer Exceptional Condition

When the interval timer must be stepped (decremented), a timer exceptional condition is generated (recognized by a trigger that is set by a positive swing of the line frequency). When the instruction in progress is finished, the CPU generates main storage address 80 (decimal), the location of the timer value. The timer value is fetched, stepped, and returned to location 80. If the timer value is stepped from a positive value to a negative value, a timer external interruption is processed next. Otherwise, processing continues with the next instruction. The timer is controlled by the DISABLE INTERVAL TIMER switch on the system control panel.

CPU Store in Progress Exceptional Condition

When the next instruction must be delayed to prevent the subsequent program interruption from being indefinite, a CPU store in progress exceptional condition is generated. This condition is recognized if the storing of data into main storage would overlap the processing of the Load PSW instruction or the handling of an interruption or manual control or program store compare exceptional condition. A two-cycle CPU loop is entered to insure that any "late" storage protection check will not result in an indefinite program interruption under these conditions. If a protection check occurs, a program interruption is processed next. Otherwise, processing continues with the next instruction.

Manual Control Stop Exceptional Condition

The manual control stop exceptional condition arises when the program is to be switched from the Running to the Stopped state. The CPU enters a "stop loop", during which no instructions are processed and all interruptions are kept pending. Only the following pushbuttons are recognized: STORE, DISPLAY, SET IC, START, ROS TRANSFER, and PSW RESTART.

Manual Control Wait Exceptional Condition

The manual control wait exceptional condition arises when PSW(14) is set to a 1. CPU clock signals are inhibited. Processing continues when an external or I/O interruption or an IPL operation is initiated.

Manual Control Repeat Exceptional Condition

The manual control repeat exceptional condition arises when the repeat instruction operation (a maintenance aid) is begun and, if the REPEAT INSN (instruction) switch on the system control panel is in the SINGLE position, before each subsequent repetition of the specified instruction.

Program Store Compare Exceptional Condition

A program store compare exceptional condition results if the next instruction to be processed must again be fetched into the Q-, R- and E-registers. This need occurs after processing the Execute instruction and after some store operations. Although the instruction to be executed next is held in the instruction buffer (Q-register), it is modified in its main storage location only. Therefore, to have the correct version of the instruction in the Q-register, the next instruction is refetched before it is executed.

Invalid Instruction Address Test Exceptional Condition

The previously discussed exceptional conditions and interruptions result from unusual conditions occurring during the execution of an instruction. To identify the instruction, its address and op code are preserved by inhibiting the fetching of the next instruction until the microprogram routine that handles the interruption or exceptional condition is ended.

In an invalid instruction address test exceptional condition, however, it is the address of the *next* instruction that is invalid, protected, or incorrectly specified. Processing of the instruction is allowed to start; thus the address of the previous instruction is replaced with an erroneous address. The appropriate Interrupt Code trigger(s) is set and a program interruption is processed next.

Q-Register Refill Exceptional Condition

The Q-register refill exceptional condition arises when Q-register (instruction buffer) refilling conflicts with the start of the next instruction. The exceptional condition delays processing of the next instruction by one (or two) cycles.

CONTROL OF I/O OPERATIONS

The following paragraphs: define an I/O operation; discuss how I/O operations are controlled by instructions, commands, orders, and control words; and illustrate how the I/O system works, using the Start I/O instruction as an example.

Instructions, Commands, and Orders

- CPU executes instructions.
- Channels execute commands.
- Control units and devices execute orders.
- Five operations are available:
  Write
  Read
  Read backward
  Control
  Sense

Input/output operations are initiated and controlled by three types of information: instructions, commands, and orders. Instructions are decoded and executed by the CPU and are part of the CPU program. Commands are decoded and executed by the channel, and initiate I/O operations such as reading and writing. Instructions and commands are fetched from main storage and are common to all types of devices. Orders specify functions peculiar to an I/O device, such as rewinding tape or spacing a line on a printer. Orders are contained in the control command; they are decoded and executed by the device.

The action in an I/O device initiated by a command is termed an *I/O operation*. Five I/O operations are available: write, read, read backward, control, and sense. The channel initiates the operation by executing the associated command.

The write command initiates a write operation at the device. Data from main storage is fetched in an ascending order of addresses and transferred to the device.

The read command initiates a read operation at the device. Data is read from the device in the same sequence as it was written by a write command. Data is placed into main storage in an ascending order of addresses.

The read-backward command initiates a read-backward operation at the device. Data is read from the device in a sequence opposite to that in writing. Data is placed into main storage in a descending order of addresses.

The control command contains information, termed *orders*, that controls the selected device. Orders are unique to the particular device in use and specify such functions as backspacing or rewinding magnetic tape. Orders are fetched from main storage in an ascending order of addresses and transferred to the device.

The sense command initiates a sense operation at the device. Data transferred during a sense operation provides information about unusual conditions detected during the last operation and the status of the device. Data is placed into main storage in an ascending order of addresses.

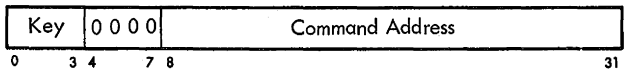
I/O Control Words

Three I/O control words are used during an I/O operation:

1. Channel Address Word (CAW), which initiates I/O sequencing.
2. Channel Command Word (CCW), which controls I/O operations and sequencing.
3. Channel Status Word (CSW), which indicates channel status.

Channel Address Word

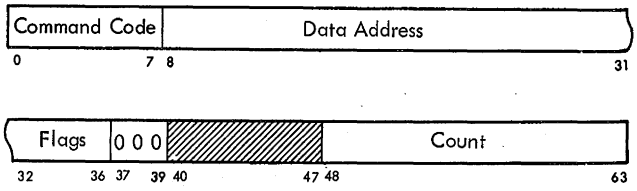
The CAW specifies the address of the first CCW associated with the Start I/O instruction. The CAW is assigned permanent main storage address 72 (decimal). The channel refers to the CAW only during execution of the Start I/O instruction. The pertinent information is stored in the channel, and the CPU program is free to change the contents of the CAW. The CAW has the following format:



Bits 0–3, Key. Specifies the storage protection key for all commands associated with the Start I/O instruction.
Bits 4–7. Must be all 0's.
Bits 8–31, Command Address. Designates location of the first CCW in main storage.

Channel Command Word

The CCW specifies the command to be executed and, for commands initiating I/O operations, designates the main-storage area associated with the operation and the action to be taken whenever data transfers to or from the main storage are completed. The CCW's can be located anywhere in main storage, and more than one can be associated with a Start I/O instruction. The channel refers to a CCW in main storage only once, whereupon the pertinent information is stored in the channel. The first CCW is fetched during execution of the Start I/O instruction. Each additional CCW is obtained when the operation has progressed to the point where the additional CCW is needed. The CCW has the following format:

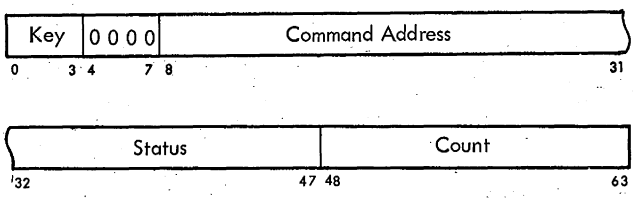


Bits 0–7, Command Code. Specifies I/O operation to be performed.
Bits 8–31, Data Address. Specifies location of an eight-bit byte in main storage; it is the first location referred to in the main storage area designated by the CCW.
Bits 32–36, Flags. Cause certain functions to be performed that modify the operation.

Bits 37–39. Must be all 0's for every CCW other than the CCW that specifies a transfer-in-channel operation.
Bits 40–47. Not used.
Bits 48–63, Count. Specifies the number of eight-bit byte locations in the main storage area designated by the data-address field in the CCW.

Channel Status Word

The CSW provides the program with the status of an I/O device or the condition under which an I/O operation has been finished. The CSW is formed, or parts of it are replaced, in the process of I/O interruptions and during execution of the Start I/O, Test I/O, and Halt I/O instructions. The CSW is placed into main storage location 64 (decimal) and is available to the program at this location until the next I/O interruption occurs or until another I/O instruction causes its contents to be replaced, whichever occurs first. The CSW has the following format:



Bits 0–3, Key. Contains the storage protection key that was used in the I/O operation initiated by the last Start I/O instruction.
Bits 4–7. Must be all 0's.
Bits 8–31, Command Address. Identifies the last CCW used.
Bits 32–47, Status. Identifies the conditions in the I/O device and channel that caused the CSW to be stored.
Bits 48–63, Count. Contains the residual count of the last CCW used.

I/O System Operation

- Start I/O instruction is given.
- Channel fetches CAW from main storage location 72 (decimal).
- CAW designates main storage address of first CCW.
- CCW specifies command and main storage area.
- Channel selects device. Start I/O instruction is terminated at this point.
- Channel controls operation and data transfers.
- Operation is terminated by two conditions: channel end and device end.

The CPU program initiates I/O operations by means of the Start I/O instruction. This instruction identifies the I/O device and causes the channel to fetch the CAW from main storage location 72 (decimal). The CAW designates the location in main storage from which the channel subsequently fetches the first CCW. The CCW specifies the command to be executed, the main storage area to be used, and the number of data bytes to be transferred, if any.

The channel attempts to select the device by sending the address of the device to all attached control units. Upon recognizing the address, the control unit associated with the addressed device connects itself logically to the channel. The channel subsequently sends the command code to the device, and the device responds by indicating whether it can execute the command.

At this time, execution of the Start I/O instruction is terminated, and the CPU continues with its program. The results of the attempt to initiate command execution are indicated in the PSW and, under certain conditions, by storing a portion of the CSW.

If the operation is initiated by the I/O device and its execution involves transfer of data, the channel responds to service requests from the device and assumes control of the operation. For operations that do not require transfer of data, the device signals the end of the operation immediately on receipt of the command code, and the channel is immediately available for a new I/O operation.

An I/O operation may involve transfer of data to or from one main-storage area, designated by a single CCW, or, when data chaining is specified, to or from a number of noncontiguous main-storage areas. In the latter case, a chain of CCW's is used in which each CCW designates an area in main storage for the continuation of the original command (operation).

Termination of the I/O operation normally is indicated by two conditions: channel end and device end. The channel-end condition indicates that the I/O device has received or provided all information associated with the operation and no longer needs channel facilities. The device-end condition indicates that the device has finished the operation.

Facilities are provided for the program to initiate execution of a chain of commands with a single Start I/O instruction. When command chaining is specified, the device-end condition causes the channel to fetch a new CCW that specifies a new operation at the device.

Conditions that initiate I/O interruptions are asynchronous with the activity in the CPU, and more than one interruption condition can occur at the same time. A priority has been established among the conditions so that only one interruption is processed at a time. The I/O interruption conditions are preserved in the I/O devices and channel until accepted by the CPU.

This section discusses: (1) the functional units (based on Diagram 3-2, FEMDM); and (2) instruction fetching and execution, the Universal instruction set by instruction class, and power considerations.

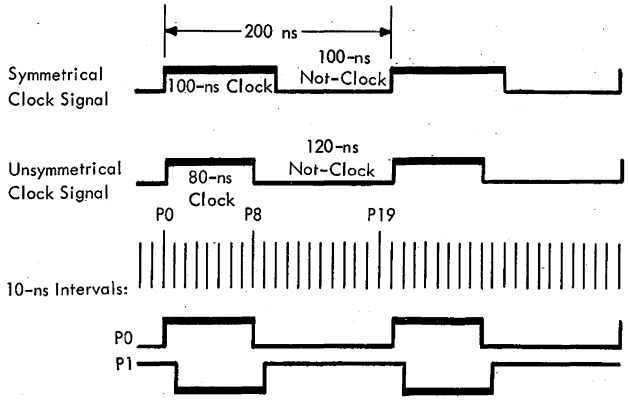
CONTROL

The 2065 CPU operates with a basic clock cycle period of 200 ns under control of ROS. The following paragraphs discuss: (1) CPU timing and data transfer; (2) ROS – what it is, how it controls the CPU, and its data flow; (3) the additional control provided by the PSW register.

CPU Timing

- Basic clock cycle period is 200 ns.
- Symmetrical clock signal consists of 100-ns clock portion and 100-ns not-clock portion.
- Unsymmetrical clock signal consists of 80-ns clock portion and 120-ns not-clock portion.

The basic CPU clock cycle period is 200 ns, divided into clock and not-clock portions. A clock signal generator provides a 5-megaHertz (5-mHz) symmetrical (100-ns/100-ns) clock signal. Two types of clock signal generators are used: a continuously running crystal-controlled oscillator in the Model G65, H65, and I65 CPU or a gated delay-line oscillator in the Model IH65 and J65 CPU. To provide additional time for CPU logic functions, the symmetrical clock signal is modified to give a 5-mHz unsymmetrical (80-ns/120-ns) clock signal. Finer intracycle control is obtained by dividing each of the two clock signals into 20 intervals of approximately 10 ns each. These intervals, named P0, P1, P2, . . . P19, are created by inverters which delay the signal by about 10 ns. Thus the notations P0, P1, P2, . . . P19 refer to signals which are inverted and are delayed 10 ns with respect to the previous signal:



The clock signals are distributed to logic gates A through E. Adjustable time delays within the logic gates synchronize the clock signals with a reference signal, thereby eliminating the various amounts of delay introduced by the distribution cables. The distribution of the clock signals to the CPU processing logic is stopped upon detection of an error or during scan, logout, single-cycle, and certain ROS operations. When the CPU clock signals are stopped, the BCU must continue to service the I/O channels and the scan logic must be operable. Therefore, the clock signals to these areas of the logic are not stopped, except under certain circumstances in the Model IH65 and J65 CPU.

Data Transfer

Data is transferred into a register, into an adder, and into and out of LS by gating signals controlled by ROS (Figure 1-16). Referring to Figure 1-16, note that data from PAL is always available at the A-register, B-register, and T-register, but is transferred only into the selected register by means of the corresponding gating signal from ROS. When gating data into an adder, timing considerations require the use of 'gate control' triggers; these triggers, which are set by the ROS decode logic, generate the required gating signal.

When transferring data into LS, the gating signal from ROS is combined with a signal from the LS addressing logic to develop a 'write LS' signal, which gates the data into LS. When transferring data from the LS, an address signal selects 1 of 24 LS registers, the contents of which are transferred to the LS bus. A second gating signal transfers the data from the LS bus to the S- or T-register.

Read-Only Storage

The CPU is controlled by ROS, a permanently recorded microprogram, supplemented by conventional control logic. A read-only storage is a storage device which contains information (1's and 0's) of a nondestructive nature. The 2065 CPU utilizes a capacitive read-only storage, in which bits are stored in the form of a capacitance between a fixed drive-plate pattern etched at right angles to a sense-plate pattern. Sense and drive plates are separated by a Mylar† film (approximately 1 mil thick), and the resulting sandwich is held together under pressure. A 1-signal is coupled from a drive line to one of a pair of sense lines, and a 0-signal is coupled to the other. Sense line outputs are detected in a differential amplifier which in turn feeds a latch. When decoded, the information in ROS controls gates to route data in the CPU. Access time is approximately 95 ns.

†Trademark of E. I. duPont de Nemours & Co. (Inc.)

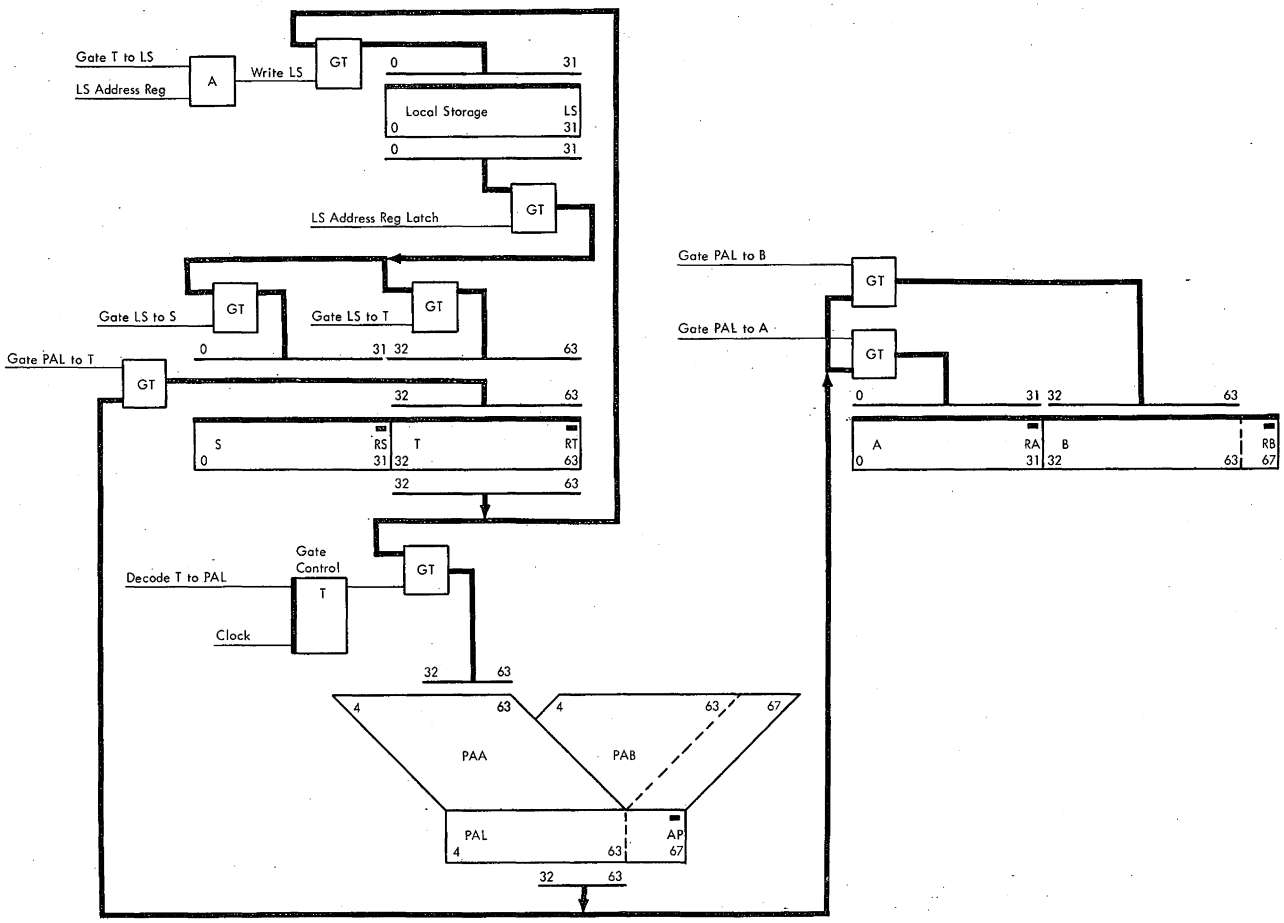


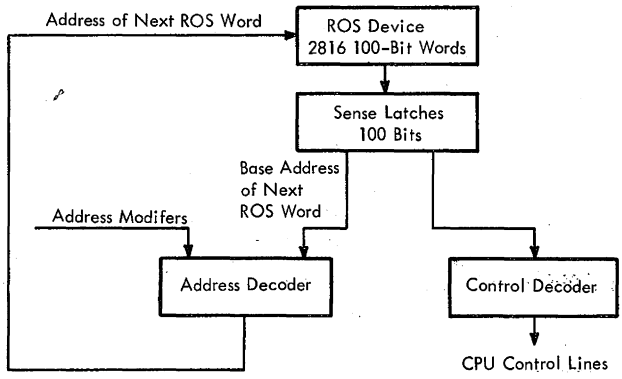
Figure 1-16. Data Transfer Scheme

Relationship of ROS to Conventional Controls

- ROS words replace most conventional sequence triggers and control lines.
- ROS word is a unique bit configuration and controls CPU during machine cycle it is in use.
- In addidion to control data, ROS word holds address of next ROS word and branch tests, if any.
- For branches, 1 ROS word is associated with each possible condition.

To understand ROS operation, it is helpful to note its relationship to conventional controls. Conventional controls may be characterized by sequence triggers, and by the control lines activated by the sequence triggers as a function of the operation to be performed and data conditions. Each cycle that the CPU may take represents a state of the CPU as defined by the control circuits. Each state, in turn, specifies which control lines are to be activated during that cycle and which state is to follow next. The defined state will cause the next sequence trigger to be set in the following cycle. In some cases, the next state may be contingent upon a branch condition in which one of two or more sequence triggers must be selected.

In ROS-controlled CPU's, the sequence triggers are replaced by micro-instructions or ROS words. Each ROS word consists of a predetermined bit pattern and represents a state of the CPU. A micro-instruction is read into the sense latches from the ROS device as follows:



Decoding of the bit pattern activates control lines which initiate operations or functions in the CPU under timing control of ROS decode logic. The base address of the next ROS word to be used is also included in each ROS word. Data conditions within the CPU may modify the address if

the bit pattern indicates a test for branching (e.g., branch if overflow occurs). One ROS word is associated with each possible condition; the base address is modified to address the ROS word which satisfies the data conditions. Thus ROS eliminates the need for most of the complex sequencing networks.

ROS Word

- ROS word is divided into 100 bits, grouped into 21 control fields.
- Number of bits within field determines number of unique control signals within field.
- Control signals are termed *micro-orders*.

The ROS is physically organized into 16 planes, each plane holding 88 200-bit words. Through addressing, the 200-bit word is further divided in half to yield 2816 100-bit words, hereafter referred to as *ROS words*. Each ROS word consists of a unique predetermined bit configuration grouped into 21 control fields (Table 1-6). The number of bits within a field determines the number of unique control signals (micro-orders) available within that field. (In a four-bit field, for example, 16 distinct micro-orders can be defined, only one of which can be activated at any one time.) The micro-orders are grouped functionally within the fields according to two rules:

1. Micro-orders that are functionally similar (such as micro-orders that control ingating to the AB register) are grouped in one field for ease of decoding.
2. All micro-orders grouped in a field must be mutually exclusive because only one micro-order within that field may be specified at a time.

Usually, rule 1 results in rule 2.

When decoded, each micro-order activates one or more control lines that condition gates to perform the function specified by the micro-order. Each micro-order is assigned a mnemonic code (up to 12 characters) that defines the control function performed. As an example, Table 1-7 lists the micro-orders and associated microcommands pertaining to control field V of the ROS word, referenced to the bit configurations of that field, and their function.

Each ROS word is represented by a block on a Control Automation System (CAS) Logic Diagram (CLD). The CLD is to the ROS microprogram what an ALD (Automated Logic Diagram) is to logic. The blocks are connected to show the logical sequence of ROS words to perform the specific function. Refer to ALD M7061 for a definition of the CLD format and content, and of the ROS block language and information contained within the block.

Table 1-6. ROS Word Breakdown

Bits	Control Field	Function Controlled
0–5	–	Spare
6–9	A	A-, B-, and IC-register ingating
10, 11	B	LS to S- and T-register ingating
12–16	C	PSW and S-, T-, D-, G-, and Q-register ingating
17–19	D†	F-register ingating, and end-op signals
20	–	Parity
21–24	E	E- and R-register ingating
25–30	F†	Status triggers and miscellaneous control lines
31–35	G†	Status triggers, IC, and miscellaneous control lines
36, 37	–	Spare
38–42	H	LS
43–46	L	Storage requests and setting of mark triggers
47–56	NA	Base address of next ROS word
57–61	K	Y-conditional branches
62–68	J	Z- (and X-) conditional branches
69–73	M	Serial adder A bus
74–77	N	Serial adder B bus
78–80	P	Parallel adder latches
81	–	Spare
82–84	Q	Hot 1's to parallel adder A-side
85	–	Parity
86	R	F- and AB-register outgating to serial adder A-bus
87–90	T	A-, B-, IC-, and F-register outgating to parallel adder B-bus
91	–	Parity
92–95	U	S-, T-, and D-register outgating to parallel adder A-bus
96	–	Spare
97–99	V	E- and Q-register outgating to parallel adder B-bus

† Control fields D, F, and G serve two functions. In the normal processing mode, they are decoded to yield standard CPU micro-orders; in the scan mode, they are identified as field S, and they yield special scan micro-orders (using common micro-order codes). The choice of modes is controlled by a 'scan mode' trigger.

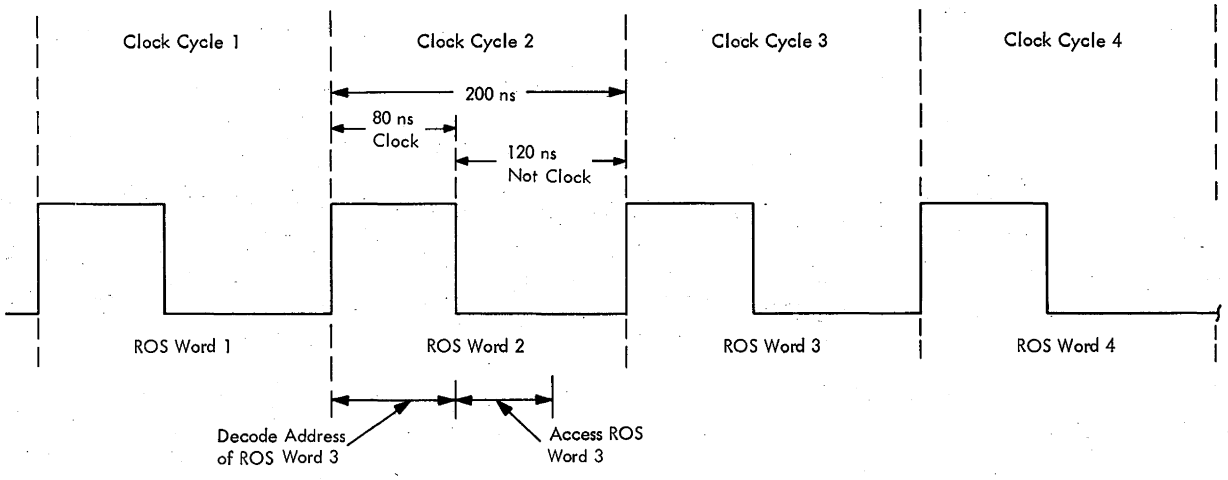
Table 1-7. Control Field V (Bits 97–99); E, Q outgates to Parallel Adder B-Bus

Bit Configuration			Micro-Order Mnemonic	Function
97	98	99		
0	0	0	0	Zero gated with parity
0	0	1	E3	E(12–15) to PB(60–63)
0	1	0	E2	E(8–11) to PB(60–63)
0	1	1	E23	E(8–15) to PB(56–63)
1	0	0	Q7	Q(52–63) to PB(52–63)
1	0	1	Q5	Q(36–47) to PB(52–63)
1	1	0	Q3	Q(20–31) to PB(52–63)
1	1	1	Q1	Q(4–15) to PB(52–63)

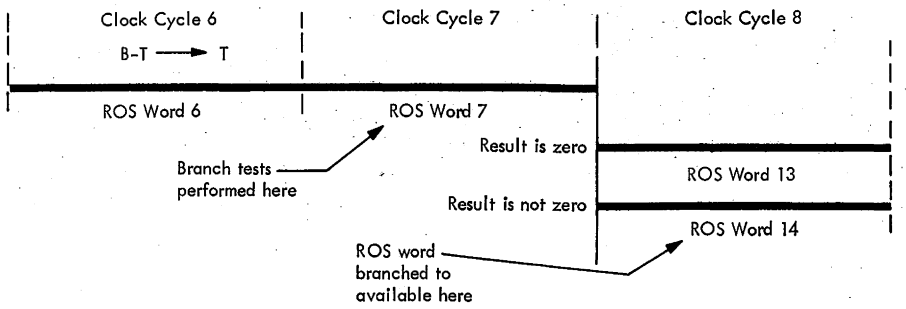
ROS Addressing and Branching

- Conditional branches are dependent on internal conditions of previous cycle.
- Word addressed as result of branch test is not available until 1 cycle later.
- ROS word is addressed by 12-bit binary address:
  0–9 is 10-bit base address.
  10 is Y-branch bit.
  11 is Z-branch bit.
- X-branch (functional branch) affects more than 1 bit.
- Overriding branch blocks 12-bit address and forces new 12-bit address into ROSAR.

As described earlier, the CPU cycle presently being executed is controlled by the ROS word addressed during the previous cycle. Referring to Figure 1-17, A, the normal sequence of ROS words is achieved by placing the address of ROS word 2 into ROS word 1, the address of ROS word 3 into ROS word 2, and so on. The address of the next ROS word is decoded during clock time of the cycle controlled by the present ROS word; the next ROS word is accessed during not-clock time of the cycle.



A. Sequential ROS Word Addressing



B. Conditional Branch ROS Word Addressing

Figure 1-17. ROS Addressing and Branching

Conditional branches are dependent on the internal conditions of the previous cycle. It is important to note that the ROS word addressed as a result of the branch test is not available until one cycle later. To explain this 1-cycle delay in addressing, assume that ROS word 6 contains the micro-orders necessary to subtract the contents of the T-register from the contents of the B-register, and to place the result into the T-register (Figure 1-17, B). Assume further that if the result is zero, a branch will be made to ROS word 13; if not zero, the next ROS word addressed will be 14. Contained in ROS word 6 is the address of ROS word 7, which defines the branch test and contains the associated branch addresses.

The results of the arithmetic operation performed in cycle 6 are tested during clock time of cycle 7. It is during this time that the address of ROS word 13 or 14 (depending on the results of the branch test) is decoded: the selected ROS word is accessed during not-clock time of cycle 7. Hence, the ROS word branched to as a result of the arithmetic operation performed during cycle 6 is not available until cycle 8.

ROS words are selected by means of a 12-bit binary address. The address is held in the ROS address register (ROSAR), whose bit positions are numbered 0 through 11, high order to low order. Bits 0 through 10 specify a 200-bit doubleword in ROS; bit 11 gates the proper 100-bit half to the ROS sense latches (Figure 1-18).

The 12-bit address is made up of three components:

1. A 10-bit base address, bits 0–9.
2. A conditional branch test, or an unconditional value of 0 or 1 applying to bit 10, designated Y-branch.
3. A conditional branch test, or an unconditional value of 0 or 1 applying to bit 11, designated Z-branch.

Included in the Z-branch field of micro-orders is a subset of branch micro-orders called X-branch or functional-branch micro-orders. The X-branch micro-orders affect more than one bit of the ROS address.

Included in the Y-branch field of micro-orders is a subset of overriding branch micro-orders. When the conditions tested by these micro-orders are satisfied, the full 12-bit address is blocked and a new 12-bit address is forced into ROSAR.

If branching conditions are to be tested, the address bits that may be affected by the branch must be set to 0's, except in the case of an overriding branch. If the branch is satisfied, 1's are forced into the ROSAR bit positions associated with that branch test; if the branch condition is not satisfied, the bits remain 0's. Thus the address is modified only if the branch is to be taken.

Addresses can be grouped into four categories: (1) no branch specified, (2) Y- and/or Z-branches, (3) X-branches, and (4) overriding branches. The following paragraphs discuss the addressing for each category. Refer to ALD M7061 for a definition of ROS addressing and branching terms used in the following paragraphs.

No Branch Specified. If no branch tests are to be made, there is only one possible ROS word that can follow, and hence only one possible next address. Accordingly, the 10-bit base address (bits 0–9) and absolute values of 1 or 0 for bits 10 and 11 are specified. The micro-orders that unconditionally set an absolute value into bits 10 and 11 are:

1. 0 in left of R-line (K0), which sets bit 10 to a 0.
2. 1 in left of R-line (K1), which sets bit 10 to a 1.
3. 0 in right of R-line (J0), which sets bit 11 to a 0.
4. 1 in right of R-line (J1), which sets bit 11 to a 1.

The appropriate Y- and Z-branch micro-orders are selected, and bits 10 and 11 are set correspondingly.

Y- and/or Z-Branches. Conditional branch addresses may be specified in which bits 10 and/or 11 are affected.

Only a Y-branch can be executed as follows. A 10-bit base address and an absolute value or X for bit 11 are specified. A branch test is defined in the Y-branch micro-order control field. If the branch condition is satisfied, bit 10 is set to a 1; if not, bit 10 remains a 0. For example, micro-order 'WCRY' sets bit 10 to a 1 if a carry is detected in the serial adder. If there is no carry, bit 10 remains a 0.

Conversely, only a Z-branch can be executed as follows. Here, a 10-bit base address and an absolute value for bit 10 are specified. If the branch test defined in the Z-branch micro-order control field is satisfied, bit 11 is set to a 1; if not, bit 11 remains a 0.

Certain situations require the use of both Y- and Z-conditional branches simultaneously. The 10-bit base address is specified, and bits 10 and 11 may assume one of the following values:

Bits 10	Bits 11	Branch Results
0	0	Y- and Z-branch conditions both unsatisfied.
0	1	Z-branch condition only satisfied.
1	0	Y-branch condition only satisfied.
1	1	Y- and Z-branch conditions both satisfied.

X-Branches. Where a branch to one of four or more possible addresses is required (as well as some special 64-way branch tests), an X-branch is used. The X-branch may affect bits 10 and 11 (four-way branch), bits 9–11 (eight-way branch), bits 8–11 (16-way branch), or bits 6–11 (64-way branch). An example of an X-branch is the 64-way branch, 'E(2–7)→ROA', made at the end of the I-Fetch sequence per the op code to enter the execution phase of the specific instruction.

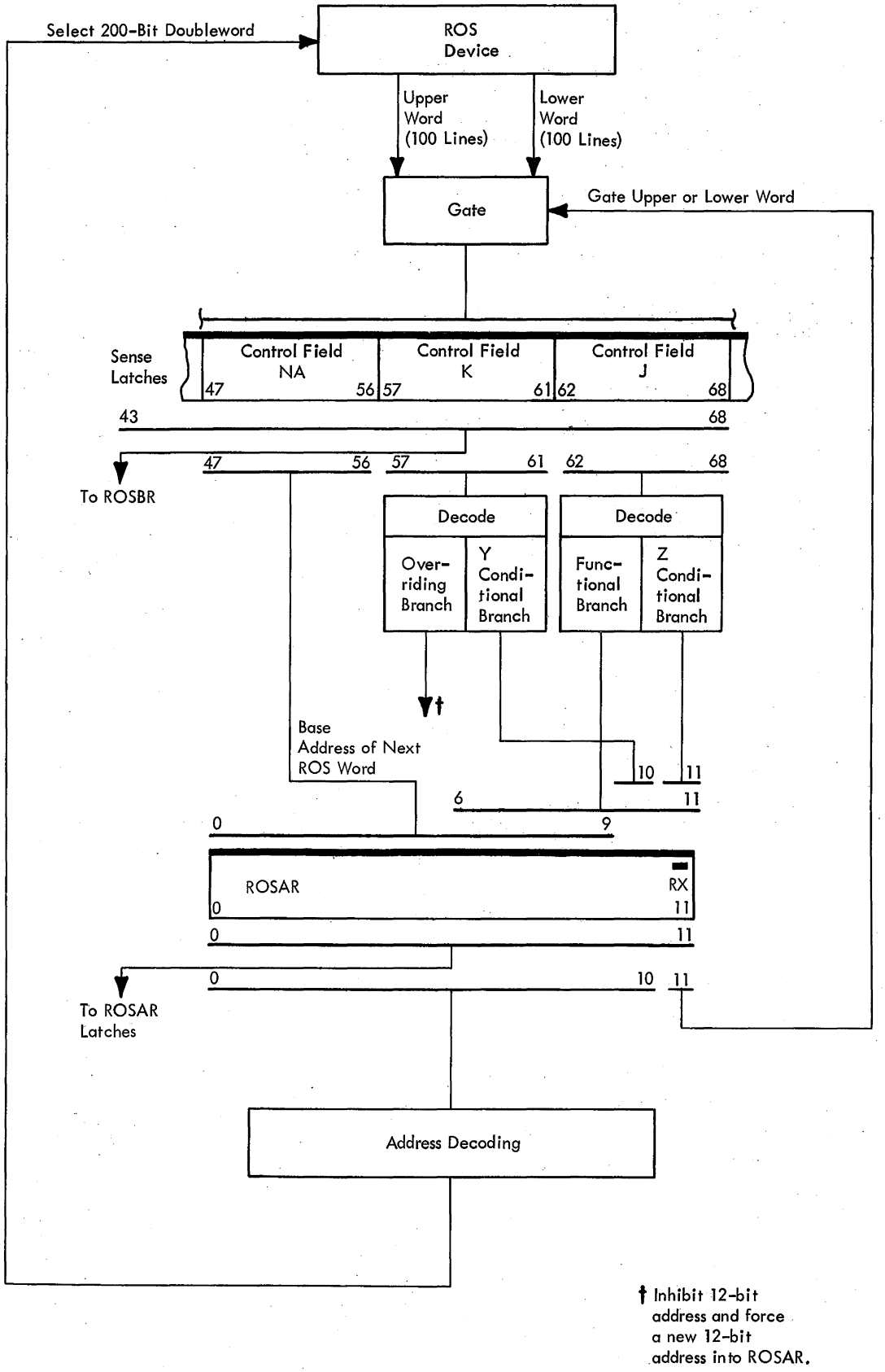


Figure 1-18. ROS Addressing Block Diagram

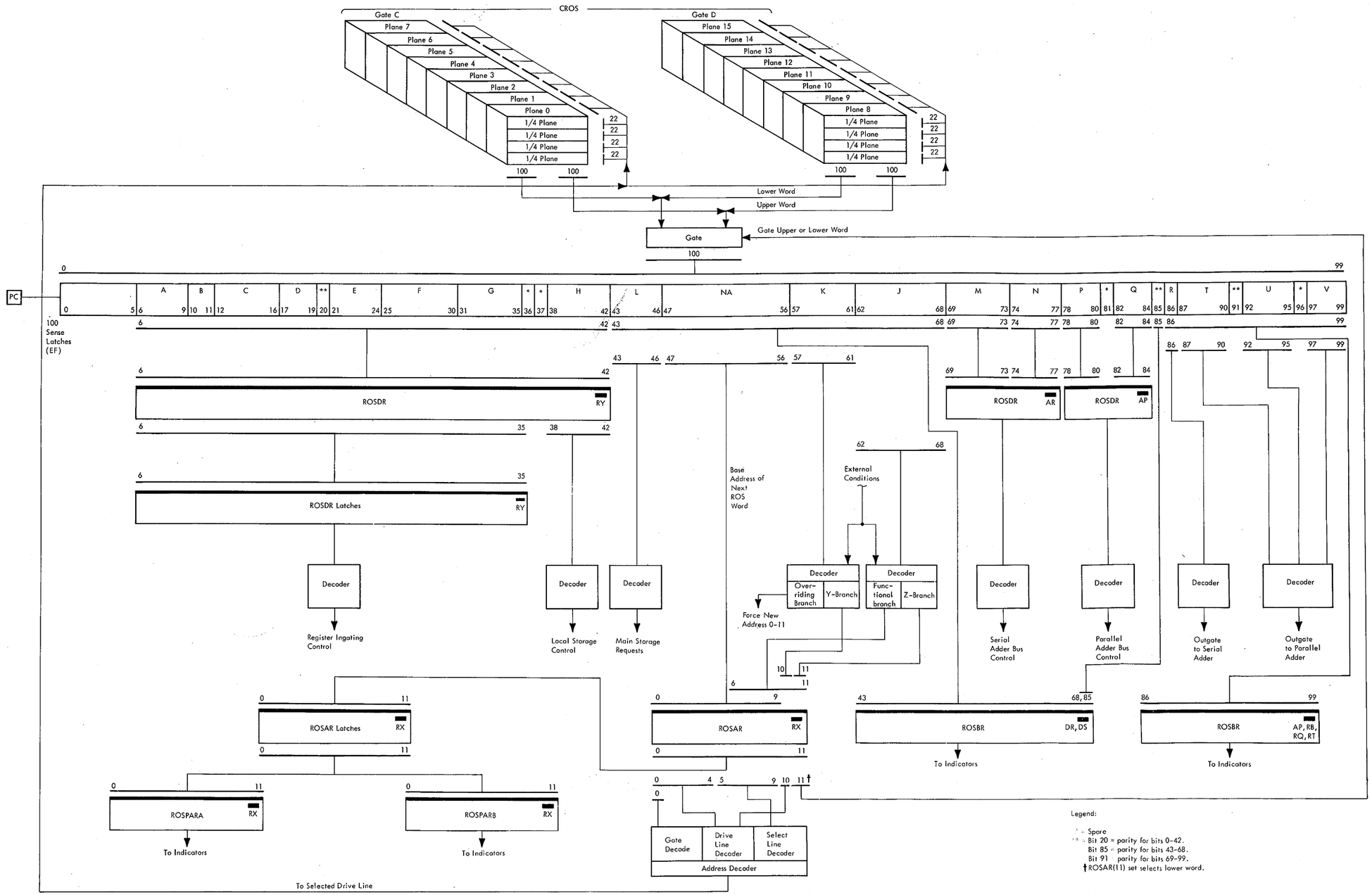


Figure 1-19. ROS Data Flow

For these multiway branches, one condition sets the associated address bit to a 1. To illustrate, assume conditions A, B, and C sets bits 9, 10, and 11, respectively, to a 1. The possible results are:

Bits 9	Bits 10	Bits 11	Branch Results
0	0	0	None of the conditions is satisfied.
0	0	1	Condition C is satisfied.
0	1	0	Condition B is satisfied.
0	1	1	Conditions B and C are satisfied.
1	0	0	Condition A is satisfied.
1	0	1	Conditions A and C are satisfied.
1	1	0	Conditions A and B are satisfied.
1	1	1	All three conditions are satisfied.

The addressing is similar to that previously discussed. A 10-bit base address is specified, with those bits that may be affected by the X-branch set to 0. Thus, for the example given above, bit 9 of the base address is set to 0, the Y-branch micro-order control field contains micro-order 0 in left of R-line to set bit 10 to 0, and bit 11 is automatically set to 0 when the X-branch is specified. Subsequently, the bit(s) associated with successful condition(s) is set to 1. The ROS word addressed will be that ROS word whose address satisfies the branch conditions.

Overriding Branches. There are exceptional machine conditions (such as interruptions) for which the normal ROS word sequence must be stopped and a new sequence started. This change is accomplished by an overriding branch specified in the Y-branch micro-order control field.

The normal sequencing address is made up of (1) the 10-bit base address and (2) bit 11 set to 0 automatically because the overriding branch is a function of the Y-field. Because the overriding branch is specified in the Y-control field, no Y-branch can be specified.

If the overriding branch condition is satisfied, the normal full 12-bit address is blocked and a new address, as determined by the overriding branch condition, is forced into ROSAR.

ROS Data Flow

- ROS data flow units are:
  100 sense latches, 1 per ROS word bit
  ROSAR
  ROSAR latches
  ROSPARA and ROSPARB
  ROSDR
  ROSDR latches
  ROSBR
  Decode logic
- Control fields may be:
  Decoded directly from sense latches.
  Placed into ROSDR and subsequently decoded.
  Placed into ROSDR, sent to ROSDR latches, and then decoded.

The 100-bit ROS word is divided into 21 control fields. When read out from ROS, the ROS word is placed into 100 sense latches, one latch for each bit position. The control fields are handled according to the functions they control. They may be (Figure 1-19):

1. Decoded directly from the sense latches (control fields L, K, J, R, T, U, and V) or transferred directly to ROSAR (control field NA).
2. Placed into the ROS data register (ROSDR) and decoded (control fields H, M, N, P, and Q).
3. Transferred to ROSDR latches from ROSDR (control fields A–G).

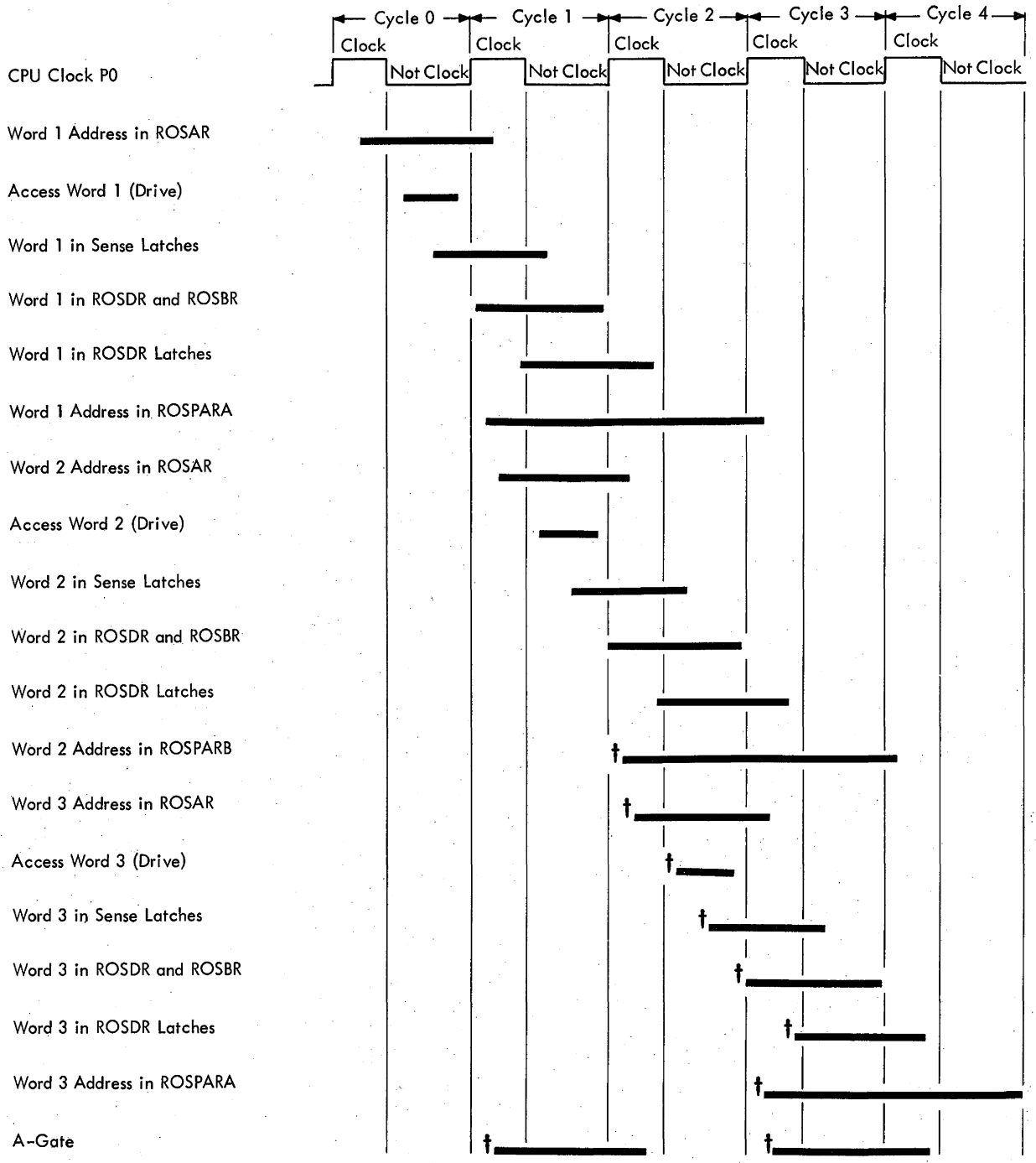
Assuming ROS words and the cycles they control are designated 1, 2, and 3, ROS word 1 is set into the sense latches during not-clock time of cycle 0 (Figure 1-20). The control fields used during clock time of cycle 1 are decoded directly from the sense latches (case 1 above). These control fields, which may be considered critical timing fields, control inputs to the adders, define the base address and branch tests for the next ROS word, and control the storage-request and mark triggers.

Although the sense latches are cleared at not-clock time of cycle 1, the control fields of ROS word 1 that are required during that time (case 2 above) are placed into ROSDR at clock time of cycle 1. These signals control the adders and LS. Note that both portions of the ROSDR associated with the adders are packaged physically with the adders they control. The balance of the ROSDR serves control fields A–H.

Control fields A–G control register inputs and triggers that are to be set during clock time of cycle 2 (case 3 above). Although the ROSDR is reset at the end of cycle 1, the ROSDR latches keep control fields A–G available for that additional 80 ns (Figure 1-20).

Control fields L, NA, K, J, R, T, U, and V are sent to the ROS backup register (ROSBR) from the sense latches. The ROSBR does not play a part in the ROS functions; it provides an indication of the subject fields during maintenance (test) mode. When the CPU stops on an error during test mode, the ROSBR contents can be used by the CE to help isolate malfunctions.

The NA control field, in addition to being stored in the ROSBR, is stored in ROSAR and provides the base address as previously explained. During each ROS cycle, the contents of ROSAR are sent to the ROSAR latches which, in turn, are alternately gated (by means of an 'A-gate' signal) to the ROS previous address A (ROSPARA) and ROS previous address B (ROSPARB) registers. These registers serve the same purpose as the ROSBR; i.e., provide the CE with an indication for maintenance use during test



† If an error is detected in cycle 1, these steps are not performed.

Figure 1-20. ROS Timing

mode. When an error causes the CPU to stop in test mode, ROSAR, ROSPARA, and ROSPARB provide the addresses of the next ROS word, current ROS word, and previous ROS word (Figure 1-20).

ROS Control of CPU

Efficient control of CPU operations is achieved by overlapping ROS words. Clock signals (P0–P19) time ROS sense latches, ROSDR, and ROSDR latches, thus allowing the processing of parts of more than one ROS word simultaneously To illustrate, Figure 1-21 shows the ROS word timing relationship for a hypothetical example.

The address of word 1 is gated into ROSAR from the ROS sense latches at P5 of word 0. [ROSAR(11) may be set as late as P7.] Information from word 1 enters the ROS sense latches at P0 + 160 ns (P16). (In normal operation, a

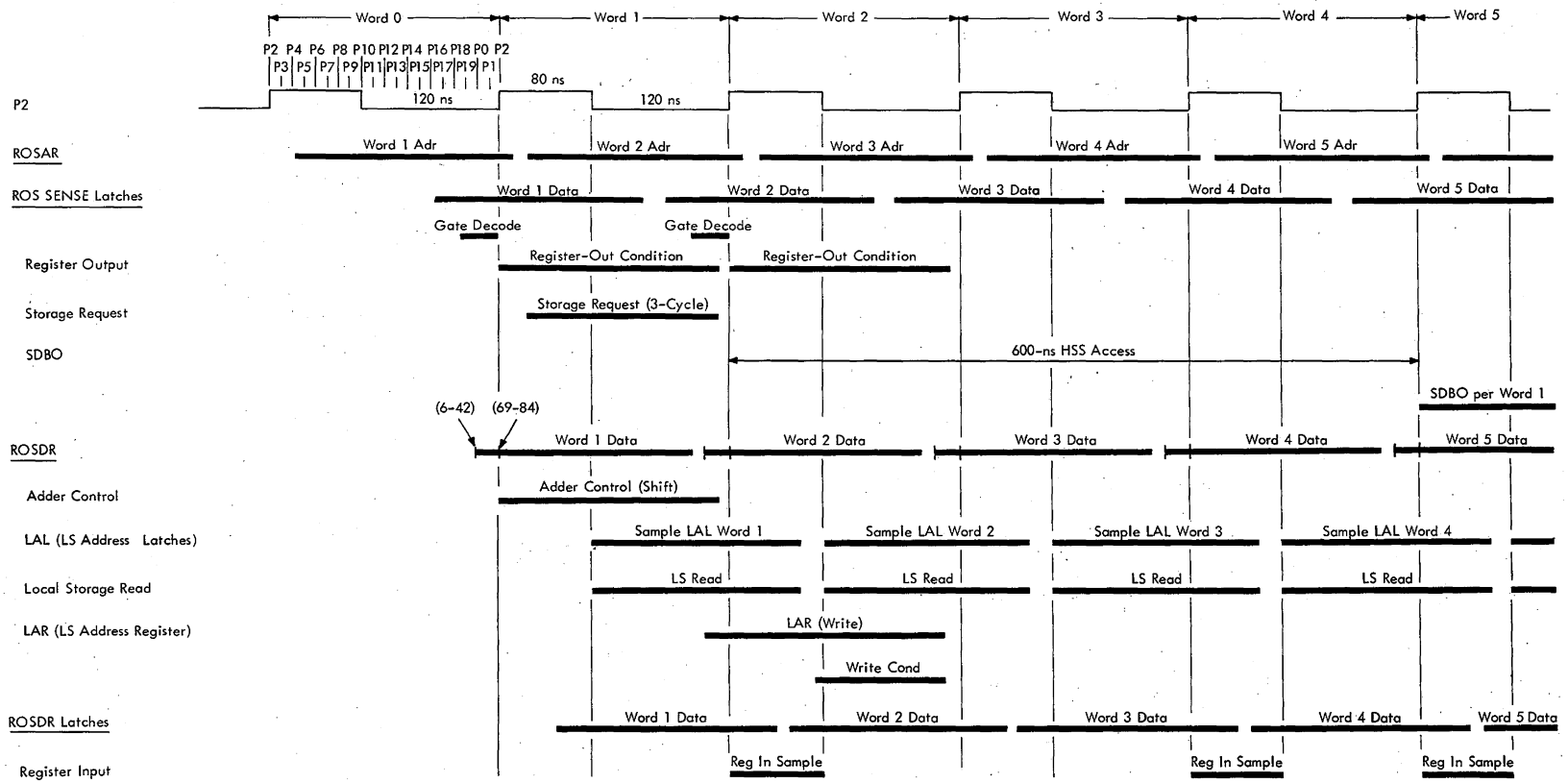


Figure 1-21. ROS Control of CPU Operations

new word enters the ROS sense latches every 200 ns.) In the example, word 1 controls: (1) register output, (2) main storage request, (3) adder shift, (4) local storage write, and (5) register input. A register output micro-order gates register data into an adder at P2 by means of 'gate control' triggers. A three-cycle main storage request is initiated at P4 to fetch information which is used three cycles later. Note that register output and the main storage request are decoded directly from the ROS sense latches.

Bits 6–35 and 38–42 enter ROSDR at P0, and bits 69–80 and 82–84 enter ROSDR at P2; they are decoded for adder control and LS operations. In the example, micro-orders generate an adder shift and a local storage write operation. The shift is performed immediately, but the local storage write operation is delayed because a local storage read operation is automatically set up first. The LS address is entered into the local storage address latches (LAL); a read operation is performed, but data is not gated into a register.

The address then is gated into the local storage address register (LAR) to perform the write operation. Note that the write condition, ordered by word 1, starts after word 2 has been transferred to the ROSDR. The figure shows an LS read operation for every word because this sequence happens even if it is not ordered. When no order is given to LS, a readout of LS address 0 is performed but the data is not used. Forcing the read operation saves time if it is needed.

ROSDR(6–35) is gated to the ROSDR latches at P7 to retain word 1 information at the same time word 2 is set into ROSDR. In the example, word 1 transfers data into a register at clock time (P2) of word 2. This action, along with an LS write operation, illustrates ROS word overlap because these operations are performed at the same time the register out condition is energized from the decoding of word 2.

Word 2 has only one micro-order, register output, but gates are conditioned to transfer word 2 data from the ROS sense latches to the ROSDR and then to the ROSDR latches. Note that the LS read micro-order is active, though not ordered. Word 3 operates in a similar manner, but the only micro-order is a register input which takes place during word 4.

As the ROS words are executed, the main storage request is processed and data is returned on the Storage Data Bus Out (SDBO); word 4 contains the micro-order to gate the data into a register for further processing.

PSW Register

Program status words (PSW's) contain detailed information pertaining to the particular mode in which the CPU is operating. These status words are composed of a system mask, storage key, program state, interruption code, instruction length code, condition code, program mask, and an instruction address that enables the interrupted program to resume at the correct location.

Status information concerning the current operating program is contained in several groups of triggers, from where it controls all system operations essential to that particular program mode. These groupings of control triggers, although not adjacently located in logic, are collectively referred to as the PSW register. Although a completely assembled PSW is 64 bits long, only 24 positions of status word data are contained in the PSW register. The remaining information (generated by the CPU at the time of the interruption) is not retained when a previously stored PSW is reloaded, because its function is only to identify the cause of the interruption and to return the CPU to the correct program location. (This information is gated directly from ST when the old PSW is recalled from main storage.) Figure 1-22 shows an assembled PSW and those areas of control information retained in the PSW register.

BUS CONTROL UNIT

In the 2065, the BCU responds to storage requests from the CPU and from up to seven I/O channels, all of which may be operating asynchronously with respect to each other. The flow of information between main storage and all units serviced by the BCU is effected through a single bus system. Because the bus system is shared by all units, the BCU must resolve conflicts between simultaneous storage requests from these units and ensure that the storage bus system is available to one unit at a time. Thus, the major function of the BCU is to provide for efficient time-sharing of the storage bus system by all units.

Each unit requiring access to storage communicates with the BCU through individual control lines. These lines are monitored in the BCU to establish priority between storage requests and to inform the requesting units of the bus system availability.

When priority is awarded to a requesting unit, the BCU decodes the high-order bits, and bit 20, of the storage address supplied by that unit. The BCU then sends a 'select' signal to the proper storage unit. The selected main storage unit initiates a storage cycle and decodes the low-order bits (6–19) of the address supplied on the storage address bus (SAB) to access the required doubleword location. When the BCU grants storage access to one unit, storage requests from all other units remain pending until the current data transfer operation is completed. At that time, the BCU performs a new priority test on all units (including the unit just serviced).

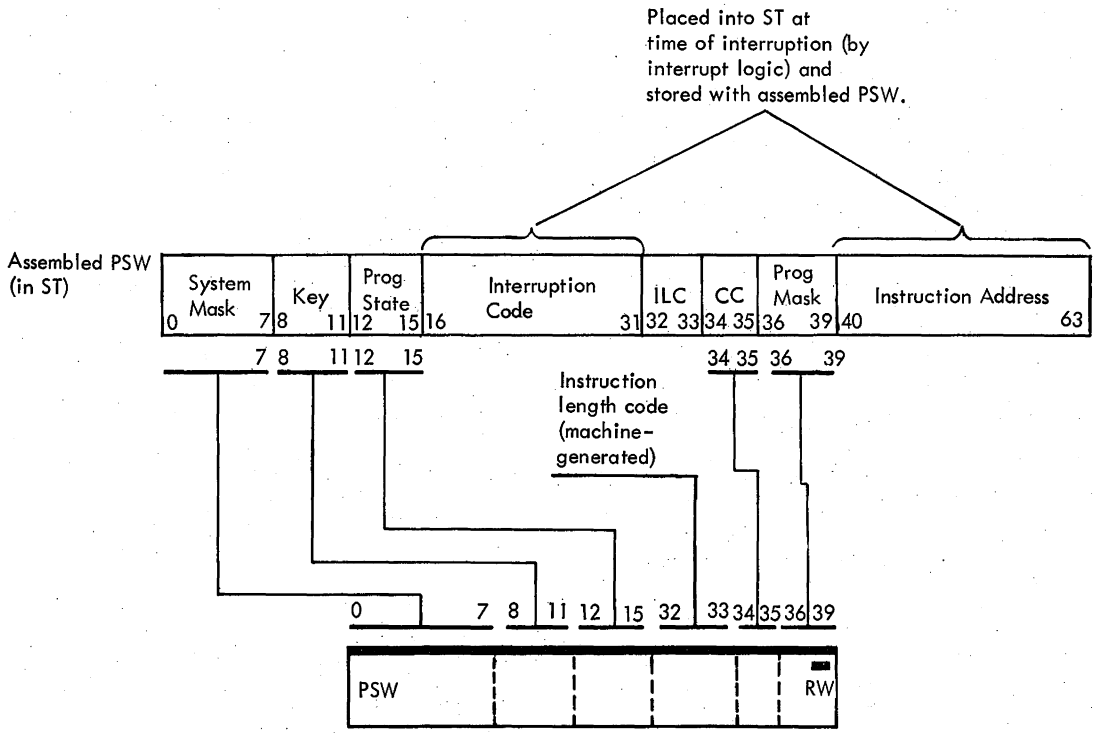


Figure 1-22. Status Information Contained in PSW Register

Major Interface Lines

A simplified diagram of the BCU interface with the Model 65 system is shown in Figure 1-23. This interface consists of multiplex data bus lines, and of simplex and multiplex control lines. By definition, a multiplex line is a line that is shared by *more than two* units of the system; for example the SAB, which connects the BCU and all the I/O channels to all storage units in the system. A simplex line is a line that is used exclusively between two units of the system. For example, individual 'select' lines are used to interconnect the BCU to *each* storage unit in the system.

The major interface lines shown in Figure 1-23 are defined as follows:

1. Address and data buses.
   a. Storage Address Bus (SAB). This multiplex bus connects the CPU and all I/O channels to all main storage units in the system. The SAB consists of 21 address lines (plus three parity lines) and specifies the address of a doubleword contained in one of the main storage units associated with the system. Note that the SAB path is different from the other buses. This difference is necessary because SAB(0–6, 20) from the CPU and from the channels must enter the BCU storage address decoder to select a storage unit.
   b. Storage Data Bus Out (SDBO). This multiplex bus connects the CPU and all I/O channels to all main storage units in the system. The SDBO consists of 64 data lines (plus eight parity lines) and carries data requested from main storage by the CPU or the I/O channels.
   c. Storage Data Bus In (SDBI). This multiplex bus connects the CPU and all I/O channels to all main storage units in the system. The SDBI consists of 64 data lines (plus eight parity lines) and carries data to be stored into main storage. This data may be supplied by the CPU or by the I/O channels.
   d. Mark Bus. This multiplex bus connects the CPU and all I/O channels to all main storage units in the system. The mark bus consists of eight mark lines (plus one parity line) and designates the bytes on the SDBI that are to be stored into main storage. (There is one mark line for each byte on the SDBI; complete absence of mark signals occurs only on a fetch operation.)
   e. Key-In Bus. This multiplex bus connects the CPU and all I/O channels to all main storage units in the system. The key-in bus consists of five key lines (plus one parity line) and transfers the storage protection key from the CPU or the I/O channels to the storage protection area in the selected main storage unit.
   f. Key-Out Bus. This multiplex bus connects the CPU to all main storage units in the system. The key-out bus, which consists of five key lines plus one parity line, is used only during execution of the insert-key operation by the CPU. This operation transfers the storage protection key from the selected main storage unit to the CPU.

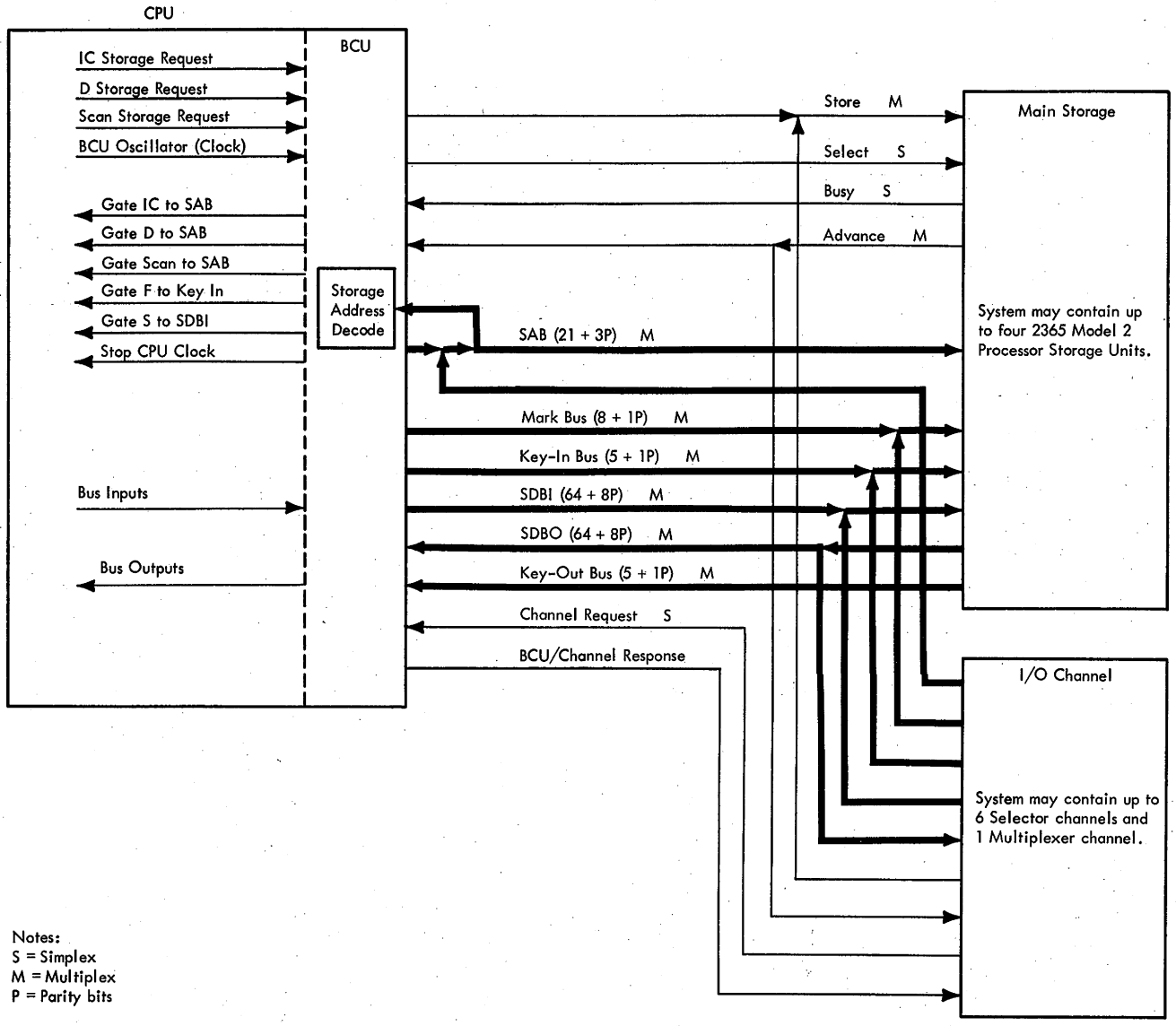


Figure 1-23. Basic BCU Interface

On channel operations, the mark bus, key-in bus, SDBI, and SDBO are independent of the BCU.

2. Major control signals.
   a. 'Select'. This simplex signal is issued by the BCU to the selected storage unit. This signal causes the selected unit to gate in the address from SAB and initiate a storage cycle.
   b. 'Store'. This multiplex signal is issued on all store operations performed by the CPU or by the channels to instruct the storage unit to store the contents of the SDBI as specified by the mark signals. The absence of a 'store' signal indicates a fetch operation.
   c. 'Busy'. This simplex signal is sent from a storage unit to the BCU to signify that the storage unit is in a store cycle.
   d. 'Advance'. This multiplex signal is issued by the storage unit to the BCU and all channels to indicate that the storage unit is about to gate data onto the SDBO. (The 'advance' signal is received approximately 200 ns before the data arrives on the SDBO.)
   e. 'Channel request'. This simplex signal is issued by a channel to the BCU when the channel needs access to main storage.
   f. 'BCU/channel response'. This simplex signal is issued by the BCU to the requesting channel when the BCU has awarded priority to that channel.

Within the CPU, storage requests are generated and sent to the BCU to develop a CPU request. The signals that perform this function are related to the source of the storage address, as follows:

1. IC storage request. The storage address is in the instruction counter.
2. D-storage request. The storage address is in the D-register.
3. Scan storage request. The storage address is developed in scan logic.

Representative signals developed by the BCU to control the buses are:

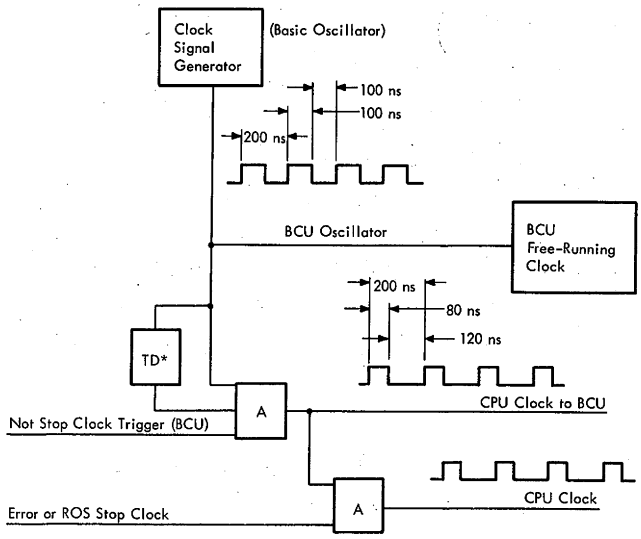
1. 'Gate IC to SAB'. Gates the contents of the instruction counter to SAB.
2. 'Gate D to SAB'. Gates the contents of the D-register to SAB.
3. 'Gate scan to SAB'. Gates the address developed by the scan logic to SAB.
4. 'Gate F to key-in'. Gates the contents of the F-register to the 'key in' bus.
5. 'Gate S to SDBI'. Gates the contents of the ST register to SDBI.
6. 'Stop CPU clock'. Stops the CPU clock when unable to grant priority to a CPU request.
7. 'Gate SDBO to AB'. Gates the data on the SDBO into the AB register.
8. 'Gate SDBO(0–31) to S(0–31)'. Gates SDBO(0–31) into the S-register.
9. 'Gate SDBO(32–63) to T(32–63)'. Gates SDBO (32–63) into the T-register.
10. 'Gate SDBO to Q(0–63)'. Gates SDBO(0–63) into the Q-register.

BCU Clocks

The timing of latches and triggers within the BCU is controlled by signals from the 2065 clock signal generator (Figure 1-24); thus the BCU operation is synchronized with the CPU. The BCU has two clocks: a free-running clock, which is active whenever power is applied to the 2065, and a 'CPU clock to BCU' signal. As shown in Figure 1-24, an error or the ROS microprogram can stop the CPU clock without interfering with the BCU clocks.

The free-running clock controls the timing of priority selection, storage unit selection, and channel signals. Additional controls are unnecessary for servicing channel requests because the BCU transfers control of the buses to the channel after the storage unit selection is made. Thus, channel storage requests are made independently of the CPU.

CPU requests, however, require additional gating controls. To control information to and from the buses, the BCU retains control of the buses and develops gating signals which are timed by the 'CPU clock to BCU' signal. When a CPU request cannot be processed because the BCU or storage unit is busy, the CPU clock and the BCU-developed gating signals are disabled by a 'stop clock trigger' signal to prevent transfer of the wrong information (Figure 1-24). The BCU restarts the clocks after it awards priority to the CPU.



* TD converts symmetrical clock signal to unsymmetrical clock signal.

Figure 1-24. BCU Clock Logic

BCU Operation

The BCU provides efficient time-sharing of the main storage by the I/O channels and the CPU. Storage requests are not honored on a first-come, first-served basis, but rather on a priority basis. Figure 1-25 shows the BCU priority logic; channel 1 has highest priority, followed by channels 2, 0, 3, 4, 5, 6, and CPU. When a channel or the CPU is granted

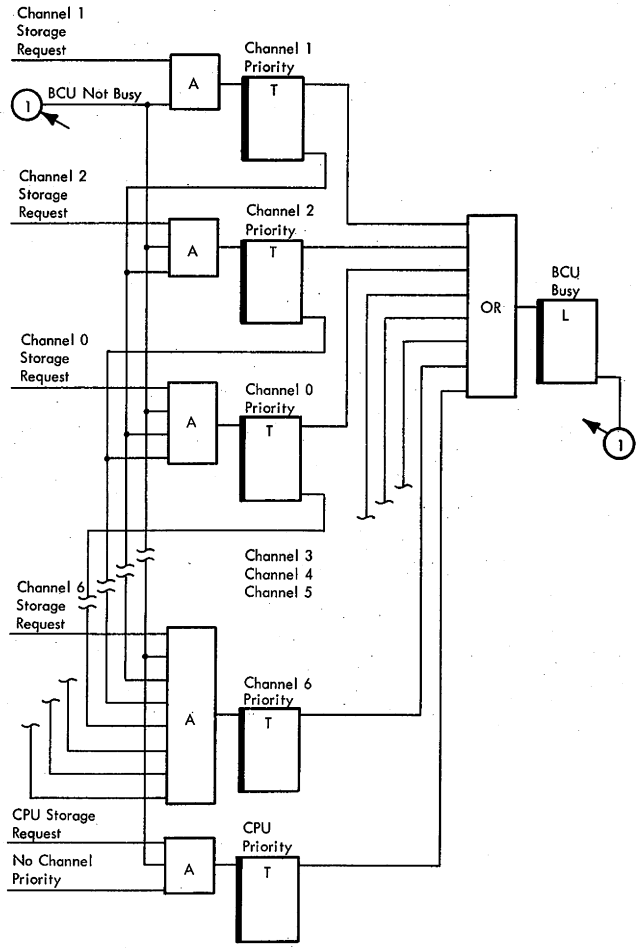


Figure 1-25. BCU Priority Logic

priority, the 'BCU busy' latch is set so that additional requests are not honored until the storage cycle is completed. For example, if after a channel 1 request a second channel 1 request is made before the 'BCU busy' latch is reset, channel 1 is selected again, even if other requests are pending. As another example, if a channel 0 request is made during a storage cycle and is followed by a channel 2 request before the end of the cycle, channel 2 is awarded priority.

CPU Request

A CPU request for storage is generated from one of three sources: (1) instruction fetch logic to refill the instruction buffer (Q-register), (2) microprogram to fetch or store data, and (3) scan logic. In each case, the request is made three or four cycles before the data is needed to allow time for storage access.

The address of the desired doubleword in storage is located in the instruction counter or the D-register, or is generated by the scan logic. BCU develops the gating signals to supply the address from the specified source to SAB. At the same time, BCU controls gating for SDBI, SDBO, 'mark' bus, and the storage address protection keys. When a CPU request for storage is sensed at the BCU, the 'CPU request' trigger is set (Figure 1-26). If the CPU is granted priority during the next priority test, the 'CPU priority' trigger is set, and the storage sequence is started immediately.

If the BCU is busy and cannot grant priority to the CPU, a 'stop clock trigger' signal is sent from the BCU to stop the CPU clock (Figure 1-24). This action stops all CPU activity until the BCU grants priority to the CPU and the storage cycle starts. When the BCU and storage are available, the CPU clock is restarted and the CPU sends or receives data over the buses. The clock signal from the CPU keeps the BCU synchronized with ROS and other CPU functions.

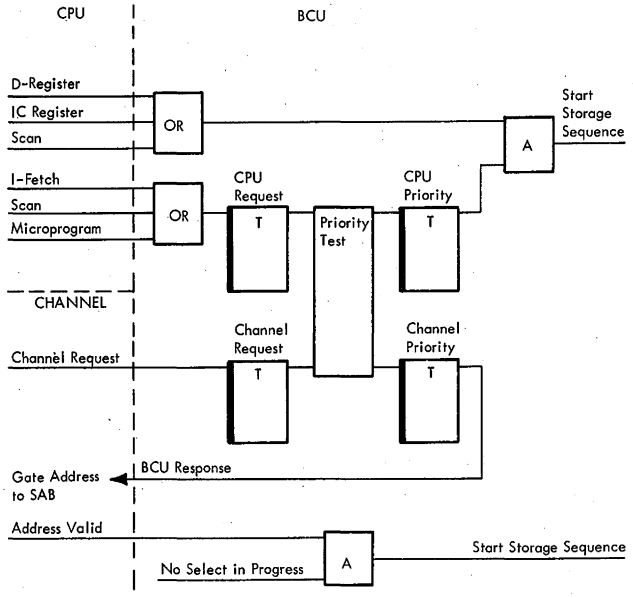


Figure 1-26. Start Storage Sequence Logic

Channel Request

When a channel has been granted priority, the BCU sends a 'BCU response' signal to the channel so that the channel will put the storage address on SAB (Figure 1-26). After the address has been gated to SAB, the channel sends an 'address valid' signal to the BCU. This signal does *not* denote a correct address, but signifies that the available address has been gated to SAB. The BCU uses the 'address valid' signal to sample the SAB.

After the storage unit has been selected, the BCU turns control of the multiplex buses to the channel. Channel registers provide data, marks, and address protection keys as needed. When the CPU receives a 'release' signal from the channel, the CPU is free to continue processing except for conflicts in requesting main storage.

Operation with Main Storage

- Address is decoded to select storage unit.
- Invalid address results if storage capacity is exceeded or if power is off at the selected unit.
- Interleaving reduces storage access time.

A portion of SAB is decoded in the BCU to select the storage unit that contains the addressed doubleword. SAB(4,5) determines the high speed storage (HSS) unit, and SAB(6 or 20) specifies the odd or even half for interleaving. Figure 1-27 shows the data path for the address decoding for HSS and the optional large capacity storage (LCS). Decoding of a 'select' line sends a 'select' signal on a simplex line to the desired storage unit where the address on SAB is decoded to locate the addressed doubleword.

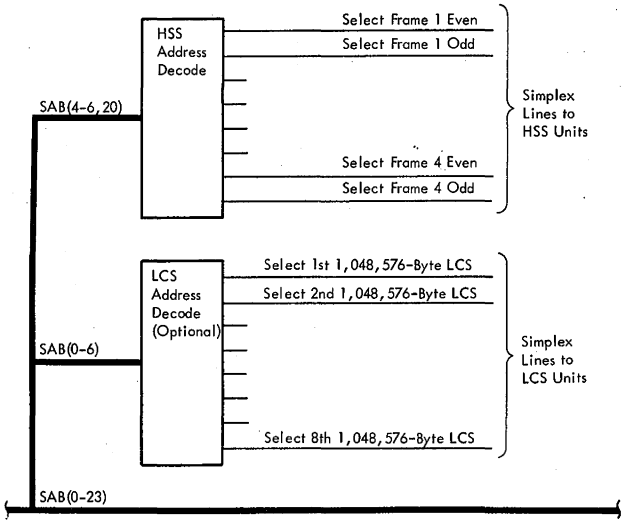


Figure 1-27. Storage Selection

Within the BCU decoder, a test is made for an invalid address condition. Jumper cards are set to define the limit of the storage attached. If the address specifies a value beyond the storage limit, an 'invalid address' signal is developed. The 'invalid address' signal is also developed if power is off at the specified storage unit. The unit originating the address is notified, and an interruption may occur.

The basic storage cycle of the 2365 Processor Storage is 750 ns, but interleaving between odd and even sections reduces the effective storage time to 400 ns. This reduction is possible because each area of storage contains its own bus arrangement, thus enabling independent operation. An address enters storage and is used during the first half cycle to set up the storage access. The second half cycle is used to bring out and store data. During the second half cycle, a new address is sent on the SAB to the other half (odd or even) of storage, thus allowing a storage access in approximately half the normal storage cycle. A delayed (by 80 ns) 'select' signal conditions the BCU reset logic. The delay holds the BCU busy through the cycle after the storage request to prevent a second storage request within 400 ns.

