IBM®

Field Engineering
Theory of Operation

2030 Processing Unit
System/360 Model 30

PREFACE

This manual contains information about the IBM 2030 Processing Unit. A companion manual, IBM 2030 I/O Control, Theory of Operation Manual, Form Y24-3362, should be obtained for information pertaining to the attachment of I/O devices to the IBM System/360 Model 30. For maintenance information on the IBM 2030, refer to the IBM 2030 Maintenance Manual, Form Y24-3390. The IBM 2030 Maintenance Diagram Manual, Form Y24-3466, contains flowcharts of specific op-code microprograms for the basic machine and the IBM 1400 Compatibility Feature. The IBM 1620 Compatibility Feature Diagram Manual, Form Y25-3478, contains flowcharts of 1620 emulation.

The following SRL publications contain much useful information about operation and application of the IBM System/360 Model 30:

Title	Form
IBM System/360 Model 30 Functional Characteristics	A24-3231
IBM System/360 Model 30 Configurator	A24-3232
IBM System/360 Model 30 1401 Compatibility Feature	A24-3255
IBM System/360 Model 30 1620 Compatibility Feature	A24-3365
IBM System/360 Model 30 Operators Guide	A24-3373
IBM System/360 Model 30 Channel Characteristics	A24-3411
IBM System/360 Principles of Operation	A22-6821

Fifth Edition, June 1967

This edition, Y24-3360-1, is a minor revision of the previous edition, Y24-3360-0, but does not obsolete it. Minor changes, which are primarily in the IBM 1620 Compatibility Feature section, are marked by a vertical line to the left of the affected text, or by a dot (•) next to the title of an affected figure. In addition, other non-technical, typographical corrections have been made throughout the manual.

Significant changes or additions to the specifications contained in this publication will be reported in subsequent revisions or in FE Supplements.

This manual has been prepared by the IBM Systems Development Division, Product Publications, Dept. 171, PO 6, Endicott, New York. Address comments concerning the manual to this address.

List of Abbreviations

ALU	Arithmetic Logic Unit
ASCII	American Standards Code for Information Interchange
Aux Stor	Auxiliary Storage
BCD	Binary Coded Decimal
bin	binary
CAW	Channel Address Word
CID	Compatibility Initialization Deck
CLD	CAS Logic Diagram
CPU	Central Processing Unit
CROS	Capacitor Read Only Storage
CU	Control Unit
dec	decimal
EBCDI	Extended Binary Coded Decimal Interchange
EOF	End of File
GM	Group Mark
GMWM	Group Mark Word Mark
IC	Instruction Counter
I/O	Input-Output
IPL	Initial Program Loader
L	length code
LS	Local Storage
MPX	Multiplexor
MS	main storage
PSW	Program Status Word
RBC	Read Back Check
ROAR	Read Only Address Register
ROS	Read Only Storage
RPG	Report Program Generator
RR	Register-to-register
RS	Storage-to-register
RX	Storage-to-register
SA	Stack Address
SAL	Sense Amplifier Latch
SI	Storage-immediate
SLD	Simplified Logic Diagram
SRL	System Reference Library
SS	Storage-to-storage
UCW	Unit Control Word
WLR	Wrong Length Record
WM	Word Mark

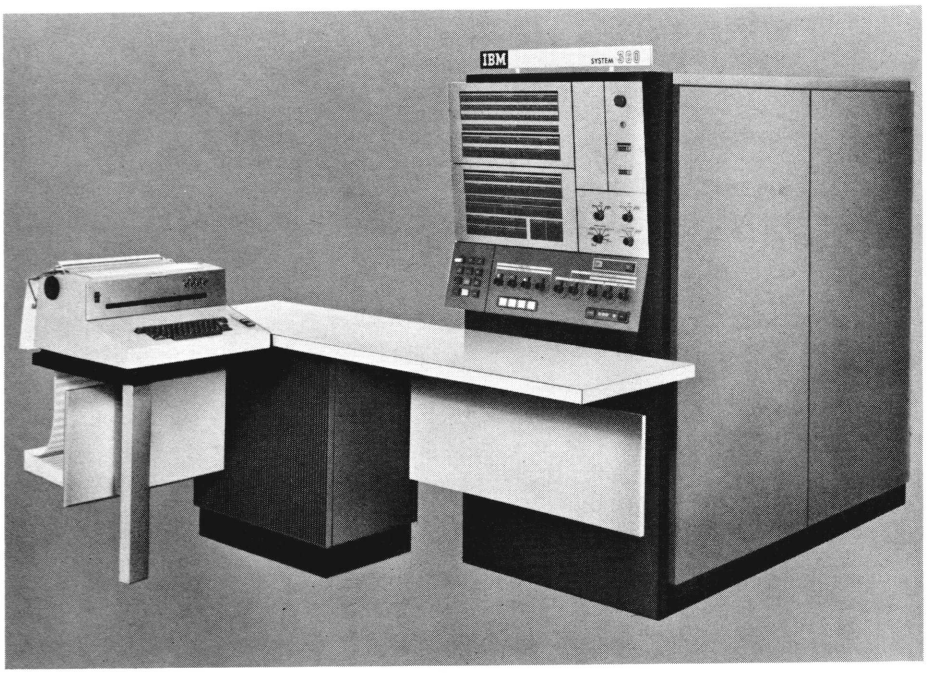


Figure 1-1. IBM System/360 Model 30 with IBM 1052 Documentary Console

CHAPTER 1. INTRODUCTION

The first part of this chapter is an introduction to IBM System/360 Model 30 characteristics. Basic System/360 information (such as data formats and basic programming concepts) is subsequently described in the IBM System/360 General Information section of this chapter. The material in this general information section covers many of the topics included in the Field Engineering System/360 Introductory Programming Student Self-Study Course. (The self-study course is a prerequisite to this publication.) The general information section is primarily for reference and review purposes.

The last part of Chapter 1 is an introduction to certain programming systems concepts with which you should be familiar.

OVERALL DATA FLOW

- Overall control of system operations is provided by control circuitry that interprets instructions and regulates the actions called for by instructions.
- Three basic areas controlled are:
  1. The arithmetic logic unit (ALU)
  2. Core storage
  3. Channels
- A channel is a control and data link between I/O control units and the processing unit.
- An I/O control unit responds to the channel in a standard way over the standard I/O interface cable.

Any data processing system performs three basic operations:

1. Information is entered into the system by use of an input device, such as a card reader.
2. The input information is processed. (Processing includes arithmetical and logical manipulations of source information. The processed information is then usually put into some predetermined format.)
3. The formatted information is sent to an output device, such as a printer or card punch, which then produces a meaningful record of the processed information.

Control of these input, arithmetic, logic, and output functions must be provided. This control is achieved by interaction of two factors:

1. A series of instructions (program) that indicates the operations to be performed.
2. Machine control circuitry that is capable of interpreting and then directing performance of the operations called for by the program.

Because speed is an important factor, each instruction must be obtained quickly by the machine. In System/360 Model 30, the program controlling the system is located in high-speed main storage. (How a program is initially put into main storage is not pertinent to this discussion.)

The control circuitry of the system interprets an instruction fetched from main storage and directs performance of the indicated operation. The next instruction is then obtained from storage and its operation is performed. This sequence is repeated until the job is completed or terminated at an intermediate step.

In the System/360 Model 30, during the processing of any instruction, one (or two, or all three) of three basic areas must be controlled. (Figure 1-2):

1. The ALU (Arithmetic Logic Unit) in

which arithmetical and logical manipulations of information are performed.

2. Core storage, either (or both) main storage or an auxiliary storage (that contains, among other things, areas devoted to general registers, floating point registers, and certain controlling information for I/O operations).

3. Channels, which are the main controlling elements in I/O operations (which, in general, take precedence over non-I/O operations).

Notice in Figure 1-2 that any input/output channel is a link between I/O control units and the control circuitry in the IBM 2030 (the processing unit for System/360 Model 30).

A CU (Control Unit) is necessary for the operation of any I/O device attached to the System/360 Model 30. The CU may be an integral part of an I/O unit or it may be a separate unit to which the I/O device is attached. In either case, the CU has circuitry that allows it to communicate with a channel. The data and control information exchanged between a channel and any CU is in a standard form; therefore, a channel can communicate with any CU as long as the circuitry in the CU is able to operate through the use of the standard signals recognized by the channel. A cable that connects CU's with a channel is called a standard I/O interface cable.

Note, however, one exeption: The 1050 Documentary Console is not attached, on the Model 30, to a standard I/O interface cable.

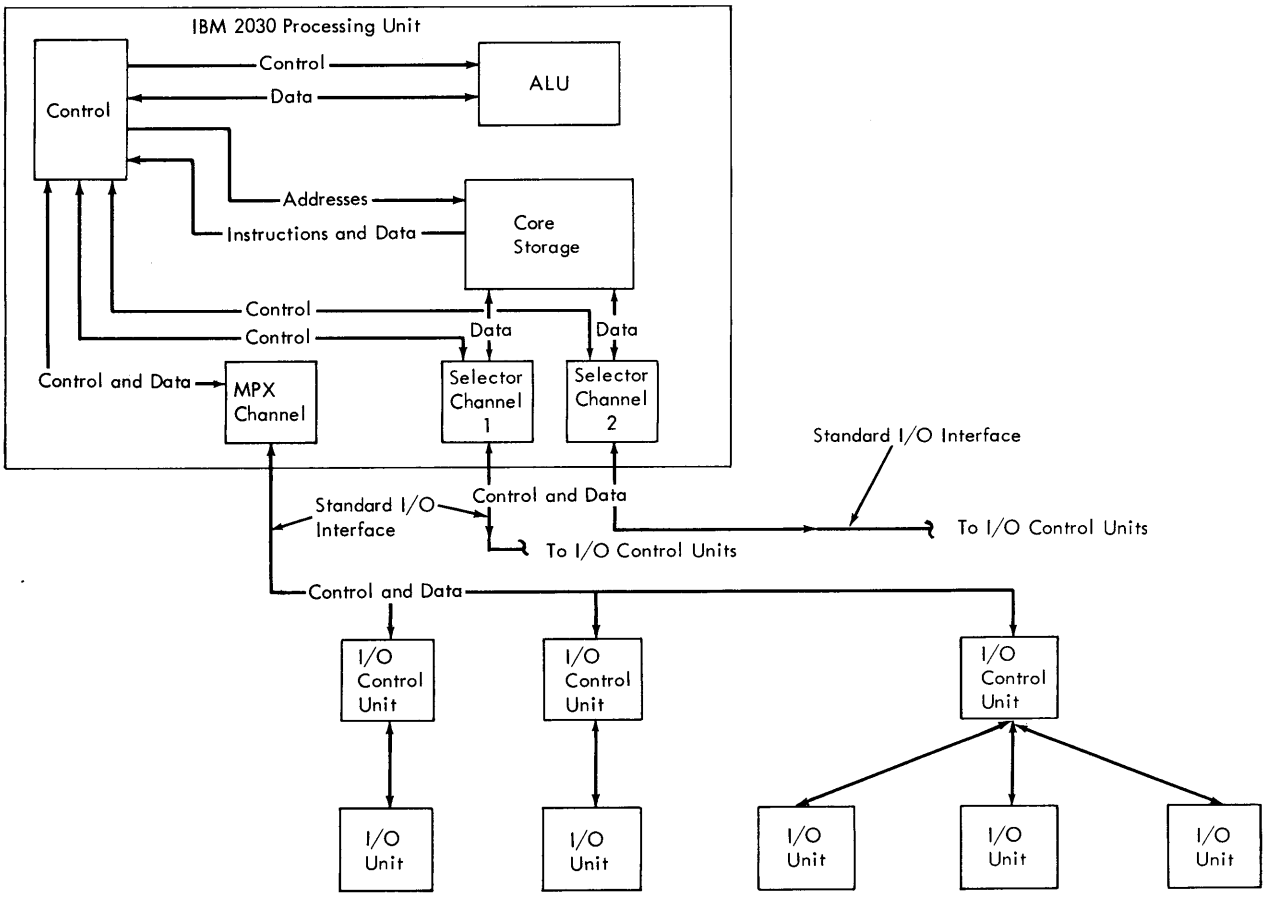


Figure 1-2. System/360 Model 30 Overall Data Flow

ROS (READ-ONLY-STORAGE) CONTROL

- Read-only-storage (ROS) is the basic control circuitry for System/360 Model 30.

Control circuitry is the guiding or regulating medium of the system. There are, however, various levels of control. For example, suppose that a specific byte is to be read out of main storage. To read out the specified byte, the following actions occur:

1. The address of the byte is set into storage address registers.
2. The output of these registers is used to specify the storage location.
3. The byte is read out of storage and placed into a machine register.

In a sense, the outputs of the storage address registers control addressing of storage. However, the storage address registers themselves are controlled by ROS, both when the address is initially set into them, and when it is read out.

In System/360 Model 30, basic controlling circuitry is called ROS (Read-Only-Storage). Outputs of ROS circuitry determine which circuit elements (such as registers) are used and how they are used for each operation. For information about the functions and physical makeup of ROS, refer to Chapter 2 of this publication.

ALU (ARITHMETIC LOGIC UNIT)

- Arithmetic and logical operations are performed on binary and packed decimal data (if decimal feature is used) by the ALU.
- Two registers (B and A) provide input to the ALU.
- Control circuitry (ROS) directs, as indicated by an instruction, both the operation to be performed by ALU and how the data in the B- and A-registers is to be used by ALU.
- Parity is not carried through the ALU circuits. Correct parity is generated for the resulting byte after the information has passed through the ALU.
- Data is sent through the ALU in both true and complemented form, thereby providing a check of ALU operations.

The ALU performs:

1. Arithmetic operations of:
   a. Adding and
   b. Subtracting.
2. Logical operations of:
   a. ANDing,
   b. ORing, and
   c. Exclusive ORing.

The ALU performs binary addition and subtraction (i.e., complement addition) on fixed-point data, two bytes (one from each operand) at a time. If the decimal feature is used, additions and subtractions are performed on packed decimal operands. Here, each byte contains two packed decimal digits; one digit is in the four high-order bits, and the other in the four low-order bits. (The sign is carried in the four low-order bits of the low-order byte.) A packed decimal digit is valid only if the four bits that represent it are in the range 0000 to 1001 (binary).

Two registers (the B- and A-registers) provide the basic information-inputs to ALU. The original information set into these two registers can come from a variety of sources. The sources used depend upon the operation performed.

ROS output:

1. Controls the manner in which the contents of the B- and A-registers are

sent to ALU. (Some ALU operations do not require use of two full bytes of data. For example, comparison against four bits of a mask field requires only two separate four bit entries into ALU.)

2. Specifies the type of operation to be performed (true or complement, binary or decimal, add, AND, OR, exclusive OR) as indicated by the instruction being processed.

Parity is not carried through ALU circuitry. Input line levels are complemented so that input to ALU is in both true and complemented form. Exclusive OR circuitry is used to check that each output line at an up level has a corresponding complemented line at a down level. Correct parity for the result byte is generated after the data has passed through ALU.

STORAGE SIZES AND CYCLE TIMES

- Model 30 uses either a 1.5 or a 2.0 microsecond storage cycle (i.e., read/write cycle).
- Information is handled one byte at a time in System/360 Model 30 core storage.
- Auxiliary storage is made up of local storage and MPX (multiplexor) storage.
- The sixteen general registers and the four floating point registers are in local storage.
- MPX storage contains the multiplexor channel's Unit Control Words.

The IBM 2030 Processing Unit contains core storage and logic, arithmetic, and control circuits for IBM System/360 Model 30. Four models are available; the primary characteristic of each model is its amount of main storage. The letter prefix in the model designation indicates the amount of main storage:

System/360 Model	Main Storage (in bytes)
C30	8192
D30	16384
E30	32768
F30	65536

Each one of the four models has either (but not both) a 1.5- or a 2.0-microsecond storage cycle (such as read from and then immediately write into storage).

Refer to the Data Width column in Figure 1-3 or Figure 1-4. Notice that one byte (eight bits plus parity) at a time is handled in Model 30. This is true for the general and floating point registers as well as for main storage. Handling a word (four bytes) in a general register requires at least 6 microseconds in the 1.5-microsecond storage cycle system (Figure 1-3). What is meant here is that one byte at a time is read from a general register and then written, for example, into a main storage location. Other operations, such as computations, may extend the time, but to read and write in succession requires at least 1.5 microseconds per byte. (Other models of System/360 [Model 40, Model 50, etc.] handle more than one byte per storage access cycle.)

An additional core storage area, called auxiliary storage, is contained in the 2030 (Figure 1-5). Auxiliary storage is a part of the main storage array. However, auxiliary storage is addressed differently and does not use any of the main storage locations. The amount of available auxiliary storage is, in general, dependent upon the size of the main storage array. Standard auxiliary storage capacity for each model is:

Model	Auxiliary Storage (bytes)
C30	512
D30	1024
E30	1024
F30	1024

Auxiliary storage is made up of two areas:

Local storage, and
MPX (Multiplexor) storage.

Local storage contains the sixteen general registers, the four floating point registers, and other miscellaneous areas. Every Model 30 has 256 bytes of local storage.

Characteristics	Speed (in microseconds)	Data Width Bits (Bytes)
Basic Machine Cycle	.75	—
Main Storage: Model C30 -- 8192 Bytes Model D30 -- 16384 Bytes Model E30 -- 32768 Bytes Model F30 -- 65536 Bytes	1.5	8 (1)
Registers Accessible to Programmer:		
Sixteen General Registers *	6	32 (4)
Four Floating-Point Registers *	12	64 (8) Double Precision
	6	32 (4) Single Precision
System Control: Read Only Storage (ROS)	.75	—

* These registers are in local storage (a storage area that is in addition to the main storage capacity).

Figure 1-3. CPU Characteristics

The remainder of auxiliary storage is used to contain multiplexor channel UCW's (Unit Control Words). Each one of these UCW's contains the information necessary to control the I/O unit, on the multiplexor channel, to which the UCW pertains. There are 32 UCW's, each eight bytes long, in the MPX storage of Model C30. Hence, a maximum of 32 subchannels can be controlled from information in MPX storage in Model C30. Models D30, E30, and F30 can use up to 96 UCW's in MPX storage as a standard feature. Models E30 and F30, however, can have the Additional Multiplexor Subchannels optional feature that allows use of up to 224 subchannels. (If this feature is installed, 2048 bytes are used for auxiliary storage.) If the 1400 or 1620 compatibility feature is installed, certain parts of MPX storage are used for purposes other than storing UCW's.

Characteristics	Speed (in microseconds)	Data Width Bits (Bytes)
Basic Machine Cycle	1	—
Main Storage: Model C30 -- 8192 Bytes Model D30 -- 16384 Bytes Model E30 -- 32768 Bytes Model F30 -- 65536 Bytes	2	8 (1)
Registers Accessible to Programmer		
Sixteen General Registers *	8	32 (4)
Four Floating-Point Registers *	16	64 (8) Double Precision
	8	32 (4) Single Precision
System Control: Read Only Storage (ROS)	1	—

* These registers are in local storage (a storage area that is in addition to the main storage capacity).

Figure 1-4. CPU Characteristics (2.0 Microsecond Read/Write Cycle)

Core Storage Array

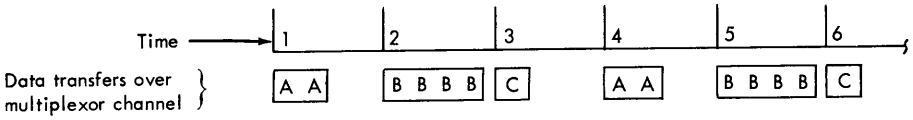
Main Storage	
Model	Capacity (in Bytes)
C30	8,192
D30	16,384
E30	32,768
F30	65,536

Auxiliary Storage		
Local Storage	MPX Storage	
256 Bytes in Every System/360 Model 30 (Contains 16 general purpose registers, 4 floating point registers, and other miscellaneous areas.)	Model	Number of UCWs
	C30	32
	D30	96
	E30	96*
	F30	96*
	* Model E30 or F30 can have 224 UCWs if the Additional Multiplexor Subchannels Optional feature is installed	

Figure 1-5. Core Storage Allocations

CHANNELS

- Up to three channels can be installed in System/360 Model 30:
  1. One Multiplexor Channel
  2. Selector Channel 1
  3. Selector Channel 2
- The multiplexor channel can operate in either multiplex or burst mode; a selector channel operates only in burst mode.



Key:
A Data byte from device A to main storage.

B Data bytes to device B from main storage.

C Data byte from device C to main storage

Figure 1-6. Multiplex Mode Operation

The 2030 can have up to three channels:

1. A multiplexor channel (standard feature)
2. Selector Channel 1 (special feature)
3. Selector Channel 2 (special feature)

The main purpose of the multiplexor channel is to provide for operation of lower speed I/O devices in multiplex (data interleaved) mode (Figure 1-6). In the multiplex mode, information is transferred in groups of bytes between the processing unit and several I/O devices concurrently. For example, multiplexing service for two serial unbuffered card readers could proceed as follows:

1. One byte of data is sent from the control unit of the first card reader to the processing unit.
2. Next, one byte of data is sent from the control unit of the second card reader to the processing unit.

Steps 1 and 2 are repeated until a complete record is transferred for one of the units. Servicing for the other unit is then completed alone.

While some I/O units always operate in burst mode regardless of the channel, buffered units (except the 2520) attached to the multiplexor channel can operate in burst mode as well as in multiplex mode. This capability is provided by a switch associated with the buffered unit. In burst mode (Figure 1-7), the data transfer is completed on a record basis.

Multiplexing operations are not allowed on the multiplexor channel during the time that a unit attached to the multiplexor channel is operating in burst mode. Therefore, a burst mode unit should not be started (on the multiplexor channel) while units that are multiplexed are operating.

Selector channels operate only in burst mode. An I/O control unit obtains control of the channel and transfers an entire record (i.e., multiplexing does not occur) for the associated I/O unit. After the record is transferred (and if no chaining for the same unit occurs), another I/O control unit can obtain control of the selector channel for record transfer.

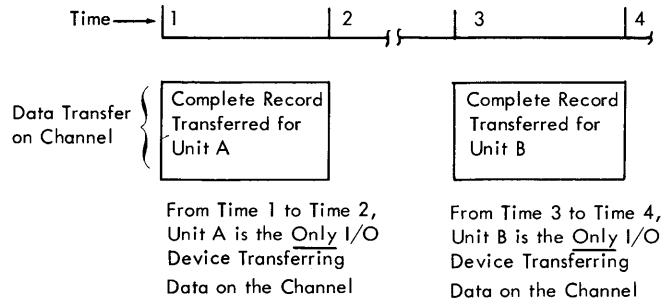


Figure 1-7. Burst Mode Operation

Multiplexor Channel

- Some of the CPU circuits are used by the multiplexor channel for its operations.
- Certain information used in CPU instruction processing is stored in local storage during multiplexor channel operations.
- UCW information is used to indicate how an I/O unit's operation is controlled on the multiplexor channel.
- The maximum number of I/O units that can be addressed on the multiplexor channel is dependent upon:
  1. The amount of MPX storage available.
  2. The number of shared subchannels used.
  3. The fact that a maximum of eight CU's can be attached to the standard I/O interface cable.

The terms "concurrent", "simultaneous", and "multiplex" are used consistently in System/360 publications. What, however, is meant by these terms when applied to System/360 Model 30 multiplexor channel operations? Consider a typical multiplexor channel data transfer.

In the 2030, certain CPU circuits are shared with the multiplexor channel. Therefore, CPU instruction processing operations are stopped during the time that a multiplexor channel operation (data transfer or chaining) is in progress. Assume that an add operation is being executed in the CPU and that a 1442 card read operation is in progress. CPU control circuitry, including certain registers (not the 16 general or 4 floating point registers), contains information that is updated as the add instruction is executed. Now suppose that the I/O CU (Control Unit) of the 1442 requests channel service (i.e., the CU has a data byte ready for transfer to storage). The CU can wait to transfer the data byte for only a certain time period. This time is dependent upon when the next card column is read. If the first byte is not transferred before the next byte is ready, data is temporarily lost. (To recover the data, the operator must reload the 1442 with the appropriate cards.)

In the CPU, the information needed for execution of the add instruction is taken out of CPU registers and placed in local storage. The UCW (Unit Control Word) that pertains to the 1442 is then taken out of MPX storage and placed in the appropriate CPU registers. The UCW is used to indicate how the byte from the 1442 should be handled (such as where it should be stored in main storage). As soon as the byte is processed and the UCW contents are updated, the UCW is stored into MPX storage. The CPU registers are loaded from local storage with the necessary add instruction information, and the add operation is continued. The next request for service by the 1442 results in repetition of the operations just described.

CPU information is not always restored into CPU registers after a multiplexor channel data byte transfer. If another request for I/O data transfer is made soon enough, then that data is processed. This operation can occur, for example, when a high-speed device (such as a magnetic tape unit) is run on the multiplexor channel.

The maximum number of I/O units that can be attached to the multiplexor channel depends upon:

1. The number of available UCW's in MPX storage.
2. The number (if any) of shared subchannels used.
3. The restriction that a maximum of eight adapters (CU's) can be connected to the standard I/O interface cable.

Item 1 depends upon the model. Up to 32 subchannels (UCW's) can be used in Model C30; up to 96 in Models D30, E30, and F30 as a standard feature. Models E30 and F30 can have the Additional Multiplexor Subchannels optional feature that provides for use of up to 224 subchannels. In this last case, shared subchannels are not allowed.

A shared subchannel is used for multiple I/O units controlled, one at a time (i.e., no multiplexing between the sharing units), by a single CU. An example of this type of configuration is several direct access mechanisms (such as 2311 disk storage drives) connected to one CU. Only one UCW is used to store controlling information for operation of one of the direct access mechanisms at a time. Use of certain unshared UCW's (i.e., a UCW devoted to only one I/O unit) is excluded if shared subchannel addresses are used. For further information on multiplexor channel addressing, refer to Field Engineering Manual, IBM 2030 I/O Control, System/360, Model 30, Form Y24-3362.

Selector Channels

- Data transfer for an I/O unit is completed on a record basis before another I/O unit can be started on the same selector channel.
- A selector channel uses its own circuitry (including clock) to effect data transfers between its attached I/O units and main storage.
- Selector channels use CPU circuitry during starting, chaining, and ending procedures. If any overlapping CPU instruction processing is also taking place, the CPU instruction information is stored in local storage until completion of the selector channel operation.
- Up to 256 I/O addresses can be used to address units on a selector channel. The actual number of I/O units will probably be less than the maximum, however, because only eight adapters (I/O control units) can be attached to a standard I/O interface cable.

Either of the two selector channels available for Model 30 operates in burst mode only. Only one device at a time can be actively engaged in a data transfer on a specific selector channel.

Each selector channel has its own circuitry for use in data transfers. When a selector channel data transfer occurs, CPU instruction processing is stopped only for the time necessary to transfer the data byte between main storage and the channel. Because only one device at a time can be operated, a separate area for storage of selector channel UCW's is unnecessary. That is, for each selector channel there is only one current UCW, which is handled by selector channel circuitry. Operation indicated by the current UCW is completed before another I/O operation can be started on the same selector channel. Hence, current CPU instruction information is not stored into local storage (as it is during multiplexor channel data transfers).

During transfer of a byte of data between main storage and a selector channel the CPU clock is not used. Rather, each selector channel has its own clock to control operation of storage. After the data transfer is completed, the CPU clock is used for processing the CPU instruction in progress.

Note that in selector channel starting, chaining, and ending operations, CPU control circuitry is used. For chaining and ending operations, information related to the CPU instruction in progress is placed in local storage.

CPU instruction-processing information is restored into machine registers from local storage at completion of the chaining or ending procedure. Instruction processing is then continued in overlap fashion with any selector channel data transfers that occur. (Note that during starting of an I/O operation, the CPU instruction in progress is an I/O instruction such as START I/O.)

The eight-bit unit addressing scheme allows for up to 256 separate I/O addresses on a selector channel. However, because only eight CU's (also called adapters) can be attached to a standard I/O interface cable, the actual number of I/O units will probably be less than the maximum addressable number.

CPU DATA FLOW

- The series of logical steps used to control information flow between machine elements (such as ALU, machine registers, and storage) for a particular operation, is called a ROS-microprogram routine.
- The ROS microprogram is not written by the user or problem programmer; its routines are established in circuitry, and ROS micro instructions are not stored in core storage.

Recall that ROS (Read-Only-Storage) is the basic control circuitry in the 2030. The particular series of ROS steps taken to control an operation is a microprogram routine. The ROS-microprogram routine for any specific machine operation is, in general, dependent upon:

1. The requirements of the operation (i.e., what machine elements must be used to achieve the desired results).
2. The logical methods used by the microprogrammer.

It is important to realize that the microprogram is part of the machine circuits and has nothing to do with the writing of problem or control programs. ROS microinstructions are not stored in core storage. A description of ROS is provided in Chapter 2 of this publication.

The function of a particular machine register, as used in a specific operation, is dependent, to some degree, on how the microprogram for that operation is written. Hence, in this chapter, subsequent introductory descriptions of machine registers and their general functions do not necessarily apply to all operations. Rather, the most usual functions are described.

Many times, reference is made to a bit position in a register. Most registers can hold one byte (eight information bits plus one parity bit) of data. Reference to a bit position within a register is done by prefixing the bit position with the letter-name of the register. For example, the high-order bit in the R register is referenced by R0.

BUSSES

- Busses are circuits that provide for transfer of information between various machine elements.

Busses provide the information-paths between machine elements such as registers, ALU, and core storage. In many operations, microprogram steps call for transfer of information from a register to a bus and from there to another register. For example, an address byte can be incremented by 1, by:

1. Gating the original address byte out of a register to a bus,
2. Sending the address byte through ALU while adding 1 to it as it passes through ALU,
3. Sending the result byte from ALU to another bus, and

4. Sending the result byte back into the original register.

Most busses in the 2030 handle 8 information bits plus 1 parity bit (one byte). Some busses handle less than a byte. The need for busses of differing capacities will become more evident when you study detailed machine circuitry.

MACHINE REGISTERS

- The M and N registers are set with information used to address core storage locations.
- The R register (storage data register), in general, is:
  1. The immediate source register for a byte to be stored into a core storage location.
  2. The immediate destination register for a byte read out of core storage.
- In general:
  1. Information used to address instructions is sent from the I and J registers to the M and N registers.
  2. Information used to address data is sent from the U and V registers to the M and N registers.
  3. Information used to address certain auxiliary storage locations is sent from the T register to the N register.
- The G register usually contains the operation code.

Any position of core storage can be located, for reading or writing purposes, by use of address information placed in the M and N registers. Each of these registers can contain one byte. Hence, a maximum address of 65,535 (decimal) can be represented by the 16 bits in the M and N registers. (That is, the maximum number represented by 16 binary digits is $2^{16}-1$.) This arrangement provides for addressing from 0000 to FFFF, or a total of 65,536 (decimal) storage locations.

In the 2030, the M and N registers always have the capacity to hold the bit structure that represents the address FFFF (i.e., for Model 30F). However, any address reference outside of the actual range of main storage positions available (Model 30C has 8192 bytes; Model 30D has 16384 bytes; Model 30E has 32768 bytes) may cause an addressing exception (a program interruption).

Other circuits are used in conjunction with the output of the M and N registers when auxiliary storage (local or one of the MPX storages) is addressed. The actual address is specified in the M and N registers while these other circuits determine to which storage area (main, local, or a MPX) the address applies. The 8 high-order bits of the address are set into the M register, while the 8 low-order bits are set into the N register. The M and N storage address registers are frequently referred to as one register (viz., the MN-register).

Addressing compatibility is maintained with other System/360 Models because of the base-displacement addressing method used. Recall that addresses are derived from a 12-bit displacement plus a 24-bit base. For example, a program segment that might be written for a System/360, Model I 65 might use storage addresses in the 500,000 to 504,000 (decimal) range. Here, the base register used could have 500,000 as the base address. Displacement values could then run from 0 to 4,000 (decimal). To run this program segment on a Model D30, the base register could be loaded with the address 0 (decimal), and the displacement values left unchanged. Because displacement values cannot exceed FFF (i.e., 12 bit positions are used for displacement) any Model 30 can handle any displacement value in its M and N registers. However, any base-plus-displacement value that exceeds the storage capacity of the System/360 model used may, as previously mentioned, cause an addressing exception.

In general, data written into (or read from) storage passes through the R register. (One notable exception is that when data is transferred between a selector

channel and main storage, it does not pass through the R register. Refer to the Field Engineering Manual of Instruction, IBM 2030 I/O Control, Form Y24-3362.)

Notice in Figure 1-8 that the MN-bus (really the M and N busses--eight information bits plus one parity bit for each bus) provides input paths to the MN-registers. The following table summarizes the address information source inputs to the MN-registers:

Source Register	Destination	Usual Immediate Source of
I	M	High order address bits for an instruction byte
J	N	Low order address bits for an instruction byte
U	M	High order address bits for a data byte
V	N	Low order address bits for a data byte
T	N	Address bits for certain auxiliary storage locations

Addresses are frequently obtained from instructions which are in main storage. Hence, there must be a path, during normal instruction processing, over which these addresses can be set into the U, V, I, J, or T registers. One path is from storage, to the R register, through the A-register inputs to ALU, through ALU to the Z bus, and from there to the appropriate register (Figure 1-8). This description is not meant to imply that every time a byte is sent from storage it follows the path just described into all registers. The microprogram specifies which registers are to take part in the operation, and, as already pointed out, the microprogram steps used depend upon the operation being performed.

During instruction processing, the G-register usually contains the instruction operation code. Hence, the values of the bit positions of this register indicate such items as instruction length and format.

Many other registers are used. However, how a register is used is mainly dependent upon the operation performed. The following table summarizes the usual functions of some important registers in the data flow (Figure 1-8):

Register	Usual Function
I	Instruction address (high-order bits)
J	Instruction address (low-order bits)
U	Data address (high-order bits)
V	Data address (low-order bits)
L	Data length
T	Auxiliary storage address
D	General purpose data register
R	Storage data register
S	Status (CPU)
G	Instruction operation code
H	Priority status register
Q	Storage-Protection key in PSW (High 4 bits); Storage-Protection key of block of storage just used (low 4-bits)
C	Interval Timer Count
F	External Interrupt: Interval Six direct-control interruptions (bits 2 through 7).

The W and X registers hold information that is used to address ROS. A maximum of 13 bits are needed to address any ROS word. The W register holds the 5 high-order bits and the X register holds the 8 low-order bits. (Note that the W register has only five bit positions which are W3, W4, W5, W6, and W7.) In addition, each of these registers has a parity bit position.

The FW-FX and GW-GX registers are backup registers for ROS addresses. The FW-FX registers are used to retain the ROS address just held by the WX registers when certain multiplexor channel operations break into CPU instruction processing. The GW-GX registers provide backup for addresses in WX when selector channel one requires use of ROS (such as in chaining operations). A similar set of registers (HW-HX) is used during ROS operations for selector channel two. For detailed information about the WX registers, refer to Chapter 2 of this publication. Multiplexor and selector channel operations and register usage) are described in Field Engineering Manual of Instruction, IBM System/360 Model 30, 2030 I/O Control, Form Y24-3362.

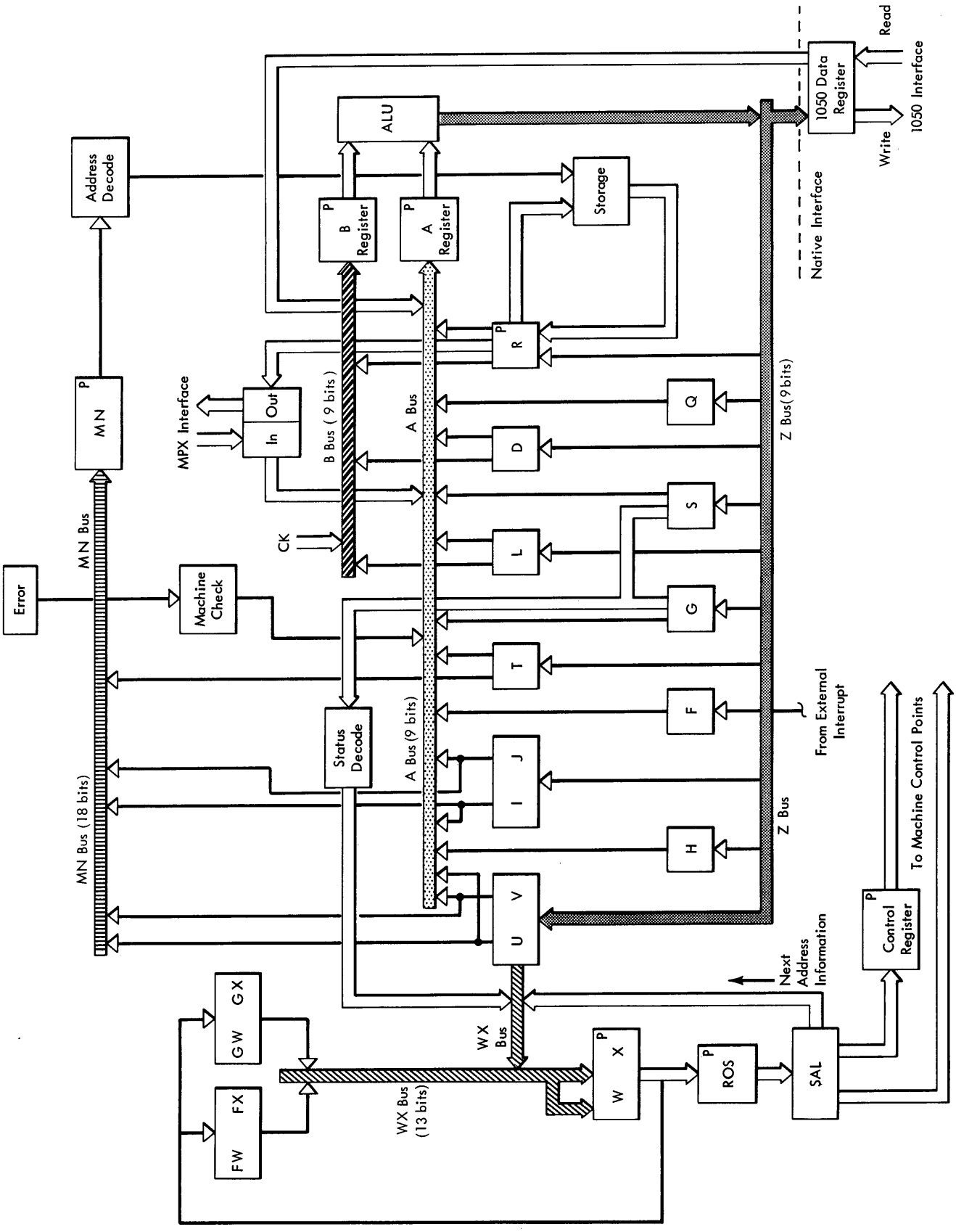


Figure 1-8. 2030 Basic Data Flow

ARITHMETIC OPERATIONS

- The B-register input to ALU is complemented in certain arithmetic operations. In some packed decimal operations the B-register input to ALU is incremented by 6.
- Each second operand (source) byte is sent to the B-register during arithmetic operations.

Certain arithmetic operations require complementation of second (source) operand bytes. Some packed decimal operations require addition of 6 to each second operand byte. These two functions are handled by circuits that affect the outputs of the B-register (but not the A-register outputs). Therefore, a source byte is set into the B-register input to ALU and the destination (first operand) byte is set into the A-register input to ALU.

In the following descriptions, actual circuit functions (including ROS controls) are not presented. Rather, the general arithmetic procedures used by ALU are presented. If you need to review binary or hexadecimal numbering systems in order to understand the following descriptions of arithmetic operations, refer to the Numbering Systems section in Chapter 1 of this publication.

FIXED POINT ARITHMETIC

- In fixed-point numeric operands, all bit positions to the left of the high-order significant digit have the same value as the sign bit.
- The maximum positive number that can be contained in a binary field of n digits is equal to 2^n-1.
- The maximum negative number (in two's-complement form) that can be contained in a binary field of n digits is equal to 2^n.

Recall that fixed-point binary operands are stored as half words or words with the sign indicated in the high-order bit position. When the high-order position has a value of 0, the binary number is positive. A negative number has a sign bit at a value of 1. The remainder of the halfword or word is used to designate the magnitude of the number. However, all bit positions between the leftmost significant digit and the sign bit have the same value as the sign bit. For example, either of the following are positive numbers:

```
+S
 0 000 0000 0000 1000

+S
 0 111 1111 1111 1000
```

But, both of the following are negative numbers:

```
-S
 1 000 0000 0000 0000

-S
 1 111 1111 1111 1011
```

When all of a given number of binary digits are 1, the largest positive quantity that can be represented by that number of digits is given. For example, the maximum positive number represented by two binary digits is 11 (decimal 3). The maximum positive quantity represented by a binary field can be expressed in decimal notation by:

1. Counting the number of binary digit positions in the field.
2. Raising 2 to a power equal to the count determined in step 1.
3. Subtracting 1 from the product obtained in step 2.

Hence, the maximum positive quantity represented by four binary bits is 2^4-1 (15 in decimal).

Because one of the sixteen bit positions in a fixed-point half word is used for the sign, fifteen bit positions can be used for the integer. Therefore, the maximum positive quantity that can be represented in a

fixed-point binary half word is $2^{15}-1$ (32,767 in decimal).

In fixed point operations, negative numbers are carried in two's-complement form. For example, the true binary form of the decimal value +26 is changed to a negative quantity by complementing it:

Decimal Value	Sign
+26	0 000 0000 0001 1010
	1 111 1111 1110 0101
	+1
-26	1 111 1111 1110 0110

Notice that the two's-complement of 11010 (+26) is 00110. The remainder of the bit positions are at a value of 1 to indicate a negative quantity.

The maximum negative number that can be represented in a half word is:

S
1 000 0000 0000 0000

This is the complement of 1 0000000 0000 0000 which should represent a positive quantity. However, the convention is that the high-order bit of a half word is 0 when the quantity is positive. To show 1 0000000 0000 0000 as positive would require an extra high-order position at a 0 value. But this is impossible because only 16 positions are provided in a half word. Hence, in a half word, the absolute value of the largest negative number is one greater than the absolute value of the largest positive number. This concept also applies to quantities represented in a word. A summary of the magnitude of binary numbers that can be represented in a word is shown in Figure 1-9.

Number	Decimal	S	Integer
$2^{31}-1$	= 2 147 483 647	=0	1111111 11111111 11111111 11111111
2^{16}	= 65 536	=0	0000000 00000001 00000000 00000000
2^{0}	= 1	=0	0000000 00000000 00000000 00000001
0	= 0	=0	0000000 00000000 00000000 00000000
-2^{0}	= -1	=1	1111111 11111111 11111111 11111111
-2^{1}	= -2	=1	1111111 11111111 11111111 11111110
-2^{16}	= -65 536	=1	1111111 11111111 00000000 00000000
$-2^{31}+1$	= -2 147 483 647	=1	0000000 00000000 00000000 00000001
-2^{31}	= -2 147 483 648	=1	0000000 00000000 00000000 00000000

Figure 1-9. Fixed Point Numbers

Fixed Point Addition

- An overflow is indicated when the carry-conditions, out of the high order digit position and out of the sign position, do not agree.

An overflow occurs when two numbers are arithmetically manipulated into an area, such as a half word, that is not large enough to contain the result. In fixed-point operations, an overflow condition is indicated when the carry out of the high-order digit position and the carry-out of the sign position do not agree.

The following addition examples illustrate fixed point binary addition. Only eight bit positions are used; the high-order bit is the sign. Carry conditions and any consequent overflow results are summarized for each example:

S
1. +57 = 00111001
<u>+35</u> = <u>00100011</u>
92 = 01011100 (true form)

a. No carry out of high order digit position.

b. No carry out of sign position.

c. Carries agree; therefore, no overflow.

S
2. +57 = 00111001
<u>-35</u> = <u>11011101</u>
+22 = 00010110 (true form)

a. Carry out of high order digit position.

b. Carry out of sign position.

c. Carries agree; therefore, no overflow.

```
          S
3.  +35 = 00100011
    -57 = 11000111
    -22 = 11101010 (complement form)
```

a. No carry out of high order digit position.

b. No carry out of sign position.

c. Carries agree; therefore, no overflow.

```
          S
4.  -57 = 11000111
    -35 = 11011101
    -92 = 10100100 (complement form)
```

a. Carry out of high order digit position.

b. Carry out of sign position.

c. Carries agree; therefore, no overflow.

```
          S
5.  -57 = 11000111
    -92 = 10100100
   -149 = 01101011
```

a. No carry out of high order digit position.

b. Carry out of sign position.

c. Carries do not agree; therefore, overflow.

```
          S
6.  +57 = 00111001
    +92 = 01011100
   +149 = 10010101
```

a. Carry out of high order digit position.

b. No carry out of sign position.

c. Carries do not agree; therefore, overflow.

Fixed Point Subtraction

- The two's-complement of the second operand is added to the first operand in fixed-point subtract operations.
- An overflow occurs when the carry conditions out of the high-order digit position and out of the sign position do not agree.

Fixed-point subtraction is done by adding the two's-complement of the second operand to the first operand.

An example is subtraction of +456 from +678. +456, the second operand, is complemented and added to +678, the first operand.

```
      +678  001010100110 (first operand)
(-)   +456  111000111000 (2nd operand comp.)
      +222  000011011110
```

Overflow occurs when the carry out of the high order digit position does not agree with the carry out of the sign position. For example:

```
                S
       32,766   1 000 0000 0000 0001
(-)  +     20   1 111 1111 1111 1100 (Comp)
     -32,786   0 111 1111 1110 1101
```

a. No carry out of high order digit position.

b. Carry out of sign position.

c. Carries do not agree; therefore, overflow.

In any fixed-point binary subtract operation, each second operand byte, as it is operated on by ALU, is effectively complemented. Recall that data is sent through ALU in complemented and true form. In the binary subtract operation, the complement lines of the second operand byte are added to the true lines of the first operand byte. The complement lines are really the one's-complement of the second operand byte. To obtain the correct result, a one is forced (by control circuitry) into the low-order bit position (bit7) of ALU when the low-order bytes are added. In this process, then, the inversion plus the one in the low-order position effectively results in addition of the two's-complement of the second operand byte to the first operand byte. For example:

1. Operation: fixed-point binary subtract.
2. First operand: 00000001
3. Second operand: 11111111

4. Action in ALU:

```
First operand in true form = 0 000 0001
Second operand inverted =    0 000 0000
Forced carry =                         1
Result =                     0 000 0010
```

The forced carry is automatic only for the low order byte addition. Consider, for example, subtraction of -1 from +496. (Half word operands are used.)

1. First operand = 0 000 0001 1111 0000

2. Second operand = 1 111 1111 1111 1111

3. Operation on low order bytes:

```
First operand in true form =  1111 0000
Second operand inverted =     0000 0000
Forced carry =                         1
Result low-order byte =       1111 0001
```

4. Operation on high order bytes:

```
1st operand byte =            0000 0001
2nd oper. byte inverted =     0000 0000
No forced carry.
Result high-order byte =      0000 0001
```

5. Resulting half word:

0 000 0001 1111 0001 = +497 (decimal)

6. Equivalent operation in decimal notation:

+496 - (-1) = +497

If a carry occurs out of the high order bit position of a result byte, then that carry is added to the low order position of the next two-byte addition.

PACKED DECIMAL ARITHMETIC

- In packed decimal add or subtract:

  1. An even number of minus signs indicates a true add.

  2. An odd number of minus signs indicates a complement add.

Recall that the sign of a packed decimal field is in the four low-order bits of the low-order byte. Sign analysis must be made before any adding or subtracting is started. The result of the sign analysis indicates whether or not the second operand bytes are to be complemented. When the system is using the EBCDI code, the sign bit-combinations are:

Bit Combination	Sign Represented
1100	+
1111	+
1101	-

Three conditions are analyzed to determine how the operation is to proceed:

1. The operation: add (+) or subtract (-)

2. The sign of the first operand: + or -

3. The sign of the second operand: + or -

An even number of minus signs specifies a true-add operation, while an odd number of minus signs specifies a complement-add operation. The eight possible combinations are:

Operation	Sign of First Operand	Sign of Second Operand	True or Complement Add
add (+)	+	+	true
add (+)	-	+	complement
add (+)	-	-	true
add (+)	+	-	complement
subtract (-)	+	+	complement
subtract (-)	-	+	true
subtract (-)	-	-	complement
subtract (-)	+	-	true

Packed Decimal True Addition

- Decimal corrector circuits are used to prevent a four bit binary sum from representing a hexadecimal digit rather than the desired decimal digit.

Values of four binary bits in the range 0000 to 1001 can represent decimal digits in the range 0 to 9. Addition of two four-bit binary numbers results in a total that represents a decimal digit as long as the total does not exceed 9. If the total exceeds 9, then the result is outside the range of single decimal symbols representable by the four binary bits. For example:

0001 + 1000 = 1001

(1 + 8 = 9).

But:

0010 + 1000 = 1010

(2 + 8 = 10)

In the last addition, the resulting four binary bits represent two decimal digits. Four bits in packed decimal fields, however, must represent only the single decimal digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, or 9. Decimal corrector circuits are used to prevent two four-bit groups from giving a result outside of the range of a single decimal symbol.

After sign analysis, packed-decimal true add proceeds as follows:

1. Six (0110 in binary) is added to each four bit digit group of the second operand byte.
2. The entire first operand byte (or only four high-order bits for low order byte) is added to the step 1 sum. Any carry out of a four-bit total is noted.
3. If, in step 2, a carry occurred out of the high-order bit of a four-bit sum, add 0000 to that sum. If such a carry did not occur, add the complement of 0110 (i.e., 1010) to that sum.

For example, ignoring sign analysis, add+18 (first operand) to +16 (second operand):

1. Add 6 (0110) to each four bit group of the second operand:

Second operand (16) =	0001	0110
	0110	0110
	0111	1100

It is of some interest to note that this addition has resulted in conversion from ten's complement notation to sixteen's complement notation. That is, 1 (0001) is the ten's complement-of-9 (1001). Addition of 6 (0110) to the ten's complement-of-9 produces the sixteen's complement-of-9 which is 7 (0111). Also, 6 (0110) added to the ten's complement-of-4 (which is 6 or 0110) produces the sixteen's complement-of-4 (which is 12 or 1100).

2. Add the first operand byte to the sum obtained in step 1. (Any carry out of the high bit position of the four low bits is carried into the low order bit of the four high bits.)

Sum from step 1 =	0111	1100
First operand (18) =	0001	1000
	1001	0100

If this addition results in a carry out of a four-bit group, then the maximum hexadecimal digit (F) has been exceeded. But the first ten hexadecimal symbols (0 through 9) equate directly to the corresponding decimal symbols. Therefore, if a carry-out does occur, the digit represented by that four-bit group must be in the range 0000 to 1001 (0 to 9). On the other hand, if no carry-out occurs, then the four-bit group does not represent the desired decimal digit, and the 6 originally added into that group must now be subtracted.

3. Add 0000 to a four bit group if a carry out of its high order position occurred. Add 1010 (complement of 6) to a four bit group if a carry did not occur out of its high order position.

1001	0100	
1010	0000	
0011	0100	or +16+18=+34

(Note that the carry out of the high order position is not used as part of the total.)

As a second example, consider addition of two packed decimal fields:

1. Second operand (138+): 0001 0011 1000 1100
   First operand (117+): 0001 0001 0111 1100

2. The low order bytes are sent to ALU. A true add is indicated because the number (zero) of minus signs is even.

3. The four high order bits of each low order byte are added:

   a. First, six is added to the second operand:

```
  1000
 +0110
  1110
```

   b. Then the first operand is added to the sum obtained in step 3a:

```
  1110
 +0111
  0101
```

   The carry is retained for use in addition of the next two four bit groups.

   c. Because a carry occurred out of the high order position, 0000 is added to the result of step 3b:

```
  0101
 +0000
  0101 + = (5 decimal)
```

   0101 represents the result low order decimal digit. 0101 and the sign 1100 are stored in the low order byte position of the destination field (first operand location).

4. The next two bytes (one from the first operand, one from the second operand) are sent to ALU.

5. The two bytes are added:

   a. Six is added to each four bit group of the second operand byte:

```
  0001   0011
 +0110  +0110
  0111   1001
```

   b. The first operand and the carry from step 3b are added to the sum obtained in step 5a:

```
Carry from step 3b =          1
Sum from step 5a =  0111   1001
First operand =     0001   0001
                    1000   1011
```

   c. Because a carry did not occur out of either four bit group, 1010 is added to each group:

```
 1000  1011
 1010  1010
 0010  0101
```

   Notice that the high bit carries of each four bit group are not used in the total. The result is placed in the first operand location. Summarizing this operation:

	First Operand	Second Operand
Before operation	0001 0011 1000 1100	0001 0001 0111 1100
After operation	0010 0101 0101 1100	0001 0001 0111 1100

(In decimal notation: + 138 + 117 = +255.)

Packed Decimal Complement Addition

- In complement add, the complement value of the second operand byte is added to the true value of the first operand byte.

Packed decimal complement addition employs decimal correction circuits in a different way than true add. After sign analysis, packed decimal complement add proceeds as follows:

1. The portion of the second operand being operated on is complemented.
2. The portion of the first operand being operated on is added to the result of step 1. Carry conditions out of each participating four-bit group are noted.
3. If, in step 2, a carry occurred out of a four-bit group, add 0000 to that group; if no carry occurred, add 1010 (the complement of 0110) to that group.
4. If there was a carry out of the high-order four bits in step 2, the answer is in true form. If there was no carry out of the high-order four bits in step 2, the answer is in complement form and must be recomplemented. In this case, take the 2's complement of the number and decimal correct those four-bit groups where no carry occurred. The procedure is:
   a. Invert each position of the complement answer and add 1 to the low-order position.
   b. Decimal correct by adding 0000 to each four-bit group that has a carry out, and 1010 (2's complement of 6) to each four-bit group that has no carry out.

For example, subtract 15 (second operand) from 18 (first operand):

1. Complement the second operand.

   Second operand (16) = 0001 0110

   (complemented) = 1110 1010

2. Add first operand to result of step 1.

   ```
   Result of step 1   = 1110 1010
   First operand (18) = 0001 1000
                        0000 0010
   ```

3. Carries occurred out of each four bit group. Therefore, add 0000 to each group. The result is in true form.

   ```
   0000  0010
   0000  0000
   0000  0010
   ```

   In decimal:

   18 - (+16) = 2

As a second example (summarized in Figure 1-10), consider subtraction of +200 from +190:

1. First operand = 0001 1001 0000 1100

   Second operand = 0010 0000 0000 1100

2. The low order bytes are sent to ALU. A complement add is indicated because the number of minus signs is odd (one minus for the subtract operation, each operand is plus).
3. Invert the four high order bits of the low order second operand byte:

   0000 (inverted) = 1111

4. Add the four high order bits of the first operand byte to the result of step 3. Because this is the units position, add 1 to make the notation 2's complement.

   ```
   Result step 3 = 1111
   First Operand = 0000
                     1
                   0000
   ```

5. Because a carry did occur from the four-bit group, no decimal correction is necessary.
6. The next two bytes (one from each field) are sent to ALU.

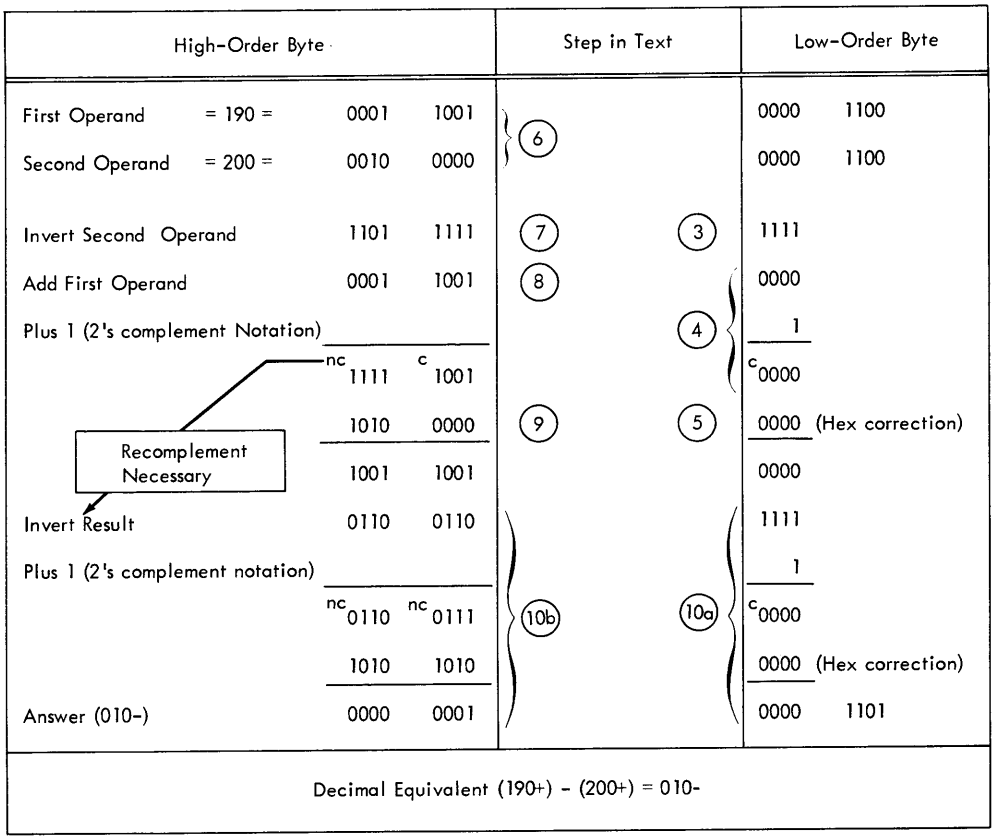


Figure 1-10. Packed Decimal Complement Add Example

7. Invert the second operand byte.

8. The first operand byte is added to the inverted second operand. (Note that a carry out of the high-bit position in the first addition--step 4--did not occur. Therefore, no additional carry is used to form this step 8 total.)

```
Step 7 result       =  1101  1111
First operand byte  =  0001  1001
Result              =  1111  1000
```

9. A carry did not occur out of the high-order four bit group. Therefore, add 1010 to that group. A carry did occur from the low-order four-bit group. Add zero to this group.

```
1111  1001
1010  0000
1001  1001
```

10. There was no carry out of the high order bit of the step 8 result. Therefore, the answer is in complement form and must be recomplemented to produce a true result.

a. Read out the low-order byte resulting from step 5. Invert the high-order four bits and add 1 to produce the correct 2's complement notation.

```
High-order four bits =   0000
Inverted =               1111
Plus 1 (2's Comp) =         1
Low-order byte answer =  0000
```

There is a carry from this four bit group so hexadecimal correction is not needed. Therefore, add 0000 to produce the low-order digit of the answer.

b. Read out the high-order byte. Invert the bits and add the carry from the low-order byte (step 10a).

```
High-Order byte =        1001 1001
Inverted =               0110 0110
Low-order byte carry =   ____ ___1
                         0110 0111
```

There was no carry from either four-bit group. Therefore, both groups must be hexadecimal-corrected by adding 1010 to each group.

```
Result from above =     0110 0111          Result = 0000 0001 0000 1101
Hex-correction =        1010 1010
High-order byte answer  0000 0001          Decimal
                                           Equiv. =  0    1    0    -
```

11. The result of the addition is stored in the first operand location.

Floating Point Arithmetic

- Floating point operands are made up of three fields:
  a. fraction sign-bit
  b. characteristic (represents -64 to +63)
  c. fraction (made up of hexadecimal digits)
- Short precision operands are a word in length.
- Long precision operands are a double word in length.
- A normalized fraction has a high order non-zero hexadecimal digit; an unnormalized fraction has a high-order hexadecimal digit of zero.

Floating point is not a numbering system. Rather, it is a way of representing a quantity in any numbering system. This representation takes the form of a series of digits multiplied by the base (of the numbering system used) which is raised to a power. For example, in the decimal system, the number 1,234 is equal to any one of the following:

123.4×10^{1}
12.34×10^{2}
1.234×10^{3}
$.1234 \times 10^{4}$
$.01234 \times 10^{5}$

Notice that the decimal point is located at a different position in each of the preceding numbers. It is in fact a floating decimal point.

The significant digit portion of a floating point number is called the fraction, and the power to which the base is raised is called the characteristic. For example, in $.1235 \times 10^{4}$, .1235 is the fraction and 4 is the characteristic. Notice that the fraction can be either positive or negative and the characteristic can be either positive or negative:

$.1235 \times 10^{4}$
$-.1235 \times 10^{4}$
$.1235 \times 10^{-4}$
$-.1235 \times 10^{-4}$

Very large and very small quantities can be conveniently represented in floating point format. For example:

$.12 \times 10^{20} = 12,000,000,000,000,000,000$

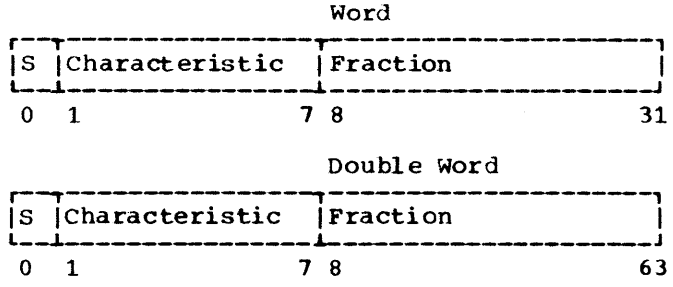
$.567 \times 10^{-20} = .00000000000000000567$

Quantities of such magnitudes are frequently used in scientific computations. Hence, floating point (a special feature) is mainly applicable to processing of scientific problems.

In System/360 floating point operands are fixed in length:

1. Short precision operands are a word in length.
2. Long precision operands are a double word in length.

Floating point operands represent hexadecimal numbers. The areas in a floating point word and double word are:



The sign bit position is at a value of 0 for positive fractions and at a value of 1 for negative fractions:

Hexadecimal Fraction	Sign Bit Value
+.12	0
-.12	1

Up to 6 hexadecimal digits can be represented by the 24 bits of the fraction field in a word. (During addition, subtraction, and division operations, however, a seventh digit-the guard digit-is used to increase the precision of the result.) A double word fraction can contain up to 14 hexadecimal digits. (A guard digit is not used here.)

The hexadecimal point of the fraction is assumed to be immediately to the left of the high-order fraction digit.

The sign of the fraction is taken care of by the sign bit, but notice that there is no sign bit for the characteristic. The characteristic portion in a word or double word is seven bits long. The maximum magnitude of a positive number represented by 7 binary bits is $2^7 - 1$ or 127; the smallest magnitude is 0. In order to represent positive and negative exponents, the value 1000000 (64 in decimal), of the seven bits that comprise the characteristic, is recognized by the system as a characteristic of 0. The maximum positive characteristic is then 1111111 and the maximum negative characteristic is 0000000. Hence, the characteristic is negative if the bit structure of the characteristic field is in the range 0000000 to 0111111 (0 to 63 decimal). This provides a range of negative characteristics from -1 to -64. In other words a characteristic field of 0000011 is recognized by the system as a characteristic of -61. This convention of using 1000000 as zero is called excess 64 notation.

If the characteristic is negative (i.e., a characteristic field in which the high-order bit is zero), its value can be determined by subtracting it (by complement addition) from 1000000 (64 decimal). For example, if the characteristic field is 0010111:

```
Inverted char.        = 1101000
plus 1                = ______1
                        1101001
```

2. Add complement to 1000000:

```
  1000000
  1101001
  0101001
```

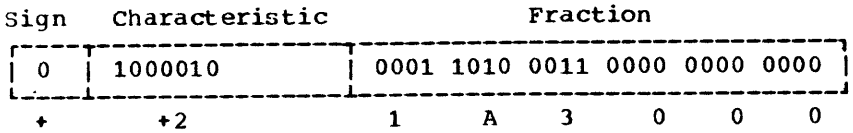
The carry indicates that the answer is in true form, but it is ignored in the result. Therefore: 0101001 = 41 (decimal). Hence, the characteristic represented by 0010111 is -41 (decimal).

To determine the decimal value of a <u>positive characteristic</u>:

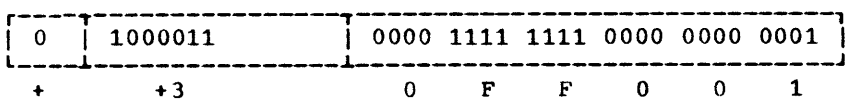
1. Note that the high order bit must be at a value of 1 for positive characteristics.
2. Convert the remaining bit positions to the appropriate decimal value.

For example, 1001001 is a positive characteristic because the high-order digit is 1. The remainder of the characteristic (1001) is equal to 9 in decimal notation. Hence, 1001001 represents a characteristic of +9.

A normalized floating-point number has a non-zero high-order hexadecimal fraction digit. For example, the hexadecimal number $+.1A3 \times 16^2$ in a normalized word (floating point format) is:



Notice that the three high-order binary fraction digits are 0, but that the high order hexadecimal fraction digit is 1. The fraction is normalized, however, because it is normalized with respect to hexadecimal digits and not with respect to binary digits. An example of an unnormalized fraction is:



Normalization is done by left-shifting the fraction digits until the high order hexadecimal digit is non-zero. For each left shift, the characteristic is decreased by one.

When normalization is done prior to an arithmetic operation, it is called prenormalization. Postnormalization is a process that changes an intermediate arithmetic result to its normalized form.

Floating-Point Addition

- The operand fraction with smallest characteristic is right-shifted until the two operands have equal characteristics; then the operands are added.

Before addition starts, the characteristics of both operands are compared. The fraction with the smaller characteristic is right-shifted. For each right-shift, the characteristic is increased by one. When the characteristics of the two operands are equal, shifting stops. The fractions are then added algebraically to form an intermediate sum. If a carry occurs out of the high-order hexadecimal sum digit, the sum fraction is shifted right once and its characteristic is increased by one. If a characteristic overflow occurs as a result of this increase, an exponent-overflow exception (program interruption) occurs.

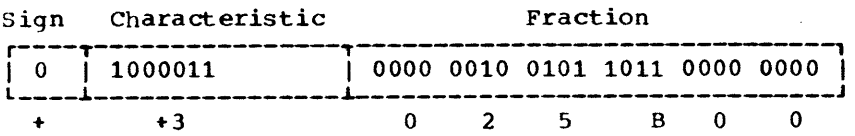
There are two floating-point add instructions: add normalized and add unnormalized. When the add unnormalized instruction is executed, the sum is stored without normalization. In the normalized add instruction, however, the intermediate sum is left-shifted until the high-order hexadecimal digit is non-zero. For each left-shift of the hexadecimal digits, the characteristic is decreased by one.

For example, add .124 X 16^2 to .0127 X 16^3 using the add unnormalized instruction (short precision assumed):

1. .124 X 16^2 is right-shifted once before the addition starts.
   .124 X 16^2 (right-shifted once) = .0124 X 16^3
2. The fractions are added:

```
  .0127
+.0124
 .025B
```

3. The result is not normalized but it is stored as is:



The same operation using an add normalized instruction is:

1. .124 X 16^2 is right-shifted once before the addition starts:

   .124 X 16^2 (right-shifted once) = .0124 X 16^3

2. The fractions are added:

```
  .0127
+.0124
 .025B
```

3. The result is normalized by one left-shift:

   .025BX 16^3 = .25BX 16^2 (normalized)

4. The result is then stored:

Sign	Characteristic	Fraction
0	1000010	0010 0101 1011 0000 0000 0000
+	+2	2 5 B 0 0 0

Floating-Point Subtraction

- The sign of the second operand fraction is changed before the operation starts.
- The operation follows the sign rules of algebra so that, if necessary, the second operand bytes are complemented during the operation.

Floating-point subtract is similar to floating-point add:

1. The fraction with smallest characteristic is right-shifted before subtraction starts. The characteristic is increased by one for each right-shift of the hexadecimal digits.
2. There are two floating-point subtract instructions:
   a. Subtract normalized.
   b. Subtract unnormalized.

These instructions are executed in basically the same way as the corresponding add instructions.

Note that the sign of a negative fraction is indicated in the sign bit position of a floating point operand. The fraction, however, is carried in true form (i.e., not two's-complement form as in fixed-point operands) as a series of hexadecimal digits.

Before the actual subtraction, the sign of the second operand (always the operand that is subtracted from the first operand) is changed. Then, if necessary, each second operand byte is complemented as it enters ALU. The operation follows the sign rules of algebra. Hence, for a subtraction operation:

Sign of 1st Operand	Original Sign of 2nd Operand	Complement 2nd Operand Bytes?
+	+	Yes
+	-	No
-	+	Yes
-	-	No

IBM SYSTEM/360 GENERAL INFORMATION

NUMBERING SYSTEMS

- A number is a sum of terms; each term is a product of a digit symbol times some power of the base of the numbering system.
- A carry out of a position occurs when a one is added to the highest valued symbol in that position.

Combinations of the symbols of a numbering system represent quantities or amounts. A quantity can relate to specific items (such as five apples), or it can be abstract (2 + 3 = 5). In either case, symbols are used to express the quantity. For example, the quantity can be represented by:

1. The written numeral 5.
2. The written word five.
3. The sound produced when the word five is spoken.

The symbol for any quantity depends upon the numbering system used. For example, the Roman Numeral symbol for the number 5 is V. The symbol(s) used to express a quantity have been adopted by convention. We are familiar with the meaning of the symbol 5, but if we had always represented 5 with a #, we would know what the # stands for.

Consider some basic decimal-numbering-system conventions. These will help you to understand less familiar numbering systems. A decimal number is composed of symbols that are called digits. The decimal number 12 is represented by the digits 1 and 2, but these symbols must be written in a specific position. (12 is a different quantity than is 21.) Each decimal digit has a meaning that is determined by its position in a string or series of digits. Some of the decimal positions are defined:

```
1 2 3 4
↑ ↑ ↑ ↑
| | | units
| | tens
| hundreds
thousands
```

Each of these positions is defined in terms of powers of ten (the base of the decimal numbering system). Any decimal number can be represented by multiplying each position by the appropriate power of ten and adding the products. For example, 1234 is the same quantity as:

$$1 \times 10^3 + 2 \times 10^2 + 3 \times 10^1 + 4 \times 10^0 = 1234$$

$$1000 + 200 + 30 + 4 = 1234$$

Notice that the units digit (4) is multiplied by 10^0. Any decimal number raised to the zero power is 1. This is also true of the other numbering systems described in this publication.

In proceeding to the left from the units position, the power of the base (10) is raised one degree for each position moved. Fractions are handled similarly, except that in proceeding to the right from the units position, the power of the base is lowered one degree for each position moved. Hence, the number 2.34 is the same as:

$$2 \times 10^0 + 3 \times 10^{-1} + 4 \times 10^{-2}$$

One other convention you should understand is the way in which carries are handled. In the decimal system, counting can proceed from a zero quantity up to nine without any carry:

0
1
2
3
4
5
6
7
8
9

At this point, however, there are no more decimal symbols to use. Of course, the next number is 10; but why? Notice the significance of the carry from the units to the tens position. What happens is that when we run out of symbols, we carry to the next higher position. The zero indicates that we start counting all over again.

Counting then proceeds as follows:

10
11
12
13
14
15
16
17
18
19

Again we have run out of symbols to represent the unit position quantity. So, we carry one over to the tens position and the zero tells us to start counting again. This procedure continues until we reach 99. At this point we have run out of symbols in both units and tens positions. So, we carry from the units to the tens position and from the tens to the hundreds position. We now have 100 and can start counting units again. Each time we run out of units symbols we again carry to the tens position until we run out of symbols in both the units and tens positions, in which case another carry is made to the hundreds position.

These same concepts of carrying and starting again at zero are used in the binary and hexadecimal numbering systems. In these systems, however, the number of symbols (two for binary, sixteen for hexadecimal) used is different than the number of decimal symbols.

BINARY

- The only two digit symbols in the binary system are 0 and 1.
- Binary addition rules are:

  0 + 0 = 0

  1 + 0 = 1

  0 + 1 = 1

  1 + 1 = 0 plus a carry

  1 + 1 + 1 = 1 plus a carry

- To obtain the two's-complement of a binary number:

  1. Invert each position of the original number (i.e., change all 0's to 1's and all 1's to 0's).

  2. Add 1 to the low order position of the inverted number.

- In order to subtract B from A (both binary numbers) complement B and add it to A. A carry out of the high-order position signifies that the answer is in true form; no carry out of the high-order position indicates that the answer (which must be recomplemented to obtain an answer in true form) is in two's-complement form.

The binary numbering system uses a base of 10 (2 in decimal notation) and has two symbols (0 and 1). Let's try the principles of carry and starting-at-zero that we used for decimal numbers and add two binary numbers.

1 + 1 = 10

Notice that we started out with 1 and then added 1 to it. We were already out of symbols when we started, so a carry was made to the next position. The 0 in 10 means that we can start counting in the low-order position again. Let's add 1 more to the total:

10 + 1 = 11

A carry did not occur because the low-order position could accomodate one additional count without running out of symbols.

Just as in the decimal system, the zero in binary is a quantity which when added to a second quantity results in a sum equal to the second quantity.

For example:

1 + 0 = 1

or

0 + 1 = 1

We can now summarize all the facts you need in order to add in binary:

0 + 0 = 0

1 + 0 = 1

0 + 1 = 1

1 + 1 = 0 (with a carry to the next position) = 10

1 + 1 + 1 = 1 (with a carry to the next position) = 11

The last item of the list can be broken down into:

1 + 1 = 10 + 1 = 11

Subtraction of binary numbers can be performed by using the same basic principles used in decimal subtraction. However, the System/360 uses complement addition rather than subtraction. Therefore, complement addition is described here.

The two's complement of a binary number is used in complement addition. It is obtained by inverting each position of the original number and adding 1 to the low order position. To find the two's complement of 1100 invert each position:

1100 inverted is 0011

Then add 1:

```
 0011
+  1
 0100
```

Using this principle, let's subtract 0100 from 1000 by complement addition.

The two's complement of 0100 is:

1011 + 1 = 1100

Now the complement of 0100 is added to 1000:

```
  1000
  1100
1 0100
```

The high-order digit indicates that the answer is in true form and this digit is not part of the total. The same operation using decimal notation is:

Binary Subtraction	Complement Addition	Decimal Subtraction
1000	1000	8
-0100	+1100	-4
0100	0100	4

Now subtract 0011 from 0010. First 0011 is complemented:

1100 + 1 = 1101

Then the addition is performed:

```
 0010
 1101
 1111
```

Notice that there was no carry out of the high order position. This lack of a carry indicates that the answer is in two's-complement form. To obtain an answer in true form, recomplement the answer:

1111

0000 + 1 = 0001

Or, 3 (0011) subtracted from 2 (0010) is a minus 1.

Binary to Decimal Conversion

- To convert from binary to decimal, sum the appropriate powers of two that correspond to 1's in the original binary number.