INSTRUCTION FETCHING

The processing of an instruction is divided into two phases: instruction fetching and execution. Instruction fetching, or I-Fetch, retrieves instructions from main storage and performs operations common to many instructions. For the most part, I-Fetch, which is controlled by ROS and conventional hardware:

1. Determines the address to be placed into the instruction counter (IC).
2. Fetches instructions from main storage.
3. Determines the instruction format (RR, RX, RS, SI, or SS).
4. Calculates the effective operand address (adds the D-field, the contents of the LS register designated by the B-field, and the index, if required) for those formats that require that function.
5. Places the operands specified by RX format instructions into the applicable registers (AB, ST, and D). For the other formats, I-Fetch issues a storage request for the second operand. The second operand is placed into the registers during the execution phase.
6. Passes control to the specific execution phase by means of a 64-way branch.

The transition from the execution phase of an instruction to the I-Fetch sequence of the next instruction is achieved by an end-op cycle, the last cycle of the execution phase. The end-op cycle completes the execution phase of the instruction being processed by:

1. Setting the condition code to reflect the result of the instruction, if applicable.
2. Detecting exceptional conditions and interruptions.

The end-op cycle initiates the I-Fetch sequence for the next instruction by:

1. Decoding the format of the next instruction.
2. Initiating operand fetches as required by that format.
3. Performing a 64-way branch to establish the correct I-Fetch sequence for that format.
4. Fetching more instruction halfwords, if required.

Functional Units Used

Five registers play vital roles in the I-Fetch sequence: Q, R, E, IC, and D. The following paragraphs discuss the functions generally performed by these five registers.

Q-Register

- Buffers four instruction halfwords received from main storage.
- Provides for overlap of I-Fetch and instruction execution.
- Transfer of B- and D-fields from Q reduces instruction processing time.

The Q-register is a 64-bit (plus eight parity bits) trigger register that buffers all instructions entering the CPU from main storage (Diagram 3-2, FEMDM). It is divided primarily into four halfword (16-bit) areas. This arrangement provides for the buffering of four instruction halfwords (eight bytes), thus increasing CPU efficiency and reducing the amount of main storage time required by the CPU. The Q-register is loaded directly from the SDBO; information is transferred to the LS address register [LAL (Read) and LAR (Write)], to the R-register, and to the parallel adder.

After being loaded with a doubleword from main storage, those Q-register halfwords containing instruction op codes are sequentially transferred to R (for subsequent execution in E). When the last op-code halfword has been transferred from Q, a new doubleword is again loaded into Q from main storage. This process of continuously refilling Q with instructions is overlapped with instruction execution whenever possible.

Additional Q-register information selects the instruction fields to be sent to LAR and to the parallel adder. Such information consists of four four-bit fields (B-fields) specifying LS registers and four 12-bit fields (D-fields) containing the displacement for main storage addresses. Transferring this information directly from Q instead of via R or E provides a lookahead capability by allowing both LS and effective addresses to be available before the execution time of the associated instruction. Transferring of these 4- and 12-bit fields is performed selectively so that the information is associated with the correct instruction.

Before an instruction is executed, it is tested for odd parity. The op-code halfword is tested in the E-register. The remaining halfwords, if any, are tested by the parallel adder half-sum checking circuits as the effective address is calculated.

R-Register

- Only op-code halfwords are transferred from Q to R.
- Selection of op-code halfword is determined by IC(21,22).

The R-register is a halfword (16 bits plus two parity bits) trigger register, providing intermediate buffering of op-code halfwords between Q and E (Diagram 3-2, FEMDM). This buffering extends the total instruction buffering capability to five halfwords (five instructions in the event of all RR formats), as Q is normally refilled after the last op-code halfword has been transferred to R. The use of two separate registers (Q and R) for containing op-code halfwords also provides double buffering. This scheme allows storage requests to be generated immediately upon transferring the last op-code halfword from Q, instead of having to wait until the instruction in E has been executed, as would be required if halfwords from Q were transferred directly to E.

Because op-code information is all that is required to initiate execution, only those halfwords in Q containing op codes are gated to R. Also, because RX, RS, SI, and SS instructions are composed of either two or three halfwords (only the first of which contains the op code), it is necessary to select the proper halfword to be transferred to R, rather than merely proceeding sequentially through the four halfwords. Selection of the halfword for transfer to R is determined by IC(21,22) as follows. Depending on the format, instructions may be 1, 2, or 3 halfwords long. The number of halfwords in an instruction is specified by the first two bits of the op code as follows:

Format	Op Code Positions 0 and 1	Instruction Length in Halfwords
RR	00	1
RX	01	2
RS and SI	10	2
SS	11	3

Because the op code of the next instruction to be executed is always in R, its format (positions 0 and 1) can be predecoded to determine the number of halfwords that compose that instruction and thus indicate which of the four Q-register halfwords contains the *next* sequential instruction op code. This predecoding occurs at end-op time of each instruction; the result (Q halfword number) is set into IC(21,22), which in turn selects a subsequent I-Fetch ROS word that specifies the next op-code Q-halfword to be transferred to R. The IC(21,22) values associated with each Q-register halfword are illustrated in Figure 1-28.

R(8–11) or R(12–15) is sent to LS address register to prefetch an operand for RR format instructions during end op of the preceding instruction. This transfer can be done

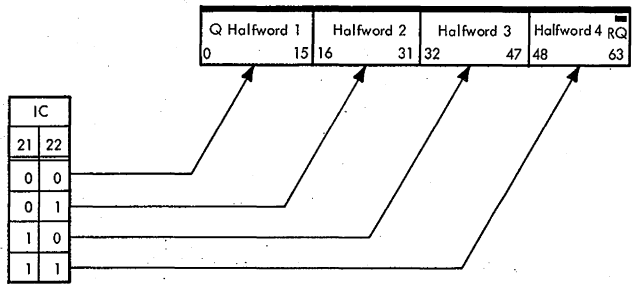


Figure 1-28. Q-Register Halfword Outgating per IC(21,22)

from R rather than from Q because RR instructions are completely contained in R; eight additional paths from Q to LAL are, therefore, not needed.

E-Register

The E-register (Diagram 3-2) is a halfword (16 bits plus two parity bits) trigger register that contains the first halfword (op-code halfword) of the instruction being executed. Portions of the op-code halfword in E are transferred to LAL, the op-code decoder, the parallel adder, the E-register incrementer, and, if the Direct Control feature is installed, an external device. The contents of the E-register are parity-checked.

Instruction Counter

- IC is divided into two sections: IC(0–20) and IC(21–23).
- IC(0–20) addresses a doubleword from main storage.
- IC(21) specifies left or right word within accessed doubleword: IC(21) = 1, select right word; IC(21) = 0, select left word.
- IC(22) selects left or right halfword from selected word: IC(22) = 1, select right halfword; IC(22) = 0, select left halfword.
- IC(21,22) specifies Q-register halfword that contains op code of next instruction to be executed.
- During VFL operations, IC(21–23) specifies addressed byte within doubleword addressed by IC(0–20).

The Instruction Counter (IC), Diagram 3-2, is a 24-bit trigger register used primarily for accessing the next sequential doubleword of instructions from main storage (excluding those specified by branch instructions, which are handled by the D-register). Source operand data is also addressed by the IC during VFL instructions.

The IC is divided into two logical sections: IC(0–20) and IC(21–23). These sections function in the following manner. The main storage area used with the CPU is addressable on a byte (eight-bit) basis, each address placed into the IC referring to a particular byte. However, because the Q-register is of doubleword (64-bit) length, instructions

are accessed from main storage in doubleword (eight-byte) groups. The address of the first byte of each doubleword is all that is required in accessing these doublewords from main storage, and this address is obtained from positions IC(0–20), regardless of the complete address. [Any address represented only by IC(0–20) is a multiple of 8 and lies on a doubleword integral boundary.]

Following the accessing of each doubleword from main storage, IC(0–20) is incremented by 8 (via the parallel adder) to develop the next sequential doubleword address in main storage (eight byte addresses ahead of the doubleword previously accessed).

Once the doubleword addressed by IC(0–20) is read into the CPU, the remaining portion of that complete address [IC(21–23)] selects either instruction halfwords or data bytes from the doubleword. When instructions are addressed by the IC, IC(21,22) only is used to extract the op-code halfword of the addressed instruction in the Q-register; IC(21) selects the right or left word within the doubleword, and IC(22) then selects the right or left halfword from the specified word. (In both cases, a 1 specifies the right portion and a 0 the left portion.) IC(21,22) values of 00, 01, 10, and 11 correspond respectively to the four (1–4) Q-register halfword portions. Figure 1-29 illustrates the Q-register halfword selection for a specified main storage instruction address of 468 (1D4 hex or 111010100 binary).

Note: Because instructions are restricted to even-numbered storage locations, IC(23) must always contain a 0 during instruction addressing. Detection of a 1 in IC(23) during instruction addressing produces an exceptional condition followed by a program interruption.

At end op of each instruction, the format of the instruction just transferred from Q and its location in Q [per IC(21,22)] are examined to determine the location in Q of the op-code halfword of the next sequential instruction.

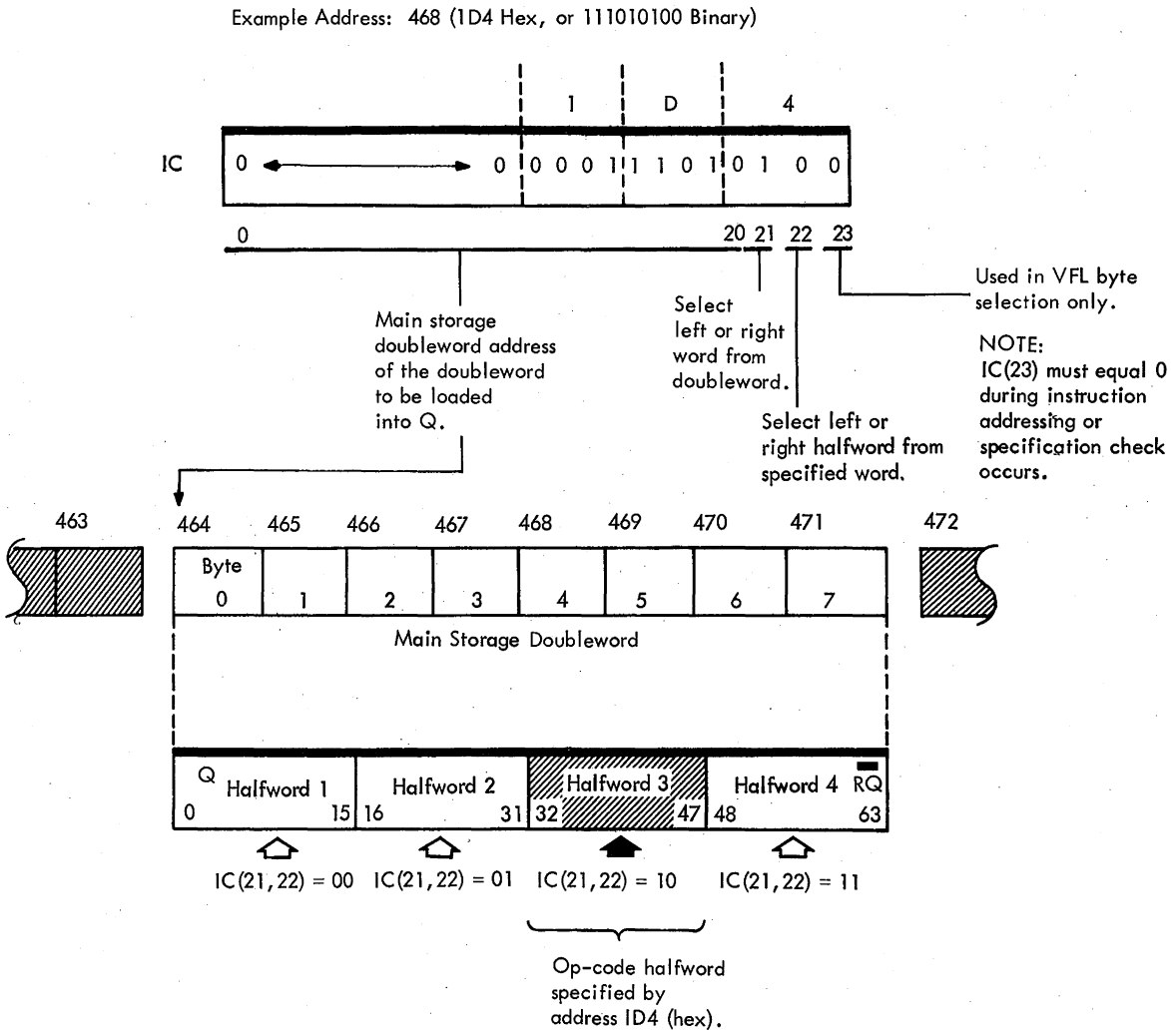


Figure 1-29. Instruction Addressing

Both format and Q-register location must be considered to avoid transferring the remaining non-op-code halfwords of a multi-halfword instruction (RX, RS, SI, or SS) to R.

When the IC is used for addressing source operands during VFL operations, doublewords containing the addressed byte(s) are referenced by IC(0–20) in the same manner as in instruction addressing. However, the accessed doubleword is read out to AB instead of to Q. IC(21–23) then specifies the addressed byte within this doubleword to be gated to the serial adder. [The initial IC(21–23) value is set into the AB counter, which is incremented or decremented, as required, to perform right-to-left or left-to-right processing of the data in AB.] Figure 1-30 illustrates the byte selection as determined by IC(21–23).

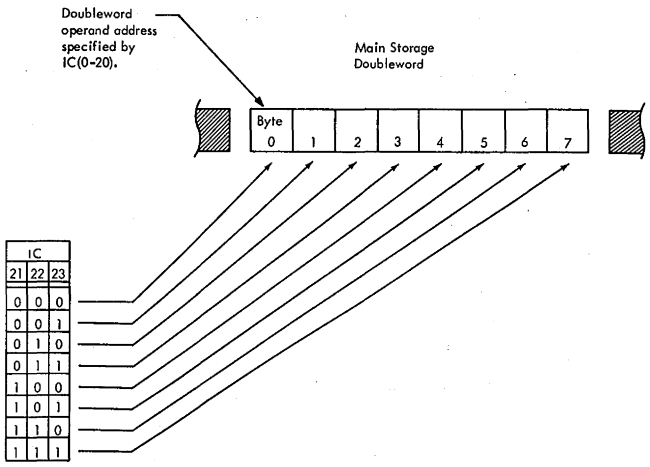


Figure 1-30. Operand Data Byte Selection per IC(21-23)

Note: During VFL operation, the instruction address contained in the IC is temporarily stored into the LS working register (LSWR).

Storage requests are generated to access the next doubleword whenever IC(21–23) indicates that all the information in the present doubleword (op-code halfword in Q-register or data bytes in AB) has been processed.

D-Register

The D-register (Diagram 3-2) is a 24-bit (plus three parity bits) trigger register that functions as a main storage address register during manual-control, branching, and certain arithmetic operations, and as a channel and unit address register during I/O instructions. When addressing main storage, D(0–20) references a doubleword; D(21,22) then extracts the desired instruction halfword and D(23) extracts the desired byte, depending on the operation.

For RS instructions, the I-Fetch routine adds the base and displacement values and places the result into D. Normally, this result is the effective second operand address. For shift instructions, however, this total specifies the number of bit positions to be shifted.

In the Stopped state, D contains the main storage address of the next instruction to be executed (address of instruction in R). (This address is generated and placed into D by the stop-loop microprogram that is in process whenever the CPU enters the Stopped state.) The stop-loop routine subtracts 8 or 16 (decimal) from the updated IC address and places the result into D. In this case, D(0–20) indicates the doubleword address of the instructions in Q, and D(21,22) specifies the location of the op-code halfword within that doubleword.

Instruction Path

- Instructions are fetched into Q from main storage four halfwords at a time.
- R contains first halfword of instruction to be executed next.
- E contains first halfword of instruction being executed.
- IC specifies location in main storage from which next instructions will be fetched and also instruction in Q to be executed next:
  IC(0–20) addresses main storage.
  IC(21,22) indicates which op-code halfword in Q has been transferred to R and is to be executed next.
  IC(23) must be 0 when addressing instructions.

The basic path for instructions entering the CPU is illustrated in Figure 1-31. The first register in the instruction path is the four-halfword instruction buffer called the Q-register. For each access, four instruction halfwords are fetched from a doubleword location in main storage (addressed by the IC) and loaded into the Q-register from the SDBO. Because instructions can vary from one to three halfwords in length, as many as four complete instructions (RR format) or as few as 1-1/3 instructions (SS format) may reside in the Q-register.

Instructions in the Q-register are sequentially selected for processing by means of IC(21,22), which indicates the first halfword (the halfword containing the op code) of the instruction to be executed next. The op-code halfword thus selected is transferred to the R-register, where format predecoding takes place during the end-op cycle. If the instruction is of the single halfword RR-format, the R-register contains the entire instruction. In the case of a two- or three-halfword instruction (RX, RS, SI, or SS format), the R-register contains only the first halfword; the balance (second or second and third halfwords) is not transferred but remains in the Q-register. For this reason, each halfword field of the Q-register is equipped with appropriate transfer paths for processing of the B and D fields of the instruction.

The format of the upcoming instruction (in R) is established by examining R(0,1). This predecoding groups

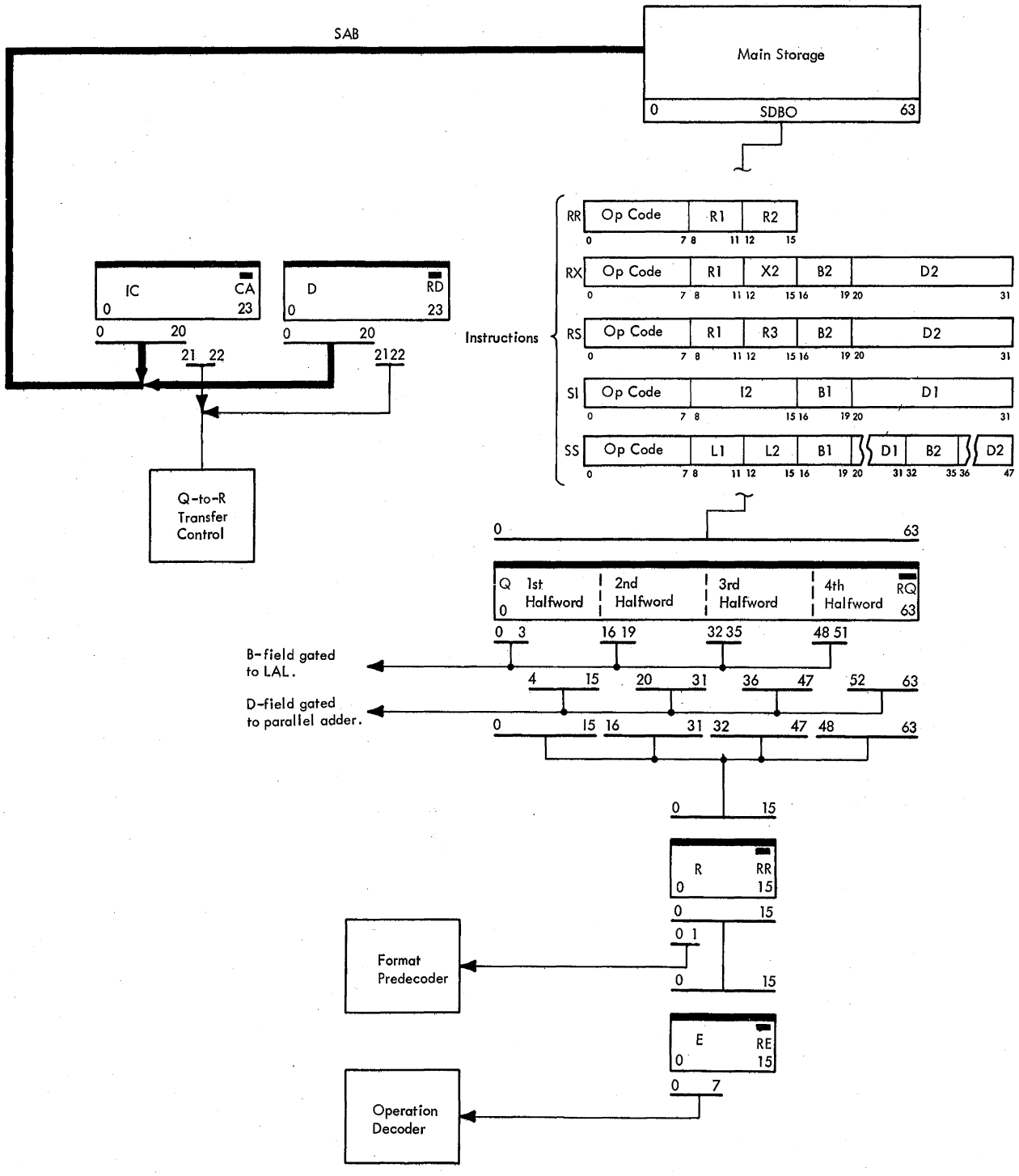


Figure 1-31. Basic Instruction Path

the instructions into four general categories (RR, RX, RS or SI, and SS) to allow loading of the appropriate data registers with operands and operand addresses. Thus operand prefetching is initiated before execution time.

The 16-bit E-register (Figure 1-31) contains the op-code halfword of the instruction presently being executed. This halfword remains in the E-register until the execution phase is completed, at which time it is replaced by the op-code halfword of the next instruction. During each execution phase, the instruction op code contained in E(0–7) is decoded and the specific operations necessary to execute the instruction are performed.

The functions performed during the end-op cycle and the I-Fetch sequence are implemented while the instruction halfwords are in the Q-, R-, and E-registers. The path and the movement of the op-code halfword between the registers for the five formats are shown in Figure 1-32. To illustrate, the following paragraphs trace an RR instruction through the Q-, R-, and E-registers (Figure 1-32, A). Note that an RR instruction is composed of only the op-code halfword; therefore, the complete instruction fits in the R- and E-registers. For the other formats, only the op-code halfword moves through the R- and E-registers; the other halfwords are not transferred from the Q-register. For the example, assume that:

1. All instructions have the RR format.
2. The instruction being executed is No. 4, the next instruction is No. 5, the following instruction is No. 6, and so on.
3. Instruction No. 5 is the one under consideration.

During the I-Fetch sequence of No. 4, instruction No. 5 is placed into the R-register. Here, during the end-op cycle of No. 4, the format of No. 5 is established, operand prefetching is initiated, No. 5 is transferred to the E-register, and the I-Fetch sequence for No. 5 is entered.

During the I-Fetch sequence for No. 5, prefetching of operands for No. 5 is completed, the execution routine for No. 5 is established, and No. 6 is transferred to the R-register. The CPU enters the execution phase for No. 5. During its end-op cycle, the condition code is set (if applicable) and any exceptional conditions and interruptions are detected. The remaining functions performed during the end-op cycle of No. 5 are devoted to initiating the I-Fetch sequence for No. 6.

Prefetching of Operands

- Operand prefetching starts before instruction execution.
- Depending on instruction format, operands are in LS or main storage:
  RR – both operands are in LS.
  RX, RS, SI – one operand is in LS, the other is in main storage.
  SS – both operands are in main storage.
- Address computation for main storage operands always starts first.

To increase the speed of instruction processing, the operands and operand addresses specified in the upcoming instruction are assembled into appropriate registers. For RR instructions, the operands are obtained directly from the LS. For instructions specifying operand addresses in main storage, address calculations take place, and the D-register prefetches an operand from main storage.

The major registers employed for operand prefetching are shown in Figure 1-33. Prefetching of operands begins when the op-code halfword of the instruction is in R and is completed after this halfword has been transferred to E. R(0,1) establishes the instruction format and, consequently, the type of operand fetch that must be performed.

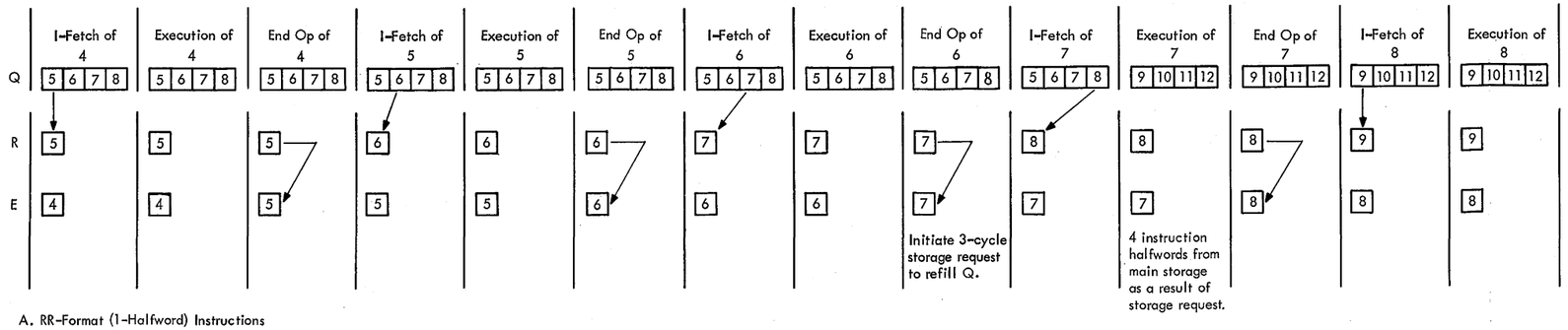
For one-halfword instructions (RR format), R contains the entire instruction. The first operand is fetched by transferring R(8–11) to LAL. After the first operand is retrieved from the LS, it is usually placed into A, B, and D. The second operand is usually fetched after the instruction is transferred to E by transferring E(12–15) to LAL. When the second operand is accessed, it is normally placed into S and T.

For two-halfword instructions (RX, RS, and SI formats), the first halfword is transferred to R while the second halfword is processed directly from Q. Address calculation for the operand in main storage is performed first so that this operand may be requested as soon as possible. This calculation is accomplished by transferring the appropriate B-field from Q to LAL. If the B-field is not zero, the contents of the LS register specified by the B-field are then routed to the parallel adder, where they are added to the D-field (transferred directly from Q). The sum constitutes the operand address specified by RS and SI instructions. This address is transferred to D, from which a storage request for the operand is made.

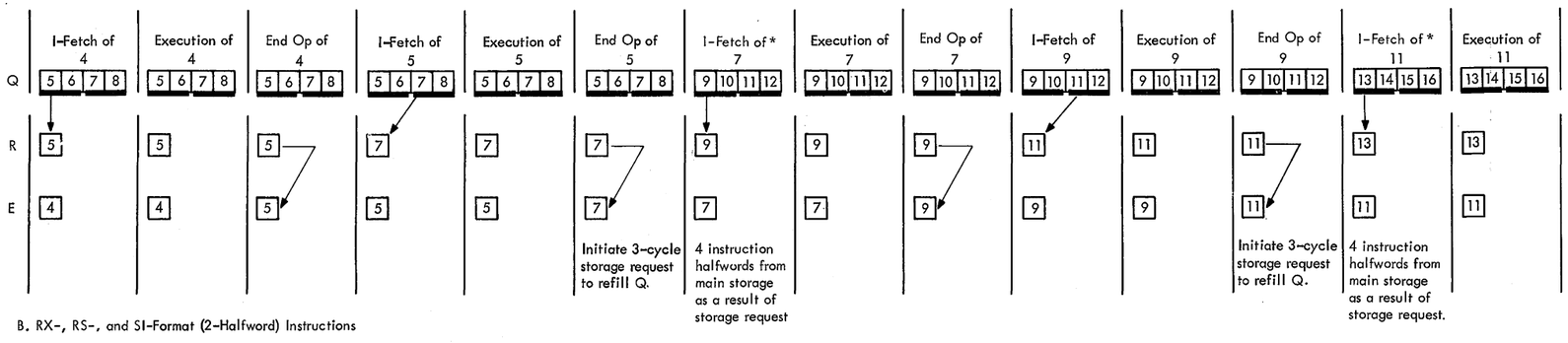
For indexed RX instructions, that is, when the X2 field is not zero, an additional step is required to derive the operand address. Consequently, the partial sum (LS contents per B-field, plus D-field) is temporarily stored into B. The LS is then addressed by the X2 field of the instruction. At this time, the instruction op-code halfword is in E, with E(12–15) containing the X2 field. The contents of the LS register accessed by the X2 field are then summed with the contents of B in the parallel adder to obtain the operand address. This address is transferred to D, and a storage request for the operand is initiated.

The operand setup for two-halfword instructions is completed by fetching the first operand from the LS (not used by SI instructions). This action is performed by transferring E(8–11) to LAL. The first operand is usually loaded into A and B. S and T are usually loaded with the second operand when it arrives on the SDBO during the execution phase.

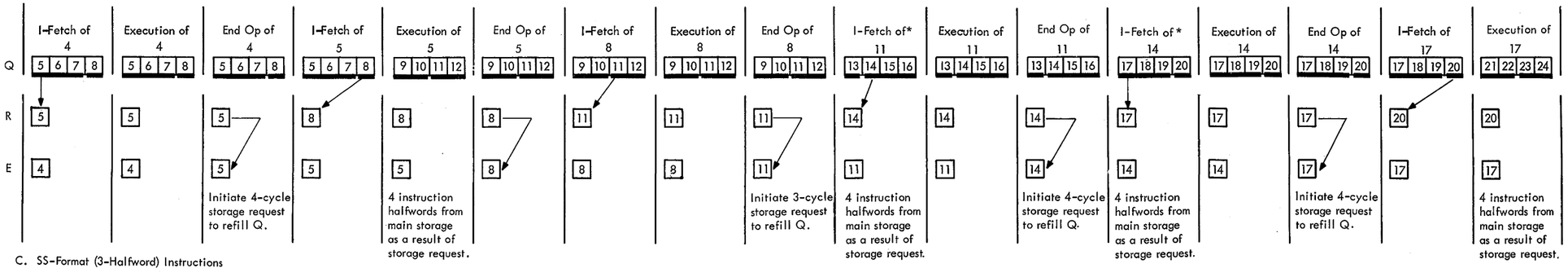
For three-halfword instructions (SS format), the first halfword is transferred to R and the remaining two halfwords are processed directly from Q. The main storage addresses for the first and second operands are calculated in a manner similar to that of two-halfword instructions. The first-operand address is computed first and loaded into D, and a storage request to prefetch the operand is initiated. The second-operand address is then computed while the contents of the IC are transferred (via the parallel adder) to the LSWR. When the second-operand address is computed, the address is loaded into the IC, from which a storage request for the operand is later made. Upon execution of an SS instruction, the instruction address is restored to the IC so that it again selects the next instruction.



A. RR-Format (1-Halfword) Instructions



B. RX-, RS-, and SI-Format (2-Halfword) Instructions



C. SS-Format (3-Halfword) Instructions

Figure 1-32. Path Through Q-, R-, and E-Registers of Op-Code Halfword

Note: Numbers indicate halfword sequence referenced to their location in Q.

For 2- and 3-halfword instructions, lines at bottom of Q group the 4 halfwords into instructions or portions of instructions.

Movement is referenced to associated ROS control word.

* Delay transferring to R until instruction halfwords arrive from main storage.

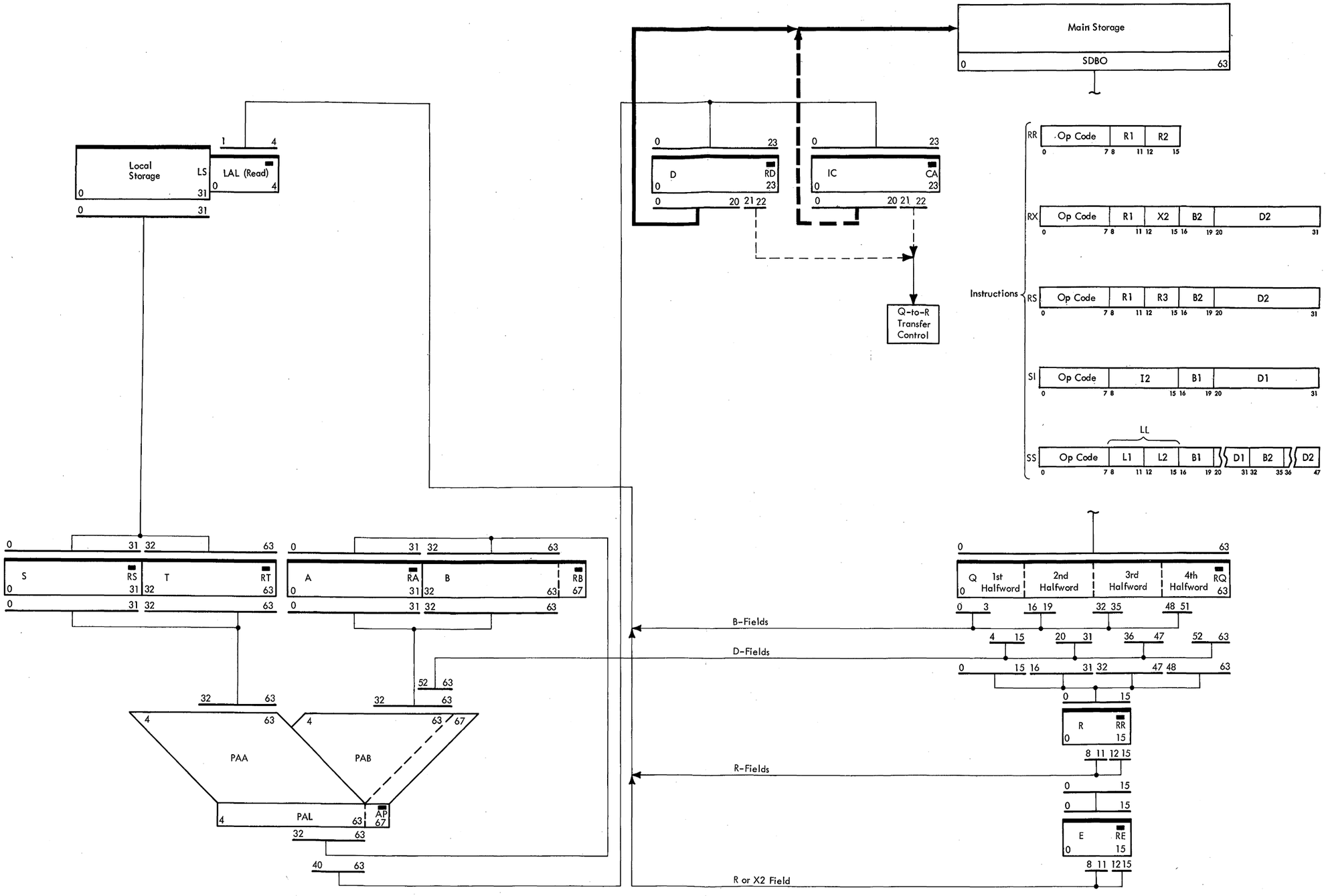


Figure 1-33. Basic Scheme for Operand Prefetching

Obtaining New Instructions from Main Storage

- Requests for new instructions are made before CPU exhausts instructions in Q.
- Usually, three- or four-cycle requests are made from IC.
- Settings of IC(21,22) and R(0,1) determine whether request is needed.
- IC is incremented by 8 after each request.
- For branch instructions, requests are made from D.
- If branch is unsuccessful, instructions accessed by request are ignored.
- If branch is successful, instructions are used and instruction address is transferred from D to IC.

Requests to refill Q with new instructions overlap most of the storage access time with processing of the remaining instructions in Q. The basic scheme used in requesting new instructions from main storage is shown in Figure 1-34.

During normal instruction sequencing, IC(21,22) is examined to establish the number of halfwords in Q that remain to be processed. These bits indicate the op-code halfword of the instruction that has been transferred to R and is to be executed next. Depending on the format of the upcoming instruction decoded from R(0,1), a request for new instructions may be initiated when 2, 3, or all 4 halfwords in Q remain to be processed. The time required to access new instructions from main storage is then used to process the remaining instruction(s) in Q. This access time is specified by the type of request generated by the CPU. Depending on its instruction status, the CPU can generate a three- or a four-cycle request. When a three-cycle request is issued, three CPU cycles must elapse before instructions arrive from main storage. Thus, new instructions are gated into the CPU on the fourth cycle following the request. Similarly, when a four-cycle request is generated, new instructions are gated into the CPU on the fifth cycle following the request. The difference in access times for a three- and four-cycle request is illustrated in Figure 1-34.

IC(0–20) normally specifies the address of new instructions to be accessed from main storage. When it is established that the CPU needs instructions, a request is made and IC(0–20) is transferred to the SAB. The IC is then incremented to address the next successive storage location from which subsequent instructions are to be fetched. (Because the BCU dictates that the address of each successive main storage location must be valid in the IC for at least two cycles, updating of the instruction address is not initiated until two cycles following the request.) Depending on the format of the upcoming instruction, incrementing of the IC is controlled by the CPU hardware (for all formats except SS) or by the ROS microprogram (for SS format). In either case, the IC is incremented by transferring its contents to the parallel adder, where a 1 is added to IC(20) (equivalent to advancing the IC address by 8). The updated address is then routed back to the IC so that a new storage request may be initiated immediately upon detecting the need to refill Q.

A departure from the normal sequencing described above occurs when the instruction being fetched is a branch instruction. To anticipate branch instructions, R (which always contains the first halfword of an upcoming instruction) is connected to a branch predecoder. Execution of a branch instruction may alter the main storage address from which new instructions are to be fetched. If the branch instruction is successful, the address specified by that branch becomes the new instruction address. If, however, the branch is not successful, the address specified by that branch must be ignored. Because it is assumed that branch instructions are successful (the only exception is the Branch on Condition instructions), a request for instructions is initiated as soon as the address specified by the branch is placed into D. Thus, a request is made and the contents of D are transferred to the SAB before establishing that the branch is indeed successful. If it is later found that the branch instruction is not successful, the instructions accessed by that branch are not transferred into Q and a new storage request is generated from the IC, if necessary. Otherwise, upon establishing a successful branch, the contents of D are transferred to the parallel adder, incremented by 8, and transferred to the IC. Normal sequencing is resumed by the IC until another branch instruction is encountered.

The Branch on Condition instructions must be treated differently from other branch instructions. For this reason, a separate detection circuit is provided to anticipate this branch. Whether a Branch on Condition instruction is successful depends on the condition code setting established by a previous instruction. Therefore, the outcome of this branch instruction is known at the time of the request. If the branch is unsuccessful, the request from D is inhibited and, if Q needs refilling, an IC request is generated instead.

Another instruction that falls in the branch category and requires unique treatment is the Execute instruction. This instruction designates a single instruction (subject instruction) to be inserted into the instruction sequence. Briefly, the Execute instruction initiates the following I-Fetch actions:

1. During I-Fetch of an Execute instruction (RX format), IC(21,22) is advanced to indicate the first halfword of the next instruction; that is, the instruction immediately following the Execute instruction in Q.
2. The address of the subject instruction specified by the Execute instruction is computed and placed into D. A storage request from D is then made.
3. When the four instruction halfwords accessed by this request are gated to Q, the subject instruction is selected from Q by examining D(21,22).

4. Upon executing the subject instruction, and if the subject instruction is not a successful branch, a storage request for the instructions previously contained in Q is made from the IC. This request is performed by the program store compare exceptional condition. (If the subject instruction is a successful branch, the request for new instructions is made from D and normal processing is resumed.)
5. When the instructions previously contained in Q are refetched, IC(21,22) is examined to select the instruction following the Execute instruction, and normal processing is resumed.

CPU Interruption and Exceptional Condition Recovery

I-Fetch recognizes the five classes of interruptions: external, supervisor call, program, machine check, and I/O. CPU recovery from these interruptions requires additional processing time for storing the old PSW into main storage and fetching a new PSW and new instructions from main storage into the CPU.

In addition, I-Fetch also recognizes eight exceptional conditions; these conditions require special handling by the CPU. The exceptional conditions and the corresponding actions performed by the CPU are:

Exceptional Condition	CPU Action
Timer	When updating is required, the CPU fetches the timer value from main storage, decrements this value (amount of decrement depends on the line frequency), and stores it back into main storage.
CPU Store in Progress	Extra cycles are added to I-Fetch.
Manual Control Stop	An address is forced into ROSAR that places the CPU into the stop loop.
Manual Control Wait	An address is forced into ROSAR that places the CPU into the Wait state.
Manual Control Repeat	An address is forced into ROSAR that places the CPU into the repeat instruction loop.
Program Store Compare	The CPU obtains the address for the next instruction and refetches it from main storage.
Invalid Instruction Address Test	A program interruption is forced when the CPU tries to use an erroneously addressed instruction.
Q-Register Refill	One extra I-Fetch cycle is added, delaying execution of the next instruction by one CPU cycle.

When one or more interruptions or exceptional conditions occur, processing of the next instruction is delayed until all such conditions are cleared in the order of their priority.

Processing of interruptions and exceptional conditions is initiated during the first cycle of I-Fetch. This cycle issues an 'EXCEP' micro-order to establish if any interruptions or exceptional conditions have resulted from execution of the preceding instruction. If one or more interruptions or exceptional conditions did occur, the 'EXCEP' micro-order overrides the basic I-Fetch actions and transfers control to an appropriate microprogram for clearing the condition that is assigned the highest priority. After processing this condition, the remaining interruptions and exceptional conditions, if any, are handled in the order of their priority. When all interruptions and exceptional conditions have been processed, the microprogram resumes the I-Fetch sequence for the next instruction. The invalid instruction address test and conditions do not override the basic I-Fetch actions but instead augment them.

INSTRUCTION EXECUTION

The execution phase of instruction processing manipulates the data to execute the instruction prepared by I-Fetch. The following paragraphs introduce the functional units that are used primarily in the execution phase and discuss the seven classes of instructions: fixed-point, floating-point, decimal, logical, branching, status switching, and I/O.

Functional Units Used

The following discussion of functional units is based on Diagram 3-2, FEMDM. For a detailed discussion, refer to Chapter 2.

AB Register

The AB register is a 64-bit (plus eight parity bits) trigger register that functions as a working register and also as a buffer for doubleword operands received from main storage.

The AB register is logically divided into two 32-bit (plus four parity bits) registers, A and B. A four-position extension, B(64–67), provides for retaining low-order significance during certain arithmetic and shifting operations.

Byte gating into and out of A and B facilitates their use with the serial adder for VFL operations. A three-position counter, the AB byte counter (ABC), controls the outgating selection of the eight bytes in the AB. AB information is also processed in the parallel adder. Both A and B outgating controls are capable of shifting data left two positions en route to the parallel adder. (Combined shifting capabilities of ST, AB, and the parallel adder thus enable any amount of shift in any direction.)

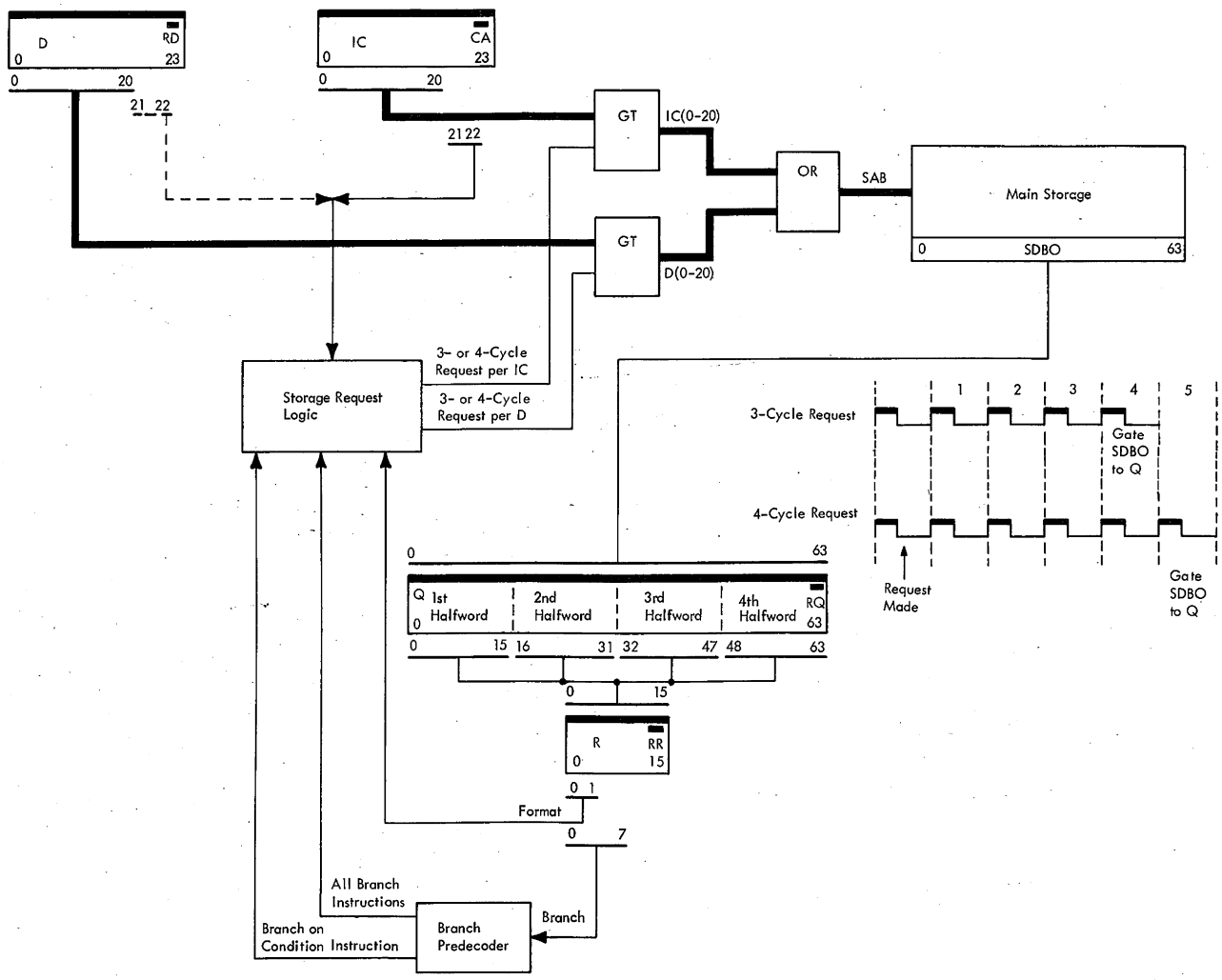


Figure 1-34. Q-Register Refill Addressing Scheme

ST Register

The ST register is a 64-bit (plus eight parity bits) trigger register that functions as an operand buffer between main storage, LS, and the CPU, and also as a working register for arithmetic and logical operations.

The ST register is logically divided into two 32-bit (plus four parity bits) registers, S and T, which serve as working registers for all operations. They also serve as a final assembly area for resultant data to be entered into either LS (T only) or main storage (S and T). Byte gating of ST inputs and outputs facilitates their use with the serial adder for VFL operations. A three-position ST byte counter (STC) and incrementer control the gating of the eight ST bytes.

ST information can also be processed in the parallel adder. T-data can be sent to the parallel adder in either true or complement form, and with a left 1 shift for certain operations. (Scan operations also utilize ST inputs from the parallel adder.) Multiply logic extracts data, in byte lengths, from S via the multiplier bus, and PSW information is sampled from ST(0–39) via the PSW bus.

AB and ST Byte Counters

For operations involving the serial adder, it must be possible to extract bytes from doublewords contained in AB and ST and to assemble bytes in ST for subsequent storage. These capabilities are provided by two byte counters: the ABC for controlling AB byte transfer and the STC for controlling ST byte input and output.

Mark Triggers

Eight mark triggers are contained within the CPU. During store-data operations, these triggers indicate to main storage which bytes of doubleword data placed on the SDBI are to enter storage. The mark triggers thus serve to store CPU data into main storage on a byte (eight-bit plus parity) basis.

An active mark bus line specifies the corresponding byte for storage entry: mark trigger 0 specifies byte 0, or SDBI(0–7); mark trigger 1 specifies byte 1, or SDBI (8–15); and so on, through mark trigger 7, which specifies byte 7 or SDBI(56–63). Any mark trigger not set causes its corresponding byte of original storage data to be regenerated into the addressed location. Complete absence of mark signals on the mark bus occurs only on a fetch operation.

F-Register

The F-register is a one-byte (eight-bit plus parity) trigger register used in certain arithmetic, logical, and data-transfer operations. Functions such as developing quotients, saving floating-point characteristics, converting routines, and processing storage-protection keys are performed in F, and, during direct-control read operations, F serves as a data entry buffer for the external device.

F-register parity circuits generate correct parity for all information received by the register.

G-Register

The G-register is an eight-bit trigger register used for buffering a byte of data between the CPU and an external device during direct-control write operations. G contains no parity bit.

Serial Adder

The serial adder processes information from ST, AB, and F on a byte basis, and is capable of performing binary and decimal arithmetic in addition to logical AND, OR, and Exclusive-OR operations. Other miscellaneous functions performed by the serial adder logic include sign insertion and correction, digit insertion, invalid character detection, zone correction (EBCDIC and USASCII-8), zero and nonzero recognition, and parity adjustment. Parity-predict logic and carry lookahead logic are employed to improve operational speeds, with checking performed on both a half-sum and full-sum basis.

Arithmetic Functions. Highlights:

- Operates on binary or decimal data.
- Processes decimal bytes as two four-bit groups.
- Each four-bit group represents one BCD digit.
- Employs excess-6 arithmetic when processing decimal data.
- Decimal arithmetic results are produced in BCD form.
- Multiply/divide results are assembled in serial adder.

Serial adder logic is capable of performing both binary and decimal arithmetic operations. For binary operations, the adder functions as an eight-bit (plus parity) binary adder, processing bytes from ST and either AB or F and developing eight-bit (plus parity) results. High-order carries resulting from arithmetic operations are stored in Status Trigger (STAT) H for use in processing the next two bytes of that operation.

For decimal operations, each byte of data from ST, AB, or F is treated as two individual four-bit groups, which are then processed with excess-6 arithmetic. This feature provides a programming advantage by enabling the adder to accept data in binary-coded decimal (BCD) form (packed digits) and to produce results which are also in decimal format.

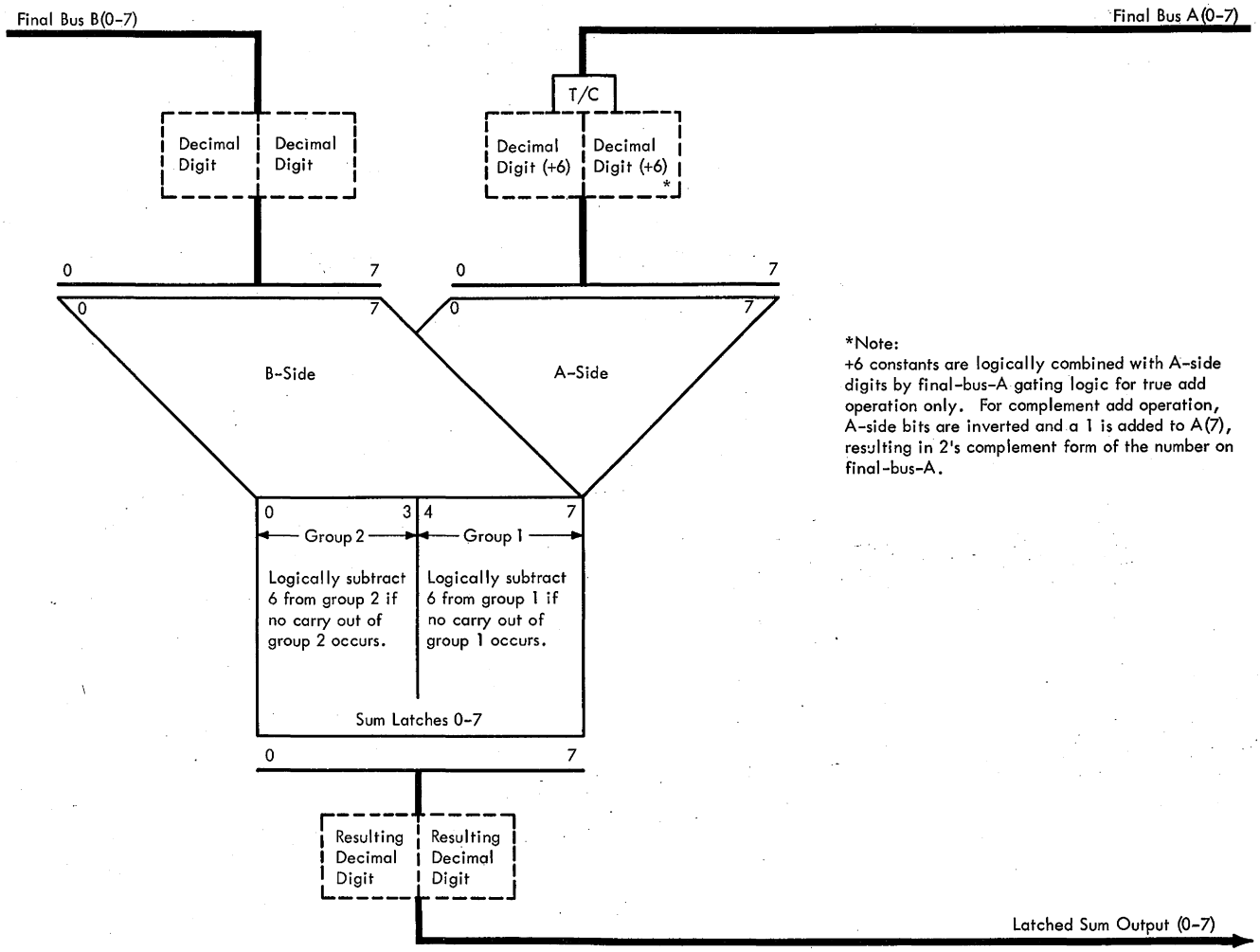
Excess-6 arithmetic involves adding 6 (under ROS control) to incoming first operands. This is necessary to preserve the decimal value in a four-bit binary character. Each binary character (four bits) has a maximum decimal value of 15 (1111 binary), which is 6 more than the maximum valid decimal character of 9 (1001 binary). When the second operand is added and the total exceeds 15 (1111 binary), the carry is a decimal value carry and leaves a correct decimal value in the four binary bits. If no carry occurs after the addition, the character is an erroneous decimal value (it is too large by 6) and the excess 6 is subtracted. The decimal values of the binary character under no-carry conditions are 6–15 (actual value of 0–9 after correction).

The following examples show excess-6 addition. Note that correction occurs when there is no carry irrespective of the decimal validity of the sum.

	1 + 1	3 + 5	6 + 9
Operand 1 (SBA)	0001	0011	0110
Excess-6	0110	0110	0110
SAA	0111	1001	1100
Operand 2 (SBB)	0001	0101	1001
Sum	1000	1110	←0101
Correction	0110	0110	----
Result	0010	1000	←0101
	(2)	(8)	(15)

In the first case (1 + 1) the sum is a valid decimal character (8), but the absence of a carry indicates an erroneous result calling for subtracting the excess 6. The second case (3 + 5) is similar except that the sum is not a decimal character. In the third case (6 + 9) there is a carry into the decimal tens position and the sum (5) does not need correction.

Figure 1-35 shows the excess-6 data paths through the serial adder and illustrates the adder operation by means of the decimal example: 46 + 28 = 74. Note that +6 constants



Decimal Arithmetic Example: 46 + 28 = 74

A-side operand byte (46_{10})	0100	0110
Convert A-side digits to excess-6 (Logically add +6 to both groups at final bus.)	0110	0110
A-side digits in excess-6 value	1010	1100
B-side operand byte (28_{10})	0010	1000
	Group 2	Group 1
A-side adder entry	1010	1100
B-side adder entry	0010	1000
Logical sum	1101	0100
Decimal correction (Logically subtract 6 from group 2.)	0110	
Decimal result (74_{10})	0111	0100

Decimal Arithmetic Example: 46 - 28 = 18

A-side operand byte (28_{10})	0010	1000
Complement	1101	0111
+1 to A(7)		1
2's complement of A-side digit	1101	1000
B-side operand (46_{10})	0100	0110
	Group 2	Group 1
A-side adder entry	1101	1000
B-side adder entry	0100	0110
Logical sum	0001	1110
Decimal correction (Logically subtract 6 from group 1)		0110
Decimal result (18_{10})	0001	1000

Figure 1-35. Decimal Format Serial-Adder Data Flow

are logically added to the A-side digits that are in packed format before entry into the adder (by final-bus-A-gating logic), and that subtraction of the +6 constants (decimal-correct) is performed on each four-bit group in which a group carry (carry from high-order position of group) does *not* occur as a result of the arithmetic operation.

When a ROS micro-order calls for complement add, the data entering SAA is converted to 2's-complement form. This conversion is accomplished by complementing the bit configuration on the A-side entry and adding a hot carry to the input of SAL(7). For complement add decimal operation, +6 constants are not combined with SBA inputs. When complementing BCD, the excess-6 is effectively added because the resultant complement is a character based on 16 rather than 10. To illustrate, the 10's-complement of 7 is 3, but the 2's-complement of 7 (0111) is 9 (1001) or 6 more than the 10's-complement. The addition then occurs as in true +6 add, and the absence of a carry likewise forces decimal correction of the sum. A carry out of the high-order adder position [serial adder bit carry (0)] sets STAT H for use in processing the next data byte of that operation.

Figure 1-35 also shows the serial adder operation for a complement add example: 46 − 28 = 18. Note that +6 constants are not added to the A-side digits, but decimal correction is performed in the same manner as for a true add operation.

Logical Functions. The serial adder also performs logical AND, OR, and Exclusive-OR functions. To implement the logical functions, each bit position of the serial adder is, in effect, a separate unit. The logical functions are defined as follows:

1. AND. If *both* operand bits are 1's, the resulting bit is a 1; otherwise, the result is a 0. (Carries between bits are suppressed.)
2. OR. If *either* operand bit is a 1, the resulting bit is a 1; otherwise, the result is a 0. (Carries between bits are suppressed.)
3. Exclusive-OR. If *one and only one* of the operand bits is a 1, the resulting bit is a 1; otherwise, the result is a 0. (Carries between bits are suppressed.)

If the conditions for the corresponding function is met, the associated serial adder latch is set.

Parallel Adder

- 60-bit (plus parity) full-binary adder.
- Inputs are from S, T, D, A, B, Q, IC, E, and F.
- Inputs from T and D are 2's complemented for subtract and compare operations and address updating.
- Output data can be shifted left 4 or right 4 into the parallel adder latches; parity is adjusted accordingly.
- Adder employs carry-lookahead and parity-predict logic.
- Adder includes half-sum and full-sum error checking.

The parallel adder is a 60-bit (plus parity) full-binary arithmetic unit. In addition to arithmetic functions, the parallel adder performs certain logical operations (e.g., convert routines) and is involved in most intra-CPU data transfers. Correct parity (odd) is generated with all adder output data. Immediate left 4 and right 4 shifting capabilities are available at the adder output, with parity adjusted accordingly. Error-checking facilities within the adder provide for validity checking of both incoming operands and full-sum results.

The parallel adder has true-complement gating controls on adder entries from T and D. For subtract and compare instructions, operands from T are 2's complemented and presented to the A-side of the adder. (The binary bits are inverted and a hot-carry is added to position 63.) From this point, add and subtract operations are the same. Complement entries from D are used for address compare and address update operations and for floating-point operations.

The parallel adder has full-binary capabilities (half-adder and full-sum functions), with immediate left-4/right-4 shift logic included on its output to facilitate data shifting without the need of an additional machine cycle. Figure 1-36 illustrates the logical functions of the parallel adder. Note the four-position adder extension, PAB(64–67); it serves to retain low-order significance during certain right-shift operations.

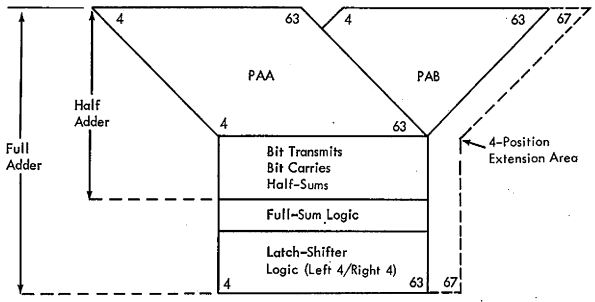


Figure 1-36. Parallel Adder Logical Functions

Half-adder functions supply information concerning incoming operands for use in both checking the validity of the operands and producing carry information for full-sum development. Full-sum logic combines half-adder outputs with carry information to produce the final or full-sum result. Parity information is also developed by logically combining half-adder and carry functions and is normally supplied on a byte basis, although certain adder areas (bits 4–8 and 64–67) require half-byte or four-bit group parity.

Carry-lookahead and parity-predict facilities are employed within the parallel adder. These features provide for the immediate development of full-sum results (and parity), without the need for additional cycles in which to incorporate carry information and generate parity. The lookahead and predict circuitry is implemented through logic in which the 60 bit positions are arranged in four-bit

groups, and these groups divided into four sections. Figure 1-37 illustrates the logical grouping of the 60 bit positions.

All adder results are checked using half-sum and full-sum error-checking logic. Half-sum checking determines the validity of incoming operands by comparing the odd/even bit count with the assigned parity. Full-sum checking involves comparing the full-sum resultant bit count with the independently generated full-sum parity; an inconsistency in either causes a full-sum error.

Local Storage

A high-speed transistor storage area, local storage (LS), is located within the CPU to reduce the number of main storage references required by the CPU during each operation. The LS consists of 25 registers for use in storing address information, fixed-point, logical, and floating-point operands, and the IC contents (IC contents stored in LS working register, LSWR, only). Local storage data is available to the CPU at 200-ns intervals. In addition to reducing the main storage reference, this access time increases operational speeds within the CPU.

The 25 LS registers are grouped as follows: 16 (0–15) general-purpose registers (GPR's), 8 (16–23) floating-point registers (FPR's), and 1 (24) working register called the LSWR. Each register contains 32 data (plus 4 parity) positions, and is directly addressable by the R1, R2, R3, B1, B2, and X2 fields of the instructions, with the exception of register 24. Register 24 (LSWR) is not available to the operational program. It is reserved for use by ROS microprograms in manipulating information during execution of certain instructions. (Such applications include temporary storage for the contents of the IC when the IC is to be used as an operand address register, and temporary storage for floating-point second operands while prenormalizing the first operand.)

The eight FPR's (16–23) function as four double-length (64-bit) registers. Each double-length register consists of two single-length (32-bit) registers coupled as follows: 16 and 17, 18 and 19, 20 and 21, and 22 and 23. Only the leftmost 32 bit positions are used in short-operand floating-point instructions, with all 64 positions participating in long-operand floating-point instructions. For either short or long operands, only the leftmost (even-numbered) registers must be addressed.

Data transferred from LS is checked only upon being processed in the adders or when entered into main storage.

Status Triggers

The CPU contains eight commonly available status triggers (STAT's) to record information that may be significant in the execution of present instruction operations. These eight triggers are designated as STAT's A–H, and retain such information as invalid digit-detection, overflow and carry conditions, and negatively signed operands. The STAT's are reset at each I-Fetch.

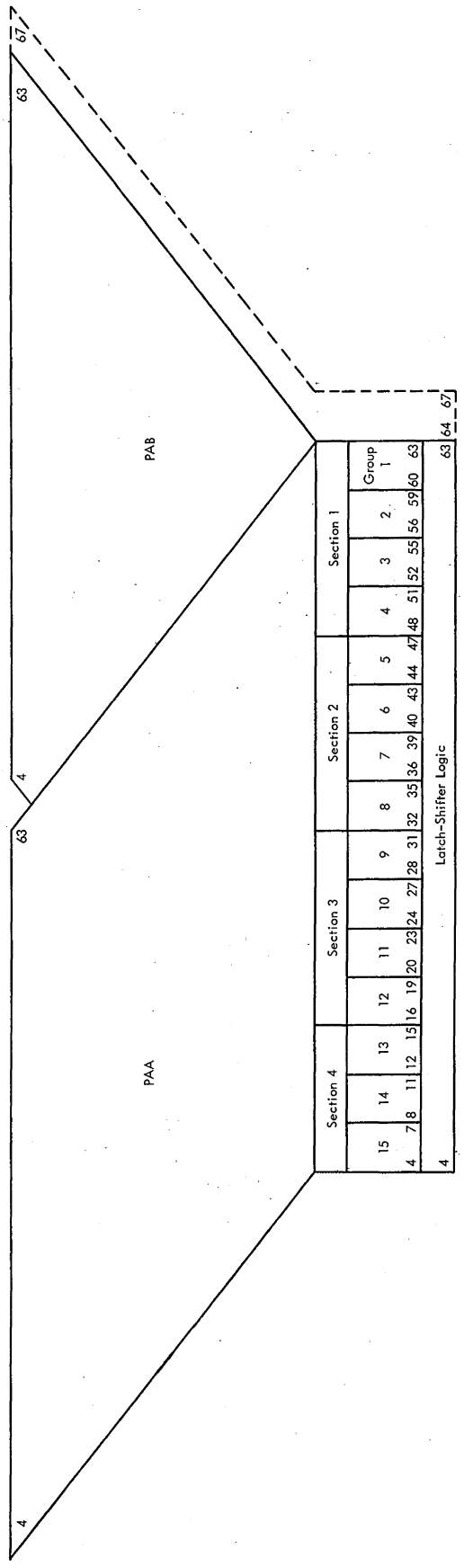


Figure 1-37. Parallel Adder Group/Section Breakdown

Certain STAT's serve multiple functions and are capable of receiving several types of information for use with different instructions. The outputs of these multiple-use triggers are distributed, via line-sense amplifiers, to the CPU areas requiring this information.

Scan operations test STAT's; during scan in, the eight STAT's are set to the state of T(38) and T(54–60).

All STAT's are reset by either 'system reset' or 'I-Fetch reset' signals, with certain STAT's containing additional individual resets.

In general, all STAT's are reset and set during clock time of the basic machine cycle. However, certain clock signals that control these triggers are delayed 180 ns. This delayed operation is necessary to prevent timing problems that could arise if the detected conditions occurred too late to set a STAT with normal clock signals. All clock signals that control the STAT's are inhibited during scan-in operations by an 'FLT inhibit clock' signal.

Fixed-Point Instructions

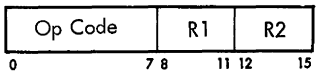
The fixed-point instruction set performs binary arithmetic on operands serving as data, addresses, index quantities, and counts. Instructions are provided for loading, adding, subtracting, comparing, multiplying, dividing, shifting, storing, and converting from binary to decimal and from decimal to binary. Table 1-8 lists the fixed-point instructions.

For a discussion of number representation, data formats, and operand addressing, refer to Section 2 of this chapter.

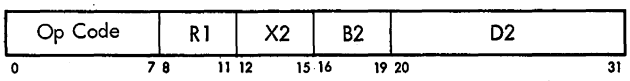
Instruction Formats

The fixed-point instruction set uses three instruction formats:

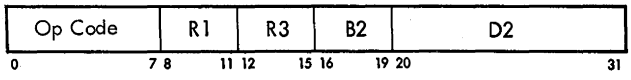
RR



RX



RS



In the RR format, R1 specifies the address of the GPR containing the first operand and R2 specifies the address of the GPR containing the second operand. Both the first and second operands may be specified by the same GPR.

In the RX format, R1 specifies the address of the GPR containing the first operand. The contents of the GPR specified by the X2 and B2 fields are added to the contents of the D2 field to form an address designating the main storage location of the second operand.

In the RS format, R1 specifies the address of the GPR containing the first operand. The contents of the GPR specified by the B2 field are added to the contents of the D2 field to form an address. This address designates the main storage location of the second operand for Load Multiple and Store Multiple instructions. In shift operations, the low-order six bits of the address specify the number of bit positions to be shifted. The R3 field specifies the address of GPR for Load Multiple and Store Multiple instruction and is ignored in the shift operations.

Data Flow

The data flow path for fixed-point operations is shown in Diagram 3-3, FEMDM. The functional units used to perform the major functions for fixed-point operations are:

1. ST. Holds the second operand and assembles data before it is sent to LS or main storage. (T is the only register that can transfer data to the LS.)
2. AB. Holds the first operand and assembles data during an operation.
3. F. Assembles product and quotient bits during multiply and divide operations; holds the binary bits to be converted during convert operations.
4. E. Controls product and quotient derivation; also contains instruction op code and number of shifts when performing shift instructions.
5. Parallel adder. Manipulates the operands to obtain the desired result. Is also the central point in the data path between ST and AB.
6. Serial adder. Calculates product and quotient bytes. Is also the central point in the data path between F and AB and between F and ST.
7. STC. Controls selection of data from and placement of data into ST, primarily during multiply, divide, and convert operations.
8. D. Addresses second operand located in main storage.

Program Interruptions

Six program interruptions can occur during execution of fixed-point instructions. Of the six, only fixed-point overflow can be masked off; the others are unconditionally taken. If the fixed-point overflow mask bit [PSW(36)] is a 0, the fixed-point overflow interruption is ignored; if a 1, it is taken.

The six interruptions and their causes are:

1. Protection. The storage key of a main storage location does not match the storage protection key in the PSW. The instruction is suppressed for a store violation, unless it is the Store Multiple instruction, which is terminated. For a fetch violation, the instruction is terminated.

Table 1-8. Fixed-Point Instructions

Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Add	A	5A	RX	R1 D2(X2, B2)	Algebraically add 2nd opr (in stg) to 1st opr (in GPR per R1) & place result into 1st opr location. D(21) determines which word of doubleword from stg is 2nd opr: if 1, right word; if 0, left word.	Prot (F) Adr Spec Fix-Pt Ovflo	0 : Sum = 0 1 : Sum < 0 2 : Sum > 0 3 : Overflow
Add	AR	1A	RR	R1 R2	Algebraically add 2nd opr (in GPR per R2) to 1st opr (in GPR per R1) & place result into 1st opr location.	Fix-Pt Ovflo	0 : Sum = 0 1 : Sum < 0 2 : Sum > 0 3 : Overflow
Add Halfword	AH	4A	RX	R1 D2(X2, B2)	Algebraically add halfword 2nd opr (in stg) to 1st opr (in GPR per R1) & place result into 1st opr location. 1. D(21) determines which word of doubleword from stg contains halfword 2nd opr: If 1, right word; if 0, left word. 2. D(22) determines which half of word is halfword 2nd opr: If 1, right half; if 0, left half. 3. Halfword 2nd opr is expanded to full word before addition by propagating sign bit through 16 high-order bits.	Prot (F) Adr Spec Fix-Pt Ovflo	0 : Sum = 0 1 : Sum < 0 2 : Sum > 0 3 : Overflow
Add Logical	AL	5E	RX	R1 D2(X2, B2)	Algebraically add 2nd opr (in stg) to 1st opr (in GPR per R1) & place result into 1st opr location. 1. D(21) determines which word of doubleword from stg is 2nd opr: if 1, right word; if 0, left word. 2. Sign bit of result is treated as high-order integer & is tested for carry to determine CC.	Prot (F) Adr Spec	0 : Sum = 0 (no carry) 1 : Sum ≠ 0 (no carry) 2 : Sum = 0 (carry) 3 : Sum ≠ 0 (carry)
Add Logical	ALR	1E	RR	R1 R2	Algebraically add 2nd opr (in GPR per R2) to 1st opr (in GPR per R1) & place result into 1st opr location. Sign bit of result is treated as high-order integer & is tested for carry to determine CC.	None	0 : Sum = 0 (no carry) 1 : Sum ≠ 0 (no carry) 2 : Sum = 0 (carry) 3 : Sum ≠ 0 (carry)
Compare	C	59	RX	R1 D2(X2, B2)	Algebraically compare 1st opr (in GPR per R1) with 2nd opr (in stg) & set CC according to result. D(21) determines which word of doubleword from stg is 2nd opr: if 1, right word; if 0, left word.	Prot (F) Adr Spec	0 : Opr 1 = Opr 2 1 : Opr 1 < Opr 2 2 : Opr 1 > Opr 2
Compare	CR	19	RR	R1 R2	Algebraically compare 1st opr (in GPR per R1) with 2nd opr (in GPR per R2) & set CC according to result.	None	0 : Opr 1 = Opr 2 1 : Opr 1 < Opr 2 2 : Opr 1 > Opr 2
Compare Halfword	CH	49	RX	R1 D2(X2, B2)	Algebraically compare 1st opr (in GPR per R1) with halfword 2nd opr (in stg) & set CC according to result. 1. D(21) determines which word of doubleword from stg contains halfword 2nd opr: if 1, right word; if 0, left word. 2. D(22) determines which half of word is halfword 2nd opr: if 1, right half; if 0, left half. 3. Halfword 2nd opr is expanded to full word before comparison by propagating sign bit through 16 high-order bits.	Prot (F) Adr Spec	0 : Opr 1 = Opr 2 1 : Opr 1 < Opr 2 2 : Opr 1 > Opr 2
Convert to Binary	CVB	4F	RX	R1 D2(X2, B2)	Convert radix of 2nd opr (in stg) from decimal to binary & place result into 1st opr location (in GPR per R1). 1. 2nd opr is doubleword in packed format. 2. High-order word is converted first. 3. Max positive integer that can be converted is +2,147,483,647. 4. Max negative integer that can be converted is -2,147,483,648.	Prot (F) Adr Spec Data Fix-Pt Div	Unchanged

Table 1-8. Fixed-Point Instructions (Cont)

Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Convert to Decimal	CVD	4E	RX	R1 D2(X2, B2)	Convert radix of 1st opr (in GPR per R1) from binary to decimal & place result into 2nd opr location (in stg). 1. Result is in packed format on doubleword boundary. 2. Low-order 4 bits of field are sign. 3. If PSW(12) = 1, use USASCII-8 code for sign; if PSW(12) = 0, use EBCDIC code.	Prot (S) Adr Spec	Unchanged
Divide	D	5D	RX	R1 D2(X2, B2)	Divide 1st opr (in GPR per R1 & R1 + 1) by 2nd opr (in stg) & place result into 1st opr location (remainder in GPR per R1; quotient in GPR per R1 + 1). 1. R1 must be even adr. 2. D(21) determines which word of doubleword from stg is divisor: if 1, right word; if 0, left word. 3. Relative value of opr's must result in quotient expressible in 32-bit signed integer. 4. Sign of quotient is determined algebraically, except 0 quotient is positive. 5. Sign of remainder is same as sign of dividend, except 0 remainder is positive.	Prot (F) Adr Spec Fix-Pt Div	Unchanged
Divide	DR	1D	RR	R1 R2	Divide 1st opr (in GPR per R1 & R1 + 1) by 2nd opr (in GPR per R2) & place result into 1st opr location (remainder in GPR per R1; quotient in GPR per R1 + 1). 1. R1 must be even adr. 2. Relative value of opr's must result in quotient expressible in 32-bit signed integer. 3. Sign of quotient is determined algebraically, except 0 quotient is positive. 4. Sign of remainder is same as sign of dividend, except 0 remainder is positive.	Spec Fix-Pt Div	Unchanged
Load	L	58	RX	R1 D2(X2, B2)	Load 2nd opr (in stg) into 1st opr location (in GPR per R1). 1. D(21) determines which word of doubleword from stg is to be stored: if 1, right word; if 0, left word. 2. 2nd opr is unchanged.	Prot (F) Adr Spec	Unchanged
Load	LR	18	RR	R1 R2	Load 2nd opr (in GPR per R2) into 1st opr location (in GPR per R1). 2nd opr is unchanged.	None	Unchanged
Load & Test	LTR	12	RR	R1 R2	Load 2nd opr (in GPR per R2) into 1st opr location (in GPR per R1) & set CC according to result. 2nd opr is unchanged.	None	0 : Result = 0 1 : Result $<$ 0 2 : Result $>$ 0
Load Complement	LCR	13	RR	R1 R2	Load 2's complement of 2nd opr (in GPR per R2) into 1st opr location (in GPR per R1) & set CC according to result. Overflow occurs only if max negative number is 2's complemented.	Fix-Pt Ovflo	0 : Result = 0 1 : Result $<$ 0 2 : Result $>$ 0 3 : Overflow
Load Halfword	LH	48	RX	R1 D2(X2, B2)	Load halfword 2nd opr (in stg) into 1st opr location (in GPR per R1). 1. D(21) determines which word of doubleword from stg contains halfword 2nd opr: if 1, right word; if 0, left word. 2. D(22) determines which half of word is halfword 2nd opr: if 1, right half; if 0, left half. 3. Halfword 2nd opr is expanded to full word before loading by propagating sign bit through 16 high-order bits.	Prot (F) Adr Spec	Unchanged

Table 1-8. Fixed-Point Instructions (Cont)

Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Load Multiple	LM	98	RS	R1 R3 D2(B2)	Load 2nd opr (as many words as required; in stg) into GPR's, in ascending order, starting with 1st opr location (per R1) & ending with 3rd opr location (per R3). 1. 2nd opr is unchanged. 2. If R1 = R3, only 1 word is loaded. 3. If R3 < R1, GPR adr's wraparound from 15 to 0. 4. D(21) determines which word of doubleword from stg is to be loaded into LS: if 1, right word; if 0, left word.	Prot (F) Adr Spec	Unchanged
Load Negative	LNR	11	RR	R1 R2	Load 2nd opr (unchanged if negative, 2's complemented if positive; in GPR per R2) into 1st opr location (in GPR per R1). If 2nd opr = 0, unchanged with plus sign.	None	0 : Result = 0 1 : Result < 0
Load Positive	LPR	10	RR	R1 R2	Load 2nd opr (unchanged if positive, 2's complemented if negative; in GPR per R2) into 1st opr location (in GPR per R1). Overflow occurs only if max negative number is 2's complemented.	Fix-Pt Ovflo	0 : Result = 0 2 : Result > 0 3 : Overflow
Multiply	M	5C	RX	R1 D2(X2, B2)	Multiply 1st opr (in GPR per R1 + 1) & 2nd opr (in stg) & place 64-bit result into 1st opr location (in GPR per R1 & R1 + 1). 1. R1 must be even adr. 2. D(21) determines which word of doubleword from stg is 2nd opr: if 1, right word; if 0, left word.	Prot (F) Adr Spec	Unchanged
Multiply	MR	1C	RR	R1 R2	Multiply 1st opr (in GPR per R1 + 1) by 2nd opr (in GPR per R2) & place 64-bit result into 1st opr location (in GPR per R1 & R1 + 1). R1 must be even adr.	Spec	Unchanged
Multiply Halfword	MH	4C	RX	R1 D2(X2, B2)	Multiply 1st opr (in GPR per R1) & halfword 2nd opr (in stg) & place low-order 32 bits of result into 1st opr location. 1. D(21) determines which word of doubleword from stg contains halfword 2nd opr: if 1, right word; if 0, left word. 2. D(22) determines which half of word is halfword 2nd opr: if 1, right half; if 0, left half. 3. Halfword 2nd opr is expanded to full word before multiplication by propagating sign bit through 16 high-order bits.	Prot (F) Adr Spec	Unchanged
Shift Left Double	SLDA	8F	RS	R1 D2(B2)	Shift 1st opr (in GPR per R1 & R1 + 1) left number of bit positions specified by low-order 6 bits of 2nd opr adr & place result into 1st opr location. 1. R1 must be even adr. 2. High-order bits of 1st opr are shifted out & lost; low-order vacated bits are made 0's. 3. If bit unlike sign bit is shifted out of bit position 1 of even register, fixed-point overflow occurs.	Spec Fix-Pt Ovflo	0 : Result = 0 1 : Result < 0 2 : Result > 0 3 : Overflow
Shift Left Single	SLA	8B	RS	R1 D2(B2)	Shift 1st opr (in GPR per R1) left number of bit positions specified by low-order 6 bits of 2nd opr adr & place result into 1st opr location. 1. High-order bits of 1st opr are shifted out & lost; low-order vacated bits are made 0's. 2. If bit unlike sign bit is shifted out of bit position 1 of even register, fixed-point overflow occurs.	Fix-Pt Ovflo	0 : Result = 0 1 : Result < 0 2 : Result > 0 3 : Overflow

Table 1-8. Fixed-Point Instructions (Cont)

Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Shift Right Double	SRDA	8E	RS	R1 D2(B2)	Shift 1st opr (in GPR per R1 & R1 + 1) right number of bit positions specified by low-order 6 bits of 2nd opr adr & place result into 1st opr location. 1. R1 must be even adr. 2. Low-order bits of 1st opr are shifted out & lost; high-order vacated bits are made equal to sign bit.	Spec	0 : Result = 0 1 : Result < 0 2 : Result > 0
Shift Right Single	SRA	8A	RS	R1 D2(B2)	Shift 1st opr (in GPR per R1) right number of bit positions specified by low-order 6 bits of 2nd opr adr & place result into 1st opr location. Low-order bits of 1st opr are shifted out & lost; high-order vacated bits are made equal to sign bit.	None	0 : Result = 0 1 : Result < 0 2 : Result > 0
Store	ST	50	RX	R1 D2(X2, B2)	Store 1st opr (in GPR per R1) into 2nd opr location (in stg). 1. PAL(61) determines into which word of doubleword in stg 1st opr is to be stored: if 1, right word; if 0, left word. 2. 1st opr is unchanged.	Prot (S) Adr Spec	Unchanged
Store Halfword	STH	40	RX	R1 D2(X2, B2)	Store halfword 1st opr (in GPR per R1) into 2nd opr location (in stg). 1. ABC selects 16 low-order bits of 1st opr for storage; high-order bits are ignored. 2. STC [D(21–23)] positions 16 low-order bits of 1st opr into doubleword 2nd opr location. 3. 1st opr is unchanged.	Prot (S) Adr Spec	Unchanged
Store Multiple	STM	90	RS	R1 R3 D2(B2)	Store into 2nd opr location (as many words as required; in stg) contents of GPR's, in ascending order, starting with 1st opr location (per R1) & ending with 3rd opr location (per R3). 1. GPR adr's wrap around from 15 to 0. 2. D(21) determines into which word of doubleword in stg contents of 1st GPR are to be stored: if 1, right word; if 0, left word. 3. If R1 = R3, 1 word is stored.	Prot (S) Adr Spec	Unchanged
Subtract	S	5B	RX	R1 D2(X2, B2)	Algebraically subtract 2nd opr (in stg) from 1st opr (in GPR per R1) & place result into 1st opr location. D(21) determines which word of doubleword from stg is 2nd opr: if 1, right word; if 0, left word.	Prot (F) Adr Spec Fix-Pt Ovflo	0 : Dif = 0 1 : Dif < 0 2 : Dif > 0 3 : Overflow
Subtract	SR	1B	RR	R1 R2	Algebraically subtract 2nd opr (in GPR per R2) from 1st opr (in GPR per R1) & place result into 1st opr location.	Fix-Pt Ovflo	0 : Dif = 0 1 : Dif < 0 2 : Dif > 0 3 : Overflow
Subtract Halfword	SH	4B	RX	R1 D2(X2, B2)	Algebraically subtract halfword 2nd opr (in stg) from 1st opr (in GPR per R1) & place result into 1st opr location. 1. D(21) determines which word of doubleword from stg contains halfword 2nd opr: if 1, right word; if 0, left word. 2. D(22) determines which half of word is halfword 2nd opr: if 1, right half; if 0, left half. 3. Halfword 2nd opr is expanded to full word before subtraction by propagating sign bit through 16 high-order bits.	Prot (F) Adr Spec Fix-Pt Ovflo	0 : Dif = 0 1 : Dif < 0 2 : Dif > 0 3 : Overflow

Table 1-8. Fixed-Point Instructions (Cont)

Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Subtract Logical	SL	5F	RX	R1 D2(X2, B2)	Algebraically subtract 2nd opr (in stg) from 1st opr (in GPR per R1) & place result into 1st opr location. 1. D(21) determines which word of doubleword from stg is 2nd opr: if 1, right word; if 0, left word. 2. Sign bit of result is treated as high-order integer & is tested for carry to determine CC.	Prot (F) Adr Spec	1 : Dif ≠0 (no carry) 2 : Dif = 0 (carry) 3 : Dif ≠0 (carry)
Subtract Logical	SLR	1F	RR	R1 R2	Algebraically subtract 2nd opr (in GPR per R2) from 1st opr (in GPR per R1) & place result into 1st opr location. Sign bit of result is treated as high-order integer & is tested for carry to determine CC.	None	1 : Dif ≠0 (no carry) 2 : Dif = 0 (carry) 3 : Dif ≠ (carry)

2. Addressing. An address designates a location outside the available main storage capacity. The instruction is terminated except for the Store, Store Halfword, and Convert to Decimal instructions, which are suppressed. Operand addresses are tested only when used to address storage. Addresses used as a shift amount are not tested. The address restrictions do not apply to the D2 field or to the contents of the GPR's addressed by the X2 and B2 fields.
3. Specification. A data, instruction, or control word address does not specify an integral boundary for the unit of information, or the R1 field of an instruction specifies an odd register address for a pair of GPR's that contain a doubleword operand. The operation is suppressed.
4. Data. A sign or digit code of the decimal operand in the Convert to Binary instruction is incorrect. The operation is terminated.
5. Fixed-point overflow. A high-order carry occurs, or high-order significant bits are lost in load, add, subtract, or shift operations. The instruction is completed by ignoring the overflow. The interruption may be masked off by making the fixed-point overflow mask bit [PSW(36)] a 0. If the mask bit is a 1, the interruption is taken.
6. Fixed-point divide. The quotient of a division, including division by zero, exceeds the register size, or the result of the Convert to Binary instruction exceeds 31 bits. If the interruption occurs during division, the operation is suppressed. If the interruption occurs during the Convert to Binary instruction, the conversion is completed but only the low-order 32 bits of the converted data are placed into LS.

Condition Codes

The results of fixed-point load, add, subtract, compare, and shift instructions set the CC in the PSW (Table 1-8). All other fixed-point instructions leave the CC undisturbed.

For fixed-point arithmetic operations, the CC can be set to reflect three types of results:

1. For most operations, codes 0, 1, and 2 indicate the result is zero, less than zero, or greater than zero, respectively, and code 3 indicates fixed-point overflow.
2. For compare operations, codes 0, 1, and 2 indicate that the first operand is equal to, lower than, or higher than the second operand, respectively.
3. For Add Logical and Subtract Logical instructions, codes 0 and 1 indicate a zero and non-zero result, respectively, in the absence of a logical carry out of the sign position; codes 2 and 3 indicate a zero and nonzero result, respectively, with a carry out of the sign position.

Floating-Point Instructions

The floating-point instructions serve to load, add, subtract, compare, halve, multiply, divide, and store floating-point numbers. These instructions may occur in the RR format for register-to-register transfers or in the RX format for register-to-storage transfers. Eight 32-bit FPR's in LS are reserved exclusively for floating-point instructions. They are logically connected by pairs to form four 64-bit FPR's. At the end of the execution of the floating-point add, subtract, compare, and certain load instructions, a CC is set.

Operands may be either short or long. Short operands are a word long (32 bits) and long operands are a doubleword long (64 bits). Long operands provide greater precision; however, where great precision is not necessary, short operands reduce instruction execution time and the amount of storage required.

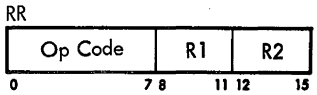
Operands and final arithmetic results are always in true form (as opposed to complement form). A 0 in the sign position indicates a positive fraction; a 1, a negative fraction. If intermediate results are in complement form, they are changed to true form before the final result is stored into the first operand location. For the add, subtract, multiply, and divide instructions, the result signs are determined algebraically.

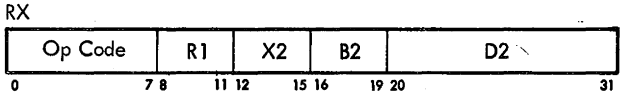
Table 1-9 lists the floating-point instructions. For a discussion of number representation, data formats, normalization, and operand addressing, refer to Section 2 of this chapter.

Instruction Formats

- Floating-point instructions use RR and RX formats.
- Programmer must specify even FPR of even/odd pair (0, 2, 4, or 6).
- Main storage address of second operand must designate word boundaries (bits 22 and 23 = 00) for short operands and doubleword boundaries (bits 21, 22, and 23 = 000) for long operands.

Floating-point instructions occur in the RR and RX formats:





In these formats, R1 is the address of an FPR that contains the first operand. The second operand location is defined differently for the two formats.

In the RR format, R2 is the address of an FPR containing the second operand. The same FPR may be specified for the first and second operands.

The R1 and R2 fields must specify 0, 2, 4, or 6, or a specification program interruption occurs. The specification check is made by testing E(8) and E(11) for zero; for RR instructions, E(12) and E(15) are tested for zero. If E(11) or E(15) does *not* equal zero, an odd address has been specified. If E(8) or E(12) does *not* equal zero, the specified FPR address is greater than 7. Thus, if any of the tested E bits equals 1, a specification program interruption is taken.

In the RX format, the contents of the GPR's specified by X2 and B2 are added to the contents of the D2 field to form an effective address designating the main storage location of the second operand. A zero in an X2 or B2 field indicates that no index or base component is to be used. The main storage address should designate word boundaries for short operands (bits 22 and 23 = 00) and doubleword boundaries (bits 21, 22, and 23 = 000) for long operands. Otherwise, a specification program interruption occurs.

The results replace the first operand except for store operations, where result replaces the second operand. Except for the storing of the final result, the contents of all LS registers and main storage locations participating in operand addressing or operation execution remain unchanged.

Data Flow

- Eight 32-bit LS registers are reserved for floating-point instructions.
- Micro-orders control low-order fraction fetch.
- LS FPR address specified must be even (0, 2, 4, or 6).
- Sign-handling is achieved via serial adder or STAT's.
- Characteristic-handling is performed via serial adder.
- Fraction-handling is performed via parallel adder.

Eight 32-bit LS registers (addresses 16–23) are reserved for floating-point instruction operands and results (Diagram 3-4, FEMDM). An even/odd pair of these registers functions as a double-length (64-bit) register with an assigned address of 0, 2, 4, or 6. A 0 in the R1 or R2 field of a floating-point instruction specifies LS locations 16 and 17; a 2 specifies locations 18 and 19; a 4 specifies locations 20 and 21; a 6 specifies locations 22 and 23.

In instructions other than floating-point, addressing is limited to 16 GPR's because the R1 and R2 fields contain four bits each. The LS address register (LAL), however, contains five bits; LAL(0) is used to address the FPR's. Because floating-point instructions must specify an LS address of 0, 2, 4, or 6 in the R1 and R2 (RR only) fields, and use only the FPR's for operands, a 1 is forced into LAL(0) when accessing LS during execution of a floating-point instruction. (Note that, for RX format instructions, the base and index register fields specify GPR's.) For example, if address 0 is specified by the R1 or R2 field, LS accesses LS register 16 (LAL = 10000). Short operand instructions fetch only 32 bits (single word) from the specified FPR. Because ingating and outgating of LS are limited to 32 bits each, long floating-point operands must be divided into two 32-bit words stored in an even/odd pair of FPR's. Under micro-order control, a 1 is forced into the low-order bit position of LAL [LAL(4)] to fetch or store the low-order 32 bits of a long operand from R1 plus 1 or R2 plus 1. For example, the 'RF*E3Ω1' micro-order specifies the FPR addressed by E(12–15) + 1. The R1 + 1 and R2 + 1 registers are the odd-numbered addresses of FPR's.

At the beginning of the execution phase of floating-point instructions, a specification test establishes that:

1. An even register is specified in the R1 and R2 (RR format only) fields.
2. A register address greater than 6 is not specified in the R1 and R2 (RR format only) fields.
3. The effective main storage address is on a doubleword boundary for long operands and on a word boundary for short operands.

Data flow may be divided into two paths: the fraction path and the sign and characteristic path. The fractions are transferred, added, or shifted via the parallel adder. The operands are located in DT, ST, and AB. For floating-point

Table 1-9. Floating-Point Instructions

Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Add Normalized (long)	AD	6A	RX	R1 D2(X2, B2)	Algebraically add 2nd opr (in stg) to 1st opr (in FPR per R1 & R1 + 1) & place normalized result into 1st opr location. 1. Low-order fraction of 1st opr must be fetched from LS. 2. Set CC per result sign & magnitude.	Prot (F) Adr Spec Exp Ovflo Exp Unflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0
Add Normalized (long)	ADR	2A	RR	R1 R2	Algebraically add 2nd opr (in FPR per R2 & R2 + 1) to 1st opr (in FPR per R1 & R1 + 1) & place normalized result into 1st opr location. 1. Low-order fractions of 1st & 2nd opr's must be fetched from LS. 2. Set CC per result sign & magnitude.	Spec Exp Ovflo Exp Unflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0
Add Normalized (short)	AE	7A	RX	R1 D2(X2, B2)	Algebraically add 2nd opr (in stg) to 1st opr (in FPR per R1) & place normalized result into 1st opr location. 1. Low-order half of FPR is ignored & unchanged. 2. D(21) determines which half of doubleword from stg is 2nd opr; if 1, right half; if 0, left half. 3. Set CC per result sign & magnitude.	Prot (F) Adr Spec Exp Ovflo Exp Unflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0
Add Normalized (short)	AER	3A	RR	R1 R2	Algebraically add 2nd opr (in FPR per R2) to 1st opr (in FPR per R1) & place normalized result into 1st opr location. 1. Low-order halves of FPR's are ignored & unchanged. 2. Set CC per result sign & magnitude.	Spec Exp Ovflo Exp Unflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0
Add Unnormalized (long)	AW	6E	RX	R1 D2(X2, B2)	Algebraically add 2nd opr (in stg) to 1st opr (in FPR per R1 & R1 + 1) & place unnormalized result into 1st opr location. 1. Low-order fraction of 1st opr must be fetched from LS. 2. Set CC per result sign & magnitude.	Prot (F) Adr Spec Exp Ovflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0
Add Unnormalized (long)	AWR	2E	RR	R1 R2	Algebraically add 2nd opr (in FPR per R2 & R2 + 1) to 1st opr (in FPR per R1 & R1 + 1) & place unnormalized result into 1st opr location. 1. Low-order fractions of 1st & 2nd opr's must be fetched from LS. 2. Set CC per result sign & magnitude.	Spec Exp Ovflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0
Add Unnormalized (short)	AU	7E	RX	R1 D2(X2, B2)	Algebraically add 2nd opr (in stg) to 1st opr (in FPR per R1) & place unnormalized result into 1st opr location. 1. Low-order half of FPR is ignored & unchanged. 2. D(21) determines which half of doubleword from stg is 2nd opr: if 1, right half; if 0, left half. 3. Set CC per result sign & magnitude.	Prot (F) Adr Spec Exp Ovflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0
Add Unnormalized (short)	AUR	3E	RR	R1 R2	Algebraically add 2nd opr (in FPR per R2) to 1st opr (in FPR per R1) & place unnormalized result into 1st opr location. 1. Low-order halves of FPR's are ignored & unchanged. 2. Set CC per result sign & magnitude.	Spec Exp Ovflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0
Compare (long)	CD	69	RX	R1 D2(X2, B2)	Algebraically compare 1st opr (in FPR per R1 & R1 + 1) with 2nd opr (in stg); CC indicates result. 1. Low-order fraction of 1st opr must be fetched from LS. 2. Opr's remain unchanged.	Prot (F) Adr Spec	0 : Opr 1 = Opr 2 1 : Opr 1 < Opr 2 2 : Opr 1 > Opr 2

Table 1-9. Floating-Point Instructions (Cont)

Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Compare (long)	CDR	29	RR	R1 R2	Algebraically compare 1st opr (in FPR per R1 & R1 + 1) with 2nd opr (in FPR per R2 & R2 + 1); CC indicates result. 1. Low-order fractions of 1st & 2nd opr's must be fetched from LS. 2. Opr's remain unchanged.	Spec	0 : Opr 1 = Opr 2 1 : Opr 1 < Opr 2 2 : Opr 1 > Opr 2
Compare (short)	CE	79	RX	R1 D2(X2, B2)	Algebraically compare 1st opr (in FPR per R1) with 2nd opr (in stg); CC indicates result. 1. Low-order half of FPR is ignored. 2. D(21) determines which half of doubleword from stg is 2nd opr: if 1, right half; if 0, left half. 3. Opr's remain unchanged.	Prot (F) Adr Spec	0 : Opr 1 = Opr 2 1 : Opr 1 < Opr 2 2 : Opr 1 > Opr 2
Compare (short)	CER	39	RR	R1 R2	Algebraically compare 1st opr (in FPR per R1) with 2nd opr (in FPR per R2); CC indicates result. 1. Low-order halves of FPR's are ignored. 2. Opr's remain unchanged.	Spec	0 : Opr 1 = Opr 2 1 : Opr 1 < Opr 2 2 : Opr 1 > Opr 2
Divide (long)	DD	6D	RX	R1 D2(X2, B2)	Divide 1st opr (in FPR per R1 & R1 + 1) by 2nd opr (in stg) & place normalized quotient into 1st opr location. 1. Low-order fraction of 1st opr must be fetched from LS. 2. Opr's are prenormalized. 3. Remainder is not saved.	Prot (F) Adr Spec Exp Ovflo Exp Unflo Flt-Pt Div	Unchanged
Divide (long)	DDR	2D	RR	R1 R2	Divide 1st opr (in FPR per R1 & R1 + 1) by 2nd opr (in FPR per R2 & R2 + 1) & place normalized quotient into 1st opr location. 1. Low-order fractions of 1st & 2nd opr's must be fetched from LS. 2. Opr's are prenormalized. 3. Remainder is not saved.	Spec Exp Ovflo Exp Unflo Flt-Pt Div	Unchanged
Divide (short)	DE	7D	RX	R1 D2(X2, B2)	Divide 1st opr (in FPR per R1) by 2nd opr (in stg) & place normalized quotient into 1st opr location. 1. Low-order half of FPR is ignored & unchanged. 2. D(21) determines which half of doubleword from stg is 2nd opr: if 1, right half; if 0, left half. 3. Opr's are prenormalized. 4. Remainder is not saved.	Prot (F) Adr Spec Exp Ovflo Exp Unflo Flt-Pt Div	Unchanged
Divide (short)	DER	3D	RR	R1 R2	Divide 1st opr (in FPR per R1) by 2nd opr (in FPR per R2) & place normalized quotient into 1st opr location. 1. Low-order halves of FPR's are ignored & unchanged. 2. Opr's are prenormalized. 3. Remainder is not saved.	Spec Exp Ovflo Exp Unflo Flt-Pt Div	Unchanged
Halve (long)	HDR	24	RR	R1 R2	Divide 2nd opr (in FPR per R2 & R2 + 1) by 2 & place normalized quotient into 1st opr location (in FPR per R1 & R1 + 1). Low-order fraction of 2nd opr must be fetched from LS.	Spec Exp Unflo	Unchanged
Halve (short)	HER	34	RR	R1 R2	Divide 2nd opr (in FPR per R2) by 2 & place normalized quotient into 1st opr location (in FPR per R1). Low-order halves of FPR's are ignored & unchanged.	Spec Exp Unflo	Unchanged
Load (long)	LD	68	RX	R1 D2(X2, B2)	Load 2nd opr (in stg) into 1st opr location (in FPR per R1 & R1 + 1).	Prot (F) Adr Spec	Unchanged

Table 1-9. Floating-Point Instructions (Cont)

Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Load (long)	LDR	28	RR	R1 R2	Load 2nd opr (in FPR per R2 & R2 + 1) into 1st opr location (in FPR per R1 & R1 + 1). Low-order fraction of 2nd opr must be fetched from LS.	Spec	Unchanged
Load (short)	LE	78	RX	R1 D2(X2, B2)	Load 2nd opr (in stg) into 1st opr location (in FPR per R1). 1. D(21) determines which half of doubleword from stg is 2nd opr: if 1, right half; if 0, left half. 2. Low-order half of FPR is ignored & unchanged.	Prot (F) Adr Spec	Unchanged
Load (short)	LER	38	RR	R1 R2	Load 2nd opr (in FPR per R2) into 1st opr location (in FPR per R1). Low-order halves of FPR's are ignored & unchanged.	Spec	Unchanged
Load & Test (long)	LTDR	22	RR	R1 R2	Load 2nd opr (in FPR per R2 & R2 + 1) into 1st opr location (in FPR per R1 & R1 + 1). 1. Low-order fraction of 2nd opr must be fetched from LS. 2. Set CC according to sign & magnitude.	Spec	0 : 2nd opr fract = 0 1 : 2nd opr <0 2 : 2nd opr >0
Load & Test (short)	LTER	32	RR	R1 R2	Load 2nd opr (in FPR per R2) into 1st opr location (in FPR per R1). 1. Low-order halves of FPR's are ignored & unchanged. 2. Set CC according to sign & magnitude.	Spec	0 : 2nd opr fract = 0 1 : 2nd opr <0 2 : 2nd opr >0
Load Complement (long)	LCDR	23	RR	R1 R2	Load 2nd opr (in FPR per R2 & R2 + 1) into 1st opr location (in FPR per R1 & R1 + 1) with sign complemented. 1. Low-order fraction of 2nd opr must be fetched from LS. 2. Set CC according to original sign & magnitude.	Spec	0 : 2nd opr fract = 0 1 : Orig sign + 2 : Orig sign -
Load Complement (short)	LCER	33	RR	R1 R2	Load 2nd opr (in FPR per R2) into 1st opr location (in FPR per R1) with sign complemented. 1. Low-order halves of FPR's are ignored & unchanged. 2. Set CC according to original sign & magnitude.	Spec	0 : 2nd opr fract = 0 1 : Orig sign + 2 : Orig sign -
Load Negative (long)	LNDR	21	RR	R1 R2	Load 2nd opr (in FPR per R2 & R2 + 1) into 1st opr location (in FPR per R1 & R1 + 1) with sign made minus. 1. Low-order fraction of 2nd opr must be fetched from LS. 2. Set CC according to result sign & magnitude.	Spec	0 : 2nd opr fract = 0 1 : 2nd opr <0
Load Negative (short)	LNER	31	RR	R1 R2	Load 2nd opr (in FPR per R2) into 1st opr location (in FPR per R1) with sign made minus. 1. Low-order halves of FPR's are ignored & unchanged. 2. Set CC according to result sign & magnitude.	Spec	0 : 2nd opr fract = 0 1 : 2nd opr <0
Load Positive (long)	LPDR	20	RR	R1 R2	Load 2nd opr (in FPR per R2 & R2 + 1) into 1st opr location (in FPR per R1 & R1 + 1) with sign made plus. 1. Low-order fraction of 2nd opr must be fetched from LS. 2. Set CC according to result sign & magnitude.	Spec	0 : 2nd opr fract = 0 2 : 2nd opr >0
Load Positive (short)	LPER	30	RR	R1 R2	Load 2nd opr (in FPR per R2) into 1st opr location (in FPR per R1) with sign made plus. 1. Low-order halves of FPR's are ignored & unchanged. 2. Set CC according to result sign & magnitude.	Spec	0 : 2nd opr fract = 0 2 : 2nd opr >0

Table 1-9. Floating-Point Instructions (Cont)

Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Multiply (long)	MD	6C	RX	R1 D2(X2, B2)	Multiply 1st opr (in FPR per R1 & R1 + 1) & 2nd opr (in stg) & place normalized product into 1st opr location (in FPR per R1 & R1 + 1). Opr's are prenormalized.	Prot (F) Adr Spec Exp Ovflo Exp Unflo	Unchanged
Multiply (long)	MDR	2C	RR	R1 R2	Multiply 1st opr (in FPR per R1 & R1 + 1) & 2nd opr (in FPR per R2 & R2 + 1) & place normalized product into 1st opr location (in FPR per R1 & R1 + 1). Opr's are prenormalized.	Spec Exp Ovflo Exp Unflo	Unchanged
Multiply (short)	ME	7C	RX	R1 D2(X2, B2)	Multiply 1st opr (in FPR per R1) & 2nd opr (in stg) & place normalized product into 1st opr location (in FPR per R1 & R1 + 1). 1. D(21) determines which half of doubleword from stg is 2nd opr: if 1, right half; if 0, left half. 2. Opr's are prenormalized.	Prot (F) Adr Spec Exp Ovflo Exp Unflo	Unchanged
Multiply (short)	MER	3C	RR	R1 R2	Multiply 1st opr (in FPR per R1) & 2nd opr (in FPR per R2) & place normalized product into 1st opr location (in FPR per R1 & R1 + 1). Opr's are prenormalized.	Spec Exp Ovflo Exp Unflo	Unchanged
Store (long)	STD	60	RX	R1 D2(X2, B2)	Store 1st opr (in FPR per R1 & R1 + 1) into 2nd opr location (in stg). 1st opr is unchanged.	Prot (S) Adr Spec	Unchanged
Store (short)	STE	70	RX	R1 D2(X2, B2)	Store 1st opr (in FPR per R1) into 2nd opr location (in stg). 1. PAL(61) determines into which half of doubleword in stg 1st opr is to be stored: if 1, right half; if 0, left half. 2. Low-order half of FPR is ignored. 3. 1st opr is unchanged.	Prot (S) Adr Spec	Unchanged
Subtract Normalized (long)	SD	6B	RX	R1 D2(X2, B2)	Algebraically subtract 2nd opr (in stg) from 1st opr (in FPR per R1 & R1 + 1) & place normalized result into 1st opr location. 1. Low-order fraction of 1st opr must be fetched from LS. 2. Set CC per result sign & magnitude.	Prot (F) Adr Spec Exp Ovflo Exp Unflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0
Subtract Normalized (long)	SDR	2B	RR	R1 R2	Algebraically subtract 2nd opr (in FPR per R2 & R2 + 1) from 1st opr (in FPR per R1 & R1 + 1) & place normalized result into 1st opr location. 1. Low-order fractions of 1st & 2nd opr's must be fetched from LS. 2. Set CC per result sign & magnitude.	Spec Exp Ovflo Exp Unflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0
Subtract Normalized (short)	SE	7B	RX	R1 D2(X2, B2)	Algebraically subtract 2nd opr (in stg) from 1st opr (in FPR per R1) & place normalized result into 1st opr location. 1. Low-order half of FPR is ignored & unchanged. 2. D(21) determines which half of doubleword from stg is 2nd opr: if 1, right half; if 0, left half. 3. Set CC per result sign & magnitude.	Prot (F) Adr Spec Exp Ovflo Exp Unflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0
Subtract Normalized (short)	SER	3B	RR	R1 R2	Algebraically subtract 2nd opr (in FPR per R2) from 1st opr (in FPR per R1) & place normalized result into 1st opr location. 1. Low-order halves of FPR's are ignored & unchanged. 2. Set CC per result sign & magnitude.	Spec Exp Ovflo Exp Unflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0

Table 1-9. Floating-Point Instructions (Cont)

Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Subtract Unnormalized (long)	SW	6F	RX	R1 D2(X2, B2)	Algebraically subtract 2nd opr (in stg) from 1st opr (in FPR per R1 & R1 + 1) & place unnormalized result into 1st opr location. 1. Low-order fraction of 1st opr must be fetched from LS. 2. Set CC per result sign & magnitude.	Prot (F) Adr Spec Exp Ovflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0
Subtract Unnormalized (long)	SWR	2F	RR	R1 R2	Algebraically subtract 2nd opr (in FPR per R2 & R2 + 1) from 1st opr (in FPR per R1 & R1 + 1) & place unnormalized result into 1st opr location. 1. Low-order fractions of 1st & 2nd opr's must be fetched from LS. 2. Set CC per result sign & magnitude.	Spec Exp Ovflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0
Subtract Unnormalized (short)	SU	7F	RX	R1 D2(X2, B2)	Algebraically subtract 2nd opr (in stg) from 1st opr (in FPR per R1) & place unnormalized result into 1st opr location. 1. Low-order half of FPR is ignored & unchanged. 2. D(21) determines which half of doubleword from stg is 2nd opr: if 1, right half; if 0, left half. 3. Set CC per result sign & magnitude.	Prot (F) Adr Spec Exp Ovflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0
Subtract Unnormalized (short)	SUR	3F	RR	R1 R2	Algebraically subtract 2nd opr (in FPR per R2) from 1st opr (in FPR per R1) & place unnormalized result into 1st opr location. 1. Low-order halves of FPR's are ignored & unchanged. 2. Set CC per result sign & magnitude.	Spec Exp Ovflo Signif	0 : Fract = 0 1 : Fract < 0 2 : Fract > 0

instructions, the parallel adder shifts operands right or left four bit positions under micro-order control. For floating-point, the contents of PAL can be gated to T, D, A, and B.

The sign and characteristic path is from ST or AB to F, via the serial adder. The byte gated to the inputs of the serial adder depends upon the STC and the ABC values. For floating-point, the STC is normally set to 4 to specify the first byte of T, and the ABC is set to 0 to specify the first byte of A. The data from F is gated to the serial adder. From the serial adder, the data is transferred to ST per the STC. For floating-point operations, the serial adder adds 1 to, or subtracts 1 or 64 from, the characteristic at the inputs of the serial adder under micro-order control.

In floating-point instructions, the signs are saved in STAT's under micro-order control. When the 'SAVE SIGNS' micro-order is executed, bit 0 of the ST byte selected by the STC is gated in true or complement form, depending upon the instruction, to STAT C. Because the STC is set to 0 or 4 before issuing the 'SAVE SIGNS' micro-order, the contents of either S(0) or T(32), whichever contains the sign of the operand, is saved in STAT C via the serial adder. At the same time, the sign of the operand in AB is sent to STAT F. If the instruction is a multiply or divide and SAL(0) = 1 (indicating a carry resulted from the characteristic addition or subtraction), STAT D is set.

When the results of the execution of a floating-point instruction are to be stored, the STAT's are decoded, under micro-order control, to determine the sign of the result. If the sign is minus, the 'INSERT SIGN' micro-order forces a 1 to bit 0 of the FPR (on the LS bus in) addressed by R1 and inhibits gating of T(32) to LS. The result sign is minus under the following conditions:

1. Multiply or divide and signs (STAT's C and F) are unlike.
2. Load complement and STAT C equals 0.
3. Halve, load, or load and test, and STAT C equals 1.
4. Add, subtract, or compare and sign of the larger operand is minus.
5. Load negative.

Program Interruptions

Seven program interruptions can occur during execution of floating-point instructions. Of the seven, "exponent underflow" and "significance" can be masked off; the others are unconditionally taken. If the associated mask bit [PSW(38) and PSW(39), respectively] is a 0, the interruption is ignored; if a 1, it is taken.

The seven interruptions and their causes are:

1. Protection. The storage key does not match the protection key in the PSW for all RX instructions. When an

instruction causes a fetch-protection violation, instruction execution is terminated, the program execution is altered by a program interruption, and a protection program interruption is indicated in the old PSW. When an instruction causes a store-protection violation, the operation is suppressed.

2. Addressing. An address designates a location outside the available storage for the installation. The operation is terminated.
3. Specification. A short operand is not located on a word boundary, a long operand is not located on a double-word boundary, or an FPR address other than 0, 2, 4, or 6 is specified. The instruction is suppressed. The address restrictions do not apply to the components (contents of the D2 field and the contents of the LS registers specified by X2 and B2) from which an address is generated.
4. Exponent overflow. The result exponent (characteristic) of an addition, subtraction, multiplication, or division overflows, and the result fraction is not zero. The operation is completed by making the characteristic 128 smaller than the true result; the sign and fraction remain unchanged.
5. Exponent underflow. The result of an addition, subtraction, multiplication, or division underflows, and the result fraction is not zero. A program interruption occurs if the exponent-underflow mask [PSW(38)] is a 1. The operation is completed by replacing the result with a true zero, if the mask is off. If the mask is on, the characteristic is made 128 larger than the true result and the sign and fraction remain unchanged.
6. Significance. The result fraction of an addition or subtraction is zero. A program interruption occurs if the significance mask [PSW(39)] is a 1. The mask bit also affects the result of the operation. When the significance mask bit is a 0, the operation is completed by replacing the result with a true zero. When the significance mask bit is 1, the operation is completed without further change to the characteristic of the result. In either case, the CC is set to 0.
7. Floating-point divide. Division by a number with a zero fraction is attempted. The division is suppressed, but the CC and the data in storage remain unchanged.

Condition Codes

The results of floating-point add, subtract, compare, and certain load operations set the CC (Table 1-9). Multiplication, division, and storing leave the CC unchanged.

The CC can be set to reflect two types of results for floating-point arithmetic. For most operations, CC's of 0, 1, and 2 respectively indicate that the result register contains zero, less than zero, and more than zero. A zero result is indicated whenever the result fraction is zero, including a forced zero. A CC of 3 is never set by floating-point instructions.

For compare instructions, CC's of 0, 1, and 2 respectively indicate that the first operand is equal to, lower than, and higher than the second operand.

Decimal Instructions

The decimal instructions provide for addition, subtraction, comparison, multiplication, division, and format conversion of variable-field length (VFL) operands. The VFL data, which may range from 1 to 16 bytes in length, resides in main storage only. All decimal instructions are therefore in the SS format to provide for storage-to-storage operations. In general, most decimal instructions require fetching the operands from main storage, performing the operations specified by the instruction op code, and storing the results in main storage. A list of the decimal instructions is contained in Table 1-10.

For a discussion of number representation, data formats, and operand addressing, refer to Section 2 of this chapter.

Data Handling

- Decimal arithmetic is performed by either true add or complement add sequence, using excess-6 arithmetic.
- True add sequence adds 6 to each digit gated to A-side of serial adder.
- Complement add sequence gates 2's complement of each digit to A-side of serial adder.
- Inputs to B-side of serial adder are unchanged.
- Each digit which did not cause a carry at output of serial adder is reduced by 6 (decimal corrected).
- If no carry from high-order digit, result is in complement form and must be recomplemented.

Decimal arithmetic operations are performed in the serial adder on a byte basis. A true add or a complement add sequence is used depending upon the instruction and the operand signs. Because decimal digits are in BCD format, excess-6 arithmetic is used.

As stated previously, the decimal digits are represented by a binary code. Each digit consists of a four-bit field, bit combinations 0000–1001 corresponding to decimal digits 0–9. This system of decimal notation allows relatively simple binary techniques to be applied when operating with decimal data, and also facilitates direct reading of decimal results. However, two problems are encountered. One problem is that the four-bit field used to represent decimal

Table 1-10. Decimal Instructions

Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Add Decimal	AP	FA	SS	D1(L1, B1) D2(L2, B2)	Algebraically add 2nd opr (in stg) to 1st opr (in stg) & place result into 1st opr location. 1. Opr's & result are in packed format. 2. Opr fields may overlap if low-order bytes coincide. 3. Right to left, byte by byte. 4. Shorter opr is extended with high-order 0's. 5. 1st opr field must be large enough to contain all 2nd opr significant digits.	Prot (S,F) Adr Data Dec Ovflo	0 : Sum = 0 1 : Sum < 0 2 : Sum > 0 3 : Overflow
Compare Decimal	CP	F9	SS	D1(L1, B1) D2(L2,B2)	Algebraically compare 1st opr (in stg) with 2nd opr (in stg) & set CC according to result. 1. Opr's are in packed format. 2. Shorter opr is extended with high-order 0's. 3. Opr fields may overlap if low-order bytes coincide. 4. Right to left, byte by byte. 5. Result is not stored & opr fields are unchanged.	Prot (F) Adr Data	0 : Opr 1 = Opr 2 1 : Opr 1 < Opr 2 2 : Opr 1 > Opr 2
Divide Decimal	DP	FD	SS	D1(L1, B1) D2(L2, B2)	Divide 1st opr (in stg) by 2nd opr (in stg) & place result into 1st opr location (quotient is leftmost in 1st opr location; remainder, rightmost). 1. Opr's are in packed format. 2. Dividend must contain at least 1 high-order 0. 3. Max dividend field = 16 bytes (31 digits & sign); L1 = 15. 4. Max divisor field = 8 bytes (15 digits & sign); L2 = 7. 5. Divisor field must be < dividend field (L2 < L1). 6. Max quotient field = 15 bytes. 7. Quotient field = dividend field minus remainder (divisor) field (L1 minus L2). 8. Remainder field = divisor field. 9. Opr fields may overlap if low-order bytes coincide. 10. Sign of quotient is determined algebraically, except 0 result is positive. 11. Sign of remainder is same as dividend sign.	Prot (S,F) Adr Spec Data Dec Div	Unchanged
Move with Offset	MVO	F1	SS	D1(L1, B1) D2(L2, B2)	Store 2nd opr (in stg) to left of and adjacent to low-order 4 bits of 1st opr (in stg). 1. Opr's are in packed or unpacked format. 2. If 2nd opr is shorter than 1st opr, fill 1st opr field with high-order 0's. 3. If 2nd opr is longer than 1st opr, ignore excess 2nd opr high-order digits. 4. Right to left, byte by byte.	Prot (S,F) Adr	Unchanged
Multiply Decimal	MP	FC	SS	D1(L1, B1) D2(L2, B2)	Multiply 1st opr (in stg) by 2nd opr (in stg) & place result into 1st opr location. 1. Opr's are in packed format. 2. Product must contain at least 1 high-order 0. 3. Max multiplicand field = 16 bytes (31 digits & sign); L1 = 15. 4. Max multiplier field = 8 bytes (15 digits & sign); L2 = 7. 5. Multiplier field must be < multiplicand field (L2 < L1); max value of L2 = 7. 6. Multiplicand field initially contains high-order 0-field equal in length to multiplier field. 7. Max product field = 16 bytes (31 digits & sign). 8. Sign of product is determined algebraically, except 0 result is positive.	Prot (S,F) Adr Spec Data	Unchanged

Table 1-10. Decimal Instructions (Cont)

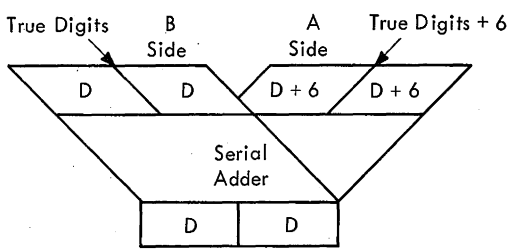
Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Pack	PACK	F2	SS	D1(L1, B1) D2(L2, B2)	Convert format of 2nd opr (in stg) from zoned to packed & place result into 1st opr location (in stg). 1. 2nd opr is in zoned format. 2. No restriction on overlapping fields. 3. Extend 2nd opr with high-order 0's, if necessary. 4. If 1st opr field is too short to contain all significant digits of 2nd opr field, ignore excess 2nd opr high-order digits. 5. Right to left, byte by byte.	Prot (S,F) Adr	Unchanged
Subtract Decimal	SP	FB	SS	D1(L1, B1) D2(L2, B2)	Algebraically subtract 2nd opr (in stg) from 1st opr (in stg) & place result into 1st opr location. 1. Opr's & result are in packed format. 2. Opr fields may overlap if low-order bytes coincide. 3. 1st opr field must be large enough to contain all 2nd opr significant digits. 4. Shorter opr is extended with high-order 0's. 5. Right to left, byte by byte.	Prot (S,F) Adr Data Dec Ovflo	0 : Dif = 0 1 : Dif < 0 2 : Dif > 0 3 : Overflow
Unpack	UNPK	F3	SS	D1(L1, B1) D2(L2, B2)	Convert format of 2nd opr (in stg) from packed to zoned & place result into 1st opr location (in stg). 1. 2nd opr is in packed format. 2. No restriction on overlapping fields. 3. Extend 2nd opr with high-order 0's, if necessary. 4. If 1st opr field is too short to contain all significant digits of 2nd opr field, ignore excess 2nd opr high-order digits. 5. If PSW(12) = 1, use USASCII-8 code for zones; if PSW(12) = 0, use EBCDIC. 6. Right to left, byte by byte.	Prot (S,F) Adr	Unchanged
Zero & Add	ZAP	F8	SS	D1(L1, B1) D2(L2, B2)	Place 2nd opr (in stg) into 1st opr location (in stg). 1. 2nd opr is in packed format. 2. Opr fields may overlap if low-order byte of 1st opr coincides with or is to the right of low-order byte of 2nd opr. 3. 1st opr field must be large enough to contain all 2nd opr significant digits.	Prot (S,F) Adr Data Dec Ovflo	0 : Result = 0 1 : Result < 0 2 : Result > 0 3 : Overflow

digits has 16 possible codes, of which 6 (binary combinations for 10 through 15 inclusive) are invalid as decimal digits. Thus means must be provided to correct invalid results when they occur in an arithmetic operation. For example, the addition of decimal digits 0110 (six) and 0101 (five) must yield a decimal result of 0001 0001 (eleven). If, however, a pure binary addition is carried out, it will yield an unacceptable result:

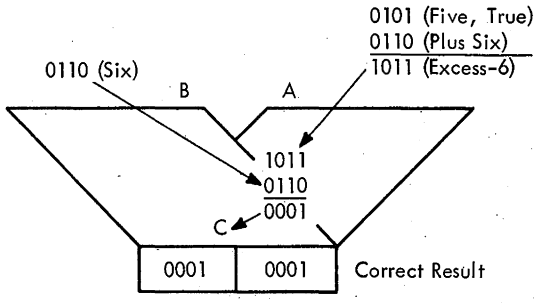
```
  0110   (decimal, or binary 6)
  0101   (decimal, or binary 5)
 ------
  1011   (invalid as decimal, but 11 in binary)
```

The second problem is in the generation of a decimal carry. When the sum of two decimal digits exceeds 9, a carry must be sent to the next high-order digit. However, a pure binary addition does not yield a carry unless the sum of the digits exceeds 1111 (15), which has the effect of a hex carry; i.e., carrying the order of 16 rather than 10.

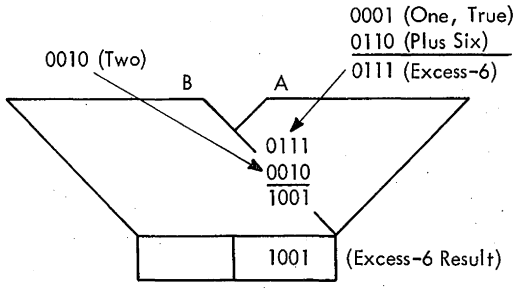
Both of the above problems are solved by the excess-6 arithmetic scheme and the decimal correction functions of the serial adder. In the excess-6 scheme, often referred to as true +6 arithmetic, a 6 is added to each digit as it is gated to the A-side of the adder, one byte (two digits) at a time from the second operand; the digits gated to the adder B-side, one byte at a time from the first operand, are not affected:



If the sum of the two digits to be added is 10 or greater, the true +6 scheme automatically eliminates the unwanted binary configuration and also supplies a decimal carry in terms of a hex carry. In true +6 arithmetic, the previous add example of digits 5 and 6 is executed as follows:

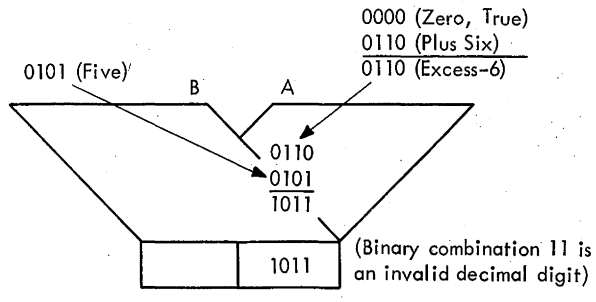


Addition of a 6 in all cases, however, may create an erroneous and sometimes invalid result. This occurs if the sum of the two digits to be added is less than 10. For example, consider the addition of decimal digits 1 and 2:



In the above case, the result (9) is clearly in excess-6 form; the digit 6 must be subtracted from the result to obtain the correct answer.

A further example illustrates how an excess-6 digit may generate an invalid result. Consider the addition of decimal digits 0 and 5:

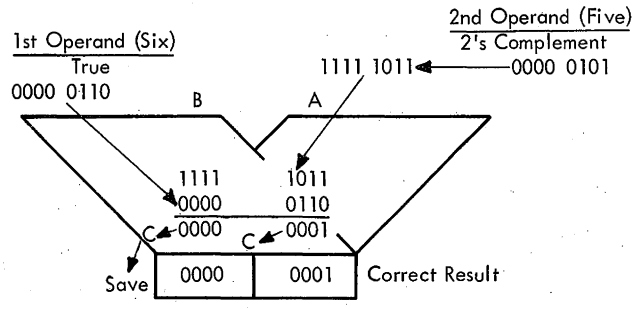


Note that both the erroneous and the invalid results are characterized by a no-carry to the next high-order digit. This condition holds true in all cases when incorrect data is generated, and is utilized by the decimal correction logic of the adder. When a no-carry condition is detected, this logic automatically deducts 6 from the result, thus supplying the correct digit to the adder output.

The decimal correct function of the adder is also used during complement add operations. The binary codes of the decimal digits at the adder A-side are gated in 2's complement form; excess 6's are ***not*** supplied. The digits at the B-side of the adder are gated in true form. The result of a complement add operation may be in true or complement form.

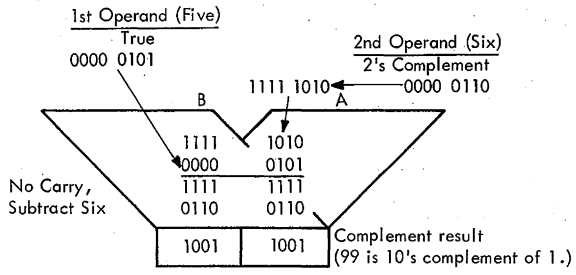
For clarity, the previous examples have shown operations that use only one digit. However, the serial adder normally handles one byte (two digits) at a time. To demonstrate the operation of the serial adder during decimal operations, the following examples deal with a byte of data.

If the first operand is larger than the second, the result is in true form. Consider complement addition of decimal digit 5 to 6; that is 6 minus 5:



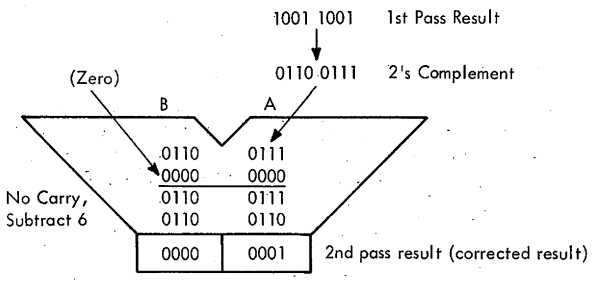
A true result during a complement add operation is always characterized by a carry from the last high-order digit. As in the case of the true add operation, a carry to the next digit indicates that no decimal correction of that digit is necessary.

If the first operand is smaller than the second, the result is in complement form. Because decimal data is always stored in true form, the result must be recomplemented. Consider complement addition of the decimal digit 6 to 5; that is, 5 minus 6:



Note that the decimal correction feature of the adder always subtracts 6 from each digit position which does not produce a carry. In a complement add operation, a no-carry condition from the last high-order digit also indicates that

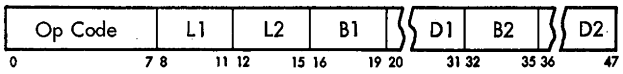
the result is in complement form and must be recomplemented. This requires a second pass through the adder:



Instruction Format

- Instructions specify two addresses.
- B1 (contents) + D1 + L1 specifies rightmost byte of 1st operand.
- B2 (contents) + D2 + L2 specifies rightmost byte of 2nd operand.
- Results are stored in true form at first operand location.

All decimal instructions use the SS format:



An SS instruction operates on two operands in 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 contents of the GPR specified by the B1 field are added to the D1 field to form an address. This address specifies the leftmost byte of the first operand. The number of operand bytes to the right of this byte is specified by the L1 field of the instruction. The L1 field may specify up to 16 bytes. Similarly, the address of the second operand is specified by the B2, D2, and L2 fields of the instruction. A zero in the B1 or B2 fields indicates the absence of the corresponding address component.

Normally, decimal operands are processed from right to left. Thus the address for the initial operand fetch is:

LS register per B-field + D-field + L-field.

Operands are fetched from main storage one doubleword, or eight bytes, at a time. Because the L-field may specify up to 16 bytes, several operand fetches may be required to completely access the operand. After each fetch, the operand address is decremented by 8 to access the next high-order eight bytes of the operand.

The results of decimal operations are placed into the first operand field and must be in true form. The result is never stored outside the first operand field specified by the instruction. If the first operand is longer than the second, the second operand is extended with high-order zeros up to the length of the first operand. Such extension does not modify the second operand in main storage, where it remains unchanged.

Data Flow

- All decimal instructions use serial adder.
- First operand is placed into ST; second operand into AB.
- STC specifies which ST byte is to be processed. ABC specifies which AB byte is to be processed.
- Destination bytes replace first operand bytes in ST.
- D contains first operand and destination address.
- IC contains second operand address.
- L1 and L2 specify number of first and second operand bytes, respectively, to be processed.

The data path used for decimal operations consists primarily of ST, AB, and the serial adder (Diagram 3-5, FEMDM). ST contains the first operand, and AB the second. The input byte to the adder A-side is selected from AB under control of the ABC. The input to the B-side of the adder is selected from ST under STC control. The selected bytes are gated to the adder simultaneously.

The serial adder handles the data at a rate of one byte per cycle; i.e., for each two input bytes, one output byte is generated at the SAL. The output byte is gated from SAL to ST under control of the STC, after which the ABC and the STC are decremented and a new cycle is started. Thus, as the operation progresses, the first operand bytes in ST are replaced by the destination bytes.

The number of first and second operand bytes processed depends upon length fields L1 and L2, respectively. The L1 count contained in E(8–11) is decremented once for each byte of first operand that is processed. Similarly, the L2 count in E(12–15) is decremented once for each second operand byte processed.

D contains the address of the first operand, which is also the address of the destination. The initial address in D specifies the doubleword containing the rightmost byte of the operand. When the STC is decremented to zero, indicating that all first operand bytes in ST have been processed, the contents of ST are stored into main storage. If additional first operand bytes remain in main storage (the L1 count has not stepped to zero), the D-address is decremented by 8, and a fetch of the next operand doubleword is made to ST.

Storage of the destination bytes in ST is controlled by the mark triggers. The mark triggers permit alteration of only those bytes in main storage that belong to the field being processed. There is one mark trigger for each of the eight bytes in ST. As a byte of processed data is gated to ST, the corresponding mark trigger is set, thus designating the byte for main storage.

The IC contains the address of the second operand. (The instruction address is held in the LSWR during execution of SS instructions.) The initial IC address specifies the doubleword containing the rightmost byte of the second operand. When the ABC is decremented to zero, all operand bytes in AB have been processed. If additional second operand bytes remain in main storage (L2 count has not stepped to zero), the IC address is decremented by 8 and a fetch is made of the next second operand doubleword to AB.

This pattern of fetching data, processing via the serial adder, assembling the results in ST, and storing the contents of ST into main storage is continued until either the first or the second operand length field (L1 or L2 count, respectively) has counted below zero. The operation at this point depends upon the individual instruction. If L2 has been exhausted but not L1, some instructions may require extension of the remaining first operand bytes with high-order zeros. On the other hand, if L1 is exhausted before L2, the instruction may test the remaining second operand bytes for presence of significant digits. This test is performed to detect a possible overflow condition and is accomplished by running the excess second operand bytes through the serial adder and sensing nonzeros. In all cases, if both L1 and L2 counts are exhausted, the instruction execution ends after the last destination word is stored into main storage.

Some decimal operations require use of the parallel adder to perform a right-4 or left-4 shift of the entire operand. The "spilled" bits generated during the shift are held in B(64–67).

Program Interruptions

Six program interruptions can occur during execution of decimal instructions. Of the six, only decimal overflow can be masked off; the others are unconditionally taken. If the decimal overflow mask bit [PSW(37)] is a 0, the decimal overflow interruption is ignored; if a 1, it is taken.

The six interruptions and their causes are:

1. Protection. The storage key does not match the protection key in the PSW. The operation is terminated for either a store or a fetch violation.
2. Addressing. An address designates a location outside the available storage for the installed system. The operation is terminated.
3. Specification. A multiplier or a divisor size exceeds 15 digits and sign, or a divisor is equal to or greater than the dividend, or a multiplier is equal to or greater than the multiplicand. The instruction is suppressed.
4. Data. A sign or digit code of an operand specified in the Add, Subtract, Compare, Zero and Add, Multiply, or Divide instruction is incorrect, a multiplicand has insufficient high-order zeros, or the operand fields in these instructions overlap. The operation is terminated before any original data is changed in main storage, except for an invalid digit code which is detected after the first store cycle.
5. Decimal overflow. Execution of the Add, Subtract, or Zero and Add instruction results in an overflow condition. The program interruption occurs only when the decimal-overflow mask [PSW(37)] is a 1. The operation is completed by placing the truncated low-order result into the result field and setting the CC to 3. The sign and low-order digits contained in the result field are the same as they would have been for an infinitely long result field.
6. Decimal divide. The quotient exceeds the specified data field, including division by zero. Division is suppressed. Therefore, the dividend and divisor remain unchanged in storage.

Condition Codes

The results of the Decimal Add, Subtract, Compare, and Zero and Add instructions set the CC as shown in Table 1-10.

Logical Instructions

The logical instructions provide for logical manipulation of data: moving, comparing, bit testing, bit connecting, translating, editing, and shifting. The logical instructions use all five instruction formats and work with both fixed- and variable-field length data. Table 1-11 lists the logical instructions.

For a discussion of the eight-bit zoned character codes, data formats, and operand addressing, refer to Section 2 of this chapter.

Instruction Formats

Logical instructions use the following five formats:

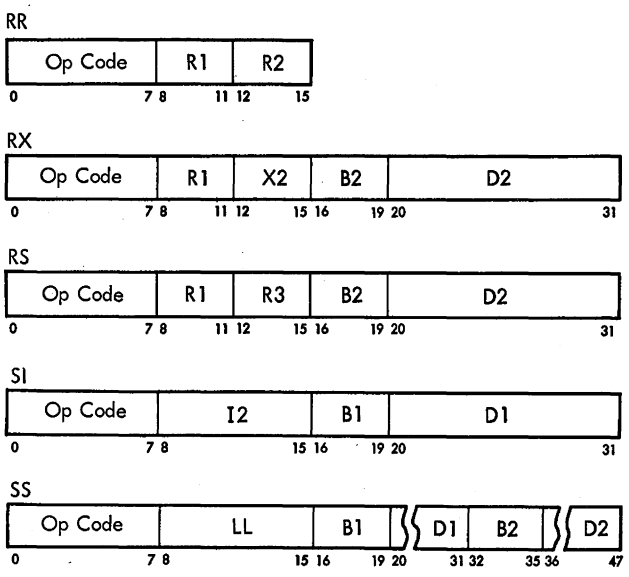


Table 1-11. Logical Instructions

Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
AND	N	54	RX	R1 D2(X2, B2)	AND 1st opr (in GPR per R1) with 2nd opr (in stg) & place result into 1st opr location. Left to right, byte by byte.	Prot (F) Adr Spec	0 : Result = 0 1 : Result ≠0
AND	NC	D4	SS	D1(L, B1) D2(B2)	AND 1st opr (in stg) with 2nd opr (in stg) & place result into 1st opr location. 1. Left to right, byte by byte. 2. Max number of bytes is 256.	Prot (S,F) Adr	0 : Result = 0 1 : Result ≠0
AND	NI	94	SI	D1(B1) I2	AND immediate opr (I2 of inst) with 1st opr (in stg) & place result into 1st opr location.	Prot (S) Adr	0 : Result = 0 1 : Result ≠0
AND	NR	14	RR	R1 R2	AND 1st opr (in GPR per R1) with 2nd opr (in GPR per R2) & place result into 1st opr location. Left to right, byte by byte.	None	0 : Result = 0 1 : Result ≠0
Compare Logical	CL	55	RX	R1 D2(X2, B2)	Binarily compare 1st opr (in GPR per R1) with 2nd opr (in stg) & set CC according to result. 1. Left to right, byte by byte. 2. Terminate on inequality or end of fields.	Prot (F) Adr Spec	0 : Opr 1 = Opr 2 1 : Opr 1 < Opr 2 2 : Opr 1 > Opr 2
Compare Logical	CLC	D5	SS	D1(L, B1) D2(B2)	Binarily compare 1st opr (in stg) with 2nd opr (in stg) & set CC according to result. 1. Left to right, byte by byte. 2. Max number of bytes is 256. 3. Terminate on inequality or end of fields.	Prot (F) Adr	0 : Opr 1 = Opr 2 1 : Opr 1 < Opr 2 2 : Opr 1 > Opr 2
Compare Logical	CLI	95	SI	D1(B1) I2	Binarily compare 1st opr (in stg) with immediate opr (I2 of inst) & set CC according to result. 1. Left to right. 2. Terminate on inequality or end of fields.	Prot (F) Adr	0 : Opr 1 = Opr 2 1 : Opr 1 < Opr 2 2 : Opr 1 > Opr 2
Compare Logical	CLR	15	RR	R1 R2	Binarily compare 1st opr (in GPR per R1) with 2nd opr (in GPR per R2) & set CC according to result. 1. Left to right, byte by byte. 2. Terminate on inequality or end of fields.	None	0 : Opr 1 = Opr 2 1 : Opr 1 < Opr 2 2 : Opr 1 > Opr 2
Edit	ED	DE	SS	D1(L, B1) D2(B2)	Change format of source (2nd opr; in stg) from packed to zoned, edit source under control of pattern (1st opr; in stg), & place result into 1st opr location. 1. Left to right, byte by byte. 2. Max number of bytes is 256.	Prot (S,F) Adr Data	0 : Result = 0 1 : Result < 0 2 : Result > 0
Edit & Mark	EDMK	DF	SS	D1(L, B1) D2(B2)	Change format of source (2nd opr; in stg) from packed to zoned, edit source under control of pattern (1st opr; in stg), place result into 1st opr location, & place location of each 1st significant result digit into GPR1. 1. Left to right, byte by byte. 2. Max number of bytes is 256.	Prot (S,F) Adr Data	0 : Result = 0 1 : Result < 0 2 : Result > 0
Exclusive OR	X	57	RX	R1 D2(X2, B2)	Exclusive-OR 1st opr (in GPR per R1) with 2nd opr (in stg) & place result into 1st opr location. Left to right, byte by byte.	Prot (F) Adr Spec	0 : Result = 0 1 : Result ≠0
Exclusive OR	XC	D7	SS	D1(L, B1) D2(B2)	Exclusive-OR 1st opr (in stg) with 2nd opr (in stg) & place result into 1st opr location. 1. Left to right, byte by byte. 2. Max number of bytes is 256.	Prot (S,F) Adr	0 : Result = 0 1 : Result ≠0
Exclusive OR	XI	97	SI	D1(B1) I2	Exclusive-OR immediate opr (I2 of inst) with 1st opr (in stg) & place result into 1st opr location.	Prot (S) Adr	0 : Result = 0 1 : Result ≠0
Exclusive OR	XR	17	RR	R1 R2	Exclusive-OR 1st opr (in GPR per R1) with 2nd opr (in GPR per R2) & place result into 1st opr location. Left to right, byte by byte.	None	0 : Result = 0 1 : Result ≠0

Table 1-11. Logical Instructions (Cont)

Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Insert Character	IC	43	RX	R1 D2(X2, B2)	Insert 2nd opr (byte; in stg) into bits 24–31 of 1st opr location (in GPR per R1). Remaining bits in GPR are unchanged.	Prot (F) Adr	Unchanged
Load Address	LA	41	RX	R1 D2(X2, B2)	Insert 2nd opr adr into bits 8–31 of GPR specified by R1. 1. Bits 0–7 in GPR are made 0's. 2. 2nd opr is not fetched from stg.	None	Unchanged
Move	MVC	D2	SS	D1(L, B1) D2(B2)	Place 2nd opr (in stg) into 1st opr location (in stg). 1. Left to right, byte by byte. 2. Max number of bytes is 256. 3. Move operation can be high or low speed.	Prot (S,F) Adr	Unchanged
Move	MVI	92	SI	D1(B1) I2	Place immediate opr (I2 of inst) into 1st opr location (in stg).	Prot (S) Adr	Unchanged
Move Numerics	MVN	D1	SS	D1(L, B1) D2(B2)	Place numeric portion (low-order 4 bits) of each byte of 2nd opr (in stg) into low-order 4 bits of corresponding byte of 1st opr (in stg). 1. Left to right, byte by byte. 2. Max number of bytes is 256. 3. Zones (high-order 4 bits) in both opr's are unchanged. 4. No restriction on overlapping fields.	Prot (S,F) Adr	Unchanged
Move Zones	MVZ	D3	SS	D1(L, B1) D2(B2)	Place zone portion (high-order 4 bits) of each byte of 2nd opr (in stg) into high-order 4 bits of corresponding byte of 1st opr (in stg). 1. Left to right, byte by byte. 2. Max number of bytes is 256. 3. Numerics (low-order 4 bits) in both opr's are unchanged. 4. No restriction on overlapping fields.	Prot (S,F) Adr	Unchanged
OR	O	56	RX	R1 D2(X2, B2)	OR 1st opr (in GPR per R1) with 2nd opr (in stg) & place result into 1st opr location. Left to right, byte by byte.	Prot (F) Adr Spec	0 : Result = 0 1 : Result ≠ 0
OR	OC	D6	SS	D1(L, B1) D2(B2)	OR 1st opr (in stg) with 2nd opr (in stg) & place result into 1st opr location. 1. Left to right, byte by byte. 2. Max number of bytes is 256.	Prot (S,F) Adr	0 : Result = 0 1 : Result ≠ 0
OR	OI	96	SI	D1(B1) I2	OR immediate opr (I2 of inst) with 1st opr (in stg) & place result into 1st opr location.	Prot (S) Adr	0 : Result = 0 1 : Result ≠ 0
OR	OR	16	RR	R1 R2	OR 1st opr (in GPR per R1) with 2nd opr (in GPR per R2) & place result into 1st opr location. Left to right, byte by byte.	None	0 : Result = 0 1 : Result ≠ 0
Shift Left Double Logical	SLDL	8D	RS	R1 D2(B2)	Shift 1st opr (in GPR per R1 & R1 + 1) left number of bit positions specified by low-order 6 bits of 2nd opr adr. 1. R1 must be even adr. 2. High-order bits of 1st opr are shifted out & lost; vacated low-order bits are made 0's.	Spec	Unchanged
Shift Left Single Logical	SLL	89	RS	R1 D2(B2)	Shift 1st opr (in GPR per R1) left number of bit positions specified by low-order 6 bits of 2nd opr adr. High-order bits of 1st opr are shifted out & lost; vacated low-order bits are made 0's.	None	Unchanged
Shift Right Double Logical	SRDL	8C	RS	R1 D2(B2)	Shift 1st opr (in GPR per R1 & R1 + 1) right number of bit positions specified by low-order 6 bits of 2nd opr adr. 1. R1 must be even adr. 2. Low-order bits of 1st opr are shifted out & lost; vacated high-order bits are made 0's.	Spec	Unchanged

Table 1-11. Logical Instructions (Cont)

Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Shift Right Single Logical	SRL	88	RS	R1 D2(B2)	Shift 1st opr (in GPR per R1) right number of bit positions specified by low-order 6 bits of 2nd opr adr. Low-order bits of 1st opr are shifted out & lost; vacated high-order bits are made 0's.	None	Unchanged
Store Character	STC	42	RX	R1 D2(X2, B2)	Store bits 24–31 of 1st opr (in GPR per R1) into 2nd opr location (in stg).	Prot (S) Adr	Unchanged
Test Under Mask	TM	91	SI	D1(B1) I2	Set CC according to state of 1st opr bits (in stg) selected by mask bits (I2 of inst). 1. If mask bit = 1, test corresponding 1st opr bit; if mask bit = 0, ignore corresponding 1st opr bit. 2. Character in stg is unchanged.	Prot (F) Adr	0 : Selected bits all 0's (mask is all 0's) 1 : Selected bits mixed 0's & 1's 3 : Selected bits all 1's
Translate	TR	DC	SS	D1(L, B1) D2(B2)	Add 1st opr byte (argument; in stg) to effective 2nd opr adr, use result as stg adr, & place function byte from resulting stg adr into corresponding 1st opr byte location. 1. Effective 2nd opr adr = contents of GPR adr by B2, + D2. 2. LL = number of bytes to be translated. 3. 1st opr bytes are processed left to right.	Prot (S,F) Adr	Unchanged
Translate & Test	TRT	DD	SS	D1(L, B1) D2(B2)	Add 1st opr byte (argument; in stg) to effective 2nd opr adr, use result as stg adr, & test function byte from resulting stg adr. If 0, translate & test next argument byte; if non-0, complete operation by inserting related argument adr into GPR1 & function byte into GPR2. 1. Effective 2nd opr adr = contents of GPR adr by B2, + D2. 2. LL = number of bytes to be translated. 3. 1st opr bytes are processed left to right. 4. Set CC according to ending condition.	Prot (F) Adr	0 : All bytes tested are all 0's 1 : Non-0 byte found before last byte to be tested 2 : Non-0 byte found as last byte to be tested

In the RR, RX, and RS formats, the contents of the GPR specified by R1 are called the first operand. In the SI and SS formats, the contents of the GPR specified by B1 are added to the contents of the D1 field to form an address. This address designates the leftmost byte of the first operand field. The number of bytes to the right of this first byte is specified by the LL field in the SS instruction. In the SI format, the operand size is one byte.

In the RR format, the R2 field specifies the GPR containing the second operand. The same GPR may be specified for the first and second operands.

In the RX format, the contents of the GPR's specified by the X2 and B2 fields are added to the contents of the D2 field to form the address of the second operand in main storage.

In the RS format, used for shift operations, the contents of the GPR specified by the B2 field are added to the contents of the D2 field. This sum is not used as an address but the low-order six bits specify the number of bits of the shift. The R3 field is ignored in the shift operations.

In the SI format, the second operand is the eight-bit immediate data field, I2, of the instruction.

In the SS format, the contents of the GPR specified by B2 are added to the contents of the D2 field to form the address of the second operand. The second operand field has the same length as the first operand field.

A 0 in the X2, B1, or B2 field indicates the absence of the corresponding address or shift-amount components. An instruction can specify the same GPR both for address modification and for operand location. Address modification is always completed before operation execution.

Data Flow

Data paths used by the logical instructions are identical to those used by the decimal instructions, with one exception (Diagram 3-5, FEMDM). For decimal instructions, E(8–15) is divided into L1 and L2 fields. For logical instructions, E(8–15) is one field (LL).

The logical instructions operate on data which may range from 1 to 256 bytes in length. The operands are obtained either from the main storage or from a GPR. Sometimes, the operand may be contained in the instruction itself.

Processing of data in main storage proceeds from the high-order to the low-order address, or from left to right. The initial byte selected for processing may be at either an odd or even main storage address. As a rule, processing of

data in a GPR involves the complete register contents. Except for the editing instructions, data is not treated as numbers.

Generally, the operands are treated as eight-bit bytes. In a few cases, the left or right four bits of a byte are treated separately or operands are shifted a bit at a time.

Results replace the first operand, except in the Store Character instruction, where the result replaces the second operand. A variable-length result is never stored outside the field specified by the address and length.

The contents of all GPR's and storage locations participating in the addressing or execution of an operation generally remain unchanged. Exceptions are the move instructions, and the result locations, GPR1 in the Edit and Mark instruction, and GPR's 1 and 2 in the Translate and Test instruction.

Editing operations provide transformation from packed decimal digits to alphanumeric characters; i.e., editing requires a packed decimal field and generates zoned decimal digits. The digits, signs, and zones are recognized and generated as for decimal arithmetic; all bit configurations are considered valid.

The translating operations use a list of arbitrary values. A list provides a relation between an argument (the quantity used to reference the list) and the function (the contents of the location related to the argument). The purpose of the translation may be to convert data from one code to another code or to perform a control function. The list is specified by an initial address, the address designating the leftmost byte location of the list. The byte from the operand to be translated is the argument. The address used to address the list is obtained by adding the argument to the low-order positions of the initial address. As a consequence, the list contains 256 eight-bit function bytes. Where it is known that not all eight-bit argument values will occur, it may be possible to reduce the size of the list.

Use of GPR1 is implied in Edit and Mark and in Translate and Test instructions. A 24-bit address may be placed into this register during these operations. The Translate and Test instruction also implies GPR2. The low-order eight bits of GPR2 may be replaced by a function byte during a translate-and-test operation.

Program Interruptions

Four program interruptions can occur during execution of logical instructions:

1. Protection. The storage key of a result location in main storage does not match the protection key in the PSW. The operation is suppressed on a store violation. The only exceptions are the variable length storage-to-storage operations, which are terminated. The operation is terminated on a fetch violation.
2. Addressing. An address designates a location outside the available storage for the installed system. In most cases, the operation is terminated. The exceptions are the AND, Exclusive-OR, OR, and Move instructions that have the SI format, and the Store Character instruction. These instructions are suppressed. Operand addresses are tested only when used to address storage. Addresses used as a shift amount are not tested. Similarly, the address generated by the use of the Load Address instruction is not tested. The address restrictions do not apply to the contents of the D1 and D2 fields, or to the contents of the GPR's specified by X2, B1, and B2.
3. Specification. A full-word operand in a storage-to-register operation is not located on a 32-bit boundary, or an odd register address is specified for a pair of GPR's containing a 64-bit operand. The operation is suppressed.
4. Data. A digit code of the second operand in the Edit or Edit and Mark instruction is invalid. The operation is terminated.

Condition Codes

The results of most logical operations set the CC in the PSW (Table 1-11). The Load Address, Insert Character, Store Character, Translate, and the moving and shift instructions leave this code unchanged.

The CC can be set to reflect five types of results for logical operations. For the Compare Logical instructions, the 0, 1, and 2 states indicate that the first operand is equal to, less than, or greater than the second operand, respectively.

For the logical AND, OR, and Exclusive-OR instructions, the states 0 and 1 indicate a zero or nonzero result field, respectively.

For the Test under Mask instruction, the states 0, 1, and 3 indicate that the selected bits are all-zero, mixed zero and 1, or all-1, respectively.

For the Translate and Test instruction, the states 0, 1, and 2 indicate an all-zero function byte, a nonzero function byte with the operand incompletely tested, or a last function byte nonzero, respectively.

For editing, the states 0, 1, and 2 indicate a zero, less-than zero, or greater-than-zero content of the last result field, respectively.

Branching Instructions

- Branching causes departure from normal instruction sequencing.
- Branch address is introduced as next sequential address.
- Branch address is obtained from GPR or specified as 2nd operand address.
- Branch may be conditional or unconditional.

- Conditional branches (may or may not use branch address):
  Branch on condition
  Branch on count
  Branch on index
- Unconditional branches (always use branch address):
  Branch and link
  Execute
- On branch, normal storage request per IC to fill Q is blocked; branch logic will make request for IC if the branch is unsuccessful and Q needs to be refilled.
- If branch is unsuccessful, Q is refilled if required.

Normally, the CPU is controlled by instructions taken in sequential order. That is, an instruction is fetched from a main storage location specified by the instruction address in the IC. The address is then increased by the number of bytes needed to address the next instruction in sequence, and this updated address replaces the old address in the IC. The current instruction is executed, and the same steps are repeated using the updated instruction address to fetch the next instruction.

A departure from the normal instruction sequence occurs when branching is performed. A branch address is introduced as the next instruction address. This branch address may be obtained from one of the GPR's or it may be the second operand address specified by a particular instruction. Depending upon the format and the instruction, branching may be either conditional or unconditional. The conditional branches are branch on condition, branch on count, and branch on index. The unconditional branches are branch and link and execute. Conditional branches may or may not use the branch address. If the branch is successful (that is, the branch is taken), the branch address is used and the storage request issued per the IC during I-Fetch is blocked. If the branch is unsuccessful, the instruction address in the IC is used to fill Q. Unconditional branches are always taken and use the branch address.

Whether a conditional branch is successful depends upon the result of operations concurrent with the branch or preceding the branch. The first case is represented by the branch on count and branch on index instructions. The second case is represented by the branch on condition instructions, which inspect the CC that reflects the result of a previous arithmetic, logical, or I/O operation.

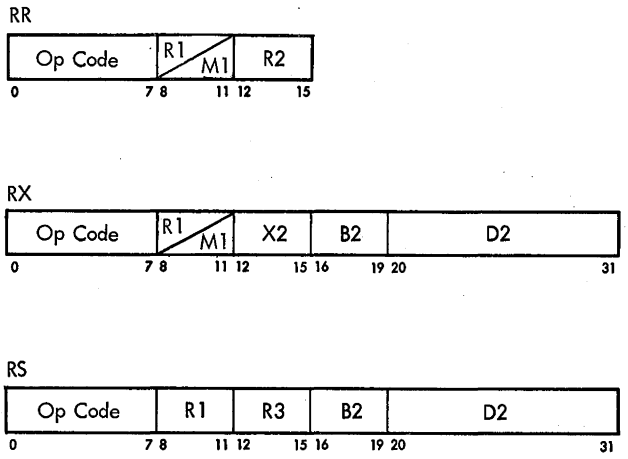
Branching is used to reference a subroutine, to resolve a two-way choice, or to repeat a portion of a program. To save time and increase the speed of the operating program, branching is always considered to be successful unless proven otherwise. (The branch conditions for branch on condition instructions is tested during I-Fetch for a successful or unsuccessful branch, and a D or IC request is issued dependent upon this test.) Therefore, whenever a branch instruction is decoded during I-Fetch, the next instruction address is the branch address located in D. If the branch is found to be unsuccessful (determined during execution of the branch- instruction), the instruction address from D is ignored, and the correct instruction address is obtained from the IC.

There are two methods of performing an end-op cycle in the branch operations: normal end op and branch end op. The normal end-op cycle allows decoding of the next instruction format from R and of the instruction address from the IC, and is normally used when ending an operation. Decoding off R is possible because the data placed into the register has become stable by the time the end-op cycle begins. The branch end-op cycle, on the other hand, allows decoding of the next instruction format from the SDBO and of the instruction address from D. This end-op cycle is used when the data, which has been placed into R, is not yet stable and is some halfword other than the last halfword of Q. Decoding from the SDBO saves the time it takes for the data to stabilize in R and the instruction address to stabilize in the IC.

Table 1-12 lists the branching instructions.

Instruction Formats

Branching instructions use the RR, RX, and RS formats:



In the formats shown above, bits 8–11 are normally the R1 field that specifies the address of a GPR containing the first operand. In the branch on condition instruction, however, bits 8–11 are designated as M1 and contain mask bits used in conjunction with the PSW CC to determine whether the branch is successful.

In the RR format, the R2 field specifies the address of a GPR that contains the branch address, except when R2 = 0, in which case no branching is to take place.

In the RX format, the contents of the GPR's specified by the X2 and B2 fields are added to the D2 field to form the branch address.

Table 1-12. Branching Instructions

Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Branch & Link	BAL	45	RX	R1 D2(X2, B2)	Store PSW(32–63), link information, into GPR (adr by R1) & branch to location specified by 2nd opr adr. 1. Branch is unconditional. 2. Link information is stored whether or not branch is successful.	Prot (F)†	Unchanged
Branch & Link	BALR	05	RR	R1 R2	Store PSW(32–63), link information, into GPR (adr by R1) & branch to location specified by GPR (adr by R2). 1. Branch is unsuccessful if R2 = 0; use next sequential instr adr. 2. Link information is stored whether or not branch is successful.	Prot (F)†	Unchanged
Branch on Condition	BC	47	RX	M1 D2(X2, B2)	Branch to location specified by 2nd opr adr if state of CC is as specified by M1. 1. Branch is unconditional if M1 is all 1's. 2. Branch is unsuccessful if M1 is all 0's; use next sequential instr adr.	Prot (F)†	Unchanged
Branch on Condition	BCR	07	RR	M1 R2	Branch to location specified by GPR (adr by R2) if state of CC is as specified by M1. 1. Branch is unconditional if M1 is all 1's and R2 $\neq$ 0. 2. Branch is unsuccessful if R2 = 0 or if M1 is all 0's; use next sequential instr adr.	Prot (F)†	Unchanged
Branch on Count	BCT	46	RX	R1 D2(X2, B2)	Subtract 1 from 1st opr (in GPR per R1); if result $\neq$ 0, branch to location specified by 2nd opr adr. 1. Place result of subtraction into 1st opr location. 2. Branch is unsuccessful if result = 0; use next sequential instr adr. 3. If 1st opr = 1, no branching occurs.	Prot (F)†	Unchanged
Branch on Count	BCTR	06	RR	R1 R2	Subtract 1 from 1st opr (in GPR per R1); if result $\neq$ 0, branch to location specified by GPR (adr by R2). 1. Place result of subtraction into 1st opr location. 2. Branch is unsuccessful if result = 0 or if R2 = 0; use next sequential instr adr. 3. If 1st opr = 1, no branching occurs.	Prot (F)†	Unchanged
Branch on Index High	BXH	86	RS	R1 R3 D2(B2)	Add increment (3rd opr; in GPR per R3) to 1st opr (in GPR per R1), algebraically compare result (index) with comparand (in odd-adr GPR specified by R3 or R3 + 1); if index $>$ comparand, branch to location specified by 2nd opr adr. 1. Place index into 1st opr location. 2. Branch is unsuccessful if index = or $<$ comparand; use next sequential instr adr.	Prot (F)†	Unchanged
Branch on Index Low or Equal	BXLE	87	RS	R1 R3 D2(B2)	Add increment (3rd opr; in GPR per R3) to 1st opr (in GPR per R1), algebraically compare result (index) with comparand (in odd-adr GPR specified by R3 or R3 + 1); if index = or $<$ comparand, branch to location specified by 2nd opr adr. 1. Place index into 1st opr location. 2. Branch is unsuccessful if index $>$ comparand; use next sequential instr adr.	Prot (F)†	Unchanged
Execute	EX	44	RX	R1 D2(X2, B2)	Execute subject instr at location specified by 2nd opr adr. Subject instr may be modified by 1st opr (in GPR per R1) if E(8–11) $\neq$ 0. Modification is achieved by OR'ing bits 8–15 of subject instr with bits 24–31 of 1st opr; if R1 = 0, no modification takes place.	Execute Prot (F) Adr Spec	Set by subject instr

† Fetch protected: bit 4 of storage protect set.

In the RS format, which is used in branch on index operations, the contents of the GPR specified by the B2 field are added to the D2 field to form the branch address. The R3 field specifies the address in LS of an increment value (third operand) which is added to the first operand to determine the index value. R3, if odd, also is the comparand; if R3 is even, R3 + 1 is the comparand.