Just as in the decimal system, any binary number can be represented as a series of multiplications that are added together. For example:

$1101 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 13$ (decimal)

This example really presents a way of converting binary to decimal. Recognize that we have only two symbols in binary (0 and 1) but we have used the symbols 0, 1, 2, and 3 in this conversion. Shown in true binary, the above series is:

$1 \times 10^{11} + 1 \times 10^{10} + 0 \times 10^{1} + 1 \times 10^{0}$

The arithmetic would look like this:

$1 \times (10)\ (10)\ (10) + 1 \times (10)\ (10) + 0 \times 10 + 1 \times 1 = 1101$

or

$1 \times 1000 + 1 \times 100 + 0 \times 10 + 1 \times 1 = 1000 + 100 + 0 + 1 = 1101$

There is no need to use this second procedure. What we are primarily interested in is a conversion process from binary to decimal so that a quantity can be represented in the familiar decimal numbering system.

Notice in the original conversion to decimal that each binary position value, of either 1 or 0, is multiplied by a power of two. Therefore, in converting from binary to decimal, you need sum only those powers of 2 that are multiplied by a 1. (0 times anything is 0.) Using powers of 2, convert 11011 to decimal. The powers of two used are:

$2^4, 2^3, 2^2, 2^1, 2^0$

or

16, 8, 4, 2, 1

By placing these over the appropriate binary positions, you can quickly add to find the equivalent decimal symbols:

16	8	4	2	1
1	1	0	1	1

$16 + 8 + 2 + 1 = 27$ (decimal)

The four was multiplied by zero, and therefore, it was not used in the summation.

Conversion of fractions is not generally necessary. The following information is, therefore, presented for reference purposes only. To convert a binary fraction to a decimal fraction:

1. Express the binary fraction as a decimal series using powers of two:

   $.111 = 1 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3}$

2. Express the series as a fraction:

   $1 \times 2^{-1} \times 1 \times 2^{-2} + 1 \times 2^{-3} = 1/2 + 1/4 + 1/8 = 7/8$

3. Divide the numerator by the denominator to form the decimal fraction:

   $\frac{7}{8} = .875$ (decimal) $= .111$ (binary)

Decimal to Binary Conversion

- To convert a number in decimal notation to a number in binary notation:

  1. Divide the original number and subsequent quotients by two.
  2. Each remainder is a successively higher ordered binary digit.
  3. The last quotient (always a 1) is the high-order binary digit.

The following procedure can be used to convert a number in decimal notation to a number in binary notation:

1. Divide the decimal number by 2. The remainder is the low-order binary digit.
2. Divide the quotient obtained in step 1 by 2. The remainder is the next binary digit.
3. Continue dividing subsequent quotients by two, to obtain each binary digit, until a final quotient of 1 is reached. This last quotient is the high-order binary digit.

For example, the conversion of 24 (decimal) to a binary number is shown in Figure 1-11.

Decimal fraction to binary fraction conversion is more easily accomplished by use of hexadecimal notation. See the Decimal to Hexadecimal Conversion section.

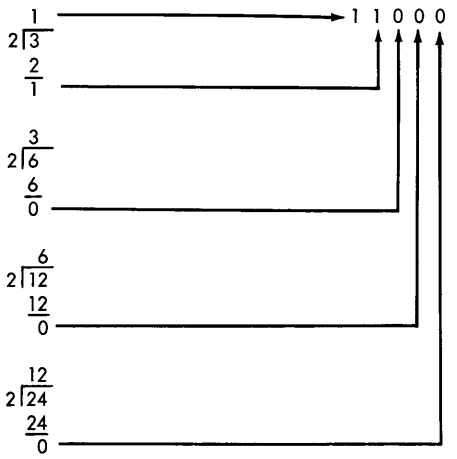


Figure 1-11. Conversion of 24 (Decimal) to Binary Notation

HEXADECIMAL

- The hexadecimal system has 16 symbols (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F).
- The 16's-complement of a hexadecimal number is found by:

  1. Subtracting the digits of the original number from an equivalent number of F's.
  2. Adding 1 to the low-order position of the result obtained in step 1.

- Hexadecimal subtraction of B from A can be done by complementing B and adding; the answer is in true form if a carry-out of the high-order position occurs; the answer is in 16's-complement form if a carry does not occur out of the high-order position.

The hexadecimal numbering system uses a base of 10 (16 in decimal notation) and has the following symbols:

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

Addition of hexadecimal symbols is similar to decimal addition except that a carry does not occur until the unit sum exceeds F (15 in decimal). Hence, in hexadecimal notation:

F + 1 = 10
F + 2 = 11
F + 3 = 12
F + 4 = 13
F + 5 = 14
F + 6 = 15
F + 7 = 16
F + 8 = 17
F + 9 = 18
F + A = 19
F + B = 1A
F + C = 1B
F + D = 1C
F + E = 1D
F + F = 1E
F + F + 1 = 1F
F + F + 2 = 20

An example of hexadecimal addition written in another form is:

In hexadecimal	In decimal
B A C	2988
+1 F E	+510
D A A	3498

In other words:

C + E = A plus a carry

A + F + 1 (the carry) = A plus a carry

B + 1 + 1 (the carry) = D

Subtraction of hexadecimal numbers can be performed by complement addition. To subtract 1FE from BAC, find the 16's complement of 1FE and add it to BAC. (The 16's complement of a hexadecimal number is determined by subtracting each position from F and adding one to the low-order position of the result.)

The operation proceeds as follows:

1. Problem BAC - 1FE = ?

2. Complement 1FE:
   FFF
   (-) 1FE
   E01 + 1 = E02

3. Addition:

   BAC
   +E02
   9AE

4. The carry out of the high order position is ignored in the result and the answer is 9AE.

If there is no carry out of the high order position, the result is in 16's-complement notation and must be recomplemented if an answer in true form is desired.

Binary to Hexadecimal Conversion

- Four binary digits can be represented by one of the 16 hexadecimal symbols.
- To convert from binary to hexadecimal:
  1. Divide the binary field into four-digit groups, starting from the binary point.
  2. Substitute, in order, the appropriate hexadecimal symbol for each four-digit group.

Four binary digits can be represented by a single hexadecimal digit. This results from the fact that 16, the base of the hexadecimal system, is equal to 2^4 which is the fourth power of the base of the binary system. Therefore, conversion of binary to hexadecimal is accomplished as follows:

1. Divide the binary field to be converted into four-digit groups. Start counting groups of four from the binary point (i.e., the point that separates binary fractions from whole numbers). For example, 111000100111 can be divided into:

   1110 0010 0111

2. Convert each four-digit group into its decimal equivalent:

Binary	Decimal Equivalent
1110	14
0010	2
0111	7

3. Substitute the appropriate hexadecimal symbol for each decimal quantity:

Decimal Quantity	Hexadecimal Symbol
14	E
2	2
7	7

4. Arrange the hexadecimal symbols in the sequence that corresponds to the original number:

   1110 0010 0111 = E27

As you become more proficient in using hexadecimal symbols you will probably omit step 2.

Hexadecimal to Binary Conversion

- Each hexadecimal digit is equivalent to the quantity represented by four binary digits.
- Conversion from hexadecimal to binary is accomplished by substituting, in sequence, four binary digits for each hexadecimal digit.

To convert from hexadecimal to binary, reverse the hexadecimal to binary procedure. For example, convert F26B to binary:

Hexadecimal Symbol	Decimal Equivalent	Binary Equivalent
F	15	1111
2	2	0010
6	6	0110
B	11	1011

Or:

F26B = 1111 0010 0110 1011

Hexadecimal to Decimal Conversion

- Because each hexadecimal place value is determined by powers of 16, conversion to decimal notation is effected by repeated multiplications by 16.

Use the following procedure to convert a hexadecimal to a decimal number:

1. Convert each hexadecimal digit to an equivalent decimal number.
2. Multiply the high order digit (equivalent decimal number) by 16.
3. Add the next lower order digit (equivalent decimal number) to the product obtained in step 1.
4. Multiply the sum obtained in step two by 16.
5. Add the next lower order digit to the product obtained in step 3.
6. Continue forming products and adding the next lower order digit until the units position is reached.
7. Add the units position to the last product formed and stop. Do not form another product by multiplying by 16.

For example, convert 1FE1 to decimal notation:

1. Conversion of each digit of 1FE1 to equivalent decimal numbers produces: 1 15 14 1

2. 1 15 14 1

   x16
   16

3. +15 ←
   31

4. x16
   496

5. +14 ←
   510

6. x16
   8160

7. + 1 ←
   8161 (answer)

Conversion of hexadecimal to decimal fractions may be useful if you are considering floating point operations. This conversion, however, is presented here for reference purposes only. To convert a hexadecimal fraction to an equivalent decimal fraction:

1. Express the hexadecimal fraction as a sum of equivalent decimal numbers (1 through 15) times powers of 16.
2. Form fractions for each term.
3. Express each fraction in terms of a common denominator and form one fraction.
4. Divide the numerator of the fraction by the denominator. (Round off the result to obtain the desired accuracy.) The result is the approximate decimal equivalent of the original hexadecimal fraction.

As an example, convert .1FE to a decimal fraction:

1. $.1FE = 1 \times 16^{1} + F \times 16^{-2} + E \times 16^{-3}$

   $.1FE = 1 \times 16^{-1} + 15 \times 16^{-2} + 14 \times 16^{-3}$

2. $\frac{1}{16} + \frac{15}{256} + \frac{14}{4096}$

3. $\frac{256 + 240 + 14}{4096} = \frac{510}{4096}$

4. $\frac{510}{4096} = .1245$ (rounded off)

Decimal to Hexadecimal Conversion

- Because each hexadecimal place value is in terms of powers of 16, conversion from decimal to hexadecimal notation is effected by successive divisions by 16.

To convert a decimal to a hexadecimal number:

1. Divide the decimal number by 16. The remainder is the low order hexadecimal digit.
2. Divide the quotient obtained in step 1 by 16. The remainder is the next higher order hexadecimal digit.
3. Continue dividing quotients by 16 and using the remainders for each succeeding hexadecimal digit until the quotient becomes less than 16.
4. The final quotient (less than 16) is the high order hexadecimal digit.

For example, convert 510 (decimal) to hexadecimal notation:

1. $16\overline{)510}$ = 31, remainder 14

14 is the remainder and represents a low order hexadecimal digit of E.

2. $16\overline{)31}$ = 1, remainder 15

15 is the remainder and it represents the next higher order hexadecimal digit, F.

3. The quotient in the preceeding division is 1, which is less than 16. The 1, then, is the high order hexadecimal digit.

4. 510 (decimal) = 1FE (hexadecimal)

Conversion of a decimal fraction to a hexadecimal fraction is applicable mainly if you are working with floating point operations. This conversion process is presented here primarily for reference purposes. To convert a decimal fraction to a hexadecimal fraction:

1. Change the decimal fraction from its decimal-point form to an equivalent ten thousandths fraction.
2. Multiply the numerator of the fraction obtained in step 1 by 65,536.
3. Divide the product (obtained in step 2) by 10,000, rounding off to the nearest unit.
4. Convert the decimal number obtained in step 3 to an equivalent hexadecimal number. (You can use the preceeding method described in this section.)
5. The low order hexadecimal digit is four places to the right of the hexadecimal point (if a four place decimal fraction was converted). Insert any necessary zeros to the left of the high order hexadecimal digit to obtain a four place fraction.

For example, convert .1245 (decimal) to a hexadecimal fraction, as follows:

1. $.1245 = \frac{1245}{10,000}$
2. $\frac{(1245)\ (65536)}{10,000} = \frac{81,592,320}{10,000}$
3. $\frac{81,592,320}{10,000} = 8159$ (rounded off)
4. a. $\frac{8159}{16} = 509$ with remainder of 15.

   Therefore, F (15 decimal) is the low order hexadecimal digit.

   b. $\frac{509}{16} = 31$ with remainder of 13 (D in hexadecimal).

   c. $\frac{31}{16} = 1$ with remainder of F

   d. The last quotient was less than 16 and is therefore the high order hexadecimal digit.

   e. Therefore, .1245 (decimal) is approximately equal to .1FDF.

Notice that the decimal fraction used (.1245) is the result of converting .1FE to a decimal fraction in the Hexadecimal to Decimal Conversion section. But the answer obtained when .1245 is converted back to hexadecimal is .1FDF. The differences occur because rounding was used in each conversion process. Hence, .1FDF rounded off one place higher is .1FE.

Conversion of a decimal fraction to a hexadecimal fraction is equivalent to conversion of a decimal fraction to a binary fraction. That is, the hexadecimal fraction obtained can be converted to a binary fraction. Hence, .1FE is equivalent to the binary fraction .00011111 1110.

It is of some interest to note why the 65536 factor is used in the conversion process. A four place decimal fraction is four decimal digits (the numerator) divided by 10,000 (the denominator).

Or, $.1234 = 1234 \times 10^{-4} = \frac{1234}{10,000}$

Similarly, each hexadecimal fraction is a numerator times some power of 16. The hexadecimal fraction place values are (in decimal notation):

$$\frac{1}{16}, \quad \frac{1}{256}, \quad \frac{1}{4096}, \quad \frac{1}{65536}, \quad \text{etc .}$$

In the conversion process, then, a four place decimal fraction is approximately equal to some numerator over 65536:

$$\frac{DDDD}{10,000} = \frac{XXXX}{65,536} = \frac{\text{Hexadecimal Digits}}{10^4}$$

(Note that $10^4 = (F+1)^4$, in the fraction on the right.) The hexadecimal numerator is found in terms of decimal digits (XXXX) and converted to an equivalent hexadecimal number.

If decimal fractions of more than four places are to be converted to hexadecimal notation, a factor larger than 65536 must be used.

For example, to convert a five place decimal fraction to a hexadecimal fraction, use 16^5 instead of 65536 in the conversion process. Also, the resulting hexadecimal point's position is determined by the power of 16 used. A five place hexadecimal fraction (rounded off) results from conversion of a five place decimal fraction. Notice, however, that zeros may have to be inserted to the left of the hexadecimal digits. The number of zeros added between the significant hexadecimal digits and the hexadecimal point is:

Number of inserted 0's = power of decimal base used minus number of hexadecimal digits in result.

INFORMATION FORMATS

- The smallest addressable unit of storage - the byte - is made up of eight information bits plus a parity bit.
- Fixed-length information is carried in fields that are:
  1. One half word (two bytes) long,
  2. One word (four bytes) long, or
  3. One double word (eight bytes) long.
- Fixed-length fields are addressed at their leftmost byte which must be located at storage locations whose addresses are divisible by:
  1. Two for half words.
  2. Four for words.
  3. Eight for double words.
- If fixed-length fields are not addressed according to the preceeding rules, a specification exception (program check) occurs.
- A variable-length field, regardless of its length, can start at any main storage address.

The basic information unit used by System/360 is the byte. The byte is the smallest addressable unit of main storage. A byte is composed of eight information bits plus, for checking purposes, a P (parity) bit. Bit positions of a byte are:

P 0 1 2 3 4 5 6 7

Each byte bit-position can have a value of 0 (off) or 1 (on). The P bit is used to maintain odd parity. If an even number of bits 0 through 7 are at a 1 value, then the P bit is set to a 1 value. The P bit is set to 0, however, if an odd number of bits 0 through 7 are set to 1. For example, if bits 6 and 7 (an even number) are at a 1 value, then the P bit is 1:

Bit position	P	0	1	2	3	4	5	6	7
Bit value	1	0	0	0	0	0	0	1	1

If, however, bits 5, 6, and 7 (an odd number) are at a 1 value, then the P bit is 0:

Bit position	P	0	1	2	3	4	5	6	6
Bit value	0	0	0	0	0	0	1	1	1

The P bit does not always accompany a byte. For instance, parity is not carried through the ALU (in System/360, Model 30) but it is generated for the result byte at the output of ALU. In general, succeeding descriptions and figures in this manual do not show the P bit position. Its presence is assumed unless otherwise noted.

A two-byte field is a half word. (Each byte in the half word has its own parity bit.) Numbering of bit positions for a half word proceeds left to right (0 to 15) through both bytes (Figure 1-12).

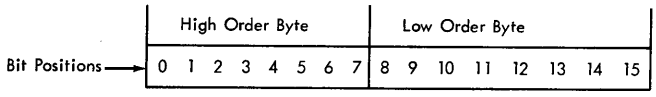


Figure 1-12. Half Word

A four-byte field (two half words) is called a word. Bit positions are numbered left to right, 0 to 31 (Figure 1-13).

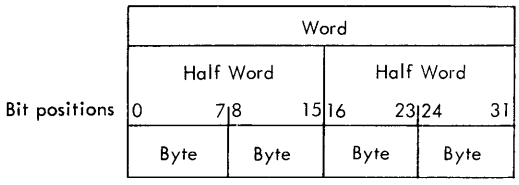


Figure 1-13. Word

Eight bytes (two words) comprise a double word. Bit positions are numbered left to right, 0 to 63 (Figure 1-14).

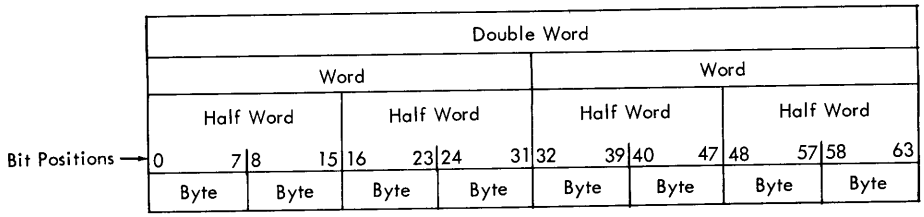


Figure 1-14. Double Word

The sizes of fixed-length fields are defined in terms of a half word, a word, or a double word. All instructions and many data fields are fixed in length. Instructions, for example, are always one, two, or three half words long.

Certain address restrictions must be followed when fixed-length operations are performed. The rule is that fixed-length information must reside on the correct boundaries in main storage. Fixed-length information is addressed at its high-order (left-most) byte location. This address must be divisible by (Figure 1-15):

1. Two for half words.

2. Four for words.

3. Eight for double words.

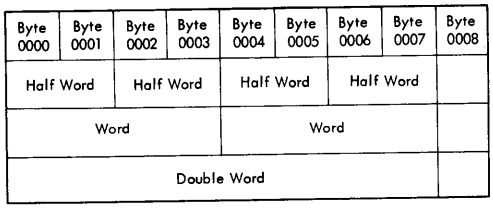


Figure 1-15. Boundary Restrictions

In other words, the low order byte of the address for the fixed-length information must have:

1. Its low-order bit set to zero in order to address a half word.

2. Its two low-order bits set to zero in order to address a word.

3. Its three low-order bits set to zero in order to address a double word.

If any one of these boundary restrictions is violated, a program check occurs. This check is called a specification exception. Hence, it is the responsibility of the programmer to make sure that these boundary restrictions are not violated. (The specification exception does not cause a machine check.)

These boundary restrictions apply to fixed-length information only. A variable-length operand, even if it is a half word, word, or double word in length, can start at any main storage location.

The bit settings in one byte can represent:

1. Special (or conditional) information,

2. A binary number (or part of a binary number),

3. An alphabetic or special character in zoned format,

4. A single decimal digit in zoned format,

5. Two decimal digits in packed decimal format,

6. The characteristic and sign, or part of the fraction of a floating point number.

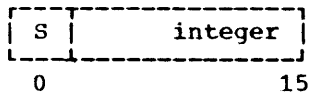
Item 1 relates to cases in which a number (or part of a number) or character (alphabetic, special, or numeric digit) is not represented by the byte. Rather, the setting of a bit or bits indicates that a particular condition does or does not exist or that a certain action is or is not allowed. For example, the first eight bit positions of a PSW are called the system mask. A bit in this byte when on (value of 1) indicates that a certain operation is allowed. (What the mask bits specifically indicates is not pertinent to this description.) When the same bit is off (value of 0), it indicates that the operation is not allowed. Hence, each of these bits represents a condition and not a number or a character.

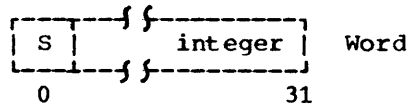
Items 2 through 5 in the preceding list are described in the following paragraphs. For information about item 6, refer to the Floating Point Arithmetic section.

FIXED POINT NUMERIC FORMATS

- Fixed point numeric fields contain representations of binary numbers.
- The high order bit of fixed point numeric fields is the sign bit; it has a value of:
  1. 0 for a positive field.
  2. 1 for a negative field.

Fixed-point numeric fields represent binary numbers. The high-order bit indicates the sign of the field. Positive numbers are represented in true binary form with the sign bit set to 0. Negative numbers are carried as two's - complements of their true binary form with the sign bit set to 1. Fixed-point operands are usually half words or words:





In some operations, such as convert-to-decimal, one of the operands is a double word.

Fixed-length instructions (all part of standard instruction set) are in the RR, RX, or RS format.

ZONED FORMAT

- The EBCDI (Extended Binary Coded Decimal Interchange) code and ASCII (American Code for Information Interchange) can be used in the System/360 eight-bit byte environment.
- The four high-order bits of a zoned-format byte contain the zone; the four low-order bits contain the digit.
- The zoned format is used primarily for character sensitive I/O devices.
- The sign of a zoned-format numeric field is contained in the four high-order bits (0 through 3) in the low-order byte of the field.

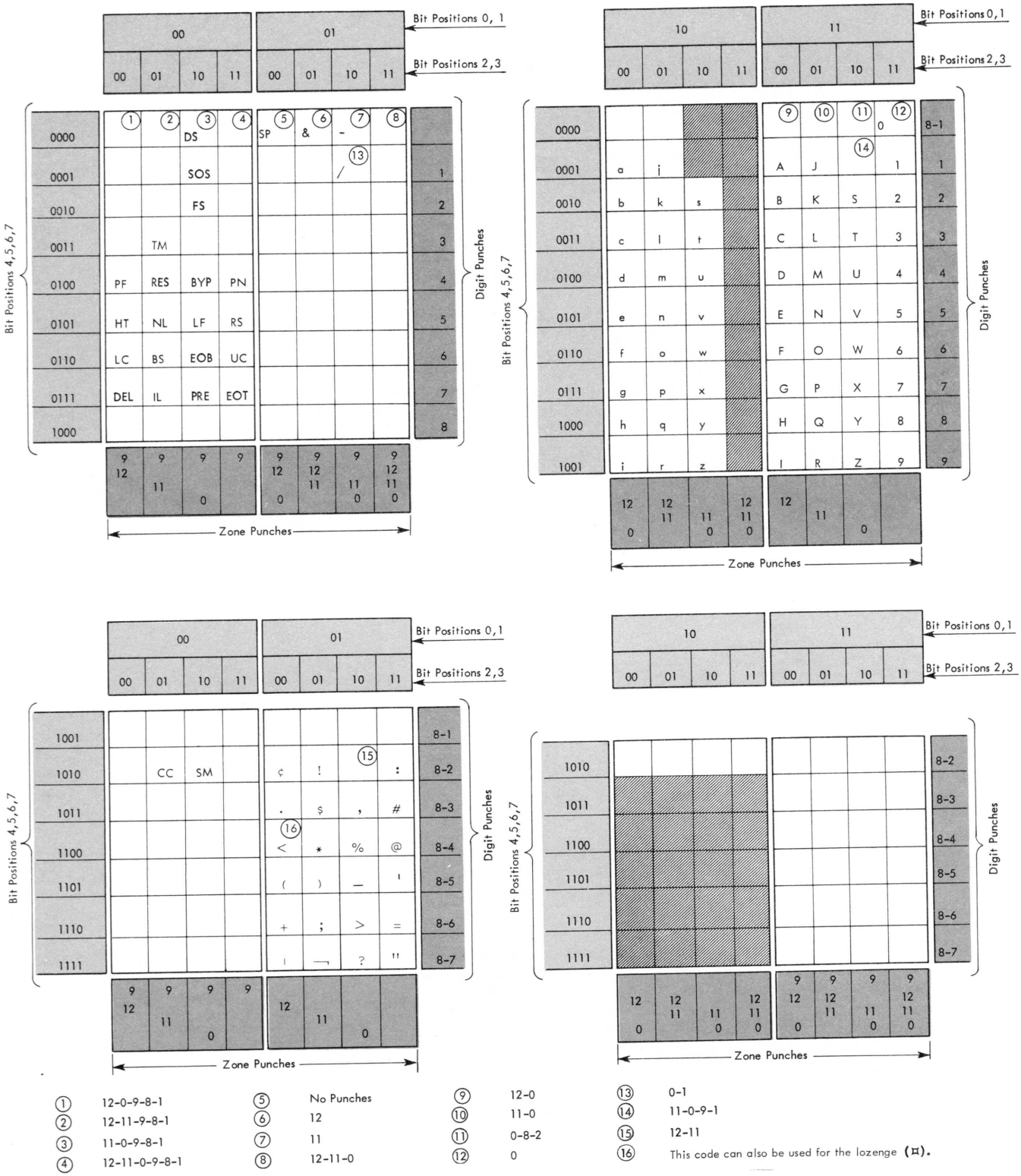
The EBCDI (Extended Binary Coded Decimal Interchange) code is designed for use in eight bit environments. Any one of the characters shown in Figure 1-16A can be represented by one eight bit byte. (The bits in a byte can be set to any one of 256 different combinations.) For example, the EBCDI code for the letter A (see Figure 1-16A) is:

Bit position	0	1	2	3	4	5	6	7
Bit value	1	1	0	0	0	0	0	1

The ASCII (American Standards Code for Information Interchange) is another code that can be used by System/360. In this publication, however, we will deal primarily with the EBCDI code. (For further information about ASCII, refer to IBM System/360 Principles of Operation, Form A22-6821.)

Information carried in the EBCDI code is in zoned format. That is, bits 0 through 3 contain the zone portion of the code while bits 4 through 7 contain the numeric portion. Hence, the numeric character 4 in EBCDI code (see Figure 1-16A) is:

	Zone Portion				Numeric Portion			
Bit position	0	1	2	3	4	5	6	7
Bit value	1	1	1	1	0	1	0	0



① 12-0-9-8-1
② 12-11-9-8-1
③ 11-0-9-8-1
④ 12-11-0-9-8-1
⑤ No Punches
⑥ 12
⑦ 11
⑧ 12-11-0
⑨ 12-0
⑩ 11-0
⑪ 0-8-2
⑫ 0
⑬ 0-1
⑭ 11-0-9-1
⑮ 12-11
⑯ This code can also be used for the lozenge (¤).

Figure 1-16A. Extended Binary Coded Decimal Interchange Code

Control Characters

PF	Punch Off	BS	Backspace	PN	Punch On
HT	Horizontal Tab	IL	Idle	RS	Reader Stop
LC	Lower Case	BYP	Bypass	UC	Upper Case
DEL	Delete	LF	Line Feed	EOT	End of Transmission
RES	Restore	EOB	End of Block	SP	Space
NL	New Line	PRE	Prefix		

Special Graphic Characters

¢	Cent Sign	*	Asterisk	>	Greater-than Sign
.	Period, Decimal Point	)	Right Parenthesis	?	Question Mark
<	Less-than Sign	;	Semicolon	:	Colon
(	Left Parenthesis	¬	Logical NOT	#	Number Sign
+	Plus Sign	-	Minus Sign, Hyphen	@	At Sign
|	Vertical Bar, Logical OR	/	Slash	'	Prime, Apostrophe
&	Ampersand	,	Comma	=	Equal Sign
!	Exclamation Point	%	Percent	"	Quotation Mark
$	Dollar Sign	_	Underscore		

Examples	Type	Bit Pattern Bit Positions 01 23 4567	Hole Pattern	
			Zone Punches	Digit Punches
PF	Control Character	00 00 0100	12 -9 - 4	
%	Special Graphic	01 10 1100	0 - 8 - 4	
R	Upper Case	11 01 1001	11 - 9	
a	Lower Case	10 00 0001	12 -0 - 1	
	Control Character, function not yet assigned	00 11 0000	12 - 11 - 0	-9 - 8 - 1

Figure 1-16B. Key to Figure 1-16A

Zoned information is used by character-sensitive I/O devices. For example, a print record is sent to a 1403 printer in the zoned format. Each specific byte bit-combination represents one character to the 1403. If the 1403 receives an unrecognizable code, then it does not print a character for that position of the print record. Hence, it is important that information is sent to the 1403 in a recognizable (to the 1403) form. Note that some I/O devices are not character sensitive. For example, no matter what the bit pattern of a byte is (correct parity assumed), it can be sent to and handled by a disk storage or magnetic tape unit. The disk or tape unit stores the bit pattern, which may or may not represent a specific character in the ASCII or EBCDI code.

When a signed decimal numeric field is carried in zoned format, the sign of the field is in the four high-order bits of the low-order byte (Figure 1-17). Decimal numeric fields are in the zoned format when read into storage from a character-sensitive I/O device or when sent from storage to a character-sensitive I/O device.

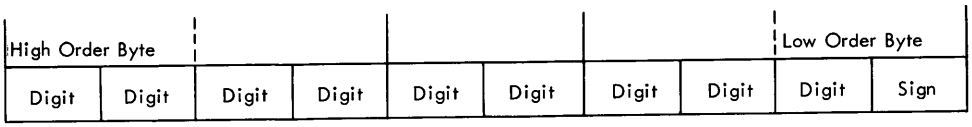
High Order Byte				Low Order Byte	
0 3	4 7	0 3	4 7	0 3	4 7
Zone	Digit	Zone	Digit	Sign	Digit
1111	0000 to 1001	1111	0000 to 1001	1100 = + 1111 = + 1101 = -	0000 to 1001

Note: Sign Bit Combinations are for EBCDI Code

Figure 1-17. Decimal Numeric Field in Zoned Format

PACKED DECIMAL FORMAT

- In the packed-decimal format, representation of two decimal digits is carried in each byte (except the low order byte).
- The sign of a packed-decimal field is carried in the four low-order bits of the field's low-order byte.
- Valid binary codes for each decimal digit are in the range 0000 to 1001 (0 to 9 decimal).
- Zoned-decimal fields can be converted to packed-decimal fields by use of the Pack instruction; packed-decimal fields are converted to zoned-decimal fields by use of the Unpack instruction.



Notes: EBCDI Signs,
1100 = plus
1111 = plus
1101 = minus
All digit values are in the range: 0000 to 1001 (0 to 9 decimal)

Figure 1-18. Packed Decimal Format

If the decimal feature is installed in the 2030, decimal arithmetic operations can be performed when the operands are in the packed-decimal format. Each byte (except the low-order byte) in a packed-decimal field has bits that represent two decimal digits (one digit in the four high-order bits, a second digit in the four low-order bits). For example, the number 19 (ignoring the sign) is represented as 0001 1001. Because the information is decimal, the only valid digits are 0 to 9 (0000 to 1001 in binary). The sign of the packed-decimal field is carried in the four low-order bits of the low-order byte (Figure 1-18).

Zoned-decimal fields can be used to form packed-decimal fields by use of the Pack instruction (Figure 1-19). The Unpack instruction is used to form zoned-decimal fields from packed-decimal fields (Figure 1-19).

Packed-decimal fields are variable in length and are composed of from 1 to 16 bytes. All of the decimal feature instructions are in the SS (storage to storage) format. Hence, all packed-decimal operands are handled in main storage rather than in one of the general registers.

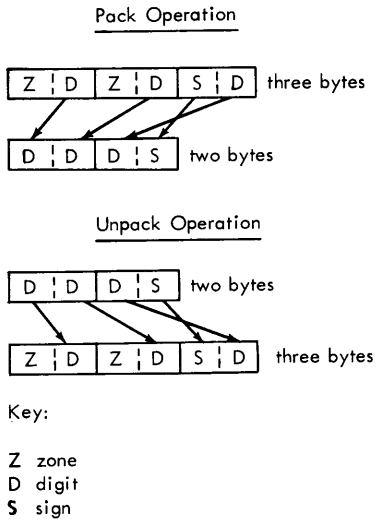


Figure 1-19. Pack and Unpack

BASIC PROGRAMMING

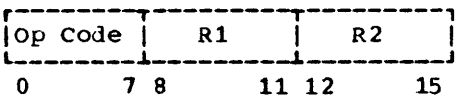
INSTRUCTION FORMATS AND LENGTHS

- Register-to-register (RR) instructions are one half-word long.
- Storage-to-register (RX and RS) and storage-immediate (SI) instructions are two halfwords long.
- Storage-to-storage (SS) instructions are three halfwords long.
- Instructions must reside on half word boundaries in main storage (i.e., low-order bit of instruction address equals zero).

Instructions specify the operation to be done and the locations of the operands that are to participate in the operation. Data referenced in an instruction can be in:

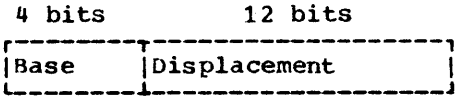
1. A general purpose register,
2. A floating point register, or
3. Main storage.

When an operand is in a general register, then that register is specified by a four bit field in the instruction. RR (register-to-register) instructions are one half word long and have the format:



Here, only one halfword is needed for the entire instruction.

Data in main storage is addressed by the sum of a base and a displacement, or, in some instructions, by the sum of an index, a base, and a displacement. The base is the value in the 24 low-order bits (bits 8 through 31) of a general register. Similarly, the index is the value in the 24 low-order bits of a general register. The displacement is a 12-bit field contained in the instruction. Base-displacement addressing is indicated in an instruction by:

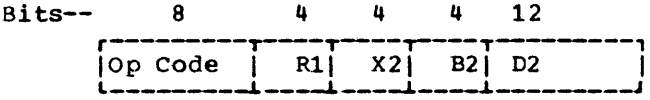


The four bits in the base field are the address of one of the 16 general registers. When (during instruction processing) the actual address is generated, the number in bits 8-31 of the general register is added to the number in the instruction's displacement field. (The desired base has previously been placed, by program control, in the specified general register.)

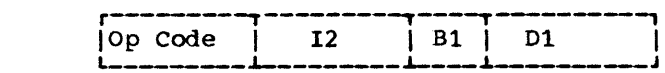
Note, however, that a maximum of 16 low-order bits of the generated address can be used in Model F30 to address a main storage location. Models C30, D30, and E30 use even fewer bits. These restrictions are imposed by the main storage capacities of the models used.

Because instructions that specify operands in main storage must use the base-displacement (or index-base-displacement) method of addressing, they must be longer than one halfword. The formats are:

1. RX or RS, both of which are two halfwords long:



2. SI which is two half words long and contains one of the operands (the I or immediate operand):



3. SS which is three half words long:

In these formats:

R specifies a general register that contains an operand.

X specifies a general register that contains an index.

B specifies a general register that contains a base.

D is a displacement in the instruction.

I is an immediate operand in the instruction.

L is the length of a variable-length operand.

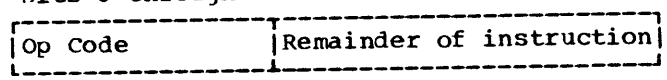
Because instructions are considered fixed-length information, they must be located at halfword boundaries in main storage. That is, the address for any instruction must have its low-order bit set to a value of zero. If this low-order address bit is a one, a specification exception occurs when the instruction is addressed.

OPERATION CODE

- Bits 0 and 1 of an op (operation) code specify:
  1. Instruction length in halfwords, and
  2. General locations of operands.
- Bits 2 and 3 of an op code specify the type of data:
  1. Fixed- or variable-length,
  2. Decimal, binary, or floating point.
- Bits 4 through 7 of an op code specify the operation (such as add or compare).

The high-order byte of every instruction is the op (operation) code:

Bits 0 through 7



Bits 0 and 1 specify the instruction length and the general location of data:

Bits 0 and 1	Instruction Length (half words)	General Location of Data
00	1	Both operands in general or floating point registers
01	2	One operand in main storage
10	2	One operand in main storage
11	3	Both operands in main storage

Bits 2 and 3 specify the type of data to be operated on (i.e., fixed- or variable-length; decimal, binary, or floating-point). Bits 4 through 7 indicate the operation (such as move, subtract, or multiply).

Op codes are frequently represented in hexadecimal notation rather than by eight binary digits. For example, the op code for a fixed-point add instruction in RR format is 1A in hexadecimal notation.

ADDRESSING GENERAL OR FLOATING POINT REGISTERS

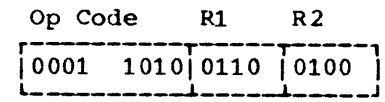
- The four floating point and the sixteen general registers are in the local storage of any System/360 Model 30.
- The instruction's op code indicates whether general or floating point registers are specified by the instruction.

The sixteen general registers can contain fixed-point binary operands while the four floating-point registers can contain only floating-point operands. All sixteen general and the four-floating point registers are in local storage.

The addresses, in instructions, that are used to specify general registers 0, 2, 4, and 6 correspond identically to the four floating-point register addresses. The op code of the instruction, however, specifies whether a general or a floating-point register is to be used in the operation.

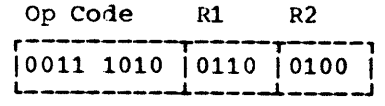
The general registers, as the name general implies, can be used for purposes other than containing fixed-point operands. For example, a general register can contain a base or index used in address generation. (Note, however, that when general register zero is specified as a base or index register, the base or index is zero, no matter what the actual content of general register zero is.)

An example of specification of general registers in an instruction is shown in the following fixed point binary add instruction:



This instruction calls for addition of the contents of general register 0100 (4 in hexadecimal) to the contents of general register 0110 (6 in hexadecimal).

An example of an add normalized floating-point instruction is:



Here, the contents of floating-point register 0100 (4 in decimal) is to be added to the contents of floating-point register 0110 (6 in decimal). The op code indicates a floating-point operation, so that the floating-point registers, and not the general registers, participate in the operation.

MAIN STORAGE ADDRESSING

- Storage addresses are generated by adding a displacement value to a base value.
- The general register that contains the base portion of the address is called the base register.
- The instruction can contain a displacement value as well as the address of a general register that contains the base value. (Sometimes the instruction also contains the address of a general register that contains an index value.)
- Only registers 1-15 can be used as base registers or index registers.
- If register 0 is specified as the base register or index register, its contents are ignored. Instead, a base address or index value of 0 is used.
- The generation of storage addresses does not change the instruction or the base register contents.

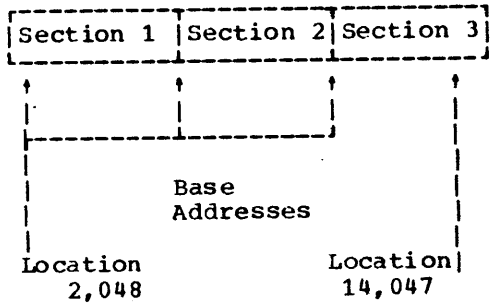
To use a 24-bit address in the instruction for each operand would consume storage space that could be used for other purposes. In the smaller models of System/360 (such as the Model 30 with approximately 8K storage), the amount of main-storage space is definitely limited. One solution would be to use 24-bit addresses on the larger models such as Model 70 and to use shorter addresses on the smaller models. This would mean that programs used on the various System/360 Models would no longer be compatible because of the different address lengths. So we must look for another solution that will reduce the length of the instructions and still maintain compatibility.

There are other features desirable in main storage addressing besides a simple reduction in the length of instructions. It is also desirable that, each time the program is loaded into the computer, the program can start at a different address without having to change the addresses in each instruction. This is known as program relocation, which is a valuable tool in IBM's latest programming systems.

Besides the features of program relocation and shorter instructions, it is also desirable to be able to index instructions.

Assume that System/360 programs are written in sections. Each section is 4096 (decimal) bytes in length. (Of course programs that are less than 4096 bytes can be written as one section.) The beginning of each section is called the <u>base</u> address for that section.

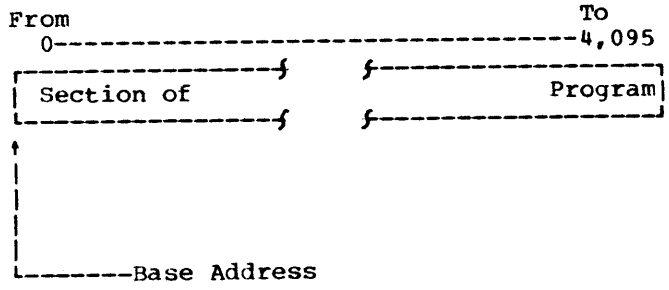
Consider the case of a program that requires 12,000 bytes. By sectioning it into 4096 byte groups, we have three program sections with a base address for each section. For the following example, the program starts at main-storage location 2,048. (The program could also be started at other locations.)



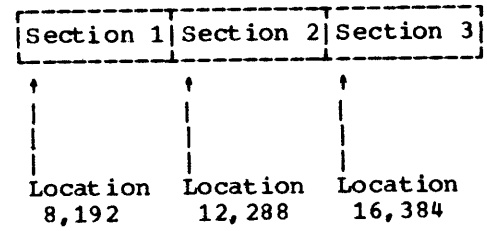
As can be seen in this example, the 12,000 byte program starts at location 2,048 and runs through location 14,047. The first two sections are each 4096 bytes long while the remainder of the program (the last 3,808 bytes) is in section 3.

Now that the program has been sectionalized and base addresses are known, how can this help in addressing main storage?

Because each section is a maximum of 4096 bytes long, any byte in a section can be located by adding to the base address a number in the range of 0-4095. This number is called the displacement. That is, each byte is displaced from the base address from 0 to 4095 places.



Suppose that the program we have been using as an example is moved so that it starts at location 8,192.



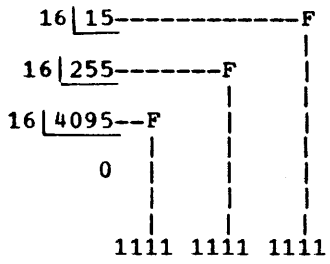
The base address for Section 1 is now 8,192 and the base addresses for Sections 2 and 3 are 12,288 and 16,384. The displacement for each byte in the program has not changed. The last byte of section 1 is still displaced from its base address by 4,095.

The preceding demonstrates the ease with which a System/360 program can be relocated. To relocate a System/360 program, the <u>base</u> addresses are changed while the <u>displacements</u> remain the same.

Main storage addresses are 24 bits long. This allows for compatibility throughout the range of storage capacities for System/360 models, as well as for addressing up to about 16 million bytes. Because a program can start anywhere in main storage, the base addresses for the program must be 24 bits long. (During actual addressing, no Model 30 uses all 24 bits.)

The displacement range for any particular base address is 0-4095. To express

this range requires 12 binary bits. (You can calculate this by converting 4095 to hexadecimal and then to binary.)



Any byte in main storage can be located by adding a 12-bit displacement to a 24-bit base address.

The use of a base address and a displacement makes it easier to relocate a program each time it is loaded into the computer. However, we also want a shorter instruction. To put both the base address and displacement in the instruction would make the instruction longer. It would also mean that each instruction would have to be changed (base address) every time the program is relocated. The manner in which the System/360 handles this is to carry the base address in one of the general registers. When a general register contains a 24 bit base address, it is referred to as a base register. The address of the base register and the 12 bit displacement are carried in the instruction.

Let's take a look at a typical instruction used to add an operand in main storage to an operand in one of the general registers. When only one of the operands is in main storage, the instruction is two halfwords in length. To add a main storage operand (source operand) to a general register operand, (destination operand) several items are necessary. They are:

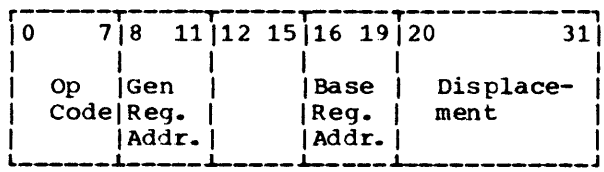
1. 8 bit Op Code

2. 4 bit General Register Address } Destination Operand Address

3. 4 bit Base Register Address } Source Operand Address

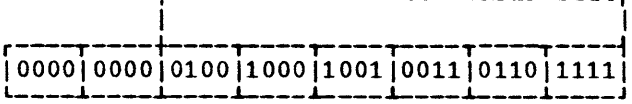
4. 12 bit Displacement }

The instruction format for this operation



Bits 12-15 of this instruction could be used for further modification of the main storage address. We will, however, ignore them for the present.

Given a displacement of 100110110010 and base-register 11 (whose contents are shown below), the effective storage binary-address is 010010001001110100100001.

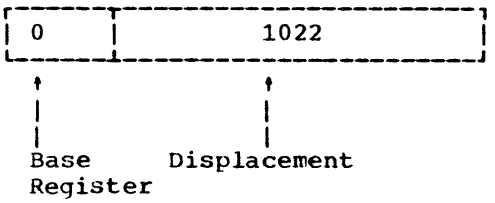


General Register 11 Contents

Remember that you add the 12 binary bits in the displacement to the low-order 24 binary bits of the base register.

The address generated by adding the displacement and base address is used for addressing main storage. The original instruction and the base register's contents remain unchanged.

As previously mentioned, only general registers 1-15 can be used as base registers. If general register 0 is specified as the base register, the base address is assumed to be zero, regardless of the contents of register 0.

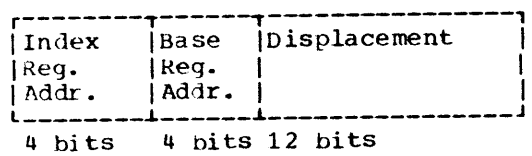


For this example, the contents (in decimal) of register 0 is 2048.

Given these address fields in the instruction and the contents of register 0, the effective storage address is 1022. Because register 0 is specified as the base register, a base address of 0 is used. The contents of register 0 are ignored.

All storage addresses are generated by using base and displacement. In some instructions, however, a third factor is used. The third factor is called the index value. It is contained in a general register.

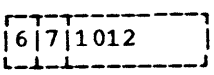
In those instructions that include an indexing factor, the address fields have the format:



The effective storage address is generated by adding:

1. Displacement,
2. Contents of the base register, and
3. Contents of the index register.

For example, suppose the address portion of an instruction is:



Register 6 contains the value 2048, and register 7 contains the value 6024:

1. The effective storage address is 9084.
2. The address portion of the instruction is unchanged.
3. The values in the base and index registers are unchanged.

The storage address is generated by adding the contents of the base register (6024) plus the contents of the index register (2048) to the displacement value given in the instruction (1012). The values in the specified registers and the displacement value in the instruction remain unchanged.

INSTRUCTION FIELDS

- There are five basic instruction formats: RR, RX, RS, SI, and SS.
- In most operations, the first operand is replaced by the contents of the second operand or by the results of the operation.
- The number in the length code (L) in the SS format is one less (when actual machine language is used) than the true length of the data field.

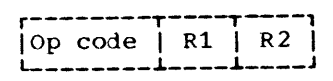
Instructions are 1, 2, or 3 half words long, depending on the locations of the operands.

RR FORMAT: A 1--half word instruction is used when each operand is in a general register or in a floating-point register.

An RR format instruction has:

1. An 8-bit op code.
2. A 4-bit register address for the first operand (destination).
3. A 4-bit register address for the second operand (source).

The RR (register-to-register) format is:



Bits 0 and 1 of the op code indicate the length of the instruction and the location of the operands. For the RR format, bits 0 and 1 are both at a 0 value.

The second byte of the RR format is divided into two fields: R1 and R2. The R1 field gives the register address of the first operand while the R2 field is the address of the second operand. The R suffix numbers in the address fields of the RR formats (and all other formats) indicate whether the operand is the first or second (and in some cases, the third) operand. For most operations, the results replace the first operand.

RX FORMAT: Instructions that are two halfwords in length have one of three different formats. As you recall, if bits 0 and 1 of the op code are either 01 or 10, the instruction is two halfwords in length. Furthermore if bits 0, 1 of the op code are set to 01, they indicate a specific format, known as the RX format:

Op Code	R1 Gen. Reg.	X2 Index Reg.	B2 Base Reg.	D2 Displacement

In the RX format, the effective address of the second operand is generated by adding the contents of the base register and the index register to the displacement. The RX format is used for storage to register operations. The destination register address is specified by the R1 field.

ADD	3	7	4	1024

For the preceding RX format instruction, the storage address is generated by adding the 24 low-order bits of the contents of registers 7 and 4 and the displacement value of 1024. The storage (source) operand is added to the contents of register 3 and the sum is placed in register 3.

RS FORMAT: Storage-to-register instructions in which the storage address does not include an indexing factor are called the RS format. The format is:

Op Code	R1	R3	B2	D2

The RS format (two half words long) is identified by a 10 in bits 0 and 1 of the op code. The R3 field in the RS format specifies the general register used for the third operand. (In some RS instructions, the R3 field is ignored.) An example of an instruction that uses the R3 field is Load Multiple. During execution of a Load Multiple instruction, data in main storage is loaded (or placed) into general registers. Loading begins with the register specified by the R1 field and continues consecutively until the register specified by the R3 field has been loaded. For example:

Op Code	4	7	0	0100

In this Load Multiple example, the effective storage address is 0100. This is because register 0 is specified as the base register (whose contents are ignored during address generation).

Here, registers 4 through 7 are loaded with the data from main storage. Because each register can hold one full word, registers 4-7 are loaded with the data in storage location 0100 through 0115 (decimal). (Each storage address is used to address one byte of data.)

SI FORMAT: An SI format instruction is also two halfwords in length. This format is used when one operand is in main storage and the other operand (called the immediate operand) is carried in the instruction itself. The SI format is also identified by a 10 in bits 0 and 1 of the op code, just like the RS format. The SI format is:

Op Code	I2	B1	D1

In the SI format, the storage operand is the first operand. Its effective address does not include an indexing factor. The immediate operand is fixed in length and is one byte long.

An SI format instruction example is Move Immediate. Execution of this instruction moves the immediate operand byte (I2) from the instruction to the storage location. The immediate operand remains unchanged in the instruction after completion of the operation. For example:

Op Code	I2	0	1000

In this Move Immediate instruction, the contents of the I2 field are placed in storage location 1000.

Because bits 0 and 1 of the op code have a value of 10 for both the RS and SI formats, the remaining bits of the op code indicate whether the instruction is in the RS or the SI format.

SS FORMAT: In the four previous formats, the operands are fixed-length. Variable-length operands are specified by the SS (storage-to-storage) format instructions:

Op Code	L	B1	D1	B2	D2

Length Code

Location of 1st Operand (destination)

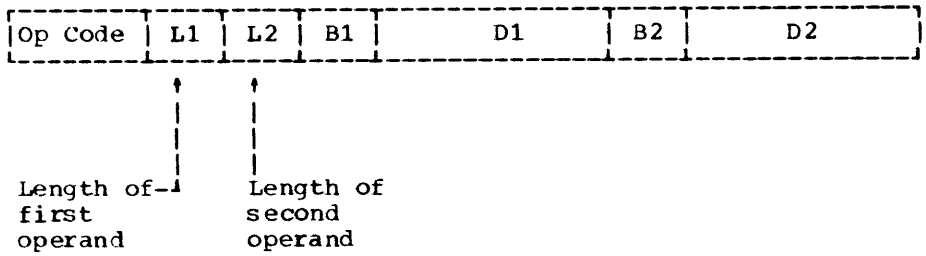
Location of 2nd Operand (source)

Because both operands are in storage, the SS format instruction is three halfwords long, and is identified when bits 0 and 1 of the op code contain 11.

In the SS format, an indexing factor is not included in the generation of storage addresses. The second byte of the SS format is the length code which consists of 8 binary bits. The maximum value that can be expressed with 8 binary bits is 255 (decimal).

Because all operands are at least one byte long, a length code is used to tell how many additional bytes are needed. For instance, a length code of 15 indicates that the operand is 16 bytes long. If an operand is one byte long, the length code is zero.

So far we have been treating the length code as one 8-bit binary number. However, we are dealing with two operands. Do they both have to be of the same length? The answer is: not always. The lengths depend on the particular operation. If we are concerned with moving a data field from one area of storage to another, we only need one length code. If, however, we are adding one storage field to another, then we need to know the length of both operands. For arithmetic SS operations, the length code is split in two:



With the length code split into two 4-bit fields, the maximum length of arithmetic variable-length operands is 16 bytes. The effective length of variable-length fields is one more than the length code.

INSTRUCTION SEQUENCING AND BRANCHING

- Unless otherwise specified, instructions are processed sequentially.
- Instructions are fetched from main storage during I-time and executed during E-time.

The instructions of the stored program are read out of main storage and then executed, one at a time. Each instruction is decoded in the control section of the Central Processing Unit (CPU).

After being decoded in the control section of the CPU, the instruction is executed. Arithmetical or logical operations are performed in ALU. During processing of every instruction, there are two periods of time. The time during which the instruction is read out (fetched) from main storage and interpreted is I-time (Instruction-time). The operation specified by the instruction is performed during E-time (Execution-time). Data is the name generally given to information read out of main storage during E-time. Instructions are read out of main storage during I-time. An instruction may be treated as data and changed if it is read out during E-time.

In the System/360 there is no clear division between I-time and E-time. That is, before the instruction has been completely read out and analyzed by the control section, some part of the execution may have already been started. But for simplicity, we can think of I-time as being separate from E-time.

The instructions of a stored program are generally read out and executed in a sequential manner. The sequential manner of instruction fetching and execution can be changed by instructions known as branch instructions.

Recall that instructions are generally thought of as having two basic parts. The op code of the instruction is used to tell the computer what to do (such as add or branch). The other portion of the instruction generally tells the computer where data is located. For this reason it is called the address portion.

An instruction may contain information other than data addresses. The address of the next instruction to be executed can be specified by a branch instruction. (In some instructions the data to be operated on can be contained in the instruction.)

Instruction Address Field in PSW

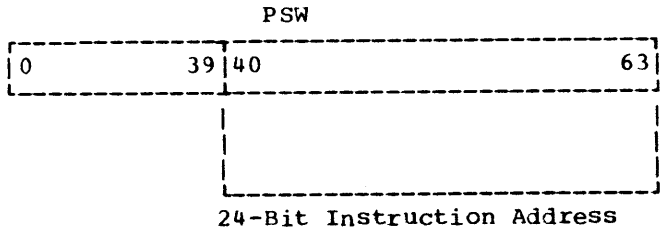
- The Program Status Word (PSW) is a double-word containing 8 bytes (64 bits) of control and status information.
- The current PSW is maintained in machine circuitry.
- The address of the next sequential instruction to be fetched from main storage is contained in bits 40-63 (24 bits) of the PSW.

In the System/360 there is a doubleword of information used to indicate the status of the program as well as to control the program. This doubleword is called the Program Status Word (PSW). As in all doublewords, the bits of the PSW are numbered 0 to 63, from left to right. The PSW includes status information such as:

1. The location of the next instruction.
2. Whether an arithmetic operation has resulted in a positive or negative answer. (Possibly the operation ended with a zero balance or an overflow.)

The current PSW reflects the status and controls the program currently being executed. The current PSW is not stored in any of the 16 general registers or addressable locations in main storage. It is kept in some internal areas of the System/360 that are not addressable by the program. Although the current PSW may be scattered throughout the CPU, it is considered as one double word of information.

The location of the next instruction to be fetched from main storage is indicated by bits 40-63 of the PSW.



The instruction address portion of the current PSW is updated for each instruction that is fetched and executed. That is, if an RR type instruction is fetched from location 1000, the instruction address portion of the current PSW is updated. The location of the next sequential instruction is 1002 because an RR-format instruction is one halfword (two bytes) long. Thus the instruction address portion of the PSW is updated to 1002.

After the RR type instruction at location 1000 has been executed, the instruction address portion of the PSW (which contains 1002) is used to fetch the next instruction. If the instruction at location 1002 is the RX (two halfwords long) type, the instruction address portion of the current PSW is then changed to 1006.

Because instruction length is always a multiple of halfwords, the instruction address portion of the current PSW is updated by some multiple of two (except after execution of a branch). The instruction address in the current PSW is increased by 2, 4, or 6 depending on bits 0 and 1 of the current instruction's op code. For example, if bits 0 and 1 of the current instruction's op code contain 11, the instruction address in the current PSW is increased by 6.

Instruction Branching

- A branch instruction is used to make program decisions.
- A branch instruction provides a way to leave one instruction sequence and branch to another instruction sequence.
- The instruction address field of the current PSW is changed to the branch-to-address when the program branches.

Decision blocks in a program flow chart are represented by a diamond shaped symbol. The use of this symbol in a program represents a decision as to what instruction to use next. Should the program continue with its present sequence of instructions, or should it branch out to another sequence of instructions? Sometimes the program is trying to decide which of two or more new sequences to branch to.

As you know, the instruction address portion of the current PSW is used to fetch the next sequential instruction. However, whenever a branch is executed, the contents of the instruction address portion of the current PSW are replaced by the address of the instruction being branched to.

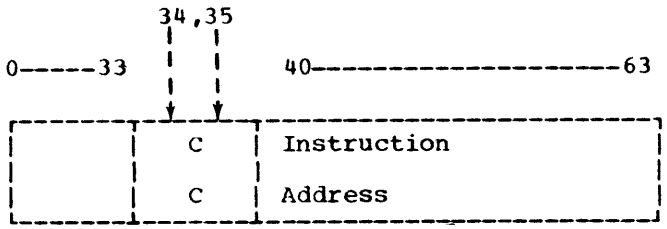
For example, if an RX instruction (at location 1000) is fetched, the instruction address portion of the current PSW is normally changed to 1004. If however, the instruction at 1000 says to branch to location 2000, the instruction address portion of the current PSW is changed to 2000.

Here, bits 40-63 of the current PSW might be updated to 1004 and then changed to 2000. The action depends on the particular branch instruction used. However, at the time of the branch, the address of the branch-to location is placed in bits 40 to 63 (instruction address portion) of the current PSW.

Condition Code Field

- The condition code occupies bits 34 and 35 of the current PSW.
- The 4 combinations of the condition code are 00, 01, 10 and 11.
- The condition code indicates the results of certain instructions (such as add, subtract, or compare).
- Some instructions do not affect the condition code.

The condition code is located in bits 34 and 35 of the PSW.



Condition Code

The condition code can have any one of four bit combinations:

1) 00
2) 01
3) 10
4) 11

The condition code is set to one of its four combinations after an instruction has been executed. Then it is placed in the condition code portion of the current PSW. Not all instructions affect the condition code.

One of the uses of the condition code is to indicate the result of arithmetic operations, such as add or subtract. There are four possible results of an algebraic add or subtract:

1) Positive number,
2) Negative number,
3) Zero balance, or
4) An overflow.

The condition code reflects the results with these settings:

Condition Code	Arithmetic Results
00	zero balance
01	< zero (negative)
10	> zero (positive)
11	overflow

The condition code is set at the end of algebraic add or subtract operations (either decimal or binary). The condition code retains its setting until the end of the next instruction that can change it.

Another use of the condition code is to indicate the result of a compare operation. A compare operation consists of comparing the first operand to the second operand. The condition code is set to indicate the result. Neither operand is changed. The condition code is set and indicates whether the first operand is equal to, less than, or greater than the seccnd operand, as follows:

Condition Code	Comparisen
00	equal
01	low
10	high

Note that a condition code setting of 11 is not possible after a compare operation. Note also that the condition code is used to indicate more than just the result of an algebraic or comparison operation. The actual meaning of the ccndition code depends on the results of the operation that caused it to be set.

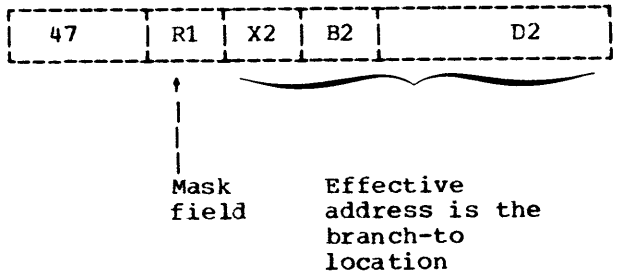
Condition Code Branching

- The instruction that tests the condition code is called Branch on Condition.
- Branch on Condition can have either the RX or RR format.
- The R1 field is used as the mask field to test for a specific setting of the condition code (one bit set in mask field) or a multiple condition code setting (two or more bits set in mask field).
- A mask field of 0000 results in a NO-OP instruction.
- A mask field of 1111 results in an unconditional branch instruction.

One of the instructions of the System/360 is an instruction called Branch-on-Condition. This instruction causes the system to examine the condition code and branch if the condition code setting matches that of a code in the Branch-on-Condition instruction.

The Branch-on-Condition instruction can be either in the RR or the RX format. In either case, the R1 field is coded so that the condition code can be tested. The Branch on Condition (RR format) instruction is:

The branch-to address is in the general register specified by the R2 field. The RX format is:



The R1 field in the Branch-on-Condition instruction is referred to as the mask field. The condition code is tested by being matched against the mask field.

The mask field is tested against the condition code according to the following chart:

Mask Field	Condition Code
1000	00
0100	01
0010	10
0001	11

Any of the possible condition code settings can be tested by setting the appropriate bit of the mask field. If bits 8-11 of a Branch on Condition instruction contain 1000, a branch occurs only if the condition code has a setting of 00. If the condition code is 01 and the mask field is 0010, a branch does not occur.

Sometimes the four possible settings of the condition code are referred to as decimal digits:

Condition Code	Decimal Equivalent
00	0
01	1
10	2
11	3

The bits of the Branch-on-Condition instruction's mask field correspond to the condition code settings in a left to right fashion.

To test for a specific condition code setting, the corresponding bit of the mask field must contain a 1.

If the mask field contains 0000, none of the possible condition code settings can cause a match and a branch can not occur.

If the mask field contains 1111, all or any of the possible condition code settings match corresponding mask bits. Because the condition code always contains at least one of the four possible settings, a mask field of 1111 always results in a branch.

In summary, the branch on condition instruction:

a. Can be used as a NO-OP instruction, when its mask field is 0000.

b. Can test for a specific result (such as an equal compare) when one of the bits of the mask field is set on.

c. Can test for a multiple result (such as an equal or low compare) when two or more bits of the mask field are set on.

d. Can be used as an uncoditional branch when its mask field is 1111.

SYSTEM/360 AND INTERRUPTIONS

Supervisor Concepts

- Control programs perform such functions as program loading, storage protection assigning, I/O operation handling, interruption handling, job flow handling, and operator communications handling.
- One control program (the supervisor), in general, remains in core storage at all times.
- Basic functions of the supervisor program are I/O control and interruption handling.

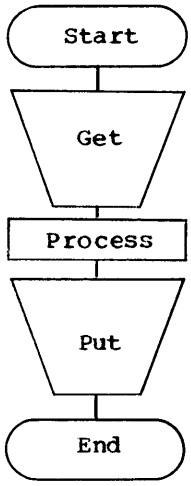
A program is a sequence of instructions designed to solve a problem. For example, a payroll problem program could:

1. Get an employee's record,
2. Calculate gross and net pay, and
3. Put the results out in the form of a pay check.

The payroll program then gets the next employee's record and repeats the process. This sequence of instructions continues until all employee's records are processed. Admittedly, this is a simplification of a payroll problem. Most programs, however, are similar to this payroll example in that they can be broken down into the three operations:

1. Get record.
2. Process record, and
3. Put record into an output file.

These problem solving programs are referred to as problem programs.

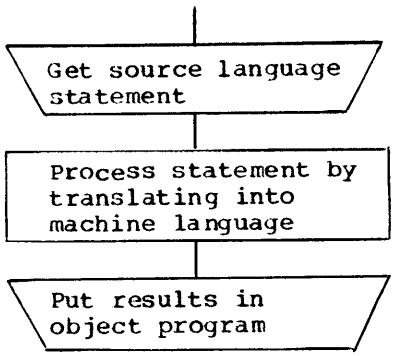


Problem Program Logic

Another example of a Problem Program is an assembly program. Here the problem is different, but the three basic operations are the same. The problem consists of:

1. Getting a symbolic (source language) statement,
2. Processing it by translating the statement into machine language, and

3. Putting the results in the output file (object program).



During recent years, data processing machines have been developed with faster and faster internal processing speeds. As a result, the execution times for these problem programs has been continually reduced, but with no corresponding reduction in the time it takes for an operator to load-in the next problem program and manually set-up input data. In some data processing installations, the average "set up" time is about equal to the average "execution" time. In other words, the data processing system is idle about half the time, while the operator is "setting up" for the next problem program. Clearly this is an inefficient way to control an installation.

In an attempt to reduce this idle time and keep the system running, programmers began to use stored programs to control the execution of problem programs. These programs are called Control Programs. (Other names used are "Monitors" and "Supervisors".) These Control Programs were at first written only for the requirements of a particular installation. Later, as the similarities between control programs became obvious, IBM began to supply generalized control programs which could be tailored to the requirements of each installation.

The simplest type of control program is used to supervise the loading of problem programs; it might operate in the following manner:

1. An input tape is prepared. This tape contains the problem programs and associated data (Figure 1-20).

2. The operator loads the control program into main storage from a second tape.

3. The control program loads the first problem program and then passes control (via a branch) 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 does not issue a halt instruction. Instead it passes control (by branching) 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 problem programs have been executed.

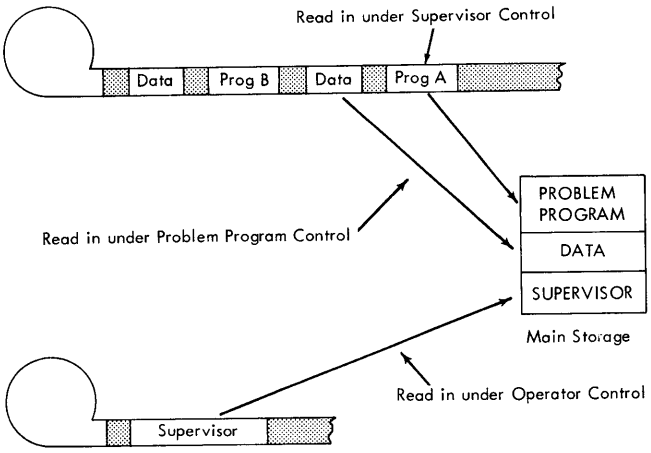


Figure 1-20. Program-Loading Control Program

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

1. The system never halted between jobs.

2. The control program remained in main storage during problem program execution.

3. The control program served as a link between jobs. Its only function was to bring in a new problem program as each job was finished.

4. The problem programs handled their own input-output operations (Figure 1-21).

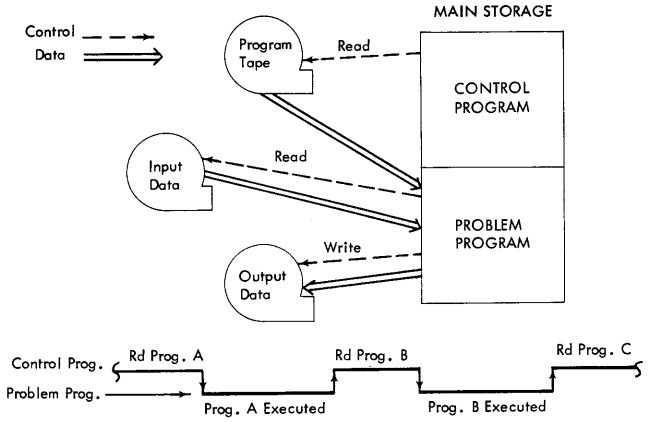


Figure 1-21. I/O Operations Handled by Problem Program

This is one example of the use of a somewhat limited control program. Here, the entire control program is in main storage. Other functions, however, can be included as part of a control program. One such function is the initiation of input-output operations. The problem program is mainly interested in processing data. The actual read and write operations necessary to transfer data between input-output devices and main storage can be handled by the control program (Figure 1-22).

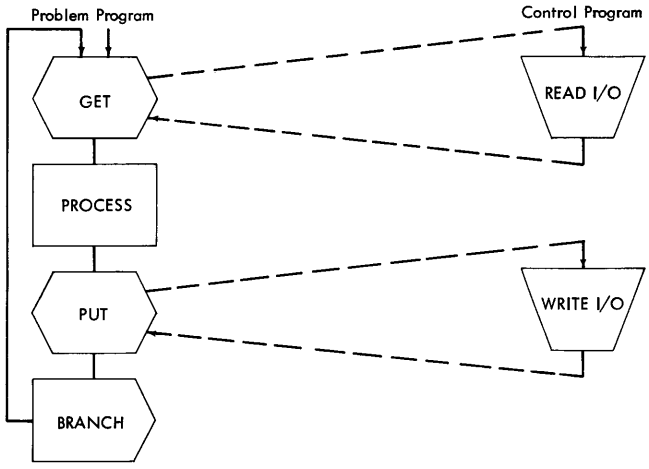


Figure 1-22. Control Program Handling I/O

In this I/O handling function of a control program, control passes back and forth between the problem and control program during the execution of the problem program.

In the first control program example, the only time the control program was in control was between jobs. Now, however, the control program not only reads in new problem programs, but it also (during the execution of the problem program) is used to start the necessary I/O units for input-output data (Figure 1-23).

The control program can be given other functions as well. Of course, the more functions that a control program has, the more main storage space it requires, thereby leaving less available storage for problem programs. This problem is solved, to some degree, by placing those sections of the control program that are used infrequently on a high speed fast access I/O device, such as a disk storage unit. Only those sections that are necessary to supervise the running of problem programs are kept in main storage. The portion of the control program that resides in main storage is known as the Supervisor. The supervisor program calls in (from disk to main storage) other sections of the control program when necessary.

Control programs have come into general acceptance because of the need to reduce machine idle time and manual intervention and to increase the overall efficiency of a data processing installation.

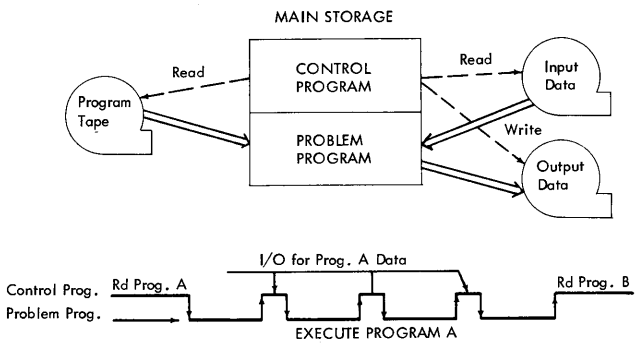


Figure 1-23. Control Program Sequencing

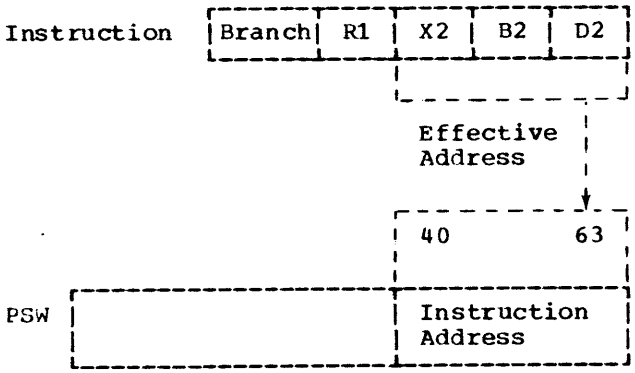
Interruptions and the PSW

- An interruption terminates the current sequence of instructions and causes a machine forced-branch to the supervisor program.
- An interruption results in storing of the current PSW in main storage, and fetching of a new PSW from main storage.
- Processing resumes at the instruction address specified by the instruction address portion of the new PSW, which is now the current PSW.
- There are five classes of interruptions. Each has both an old and a new PSW location in main storage.

Because there is no halt instruction in System/360, a problem program, when finished, must be able to branch into the supervisor so that a new problem program can be loaded. Also when a machine or program check occurs, an automatic branch to the supervisor is usually desired.

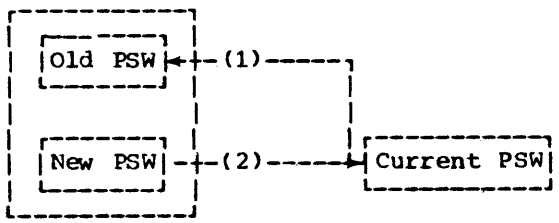
These automatic branches into the supervior are called Interruptions. That is, the current sequence of instructions is interrupted and an automatic branch is taken to a new sequence of instructions. Both machine checks and program checks can cause automatic branches or interruptions. Also, when a problem program is finished, it signals the supervisor via an interruption.

An interruption is similar to a branch. However, it does much more than a branch instruction. A branch instruction can only cause the instruction address portion of the current PSW to be replaced.



When an interruption occurs:

1. The current PSW is placed in main storage where it is called the old PSW, and
2. A new PSW is brought out of main storage, and it becomes the current PSW.



Main Storage

Assuming that the instruction address portion of the new PSW contains 1096, the first instruction processed after the interruption is at address 1096.

There are five distinct classes of interruptions:

1.	External	Can be caused by pressing the Interrupt key on the system ccnsole.
2.	Supervisor	Caused by the Supervisor Call instruction.
3.	Program	Caused by a program check.
4.	Machine	Caused by a machine check.
5.	I/O	Caused by an Input/Output operation.

Each of the five classes of interruptions has its own main storage locations for new and old PSW's as follows (decimal notation used):

Interruption	Old PSW	New PSW
External	0024	0088
Supervisor	0032	0096
Program	0040	0104
Machine	0048	0112
I/O	0056	0120

For example, a machine check causes the current PSW to be placed in location 0048 and a new PSW to be brought out from loca-

tion 0112. Notice that these locations are all divisible by eight because PSW's are doublewords, and must reside on doubleword boundaries. (It is interesting to note that each new PSW is located 64 storage locations higher than the corresponding old PSW.)

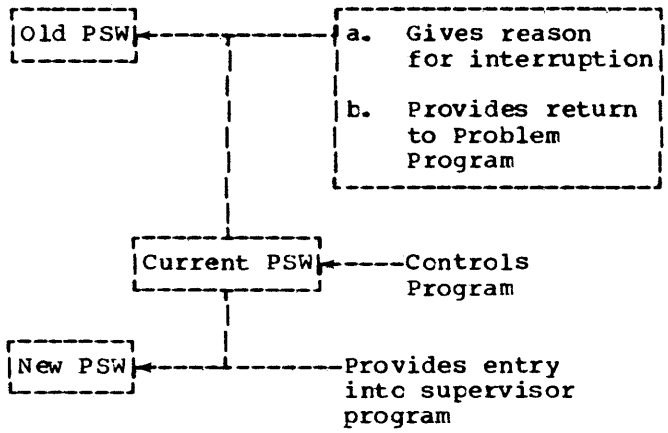
Although an interruption may be initiated by an instruction (such as when the supervisor call instruction initiates a supervisor interruption), the actual storing and loading of the PSW is done automatically by circuitry.

Interruptions occur only at the end of an instruction and never in the middle of one. The current instruction is completed before an I/O, external, or supervisor call interruption is taken. In the case of program and machine interruptions (which indicate programming and circuit errors, respectively), the interruption still occurs at the end of execution of the instruction. However, in these two cases, the end may be forced by:

1. Suppressing the instruction's execution when a programming error is detected during instruction fetch time, or

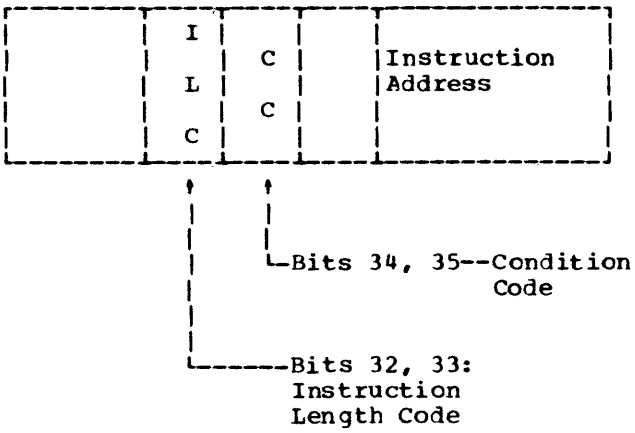
2. Terminating its execution when a programming or machine error is detected during execution time (Figure 1-24).