Data Flow

Diagram 3-6, FEMDM, is a diagram of the basic data flow for the branching instructions. The main functional units used to determine addresses and instructions in the branching operations are Q, R, E, D, and the IC. The secondary functional units, T, AB, parallel adder, STC, and ABC, determine whether the branch is successful when the branch being executed is a conditional branch. The purpose of each functional unit is as follows:

1. Q. Holds the doubleword that contains the instruction addressed by the branch instruction if the branch is successful.
2. R. Contains the instruction to be performed after execution of the branch instruction.
3. E. Contains the branch instruction presently being executed.
4. D. Holds the address of the doubleword which, if the branch is successful, contains the next instruction to be executed.
5. IC. Holds the address of the doubleword which, if the branch is unsuccessful, contains the next instruction to be executed.
6. T. Buffers the operand being tested and operated on.
7. AB. Holds the first operand when added to some other value to determine whether the branch is successful.
8. Parallel adder. Determines whether conditions have been met when a conditional branch is being executed.
9. STC. Allows transfer of last byte of T during an Execute instruction when modifying the subject instruction of the Execute instruction.
10. ABC. Selects data being modified in the subject instruction during an Execute instruction.

Program Interruptions

Four program interruptions can occur during execution of branching instructions:

1. Execute. The subject instruction of an Execute instruction is another Execute instruction. The operation is suppressed.
2. Protection. The branch address of an Execute instruction is protected. The branch-to address of any branch instruction may be fetch-protected. In this case, the PSW key must match the storage key or must be a master key of 0. The operation is suppressed.
3. Addressing. The branch address of an Execute instruction designates an instruction-halfword location outside the available storage area. The operation is suppressed.
4. Specification. The branch address of an Execute instruction is odd. The operation is suppressed.

Condition Codes

The branching instructions leave the CC unchanged, except for the Execute instruction. If the CC is set during the Execute instruction, it is set by the subject instruction.

Status Switching Instructions

- Load PSW, Set Program Mask, Set System Mask, and Supervisor Call instructions control status of CPU.
- Set Storage Key, Insert Storage Key, and Test and Set instructions control status of data in main storage.
- Write Direct and Read Direct instructions control status of external device (also transfer data bytes).
- Diagnose instruction controls status of CPU and channels.

The status switching instructions can change the status of the CPU, the channels, the external device, and the data in main storage. The status of a unit may also be changed by manual intervention and by interruptions (described elsewhere in this manual). The overall status of the CPU is determined by the current PSW and associated logic. (For a discussion of the PSW and of the eight CPU program states, refer to Section 3 of this chapter.) Any field in the current PSW may be changed directly by the Load PSW instruction, if the CPU is in the Supervisor state. Thus, the Load PSW instruction may be used to switch from the Supervisor state to the Problem state, between the Wait and Running states, and between the Masked and Interruptable states. At any time, the Set Program Mask instruction may be used to switch any of the four program mask bits between the Masked and Interruptable states. When in the Supervisor state, the Set System Mask instruction may be used to switch any of the eight system mask bits between the Masked and Interruptable states. The Supervisor Call instruction allows a problem program to switch the CPU from the Problem state to the Supervisor state; simultaneously, a byte of information is passed to the supervisor program via the interrupt code of the Supervisor Call old PSW.

Three instructions control the protection status of data in main storage. The Set Storage Key and Insert Storage Key instructions are privileged instructions for controlling the protection status of main storage data in 2048-byte blocks. The Set Storage Key instruction changes the storage protection keys in main storage. The Insert Storage Key instruction fetches the keys from main storage for inspection by the program. The Test and Set instruction, on the other hand, may be used in either the Supervisor or Problem state for protecting data in main storage in blocks of any length; this application is described in IBM Systems Reference Library, *IBM System/360 Principles of Operation,* Form A22-6821.

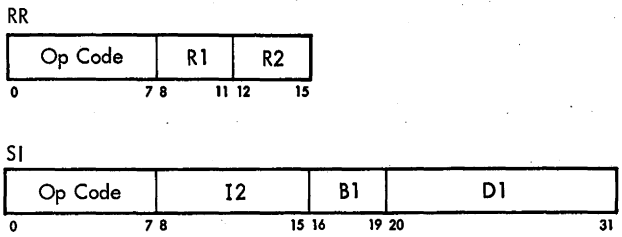
The Write Direct and Read Direct instructions, which are part of the Direct Control feature, may be used to switch the status of an external device by means of a code in the I2 field. If the external device is another System/360, the Write Direct and Read Direct instructions can be used to externally interrupt the receiving CPU. For example, assume CPU 1 and CPU 2 are both operating and have their Direct Control Features enabled (Diagram 5-607, Sheet 2, FEMDM). CPU 1 requests data from CPU 2 by executing a Read Direct instruction, sending an external interruption code to CPU 2 on the 'timing signal bus out' lines. If CPU 2 is masked on for external interruptions, if no other interruptions or exceptional conditions of higher priority are pending, and as soon as the instruction being processed is finished, CPU 2 is externally interrupted. After CPU 2's program decodes the interruption code and determines the operation to be performed, it executes a Write Direct instruction, sending its external interruption code and a 'direct control write out' signal to CPU 1 and putting the requested data on the 'direct control bus out' lines. CPU 1 can now be similarly interrupted and again execute a Read Direct instruction (but this time with its I2 field clear). CPU 1 waits until the 'direct control write out' signal (received as a 'direct control hold in' signal) is removed and then transfers the requested data from the 'direct control bus in' lines to main storage.

The Diagnose instruction controls the status of the CPU and the channels. Unlike the Load PSW, Set Program Mask, Set System Mask, and Supervisor Call instructions that switch the CPU's status by changing the current PSW, the Diagnose instruction switches the CPU's status by setting control triggers (such as 'defeat interleave', 'emulation mode', and 'diagnose FLT') through the use of a maintenance control word. The Diagnose instruction may also be used to switch the channels between a normal operating mode and a test mode.

Table 1-13 lists the status switching instructions.

Instruction Formats

Status switching instructions have two formats:



In the RR format, the R1 and R2 fields specify GPR's except when used in the Supervisor Call instruction. The R1 and R2 fields in the Supervisor Call instruction are replaced by an I-field which contains an eight-bit interruption code. In the Set Program Mask instruction, the R2 field is ignored.

In the SI format, the I2 field is ignored for the Load PSW, Set System Mask, and Test and Set instructions. In the Write Direct and Read Direct instructions, the I2 field contains a timing signal code that is sent to an external device. In the Diagnose instruction, the I2 field contains a code for controlling certain maintenance aids and an optional Compatibility feature. The contents of the GPR specified by the B1 field are added to D1 to form a main storage address of an operand to be fetched by the instruction specified, except for Read Direct. The Read Direct instruction uses the address derived for storing data from an external device. Only one storage address is required in status switching operations. A 0 in the B1 field indicates the absence of the base address component.

Data Flow

- Each status switching instruction has different data flow.
- ST is used by most instructions as buffer before final data transfer.

The status switching instructions transfer data from one unit to another; except for the Insert Storage Key instruction, there is no intermediate processing. Depending on the instruction, the data is obtained from LS, main storage, or the 'direct control bus in' lines and is transferred to either LS, main storage, CPU control triggers, or the 'direct control bus out' lines. A generalized data flow is shown in Diagram 3-7, FEMDM. The following is a list of the functional units and their purposes.

1. BCU. Primarily used for 3-cycle fetches of storage operands per D. During the Load PSW instruction, a 3-cycle fetch per the IC is made for the next instruction after the new PSW has been loaded into the CPU. For the Set Storage Key, Insert Storage Key, and Test and Set instructions, the BCU performs a 4-cycle set-key operation per D, a 3-cycle insert-key operation per D, and a 3-cycle test-and-set operation per D, respectively. During the Read Direct instruction, a 4-cycle store operation per D is made.
2. Q. Holds the doubleword containing the instruction being executed. It may also hold the next sequential doubleword if a Q-refill operation occurred during I-Fetch. The Load PSW instruction refills Q with the next instruction regardless of its storage location.
3. R. Contains the instruction to be performed after execution of the status switching instruction.
4. E. Contains the status switching instruction (or the first 16 bits of the instruction) being executed. E(0–7) contains the R code for the Write Direct, Read Direct, and Diagnose instructions, and E(8–15) contains an immediate operand. For the Supervisor Call instruction, E(8–15) contains a supervisor call interruption code. For the Insert Storage Key instruction, E(8–11) contains the address of the GPR into which the protection key is to be inserted.

Table 1-13. Status Switching Instructions

Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Diagnose	None	83	SI	D1(B1) I2	Load word designated by stg opr adr into MCW, set or reset certain control triggers, & branch to ROS adr specified by MCW.	Priv Oper Prot (S,F) Adr Spec	Unpredictable
Insert Storage Key	ISK	09	RR	R1 R2	Insert stg protection key for 2048-byte stg block, adr by bits 8–20 of 2nd opr (in GPR per R2), into bits 24–28 of 1st opr (in GPR per R1). 1. 1st opr: bits 0–23 are unchanged; bits 29–31 are cleared. 2. 2nd opr: bits 0–7 & 21–27 are ignored; bits 28–31 must = 0's. 3. Key is fetched twice because of 2-way interleaving.	Priv Oper Adr Spec	Unchanged
Load PSW	LPSW	82	SI	D1(B1)	Load doubleword stg opr (designated by stg opr adr) into CPU, thus replacing current PSW, & branch to new instr sequence. 1. Bits 0–15: system mask, protection key, program state. Bits 16–33: ignored. Bits 34–39: CC, program mask. Bits 40–63: instr adr. 2. If PSW(14) = 1, enter Wait state. 3. If PSW(15) = 1, enter Problem state. 4. Load PSW instr is only instr available for entering Problem or Wait state.	Priv Oper Prot (F) Adr Spec	Set by new PSW bits 34 & 35
Read Direct	RDD	85	SI	D1(B1) I2	Send 'direct control read out' signal & timing signal code (I2; in instr) to external device for about 0.6 usec; store 1 data byte from external device into stg (per stg opr adr) when 'direct control hold in' signal is absent.	Oper Priv Oper Prot (S) Adr	Unchanged
Set Program Mask	SPM	04	RR	R1	Replace CC & program mask (bits 34–39) of current PSW with bits 2–7 of 1st opr (in GPR per R1).	None	Set by opr 1 bits 2 & 3
Set Storage Key	SSK	08	RR	R1 R2	Set stg key (bits 24–28 of 1st opr; in GPR per R1) for 2048-byte stg block (adr by bits 8–20 of 2nd opr; in GPR per R2) into stg protection logic in main storage. 1. 1st opr: bits 0–23 & 29–31 are ignored. 2. 2nd opr: bits 0–7 & 21–27 are ignored; bits 28–31 must = 0's. 3. Key is set twice because of 2-way interleaving.	Priv Oper Adr Spec	Unchanged
Set System Mask	SSM	80	SI	D1(B1)	Replace system mask (bits 0–7) of current PSW with byte from location designated by stg opr adr.	Priv Oper Prot (F) Adr Multisys	Unchanged
Supervisor Call	SVC	0A	RR	I	Cause supervisor call interruption; replace old PSW(24–31) with I-field (bits 8–15) of instr, providing interruption code. 1. Clear PSW(16–23). 2. Store old PSW at stg location 32 (decimal). 3. Fetch new PSW from stg location 96 (decimal).	None	Unchanged
Test & Set	TS	93	SI	D1(B1)	Test high-order bit (bit 0) of stg opr byte (in stg), set CC according to state of tested bit, & set addressed byte back into stg as all 1's.	Prot (S,F) Adr	0 : High-order bit = 0 1 : High-order bit = 1
Write Direct	WRD	84	SI	D1(B1) I2	Send 'direct control write out' signal & timing signal code (I2; in instr) to external device for about 0.8 usec; make 1 data byte from stg (per stg opr adr) available to external device until next WRD is executed.	Oper Priv Oper Prot (F) Adr	Unchanged

5. D. Contains the main storage address for storage requests issued during execution of the Set Storage Key, Insert Storage Key, Read Direct, and Test and Set instructions. This register also selects the byte to be used in the Set System Mask and Write Direct instructions, and selects the halfword containing the instruction to be executed after the Load PSW instruction.
6. IC. Contains the main storage address of the next instruction during execution of the Load PSW instruction.
7. AB. Buffers operands for the serial adder and the parallel adder. During I-Fetch of the Set Program Mask, Set Storage Key, and Insert Storage Key instructions, the first operand is placed here. During the Set System Mask, Diagnose, and Test and Set instructions, doublewords from storage are received here.
8. ST. Buffers operands for the serial adder and the parallel adder. Data is received here from main storage for the Load PSW instruction and from LS for the Set Storage Key and Insert Storage Key instructions. Data is stored from here into main storage during the Read Direct instruction and from LS during the Insert Storage Key instruction. The Load PSW, Set Program Mask, Set System Mask, and Test and Set instructions cause all or part of the PSW register to be changed per ST. The Diagnose instruction causes the MCW register, scan counters, and ROSAR to be changed per ST.
9. ABC and STC. Controls selection of data from and placement of data into AB and ST, respectively. Also, during the Read Direct and Test and Set instructions, STC sets a mark trigger.
10. Mark. Identifies the byte to be used by main storage during the Read Direct and Test and Set instructions. All mark triggers are set during the Set Storage Key instruction by a ROS micro-order.
11. F. Buffers the storage key before it is placed into main storage during the Set Storage Key instruction and after it is taken from main storage during the Insert Storage Key instruction. This register also buffers data received from an external device during the Read Direct instruction.
12. G. Buffers a byte of data being sent to an external device when executing a Write Direct instruction.
13. PSW register. Contains a portion of the current PSW. All or part of the PSW register contents is changed directly by the Load PSW, Set Program Mask, Set System Mask, and Test and Set instructions. Because the Supervisor Call, Write Direct, Read Direct, and Diagnose instructions may cause an interruption after being executed, they may indirectly change all of the PSW register contents.
14. MCW register. Controls CPU or channel diagnostic functions during and after execution of the Diagnose instruction.
15. Parallel adder. Provides the data transfer path between AB, ST, D, and the IC. Adds 8 to the IC and D for address updating. Subtracts 8 from A during the Set Storage Key and Insert Storage Key instructions so that a re-entrant loop may be constructed. Calculates IC - D + 7 for the address store compare tests made during Read Direct, Diagnose, and Test and Set instructions.
16. Serial adder. Provides the data transfer path from AB to ST and G; also, during the Read Direct instruction, assigns odd parity to the data byte transferred from F to ST. During the Insert Storage Key instruction, the contents of F are logically OR'ed with the contents of T via the serial adder.
17. LS. Contains operands required by the Set Program Mask, Set Storage Key, and Insert Storage Key instructions. Only the Insert Storage Key instruction transfers data into LS.

Program Interruptions

Six program interruptions can occur during execution of status switching instructions:

1. Operation. Occurs if the Direct Control feature is not installed or not enabled and the instruction being executed is either Read Direct or Write Direct. The operation is suppressed.
2. Priviliged Operation. Occurs if a Load PSW, Set System Mask, Set Storage Key, Insert Storage Key, Write Direct, Read Direct, or Diagnose instruction is encountered while the CPU is in the Problem state. The operation is suppressed.
3. Protection. Occurs if the storage key of the location designated by the instruction does not match the protection key in the current PSW. The instruction is suppressed on a store violation, except for the Read Direct and Test and Set instructions, which are terminated. The operation is terminated on a fetch violation.
4. Addressing. Occurs if an address designates a location outside the available main storage. The operation is terminated, except for the Diagnose instruction, which is suppressed.
5. Specification. Occurs if (1) the operand address of a Load PSW or Diagnose instruction does not have 0's in the three low-order bit positions, or (2) the block address specified by the Set Storage Key or Insert Storage Key instruction does not have 0's in the four low-order bit positions. The operation is suppressed.
6. Multisystem (Multisystem feature only). Occurs if Set System Mask instruction is encountered when in Multisystem mode. The operation is suppressed.

Condition Codes

Three status switching instructions affect the condition code: Load PSW, in which the CC is set by new PSW(34,35); Set Program Mask, in which the CC is set by

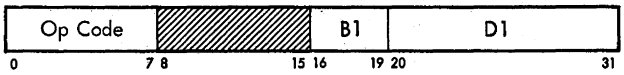
bits 2 and 3 of the first operand; and Test and Set, in which the CC is set to 0 if the high-order bit of the addressed byte in storage equals 0 or set to 1 if the high-order bit equals 1. The remaining instructions leave the CC unchanged, except for the Diagnose instruction in which the CC is unpredictable.

Input/Output Instructions

The 2065 has four I/O instructions: Start I/O, Test I/O, Halt I/O, and Test Channel (Table 1-14).

Instruction Format

The four I/O instructions use the SI format:



Bits 8–15 are ignored. The base plus the displacement determines the channel and I/O unit address: bits 16–23 of the sum are the channel address (of which only bits 21–23 are valid), and bits 24–31 of the sum are the I/O unit address.

Data Flow

Although the CPU operation is essentially the same for all I/O instructions, the Start I/O instruction is used to illustrate the data flow because the channel operation is more extensive. The Halt I/O, Test I/O, and Test Channel instructions do not fetch a CAW or a CCW, do not transfer data, and do not cause an I/O interruption of succeeding CPU operations.

To illustrate the channel operation, the IBM 2860 Selector Channel is used. The Start I/O instruction sets the 'timing gate' trigger, which gates a 'channel select' signal to select 1 of 7 channels (Diagram 3-8, FEMDM). With the 'timing gate' trigger set, the CPU cannot proceed to the next instruction. The I/O unit address is sent to all channels via the 'unit address bus out' (UABO) line, addressing the proper control unit and I/O unit as specified in the instruction. The channel requests the CAW from main storage address 72 (48, hex), and the first command address (address of the CCW) is set into the data address register via the SDBO lines. The channel requests the first CCW from the main storage address specified in the CAW (now stored in the data address register). Parts of the CCW are set into various channel registers to control the I/O operation. During the CCW fetch cycle, the first command address is incremented by eight bytes and is stored into the command address register to be used to fetch the next CCW, if chaining is specified in the first CCW.

Depending upon the availability of the control unit and I/O unit, the I/O operation may be initiated. In any case, the appropriate CC is generated and sent to the CPU together with a 'release' signal, thus resetting the 'timing gate' trigger. The CPU performs an end-op cycle and proceeds to fetch and execute the next instruction.

If the selected control unit and I/O unit are available, the I/O operation is initiated and the channel performs the operation specified by the CCW; the CPU and the channel

Table 1-14. I/O Instructions

Instruction	Mnemonic	Op Code	Format	Operands	Function	Program Interruptions	Condition Code
Halt I/O	HIO	9E	SI	D1(B1)	Terminate current I/O operation at selected channel & I/O unit. 1. D(13–15) is channel adr. 2. D(16–23) is I/O unit adr.	Priv Oper	0 : Interruption in channel 1 : CSW stored 2 : Halted 3 : Unavailable
Start I/O	SIO	9C	SI	D1(B1)	Select specified I/O unit & initiate channel command to that unit. 1. D(13–15) is channel adr. 2. D(16–23) is I/O unit adr. 3. CAW, which specifies address of 1st CCW, is fetched from location 72 (48, hex).	Priv Oper	0 : Available 1 : CSW stored 2 : Working 3 : Unavailable
Test Channel	TCH	9F	SI	D1(B1)	Test state of selected channel & set CC accordingly. 1. D(13–15) is channel adr. 2. D(16–23) is ignored. 3. State of channel is not affected.	Priv Oper	0 : Available 1 : CSW ready 2 : Working 3 : Unavailable
Test I/O	TIO	9D	SI	D1(B1)	Clear interruption condition in addressed channel or associated I/O units, & set CC according to status of addressed channel & I/O units. 1. D(13–15) is channel adr. 2. D(16–23) is I/O unit adr. 3. CSW is stored at location 64 (40, hex) if: a. I/O unit or control unit contains pending interruption. b. I/O unit or control unit is executing previous operation, or there is pending channel-end/control unit-end for another I/O unit. c. I/O unit or its control unit detects machine error.	Priv Oper	0 : Available 1 : CSW stored 2 : Working 3 : Unavailable

continue to share main storage under control of the priority function of the BCU. If an I/O operation was not initiated or was completed, the channel operation is ended when the CC is set into the CPU, and the channel is freed to perform further operations as initiated by the CPU.

For an I/O write operation, a doubleword is fetched from the main storage location specified by the data address in the CCW (now in the data address register), and is stored into the channel A-register. The contents of the channel A-register are transferred to the channel B-register, the data address register is updated, and another doubleword is fetched from main storage and stored into the channel A-register. The contents of the channel B-register are gated to the bus-out register, a byte at a time, and are transmitted over the bus-out lines to the control unit and to the I/O unit as required. When the channel B-register has transferred the last byte, the contents of the channel A-register are transferred to the channel B-register, and another doubleword is fetched from main storage and stored into the channel A-register.

For an I/O read operation, data is received over the bus-in lines, one byte at a time, from the I/O unit via the control unit, and is set into the bus-in register. The contents of the bus-in register are transferred to the appropriate byte location in the channel B-register; when B is full, its contents are transferred to the channel A-register. The contents of A are stored into main storage, a doubleword at a time, according to the address in the data address register.

When the read or write operation is completed or is terminated because of an error condition, the channel requests an I/O interruption of the CPU to present to the CPU the CC and status byte describing the condition of the channel and I/O unit. If the CPU accepts the interruption request, the channel stores a CSW into main storage address 64 (40, hex) and is freed for further operations. Until the CPU accepts the interruption request, the channel remains unavailable to the CPU.

Program Interruption

The only program interruption that may occur for an I/O instruction is the privileged-operation interruption. It occurs if the CPU is in any state other then Supervisor. The instruction is suppressed before the channel is selected. The CSW, the CC in the PSW, and the state of the addressed channel and of the I/O unit remain unchanged. The interruption code in the program old PSW(16–31) is 00000000 00000010.

Condition Codes

When the CPU is released from an I/O instruction, 1 of 4 CC's is set into the CC register of the CPU and becomes a part of the current PSW. This CC is the result of tests by the CPU, the channel, or the I/O unit, and indicates various conditions that exist in the channel, the control unit, or the I/O unit. The CC's for the four I/O instructions are summarized in Table 1-14. For a detailed discussion of the setting of the CC's, refer to the applicable I/O channel FETOM.

POWER

The CPU wall contains a 75-amp, 2.5-kHz converter/inverter and the necessary high-frequency regulator modules to provide the CPU with dc power. The power of just the CPU may be turned on and off by the CPU READY/OFF switch and the CPU ON pushbutton on the enclosed CE panel. Normally, however, CPU power is turned on and off together with the power of the system. The POWER ON and POWER OFF pushbuttons on the system control panel (and the 2150 Console if it is installed) control the power of the system.

Depressing the POWER ON pushbutton causes the units of the system to be turned on one at a time. This sequencing is done so that the higher turn-on current of all the units is not required at the same time. The CPU is turned on first, followed by the channels and, lastly, the main storage units. The units attached to one channel are turned on before the power of the next channel is turned on. The main storage units are turned on last, preventing inadvertent storing by the CPU or channels during their power-on transition before their control triggers have been reset. The system power-on sequence is completed with a system reset operation. The entire system power-on sequence is performed in seconds.

The system power-off sequence turns off the dc power in the main storage units first, followed by the rest of the system. Again, this sequence is followed to prevent inadvertent storing during the power transition while all control triggers may not be turned off. A switch on most of the units in the system (such as CPU READY/OFF or LOCAL/REMOTE) allows the unit to be bypassed during power-off and power-on sequences. In an emergency, the power of the complete system, regardless of the position of these switches, is turned off by pulling the EMERGENCY PULL switch on the system control panel.

To protect the CPU from thermal damage, it is forced-air-cooled by blowers whenever the CPU is turned on; if the internal temperature gets too high, the CPU is turned off automatically. The CPU is also automatically turned off in an orderly, interlocked manner if an overcurrent, overvoltage, or undervoltage condition is detected on the output of any of the CPU's regulators. Whenever the CPU's power is turned off by any means other than the POWER OFF pushbutton, two manual actions are needed to turn the power back on. The intention of this precaution is to prevent inadvertent restoration of power.

Indicators at the top of the CPU's system control panel warn of a power fault in the CPU, channels, and 2365 Processor Storage units. Also located there are controls and a voltmeter for margining the 6V dc supplies in those units and the 18V dc ROS supply in the CPU.

This chapter discusses the functional units composing the 2065 CPU, and is divided into seven sections:

Section 1, Timing and Clock Control.
Section 2, Read-Only Storage.
Section 3, Bus Control Unit.
Section 4, Data and Control Registers.
Section 5, Local Storage.
Section 6, Serial and Parallel Adders.
Section 7, Status and Control Triggers.

Each functional unit is described separately, as concerns operation, operational timing, and functional application. Supporting the descriptions are simplified, positive-logic upper-level diagrams, flowcharts, and timing charts.

Section 1. Timing and Clock Control

The 2065 CPU operates with a basic CPU clock cycle period of 200 ns; i.e., a 5-megahertz (5-mHz) clock frequency. Each cycle is composed of clock and not-clock portions, used for data transfer and logic functions, respectively. The 200-ns clock cycle period is divided into twenty 10-ns intervals for intracycle timing.

CLOCK SIGNAL GENERATORS

Two types of clock signal generators are used in the CPU, depending on the system model. CPU's in a Model G65, H65, or I65 system, with a maximum of two high-speed storage (HSS) units, use a 10-mHz continuously running crystal-controlled oscillator and divide-by-two logic to provide the basic 5-mHz (200-ns) clock signal. CPU's in a Model IH65 system with three HSS units, and in a Model J65 system with four HSS units, use a 5-mHz gated delay-line oscillator to provide the basic clock signal. The longer cables to HSS units 3 and 4 cause an additional 10-ns to 15-ns storage response delay, which is accounted for by inhibiting the clock. The gated delay-line oscillator is also used if a 2361 Core Storage Unit (Large Capacity Storage or LCS) is attached or if the Multisystem feature is installed, regardless of the CPU model. The gated delay-line oscillator reduces the time required to restart the clock after the storage request sequence that required it to be inhibited. It is not necessary to wait for the next full oscillator cycle to restart the clock as would be the case if a continuously running oscillator were used (as much as the full 200 ns might be required).

Both types of clock signal generators may be operated with a higher output frequency (5.128 mHz), thereby shortening the clock cycle period 2.5% (from 200 ns to 195 ns) for test purposes. The higher frequency is obtained from the clock signal generators by setting the FREQUENCY ALTERATION switch to the down position and by turning the CE Key switch to the CE position. Both switches are located on the CPU system control panel.

Model G65, H65, and I65 CPU Clock Signal Generator

The Model G65, H65, and I65 CPU's have a continuously running crystal-controlled oscillator with two crystals: 10 mHz and 10.256 mHz. The crystal used is determined by a reed relay. The relay is operated by the FREQUENCY ALTERATION switch and the CE Key switch (Diagram 4-1, FEMDM). With either switch in the normal operating position, the relay is not picked and the 10-mHz crystal is used. With both switches set to the test mode position, the relay is picked and the 10.256-mHz crystal is used.

The output of the oscillator circuit drives the frequency dividing logic, consisting of six inverting AND's. The output of the dividing logic provides the basic 5-mHz (200-ns) or 5.128-mHz (195-ns) clock signal. This oscillator cannot be inhibited by the logic.

Model IH65 and J65 CPU Clock Signal Generator

The Model IH65 and J65 CPU has two gated delay-line oscillators operating at 5 mHz and 5.128 mHz. The oscillator used is determined by the FREQUENCY ALTERATION switch and by the CE Key switch (Diagram 4-2, FEMDM). With either switch in the normal operating position, the 5-mHz oscillator is enabled. With both switches set to the test mode position, the 5.128-mHz oscillator is enabled. The output of the oscillators provides the basic 5-mHz (200-ns) or 5.128-mHz (195-ns) clock signal.

The gated delay-line oscillator can be inhibited by the BCU during a storage request sequence to either HSS unit 3 or 4 (Diagram 4-2). The oscillator is inhibited at the

time the BCU would normally expect the data from the accessed HSS unit (600 ns after the 'select' signal was sent to the unit) and is automatically restarted at the time the data arrives from the unit by a fixed (pre-adjusted) time delay. The oscillator provides stable clock signals immediately; the negative-going signal into the oscillator not only starts the oscillator but becomes part of the first output cycle.

A continuously running crystal-controlled oscillator (identical to the clock signal generator in the Model G65, H65, or I65 CPU; see Diagram 4-1) provides a 5-mHz reference frequency for adjusting the 5-mHz delay-line oscillator frequency. A comparator circuit (Diagram 4-2) mixes the two signals and provides an output that is the absolute difference frequency between the two signals. When the two signals are within 1 kHz of each other, the output of the comparator circuit is a null or is a difference frequency of less than 1 kHz, indicating that the delay-line oscillator frequency is within 0.02% of the crystal-controlled oscillator frequency.

CLOCK TIMING

- Triggers are set and reset at clock time.
- Latches are set and reset at not-clock time.
- Logic operations normally occur at not-clock time; subsequent data transfers occur at following clock time.
- Symmetrical and unsymmetrical clock signals are used, depending on logic function.
- Twenty 10-ns delay intervals provide for intracycle timing.

Throughout the CPU, trigger and latch logic is used for all data-handling and control functions. Although implemented with the same logic components (AND's, OR's, and inverters), triggers (by definition) are set or reset at clock time, whereas latches (by definition) are set or reset at not-clock time. In general, the data registers consist of triggers, and all intermediate logic units (such as adders, incrementer/decrementers, and decoders) consist of latches. Control logic consists of both triggers and latches. The CPU design provides for all intra-CPU data manipulation to be done by register-to-latch-to-register (trigger-to-latch-to-trigger) sequences, in lieu of direct register-to-register (trigger-to-trigger) transfer. Continuous availability of stable data results from the overlapping of the trigger and latch set/reset states (Figure 2-1). Note that the data (e.g., "A" in the figure) is stable in either a trigger or a latch at any one time during the indicated two clock cycles, but that the trigger and latch are available for new data ("B") in the next clock cycle.

Each logic component within the CPU introduces some degree of signal delay which must be considered in the CPU operation. All logic blocks with inversion introduce between 3 ns and 20 ns of delay, with 10 ns as the average. (Some special circuits will introduce either 30 ns or 700 ns.) All logic blocks without inversion introduce less than 3 ns of delay. In addition, approximately 10-ns to 12-ns delay is created by every 6 feet of signal transmission line.

Because of these delays, the clock signal is converted from a 5-mHz symmetrical signal to a 5-mHz unsymmetrical signal, with an 80-ns clock time and a 120-ns not-clock time. This unsymmetrical signal provides the needed extra time for logic operations during not-clock time and still leaves sufficient time for trigger input at clock time.

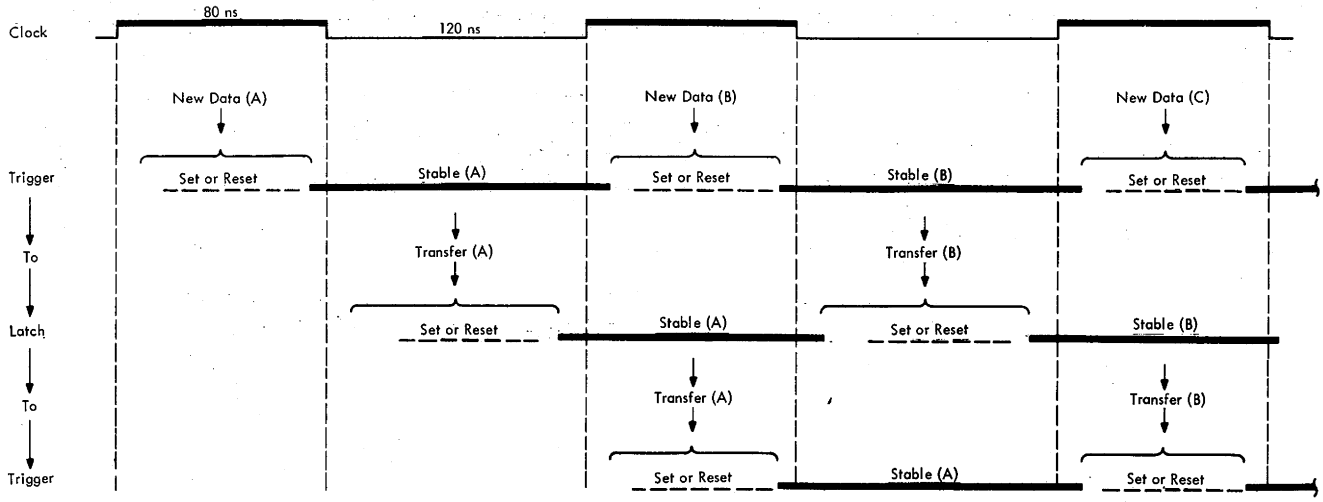


Figure 2-1. Trigger and Latch Data Relationship

Many logic blocks are used in the parallel adder at clock time, thus causing excess delay. To overcome this delay, the unsymmetrical clock signal is extended to a symmetrical clock signal with a 100-ns clock time and a 100-ns not-clock time; this conversion is made within the parallel adder. Some of the parity checking and sign propagating circuits are timed by the unsymmetrical clock signals. Primary concern in clock timing is to provide stable information at sampling time.

Oscillators A through E (Diagram 4-3, FEMDM) set up controls to stop, start, or control the clock. These signals are used in advance of clock time so that the controls are stable when they are needed.

With most of the CPU logic blocks introducing 10 ns of delay, the 200-ns clock cycle period is divided into twenty 10-ns delay intervals to provide timing within a clock cycle (Figure 2-2). These delay intervals are called "B" time or "P" time and are relative to the start of the clock cycle. "B" time refers to symmetrical clock signals; "P" time, unsymmetrical clock signals. The delayed clock signals are created by series inverters (each inverter introducing one 10-ns delay interval) or by time delays (e.g., B0, B1, B2, ... , and P0-1, P0, P1, P2, ...).

CLOCK CONTROL AND SIGNAL DISTRIBUTION

- Manual, BCU, and ROS operations, and errors, affect clock signal availability.
- One symmetrical and one unsymmetrical clock signal are distributed throughout logic gates.
- Time delays unskew and synchronize clock signals within and between sections of logic.

The availability of the clock signals to the CPU processing logic is controlled by the clock-stopping logic (Diagram 4-3). Usually, the clock signals to the BCU are not stopped because the BCU must continue to service the I/O channels even if the CPU is not operating. In the Model IH65 or J65 CPU, however, the BCU has the ability to inhibit the clock signal generator to account for the cabling delays to HSS units 3 and 4 (Diagram 4-2). During maintenance operations (such as scan, log-out, and single-cycle operations), the clock signals may be stopped or permitted to run intermittently.

The 'pass pulse' trigger (Diagram 4-3) provides clock signal distribution control during both normal (continual) and single-cycle operations. A start, load, or reset operation sets the 'pass pulse' trigger and permits the clock signals to be passed on to the logic. When in the single-cycle mode, the 'block' trigger (set by the same operations) resets the 'pass pulse' trigger and blocks the clock signals before the next clock cycle, unless the BCU holds the clock on.

The 'stop clock' trigger (Diagram 4-3) provides BCU control of the clock signal distribution. If the BCU cannot process a CPU storage request immediately or if a request to a large-capacity storage (LCS) unit has been made, the BCU sets the 'stop clock' trigger and blocks the clock signals. The 'BCU cleanup' signal for the HSS, the 'advance' signal for the LCS, or a start, load, or reset operation resets the 'stop clock' trigger.

During certain operations, the ROS microprogram may stop the CPU clock signal distribution for one or two cycles ('STOP1' or 'STOP2' micro-orders). The 'stop clock ROS' trigger provides this control from the bit configuration of ROS word bits 45 and 46 (Diagram 4-3).

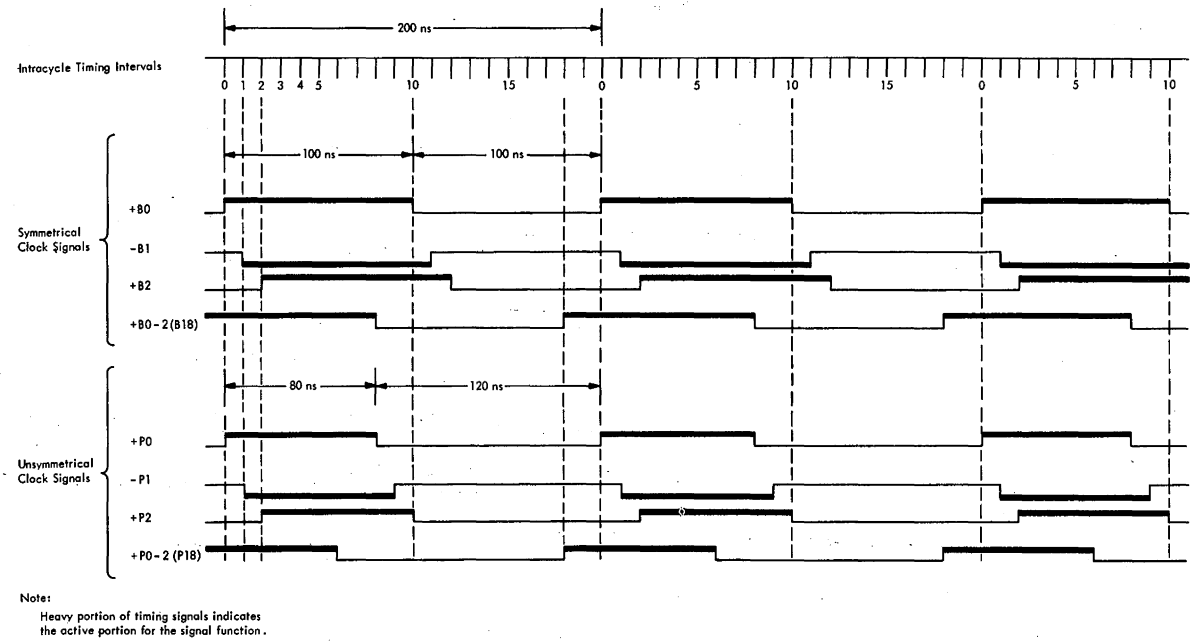
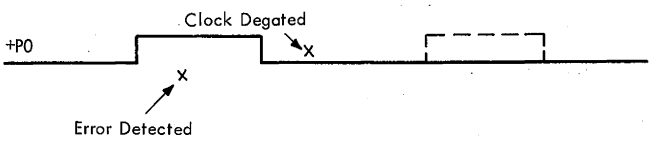


Figure 2-2. Typical Clock Signals

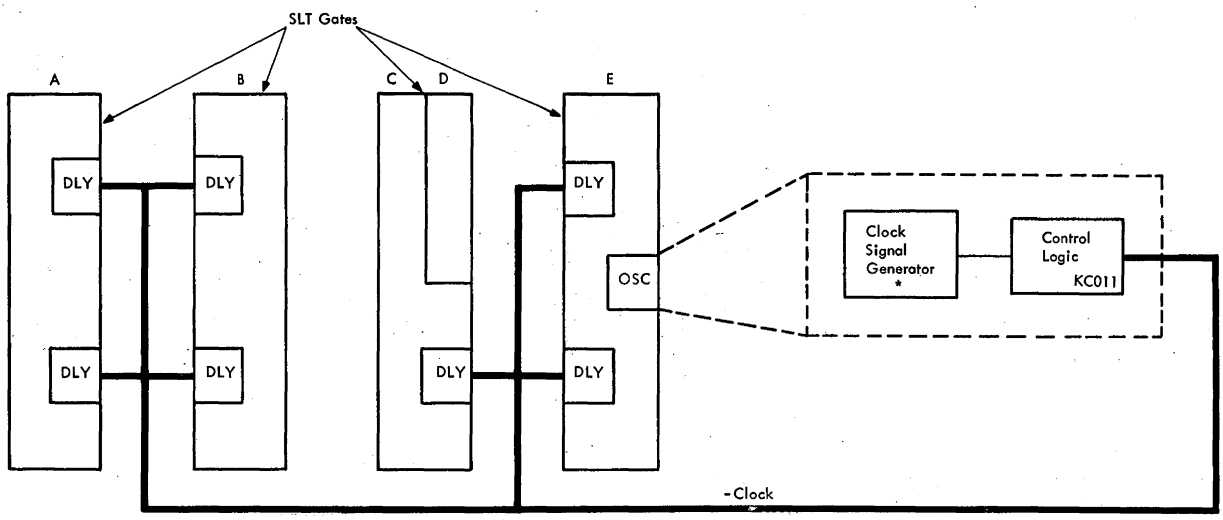
The distribution of clock signals to portions of the ROS logic is stopped while the CPU is in the Wait state (Diagram 4-3). The Wait state is entered if PSW(14) = 1 at end op. Entering the stop loop or an interruption removes the wait state block of the ROS signals. The 'time clock step' trigger also removes the wait state block, allows the exceptional branch to step the time clock (Diagram 8-20, FEMDM), and then allows the CPU to re-enter the wait state, if there has been no change in PSW(14).

Clock signal distribution to the CPU processing logic is stopped when the 'error' trigger is set and the CPU CHECK switch is not in the DSBL (disable) position. Error detection occurs during clock time, and the clock signals cease with the next not-clock time:

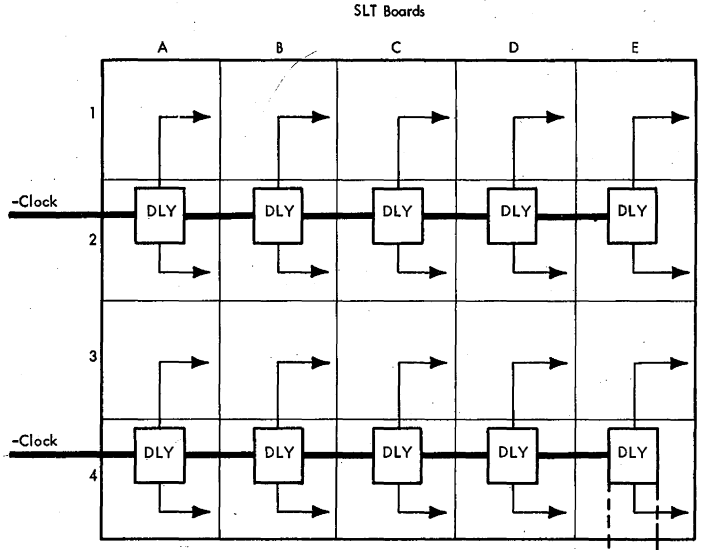


Necessary information for error analysis is thus held in latches and triggers to be examined directly or to be stored by a logout operation. Note that the DLY (delay) units allow the clock to finish the cycle started by the last clock signal. The clock may be restarted by the CPU CHECK switch, internal circuits, or by resetting the 'error' trigger (Diagram 4-3).

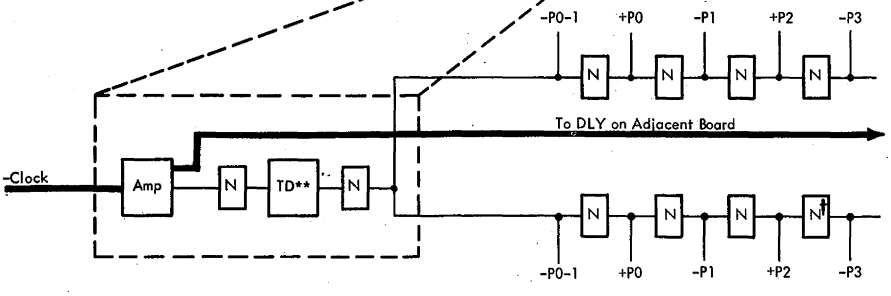
The clock signal development and distribution concept is shown in Figure 2-3; note that it is not representative of any logic gate. Figure 2-3 (A) shows how the master 'clock' signal is distributed to the gates. Figure 2-3 (B) shows distribution within a typical gate, with a separate delay logic for every two SLT boards. The adjustable time delay shown in Figure 2-3 (C) allows for unskewing of the clock signals; i.e., aligning all 'P0' signals. The 'special sync for P0 B0' signal is used to adjust all time delays so that all 'P0' signals occur simultaneously; see ALD M8001 for the timing procedure.



A. Clock Distribution



B. Typical 5 X 4 Gate



C. Unskewing Delay Logic

Note: Total delay includes 10 ns per 6 feet of wire plus adjustable time delay.

* ALD KC021 for Models G65, H65, and I65.
ALD KC101 for Models IH65, and J65.

** TD is adjusted to synchronize each P0 with 'special sync for P0 B0' signal shown on ALD ZA001. Each P0 must coincide with every other P0 in any SLT gate.

† Number of inverters varies from gate to gate, depending on need for particular signal.

Figure 2-3. Clock Signal Development and Distribution

The read-only storage (ROS) is a device containing a permanently recorded microprogram used to control CPU operations. The microprogram is in the form of 100-bit micro-instructions (ROS words), each of which has a unique predetermined bit pattern. The ROS words can be read out as required, but a physical modification is necessary to change the stored information. When decoded, the bits of the ROS word condition gates whose outputs perform the necessary functions to execute an operation. Thus, ROS eliminates the need for most complex instruction decoders and sequencing networks, and introduces a flexibility to machine design not previously available in control hardware. This flexibility allows changes to be made to control circuits for special features by replacing printed circuit sheets in ROS.

CAPACITIVE READ-ONLY STORAGE ARRAY

The capacitive read-only storage (CROS) array consists of 2816 100-bit ROS words which are addressed by a 12-bit ROS address register (ROSAR). The array consists of 16 planes, each of which is divided into 4 quarter planes. Each quarter plane has one array driver energizing 1 of 22 select lines. Each select line causes two ROS words to be read out. To address a particular drive line from ROSAR, bits 0–3 select a plane, bits 4 and 10 select a quarter plane, and bits 5–9 energize one select line. Bit 11 of ROSAR selects one of the two ROS words (upper or lower) read out each cycle.

CROS Electrical Theory

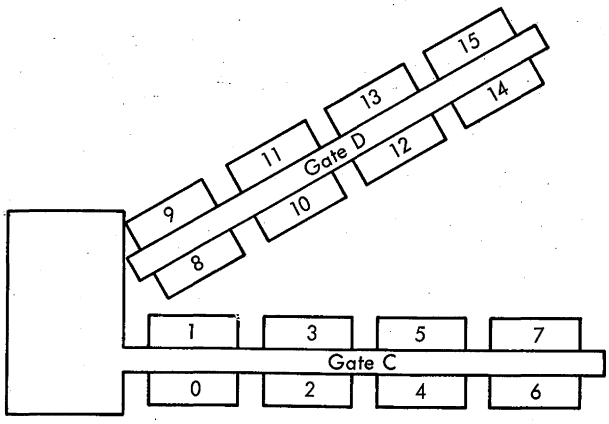
The CROS operates on the presence or absence of a capacitor between a drive line and a sense line. Only one driver at a time may be energized. In the example shown in Figure 2-4, when driver 1 is energized an impulse is coupled through the capacitor C1 to differential sense amplifier D, producing the "D" bit. The same drive results in inputs to differential sense amplifiers A, B, and C, but the polarity is reversed and no bits are generated. To equalize the capacitive load (impedance) to all sense amplifiers, a balance line is provided with each driver and is allowed to "float".

Note: Because the balance line function was found to be unnecessary, later machines do not use the balance line, although it is still printed on the ROS planes.

Some unwanted capacitive coupling exists in this type of matrix. In Figure 2-4, when driver 1 is energized, C1 couples the voltage shift to sense line D, C2 couples the voltage shift to drive line 2, and C3 couples the voltage shift to sense line B. This unwanted signal is very low because it passes through three elements in cascade. The threshold of the sense amplifier is designed so that the low signal is rejected while the desired signal is amplified.

CROS Planes

The 2816 words of ROS are stored in 16 planes. Each plane contains the capacitors, drive lines, balance lines, and sense lines for 176 100-bit ROS words. The drive and balance lines are independent, whereas the sense lines feed common sense amplifiers. Planes 0–7 are on gate C, and planes 8–15 are on gate D:



Drive and Balance Lines (Bit Plates)

The drive and balance lines are photo-etched from a sheet of copper that is bonded to epoxy glass (Figure 2-5). The resulting epoxy sheet with copper drive and balance lines is called a *bit plate*. A separate bit plate controls the bit configuration for each CROS plane.

Tabs at the top and bottom of the bit plate are used for electrical connections to the drive and balance lines. The top tabs connect the drive and balance lines to terminating resistors. The bottom tabs connect the drive lines to the drive circuits.

Four holes in the bit plate align the bit plate to the sense plane. The two outer holes snap over locating studs in the sense plane, and the inner two holes provide clearance for the center studs.

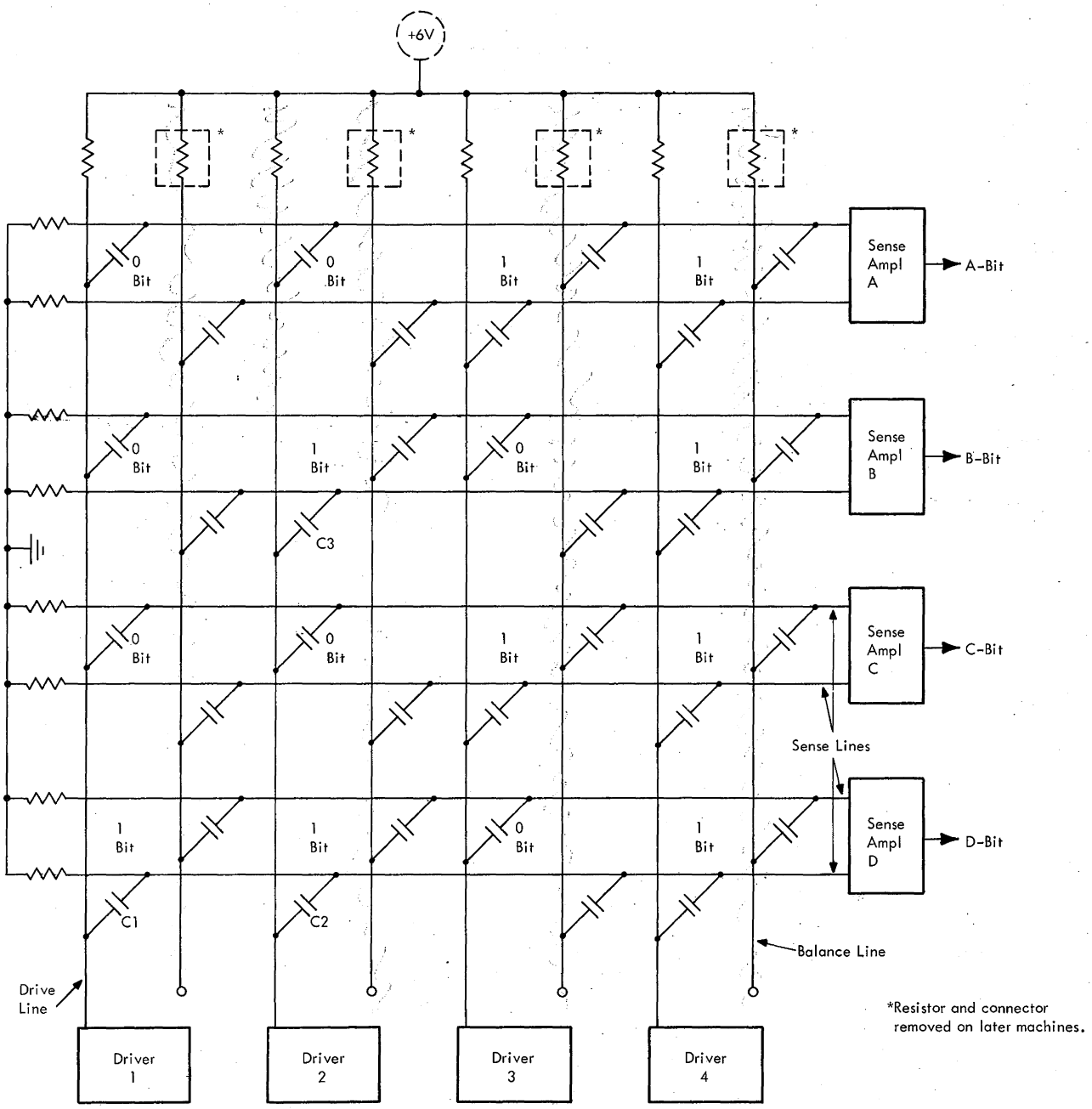


Figure 2-4. Basic 4 X 4 CROS Matrix

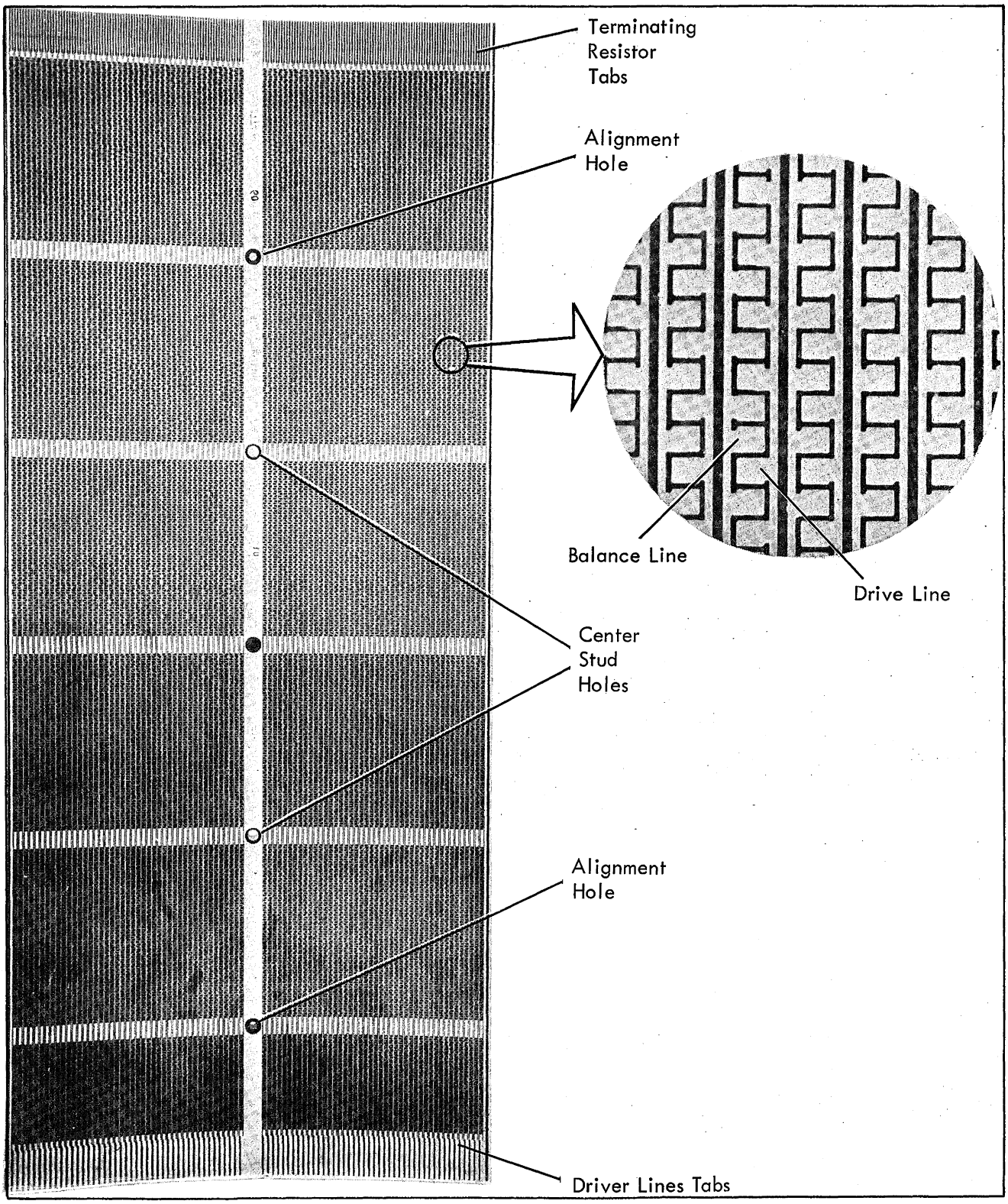


Figure 2-5. Bit Plate

Sense Lines

- 200 pairs of sense lines are in each CROS plane.
- Pair of sense lines carries signal for one ROS-word bit position.
- 200 pairs of sense lines read out two 100-bit ROS words simultaneously.

The sense lines are photo-etched into copper-covered epoxy-glass plates (Figure 2-6). The sense-line plates are permanently mounted to the array gates. Electrical connections from the sense lines to the terminating resistors and sense amplifiers are made with low-temperature solder.

There are 200 pairs of sense lines in each CROS plane. Two sense lines are required to read out one bit position of the ROS word. One drive line simultaneously reads out two 100-bit ROS words, which use 200 pairs of sense lines.

Figure 2-7 shows the layout of the sense lines in the ROS planes. The top pair of sense lines is bit 0 of the upper word. The next lower pair of sense lines is bit 0 of the lower word. This order continues to the bottom pair of sense lines, which is bit 99 of the lower word. The upper and lower words are read out simultaneously. Each sense line is terminated through a resistor to ground. Note the distribution of sense lines through the planes to the differential sense amplifiers. The sense lines through the planes on both sides of a gate are tied together for each bit. The pair of sense lines from each gate is then OR'ed in the sense amplifier for each bit. Because only one plane has an active drive line for a given ROS address, the sense amplifier receives only one input signal.

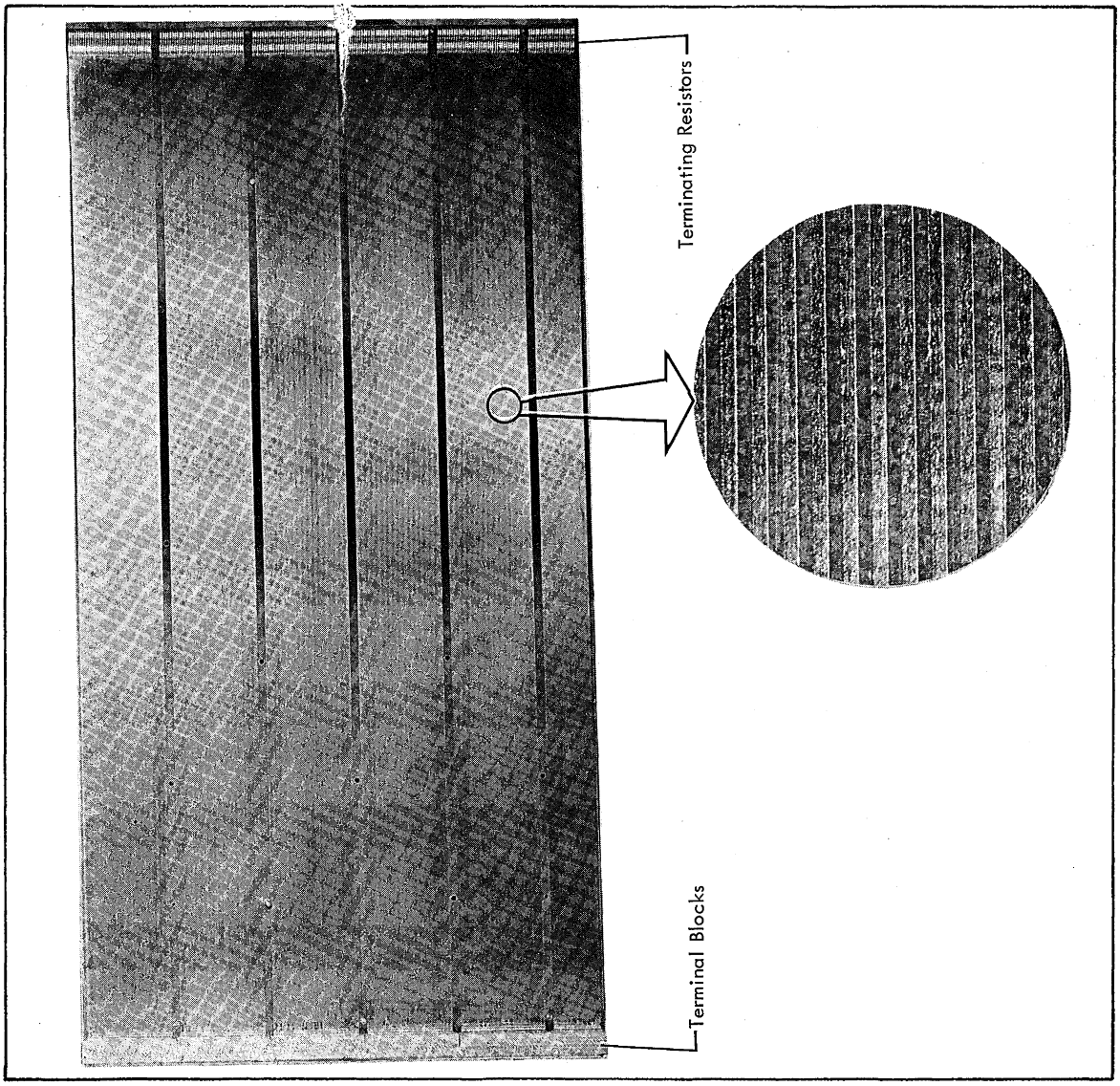


Figure 2-6. Sense Lines

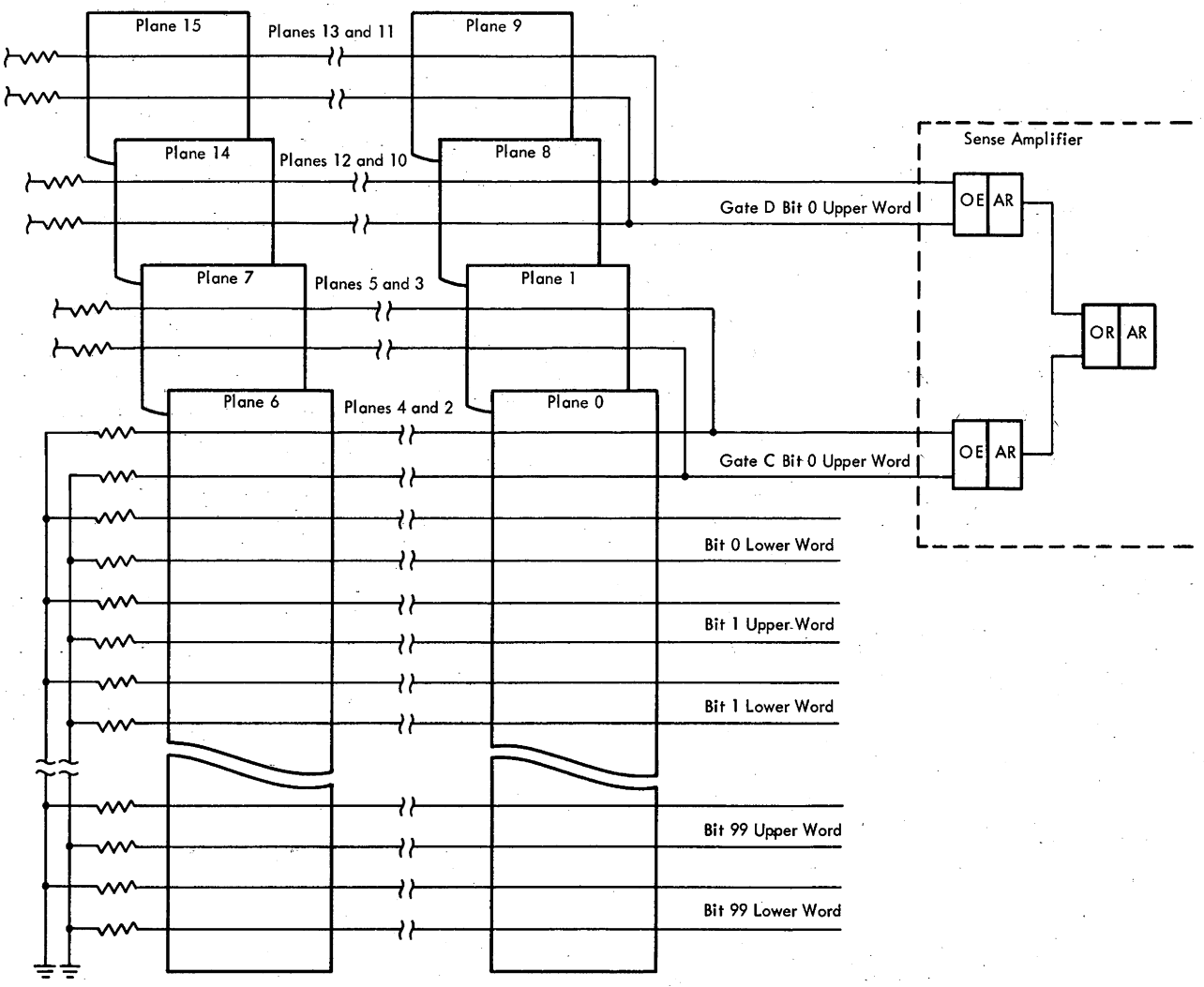


Figure 2-7. Sense Line Layout

Bit Capacitors

The bit capacitors are formed by sandwiching a sheet of Mylar† between the bit plate and the sense lines (Figure 2-8). Pressure plates hold these pieces firmly together. The Mylar is the dielectric, and the drive, balance, and sense lines become the plates of the capacitors.

Tabs on the drive and balance lines increase the size of the capacitors to form the bit configuration. The effective capacitive coupling of a drive line to a sense amplifier is equal to C1 minus C2. The size of this effective capacitor is approximately 0.5 pf.

The bit configuration within a CROS plane is controlled by the bit plate. Therefore, the ROS word can be changed by replacing the bit plate that contains the word.

Physical Package

A CROS plane consists of a sandwich composed of the sense line board, a dielectric sheet, and a bit plate. These pieces are held firmly together by pressure plates (Figure 2-9).

A pressure plate, with a neoprene pad, fits over each group of capacitors in the plane. The plates are loosely connected to a pressure frame that is bolted to the gate. Adjusting screws in the frame squeeze the pressure plate against the bit plate. Because the sense lines are on a rigidly mounted board, the pressure plate holds the bit-capacitor sandwich firmly together.

Electrical connections to the bit plate are also made through pressure connections.

ROS ADDRESSING

ROS word addresses are assembled in the 12-bit read-only storage address register (ROSAR). Each ROS word contains the basic address of the next ROS word. The basic address may be modified by machine operation or by error conditions.

†Trademark of E. I. duPont de Nemours & Co. (Inc.)

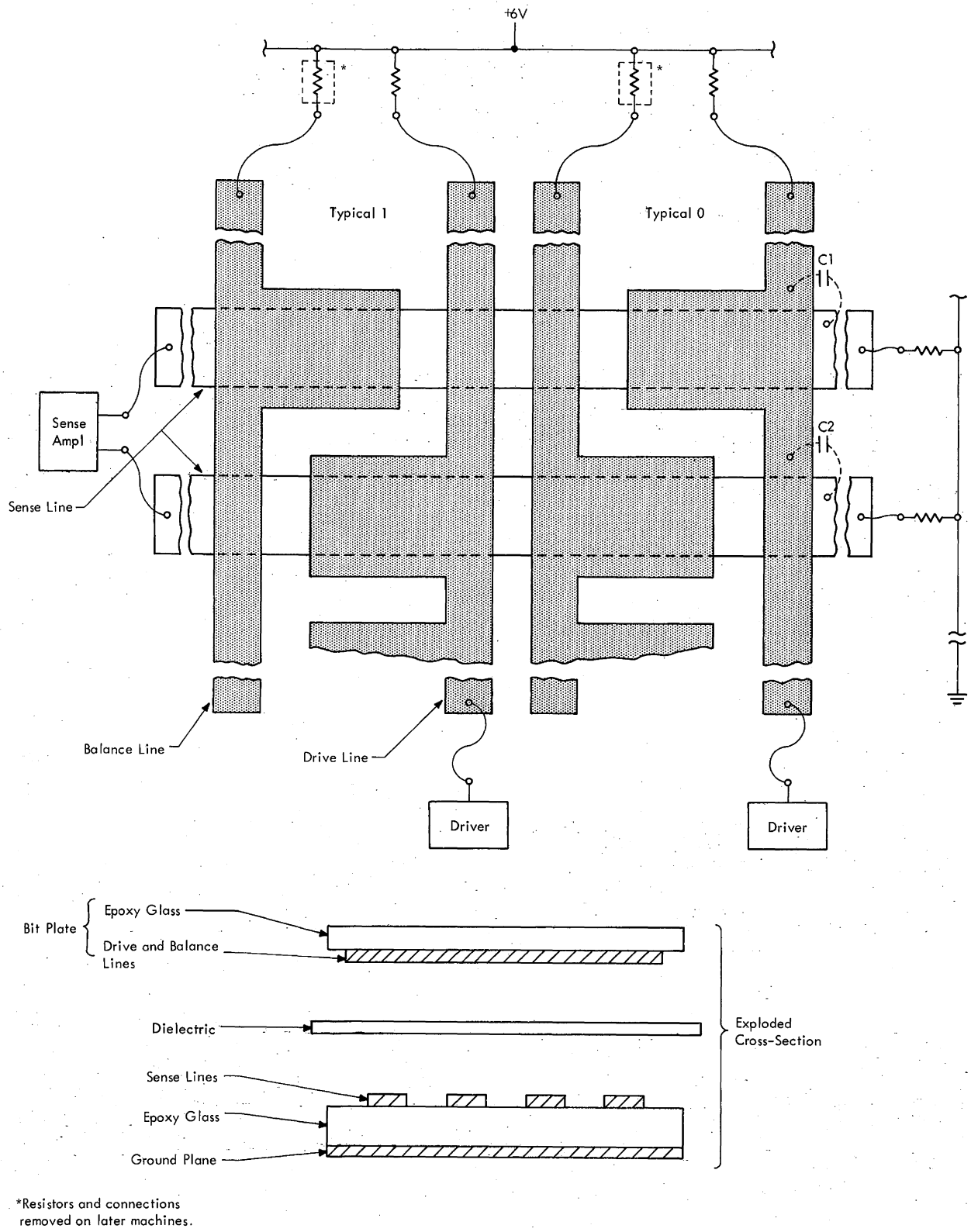


Figure 2-8. Bit Capacitors

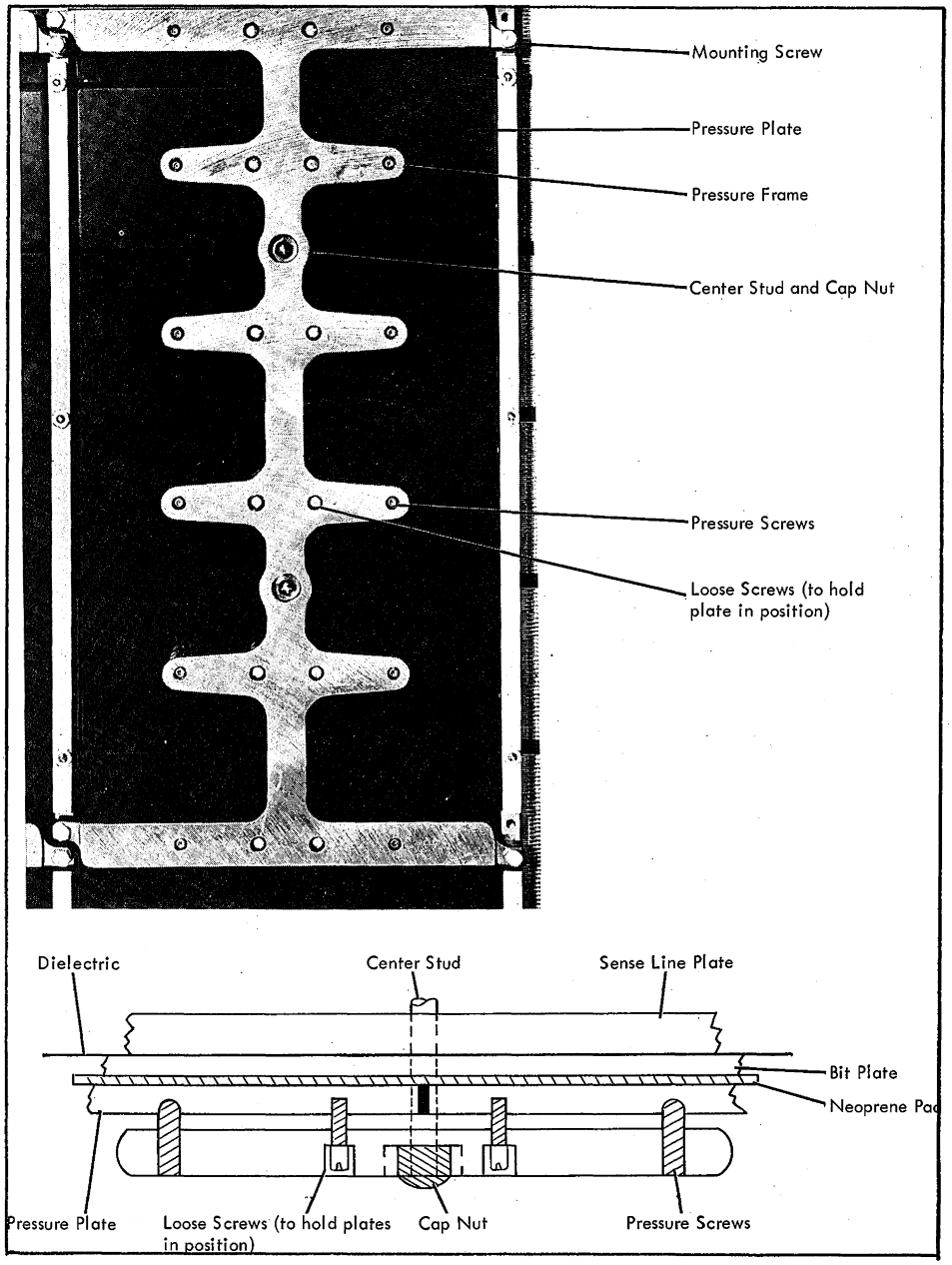


Figure 2-9. CROS Plane Pressure Mounting Assembly

Read-Only Storage Address Register

- ROSAR(0–11) supplies 12-bit address that selects next ROS word.
- Overriding branch and manual ROS operations force new address to ROSAR.

The ROSAR, a 12-position (labeled 0–11) latch register, supplies the address to select the next ROS word. The configuration of the ROSAR contents (address) is controlled by the NA, K, and J control fields of the present ROS word. A new address is available in ROSAR 50 ns after each P2 clock time. Although each ROS word contains the address of the next word to be accessed, address modifications can result by satisfying data-dependent branch conditions. These data conditions are stable at ROSAR(0–10) by P2 + 30 ns and at ROSAR(11) by P2 + 50 ns. The gate at ROSAR is a P4 clock pulse. To prevent late branching, the output of all ROSAR bits must be stable by P2 + 60 ns; at approximately P0 minus 30 ns, the ROS sense latches are sampled. This sequence is repeated every machine cycle.

The ROSAR bit positions can be arbitrarily divided into four groups according to their inputs. These groups are (1) ROSAR(0–5), which normally receives only the six high-order bits of the base address; (2) ROSAR(6–9), which can receive the four low-order bits of the base address and/or the output of the X-branch decoder; (3) ROSAR(10), which can receive data from either the X-branch decoder or the Y-branch decorder; and (4) ROSAR(11), which can receive data from either the X-branch decoder or the Z-branch decoder. [Note the overlap between the base address and X-branches on ROSAR(6–9).] An overriding branch, however, affects all positions of the ROSAR.

The following micro-orders can cause an overriding branch: 'T→RAR,' 'EXCEP', and 'SPEC'. If control field K contains the 'T→RAR' micro-order, the base address is inhibited and T(40–51) is transferred to ROSAR(0–11). This branch is unconditional. An 'EXCEP' micro-order inhibits the base address if an interruption is pending. The source of the interruption provides the branch address. When an 'SPEC' micro-order is specified, a specification program interruption forces the branch address into the ROSAR. In addition, a local store write operation is blocked.

Certain FLT operations force a new address into ROSAR (Diagram 4-101, FEMDM).

Two manual operations cause the contents of the ADDRESS switches to be forced into ROSAR: depressing the ROS TRANSFER pushbutton or activating the REPEAT ROS ADDRESS switch. The operation of these controls is described in Section 1 of Chapter 6.

ROSAR(0–5)

ROSAR(0–5) (Diagram 4-101, FEMDM) can be set from one of three sources: ROS sense latch bits 48–52, T(40–45), or ADDRESS switches 0–5. Normally, positions 0–5 receive the five high-order bits of the base address from the NA control field of the current ROS word contained in the ROS sense latches. If, however, control field K contains an overriding branch and the branch condition is met, or if an FLT operation is in progress and certain conditions are present, or if certain manual operations are being performed, the base address is inhibited from entering the ROSAR and ROSAR(0–5) is set from the ST bus or the ADDRESS switches.

ROSAR(6–9)

ROSAR(6–9) (Diagram 4-102, FEMDM), in addition to receiving the low-order bits of the base address and the overriding branch address, can be set individually by X-branches (functional branches) specified by control field J (bits 62–68) of the ROS word. The control field J micro-orders that specify X-branches are listed as J96 to J124 on ALD M7021, which also shows the ROSAR bits that are set under specific conditions for each X-branch micro-order. When an X-branch is executed, the high-order positions [ROSAR(0–5)] remain unchanged; i.e., still contain the high-order bits of the base address. The base address bit positions corresponding to the bits affected by the X-branch must be set to zero when an X-branch micro-order is given. For example, if ROSAR(6) can be set to a 1 by a certain micro-order, ROS sense latch bit 53 must be 0 in the ROS word that contains the branch.

ROSAR(10)

ROSAR(10) (Diagram 4-103, FEMDM) can be set by an X-branch, an overriding branch, and/or a Y-branch. The base address, however, has no affect on this bit. Also, because an overriding branch and a Y-branch are both decoded from bits in control field K (bits 57–61) of the ROS word, only one of the two branches can be executed at a time; they cannot be executed together. That is, if control field K specifies an overriding branch, a Y-branch cannot be specified, and vice versa.

However, a Y-branch and an X-branch can be executed together because they are functions of micro-orders in separate control fields (K and J). The result of ROSAR(10) is as follows: if neither the X-branch condition nor the Y-branch condition is met, ROSAR(10) remains a 0; if either or both of these conditions are met, ROSAR(10) is set to a 1. The Y-branch micro-orders that affect ROSAR(10) are listed in the K field on ALD M7031 in the same manner as that of X-branches.

ROSAR(11)

ROSAR(11) (Diagram 4-104, FEMDM) can be set by an X-branch, an overriding branch, or a Z-branch. The base address, just as for ROSAR(10), has no effect on ROSAR(11). Because an X-branch and a Z-branch are both decoded from bits in control field J (bits 62–68), only one of the two branches can be executed at one time. ALD M7021 lists the Z-branch micro-orders.

However, a Y-branch and a Z-branch can be executed together. The effects on ROSAR(10,11) are as follows: if neither condition is met, both ROSAR bits remain a 0; if either condition is met, the associated ROSAR bit is set to 1; if both conditions are met, both bits are set to 1's.

If control field K contains an overriding branch and control field J contains a Z-branch and the overriding-branch condition is met, the result of the Z-branch is inhibited and ROSAR(11) is set as specified by the overriding branch.

ROSAR(0–10) Decoding

ROSAR decode logic decodes the address in ROSAR(0–10) to select 1 of 1408 array drive lines. ROSAR(0–4,10) is decoded into 1 of 64 drive lines; ROSAR(5–9) is decoded into 1 of 22 select lines; and ROSAR(0) selects

gate C or D. One select line and one drive line then select 1 of the 1408 array drive lines. See Diagram 4-105, FEMDM, for decode flow. ROSAR(11) is decoded to select one of the two ROS words read out each cycle. (If bit 11 = 1, the lower word is selected; if bit 11 = 0, the upper word is selected.)

Strobed Drive Lines

An ROSAR address selects 1 of 64 strobed drive lines by decoding ROSAR(0–4,10), as shown in A of Diagram 4-105, FEMDM. The decoded address is a gate-drive signal to the array drivers. Each strobed drive line controls the array drivers for one CROS quarter plane.

Decoding is accomplished in two levels. In the first level, bits 3, 4, and 10 are decoded to activate 1 of 8 lines, and bits 0, 1, and 2 activate 1 of 8 lines. The outputs of the two first-level decoders are then combined with a gate-drive signal to activate 1 of 64 drive lines.

Select Lines

One of 44 select lines is activated by decoding ROSAR(0,5–9) (B of Diagram 4-105). Twenty-two of these select lines are connected to gate C, and 22 to gate D. ROSAR(0) is decoded to select the gates, and ROSAR(5–9) is decoded to activate a select line within a gate. Although ROSAR(5–9) can be decoded 32 different ways, only the first 22 combinations are considered valid addresses; the other 10 combinations are not tested. If an illegal bit combination is entered into these bit positions, no select line is activated. Illegal addresses are addresses in which ROSAR(5,6) = 11.

Decoding is accomplished in two levels. In the first level, bits 7, 8, and 9 are decoded to activate 1 of 7 lines, and bits 0, 5, and 6 are combined with a 'gate word select' signal (clock P2 delayed) to activate 1 of 6 lines. (Note that when bits 5 and 6 = 11, no signals are developed from the first-level decoder.) The outputs of the two first-level decoders are then combined in the second-level decoder to activate one of the 44 select lines.

Array Drivers

There are 1408 array drivers in ROS, 704 on gate C, planes 0–7, and 704 on gate D, planes 8–15. Diagram 4-105, C, shows how the drive-line signals are developed and distributed. Each array driver is an AND that AND's 1 of 64 drive lines with 1 of 22 select lines (details are shown in Diagram 4-106, FEMDM). The AND's are single transistors: the drive lines condition the emitters while the select lines control the bases. Voltage is supplied to the collectors through the array drive-line-terminating resistors.

Sense Amplifiers

The sense amplifiers increase the voltage difference between paired 1 and 0 sense lines. The first stage of the sense amplifiers (D of Diagram 4-105, FEMDM) consists of two differential amplifiers, one for gate C sense lines and one for gate D sense lines. Because only one array driver is active for a machine cycle, the sense lines of only one gate carry a signal during a machine cycle. The first-stage differential amplifier increases the voltage difference between paired sense lines and sends this signal to the second-stage amplifier. The second-stage further amplifies the signal and transmits it to the ROS sense latches.

ROSAR(11) Function

Each cycle, 200 bits are sent to the sense latches. ROSAR(11) divides this information into two 100-bit words. If ROSAR(11) = 1, the lower word is selected; if ROSAR(11) = 0, the upper word is selected.

ROS DATA FLOW

ROS word data is transferred from the sense amplifiers to the sense latches. A portion of the word is immediately decoded, while other portions step through registers and latches to provide a delay so that the data is available at the desired time. The ROS word data flow is shown in F of Diagram 4-105.

ROS Sense Latches

The 100 ROS sense latches hold the ROS word for decoding and for setting ROSDR at P0 of the next machine cycle. The sense latches are set by strobing either the upper or lower word sense amplifier outputs and are reset approximately 120 ns after P0 of the machine cycle (E of Diagram 4-105).

ROS Data Register and ROSDR Latches

The ROS data register (ROSDR) holds fields A through H, and M, N, P, and Q of the ROS word for use in the next machine cycle. Fields H, and M, N, P, and Q are decoded directly from the ROSDR to control LS and the adders, respectively. Fields A–G, however, are further delayed by holding them in the ROSDR latches. These fields are used for register ingating.

The ROSDR latches allow a ROS word to control certain gates during the register set time of the next cycle. For example, one ROS word may contain the micro-instruction: add the contents of T and A, and store the answer into A. The ROS word adds the contents of the

registers on one machine cycle and stores the sum from the parallel-adder-out bus at register set time of the next cycle.

Diagram 4-107, FEMDM, is a simplified diagram of ROSDR. In this diagram, each main division of ROSDR is represented by a single bit position. At clock P0-1 of each machine cycle, ROSDR is reset. At not-clock P0-1, the contents of the ROSDR (bits 6–36) are sent to the ROSDR latches. At clock P0, the contents of the sense latches are transferred to the ROSDR. The output of the ROSDR latches (containing the previous ROS word) and the output of ROSDR (38–42, 69–77, and 78–84) are then decoded to perform the selected micro-orders.

ROS Decoders

The ROS decoders use the bits from the ROS word to develop control lines. One micro-instruction may activate a number of control lines. Timing consideration governs the source of the lines; i.e., sense latches, ROSDR, ROSDR latches.

Field A (bits 6–9 of the ROS word) in Figure 2-10 is an example of a decoding network to develop control lines from ROS bits. Line A ['gate M1M2 to PAL(64, 65)'] is decoded from ROSDR because it updates PAL(64,65) before PAL is gated to AB. The 'B38M' micro-order (A7), shown on ALD M7001, uses line A to update PAL(64,65) during not-clock time. Then, lines B, F, and G are developed from ROSDR latches 6–9 to gate PAL(24–67) to AB(24–67) during the next clock time. Note that the 'B38M' micro-order results when bit 6 = 0 and bits 7, 8, and 9 = 111.

This example demonstrates the register-to-latch-to-register timing which controls the source of the decoded control lines. The other micro-order control fields are decoded in a similar manner to provide the control lines at the proper time.

ROS Timing

ROS timing is controlled by the master clock signals. At P0 + 160 ns, the ROS word is strobed (gated) into the sense latches, which are reset at P0 + 120 ns. Data from the sense latches is stable and available at P0-5 ns when it conditions ROSDR(6–42) for setting at clock P0 and ROSDR(69–84) for setting at clock P2. Gate controls from the sense latches (register data transfer) are activated at clock P2, and remain up for 190 ns. ROSDR latches are set at P7 ('not clock P0-1' signal) and initiate register inputs during the following 200 ns. Figure 2-11 shows the timing relationships of the registers and latches. These timings are theoretical and do not show the delays caused by the signals passing through inverters.

Note: Initially, ROSDR is set to all 1's; a 0 in a sense latch position resets the corresponding ROSDR position.

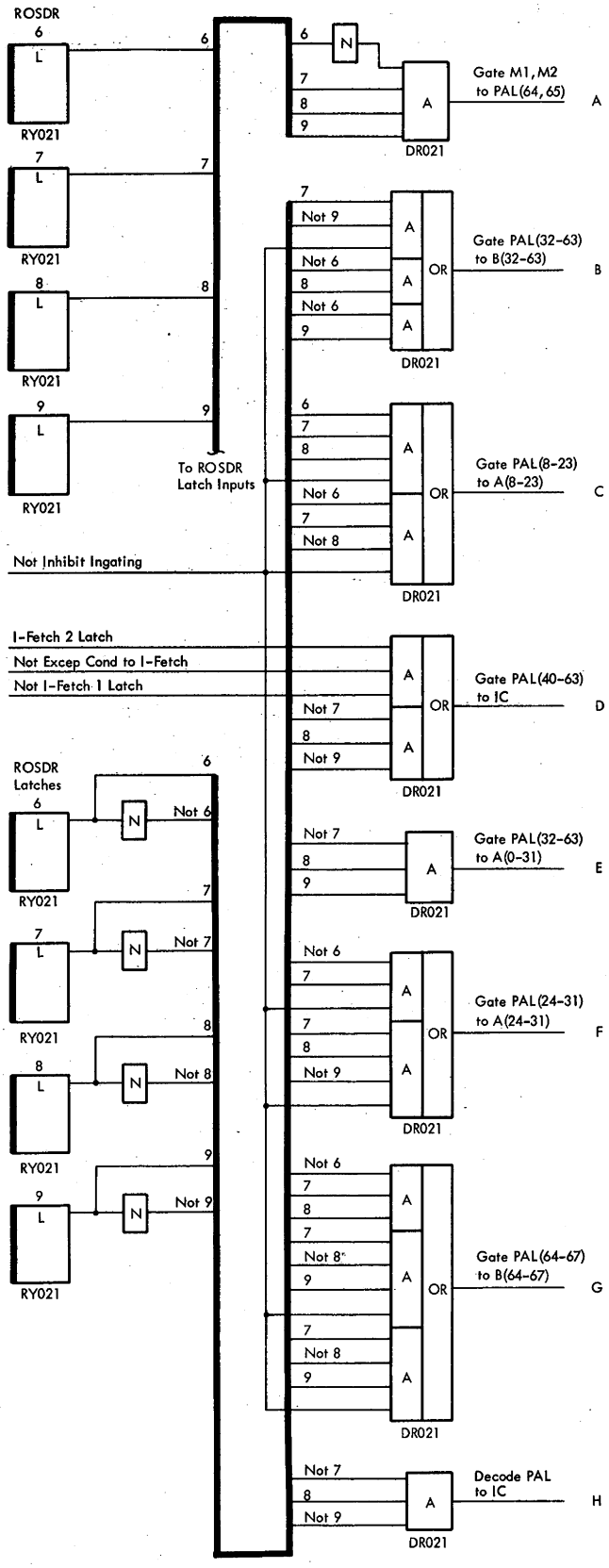


Figure 2-10. Control Field A Decoder

Maintenance Aids

When an error occurs, data leaves ROS registers and latches before the clock is stopped. To retain this information, which identifies the instruction which resulted in the error, secondary registers (which have no other purpose) are provided: ROSAR latches, ROS previous address registers A and B, and ROS backup register.

ROSAR Latches

The ROSAR latches are loaded from ROSAR at P11 time (not-clock P3 time) of each ROS cycle. At P4 time, the latch output is gated to the ROS previous address registers A and B (ROSPARA and ROSPARB) by an alternator. At the next P10 time (not-clock P2 time), the ROSAR latches are reset.

ROS Previous Address Registers

The contents of the ROSAR latches are alternately gated to ROSPARA and ROSPARB, which alternately contain the address of the current and previous ROS words. These registers, which are loaded at P4 time of the ROS cycle, comprise polarity-hold circuits and retain their values until gated into again. Thus, if ROSPARA is loaded on one cycle and ROSPARB on the next cycle, the contents of ROSPARA are maintained until the third cycle, at which time a new address is loaded. The contents of ROSPARA and ROSPARB are indicated on the roller switch indicators: roller 1, position 4, bits 12–23 and bits 24–35, respectively (Diagram 8-2, FEMDM).

ROSPARA and ROSPARB Alternator

The ROSPARA and ROSPARB alternator (Figure 2-12) causes the contents of the ROSAR latches to be sent alternately to the ROSPARA and ROSPARB indicators. Referring to Figure 2-12, assume that the latch is reset, the CPU is at clock P0 time, the A-gate is conditioned, and the B-gate is deconditioned.

At the following P3 time, AND 1 becomes conditioned, which in turn conditions AND 3. The output of AND 1 is also sent to AND 2, but AND 2 cannot be conditioned because the A-gate is set. The output of AND 3 gates the contents of the ROSAR latches to the ROSPARB indicator circuits.

Because the B-gate is deconditioned at P6 time, the latch is set. On the rise of the -P5 signal, the latch being set causes the B-gate to be conditioned and the A-gate to be deconditioned. With the A-gate deconditioned, AND 2 is activated at P3 time (via AND 1) of the following cycle. This action gates the contents of the ROSAR latches to the ROSPARA indicator circuits.

When the -P5 signal drops, the latch is reset. The gates remain in this condition (A deconditioned, B conditioned) until the rise of the -P5 signal. At that time, the A-gate is conditioned by the reset latch, and the B-gate is deconditioned by the A-gate.

During the next two cycles and each cycle thereafter, the operation described above is repeated until the CPU clock is stopped. At that time, the contents of the ROSAR latches are gated to the ROSPAR indicators associated with the deconditioned gate. To indicate which ROS address is in which set of indicators, the latch output

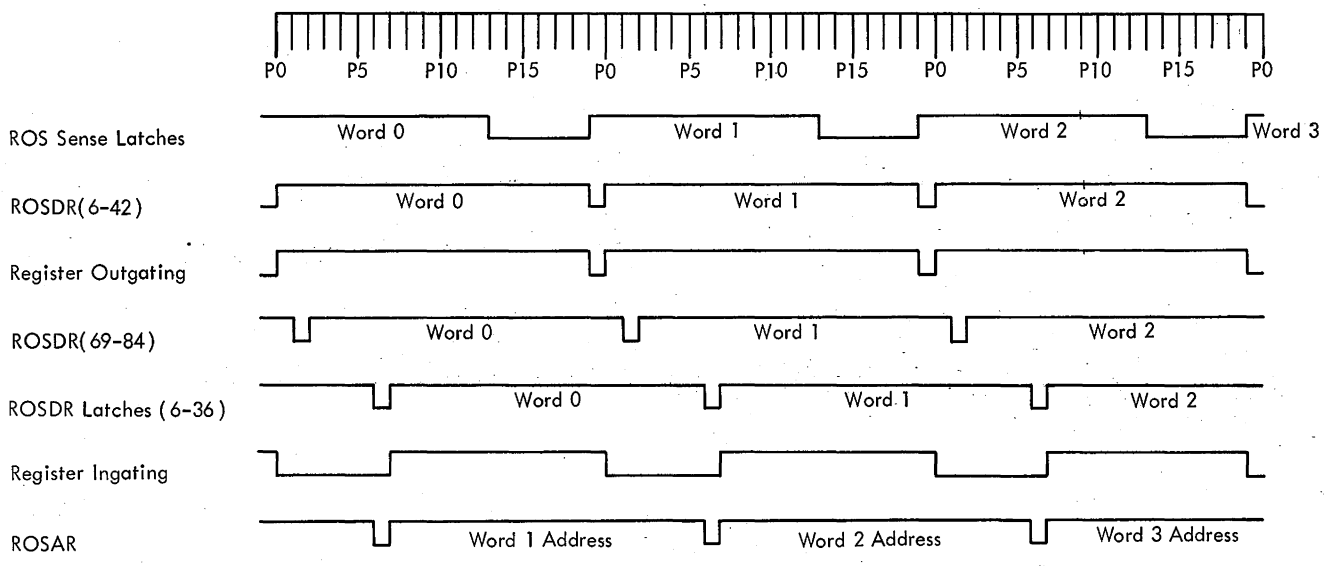
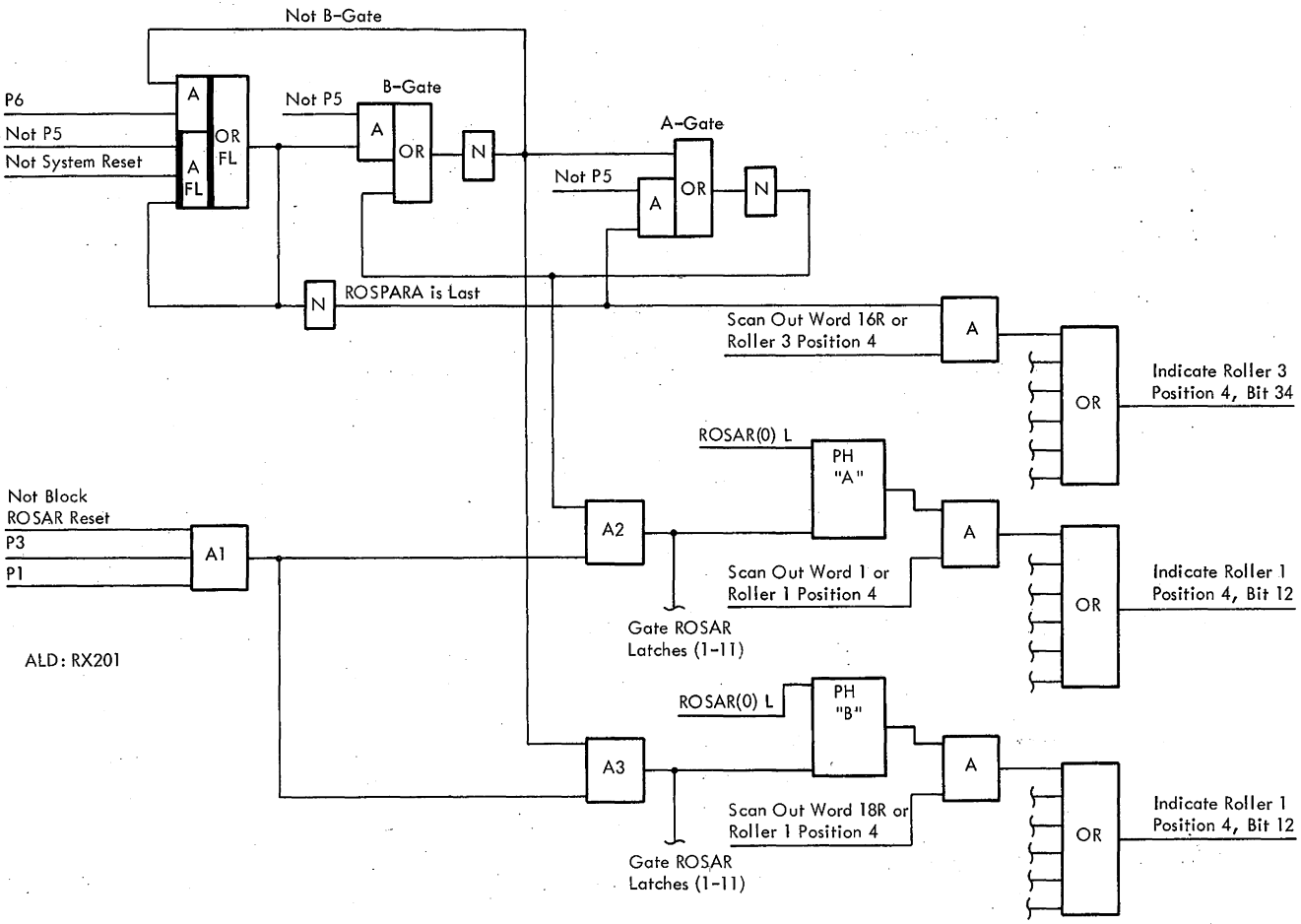


Figure 2-11. Detailed ROS Timing



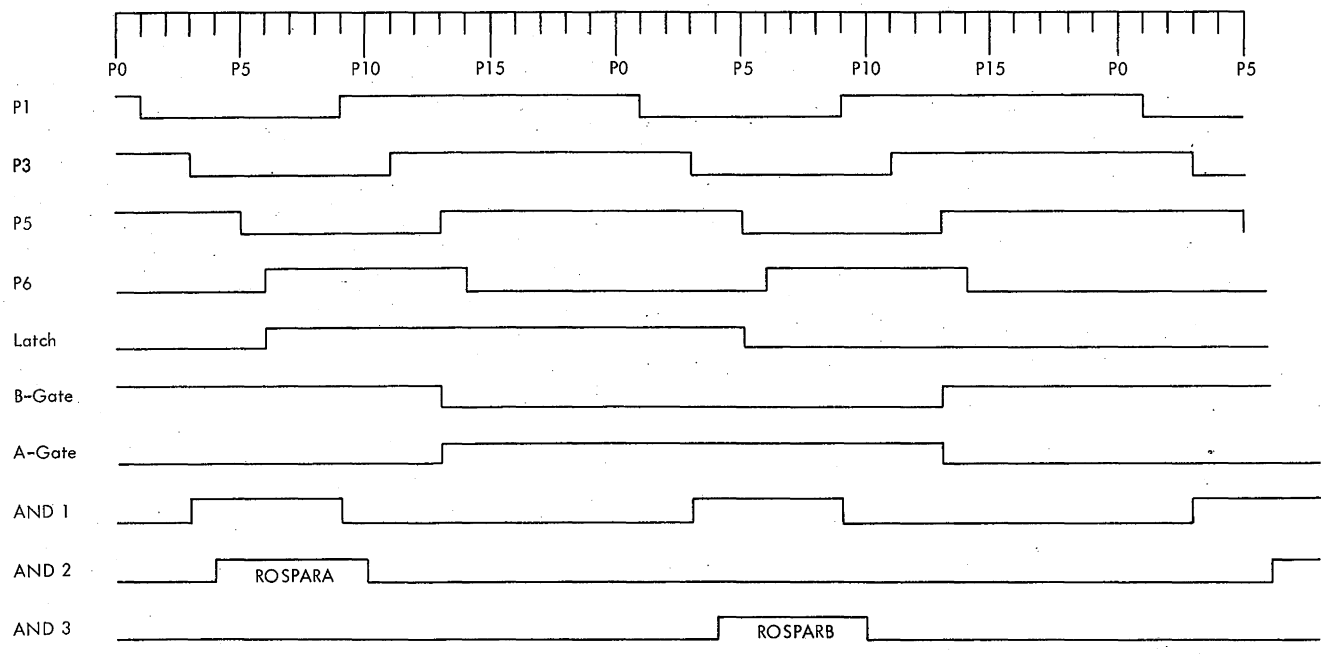


Figure 2-12. ROSPARA and ROSPARB Alternator

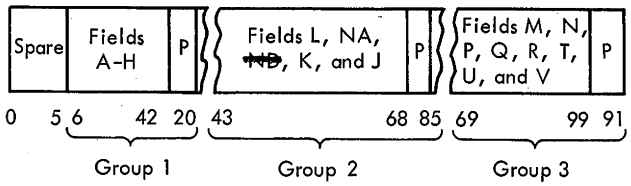
(inverted) is sent to the ROSPARA indicator (roller 3, position 4, bit 34). When the latch is reset, the indicator is on, indicating that ROSPARA contains the address of the current ROS word and that ROSPARB contains the address of the previous ROS word; if off, the contents of ROSPARA and ROSPARB are reversed.

ROS Back-Up Register

The ROS back-up register (ROSBR) holds fields L, NA, K, and J (which are indicated on roller 3, position 4, bits 7–32), and fields R, T, U, and V (which are indicated on roller 4, position 4, bits 17–30). These indicators combined with the ROSDR indicators provide the CE with a picture of ROS word contents when the CPU stops during maintenance (test) mode.

ROS Error Checking

Before each ROS word is decoded, it is checked for correct parity. Parity is checked in three groups from the ROSDR and the ROSBR:



Each group contains its own parity bit and must have an odd number of bits to result in correct parity. (There is also a parity bit for the entire ROS word, bit 0. At the present time, however, this bit is not checked.) Figure 2-13 illustrates ROS parity checking for the three groups: A of the figure shows how parity is checked from the ROSDR for group 1; B of the figure shows how parity is checked from the ROSBR for group 2; C of the figure shows how parity is checked from the ROSDR and ROSBR for group 3.

The clock reset is blocked in that part of the ROSDR or ROSBR containing the failing ROS word. The part or parts not in error are reset, and the next ROS word is gated to its respective register part(s). For example, if bits 43–68 of a ROS word contain an error, the bits are retained for observation. The other two groups not in error (bits 0–42 and 69–99) will change. The groups that change belong to the data word accessed by the failing word when the data register is examined. Thus, when a ROS parity error occurs in one part, the ROS bit indicators on the system control panel comprise bits from two different ROS words.

A ROS parity error also prevents stepping ROSAR, ROSPARA, and ROSPARB, thus enabling the operator to establish the address of the current ROS word, the address of the previous ROS word, and the address of the next ROS word. The address of the previous ROS word should be particularly helpful when parity errors are caused by late ROS branches.

Assume that ROS bit 40 fails. A parity error in bits 6–42 is indicated, and the ROS previous address register indicated the failing word. Reference to the ROS bit plane description shows the expected bit content of the failing word. The incorrect bit (bit 40) can be determined directly by comparing the bit plane description with the indicators.

To summarize, if a machine check is not disabled and a ROS parity error occurs, the parity group in error is not reset at CPU clock time of the next cycle. The CPU clock set-reset signal is blocked to the group that contains the parity error (Diagram 4-107, FEMDM). The new ROS word, however, is gated to the two groups not in error. The bit in error can be determined by displaying the ROS micro-instruction, noting which group of bits is in error, deciding which ROS address is in error, and referring to the listing of ROS micro-instructions. If a ROS parity error occurs and the machine check mask bit [PSW(13)] is set, a logout occurs.

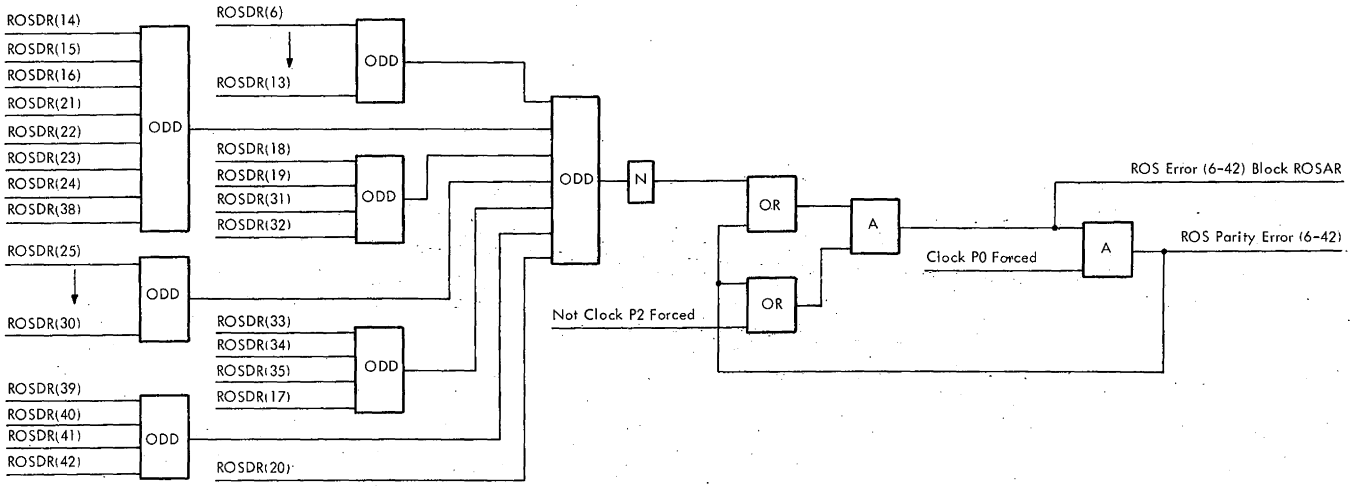
Scan Mode Operations

Scan mode operations affect three fields of ROSDR: field D (bits 17–20), 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.

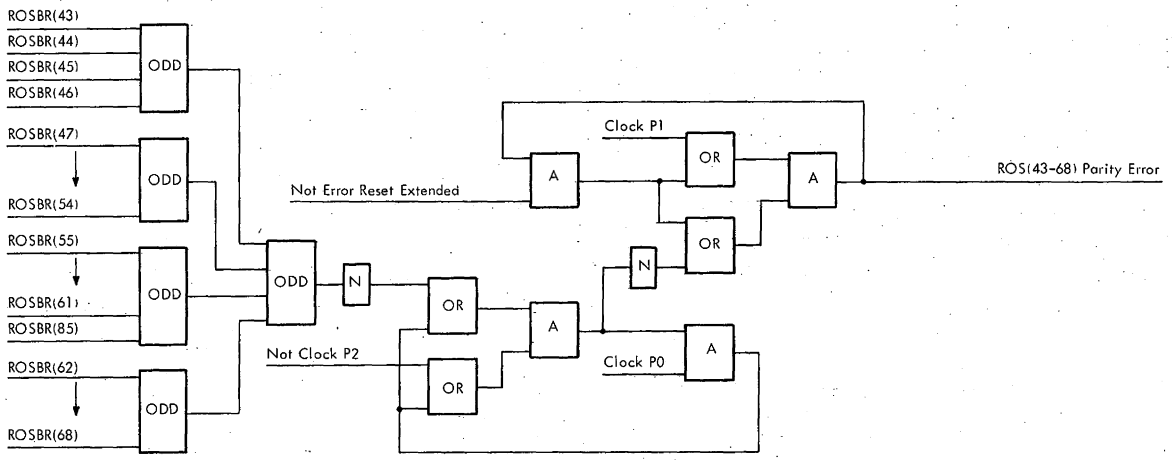
The scan mode is controlled by the 'scan mode' trigger. When the 'scan mode' trigger is reset, the standard decode path is used. When the 'scan mode' trigger is set, however, the standard control lines are blocked and scan control lines (using common CPU control line codes) are activated.

The 'scan mode' trigger can only be set in normal CPU mode and reset only in scan mode. The scan control logic generates an 'inhibit register ingating' signal which is sent to the ROSDR fields to block register inputs and to allow scan control use of the ROS in sequencing through its test operations.

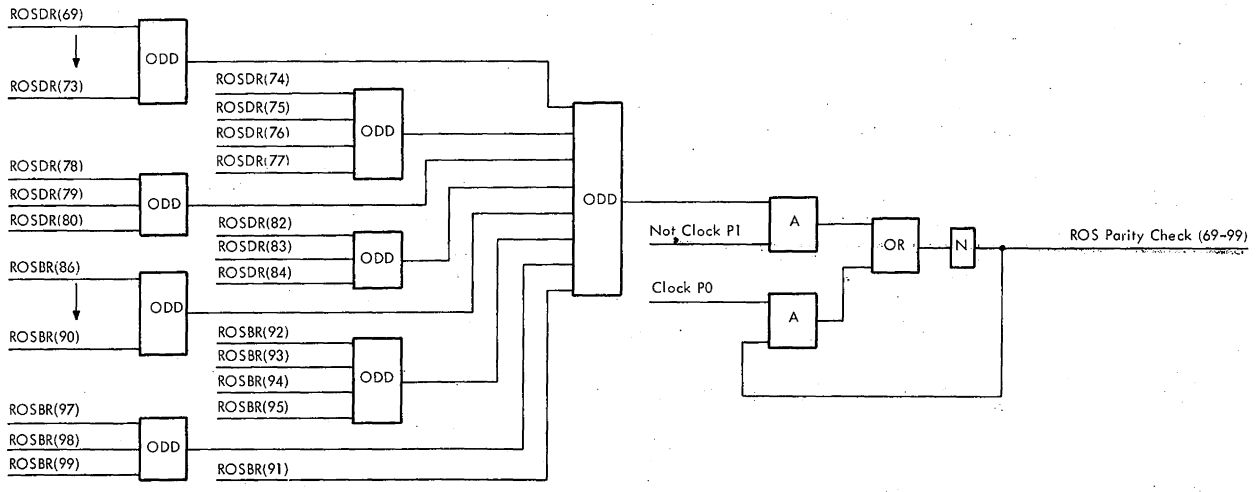
Scan also affects ROS microbranching. (See "Scan Mode Control of ROS" in Section 2 of Chapter 6.)



A. ROSDR(6-42)



B. ROSBR(43-68,85)



C. ROSDR(69-84) AND ROSBR(86-99)

Figure 2-13. ROS Parity Checking

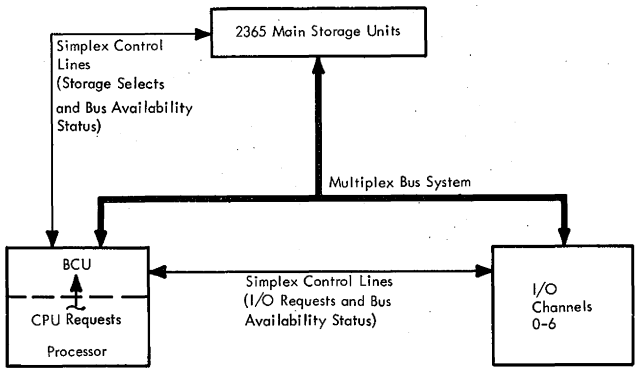
Section 3. Bus Control Unit

The Bus Control Unit (BCU) governs the flow of all information to and from main storage. This unit regulates the flow of addresses, data, key information, and other control signals associated with main storage. In the Model 65 system, the BCU responds to the CPU and to as many as seven I/O channels, all of which are operating asynchronously in respect to each other.

Note: If the Multisystem feature is installed, storage address decoding, 'select' signal generation, and BCU resetting is affected. Refer to Chapter 4, Section 2, for details.

GENERAL DESCRIPTION

The System/360 Model 65 uses a common, or multiplex, bus arrangement to transfer all information to and from main storage:



Each unit requiring access to storage is equipped with a set of receivers and drivers which tap into the multiplex bus. The major function of the BCU is to provide for efficient time-sharing of the multiplex bus by all units. To accomplish this task, each main storage unit and each unit requiring access to main storage communicates with the BCU through individual, or simplex, control lines. These lines are monitored by the BCU to establish priority between the requesting units and to inform the units of the bus availability status.

Basic Interface Considerations

Figure 2-14 is a simplified diagram of the BCU interface with the system. The functions of the major buses and control lines are explained below.

1. Multiplex Buses
   a. Storage Address Bus (SAB). This bus, 21 address lines and 3 parity lines, specifies the address of a doubleword in main storage.
   b. Storage Data Bus Out (SDBO). This bus, 64 data lines and 8 parity lines, carries data sent by the main storage unit.
   c. Storage Data Bus In (SDBI). This bus, 64 data lines and 8 parity lines, carries data sent by the CPU or the channel to main storage.
   d. Mark Bus. This bus, 8 control lines and 1 parity line, designates which data bytes on the SDBI are to be stored into main storage; there is a mark line corresponding to each byte on the SDBI. Complete absence of mark signals on the mark bus occurs only on a fetch operation.
   e. Key-In Bus. This bus, 5 key lines and 1 parity line, is used during storage operations and the set-key operation. These operations transfer the storage protection key from the CPU or channel to the storage protect area in the selected main storage unit.
   f. Key-Out Bus. This bus, 5 key lines and 1 parity line, is used only during the insert-key operation. This operation transfers the storage protection key from main storage to the CPU.
2. Control Lines
   a. Select. Upon selecting the required storage unit, the BCU issues a 'select' signal to that unit. This signal causes the selected unit to perform a storage cycle.
   b. Busy. Throughout the duration of the storage cycle, the storage unit generates a 'busy' signal to the BCU.
   c. CPU Request. Upon requiring access to main storage, the CPU issues a request signal to the BCU.
   d. Channel Request. Upon requiring access to main storage, the channel issues a request signal to the BCU.
   e. BCU-Channel Response. This signal is generated by the BCU to the requesting channel in response to the channel-request signal. This signal indicates that priority has been granted to the channel and that a storage address is now required.
   f. Advance. Indicates to the CPU or channel that the storage unit is about to place data on the SDBO.
   g. Store. This multiplex line indicates to the storage unit that a store operation is in progress.

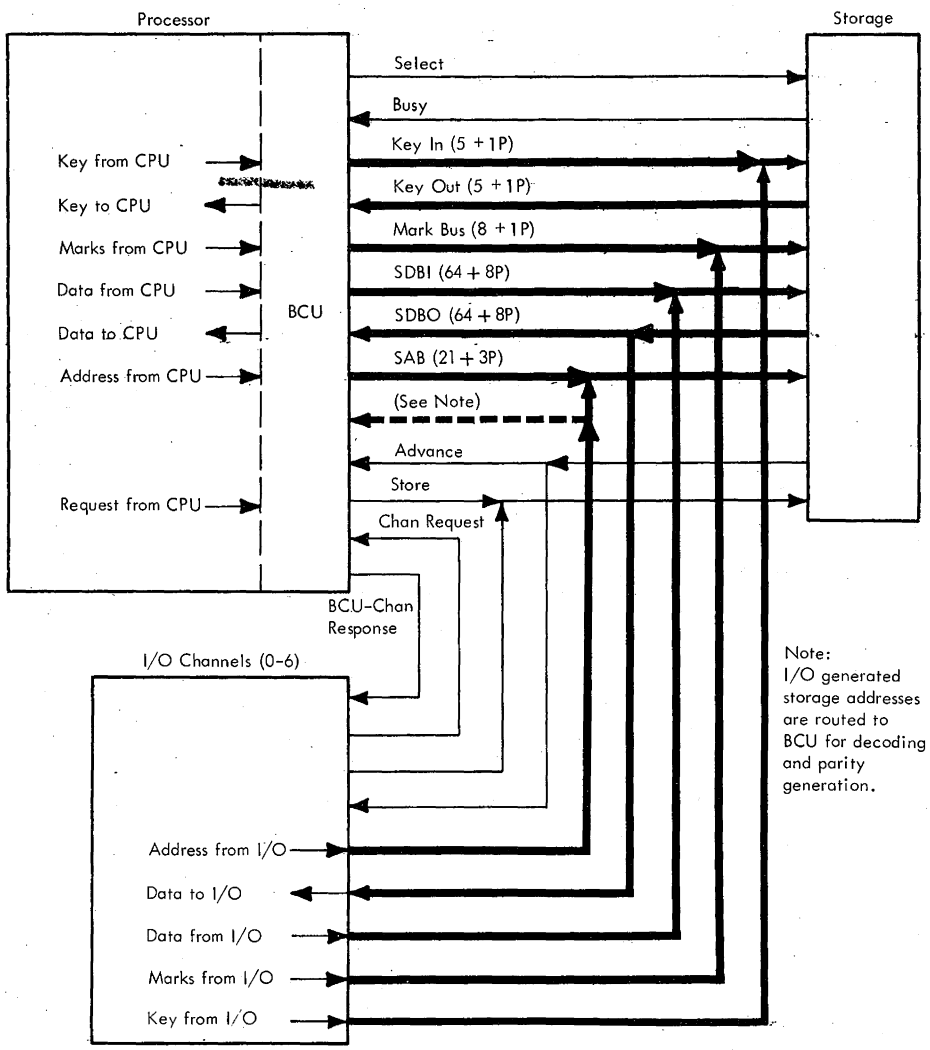


Figure 2-14. Primary BCU Interface Signals

Special multiplex lines for operation with the 2361 Core Storage Unit (Large Capacity Storage or LCS), not shown on Figure 2-14, include:

1. LCS Pre-Advance. Warns the BCU that the LCS is preparing to send data on the SDBO. This signal prevents the start of another request which might interfere with the incoming data.
2. LCS Advance. Precedes the data from the LCS so that a register will be ready to receive the data.
3. Set LCS Priority. Notifies the requesting channel that it should look for an 'LCS advance' signal and ignore the 'advance' signal from the 2365 Core Storage Unit (High-Speed Storage or HSS).

A simplex line, 'LCS operation', from each channel sets the 'channel X waiting' trigger to allow other channels to operate with the HSS while the LCS is busy.

Basic Operating Considerations

The following paragraphs briefly examine the major operational characteristics of the processor, channels, main storage units, and large-capacity storage (LCS) units from a BCU viewpoint.

Operation with Processor

- Requests are issued by (1) I-Fetch, (2) microprogram, (3) Scan.
- CPU clock is stopped if request is not serviced within 3 cycles.

The BCU resides in, and is a logical but independent part of, the CPU. To enable efficient handling of CPU storage requests, the BCU and CPU clocks are synchronized. The BCU is designed to accept storage requests from the CPU every 0.4 usec (every other CPU clock cycle). However, the speed with which these requests are serviced depends on the storage availability and on the amount of channel interference.

The CPU can issue five types of requests to the BCU: fetch data, store data, insert kev. set key, and test and set. Furthermore, the fetch data and store data requests can be issued by various functional areas of the CPU and are categorized accordingly. For example, the fetch data requests can be issued by the I-Fetch control hardware, by the microprogram, and by Scan controls.

Whenever possible, requests from the processor are handled by the BCU in a manner to provide for maximum CPU operation; i.e., processing of data in the CPU is overlapped with handling of the request by the BCU. However, the degree of overlap is limited, and the BCU must stop the CPU clock when a channel request has taken priority, when the storage unit requested by the CPU is busy, and when the data requested by the CPU cannot be retrieved within the required number of cycles.

A need to fetch new data is detected in the processor three or four cycles before this data is required. Accordingly, the CPU issues a 3- or 4-cycle fetch request indicating to the BCU that data will be required after three or four cycles have elapsed from the time of the request (i.e., the transfer of data into the CPU must take place on the fourth or fifth cycle following the request). It is the responsibility of the BCU to ensure that the requested data is present at the SDBO at the specified time. To do this, the BCU allows the CPU clock to cycle three times regardless of whether storage can be accessed within this specified period. If data is not available at the SDBO at the end of the third cycle, the BCU stops the CPU clock. CPU processing is thus halted until its storage request is completed, at which time the BCU restarts the CPU clock (see ALD M8251).

Requests to store data are initiated in the CPU by the microprogram or by the Scan controls; these requests are always accompanied by the 'mark' signals, which designate the bytes to be stored. Store requests are not categorized into the 3- and 4-cycle types because no critical transfer into the CPU is involved; once the storage unit is selected, the CPU no longer depends upon storage unit operations.

The insert-key, set-key, and test-and-set requests are issued by the CPU microprogram. Basically, the insert-key request is a fetch operation to obtain the protection key from main storage. The set-key request is a store operation which transfers the five-bit (plus parity) storage protection key from the CPU into a specified storage protect area of main storage. The test-and-set request is essentially a combined fetch/store operation effected in a single storage cycle.

Operation with I/O Channels

- Requests are issued by up to seven channels.
- Requests are handled in the following order of priority: channels 1, 2, 0, 3, 4, 5, 6; then CPU.

Up to seven channels (six selector plus one multiplexer) are available on the Model 65. All channels may operate at the same time. The channel sustains a maximum data rate when it is servicing a high-speed drum: 1.25 million bytes per second. Because data is transferred between the channels and main storage on an eight-byte basis, the channel makes a storage request every 6.4 usec at this speed. When servicing the 340-kHz and 90-kHz magnetic tape devices, the channel requires access to storage at 23- and 89-usec intervals, respectively.

The channel can issue two types of requests: fetch data or store data. Requests from channels are processed by the BCU on a priority basis. Where there is a conflict between two I/O channels or between an I/O channel and the CPU, the BCU gives priority to the highest-ranking channel. Priority is preassigned in the following manner: Channel 1 has the highest priority, followed by channels 2, 0, 3, 4, 5, and then 6 (channel 0 is the multiplexer channel); when no channel requests remain, a CPU request is honored. Priority is therefore not established on a first-in/first-out basis; instead, as each request is serviced, a priority test is performed again on all channels, including the channel just serviced.

Operation with Main Storage

- BCU allows interleaving of odd and even requests.
- Storage keys protect storage contents.
- Insert key operation fetches key to CPU.
- Set key operation replaces key in main storage.

The 2365 main storage units, also referred to as the high-speed storage (HSS), operate on a basic cycle of 750 ns. Each HSS is divided into an even- and an odd-address storage area equipped with its own SAB, SDBO, SDBI, and 'mark' bus. The access path to these areas is through a common interface (Figure 2-15). This arrangement (internal to the HSS) allows the BCU to interleave even and odd storage requests; i.e., an odd request can be issued by the BCU halfway through an even storage cycle, and vice versa. By interleaving references to even- and odd-address areas, the effective storage cycle approaches one-half the cycle time of the unit. (If desired, the interleaved mode of operation can be defeated to allow operation of a program solely in the odd or even area of storage.)

Each time a storage unit is accessed by the CPU or I/O channels, information is either stored into or fetched from

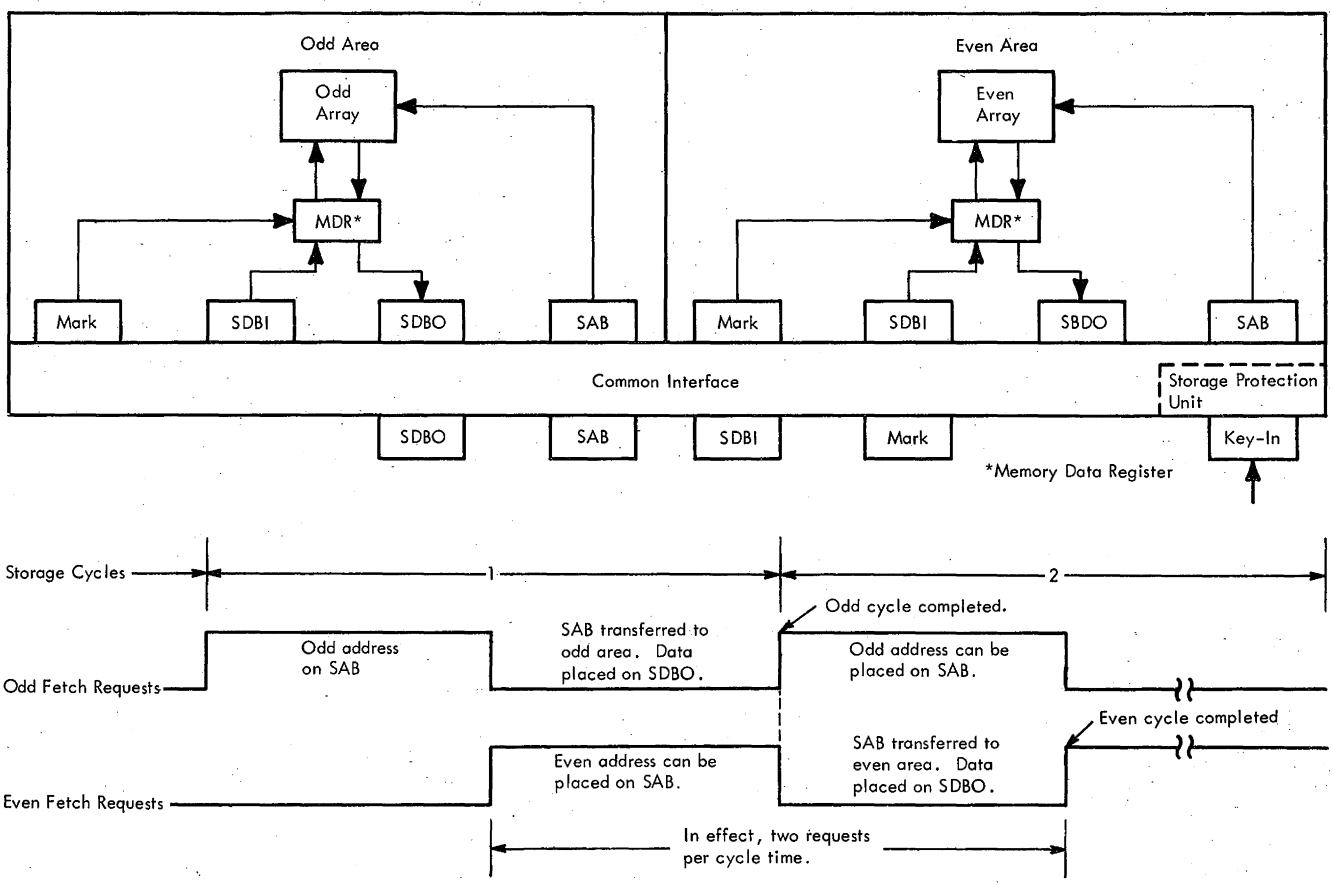


Figure 2-15. Basic Organization of HSS Unit

that unit. A store operation is performed when the data on the SDBI is accompanied by one or more 'mark' signals on the 'mark' bus. There is a 'mark' line for each of the eight bytes that can be placed on the SDBI. The presence of signals on these lines indicates which SDBI bytes are to be stored. Thus, data can be stored selectively by bytes, or, if all marks are active, the entire SDBI contents are stored at the doubleword address specified by the SAB. When the CPU or I/O channel requests a storage cycle and no signals are present on the 'mark' bus, a fetch operation is initiated. In a fetch operation, the doubleword addressed by the SAB is placed on the SDBO and is made available to the requesting unit.

A storage protection capability is provided to protect the contents of main storage from unauthorized use or destruction. Both fetch and store operations are subject to this protection. Protection is implemented by subdividing each storage unit into 2048-byte blocks and assigning a one-byte protection key pattern to each block. The assigned key patterns for the addresses for all 2048-byte blocks within a storage unit are recorded in the storage protection mechanism of that unit. During a storage request, the storage protection mechanism compares the key pattern for the addressed block with the key pattern supplied by the requesting CPU or channel. (Keys are gated from the requesting unit over the 'key in' bus at the beginning of each storage cycle.) A mismatch in keys results in a protection violation; the storage request is not honored, and a 'protect violation' signal is sent to the requesting unit.

The protection key for any 2048-byte block of storage can be fetched from the protection key mechanism and brought into the CPU for inspection. This operation is performed through execution of the Insert Storage Key (ISK) instruction, which issues a fetch request directly into the protection key area of the storage unit. Conversely, the protection key for any 2048-byte block of storage can be changed through execution of the Set Storage Key (SSK) instruction by the CPU. This instruction provides a new key pattern and issues a store request into the protection area.

The ISK and SSK instructions can be executed only in the Supervisor state and are not available to the problem programmer. A programmed protection of storage can be achieved by means of the Test and Set (TS) instruction. When the TS instruction is executed by the CPU, a doubleword is fetched from main storage, and the CPU inspects one byte in that doubleword. The main storage sets the byte inspected by the CPU to all 1's, while the remaining bytes in the doubleword are not changed. Thus

programmed storage protection is achieved in the sense that the CPU can later inspect the first byte of a particular storage block to establish whether this block has been previously processed.

A test-and-set storage request combines aspects of both fetch and store operations; therefore, it requires special handling by the BCU and the storage unit. From the BCU viewpoint, the TS instruction is the only one that generates a 'mark' signal during a fetch request. The BCU sends the 'mark' signal and the 'test and set' signal to the storage unit. During the storage write cycle, all bytes (excluding the byte specified by the 'mark' signal) are regenerated into the core array; all 1's are generated into the byte location specified by the 'mark' signal.

Operation with LCS (Optional Feature)

The LCS is an optional feature that extends the storage capacity of the system. One 2361 Model 1 (1, 048, 576 bytes), or up to four 2361 Model 2's (2, 097, 152 bytes each) may be installed without interleaving. By installing units in pairs, two 2361 Model 1's or two or four 2361 Model 2's, the addressing can be split so that one unit of a pair contains all even addresses and the other unit contains all odd addresses. When these units are addressed by the BCU on an interleaved basis, the normal 2361 internal cycle time of 8 usec is reduced to an effective cycle time of 4 usec.

Basic Control and Timing Considerations

- Requests are recorded by sync trigger/latches.
- Priority test is performed when BCU is not busy.
- Once priority is granted, BCU decodes storage address.
- 'Select' signal is sent to storage if unit is not busy.

The basic scheme for processing storage requests by the BCU is shown in Figure 2-16. Requests from the CPU and I/O channels are entered into the BCU request sensing logic and are recorded by the corresponding sync trigger/latch circuits of the BCU. If the BCU is not busy processing a previous request, it examines the status of the sync latches to perform a priority test. When priority is granted to a particular unit, the priority trigger for that unit is set, and the storage address from the unit is made available to the BCU for decoding. At the completion of the decoding, the BCU initiates a storage cycle by gating the storage address to the SAB and sending a 'select' signal to the even or odd storage area of the selected storage unit. Upon receipt of the 'select' signal, the storage unit proceeds to read out or write in the data at the addressed location.

On a fetch data request, the storage unit gates one doubleword of data onto the SDBO for sampling by the requesting CPU or channel. On a store data request, the storage unit replaces the contents of the addressed location with the data sent over the SDBI from the CPU

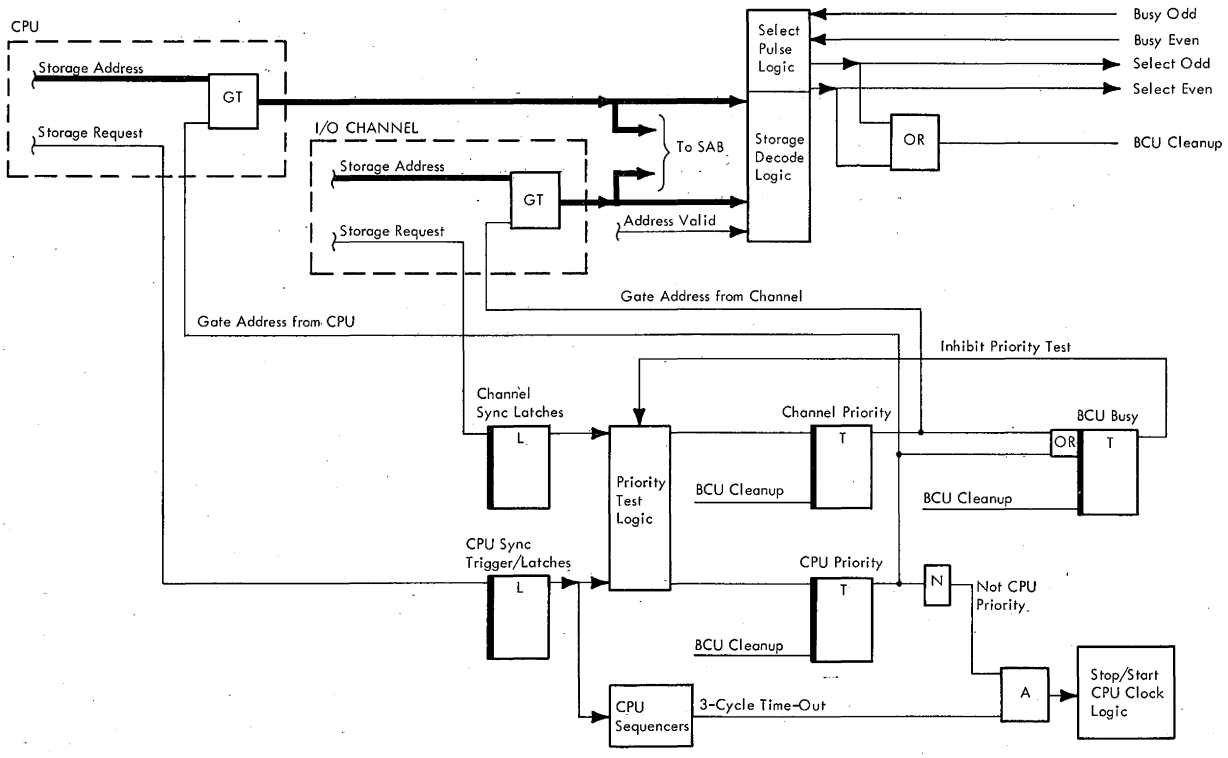


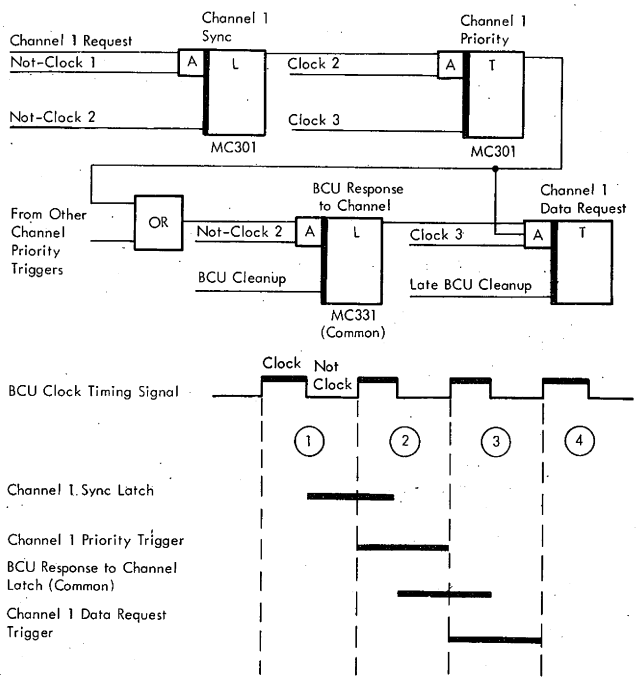
Figure 2-16. Basic BCU Scheme for Processing Storage Requests

or channel. Only those bytes designated by the 'mark' signals are placed into main storage; in the absence of 'marks', bytes already in storage remain unaltered.

During the time when the storage unit is actually performing a fetch or store cycle, it is not available for reselection. To prevent the BCU from doing so, a busy indication is presented to the BCU from either the even or odd storage area, depending on whether the storage unit is performing an even or odd request.

The access time to main storage is 600 ns; i.e., data becomes available from or is stored into main storage 600 ns after the request is issued. The basic timings for the fetch and store operations are shown in Figure 2-17.

To provide the necessary control signals at the correct time, the BCU makes extensive use of trigger/latch circuits. All BCU triggers are set at clock time of the machine timing signal and are reset at the following clock time. Conversely, the latch circuits are set at not-clock time of the timing signal and are reset at the following not-clock time. Thus, BCU timing sequences are implemented (essentially) by sequential shifting of status information through latch-to-trigger-to-latch circuits. A typical arrangement is shown in the adjacent column. Note that the state of a particular trigger or latch at any particular time is indicative of the progress made since the issue of the request:



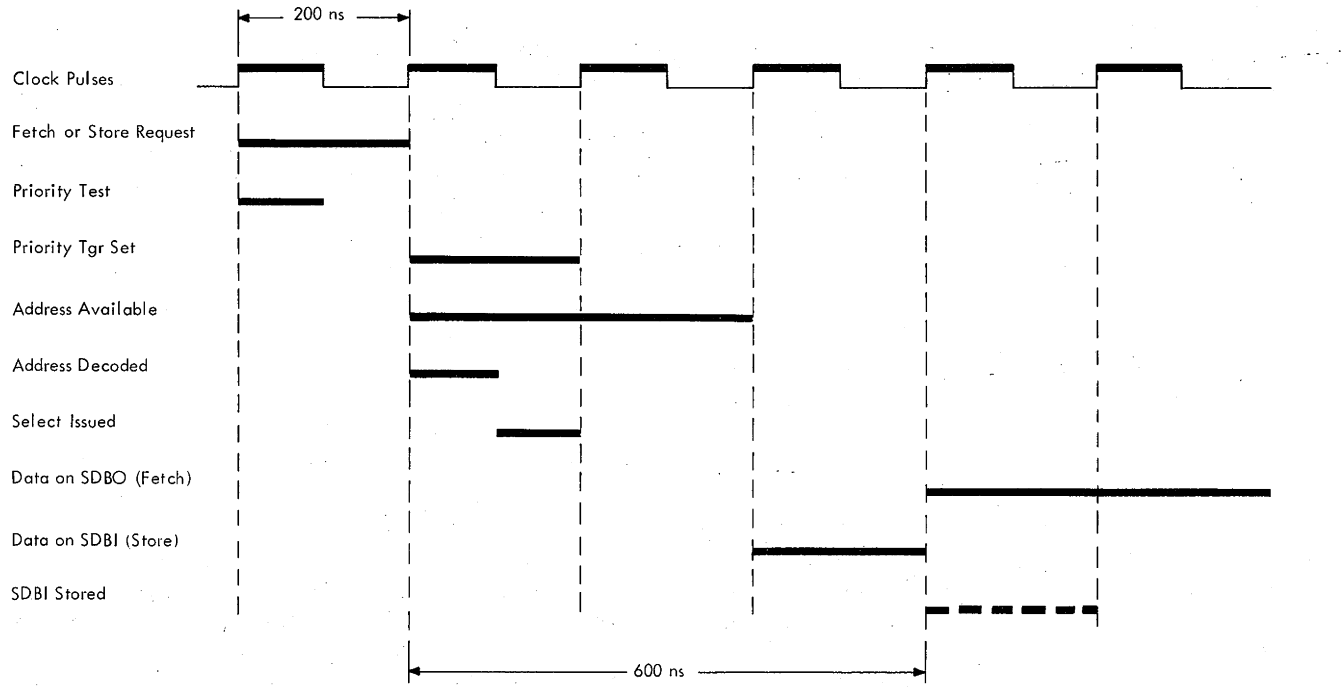
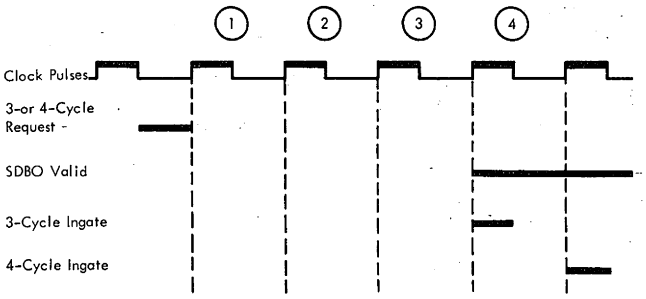


Figure 2-17. Basic Request Timing

As mentioned previously, the CPU can generate either a 3- or 4-cycle fetch request to specify the exact time at which the SDBO is to be gated into the processor. If the CPU is granted priority, and if a successful storage selection has been made, the SDBO becomes "valid" after 3 cycles have elapsed from the time of request. To meet the requirements of a 4-cycle request, the SDBO is held valid for two successive machine cycles. This provision ensures that data will also be present at the SDBO during the 4-cycle sample pulse:



LCS operation is similar to HSS operation, except for the time involved. After the LCS unit is selected, it takes 3 usec for data to return from the selected storage; for about 2 usec of that time the BCU is free to process channel requests. In the case of a CPU fetch request, the CPU clock is stopped until the 'LCS advance' signal restarts it to process the incoming data. In the case of channel requests, the channel waits for the 'LCS advance' signal, but does not hold up other operations.

Basic Operational Sequence

- CPU sequencers count CPU cycles following CPU request.
- Sending of 'select' signal to storage signifies successful completion of request.
- 'Invalid address' signal is generated if 'select' signal is not sent to storage.

The basic BCU operational sequence in handling storage requests from either the CPU or the I/O channels is shown in Figure 2-18. The handling of CPU requests is described first.

The CPU storage requests are issued by one of three functional areas within the processor: I-Fetch logic, ROS microprogram, or scan controls. Once the request is issued, a group of CPU sequencers is activated in the BCU. These sequencers count the CPU cycles that elapse after the request is issued. If the request cannot be completed within a specific time defined by the processor (3 or 4 cycles, depending on the type of request), the sequencers stop the CPU clock to prevent processing in the CPU from advancing beyond the specified cycle. The CPU clock remains stopped until the BCU is able to complete the request.

When the BCU is not busy processing a previous request, a priority test is performed on all the requests (CPU and channel) recorded by the BCU sync latches. After priority is granted to a specific request, the BCU is immediately placed into the busy status to inhibit further priority testing until the accepted request is completed. (During BCU busy status, requests are still received by the BCU and are stored into the latches for consideration on the next priority test.)

If the BCU is not busy, and if no channel requests are pending at the time, the CPU request is granted priority. This action allows the storage address supplied by the CPU to be gated to the SAB. The BCU then decodes the SAB address and generates a 'select' signal to send to the specified storage unit. Whether or not the signal is actually sent depends on the SAB specifying a unit that is within the addressing range of the physical storage and has its power turned on.

If the address supplied by the CPU is outside the physical address range of the installation or specifies a storage unit whose power is down, the 'select' signal is not sent to the storage unit. In this case an 'invalid address' signal is sent to the CPU, and the request is cancelled.

In a case where the unit address is correctly defined and its power turned on, the attempt to issue a 'select' signal will be made. However, the sending of the 'select' signal to the unit may be delayed if the unit is busy servicing a previous request. A busy storage unit causes another selection attempt to be made by the BCU on the following cycle. Continuous selection attempts are thus made until a 'select' signal is sent to storage.

The sending of a 'select' signal to a storage unit signifies to the BCU that a successful selection has taken place. A BCU cleanup operation is then initiated to reset the BCU control circuits and to allow a priority test to be made on pending requests.

Storage requests from the seven I/O channels are handled by the BCU in a sequential order of priority: the highest priority is assigned to channel 1, then to channel 2, then to channel 0 (the multiplexer channel), and then sequentially to channels 3, 4, 5, and 6. When the BCU grants priority to a requesting channel, a BCU response signal is sent to that channel. The channel then gates the storage address to the SAB and responds to the BCU with an 'address valid' signal. Upon receipt of the 'address valid' signal, the BCU decodes the SAB address and attempts to issue a 'select' signal to the specified storage unit. Basically, from this point on, the channel request is handled in the same manner as a CPU request. When the channel is performing a store operation, the 'mark' bus, SDBI, and 'store' signal are gated directly from the channel to the addressed storage unit. If an invalid address is decoded by the BCU during processing of a channel request, an invalid address indication is sent to the channel and to the I/O interrupt logic of the CPU.

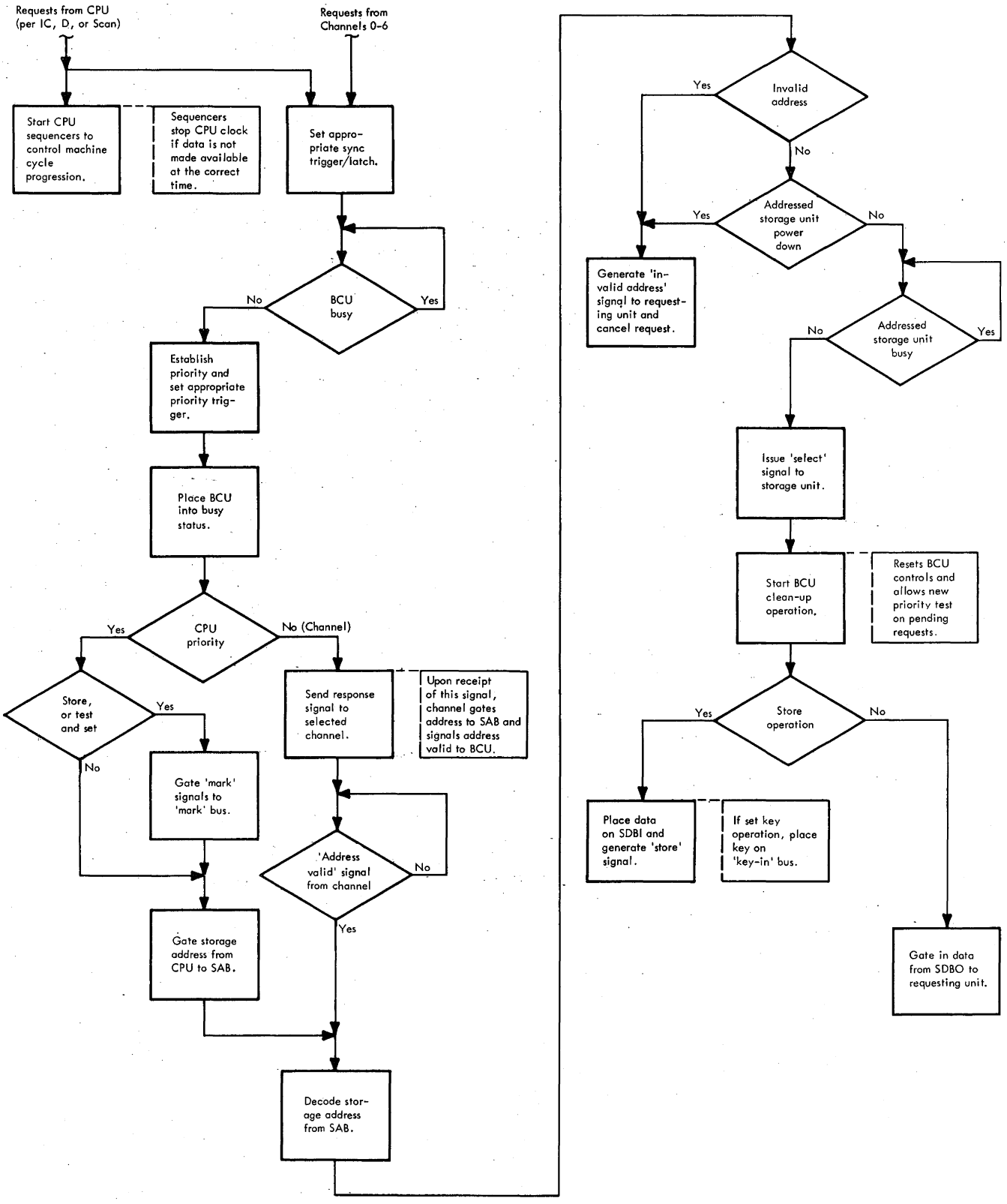


Figure 2-18. Basic BCU Operational Sequence

DETAILED ANALYSIS OF BCU FUNCTIONS

For purposes of discussion, the BCU is divided into a number of functionally distinct logic areas. The subsequent paragraphs describe the functions performed by each area and explain how these functions fit into the overall operational sequence of the BCU (Diagram 4-201, FEMDM).

Initial Handling of Requests

- CPU requests during clock time are recorded by sync triggers.
- CPU requests during not-clock time are recorded by sync latches.
- I/O requests are always recorded by sync latches.

All requests for main storage originate in either the CPU or the I/O channel. The BCU logic used for sensing and recording these requests is shown in Diagram 4-202, FEMDM.

Storage requests from the processor can be issued to the BCU either at clock or not-clock time of the machine cycle. To synchronize the clock and not-clock requests, the BCU employs a trigger/latch sync arrangement. Requests received at clock time are first entered into the BCU sync triggers; they are then propagated (at not-clock time) into the sync latches. Requests received at not-clock time are entered directly into the sync latches. Thus, at the completion of one machine cycle, all requests are reduced to a common time-reference frame.

From the BCU "viewpoint", the storage requests issued by the CPU can be placed into one of three general categories:

1. Requests Generated by Microprogram. These requests are decoded at clock time and, depending on the address source (IC or D), are entered into the corresponding BCU sync triggers. Furthermore, all fetch requests must be specified as being either 3 or 4 cycles in duration. This is to inform the BCU of the specific time at which the requested data must be gated into the processor. The presence or absence of the '3-cycle request' signal from the CPU indicates whether a 3- or 4-cycle fetch has been initiated; i.e., the '3-cycle sync' trigger (in the BCU) is set on all 3-cycle fetch requests and reset on all 4-cycle fetch requests. Upon the setting of the appropriate sync trigger in the BCU, the request is propagated (at not-clock time) into the corresponding sync latch.
2. Requests Generated by I-Fetch Hardware. These requests are decoded at not-clock time and, therefore, are entered directly into the corresponding IC or D latch and the '3-cycle sync' latch.
3. Requests Generated by Scan Hardware. These requests, generated during logout and ROS test operations, are decoded at clock time of the machine cycle. Accordingly, the requests are first entered into the scan-sync trigger and are then propagated into the scan-sync latch at not-clock time. Fetch requests initiated by scan operations are always specified as 4-cycle requests; i.e., the 3-cycle trigger is not set.

At clock time of the machine cycle following the requests, the signals from the sync latches are further propagated into the appropriate request triggers and into the 'CPU request' trigger. The output of the 'CPU request' trigger feeds the priority test logic, which enters the CPU request into priority contention with the channels; it also feeds the CPU sequencer and clock control logic, which stops the CPU clock if the request cannot be handled within the specified time. If the priority test establishes that the CPU request can be handled immediately, the outputs of the IC, D, or scan sync latches and request triggers are used to gate the storage address from the appropriate CPU source (IC, D, or scan address generator) into the BCU.

In conjunction with a request per D, the processor may issue an 'insert key', 'set key', or 'test and set' signal to the BCU. These signals are recorded into the appropriate BCU triggers and are later used to modify the handling of the storage request. Basically, this modification is as follows:

1. The 'insert key' trigger causes an 'insert key' signal to be sent to storage during the handling of the request.
2. The 'set key' trigger issues a 'set key' signal to storage and gates the key bus to storage.
3. The 'test and set' trigger causes a 'test and set' signal to be sent to storage during the handling of the request.

The store-data requests from the CPU are detected by the BCU whenever any mark triggers are set in the processor at the time of the request; a 'store' trigger is then set in the BCU. The output of the 'store' trigger activates a 'store' signal to the selected storage unit and also sets the 'BCU data gate' latch, which gates ST to the SDBI. In addition, the 'mark' signals are transmitted to the selected storage unit via the 'mark' bus. The 'mark' signals specify to the storage unit which bytes on the SDBI are to be entered into the addressed doubleword location.

Storage requests from the I/O channels are made to the BCU by means of channel-request signals. These request signals are detected at not-clock time and are entered into the corresponding BCU sync latches. The sync-latch outputs are in turn applied to the priority test logic to establish whether the particular channel can be granted priority at that time.

The BCU does not distinguish between fetch or store requests from the I/O channels: both types of requests are handled by a common BCU response sequence. When a store-data operation is performed by the channel, the 'mark' bus, SDBI, and 'store' signal are transmitted directly from the channel to the storage unit selected by the BCU.

A channel request to a busy LCS unit energizes a channel lockout circuit. This circuit allows selection of lower priority units while waiting for the completion of the LCS operation. The 'LCS busy' signal forces a priority test to reset the channel priority trigger. The 'channel X waiting' trigger is set, and the channel sync latch is deconditioned by the channel lockout (Diagram 4-202). The channel request remains pending until the 'LCS precomplete' signal resets the 'channel X waiting' trigger, at which time the channel sync latch is set and the channel request is processed.

Establishing Priority

- Priority test sets appropriate priority trigger.
- Once priority is established, BCU is placed in busy status.

A priority test is performed whenever the BCU finishes processing a storage request; it is repeated on each subsequent cycle until the BCU accepts a request and attempts to process it. The BCU is then placed in a busy status, inhibiting further priority tests until that request is completed. When the request is completed, the BCU is again placed in a not-busy state (by a BCU clean-up operation), and a priority test is forced at the beginning of the next machine cycle. The next request is then accepted from the sync latches, or, if no requests are present in the sync latches, a priority test takes place on each following machine cycle until a request is received.

Priority for a waiting storage request is established by transferring the particular request from its sync latch to its associated priority trigger. Setting a priority trigger causes the proper storage address (IC, D, or scan in the case of a CPU request; an I/O channel in the case of a channel request) to be gated to the SAB. On CPU storage requests, processing continues under control of the CPU sequencers, while the BCU selects the addressed storage unit. On channel requests, setting a priority trigger sets a response latch, which in turn initiates the controlling request/response sequence between the BCU and the I/O channel.

The priority triggers for the CPU and I/O channels (Diagram 4-203, FEMDM) are implemented so that there is a definite sequence in which they can be set on successive priority-test operations: the setting of a priority trigger is inhibited if a higher-order priority trigger is about to be set. Thus, only one priority trigger can be set at a given time, with channel 1 having highest priority and the CPU rating lowest priority. Note also that when a priority test is made all requests received up to the point of the test are considered, including requests for the unit just serviced. For example, if two requests are present at the time a priority test is made, the request having highest priority is serviced, and a priority test is again initiated. If, however, another request for the unit just serviced has been received during the interim, that unit is again serviced, and the lower-priority request is kept waiting.

Gating the Address to SAB

- Address gating is initiated by set priority trigger.

When priority is established for either the CPU or the I/O channel, the set state of the corresponding priority trigger initiates gating of the storage address into the BCU. The address gating logic is illustrated in Diagram 4-204, FEMDM.

When the 'CPU priority' trigger is set, gating of the storage address is under control of the IC, D, or scan sync latches. If the storage address is in the process of being transferred into IC or D at the time the CPU has been granted priority, the PAL gate trigger permits a direct transfer of the address from PAL to SAB.

When an I/O channel has been granted priority, the corresponding priority trigger is set for that channel. A 'BCU response' signal is then sent to the channel to initiate the gating of the storage address to the SAB. An 'address valid' signal is then returned to the BCU from that channel to indicate that the address is on the SAB and to start the storage sequence. A channel-data-request trigger for that channel is set on the next cycle, signalling the requesting channel to place data to be stored on the SDBI.

Stopping the CPU Clock

- CPU sequencers control distribution of clock timing within CPU during handling of CPU requests.
- CPU sequencers are started at clock time of cycle following request.
- Sequence in which sequencers are stepped varies with request being processed:
  1. 4-cycle fetch: 'CPU 2' trigger, 'CPU 2' latch, 'CPU 3' trigger, 'CPU 3' latch, 'CPU 4' trigger, 'CPU 4' latch, 'CPU 5' trigger, 'CPU 5' latch.
  2. 3-cycle fetch: 'CPU 2' trigger, 'CPU 2' latch, 'CPU 3' trigger, 'CPU 4' latch, 'CPU 5' trigger, 'CPU 5' latch.

3. Store or set key: 'CPU 2' trigger, 'CPU 2' latch, 'CPU 3' trigger.
4. Insert key: 'CPU 5' trigger, 'CPU 5' latch.

- Conditions that stop CPU clock:
  1. 'CPU 2' latch is set, and either CPU has not received priority or storage is busy.
  2. 'CPU 4' latch is set and CPU did not receive 'advance' signal from storage. (This function is disabled if LCS feature is not installed.)
  3. Insert-key operation.

A group of CPU sequencers (four trigger/latch combinations) is used in the BCU to control CPU cycle progression after each CPU storage request. These sequencers control the 'stop CPU clock' trigger in the BCU. The 'stop CPU clock' trigger has direct control of the CPU clock: setting this trigger stops the CPU (on the following cycle); resetting this trigger starts the CPU (on the following cycle). Diagram 4-205, FEMDM, shows the CPU sequencers and the control logic for the 'stop CPU clock' trigger. The sequencers are started on the CPU cycle following a storage request and are advanced by the subsequent CPU clock signals: the 'CPU 2' trigger and latch are set during the first CPU cycle following the request; the 'CPU 3' trigger and latch are set during the second CPU cycle following the request; and so on.

The 'stop CPU clock' trigger is so implemented that if its reset logic is active its set logic is prevented from setting the trigger. Because of this method of implementation (as seen in Diagram 4-205), if the 'CPU 2' latch is set (first cycle following a storage request), the 'stop CPU clock' trigger will be set on the next cycle (second cycle following the request), and the CPU clock will be inhibited from performing the third processing cycle. This clock stopping sequence occurs during both fetch and store operations, retaining the storage address in the IC, D, or scan controls until the BCU and the storage unit become available. When the BCU and storage unit become available, and when the CPU is awarded priority, the 'stop CPU clock' trigger is reset by the BCU 'cleanup' signal. Once the CPU clock is started, both CPU processing and further sequencer stepping is continued. (If LCS units are attached to the Model 65, the CPU clock is stopped again at latch-4 time provided the advance signal from storage is not received.)

The 'CPU 5' trigger and 'CPU 5' latch are held in the set condition when the 'LCS request' signal sets the 'stop CPU clock' trigger. The CPU is inhibited for about 3 usec until the 'LCS advance' signal resets the 'stop CPU clock' trigger (Figure 2-19). The 'LCS pre-advance' signal prepares to restart the CPU. First, the 'advance waiting' trigger is set to prevent honoring another request by diverting 'issue a select' signals to the 'storage 1' trigger. The 'storage 1' latch retains the 'select' signal until the 'LCS advance' signal resets the 'advance waiting' trigger, thus allowing the 'select' signal to set the 'storage 2' trigger. The 'LCS advance' signal also resets the 'stop CPU clock' trigger if the 'CPU 5' latch is set. The clock is restarted, data arrives on the SDBO, and requests are honored in the normal manner again.

For store data operations, the CPU sequencers are started in the normal manner. However, the output of the 'store' latch modifies the subsequent sequencer stepping as follows: 'CPU 2' trigger and 'CPU 2' latch are set on the first cycle following the request; 'CPU 3' trigger is set on the second cycle following the request. Further sequencer advance is inhibited during store operations because ingating is not required.