The branch is effected automatically by internal circuitry. The current PSW is placed in a fixed location in main storage and becomes the old PSW. The old PSW gives the specific reason for the interruption and also provides a return to the interrupted program. A new PSW is fetched from a fixed location in main storage and becomes the current PSW. The new PSW provides an entry into the correct routine in the supervisor program.



PSW INSTRUCTION LENGTH FIELD: Once an instruction has been read out of main storage, the instruction address portion of the PSW is updated and specifies the next instruction's address. Interruptions can occur 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. Instead, it contains the address of the next instruction that would have been executed if the interruption had not occurred.

When the interruption is completed, the supervisor may elect to return to the point of departure from the problem program. It does this by examining the old PSW. In some cases, the problem program instruction, performed just before the interruption, may have to be performed again. Because the instruction address portion of the old PSW is updated before the interruption occurs, and because instruction lengths vary, the supervisor needs additional information to derive the instruction address. This additional information is contained in bits 32 and 33 of the old PSW, and is called the instruction length code.



When the supervisor retrieves the old PSW to determine where to re-enter the problem program, the instruction length code indicates what value must be subtracted from the old PSW instruction address field to produce the address of the op code of the last instruction executed before the interruption occurred. The instruction length code is valid only for certain types of interruptions. The supervisor program must determine if this information is to be used.

Bits 32-33 of the PSW are set to 01, 10, or 11 (depending of the length of the instruction) before the current PSW is stored as the old PSW.

PSW Bits 32-33	Instruction Length
01	1 Halfword
10	2 Halfwords
11	3 Halfwords

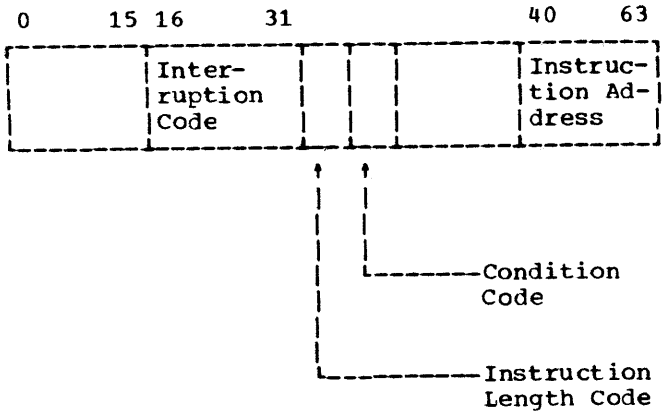
For example, the instruction length code in the PSW is set to 10 (2) for an RX format instruction.

If the instruction address portion of the old PSW contains 4000 (decimal) and the instruction length code contains 11, the op code of the last instruction prior to the interrupt is located at 3994 (decimal).

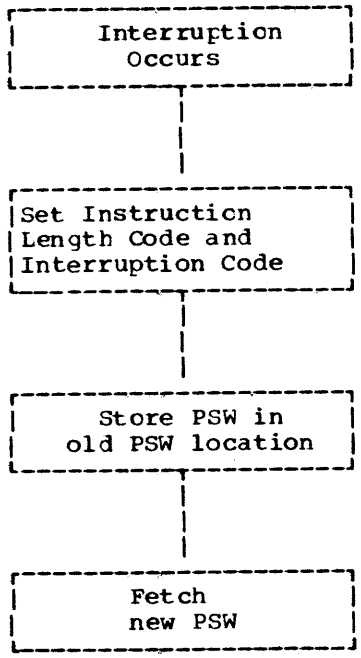
A program routine must be provided for each of the five classes of interruptions. Each of these interruption handling routines process the interruptions in a different way. It is not always important to be able to determine the last instruction executed before the interruption. In the case of program, machine, or supervisor interruptions, an instruction in the problem program caused the interruption.

In the case of external and I/O interruptions, the problem program did not cause the interruption. As a result, it is unimportant to the supervisor program what instruction was executed last in the problem program. After the interruption routine is completed, the next sequential problem program instruction (address in old PSW) is processed.

PSW INTERRUPTION CODE FIELD: Another field in the PSW that is of value to the supervisor program is the interruption code field. It is held in bits 16-31 of the PSW.



When an interruption occurs, the current PSW is stored in one of five locations reserved for old PSW's. It is at this time that the interruption code of the current PSW is set.



The five classes of interruptions tell the supervisor only the general reason for the interruption. For instance, if the new PSW is brought out of location 0040, then 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 in the PSW. By examining the interruption code in bits 16-31 of the old PSW, the program check routine in the supervisor can tell specifically whether it was a specification, addressing, or some other program exception. In the case of I/O interruptions, the interruption code carries the address of the channel and I/O unit that caused the I/O interruption. (Figure 1-24).

For example, when a program interruption is caused by a fixed-point overflow, the interruption code of the old PSW contains 0000000000001000. (Refer to Figure 1-24.)

For brevity's sake, the interruption code is often represented as 4 hexadecimal digits:

Binary	Hexadecimal
0000000000001000	0008

There are five old PSW's in main storage. How does the supervisor know which one to use? The answer is, that each of five new PSW's point to different routines in the supervisor. These routines in turn use the old PSW location that corresponds to the particular class of interruption. For example, the program check routine in the supervisor uses the old PSW at location 0040, while the supervisor call routine uses the old PSW at location 0032.

Interruption	Old PSW	New PSW
External	0024	0088
Supervisor	0032	0096
Program	0040	0104
Machine	0048	0112
I/O	0056	0120

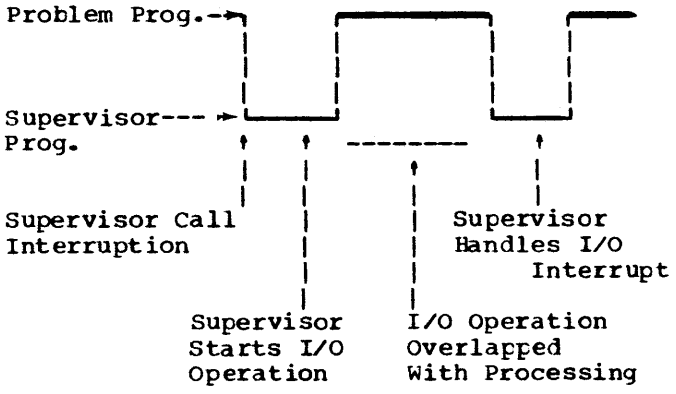
In the case of an interruption caused by a machine check, the PSW that was controlling the program prior to the interruption is stored automatically in location 0048. Then the doubleword at location 0112 is brought out and becomes the current PSW. This PSW directs the system to that area of the supervisor program that handles machine checks. The machine check handling routine of the supervisor is written so that the doubleword at location 0048 is processed as the old PSW.

In the case of an interruption caused by a program check, the current PSW at the time the interruption occurs is stored automatically as the old PSW at location 0040. Then the doubleword at location 0104 is brought out and becomes the current PSW. This PSW directs the system to the supervisor routine that handles program checks. The program check handling routine of the supervisor is written so that the doubleword at location 0040 is processed as the old PSW.

In the case of an interruption caused by the Supervisor Call instruction, the current PSW is stored in location 0032. Then the doubleword at location 0096 is brought out and becomes the current PSW. This PSW directs the system to that portion of the supervisor that handles supervisor calls. One way a problem program could notify the supervisor that the program is finished is to issue a Supervisor Call instruction. Thus the last instruction of a problem program would probably be a Supervisor Call instruction.

If the Interrupt key on the system console is pressed, an external interruption occurs. In this case, the current PSW is automatically stored at location 0024. For an external interruption, the doubleword at location 0088 is brought out and becomes the current PSW.

I/O interruptions generally occur at the end of an I/O operation. Most I/O operations are overlapped with processing. The I/O interruption signals the supervisor that the I/O operation is finished. An I/O interruption causes the current PSW to be stored at location 0056. The new PSW at location 0120 is brought out and becomes the current PSW. This PSW directs the system to that section of the supervisor program that handles I/O interruptions.



Interruption Source Identification	Interruption Code PSW Bits 16-31		Mask Bits	ILC Set	Instruction Execution
Input/Output (Old PSW 56, New PSW 120)					
Multiplex Channel	00000000	aaaaaaaa	0	x	Complete
Selector Channel 1	00000001	aaaaaaaa	1	x	Complete
Selector Channel 2	00000010	aaaaaaaa	2	x	Complete
Selector Channel 3	00000011	aaaaaaaa	3	x	Complete
Selector Channel 4	00000100	aaaaaaaa	4	x	Complete
Selector Channel 5	00000101	aaaaaaaa	5	x	Complete
Selector Channel 6	00000110	aaaaaaaa	6	x	Complete
Program (Old PSW 40, New PSW 104)					
Operation	00000000	00000001		1,2,3	Suppress
Privileged Operation	00000000	00000010		1,2	Suppress
Execute	00000000	00000011		2	Suppress
Protection	00000000	00000100		0,2,3	Suppress/Terminate
Addressing	00000000	00000101		0,2,3	Suppress/Terminate
Specification	00000000	00000110		1,2,3	Suppress
Data	00000000	00000111		2,3	Terminate
Fixed-Point Overflow	00000000	00001000	36	1,2	Complete
Fixed-Point Divide	00000000	00001001		1,2	Suppress/Complete
Decimal Overflow	00000000	00001010	37	3	Complete
Decimal Divide	00000000	00001011		3	Suppress
Exponent Overflow	00000000	00001100		1,2	Terminate
Exponent Underflow	00000000	00001101	38	1,2	Complete
Significance	00000000	00001110	39	1,2	Complete
Floating-Point Divide	00000000	00001111		1,2	Complete
Supervisor Call (Old PSW 32, New PSW 96)					
Instruction Bits	00000000	rrrrrrrr		1	Complete
External (Old PSW 24, New PSW 88)					
External Signal 1	00000000	xxxxxxx1	7	x	Complete
External Signal 2	00000000	xxxxxx1x	7	x	Complete
External Signal 3	00000000	xxxxx1xx	7	x	Complete
External Signal 4	00000000	xxxx1xxx	7	x	Complete
External Signal 5	00000000	xxx1xxxx	7	x	Complete
External Signal 6	00000000	xx1xxxxx	7	x	Complete
Interrupt Key	00000000	x1xxxxxx	7	x	Complete
Timer	00000000	1xxxxxxx	7	x	Complete
Machine Check (Old PSW 48, New PSW 112)					
Machine Malfunction	00000000	00000000	13	x	Terminate

Notes: a = I/O Device Address
r = Bits 8-15 of Supervisor Call Instruction
x = Unpredictable

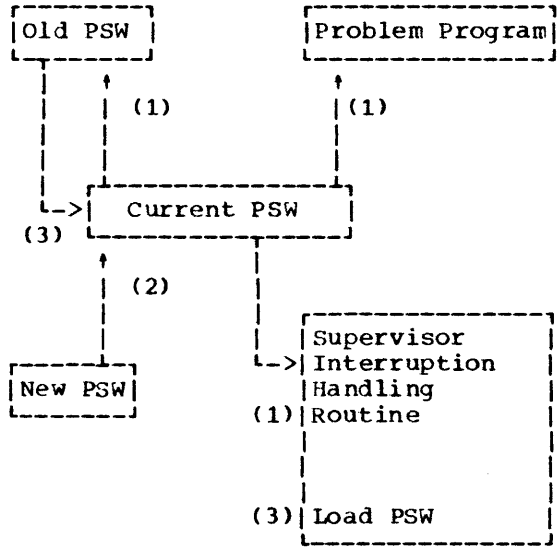
Figure 1-24. Interruption Code and Action Chart

Load PSW Instruction

- The Load PSW instruction is used to return to the problem program after an interruption.
- The Load PSW instruction is in the SI format. The I2 field is ignored.
- A doubleword is loaded into current PSW circuitry (from locations in main storage) by the Load PSW instruction.

After the end of the I/O interruption routine in the supervisor, it is desirable to return to processing the problem program. Simply branching back to the problem program would not be desirable. A branch instruction only affects the instruction address portion of the PSW. Other parts of the PSW are also important in controlling the processing of a program. For one thing, the condition code setting in the controlling PSW for the I/O interruption routine would not necessarily be the same as it was before the I/O interruption occurred. It would be best to be able to give control back to the problem program with the same PSW the problem program was using when the I/O interruption occurred.

This can be done in the System/360 with an instruction known as Load PSW. This instruction is used by the supervisor to load the old PSW back in the system's control section. This is the last instruction in the supervisor's interruption handling routine. Note that this return (by replacing the PSW) to the problem program is done by means of an instruction (load PSW) and is not automatic, as is an interruption.

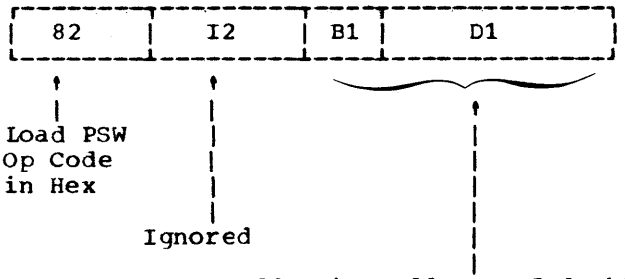


As can be seen from the preceding diagram, interruption action is as follows:

1. At the time of the interruption, the current PSW (which is controlling the problem program) is stored in the old PSW location. This is done automatically by machine circuits. The old PSW interruption code gives the reason for the interruption. The instruction address portion of the old PSW indicates the point at which the problem program was left.
2. A new PSW is then brought out of storage and becomes the current PSW. This new PSW points to the first instruction of the interruption handling routine which is part of the supervisor program.
3. After the interruption has been taken care of, the last instruction of the interruption handling routine is Load PSW. Processing this instruction causes the old PSW to become the current PSW, and a return is made to the problem program.

The Load PSW instruction is of the SI format:

In the Load PSW instruction, the I2 field is ignored.



The Load PSW instruction can be used by a supervisor program to change the current

PSW. The main use of the Load PSW instruction is to return to the problem program after an I/O, supervisor call, or external interruption has been serviced. It could also be used to load the PSW for a new problem program after the new program has been read into the machine by the supervisor program.

To return to a problem program after an I/O interruption has been serviced, the effective address generated by the B1 and D1 fields of a Load PSW instruction should be 0056 (38 in hexadecimal). Refer to Figure 1-24.

Supervisor Call Instruction

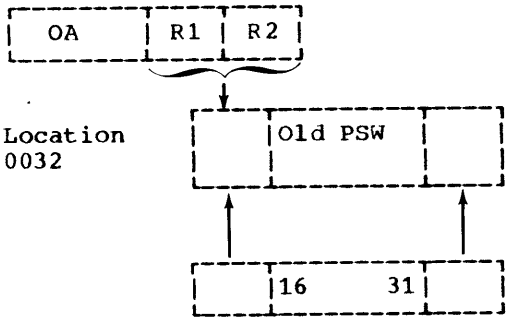
- The Supervisor Call instruction (RR format) is used by the problem program to pass control to the supervisor program by causing a supervisor call interruption.
- The R1 and R2 fields of a Supervisor Call instruction are placed in the interruption code field of the supervisor call old PSW.

The supervisor call interruption is used by the problem program to pass control to the supervisor program. There are a number of reasons why the problem program might want to call the supervisor program. Two of the major reasons are:

1. To tell the supervisor program that the problem program is done. The supervisor might then read in a new problem program and load its PSW.
2. To request the supervisor program to start an I/O operation for the problem program.

The Supervisor Call instruction is of the RR format:

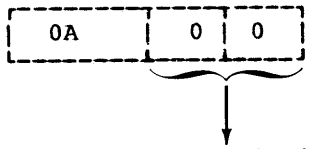
The Supervisor Call instruction causes a supervisor call interruption. The eight bits of the R1 and R2 fields are placed in the interruption code of the old PSW.



Current PSW

Location 0096 New PSW

Because the bits of the R1 and R2 fields are stored as the interruption code, they can tell the supervisor program the reason for the interruption. Resulting actions vary, depending on who wrote the supervisor program. For instance:



Supervisor Call Instruction

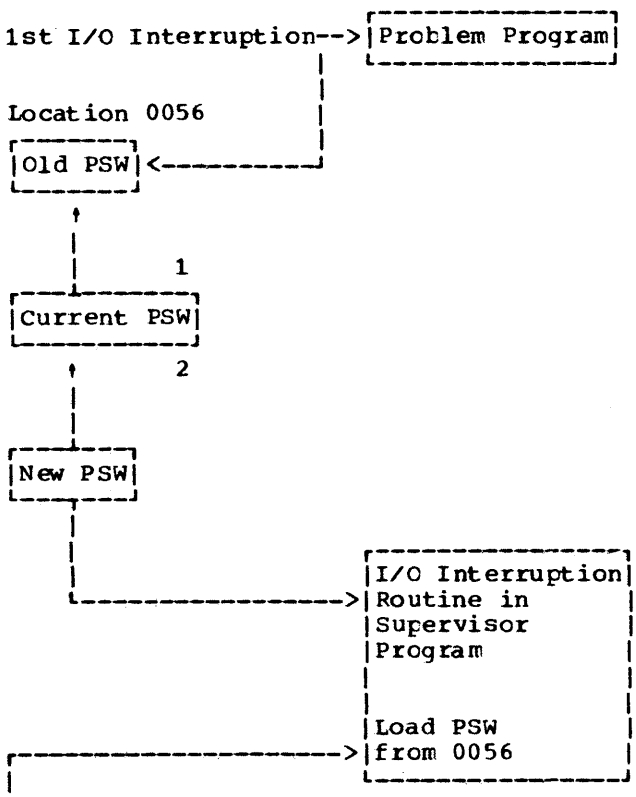
This interruption code might be used to signal a supervisor program that the problem program is finished.

Given the following supervisor call instruction (in hex), the binary bit structure that would be place in the interruption code of the old PSW is 11010000 (bits 16-31 of the old PSW in location 0032):

Masking Interruptions

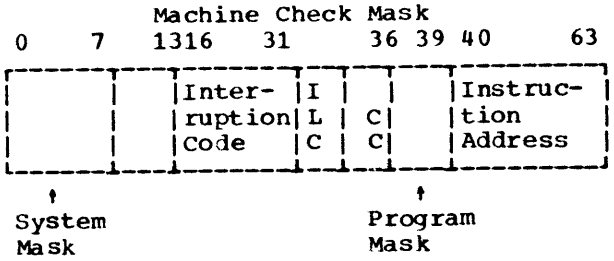
- Some interruptions are prevented from occuring by mask bits in the current PSW.
- The system mask is in bits 0 to 7 of the PSW. These bits, when zero, prevent (mask) external and I/O interruptions.
- The machine check mask is bit 13 of the PSW. When off, it allows machine errors to be ignored. Machine checks are not normally masked off except as a diagnostic aid.
- The program mask is stored in bits 36 to 39 of the PSW. When zero, these bits are used to prevent 4 of the 16 program checks from causing interruptions.
- Eleven program check interruptions and the supervisor call interruption cannot be masked off.

Sometimes, it is not desirable to allow an interruption. Consider, for example, an I/O interruption. In the System/360 it is possible to have simultaneous I/O operations on two or more channels. When one operation is completed, an I/O interruption usually occurs. The PSW is stored to give the supervisor program the reason (which I/O unit) for the interruption. This old PSW also gives the supervisor program a way in which to return to the interrupted problem program. If another I/O interruption is allowed before the first one has been completely handled, the old PSW (from the problem program) is lost. The supervisor program would then not be able to return to the problem program via the Load PSW instruction.



If a second I/O interruption were allowed to occur before the Load PSW instruction is executed, the current PSW at this point would be stored in location 0056. This would cause the old PSW (also in location 0056) from the problem program to be destroyed.

SYSTEM MASK: How does the supervisor program prevent this second undesirable I/O interruption until it has processed the first one? It does this by proper use of mask bits in the PSW.



Notice that:

1. Bits 0-7 are the system mask bits.

2. Bit 13 is the machine check mask bit.

3. Bits 36-39 are the program mask bits.

When any one of these mask bits is set to zero, the corresponding interruption is masked or prevented. Let's first consider the system mask bits. These eight bits can be used selectively or collectively to mask I/O and external interruptions as follows:

PSW Bit	Masks Interruption from
0	Multiplexor Channel
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	External

(Note that the only system mask channel-bits applicable in a Model 30 are bits 0, 1, and 2.)

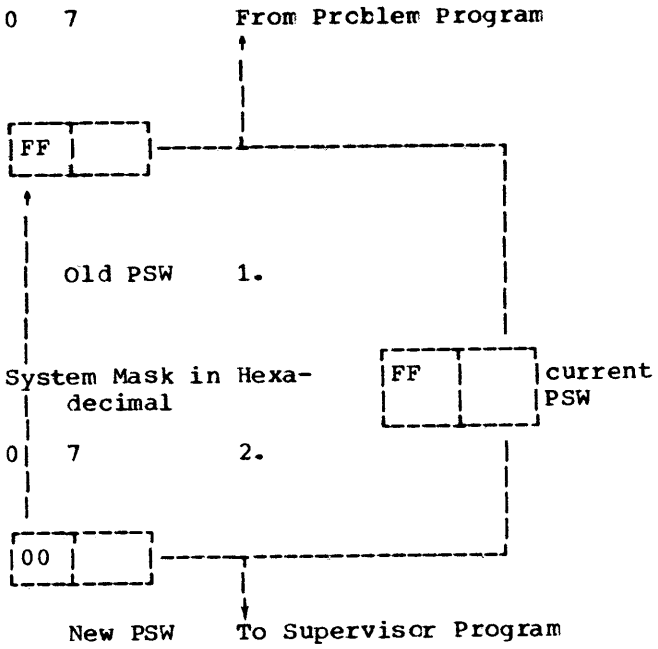
To prevent (mask) all I/O and external interruptions, bits 0-7 of the current PSW must contain zeros.

Notice that there is only one I/O interruption. However, each of the six selector channels and the multiplexor channel can be selectively prevented from causing an I/O interruption.

A system mask of 00111110 masks some I/O and all external interrupts. A system mask of 10000001 prevents I/O interruptions by all selector channels.

The system mask that determines whether or not to prevent any I/O or external interruptions is in the current PSW. In the case of an I/O interruption, the address of the device and channel causing the interruption is stored in the interruption code of the old PSW.

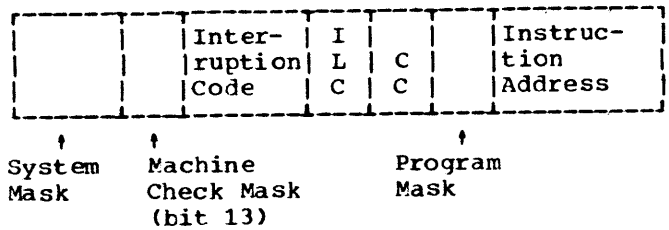
To prevent a second I/O interruption before a first one has been completely processed, the system mask of the new PSW should contain zeros.



One more point should be made concerning the system mask. When it contains zeros, I/O and external interruptions remain pending. As soon as the system mask is set to ones, another interruption can be taken.

The last instruction in the I/O interruption routine of the supervisor program is Load PSW. The old PSW in main storage location 056 (decimal) is brought out and becomes the current PSW. Once this is done, I/O interruptions can once more occur because the system mask of the problem program's PSW probably contains all ones (FF). Of course, a system mask of all ones allows not only I/O interruptions but also external interruptions.

MACHINE CHECK MASK:



A machine check interruption can be masked by means of bit 13 of the PSW. If this bit contains a zero, machine checks are ignored, and no machine interruption can occur. Of course, this is not the usual state of the machine check mask bit. It is usually set to one, so that any machine check will cause an interruption. The Check Control switch, on the system console, when set to the stop position, causes an error stop (even if PSW bit 13 is

set to zero) rather than an interruption when a machine check occurs. The usual mode of operation is to have this switch set to the process position and PSW bit 13 set to one. This means that when a machine check (such as even parity) occurs, an error stop does not occur. Instead an interruption occurs.

In summary then, there are three basic courses of action when a machine check occurs:

1. A machine check interruption (the PSW is stored in location 0048 and a new PSW is fetched from location 0112.)

2. An error halt when the Check Control switch is set to the stop position.

3. The check is ignored if PSW bit 13 is zero and the Check Control switch is set to the process position.

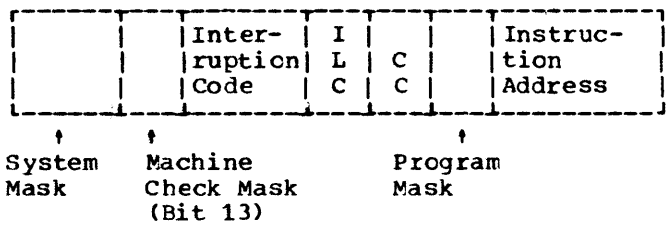
Depending upon the settings of switches on the system console, other actions can occur when a machine check occurs. These actions are described in Appendix B of this publication.

There is one other item of information concerning machine checks. It is called log-out. Unless the machine check is being ignored, information concerning the status of internal circuitry is automatically placed in storage starting at machine location 0128 (decimal). This log-out occurs prior to loading of the machine new PSW that is used to control the program error handling routine.

Just how much information is contained in a log-out and what it means depends on the particular model of System/360. However, log-out always occurs prior to a machine interruption. This log-out information reflects the status of the machine's internal circuitry. As such, it is meaningful only to someone who has a knowledge of the machine's internal circuitry.

For a Model 30, the maximum log-out area used includes main storage locations 128, 129, 130, 131, 133, 134, 135, 137, 138, and 139 (decimal).

PROGRAM MASK: Program checks (such as a specification exception) can also cause an interruption. On a program interruption, the PSW is stored in location 0040 and a new PSW is fetched from location 0104. Certain program interruptions can be masked off by use of bits 36-39 of the PSW.

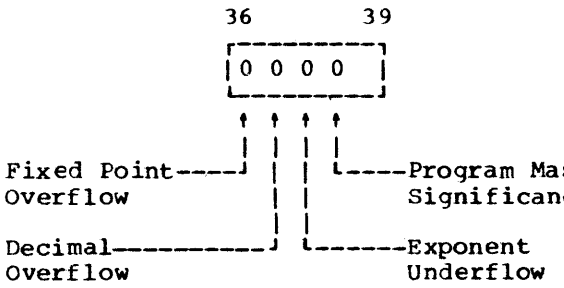


There are 15 possible exceptions that can cause a program check (see Figure 1-24). On occasion, four of these may not be considered as program checks. The four exceptions are:

1. Fixed-Point Overflow
2. Decimal Overflow
3. Exponent Underflow
4. Significance

} Concerned with Floating Point

When one of the general registers is being used as a counter in a program, it may be desirable to test the counter for an overflow. In such cases, an overflow should not be treated as a program check. As a result, the program mask in the PSW is available to the programmer to mask program check interruptions caused by four exceptions, as follows:

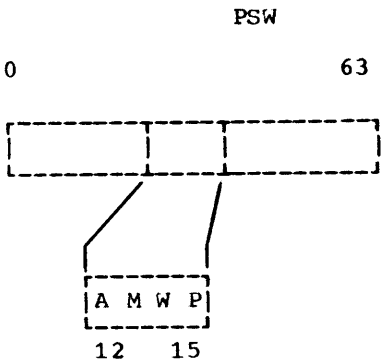


All other programming exceptions (such as specification) are always treated as programming errors and always cause a program interruption.

It is important to know which classes of interruptions cannot be masked. They are the supervisor call interruption and program interruptions caused by all but the four programming exceptions indicated in bits 36-39 of the PSW.

System/360 Status Bits

- Three bits in the PSW are used to control the System/360 mode or state.
- The ASCII mode bit (PSW bit 12) determines if decimal operations are done in EBCDIC mode (0) or ASCII mode (1).
- The wait state bit (PSW bit 14) determines if the System/360 is in the running (0) or wait (1) state.
- An external or I/O interruption causes the System/360 to go from the wait state to the running state.
- The problem state bit (PSW bit 15) determines if the System/360 is in the problem (1) or supervisor (0) state.
- Privileged instructions can be processed only when the system is in the supervisor state. A program interruption occurs in the problem state, if execution of a privileged instruction is attempted.



Of bits 12-15, you are already familiar with bit 13. It is the machine check mask bit.

ASCII MODE BIT: Bit 12 is the ASCII mode bit. ASCII is an information interchange code adopted by the American Standards Association to be used for data communication. The ASCII mode bit determines the mode in which decimal operations are done (i.e., in EBCDIC or ASCII mode). If bit 12 of the PSW contains a one, the ASCII sign (+ and -) codes and zones are internally generated, rather than the EBCDI codes. For example,
The number 1 in EBCDIC is:

1 1 1 1 0 0 0 1
Zone Numeric

The number 1 in ASCII is:

0 1 0 1 0 0 0 1
Zone Numeric

When processing data with the instructions of the decimal feature, the following are the standard signs generated:

1 1 0 0 = Plus
EBCDIC
1 1 0 1 = Minus

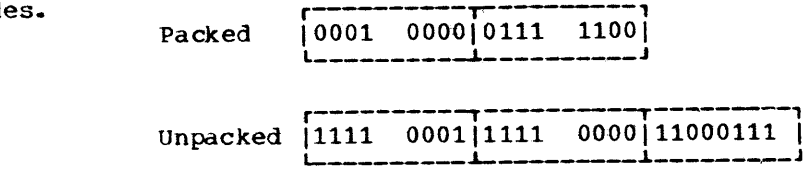
If bit 12 of the PSW contains a one, the signs that are generated when using the decimal feature are:

1 0 1 0 = Plus
ASCII
1 0 1 1 = Minus

For example, +107 is:

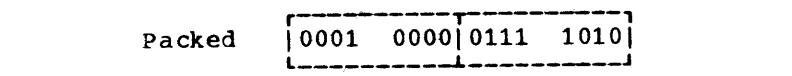
If PSW bit 12 is 0	D 0001	D 0000	D 0111	S 1100	EBCDIC
If PSW bit 12 is 1	0001	0000	0111	1010	ASCII

When a packed decimal field is converted back to the unpacked format by the Unpack instruction, the zone bits that are inserted depend on the ASCII mode bit in the PSW. For instance, +107 in EBCDIC mode is unpacked as follows:



Zones and sign inserted if PSW bit 12 is 0 (EBCDIC Mode)

If +107 is unpacked when in ASCII mode, the following results are obtained:



Unpacked	0101 0001	0101 0000	10100111

Zones and sign inserted if PSW 12 bit is 1 (ASCII Mode)

WAIT STATE BIT: If the wait bit (PSW bit 14) contains a one, instructions are no longer fetched and executed. Instead the System/360 waits until an interruption occurs and changes the PSW. The new PSW would normally contain a zero in bit position 14.

Only the occurrence of I/O or external interruptions can change the status of the CPU from wait to running state. Machine, program, and supervisor call interruptions can occur only when the CPU is in a running state.

PROBLEM STATE BIT: The Model 30 can be executing either the supervisor program or the problem program. Accordingly, the system is in either the supervisor state or the problem state.

All instructions can be executed when in the supervisor state. However, certain instructions are not allowed in the problem state. For example, all I/O instructions must be issued by the supervisor program.

Bit 15 of the PSW is called the problem state bit. When bit 15 of the PSW is zero, the instruction associated with that PSW is part of the supervisor program. When bit 15 of the PSW is one, the instruction associated with that PSW is part of the problem program. Thus, regardless of which PSW is used, bit 15 identifies the state of the System/360.

The program state bit allows the system to be sure that those instructions reserved for the supervisor state are executed only by the supervisor program. If the problem program attempts to execute an instruction reserved for the supervisor state, a program interruption occurs.

Normally:

1. Bit 15 is set to a 1 in the old PSW's in main storage.
2. Bit 15 is set to a 0 in all five new PSW's in main storage.

The old PSW indicates the next problem program instruction, while each new PSW indicates a supervisor program instruction.

Privileged Instructions

- Privileged instructions are those which can be executed only in the supervisor state (bit 15 of PSW is 0).
- An attempt to execute a privileged instruction in the problem state (PSW bit 15 set to 1) results in a privileged operation exception (a program interruption).

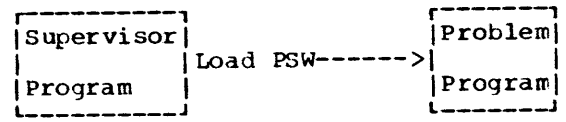
Not all privileged instructions are described here. However, you should be aware of the considerations that determine which instructions are privileged.

For example, the supervisor program has more control over changing PSW fields than has the problem program. The following table indicates how certain PSW fields can be changed:

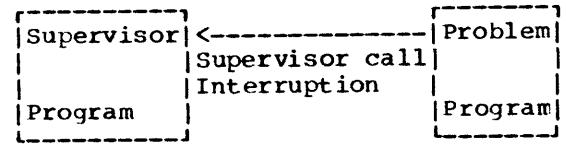
Bits	Field	Changed By
0-7	System Mask	The Set System Mask instruction
16-31	Interruption Code	An interruption
32-33	Instruction Length Code	An interruption
34-35	Condition Code	Many instructions
36-39	Program Mask	The Set Program Mask instruction
40-63	Instruction Address	Execution of program

Notice that some of the PSW fields can be changed by an instruction. Other fields can be changed only by changing the entire PSW. Basically, there are two ways of changing the entire PSW. One is by way of an interruption. The other is by way of the Load PSW instruction. It would not be desirable to allow the problem programmer to use the Load PSW instruction because this instruction changes all parts of the PSW. The problem program should not have that much control over the machine. Only the supervisor program should retain this control. As a result, Load PSW is a privi-

leged instruction. It can only be used in supervisor mode (when bit 15 of the PSW is 0). The programmer could use the Load PSW to change any part (or all) of the PSW when the system is in supervisor mode. This instruction can be used to return to the problem program after an interruption has been serviced.



The problem program can "branch" to the supervisor program by way of a supervisor call interruption.



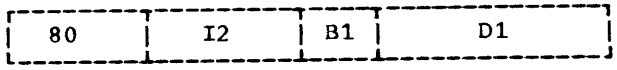
Notice, however, that a branch instruction is not used because it can not change the problem state bit (bit 15) in the PSW. The problem program cannot use the Load PSW instruction because it is a privileged operation. The problem program can only use the Supervisor Call instruction to go from the problem state to the supervisor state (PSW bit 15). Of course, this assumes that the new PSW in location 0096 (for supervisor call interruptions) has a zero in bit 15.

Besides the Load PSW instruction, there are two other instructicns which can change the PSW. They are: Set System Mask and Set Program Mask. The Set Program Mask is not a privileged instruction. Hence, the problem programmer can use it to change the program mask portion of the PSW. Actually the Set Program Mask instruction changes bits 34-39 of the PSW (which include the PSW condition code field).

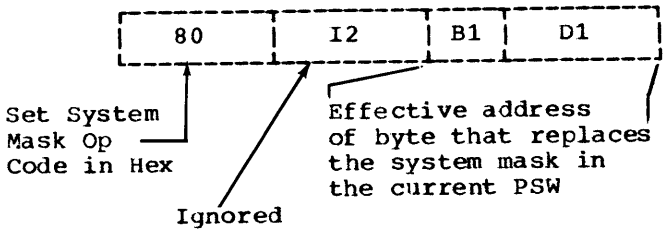
Set System Mask Instruction

- The Set System Mask instruction is used by the supervisor program to change the PSW system mask field.
- Set System Mask is a privileged instruction.
- Set System Mask is in the SI format, but the I2 field is ignored.

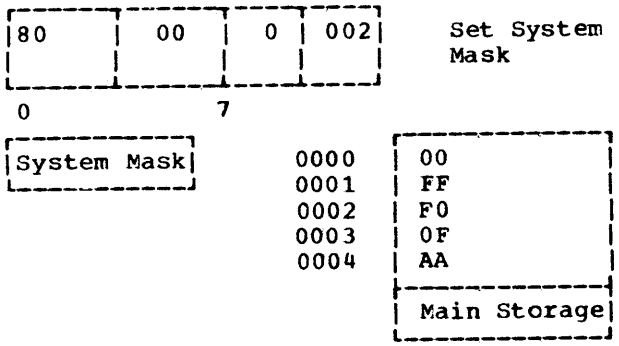
The Set System Mask instruction is a privileged instruction. Recall that the system mask affects I/O interruptions, but System/360 is designed to have the supervisor handle all I/O operations. For this reason, the Set System Mask instruction and the four I/O instructions are privileged operations. The Set System Mask instruction is:



This instruction is similar to the Load PSW instruction in that the I2 field is ignored.



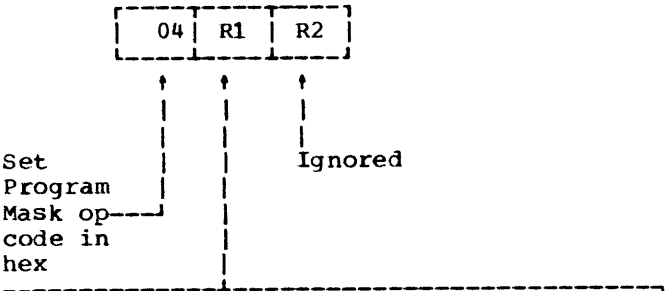
Given the following Set System Mask Instruction (in hexadecimal), the binary bit structure that is placed in bits 0-7 of the current PSW is 11110000.



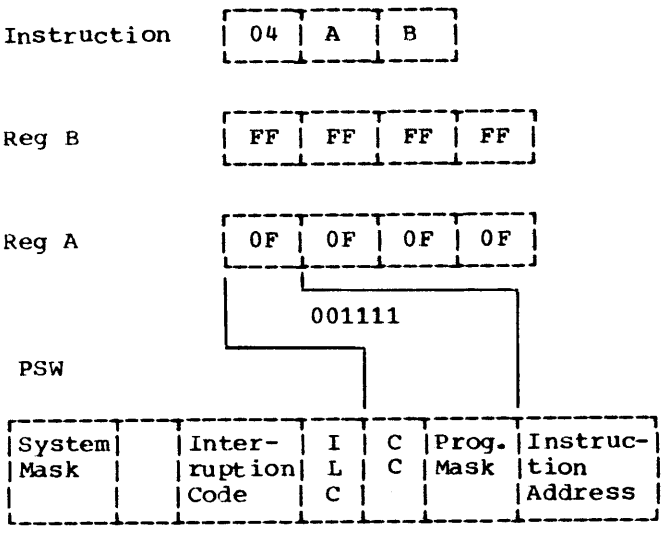
Set Program Mask Instruction

- The Set Program Mask instruction is used to change the setting of the condition code and the program mask in the current PSW.
- Set Program Mask is in the RR format and the R2 field is ignored.

The Set Program Mask instruction (RR format) is:

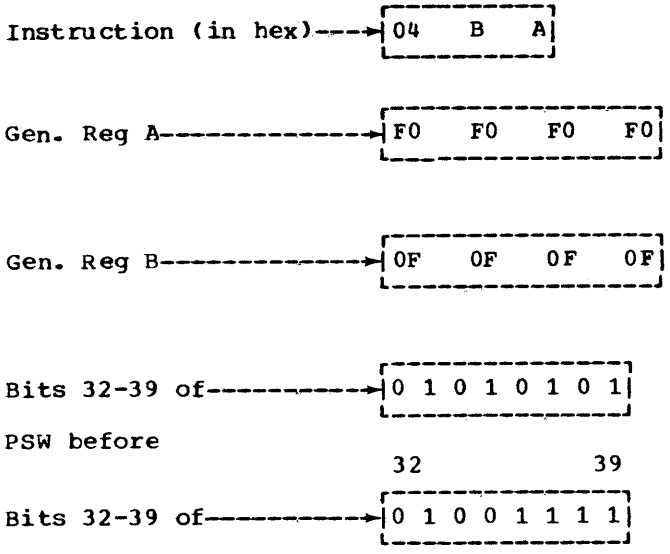


Example (note that B register is ignored):



Bits 2-7 (001111) of reg A are placed in positions 34-39 of the PSW. (Notice that the contents of general register B are ignored.) This action replaces the condition code and the program mask. With a program mask of all ones, any fixed point and decimal overflows would be treated as errors and a program interruption would occur.

Let's try another example. Given the following Set Program Mask instruction, the binary bit structure of bits 32-39 of the current PSW after the instruction is executed are as shown. Bits 32-33 are the instruction length code.



Remember that the program mask is used to determine which program checks can cause interruptions. For example, with a program mask of all zeros, a fixed point or decimal overflow is not treated as a programming error and a program interruption does not occur. Instead, an overflow sets the condition code to 11. This is normal regardless of the program mask. But now an interruption does not occur and the problem programmer can use the branch-on-condition instruction to test for overflow.

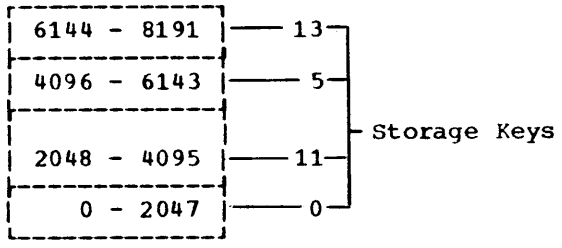
STORAGE PROTECTION

- A four bit storage key is associated with each main storage block of 2048 bytes.
- A protection key is held in bits 8-11 of the PSW.
- Every time a storage modification cycle is attempted, the associated storage key and the PSW key are compared.
- If the two keys are not the same (and the PSW protection key is not zero), a protection exception occurs, causing a program interruption so that the storage modification cycle is not taken.

Any information in main storage can be modified or completely changed. The five new PSW's in storage locations 0088-0127 can be changed. It is desirable to allow the supervisor program to modify these new PSW's. However, the problem program should not modify this same area. It is undesirable to have any part of the supervisor program changed by the problem program. What is needed is some means by which the supervisor program can change any area of main storage while the problem program can change only its own assigned area. The System/360 has a storage protection feature which prevents a program from altering main storage contents in specified areas of storage. Storage protection is a special feature in System/360 Model 30.

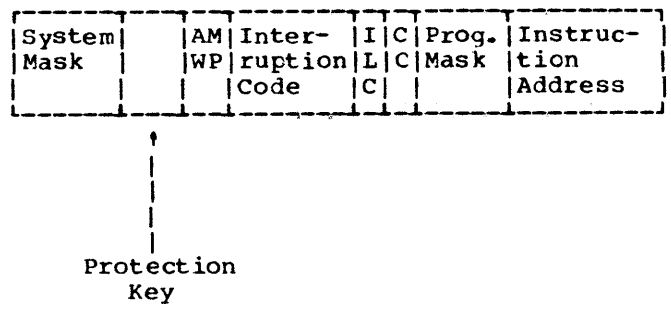
To implement the storage protection feature, each main storage block of 2048 bytes has a key associated with it. This storage key is four bits long and can contain any number from 0 to 15. Storage keys need not be assigned in any order. Any of the 16 keys can be used regardless of storage size.

For example, storage keys for a system with 8K main storage could be:



A 16K main storage unit would need eight storage keys if each 2048 byte increment were assigned a different key.

Besides the storage key associated with each block of 2048 bytes, there is a Protection Key in bits 8-11 of the PSW:



Any time the main storage unit takes a storage modification cycle, the storage protection feature is in operation. A storage modification cycle is one in which the information brought out of main storage is not regenerated. Instead new information is placed back into the same main storage location. The fetching of an instruction is not an example of a storage modification cycle, because the instruction is placed back into storage without modification.

The operation of the storage protection feature is as follows:

1. On every storage modification cycle, the protection key in the PSW is compared with the storage key associated with the block of main storage in which information is addressed.
2. A protection exception results in a program interruption if the two keys are not identical.
3. If the PSW protection key is zero, any main storage area can be modified regardless of its storage key. An interruption does not occur.

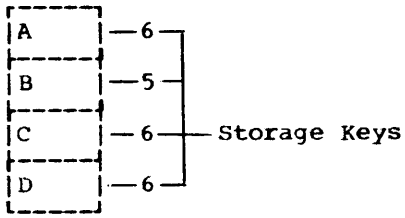
For example, if the protection key in the PSW contains a six and a storage modification cycle is attempted in an area whose storage key is five, a program interruption occurs.

If the key in the PSW is zero and the

storage key is six, however, a program interruption will not occur.

Whenever a program interruption occurs, the interruption code, placed in the old PSW, indicates the reason for the interruption. When a storage protection mismatch occurs, a protection exception is indicated in the interruption code of the old PSW (see Figure 1-24).

Assuming that the PSW has a protection key of six, the 2K blocks of main storage labeled A, C and D can be successfully modified because their storage keys match the PSW's protection key:



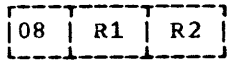
If the PSW protection key is zero, however, all four areas can be modified.

Thus, when the PSW has a protection key of zero, the current program can successfully modify data anywhere in main storage. A protection key of zero would probably be in a PSW used by a supervisor program that has a storage key of zero.

Set Storage Key Instruction

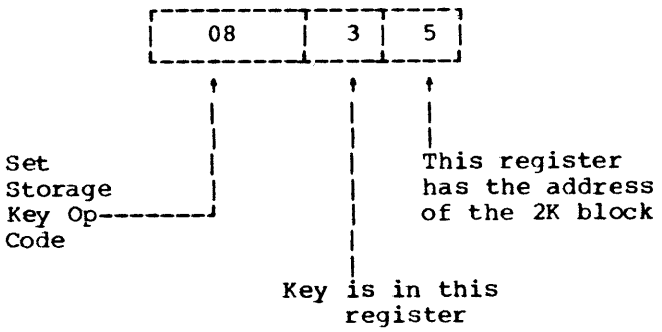
- This instruction is used to change storage keys associated with each 2048-byte block of storage.
- Set Storage Key (RR format) is a privileged instruction.

The protection key in bits 8-11 of the PSW cannot be altered except as a result of changing the entire PSW. The entire PSW is changed only by the Load PSW instruction or by an interruption. However, the storage keys for each block of 2048 bytes can be changed by an instruction known as Set Storage Key (RR format). This instruction sets the storage key for one block of 2048 bytes:



To set all the storage keys for a 16K main storage unit would require execution of eight set storage key instructions.

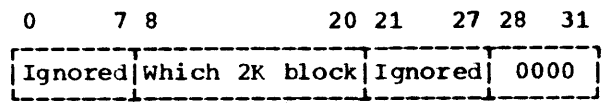
The desired storage key (0-15) is in bits 24-27 of the general register specified by the R1 field. The remainder of register R1 is ignored. The 2048 byte block of storage in which the storage key is to be set is determined by the address in the general register specified by the R2 field.



Storage addresses in the System/360 are 24 bits long. General register capacity is 32 bits. As discussed previously, storage addresses are placed in the low-order 24 bit-positions in a general register (bit positions 8-31). Because we are concerned only with 2048-position blocks of storage, and not specific storage addresses, we have to examine only those bits that define 2048-position blocks. This information can be determined from bits 8-20. Address bits 8-20 for an 8K storage system are:

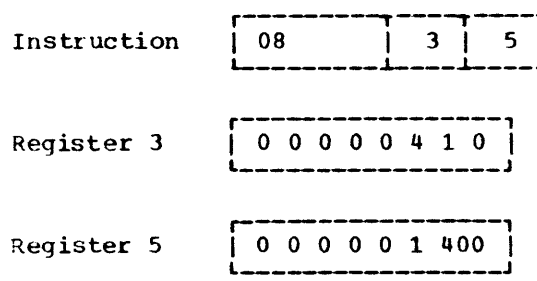
```
8                   20
0000 0000 0000 0 = addresses 0000-2047
0000 0000 0000 1 = addresses 2048-4095
0000 0000 0001 0 = addresses 4096-6143
0000 0000 0001 1 = addresses 6144-8191
```

A specification is given to the programmer that requires the general register's four low-order bits (28-31) to be zero. Thus, the structure of data in the general register, as far as the set storage key instruction is concerned, is:



Any address can be used, as long as the four low-order bits are zero. This means that the storage key can be set using any address that is divisible by 16.

Given the following, the storage key of block D is set to 1:



Storage Block		Key
A	6144-8191	0
B	4096-6143	0
C	2048-4095	0
D	0000-2047	1

General register 5 contains the hexadecimal address 140. This means that bit-positions 8-20 of register 5 are zero. Thus block D, the first block of 2048, has its storage key set to 1.

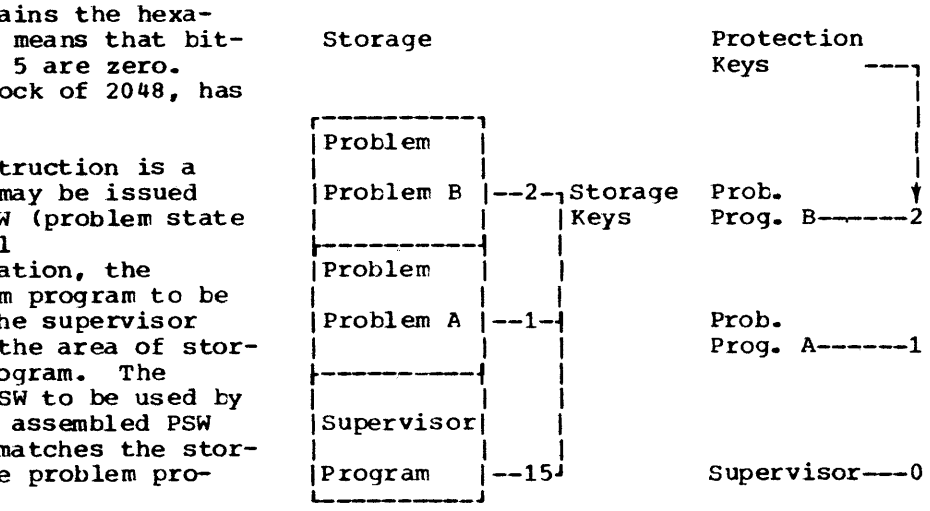
The set storage key instruction is a privileged operation. It may be issued only when bit 15 of the PSW (problem state bit) is zero. In a typical supervisor-controlled operation, the supervisor causes a problem program to be read into main storage. The supervisor sets the storage keys for the area of storage used by the problem program. The supervisor assembles the PSW to be used by the problem program. This assembled PSW has a protection key that matches the storage key associated with the problem program.

Once the function of loading a problem program into main storage and assigning the keys for storage protection is done, the supervisor passes control to the problem program with the Load PSW instruction which specifies the assembled PSW (Figure 1-25).

The protection key in the PSW used by the supervisor program is zero. This allows the supervisor program to modify data anywhere in main storage. The main storage area occupied by the supervisor program has a storage key of 0. This means that unless a program has a key of 0 in its PSW, it will not be able to modify or change information in the area being used by the supervisor program.

Each block of 2048 bytes does not have to have a different number set in its storage key. However, each program in main storage should have a different storage key assigned in order to protect one program from another. For instance, the supervisor program may take up one block of 2048 bytes which is assigned a storage key of 0. This storage key would most likely be assigned by the supervisor program just after it is read into the system. The problem program is then read into the machine under supervisor control. This program (Figure 1-25) takes up three blocks of 2048 bytes each. Each of these three blocks is assigned the same storage key (1) by the supervisor program. The PSW for the problem program is given a protection key that matches its storage keys. This allows the problem program to alter itself if necessary, but prevents it from altering another problem program or the supervisor.

So far, we have only discussed the concept of two programs in the computer: a supervisor program and a problem program. There may, however, be two or more problem programs in storage at the same time.



In the preceding diagram, each problem program has a different storage key. The protection keys used by each program are also different. Each matches its program's storage key. Notice that the supervisor's protection key does not match its storage key. Because the supervisor's protection key (in its PSW) is zero, it does not have to match a storage key. It can unlock any

area of main storage and alter its contents if necessary.

Assume: 1. That the problem program takes 5,000 bytes and begins at location 2048.
2. That the supervisor is in locations 0000 - 2047 and has a storage key of 0 and a protection key of 0.

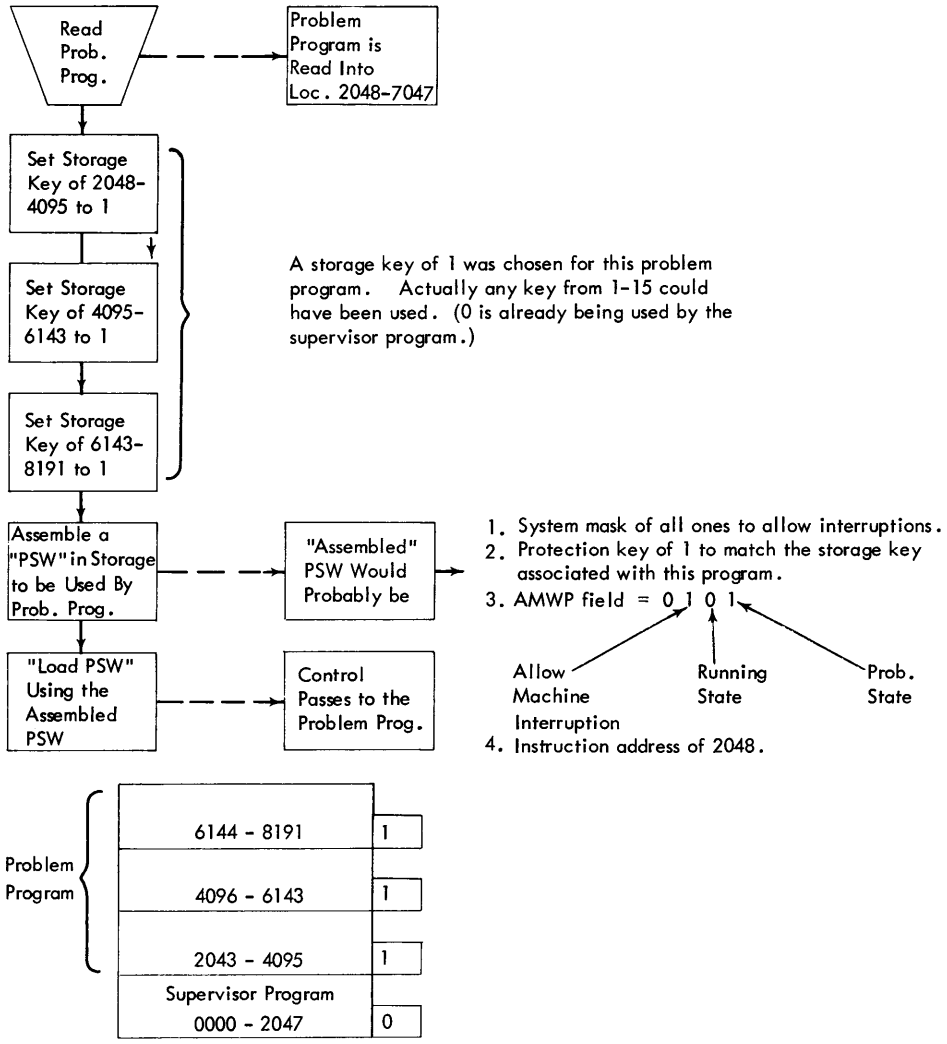
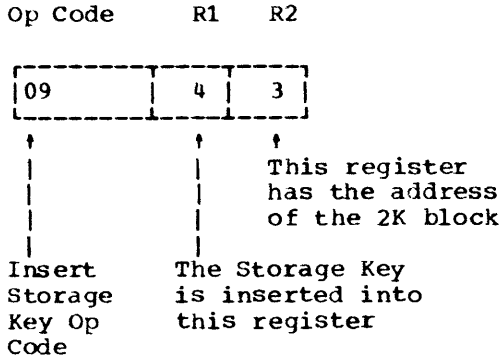


Figure 1-25. Using Storage Protection

Insert Storage Key Instruction

- The Insert Storage Key (RR format) instruction is used to examine the current value of a storage key.
- Insert Storage Key is a privileged instruction.

The insert storage key (RR format) instruction does not change any storage keys. Its purpose is to inspect or examine a storage key.



Here, the storage key of the block addressed by the contents of the register specified by the R2 field is inspected. This storage key is then inserted into bits 24-27 of the register specified by the R1 field. Bits 28-31 of this register are made zero and bits 0-23 remain unchanged.

Example:

	Storage Block	Key
	2048-4095	1
	0000-2047	F
Instruction	0 9 4 3	
Register 3 before	0 0 0 0	0 F 0 0
Register 3 after	0 0 0 0	0 F 0 0
Register 4 before	8 7 6 5	4 3 2 1
Register 4 after	8 7 6 5	4 3 1 0

Notice that the storage key (1) of block 2048-4095 is inserted into bits 24-27 of register 4 while bits 28-31 are made zero. The remainder of the register is unchanged. The storage key remains unchanged for the storage block referenced.

PROGRAMMING SYSTEMS

- Programming systems are designed to lessen the programming effort required to produce application programs.
- Each programming system requires that the machine system have certain minimum features and I/O units.
- Three basic categories of programs are:
  1. Control,
  2. Processing, and
  3. System service.

A wide variety of programming support is provided for use with IBM System/360:

1. Operating System/360,
2. Basic Operating System/360,
3. Basic Programming Support, and
4. System/360 Model 20 programming support.

All are designed to minimize the time and effort required by the user to produce and process programs. (Operating System/360 is summarized in IBM Operating System/360 Concepts and Facilities, Form C28-6535; Basic Programming Support and Basic Operating System/360 are summarized in IBM System/360 Basic Programming Support and IBM Basic Operating System/360 Programming Systems Summary, Form C24-3420.) System/360 Model 20 programming support is not applicable to Model 30 and therefore is not described here.

BPS (Basic Programming Support) is a programming system used without dependence upon any other program. Each BPS program serves a specific and limited application for minimum card and/or tape configurations.

By contrast, BOS (Basic Operating System/360) and Operating System/360 furnish centralized control for all programs. In these systems, programs are stored on a tape reel (usually file protected) or a disk pack, thus providing a high degree of program security. (That is, frequent operator handling of programs, that otherwise would be stored on cards, is not required.) At the direction of the user, these resident programs are retrieved and brought into storage by the control program when needed. This overall control results in automation of system operations with a minimum of operator intervention.

Choice of a particular programming system is dependent upon many factors. The user determines the main storage size and I/O configuration required by his applications. He chooses a programming system that gives him the most effective use of his system. Choice of a programming system, however, may influence, to some degree, the amount of storage and types of I/O units he will need.

Each programming system requires a certain amount of main storage. For example, BOS (Basic Operating System/360) 8K Disk requires a System/360 with at least 8,192 bytes of main storage. An autotest program can be obtained to assist the user in testing programs in the BOS 8K Disk environment. This autotest program, however, requires that the system have at least 16,384 bytes of main storage.

Also, each programming system requires some minimum machine configuration. If, for example, BOS 16K Disk is used, the system configuration must include, besides other I/O units, at least one IBM 2311 Disk Storage Drive. BOS 16K Tape, however, requires magnetic tape units.

Generally, within a programming system (though not in all systems) are three basic categories of programs:

1. Control,
2. Processing, and
3. System service.

A control program handles functions that are not directly related to problem solving. Such functions are control of problem program loading and control of I/O operations. The control functions achievable by any programming system depend upon the facilities of the programming system and the system configuration (number and types of I/O units and features).

Processing programs operate under control of the control program and are more directly aimed at specific applications (such as sorting and merging data) than are control programs.

System service programs are, in general, used to:

1. Create and maintain libraries (refer to the Libraries section of this manual).
2. Edit programs (refer to the Linkage Editor Section of this manual), and
3. Generate the system [i.e., set up the overall program (including control and problem) in main storage and in external storage devices (such as disk) so that desired functions can be performed)].

The particular characteristics of each programming system are not described here. Rather, a general description related to certain control, processing, and system service program components is presented. Note also that the following topics do not necessarily apply to all programming systems and do not include all possible functions of the programs described. What is presented is general programming system information that you are likely to encounter.

CONTROL PROGRAM

Supervisor

- The entire supervisor may be in main storage during problem program runs, or it may have its primary routines in main storage and less frequently used routines in an external storage medium.
- The checkpoint/restart facility provides for recording program information at intermediate points so that, if a higher priority program requires processing, the checkpointed program can later be started at the intermediate point rather than at its beginning.

The supervisor performs such functions as:

1. Interruption handling (supervisor call, external, etc.)
2. Channel scheduling (i.e., schedule I/O requests for each channel; initiate I/O operations; handle I/O interruptions),
3. I/O device error recovery,
4. Operator communication,
5. Program retrieval (from external storage, such as disk storage),
6. End-of-job indication (thereby turning control over to a job control program which may then load the next problem program).

Depending upon the programming system used, the entire supervisor may be in main storage during problem program processing. In other programming systems, the most frequently used routines of the supervisor are normally in main storage, while infrequently used routines are kept in resident storage (such as on magnetic tape). These infrequently used routines are loaded into a transient main storage area when needed. (The transient area may be used by a number of routines, but usually only by one routine at a time.)

In some cases, a single, generalized supervisor is used by all problem programs. In other situations, the supervisor is tailored to a specific application and run with only certain problem programs. The method used is usually determined by the particular programming system and, in some cases, by the application (problem) programs.

In some supervisors a checkpoint/restart facility is provided. Here, records of program conditions are made at intermediate points during job processing. These records are usually stored on magnetic tape or disk. If a higher priority program then requires processing, the checkpointed records are retained so that the original program can later be restarted at an intermediate step rather than repeating the entire program run.

IPL (Initial Program Loader)

- IPL loads the supervisor into main storage at the start of system operations.

This program loads the supervisor into main storage when system operation is initiated. (Not all programming systems have a control program component called IPL.) IPL is loaded from an I/O unit by dialing that I/O unit's address into the load-unit switches (on the system console) and pressing the start key.

IPL may initially clear all of main storage (except the area used by IPL) before loading the supervisor.

Program Loader

- The program loader loads problem programs into main storage.

When a distinct component called program loader is used, it generally performs the function of loading problem programs. In some programming systems, the functions of the program loader are handled by the supervisor or some other control program component.

Job Control

- Job control, between job runs, prepares jobs to be run.

Job control prepares jobs to be run. It performs its functions between jobs and is generally not in core storage while a job is being run. It may perform such functions as:

1. Assign actual I/O device addresses to the appropriate symbolic names used in the program to be run.
2. Set program switches according to the requirements of the program to be run.
3. Indicate that program execution is to begin.

PROCESSING PROGRAMS

Language Translators

- Language translators convert source programs to object programs.
- Programming languages used with System/360 are:
  1. Assembler,
  2. COBOL,
  3. RPG,
  4. FORTRAN, and
  5. Programming Language/I (PL/I).

Language translators are programs that convert symbolic (source) programs into machine language (object) programs. Two terms are frequently used to describe the conversion process: assembling and compiling.

In general, assembling means to produce one machine language instruction for each symbolic source statement written by the programmer; compiling means that more than one machine language instruction is produced for each input source statement.

The distinction is not always clear. For example, when source statements written in the assembler language are translated, one machine language instruction is generally produced for each input assembler statement. However, macro instructions can be written by the programmer and each of these effectively results in several machine lanuage instructions that can be used by the object program.

Depending on the programming system, one or more of the following programming languages can be used:

1. Assembler, which is a flexible, symbolic language that is machine-oriented

and applicable to both commercial and scientific problems,

2. Report Program Generator (RPG), the principal function of which is to accumulate data from existing files and generate reports from this data,

3. COBOL, which is applicable mainly to commercial problems,

4. FORTRAN, which is specifically directed to the solution of scientific problems, and

5. Programming Language/I, which is used in both scientific and commercial problems.

Depending on the programming system, a specific "level" of a programming language is usually used. For example, the basic assembler language does not include all the capabilities of the assembler language. The basic assembler is used, for example, with the BPS (8K Card) Card Assembler. The assembler language, however, can be used in the BOS (8K Disk) programming system. It is interesting to note that a program written in the basic assembler language can generally be translated and run by a programming system that normally uses the assembler language. However, the reverse is not usually true because the assembler language has greater capabilities than the basic assembler language.

Sort/Merge Programs

- Sort/merge programs sort and merge data files contained on disk or magnetic tape.

Sort/merge programs, in general, provide for sorting files of random records or merging multiple files of sequenced records into one sequential file. Records can be sorted or merged into ascending or descending sequence, and an individual sequence can be specified for each control-data field. (A control-data field is a group of contiguous bytes within a data record. The data in this field, in effect, is compared with the data in the corresponding field of every record in a file to determine the sorted or merged sequence of the records.)

Programs are provided to sort/merge files that are on disk or on magnetic tape. The user provides specifications (or parameters) that define the job to be run and the data input.

Utility Programs

- Most utility programs can be categorized as:

  1. File-to-file (such as card to tape),

  2. Multiple file-to-file (such as multiple disk to printer), or

  3. Initializing (such as preparing a disk pack for use).

IBM provides several types of utility programs to perform:

1. Transfer of information from one I/O device to another (file-to-file).

2. Transfer of information among several I/O devices (multiple file-to-file).

3. Initialization of a tape or disk volume. (A volume is the portion of a single unit of storage media that is accessible to a single read/write mechanism. For example, a reel of magnetic tape on a 2400 series magnetic tape drive or a disk pack on a 2311 disk storage drive is a volume.) An initialize-disk program, for example, is used to write standard home addresses and track description records and to make a disk surface analysis to identify defective recording surfaces (if any).

In some programming systems, either batch or SPOOL file-to-file utilities can be processed. Batch file-to-file utilities are run independently when no other program is being run. SPOOL (Simultaneous Peripheral Operations On Line) utility programs are designed to maximize total job throughput. For example, if one program does not require the full I/O capacity of the system, other peripheral operations can concurrently use the I/O facilities that would otherwise be idle. Also, a program that

normally uses slow speed I/O devices (such as printers, card readers, and card punches) can direct its output to high speed I/O devices (such as magnetic tape or disk units). Later, a SPOOL operation can transfer this data from the high-speed to a low-speed I/O device if concurrent programs do not need this pair of I/O devices.

Autotest

- An autotest program provides testing facilities for application programs.

Autotest programs provide debugging capabilities for assembled program decks as they are test-run. In general, the user can batch (run several programs, one after another) a number of individual test jobs and get extensive diagnostics and testing services with just one loading procedure.

SYSTEM SERVICE PROGRAMS

Linkage Editor

- The linkage editor links together and relocates object program segments (routines).

This program edits the output of language translators and produces executable phases (an entire problem program may be a phase) in a library (see Libraries). The linkage editor relocates programs or program sections and links together separately assembled sections. Linking is the cross referencing of program routines. For example, a subroutine may have to be inserted into another routine before the program can be run. Cross-references (specified by the programmer) between the routines are used by the linkage editor to establish the correct relationships between the routines.

Libraries

Core Image Library

- The core image library, on disk or magnetic tape, contains program phases in a form identical to that which they have when in core storage.

The core image library (not a program) is a grouping of programs, each comprising one or more phases. Each phase is the image of (i.e., identical to) its form in main storage. (The core image library is on magnetic tape or disk.) Programs that may be in the core image library are:

1. User problem programs,
2. Job control,
3. Linkage editor,
4. Language translators,
5. Library maintenance programs, and
6. Sort/merge programs.

The desired program is moved from the core image library (on tape or disk) to main storage when it is to be processed.

Macro Library

- The macro library (on tape or disk) contains a number of series-of-instructions each of which can be referenced by a macro-instruction statement.

The macro library (on disk or tape) contains instruction routines; each routine can be referenced by a macro-instruction statement. Macro-instruction statements cause the assembler language translator program to retrieve a specially-coded symbolic routine from the macro library, modify the routine according to the information in the macro instruction, and insert the modified routine into the source program for translation into machine language. IBM provides specially coded routines as part of a macro library and the user can, in some programming systems, define his own macro-library routines. He can then reference these routines through macro-instruction statements that he defines himself.

Relocatable Library

- The relocatable library contains object modules (program sections) that can be located into various areas of core storage.

This area (on tape or disk) is used to store object (machine language) modules (a separate program section that can be combined with other sections) in relocatable format. Relocatable means that the module can have its addresses (with reference to main storage) changed, and hence it can be placed in various areas in main storage. Note that some programs, due to the manner in which they are written, cannot be relocated. The object modules stored in this library can be combined with other object modules (that are either in the relocatable library or are read in from an I/O unit) by the linkage editor when it edits a program in the core image library.

Library Maintenance Programs

- Library maintenance programs provide services to enter or delete library sections, to print out the contents of a library, and to rearrange library sections.

These programs are used to:

1. Enter or delete phases (in the core image library) and macro definitions (in the macro library),
2. Translate information from a particular library to printed (or displayed) or punched output, and
3. Reallocate and condense libraries.

Load System Program

- When used, the load system program generates (or sets up for use) a minimum resident system.

This program may be used to create a minimum resident system. The system created may be used to generate other specialized systems, or the load system program itself may be used to produce specialized systems. Many times it is unnecessary to use the load system program. In this case, system generation is accomplished by other means which depend upon the programming system used.

CHAPTER 2. FUNCTIONAL UNITS

SYSTEM CLOCK

- The basic timing pulses for the IBM 2030 are generated by the system clock.
- A crystal oscillator drives a four-stage latch ring.
- Latch ring outputs travel via transmission lines to the SLT large cards.
- Specific timing pulses are created at the large cards by mixing the latch ring pulses.

Four latches are connected to form an overlapped latch ring for creating the basic clock pulses (Figure 2-1). A free-running crystal oscillator provides the pulses that drive the latch ring circuit. The latch ring circuit is reset with the clock 4 latch on. When the clock is to start, the clock start latch is turned on. This allows clock 1 latch to turn on. The latches turn on in progression. Clock 2 latch turns on before clock 1 latch turns off, clock 3 latch turns on before clock 2 latch turns off. The result is four overlapping timing pulses called P1, P2, P3, and P4. These four pulses are sent via transmission lines to the large cards. (Figure 2-2) At the large cards, logic circuits combine the P-pulses to develop the specific timing pulses needed at the large cards. These pulses are shorter than the P=pulses and are called T1, T2, T3, and T4. Use of the transmission line distribution system, allows the subdistribution centers to be close to the logic. Thus, ringing and noise are minimized.

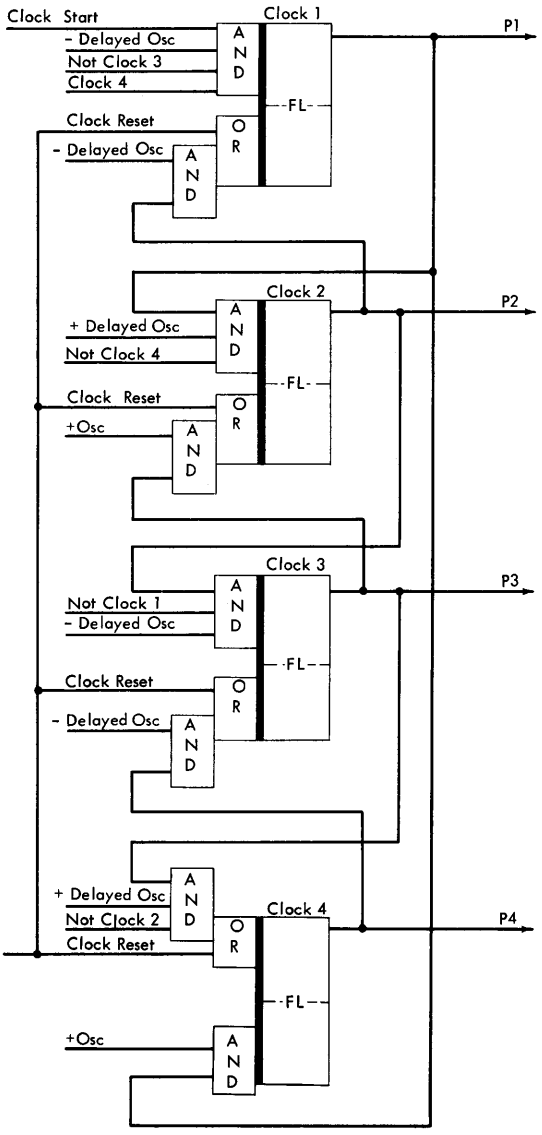


Figure 2-1. Clock Pulse Generation

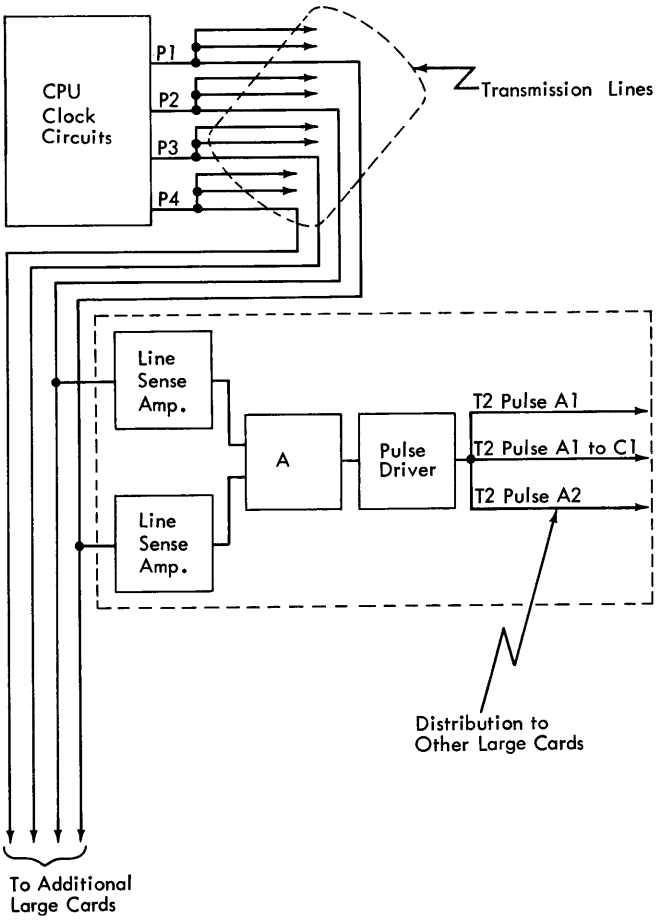


Figure 2-2. Clock Pulse Transmission

The system clock operates on either a 750 nanosecond or a 1 microsecond clock cycle depending on the type of core storage unit in the 2030. The M2 core storage unit operates on a 2 microsecond read/write cycle, and therefore it requires a one microsecond system clock (Figure 2-3). For the one microsecond clock, the oscillator runs at 2.0 megacycles per second. Turn-on and turn-off of the clock latches produce 500 nanosecond, overlapped P1, P2, P3, and P4 pulses. These pulses are brought together at the large cards to form timing pulses that are 250 nanoseconds long.