Selecting the Storage Unit

- Storage address must correspond to physical storage unit.
- Storage capacity is defined by pluggable jumper cards in CPU.

The capacity of main storage (also referred to as the high-speed storage, or HSS) varies with the model:

Model	Capacity	Description
G65	131,072 bytes	1 storage unit without interleaving capability
H65	262,144 bytes	1 storage unit with interleaving capability
I65	524,288 bytes	2 storage units with interleaving capability
IH65	786,432 bytes	3 storage units with interleaving capability
J65	1,048,576 bytes	4 storage units with interleaving capability

If the system is equipped with more than one storage unit (Models I65, IH65, or J65), the starting address for each unit varies in increments of 262,144 bytes; i.e., addresses 0–262,143 are assigned to unit 1, 262,144–524,287 to unit 2, and so on. Thus, when a storage address is supplied to the BCU, the BCU must select the physical storage unit referred to by this address. Furthermore, if interleaving is desired, the BCU must establish whether the address pertains to the odd or the even area of the selected unit. Once the physical storage unit and the odd or even area within it have been established, the BCU issues an 'odd-' or 'even-select' signal to that unit. This signal commands the storage unit to sample the address on the SAB.

Two identical address decoder circuits are provided in the BCU: one circuit decodes the addresses supplied by the CPU, and the other circuit decodes the addresses

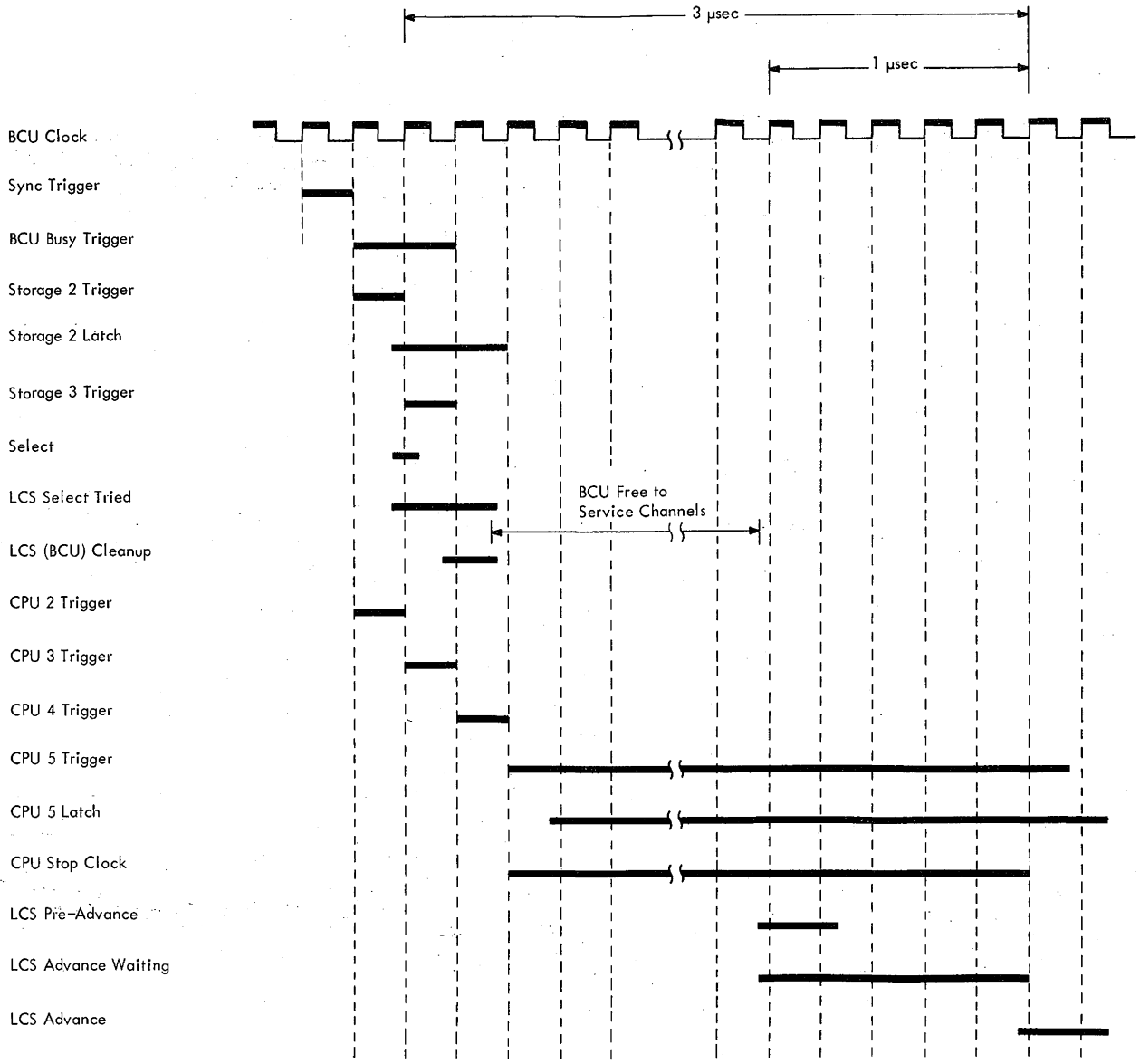


Figure 2-19. Typical Timing for CPU Fetch Request to LCS

supplied by the I/O channels. The need for two decoders (Figure 2-20) is due to a time lag between the CPU- and channel-supplied addresses. Note that a CPU-supplied address bypasses the driver/receivers of the multiplex bus and is applied directly to the address decoder; a channel-supplied address must pass through the channel drivers and through the BCU receivers before being applied to the address decoder. The time lag introduced by the driver/receivers in the multiplex bus is approximately 55 ns.

The optional LCS unit(s) can increase the storage capacity by from 1,048,576 bytes to 8,388,608 bytes. When one or more LCS units are used, it is the function of the BCU to establish whether the particular storage address refers to the HSS or the LCS and to select the correct unit within each type. To perform this function, separate HSS and LCS decoding logic is used in the BCU. Figure 2-21 shows the basic scheme used for selection of the correct storage unit.

At the time the system is set up, pluggable jumper cards are set and inserted into the CPU to define the capacity of the HSS and the LCS and to customize the address decoding logic to the available storage.

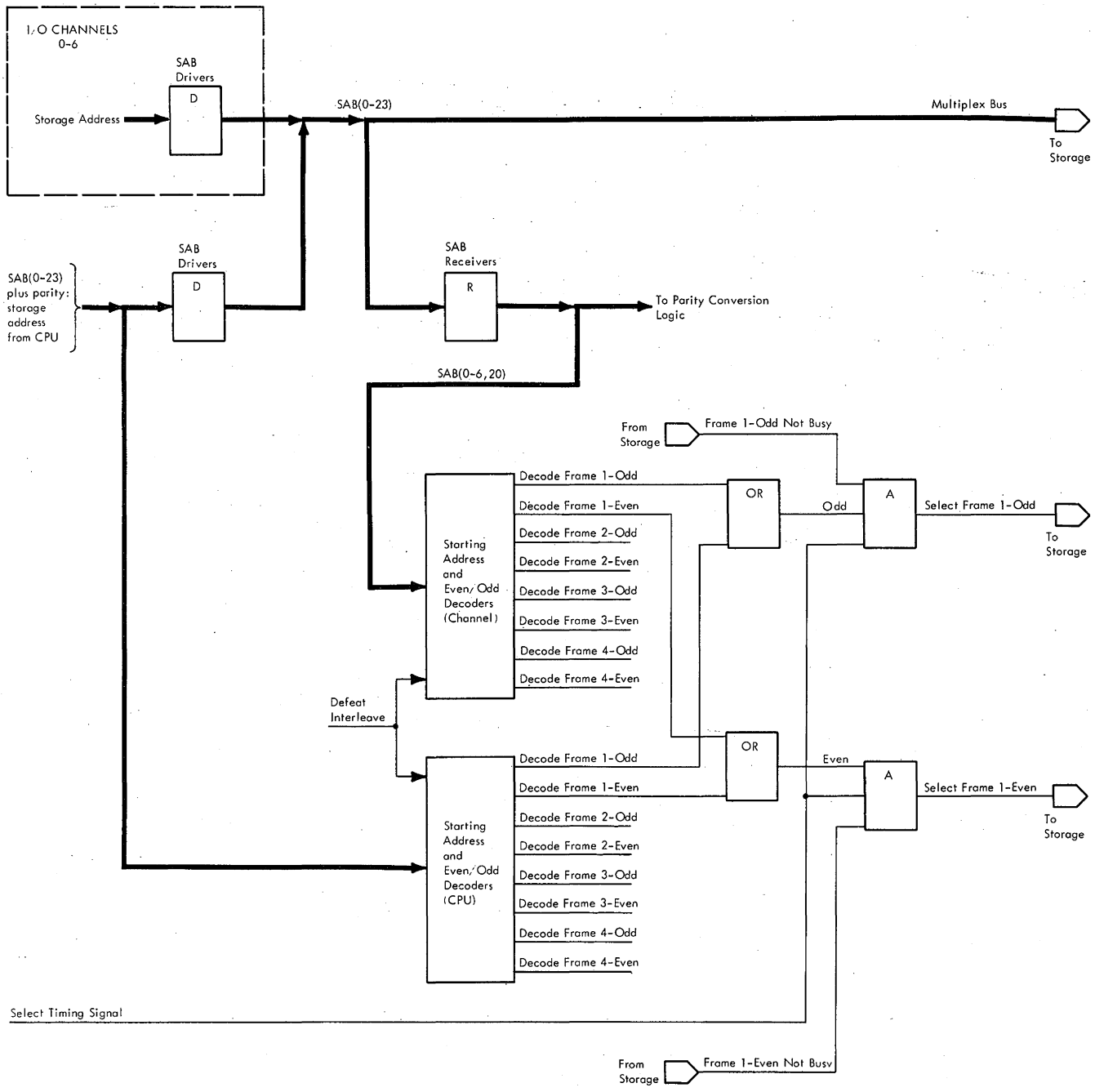
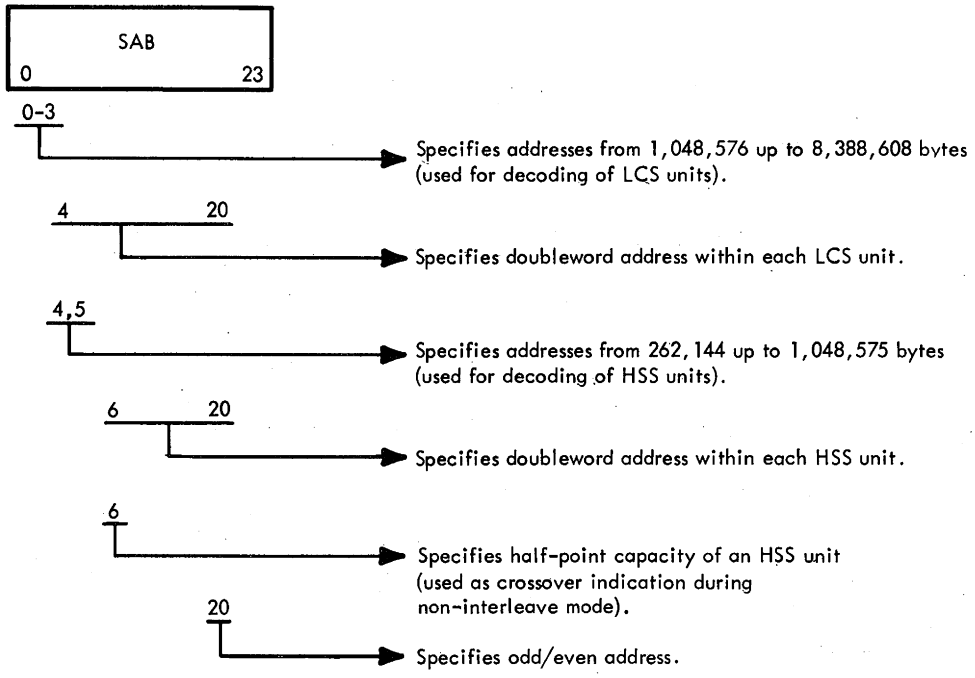


Figure 2-20. Gating of Storage Address From CPU and Channels to Address Decoders in BCU

Each HSS unit of 262,144 bytes is divided into two 131,072-byte areas: even and odd. In normal processing operations, HSS accesses are interleaved between the two storage areas, permitting a higher data transfer rate than is possible without interleaving. Selection of the even or odd storage area within the HSS unit is determined by the setting of the DEFEAT INTERLEAVING switch on the system control panel and SAB(6) or SAB(20).

LCS units may also be interleaved to reduce access time. When interleaving is used, the first LCS unit is accessed (about 3 usec) and the system is released although the LCS unit must be unavailable an additional 5 usec while it is completing its cycle. However, a different LCS unit may be addressed immediately after the 'LCS advance' signal. Because the access time is less than half the cycle time of the LCS unit, the effective access rate is reduced to 4 usec in interleaved mode.

The address decode logic used for decoding CPU- and channel-supplied addresses is shown in Diagram 4-206, FEMDM. When the DEFEAT INTERLEAVING switch is



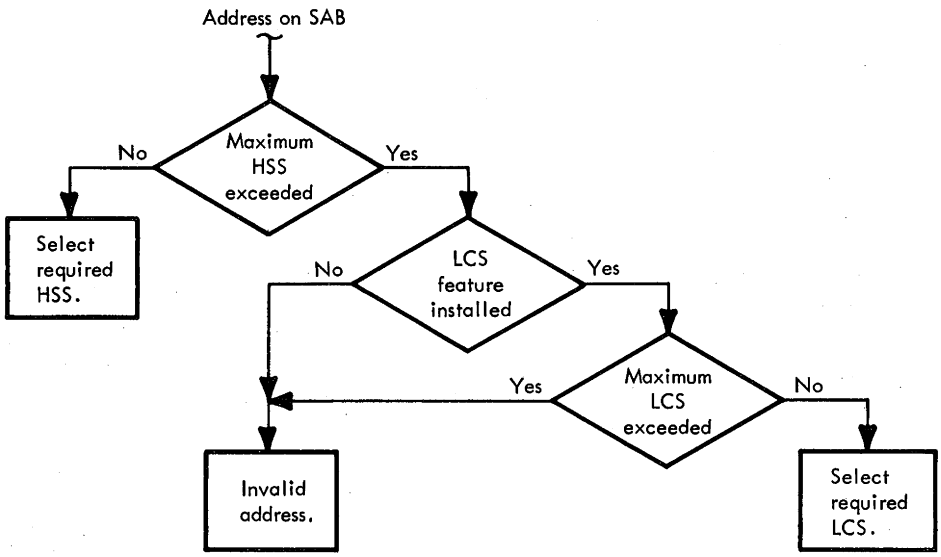


Figure 2-21. Selection of Correct Storage Unit

set to the normal or PROC position, SAB(20) determines the even or odd storage area: if SAB(20) = 0, then even; if SAB(20) = 1, then odd. When the switch is set to the NO REV position (defeat interleaving and no reversal of the even and odd storage areas), SAB(6) determines the even or odd storage area: if SAB(6) = 0, then even; if SAB(6) = 1, then odd. However, the function of SAB(6) is reversed if the switch is set to the REV position (defeat interleaving with reversal of storage addresses). In this case, SAB(6) = 0 specifies an odd storage area; SAB(6) = 1, an even storage area.

The LCS addresses are decoded in 1,048,576-byte groups, the first of which is contiguous with the last HSS address. If the available HSS capacity is exceeded, the 'select' signals to HSS units are blocked, and the LCS byte group is determined by SAB(0–3) and an above/below detector. The above/below detector is conditioned by pluggable jumper cards defining the HSS byte limit and, depending on the HSS byte limit, by SAB(4) or SAB(4,5).

The following examples show this operation. Assume a

524,288-byte HSS capacity and a 2,097,152-byte LCS capacity. Assume, also, an address of 1,179,648 is on the SAB [SAB(3) and SAB(6) = 1]. The HSS 'select' signals are blocked because the HSS has been exceeded and the 'below' signal is generated. The 'below' signal results from the 524,288-byte HSS limit signal being active (pluggable jumper card) and SAB(4) = 0. The LCS byte group gating finds SAB(0–3) = 0001 and the 'below' signal present; the only LCS byte group requiring these conditions is the first. Therefore, the address 1,179,648 is in the first 1,048,576 LCS byte group. (The address boundaries for this first group are 524,288 and 1,572,863.)

Converting SAB Parity

- Generate parity bit P(A) for SAB(6–12).
- Generate parity bit P(B) for SAB(13–19).

The SAB-parity-conversion logic subtracts bits 0–5 and bits 20–23 from SAB(0–23). On the basis of this subtraction, two new parity bits are generated: P(A) for SAB(6–12), and P(B) for SAB(13–19). In generating these new parity bits, the parity-conversion logic must take into account the setting of the DEFEAT INTERLEAVING switch on the system control panel. If this switch is not in the PROC position, it will cause the storage unit to reverse SAB(6) and SAB(20). Accordingly, the same bit reversal must be performed by the parity-conversion logic in generating P(A) and P(B) parity bits.

The basic decoding performed in the BCU and in the storage unit to select a specific core location is shown in Figure 2-22. Note that in the storage unit, only SAB(6 or 20) and SAB(7–19) are gated into the odd or even memory address register (MAR); i.e., the address of any doubleword location in the odd or even area of the storage unit is specified by 14 address bits.

During operation in the interleave mode, SAB(6–19) remains constant for two storage cycles, and two consecutive storage accesses are made per the same address: once from the even and once from the odd area of the storage unit. The selection of the odd/even area is made in the BCU by decoding SAB(20); this bit changes once for each storage access.

During operation in the defeat-interleave mode, SAB(6) and SAB(20) are reversed: the BCU now decodes SAB(6) to select the odd/even storage area; and the storage unit gates SAB(20) as the highest-order bit in the address of a core location.

From a system standpoint, SAB(6) designates the one-half capacity of a storage unit; or, the total capacity of an odd- or even-area in the storage unit. Accordingly, the BCU continues to issue 'select' signals to the *same* even, or the *same* odd, storage area until all core locations in that area are accessed; i.e., SAB(6) changes from 1 to 0, or vice versa.

From a system standpoint, SAB(20) designates the doubleword boundary and is updated for each storage request. Because the storage unit substitutes SAB(20) for SAB(6), the highest-order address bit (within the selected odd/even area) is changed on each storage cycle. This changing leads to interleaved accesses which are restricted *solely* to the odd or to even area of the storage unit; i.e., storage accesses alternate between the upper and lower halves of the area.

The SAB-parity-conversion logic is shown in Figure 2-23 and in Diagram 4-207, FEMDM. This logic utilizes Exclusive-OR circuits to generate, subtract, and add SAB parity bits until the required P(A) and P(B) results are obtained. Note that the output of an Exclusive-OR always excludes those bits which are applied simultaneously on *both* circuits inputs. Thus, depending on the input bits, an Exclusive-OR can be used as an adder or as a subtractor. When the parity bit of SAB(0–3) is combined with SAB P(0–3,6 or 20,7) at an Exclusive-OR, the result is SAB P(6 or 20,7) and SAB P(0–3) has been canceled or subtracted. SAB P(6 or 20,7) is then combined with SAB P(8–12) to produce the parity of SAB(6 or 20,7–12) by addition of parity. In this manner, the parity of SAB(0–23) is converted to parity of SAB(6 or 20,7–19) and then sent to the storage unit.

Generating 'Select' Signal to Storage

When priority is granted to either the CPU or the channel, the BCU attempts to issue a 'select' signal to storage (Diagram 4-208, FEMDM). The manner in which the 'select' signal is generated depends on whether the LCS feature is installed in the system.

Generating 'Select' Signal if LCS Is Not in System

- 'Select' signal is initiated by 'CPU priority' signal for CPU requests.
- 'Select' signal is initiated by 'address valid' signal for channel requests.

The 'select' signal logic is activated by the 'CPU priority' signal or, in the case of channel requests, by the 'address valid' signal. If no previous selection attempt has been made ('storage 2' latch not set), detection of the 'CPU priority' or 'address valid' signal results in the setting of the 'storage 2' trigger. The 'storage 2' trigger sets the 'storage 2' latch to gate a 135-ns 'select' timing signal to the final gating logic for HSS units 1 through 4. This 135-ns timing signal is sent to storage if the unit power is on, if a valid unit address is decoded, and if the odd/even area of the unit is not busy servicing a previous request.

The absence of storage power or the failure to decode a valid storage address results in an invalid address condition. (Refer to "Detection of Invalid Address.") However, if the 'select' signal is not sent due to a busy storage

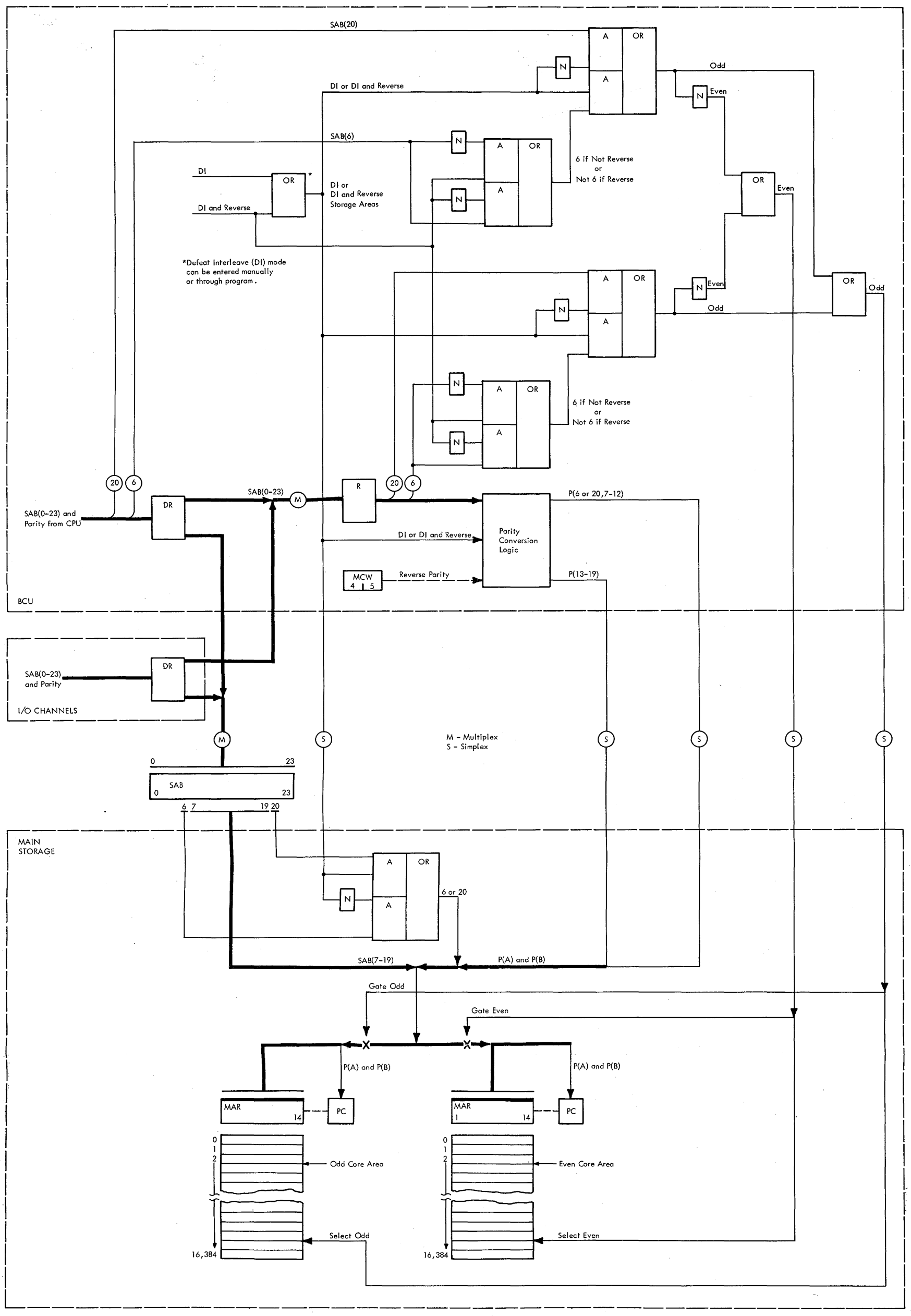


Figure 2-22. Basic SAB Decoding Circuits in BCU and HSS Unit

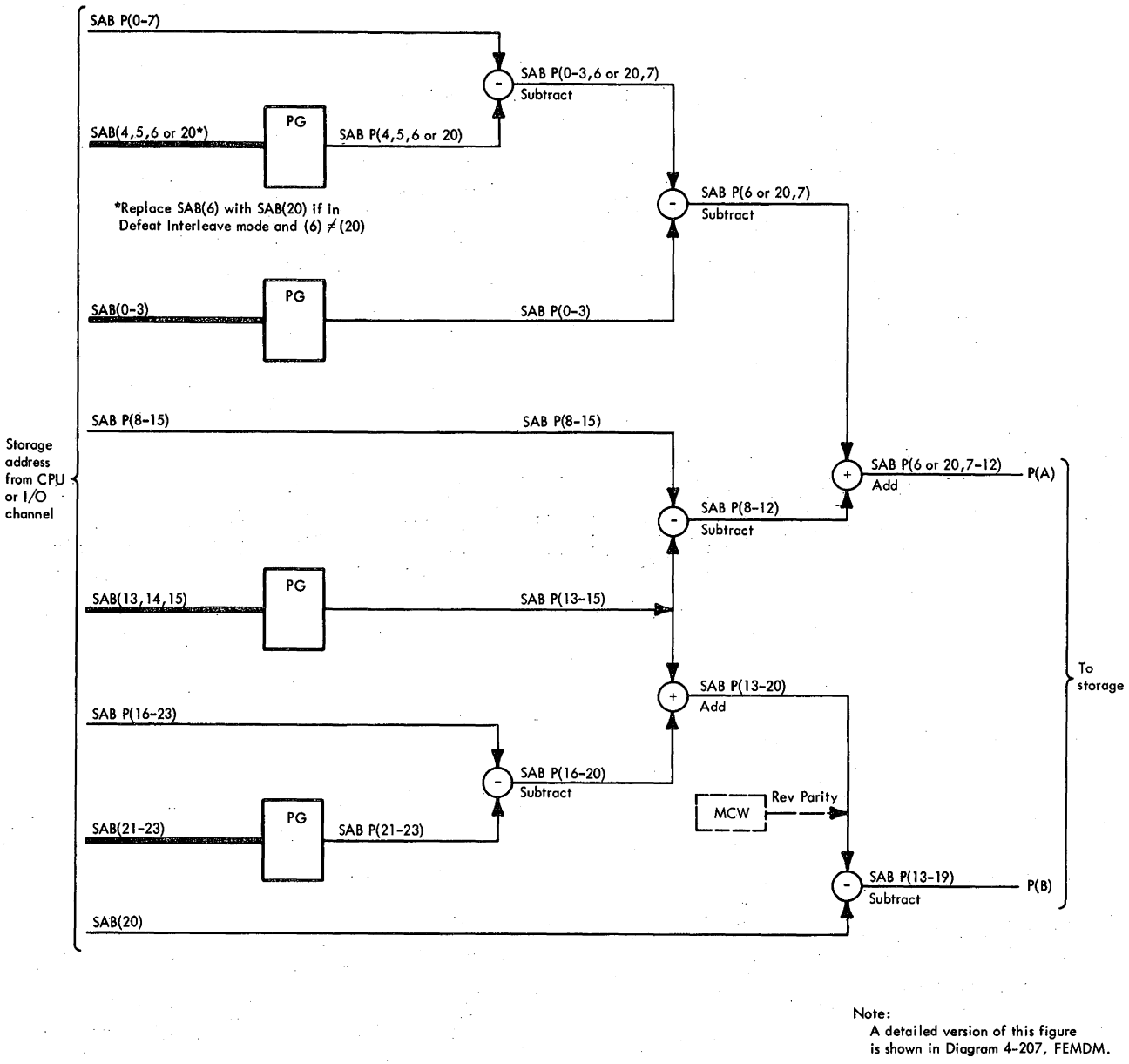


Figure 2-23. SAB Parity Conversion Logic

condition, the 'HSS select sent' trigger is not set. The reset state of this trigger and the set state of the 'storage 2' latch initiates another selection attempt. Successive attempts are thus made until the storage unit becomes not busy and a 'select' signal is sent to that unit. The sending of the 'select' signal to storage initiates a BCU cleanup operation which restarts the CPU clock and resets the BCU control circuits for handling of the next request.

Generating 'Select' Signal if LCS Is in System

- 'Storage 2' trigger is set only when storage not busy.

Detection of the 'CPU priority' or of the 'address valid' signal by the select logic results in setting of either 'storage 1' or 'storage 2' trigger, depending on the availability status of the storage data bus. The bus availability status is specified by the state of the 'advance waiting' trigger: if this trigger is reset (no storage advance), the bus is available; if set, unavailable.

If the storage bus is available, the 'storage 2' trigger is set; however, if the storage bus is not available, the 'storage 1' trigger is set. Note that once the 'storage 1' trigger is set, it forms a loop with the 'storage 1' latch to continue trying to set the 'storage 2' trigger. When the 'advance' signal is received from the storage using the data bus, the 'advance waiting' trigger is reset, allowing the 'storage 2' trigger to be set.

The set state of the 'storage 2' trigger results in sending the 'select' signal to the HSS or to the LCS. The conditions for LCS selection are similar to those described for HSS. However, one major difference should be noted: the 'storage 3' trigger maintains the 'storage 2' latch set for two successive cycles to compensate for a 1-cycle delay in starting the invalid address test. This delay is due to the need of sampling the state of the 'advance waiting' trigger during LCS selection.

Detection of Invalid Address

- Address is outside physical storage capacity.
- Power is down on storage unit.

An invalid address condition is detected in the BCU if power is off in the addressed storage unit or if the address supplied by the requesting unit exceeds the physical storage capacity of the system. There are two invalid address detection circuits: one to detect an invalid address from the CPU, and another to detect an invalid address from the channel. The 'invalid address' signal has a dual function: to detect an invalid address and to select an address decoder. For a channel request, an 'invalid address' signal disables the CPU decoder; for a CPU request, a similar signal disables the channel decoder. The invalid address logic is shown in Diagram 4-209, FEMDM.

The BCU detects an invalid address by examining the states of the 'test for invalid address' trigger and the 'select tried' latch: if both are set, the address is valid; if the 'test for invalid address' trigger is set but the 'select tried' latch is reset, the address is invalid. The subsequent paragraphs analyze the conditions that lead to detection of an invalid address and describe the BCU operation that follows.

The 'test for invalid address' trigger is set when a 'select' signal is issued to the final decode-gating logic of the BCU. The output of this trigger then monitors whether the 'select' signal is successfully passed by the gating logic. If the address of a storage unit has been defined correctly, and if its power is turned on, the 'select' signal is passed through the gating logic to set the 'select tried' latch.

The set state of the 'select tried' latch signifies that a successful storage selection has been made; i.e., a 'select' signal will be issued to storage even though the sending of the signal may be delayed if the unit is busy servicing a previous request. Thus, this condition indicates that a valid storage address has been decoded.

If the 'select' signal fails to pass through the final decode-gating logic, the 'select tried' latch is not set. This condition indicates to the BCU that a successful storage selection is not possible due to an incorrect address on the bus or a power failure in the storage unit.

Upon detection of an invalid address, the BCU alerts the requesting CPU or channel of the error condition and then proceeds to force a 'select' signal to storage. (Detection of an invalid address during I-Fetch does not necessarily result in a program interruption.)

The forced storage selection is made to generate a 'BCU cleanup' signal, which will allow the CPU to continue (CPU clock was stopped) or the channel to continue. The BCU forces a request to any storage unit that has power, in the ascending order: starting with unit 1 and, if unit 1 power is off, proceeding to units 2, 3, and 4. The storage unit accessed by a forced 'select' signal, however, is not allowed to complete its cycle because the BCU also issues a 'cancel' signal to that unit. The 'cancel' signal prevents storage data transfer to the SDBO; thus 0's are transferred to the selected register in the channel or the CPU.

Recording of Error Indications from Storage

- Storage errors are address check, data check, protection check.

If a storage request causes an error within the selected storage unit, an error indication is sent from that unit to the BCU. The storage-error-recording logic in the BCU is shown in Diagram 4-210, FEMDM. Note that the PROC CHK indicator is activated only if a CPU request is responsible for the storage error; the STOR CHK indicator is activated if *either* a processor or channel request has caused the error. (If a storage error is received when the CPU CHECK switch is set to the PROC position and the machine check mask bit (bit 13) is a 1, it causes a logout operation.)

The storage-error indicators (Stor Adr Chk and Stor Data Chk) are activated as a result of one of the following error conditions:

1. Storage Address Check. A parity error has been detected on either SAB or the 'mark' bus. (On store or set-key operations, the original data is regenerated into the addressed storage location to prevent loss of data.)
2. Storage Data Check. A parity error has been detected on the SDBI, the SDBO, or the storage-key bus.

In addition to the address-check and data-check error conditions, the storage unit also generates a protection check if the key supplied by the request either is not zero or does not match the protection key assigned to the storage area being addressed. (Upon detection of this condition, the storage unit cancels the request to prevent loss of data.) A protection-check indication from storage is routed via the BCU and applied directly to the CPU I-Fetch-and-interrupt logic.

Resetting of BCU Logic

- Sending of 'select' signal to storage initiates the following reset signals: 'channel accept', 'BCU cleanup', 'late BCU cleanup', 'late BCU cleanup for CPU request'.

The BCU-reset logic is shown in Diagram 4-211, FEMDM. After the BCU sends a 'select' signal to storage, this logic generates the following reset signals: 'channel accept', 'BCU cleanup', 'late BCU cleanup', and 'late BCU cleanup for CPU request'.

The reset signals are initiated as a result of the 'pseudo accept' signal; i.e., a signal which simulates an "accept" condition from storage. The start of the 'pseudo accept' signal is controlled by routing the 'select timing signal' signal through a number of inverters in the BCU which provide a total delay of approximately 80 ns. The delayed 'select timing signal' signal is then AND'ed with the 'select sent' signal to set the 'accept' latch. The output of the 'accept' latch initiates the 'channel accept' and 'BCU cleanup' signals.

The reason for delaying the 'BCU cleanup' signal (approximately 80 ns after the 'select timing signal' signal) is to prevent a new priority test from taking place on the same cycle in which the 'select' signal is issued to storage. (This condition could result in two successive 'select' signals being issued less than 400 ns apart; i.e., an early 'BCU cleanup' signal would reset the 'BCU busy' trigger before the 'BCU busy' latch had time to latch on; the resulting priority test could initiate another 'select' signal on the following BCU cycle.)

The time duration for which the 'accept' latch stays set must be such as to "extend" the 'BCU cleanup' signal past B2 time of the following clock cycle. This condition is necessary to provide the following functions:

1. Inhibit setting of the 'stop CPU clock' trigger, if the set state of the 'CPU 2' latch coincides with the issue of the 'select' signal. (The 'stop CPU clock' trigger is so implemented that if its reset logic is active, its set logic is prevented from setting the trigger; see Diagram 4-205, FEMDM.)
2. Inhibit setting of the 'test for invalid address' trigger; i.e., when a successful 'select' signal is generated, the 'BCU cleanup' signal blocks the invalid address test (Diagram 4-209, FEMDM).

The 'late BCU cleanup' and 'late BCU cleanup for CPU request' signals are generated approximately 130 ns after the 'accept' latch is set. These signals clear all BCU functional circuits associated with the previous request to allow servicing of a new request on the following cycle: the 'late BCU cleanup' signal resets all channel-priority and data-request triggers; the 'late BCU cleanup for CPU request' signal resets the 'CPU priority' trigger and removes the CPU address from the multiplex bus.

Resetting of BCU logic on an LCS operation is delayed one cycle (200 ns) because the 'accept' signal from the storage (LCS) is used instead of the 'pseudo accept' signal. This delay allows time to test the busy condition of the LCS unit. Figure 2-24 shows the logic to develop the 'LCS cleanup' signal.

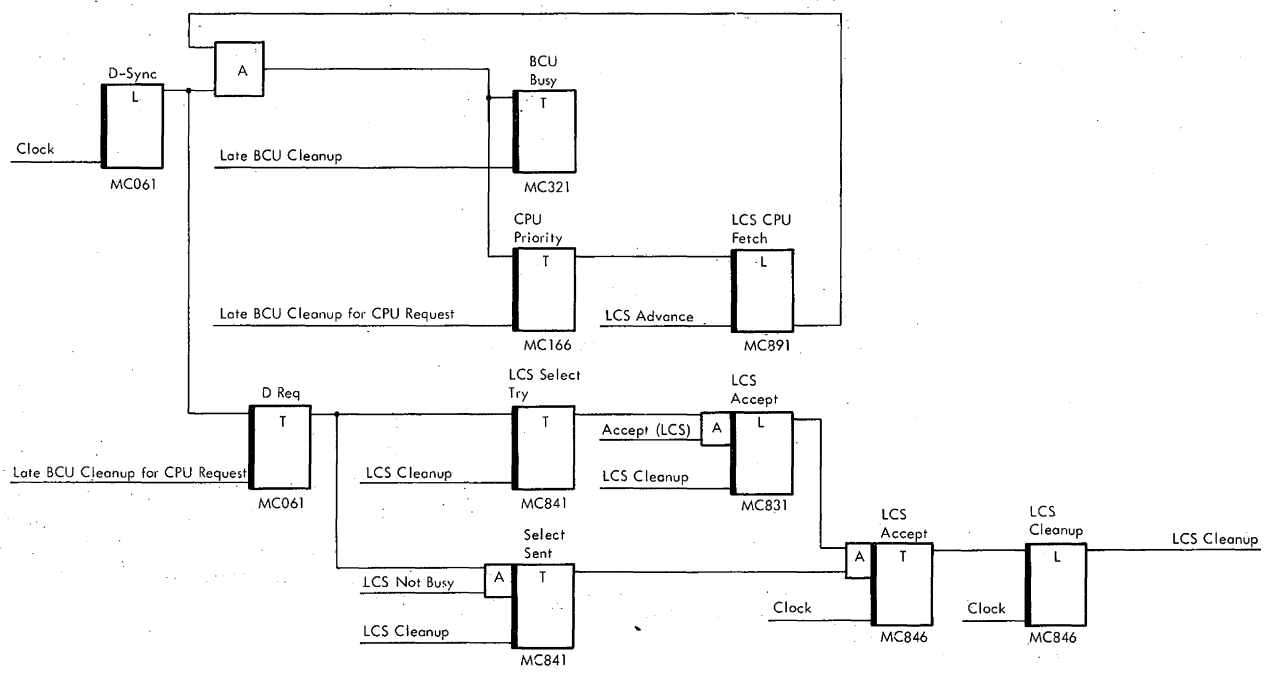


Figure 2-24. BCU Reset for LCS Operation

DETAILED ANALYSIS OF BCU OPERATIONS

The subsequent paragraphs describe the operational sequences performed in the BCU during processing of storage requests for the CPU or channel.

CPU Storage Requests

The functional sequence for CPU storage requests is illustrated in Diagram 4-201, FEMDM.

CPU storage requests are issued to the BCU from I-Fetch, ROS, and Scan logic, and both the storage request and the resulting data transfer are overlapped with processing. Although all CPU requests are handled by the BCU as basic fetch or store requests, the following variations exist: 3- and 4-cycle fetch, store, insert-key, set-key, test-and-set, and single-cycle operation. Processing of the request is interrupted when either the storage unit or the BCU is busy, or when the requested storage data is not available in the specified time.

3- and 4-Cycle Fetch Operations

- Ingating of requested storage data is specified at 3 or 4 cycles following storage request.

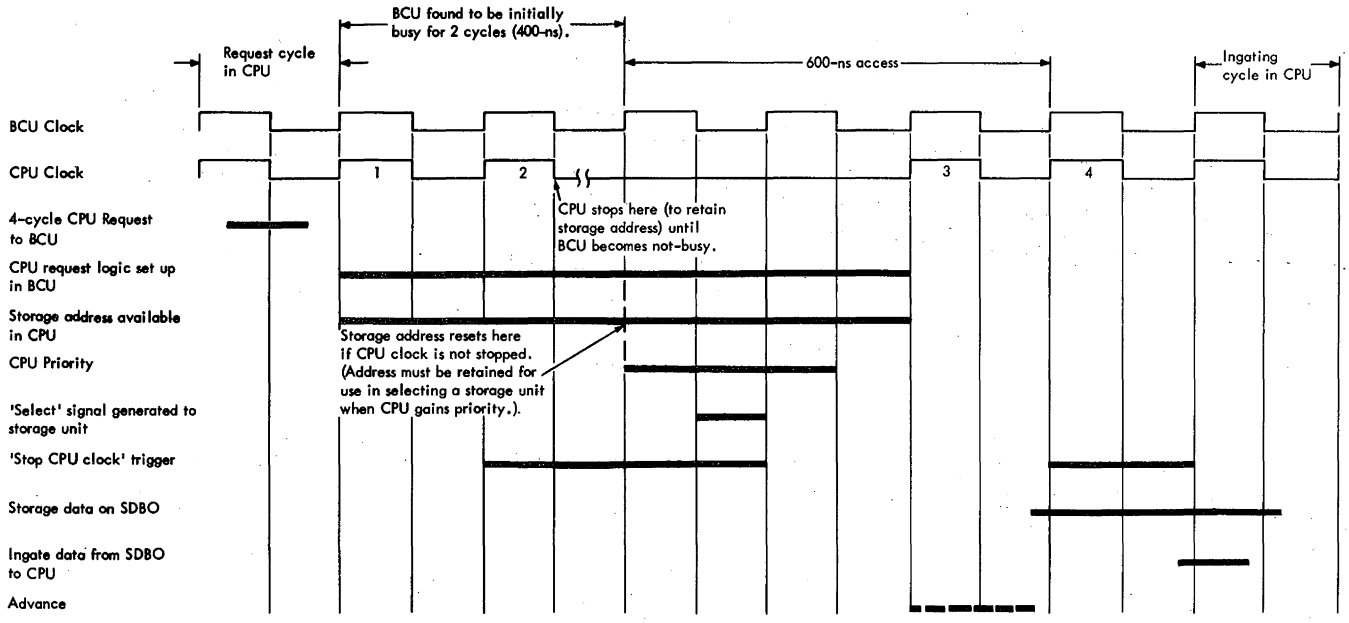
Because the BCU operates at the same machine cycle speed as the CPU, and because the access time to storage requires 3 cycles, the CPU is allowed to continue processing for 3 cycles following the request. The BCU must keep track of the CPU cycle progression so that the SDBO ingating is executed at the correct time. If data is not present at the SDBO at the end of 3 cycles, or if the SDBO is to be ingated following the 4th cycle, the CPU clock is stopped. The requested data may not be present on the SDBO for one of the following reasons:

1. A BCU-busy condition, wherein priority cannot be established immediately because of channel interference.
2. A storage-busy condition, wherein the addressed storage unit is still servicing a previous request.
3. Storage access time limitations, wherein a request has been made to an LCS unit (optional feature).

Figure 2-25 illustrates a typical clock control timing sequence for a 4-cycle fetch (with the BCU initially busy for 2 cycles). For detailed timing charts of 3- and 4-cycle fetch operations, refer to ALD's M8221, M8241, and M8251.

Store Operation

Store-data requests are always made per D. Data from ST is gated to the SDBI for transfer to the storage unit. Mark trigger settings are transferred to the storage unit (via the 'mark' bus) to specify which of the eight bytes of data are to be stored. Processing of the store request is interrupted



Notes:
1. 4-cycle fetch to 750-ns storage unit (600-ns access) with BCU initially busy for 2 cycles (400-ns).
2. CPU clock is shown stopped at the end of the 2nd cycle following the request cycle in CPU.
   Reason for stopping: to retain storage address for use in selecting a storage unit when BCU is found to be busy (store or fetch operations).

Figure 2-25. Typical CPU Clock Stopping Sequence

if the BCU or the storage unit is busy. For detailed timing of the store operation, refer to ALD M8211. Note that the 'store' signal prepares the storage unit for storing data.

Insert-Key Operation

- 'D-storage request' and 'insert key' signals are sent from ROS to BCU.
- Basic fetch operation is performed by BCU.
- Five-bit (plus parity) storage protection key is transferred from storage to CPU via 'key out' bus.

Insert-key operations are essentially fetch requests per D, in which a five-bit (plus parity) storage protection key is obtained from the specified storage protection area of main storage and inserted into F(0–4) of the CPU. These operations enable the CPU to examine the key patterns used by the storage protection mechanism.

An insert-key request sets the 'insert key' trigger in the BCU to modify the normal stepping of the CPU sequencers: the 'CPU 5' trigger is set on the first cycle following the insert-key request, and further sequencer stepping is not performed. The 'stop CPU clock' trigger is also set on the first cycle following the insert-key request. Thus, the CPU clock is stopped on the second cycle after the request.

When the BCU awards priority to the CPU, the contents of D are gated to the SAB. A 'select' signal is then generated and sent to the addressed storage unit together with an 'insert key' signal. In the storage unit, the two signals ('select' and 'insert key') initiate an insert-key operation. The seven high-order bits of the SAB are decoded to determine the protection key location, and the key pattern (five bits plus parity) is fetched from that location. The storage unit then generates an 'advance key' signal, which prepares the BCU for ingating of the 'key out' bus into the F-register of the CPU. A detailed timing chart for the insert-key operation is shown on ALD M8211.

Set-Key Operation

- 'D-storage request' and 'set key' signals are generated from ROS to BCU.
- Basic store operation is performed by BCU.
- Five-bit (plus parity) storage key is transferred from CPU to storage via 'key in' bus.

Set-key operations are essentially store requests per D in which a five-bit (plus parity) storage protection key is obtained from F(0–4) in the CPU and stored into the specified storage protection area of main storage. These operations enable the CPU to set new key patterns into the storage protection mechanism.

A set-key request sets the 'set key' trigger in the BCU. In addition, all CPU mark triggers are set during the set-key operation. Receipt of the D-storage request by the BCU then sets the 'store' trigger and starts the CPU sequencers in the normal manner. When the CPU is awarded priority, the contents of D are gated to the SAB, and the 'select' and 'set key' signals are sent to the addressed storage unit. (The sending of the 'store' signal to the storage unit is inhibited during the set-key operation.)

The 'select' and 'set key' signals initiate a set-key operation in the storage unit. The seven high-order bits of the SAB are decoded to determine the protection key location, and the 'key in' bus is gated into that location. For a detailed timing chart of the set-key operation, see ALD M8211.

Test-and-Set Operation

Although a test-and-set request combines aspects of both fetch and store operations, the basic handling of the request by the BCU is similar to a 3-cycle fetch per D. The major difference is that the BCU sends one mark and a 'test and set' signal to the storage unit specified by the D-address. The basic test-and-set timing sequence is as shown on ALD M8221.

Single-Cycle Operation

- START pushbutton provides for manual stepping through CPU cycles.
- RATE switch in SINGLE CYCLE STORAGE INHIBIT provides for manual stepping through all cycles within request sequence. (Storage unit is not selected; data transfer is inhibited.)
- RATE switch in SINGLE CYCLE enables CPU to run automatically from time 'select' signal is sent to storage until data transfer operation is completed.

When CPU operations are being tested in the single-cycle mode, the CPU clock is stepped manually; one CPU clock cycle results for each depression of START. The BCU clock, however, is not affected by the single-cycle mode and runs automatically, thus allowing the BCU to continue servicing storage requests from the CPU or from the I/O channels.

The single-cycle operation can be performed by the CPU with or without access to storage: if the RATE switch is placed in SINGLE CYCLE, the CPU must access storage whenever it steps through a cycle specifying a storage request; if the RATE switch is in SINGLE CYCLE STORAGE INHIBIT, all storage requests are ignored by the BCU.

When servicing storage requests from the CPU in the single-cycle mode, the BCU must ensure that the gating of data to or from the CPU is synchronized with the storage

unit operation. To accomplish this function, special single-cycle logic in the BCU controls the CPU clock and runs it automatically whenever synchronized ingating is required. Refer to Diagram 4-212, FEMDM, a functional flowchart of BCU operations during servicing of CPU requests in the single-cycle mode.

To enable manual stepping through as many CPU cycles as possible, the BCU delays sending the 'select' signal to storage until the time when the CPU sequencers stop the CPU clock. (This delay is implemented by blocking CPU priority, when in the single-cycle mode, until the CPU clock is stopped.) The stop-clock condition indicates to the BCU that the data transfer between CPU and storage must be executed on the next depression of START.

At this point, the nature of the request is of primary consideration. If a fetch-data request is in progress, the BCU must override the single-cycle controls and run the CPU clock automatically until the CPU executes the ROS word with the 'ingate SDBO' micro-order. If a store-data request is in progress, the BCU need not control the CPU clock because the CPU data is placed on the SDBI when CPU priority is established and a 'select' signal is sent to storage; i.e., as soon as the START pushbutton is depressed. Thus, on store-type requests, the operator can single-cycle through every ROS word of the CPU microprogram; on fetch-type requests, the operation automatically skips over one or two ROS words, depending on whether a 3- or 4-cycle request is specified. (Note that the time slice between two consecutive depressions of START does not enter into consideration; this time slice is much greater than the 600-ns time interval required to access storage.)

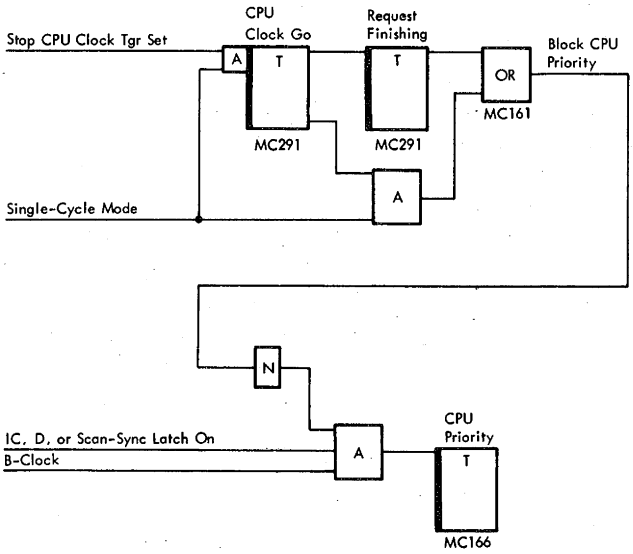
When START is depressed and a 'select' signal is sent to storage, the resulting 'BCU cleanup' signal restarts the CPU clock. The state of the CPU sequencers, after the first CPU clock signal is generated, indicates the type of request in progress and whether additional stepping of the CPU clock is required; this stepping is performed automatically under control of the 'CPU clock go' trigger. Note that the 'CPU clock go' trigger is always reset on the first clock signal after the 'CPU 5' latch is set.

If a store-data request is in progress, the 'CPU 5' latch is set before the CPU clock is restarted. Thus, as soon as the CPU clock is restarted, the 'CPU clock go' trigger is reset to indicate that no additional CPU clock cycles are required to complete the request.

If a 3-cycle fetch-data request is in progress, the 'CPU 4' latch is set before the CPU clock is restarted. When the CPU clock is restarted, the first clock signal sets the 'CPU 5' trigger/latch sequencers, and accesses the ROS word with the 'ingate SDBO' micro-order. Note, however, that the ingating of the SDBO into the CPU takes place on the following cycle. Thus, the CPU clock must be automatically stepped an additional cycle to perform the ingating. This function is performed by the 'CPU clock go' trigger, which is reset by the same clock signal that gates in the data into the CPU.

If a 4-cycle fetch-data request is in progress, the 'CPU 3' latch is set before the CPU clock is restarted. In this case, the 'CPU clock go' trigger is not reset until two cycles after clock-restart. Thus, the CPU clock is automatically stepped through two additional cycles to perform the required ingating.

As mentioned previously, sending the 'select' signal to storage is delayed when servicing requests in the single-cycle mode. This delay is accomplished as shown in the following illustration:



Note that the 'block CPU priority' signal is inactive when the CPU is not in the single-cycle mode. In this case, the 'CPU priority' trigger is set on the cycle following the setting of the appropriate IC, D, or scan-sync latch.

When servicing requests in the single-cycle mode, CPU priority is initially blocked by the reset state of the 'CPU clock go' trigger. When the BCU is ready to issue a 'select' signal, however, the 'CPU clock go' trigger is set and remains set until the request is completed. During the final stage of request servicing, the 'CPU priority' trigger is reset by the 'BCU cleanup' signal. The 'request finishing' trigger prevents immediate setting of the 'CPU priority' trigger if another sequential CPU request (in single-cycle mode) has been entered into the sync latch. After the current request has been serviced, the 'CPU clock go' and 'request finishing' triggers are both reset. The 'CPU clock go' trigger then blocks CPU priority until the pending request stops the CPU clock.

Channel Storage Requests

- Each channel storage request transfers 64 data (plus 8 parity) bits between channel and storage (via SDBI/SDBO).

- Data transfer is performed by asynchronous request/response sequence.

I/O channel data-transfer operations are initiated as a result of the CPU program selecting an I/O device and loading I/O control words into the associated channel. When data transfer is required, a series of requests is made by the channel to the BCU. After each request is accepted by the BCU, the proper storage unit is selected, and data is transferred either from storage to the channel (via the SDBO) or from the channel to storage (via the SDBI) on a 64-data-bit (plus 8 parity bits) basis. Thus, the function of the BCU is to accept channel requests, establish channel priority, decode the storage address supplied by the channel, select the proper storage unit, and execute the data transfer.

When priority is established for a particular channel, a 'BCU response' signal causes that channel to gate the storage address to the SAB. The storage address is then decoded by the BCU, and, if the addressed storage unit is available, a 'select' signal is generated and sent to the storage unit. If a store-data operation is required by the requesting channel, a 'store' signal is generated by that channel and is combined with the 'select' signal in the addressed storage unit to produce a store operation.

When the 'select' signal is sent from the BCU to the addressed storage unit, a 'channel accept' signal is sent to the requesting channel, verifying that the storage unit has been successfully selected. If a 'channel accept' signal does not follow the 'data request' signal, the requesting channel assumes that the selection attempt was unsuccessful. Detailed timing charts for channel requests are shown on ALD M8281.

This section describes the registers employed for CPU data flow and control functions. For the overall register data flow, see Diagram 3-2, FEMDM.

Q-REGISTER

The Q-register is a doubleword (64 data bits plus 8 parity bits) buffer for instructions entering the CPU from main storage on the SDBO (Figure 2-26). Data is transferred to the local storage address latches (LAL), the parallel adder, or the R-register.

Input

Instructions are transferred from the SDBO into Q by means of a gate signal decoded from the ROSDR latches. The transfer of data is initiated by either the I-Fetch hardware or by the ROSDR latches at not-clock time; the transfer controls remain active for one cycle (200 ns). The instructions are transferred into Q at clock time.

Op-Code Transfer

- Only halfwords containing op codes are transferred to R.
- Selection of halfword containing op code is determined by IC(21,22).

Only those halfwords in Q containing op codes are transferred to R. Because RX, RS, and SI instructions are composed of two halfwords and SS instructions are composed of three halfwords (only the first of which contains the op code), it is necessary to select the proper halfword to be transferred to R. Note that because RR instructions are composed of only *one* halfword, the next halfword to be transferred to R after an RR instruction is completed is the next sequential halfword in Q.

Selection of the halfword for transfer to R is determined by IC(21,22), and transfer is performed either directly through hardware or as a result of an I-Fetch micro-order. Recall that during an I-Fetch operation the op code of the next instruction to be executed is transferred from R to E, with R then being refilled with the next sequential op code. Because the op code of the next instruction to be executed is always in R, its format (positions 0 and 1) can be predecoded to determine the number of halfwords that compose that instruction and thus indicate which of the four Q-register halfwords contains the next sequential instruction op code. This predecoding occurs at end-op time of each instruction; the result (Q halfword number) is set into IC(21,22), which in turn selects a subsequent I-Fetch ROS word that specifies the next op-code halfword to be transferred to R.

Note: IC(21,22) is not used in addressing main storage, but only specifies which of the four Q-register halfwords is to be transferred to R by the following I-Fetch sequence. The IC(21,22) values associated with each Q-register halfword are illustrated in Figure 2-27.

An exception to the normal I-Fetch ROS word method of transferring the Q-resister halfwords to R is as follows. Assume a condition whereby a four-byte (RX, RS, or SI format) instruction occupies the right half of Q, IC(21,22) = 10, and a storage request is generated to main storage. When the I-Fetch sequence loads the op code of this four-byte instruction into R, the predecode logic determines that the next doubleword being accessed from main storage contains, in its leftmost byte, the op code of the next sequential instruction. [IC(21,22) is also stepped to 00 during this particular I-Fetch.] When, during the following I-Fetch sequence, the contents of R are transferred to E for execution, R is not refilled from Q in the normal manner because the particular I-Fetch ROS word selected to control this operation does not contain a micro-order specifying refilling of R. When that doubleword being brought from main storage enters Q, however, Q(0–15) (the op-code halfword) is allowed to proceed directly on into R. Thus, R again contains the op code of the next instruction to be executed, even though the instruction was not present during the I-Fetch sequence. This function is accomplished solely through the use of hardware that constantly tests for the presence of two signals: 'I-Fetch latch 1 and 3 set' and 'IC(21,22) = 00'.

B-Field and D-Field Transfer

The instruction B-field, which specifies LS registers, and the D-field, which is the main storage address displacement, is transferred from Q to LAL and the parallel adder, respectively. To save time, this information is transferred directly from Q instead of from R or E, thus allowing LS and the address to be available before the execution time of the associated instruction. Transferring these fields must be performed selectively so that the information is associated with the correct instruction.

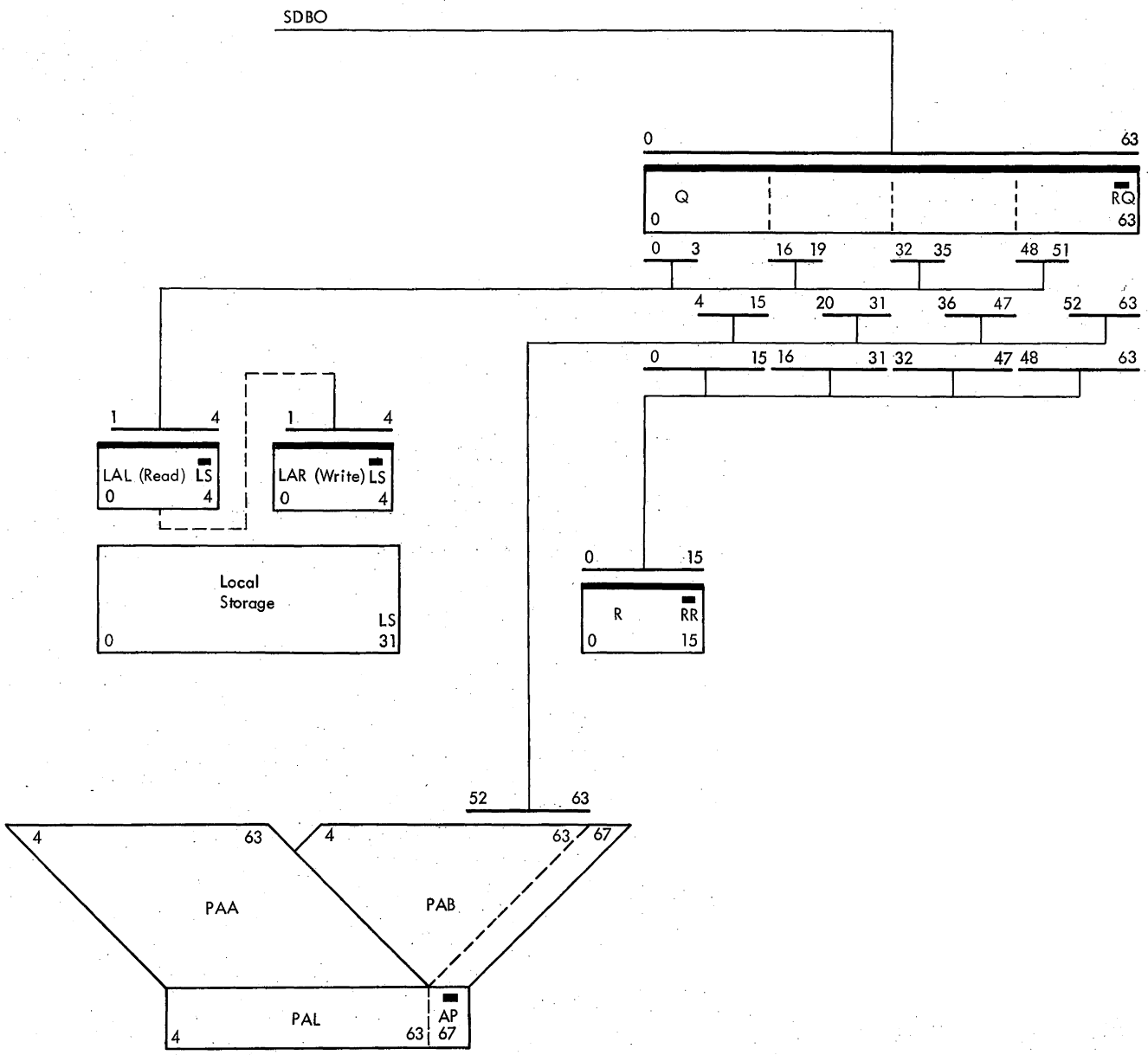


Figure 2-26. Q-Register Data Flow

B-Field Transfer

The four-bit Q-fields (B-field address data) are normally transferred to LAL at end-op time, under hardware control, per IC(21,22), or, for certain branch instructions, per D(21,22). (See Diagram 4-301, FEMDM.)

For SS instructions, however, two B-field values must be transferred to LAL. The first B-field is transferred to LAL at end-op time, per IC(21,22), under hardware control and in the normal manner. [D(21,22) is used for branch instructions.] The B-field of the second operand is then transferred to LAL (from the selected portion of Q) during I-Fetch of the SS instruction by a micro-order contained within one of the I-Fetch ROS words.

All transfer of data from Q to LAL takes place at not-clock time; the data is transferred into LS at clock time.

Note: Because an RR instruction can be contained within R and E, only halfword transfers from Q to R are required for RR instructions. All addresses for LAL can therefore be transferred directly from R or E.

D-Field Transfer

Selection of the Q-register D-field for transfer to the parallel adder (for use in address computation) is determined by the particular ROS word selected for use. The D-field transfer occurs at clock time.

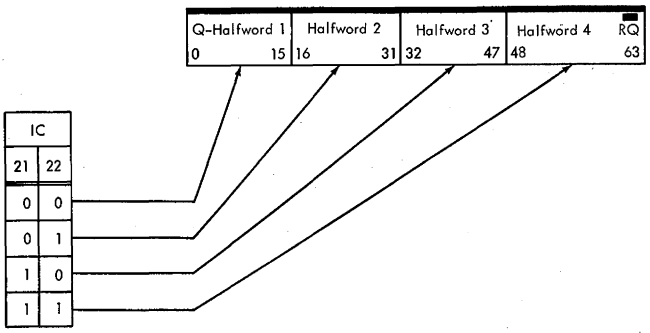


Figure 2-27. Q-Register Halfword Transfer Per IC(21, 22)

R-REGISTER

The R-register is a halfword (16 data bits plus 2 parity bits) register that provides intermediate buffering between Q and E for the halfword that contains the op code (Figure 2-28). This intermediate buffering speeds the refill of Q by allowing a storage request when the last op code has been transferred from Q but has not yet been executed.

Input

The R-register is loaded with one of the four halfwords in Q at I-Fetch time under ROS control. The contents of PAL(56–63) are also transferred to R (at I-Fetch time of the subject instruction of an Execute instruction).

Output

Whenever the instruction in R is predecoded as an RR non-branch instruction, R(8–11) is transferred to LAL at end-op time (Diagram 4-302, FEMDM). (The RR format indicates that R contains the entire instruction.)

Whenever the instruction in R is predecoded as an RR branch instruction, R(12–15) is transferred to LAL at end-op time. The contents of R are transferred to E at I-Fetch time under ROS control.

Predecoding

The R-register predecode logic samples R(0,1) at end-op time to determine the format of the next instruction. Time is saved because prefetching of operands per the format prepares data for use after the instruction is transferred to E. In addition, R(0,1) and IC(21,22) determine the need for storage requests to refill Q.

R(0–4) is tested for shift instructions. Because shifting does not require a storage request, time is saved if a shift instruction is decoded when a Q-refill request is generated 2 cycles before end op. The Q-register refill exceptional condition is eliminated because there is no interference between the shift instruction and the Q-refill storage request.

R(0–7) is sampled for branch instructions so that prefetching of the new instruction address can start immediately, thus saving time. R(12–15) is sampled for a zero condition which prevents the branch in the RR format.

On a branch end op, the instruction halfword is still in the process of being requested from storage. To save time in prefetching operands, the instruction format is predecoded from the SDBO rather than waiting until the instruction becomes stable in R.

E-REGISTER

The E-register is a halfword (16 data bits plus 2 parity bits) register which contains the op-code halfword of the instruction being executed (Figure 2-29).

Input

An op-code halfword (including two parity bits) is transferred from R to E during a normal I-fetch sequence under ROS control. On shift operations, D(18–21) is transferred to E(12–15) via the E-register incrementer. The data path from PAL(56–63) to E(8–15) is used in some SS format instructions to control the specified number of bytes.

Output

Op-code signals to control processing are decoded directly from E(0–7) without the use of gating logic.

LS is addressed by transferring E(8–11) or E(12–15) to LAL(1–4). E(8,11,12,15) is examined for an LS address specification error. When manual operations are performed using LS, E(11,12–15) is transferred to LAL(0,1–4) so that all registers may be addressed. Transferring E(8–11) to PAB(56–59), E(12–15) to PAB(60–63), and E(8–15) to PAB(56–63) provides for multiply and divide operand aligning, byte count control for SS format operation, and subsequent transfer to other registers. E(8–11), E(12–15), or E(8–15) may be sent to the E-register incrementer for alteration under ROS control.

E(8–15) is sent to an external device to provide external-control information during direct control operations. The data is transferred when the 'timing gate' trigger is set. E(8–15) is also sent to the PSW interruption code [S(24–31)] on a supervisor call interruption only, and to control triggers such as 'disable interleaving' and 'diagnose FLT'.

Incrementers

Two four-position incrementer registers are available, with ROS controls, for either treating E(8–11) and E(12–15) separately or treating E(8–15) as an entity. Positions E(8–11) and E(12–15) can be either incremented or

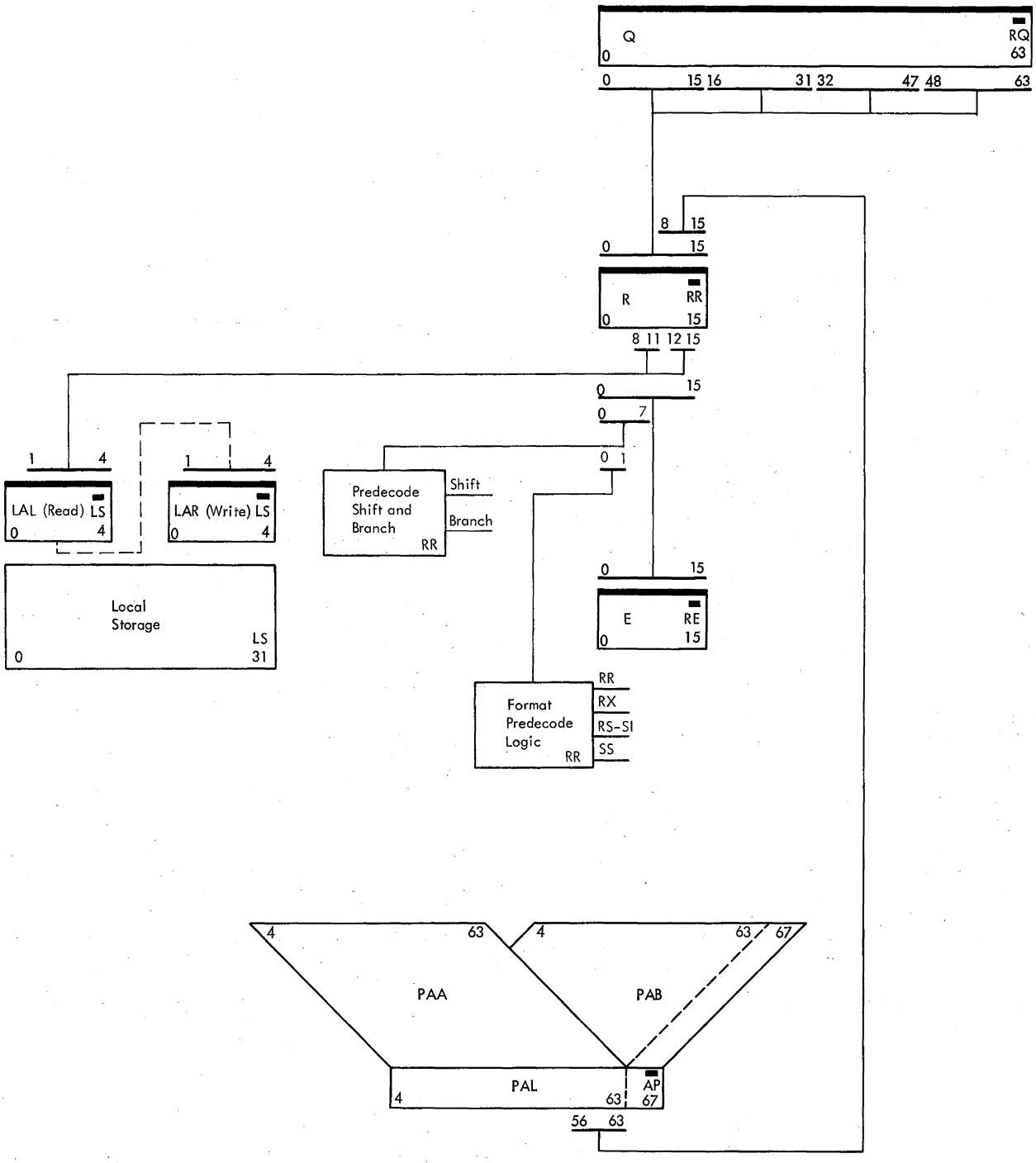


Figure 2-28. R-Register Data Flow

decremented by 1, but E(8–15) can only be *decremented* by 1 (for example, used for reducing length fields in logical VFL operations).

The E-register incrementers consist of latch circuits with logic decoder inputs (Diagram 4-303, FEMDM). The four-position incrementers are not capable of counting, but rather decode the binary information at their input, generate a binary value of 1 greater (or 1 less), and then set that value into the latches. Processing E(8–15) as an entity is accomplished by logically connecting the two four-position incrementers.

ROS controls also load constants into the E-register incrementers during execution of certain instructions (e.g., fixed-point multiply or divide) in order to select serial adder positions when developing products or quotients.

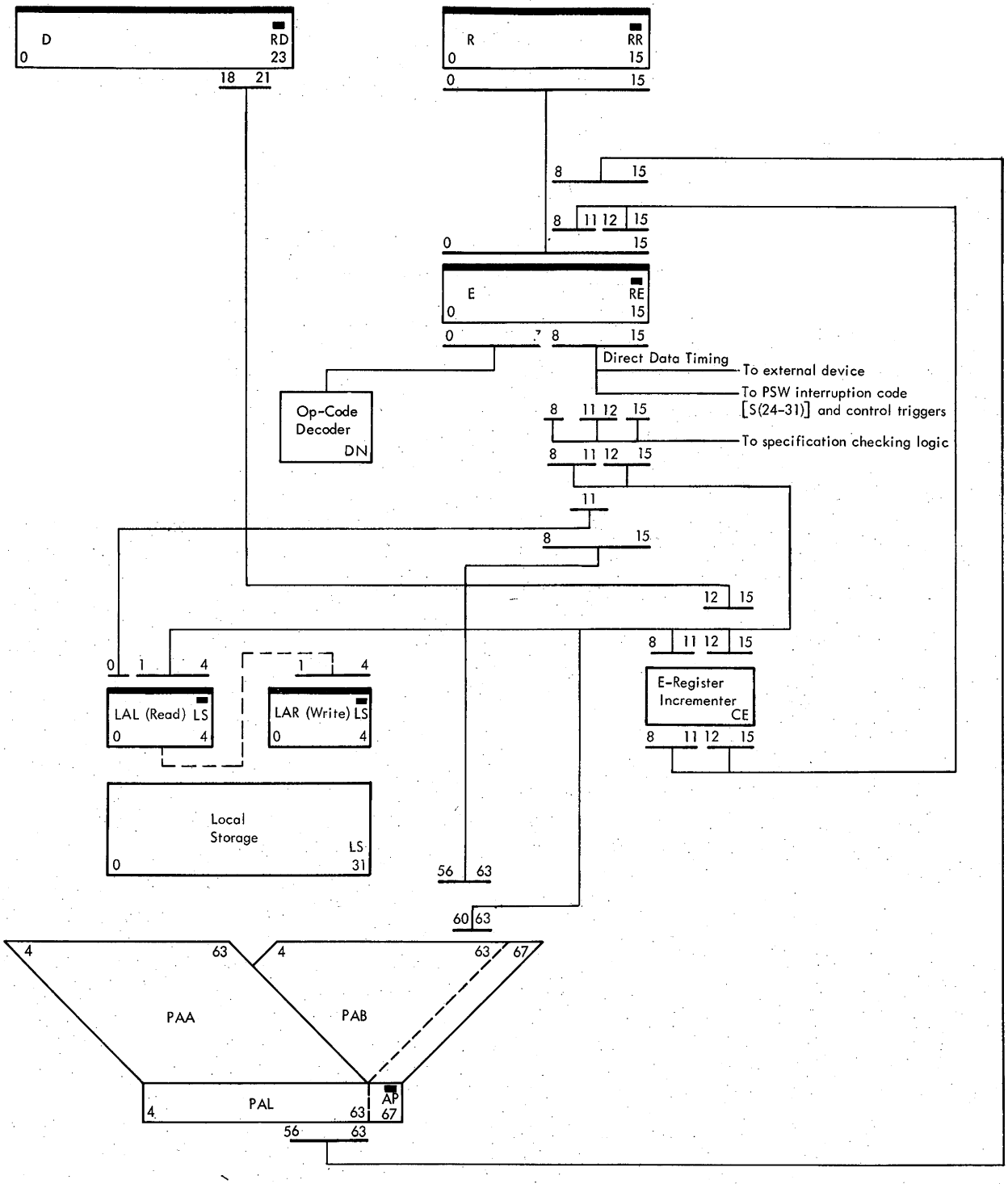


Figure 2-29. E-Register Data Flow

During shift instructions, D(18–21) is gated into the E(12–15) incrementer; decrementing functions reduce the specified shift amount as each shift operation is completed, and thus control the shift instruction.

When E(8–15) is modified in the E-register incrementer, E(8–15) parity may change. Diagram 4-304, FEMDM, shows the parity prediction logic to yield correct parity for E. If, for example, E(15) = 0 and the 'Add 1 to E(12–15)' signal is active, the 'change parity of E(12–15)' signal is developed. Assuming E(8–15) is odd, the 'INCR(8–15) bits even' latch is set which, in turn, sets the 'E(8–15) parity' trigger. Thus, parity is altered at the same time E is modified.

INSTRUCTION COUNTER

The instruction counter (IC) is a 24-bit (plus three parity bits) register used primarily in addressing doublewords of instructions from main storage (Figure 2-30).

Input

PAL(40–63) is transferred to IC(0–23) when incrementing the IC, or when entering a branch instruction as specified by D. Because IC(21,22) selects halfwords in Q, ROS controls the setting of IC(21,22) independently of the parallel adder.

Output

IC(0–23) is transferred to PAB(40–63) to be incremented by 8 so that the next sequential instruction address in main storage will be available in the IC. When called for, IC(0–20) is transferred to the storage address bus (SAB) to address the next instruction doubleword from main storage. IC(23) is transferred to the specifica-

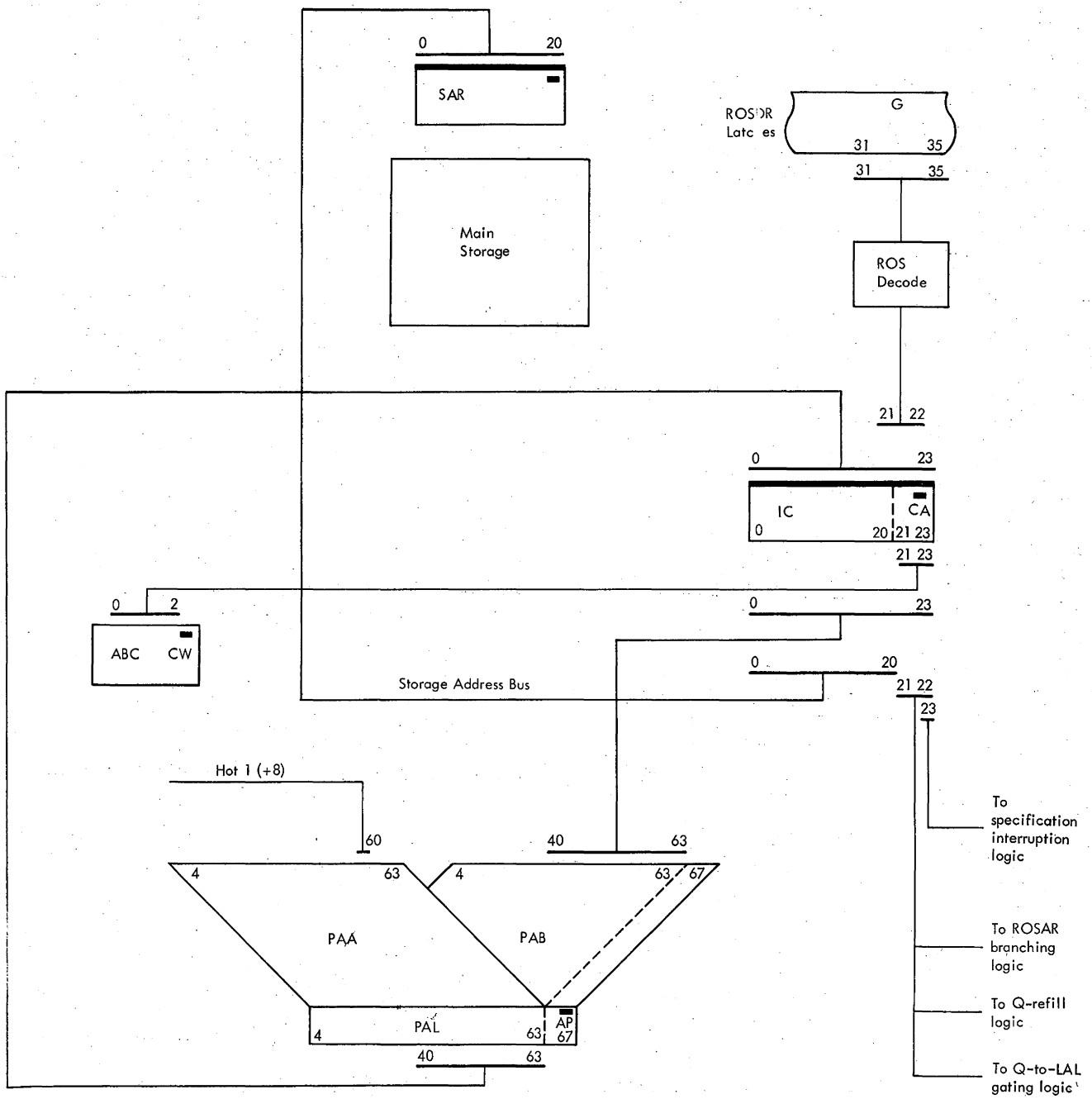


Figure 2-30. Instruction Counter Data Flow

tion logic to test for a 0-bit on instruction addressing; a 1-bit indicates a specification error.