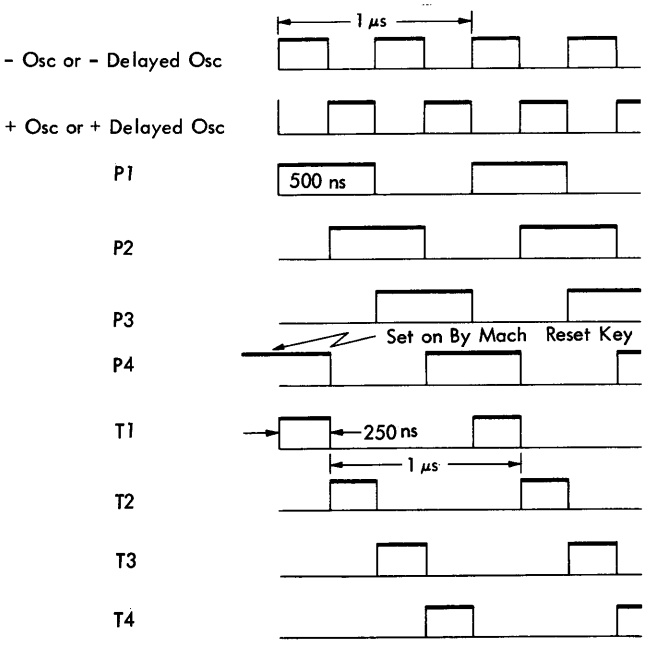


Figure 2-3. Clock Timing (1 us clock)

The 750 nanosecond clock operates the same as 1 microsecond clock. However, the timings are shorter. The oscillator operates at 2.67 megacycles per second. This produces 375 nanosecond P-pulses, and 187.5 nanosecond T-pulses (Figure 2-4).

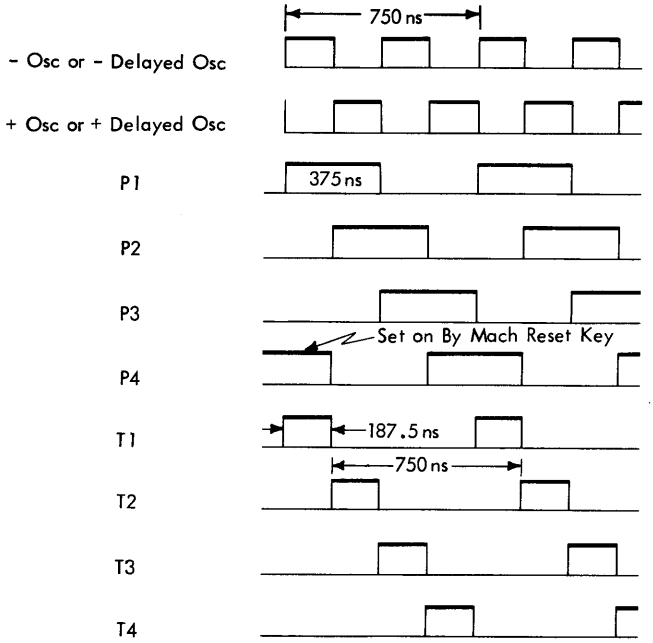


Figure 2-4. Clock Timing (750ns clock)

CLOCK CONTROL

- The clock is reset with P4 on and P1, P2, and P3 off.
- Clock start line allows oscillator pulses to reach the clock latch ring.
- The clock always stops with P4 on and P1, P2, and P3 off.

The clock oscillator runs continuously as long as power is on. When the 2030 is reset, clock latches P1, P2, and P3 are off, and clock latch P4 is on. Raising the clock-start line allows oscillator pulses to reach the latch ring. As long as the clock-start line remains up, the latch ring continues to run. When the clock-start line drops, the latch ring continues until P4 turns on and P3 turns off. At this time, the latch ring stops until the clock-start line is raised again. The clock-start line is controlled by the clock-start latch which must be on to start the clock. The clock-start latch is turned on by either the Start key or the Load key (SLD Figure 5-03C). With the clock-start latch on, the clock-start line may be blocked to prevent further clock cycles. For example, when the Power-off key is pressed, the power-off latch turns on. This blocks the clock start line which stops the clock at the end of the current cycle. The clock may be reset immediately by a machine reset. This line resets P1, P2, and P3 off and P4 on (SLD Figure 5-08A). Machine reset also raises the clock-reset line to reset the clock control latches such as the clock-start latch, the clock-start-control latch, and the load-key latch (SLD Figure 5-03C). Thus the clock-reset line prevents the latch ring from being restarted after it is reset.

REGISTERS

- The 2030 uses storage latches for registers.
- Polarity hold latches and AOI latches are used.
- Most register input and output data movement is controlled by the read only storage unit.
- Data may be moved into or out of a register manually.

Registers in the 2030 are used for storing addresses, status information, and data. Many registers have multiple functions; the function used depends on the operation being performed. These registers are made up of storage latches of either the polarity hold or AOI type. In the polarity-hold latch, the output line follows the data line when the control line is active (Figure 2-5). This means that information on the data line is set into the polarity-hold latch when the control line is raised. Notice that there is no actual reset of this latch. It is reset by raising the control line while at the same time, leaving the data line down. The I, J, U, V, T, G, L, and D registers are all samples of registers using polarity-hold latches (SLD Figure 5-05C).

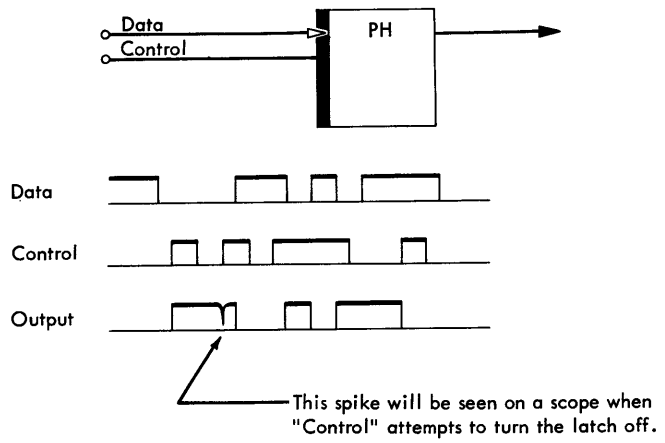
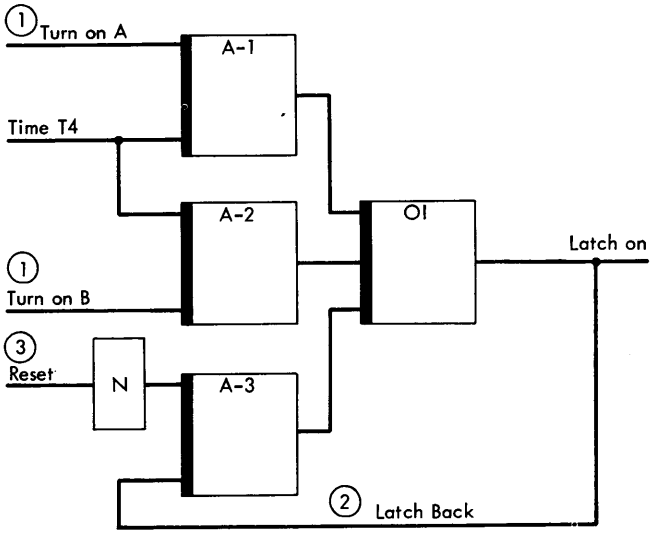


Figure 2-5. Polarity Hold Latch

The second type of latch used for registers is the AOI latch which is made of several logic blocks tied together to form a latch (Figure 2-6). This latch is used when multiple inputs are required, and when a single reset function is desired. The F-register (SLD Figure 5-04C), the MC-register (SLD Figure 5-07A), and the S-register (SLD Figure 5-07B) are all examples of AOI latches used as registers.



(1) T4 and Turn on A or B turns latch on.

(2) Turn on of latch sends Latch Back pulse to switch with (not) Reset to keep latch on.

(3) Reset deconditions A-3 to turn latch off.

Figure 2-6. AOI Latch

The information to be placed into a register may come from any one of several points in the CPU. Likewise, information in the register may be directed to several points in the CPU. In Figure 2-7, the input gating is shown for two sources, while the output gating is shown for one destination. Keep in mind that for this example, there are actually 9 PH latches, nine sets of input gating, and nine sets of output gating. In this example one input comes from the main storage unit. The second input comes from the Z-bus which is the output of the arithmetic and logic unit.

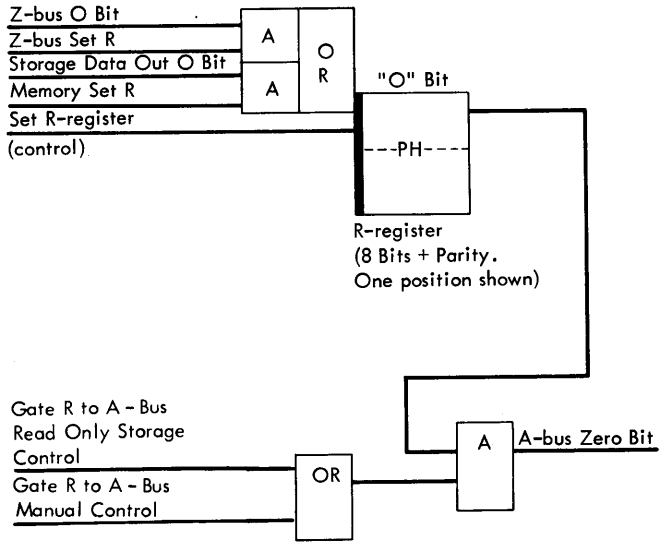


Figure 2-7. Register Control

The polarity hold registers are reset by raising the control line while keeping the data input lines down. A machine reset would cause most registers to be reset off. To prevent parity errors, the machine-reset line resets the parity latch on in most registers. One exception to this reset system is the F register where all positions except the 1A latch are reset on. The F-register is part of the interrupt mask system. Resetting all positions, except the 1A latch, on allows external interrupts to occur after the machine has been reset.

READ ONLY STORAGE AND MICROPROGRAM

- ROS (Read Only Storage) is a nondestructive read out storage device.
- Microprogram is a machine control program and is punched in special cards called ROS Cards.
- ROS cards are placed in the ROS device and are selected to read out a logical functional operation for the machine.

Before we learn the details of the ROS and microprogram lets look at some of its general concepts.

The ROS in the 2030 is a CROS (Capacitor Read Only Storage) device which uses the capacitor as a storage device. If we have a capacitor at a selected spot, we say there is a bit, or the condition is a 1. If we do not have a capacitor at a selected spot, then we do not have a bit or the condition is 0. By selecting a set of capacitors and decoding their bits (1) or no bits (0) we can control signal lines and gate inputs and outputs of registers. The only way the information in ROS can be changed is by adding or removing capacitors. Therefore, we can only read out of ROS and the read out is nondestructive as compared to core storage where the information is read out and must be written back in order to retain it. Figure 2-8 shows the general operation of ROS. An address is set into the address register and then decoded to select a certain position in the ROS device. An impulse is then sent to the capacitors. The outputs from the selected capacitors are sensed and decoded to condition a circuit which controls the input or output of one or more registers.

The microprogram is a written program. The program is divided into words; each word contains logical statements telling what function the 2030 should be performing during this one microsecond cycle. Figure 2-9 is a page from a microprogram. The microprogram is laid out on CAS (Control Automation System) CLD's (CAS Logic Diagram). Each microprogram word is placed in a logic block on the CLD.

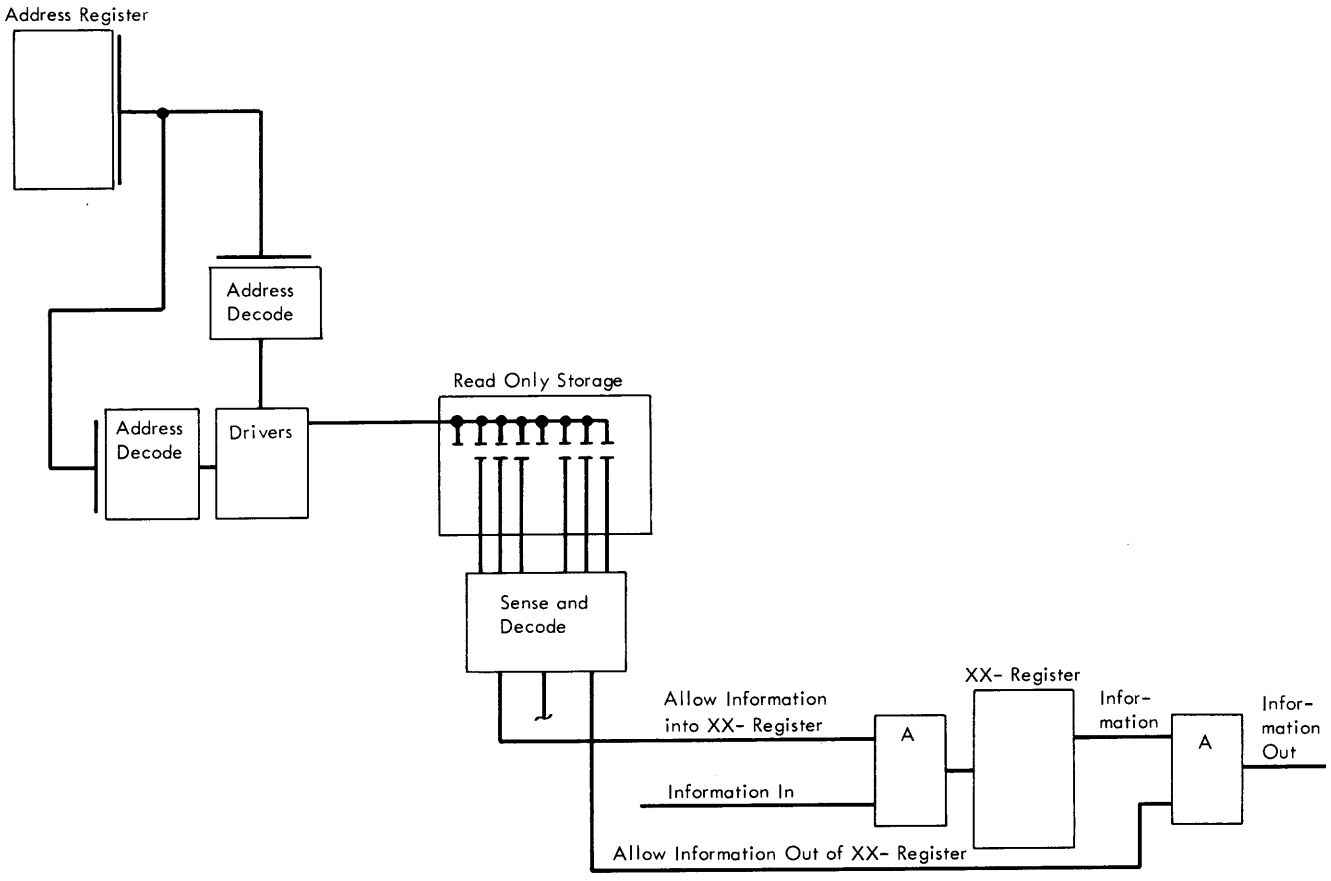


Figure 2-8. General Operation of ROS

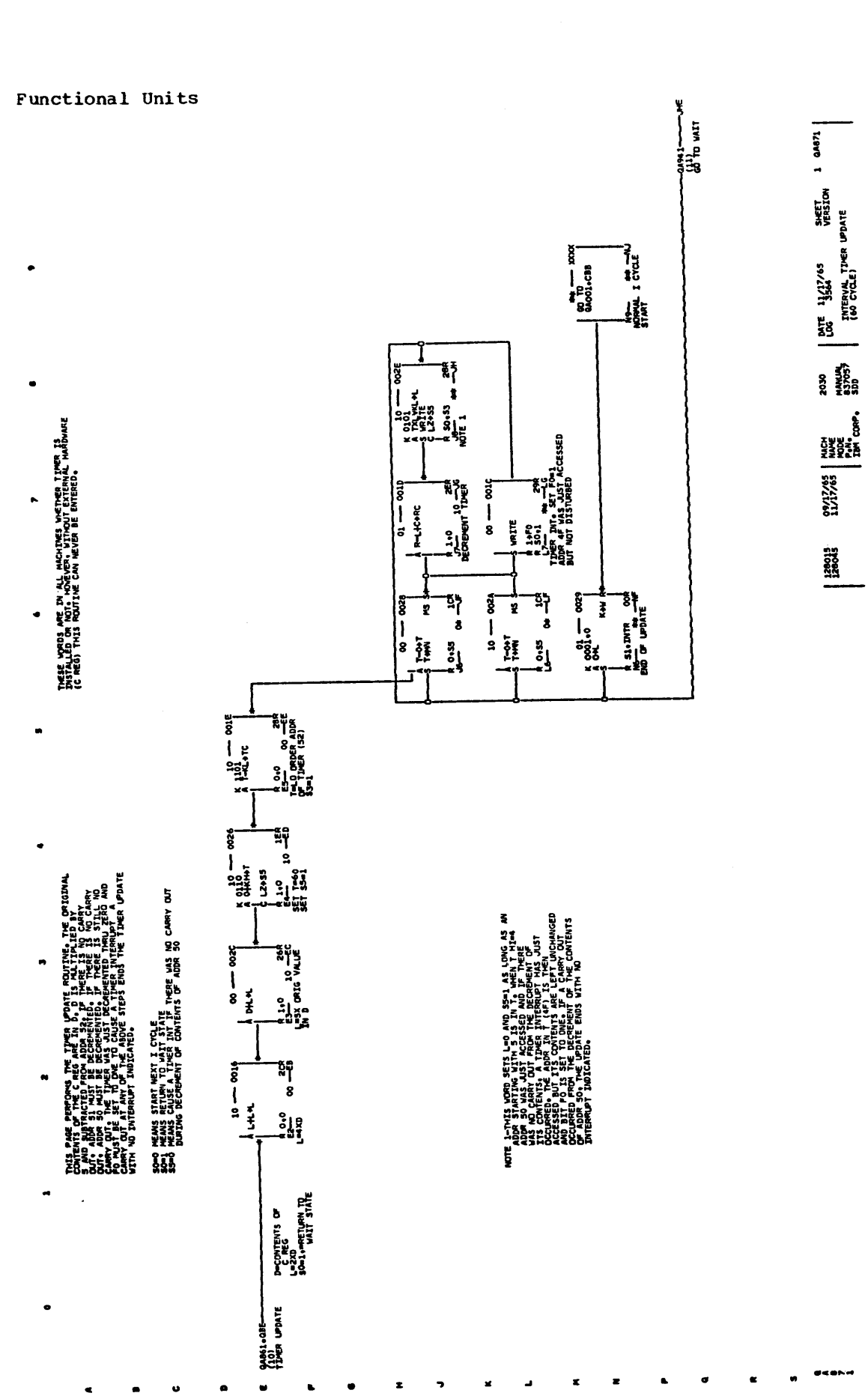


Figure 2-9. CLD Page

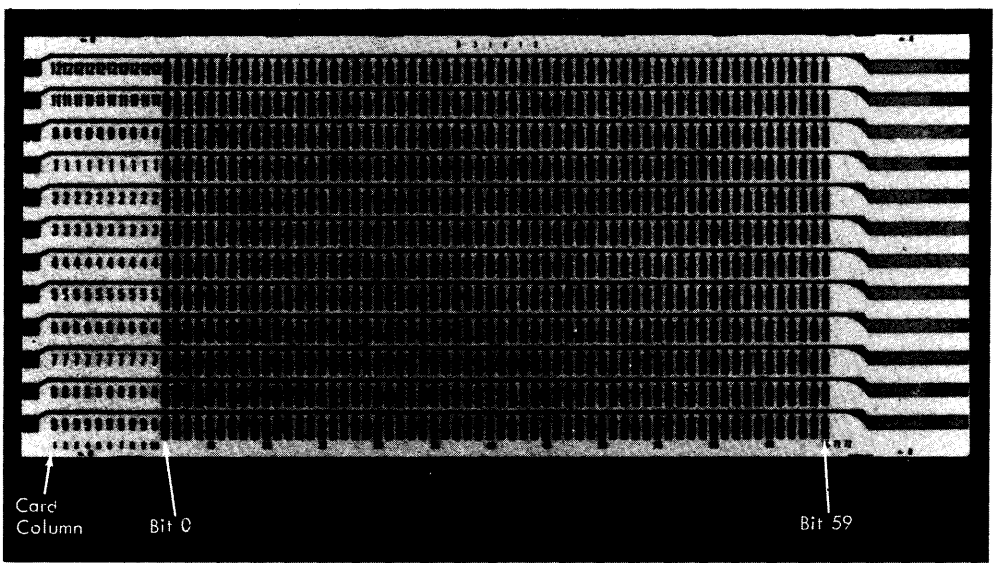


Figure 2-10. ROS Document Layout

In order to place the microprogram into the 2030, use punch-card size documents called ROS cards. The microprogram word is coded and punched into the ROS card. There are 60 positions for punching in each row of the ROS card (Figure 2-10).

Each position is one side of the capacitor used in the CROS. Therefore, by punching the card, the microprogram can be placed into the ROS device. Each row is one microprogram word.

Now that we have some idea what ROS and microprogram are and how they are used, we shall go into more detail and explain the concepts of ROS, how it is used, and how to read a microprogram.

ROS CONCEPTS

- IBM System/360 Model 30 is controlled by a microprogram.
- A ROS (microprogram) word controls each machine cycle.
- System control lines and gates are controlled by fields in the ROS word.
- CROS (Capacitor Read Only Storage) contains the microprogram in the form of ROS words.
- CROS for the 2030 can contain 8064, 60 bit words.
- Bit patterns of ROS words are determined by the presence or absence of capacitors.
- ROS words can be changed by replacing ROS cards.

The CROS device is used to hold predetermined information, such as the microprogram, that can be nondestructively read out. The microprogram is punched in the ROS cards. Up to 4,032 words (12 words per card) are used unless a compatibility feature is installed, then another 4,032 word CROS module is installed. Each ROS word contains 60 bits that control the gates and control lines of the system for each 1 microsecond machine cycle. Gating for each functional unit is controlled by the bit combination within a field of a ROS word. Later we will see where 0111 in bit positions 23-26 of the ROS word gates the R-register to the A-register bus.

The bit pattern of a ROS word determines the presence or absence of capacitors within the CROS hardware. A ROS word cannot be changed by a customer program; however, the Customer Engineer can change the information in the ROS words by replacing the ROS cards.

CROS replaces most of the system control circuits, as used in previous machines, and introduces a flexibility to machine design that we did not have before. This flexibility allows changing the control circuit for a feature by replacing or adding the necessary ROS cards.

CONTROL POINTS

- The capacitor is the most important component of the CROS.
- A line driver impulses many capacitors.
- Each control point in the 2030 data flow is controlled by a SAL (sense amplifier latch).
- The bits in the ROS word determine if the line is active or inactive for that cycle.

Using simplified block diagrams we can learn the theory and operation of CROS. In our development of CROS we will see: (1) a control point source, (2) a selection device for the source, and (3) basic operation of ROS.

The block diagram in Figure 2-11 has the control points numbered. For example, the in-gate control point for the G-register is number 3. By using the statement READ OUT R, GATE THE OUTPUT THROUGH THE LOGIC UNIT, AND STORE IT IN S, the use of control points can easily be seen.

The first part of the logic statement, READ OUT R, indicates a need to condition control point 2 (see Figure 2-11). By adding a latch called a SAL (sense amplifier latch) to this point, we have a method to allow the R-register bits on the in-bus (Figure 2-12). We used the capacitive coupling (a) from a line driver (b) to turn on the SAL which allows the R-register to be gated to the in-bus.

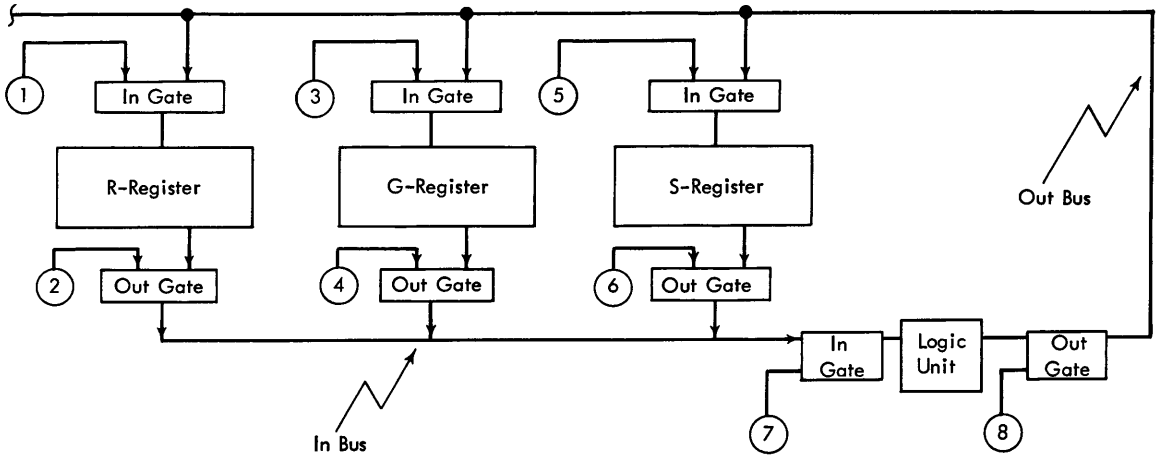


Figure 2-11. Control Points

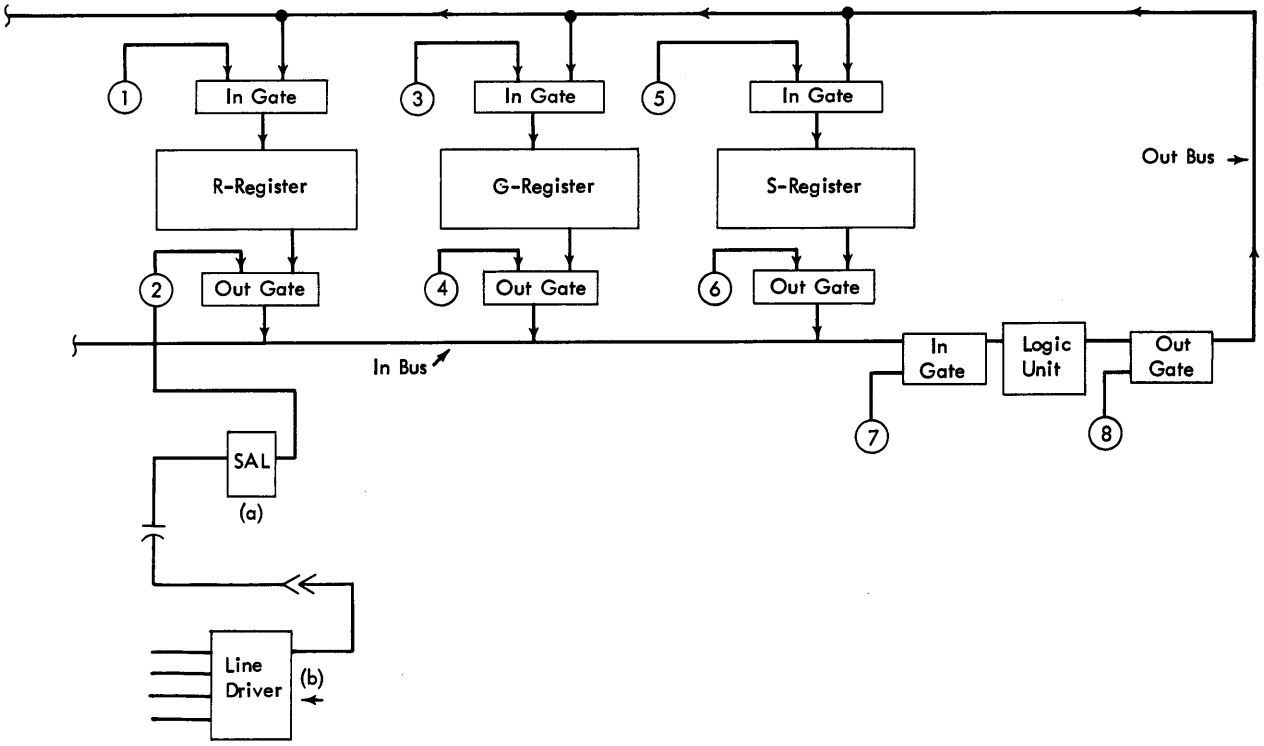


Figure 2-12. SAL Control

To do the rest of the statement, we must read in and out of the logic unit and into the S-register. Figure 2-13 shows that we do this by adding three more SAL's and connecting them to the proper control points. The three added SAL's are also capacitive coupled to the same line driver as the first SAL. Thus, the conditions of our statement have been satisfied. We have now established a source for our control points (the SAL's), and a selection for the source (the line driver and coupling capacitor).

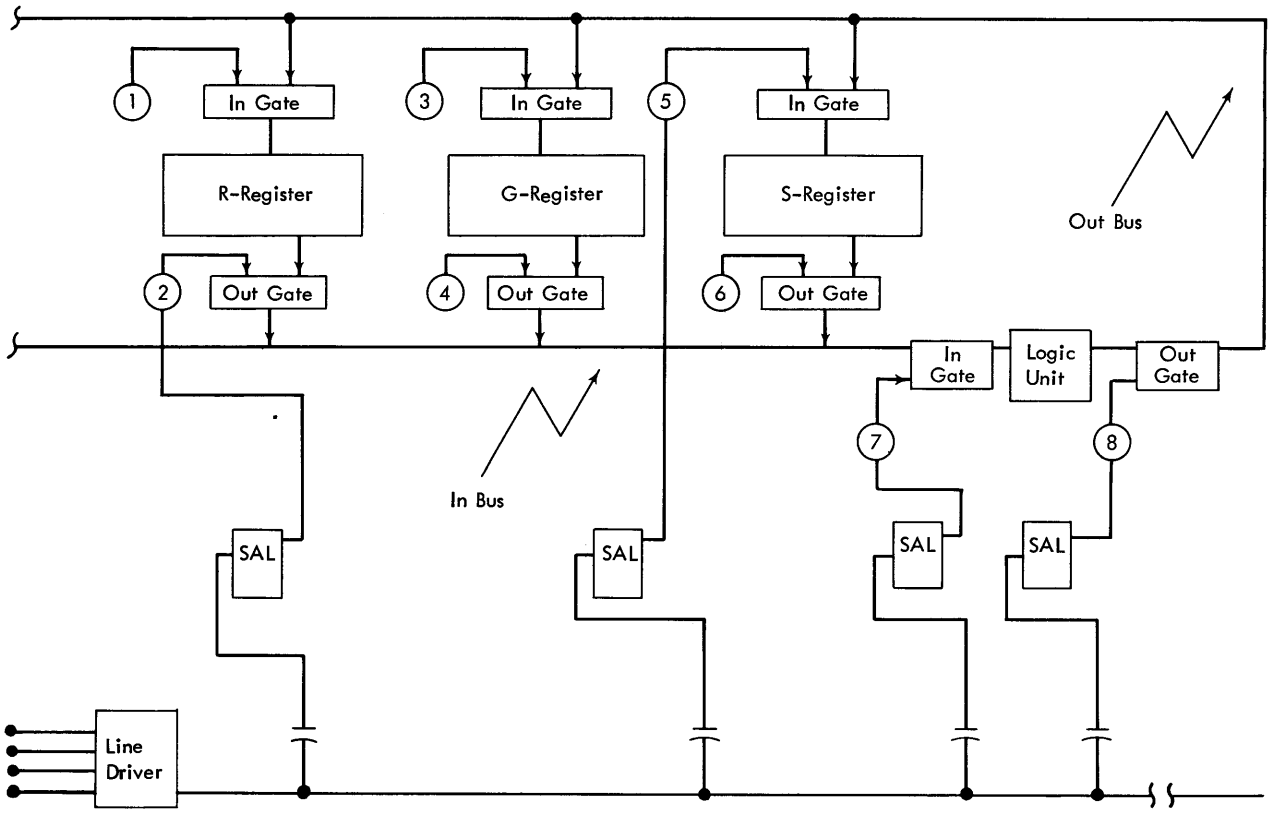


Figure 2-13. Multiple SAL's

So far we have only looked at SAL's that were active for our specific statement. In the block diagram, there are eight control points and each one has to have a source and a selective device.

What happens to our statement if we connect the R-register as shown in Figure 2-14. Beside reading out of R, we will read into R. There is nothing wrong with this electronically and it can be a legitimate operation, but the conditions set by the statement are not satisfied. Assume we can cut one of the plates off the coupling capacitor for the SAL that conditions control point 1 (Figure 2-15). Once again the statement is satisfied.

Now if we connect a SAL for each control point (Figure 2-16), but cut one plate from each coupling capacitor we do not want to use, and impulse the driver, the statement is still satisfied. Thus, we have a storage device, and each time the line driver is impulsed, the same operation is performed.

Actually we have a read only storage device made of capacitors with either one or two plates using a common drive line.

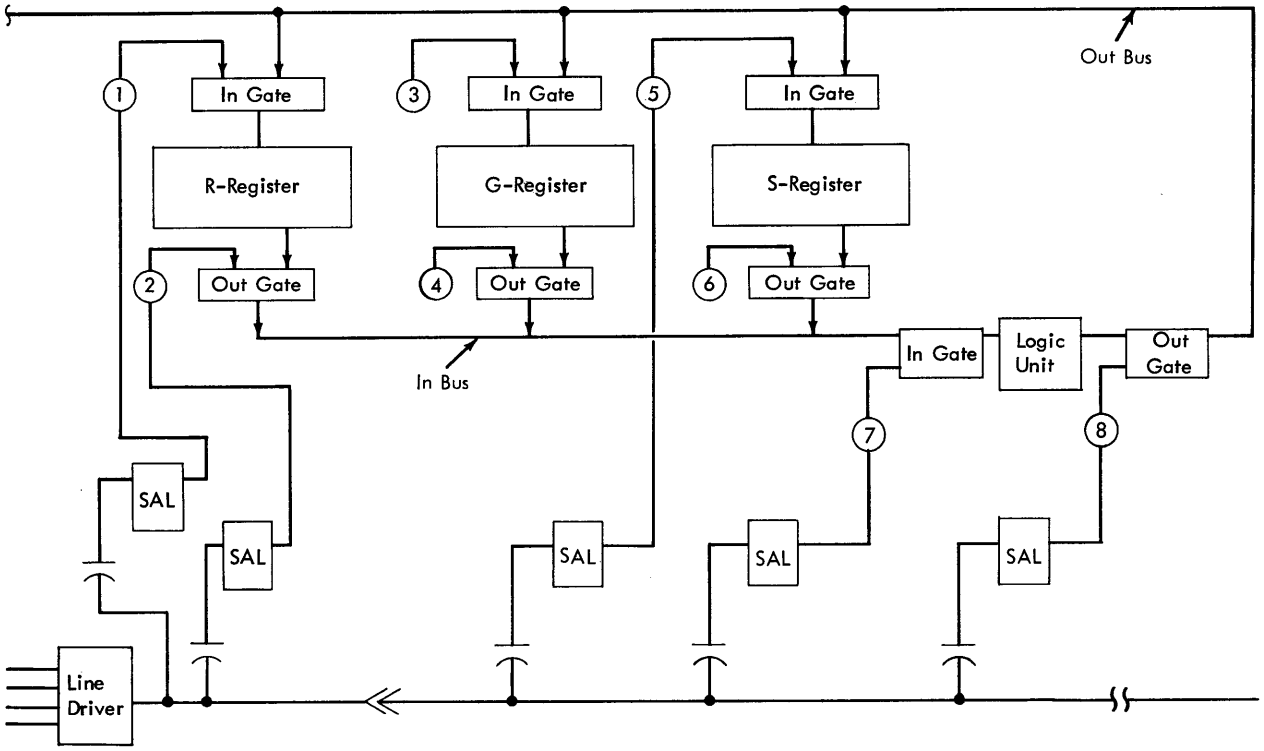


Figure 2-14. SAL Gate

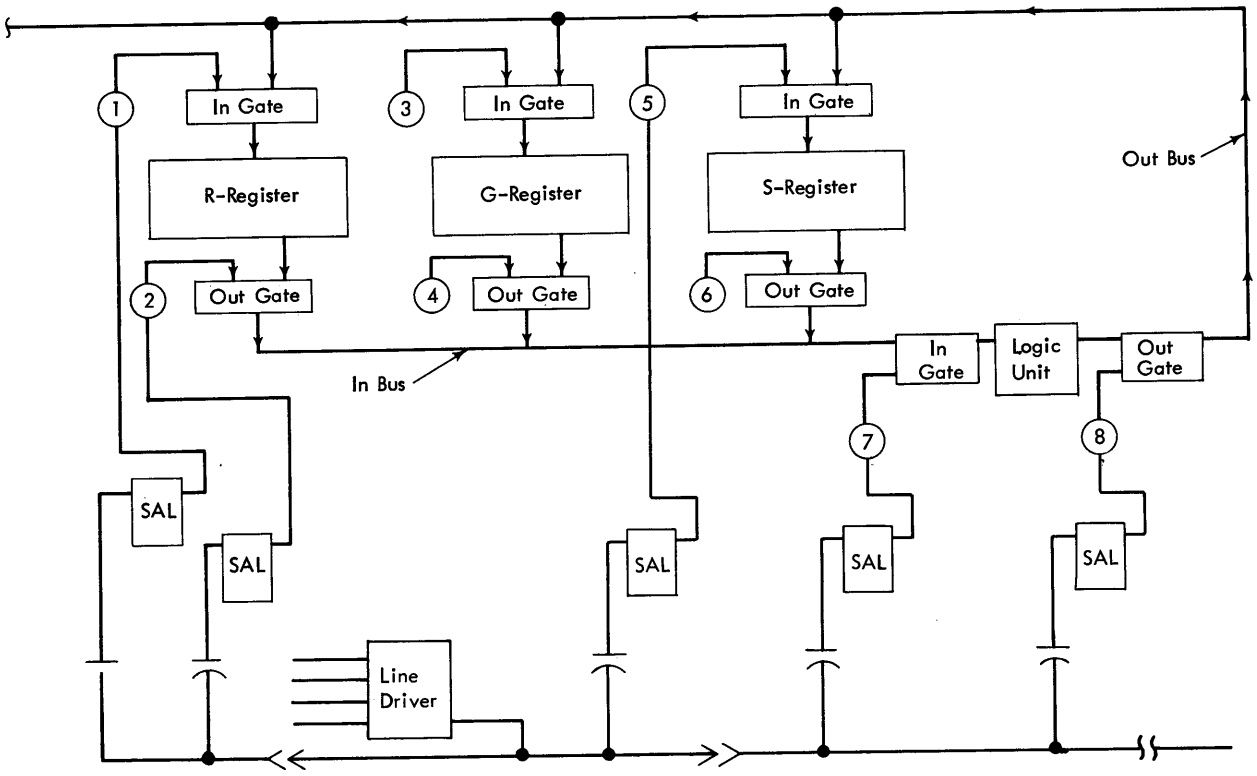


Figure 2-15. SAL Selection

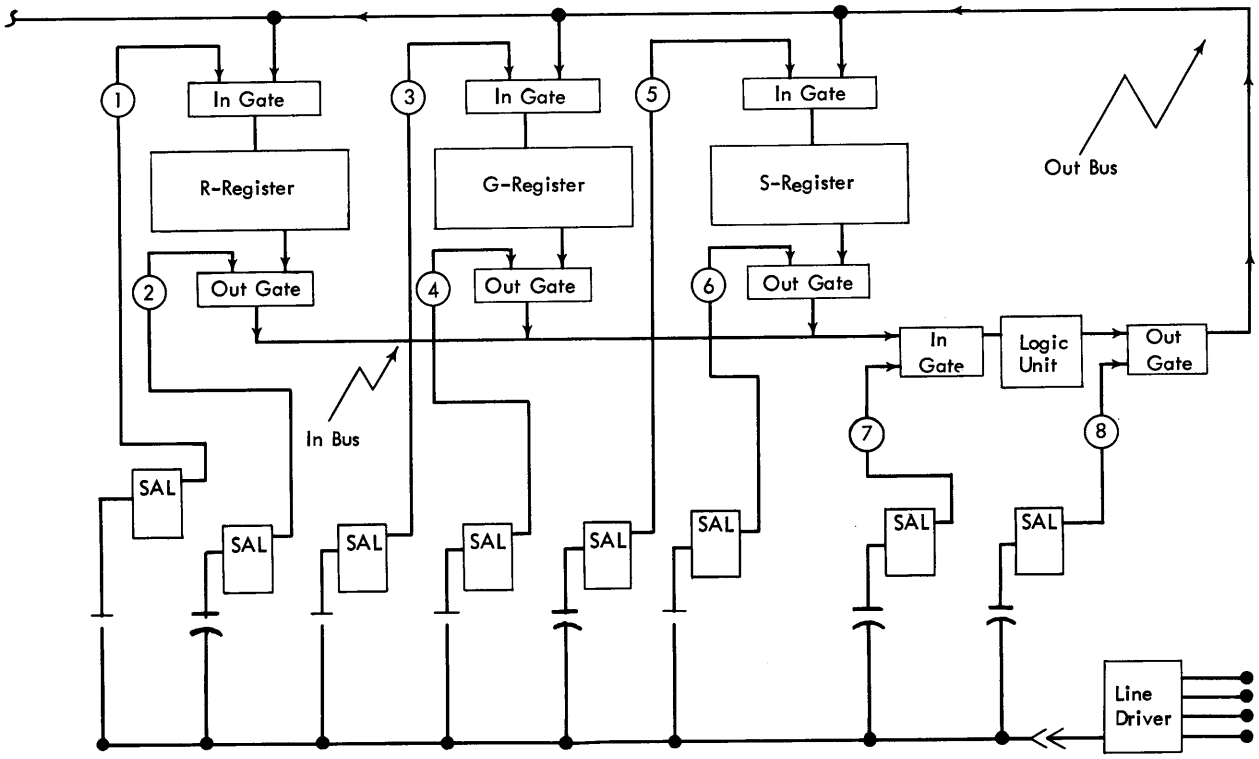


Figure 2-16. Multiple SAL Selection

ROS WORD

- Each ROS word is one step of a microprogram.
- The ROS word is made up of a string of capacitor plates having a common line feeding them.

The statement we have been working with is one ROS word. If we want to do a different function or operation, we either have to replace the first word or add another. Since we still want to do the first statement again, we add another ROS word and another line driver (Figure 2-17). This new word can perform the function READ-OUT R, TAKE THE OUTPUT THROUGH THE LOGIC UNIT, AND READ-IN G. If we impulse line driver one, we take what is in R and put it in S. If line driver two is impulsed, we take what is in R and put it in G. By adding plates to the coupling capacitors and adding more line drivers, we can create enough ROS words to perform any function our data flow can handle.

We know what we want to do but what is an easy way of doing it? First, let's look at the plates' connections to the SAL's. In Figure 2-18 the capacitor plates are shown connected serially to a SAL. A ROS board is used to do this in the 2030. The ROS board is made of laminated fiber board and the capacitor plates are made of copper which are laid on the board.

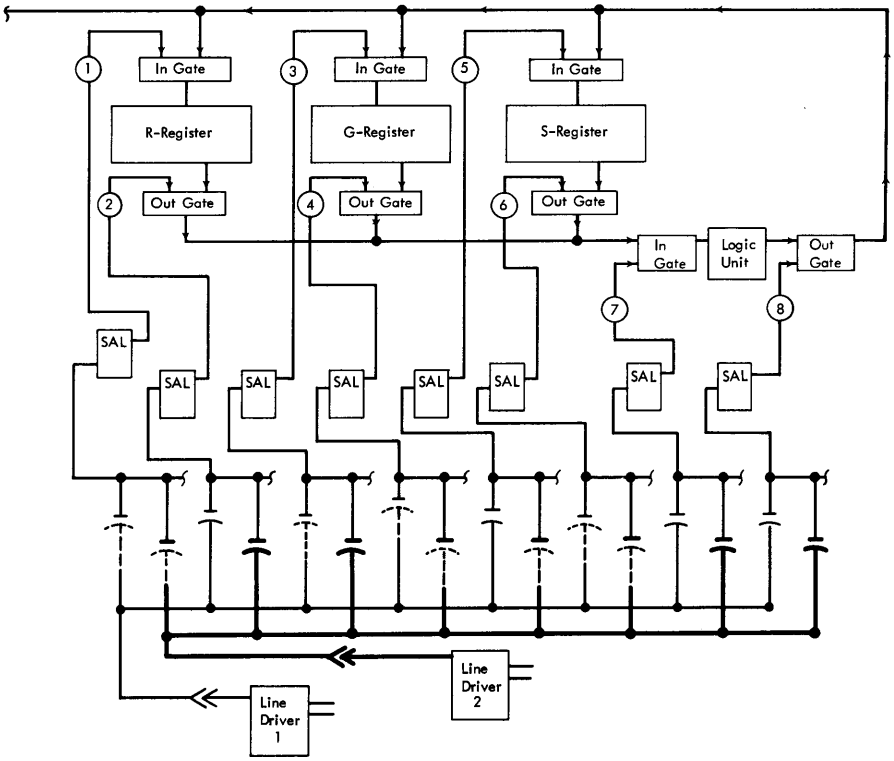


Figure 2-17. Multiple Drivers

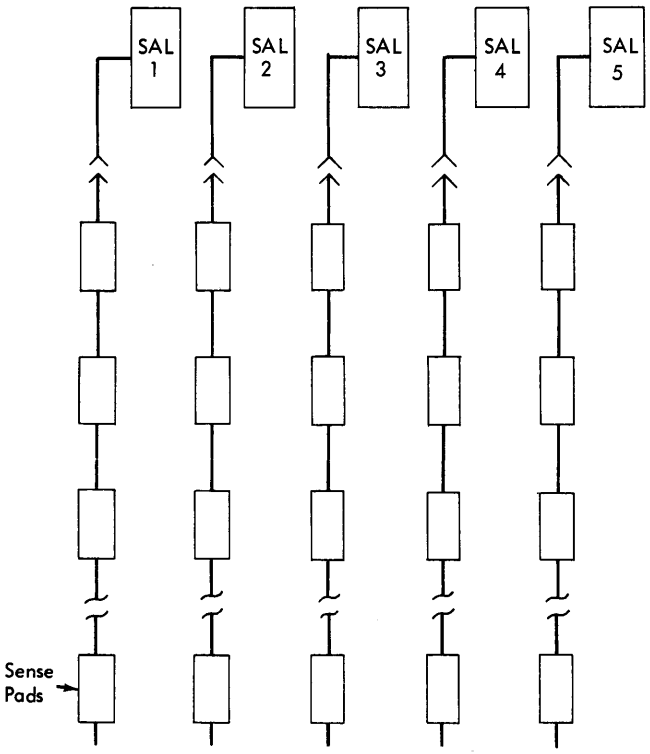


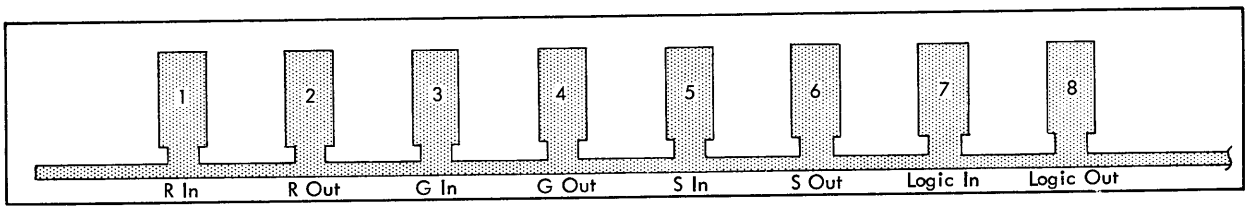
Figure 2-18. Sense Pads

The coupling capacitor plates connected to a line driver (ROS word) are connected differently. They are laid out in parallel on a Mylar* strip. Figure 2-19a shows a ROS word for our block diagram before it is programmed and Figure 2-19b shows the ROS word after it is programmed for the first statement.

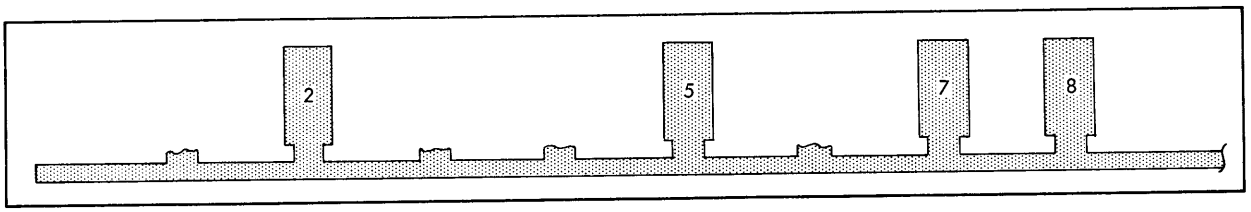
In Figure 2-20 the Mylar strips with our first and second statements programmed are shown placed over the sense pads. When the line driver for the first statement is impulsed, that function is performed. By adding more ROS words and drivers, more functions can be done.

In the 2030, the Mylar strips are called ROS cards, and each card contains 12 ROS words. The capacitor plates on the ROS cards are made of copper strips or silver ink.

*Trademark of E. I. du Pont de Nemours & Co. Inc.



(a) ROS-Word



(b) ROS-Word Programmed to Perform Logic Function:

Readout R, gate the output through the logic unit and read in S.

Figure 2-19. Programmed ROS Word

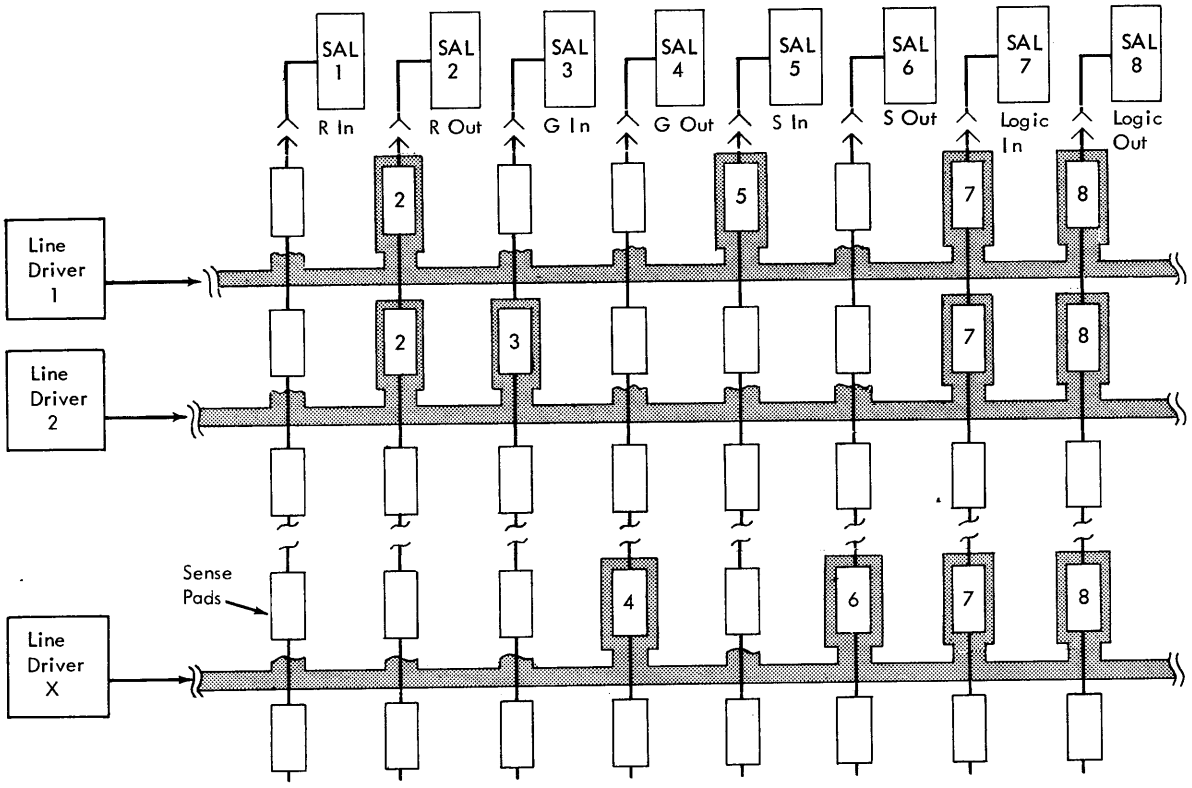


Figure 2-20. ROS Words

ROS CARD

- The primary unit of the CROS is the ROS word.
- The 2030 ROS word is 60 bits wide.
- There are 12 ROS words on each ROS card.

Figure 2-21 shows a ROS card. The primary unit of CROS is the ROS word; in the 2030 each word is 60 bits wide and there are 12 words per card. The words are positioned on a sheet of Mylar exactly the same size as an 80-column card. Each bit position is one plate of a capacitor and the plates are positioned so they coincide with the normal punching position of a card. The plates are connected to a horizontal line running from the column 1 and to the column 80 end of the card. This allows the card to be punched on existing punched-card equipment such as the IBM 24, 514, or 1402.

If the plate is punched out, we do not have a bit. Therefore, if we wish to have a control line active for a certain ROS word, we do not punch that position. When the card is placed next to the ROS board, the elongated tab on the card contacts the drive tab on the board.

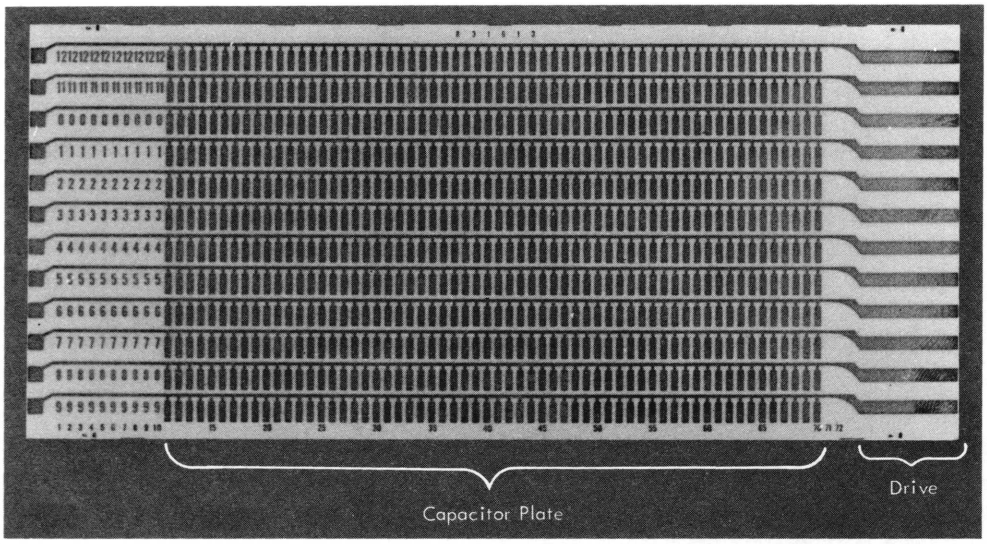


Figure 2-21. ROS Document

ROS MODULE AND ROS BOARD

- There are 43 ROS boards per full 4K ROS module.
- Depending on the capacity required, some ROS modules may contain fewer than 43 boards.
- Each ROS board has 8-ROS card positions.

A 4K ROS module contains 42 ROS boards for a total of 4,032 ROS words (Figure 2-22) plus one spare board to be used if a board fails.

The sense pad for the same bit position of each word is connected by a vertical line on the board. This line feeds the sense amplifier for that bit.

The ROS card is held against the ROS board by an air bag. Air pressure is applied to the bag which in turn forces the card in contact with the board.

The design of the board allows the boards to be placed in the ROS module so the drive-line connectors for the even boards are at the top, and at the bottom for the odd boards. All sense lines are routed from one end of the module.

The ROS board has a capacitor plate for each capacitor plate on the ROS card (Figure 2-23). The capacitor plates on the ROS board are called sense pads. Both sides of the ROS board have identical sense pad patterns, so we can have 8 (0-7) ROS cards of 12 words apiece, for a total of 96 ROS words per board.

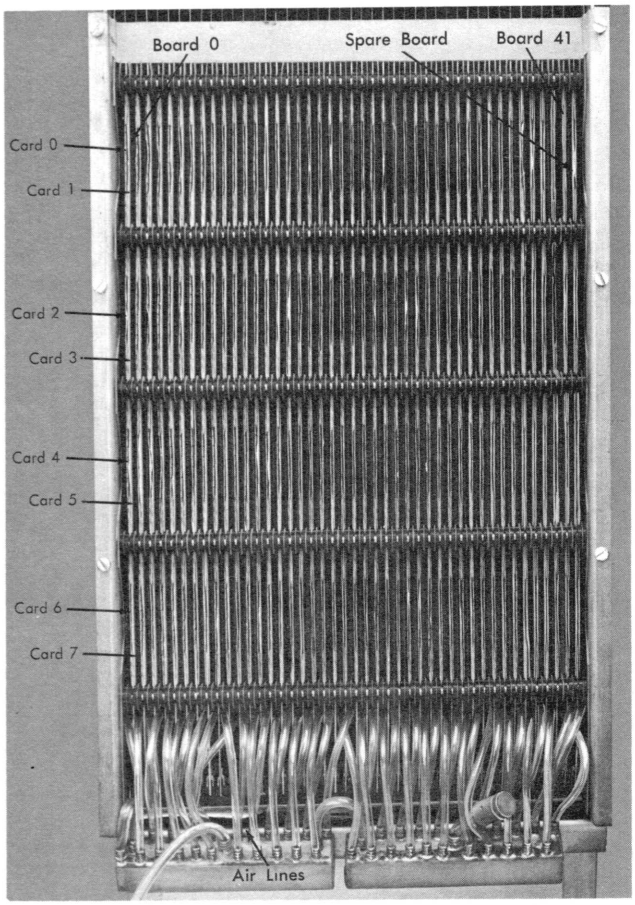


Figure 2-22 ROS Module, Front View

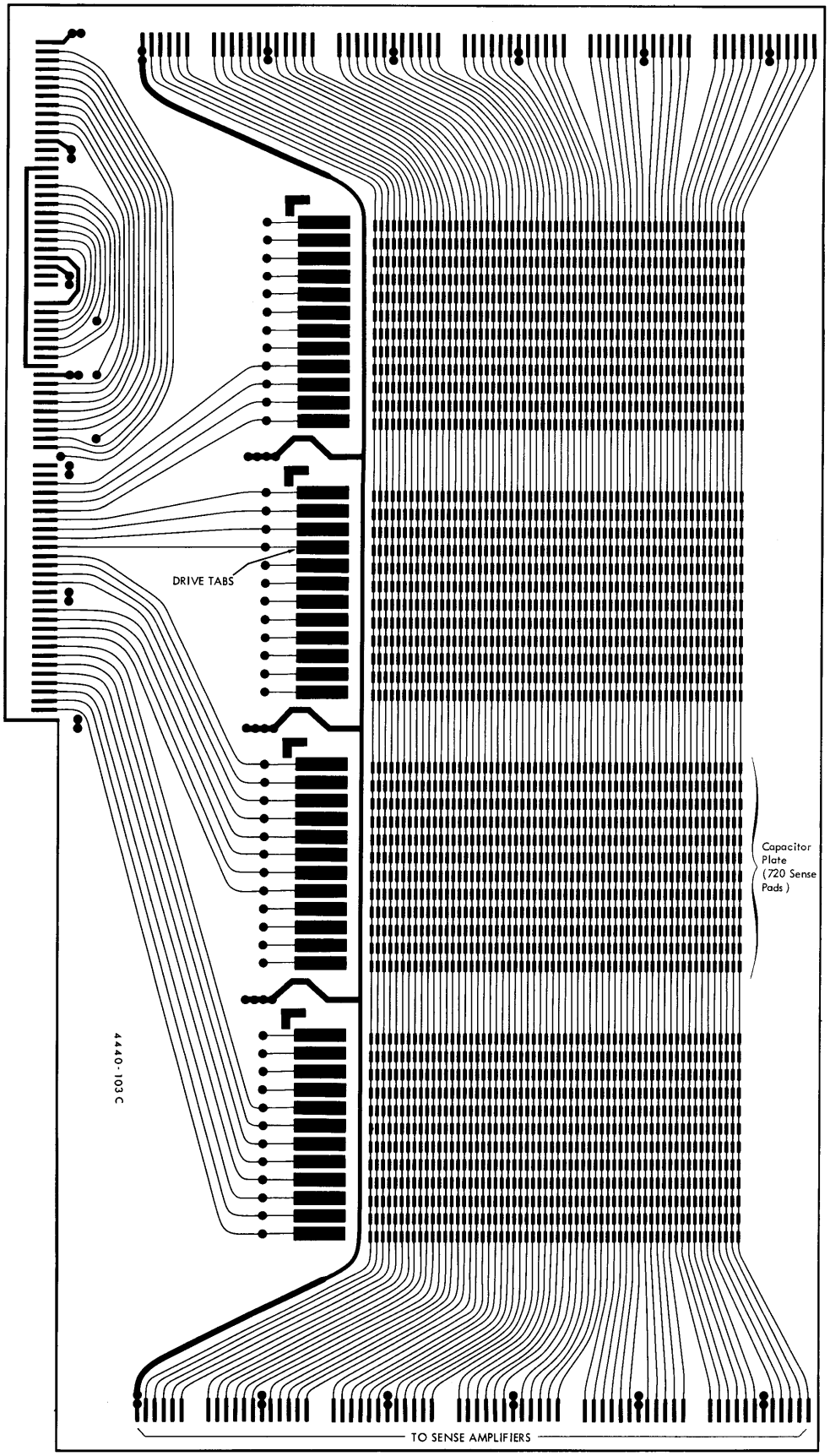


Figure 2-23 ROS Board

ROS DATA FLOW

- The W- and X-registers are used to address ROS.
- A ROS word is read out of each machine cycle.
- The information in some of the SAL's is transferred to control registers to be held because of timing conditions.
- The outputs of the SAL's and control registers are decoded and routed to control the function required by the ROS word.

The ROS address registers (W and X registers) are set at the first of the ROS cycle and the outputs of the registers are decoded to select one ROS board and two ROS words on the board. The two words are read out and one of the words is selected and set into the SAL's at a given time in the cycle. The outputs of some SAL's are needed longer than the SAL's are set, so the information is transferred to control registers. The output of the remaining SAL's and the control registers are decoded and routed to allow a logical operation to be performed (Figure 2-24).

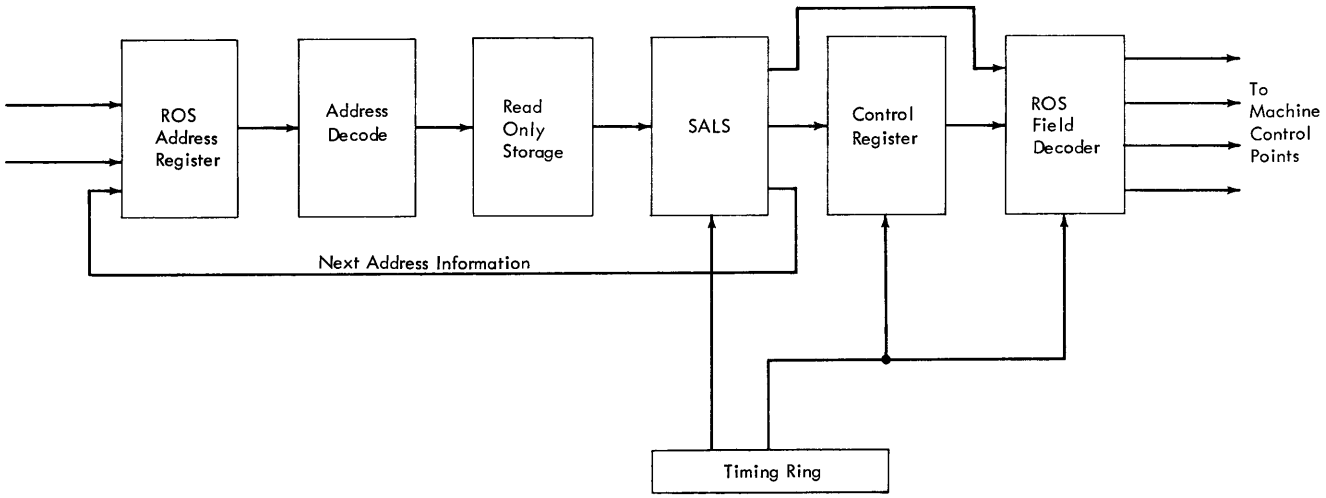


Figure 2-24. ROS Data Flow

ROS CONTROLS

- The controls for ROS include hardware for:
  1. Addressing ROS.
  2. Sensing the output of ROS.
  3. Timing for ROS.

ROS ADDRESSING (4K MODULE)

- The ROAR (Read Only Address Register), W- and X-Registers, address is decoded to:
  1. Select one out of 42 ROS boards.
  2. Select one out of 48 drivers for that board.
  3. Select one out of 2 ROS word read from that board.

Figure 2-25 shows the overall addressing method for ROS. For now, we will take sections of the total picture and explain them and then tie them together.

We know that each ROS board has 8 ROS cards and each card has 12 words of 60 bits each. Therefore, we have 96 ROS words on each board, and all are used. In a 4K module the addresses are sequential, board 0 has addresses 0 to 95 (decimal number), and board 1 has addresses 96 to 191 and etc. through board 41 with addresses 3936 to 4031. Because of the electrical connections on a ROS board, a ROAR decode selects two ROS words. However, only one of these words is gated to the SAL's. Since there are 96 words on a ROS board and two words are addressed each time, 48 drivers are needed to read out the 96 words.

Figure 2-26 represents the 48 drivers for one board and the driver's connections. The drivers are physically located on two small cards (driver card A and driver card B) and connected to the ROS board from the rear side of the ROS module (Figure 2-27). There are 24 electrically connected drivers on each small card. Each driver is a one transistor circuit. Let's consider the connection to the transistors as shown in Figure 2-28.

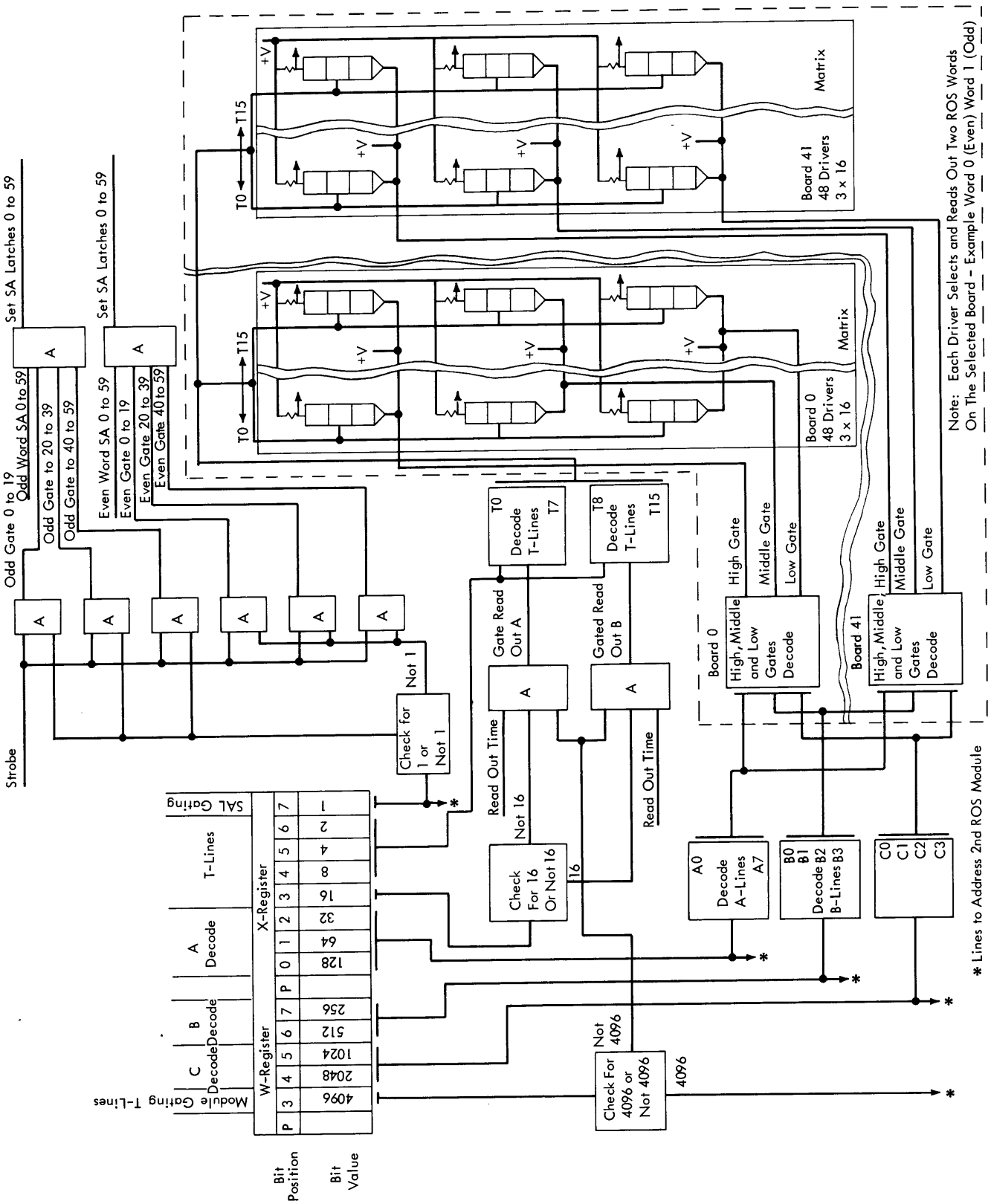


Figure 2-25. ROS Addressing, Logical

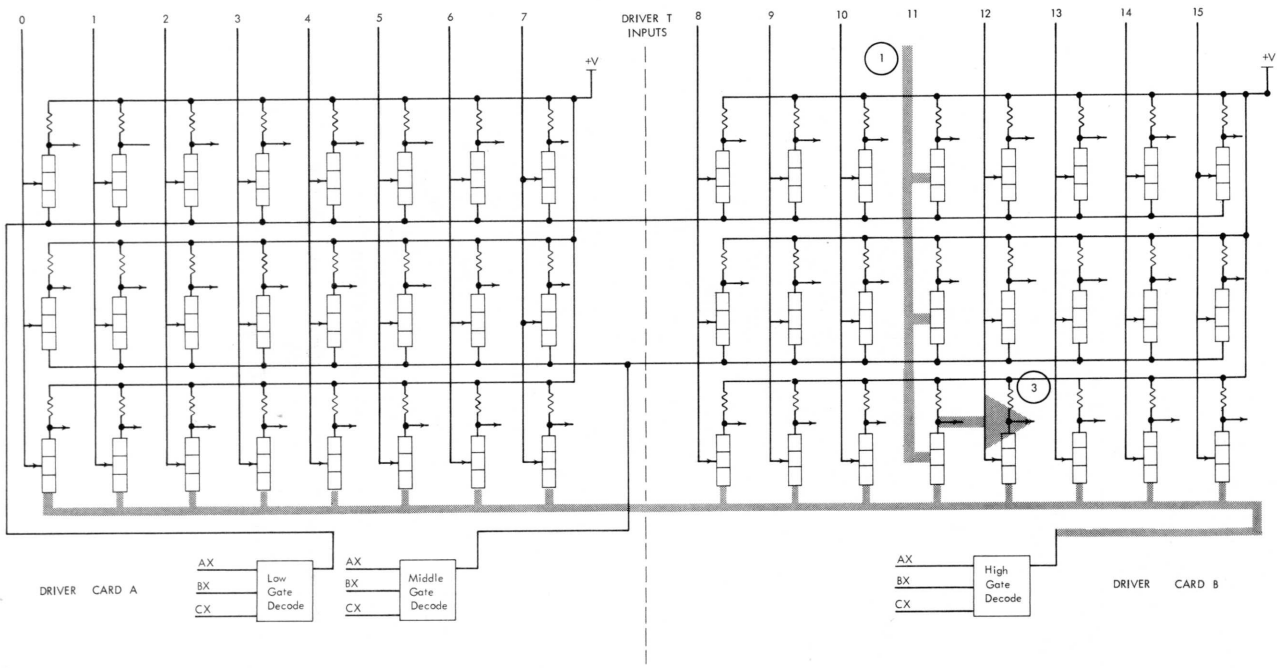


Figure 2-26. Driver Cards

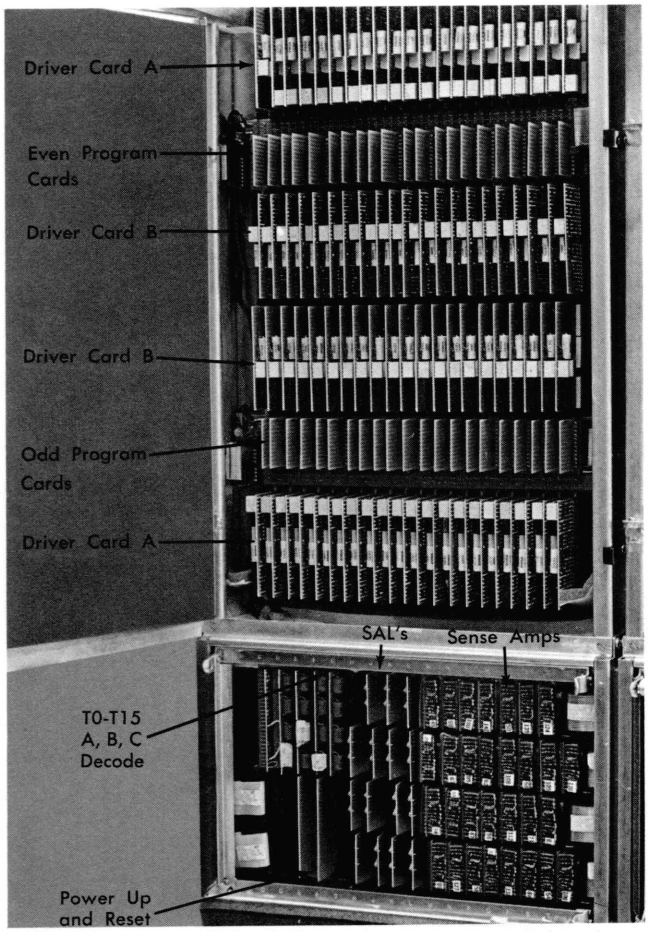


Figure 2-27. ROS Module, Rear View

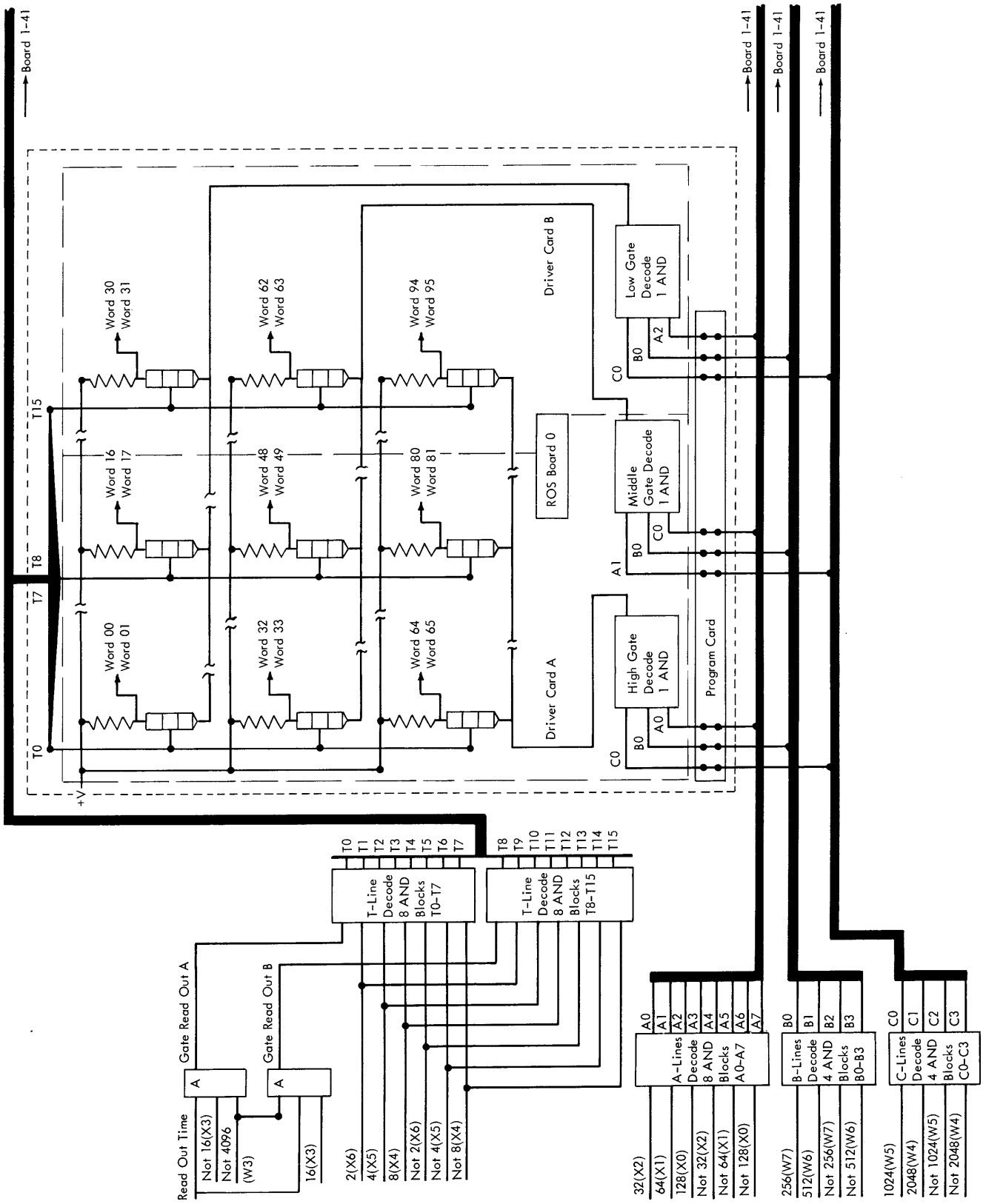


Figure 2-28. Driver Selection

1. The Base: Three driver bases are common. The darkened line that connects these three drivers is a T input line. Since one T line feed three drivers, there must be 16 T lines going to each ROS board.

2. The Emitter: Sixteen drivers have their emitters commoned. The darkened line that connects these 16 drivers is a driven gate decode line. Notice that the gate decoders are on the driver cards. Two decoders are on driver card A, and one decoder is on driver card B in reference to each ROS board.

3. The Collector: The output from the collector drives two ROS words.

The T-lines for a 4K module are developed from 16 four-input AND circuits. Only one T-line is active at a time. Both the bit and not-bit lines are routed from the CPU to condition the AND inputs. Three of the four inputs come from the X-register positions 4, 5, and 6. The fourth input is Gate Read Out A or Gate Read Out B, these lines are developed from the condition of X-register position 3 and W-register position 3 AND with Read Out Time. The T-lines are routed to all 42 ROS boards. Note that if W3 is on, the second 4K module is selected and another address decode network is used. Addresses for the first 4K are 0000 through 4031 (decimal) and for the second 4K are 4096 through 8127 (decimal).

At this time we have one T-line active and this line conditions the bases of three drivers on each ROS board. The next requirement is to condition just one driver on one ROS board.

Each ROS board has three driver gates (high, middle, and low). Each gate on a board is commoned to 16 drivers on that board. So by activating one gate on one ROS board, we can select one driver.

The selection of the gate is controlled by three lines (A, B, and C) which are developed from the X-register 0, 1, and 2 positions and the W-register 4, 5, 6, and 7 positions. By looking at Figure 2-28, we can see that eight A-lines (A0 through A7) are decoded from the bit and not bit lines of the X-register positions 0, 1, and 2. We also have four B-lines and four C-lines. The B-lines are developed by ANDing the W-register positions 6 and 7 bit and not-bit lines. The C-lines use the W-register positions 4 and 5 bit and not-bit lines ANDed together.

From Figure 2-28 we can tell that the high gate for ROS board zero is developed by ANDed A0, B0, C0. By using the address table in Figure 2-29, we can determine what five lines are used to form the three gates for any ROS board. The lines from the A, B, and C lines busses are routed from the bus to the gate decode circuits through a program card. A program card is nothing more than a pluggable card to jumper a line from one place to another.

		A0	A1	A2	A3	A4	A5	A6	A7	
C0	B0	0000	0	0095	1	0191	2	0255		
	B1	2	0287	3	0383	4	0479	5	0511	
	B2	5	0575	6	0671	7			0767	
	B3	8	0863	9	0959	10			1023	
C1	B0	10	1055	11	1151	12	1247	13	1279	
	B1	13	1343	14	1439	15			1535	
	B2	16	1631	17	1727	18			1791	
	B3	18	1823	19	1919	20	2015	21	2047	
C2	B0	21	2111	22	2207	23			2303	
	B1	24	2399	25	2495	26			2559	
	B2	26	2591	27	2687	28	2783	29	2815	
	B3	29	2879	30	2975	31			3071	
C3	B0	32	3167	33	3263	34			3327	
	B1	34	3359	35	3455	36	3551	37	3583	
	B2	37	3647	38	3743	39			3839	
	B3	40	3935	41	4031					

Figure 2-29. Address Table

When a driver is fully selected, it provides the drive to two ROS words. An even address in ROAR selects that address and the next high-order odd address. An odd address in ROAR selects that address and the next lower even address. As an example: If ROAR contains the decimal address 0063, this address and address 0062 are selected. However, only the ROS word at address 0063 is gated to the SAL's.

These two ROS words are read out to sense amplifiers.

SENSING AND DECODING ROS OUTPUT

- There are 120 sense amplifiers used to sense the two ROS words read out of the selected 4K module.
- There are 60 SAL's (sense amplifier latches) used to store the selected ROS word.

Each ROS word has 60 bits and there are two words read in each ROS cycle. Therefore, there are 120 sense amplifiers. Depending on the condition of X-register position 7, bit or not-bit, one set of sense amplifiers (Figure 2-30) are gated to the SAL at a given time (Strobe time) in the cycle. The information is held in the latches until the reset pulse is activated.

Note: The second 4K module has another 120 sense amplifiers which are routed to the same SAL and are gated by the condition of X-4. Remember only one driver is activated in either module. The module selected depends on the condition of W-3.

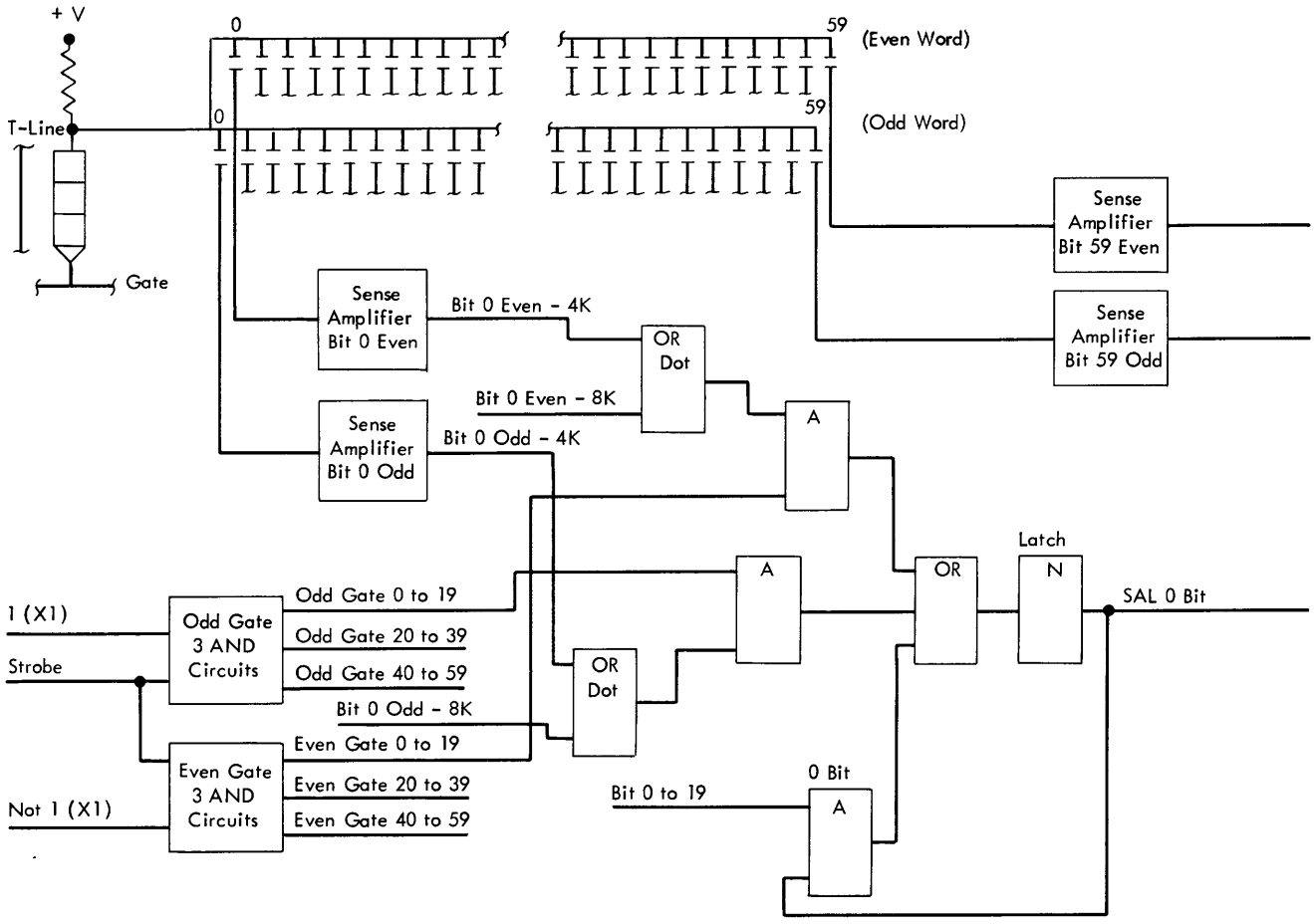


Figure 2-30. ROS Sensing

CONTROL REGISTERS

- The outputs of SAL's 34 through 51 are transferred to control registers.

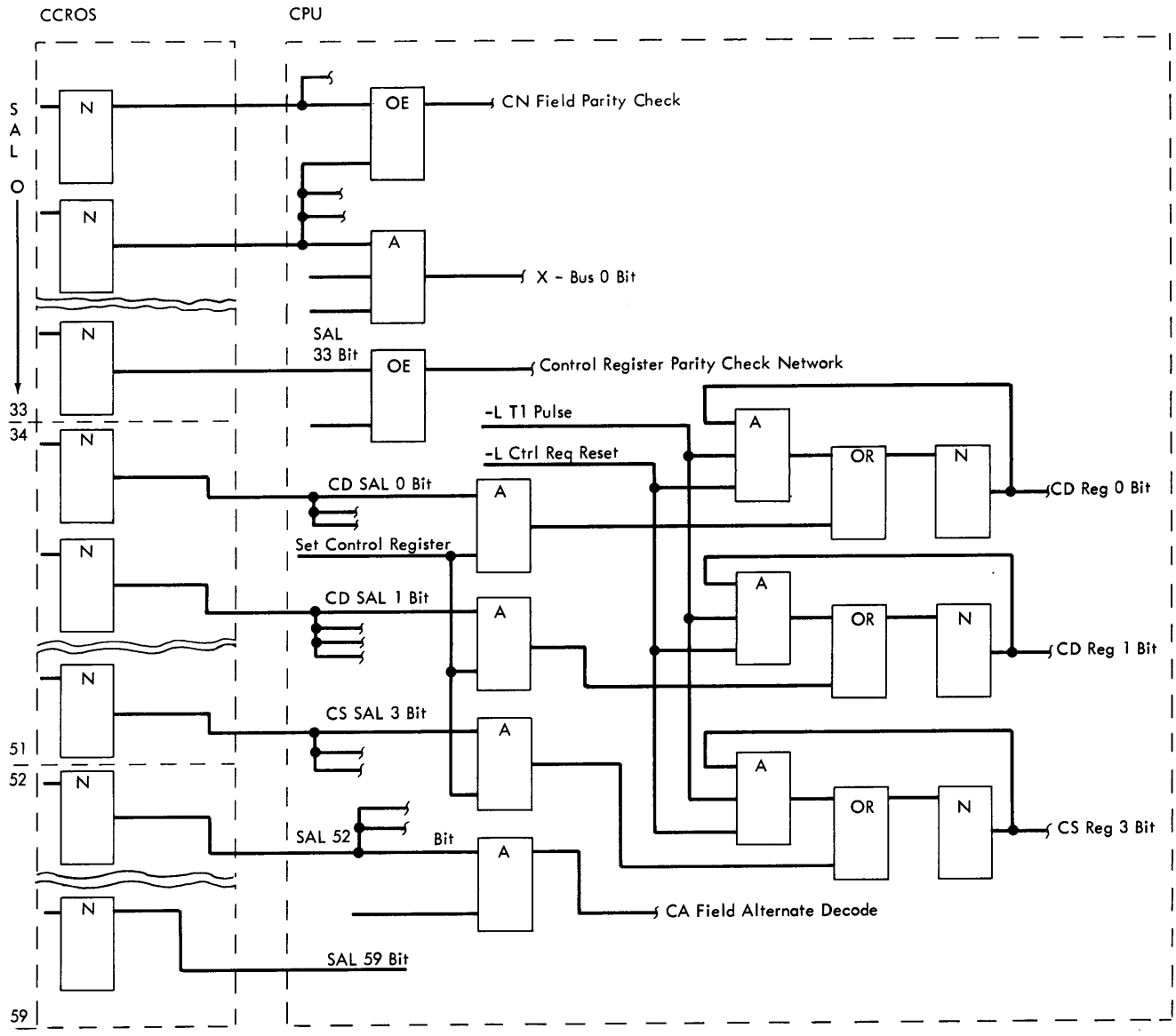


Figure 2-31. ROS SAL Output and Control Registers

Because of timing conditions, the information in SAL 34 through 51 is transferred to another group of latches called control registers (Figure 2-31).

The outputs of the SAL's are decoded and used during the first part of a ROS cycle and the outputs of the control registers are decoded and used during the latter part of the cycle.

BASIC ROS TIMING

- ROAR is set using a P1 pulse and selected set inputs.
- The CPU CROS GO pulse is used to develop a ROAR decode pulse and a SAL's reset pulse.
- The SAL's are good by T4 time.
- The control registers are reset during T1 time and set during T2 time.

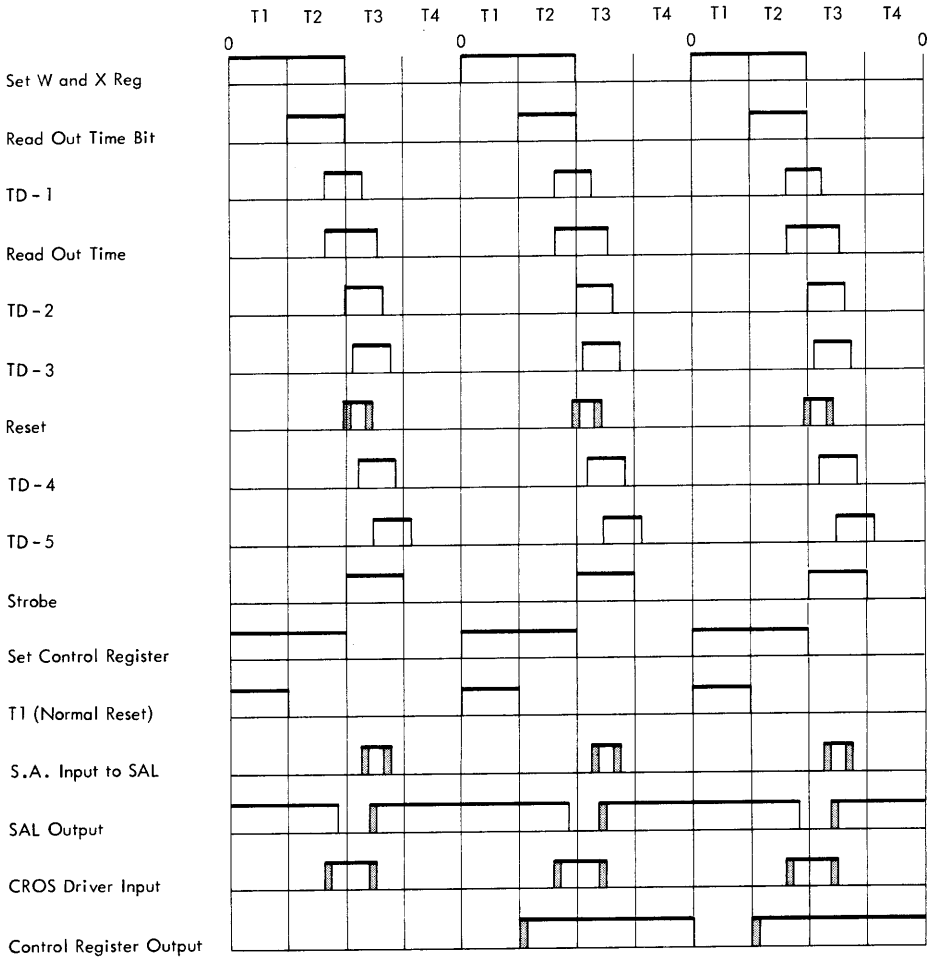


Figure 2-32. ROS Timing, Basic

Figure 2-32 shows the basic timings associated with ROS. Each cycle is divided by the CPU timing T1, T2, T3, and T4. The figure shows three ROS cycles. The first cycle represents the time to set ROAR, because before any ROS word can be read out and executed, there must be an address in ROAR. The pulse to condition the set of ROAR is a P1 pulse, but the information to set ROAR is active by the first part of T1, so ROAR is set during T1 time. Once ROAR has been set and the latches have settled down, the output of the latches can be used to bring up the gates and drive lines for addressing ROS.

This is done at T2 time using the CROS GO pulse from the CPU. The CROS GO pulse becomes a line called Read Out Time Bit and is delayed to bring up the Read Out Time to condition the decode of the ROAR output lines and a Reset pulse for the reset of the SAL's.

The decode of the ROAR output lines conditions one driver on one ROS board to impulse two ROS words. The output, representing the bits of the ROS word, of the capacitors is routed to 120 sense amplifiers. The CPU routes a line called CCROS Strobe to CROS at T3 time; this line then

is called Strobe and is ANDed with the condition of the X-register position 1 to select the correct sense amplifier to be routed to the SAL's.

The information in the SAL's is good from about T4 time of the cycle of which the ROS word was read out until the SAL's are reset during the next ROS cycle. Because of timing conditions, some of the information in the SAL's is needed after reset time. Therefore, at P1, the information is routed to the control registers. The control registers are reset during T1 time of the cycle; they are set during T2 time of the cycle.
Note: Should the clock be stopped at T4 time, the SAL's would contain the information of the ROS word just addressed and the control registers would contain the information from the previous ROS word.

SETTING OF ROAR

- The address in ROAR may be stored in one of two backup ROARs.
- The address to be set into ROAR may come from many sources.

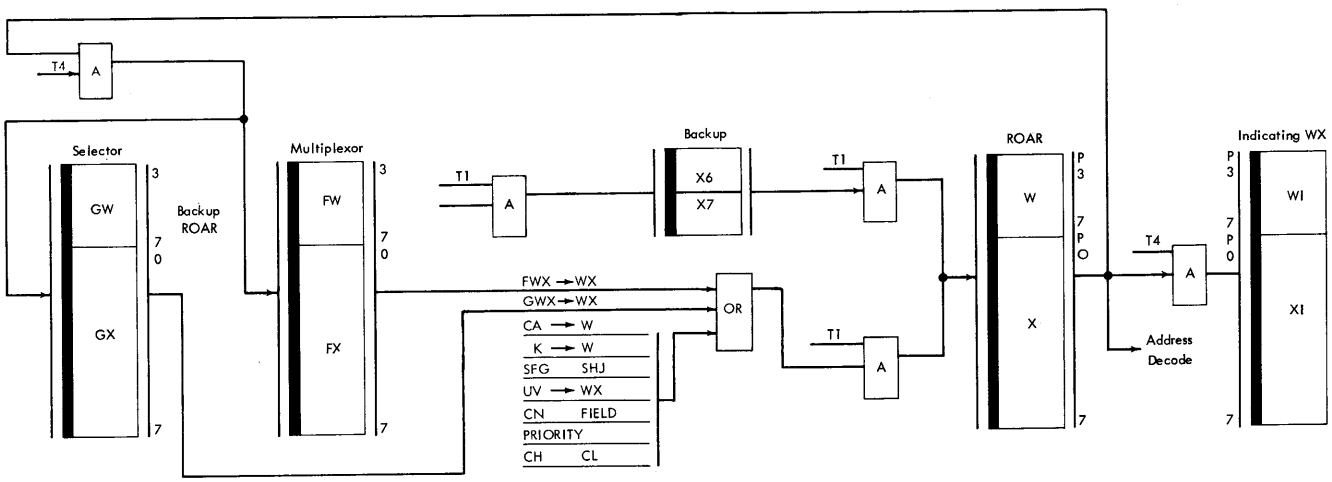


Figure 2-33. ROAR Controls

Figure 2-33 shows ROAR and two backup ROAR's, one for selector channel and one for multiplexor channel. ROAR is set under control of the microprogram by many different sources. Some of the microprogram mnemonics are shown in Figure 2-33 and are discussed in detail under the microprogram section.

ROAR is set at T1 time from one of the inputs. Should the selector channel or multiplexor channel cause a break-in, the

address in ROAR is transferred to either GW and GX or FW and FX registers at T4 time. When the channel operation is completed, the microprogram transfers back to where the interrupt occurred and the address in the backup ROAR is transferred to ROAR allowing the original program to continue.

INDICATING ROAR

- The console lights for a ROS word address are controlled by an indicating ROAR.

The indicating W- and X-registers are shown in Figure 2-33. Because of timing considerations, the output from ROAR is gated to the indicating ROAR latches at T4 time. Since the clock stops at the end of T4 time, the address displayed is the address of the ROS word we have just read out.

CROS LOCATION

Figure 2-34 shows the addresses as they appear on ROS board 0 and 1. Even addresses are on the left side of a ROS board; odd addresses on the right. As an example: address 0017 (decimal) is on ROS card number 1. Card 1 is the uppermost card on the right side of board 0. Figure 2-35 shows a 4K ROS module as viewed from the left side of the console. The ROS cards are inserted from this side.

From our knowledge of a ROS board and the fact that words 0000 and 0001 are read out at the same time, we can see that one word is read from each side of the ROS board each cycle.

Figure 2-36 shows the layout of the ROS card. The drive tabs are located on the column 80 end of the card. The card is inserted into the module, drive tab first. Because the capacitor plates on the cards must be next to the ROS board, all the ROS cards that contain words at odd addresses must first be flipped over before insertion.

If all the ROS cards are viewed with column-80 of the card to the right, then it follows that the odd addresses are numbered from the 9-edge to the 12-edge of the card. The cards that contain the ROS words at even addresses are numbered from top to bottom as viewed (Figure 2-37).

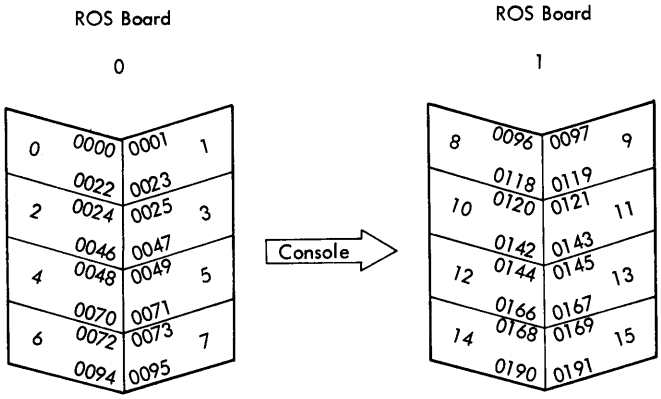


Figure 2-34. ROS Document Addresses

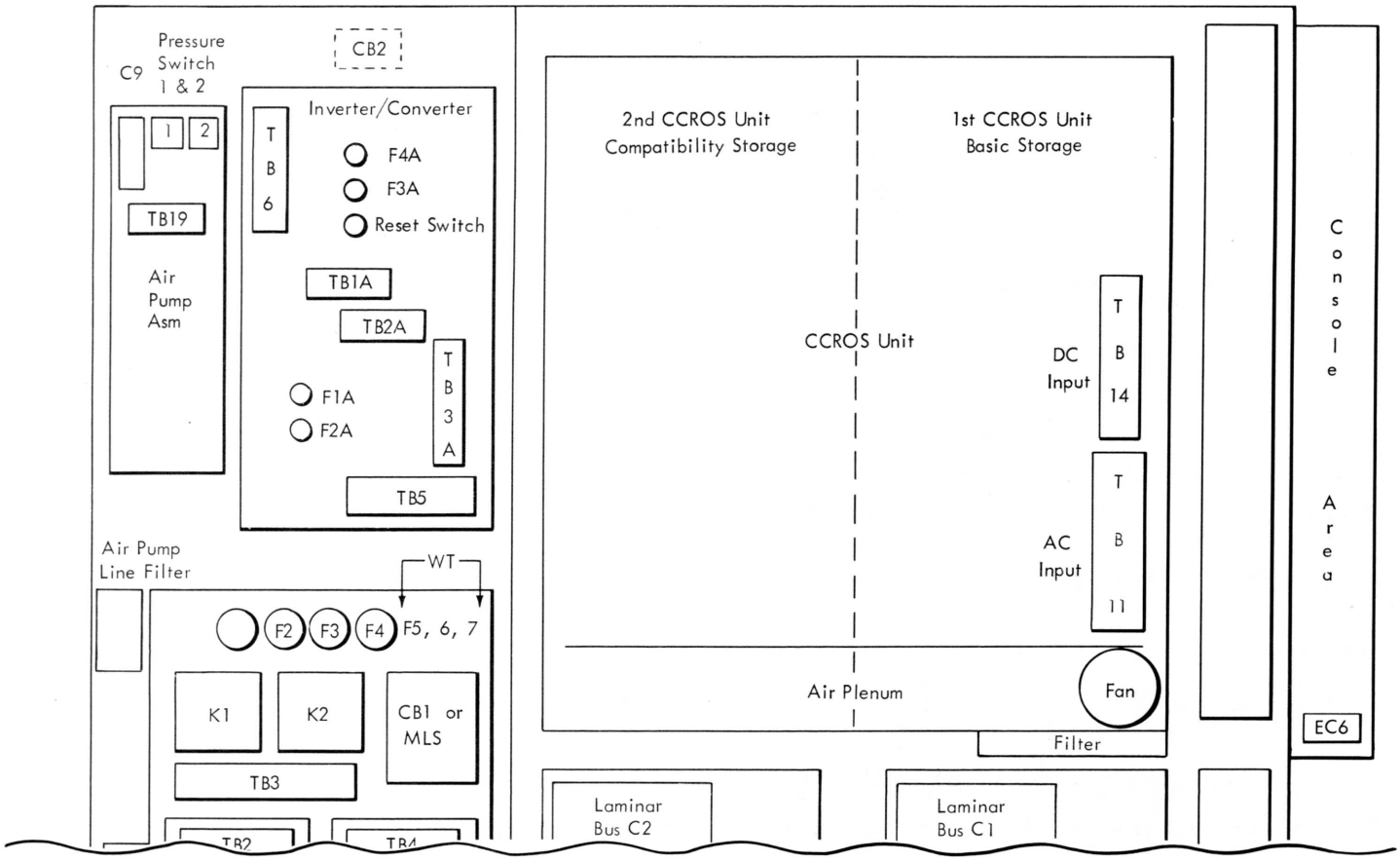


Figure 2-35. IBM 2030, Left Side

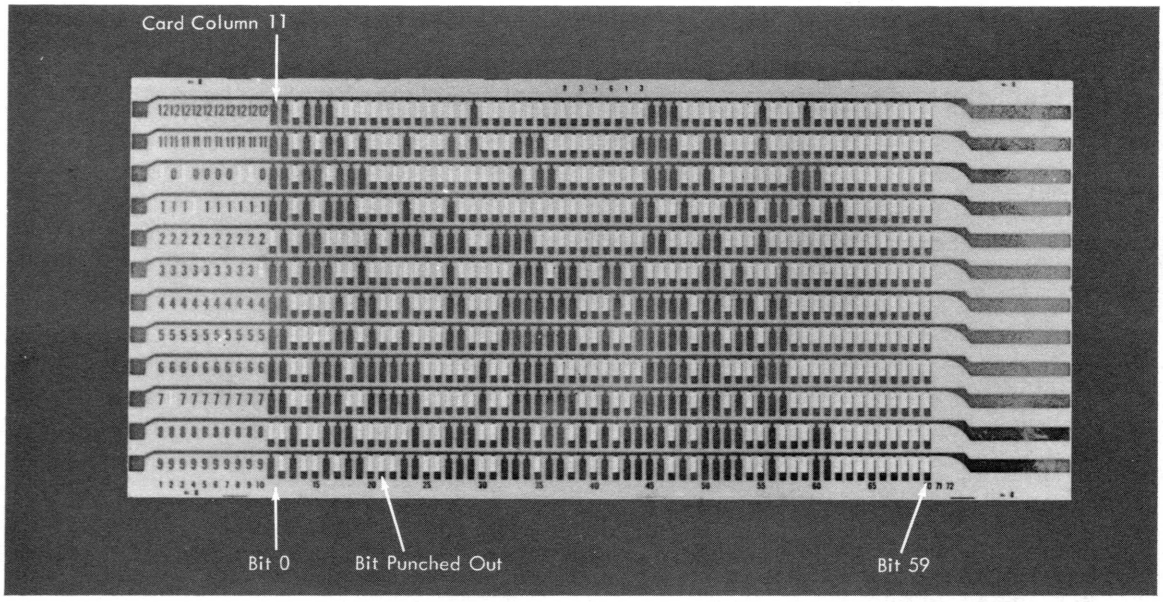


Figure 2-36. ROS Document Layout

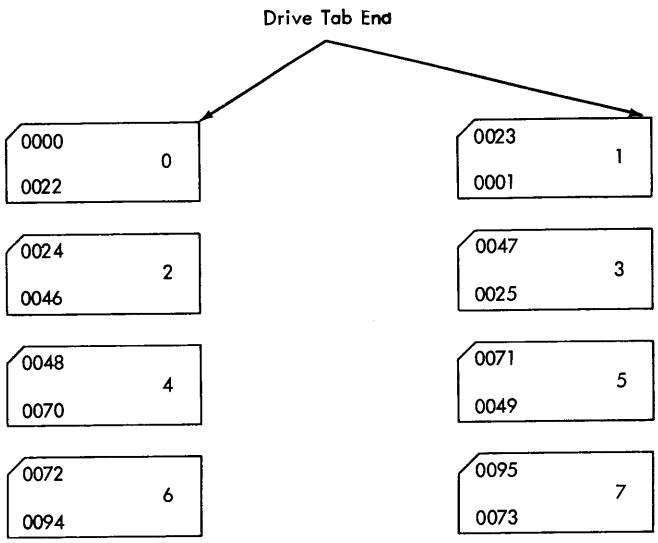


Figure 2-37. ROS Word Numbering

MICROPROGRAM

- The microprogram is used to control the function of the 2030.
- Each machine cycle is controlled by one microprogram word.
- A microprogram word is punched in a ROS card and becomes a ROS word.

In all computers it is necessary to have some method to perform a sequence of logical steps. The 2030 uses a microprogram. Within the microprogram, the microprogram word is the functional statement. The microprogram word is punched in a ROS card to form a ROS word.

A ROS word is selected by the decode of the address in ROAR (Read Only Address Register) and the ROS word contents are decoded to activate control points in the system. The ROS word consists of specific fields programmed to perform a logic statement. The activated word sends back part of the next address for ROS to ROAR. Coupled with branch control (machine status test), the partial address forms the complete address of the next ROS word. To read and understand the ROS word, we must know what the ROS word can contain and what format is used to write the word.

ROS Word Control Fields

- The ROS word used in the 2030 is 60 bits wide.
- The ROS word is divided into control fields.

The 60-bit ROS word is divided into control fields (Figure 2-38) and these fields can be separated into six broad groups:

1. Function control CA, CF, CB, CG, CC, CV, CD, CK
2. Main and auxiliary storage CM, CU
3. Branching and ROS address CN, CH, CL
4. Set and reset of status condition CS
5. Alternate AA, AS, AK
6. Parity for different sections of the control fields

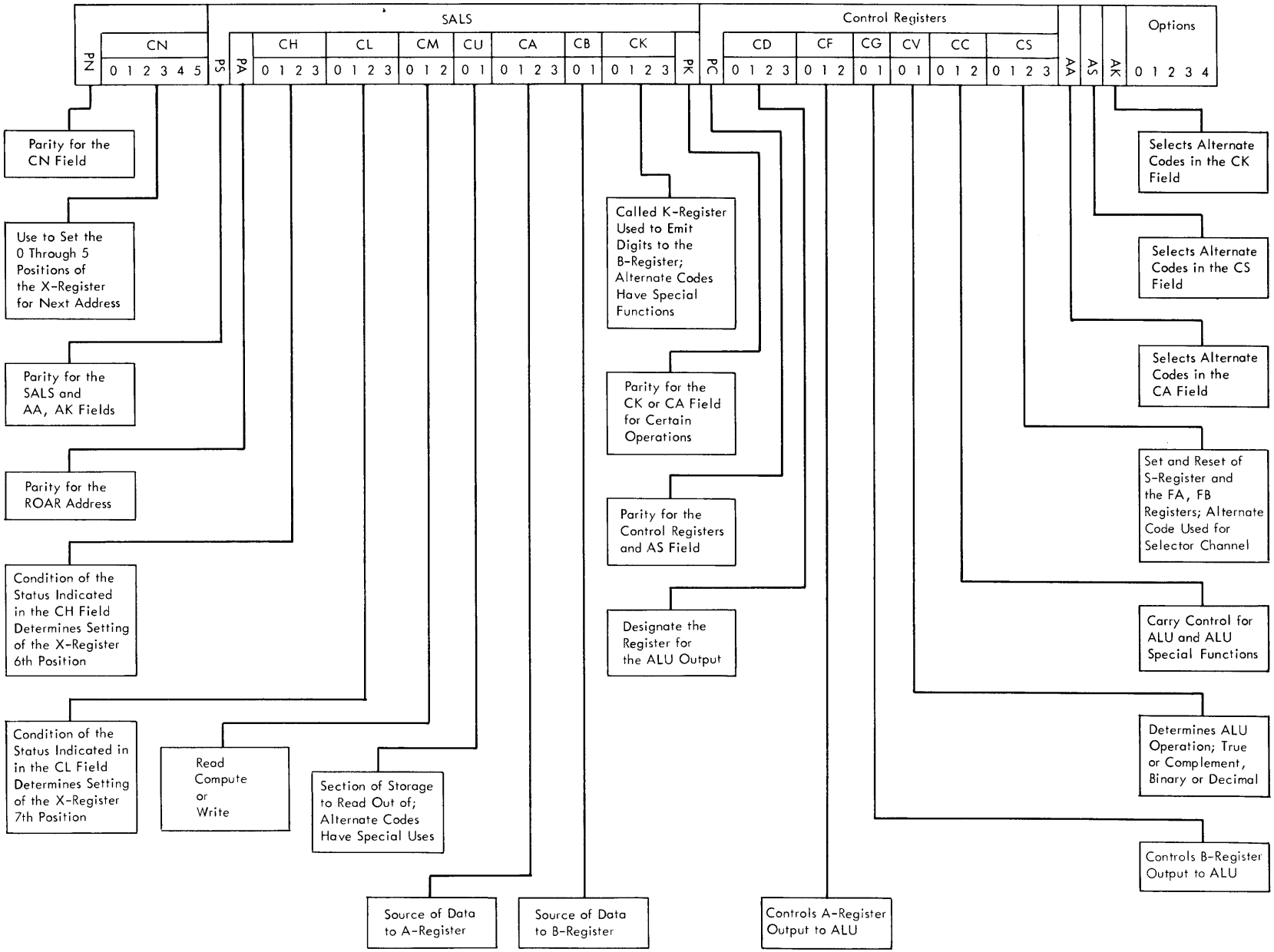


Figure 2-38. ROS Control Fields

Notice the control fields vary in numbers of bit positions. Example: the CU field is two bits wide and the CD field is four bits wide. If the field is two bits wide, we can set and decode four combinations: 0-00, 1-01, 2-10, 3-11. A three position field can be set and decoded in eight combinations, 0-000 through 7-111, and a 4-bit field has 16 combinations, 0-0000 through F-1111.

FUNCTION CONTROL. The function control fields (Figure 2-39) are used to control all data movement in the CPU and the ALU. ALL DATA MOVEMENT IS THROUGH THE ALU. The function control fields can be subdivided into four groups.

1. Source to the A-register and control of the A-register output to the ALU; CA, CF.
2. Source to the B-register and control of the B-register output to the ALU; CB, CK, CG.
3. Function and control of the ALU; CV, CC.
4. Destination of the ALU output; CD

Source to the A-register (CA): This 4-bit

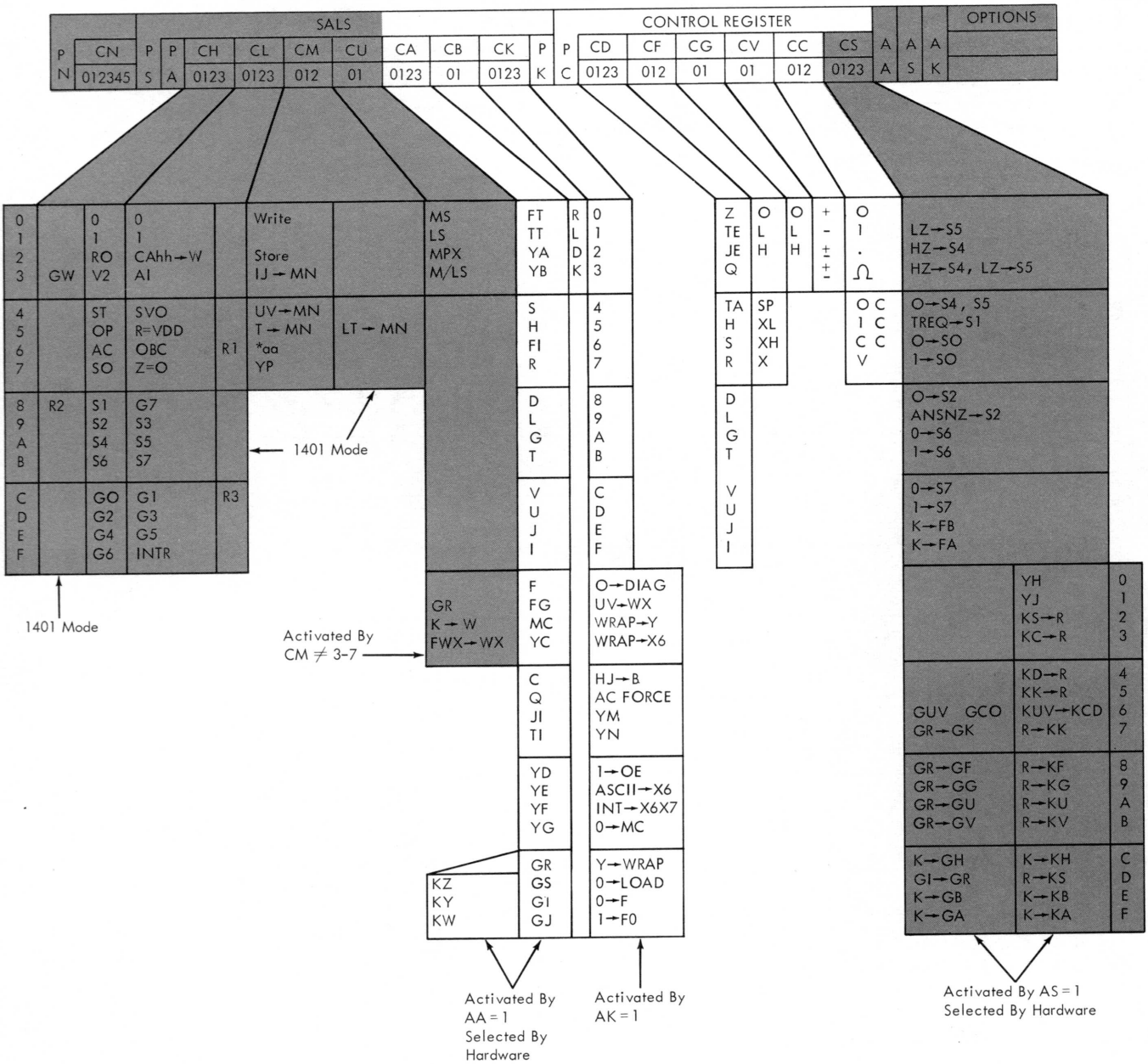


Figure 2-39. ROS Function Control Fields

field is decoded to select the data to be routed to the A-register. It can be decoded to 16 combinations, but by using the AA field (explained later), the CA field has 16 alternate sources to select. This makes 32 combinations for the A-register source.

Control of the A-register output (CF): This 3-bit field controls the method that the data from the A-register is presented to the ALU. The field is essentially bit significant. There are eight bits routed to the ALU from the A-register; we can block all of them, block the four high bits, block the four low bits, or allow all eight bits.

If we block any bits, zeros are routed to the ALU in place of the blocked bits. We can also cross the four low bits with the four high bits or cross and block four bits. Figure 2-40 shows: if the 2 bit is on, the four low bits are allowed, if the 1 bit is on the four high bits are allowed and if the 0 bit is on the high and low bits are crossed.

Cross the A-Register Four High Bits with the Four Low Bits	Block the A-Register Four Low Bits Replace with Four Zeros and Allow High Bits	Block the A-Register Four High Bits Replace with Four Zeros and Allow Low Bits	
Bit 0	Bit 1	Bit 2	
0	0	0	Block A-Register-Route Zeros to ALU
0	0	1	Block High Bits-Route 0000 and Low Bits
0	1	0	Block Low Bits-Route High Bits and 0000
0	1	1	Route A-Register to ALU
1	0	0	Conditional Machine Stop
1	0	1	Block High Bits-Route Low Bits and 0000
1	1	0	Block Low Bits-Route 0000 and High Bits
1	1	1	Route A-Register Crossing Low and High

Figure 2-40. CF Field Bit Significant

If both the 1 and 2 bits are off, the information in the A-register is blocked. Notice there are two possible conditions for this, all three bits off or just the 0-bit on. The condition of just the 0-bit on has been selected as the machine stop function since it did not serve any other useful purpose. The stop function is explained in greater detail later in this section.

Source to the B-register (CB): This 2-bit field is decoded to select the data to be routed to the B-register from either the R, L, D, or K register. The K-register is the CK field of the ROS word.

The K-register (CK): The K-register is also called the emit field or the constant field. This 4-bit field can be decoded to 16 combinations; there are 16 alternate combinations which are active when the AK field has a 1 bit.

The primary bit configuration can be used to emit a digit 0 through F. The same digit is presented to both the high and low four bits of the B-register. For example, the K-register has a 1 in it and the CB field decodes to route K-register to B-register, the 1 enters the high four bits and the low four bits giving us the number 11. By using the CG field, we can route to the ALU from the B-register the number 01, 10, 11, or 00. The W-register can be set from the CK field if desired.

The K-register can also be used to create an address to set in the N-register. This is explained in greater detail later in the section.

Control of the B-register output (CG): This 2-bit field controls how the data from the B-register is presented to the ALU. The operation is the same as the CF field except the B-register cannot be crossed. We can block the output and route eight zeros, or block either the high or low four bits and route zeros where the bits were blocked. The B-register output can be routed direct (both high and low four bits) to the ALU.

Control of ALU (CV): This 2-bit field decodes to select what type of arithmetic operation (true/complement and binary/decimal) is to be performed. The B-register input to the ALU is the true/complement side.

(CC): This 3-bit field decodes to control the carry-in and carry-out to the ALU and permits the setting of a carry-out into the carry latch. This field also decodes to control the AND, OR, and EXCLUSIVE OR function of the ALU.

MAIN AND AUXILIARY STORAGE CONTROL. The two ROS fields which control main and auxiliary storage are the CM and CU fields (Figure 2-41) and work in conjunction with each other. To understand the functions of the two fields, it is easier to explain the operation of the two fields together.

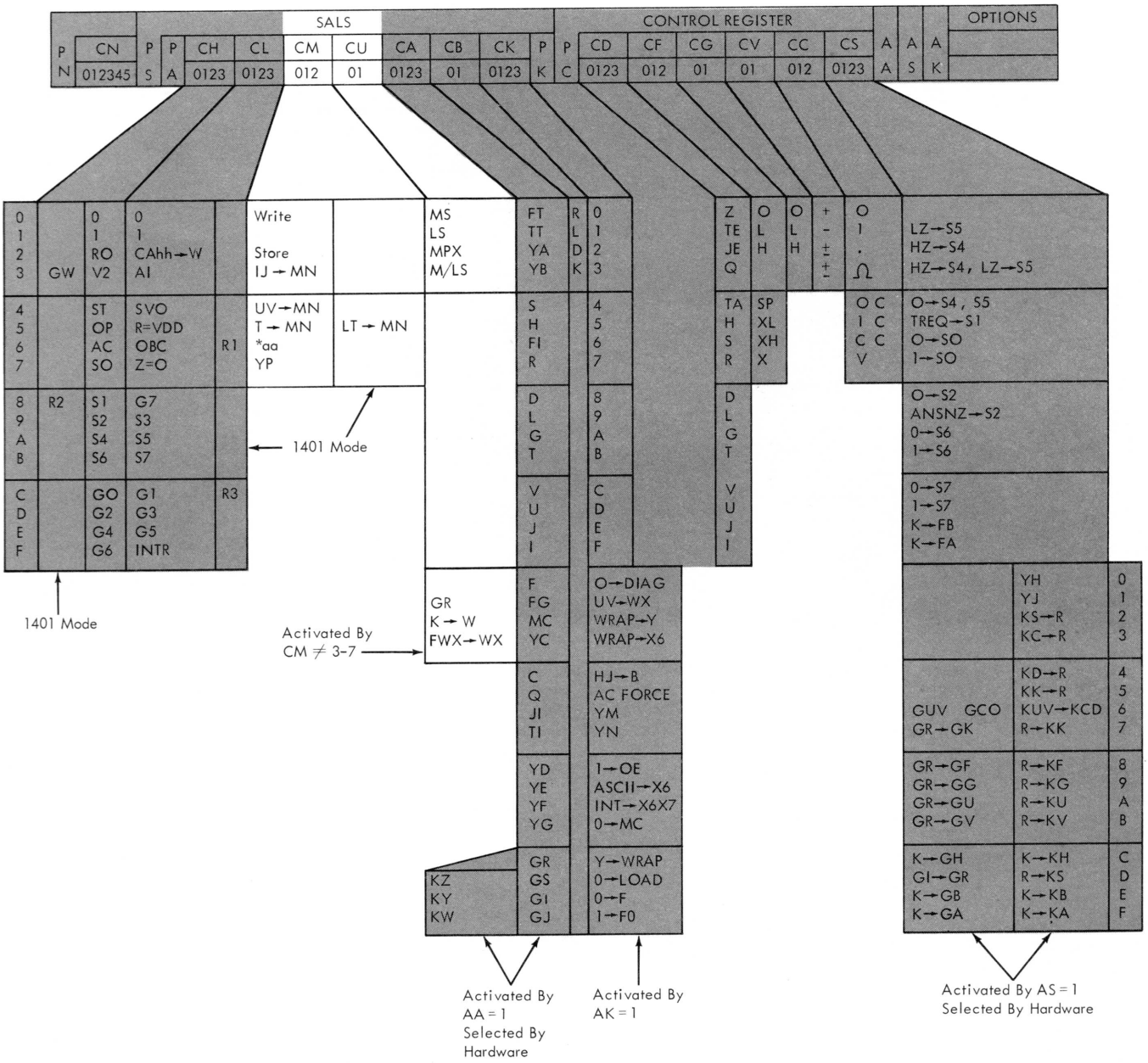


Figure 2-41. ROS Storage Control Fields

The 3-bit CM field decodes to select the type of operation - read-compute or write. The 2-bit CU field decodes to select what section of storage to operate in: main storage or auxiliary storage. Auxiliary storage includes local storage and the multiplexor storage blocks.

In the 2030, the four basic core storage cycles are:

Read, Write (R, W)

Read, Compute, Write (R, C, W)

Read, Store (R, S)

Read, Compute, Store (R, C, S)

Remember from the study of ROS hardware and timing, that the data from core storage is not ready for use until the beginning of the next ROS cycle. Therefore, if a read call is given, the next cycle must be a write, a store, or a compute cycle. Also, a write or store cycle should follow a read cycle within three ROS cycles. If this rule is not followed, it is possible to have an over-run condition of an I/O unit on the selector channel. Over-run is where

new data is ready but can not be accepted before more data is ready. There is an allow write latch on the 2030 which is used to recognize whether the last cycle was a read or a write. If a read is followed by a read, there will be a position in storage with all bits missing. This happens because the position read first had nothing written into it before its storage address was changed. If a write is followed by a write, the second write becomes a compute cycle because the allow write latch is off (set to allow a read cycle).

If the read cycle is followed by a write cycle, the data is set in the R-register and is routed to the core storage unit from the R-register during the write cycle. If the read cycle is followed by a store cycle, the output from core-storage is not used. Instead, new information is in the R-register at the end of the read cycle and is then written into core-storage during the store cycle.

If the read cycle is followed by a compute cycle, the output from core-storage during the read cycle is set into the R-register. During the next cycle, the information in the R-register may or may not be used in the computation. The next cycle is either a write or a store cycle and the R-register may contain the original information or the result of the computation. In any case, what is finally in the R-register is written in core-storage during the write or store cycle.

The core storage read-write control (CM): This 3-bit fields is decoded to determine if the cycle is a read, compute, or write cycle. A 0 or 2 decodes to a write cycle (2 is a store but brings up a write operation), a 1 is a compute cycle, while 3 through 7 are read cycles.

The section of core storage used (CU): This 2-bit field decodes to select which section of core storage is used during the read cycle. Write at the same address. The alternate decodes for the CU field are activated when writing by the CM field having a decode of 0, 1, or 2. The alternate decodes are explained later in this section.

Note: If the CU field is a 3 (M/LS), the operation must be checked further to see if main storage or local storage is to be used. This is done by checking the two high-order bits of the G-register which contain the op code during this time. If the two bits are 00, the op code format is RR and local storage is used. Any other combination of the two bits (01, 10, or 11) requires the use of main storage.

BRANCHING AND ROS ADDRESS. The complete ROS address is held in the W- and X- registers. The W- registers hold the five high-order positions of the ROS address and can be set by a ROS statement CAhh->W (detail on this ROS statement later) and the eight low-order positions of the ROS address are in the X- register. Normally the X- register is set from the CN, CH, and CL fields (Figure 2-42).

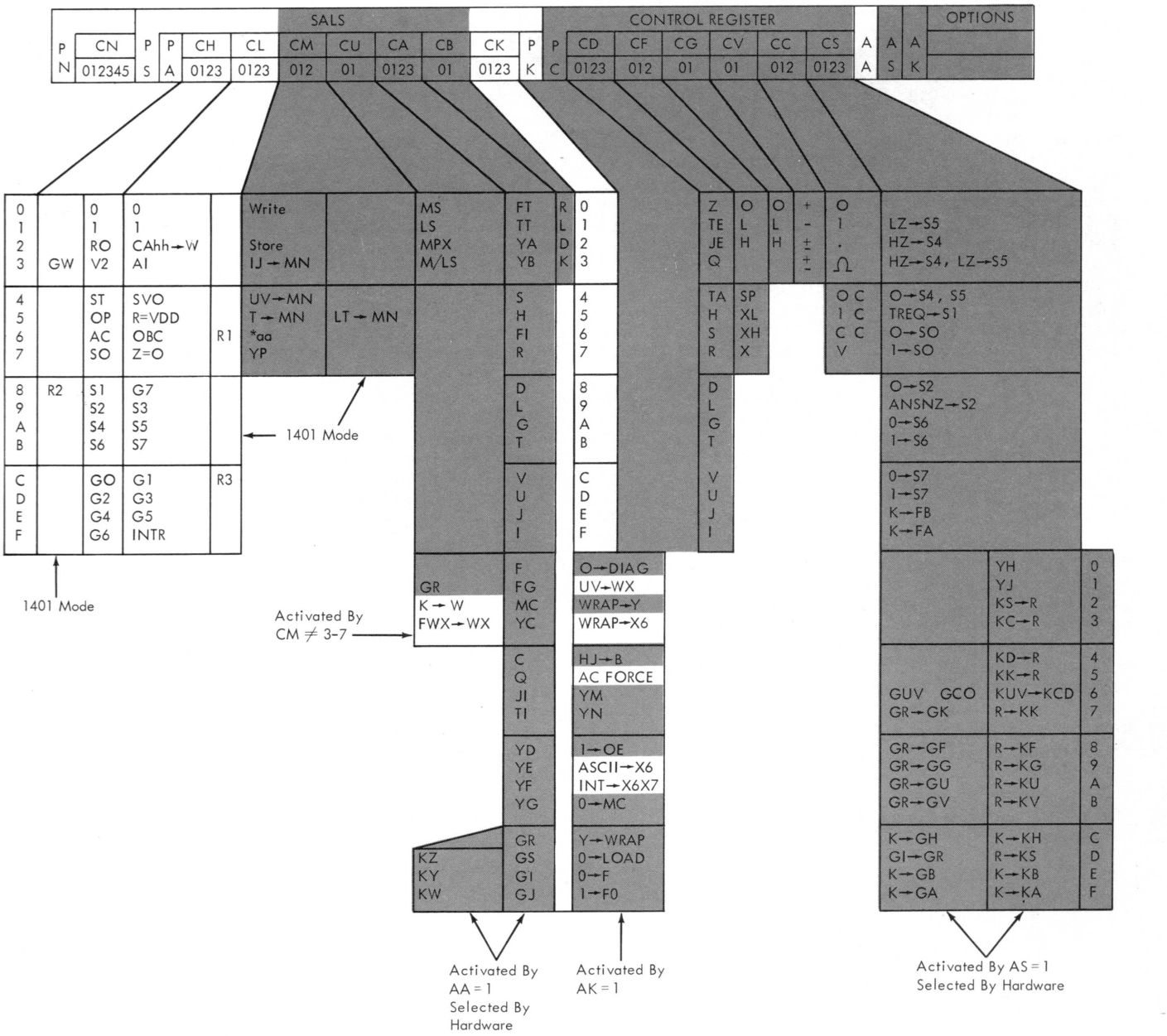


Figure 2-42. ROS Branch Control Fields

The 0 through 5 positions of the X register are set from the CN field while the 6 and 7 positions are set by decoding the CH and CL fields. If the condition of the CH field is satisfied, the 6th position of the X- register is set to the on condition and if the condition is not satisfied, the position is set to 0. The same operations for the 7 position are used except the CL field is decoded to determine the on or off condition.

STATUS SET AND RESET. Certain bit positions in the S-register are controlled by the CS field (Figure 2-43). The FB and FA latches for the multiplexor channel are also controlled by the CS field. The alternate codes of the CS field are used for the selector channel.

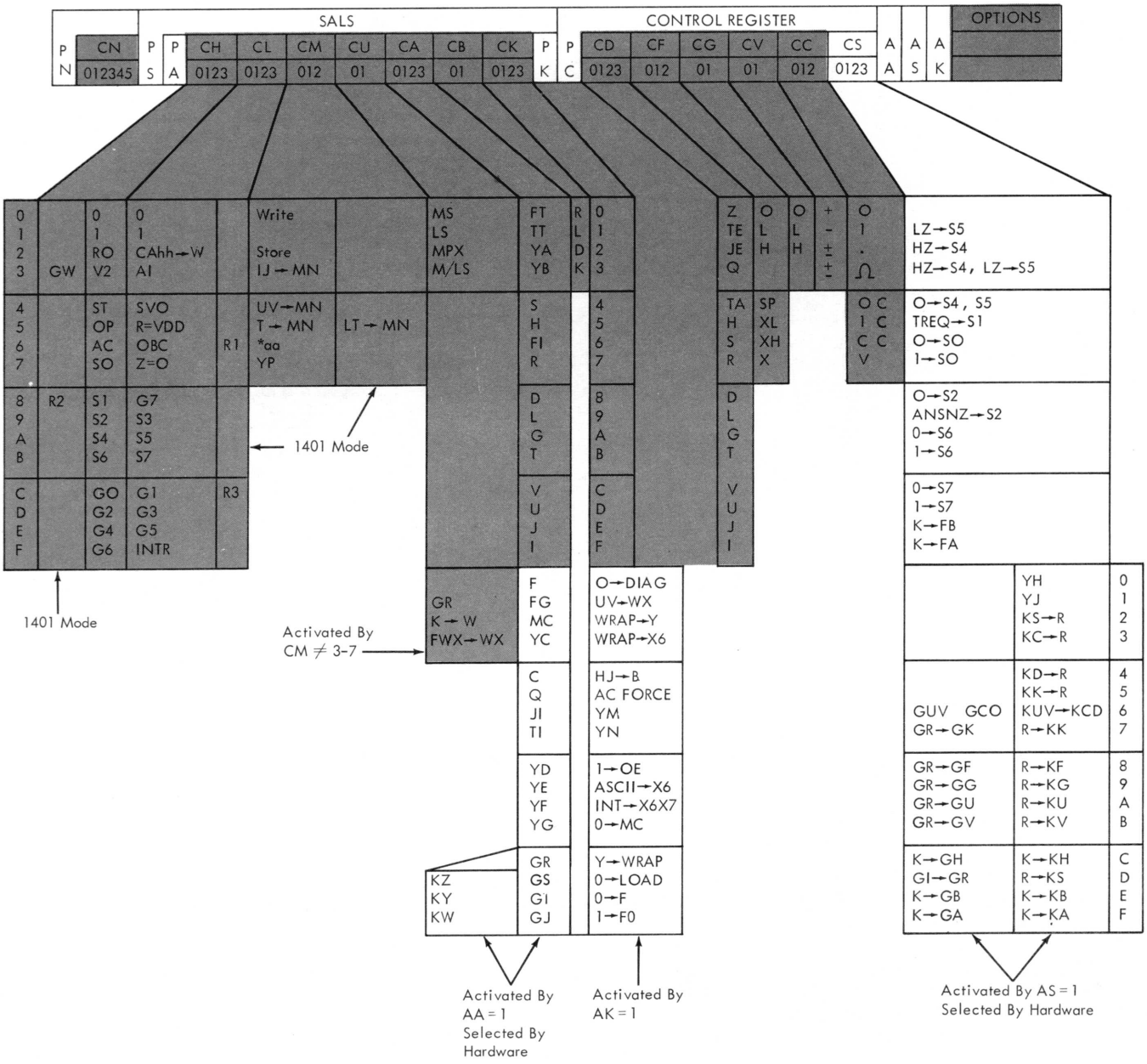


Figure 2-43. ROS Status Field and Parity

ALTERNATE DECODE. When the 1-bit AA field has a 1, the alternate codes for the CA field are used. If the 1-bit AS field has a 1, the alternate codes for the CS field are used. When the 1-bit AK field has a 1, the alternate CK codes are used (Figure 2-43).

When the 2030 is in 1401 compatibility mode the AA field needs a 1 in conjunction with the mnemonic CAhh->W, to set the ROS address.

CONTROL FIELD PARITY BITS. There are five parity bits associated with the control fields: PN, PS, PA, PK, and PC. (Figure 2-44) shows the fields and the parity bits used for each checking circuit.

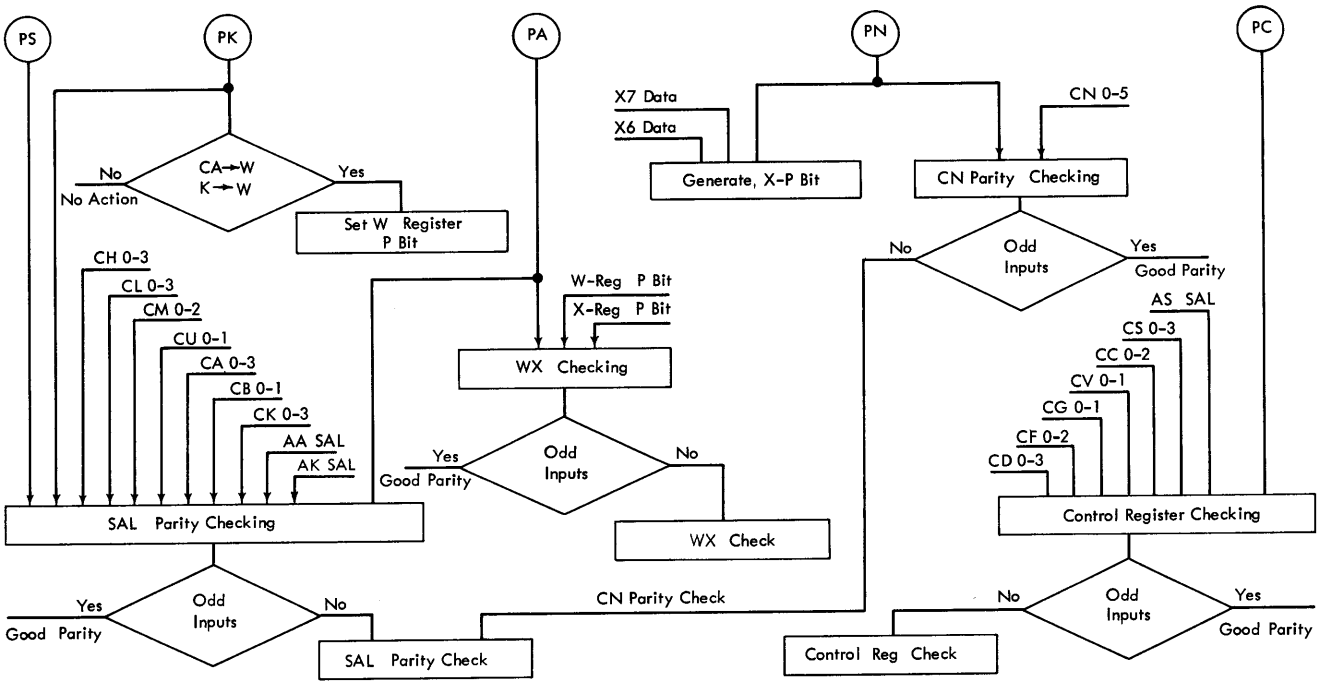


Figure 2-44. Parity Check Bits

The PN parity bit is used to maintain odd parity for the CN field. This bit is used with X6 and X7 bits to set X- register parity bit when CN is gated to the X- register. The PS parity bit is used to maintain odd parity for the AA, AK, CA, CB, CH, CK, CL, CM, and CU fields and the PA and PK bits.

The PA parity bit is used to maintain odd parity for the ROAR. As an example, if the address of the ROS word is 01BF (0000 0001 1011 1111), the PA bit must be a one to maintain odd parity.

The PK parity bit is used to maintain odd parity for either the CA or CK fields depending on the mnemonic used.

When the CK field is used as a constant in an arithmetic statement, the PK bit is not specified. In this case, the PK bit can be 0 or 1; usually 0. In the storage statement (*aa) or in a statement where K is used to change the W- register (K->W), the PK bit is used to provide odd parity on the W- register.

If the CA field is used to set the W- register (CAhh->W), the PK bit is used to maintain odd parity for the W- register.

The PC parity bit is used to maintain odd parity for the AS, CC, CD, CF, CG, CS, and CV fields.

Control Field Mnemonics

- Most of the control fields have from one to sixteen mnemonics.
- Some of the control fields have alternate mnemonics which are activated by the condition of another field.

By this time, we know the concepts of ROS, along with the names and functions of each ROS control field. Now we need to know how each control field is coded and how this coding is written in the microprogram so the microprogram can be read and punched in the ROS card.

Figure 2-45 shows the symbols used in the mnemonics and the meaning of the symbol. Figure 2-46 shows the mnemonics for each field and gives a brief description of the purpose each mnemonic serves.

Symbol New	Symbol Old	Definition	Example
+	+	True Add/Positive	A + B: B is Added (True) to A
-	-	Complement Add/Subtract, Negative	A - B: B is Complement Added to A
=		Equal	A = B: A is Equal to B
≠	≠	Unequal	A ≠ B: A is Unequal to B
→	=	Is Set Into	A→B: A is Set Into B (Destructive Read-In is Implied.)
·	*	Is ANDed With (Logical)	A . B: A is ANDed with B
,	,	AND (Non-logical)	A→B, C: A is Set Into B and C
∩	$	Is ORed With (logical)	A∩B→C: A is ORed with B and the Result is Set Into C
/	/	OR (Non-logical)	A / B→C: A or B is Set into C
⩛	⩛	Is Exclusive ORed With	A⩛B→C: A is Exclusive ORed with B and Set Into C
±		True or Complement Add/Positive or Negative	A ± B→C: B is True or Complement Added to A and the Result is Set into C
±	±	Binary Add Under T/C Control	A ± B→C: B is True or Complement Added to A and the Result is Set into C.
±	@	Decimal Add Under T/C Control	A ± B→C: B is True or Complement Decimal Added to A and the Result is Set into C.
<	<	Is Less Than	A < B: A is Less than B
¬	¬	Not (Boolean - Used as the Not Function on CLD's)	
:	:	Is Compared to	A : B: A is Compared to B
()	()	Used for Normal English Punctuation or to Enclose an Expression Within a Statement	
*		Special. 2030 Uses the * for One Mnemonic.	*a a: Explained in Mnemonic Section.
?	?	Indeterminate Function (This Describes a function which is Hardware Controlled Rather than Under the Direct Control of the Micro Program.	A ? B→C: A and B are Logically Combined (Under Hardware Control) and the Result is Set Into C.

Figure 2-45. CLD Block Symbols

Field	Hex	Mnemonic	Old Form	Operation. Location of Field is in Reference to Automated CLD Box
0 - 5 CN	-	-	-	Shown in Hex on Right Side of Line 7 in the CLD Box. Sets Position 0 through 5 of the X-Register for Next Address.
0 - 3 CH Set 6th Position of X-Register	- 0 1 2 3 4 5 6 7 8 9 A B C D E F	- 0 1 R0 • VZ ST OP AC S0 • S1 S2 S4 S6 G0 G2 G4 G6	- 0 1 R0 V=00 STI OPI AC S0 S1 S2 S4 S6 G0 G2 G4 G6	Shown on Left Side of Line 7 in the CLD Box. Set X-6 to ZERO Set X-6 to ONE Set X-6 to the Condition of R-Register Position 0 Set X-6 to ONE, if the V-Register Positions 6 and 7 are ZERO Status in (I/O) OP in (I/O) Set X-6 to ONE: if there is a Carry Out of ALU Position 0 } Set X-6 to ONE, if the Tested Position of the S-or G-Register is Equel to ONE
0 - 3 CL Set 7th Position of X-Register	- 0 1 2 3 4 5 6 7 8 9 A B C D E F	- 0 1 • CAhh →W AI SVI • R=VDD • 1BC Z=0 G7 S3 S5 S7 • G1 G3 G5 INTR	- 0 1 W=CA AI SVI RVDD 1BC Z=0 G7 S3 S5 S7 G1 G3 G5 INTR	Shown on Left Side of Line 7 in the CLD Box-Example; CH, CL Set X-7 to ZERO Set X-7 to ONE Set Value of CA Field into W-Register, Set X-7 to ONE. hh is the Hex Value of the CA Field and AA Field Address in (I/O Address) Service in (I/O) Set X-7 to ONE if the R-Register Contains Valid Decimal Digits. Set X-7 to ONE if there is a Carry Out of ALU Position One. Set X-7 to ONE if the Z-Bus (Bits 0-7) is ZERO } Set X-7 to ONE, if the Tested Position of the S-or G-Register is Equal to ONE Test for any Interrupt, Set X-7 to ONE if ther is a Interrupt.
0 - 2 CM Storage Control	- 0 1 2 3 4 5 6 7	- WRITE STORE IJ →MN UV →MN • T →MN • *aa YP	- WRITE STORE IJ UV T K GUV	Shown on Left Side of Line 4 in the CLD Box Write the Data in the R-Register into the Storage Position Addressed by the M-and N-Registers No Mnemonic-Compute Cycle, Storage not Used. Write NEW R-Register Data into the Storage Position Addressed by the M-and N-Registers Set the M-and N-Registers to the Address in the I-and d-Registers and Read from Storage at that Address Set the M-and N-Registers to the Address in the U-and V-Registers and Read from Storage at that Address Set the N-Registers to the Address in the T-Register and Read from Storage at that Address Set the N-Register using the CK Field (Note 1) Dummy Symbol-No Action or Can be Used in a Diagnostic Area. (Old From was a Selector Channel Code).
0 - 1 CU Storage Selection	- 0 1 2 3	- MS LS MPX M/LS	- MEM CPU UCW M,C	Shown on Right Side of Line 4 in the CLD Box Addressing MAIN Storage Addressing Auxiliary Storage-LOCAL Store Section Addressing Auxiliary Storage-Multiplexor UCW Section Addressing MAIN Storage or LOCAL Store Section, Depending on the OP Code-RR Format Selects LS In 1400 Mode this Selects the Local Storage Area for NPL Area
0 -1 Alternate CU	- 0 1 2 3	- GR K →W FWX →WX	- Use GR W=K WX=FWX	Shown on Right Side of Line 4 in the CLD Box No Action Use the GR-Register in the Selector Channel in Place of the R-Register for Storage Input and Output Set the W-Register to the Hex Valve of the CK Field Set the W-and X-Registers to the Address in the Multiplexor Back-Up Registers (FW and FX)
0 - 3 CA A-Register Source Control	- 0 1 2 3 4 5 6 7 8 9 A B C D E F	- FT TT YA YB S H FI R D L G T V U J I	- FT TT S H FI R D L G T V U J I	Shown on Left Side of Line 3 in the CLD Box Multiplexor Channel Tags in 1050 Tags in Dummy Symbol-No Action or Can be Used in a Diagnostic Area Dummy Symbol-No Action or Can be Used in a Diagnostic Area Gate the S-Register to the A-Register Via the A-Bus Gate the H-Register to the A-Register Via the A-Bus Multiplexor Channel Bus In } Gate the - Register to the A-Register Via the A-Bus

Figure 2-46. Mnemonics, Sheet 1

Field	Hex	Mnemonic		Old Form	Operation
0-3	-	-	-	-	Shown on Line 4 of the CLD Box
Alternate	0		F	F	Gate the F-Register to the A-Register Via A-Bus (External Interrupts).
CA	1		FG	FG	Gate the F-and G-Switches to the A- Register Via the A-Bus
Activated	2		MC	MC	Gate the Machine Check Register to the A-Register Via the A-Bus
by "AA"=1	3		YC		Dummy Symbol-No Action or Can be Used in the Diagnostic Area
	4		C	C	Gate the C-Register to the A-Register Via the A-Bus (Interval Timer)
	5		Q	Q	Gate the Q-Register to the A-Register Via the A-Bus (Protect Storage)
	6		JI	JI	Direct Data Channel Bus In
	7		TI	TI	1050 Bus In
	8		YD		} Dummy Symbols - No Action or Can be Used in the Diagnostic Area
	9		YE		
	A		YF		
	B		YG		
	C		GR	GR	Gate the GR-Register (Selector Channel) to A-Register Via A-Bus
	D	KZ	GS	GS	Gate the GS-Register (Selector Channel) to A-Register Via A-Bus
	E	KY	GT	GT	Gate the GT-Register (Selector Channel) to A-Register Via A-Bus
	F	KW	GJ	GJ	Gate the GJ-Register (Selector Channel) to A-Register Via A-Bus
0-1	-	-		-	Shown on Line 3 of the CLD Box
CB	0	R		R	Gate the R-Register to the B-Register Via the B-Bus
B-Register	1	L		L	Gate the L-Register to the B-Register Via the B-Bus
Source	2	D		D	Gate the D-Register to the B-Register Via the B-Bus
Control	3	K		K	Gate Hex Value of the CK Field to the B-Register Via the B-Bus
	-	-		-	
0-3	-	- - - -		- - - -	Shown on Line 2 of the CLD Box
CK	0	0 0 0 0		0 0 0 0	
Emit Value	1	0 0 0 1		0 0 0 1	
	2	0 0 1 0		0 0 1 0	
	3	0 0 1 1		0 0 1 1	
	4	0 1 0 0		0 1 0 0	
	5	0 1 0 1		0 1 0 1	
	6	0 1 1 0		0 1 1 0	
	7	0 1 1 1		0 1 1 1	
	8	1 0 0 0		1 0 0 0	} Binary Bit Form of the Hex Number is Routed to the Selected Area When Requested.
	9	1 0 0 1		1 0 0 1	
	A	1 0 1 0		1 0 1 0	
	B	1 0 1 1		1 0 1 1	
	C	1 1 0 0		1 1 0 0	
	D	1 1 0 1		1 1 0 1	
	E	1 1 1 0		1 1 1 0	
	F	1 1 1 1		1 1 1 1	
0-3	-	-		-	Shown on Left Side of Line 6 of the CLD Box
Alternate	0	O→DIAG		RESET DIAG	Reset the Diagnostic Latch
CK	1	UV→WX		WX = UV	Gate the U-and V-Register to the W-and X-Registers Via the WX-Bus
Activated	2	● WRAP→Y		RESTORE WRAP	Gate the Wrap Buffer Latch to the Wrap Latch
by "AK"=1	3	● WRAP→X6		TEST WRAP	Set X6 to ZERO if Wrap Latch is On
	4	HJ→B		HJ	Gate the H-and J-Switches to the B-Register Via the B-Bus
	5	● AC FORCE		AC FORCE	Set X-Register to ZERO, if a ALU Carry Occurred in Previous Cycle
	6	YM			Dummy Symbol-No Action or Used in Diagnostic Area. Old Mnemonic was Reset 1050 Line Latch
	7	YN			Dummy Symbol-No Action or Used in Diagnostic Area. Old Mnemonic was Set 1050 Line Latch
	8	● 1→OE		OE = 1	Force an ALU Check (Note 3)
	9	● ASCII→X6		TEST ASCII	Set X-6 to ZERO if the ASCII Latch is On.
	A	● INT→X6, X7		TEST INT	Set X-6 and X-7 per Stacked Interrupts (Note 4)
	B	0→MC		MC = 0	Set Machine Check Register to All ZEROS
	C	Y→WRAP		STORE WRAP	Gate the Wrap Latch to the Wrap Buffer Latch
	D	0→LOAD		LOAD	Reset the LOAD, ODD/EVEN, and INTRODUCE ALU CHECK Latches
	E	● 0→F		F = 0	Reset the F-Register to All ONES. Note: The Reset Condition of the F-Register is All ONES
	F	● 1→F0		F0 = 1	Set the F-Register Position 0 to ZERO
0-3	-	-		-	Shown on Line 3 of the CLD Box
CD	0	Z		Z	To Show That the Z-Bus is the Only Place the Output of the ALU is Routed
Destination	1	TE		TE	1050 Bus Out (Exit)
of ALU	2	JE		JE	Direct Data Channel Bus Out (Exit). Set JE-Register from D-Register, Z-Bus not Used.
Output	3	Q		Q	Gate the Output of the ALU to the Q-Register Via the Z-Bus
	4	TA		TA	1050 Tags Out■
	5	H		H	} Gate the Output of the ALU to the ____Register Via the Z-Bus.
	6	S		S	
	7	R		R	
	8	D		D	
	9	L		L	
	A	G		G	
	B	T		T	
	C	V		V	
	D	U		U	
	E	J		J	
	F	I		I	

Figure 2-46. Mnemonics, Sheet 2

Field	Hex	Mnemonic	Old Form	Operation
0-2	-	-	-	Shown on Line 3 of the CLD Box
CF	0	0	0	Block A-Register Exit to the ALU. Route All ZEROS to ALU Entry for the A-Register.
A-Register to ALU	1	L	L	Block High 4 Bits of A-Register. Route 4-ZEROS and Bits 4-7 of A-Register to the ALU
	2	H	H	Block Low 4 Bits of A-Register. Route 4-ZEROS and Bits 0-3 of A-Register to the ALU
	3			Gate the Entire A-Register to the ALU
	4	SP	STOP	Conditional Machine Stop (Note 5)
	5	XL	XL	Block A-Register Bits 4-7 Exit. Gate Register Bits 0-3 to ALU Entry Bits 4-7 and 4-ZEROS to Bits 0-3.
	6	XH	XH	Block A-Register Bits 0-3 Exit. Gate A-Register Bits 4-7 to ALU Entry Bits 0-3 and 4-ZEROS to Bits 4-7.
	7	X	X	Gate A-Register Bits 0-3 to ALU Entry Bits 4-7 and Gate A-Register Bits 4|7 to ALU Entry Bits 0-3.
0-1	-	-	-	Shown on Line 3 of the CLD Box
CG	0	0	0	Block B-Register Exit to the ALU. Route All ZEROS to the ALU Entry for the B-Register .
B-Register to ALU	1	L	L	Block High 4-Bits of the B-Register. Route 4-ZEROS and Bits 4-7 of the B-Register to the ALU.
	2	H	H	Block Low 4 Bits of the B-Registers. Route 4-ZEROS and Bits 0-3 of the B-Register to the ALU.
	3			Gate the Entire B-Register to the ALU.
0-1	-	-	-	Shown on Line 3 of the CLD Box
CV	0	+	+	True Add B-Register Data
Arithmetic	1	-	-	Complement Add B-Register Data
Functions	2	±	±	Binary Add or Subtract Depending on the Status of S0
	3	±	@	Decimal Add or Subtract Depending on the Status of S0
0-2	-	-	-	Shown on Line 3 of the CLD Box
CC	0	0	0	Block Carry
Arithmetic	1	1	1	Insert Carry
Controls	2	·	*	AND Function-Check to See if Same Bits are Set to ONE in both the A-and B-Register Using the ALU
	3	⌐┘	$	OR Function-Check to See if Either Bit in the Same Position of the A-and B-Register is Set to ONE.
	4	0C	CO	No Carryin, Set S3 to ONE if A Carryout Occurs
	5	1C	C1	Insert A Carryin and Set S3 to ONE if a Carryout Occurs.
	6	CC	CC	Allow Carryin from Carry Latch and Set S3 to ONE if a Carryout Occurs.
	7	⊻	⊻	Exclusive OR Function-Check to See if Only the A-or B-Register has the Same Bit Position Set to ONE.
0-3	-	-	-	Shown on Line 5 of the CLD Box
CS	0			No Action
Status	1	LZ→S5	S5=LZ	Set S5 to ONE if Bits 4-7 of the Z-Bus Are 0. Reset S5 if Non-Zero (Set S6 to ZERO).
Conditions	2	HZ→S4	S4=LZ	Set S4 to ONE if Bits 0-3 of the Z-Bus are 0. Reset S4 if Non-Zero (Set S5 to ZERO).
	3	HZ→S4, LZ→S5	S4, S5=HZ, LZ	Combines the Conditions of CE Field Mnemonics LZ→S5 and HZ→S4.
	4	0→S4, S5	S4, S5=0	Set S4 and S5 to ZERO.
	5	TREQ→S1	S1=TREQ	Set S1 to ONE if a 1050 Request has Occurred. Set S1 to ZERO if no 1050 Request.
	6	0→S0	S0=0	Set S0 to ZERO.
	7	1→S0	S0=1	Set S0 to ONE.
	8	0→S2	S2=0	Set S2 to ZERO.
	9	ANSNZ→S2	S2=ANSNZ	Set S2 to ONE if the Output from the ALU is Non-Zero. (Note 6).
	A	0→S6	S6=0	Set S6 to ZERO.
	B	1→S6	S6=1	Set S6 to ONE.
	C	0→S7	S7=0	Set S7 to ZERO.
	D	1→S7	S7=1	Set S7 to ONE.
	E	K→FB	FB=K	Multiplexor Channel Tags Out.
	F	K→FA	FA=K	Multiplexor Channel Tags Out.
0-3	-	- -	-	Shown on Line 5 of the CLD Box.
Alternate	0	YH -	-	} Dummy Symbols
CS	1	YJ -	-	
Selector	2	KS→R -	-	} Used for Selector Channel Operations These are Selector Channel Registers (Note 2)
Channel	3	KC→R -	-	
Activated	4	KD→R -	-	
by "AS"=1	5	KK→R -	-	
Selected	6	KUV→KCD GUV→GCD	GCD=GUV	
by	7	R→KK GR→GK	GK=GR	
Hardware	8	R→KF GR→GF	GF=GR	
	9	R→KG GR→GG	GG=GR	
	A	R→KU GR→GU	GU=CR	
	B	R→KV GR→GV	GV=GR	
	C	K→KH K→GH	GH=K	
	D	R→KS GI→GR	GR=GI	
	E	K→KB K→GB	GB=K	Selector Channel Tags Out.
	F	K→KA K→GA	GA=K	Selector Channel Tags Out.