In some instances, the address is for VFL data. Accordingly, IC(21–23) is transferred to ABC(0–2) on VFL operations to specify the desired data byte in AB. IC(21,22), through ROS branching, specifies the Q-halfword to be transferred to R; IC(23) determines the byte within that halfword.

Incrementing IC(0–20)

- After each storage request is generated, IC(0–20) is incremented by 8 to develop address of next sequential doubleword in main storage.

Incrementing (updating) IC(0–20) by 8 to develop the address of the next sequential doubleword in main storage is accomplished using the parallel adder. [IC(0–23) is gated to PAB, and a hot-1 bit is forced into PAA(60).]

At any given time, however, IC(0–20) may be either one or two doubleword addresses ahead of the doubleword in which the instruction being executed is located. These conditions occur as follows. When the instruction being executed is contained in a doubleword still present in Q, IC(0–20) has been updated and is one doubleword (8 byte addresses) ahead of the doubleword in Q. However, when the op-code halfword just transferred to R happens to occupy the last halfword portion of Q [IC(0–20) already being one doubleword address ahead], a storage request is generated to access the next doubleword and the IC is again updated by 8. [IC(0–20) is now two doubleword addresses ahead of the doubleword in which the instruction being executed is located.]

Incrementing IC(21–23)

- After each op-code halfword is transferred from Q to R, IC(21,22) is set to the value corresponding to the Q portion occupied by that halfword.

IC(21,22) values of 00, 01, 10, and 11 correspond respectively to the four (1–4) Q-register halfword portions. On the same cycle in which the op-code halfword is transferred from Q, IC(21,22) is set to the value corresponding to that halfword portion. [IC(21–23) trigger circuitry is not capable of accumulating, but only of receiving, input values.] Thus, these triggers are not stepped or incremented but, rather, set to values indicating the four Q-register halfword areas. IC(21,22) is controlled by ROS words, and, in the case of instruction sequencing, by the same ROS words that gate the Q-register op-code halfword to R.

In the event of a non-RR instruction, IC(21,22) must be changed by 2 or 3 to skip over the non-op-code halfwords remaining in the instruction. This skipping is accomplished as follows. Two factors involved in the 64-way ROS branch (NEXT-INST*IC) occurring at end-op time of each instruction are: (1) the format of the instruction op code previously transferred to R, indicating the number of halfwords composing that instruction; and (2) the contents of IC(21,22), indicating the Q-register area occupied by that instruction. These factors enable the end-op branch to access the proper I-Fetch ROS word for gating out the next sequential op-code halfword and also setting IC(21,22) to the value corresponding to that particular halfword area.

IC(23), set only during VFL operations in which an odd-numbered operand address is set into the IC, is not otherwise subject to change.

Note: IC(21–23) of the VFL operand addresses is placed into the AB counter, which then assumes the function of sequencing through the data-field bytes.

When IC(21,22) is set to new values, the parity of IC(16–23) may change. Parity adjust logic (Diagram 4-305, FEMDM) conditions the IC P(16–23) bit when the 'set IC(21,22)' micro-order is executed. IC(21,22) is set before the 'adjust parity' trigger is set, but circuit delays hold the parity adjust condition until the trigger is set. In effect, the parity adjust logic subtracts the parity of the old value of IC(21,22) and then adds the parity of the new value of IC(21,22), thus resulting in an updated IC P(16–23).

D-REGISTER

The D-register is a 24-bit (plus three parity bits) register which functions as a main storage address register for certain operations and as an I/O channel and unit address register for I/O instructions (Figure 2-31).

Input

Inputs to D are under ROS control. Address information (main storage or I/O) may come from either the parallel adder or the ADDRESS switches. Address information placed into D(17–20) is generated by the interruption logic.

Output

D(0–20) is transferred to the main storage address register (SAR) to provide storage addressing. On channel operations, D(8–15) is transferred to the channel address decoder to select a channel, and D(16–23) is transferred to the channel to provide the I/O unit address. D(18–21) is sent to the E-register, via the E-register incrementer. D(21–23) is sent to the ST byte counter, ROSAR branching logic, and specification checking logic. D(21,22) determines which halfword of Q to use to provide LS address information. The transfer of D(0–23)

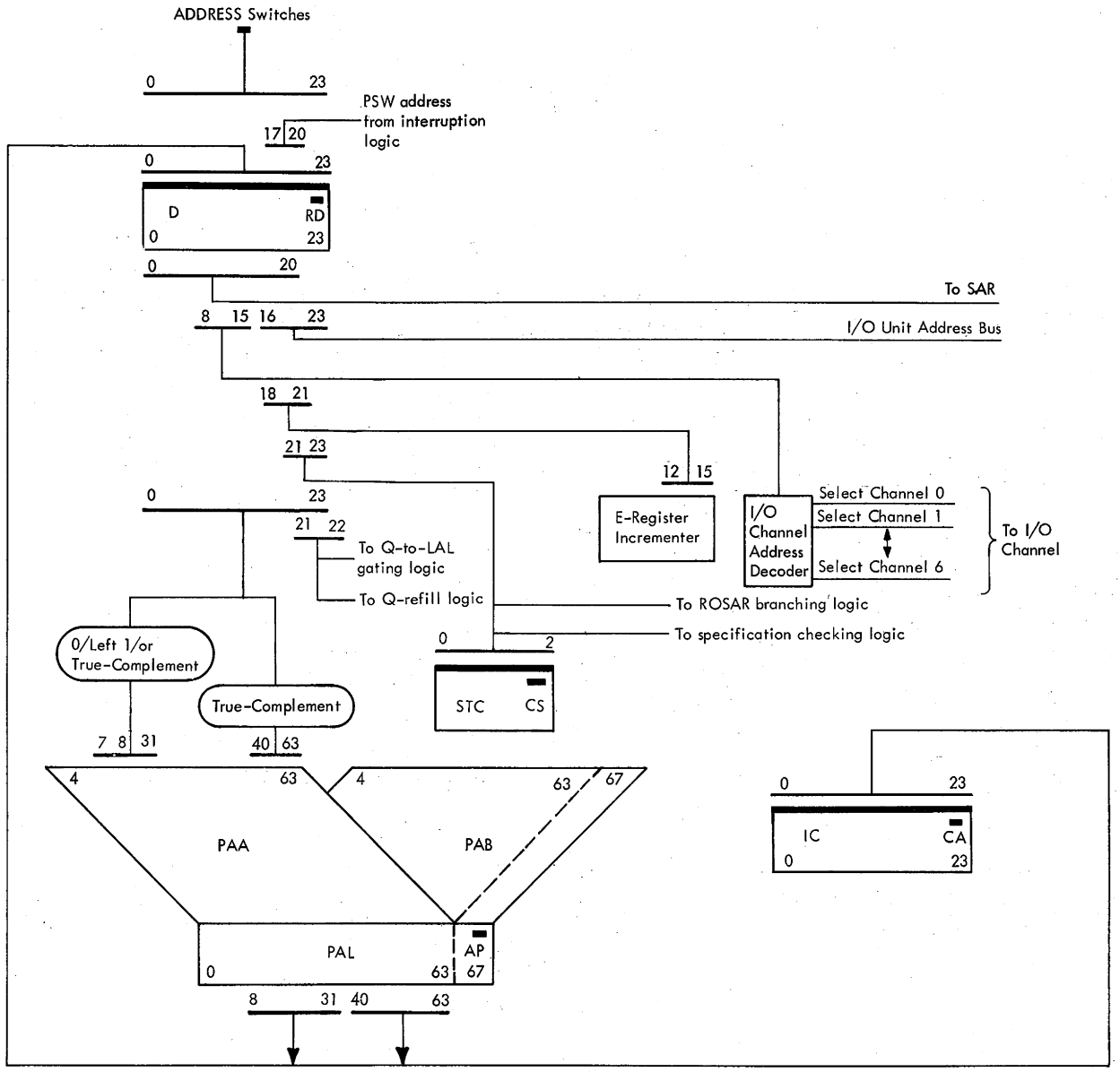


Figure 2-31. D-Register Data Flow

to PAA(8–31) or PAA(40–63) provides the path to alter or update the D-register.

Operational Functions

Operations in which D participates are: (1) branch and execute, (2) shift, (3) VFL, (4) fixed-point, (5) floating-point, (6) manual-control, (7) I/O, and (8) interruption.

Branch and Execute Operations

For branch and execute instructions, the specified successful branch address (all branches are assumed to be successful until otherwise determined) is placed into D by the normal I-Fetch sequence. A storage request is then issued to main storage (per D), and D(0–20) is gated onto SAB. D(21,22) specifies the particular op-code halfword within the doubleword in the same manner as IC(21,22) does for normal operation. If during execution of a branch instruction the branch is found to be successful (branch condition satisfied), the requested doubleword from main storage is gated into Q and the branch address in D is sent to the parallel adder and updated by 8. The result is placed into the IC (replacing the IC address), and the program proceeds in the normal manner.

If, however, the branch instruction is found to be unsuccessful, the doubleword requested from main storage (per D) is not gated into Q and the branch address

in D does not replace the IC address. The program then proceeds with the next sequential instruction.

Shift Operations

For RS instructions, I-Fetch adds the base and displacement values and places the result into D. Normally, this result is the second operand address. For shift instructions, however, this total specifies the number of bit positions to be shifted, and is used as follows. The number of shifts specified by D(22,23), a maximum of 3, are executed immediately upon being computed in the parallel adder and without the use of D. The number of shifts remaining is now specified by D(18–21), which indicates the number of shift operations necessary to complete the shift instruction, provided four shifts are accomplished by each shift operation.

Because left 4 and right 4 shifts are possible in the parallel adder, the binary number in D(18–21) is transferred to E(12–15), where the E-incrementer then controls the remaining number of left 4 or right 4 shift operations required to complete the instruction.

VFL Operations

For VFL operations, destination operand addresses are placed into D by the I-Fetch sequence. (Source operand addresses are placed into the IC.) Storage requests for destination operands are made per D(0–20), and the accessed doubleword is loaded into ST. D(21–23) is set into the ST byte counter to control ST byte transfer. The address in D is updated by 8 following each storage request, and, when the ST byte counter value reaches 7, another storage request is made per D to refill ST with destination operand data.

Fixed-Point Operations

For fixed-point operations, operand addresses are placed into D by the I-Fetch sequence. Operand storage requests are made per D(0–20), with D(21) determining which 32-bit word of the accessed doubleword is to be gated into ST. For halfword operation, D(22) determines which half of the 32-bit word specified by D(21) is to be gated into ST.

Floating-Point Operations

For floating-point operations using long operands, D and T provide for the handling of a 56-bit fraction. The high-order 24 bits of long fractions are contained in D.

Manual-Control Operations

In manual-control operations (manual mode), addresses entered into the ADDRESS switches are transferred to D. A storage request is then made per D to reference main storage for operations such as storing and displaying.

The address is entered into D as follows. Manual operation microprogram routines (for example, store or display) cause the parallel adder to generate all 1's, and then transfer them to D. Those ADDRESS switches not depressed (not set to 1) cause their associated D-register positions to be reset to 0; the resulting bit configuration in D is the address.

I/O Operations

The channel and unit address for an I/O instruction is placed into D(8–23) via the normal I-Fetch sequence, D(8–15) specifying the I/O channel and D(16–23) specifying an I/O device attached to that channel.

During the execution phase, D(8–15) is decoded in the CPU to determine the I/O channel. [Up to seven channels are currently available per system; D(8–12) must therefore always contain zeros, with the binary I/O channel address located in D(13–15).] Outputs from this channel decoder function to select one of seven possible I/O channels. D(16–23) is commonly routed to all available channels as a unit-address bus, and is decoded by the selected channel to select a particular I/O unit attached to that channel.

Interruption Operations

At end op of each instruction, ROS Y-branch (overriding branch) tests are made to check for the presence of any interruptions. Each interruption forces a unique bit configuration into D(17–20), which is generated by the interruption-decode and forced-address logic. (This logic also forces an address into ROSAR to access the first ROS word of the associated interruption-handling routine.) These four positions constitute the low-order bits of doubleword addresses in main storage that contain the new PSW for the various interruptions.

AB REGISTER

The AB register is a doubleword (64 data bits plus 8 parity bits) register that serves as a working register and as a buffer for doubleword operands received from main storage (Figure 2-32). Note that the AB register is logically divided into two 32-bit (plus four parity bits) registers, A and B, and has a four-bit extension, B(64–67), to retain low-order significance during certain shift and arithmetic operations.

Input

All AB positions are reset at clock P1 time of the cycle in which they are selected to receive information; data transfer then takes place at P2 time.

Main storage information (doubleword length) is transferred into AB under ROS control by transferring

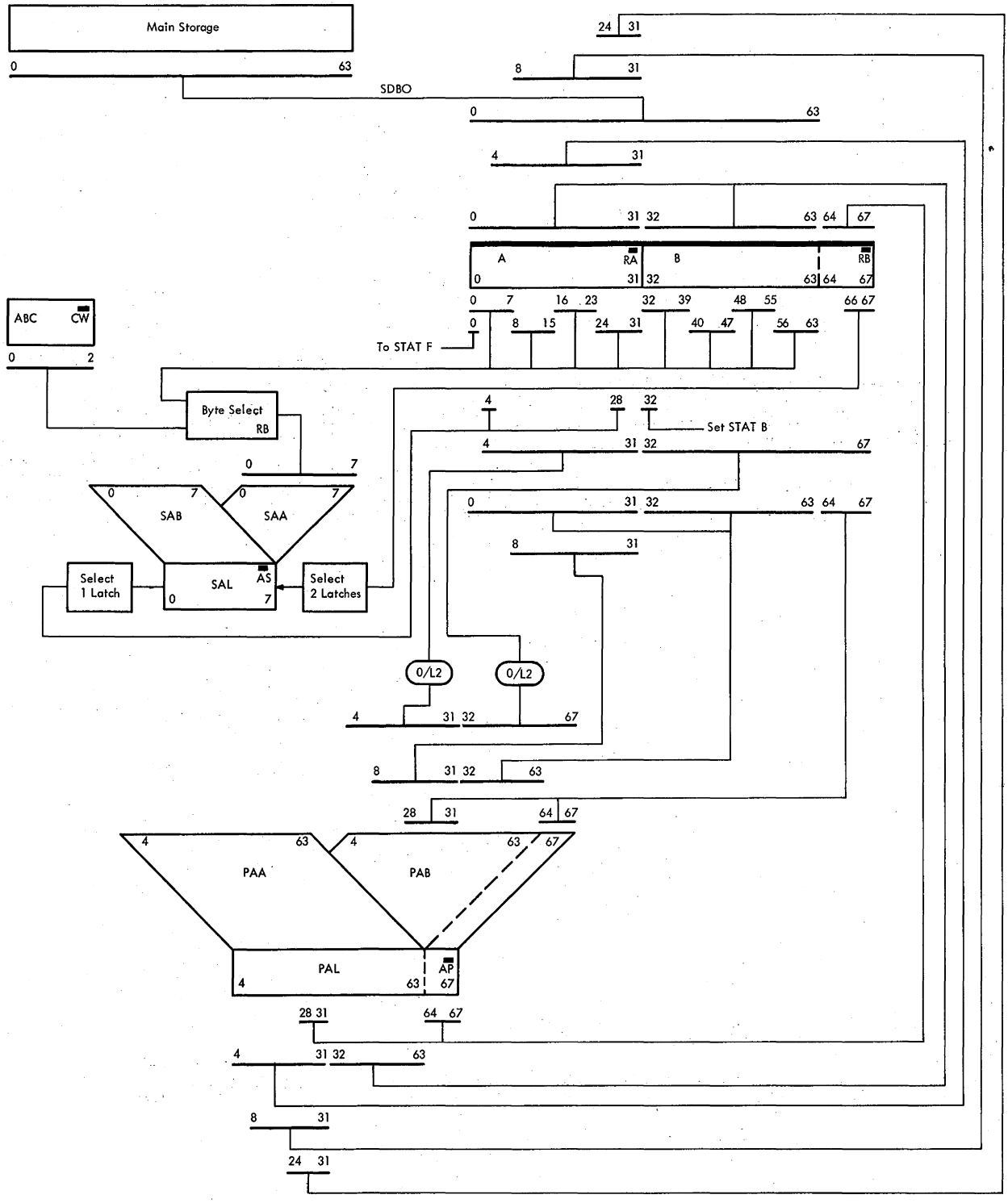


Figure 2-32. AB Register Data Flow

SDBO(0–31) and SDBO(32–63), plus parity, to A and B, respectively.

Parallel adder information (plus assigned parity) is also transferred to A and B under ROS control. Ingating of B(64–67) from PAL(64–67) or PAL(28–31) provides for maintaining high- and low-order significance during shift operations. Transferring PAL(24–31) to A(24–31) facilitates processing of fixed-point divide instructions.

Output

All AB transfer is under ROS control and is accomplished primarily through the use of gate-control triggers. All gate-control triggers are reset at P1 time of each machine cycle; the specific triggers selected for use are set at P2 time of the cycle in which they are to function. (One level of logic delay is incurred in transition; as a result, the respective transfer controls are activated at P3 time of that same cycle.) Selection of the gate-control triggers for use during any given cycle is determined either directly, through ROS decoding, or indirectly, through the AB byte counter. Parity information is transferred on a byte basis.

On multiply operations, the partial product, B(66,67), is placed directly into the serial adder latches (SAL) per E(14,15). Divide operations transfer A(4) or A(28) to one position of SAL per a ROS micro-order and E(14,15).

ST REGISTER

The ST register is a doubleword (64 data bits plus 8 parity bits) register that serves as a buffer between main storage, LS, and the CPU and also serves as a working register for arithmetic and logical operations (Figure 2-33). Note that the ST register is logically divided into two 32-bit (plus four parity bits) registers.

Input

- Inputs are from main storage, LS, parallel adder, serial adder, PSW register, interrupt logic, and DATA switches.

When new data is transferred into ST, only the bit positions involved are reset. Resetting occurs at P1 time and data transfer at P2 time. Reset signals are generated by the gating signals so that doublewords may be assembled (Diagram 4-306, FEMDM).

Main storage information is placed into ST by transferring SDBO(0–63) to ST(0–63), SDBO(0–31) to S(0–31) or T(32–63), or SDBO(32–63) to T(32–63). LS information (32 data bits plus 4 parity bits) is transferred to either S or T. All ST storage activity is controlled by ROS.

PAL(32–63) or PAL(40–63), plus parity, is transferred to T, and SAL(0–7), plus parity, is transferred to the ST byte per the ST byte counter and incrementer. ROS controls the transfer from the adders to the ST register.

PSW information is transferred from the PSW register to S(0–15) and T(32–39) under ROS control. The interruption code from the interrupt logic enters S(16–31) and is stored into the old PSW. Figure 2-34 shows S(16–31) input logic.

For manual control operations, information from the DATA switches is placed into ST for subsequent entry into main storage (or LS) in the following manner. All positions of ST are set to 1's by means of the parallel adder and LS ROS micro-orders. All DATA switches not set to 1's cause their respective ST positions to be reset to 0. Thus, ST reflects the information contained in the DATA switches.

Output

- Outputs are to main storage, LS, PSW register, serial adder, parallel adder, multiply/divide logic, and MCW.

All ST transfer is under ROS control. Transfers to the adders are performed by gate-control triggers. These triggers are reset at P1 clock time of every machine cycle; the specific triggers selected for use are set at P2 clock time of the cycle in which they are to function. (One level of logic delay is incurred in transition, and, as a result, each trigger activates its outgating circuitry at P3 clock time of that same cycle.)

Selection of gate-control triggers for use on any given cycle is controlled either directly by ROS or indirectly by the ST byte counter during ROS-controlled VFL operations. ST transfer to main storage takes place when a storage request is initiated with the 'store' trigger set; 64 data (plus eight parity) bits are then transferred to the SDBI. (Only T-register information can be gated to the LS.)

PSW information from S(0–15) and T(34–39) is transferred to the PSW register under control of ROS micro-orders.

The contents of ST, plus parity, can be transferred to PAA. In addition, T-to-PAA data transfer logic is capable of both true or complement and left 1 shift, and either T(32–47) or T(48–63) can be gated to PAA(48–63).

Byte transfer from ST (for product and quotient insertion during multiply and divide operations) is controlled by E(13–15). (The selected product/quotient bytes are transferred to the MPR bus.)

AB AND ST BYTE COUNTERS

For operations involving the serial adder, it must be possible to extract bytes from doublewords contained in AB and ST and to assemble bytes into ST for subsequent storage. These capabilities are provided by two byte counters: the ABC for controlling AB byte transfer, and the STC for controlling ST byte ingating and outgating.

AB Byte Counter

- Inputs are from PAL(61–63), T(57–59), E(13–15), and IC(21–23).
- ABC logic increments, decrements, or retains absolute values.

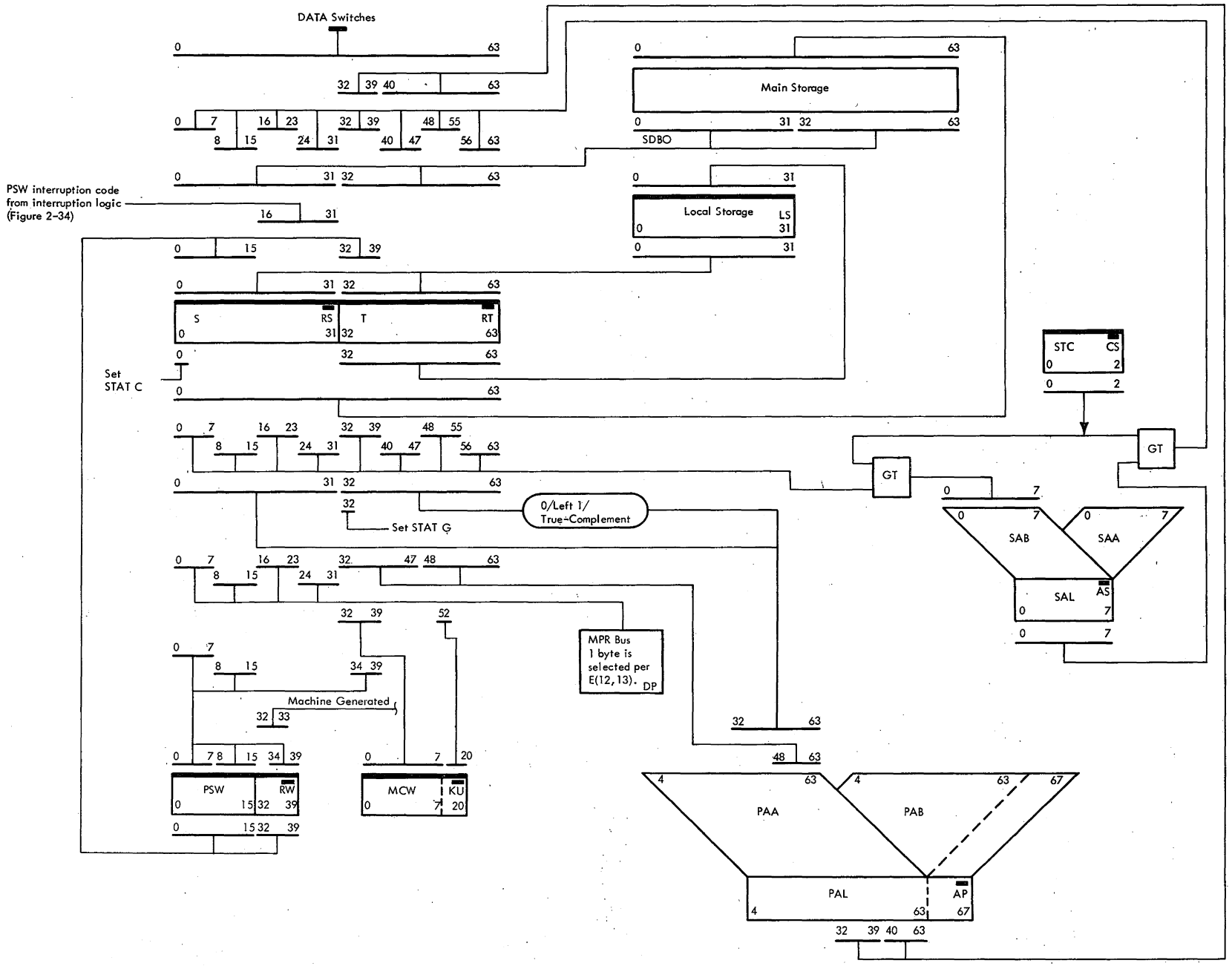


Figure 2-33. ST Register Data Flow

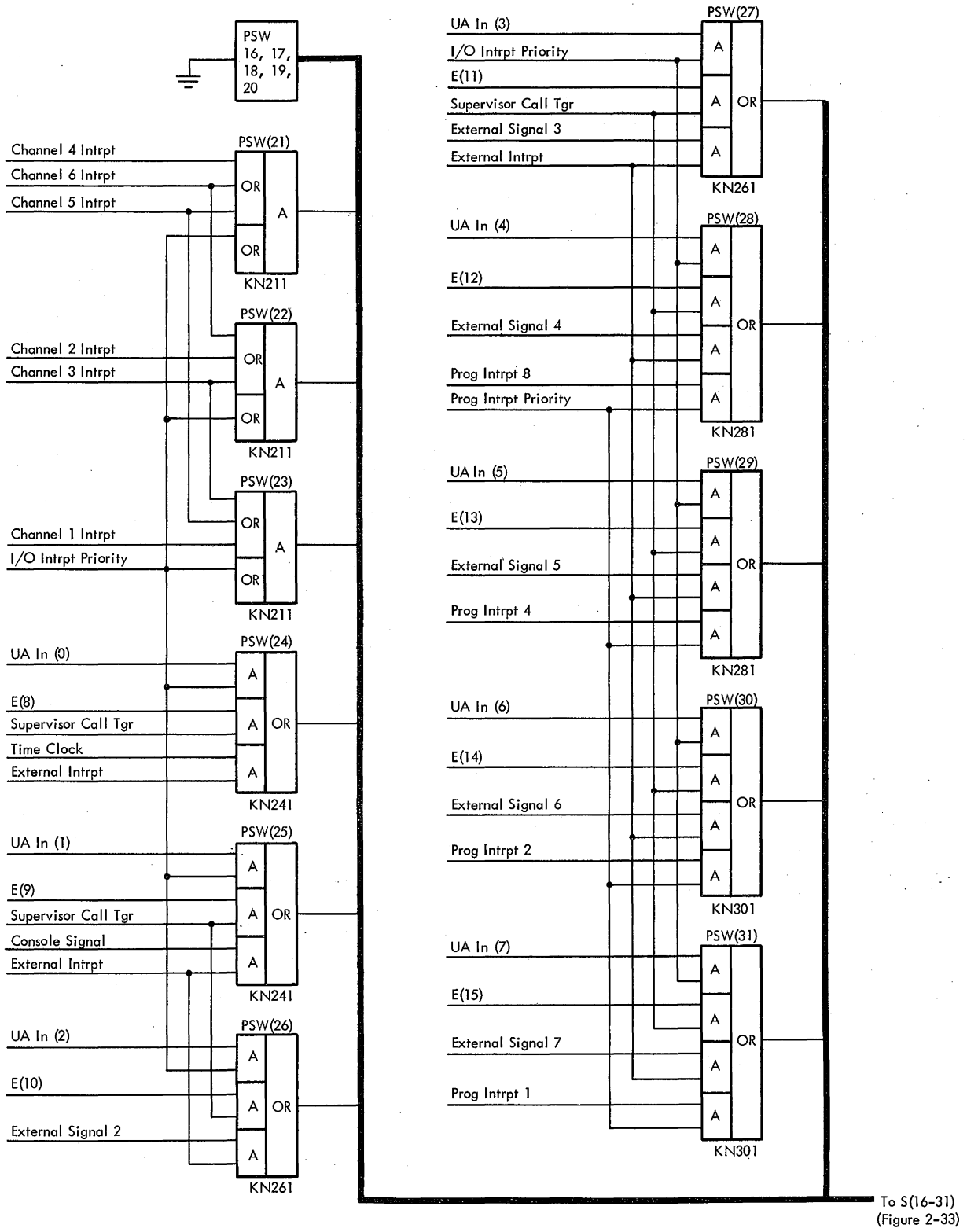


Figure 2-34. PSW Input to S(16-31)

The ABC (Diagram 4-307, FEMDM) consists of three triggers and three incrementer latches. These components are designated T0, T1, T2, and L0, L1, L2, in each group, and represent decimal values of 4, 2, and 1, respectively. Thus the ABC is capable of selecting any AB byte from 0 to 7. Both the trigger and latch groups are capable of receiving information (000–111 binary); modification (incrementing/decrementing) is performed through the use of incrementer-decoding logic on the input of the incrementer latches.

ROS controls the transfer of information into the ABC from PAL(61–63), E(13–15), and IC(21–23); data from T(57–59) is controlled by scan logic.

In operation, binary values of 000–111 (specifying AB bytes 0–7) are transferred into the ABC triggers at clock time under ROS control [ROSDR(25–30)]. The incrementer-decoding logic samples the ABC triggers and, under ROS control, sets that value, incremented by 1, decremented by 1, or absolute, into the incrementer latches at not-clock time. The incrementer latches are then sampled, and the outputs are decoded into eight lines (0–7) to select one of the eight AB bytes for transfer on the following machine cycle. In addition to controlling register transfer, ABC trigger outputs are utilized by scan operations and certain ROS-branch-decode functions.

Note: E(13–15) or IC(21–23) can also be entered directly into the incrementer latches, at not-clock time, under ROS control.

The ABC triggers are reset with a negative P1 clock pulse at clock time of each machine cycle and set with incoming data at P2 clock time. The incrementer latches are reset with a negative P2 not-clock pulse at not-clock time of each machine cycle, with data transfer occurring at P3 not-clock time.

The contents of the ABC triggers are transferred to the incrementer latches, and the same value is returned to the ABC triggers (regenerated), during each machine cycle in which no other ABC transfer controls are specified by ROS (provided that an 'I-Fetch reset' signal does not occur).

ROS control signals ('000 to ABC' and 'I-Fetch reset') reset the ABC and ABC incrementer to 0 by allowing both the triggers and the latches to reset on the following machine cycle.

ST Byte Counter

- Inputs are from PAL(61–63), T(54–56), E(13–15), and D(21–23).
- STC logic increments, decrements, or retains absolute value.

The STC (Diagram 4-308, FEMDM) consists of three triggers, three bipolar (polarity-hold) latches, and three incrementer latches. The triggers are designated as T0, T1, and T2 and the latches as L0, L1, and L2, representing decimal values of 4, 2, and 1, respectively. Thus the STC is capable of selecting any ST byte from 0 to 7. The STC triggers (with associated polarity-hold latches) and the STC incrementer latches are capable only of receiving information (000–111 binary); modification (incrementing/decrementing) is accomplished through the use of incrementer-decoding logic on the input of the incrementer latches. The polarity-hold latches retain each STC setting for one additional cycle, providing for a resultant data byte to be gated into ST at the same time the next sequential ST byte is being gated for processing.

ROS controls the transfer of information into the STC from E(13–15), D(21–23), and PAL(61–63). Entry of T(54–56) to the STC is controlled by scan logic.

In operation, binary values of 000–111 (specifying ST bytes 0–7) are transferred into the STC triggers at clock time, with the associated polarity-hold latches assuming the same value at not-clock time of that same cycle. The incrementer-decoding logic samples the contents of the STC triggers and sets that value (incremented, decremented, or absolute) into the incrementer latches. The incrementer latches are then sampled, and the outputs are decoded into eight lines (0–7) to select the ST bytes for transfer during a subsequent machine cycle.

Note: E(13–15), D(21–23), or a defined constant can also be entered into the incrementer latches under ROS control.

The polarity-hold latches are set (or reset) at not-clock time to the value of the STC triggers. This value (specifying an ST byte) is retained in the polarity-hold latches until not-clock time of the following cycle. Thus, at clock time of the following cycle, with the STC triggers having been set to a new value and the incrementer latches possibly containing a modification of this new value, the previous STC-trigger setting is still present in the polarity-hold latches. This retained value now allows information from the serial adder to be placed into the ST byte that was previously transferred out, at the same time that the incrementer latch output is transferring the next sequential ST byte to the serial adder for processing.

All incrementer latch decode lines are sent to the mark-trigger logic (specifying byte areas for main storage entry), and decode lines 0, 3, and 7 are sent to the branch logic controlling the ROSAR setting. An 'STC greater than 3' signal is also transferred to the branch logic controlling ROSAR whenever incrementer latch 0 (binary value of 4) is set.

Gating of the contents of the STC triggers into the incrementer latches and regeneration of that latch value to the triggers are performed each machine cycle in which no other STC ingating controls are specified by ROS (provided that an 'I-Fetch reset' signal does not occur).

The '000 to STC' and 'I-Fetch reset' signals reset all STC triggers and latches on the following machine cycle. The '1 to STC bit 0' and 'I-Fetch reset' (for RR instructions) signals cause the incrementer to assume a decimal value of 4; the '011 to STC' signal sets incrementer latches 1 and 2, thus setting the incrementer to a decimal value of 3.

MARK TRIGGERS

Eight mark triggers, contained in the CPU, indicate which bytes of the doubleword on SDBI are to be entered into main storage on a store operation.

Mark trigger logic (Diagram 4-309, FEMDM) is ROS controlled; operation is as follows. ROS control field L (ROS sense latch positions 43–46) is decoded to activate one or more of four mark trigger signal lines. These lines set the mark triggers as required: (1) individually, per the STC, (2) in groups of four (0–3 or 4–7), and (3) unconditionally, by setting *both* the 0–3 and 4–7 groups. ROS micro-orders to set mark triggers also set the 'store' latch to generate a 'store' signal which is sent to the selected storage unit.

F-REGISTER

The F-register is a one-byte (plus parity) trigger register that is used in certain arithmetic, logical and data-transfer operations (Figure 2-35).

Input

Inputs to F are under ROS control. All positions involved in an operation are reset at P1 clock time of the same cycle in which they are to receive information, with the ingating occurring at P2 clock time. Data and external control information is received during direct-control read operations, and serial adder outputs are received during VFL operations. F(0–3), F(0–4), and F(4–7) are utilized by Set Key and Insert Key instructions, logical instructions, and decimal multiply and divide instructions.

Output

All outputs are under ROS control. F(0–7) is transferred to the serial adder by means of a gate-control trigger at clock time, and F(4–7) is transferred to the parallel adder under control of the parallel adder input logic. F(0–3) is transferred to the storage protect area during set-key operations. F(4–7) is also used in ROS branching.

G-REGISTER

The G-register is a one-byte register that buffers one byte of data between the CPU and an external device (Figure 2-36). SAL(0–7) is the only input, and the external device is the only transfer path.

PSW REGISTER

- Inputs are from ST and interruption-control logic.
- Outputs are sent to ST and CPU control circuitry.

Although the PSW is 64 bits in length, the PSW register contains only 24 bits (Figure 2-37). The remaining information (generated by the CPU at the time of an interruption) is used to identify the cause of the interruption and to allow the CPU to return to the correct program address.

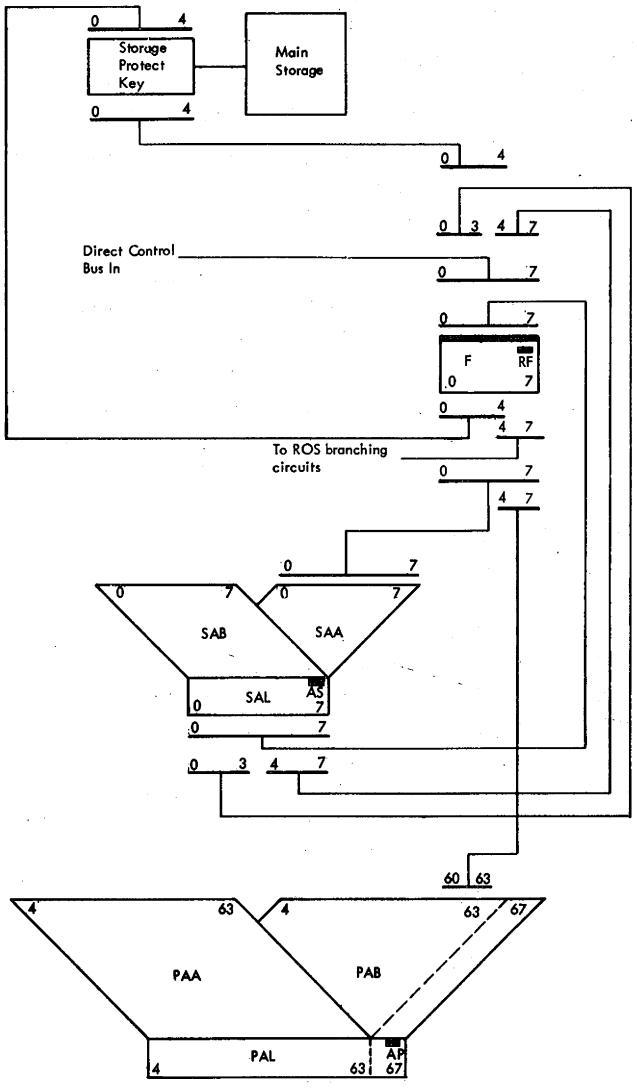


Figure 2-35. F-Register Data Flow

PSW register trigger logic is shown in Figure 2-38. The 'gate S(0–7) to PSW(0–7)' signal resets the triggers and, through the logic delay, provides the gating signal to allow the ST register information to enter the PSW register. Thus the information remains in the PSW register until it is replaced by a new PSW.

All PSW register input and transfer is initiated by ROS micro-orders. When an interruption occurs, a series of micro-orders in the accessed ROS word transfer the contents of the PSW register into ST for subsequent entry into main storage.

The format of the instruction in E (interrupted instruction) is decoded, thus providing the instruction-length code to be entered into the PSW register before transferring the contents of the PSW register to ST. Micro-orders also transfer the old PSW address (generated by the interruption control logic) for that particular interruption to D(17–20) to develop the old PSW address for that interruption. Either 8 or 16 (depending on the current instruction address) is subtracted from the IC and inserted into the instruction-address field of the assembled PSW.

Micro-orders executed by the Load PSW, Set System Mask, or Set Program Mask instructions control the transfer of PSW information from the SDBO to ST and the transfer of PSW data from ST to the PSW register and the IC. The old PSW address (contents of D + 64, decimal) is generated in the parallel adder, also under ROS control.

The PSW register does not contain data transfer logic; PSW information is constantly available throughout the CPU for use as required.

MCW REGISTER

The MCW register is a nine-bit trigger register that provides program control of scan operations (Figure 2-39). During execution of the Diagnose instruction, FLT's, or ROS tests, MCW(0–7,20) is gated from T(32–39,52) to the MCW register. The bits of the MCW are retained in the MCW register and decoded by the MCW decoder to perform the functions as specified in Chapter 6.

Note: Four MCW's use the same MCW register: (1) FLT, (2) ROS test, (3) Diagnose for CPU, and (4) Diagnose for channels. See Chapter 6, Section 2, for the format of each MCW.

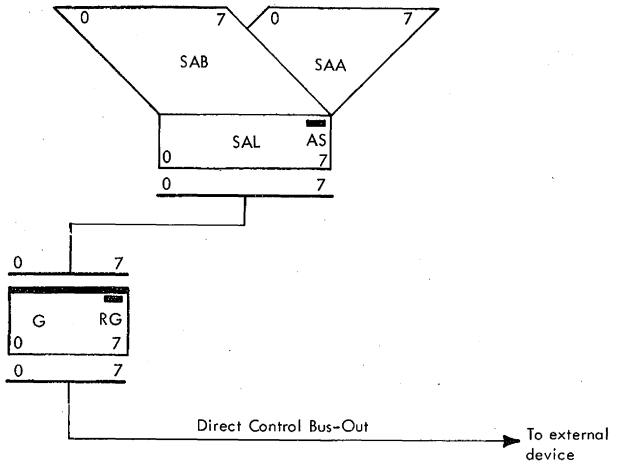


Figure 2-36. G-Register Data Flow

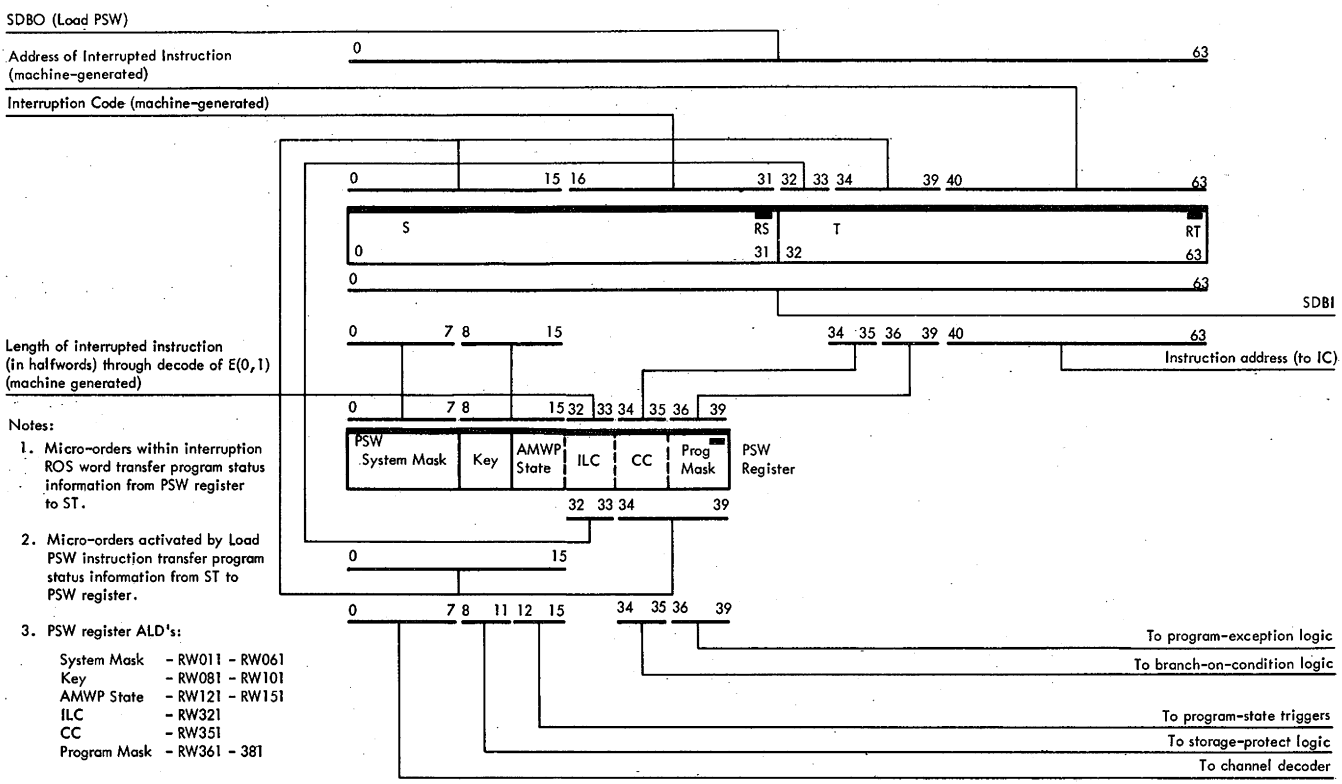


Figure 2-37. PSW Register Data Flow

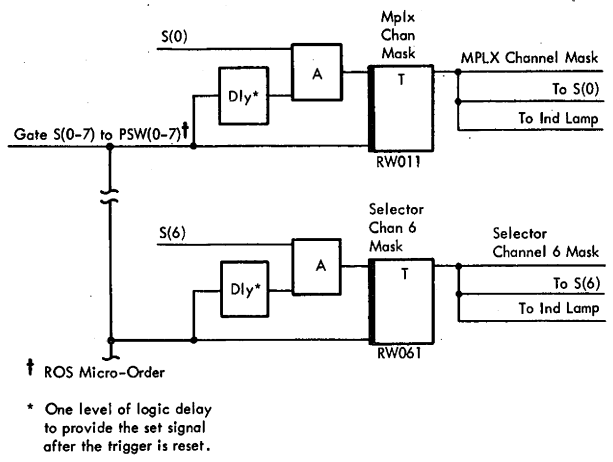


Figure 2-38. PSW Register(0,6) Logic

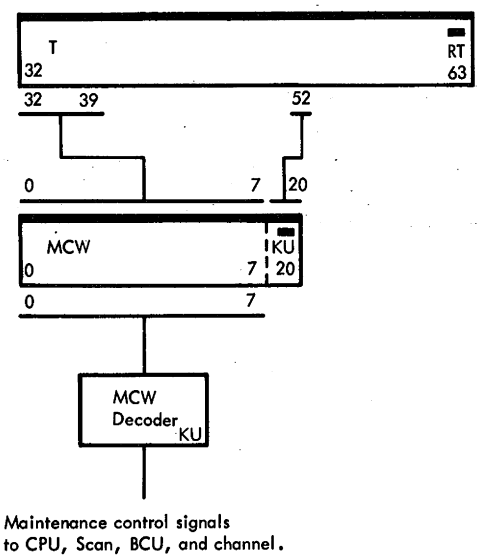


Figure 2-39. MCW Register Data Flow

This section describes the operation of the 25-register local storage (LS).

ADDRESSING AND DATA FLOW

- 5-position address registers [LAL (Read) and LAR (Write)] address LS.
- Input to LS is from T only; output is sent to S and/or T under ROS control.

Two five-position LS address registers [LAL (Read) and LAR (Write)] select the 25 individual LS registers (Fig. 2-40). The LS address is received from Q, R, or E under ROS control (or directly from ROS control words when addressing the LSWR). The particular register (Q, R, or E) and field within that register to be set into LAL is determined by decoding ROSDR(38–42). The four-bit LS addresses are gated into the four low-order positions of LAL from Q, R, or E fields. These four-bit addresses are capable of directly addressing registers 0–15 (general-purpose registers). For floating-point operations (requiring the use of registers 16–23), a 1-bit is forced into the high-order position of LAL upon decoding of the floating-point op code. This action increments the four-bit address from Q, R, or E by 16, thus forcing the use of floating-point registers 16–23.

Note: Floating-point instructions are restricted to the use of even LS addresses 0, 2, 4, and 6. Automatic incrementing of these values by 16 then generates LS addresses of 16, 18, 20, and 22.

Long-operand floating-point op codes also force a 1-bit into the low-order position of LAL, in addition to the high-order 1-bit forced by all floating-point op codes. This additional bit further increments the 16, 18, 20, and 22 floating-point addresses by 1, thus generating the second (R1 + 1) register address required for long-operand (64-bit) instructions.

For operations requiring use of the LSWR (register 24), the ROS words controlling these operations force 1-bits into the two high-order positions of LAL [LAL(0,1)]. This action generates a binary address of 24, and is the only means of selecting the LSWR.

Selection of the Q, R, or E field to be entered into LAL is determined by decoding ROSDR(38–42) of the controlling ROS word or by selecting 'NEOP' or 'BEOP' micro-orders, depending on the next programmed instruction. Regardless of the address source or of whether a read LS or write LS operation is indicated, the contents of the addressed register are always read out onto the LS data bus. If a read LS operation is indicated, decoding of ROSDR(10,11) of the controlling ROS word gates the contents of the LS data bus into S, T, or both S and T. When a write LS operation is indicated, the contents of the addressed LS registers are gated out onto the LS data bus in the same manner, but the output resulting from decoding ROSDR(10,11) remains inactive and does not condition the ST ingating controls. A 'write into LS' signal, resulting from the ROSDR(38–42) decoder, then gates the ST bus (T-data) into the addressed LS register.

DATA TRANSFER CONTROLS

The following paragraphs describe the LS logic involved in read LS and write LS operations. Diagram 4-401, FEMDM, illustrates the read/write logic of LS register 0 and also the common control circuit timings for each 200-ns LS cycle. (Registers 1–24 are identical with register 0.)

Local storage addressing and all LS register-operating logic are implemented in 10-ns circuitry; the polarity-hold latches and associated input logic are implemented in 30-ns circuitry.

Read LS Operation

- ROSDR(38–42) sets LAL from specified Q, R, or E field.
- LAL gates contents of selected LS register to LS data bus.
- ROSDR(10,11) gates LS data bus into ST.

Read LS operations are initiated at not-clock time when LAL (Read) is set with Q, R, or E information, as determined by ROSDR(38–42). The LSWR address and the floating-point register address bits are also entered into LAL at this time, depending on ROSDR(38–42). The contents of LAL (Read) are decoded, and the decoder outputs gate the contents of the selected register polarity-hold latches to the LS data bus. ROSDR(10,11) then activates the required ST input logic, and at clock time of the following cycle the LS data bus information is set into the ST triggers. Polarity-hold circuits provide nondestructive readout, eliminating the need for regeneration. (Refer to the timing chart in Diagram 4-401, for relative control timings.)

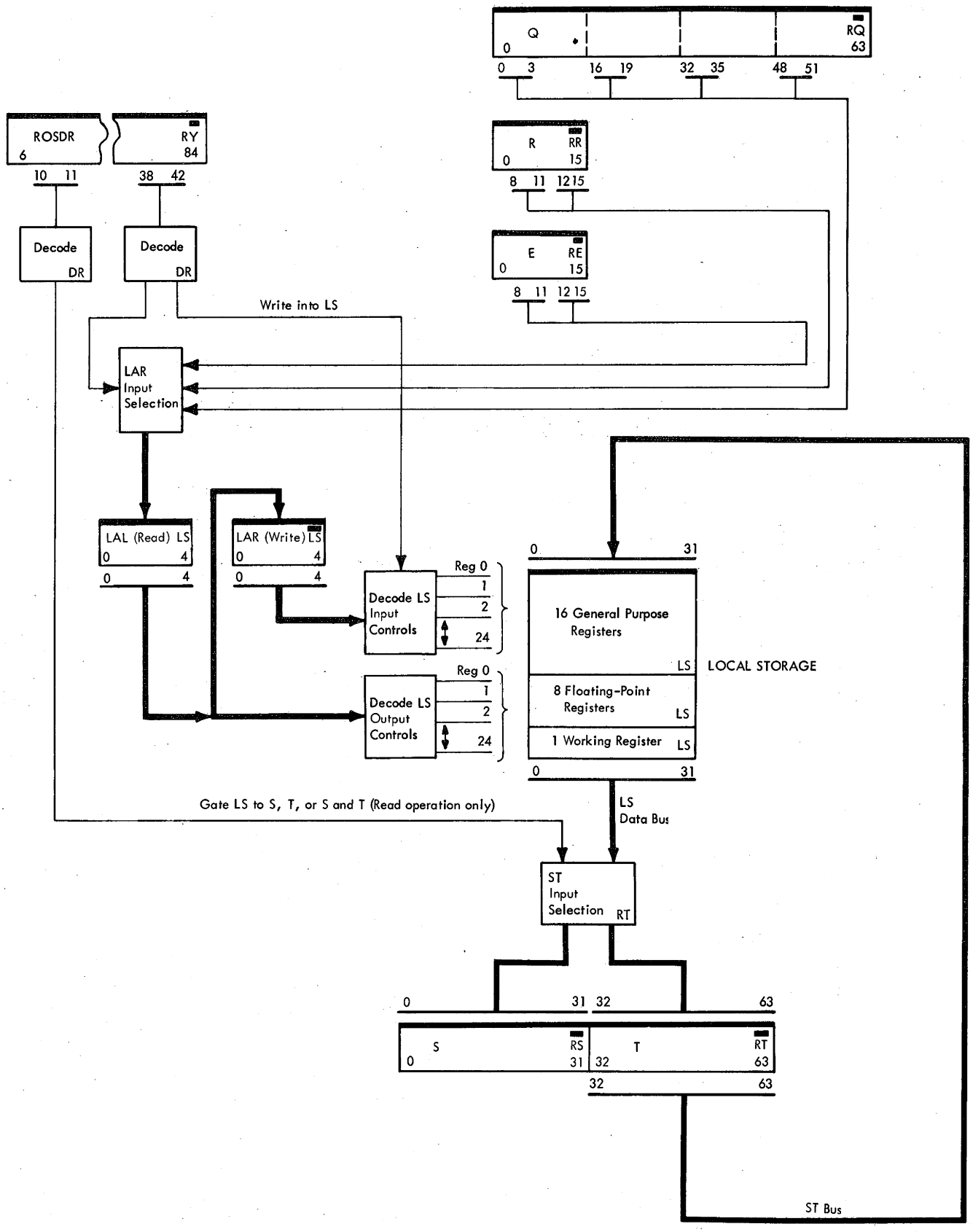


Figure 2-40. Local Storage Data Flow

Note: LAL (Read) is sampled every cycle, even if no command is given. Data from LS is thus placed on the LS data bus every cycle, but is not always gated into a register. The contents of LAL (Read) are transferred to LAR (Write) every cycle, but no addressing is performed unless a write LS micro-order is decoded in ROS.

Although LS data is available to the CPU approximately 100 ns after the setting of LAR, consecutive LS data readout is limited to 200 ns. LS cycles are therefore defined as being 200 ns long.

Write LS Operation

- ROSDR(38–42) sets LAL (Read) to specified Q, R, or E field.
- LAL (Read) contents are transferred to LAR (Write).
- LAR (Write) decoder selects specified LS register.
- ROSDR(38–42) gates ST bus data into selected LS register.

On write-LS operations, LAL (Read) latches are set at not-clock time with the specified Q, R, or E information. At the beginning of the following cycle, LAL (Read) is transferred into LAR (Write) in the same manner as for a read operation. The selected LS register is also gated to the LS data bus as in read operations; up to this point, read and write operations are identical. (On write LS operations, however, LS data bus information is not gated into the ST register.)

At the beginning of the following cycle, LAL (Read) is transferred into LAR (Write). Further decoding of ROSDR(38–42) generates a 'write into LS' signal that sets the 'write LS' trigger at P0 time. This trigger provides the signal to gate the ST bus (T-data) into the selected LS register at not-clock time of the following cycle. (Refer to the timing chart in Diagram 4-401 for relative write control timings.) Negative levels on the ST bus represent 1 bits and set the respective polarity-hold latches; positive levels represent 0's and reset the respective polarity-hold latches.

For an 'insert sign' micro-order when the result sign is minus, T(32) is forced to a 1. To preserve proper parity for this operation, the parity bit for T(32–39) is inverted before it is transferred to LS.

There are situations when writing into LS must be inhibited. When such a situation occurs, the 'SPEC' (K31) micro-order causes a set signal to the 'inhibit LS write' trigger so that the LS positions remain unchanged.

Note that a minimum of 40-ns coincidence must exist between stable ST bus data (1's or 0's) and an active 'gate T to LS' polarity-hold-latch control signal to give correct data entry. When the 'gate T to LS' signal is deactivated, polarity-hold latches of the selected register remain in their present state until new data is entered on a subsequent write LS operation.

Note, too, that LS data is not parity-checked until it enters an adder at a later time.

LS Timing

Separate address registers for reading and writing coordinate the LS timing to other CPU functions. The 'gate LS reg n' signal is generated by LAL (latch timing) to make the LS data available at clock time for entry into a register. LAR, at not-clock time, allows the data entered into T early in the cycle to be entered into LS late in the cycle. Refer to Diagram 4-401, which illustrates the entry of new data into T (reflected by the shift in the ST bus data line) and the subsequent storage of that data into LS.

Section 6. Serial and Parallel Adders

This section describes the operation and application of the serial and parallel adders.

SERIAL ADDER

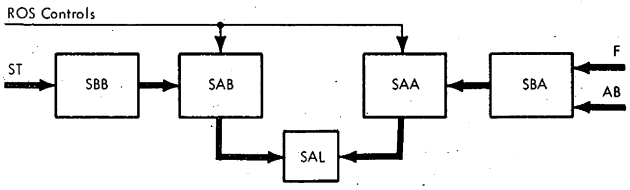
The serial adder (8 data bits plus 1 parity bit) processes data in binary or decimal format, performs logical AND, OR, and Exclusive-OR functions, and assembles multiply/divide results.

Data flow for the serial adder is illustrated in Figure 2-41. Note that data entered into the A-side of the adder (via final bus-A) comes from either AB or F under ROS control.

Data entering the serial adder is in true or complement form. For a true add operation, the data is entered directly; for a complement add operation, the input data to the serial adder A-side (SAA) is inverted (Fig. 2-42). Note that the input data may be in true or complement form. The two's complement value is achieved by forcing a hot-1 to the input logic for serial adder latch (SAL) position 7.

Input and Output

Inputs to the serial adder A-side (SAA) are the contents of F or a selected byte from AB (per the ABC); the input to the serial adder B-side (SAB) is a selected byte from ST (per the STC). A bus arrangement transfers data from the registers to the serial adder as follows:



During transfer from the serial adder bus A (SBA) to the SAA or from the serial adder bus B (SBB) to the SAB, ROS controls can alter the data being transferred. The SAA input can be altered by the following functions: decimal excess-6, complement add, shift, crossgating (interchanging of incoming bits 0–3 with bits 4–7), and zone and sign insertion. The SAB input may be altered by the following functions: sign insertion, special digit insertion, and special gating for changing destination (for example, placing bits 0–3 into bit positions 0–3 and 4–7).

After the sum has been developed and placed into SAL, gating signals from ROS allow the information to be transferred to F, to G, or to a selected byte in ST.

Adder Operation

A simplified summary of serial adder operation is shown in Figure 2-43. SAA and SAB are combined in the serial adder to produce a bit-carry, a bit-transmit, or a half-sum. A bit-carry is developed when both input bits are present, a bit-transmit when either input is present, and a half-sum when only one of the input bits is present (Figure 2-44). Carry-in and half-sum conditions combine to produce a full sum. The table in Figure 2-44 shows the conditions which produce a full-sum bit. For example, if SAA is a 1-bit and SAB is a 0-bit there will be no bit-carry, but a bit-transmit and a half-sum will be produced. If no 'carry in' signal is present, the full-sum is a 1.

The 'carry in' signal is developed by the carry lookahead logic. A test is made for a carry from the next lower position or for a carry developed from bit-transmits and a lower-order carry (Figure 2-45). The carry look-ahead logic saves time by providing an immediate carry rather than using another cycle to ripple a low-order carry through the adder.

Accurate results are achieved by parity checking and parity correction circuits. Tests for error conditions are made at half-sum and full-sum levels as well as on decimal input data.

Controls

- Selected data enters on SBA and SBB.
- Data is first modified on transfer from SBA to SAA and from SBB to SAB.
- Final modification occurs as data enters SAL.

There are three control areas: input bus, final bus, and SAL (Figure 2-46).

ROS sense latch 86, field R, selects F or AB as a data source for SBA. If the latch is set, F is selected; if it is reset, AB is selected. Gate control triggers provide the gating signals to transfer data to the buses. For SBA, if F is not selected, an AB gate control trigger is selected by the value in the ABC (Diagram 4-501, FEMDM). Similarly, the STC selects an ST gate control trigger. One byte of data is selected by each gate control trigger, and two gate control triggers are selected (one for SAA and one for SAB) each machine cycle whether or not the data will be used.

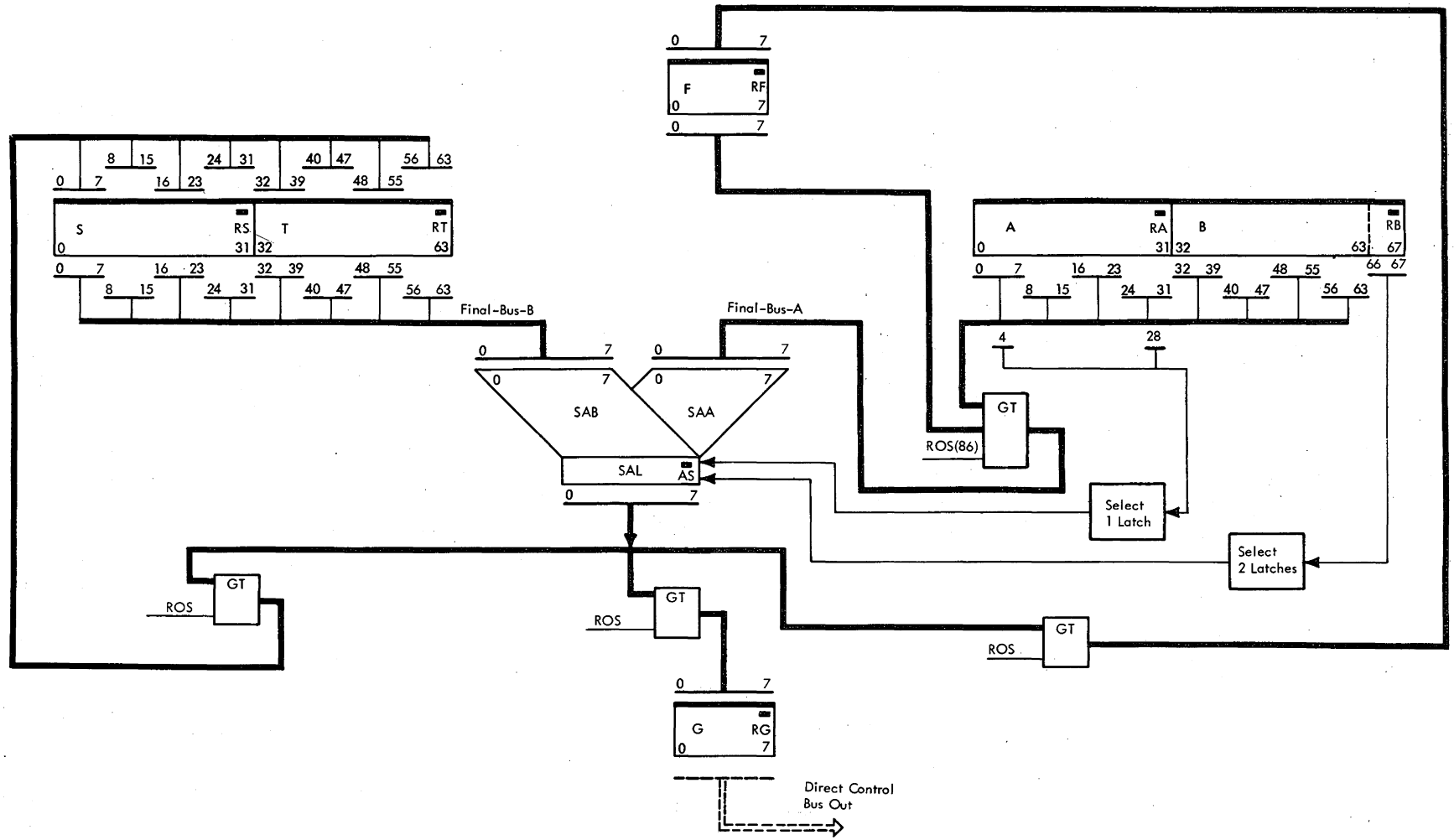


Figure 2-41. Serial Adder Data Flow

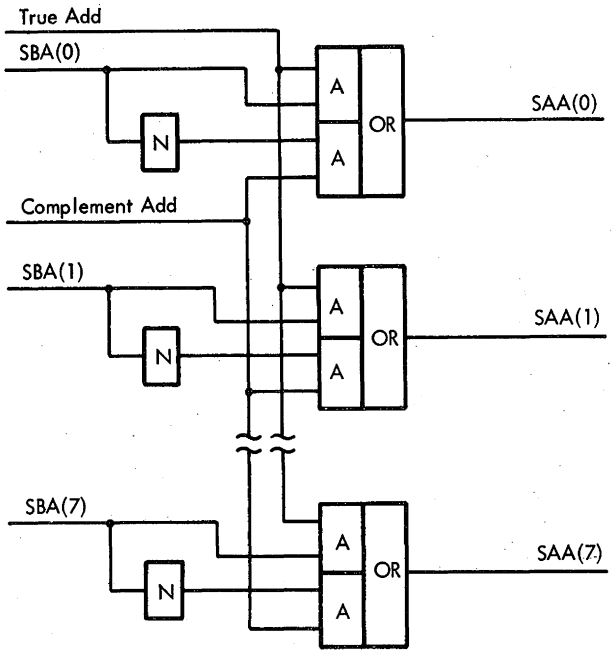


Figure 2-42. True-Complement Data Entry

ROS fields M (bits 69–73) and N (bits 74–77) govern the second area of data control; that is, the transfer of data from the input buses to the A- and B-sides of the adder. ROSDR(69–73) provides signals to control transfer from SBA to SAA. When no control is present, SAA(0–7) is, in effect, 0. Micro-orders allow true-complement transfer, crossgating, excess-6 adding for decimal operations, forcing of certain bits, and sign insertion (Diagram 4-501). ROSDR(74–77) provides similar, though less extensive, control for transfer from SBB to SAB. A list of the micro-orders generated by control fields M and N is contained in ALD M7031.

The third control area, SAL input, is also governed by ROS fields M and N. Controls include logical functions, decimal correction, product/quotient operations, and a hot-carry to SAL(3) or SAL(7) for complement add operations.

Functional Discription

Because the serial adder is used in many operations and is so versatile, a general discussion is not sufficient. Accordingly, the following paragraphs discuss the adder functions individually.

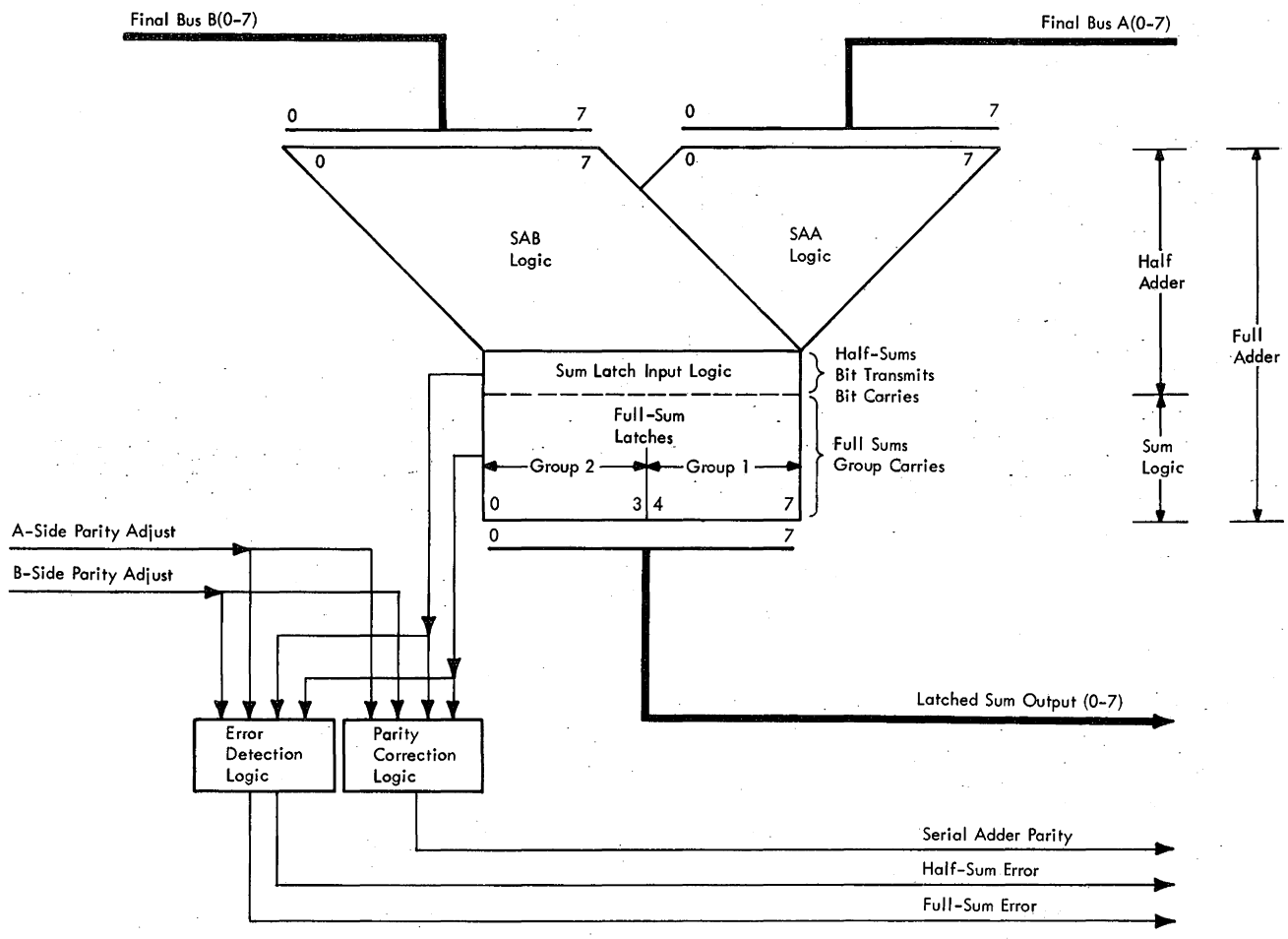
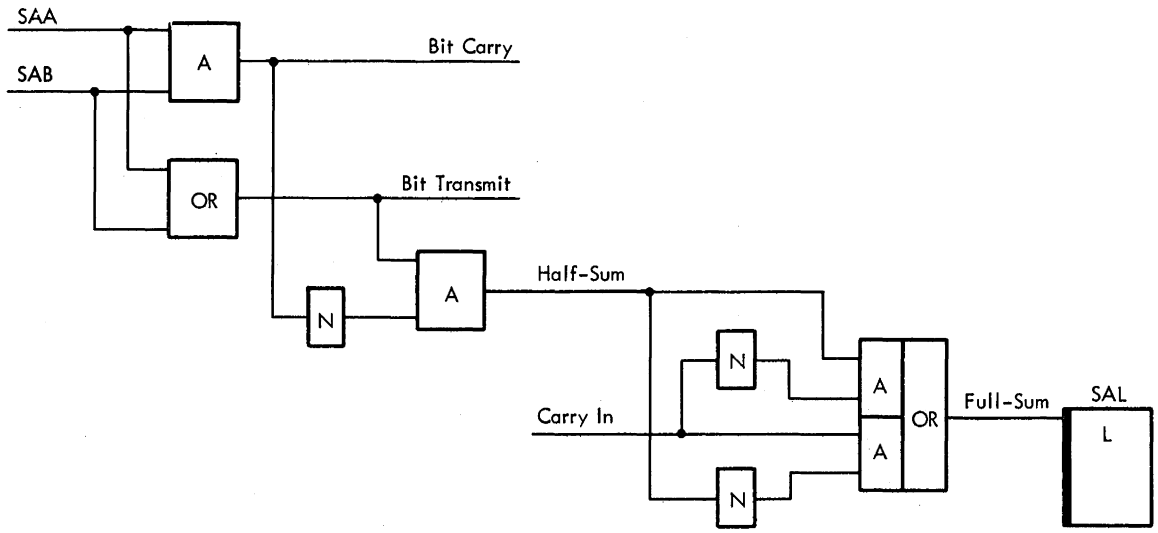


Figure 2-43. Serial Adder (Simplified)



SAA	0	1	0	1	0	1	0	1
SAB	0	0	1	1	0	0	1	1
Bit Carry	0	0	0	1	0	0	0	1
Bit Transmit	0	1	1	1	0	1	1	1
Half-Sum	0	1	1	0	0	1	1	0
Carry In	0	0	0	0	1	1	1	1
Full-Sum	0	1	1	0	1	0	0	1

Figure 2-44. Half-Sum and Full-Sum Logic

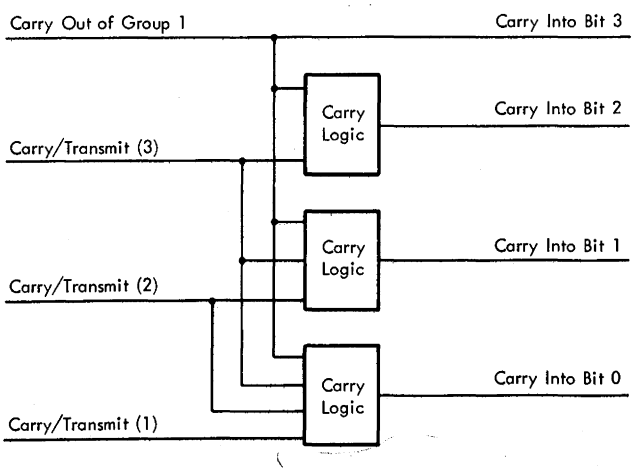


Figure 2-45. Carry Lookahead, Block Diagram

Binary Add

For binary add, data on SBA is entered in true or complement form and is combined with SBB data which may be 0's, forced bits, or data from ST. Combination takes place in the half-sum and full-sum logic with carry signals from the carry lookahead logic (Diagram 4-502, FEMDM).

Decimal Operation

- Excess-6 is provided in input logic to SAA.
- Decimal correction is made in set-SAL logic.
- Validity tests are made on input digits and signs.

The excess-6 operation for decimal instructions is implemented by logical circuits rather than by using extra adder cycles. The decimal character entering on SBA is increased by 6 as it is transferred to SAA (Diagram 4-503, FEMDM). Note that no time is lost in this operation; the circuits select the SAA positions which are 6 (0110) greater than the value on SBA.

The two operands are combined in the half-sum logic. If no group carry results, decimal correction is initiated by a ROS micro-order (Diagram 4-504, FEMDM). Decimal correction removes the excess-6 factor by using logical circuits to set SAL to a value 6 less than the full-sum value. Table 2-1 shows the decimal-corrected values for all possible erroneous characters. Again, because the circuits have been preconditioned, no cycles are lost and the decimal operation proceeds at full speed.

Table 2-1. Decimal Correction for Erroneous Numeric Characters

Group 1 Result					Group 1 Result Decimal-Corrected (Binary Position)			
Decimal	Binary Position 4	5	6	7	4	5	6	7
15*	1	1	1	1	1	0	0	1
14*	1	1	1	0	1	0	0	0
13**	1	1	0	1	0	1	1	1
12**	1	1	0	0	0	1	1	0
11***	1	0	1	1	0	1	0	1
10***	1	0	1	0	0	1	0	0
9****	1	0	0	1	0	0	1	1
8****	1	0	0	0	0	0	1	0
7*	0	1	1	1	0	0	0	1
6*	0	1	1	0	0	0	0	0
5 ↕ 0	Valid digits, no correction required							

* 1-bits in positions 5 and 6: reset positions 5 and 6.
** A 0 in position 6: set position 6 and reset position 4.
*** A 1-bit in position 6 and a 0 in position 5: set position 5 and reset position 4.
**** 0's in positions 5 and 6; set position 6.

The following is an example of decimal correction for SAL(0–3) using Diagram 4-504:

Uncorrected binary result to SAL(0–3) = 1011.
SAL(3): No correction is made. It is set to 1 per binary addition.
SAL(2): Requires carry plus half-sum, or no carry plus no half-sum (effective 0 result). Conditions are not met and SAL(2) is not set.
SAL(1): Requires effective 0 and full-sum (2). Conditions are met and SAL(1) is set.
SAL(0): Requires effective 1 plus full-sum (1 and 2). Conditions are not met and SAL(0) is not set.
Corrected result in SAL(0–3) = 0101.

Incoming data is examined for validity on decimal instructions. If either character on SBA or SBB exceeds a value of 9 (binary 1001), an 'invalid digit' signal is generated and STAT E is set (Diagram 4-505, FEMDM). At the same time, 1's are forced into SAL to yield correct parity for the number transferred to S.

The invalid digit logic is also used to test the sign character entering the serial adder. An 'invalid sign' signal is developed if the sign character does not have a value of 10–15 (binary 1010–1111).

For multiply operations, the product is sent from B(66,67) to selected pairs of SAL bits to accumulate a byte of data. On non-decimal divide operations bits are sent from A(4) and A(28) to selected SAL positions to accumulate a byte of data.

Logical Functions

Logical functions (AND, OR, and Exclusive-OR) are performed in the serial adder. These functions produce full-sum latch settings (carry information from adjacent positions is disregarded) as follows (Diagram 4-506, FEMDM):

1. AND. The combination of an active AND control signal and a bit-carry from the half-adder of that position.
2. OR. The combination of an active OR control signal with a bit-carry or a half-sum (in effect, a bit on either or both inputs) from the half-adder of that position.
3. Exclusive-OR. The combination of an active OER control signal with a half-sum (in effect, either input bit but not both) from the half-adder of that position.

Results are transferred to selected bytes in ST.