Note 1
N-Register Set as Follows:
N0-Forced to ONE
N1-Forced to ZERO
N2-CN 0Bit
N3-CK 0Bit
N4-Forced to ONE
N5-CK 1Bit
N6-CK 2Bit
N7-CK 3Bit

Note 2
These Mnemonic Depends on the Channel Requested. Mnemonics may be of Three Types for one Hex Number. Example- Alternate CA Field. Hex E can be KY, GT, or HT

Note 3
Used on Diagnostics to Force Parity

Note 4

	X6	X7
Timer/External	$\overline{0}$	$\overline{0}$
Channel 1	1	0
Channel 2	0	1
Multiplexor Channel	1	1

Note 5
Micro Program Stop or Process Loop Stop

Note 6
In Diagnostic Mode, if the "Malfunction Trap Latch" is Set, this Mnemonic will Cause Machine Stop.

Figure 2-46. Mnemonics, Sheet 3

Some of the mnemonics need a more complete explanation than given in the chart. There is a • next to the new mnemonic having a more detailed description. The column to the left of the mnemonic contains the hex number punched in the bit positions for that control field. As an example, the CH field would be punched 1010(A) for a mnemonic S4.

CAhh->W: The value in the CA field is gated to the W-register positions 4-7 and the AA field is gated to position 3. Example: to change from ROS address 01XX to 08XX, the mnemonic CA08->W is used. Also, position 7 of the X-register is set to 1 for the next address. Parity for the W-register is maintained by using the PK bit.

Note: If this mnemonic is used in the 1401 compatibility mode, the AA 1-bit field is set to 1. This is routed to the 3rd position of the W-register to select the second ROS module or if set to 0, selects the first ROS module. Also when this mnemonic is used, the add statement normally has an A entry of 0 (CF = 000).

R = VDD: Each half of the R-register is checked for a valid decimal digit (0-9). Set X7 to a 1 if both digits are valid decimal digits.

K->FA: The CK field is used to set and reset latches in the multiplexor FA-register, singly or in combination. The value of the CK field is shown on the E line of the new CLD box which is explained later. The PK bit is necessary and it's condition is also specified on the E line. If the CK field value is 3 and PK is 1, the E line will have K = 0011,1.

K->FB: The CK field and the PK bit is used to set and reset some of the multiplexor FB-register latches. This mnemonic also provides a gate for the set and reset of other latches.

Y->WRAP: An address overflow (memory wrap) may arise on a 64K core storage unit. Additional circuitry is needed to detect the error which occurs when there is a carryout of the high-order position of the I- or U-register as a result of up-dating the address.

If the I- and J-registers are used to set the M- and N-registers to address core storage, a memory wrap condition sets a wrap latch. Certain routines, which may be needed during the decode of an SS instruction, require the condition of this latch to be retained. The mnemonic Y->WRAP gates the status of the wrap latch to another latch called the wrap-buffer-latch. It is retained there until the WRAP->Y mnemonic is used.

WRAP->Y: When it becomes necessary to determine if there had been a wrap earlier, the mnemonic WRAP->Y gates the status of the wrap-buffer latch to the wrap latch. This mnemonic is also used to reset the wrap latch in some routines.

WRAP->X6: To test the status of the wrap latch for branching, the mnemonic WRAP->X6 is used. If the wrap latch is on, a 00 or 01 branch is taken. However, it if is off, the X6 portion of the branch is still controlled by the CH field. Note: A wrap condition can also occur on an 8K, 16K or 32K machine. This is detected by testing one of the three high-order positions of the M-register to see if it is set to 1. The position tested depends on the size of core-storage the machine has. This does not use the wrap circuits used on 65K machines.

AC Force: This alternate CK mnemonic causes all positions of the X-register to be set to zeros if there was a carryout of the ALU during the previous ROS word. The information in the CN, CH, and CL fields is blocked. With the X-register equal to zero, the microprogram is branched to 00 of the block addressed by the W-register decode.

1->OE: This mnemonic is used in diagnostic testing. The first time this mnemonic is used in a routine, bad parity is forced by blocking the + L Z bus 0 line and + L Z bus 4 line. The next time this mnemonic is used in the routine, an ALU check is forced by forcing all the minus SUM lines and the minus carry 0-bit line to a plus L levels (Figure 2-47).

The odd-even-control latch is turned on (EVEN) by the decoded

line 1->OE, T2 and the introduce ALU check latch set off. A circuit to gate the + L Z bus 0 line and the + L Z bus 4 line requires that the odd-even latch be ODD. The introduce-ALU-check latch is turned on at T1 when the expression 1->OE is used again. This latch blocks the -L SUM lines and the -L carry 0-bit line so the lines are all plus and an ALU check is forced. The odd-even-control control latch is turned off three ways: machine reset, reset load line (which is developed when the mnemonic 0->IPL is used), or at T2 time when the introduce-ALU-check latch is on.

ASCII->X6: This mnemonic tests the ASCII latch to see if the ASCII latch is on. If the latch is on, the 6th position of the X-register is set to 0. If the latch is off, the 6th position of the X-register is set by the CH field conditions.

H: When the H-register is specified by the CD field coding of 5 (0101), the priority-reset-control latch is set on. This latch ANDed with T3 time, turns the priority latch off so priorities may be recognized.

SP: This CF field mnemonic causes the stop latch to be turned on at T4 time (Figure 2-48). The output from the stop latch feeds two circuits. If the J-register is not specified by the CA field, one circuit causes a microprogram stop line to be active. This line stops the CPU clock by blocking the clock start circuit. If the J-register is specified and the process stop latch is on, a process-loop-stop line is made active when the stop latch comes on. The process-loop-stop line allows the CPU clock to run until all ROS share requests or multiplexor share requests have been honored. The CPU clock is stopped by turning off the clock-start latch. The process-loop stop line blocks the set of the W- and X-registers so the microprogram returns to the address of the STOP word after execution of any ROS or multiplexor share request.

*aa: This mnemonic addresses a byte in local storage. The expression on the S line of the CLD box to read addressable byte 27 from local storage is *BB LS. Figure 2-49 shows how the N-register is set to BB so byte 27 can be addressed. The CK field must be a B (1011). Since this is a constant, the CK field is specified on line E of the CLD box.

VZ: When operating in 1401 mode, this mnemonic appears as GW and is used to allow a branch on a group mark word mark combination.

S1: When operating in 1401 mode, this mnemonic appears as R2 and is used to allow a branch on the condition of the bit 2 in the R-register.

1BC: When operating in 1401 mode, this mnemonic appears as R1 and is used to allow a branch on the condition of the bit 1 in the R-register.

G1: When operating in 1401 mode, this mnemonic appears as R3 and is used to allow a branch on the condition of the bit 3 in the R-register.

T->MN: When operating in 1401 mode, this mnemonic appears as LT->MN and is used to gate the L- and T-registers to the M- and N-registers. This performs the functions of the A-star in the 1401.

0->F: The F-register is used to recognize external interrupts. This mnemonic resets the F-register so the output lines of the F-register are plus, preventing any external interrupts from being recognized until the F-register is set from an external device. Since the F-register is reset so all output lines are plus, we say it is reset to ONE's.

1->F0: Since the F-register is reset to ONE's, this mnemonic set the 0-position of the F-register to ZERO.

Special Statements

The following special statements are used in diagnostic programming:

0V±0->Z: This expression brings up the control lines to check positions

4 and 5 of the R-register. If bit 4 is a one, the ASCII latch is set. If bit 5 is a zero, the suppress malfunction trap latch is set. Decimal mode is specified on line 2 of the CLD block.

J ±0->Z: The wait latch is set on and the ROS word that contains this statement is continually executed until an interrupt occurs. Decimal mode is specified on line 2 of the CLD block.

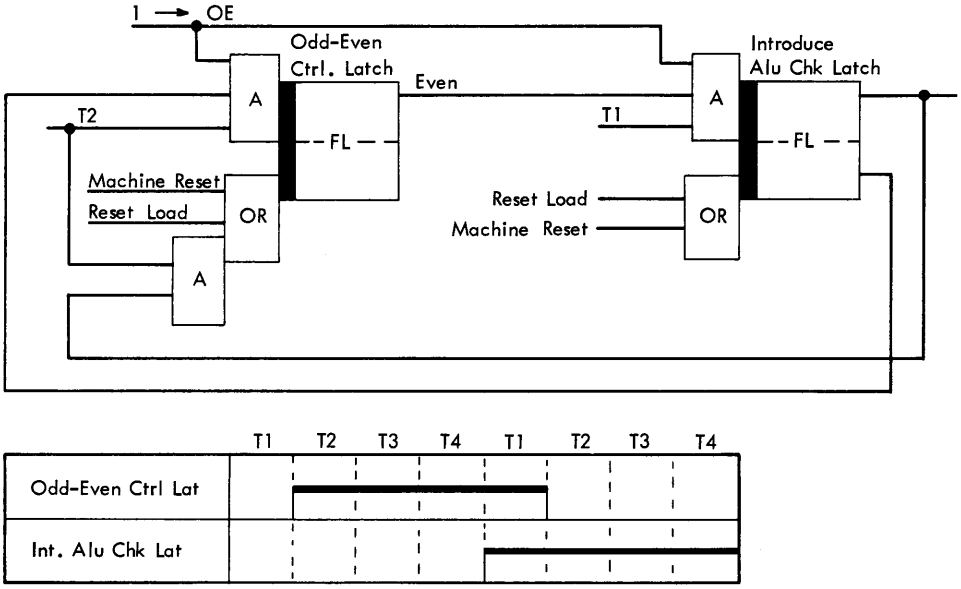


Figure 2-47. 1 -> OE Control

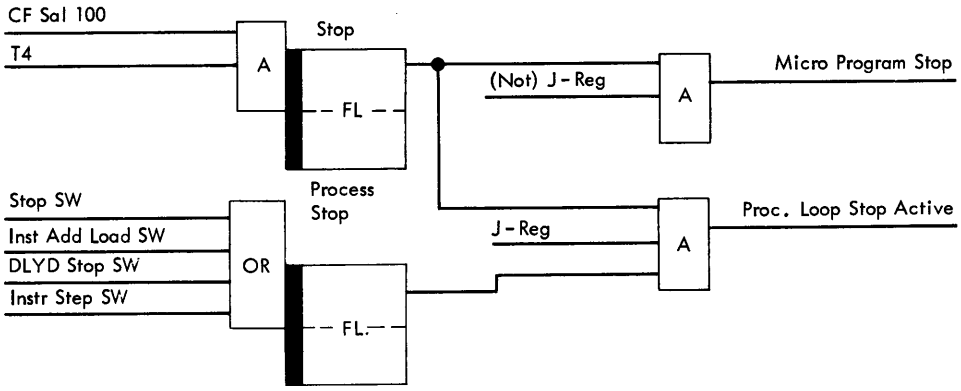


Figure 2-48. Stop Mnemonic

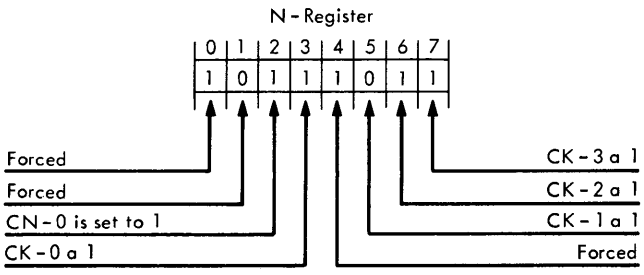


Figure 2-49. N-Register Set from *aa

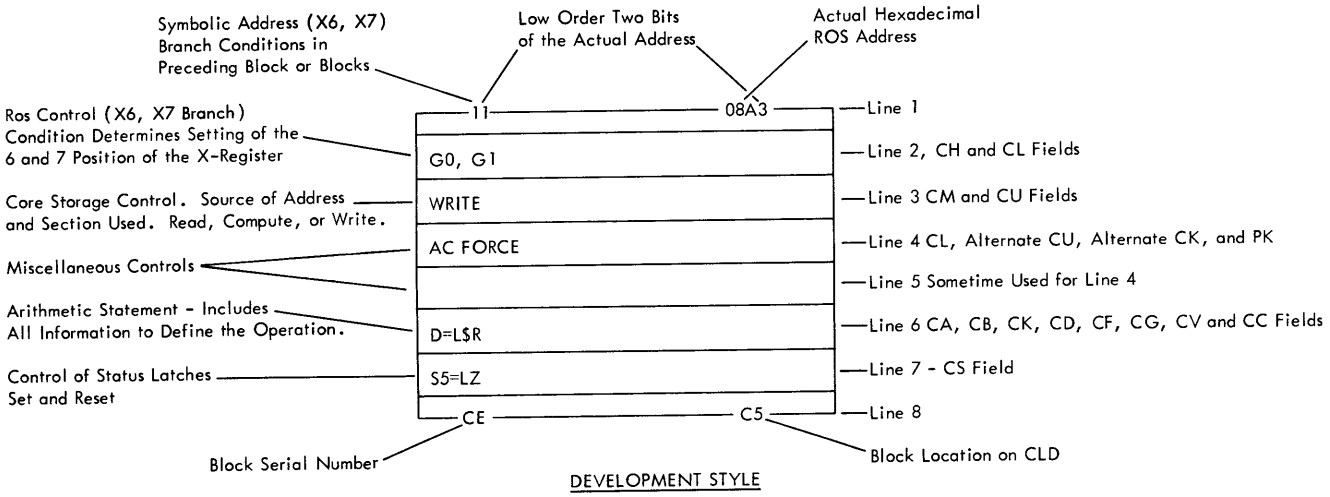
CLD BLOCKS

- Each ROS word is written in one CLD (CAS Logic Diagram) block.
- There are eight lines in a CLD block used with the 2030 micoprogram.

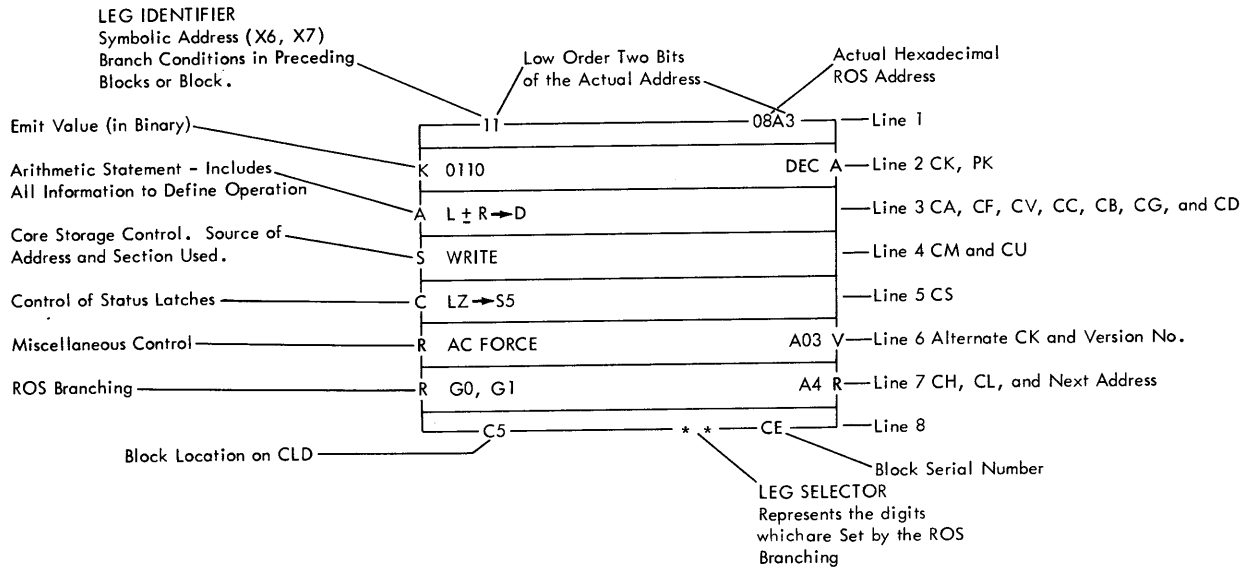
By this time, you know the different ROS fields and the mnemonics used with each field. The next step is to learn how the mnemonics are tied together into a logical statement and the format for this statement.

Figure 2-50 shows the CLD block format for both the development and unified style. We will discuss the unified style; the development style is shown in case you ever need to cross reference the two styles.

First, let's look at the eight lines as shown in Figure 2-50. Notice that the letters in the edge of the block are present only when there is information on the line.



DEVELOPMENT STYLE



UNIFIED STYLE

Figure 2-50. CLD Block Format

Line 1 contains the leg identifier, that indicates which box to branch to from the previous ROS statement branching conditions, and the actual hexadecimal address of this word.

Line 2 contains K on the left edge. The value of the CK field and the PK bit are found here, and on the right edge is an A when the type of arithmetic operation is defined here.

Line 3 has an A on the left side when the arithmetic statement for the ROS word is found here.

Line 4 has an S on the left side. The core-storage control statement is written here, and on the right side an S or an R can be found. The S is found here when there is a statement on the right side that has to do with storage control (MS), and the R is found here when a statement concerned with ROS branching is present (K->W).

Line 5 has a C on the left edge, the mnemonics from the CS field are found here. Note: The C stands for control, miscellaneous.

Line 6 can have a C (0->F), or an R (AC FORCE) on the left side; the alternate CK field is written here. On the right side of line 6 is a V; the version number of this ROS word is located here.

Line 7 has an R on the left edge; the bits or conditions to be checked to determine the setting of the X-register 6th and 7th positions are written here. Also the right edge has an R; the actual lowest hex address that can be branched to is written here.

Line 8 has the location of this block on the CLD page on the left side. In the center is the leg selector which represents the bits for ROS branching. On the right is the serial number of the block; this number normally changes if the block is moved on the CLD page.

Now let's break each line down and learn how to read a ROS statement using Figure 2-51. The left edge of line 1 has two bits (in this case 10). This means that the conditions of ROS branching in the last word caused X-6 to be set to ONE and X-7 to be set to ZERO. Also, when reading the CAS logic and it is possible to leave one block and go to any one of four blocks, this identifies the block that ROS branching selected. On the right side, the hexadecimal address of this word is written. The breakdown of the address is shown in the figure.

The left side of line 2 has the value of the CK field written in binary form. This is shown when the CK field is to be used as a constant or the alternate CK mnemonics are used. To the right of this is the PK bit condition, ZERO or ONE. Next to the PK bit, a character can be inserted to force bad parity (see Figure 2-52).

On the right side of line 2, either DEC (decimal) or BIN (binary) is written, if the Decimal feature is installed to identify if the add is decimal or binary.

In Figure 2-51, line 3 shows where each field of the add statement is found. CA, where the A-register data is gated from; CF, how the A-register is gated to ALU; CV or CC, depends on what is wanted--an add or compare of the conditions of the A- and B-registers; CB, where the B-register data is gated from; CG, how the B-register is gated to the ALU; CC, carry in condition; CD, what register the ALU output is written into; CC, carry out condition.

The CM field is found on the left side of line 4, and the CU field on the right side.

The CS field is found on line 5, control of the status set and reset.

The Alternate CK field is found on the left side of line 6, and can have things to do with status control (C), add statement (A), or ROS branching (R). The version of this ROS word is identified on the right side of line 6; a basic word is left blank.

The CH and CL fields are on the left side of line 7. These two fields are used to test the status of certain conditions and set X-6 and X-7 from the test results. The next lowest ROS address that the CN, CH, and CL fields will allow the microprogram to branch to, is written in hex on the right side of line 7. Note: just the setting of the X-register is shown.

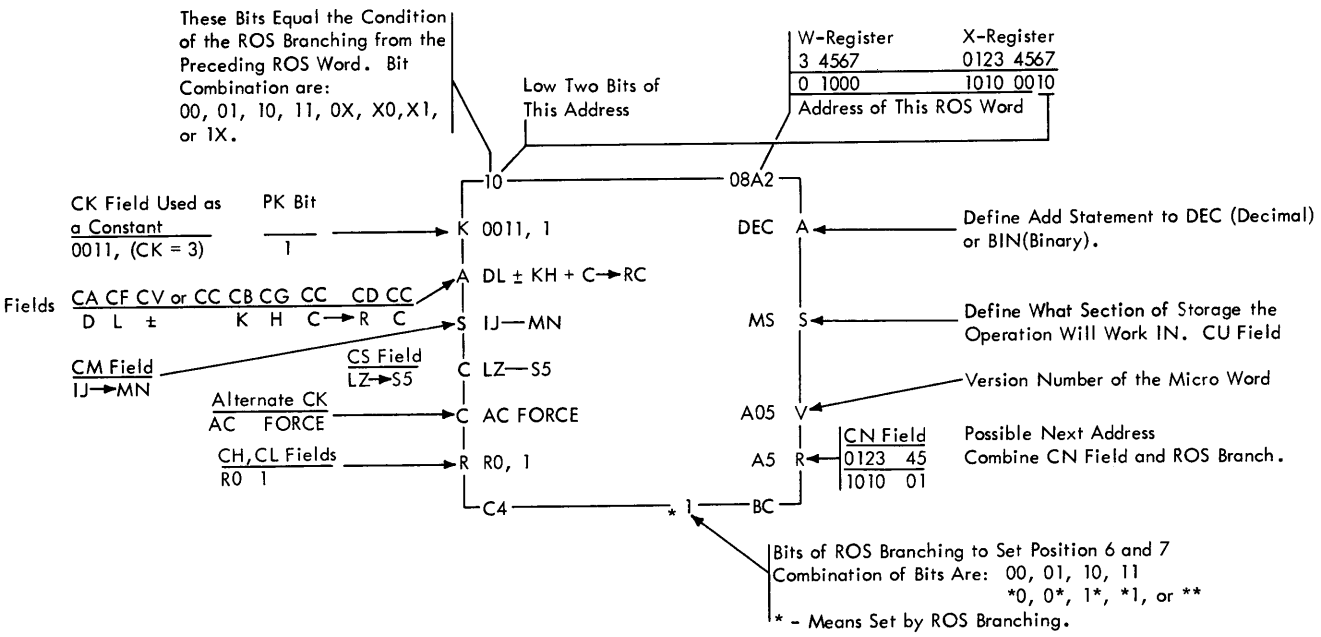
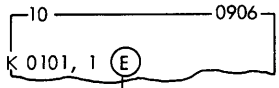


Figure 2-51. CLD Block Line Breakdown.

In the center of line 8, the leg selector is shown. This shows the bit condition that X6 and X7 are set to for the branch from this word. If the condition is determined by testing the status of some bit or condition an * is placed in that position of the leg selector.



Letter	Creates Bad Parity On PN	PS	PA	PC
F	1	1	1	1
E	1	1	1	0
C	1	1	0	0
8	1	0	0	0

Note: 1 Indicates Bad Parity.

Figure 2-52. Bad Parity Created

EXIT AND ENTRY CLD BLOCKS ARE FOUND.

Figure 2-53 shows a CLD exit and entry block. The lines for each block is as follows:

A: Entry block.

Line 1: The leg identifier and a simulated hexadecimal address (three X's).

Line 2: The word from.

Line 3: Contains the page number and the block serial number that the entry came from.

Line 4-17: Contains additional page numbers and block serial numbers that the entry can come from.

Last line: Contains the block location, leg selector, and block serial number for this block.

B: Exit Block.

Line 1: The leg identifier and a simulated hexadecimal address (three X's).

Line 2: The words go to.

Line 3: Contains the page number and the block serial number of the entry block that this exit goes to.

Line 4-7: Blank

Line 8: Contains the block location, leg selector, and block serial number for this block.

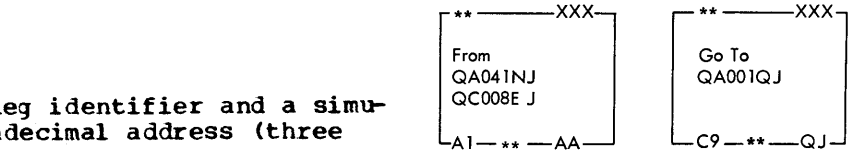


Figure 2-53. CLD Entry and Exit Blocks

MICROPROGRAM SAMPLE PROBLEM

Using Figure 2-54, determine the data in the R-register after leaving block 04B6 the second time. The conditions to start are:

1. The UV registers contain the address XXXX.
2. The D-register contains the value of 17 in binary 0001 0001.
3. The S-register is set to zero except for S2, which is set to a 1.
4. The R-register data is zero.

The data in the R-register after completing block 04B6 the second time is 0001 0111.

This is arrived at by executing the blocks in the following order.

ADDRESS 04AA: The data in the R-register

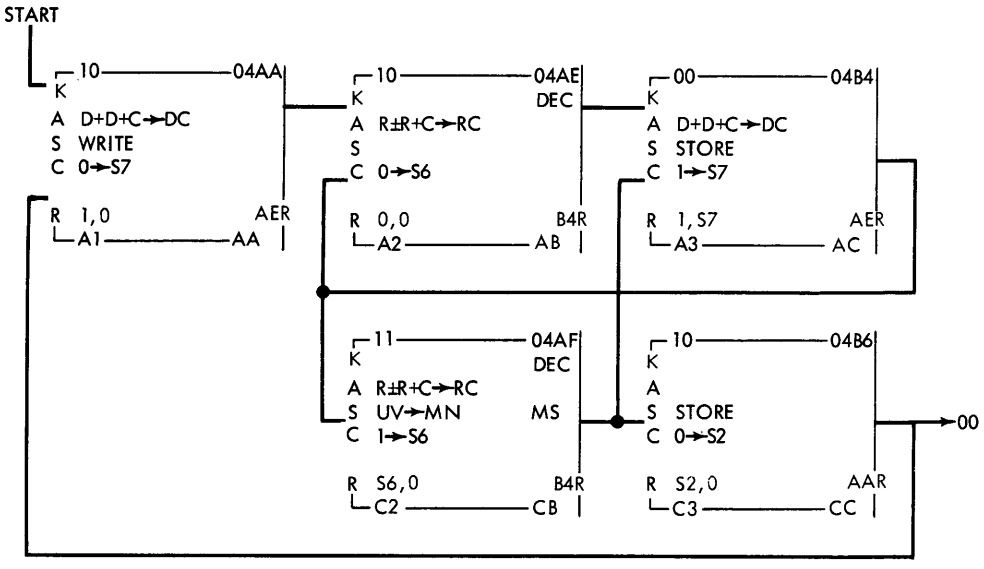


Figure 2-54. Microprogram Test

is returned to core (WRITE.) The data in the D- is presented to both the A-register and B-register inputs to ALU. There is no carry insert because position 3 of the S-register is zero. There is no carryout as a result of the addition to set S3 (D+D+C DC).

A source	0001	0001
B source	0001	0001
D-register	0010	0010

Postion 7 of the S-register is set to zero (0->S7). An unconditional 1, 0 branch is taken to address 4AE.

ADDRESS 04AE: The data in the R-register is DECIMAL added to itself (R±R+C->RC). Decimal mode is specified by the K line. S3 is still zero, therefore no carry is inserted. Because the data in the R-register is zero, the resultant addition provides no carry out to set S3. Position 6 of the S-register is set to zero (0->S6), Branch 0,0 to address 04B4.

ADDRESS 04B4: The mnemonic STORE, does nothing for us at this time because the previous cycle was not a read. The data in the D-register is again added to itself. Still, there are no carries.

A source	0010	0010
B source	0010	0010
D-register	0100	0100

You can see that every time the data in the D-register is added to itself, the data shifts one position to the left.

Position 7 of the S-register is set to a 1 (1->S7). On the branch line a test is made on S7. This test is done early in the cycle before any status is set by the C line statement. As a result, a 1, 0 branch is executed to 04AE.

ADDRESS 04AE: Again, the R-register data is decimal added to itself. And again, since the data in the R-register is zero and there is no carry insert, the resultant answer is zero with no carry out. S6 is set to zero (0->S6). Branch 0,0 to address 04B4.

ADDRESS 04B4: The S line statement, STORE, has no effect. The data in the D-register is added to itself (D+D+C->DC) with

A source	0100	0100
B source	0100	0100
D-register	1000	1000

no carryout, S3 is still zero. Position 7 of the S-register is set to a 1 (1->S7). Since S7 was set previously, the branch conditions now set up a 1,1 branch to address 04AF.

ADDRESS 04AF: The MN registers, set by UV, address main core (MS) to read data at address XXXX (UV->MN MS). The data in the R-register is again DECIMAL added to itself (R±R+C->RC). No carries are involved. Position 6 of the S-register is set to 1 (1->S6), but not before the branch test is made and a 0 0 branch is taken to address 04B4.

Note that when a bit in the branch statement (line 7) is set or reset during the same word, the branch condition is tested during the first part of the cycle. The set or reset occurs during the latter part of the cycle. Example of branching statement is S6, 0 (line 7), and an example of set or reset statement is 1->S6 (line 5). The sixth position of the X-register is set to 0 or 1, depending on the condition of S6 when the word is read out. The seventh position of the X-register is set to 1. At the end of the cycle, when the word has been executed, S6 equals 1.

ADDRESS 04B4: The data just read from address XXXX is not used and the data in the R-register, all zeros, is returned to core (STORE). The data in the D-register is added to itself with no carry in, but

A source	1000	1000
B source	1000	1000
D-register	0001	0000

with a carryout. Because of the C to the right of the arrow, the carry out sets position 3 of the S-register. Even though S7 is set to a one, the expression 1->S7 sets S7 again. A 1,1 branch is taken to address 04AF.

ADDRESS 04AF: The data in the R-register is decimal added to itself with a carry insert. The C to the left of the arrow allows S3 to set a carry into ALU. (R±R+C->RC). The data in the R-register is now 0000 0001. The C to the right of the arrow allows a carryout to set S3. Because there is no carryout, S3 is again zero. Main storage (MS) is again read (UV->MN). Position 6 of the S-register is set to one (1->S6). S6 had previously been set to a one so the branch test executes a 1,0 branch to address 04B6.

ADDRESS 04B6: The data just computed is written at address XXXX (STORE). Position 2 of the S-register is set to 0 (0->S2) but not before the branch tests (S2,0) determine that a 1,0 branch is to be taken to address 04AA. Remember, one of the conditions given before starting the problem was that S2 was set to a one.

ADDRESS 04AA: The expression WRITE has no effect at this time, because it follows the STORE operation of address 04B6. The data

in the D-register is added to itself with no carry inserted and

A source	0001	0000
B source	0001	0000
D-register	0010	0000

no carryout S3 = 0. Position 7 of the S-register is set to zero (0->S7). A 1,0 branch is taken to address 04AE.

ADDRESS 04AE: The data in the R-register is DECIMAL added to itself with no carry inserts. The resultant data in the R-register is 0000 0010. S3 remains zero (R±R+C->RC). Position 6 of the S-register is set to zero (0->S6). Advance 0,0 to address 04B4.

ADDRESS 04B4: The mnemonic, STORE, does not affect the problem because it does not follow a read call. The data in the D-register is added to itself with no carry insert (D+D+C->DC) and with

A source	0100	0000
B source	0010	0000
D-register	0100	0000

no carryout, S3=0. Position 7 of the S-register is set to a one (1->S7), but not before the branch tests determines that a 1,0 branch to address 04AE is called for.

ADDRESS 04AE: The data in the R-register is DECIMAL added to itself and becomes 0000 0100. No carryout therefore, S3 is still zero (R±R+C->RC). Position 6 of the S-register is set to zero (0->S6). Advance 0,0 to address 04B4.

ADDRESS 04B4: STORE, again accomplishes nothing for our program at this point. The data in the D-register is added to itself with no carry insert (D+D+C->DC) and

A source	0100	0000
B source	0100	0000
D-register	1000	0000

no carry out, S3=0. S7 is set to as one (1->S7). This position of the S-register was previously set therefore a 1,1 branch is taken to address 04AF.

ADDRESS 04AF: The data in the R-register is DECIMAL added to itself. There is no carry insert as S3 is a zero. No carryout results. The resultant data is 0000 1000 in the R-register. Core address XXXX is read (UV->MN MS). S6 is set to a one (1->S6) but not before a 0,0 branch is taken to address 04B4.

ADDRESS 04B4: The data just read is lost, and the computed data, 0000 1000, is returned to core (STORE). The data in the D-register is added to itself with no carry insert. A carryout results that sets S3 to a one (D+D+C->DC).

A source	1000	0000
B source	1000	0000
D-register	0000	0000

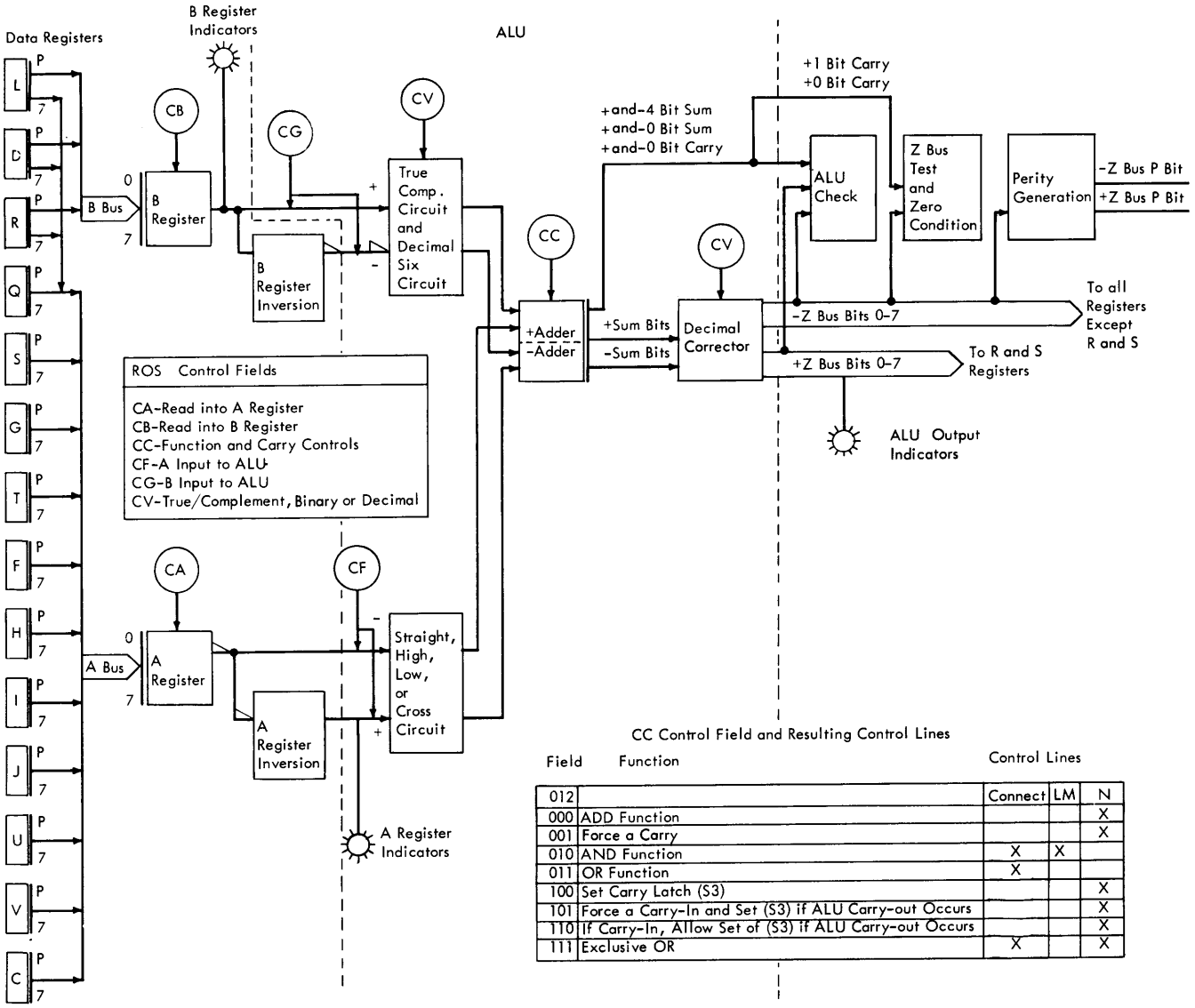
S7 is set to a one (1->S7). A 1,1 branch is taken to address 04AF.

ADDRESS 04AF: The data in the R-register is DECIMAL added to itself. A carry is inserted. No carryout results, and S3 is set to zero (R±R+C->RC). The result is R is 0001 0111. Core address XXXX is read again (UV->MN MS). S6 is again set to one (1->S6). A 1,0 branch is executed to address 04B6.

ADDRESS 04B6: The data read from address XXXX is lost and the data just computed, 0001, 0111 is returned to core (STORE). S2 is set to zero (0->S2). The data in the R-register is 0001 0111, which is 17 in decimal mode. Effectively then, this small 5 ROS-word loop has converted a binary number to a decimal number.

ARITHMETICAL LOGICAL UNIT (ALU)

- The ALU functions are controlled by ROS control fields.
- All arithmetic operations are performed in the ALU.
- ALU consists of control gating circuits for the A- or B-Registers, an adder, and a decimal corrector circuit. (Figure 2-55)



CC Control Field and Resulting Control Lines

Field	Function	Control Lines		
012		Connect	LM	N
000	ADD Function			X
001	Force a Carry			X
010	AND Function	X	X	
011	OR Function	X		
100	Set Carry Latch (S3)			X
101	Force a Carry-In and Set (S3) if ALU Carry-out Occurs			X
110	If Carry-In, Allow Set of (S3) if ALU Carry-out Occurs			X
111	Exclusive OR	X		X

Figure 2-55. ALU Data Flow and Controls

Data to be processed in the ALU is gated in from the A- and B-Registers, which are fed data from the A and B buses by ROS control fields CA and CB.

Each bit entering ALU is represented by two line levels, plus and minus. This two line system is used throughout the ALU, and for the output leaving ALU on the Z bus.

The data from the A-Register is gated to the adder under control of ROS field CF which may gate the data in the following combinations;

All eight bits straight into the adder

Block all eight bits to the adder

Four high bits only

Four low bits only

Cross the four high and low bits.

The data from the B-Register is gated to the adder under control of ROS fields CG and CV. The CG field gates the data from the B-Register in the following combinations;

All eight bits straight

Block all eight bits

Four high bits only

Four low bits only

This data is further gated by the CV field before entering the adder. The CV field indicates binary addition under true/complement control, or decimal addition under true/complement control. True or Complement binary addition, and complement decimal addition are handled in a similar manner. The data gated under these conditions is fed to the adder in true or complement form by the true/complement circuitry.

If the CV field indicates a decimal true addition, a special operation must be performed. This operation consists of adding six (0110) to both high and low 4 bit groups of the B-Register output byte before the number is gated to the adder. The reason for this operation and examples of the decimal add are contained in Chapter One under Packed Decimal True Addition.

The CC ROS control field causes the control lines, connect, LM and N to be activated in various combinations to control the operations of the adder circuitry (Figure 2-55). The major functions of the adder are:

ADD the data from the A- & B-Registers.

OR the data from the A- & B-Registers.

AND the data from the A- & B-Registers.

Exclusive OR the data from the A- and B-Registers.

Other functions of the adder, controlled by the CV field, are the handling of carries in and out of the adder, and the setting of the carry latch.

DECIMAL CORRECTOR

- Tests and corrects all decimal arithmetic results.
- All data passes through the Decimal Corrector circuitry to the Z bus.

The Decimal Corrector circuitry tests the results of a decimal operation, (either true or complement) leaving the adder. Binary results are gated through this circuitry unaltered. Each four bit group is considered separately and tested for a high-order carry. If a carry was not generated from the high bit of a four bit group, the decimal corrector circuitry will subtract six from that group. If a carry had been generated, zero is added to that four bit group. The output of the Decimal corrector is placed on the Z bus.

CARRY HANDLING

- Carries in the adder circuitry may occur from one position to the next higher position.
- Carries into adder position seven are controlled by the CC ROS field and Carry latch.
- Carry conditions are tested in various places throughout the ALU circuitry.

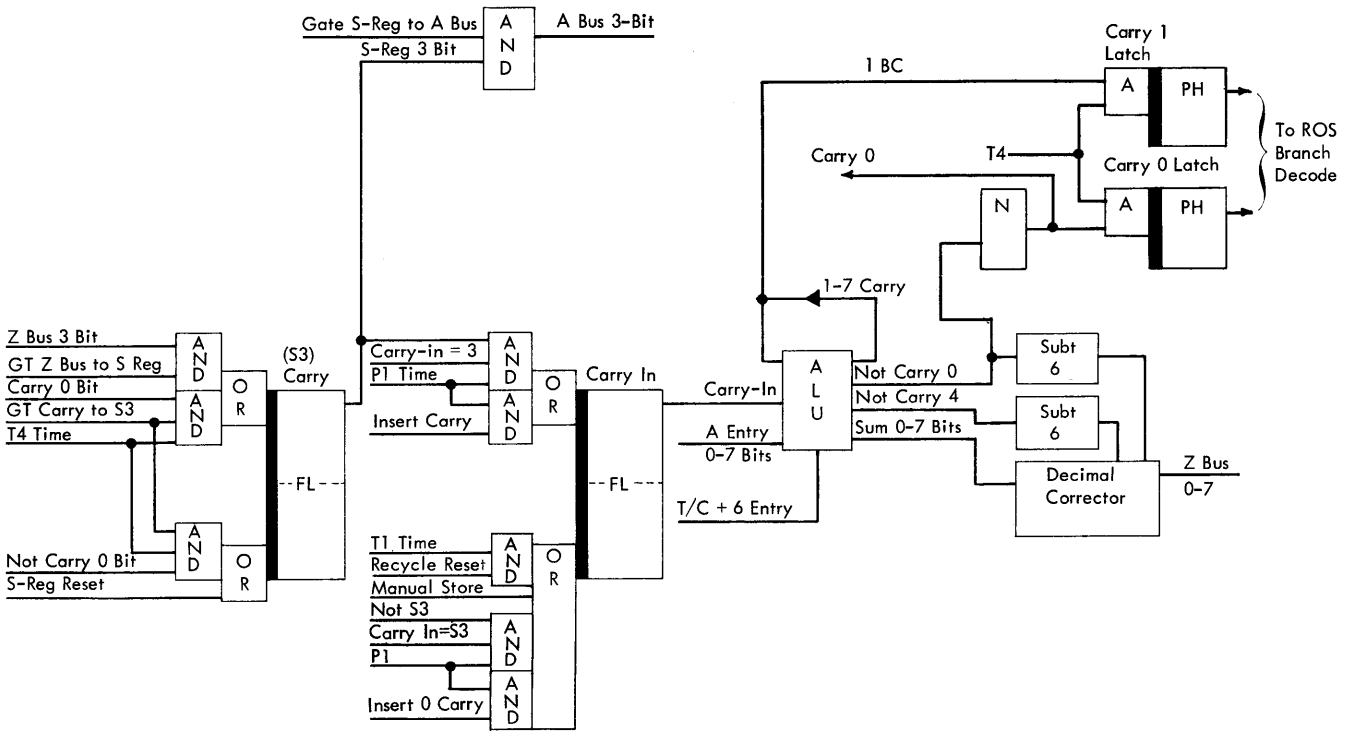


Figure 2-56. Carry Circuits

A carry that is developed within any of the adder bit positions is allowed to transfer to the next higher bit position. This is true for both binary and decimal additions.

The carry into bit position seven of the adder is gated in by the Carry-In Latch under control of the CC ROS control field. The Carry-In latch (Figure 2-56) is set by the Insert Carry line, or the Carry latch (S3) being on. The CC ROS control field also controls the setting of the Carry latch for certain microprogram functions. The three CC field decodings that control adder carry operation are:

CC Field	Function
001	Force a carry-in.
100	Set the Carry Latch (S3) if an Adder carry-out occurred.
101	Insert carry and set S3 if an adder carry-out occurs.
110	If a carry-in, allow set of S3 if adder carry-out occurs.

The carries that occur from adder positions 0 and 1 may be checked by the microprogram for, address checking, arithmetic overflows, or sign analysis. The 0 bit carry is also used to set the Carry latch (S3), if specified by the ROS control field CC. The carrier from bits 0 and 1 of the adder, may be used to set the branching bits 6, or 7 of the X-Register, if specified by the mnemonics AC or 1BC. The mnemonic AC used in the ROS branch line of a CLD box will set the 6 bit of the X-Register if an adder carry had occurred in the previous arithmetic statement. The mnemonic 1BC used in the ROS branch line will set the 7 bit of the X-Register if a carry occurred from the 1 bit position of the adder in the previous arithmetic statement.

When performing a true or complement decimal addition, the carries from adder positions 0 and 4 are tested, and the decimal corrector will subtract 6 from the 4 bit group for which a carry did not occur. Any carry produced, by this subtraction, from the high-order bit of a four bit group is ignored.

The 0 bit position of the adder is tested for a carry when doing complement decimal addition to determine if the result is in true or complement form. If a carry from the 0 bit position had occurred the result is in true form. If no carry occurred the number will have to be recomplemented.

ALU CHECK (SEE FIGURE 2-57)

- All Z-bus lines and the ALU Sum Zero, Sum Four and Carry-Zero lines are checked for complementary line levels.
- ALU Check will drop the CPU Clock Start Line if Check Stop is on.

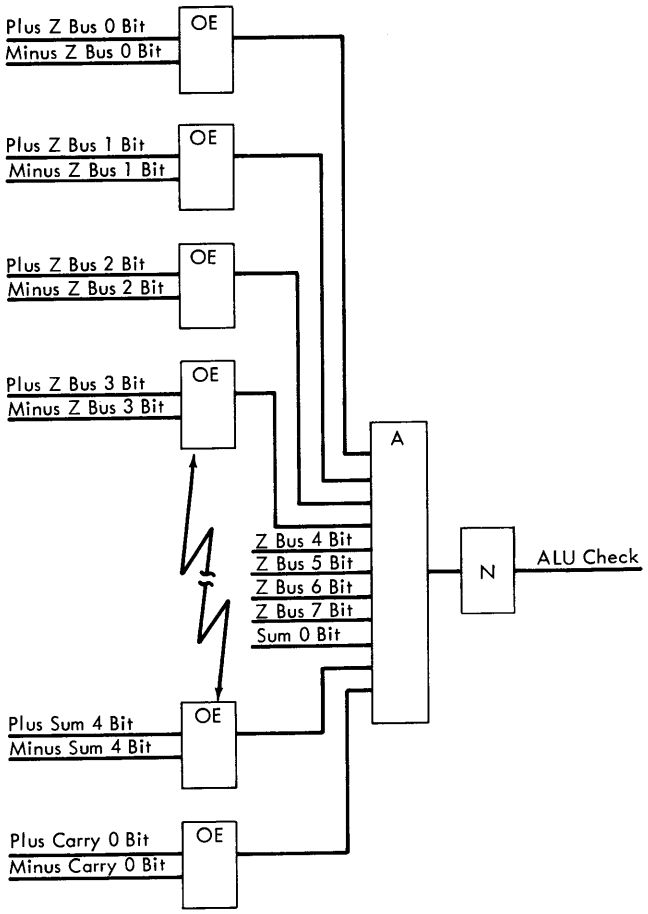


Figure 2-57. ALU Check

The use of two wire circuitry is also used on the output of the ALU decimal corrector to check for correct operation. Each bit position will have both a plus and a minus level output. For example, if bit 4 in ON, the output of the decimal corrector will produce "+L Z BUS 4 Bit" and "-L Z Bus 4 Bit". If the 4 bit position is OFF, these two lines will have the opposite voltage level output. Three bit positions are also checked directly from ALU in addition to the lines from the decimal corrector. All of these complementary lines are fed to exclusive-or circuits which produce plus level outputs if one and only one input is plus. If any OE has either both inputs plus or both inputs minus the output will be minus (-L). This minus level will produce a plus (+L) level output through an AI circuit to establish the line "+L ALU check". Correct operation will result in "-L ALU Check" (Not ALU Check). The correct operation then is to condition all OE's so that their plus outputs to the AND block in Figure 2-58 will produce a minus level out.

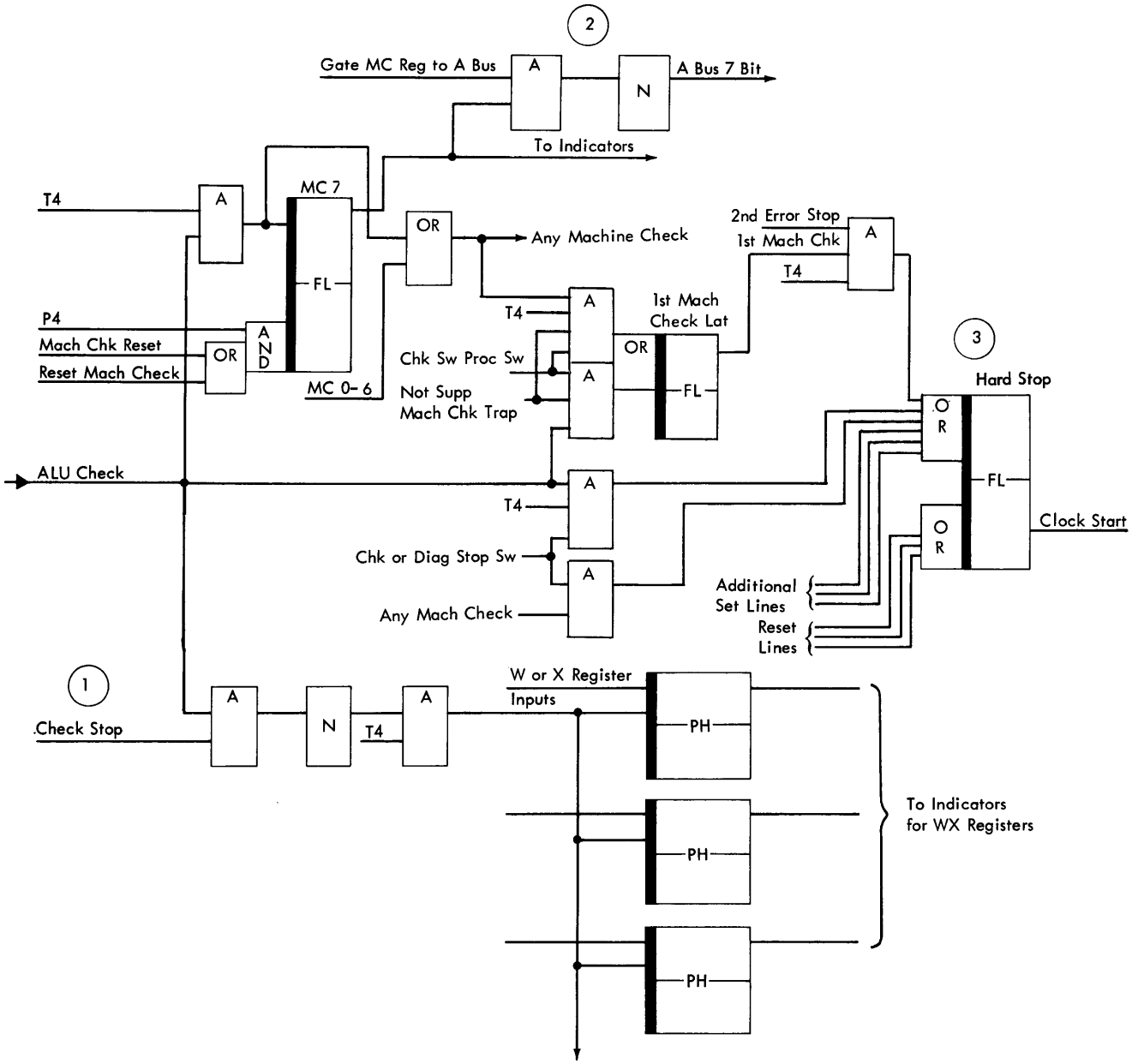


Figure 2-58. ALU Stop Check Controls

The output of the ALU check circuits is used several ways. See Figure 2-58.

1. It blocks the setting of the W and X-Register Indicating Latches if Check Stop is on.

2. It sets the Machine Register 7 Latch which can be used in microprogramming.

3. It produces "Any Machine Check" which, depending on switch settings, will cause a "Hard Stop" and therefore stop the CPU clock.

M2 CORE STORAGE UNIT

- The M2 storage unit is the 2.0 microsecond read-write storage unit for the IBM 2030.
- The M2 is a separately packaged unit within the 2030 frame.

The M2 memory provides the IBM 2030 Processing Unit with a 2.0 microsecond read-write storage unit. The basic unit of information stored is the eight-bit byte, with an additional bit added to maintain odd parity of data. Storage sizes are 8,192 positions (8K), 16,384 positions (16K), 32,768 positions (32K), and 65,536 positions (65K).

The M2 storage unit is a separately packaged unit that is installed inside the 2030. This separately packaged unit contains the controls, timing generator, core array, and sense/inhibit system for the storage unit. If the 2030 requires the full 65K of storage, two separate M2 units are installed in the base of the 2030 frame.

Because the unit is entirely separate from the 2030, communication between the two takes place over a number of signal and control lines known as the Memory/CPU interface. This interface transfers address information, input data, output data, and timing signals.

FOUR-BIT ADDRESSING

- A storage location is a place where something may be kept.
- A number assigned to a storage location is its storage address.
- Four binary digits form 16 different storage addresses.

By definition a storage location is a place where something may be kept. Examples of storage locations are shelves in a library or mail boxes. To facilitate finding things at different storage locations, it is convenient to assign a number to each storage location. These numbers become the storage addresses.

Let's begin our study of storage addressing with a simple sixteen position storage unit. We can then expand to larger systems. Finally, we will apply this addressing system to the actual core storage unit.

Using four binary bits, 16 storage locations can be assigned addresses. All items numbered 0000 that we wish to store are placed in storage location 0000; all items numbered 0001 that we wish to store are placed in storage location 0001; etc. It is now possible to find any item by selecting the storage location with the proper binary number. In Figure 2-59 storage location 0101 has been selected by the combination of binary bits that represent the number 0101.

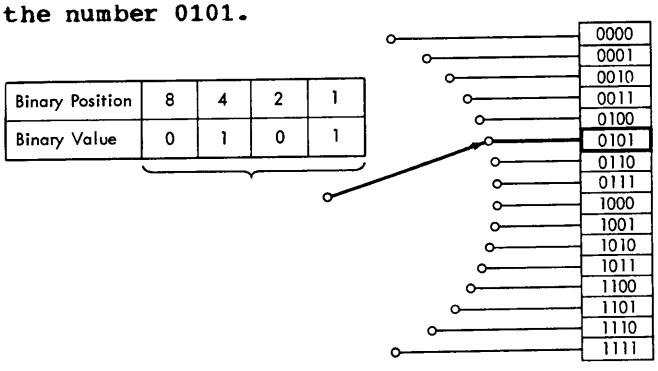


Figure 2-59. Four-Digit Addressing

SIX-BIT ADDRESSING

- Six binary bits form 64 different storage addresses.
- Addresses range from 000000 to 111111.

If the original four binary bits provide 16 combination of numbers (2^4=16), then six binary digits can be used to provide 64 combinations of numbers (2^6=64). By using these 64 numbers as storage addresses, it is possible to have 64 addressable storage locations with the address range of 000000 to 111111.

There are several ways to apply the six binary bits to an addressing scheme. To keep our theoretical addressing system compatible with the scheme used in the actual core storage unit, we will expand the original 4-digit addressing scheme shown in Figure 2-59. Thus, in Figure 2-60, the four low-order binary bits describe some number in the range 0000-1111, while the two high-order binary digits describe which of the four groups of 16 numbers is to be used. In the example shown, the four low-order digits 1111 combine with the two high-order digits 10 to select storage location 101111.

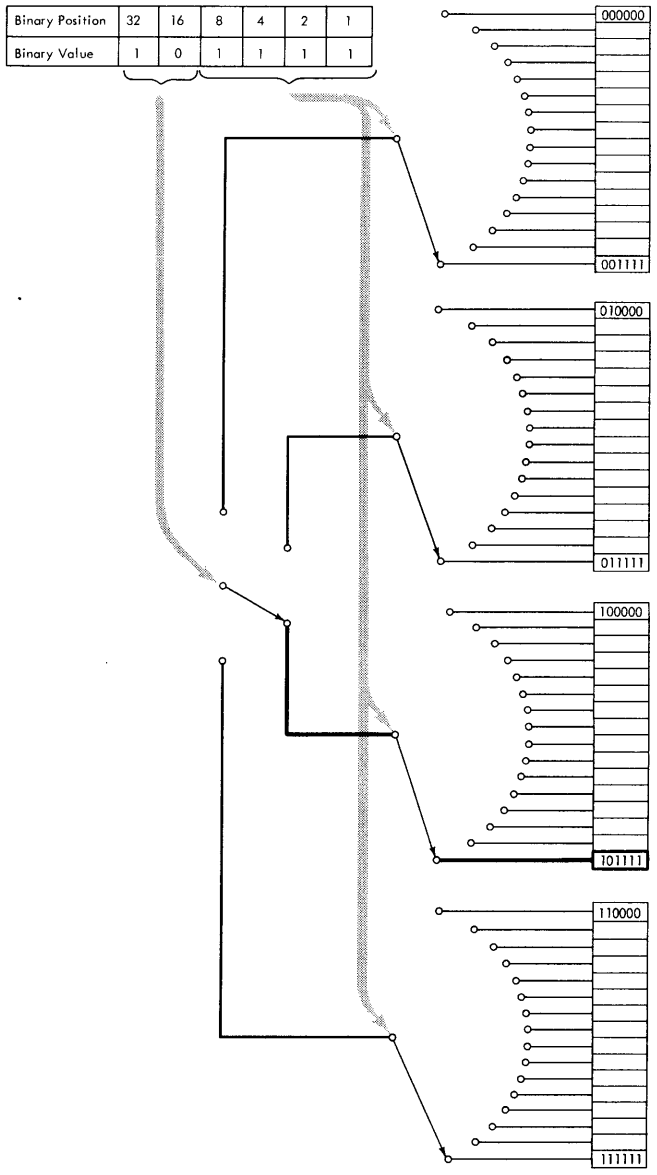


Figure 2-60. Six-Digit Addressing

TEN-BIT ADDRESSING

- Ten binary digits form 1,024 different storage addresses.
- Address range from 0000000000 to 1111111111.

If four additional binary digits are added to the 6 digit addressing scheme, it is possible to define 1,024 storage locations (2^{10}=1,024). To accommodate the extra bits in the addressing scheme, it is necessary to add another dimension (Figure 2-61). Once again, we will expand our theoretical addressing scheme in such a way as to keep it compatible with the actual core storage addressing system.

The four low-order binary bits describe some basic number from 0000 to 1111. This basic number is represented by a storage location in each of the 64 blocks of 16 storage locations. To further select the desired location, the next two binary bits describe one of for blocks of 256 storage locations. Each of these blocks is made up of 16 blocks of 16 storage locations each. The six low-order bits have narrowed the selection to 16 storage locations. With four high-order bits, it is possible to make a final selection of one of these 16. In the example shown, the four low-order bits (0000) plus the next two bits (01), plus the four high-order bits (1110), combine to form 1110010000.

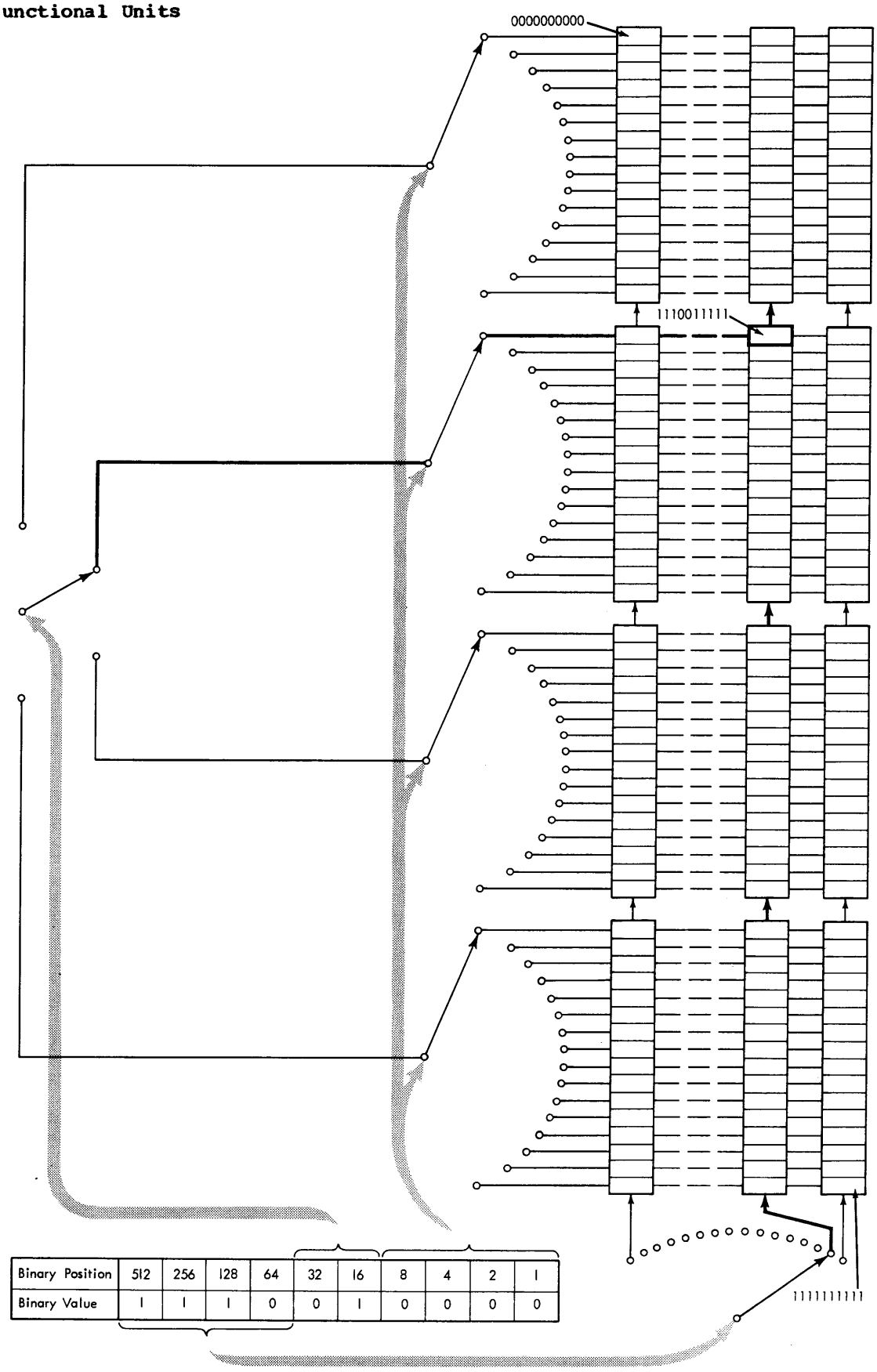
THIRTEEN-BIT ADDRESSING

- Thirteen binary bits address 8,192 storage locations.
- Address range 0000000000000 to 1111111111111.

In the first example of theoretical addressing, selection depended on one group of binary bits. This was expanded to selection by three groups of binary bits. If a fourth group is added to provide a further means of selection, the total amount of addressable storage can be increased.

With an additional three binary bits to provide eight more combinations of numbers, the total amount of addressable storage is increased by a factor of eight from 1,024 to 8,192 (Figure 2-62). These additional three binary bits provide a fourth direction to the addressing. Basic addressing is the same as shown in Figure 2-61 except that now there are eight groups of 1,024 storage locations. The three additional bits determine which of the eight groups of 1,024 is to be used. Notice that address selection depends on the coincidence of lines from four directions.

Up to this point, reference has been made only to storage locations, with no attempt to describe the actual storage device. In the examples given, the storage locations could have been in any storage device, depending on what was to be stored. In the IBM 2030 Processing Unit, a storage device is needed to store information, program instructions, constants, and data for processing. The storage device must be capable of storing and/or supplying the required information in the range of several microseconds. Thus the multiplicity of switches and boxes used to demonstrate storage addressing in Figure 2-62 are not satisfactory. However, it is possible to apply the same addressing scheme to faster storage devices. An investigation into the properites of magnetic core storage reveals that this device can be readily applied to produce an extremely fast storage device capable of storing the information required in the IBM 2030 Processing Unit.



Binary Position	512	256	128	64	32	16	8	4	2	1
Binary Value	1	1	1	0	0	1	0	0	0	0

Figure 2-61. Ten-Digit Addressing

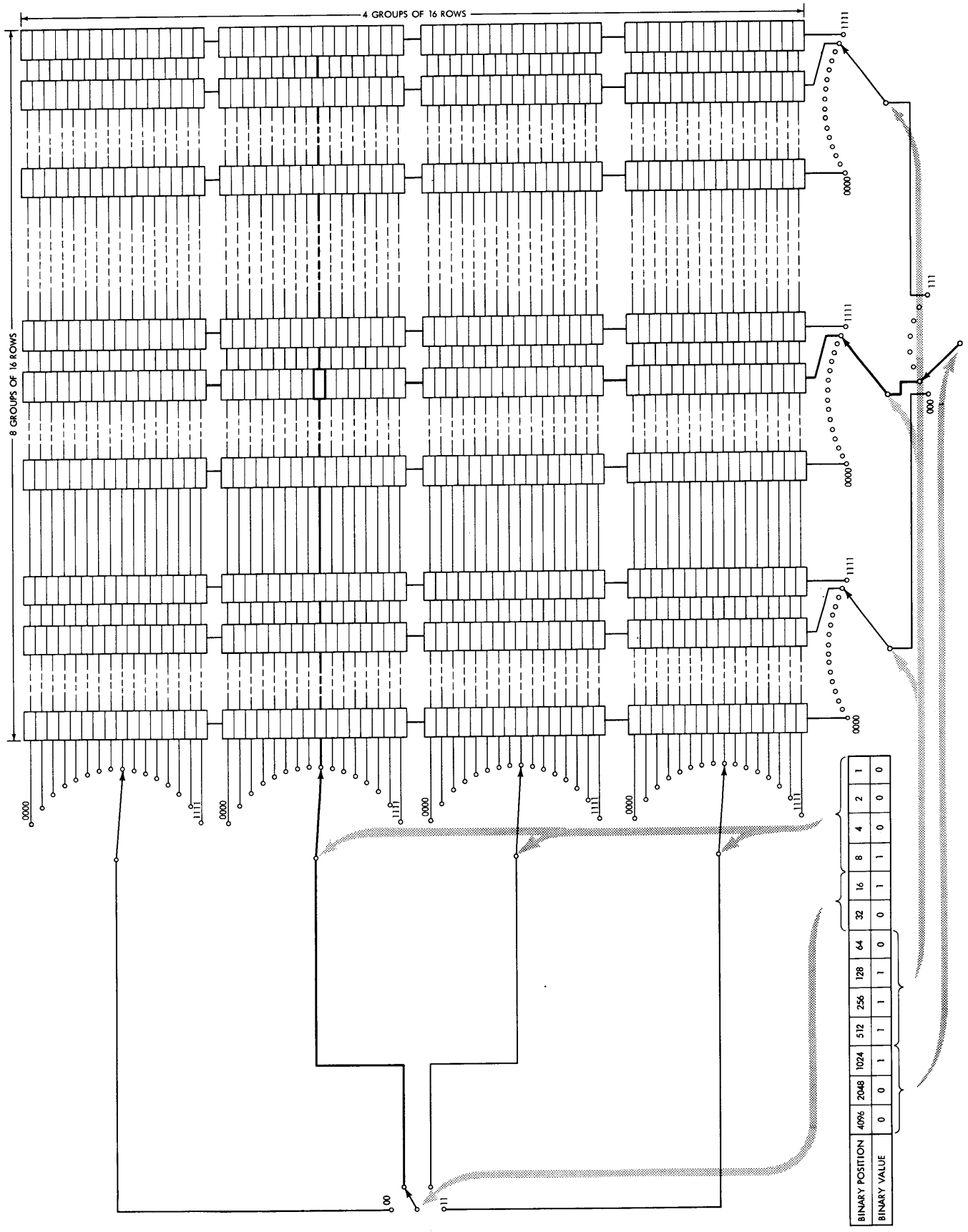


Figure 2-62. Thirteen-Digit Addressing

MAGNETIC CORE THEORY

- A magnetic core is a small, doughnut-shaped object made of ferromagnetic material.
- A core can be magnetized to either of two polarities.
- Once magnetized, the core retains its magnetism until it is deliberately changed by an external magnetizing force.
- The external magnetizing force is created by current-carrying wires.

A magnetic core is a tiny, doughnut-shaped object made of a ferromagnetic material. The properties of this material are such that if a ferromagneticic core is introduced to a sufficiently strong magnetic field, the core becomes magnetized. Furthermore, if the core is removed from the vicinity of the magnetic field, it remains magnetized. Unless it is deliberately changed, the core retains its magnetism indefinitely.

To deliberately change the core, it must be introduced to a sufficiently strong magnetic field of the opposite polarity. This causes the core to be magnetized in the opposite direction. Once again, unless deliberately changed, the core retains its magnetism indefinitely.

The fact that the core may be set to either of two states makes it a very useful binary storage device. If, when the core is magnetized in one direction a binary value of 1 is assigned, then a binary value of 0 results when the core is magnetized in the opposite direction.

Moving the core to the vicinity of a magnetic field is not a practical method of storing binary information. A more suitable method is to have a controllable magnetic field near the core itself. To magnetize the core in either of two directions, this magnetic field must be reversible in polarity. The desired result can be obtained by threading a wire through the center of the core. If a sufficiently strong current is passed through the wire, the core will be magnetized by virtue of the magnetic field created around the wire as the current passes through the wire. If the current through the wire is reversed, the core becomes magnetized in the opposite direction (Figure 2-63). Thus, by controlling the direction of current flow through the wire, it is possible to magnetize the core to a value of either binary 1 or binary 0. Changing the core from one magnetic polarity to another is called flipping the core.

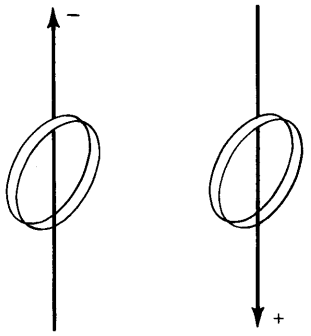


Figure 2-63. Magnetic Core

Using one wire for each core results in an expensive, inefficient storage device. With a slight change in the method of flipping the cores, it is possible to produce a more efficient device.

TWO-WIRE ADDRESSING

- Two wires pass through each magnetic core.
- Core is magnetized by additive effects of the two magnetic fields.

By passing two wires through the core, and by sending just half the current necessary to magnetize the core through each wire, the core is flipped by virtue of the additive effects of the two magnetic fields (Figure 2-64).

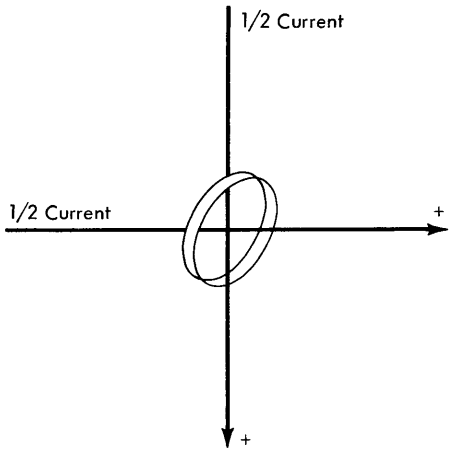


Figure 2-64. Half-Current Principle

If this half-current is passed through just one wire instead of both wires, the core is not flipped because the magnetic field is not great enough. Thus the core can be affected only by the coincidence of the two half-currents.

This half-current principle can be used to simplify the setting of cores by forming a screen of wires with a magnetic core at each intersection of the wires (Figure 2-65). By sending current in the appropriate direction through the appropriate pair of wires, the desired core can be flipped to the desired magnetic polarity without affecting the other cores in the group.

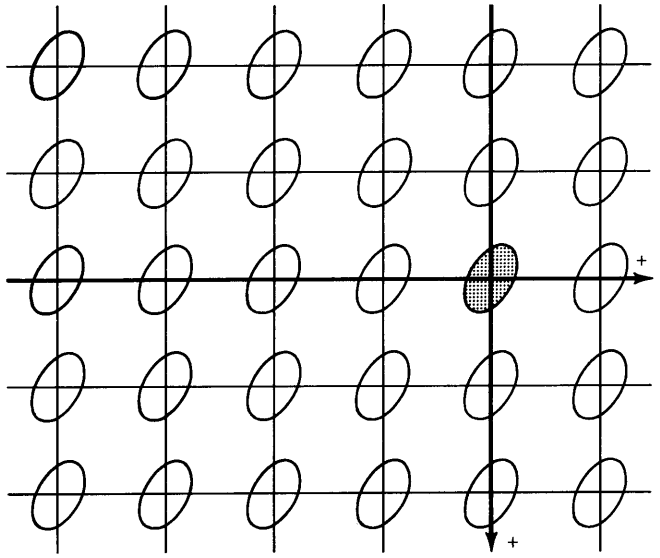


Figure 2-65. Coincident Current Addressing

CORE STORAGE ADDRESSING

- Transistors select address and provide selection current.
- Current flow in drive line determines core magnetic polarity.

In the IBM 2030, there are sixteen binary bits in each address. These sixteen bits represent four hexadecimal digits in the range 0000 to FFFF. For the 8,192-position storage device, the three high-order binary bits of the storage address are always logical zero, providing a binary address range of from 0000 0000 0000 0000 to 0001 1111 1111 1111 (hexadecimal 0000 to 1FFF). In Figure 2-66, magnetic cores have been added to the 8,192-position storage device discussed earlier. Also, hexadecimal digits have been introduced. Each box in the figure represents a magnetic core, and the lines between the boxes represent the screen of wires. If a battery is connected between the bottom address selection switch and the top address selection switch, and if a similar battery is connected between the left address selection switch and the right address selection switch, coincident current will be produced in one core. That one core will be flipped to a polarity dependent on the direction of current flow. The core addressed can be flipped to the opposite state by changing the direction of current flow.

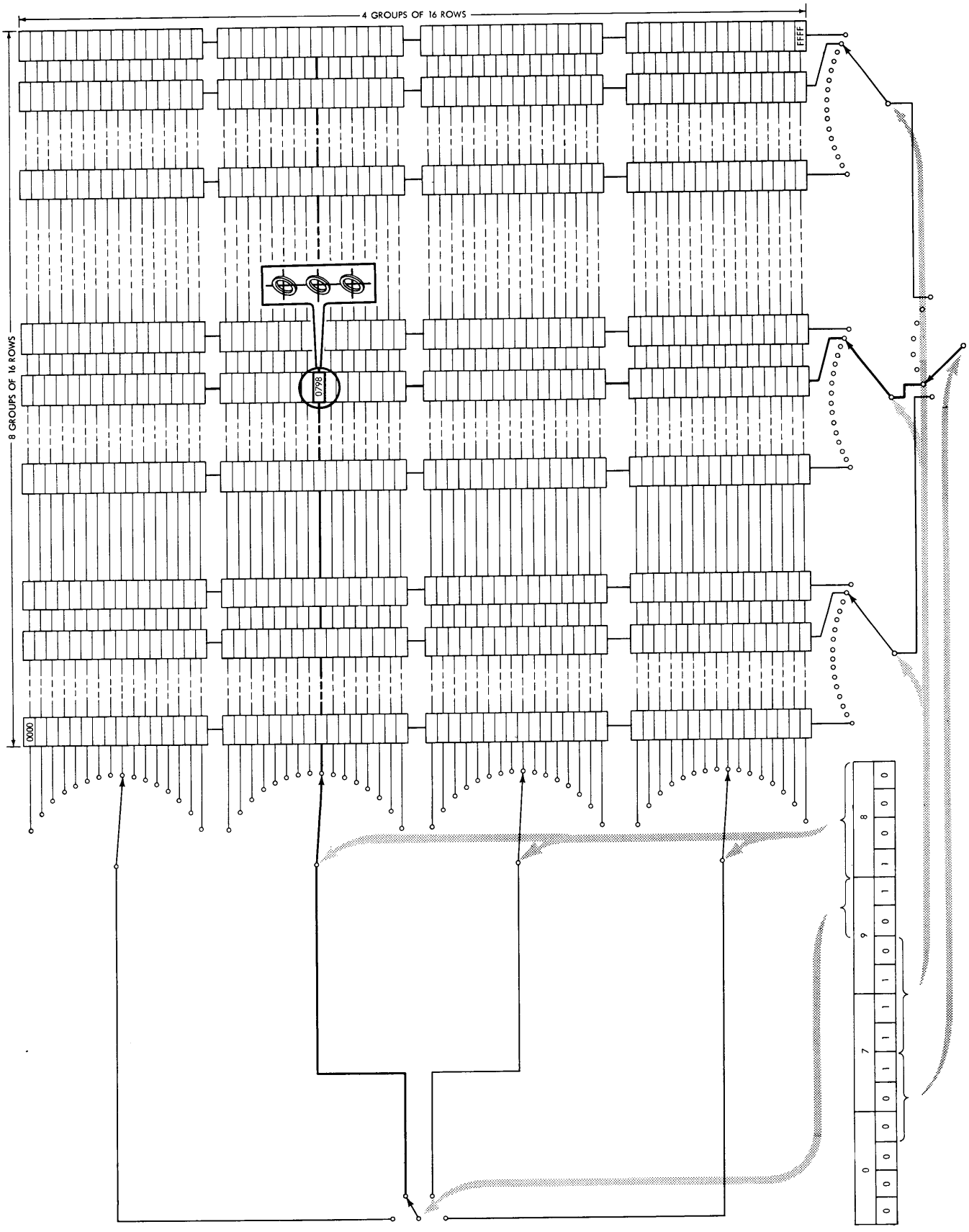


Figure 2-66. 8K Storage

The use of switches for address selection produces the desired result. However, having a series of switches is awkward. Moreover, it is impossible for such a system to be operated at the speeds required by the IBM 2030 Processing Unit. A much more practical approach is to let transistors do the switching for address selection. Figure 2-67 shows the windings through a typical core, and the method of driving the windings with sufficient current to flip the core to a logical 1.

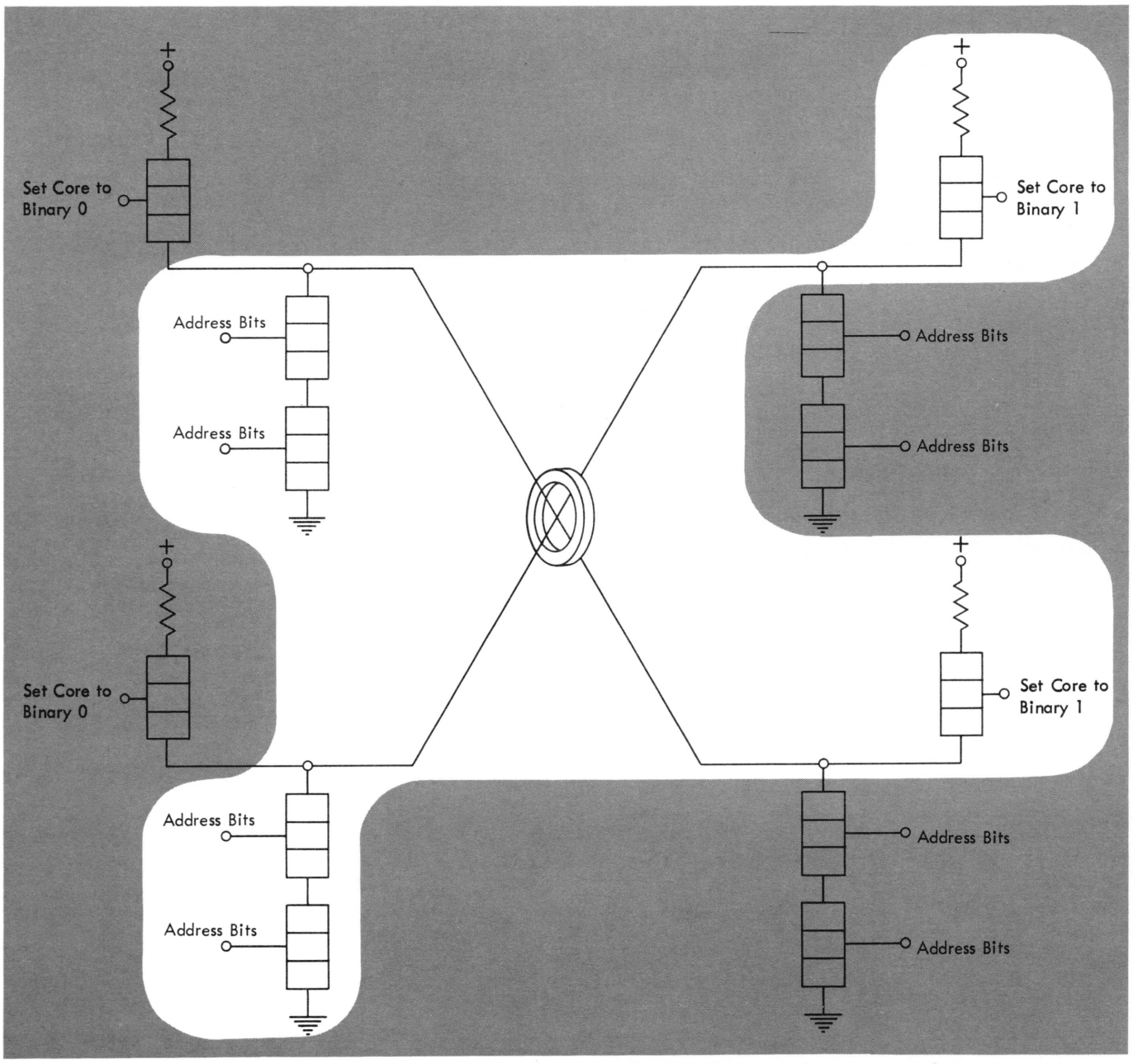


Figure 2-67. Core Storage Drive

SENSE

- Core magnetized to either of two polarities, represented as logical 1 or logical 0.
- Logical 1 called bit status, logical 0 called no-bit status.
- When core changes from bit status to no-bit status, a pulse is induced onto the sense winding.
- Changing core to bit status called writing.
- Changing core to no-bit status called reading.

A magnetic core stores information by remaining in either of two magnetized states. The two states are logical 1 and logical 0, thereby forming a binary storage device. The logical 1 state is called the bit status while the logical 0 is the no-bit status.

The stored information is of little value unless it can be retrieved from the core. To accomplish this, a wire is threaded through the core. When the core is flipped from one magnetic state to the other, a pulse is induced onto the sense wire. This pulse can be amplified and used to set a latch. The latch then provides the usable output from the core.

If a core is to contain information, it must be magnetized to the bit status. Accomplishing this requires coincident current in the proper direction. Flipping the core to the bit status is called writing, and the coincident current that causes writing is called write current. When information is to be retrieved from the core where it was stored, drive current is made to flow through the windings such that the core is flipped to the no-bit status. This causes the pulse that is amplified and used to set the latch (Figure 2-68). Retrieving this stored information from core storage is called reading, and the coincident current that causes reading is called read current. If coincident read current is made to flow through a core that is already in the no-bit status, the core does not flip, and there is no pulse induced onto the sense winding.

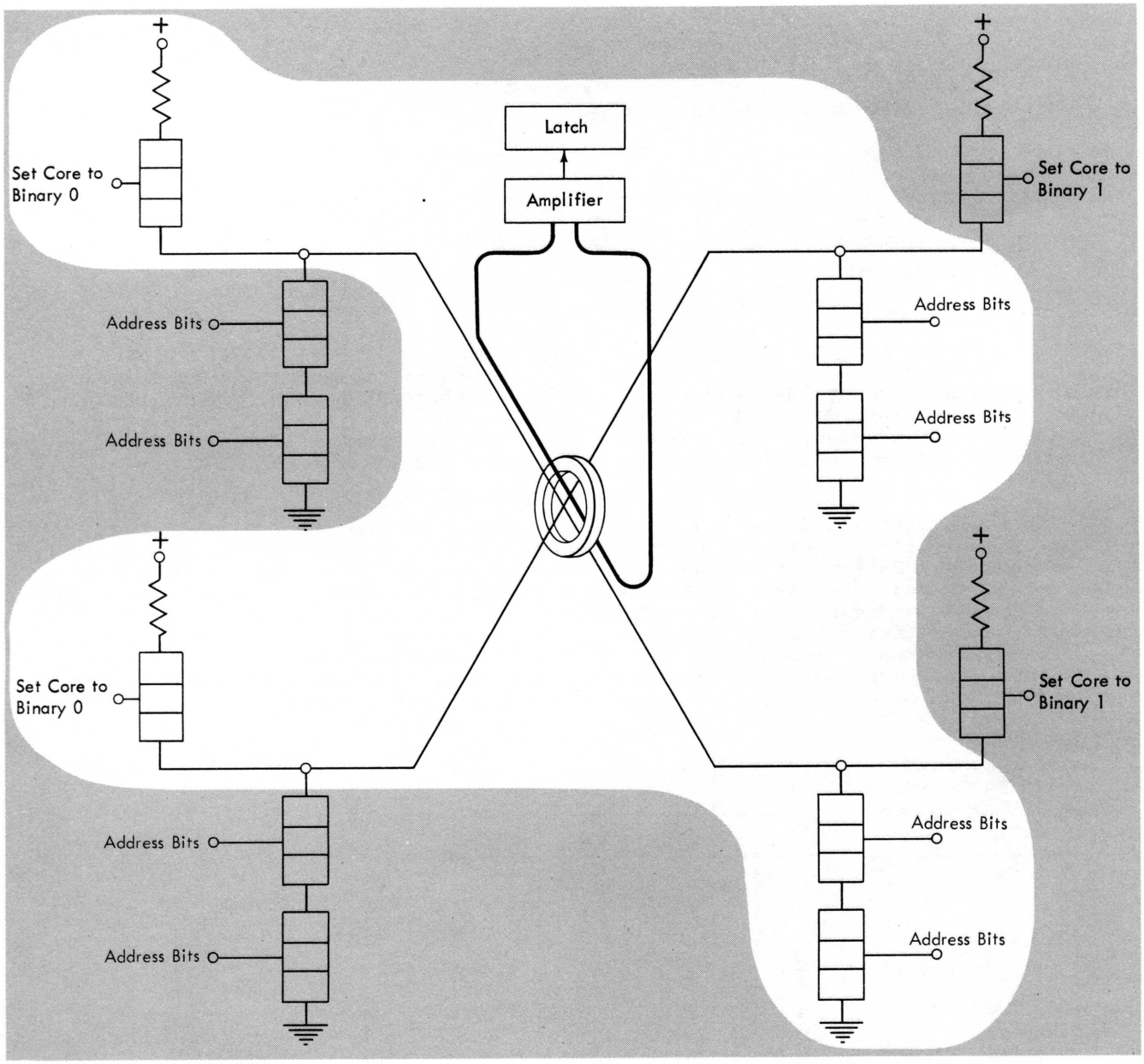


Figure 2-68. Core Read

Notice that to read out the addressed core requires the core to be flipped to the no-bit status. As far as the core itself is concerned, the information is lost. This type of information retrieval is called destructive readout. If it is necessary to have the information remain in the core after readout, it must be replaced on a subsequent write cycle.

STORAGE ADDRESS REGISTER

- M- and N-Register hold storage address.
- Together, M- and N-Register store a 16-bit binary address.
- Low-order 13 bits used to address basic 8K storage unit.

To retrieve a byte of information from core storage, the core-storage address must be available to the address decode network throughout the time when reading is taking place. Similarly, the address where a byte is to be written must be available during write time.

Two 9-bit registers (8 information bits plus a parity bit) are provided for core-storage addressing. Called the M- and N-Registers, these registers store a 16-bit binary core-storage address. The low-order position of the N-Register has the value of 1, the next position has the value of 2, and so on in binary increments up to the high-order position of the M-Register which has a binary value of 32,768 (Figure 2-69). Used together, these registers provide 65,536 different hexadecimal numbers, ranging from 0000 to FFFF. These numbers are the core-storage addresses for the core-storage unit in the 2030.

Thus far, we have discussed only the basic or 8,192-position block of core storage. To address this block requires only the low-order 13 bits of the M- and N-Registers (Hex addresses 0000 - 1FFF).

The remaining 3 high-order bits are used to complete the addressing shceme up to the maximum core storage size available (Hex addresses 2FFF - FFFF).

Name	M-Register								N-Register							
Position	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
Binary Value	3	1														
	2	6	8	4	2	1										
	7	3	1	0	0	0	5	2	1							
	6	8	9	9	4	2	1	5	2	6	3	1				
	8	4	2	6	8	4	2	6	8	4	2	6	8	4	2	1

Figure 2-69. Storage Address Register

8K STORAGE ADDRESSING

- Address decode takes place for each end of the drive lines.
- Four drivers, 16 gate decodes, and 64 gate transistors for each end of the X-drive lines.
- Eight drivers, 16 gate decodes, and 128 gate transistors for each end of the Y-drive lines.
- The gate transistor, with both base and emitter conditioned, is turned on to supply drive current.

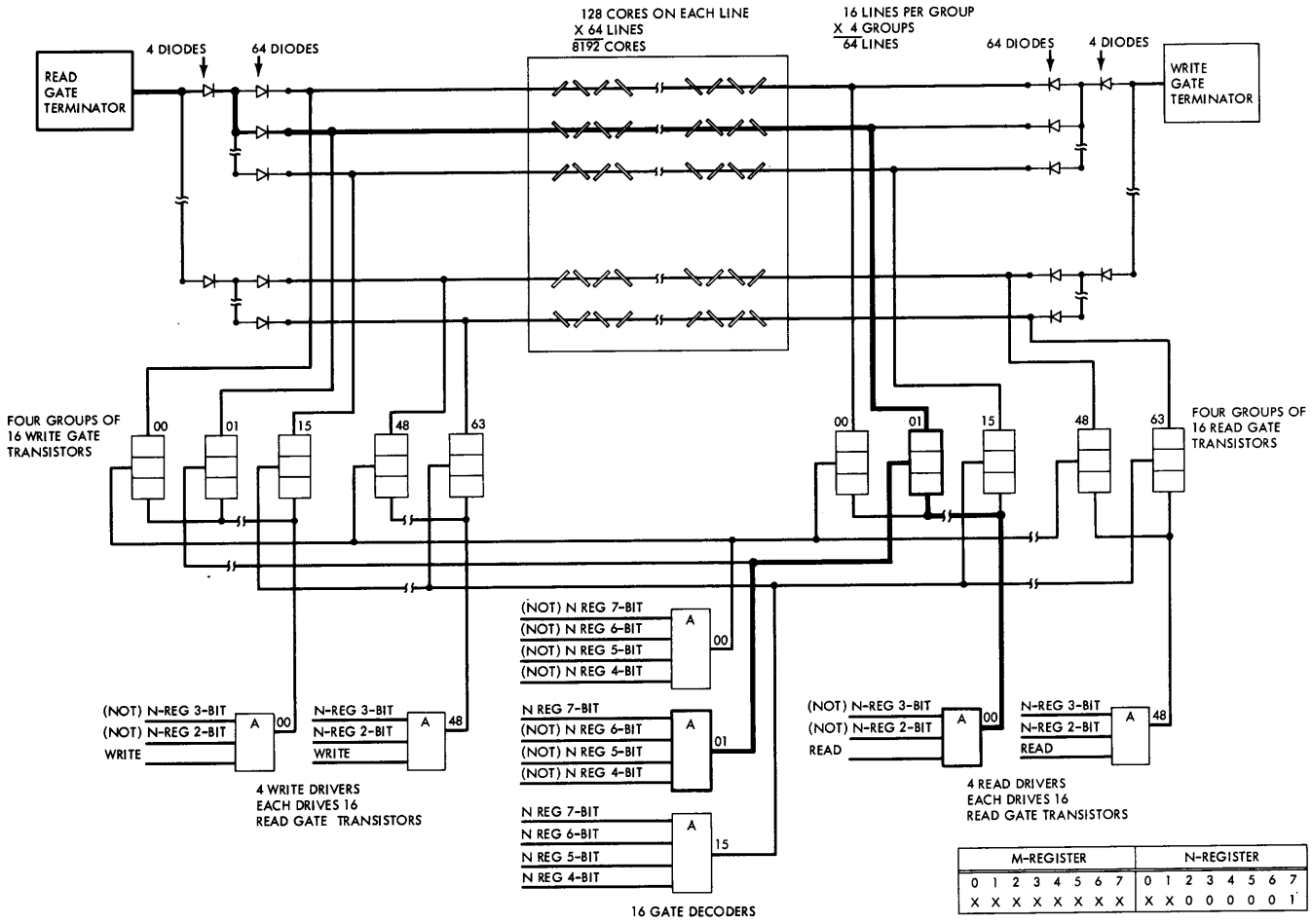


Figure 2-70. X-Drive Address Decode

The examples of Figures 2-67 and 2-68 assume certain address values to be present at the input of the transistor circuits. Developing these address values from the binary address presented to the core storage unit is known as address decoding. There is address decoding circuitry for the 64-line X-dimension, and similar address decoding circuitry for the 128-line Y-dimension. In addition, further address decoding takes place at each end of the lines. One end is address decode for read, and one end is address decode for write. Read and write address decoding for the X-dimension is shown in Figure 2-70.

Four drivers and 16 gate decode switches define which gate transistor is to conduct. In Figure 2-70, a single X-line has been selected to read from the binary value shown as follows: The N-Register 7-, 6-, 5-, and 4-bits combine with the read signal at a gate decode switch to condition the bases of four gate transistors (one in each of the four groups). From the N-Register 3- and 2-bits of the address, one of four read drivers is turned on to condition the emitters of one group of 16 read gate transistors. The connections form a matrix so that only one gate transistor will have both base and emitter conditioned for conduction. At the other end of the selected X-drive line, the read gate-terminator is turned on to complete the current path. Consequently, half-select current flows through 128 cores located on the selected drive line. To complete the addressing to a single core, one of the 128 Y-lines must be selected and driven with half-select current to provide coincident current in one core-storage position. To decode and drive a Y-line, the N-Register 1- and 0-bits, and M-Register 7- and 6-bits satisfy one of the 16 read gate decode switches. This conditions the bases of 16 of the 128 read gate transistors. The M-Register 5-, 4-, and 3-bits turn on one of the eight Y-read drivers. This conditions 16 read gate transistor-emitters. The one read gate transistor with both emitter and base conditioned turns on to provide read current for one Y-drive line.

CORE PLANES

- Nine cores required to store one byte.
- Nine core planes wired together to provide nine cores for each address.
- Coincident current produced in nine core windings.

Up to this point, we have been speaking of a single core plane consisting of 8,192 cores. This plane can store 8,192 bits of information. At any time, by correctly impulsing the proper drive line, a single bit of information can be stored or retrieved. In the IBM 2030, it is necessary to store a whole byte of information at each storage location. Each byte consists of eight information bits plus a parity bit. To store a complete byte requires nine cores (eight information cores plus a parity core). In Figure 2-71, nine 8,192-core planes have been stacked, and the address lines have been tied together serially. If two address lines are selected and are driven with coincident current, nine cores are affected (one in each core plane), because coincident current is produced in the same relative core in each of the nine identical core planes.

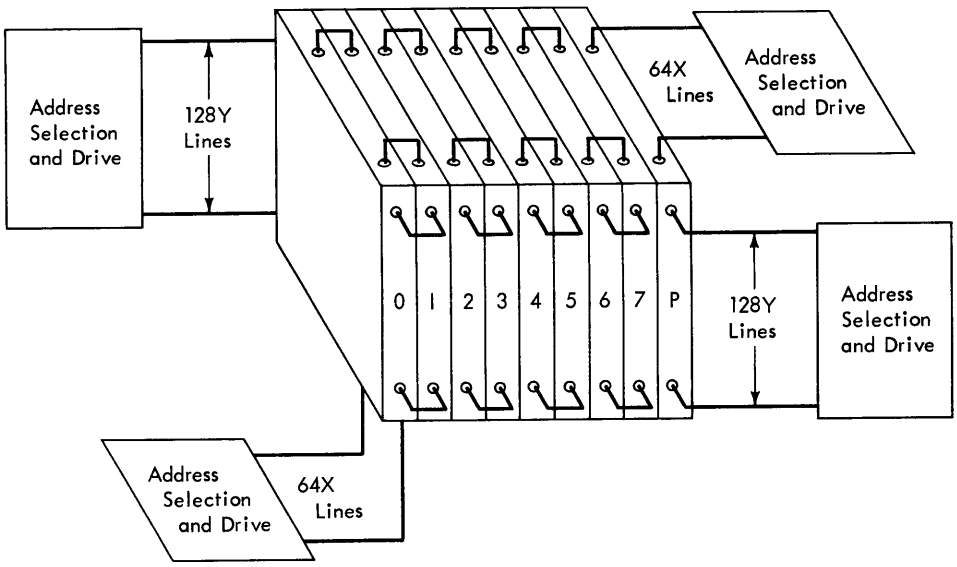


Figure 2-71. Core Plane Stacking

INHIBIT

- Controls writing in cores.
- Sense winding shared by inhibit circuits.
- Inhibit current prevents core from setting.
- Inhibit current opposes X-drive current.

The 8,192-position core storage unit shown in Figure 2-71 has a deficiency: it can store only all bits or no bits in a given storage location. To make the core-storage unit useful, we must be able to write in only the desired cores within a given core-storage location.

This is necessary because a core-storage position containing useful information has some cores set to logical 1 and some cores set to logical 0. Additional control over the writing of the cores is provided by the principle known as inhibiting. In Figure 2-68, we added a third wire to the core and used this wire to sense when the core flipped. We can now use this same wire to control writing in the core. This control is accomplished by sending current through the third wire during the time when writing is to take place (Figure 2-72). Called inhibit current, this current is equal to the drive current in the X-drive line, but is opposite in direction. The effect of this inhibit current cancels the effect of the current through the X-drive line, and the addressed core is not flipped.

A combined sense-inhibit winding is threaded through all the cores in each of

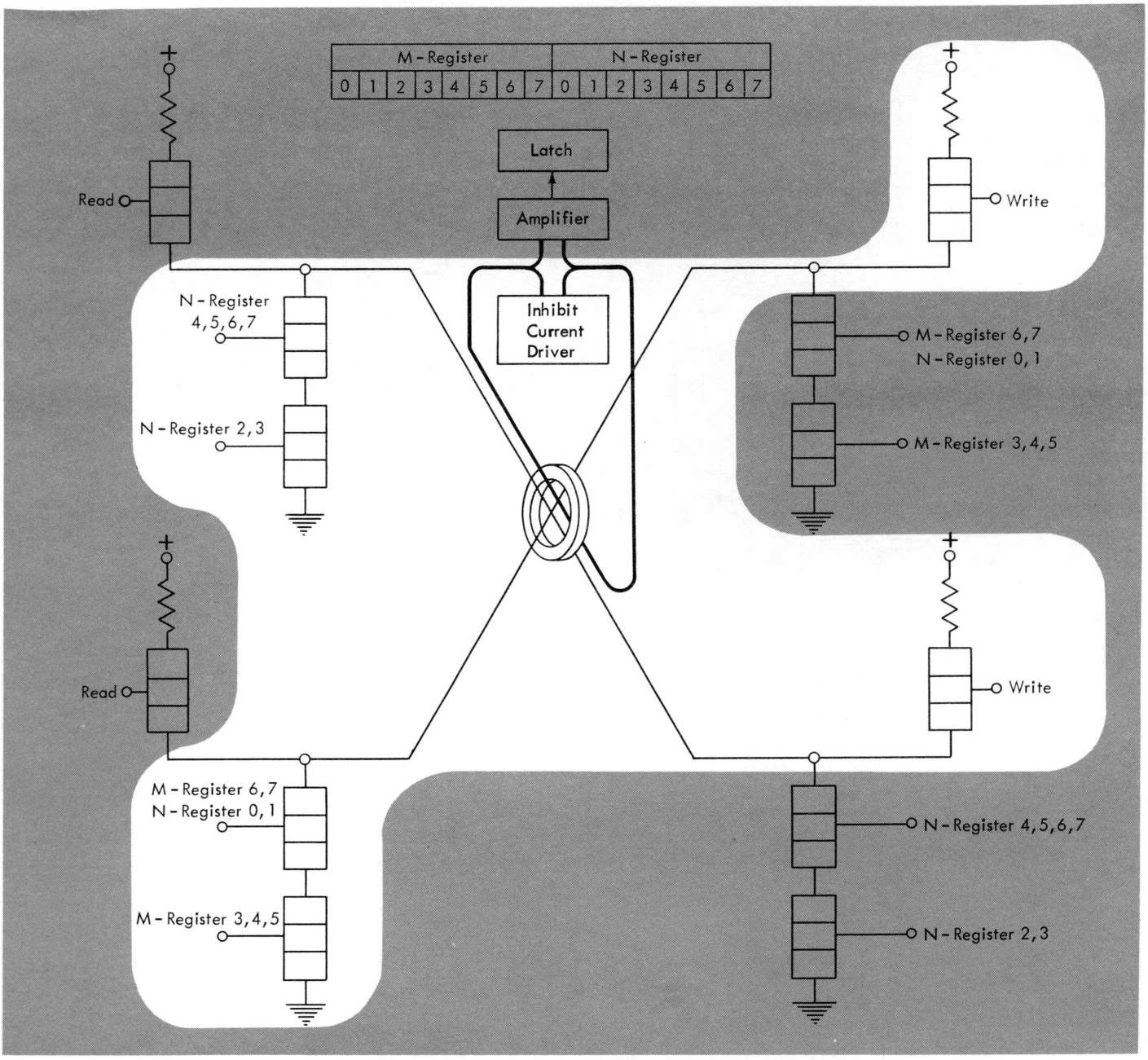


Figure 2-72. Inhibit

the nine core planes. For each core that is to be flipped to logical 1 during a write cycle, we block the inhibit current from flowing in the respective core plane. With no inhibit current flowing through the sense-inhibit winding of the addressed core, coincident current in the drive lines causes the core to flip. For each core that is to be blocked from flipping to logical 1 (ie: is to remain at logical 0), we allow inhibit current to flow in the respective core plane. Here the effect of one of the coincident currents in the drive line is cancelled by the effect of the inhibit current and the core does not flip.

For example; if a core position is to contain a byte coded with 0-, 1-, 2-, 5-, and P-bits, then inhibit current must be made to flow in the 3-, 4-, 6-, and 7-bit core planes so the 3-, 4-, 6-, and 7-bit cores in the addressed position are not set (Figure 2-73).

In the 2030 core-storage unit, each core plane has two sense-inhibit windings. Each winding is threaded through 4,096 cores. The two windings are functionally the same. However, using two windings for each core plane relaxes the design requirements for each inhibit current driver and sense amplifier, and provides more reliable operation.

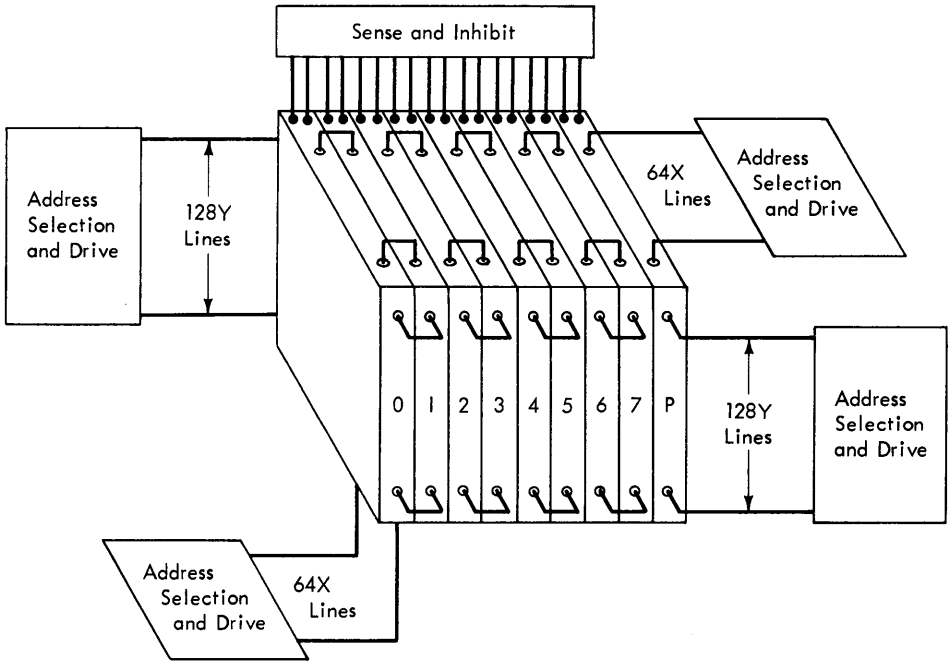


Figure 2-73. Composite Core Layout

AUXILIARY STORAGE FOR 8K

- Added area for CPU, and I/O control and status information.
- Additional addressing in Y-dimension only.
- Main-Auxiliary latch in CPU defines area to be addressed.
- M-Register 3-bit selects CPU local or MPX storage.

Included in the 8,192-position storage unit is an additional 512-position auxiliary storage section. In this section, 256 positions are reserved for use by the miltiplexor channel. The other 256 positions of local storage are used by the CPU for special and general purpose registers (Figure 2-74).

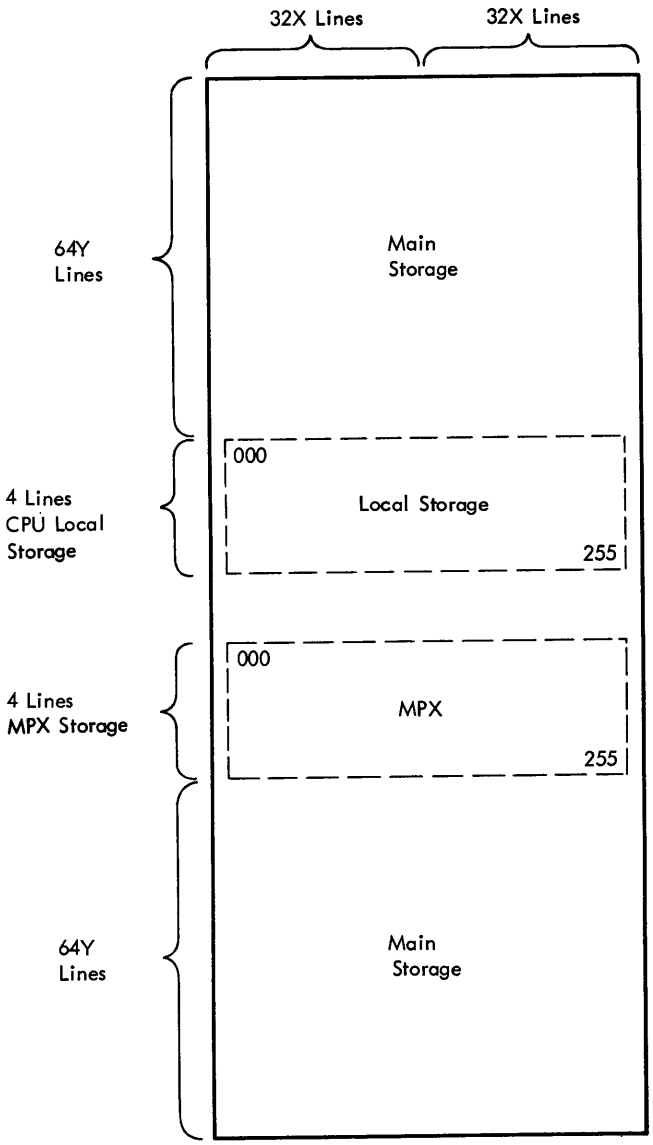


Figure 2-74. Auxiliary Storage

The additional 512 storage positions are formed by adding eight lines in the Y-direction direction (eight Y-lines intersect with 64 X-lines produce 512 additional storage positions). Eight Y read-gate transistors provide read current for the eight auxiliary Y-lines, while eight write-transistors provide write current for the auxiliary Y-lines.

At each end of the auxiliary Y-lines, the auxiliary gate-transistors are controlled by the Y-gate decode-switches and two special auxiliary drivers. When an address in the range of 00-255 is placed in the M- and N-Registers, it refers to one of three storage positions. The desired position may be in main storage, CPU local storage, or a multiplexor storage. To select which of the three areas is to be addressed, a latch in the CPU specifies whether to use main or auxiliary storage. However, just knowing that the desired address is in auxiliary storage is not enough, because there is more than one area of auxiliary storage. To select CPU local storage or multiplexor storage in the 8K storage unit, the M-Register 3-bit is set by the CPU in a code that determines which area is to be addressed. In the case of the 8K storage unit, if the M-Register 3-bit is zero, then the desired address is in multiplexor storage. If the M-Register 3-bit is one, then the desired address is in CPU local storage.

Figure 2-75 shows auxiliary Y read-gate selection when the address 174 is placed in the M- and N-Registers. The Y gate-decode switch that is turned on by the M- and N-Register contents conditions the bases of 18 Y gate-transistors (16 main Y gate-transistors and two auxilairy Y gate-transistors). However, only one Y gate-transistor is further conditioned by a Y-driver. In this case, the multiplexor read driver is turned on because the M-Register 3 bit is zero and because the CPU Main-Auxiliary latch is set to Auxiliary.

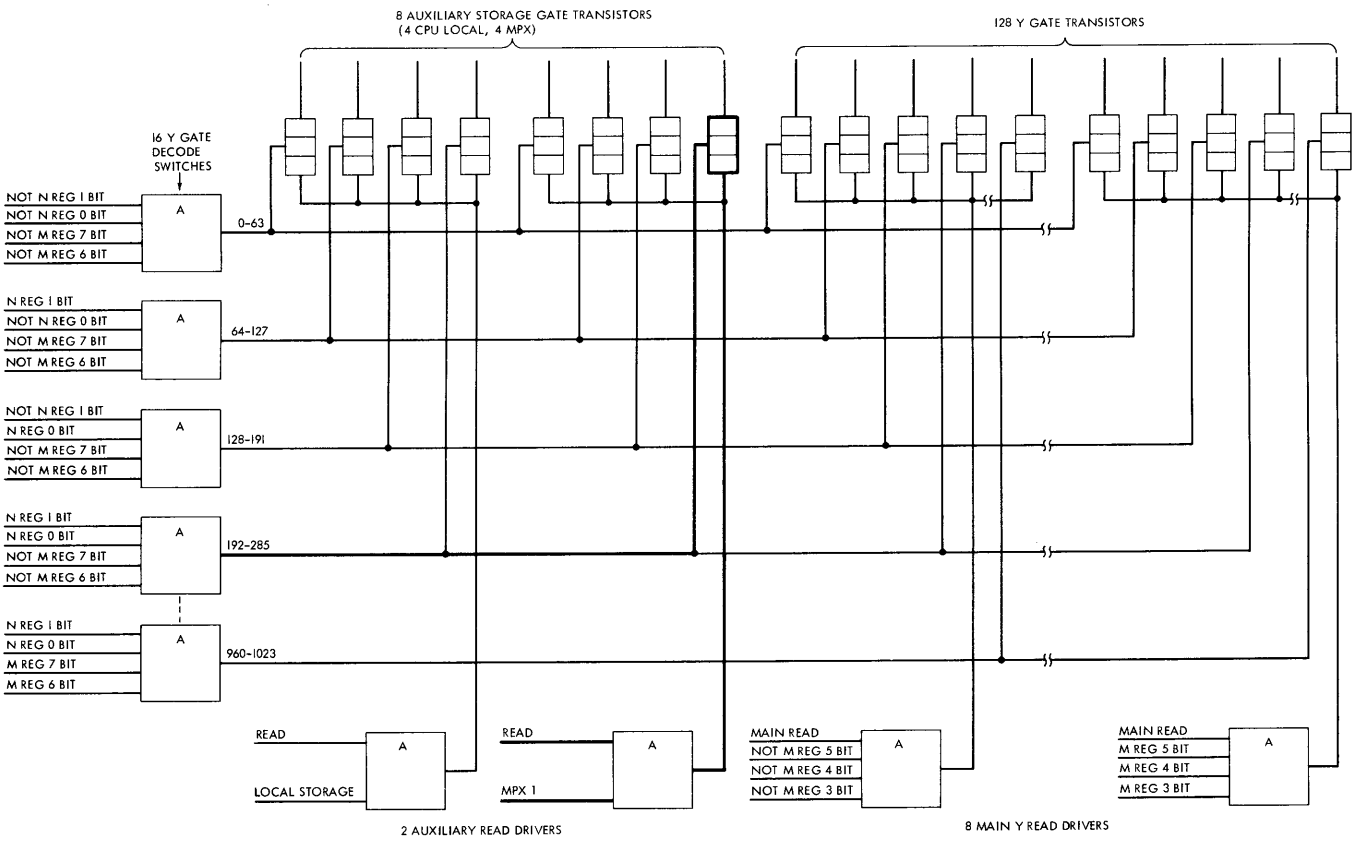


Figure 2-75. Auxiliary Storage Gate Decode

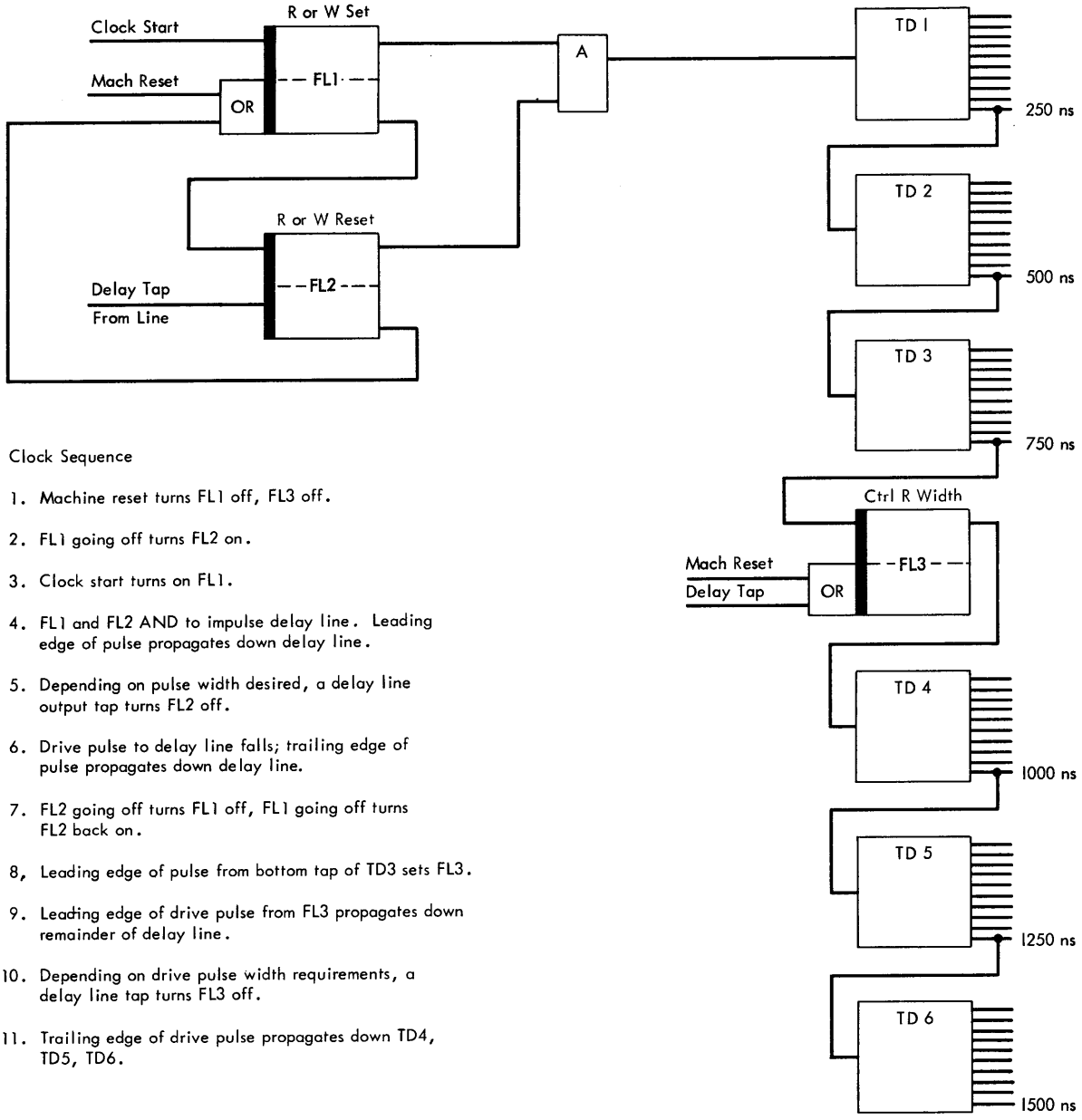
STORAGE CLOCK

- There is a separate clock for the core storage unit.
- Delay lines produce timing pulses.
- Control latches develop delay line drive pulses.
- Read and write clocking pulse latches form storage drive pulses.
- The storage clock is started by a signal from the CPU.
- Once started, the clock operates for a complete cycle.

The core-storage unit is operated on a cycle-by-cycle basis. If a byte of information is to be retrieved from the core-storage unit, a read cycle in initiated. During the subsequent read cycle, a storage position is addressed and the byte of information stored in that position is read out to the data register. If a byte of information is to be placed into core storage, a write cycle is initiated. During the subsequent write cycle, a storage position is addressed and the desired information is placed into the addressed byte location.

A storage clock provides the necessary timing pulses and gates to operate the storage unit on a cycle-by-cycle basis. This clock is started by the read or write signal from the CPU. Once started, it operates for a complete read or write cycle.

For example, suppose a position is to be read out, the byte of information thus obtained is to be used in a computation, and the result of the computation is to be placed back into the same storage position. The CPU specifies a storage location by placing a storage address in the M- and N-Register. The storage circuitry is signaled to read and the storage clock is started (Figure 2-76). A storage read cycle results, during which time the desired storage location is read out, and the resulting byte is placed in the CPU R-Register. The CPU then makes the necessary computation and places the result back into the R-Register. Once again, the storage unit clock is started. This time, however, the storage unit is signaled to write. A storage write cycle results, during which time the byte from the R-Register is written into the addressed storage position. In each case (read and write), the clock operated for a complete cycle once it had been started.



Clock Sequence

1. Machine reset turns FL1 off, FL3 off.
2. FL1 going off turns FL2 on.
3. Clock start turns on FL1.
4. FL1 and FL2 AND to impulse delay line. Leading edge of pulse propagates down delay line.
5. Depending on pulse width desired, a delay line output tap turns FL2 off.
6. Drive pulse to delay line falls; trailing edge of pulse propagates down delay line.
7. FL2 going off turns FL1 off, FL1 going off turns FL2 back on.
8. Leading edge of pulse from bottom tap of TD3 sets FL3.
9. Leading edge of drive pulse from FL3 propagates down remainder of delay line.
10. Depending on drive pulse width requirements, a delay line tap turns FL3 off.
11. Trailing edge of drive pulse propagates down TD4, TD5, TD6.

Figure 2-76. Delay Line Clock Drive

The storage clock consists of a series of delay lines, delay line control latches, and read and write clocking pulse latches. The control latches develop the timing of and control the width of the pulse that drives the delay line. The delay line consists of six separate delay lines connected in series. Each delay line has ten outputs. There is a 25 nanosecond delay between each of the ten outputs for a total delay of 250 nanoseconds per delay line. Connected in series, the six delay lines produce a total delay of 1,500 nanoseconds from the start of the drive signal.

The pulses required to operate core storage are formed by the read and write clock pulse latches. The appropriate delay-line taps are wired to the set and reset inputs of these latches to develop the required pulses at the outputs of these latches. The same delay line is impulsed regardless of whether a storage read cycle or a storage write cycle is to take place. The tap outputs are then gated to either the read clock pulse latches or the write clock pulse latches to cause either a read or a write cycle to take place (Figure 2-77 and 2-78).

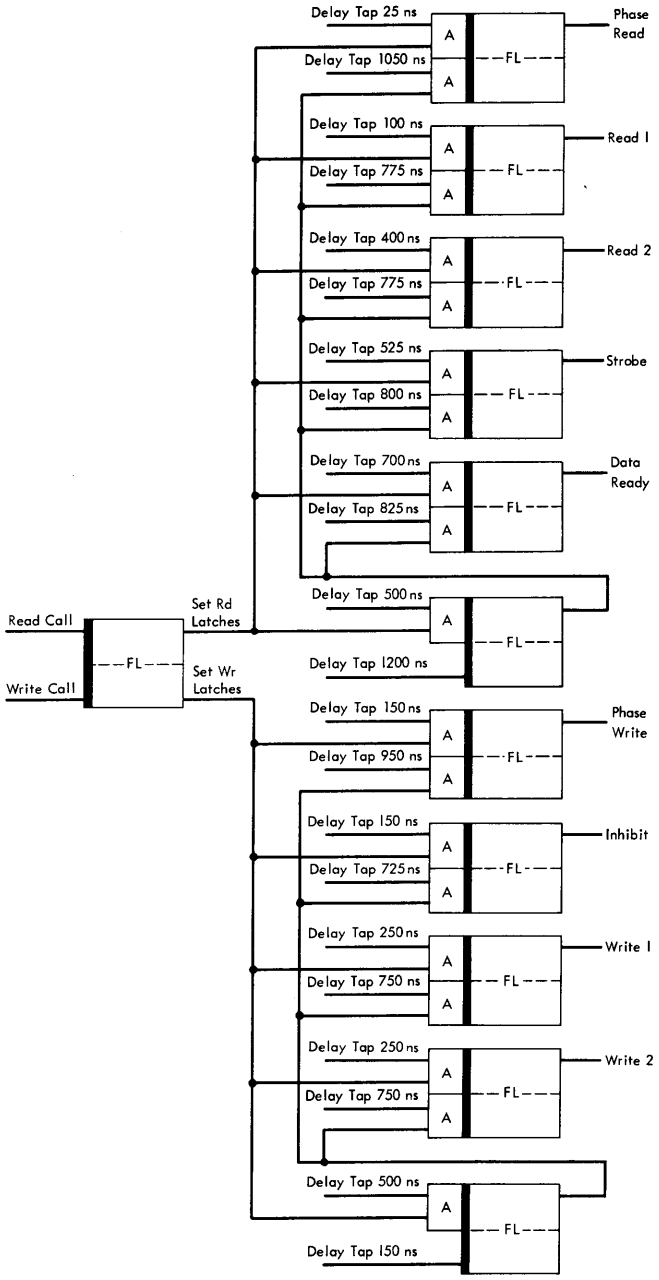


Figure 2-77. Memory Clock

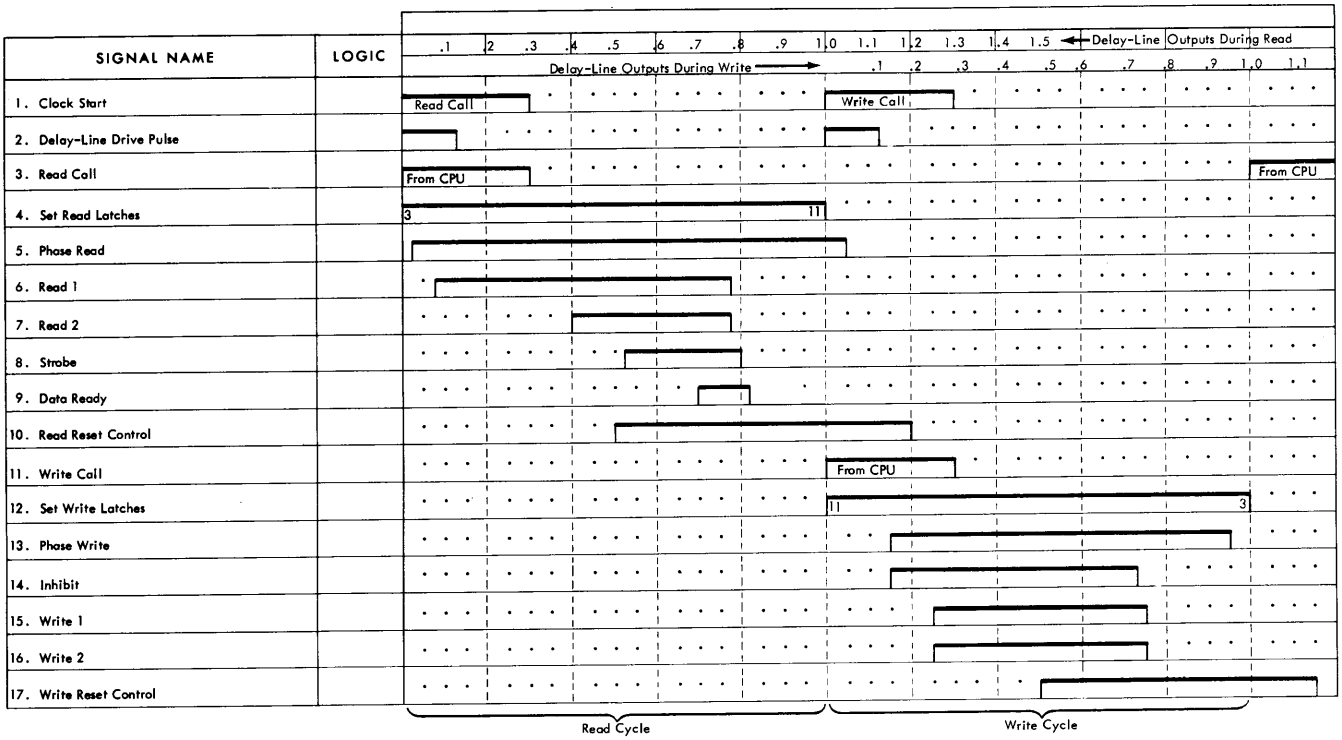


Figure 2-78. Memory Clock Timings

8K STORAGE SUMMARY

Before we go on to larger core storage units, let's review the 8K unit by listing the quantities of different components. If you understand how these quantities give the required addressing configuration, you will have an easier time understanding the larger storage units.

For an 8K storage unit, there are:

64X drive lines
64X read gate transistors
64X write gate transistors
16X gate decode switches
4X read drivers
4X write drivers

128Y drive lines
128Y read gate transistors
128Y write gate transistors
16Y gate decode switches
8Y read drivers
8Y write drivers
4Y CPU local storage drive lines
4Y MPX storage drive lines
1 CPU local storage read driver
1 CPU local storage write driver
1 MPX storage read driver
1 MPX storage write driver

As an example of how this provides a convenient review, consider the 64X lines. At each end of each of the 64X lines, there is a gate transistor. That means there is a total of 128 gate transistors. Checking the preceding list reveals that there are 64X read gate transistors and 64X write gate transistors, for a total of 128X gate transistors. For either group of gate transistors, each of the 16 gate decode switches conditions the bases of four of these gate transistors. Similarly, each driver conditions the emitters of 16 gate transistors. The resulting matrix produces only one gate transistor with both base and emitter conditioned. Therefore, only one gate transistor turns on, and only one X-drive line has current flowing through it (Figure 2-70).

PHASE REVERSAL ADDRESSING (16K)

- Phase reversal principle allows twice as many storage positions to be addressed with the same drive circuitry.
- Phase reversal takes place between 8K blocks.
- Y-drive lines wired through phase reversal plane; X-drive lines are not.
- No cores in the phase reversal plane.

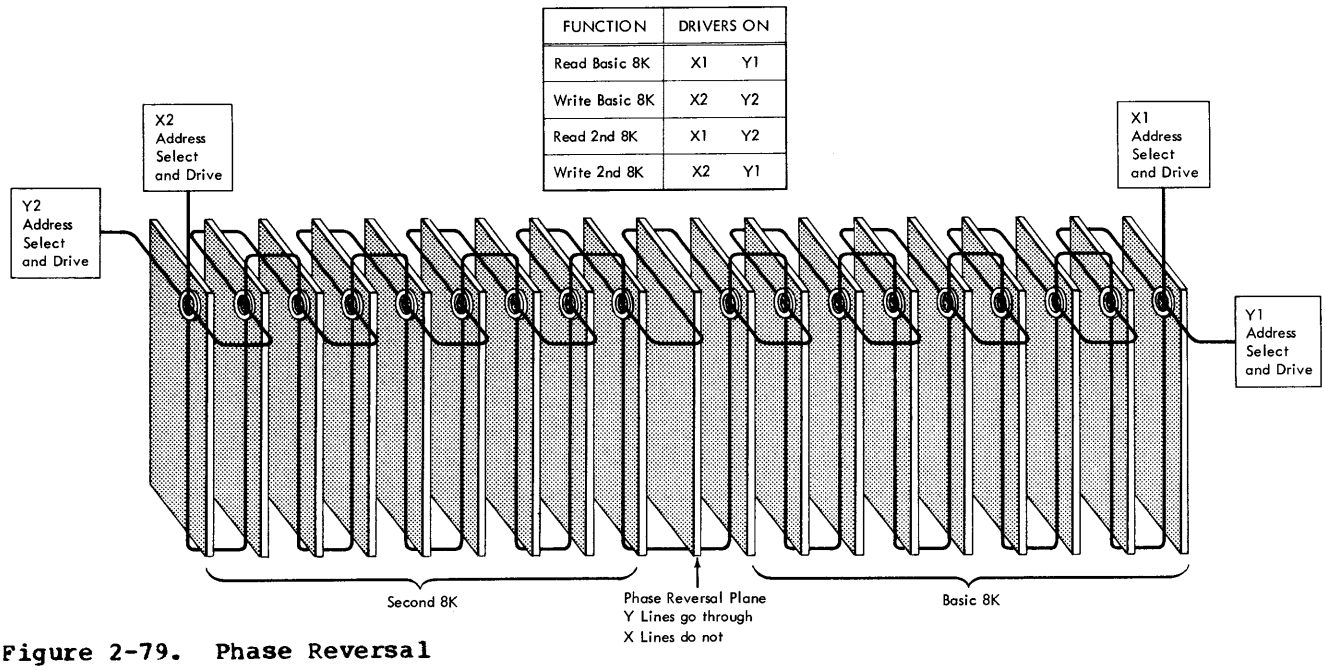


Figure 2-79. Phase Reversal

The basic 8K storage unit can be expanded to 16K without changing the basic drive scheme or the drive circuitry. This is accomplished by wiring the same drive lines through two 8K blocks of storage. Between the two 8K blocks of storage is a phase reversal plane containing no cores. The Y-drive lines are wired through the phase reversal plane, whereas the X-drive lines are not wired through the phase reversal plane (Figure 2-79).

When the addressing circuitry selects and drives one X- and one Y-drive line, two storage positions (18 cores) are addressed. However, the drive currents are in phase in one 8K section and out-of-phase in the other 8K section. Reversing the direction of one of the drive currents causes the drive currents to be in phase in the second 8K section. Reading and writing are controlled by reversing both drive currents as shown in Figure 2-79. To read out a core-storage position in the first 8K block, drivers labeled X1 and Y1 are caused to supply drive current while the circuitry at the other ends of the drive lines accepts these currents. The result is in-phase read current in the desired position in the basic 8K block. To write into the same position, drivers X2 and Y2 supply drive current and circuitry at the opposite end provides a path for these currents. The result is in-phase write currents in the desired position in the basic 8K block.

To read out a core-storage position in the second 8K block, drivers X1 and Y2 are turned on. Circuitry at the opposite ends of the drive lines is conditioned to complete the drive current paths. The result is in-phase read current in the desired position of the additional 8K block. Notice that the corresponding position in the basic 8K block is not affected because the read currents in this block are out-of-phase

Writing into a core-storage position in the second 8K block requires drivers X2 and Y1 to be turned on. Circuitry at the opposite ends of the drive lines must be conditioned to accept drive current. The result is in-phase write current in the position in the additional 8K block. Once again, the corresponding position in the basic 8K

block is not affected because the write currents in this block are out-of-phase.

Notice in Figure 2-79 that the X1-driver is turned on for each read cycle while the X2-driver is turned on for each write cycle. The Y1-driver is turned on for a read cycle in the basic 8K or for a write cycle in the second 8K. The Y2-driver is turned on for a write cycle in the basic 8K or for a read cycle in the second 8K. The desired 8K block is selected by using the M-Register 2-bit position in combination with the function desired (read, write) to condition the proper Y-driver. Absence of an M-Register 2-bit indicates an address in the range 00000 to 08191, causes Y1 to turn on for a read cycle or Y2 to turn on for a write cycle. An M-Register 2-bit indicates an address in the range 08192 to 16383 and causes Y2 to turn on for a read cycle or Y1 to turn on for a write cycle.

AUXILIARY STORAGE FOR 16K

- Four 256-byte auxiliary storage areas included in a 16K storage unit.
- M-Register 2-bit and 3-bit determine auxiliary storage area to be addressed.
- N-Register determines specific address from 000-255.
- Local storage is in second 8K storage unit.

Included in the 16,384 position storage unit are 1024 additional byte positions of auxiliary storage. These are divided into four 256-position areas called MPX 0, MPX 1, MPX 2, and local storage. When the CPU wishes to address one of these auxiliary storage areas, the main-auxiliary latch in the CPU is set to auxiliary, and the desired address is placed in the N-Register. The CPU further specifies which area of auxiliary storage is to be addressed by coding the M-Register 2-and 3-bits as follows:

M-Reg 2-bit	M-Reg 3-bit	Auxiliary Storage Area Selected
0	0	MPX 0
0	1	MPX 1
1	0	MPX 2
1	1	Local Storage

For example, if the CPU wishes to address a byte of information in the local storage area of auxiliary storage, the desired byte-address would be placed into the N-Register. The M-Register 2- and 3-bits would both be set to one. All other M-Register bits would be set to zero.

The auxiliary storage unit for 16K storage has the four auxiliary drivers: two read drivers and two write drivers. This is exactly the same as the 8K auxiliary storage arrangement. However, now the drivers must drive the lines through two 8K storage units. This means that the auxiliary drivers must be controlled by the M-Register 3-bit, the M-Register 2-bit, and the functions read and write, because of the phase reversal between the two 8K blocks of storage. The need for this selection can be seen on Figure 2-79.

16K STORAGE SUMMARY

Just as we did when we finished the 8K storage unit, let's review the quantities of drivers, gates, etc., in the 16K storage unit.

For a 16K storage unit, there are:

64X drive lines
64X read gate transistor
64X write gate transistor
16X gate decode switches
4X read drivers
4X write drivers

128Y drive lines
128Y read gate transistors
128Y write gate transistor
16Y gate decode switches
8Y read drivers
8Y write drivers

8Y auxiliary storage drive lines
2 auxiliary storage read drivers
2 auxiliary storage write drivers

Notice that the quantities are all the same as those quantities given for the 8K summary. This illustrates why the phase reversal scheme is used: double the size of storage unit can be addressed with the same drive scheme. The only quantity changed was the number of core planes, and this of course, doubled.

PHASE REVERSAL ADDRESSING (32K)

- There are four 8K blocks of core storage.
- Phase reversal occurs between the basic 8K and the second 8K, between the third 8K and the fourth 8K.
- Common Y-drive lines go through all four 8K blocks.
- Two sets of X-drive lines: one set for basic and second 8K addressing, one set for third and fourth 8K addressing.
- M-Register 2- and 1-bits control drivers.

A 32K core-storage unit is formed by tying two 16K units together in such a way that the Y-selection and drive circuitry is shared (Figure 2-80). Additional X-drivers and X-selection circuitry is required. Thus, there are two sets of Y-drivers (read and write), and four sets of X-drivers (read and write for the first 16K, and read and write for the second 16K).

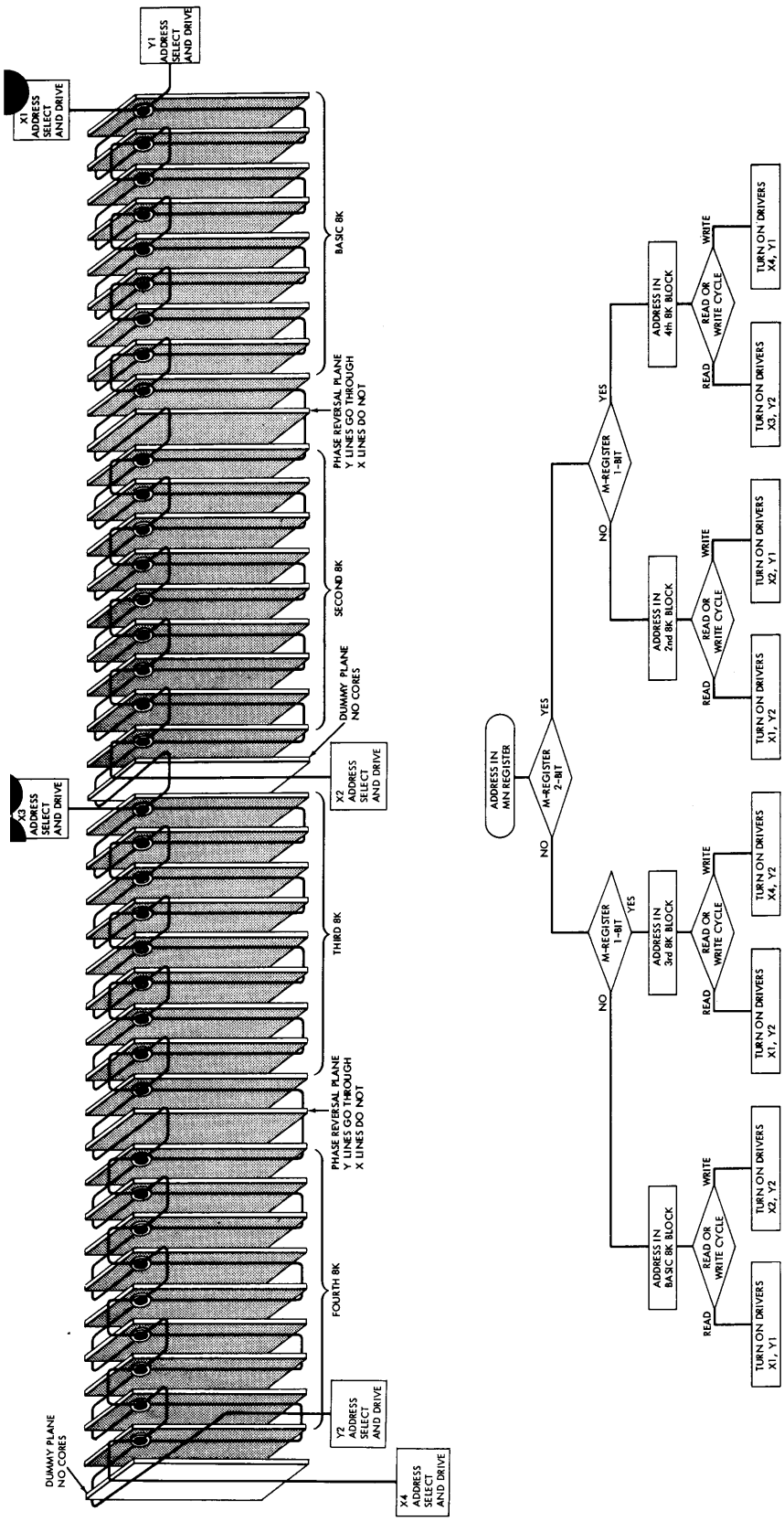


Figure 2-80. 32K Phase Reversal

Address selection above 8K is provided by the M-Register 2- and 1-bit positions. These two bit positions allow unique selection of one of the four 8K blocks. Absence of both M-Register 2- and 1-bits indicates an address in the range of 0000-1FFF, and therefore selects the basic 8K block. An M-Register 2-bit with no M-Register 1-bit specifies an address in the range of 2FFF-3FFF. This selects the second 8K block. The third 8K block has the address range of 4FFF-5FFF, and is selected by an M-Register 1-bit with no M-Register 2-bit. If the address contains both M-Register 2- and 1-bits, the fourth 8K block with the addresses 6FFF-7FFF is selected. The drivers are controlled by the M-Register 2- and 1-bits and the read or write signal.

AUXILIARY STORAGE FOR 32K

- Either four or eight 256-byte auxiliary storage areas included in a 32K storage unit.
- M-Register 1-, 2-, and 3-bits determine the auxiliary storage area to be addressed.
- N-Register determines the specific address from 00-255 within the auxiliary storage area defined.
- CPU local storage is always the high-order 256-byte auxiliary storage area.

Standard auxiliary storage for 32K storage unit is four 256-byte areas (MPX 0, MPS 1, MPX 2, and local storage). These four areas are located in the first 16K of storage, and are addressed as described under Auxiliary Storage 16K. A feature is available that provides four additional 256-byte blocks of auxiliary storage. These additional blocks of auxiliary storage provide additional subchannels for the multiplexor channel.

With the four additional blocks, auxiliary storage is composed of eight 256-byte blocks of auxiliary storage named MPX 0, MPX 1, MPX 2, MPX 3, MPX 4, MPX 5, MPX 6, and local storage. When the CPU wishes to address a specific byte-location in one of these blocks of auxiliary storage, the main-auxiliary latch in the CPU is set to auxiliary, and the desired byte-address is placed into the N-Register. The CPU further specifies which block of auxiliary storage is to be addressed by coding the M-Register 1-, 2-, and 3-bits as follows:

M-Reg 1-bit	M-Reg 2-bit	M-Reg 3-bit	Auxiliary Storage Area Selected
0	0	0	MPX 0
0	0	1	MPX 1
0	1	0	MPX 2
0	1	1	MPX 3
1	0	0	MPX 4
1	0	1	MPX 5
1	1	0	MPX 6
1	1	1	Local Storage

CLOCK CONTROL ADDRESSING (64K)

- A 2030 with 64K of core storage has two separate 32K core storage units.
- Each 32K unit is completely independent of the other.
- The 0-bit position of the M-Register determines which storage clock is started.

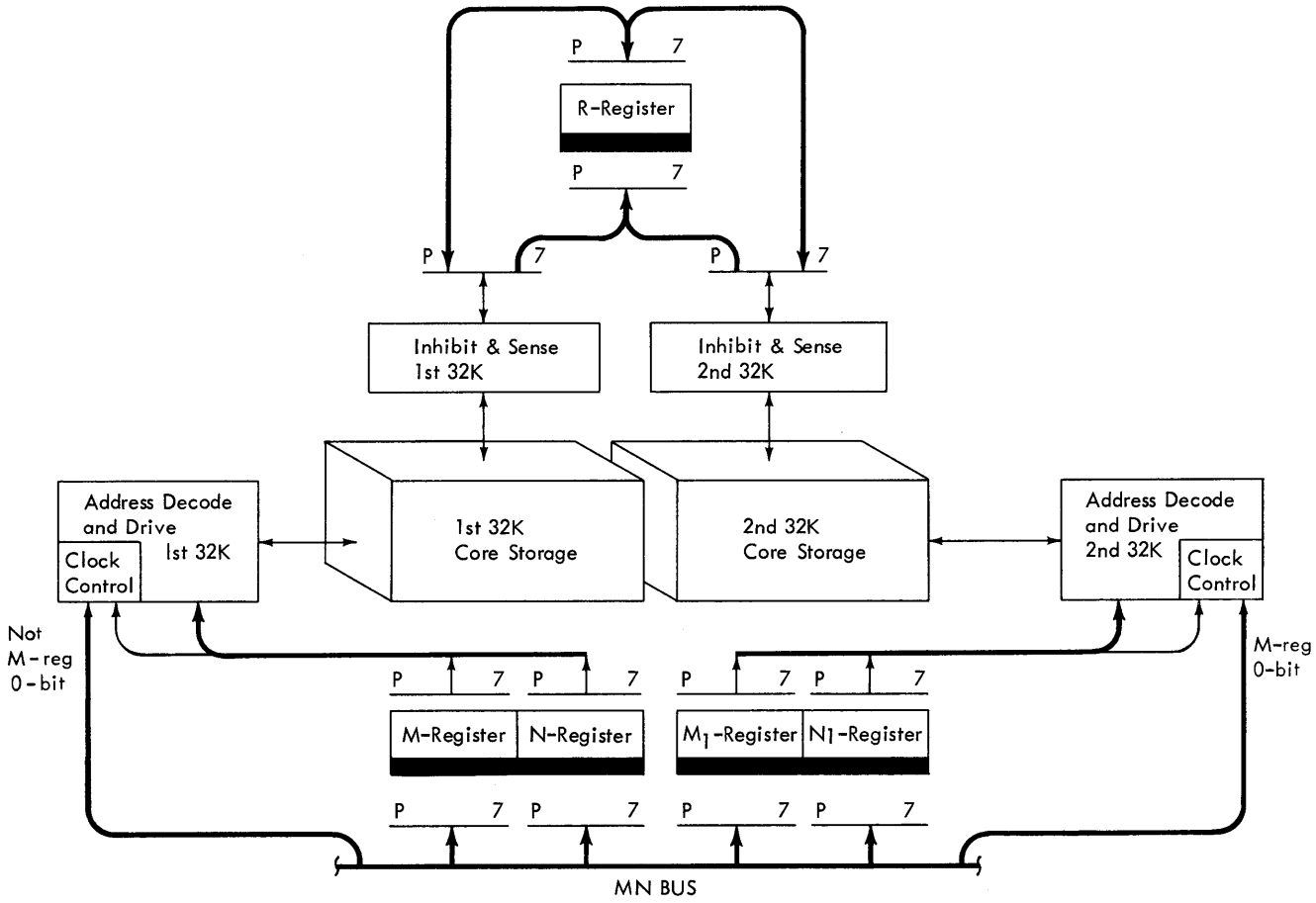


Figure 2-81. Clock Control Addressing

Each 32K core-storage unit is a complete package that cannot be further expanded by merely adding more planes to the existing array. The package includes the core planes, the addressing circuitry, and the storage clock for 32,768 positions of core storage. To expand core-storage capacity requires the addition of a similar 32K unit, complete with core planes, addressing circuitry, and storage clock. In addition, the CPU provides a second MN-Register for second 32K storage unit (Figure 2-81).