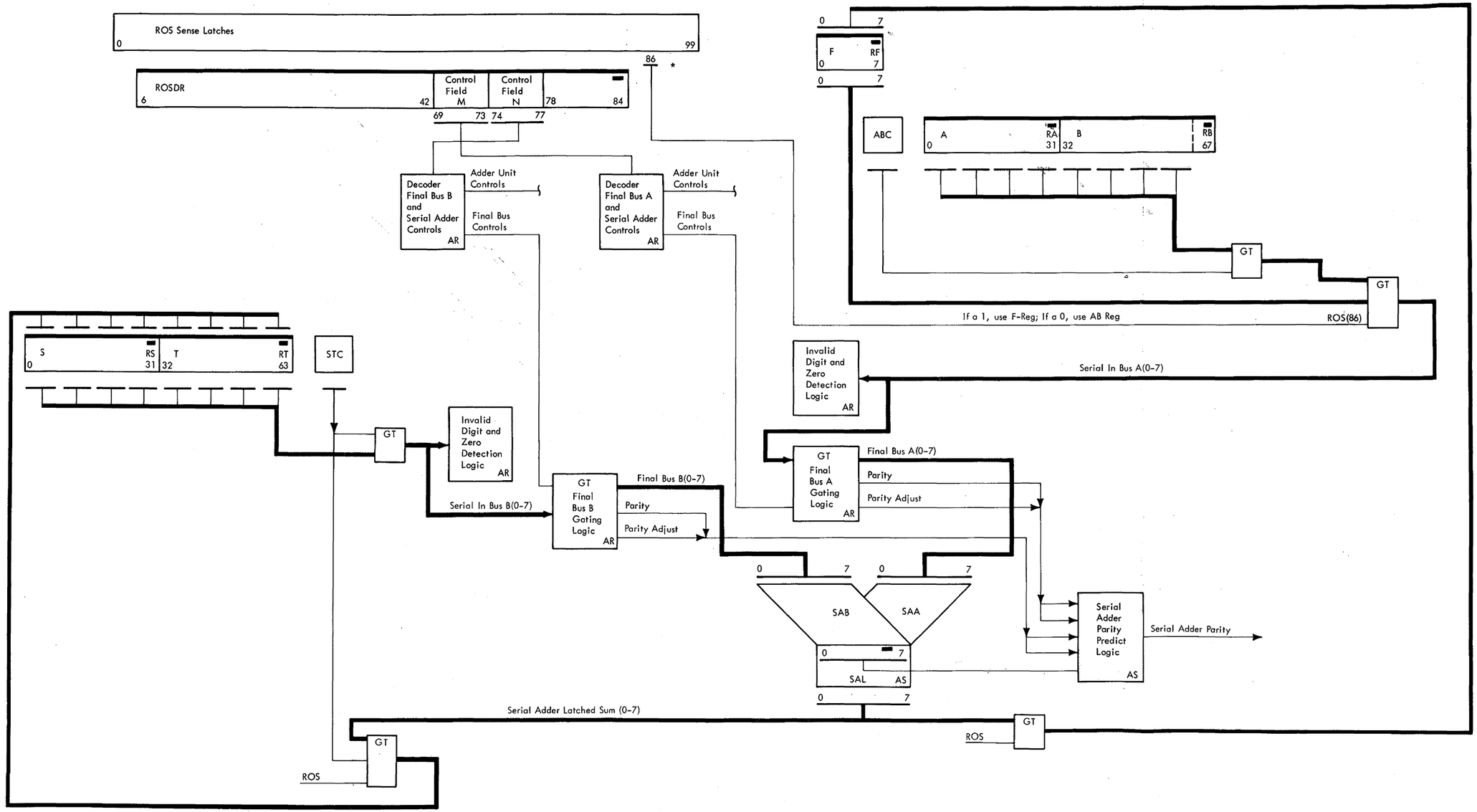
Parity Correction

- Parity bit is set if number of bits in SAL(0–3) and SAL(4–7) are both odd or both even.
- Additional logic predicts parity for decimal operations.
- Parity is reversed on multiply and divide operations if only one bit is sent to SAL.
- Logical operations develop parity through unique logic.

Correct (odd) parity for the serial adder outputs is generated in the parity predict logic (Diagram 4-507, FEMDM). There are two basic areas of parity generation: (1) arithmetic and (2) logical (AND, OR, and Exclusive-OR). The arithmetic parity generation is further divided into binary and decimal. See Figure 2-47, a block diagram of parity predict logic.

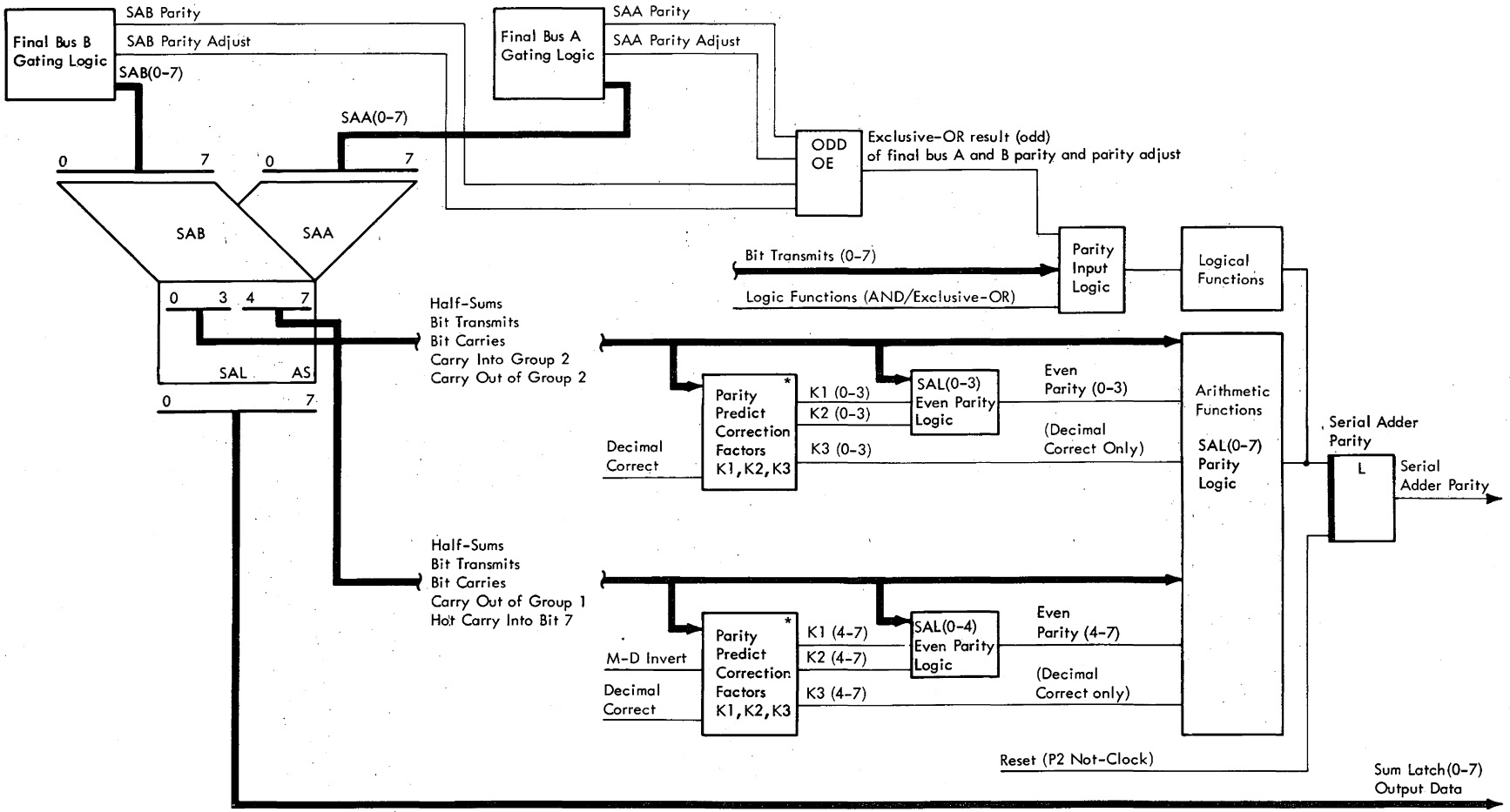
Binary parity predict logic includes factors K1 and K2, half-sum (0), and an odd or even number of transmit bits (1–3 and 5–7). K1 and K2 are established by carry and transmit bits (Diagram 4-507). The signal resulting from these factors reflects the odd or even number of bits in SAL(0–3) and in SAL(4–7). If both are odd or both are even, the serial adder parity latch is set. If only one is odd, the parity latch is not set.

The shaded area in Diagram 4-507 indicates the additional control (K3) used for decimal operations. A decimal correction must be set up (decimal operation and no group carry) to allow energizing of the K3 signal. An examination of carry and half-sum (1, 2, 5, 6) allows the prediction of parity after the excess-6 factor is subtracted from SAL.



* ROS control field R (bit 86) determines whether AB or F data is presented to final-bus-A gating logic.

Figure 2-46. Serial Adder Gating Controls



*K1, K2, and K3 are defined factors of serial adder half-sum, bit-transmit, bit-carry, group-carry, and carry-into-group functions. (K3 factor is activated during decimal-correct operations only.)

Figure 2-47. Serial Adder Parity Predict Logic

Multiply-divide operations present a different problem in that data is presented directly to SAL and is not processed through normal parity predict logic. A test is performed on the two partial product bits or the one quotient bit (Diagram 4-508, FEMDM). A 1-bit change causes an 'invert predicted parity' signal which energizes K2 (4–7). Because no carries are generated, K2 would not normally be energized; thus, the K2 signal inverts the predicted parity.

Logical operations disable arithmetic parity prediction and energize a different parity predict circuit. Three conditions are tested to set the SAL parity latch; (1) input parity, (2) parity adjust, and (3) odd-even transmit bits (Diagram 4-507). The development of each is as follows:

1. Parity. Normally presents the original parity (1 or 0) of the byte being entered on the input bus (one byte for SBA and one byte for SBB).
2. Parity-adjust. In effect, inverts the incoming parity regardless of the actual 1 or 0 parity. Parity-adjust is energized when a micro-order alters incoming data [for example, if SBA(0) enters as a 1 and a ROS micro-order forces SBA(0) to a 0, parity-adjust is energized].
3. Odd-even transmit bits. Exclusive-OR circuits analyze the developed transmit bits of all positions and produce signals denoting an odd or even number of transmit bits.

Parity generation for an OR function is based on the OR command and the OE transmit bits. Because only effective transmits are used to set SAL(0–7), parity generation needs only an even number of transmit bits to set the parity latch. The following is an example of parity generation for an OR function:

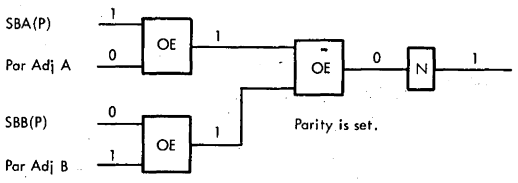
Bit	0	1	2	3	4	5	6	7	P	
SBA	1	0	1	0	0	0	1	0	0	
SBB	0	0	1	1	0	0	1	0	0	
Transmit	1	0	1	1	0	0	1	0		Even
SAL	1	0	1	1	0	0	1	0	1	

4 (even) transmit bits; Parity is set.

The Exclusive-OR function requires the parity and parity-adjust Exclusive-OR logic to generate correct parity. (Because the transmit bits do not reflect parity for Exclusive-OR, they are not used.) Regardless of the number of 1 bits, if the parity of the two input data bytes is different, the resultant will be an odd number of bits and the parity bit will not be set. An exception is caused by micro-order insertion of data over the byte data. This condition is corrected by means of the parity adjust circuits, as shown by the following example:

ROS Force 0

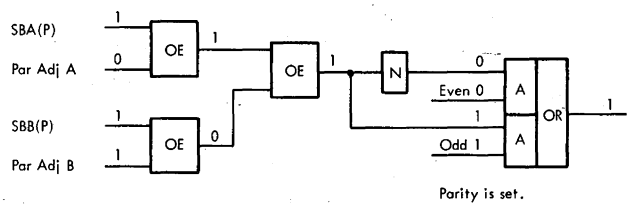
Bit	0	1	2	3	4	5	6	7	P
SBA	1	1	1	1	0	1	1	0	1
SBB	1	0	0	1	0	1	0	0	0
SAL	1	1	1	0	0	0	1	0	1



When the AND function is used, all three signals (SBA and SBB parity, SBA and SBB parity adjust, and odd-even transmit bits) are analyzed to generate parity. The serial adder parity latch is set when the result of the parity and parity-adjust signals matches the odd-even transmit bits signal, indicating an even number of bits have been generated; a parity bit is thus needed. An example of parity generation for a logical AND operation follows:

ROS Force 0

Bit	0	1	2	3	4	5	6	7	P	
SBA	0	0	1	0	1	1	0	1	1	
SBB	1	1	1	1	1	0	1	0	1	
Transmit	0	1	1	1	1	1	1	1		Odd
SAL	0	0	1	0	1	0	0	0	1	



Note: There are four special cases when the SBB input parity bit is set: (1) no-operation, when SBA is all 0's, (2) when -64 (1100 0000 binary) is forced, (3) when subtract 1 (1111 1111 binary) is forced, and (4) on partial product entry when SBB is zero. In addition, the SBB parity bit is held off on add 1 (0000 0001 binary) because it is incorrect parity.

When an invalid digit is detected on a decimal operation and all 1's are forced to SAL(0–7), the parity predict logic is bypassed, and the SAL parity latch is set to give proper parity for the byte sent to ST.

Error Detection

Serial adder data is parity-checked on both a half-sum and a full-sum basis. Error indications are retained in the half-sum error or full-sum error latches (Figure 2-48).

Half-sum error logic tests the incoming data for accuracy. The signals tested are (1) half-sum(0–7), and (2) input parity and parity-adjust for both A- and B-sides of the adder. When an input parity is in error (even parity), the combined signals produce an odd result and the 'half-sum error' trigger is set.

Full-sum error logic tests the accuracy of the final answer by combining the SAL outputs and the resulting state of the parity latch. An even result sets the 'full-sum error' trigger.

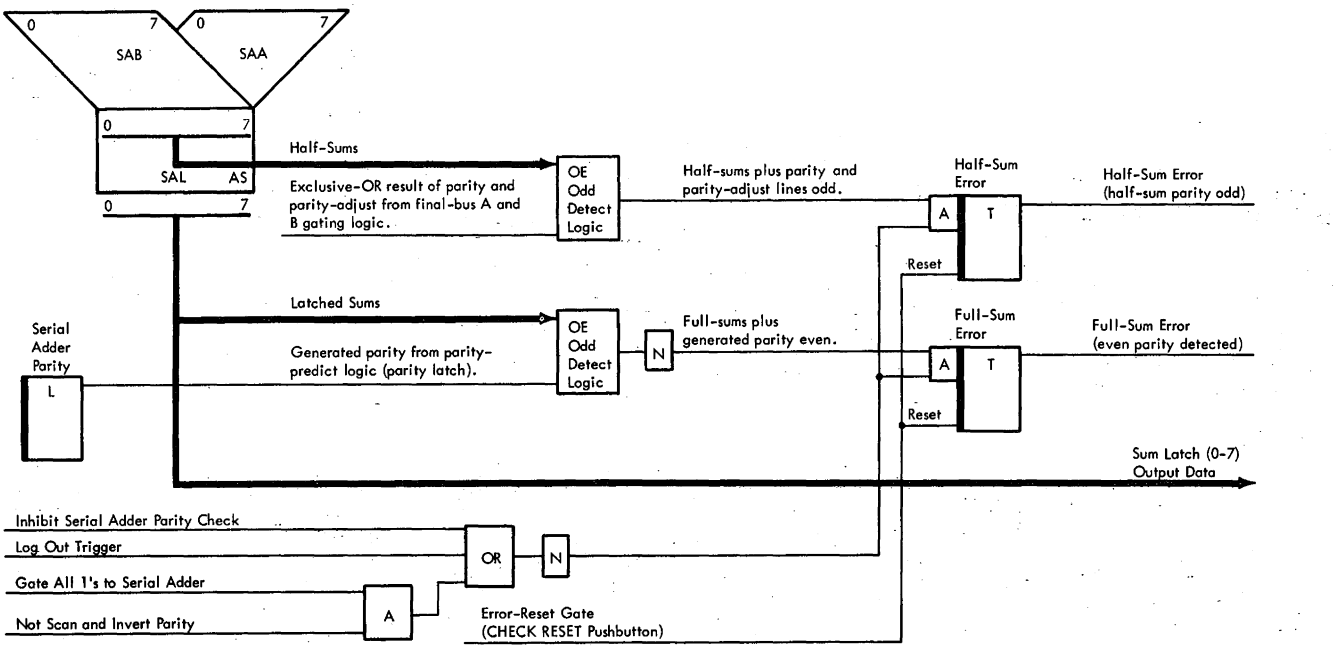


Figure 2-48. Half-Sum and Full-Sum Error Logic

The error triggers remain set until reset by the 'error reset gate' signal. To avoid meaningless error indications and subsequent logout operations, the set error trigger signals are blocked when:

1. The 'inhibit serial adder parity check' micro-order is active.
2. An invalid digit is detected during a decimal operation (all serial adder latches, including the parity latch, are set).
3. The 'logout' trigger is set.

PARALLEL ADDER

The parallel adder, 60 data bits plus parity, performs arithmetic and logical functions and is involved in most intra-CPU data transfers. Data flow for the parallel adder is shown in Figure 2-49.

The parallel adder bit positions are divided into sections and groups to implement carry lookahead and parity predict functions (Figure 2-50). Additional functions illustrated in the figure are half-sum, full-sum, and latch-shifter logic.

Data Input

Data is transferred to the parallel adder from various registers by means of input buses controlled by ROS micro-orders (Figure 2-51). More than one bus may enter the same side of the parallel adder, but only one bus is active at a given time.

ROS fields T and U control inputs to bus B and bus A, respectively. Gate control triggers for adding B and T are shown in Diagram 4-509, FEMDM. ROS sense latches are decoded at P2 time to select a gate control trigger; the trigger remains set until the following P1 time to make the data available as long as the parallel adder needs it. Note in Diagram 4-509 that only 3 of the 4 bits of the ROS fields are used; the state of the fourth bit does not affect the gate control triggers shown but does affect other gate control triggers.

Individual Bit-Position Logic

- Full-binary capabilities (half-adder and full-sum logic) are provided.
- Shift logic (latch-shifter) is included at the output.

The logic functions associated with each adder position are shown in block form in Figure 2-52. These functions (half-adder, carry-into-bit, full-sum, and latch-shifter) constitute the full-binary logic of each bit position; operation is as follows. The status of corresponding A- and B-side operand bits is entered into the half-adder, where they are combined to produce bit-transmits, bit-carries, and half-sums. The bit-transmit/bit-carry is sent to carry lookahead logic to produce predicted carry information, and the half-sum is sent to the full-sum logic. The carry-into-bit logic combines the immediately available bit-transmit/bit-carry from adjacent lower-order adder positions (representing an actual carry) with the somewhat later returning predicted carry. This predicted carry output (carry-into-bit) is also sent to the full-sum logic, where it is combined with the half-sum from the half-adder logic to generate a final full sum (1 or 0) for that adder position. This full

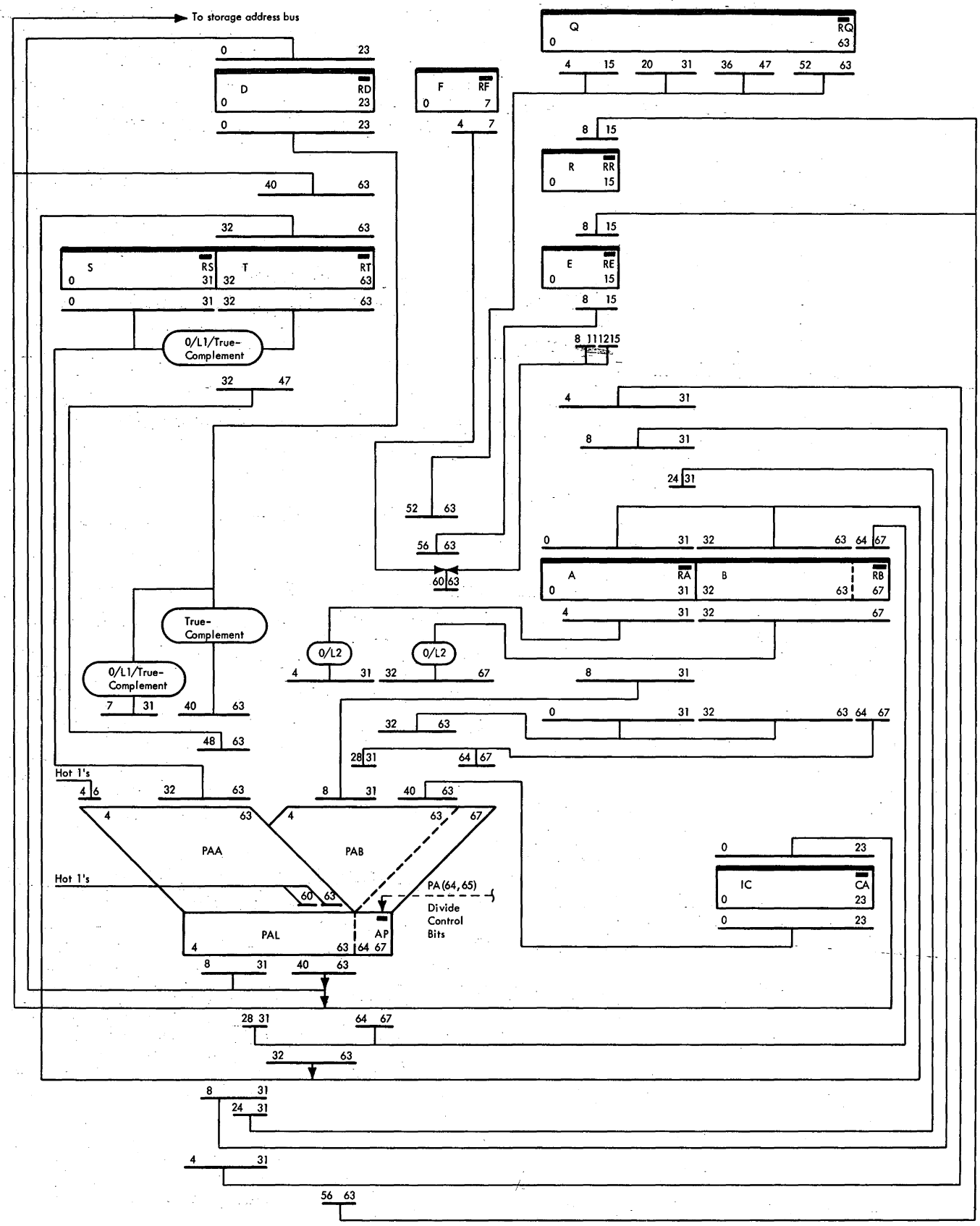


Figure 2-49. Parallel Adder Data Flow

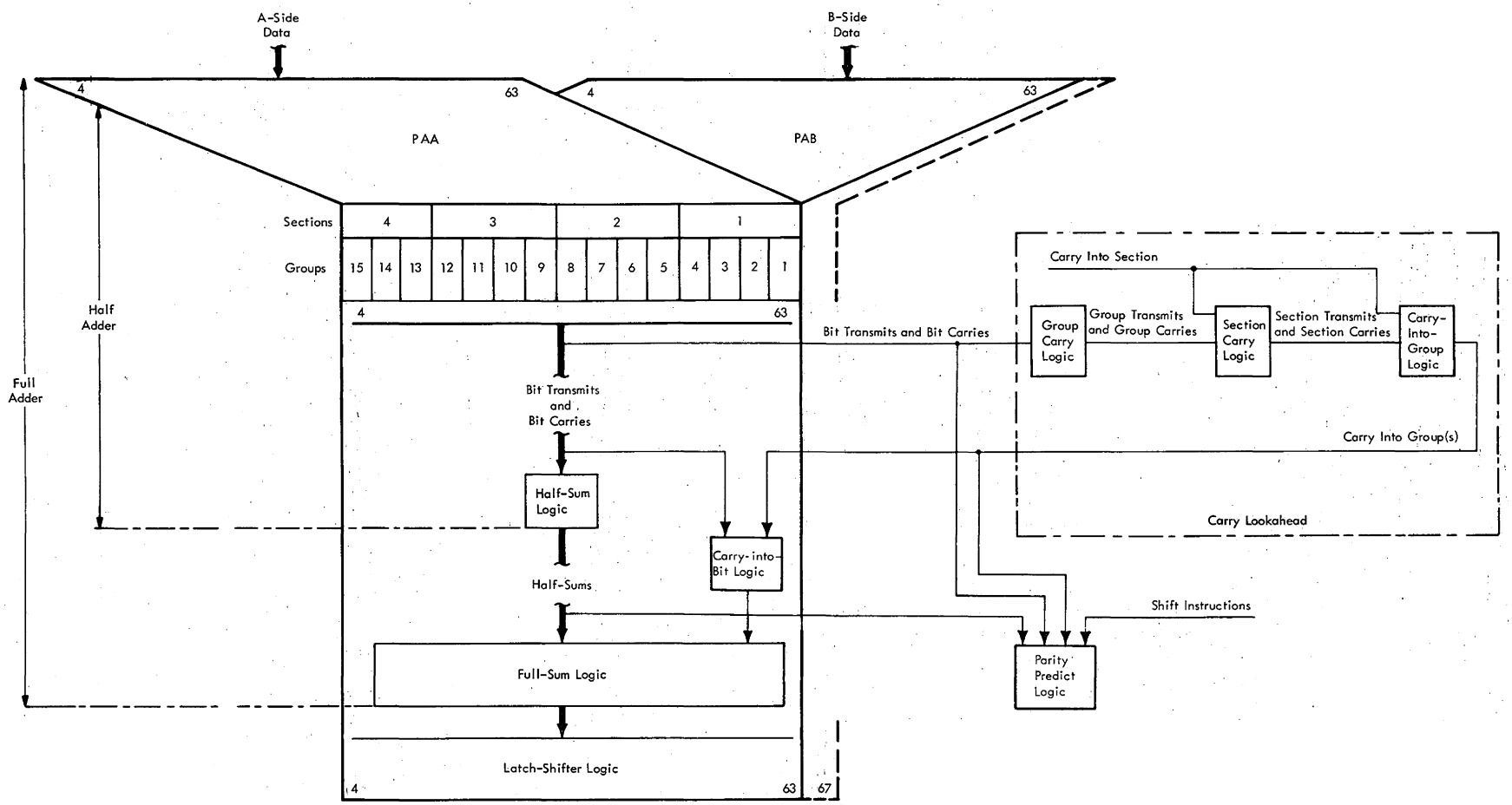


Figure 2-50. Parallel Adder Function Breakdown

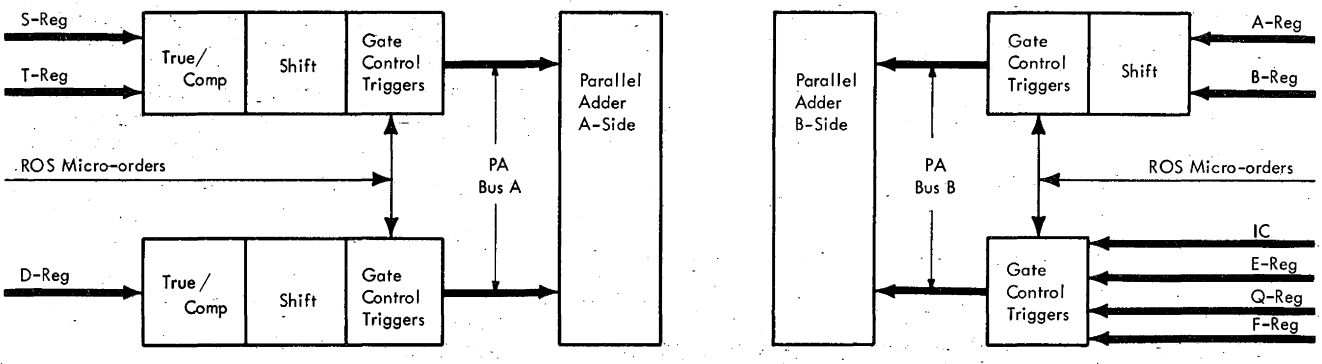


Figure 2-51. Parallel Adder Input Buses

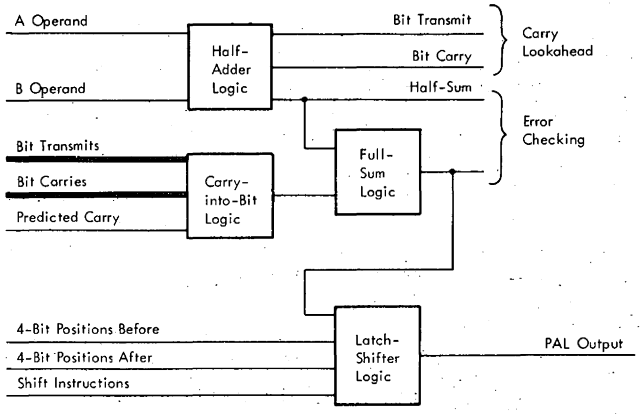


Figure 2-52. Bit Position Block Diagram

sum, or possibly the full sum from four positions to either the left or the right (depending on the particular shift control) is gated into the adder latch. Latched sum data is retained until the following cycle for sampling into the selected register(s). The following paragraphs discuss the logic involved in each function.

Half-Adder

The logic involved in bit-transmit, bit-carry, and half-sum functions are:

1. Bit-transmit. At least one *and possibly two* 1 bits are contained in the two corresponding A- and B-side operand positions.

   Note: For certain operations, the parallel adder is set to all 1's by a micro-order which forces all 1's into the A-side of the half-adder (Diagram 4-510, FEMDM).

2. Bit-carry. Two 1 bits are contained in the corresponding A- and B-side operand positions.
3. Half-sum. A single 1 bit, but not two, is contained in the corresponding A- and B-side operand positions.

Carry-into-Bit Logic

- Detects "carry-into" conditions affecting each particular bit position.

The carry-into-bit logic of each adder position detects whether a carry-into condition prevails, resulting from either an actual carry or a predicted carry. Carry-into-bit circuitry logically OR's the actual carry (prevailing carry conditions from immediately adjacent lower-order positions) with predicted carry (carry-into-group indications from lookahead logic, signifying that effective carry conditions exist in the more extreme lower-order areas).

The actual carry into any particular adder position is determined by logically testing all remaining lower-order bit positions within that same group or, if the particular adder position happens to be the low-order group position, testing all four bit positions of the next lower-order group. (Predicted carries are discussed in "Carry Lookahead".)

Note: Either an actual carry (from adjacent positions) or a predicted carry (from carry lookahead) is allowed to affect a particular position, but not both. Where both carries occur, conditions producing the actual carry also function to inhibit the predicted carry from entering the affected bit position(s). This inhibit logic is illustrated in Diagram 4-510, FEMDM, as follows: Carry conditions from positions 48–51 generate both a predicted carry to position 47 (carry-into-group 5), via carry lookahead logic, and an actual carry in the form of bit-transmit/bit-carry signals. Because group 4 positions (48–51) represent the actual carry source, the group-4-carry condition is then inverted to inhibit predicted carry entries into position 47.

Full-Sum Logic

The full-sum logic for any particular adder position combines (by means of an Exclusive-OR) the carry-into-bit output with the half-sum output of the half-adder, developing a 1 or 0 full-sum for that adder position.

Latch-Shifter Logic

Latch-shifter logic facilitates the left 4/right 4 shifting of the full sum during the same cycle in which it is

developed. (Logical and data-transfer operations also utilize this logic.) For any particular adder position, zero-shift, left-4 shift, and right-4 shift controls respectively gate the full sum into the latch associated with that position, into the latch associated with the position four places to the left, and into the latch associated with the position four places to the right. (Scan-out operations also utilize the latch-shifter logic but only for its data path facilities.) All latches retain the latched sum until the following cycle, and extended-clock signals delay resetting the latches long enough for the error-checking logic to function.

Note: An adder-hold ('→HOLD') micro-order, used during certain operations, blocks the extended-clock reset signal and causes the latches to retain their data for one additional cycle.

Carry Lookahead

- Predicts carry before full-sum development.
- Reduces time required to provide full sum.
- Lookahead logic divides 60-position adder into 15 four-bit groups, and these groups into four sections.
- Lookahead information is developed in form of bit-position carry, group carry, and section carry, and then fed back into individual positions as predicted carries.

The carry lookahead function provides the adder with the capability of entering full sums directly into the adder latches. Lookahead functions effectively predict the carry resulting from combining two operands, and use this predicted carry to convert half-sums to full sums before the entry of information into the adder latches. This sequence eliminates the additional time required by ripple operations, which would be necessary in converting half-sums to full sums if half-sums were entered directly into the latches.

For design reasons, the 60-position adder is divided into 15 four-position groups, and these groups are subdivided into four sections. This group/section arrangement reduces the logic decoding required in implementing the carry lookahead functions. Group/section arrangement and carry lookahead data flow are shown in Figure 2-53.

Lookahead logic is designed so that, for any particular position, the effective carry conditions in all lower-order positions (except for an adjacent few) are logically predicted for that position. Carry conditions which would later be produced in these same adjacent few positions as a result of propagated lower-order carries are predetermined by the lookahead functions and logically entered into that position as a predicted carry. Using this method, each bit position then requires only that logic necessary to detect prevailing carries (actual carry) in the adjacent positions, and to logically OR the actual and predicted carries when developing the full sum. Predicted carries are presented to the input logic of individual positions as 'carry into group' signals. Figure 2-54 illustrates the adder areas supplying source information for actual and predicted carry signals to adder position 44. Note that, although lower-order carry conditions exist in both examples, they are represented by an actual carry in example 1 and by a predicted carry (carry-into-group-5) in example 2. (Recall also, from the previous discussions, that where both actual and predicted carries are generated to a particular position, only the actual carry is entered; the predicted carry entry is blocked.)

In the lookahead logic, predicted carry information is developed by testing each adder group for bit-position carry conditions, combining these conditions to form group-carry conditions, and then similarly combining the group-carry indications to produce section-carry conditions. All lower-order carry conditions affecting any individual adder position (with the exception of the positions immediately adjacent to that position) are then collectively represented to the lookahead logic as section-level carry information. Section-carry information from each section is then (after being combined with lower-order section-carry indications) sent to higher-order sections, where it is combined with the group-carry conditions within these sections, to produce 'carry into group' signals. As previously described, these 'carry into group' (predicted-carry) signals are then logically OR'ed with actual carries within the carry-into-bit logic of each individual adder position, and combined with half-sum information to generate a full sum. (Recall also that, where both actual and predicted carries are generated to a particular position, only the actual carry is entered and the predicted carry entry is blocked.)

The following paragraphs give detailed descriptions of group-level and section-level carry-predict functions.

Group-Level Carry Logic

- Bit-position carry conditions are combined in four-bit groups to generate group-carry conditions.
- Group-carry logic outputs define effective status of all bit positions composing a group.
- Group-carry outputs are sent to section-carry logic.

The group-level carry information generated for any particular group is determined by logically combining the bit-transmit/bit-carry outputs of all four positions within that group. Group-transmit/group-carry signals are then

generated and sent to the section-level carry logic of the section in which the particular group is located. Group-level carry logic outputs indicate the following:

1. Group transmit. Signifies that all bit positions within that group have received at least one bit of operand data, i.e., bit-transmit conditions exist throughout the group.
2. Group carry. Signifies that bit-transmit/bit-carry conditions within that group are such that an effective carry condition exists from the high-order position of that group.

Group-level carry logic is illustrated in Diagram 4-511, FEMDM. Note in the diagram that group-transmit/group-carry outputs are determined solely by bit-transmit/bit-carry conditions, which represent incoming data only (without the use of any propagated carry information).

When group-carry conditions are sent to their associated section-level carry logic, they may also (at the same time) generate 'carry into group' signals to adjacent higher-order groups within that same section. This sequence results in the immediate propagation of group-carry information (within that same section). Carry-into-group circuits that are not activated at this particular time may be activated somewhat later by incoming section-carry signals from lower-order sections.

Because group-carry conditions are used in developing section-carry conditions, a time differential exists between the two logic functions. The timing relationships between bit-carry, group-carry, section-carry, and carry-into-group are discussed in "Arithmetic Function Sequence".

Section-Level Carry Logic

- Group-level carry conditions are combined to develop section-level carry conditions.
- Section-carry outputs define effective carry conditions of all bit positions within a particular section.
- Section-carry outputs are sent to higher-order sections as predicted carry information.

Section-level carry information is determined by logically combining the group-transmit/group-carry outputs of all groups within a section. Like group-carry generation, section-carry outputs are also determined solely by group-carry conditions (without the use of any carry propagation). Section-level carry logic outputs signify the following:

1. Section-transmit. Indicates that all bit positions of all groups within that section have received at least one bit of operand data (i.e., group-transmit conditions exist throughout the section).
2. Section-carry. Signifies that group-transmit/group-carry conditions within that section indicate that an effective carry condition exists in the high-order bit position of that section.

The section-level carry logic (Diagram 4-511) develops a 'section 1 transmit' signal from group transmits and a 'section 1 carry' signal from combinations of group-transmits and carries.

Section-Level Carry-Into Logic

A section-carry generates a carry into the next higher order section. The section-level-carry-into logic (Diagram 4-511) develops carry-into-section signals, starting with a 'carry into section 1' signal produced by a hot-carry. Carry into sections 2, 3, and 4 are developed by section-transmit and section-carry logic.

Group-Level Carry-Into Logic

Carry-into-section signals produce a carry-into-group signal for at least the low-order group of the section. Development of additional carry-into-group signals is dependent on group carry/transmit conditions (Diagram 4-511). Note that carry-into-group signals may be developed independently from the carry-into-section signals.

Bit-Level Carry-Into Logic

Carry lookahead conditions the bit-level carry-into logic of the low-order group position if no group carry is present from the next lower-order group. Other bits in the group are conditioned if intervening low-order transmit bits are present (Diagram 4-511). For example, if a bit 49 carry and a bit 48 transmit have been developed, the result is a group 4 carry that generates a section 1 carry. The section 1 carry and a section 2 transmit produce a 'carry into section 3' signal. A 'carry into group 9' signal results from the 'carry into section 3' signal. When bit-transmits 30 and 31 are present, a carry into bits 29 and 30 takes place to develop full sums.

Diagram 4-511 shows the timing relationships for carry lookahead. Note that although a direct carry occurs before a carry lookahead, this time difference does not affect the final sum development which takes place after all carry circuits have settled down.

Full-Sum Development

- Half-sums are combined with carry information (actual and predicted) to develop full sum.

The manner in which carry lookahead and half-sum functions are logically combined to produce a full-sum result is illustrated in Figure 2-55. Note that all group-level and section-level functions are arranged on a section (four-section) basis, whereas carry-into-bit and full-sum functions appear in each adder position.

The complete carry lookahead system is shown in Diagram 4-511; a summary of carry-predict operation is as follows. When operand data is presented to the A- and B-sides of the adder, the half-adders of all positions are

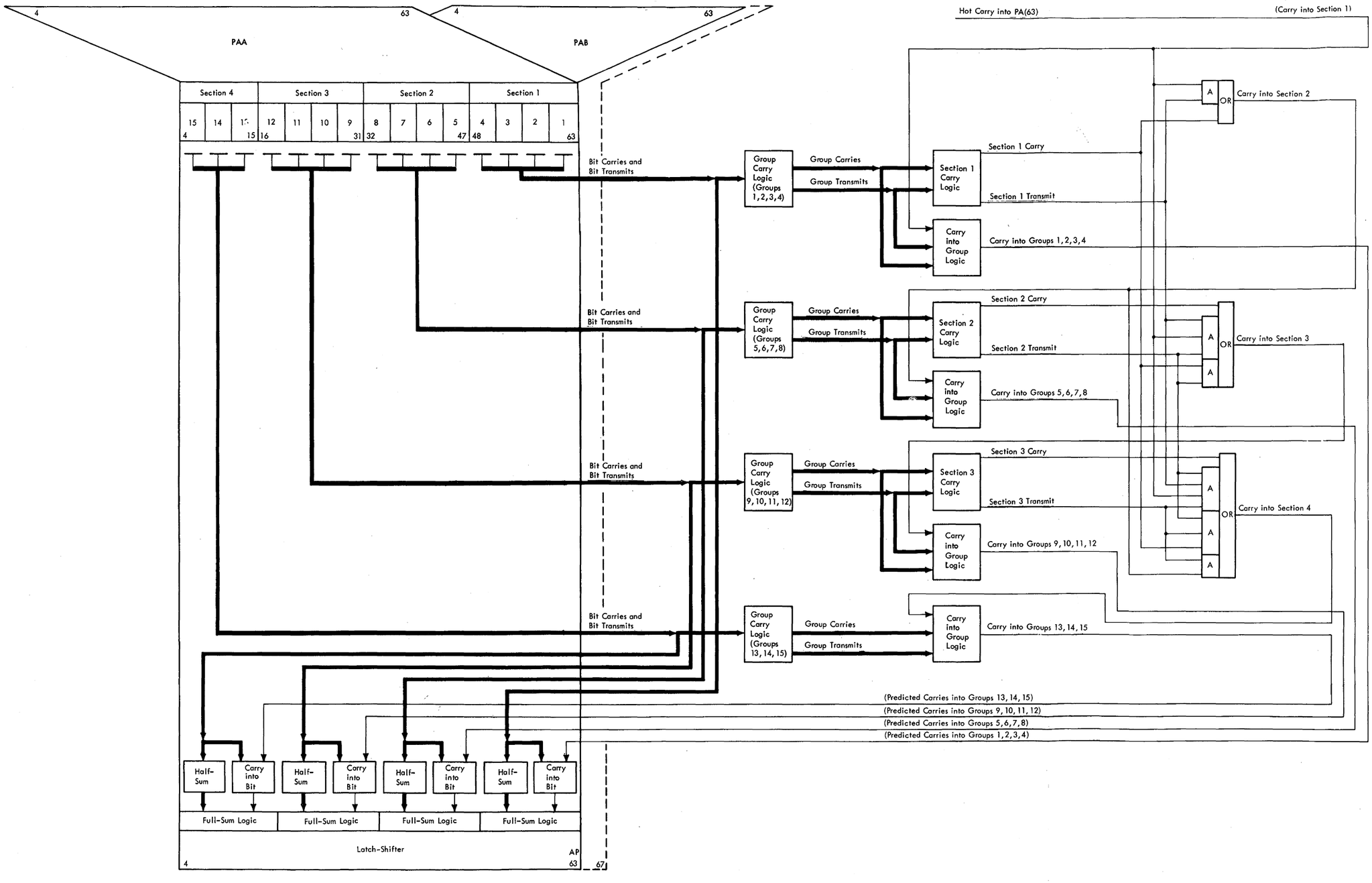


Figure 2-53. Parallel Adder Carry Lookahead Data Flow

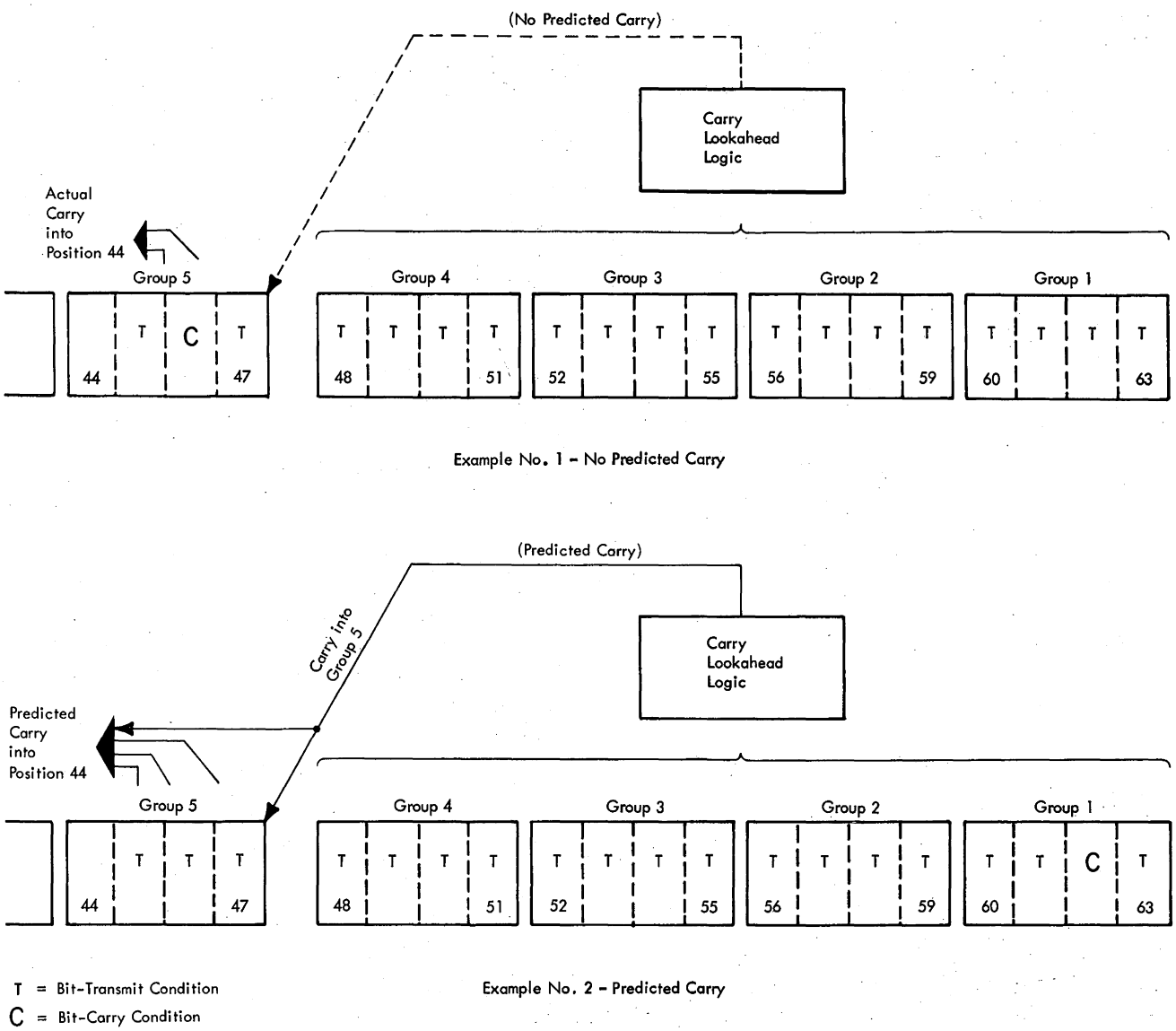


Figure 2-54. Actual and Predicted Carry Origin for PA(44)

sampled for bit-transmit/bit-carry information. (Half-sums are also generated from the half-adders and presented to the full-sum logic of each position at this time.) All bit-transmit/bit-carry information is sent to the associated group-carry logic, and all group-carry outputs are entered into their respective section-carry logic. Section-carry outputs now represent the carry status that logically prevails in the high-order position of each section (without any effects of carry propagation). All section-level carry outputs are then combined with lower-order section-carry information to determine whether a 'carry into section' (predicted-carry) signal is generated for the higher-order section(s). 'Carry into section' signals sent to higher-order sections combine with group-carry conditions within those sections to produce the carry-into-group conditions that represent predicted carries for the individual bit positions. The carry-into-bit logic of each individual position then logically OR's 'carry into group' (predicted-carry) signals with actual carry indications, and this output combines with the half-sum to produce the full-sum result.

Note that throughout the lookahead sequence no ripple operations are required. Definite cycle times, however, are associated with each predict function (bit-carry, group-carry, section-carry, carry-into-section, and carry-into-group); these times are discussed in the following paragraph.

Arithmetic Function Sequence

- Eight logical delay levels are required for arithmetic functions.
- Extended clock signals are used within adder.
- Full-sum results are latched (retained) for 1 cycle.

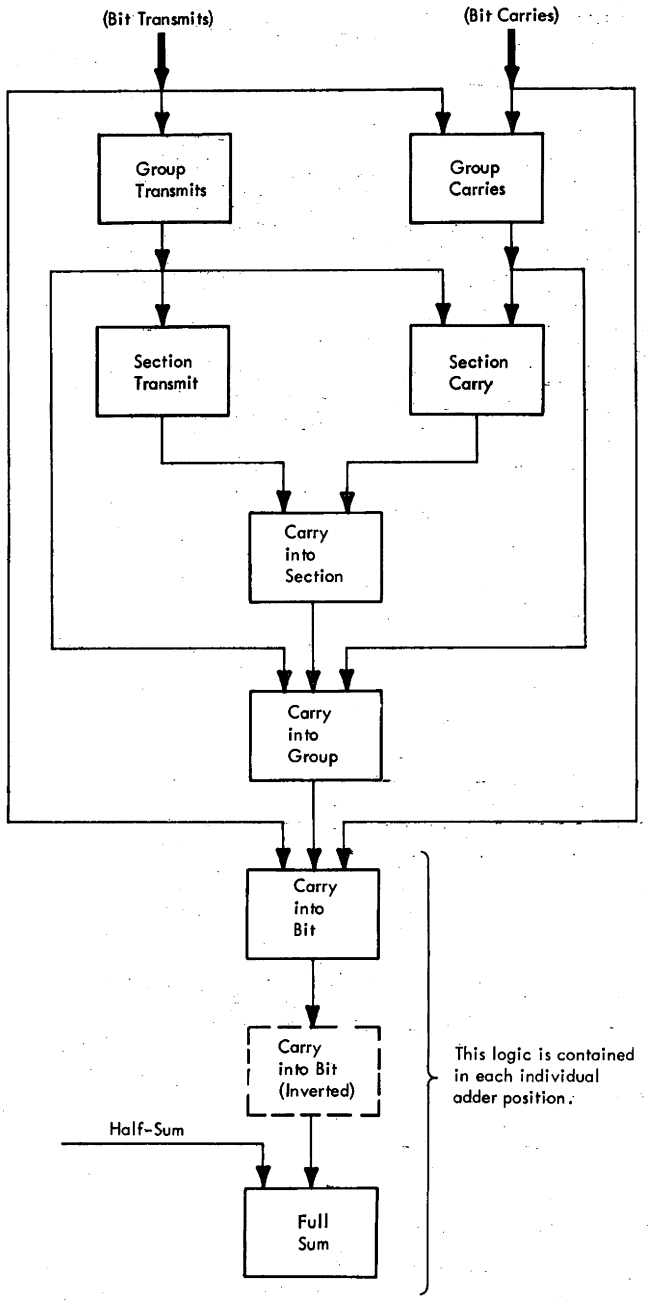


Figure 2-55. Full-Sum Development Logic

The timing sequence in which all adder logic operates to develop and check full-sum information is shown in Figure 2-56. Three delay levels (P4–P7) occur between the time at which data is placed on the adder input bus (by the associated gate-control triggers) and the time at which the same data enters the half-adders. (Two of these delay levels result from bus-gating delays; the third, from the signal cables.) Eight levels of signal delay, then, are required within the adder for the full-sum development process. As noted on the timing chart in Figure 2-56, the logic functions that require the eight delay times occur in the following sequence:

1. Bit-carry/transmit.
2. Group-carry/transmit; carry-into-bit (direct).
3. Section-carry/transmit.
4. Carry-into-section.
5. Carry-into-group.
6. Carry-into-bit (predicted).
7. Carry-into-bit (inverted).
8. Full-sum.

Note: A carry-into-bit can originate early from a direct carry or late from the predicted carry logic.

Extended clock signals are used within the parallel adder to control all latches. The clock portion of the normal CPU clock signal is extended two delay levels (approximately 20 ns), producing a symmetrical clock signal of 100-ns clock and not-clock times. These extended clock signals result in delaying both the setting and resetting of the adder latches. Delaying the setting of the full-sum latches provides additional time for carry-predict functions, and delaying the resetting of the latches retains latched sum information long enough for sampling by the error-checking logic.

Full-sum information contained in the adder latches is normally retained one cycle. For certain operations, however, an adder-hold ('→HOLD') micro-order inhibits the clock signal that resets the adder latches, thus retaining latched sum information for one additional cycle.

Parity-Predict Logic

- Odd parity is supplied with each byte (or half-byte) of adder output data.
- Parity generation is simultaneous with full-sum development.
- Parity-predict logic utilizes inputs from half-adders and carry lookahead logic.
- Parity generation is corrected accordingly for left-4/right-4 data shifting.

Odd parity is generated for each byte (or half-byte in the case of positions 4–7 and 64–67). Predict logic is employed, allowing parity information to be generated simultaneously with the development of full-sum data. (This scheme eliminates the time involved in analyzing the full-sum bit count to determine parity.) Parity is initially predicted for each four-bit group of adder output data. For the eight-bit byte outputs, the parity information predicted for the two adjacent four-bit groups that constitute a particular byte is combined (Exclusive-OR'ed) to determine the full-sum parity of that byte. Because the adder is also capable of shifting full-sum data

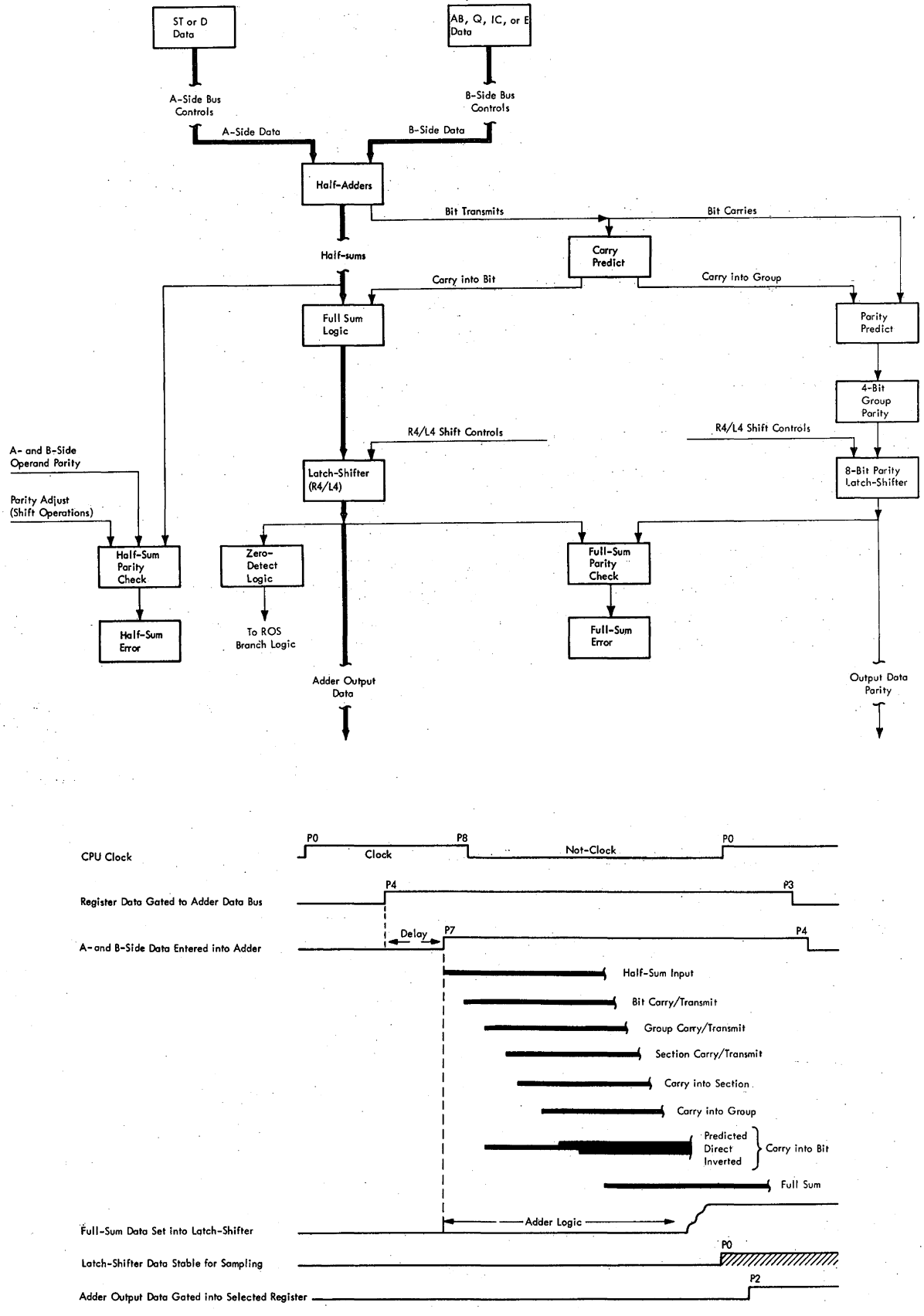


Figure 2-56. Parallel Adder Logic Function Sequence

left 4 and right 4 (before entry into the adder latches), generation of correct byte parity for left-4/right-4 operations then becomes a matter of selecting which two adjacent four-bit group parity outputs to combine when determining the parity of a particular output byte.

Parity is logically predicted through functions of the incoming operand data; operation is as follows. At the same time half-adder outputs are sent to the lookahead logic (to predict carry information), they are also sent to four-bit group parity-predict logic. A typical four-bit group parity-predict function is shown in Diagram 4-512, A, FEMDM. (Group 4 is used as an example; all groups are similar.) For each four-bit group, bit-transmit, bit-carry, and half-sum outputs from half adders and carry-into-group outputs from the lookahead logic are combined to logically predict whether the resultant full-sum bit count for that particular group will be odd or even. Note in the diagram that duplicate decoder logic is present in each four-bit group parity-predict circuit. This duplicate logic simultaneously produces the opposite polarity (out-of-phase) signals required for use in the eight-bit parity latch-shifter logic without the signal delay introduced if an additional inversion component were used.

Typical parity latch-shifter logic used in combining two adjacent four-bit group parities to determine eight-bit byte parity is shown in Diagram 4-512, B. (Adder output byte 48–55 is used as an example; all byte parity logic is similar.) Note in the diagram that the two four-bit group parity outputs to be combined (Exclusive-OR'ed) when determining byte parity are selected according to the type of shift operation in process; i.e., left 4, right 4, or no shift (straight transfer). The generated parity for each adder byte (or half-byte in the case of positions 4–7 and 64–67) is set into the corresponding parity latches for transfer with the data and sent to the full-sum error-checking logic. Because parity information is used in full-sum error checking and both parity and full-sum information are formed independently, an inconsistency in either will cause a full-sum error.

Error Checking

Parallel adder logic employs both half-sum and full-sum checking facilities. Half-sum checking verifies incoming data (in regard to assigned parity only); this test also results in verifying half-adder operations because half-sum outputs are used in half-sum checking logic. Full-sum checking logic compares the full-sum bit count (odd/even) with the generated parity information on a byte (or half-byte) basis. Because full-sum and parity information are formed independently, an inconsistency in either results in a full-sum error.

Half-Sum Checking

- Compares half-sums with incoming operand parity.

Half-sum checking logic combines the parity information assigned to incoming A- and B-side operand data with the half-sum generated when the same two operands are combined in the half-adders. This combining of parity and half-sums is performed on a byte (or half-byte) basis, with detected errors stopping the CPU clock and lighting indicators signifying the byte (or half-byte) in error.

Half-sum checking logic, illustrated in Diagram 4-513, FEMDM, operates as follows. A stage of precheck logic for each byte (or half-byte) combines half-sums with the corresponding A- and B-side parity information in odd-detect (Exclusive-OR) circuits. (The precheck logic shown in Diagram 4-513 monitors adder positions 48–55.) This logic functions so that, if the number of half-sums, plus the A- and B-side parity bits, results in an odd bit count, the half-sum precheck trigger for that associated adder area is set. Precheck outputs from all adder areas are then combined with left-shift logic at the input to the 'half-sum error' trigger. This left-shift logic determines whether an actual half-sum error exists or whether shifting the register data left 1 or left 2 positions (while en route to the adder) has forced a half-sum error.

Note: Left-shifting the adder input data left 1 or left 2 positions invalidates the assigned parity information, thus forcing half-sum errors. The number of half-sum errors created, however, should result in an even number; i.e., the half-sum errors forced into positions 4–31 should equal the number of half-sum errors forced into positions 32–63. Odd-detect logic, therefore, allows only an odd number of half-sum precheck indications to set the 'half-sum error' trigger during a left 1/left 2 shift operation.

If a valid half-sum error exists, the 'half-sum error' trigger is set, thus setting the 'final error' latch. Setting the 'final error' latch prevents the 'half-sum error' trigger from automatically resetting, which in turn prevents resetting the precheck logic for the area in which the half-sum error occurred. Inhibiting these resets causes the CPU program to stop on the following cycle (provided the CPU CHECK switch is in the STOP position), with the HALF SUM error indicator for the area incurring the error displayed on the roller switch indicators.

The half-sum error indications are reset by the 'error reset gate' signal (SYSTEM RESET or CHECK RESET pushbutton).

Full-Sum Checking

- Compares latched sum information with generated parity. ON A BYTE BASIS

Full-sum checking logic combines latched sum information with generated parity information on a byte (or half-byte) basis. (Full-sum checking for adder positions

48–55 is shown in Diagram 4-514, FEMDM.) Because the CPU operates with odd parity, combining full-sum bits with generated parity should always result in an odd bit count. Detecting an even latched-sum-plus-parity bit count sets the 'full-sum error' trigger for that particular adder byte area, which in turn sets the 'final error' latch. The error signal that sets the 'full-sum error' trigger stops the CPU program on the following cycle (provided the CPU CHECK switch is in the STOP position) and lights a FULL-SUM error indicator on the roller switch indicators, signifying the area incurring the full-sum error.

For practical reasons, combining full sums with parity is logically accomplished by first Exclusive-OR'ing the generated parity with a single latched sum position (Diagram 4-514), and then combining that result with the remaining latched sums of that particular byte. An odd result (signifying an *even* overall bit count) then sets the associated 'full-sum error' trigger.

Full-sum error conditions are reset by the 'error reset gate' signal (SYSTEM RESET or CHECK RESET push-button).

Convert-to-Decimal Operation

Special circuits, used only in the convert-to-decimal operation, provide excess-6 decimal correction when required. Excess-6 is forced on the PAB bus when a test of a four-bit group indicates a decimal value higher than 9. For this operation, parallel adder bit positions 28–63 are logically divided into four bit groups, each group representing a decimal digit in the packed format. Diagram 4-515, FEMDM, shows the development of excess-6 signals for PAB(28–31) and PAB(60–63). Note that these signals are activated only when ROS has developed the excess-6 gate and when the AB bits indicate the need for decimal correction.

For this operation, data is brought into the parallel adder one bit at a time by transferring one byte of data to the serial adder and sampling SAL(0). If SAL(0) = 1, the 'conv dec' trigger is set and a hot-carry sets PAL(63) (Fig. 2-57). The contents of the serial adder are then shifted left 1 position so that the next bit can be sent to the parallel adder (which is also shifted left one position). [For details, see Chapter 3, Section 2, "Convert to Decimal, CVD (4E)".] Because data is processed through normal parallel adder entry logic, parity generation takes place in a normal manner.

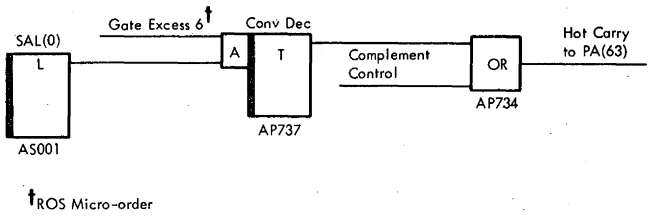


Figure 2-57. Convert-to-Decimal Data Flow to Parallel Adder

Set Condition Code

After an operation, PAL is analyzed to set the PSW condition code (CC). CC's are set in many ways with many variables for different instructions; Diagram 4-516, FEMDM, shows a typical example. Various sections of PAL are examined for a zero condition; combinations of PAL equal zero and micro-orders set STAT A, which is sampled by the instruction and the result sign to set the CC.

In the example, if PAL(32–63) is not equal to zero and the result is negative, a CC of 1 is set on a fixed-point operation. (This setting indicates a number less than zero.) Note that an overflow condition on a fixed-point instruction sets both CC bits, regardless of the condition of PAL.

For a floating-point operation PAL(7–67) is examined for zero. A not-zero condition and a plus result set a CC of 2.

This section discusses the eight Status Triggers (STAT's A–H) and miscellaneous control triggers. A summary of the conditions that set STAT's A–H is shown in Figure 2-58, and a typical example of STAT logic (showing STAT B) is illustrated in Diagram 4-601, FEMDM.

STAT A

- STAT A indicates:
  Zero condition for parallel adder.
  Non-zero condition for serial adder.
  Digit condition on edit operations.

STAT A primarily indicates zero-detect conditions. Except during scan-in, when it is set directly to the value of T(54), STAT A is normally set at P2 clock time by one of the following conditions:

1. 'Set STAA if SAL(0–7) not equal to zero' signal, with SAL(0–7) not containing all zeros.
2. 'Edit set STAA' signal (edit operations).
3. 'Serial adder (0–3 or 4–7) not zero' signal, from the 'serial adder not zero' (SNZ) latch (indicating that the serial adder latched outputs do not contain all zeros).

   Note: The 'SNZ' latch is set at not-clock time by the 'decimal correct 0–3 set STAT's AE', or 'serial carry-7 STAT's AE decimal correct 4–7' signal.

4. 'Set STAA if PAL(7–63) equals zero' signal, with PAL(7–63) latched outputs containing all zeros.
5. 'Set STAA if PAL(32–63) equals zero' signal, with PAL(32–63) latched outputs containing all zeros.
6. 'Set STAA if PAL equals zero and insert sign' signal, with E(6) = 1.

The output of STAT A is entered directly into a polarity-hold latch, which unconditionally assumes the same binary state as STAT A at P1 not-clock time. (This latch retains its assumed state until not-clock time of the cycle in which STAT A is reset.)

STAT A is reset at P1 clock time if one of the following conditions is active:

1. 'STAT trigger reset' signal (conditioned by either a 'system reset' or an 'I-Fetch reset' signal).
2. 'Reset STAA' signal, which is conditioned when any one of the following is active:
   a. 'Edit reset STAA' signal.
   b. 'Set STAA if PAL(0–63) equals zero' signal.
   c. 'Set STAA if PAL(32–63) equals zero' and signal.
   d. 'Set STAA if PAL(32–63) equals zero and insert sign' signal, with E(6) = 1.
   e. 'Reset STAA if PAL(32–63) not equal zero' signal, with STAA polarity-hold latch set.

      Note: If PAL(32–63) contains all zeros and the STAA polarity-hold latch is set, the 'reset STAA if PAL(32–63) not equal zero' signal is inhibited from resetting STAT A.

   f. 'Set STAA if SAL(0–7) not equal zero' signal.

STAT B

- STAT B indicates:
  Zero-condition for serial adder.
  Overflow condition for decimal, fixed point, left-shift operations.
  Condition of PAL(31).
  Condition of B(32).

STAT B primarily indicates overflow conditions. Except during scan-in, when it is set directly to the value of T(55), STAT B is normally set at P2 clock time by one of the following conditions:

1. 'Set STAB if SAL(0–7) equals zero' signal, with SAL(0–7) containing all zeros.
2. 'Set STAB on decimal overflow' signal, with the 'decimal overflow' latch set.

   Note: The 'decimal overflow' latch is set at not-clock time of a decimal-compare cycle in which:
   a. The 'serial adder in bus A(7)' contains a 1 bit, and either STAT A, STAT D, or STAT H is reset.
   b. The 'serial adder in bus A(0–6)' is not equal to 0.
   c. STAT H is set, with STAT C and STAT F either both set or reset.

3. 'Set STAB if PAL(31) equals 1' signal, with PAL(31) = 1.
4. 'Gate fixed-point overflow to STAB' signal, with a fixed-point overflow condition prevailing.
5. 'Set STAB on left shift overflow' signal, with a left-shift overflow condition detected.
6. 'B(32) to STAB and T(32) to STAG' signal, with B(32) = 1. [STAT B is set at P2 + 140 ns under this condition to allow B(32) to become stable before it is sampled.]

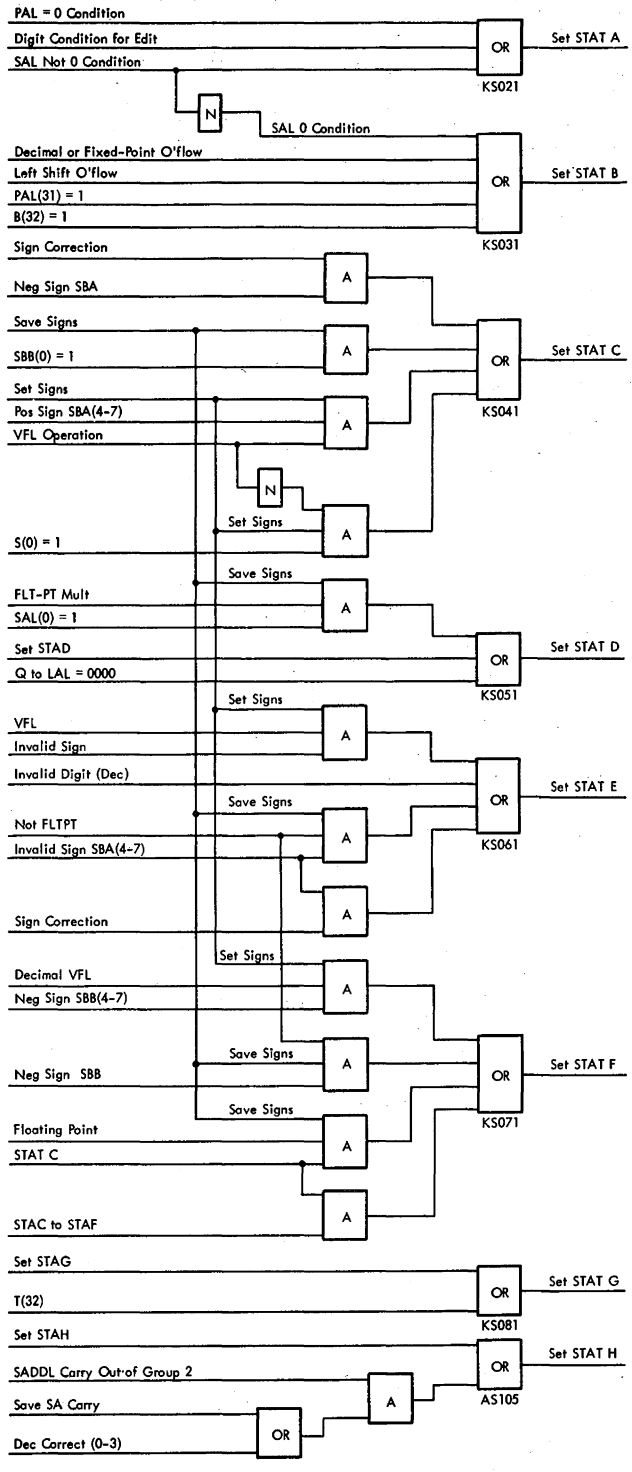


Figure 2-58. Summary of Setting of STAT's

The outputs of STAT B are sent directly to a polarity-hold latch. This latch unconditionally assumes the state of STAT B at not-clock time of the cycle in which STAT B is set, and retains this information until not-clock time of the cycle in which STAT B is reset. The output from the STAT B polarity-hold latch inhibits the resetting of STAT B whenever STAT B is set during the same cycle in which either a fixed-point overflow or a left-shift overflow is detected. (Either of these overflow conditions causes a program interruption requiring that STAT B remain set for interrogation.)

STAT B is normally reset at P1 clock time if one of the following conditions is active:

1. 'STAT reset' signal (conditioned by either a 'system reset' or an 'I-Fetch reset' signal).
2. 'Reset STAB' signal, which is conditioned if one of the following signals is active:
   a. 'Set STAB if PAL(31) equals 1'.
   b. 'Gate fixed-point overflow to STAB'.
   c. 'Set STAB on left shift overflow'.
   d. 'B(32) to STAB and T(32) to STAG' (resets STAT B at P1 + 140 ns).

STAT C

- STAT C holds:
  'Serial adder in bus A' sign on sign-correction VFL operations.
  'Serial adder in bus B' sign on save-signs VFL operations.
  Sign for set-signs on VFL and non-VFL operations.

STAT C primarily indicates the sign of a source operand. Except during scan-in, when it is set to the value of T(56), STAT C is set at P2 + 140 ns clock time by one of the following conditions:

1. 'Sign correct SA(4–7)' signal, with a negative sign detected on 'serial adder in bus A'.
2. 'Save signs' signal, with 'serial adder in bus B' position 0 containing a 1-bit during subtract or compare operations and a 0-bit during all others.
3. 'Set signs' signal during VFL instructions in which 'serial adder in bus A' positions 4–7 contain a positive sign during subtract or compare operations and a negative sign during all others.
4. 'Set signs' signal during any non-VFL operation in which S(0) = 1.

STAT C is normally reset at P1 clock time by the 'STAT trigger reset' signal (activated by either a 'system reset' or an 'I-Fetch reset' signal). STAT C is also reset at P1 + 140 ns whenever the 'set signs' signal is activated for operations other than VFL operations.

STAT D

- Used by microprogram to retain ROS branch information.
- Indicates sign for save-signs operation on floating-point multiply and divide.
- Indicates Q-to-LAL equals 0000.

STAT D stores a characteristic carry from SAL(0) during floating-point multiply and divide operations and indicates that the B1 or B2 field of an instruction equals zero. For operations other than floating-point multiply and divide and scan-in operations, STAT D is available for arbitrary microprogram use and can be unconditionally set or reset by the 'set STAT D' and 'reset STAT D' signals, respectively. (Such operations include storing of the dividend sign on fixed-point operations.) Except during scan-in operations, when it is set directly to the value of T(57), STAT D is normally set at P2 + 140 ns by one of the following conditions:

1. 'Set STAT D' signal.
2. 'Save signs' signal during floating-point multiply and divide operations in which SAL(0) = 1. (STAT D is set at P2 clock time under this condition.)
3. 'Gate Q to LAL 0000' signal.

Note: This signal is activated whenever the B1 or B2 field of an instruction is being gated from Q to LAL and is found to equal 0. Although STAT D is always set on this condition, its significance is of value only during an SS-format instruction when a B2 = 0000 indication must be retained for more than one cycle. (Used in setting ROSAR when selecting I-Fetch ROS words for SS-format instructions.)

STAT D is normally reset at P1 clock time by the 'STAT trigger reset' signal, which is activated by either a 'system reset' or an 'I-Fetch reset' signal. 'Reset STAD' and 'gate I-Fetch invalid address' signals reset STAT D at P0 + 140 ns. The 'reset STAD on decimal overflow' signal resets STAT D at P2 clock time.

STAT E

- Indicates invalid digits and signs.

STAT E primarily indicates the detection of invalid data during decimal operations. Except during scan-in, when it is set to the value of T(58), STAT E is normally set at clock P2 + 140 ns by one of the following conditions:

1. 'Set signs' signal during VFL operations in which an invalid sign is detected on either the 'serial adder in bus-A' or '-B'.
2. 'Save signs' signal for operations other than floating-point operations in which an invalid sign is detected on the 'serial adder in bus B(4–7)'.
3. 'Sign correct SA(4–7)' signal with the detection of an invalid sign on the 'serial adder in bus A(4–7)'.
4. Detection of an invalid digit on either side of the 'serial adder in bus'.
5. Detection of an invalid digit on the 'serial adder in bus A(0–3)', with the 'digit examine' latch set (edit operations).

STAT E is reset at P1 clock time by the 'STAT trigger reset' signal (activated by either a 'system reset' or an 'I-Fetch reset' signal).

STAT F

- STAT F holds:
  'Serial adder in bus B' sign on set-signs decimal operations.
  'Serial adder in bus B' sign on save-signs operations.
  Condition of STAT C.

STAT F primarily indicates the sign of VFL destination operands. Except during scan-in, when it is set to the value of T(59), STAT F is normally set at P2 + 140 ns by one of the following conditions:

1. 'Set signs' signal during a VFL decimal operation, with a negative sign detected on 'serial adder in bus B(4–7)'.
2. 'Save signs' signal during operations other than floating-point operations, with a negative sign detected on 'serial adder in bus B'.
3. 'Save signs' signal during a floating-point operation, with A(0) = 1.
4. 'STAC to STAF' signal, with STAT C set. (Sets STAT F at P2 clock time.)

STAT F is normally reset at P1 clock time by the 'STAT trigger reset' signal (activated by a 'system reset' or an 'I-Fetch reset' signal). The 'STAC to STAF' signal also resets STAT F at P0 clock time in preparation for setting STAT F again at P2.

STAT G

- Used by microprogram to retain ROS branching information.
- Indicates state of T(32).

STAT G is available for arbitrary microprogram use and for indicating the state of T(32). Except during scan-in, when it is set to the value of T(60), STAT G is normally set at P2 + 140 ns by one of the following conditions:

1. 'B(32) to STAB and T(32) to STAG' signal, with T(32) = 1.
2. 'Set STAG' signal.

STAT G is normally reset at P0 + 140 ns by the 'reset STAG' or 'B(32) to STAB and T(32) to STAG' signal. A 'system reset' signal and an 'I-Fetch reset' signal also reset STAT G.

STAT H

- Used for serial adder carry-control functions and ROS branching information.

STAT H indicates a serial-adder carry. Except during scan-in, when it is set to the value of T(38), STAT H is set by one of the following conditions:

1. 'Set STAH' signal and clock time P2 + 140 ns.
2. The output of a latch set at not-clock time by either a 'decimal correct 0–3 and set STAT's AE' or a 'save serial adder carry' signal in conjunction with a serial adder carry from group 2. The set condition is timed at P2 clock time.

STAT H is reset at P1 clock time whenever the latch referred to in item 2 is set. A 'system reset' signal and an 'I-Fetch reset' signal also reset STAT H at P1 clock time. It is also reset at P2 + 140 ns by the microprogram.

CONTROL TRIGGERS

A number of control triggers perform functions similar to STAT's. Table 2-2 lists the most significant triggers, summarizes their functions, and provides ALD and FETOM references.

Table 2-2. Control Triggers

Trigger	Function	ALD Reference	FETOM Reference	Roller Switch Indicator
Right Digit	Selects digit from AB byte on edit operations.	KZ321	Volume 2, Chapter 3, Section 5, "General Data Handling"	RT DIG Roller 4 Position 4 Bit 32
S	Indicates source character, rather than fill character, on edit character transfers.	KZ321	Volume 2, Chapter 3, Section 5, "Introduction to Edit Operation"	S Roller 4 Position 4 Bit 33
Leave	Controls 'serial adder bus B' on edit operations.	KZ201	—	LEAVE Roller 4 Position 4 Bit 34
Step ABC	Increments ABC on edit operations, if 'right digit' trigger is set.	KZ501	—	STEP ABC Roller 4 Position 4 Bit 35
Block I-Fetch	Prevents most I-Fetch functions when interruption or exceptional condition is to be processed.	KD501	Volume 2, Chapter 3, Section 1, "Block I-Fetch Trigger"	BLOCK Roller 4 Position 5 Bit 8
Branch Invalid Address	Indicates branch address of successful branch is invalid.	KD701	Volume 2, Chapter 3, Section 1, "Invalid Address Detection"	BR INVLD ADR Roller 4 Position 5 Bit 16
I-Fetch Invalid Address	Indicates Q has been refilled from invalid address.	KD711	Volume 2, Chapter 3, Section 1, "Invalid Address Detection"	INVLD ADR Roller 4 Position 5 Bit 17
Instruction Length Not Available	Resets ILC in old PSW and resets all interrupt code triggers except 'interrupt code 4' trigger. Set by "late" storage protection check.	KM851	Volume 2, Chapter 3, Section 1, "Fetch Protection Detection"	IL NOT AVAIL Roller 4 Position 5 Bit 18

Table 2-2. Control Triggers (Cont)

Trigger	Function	ALD Reference	FETOM Reference	Roller Switch Indicator
Time Clock at Limit	Indicates timer has been decremented past zero and requests external interruption	KM221	Volume 2, Chapter 3, Section 1, "Timer Exceptional Condition"	TC AT LIMIT Roller 4 Position 5 Bit 19
Timing Gate	Controls duration of I/O control or direct-control signals. Set and reset by microprogram.	KX311	Volume 2, Chapter 3, Section 7, "Write Direct, WRD (84)" and "Read Direct, RDD (85)"	TIME GATE TGR Roller 4 Position 5 Bit 35
No Retry†	Indicates to diagnostic programmer instruction retry may give unpredictable results. Set by (1) 'store per D' signal, (2) 'PAL to IC' signal and not SS format, and (3) 'write local stor' signal and LSWR not selected.	KS321	—	NO RETRY Roller 4 Position 1 Bit 18
IC in LSWR†	Indicates IC is saved in LSWR. Occurs only on SS format operations.	KS321	—	IC IN LSWR Roller 4 Position 1 Bit 19

† 'No retry' and 'IC in LSWR' triggers perform no control function but indicate machine conditions only.