When the CPU requires information from the 64K storage unit, a 2-byte address is placed into both M- and N-Registers. The gates and drivers are conditioned in both 32K storage units. However, the storage clock is started in only one of the 32K units. If there is no bit in the high-order position of the M-Register, the storage clock in the first 32K unit is started and the desired cycle is completed. This cycle may be either a read or a write cycle. A bit in the high-order position of the M-Register indicates an address above the range 0000-7FFF, and starts the clock in the second 32K storage unit.

MEMORY/CPU INTERFACE

Each 32K storage unit communicates with the CPU over an interface. All addresses, data, and control signals are transferred over this interface. A brief description of the interface signals follows.

M- and N-Register Bit Lines

Sixteen bit-lines carry the address in the M- and N-Registers to the core-storage addressing circuitry. The address is set into the M- and N-Registers at T1 time of the CPU clock cycle following the cycle when a CPU read is decoded by the control circuitry. The address does not change until the necessary CPU compute and core-storage write cycles have been taken. The M- and N-Register bit lines in order from the high-order position of the address to the low-order position of the address are:

M-Reg 0
M-Reg 1
M-Reg 2
M-Reg 3
M-Reg 4
M-Reg 5
M-Reg 6
M-Reg 7
N-Reg 0
N-Reg 1
N-Reg 2
N-Reg 3
N-Reg 4
N-Reg 5
N-Reg 6
N-Reg 7

Store Bit-Lines

The nine store bit-lines provide the data input to the core-storage unit. These lines are direct outputs of the R-Register, and they go to the core storage inhibit drivers. The nine store bit-lines are:

Store Parity Bit
Store 0 Bit
Store 1 Bit
Store 2 Bit
Store 3 Bit
Store 4 Bit
Store 5 Bit
Store 6 Bit
Store 7 Bit

Mem Sense Bit-Lines

These nine lines represent the core-storage data output. They are active at memory strobe time. If the data on the memory sense bit-lines is to be used by the CPU, the memory clock data ready pulse is allowed to set the appropriate R-Register latches from the data on the sense lines. The nine sense bit-lines present at the R-Register input are:

Mem Sense Par Bit
Mem Sense 0-bit
Mem Sense 1-bit
Mem Sense 2-bit
Mem Sense 3-bit
Mem Sense 4-bit
Mem Sense 5-bit
Mem Sense 6-bit
Mem Sense 7-bit

M Bus 0 Bit

The memory clock must be started at the beginning of T1 time so the CPU and memory stay in step. Selection of the first 32K clock or second 32K clock is dependent on the high-order position of the M-Register (M-Reg 0-bit). However, the M- and N-Register set pulse is at T1 time, and it takes approximately 50 nanoseconds to set the M- and N-Register latches. This would not allow the M-Register 0-bit to start either of the two memory clocks at zero time in the CPU clock cycle. The M bus 0 bit occurs before the M-Register has set, and is actually before zero time in the CPU clock cycle. If the M bus 0 bit signal is present at the clock control circuitry at zero time of the CPU clock cycle, the CPU read-call signal starts the second 32K clock. If the M bus 0 bit signal is not present when the CPU read-call signal arrives, the first 32K clock is started.

The M bus 0 bit signal is called early M0 in the memory circuits. It is not brought up for a write cycle because the M- and N-Registers are not changed for a memory write cycle. For a write cycle, the M-Register 0-bit line (output of M-Register) switches with the CPU write-call signal to control the two clocks.

Early Local Storage

The function of this signal is similar to that of the M-bus 0 bit signal: control of the two memory clocks. Early local storage occurs before zero time in the CPU clock cycle to signal the memory that the next access to memory will be in the first 32K. When read call occurs, the first 32K clocks starts.

Read Call

Read-call signals the memory that the CPU control circuitry has decoded a read operation. The read-call pulse occurs at T1-time of the CPU clock cycle and it is used to start the memory clock. Read-call specifies a memory-read cycle by setting up the memory clock for a read operation (Figure 2-77).

Write Call

Write-call occurs at T1 time of the next cycle after the CPU control circuitry has decoded a write operation. Write-call starts the memory clock, and specifies a memory-write cycle by setting up the memory clock for a write operation (Figure 2-77).

Data Ready

Data-ready is the memory data strobe pulse to the CPU. The data-ready signal sets the data (memory sense bit lines) into the R-Register, provided the CPU has specified memory as the source for the R-Register.

Mach Reset Sw

Machine-reset-switch is a signal from the CPU that causes all memory control latches to be reset to a starting condition when the machine reset function is being performed. The machine-reset function can occur for several reasons, such as when power on sequencing is complete or when the system reset key on the 2030 console is pressed.

Read Echo

This signal is required by the CPU for manual store operations. It signals the CPU that a memory-read cycle is taking place, and allows a write cycle to follow.

Write Echo

This signal is required by the CPU for manual store operations. It signals the CPU that a memory-write cycle is taking place, and allows a read cycle to follow.

STORAGE READ EXAMPLE

- Storage drive lines are selected by the M- and N-Register bits.
- The appropriate clock is selected by the high-order bit position of the M-Register, and is started by read-call from CPU.
- Resultant data byte is placed into the CPU R-Register.

	M-REGISTER								N-REGISTER							
Position	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
Binary Value	3 2 7 6 8	1 6 3 8 4	8 1 9 2	4 0 9 6	2 0 4 8	1 0 2 4	5 1 2	2 5 6	1 2 8	6 4	3 2	1 6	8	4	2	1
Purpose	Clock Control	Gate Terminator and Phase Reversal Control		Y-Driver Decode			Y-Gate Decode				X-Driver Decode		X-Gate Decode			
Sample Binary Address	0	1	0	0	1	1	0	1	1	0	0	0	1	0	1	1

4D8B

Figure 2-82. MN Decode

To read out a byte of data, the core-storage unit must interpret the bits in the CPU M- and N-Registers and select the appropriate lines so the desired position can be read out. To read out this position, an X-line must be driven, a Y-line must be driven, and the resultant data on the sense lines must be amplified and set into a data register. Selecting the correct lines is the result of decoding various bit groupings in the M- and N-Registers (Figure 2-82). To follow a storage read example through the core-storage circuitry, assume the address 4D8B is in the M- and N-Registers.

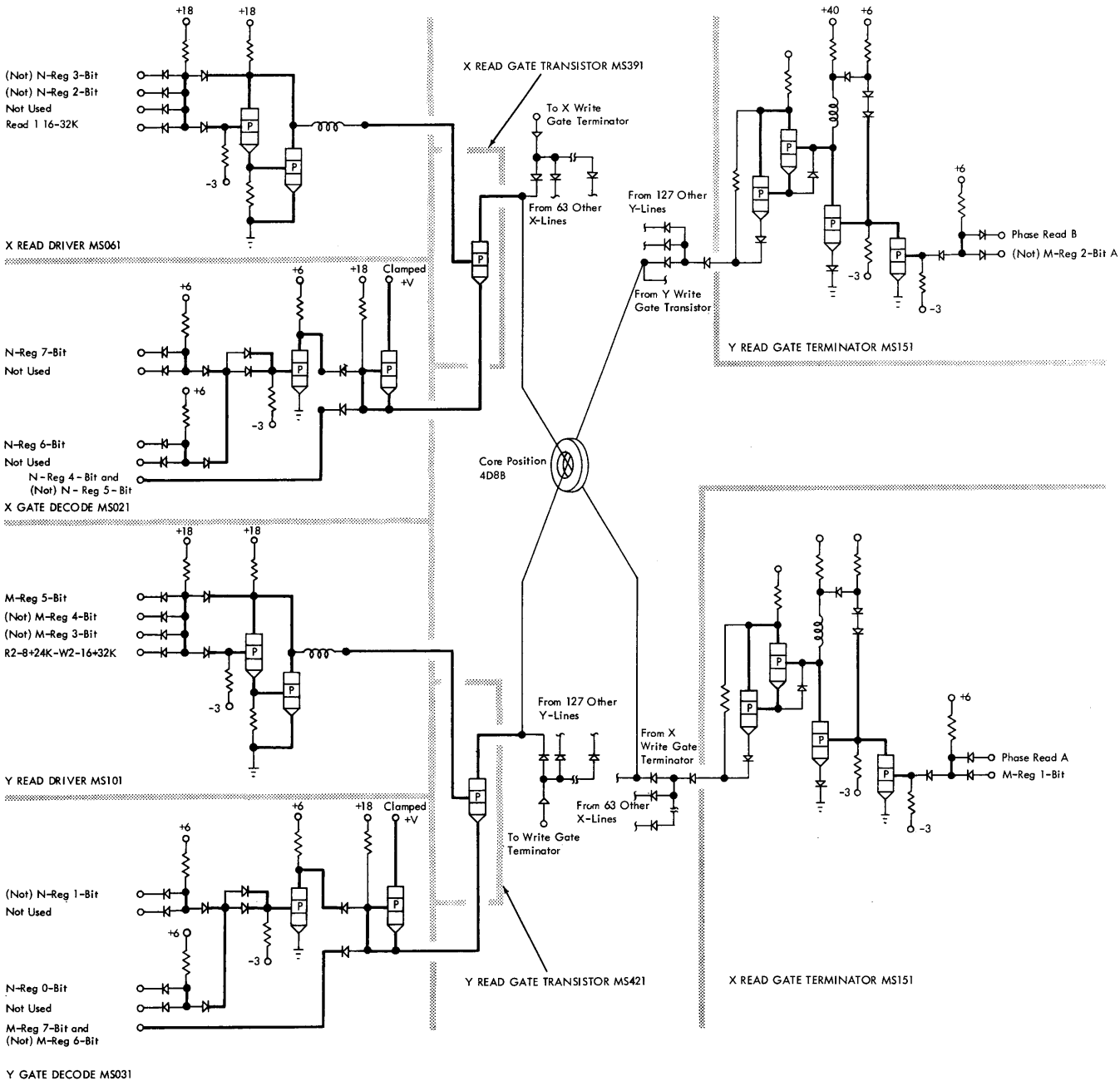


Figure 2-83. Storage Read Example

Circuit Objectives (Figure 2-83)

1. Start the clock for the first 32K storage unit MS321).
   Start 1st 32K clock
   (Not) Early M0 (M-Register 0-bit)
   Read Call

2. Define area of storage to be addressed (MS321).
   Use Main Mem
   Read Call
   (Not) Early Local Storage

3. Decode and drive an X-line.

   a. Read 1 16-32K. This read timing pulse from the clock conditions the proper X-drivers as required by the phase reversal addressing scheme (MS161).
      Read 1 (from clock)
      M-Reg 1-bit controlled

b. 11 Gate TX 16-32K. This is the X-gate decode (MS021).
   N-Reg 7-Bit
   N-Reg 6-Bit
   (Not) N-Reg 5-Bit
   N-Reg 4-Bit

c. Rd 1 0-15 16-32K. This is the X-driver decode (MS061).
   Read 1 16-32K
   (Not) N-Reg 2-Bit
   (Not) N-Reg 3-Bit

d. Ary Side C 15X Ln 11 A1. This is one end of the X-drive line (MS391).
   11 Gate TX 16-32K (conditions emitter of read gate transistor)
   Rd 1 0-15 16-32K (conditions base of read gate transistor)

e. Read 16-32K. This line provides a current path at the other end of the X-line. This requires that the X read-gate terminator be turned on (MS151).
   M-Reg 1-Bit Controlled
   Phase Read A (from clock)
   X Gate Term Current Source (from power supply)

4. Decode and drive a Y-line.

a. R2-8+24K-W2-16+32K. This read timing pulse from the clock conditions the proper Y-drivers as required by the phase reversal addressing scheme (MS161).
   Not M-Reg 2-Bit A
   Read 2 (from clock)
   (Not) Use Local

b. R-8+24K-W16+32K-3072-4095. This is the Y-driver decode (MS101).
   M-Reg 5-Bit
   M-Reg 4-Bit
   (Not) M-Reg 3-Bit
   R2-8+24K-W2-16+32K

c. 384-447 Gate TX C1. This is the Y-gate decode (MS031).
   (Not) N-Reg 1-Bit
   N-Reg 0-Bit
   M-Reg 7-Bit
   (Not) M-Reg 6-Bit

d. Ary Side D 99 Y Ln 54 A1. This is one end of a Y-drive line (MS421).
   384-447 Gate TX A1 (conditions base of read gate transistor)
   R-8+24K-W-16+32K 3072 - 4095 (conditions emitter of read gate transistor)

e. R-8+24K-W-16+32K. This line provides a current path at the other end of the Y-drive line. It is the result of the Y read gate terminator being turned on (MS151).
   Y Gate Term Current Source (from power supply)
   Phase Read B (from clock)
   Not M-Reg 2-Bit A

5. Sense and amplify the resultant data byte. Each 8K block of storage has two sets of sense amplifiers (one set for each 4,096 positions). These amplifiers are active all the time. Thus, in a 32K storage unit, there would be eight sets of sense amplifiers (two for each 8K block). To help distinguish between the byte of information from the addressed 4K block and noise from the other 4K blocks, a strobe pulse is developed for each 4K block. The M-Register 2- and 1-bit positions, and the N-Register 2-bit position determine which 4K block is strobed. In the example given, there is an M-Register 1-bit, no M-Register 2-bit, and no N-Register 2-bit, indicating the desired byte is in the first half of the third 8K block. The output of this 4K block is gated to the final amplifier by the strobe pulse that is developed. For simplicity, only the 6-bit position is shown. The other eight bit positions operate similarly.

a. Strobe 6-Bit 16-24K A (MS181).
   Strobe (from clock)
   Not M-Reg 2-Bit A
   M-Reg 1-Bit Controlled
   N-Reg 2 Bit A

b. SA 6 Bit 16-24K A. This is one half (4,096 positions) of the sense circuitry for the third 8K block (MS231).
   SA - Inh Line 6-Bit A1
   SA - Inh Line 6-Bit A2
   Strobe 6-Bit 16-24K A

c. Mem Sense 6-Bit 16-32K. (Final amplifier output MS281).
   SA 6-Bit 16-24K A

STORAGE WRITE EXAMPLE

- X- and Y-drive lines are selected by M- and N-Register bits.
- The clock is selected by the high-order bit position of the M-Register and is started by write call from the CPU.
- Data from the R-Register activates appropriate store cir-

cuits, allowing X- and Y-lines to write the data into the addressed position.

To write a byte of data into a core-storage position, the CPU signals the core-storage unit with a write call. The address in the M- and N-Registers does not change between read and write cycles. Therefore, we can assume the same address (4D8B) is in the M- and N-Registers in binary form. To write data into this position, an X-line must be driven, a Y-line must be driven, and the appropriate inhibit lines must be driven to cause the desired bit configuration to be set into the position addressed by the X- and Y-lines.

Circuit Objectives (Figure 2-84)

1. Start the clock for the first 32K storage unit (MS321). Start 1st 32K clock
   (Not) M-Reg 0-Bit
   Write Call

2. Define the area of storage to be addressed (MS321). Use Main Mem (This latch was set on during the previous read cycle and stays on until local storage is addressed on a read cycle).

3. Decode and drive an X-line

   a. Write 1 16-32K. This write timing pulse from the clock conditions the proper X drivers as required by the phase reversal addressing scheme (MS161).
      Write 1 (from clock)
      M-Reg 1-bit controlled

   b. 11 Gate TX 16-32K. This is the X Gate decode (MS021).
      N-Reg 6-bit
      N-Reg 7-bit
      (Not) N-Reg 5-Bit
      N-Reg 4-Bit

   c. WR1 0-15 16-32K. This is the X driver decode (MS061).
      Write 1 16-32K
      (Not) N-Reg 2-Bit
      (Not) N-Reg 3-Bit

   d. Ary Side A 15 X Ln 11 A1. This is one end of the X-drive line (MS381).
      11 Gate TX 16-32K
      WR1 0-15 16-32K

   e. Write 16-32K. This line provides a current path at the other end of the X-line. This requires that the X write-gate terminator be turned on (MS151).
      Phase Write A (from clock)
      M-Reg 1-Bit Controlled
      X Gate Term Current Source (from power supply)

4. Decode and drive a Y-line.

   a. R2-16+32K-W2-8+24KA. This write timing pulse from the clock conditions the proper Y-drivers as required by the phase reversal addressing scheme (MS161).
      Not M-Reg 2-Bit A
      Write 2 (from clock)
      (Not) Use Local

   b. R-16+32K-W-8+24K-3072-4095. This is the Y-driver decode (MS091).
      (not) M-Reg 3-Bit
      M-Reg 4-Bit
      M-Reg 5-Bit
      R2-16+32K-W2-8+24K A

   c. 384-447 Gate TX C1. This is the Y-gate decode (MS031).
      (not) N-Reg 1-Bit
      N-Reg 0-Bit
      M-Reg 7-Bit
      (not) M-Reg 6-bit

   d. Ary Side D 99 Y Ln 54 C1. This is one end of a Y-drive line (MS401).
      384-447 Gate TX C1 (conditions base of write gate transistor)
      R-16+32K-W-8+24K-3072-4095 (conditions emitter of write gate transistor)

   e. R-16+32K-W-8+24K. This line provides a current path at the other end of the Y-drive line. It is the result of the Y read gate terminator being turned on (MS151).
      Y Gate Term Current Source (from power supply)
      Phase Write A (from clock)
      (not) M-Reg 2-Bit

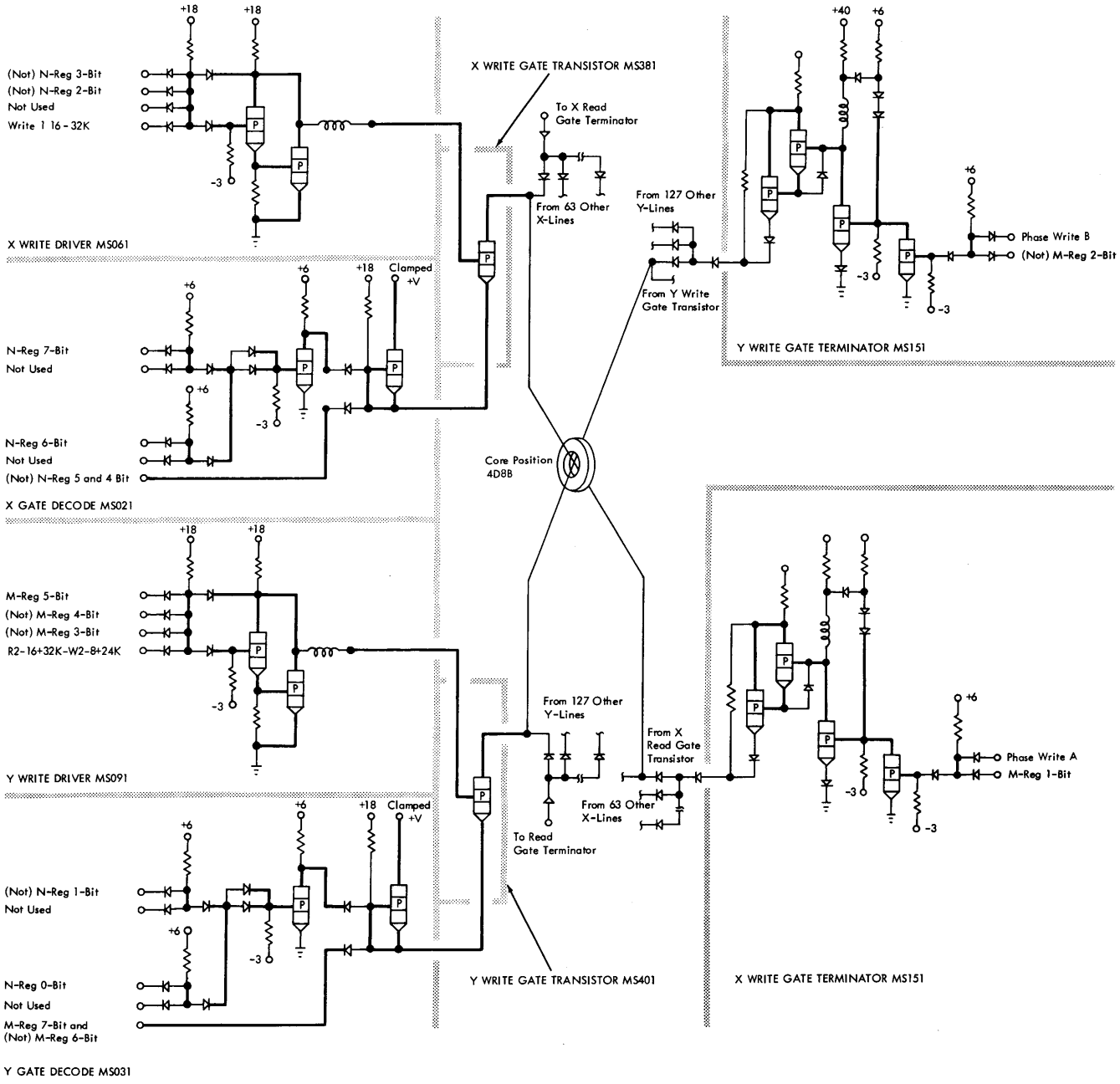


Figure 2-84. Storage Write Example

5. Activate the appropriate inhibit drivers. For each core plane in an 8K unit there are two sense/inhibit windings. Thus, there are 18 sense/inhibit windings in an 8K block of storage. To supply inhibit current, there is one set of nine inhibit current drivers for the one half of the 8K block, and one set of nine inhibit current drivers for the other half of the 8K block (Figure 2-85). Only one of these sets of nine inhibit drivers is allowed to be active during any storage write cycle. This means that a set of nine inhibit drivers must be selected as a part of the address decode. Thus, to store a properly coded byte of information in an addressed position, the proper set of nine inhibit drivers must be conditioned to turn on. Then the bit coding of the byte to be stored causes the correct inhibit drivers of that set to be turned on at inhibit time. For each bit position of the byte to be stored, presence of a bit at the inhibit driver input prevents that inhibit driver from

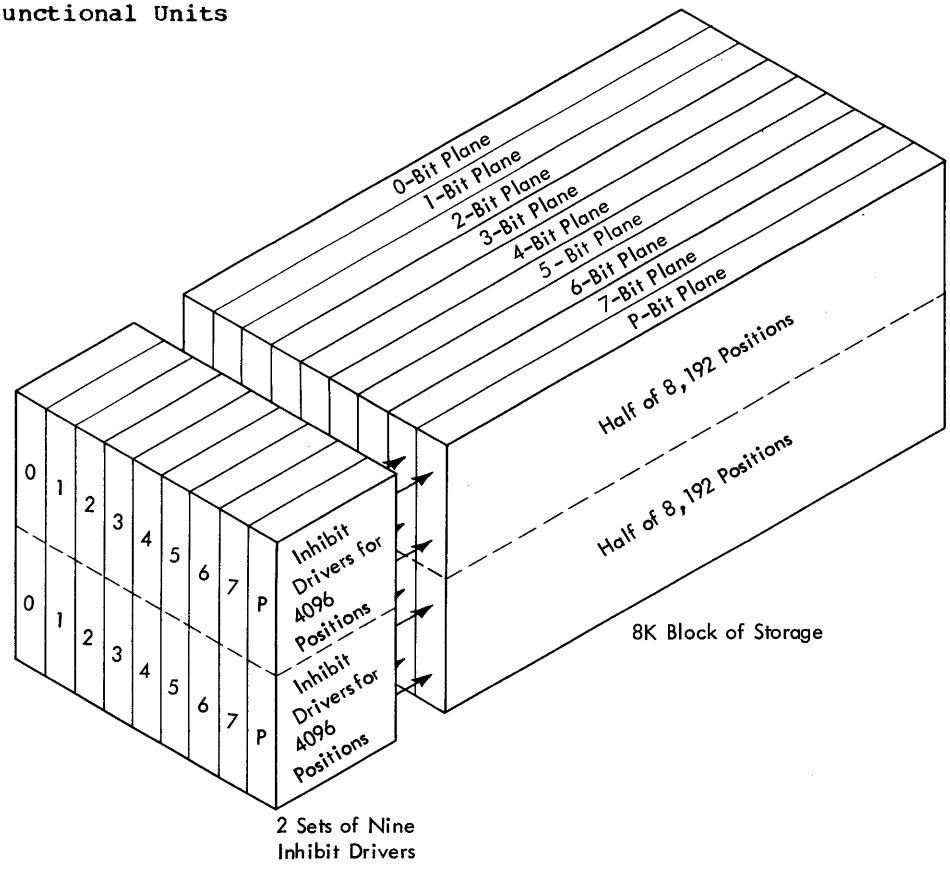


Figure 2-85. Inhibit Driver Control

turning on. Conversely, for each bit position of the byte to be stored, absence of a bit allows the inhibit driver to turn on. As a result, there is no inhibit current flowing where a bit is to be stored, and inhibit current flows where no bit is to be stored.

a. To effect inhibit driver selection, the Inhibit timing pulse is switched with the N-Register 2-bit to produce Inhibit A or Inhibit B pulse. These pulses determine which set of inhibit drivers will be allowed to turn on. In this case, there is no N-Register 2-bit. As a result, the Inhibit A pulse turns on (MS321).
   Inhibit (from clock)
   (not) N-Reg 2 Bit

b. To complete inhibit driver selection, the M-Register 2- and 1-bits define in which 8K block inhibit current is to flow.

   Inhibit 16-24K A. This line identifies the address as falling within one group of 4,096 positions of the third 8K block of storage (MS161).
   Not M-Reg 2-Bit A
   M-Reg 1-Bit Controlled
   Inhibit A

c. SA-Inh Line 6-Bit A1
   SA-Inh Line 6-Bit Aa
   These are the two ends of the inhibit winding for the desired 4,096 positions of the 6-bit plane of the third 8K block of storage. For simplicity, only the 6-bit is shown. Nine inhibit drivers are involved to set a byte of data into a storage position. Notice that the inhibit driver is conditioned and that inhibit current is made to flow because there is no store 6-bit input. Thus, a bit input prevents inhibit current which allows the X- and Y-drive lines to set the core, whereas not-bit input enables the inhibit driver. When inhibit current flows, it opposes the effect of the X-drive current and the core is not set (MS231).
   Inhibit 16-24K A
   (not) Store 6-Bit

M2-I CORE STORAGE UNIT

- The M2-I storage unit is the 1.5 microsecond read-write storage unit for the IBM 2030.
- The M2-I is a separately packaged core storage unit in the 2030.
- A memory/CPU interface transfers all information between the M2-I and the 2030.

The M2-I memory provides the IBM 2030 Processing Unit with a 1.5 microsecond read-write cycle time. The basic unit of information in the 2030 is the eight-bit byte, with an additional bit added to maintain odd parity of data. Storage sizes are the same as for the 2.0 microsecond M2 memory offered on early 2030's. The 8K, 16K, 32K, and 65K refer to 8,192 bytes, 16,384 bytes, 32,768 bytes, and 65,536 bytes of storage respectively.

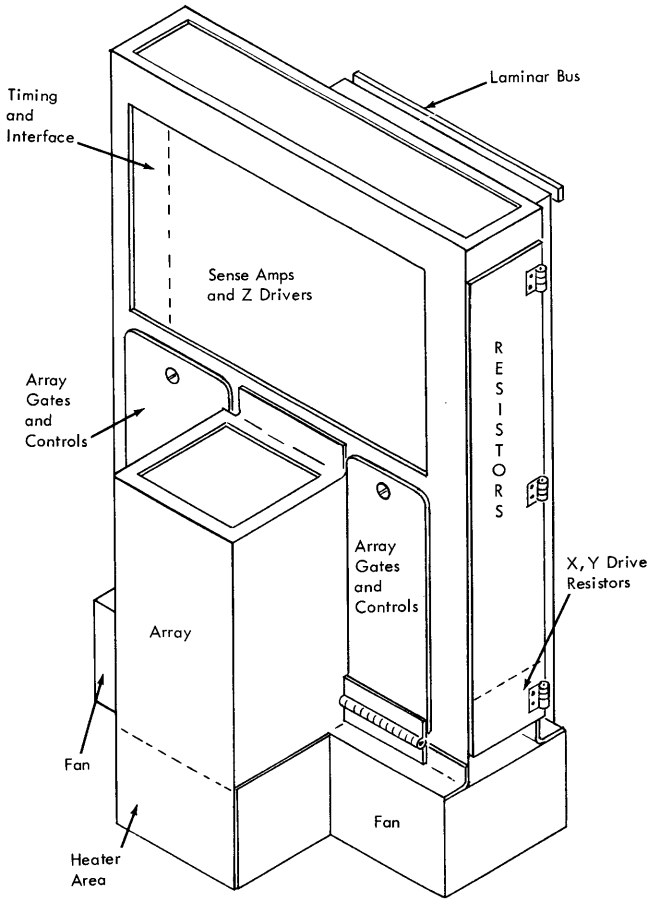


Figure 2-86. M2-I Storage Unit (8K, 16K, 32K)

Like the M2 memory, the M2-I is a separately packaged unit that is installed inside the 2030 Processing Unit. This separately packaged unit contains the controls, timing generator, core array, drive system, and sense/inhibit system. An M2-I may be 8K, 16K, or 32K in size (Figure 2-86). If the 2030 requires the full 65K of storage, two separate M2-I units are installed in the 2030.

Because the unit is entirely separate from the 2030, communication between the two takes place over a number of signal lines called the Memory/CPU interface. Essentially, this interface transfers address information, input data, output data, and timing signals. Basic data flow is as follows: the 2030 places a two-byte address into the M- and N-Registers (Figure 2-87).

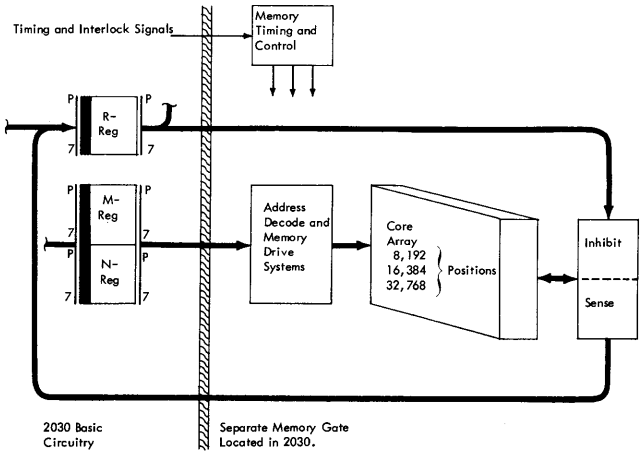


Figure 2-87. Memory to CPU Data Flow

At the appropriate time, the memory is signaled to begin a read cycle. The memory timing circuits begin a read cycle and the byte of information located at the address in the M- and N-Registers is read out and placed on the data lines to the 2030. The memory signals that the data is "ready". This allows the 2030 to set the information into the R data-register. When the read cycle is complete, the memory stops. The byte of data read out may be placed back into the addressed location, or different data may be placed into that storage position. The data to be stored is placed into the R-register. The 2030 then signals the memory to begin a write cycle. The memory timing circuit starts, and the same position is addressed again. This time, however, the information in the R-Register is placed into the addressed position.

CORE ARRAY

- The core array is composed of a number of core planes.
- Three wires go through each core.
- Horizontal drive lines are called X-lines.
- Vertical drive lines are called Y-lines.
- A common sense/inhibit winding is used.

The core storage array is composed of a number of core planes. Each core plane consists of a plastic-material frame approximately 1/8 inch thick and 6 1/2 inches square. The basic core plane contains 16,384 cores located at the intersection points of the 128 horizontal drive lines and the 128 vertical drive lines (Figure 2-88). The horizontal drive lines are called X-drive lines, whereas the vertical drive lines are called Y-drive lines. The M2-I uses a common sense/inhibit winding that is wound parallel to the X-line. Thus, three wires go through each core: one X-drive wire, one Y-drive wire, and one sense/inhibit wire.

While an X-drive or Y-drive wire go through 128 cores in a core plane, the sense/inhibit wire goes through 4,096 cores in a core plane. This means there are four sense/inhibit windings in each core plane.

The 8K core array is composed of five 16,384-core planes (Figure 2-89). The first plane forms bit-0 and bit-5, the second plane forms bit-1 and bit-6, the third plane forms bit-2 and bit-7, the fourth plane forms bit-3 and bit-P, and half of the fifth plane forms bit-4. The top half of the fifth plane is not used in the 8K array.

Each X-winding travels through five planes before crossing to the second half of the core planes via an X-return card at one end of the array. This X-return card contains printed lines that carry X-drive current from one X-winding in the lower half of the array to another X-winding in the upper half of the array. The winding pattern of the array is such that alternate X-drive lines are driven from opposite ends of the core plane (see Figures 2-88 and 2-89 for core positioning).

The Y-drive lines are positioned similarly. A Y-drive line starts at either the top or bottom of the first core plane and is wound through each plane (Figure 2-89). Thus, if drive current flows through one X-line and one Y-line, ten cores will experience the coincident drive current necessary to affect the cores. The tenth core, in the top half of the last plane, experiences coincident drive current. However, this core output is not sensed. Depending on the direction of drive current, the ten cores are read from or written into.

An 8K storage unit, such as shown in Figure 2-89, has eighteen sense/inhibit windings (two per bit). Each winding serves 4,096 cores. The sense/inhibit winding is parallel to the X-winding.

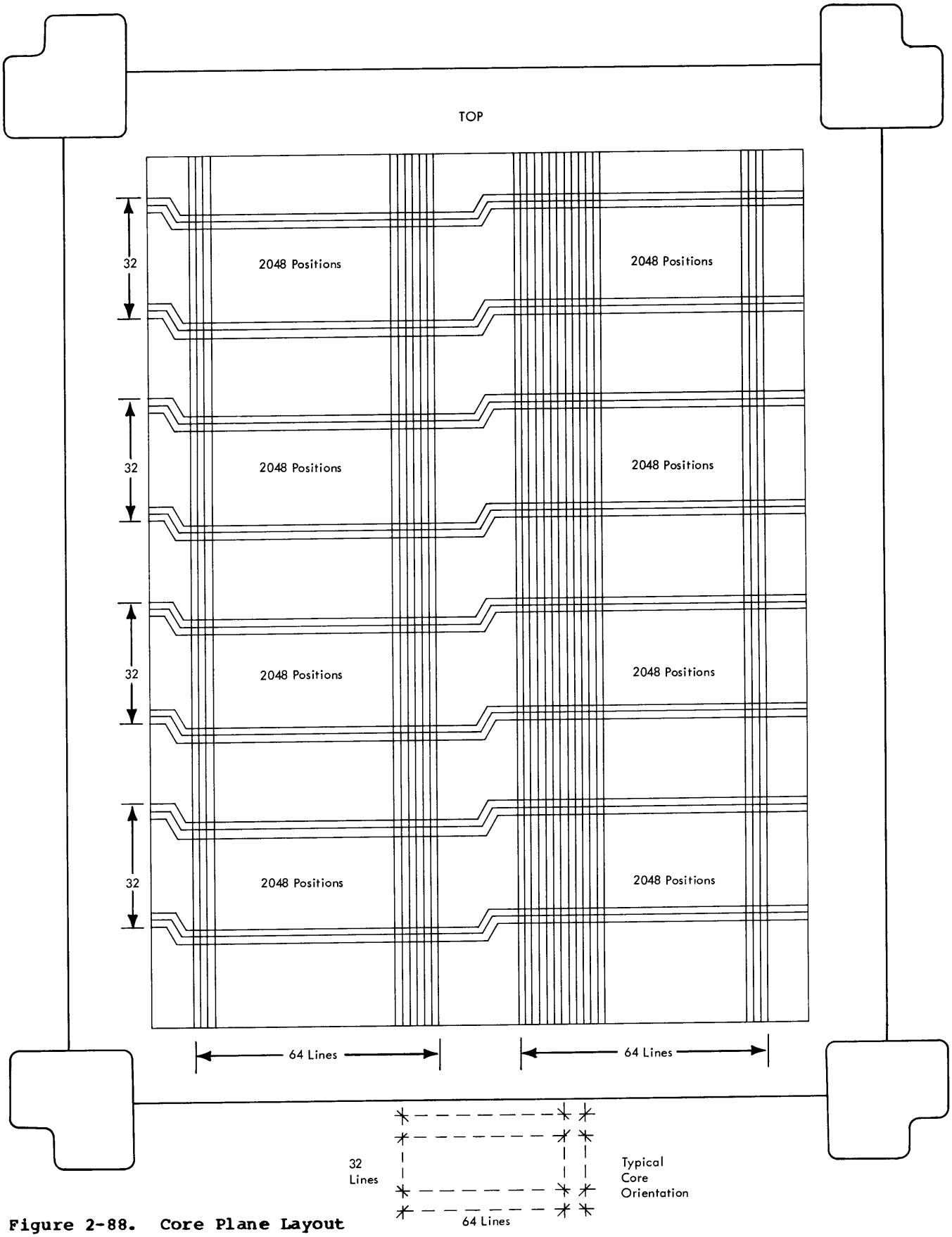


Figure 2-88. Core Plane Layout

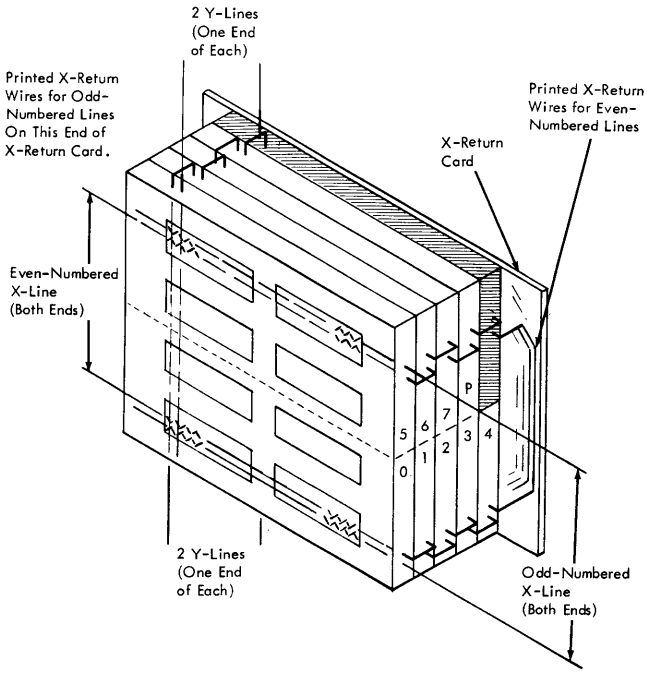


Figure 2-89. 8K Storage Winding

The 16K core array consists of nine 16,384-core planes. The first plane forms bit-0 in both first and second 8K. (Figure 2-90). The second core plane forms bit-1 in both first and second 8K. This same scheme continues through the ninth core plane which forms bit-P for both 8K. An X-winding travels through all nine planes of one 8K unit before crossing to the second 8K unit via an X-return card at the end of the array. The X-return card connects the X-winding to the second 8K unit. The X-winding then continues through all nine planes of the second 8K unit.

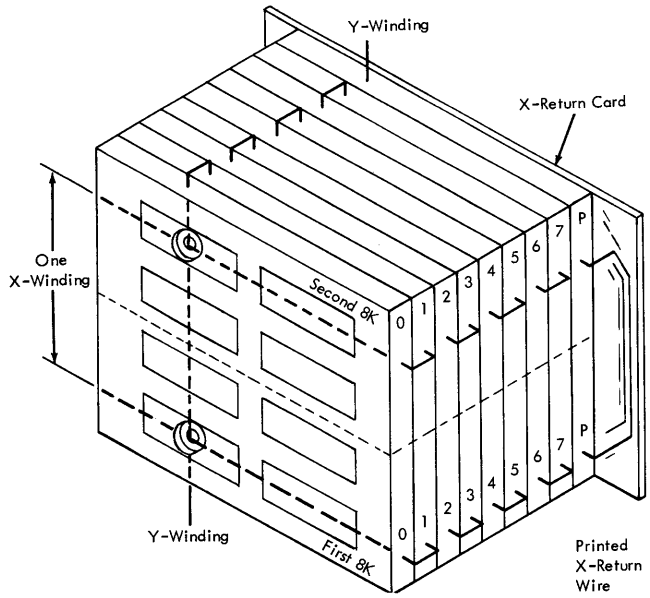


Figure 2-90. 16K Storage Winding

The Y-winding goes through both 8K units in each of the nine planes. The significant point is that the X-winding experiences a phase reversal when crossing from one 8K unit to the other. Because of this phase reversal, the X- and Y-currents are in-phase in only one 8K unit at a time.

Sense and inhibit consists of four windings per core plane for a total of 36 windings. Each winding passes through 4,096 cores in a plane.

The 32K array consists of eighteen 16,384-core planes (Figure 2-91). This array consists of two 16K arrays sharing a single set of Y-windings. The desired Y-winding is selected and driven. This Y-winding goes through all eighteen planes. The X-windings go through nine planes, starting at one end, passing through nine planes, crossing to the other half of the plane via a printed X-return wire, and returning through the other half of the same nine planes. To address a particular position, the appropriate Y-winding is selected and driven. This makes it possible to address four different core locations. The desired location is driven by selecting the appropriate set of X-windings (first 16K or second 16K) by selecting the appropriate X-winding within the selected set, and by driving the selected X-winding with current in the appropriate direction. Current direction control is necessary because of the phase reversal between 8K blocks of storage on the X-winding.

The 32K storage unit has 72 sense/inhibit windings (four windings per core plane). Each winding passes through 4,096 cores, parallel to the X-windings.

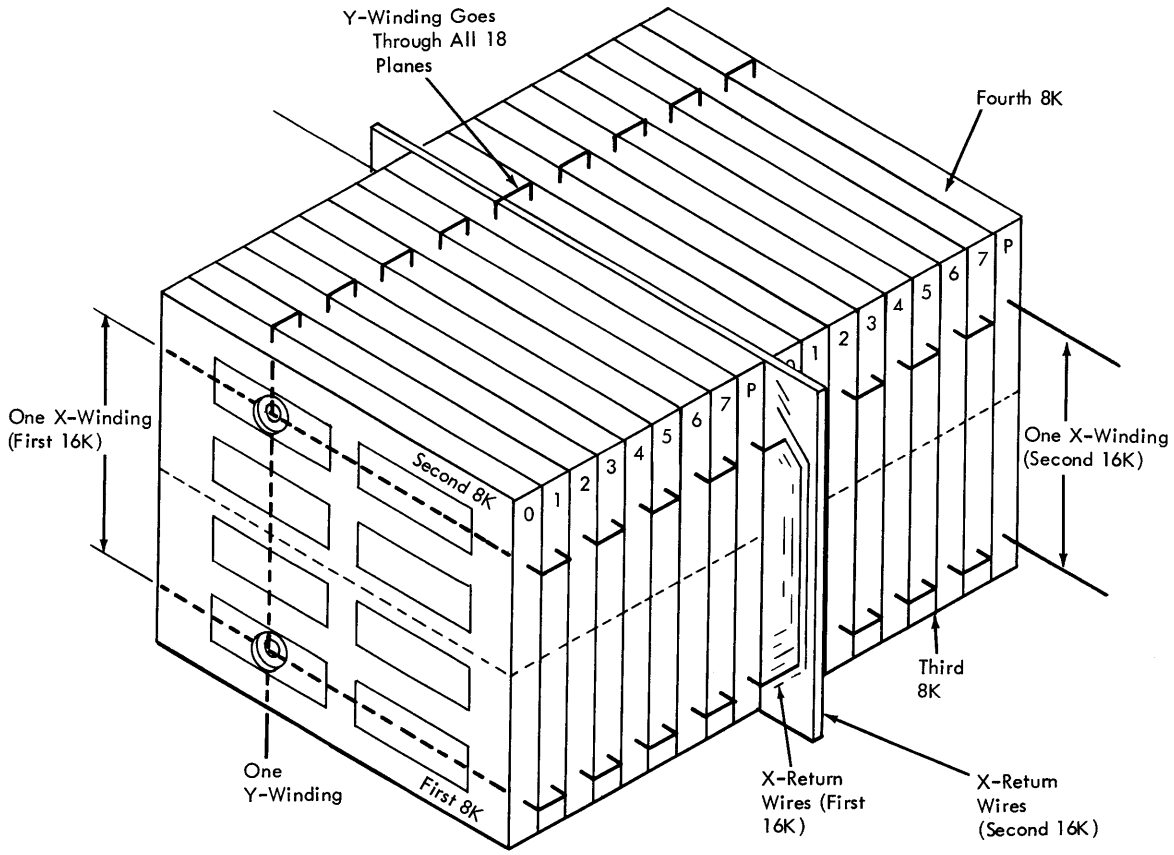


Figure 2-91. 32K Storage Winding

A 65K storage unit is actually two separate storage units. Each unit can store 32,768 bytes of information. Each of these 32K units is the same as described previously (Figure 2-91). The units are separate physically, and each mounts on a separate hinged gate in the 2030 (Figure 2-92).

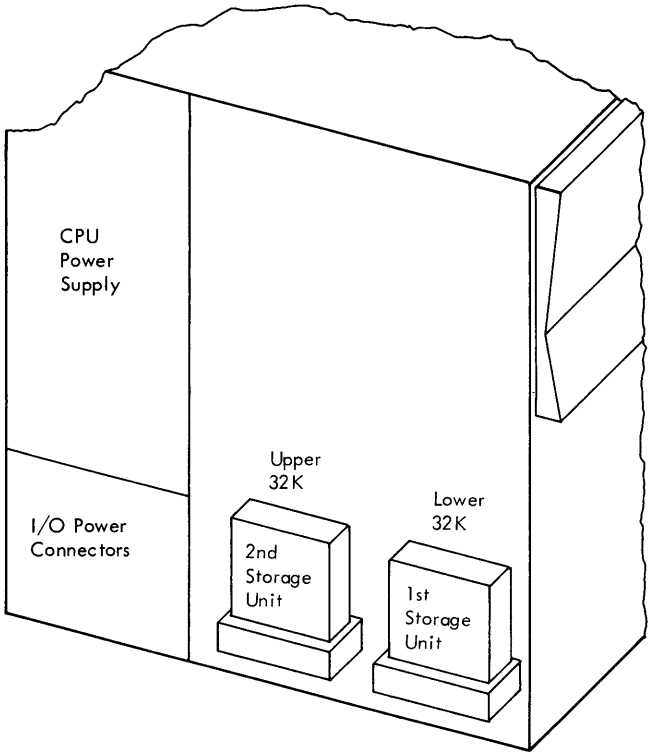


Figure 2-92. IBM 2030 Showing Two 32K Storage Units

STORAGE CLOCK

- The M2-I has a separate clock which allows it to operate independently from the 2030.
- The clock consists of delay lines and timing latches.
- The clock is started by either read call or write call from the 2030.

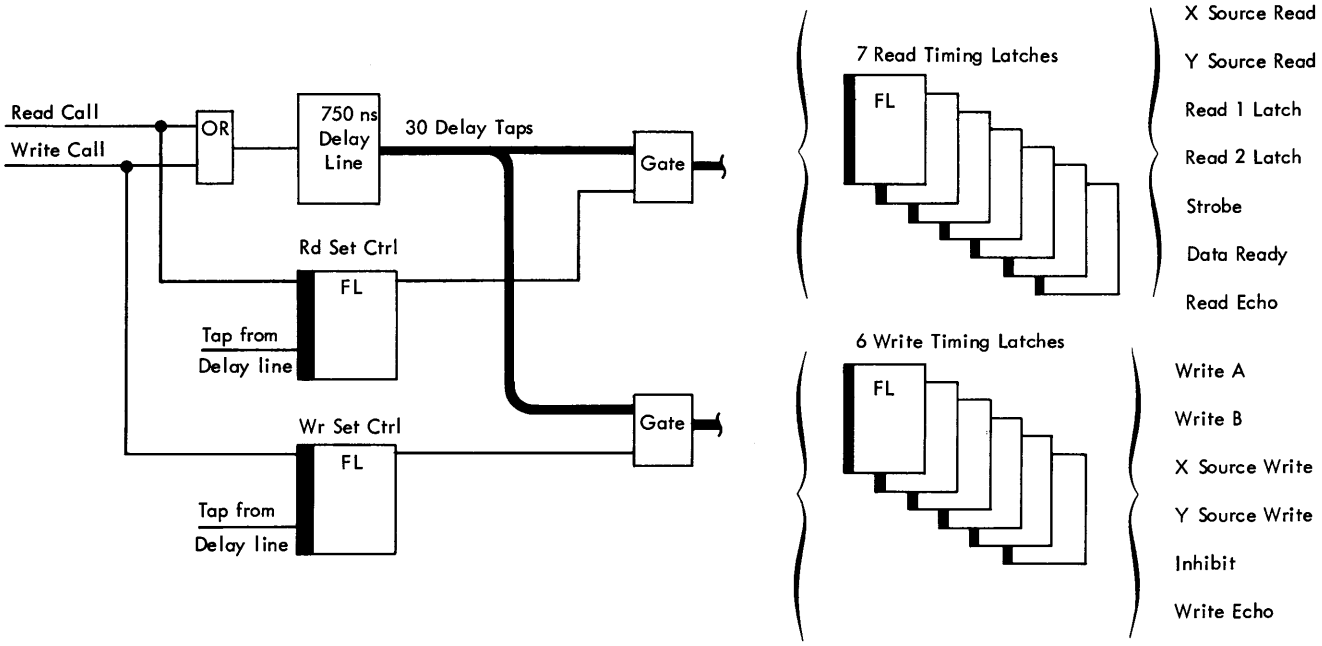


Figure 2-93. Simplified Clock Logic

The M2-I core storage unit contains a timing generator referred to as the storage clock. Having a clock spearate from the 2030 clock allows the storage unit to operate independently from the 2030 once the read call or write call signal starts a storage cycle.

The clock is composed of three 250 nanosecond delay lines tied together with appropriate controls to form a 750 nanosecond delay line. The delay line is tapped at 25 nanosecond intervals. These taps are wired to a series of latches to produce the composite timing signals required by the storage unit (Figure 2-93).

When a read call signal arrives from the 2030, the read set control latch is turned on, and the delay line is driven. The read set control latch allows the delay line tap outputs to reach the read clock latches in 750 nanoseconds--the same time as one basic 2030 machine cycle (Figure 2-94).

Write call from the 2030 turns on the write set control latch and drives the delay line. The write set control latch gates the delay line tap outputs to the write clock latches (Figure 2-95). A write cycle is completed in 750 nanoseconds--the same time required for one 2030 clock cycle.

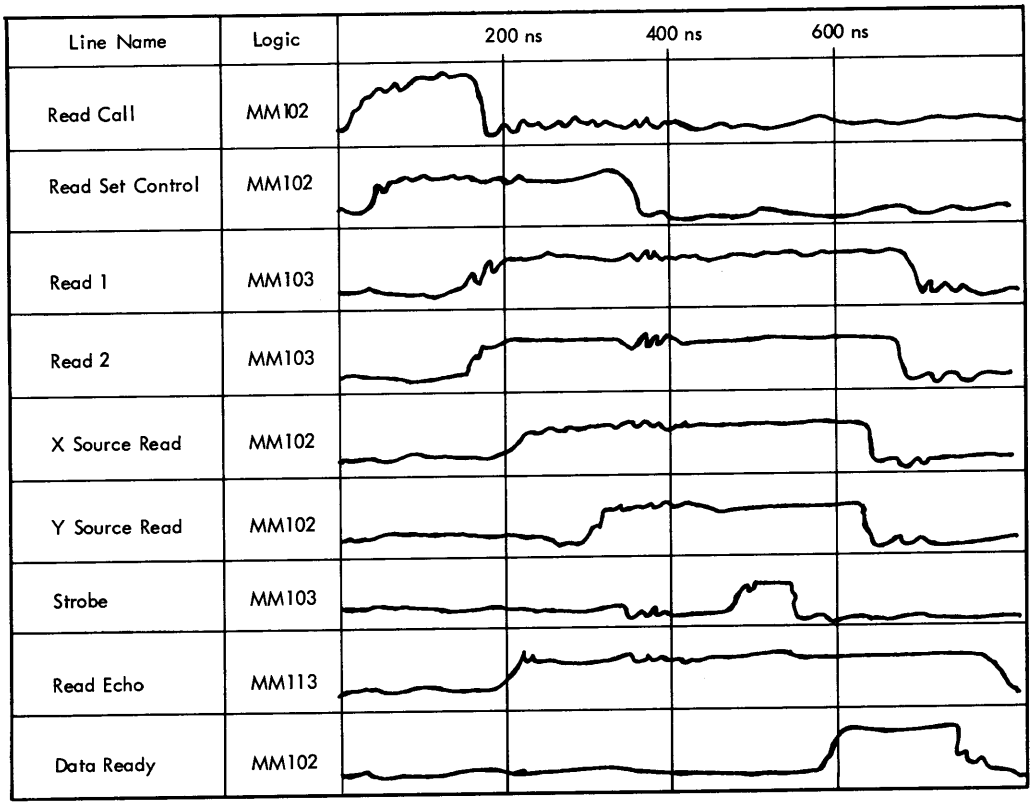


Figure 2-94. Core Storage Read timings

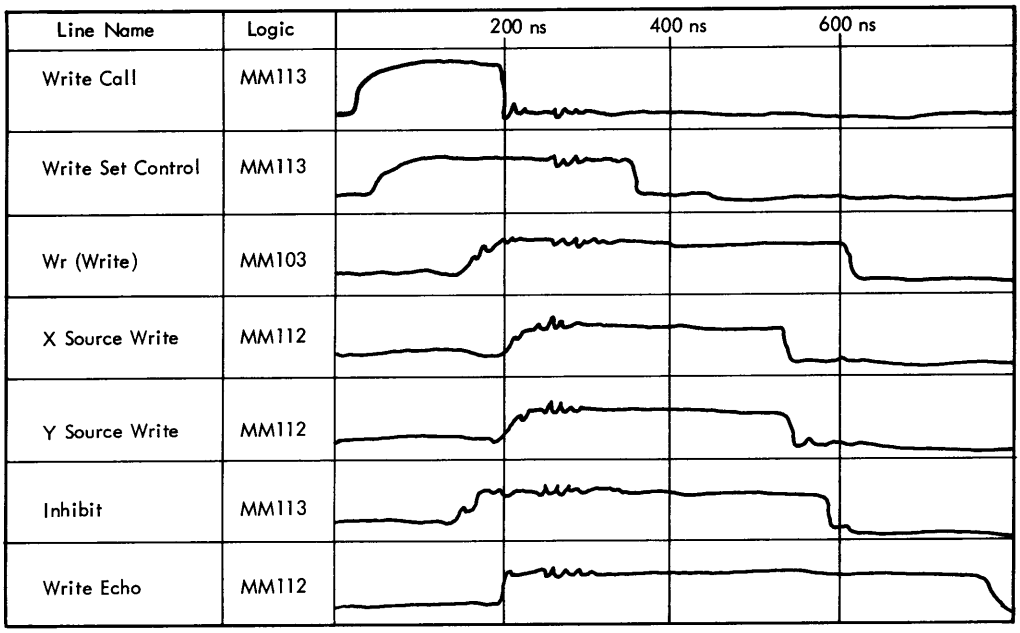


Figure 2-95. Core Storage Write Timings

CURRENT SOURCES

- Current sources supply drive current to the X- and Y-windings.
- The drive current comes from the secondary winding of a transformer.
- The primary windings of the source transformers are driven by transistors signaled to turn on by the storage clock.

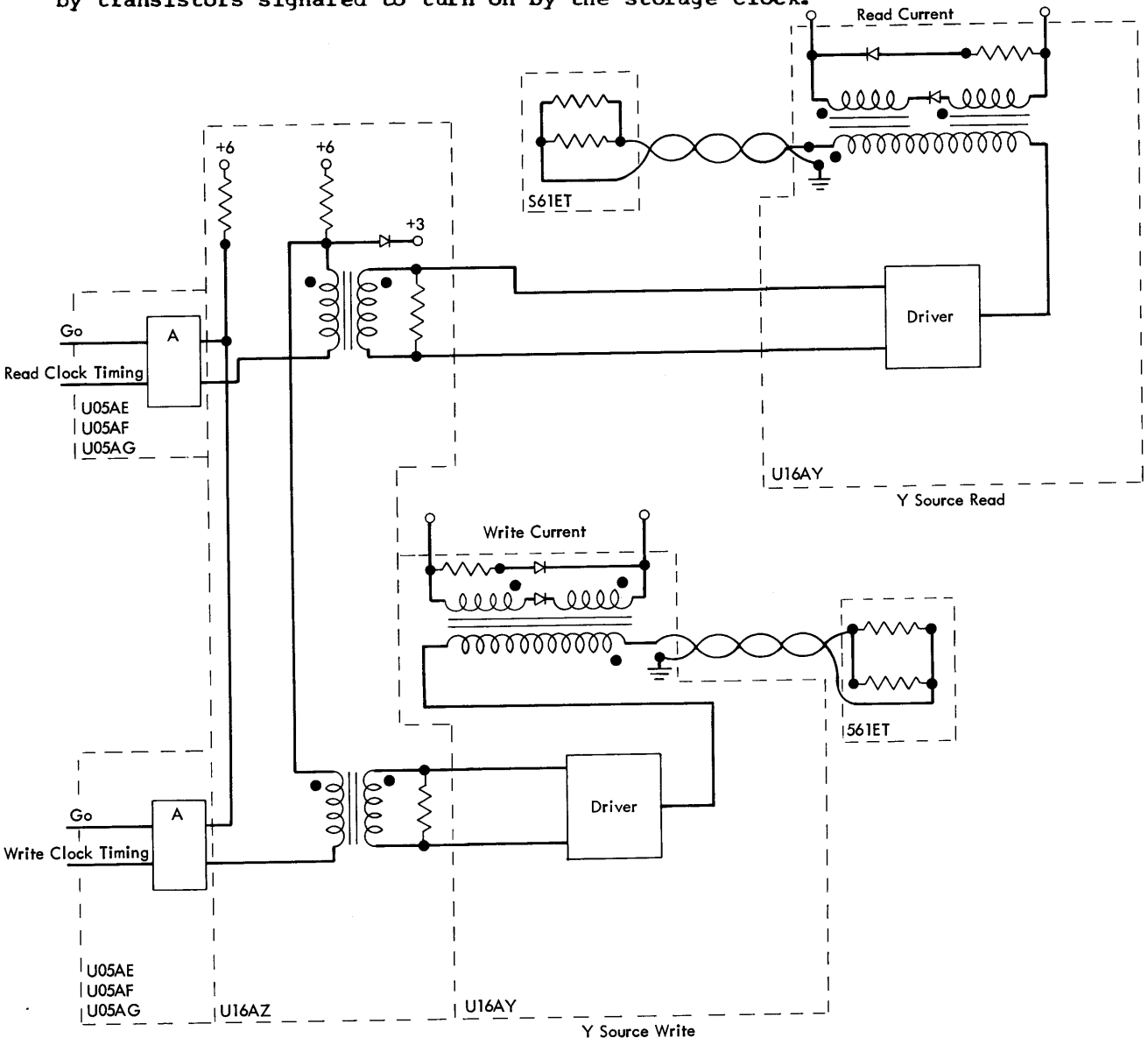


Figure 2-96. Current Sources

Current sources are special circuits designed to supply drive current to the X- and Y-windings. In the basic 8K storage unit, there are four current sources: X-source read, X-source write, Y-source read, and Y-source write. Each current source consists of a transformer secondary winding. The primary winding of each transformer is driven by a transistor circuit (Figure 2-96). When the clock signals Y-read source timing, the Y-source read circuit is turned on to cause current to flow in the Y-source transformer primary. This in turn causes transformer secondary current flow. By this time, the selection circuitry has coupled the source transformer to a single drive line and current flows through the drive line. This

same action occurs in the X-source read circuit: the clock signals when to turn on, the transistors provide transformer primary current, and the transformer secondary provides drive current for the selected X-line.

GATE AND SELECTION SYSTEM

- The gate and selection system directs drive current to a single X-line and a single Y-line.
- The gate and selection logic consists of control drivers, address decoders, and gates.

The purpose of the gate and selection system is to direct drive current from a current source to a single X-line and a single Y-line. The gate and selection system acts like a switch at each end of the drive lines to direct the current source drive current to a single drive line (Figure 2-97). Thus, the current source supplies the current, and the gate and selection circuitry simply directs the current to the appropriate drive line.

The gate and selection circuitry consists of control drivers (SI5EX), address decoders (U03AD), and gates (SI5ES, SI5ET). In Figure 2-98, the composite logic is shown for the Y-direction. Notice that the gates are turned on to direct the current source to the appropriate drive line.

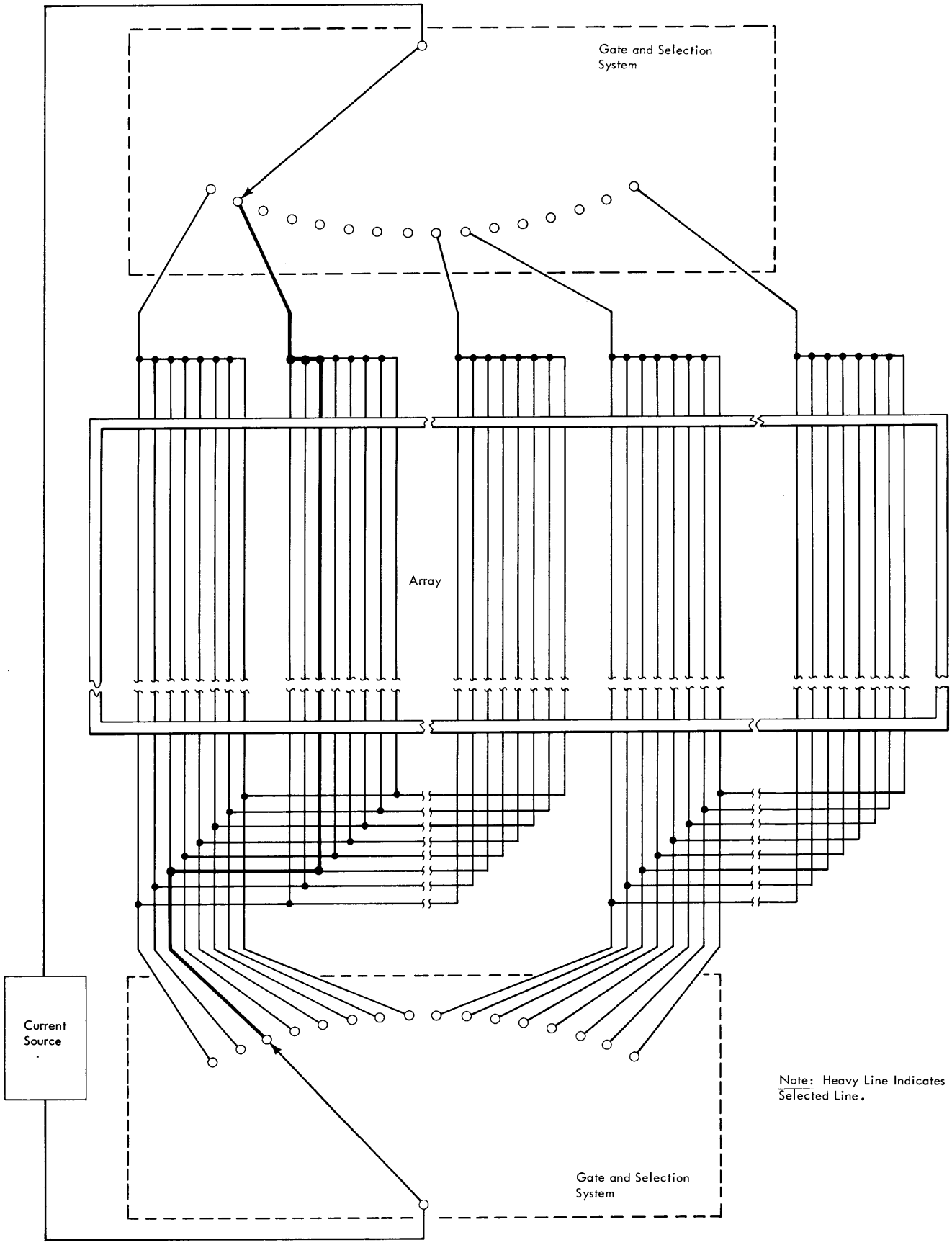


Figure 2-97. Gate and Selection System

SENSE/INHIBIT SYSTEM

- A combination sense/inhibit winding is used.
- Each sense/inhibit winding goes through 4,096 cores, parallel to the X-drive lines.
- During a read cycle, the sense/inhibit winding senses pulses caused by cores changing states.
- During a write cycle, the sense/inhibit winding can prevent cores from changing state.

The M2-I uses combination sense/inhibit windings for storing and retrieving information. This winding is wound parallel to the X-winding and it goes through 4,096 cores in a single core plane. There are four such windings for each 16,384-core plane. During a read cycle, a core that switches (was logical 1) induces a pulse onto the sense/inhibit winding. This pulse is amplified by a sense amplifier (Figure 2-99).

The sense amplifier senses a change or difference in current on the sense winding and is called a differential amplifier. To prevent unwanted noise from being amplified in other storage sections, only the sense winding outputs for the 4,096 block of storage being addressed are allowed to reach sense amplifiers. The sense amplifier gate allows the desired sense winding output to be amplified. The output of the sense amplifiers appears at the input of the detector circuit. Here the strobe pulse from the storage clock gates the sense amplifier output to a data latch which stores the bit until used by the processing unit. During a read cycle, if a core does not switch (was logical 0), no pulse is induced onto the sense winding, and therefore the data latch is not set.

During a write cycle, if a bit is to be stored in a core, the core is switched by the effect of the coincident X- and Y-drive currents. In this case, the inhibit current is not allowed to flow (Figure 2-99). During a write cycle, if the bit is to be blocked from setting, inhibit current must flow to oppose the magnetic effect of the X-write current. With the absence of a store signal at the inhibit driver input, the inhibit driver turns on, inhibit current flows and the effect of the inhibit current cancels the effect of the X-winding current. As a result, the core is not set to a logical 1 state.

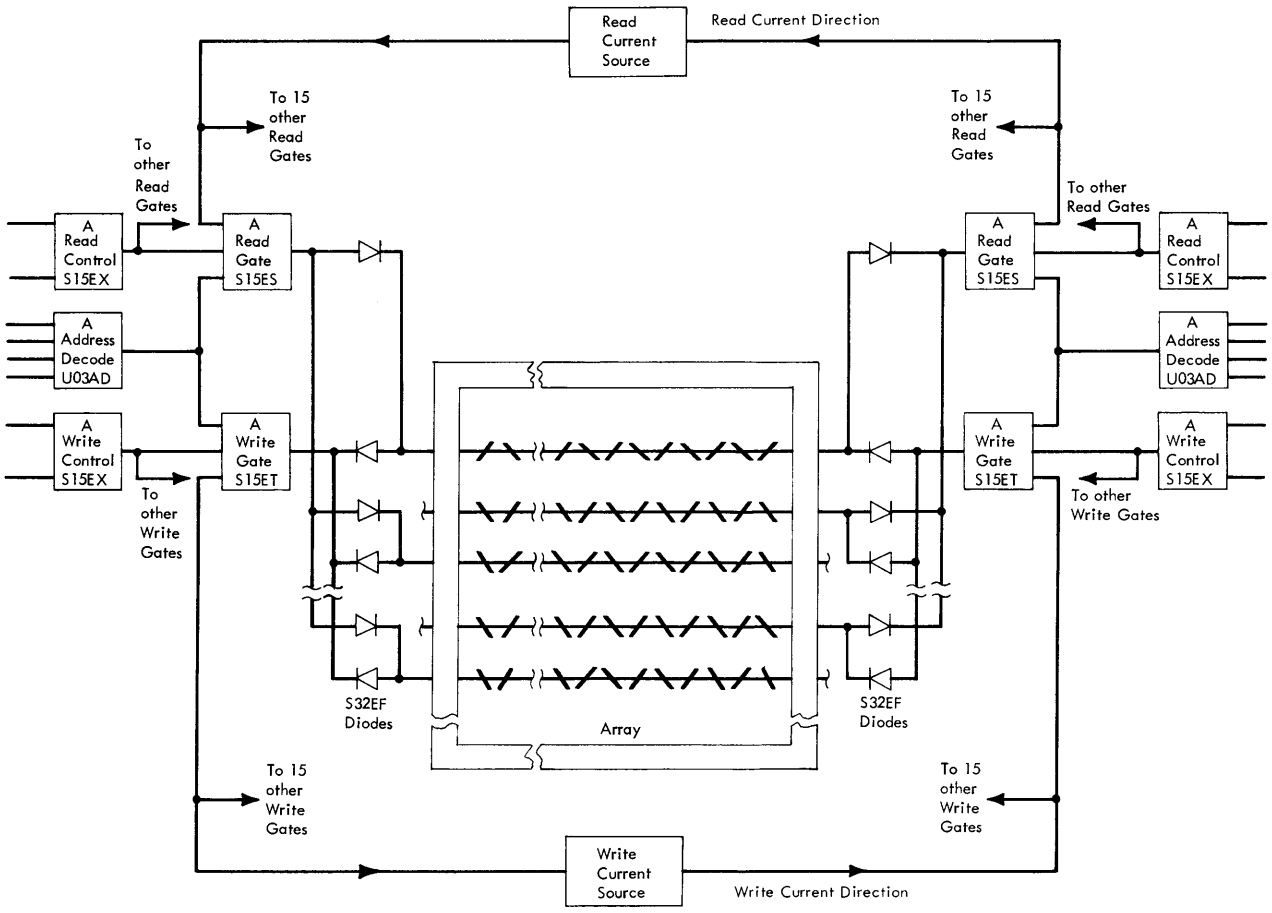


Figure 2-98. Gate and Selection Logic

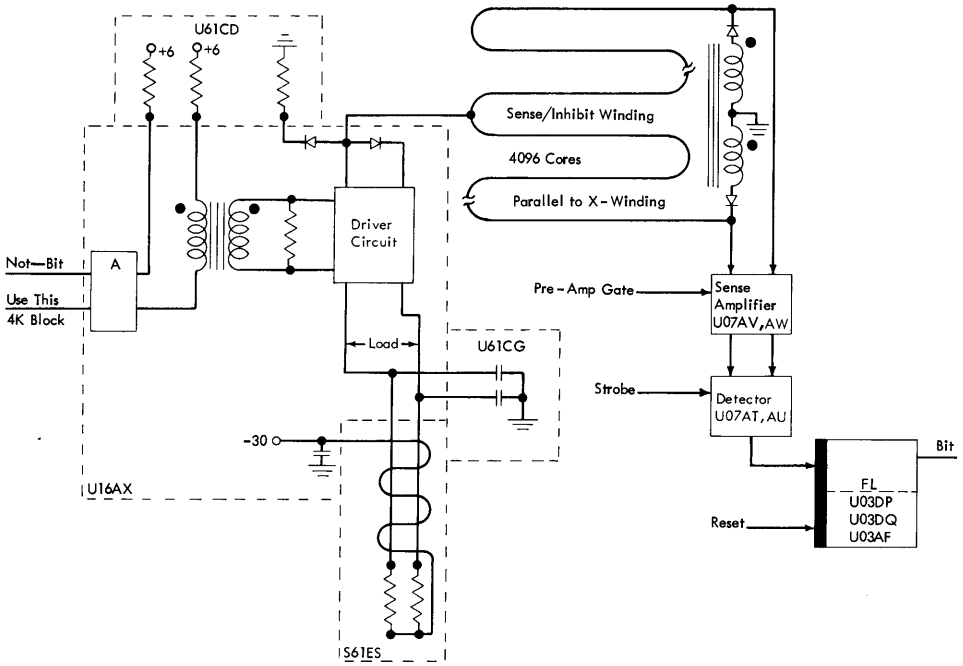


Figure 2-99. Sense and Inhibit Logic

POWER SUPPLY AND TEMPERATURE COMPENSATION

- Four power supply voltages are required for operation of the M2-I: +6, +3, -3, -30.
- The logic boards are cooled by fans.
- The core array is heated by a heater element, and cooled by a fan.
- A unit called the Proportional Controller controls the heat generated by the heater element.

The M2-I requires four dc power supply voltages for operation of the logic and drive circuitry. The voltages and the 2030 power supply from which they originate are:

+6 Power Supply 3

-30 Power Supply 6

+3 Power Supply 7

-3 Power Supply 8

A -18 volt supply is generated internally in the memory from the -30 volt supply. This special voltage supplies bias current for the sense amplifiers.

Also supplied to the memory gate is a 208 volt ac line and a 24 volt dc line for operation of the temperature control system. This system consists of two continuously-running fans to cool the logic gates, and a core array heater and fan for controlling the temperature of the core array.

A thermistor near the core array senses the array temperature. The variation in thermistor resistance signals a separate unit called a proportional controller. this unit is located behind the memory gate on the 2030 frame. Its purpose is to control the power supplied to a heater element located near the core array. Varying the power supplied to the core array heater element controls the temperature of the core array. The heater fan, located under the core array, runs continually to blow air past the heater element into the core array.

The LP light on the ROS area of the 2030 console indicates low pressure in the C-CROS air system. When the M2-I is installed, a thermostatically-controlled relay turns the LP light on if the array temperature is below its correct operating limit. If the array temperature rises above 120 deg. F, a thermal contact located in the core array area initiates a power-off sequence in the 2030.

AUXILIARY STORAGE

- Auxiliary storage is an added area for CPU, and I/O control and status information.
- Auxiliary storage requires additional addressing in the Y-dimension only.
- Auxiliary storage is referred to as bump storage.
- The amount of auxiliary storage available varies with the size of the main storage unit.

Included in the storage unit is an additional area of auxiliary storage used by multiplexor channel and by the processing unit. This auxiliary storage is formed by adding eight extra Y-lines to the basic core plane (Figure 2-100). An 8K unit, with five core planes, has 512 positions of auxiliary storage. Of these 512 positions, 256 are for CPU local storage, and 256 are for multiplexor channel usage. Eight auxiliary storage lines (Y-lines) intersect with 128 X-lines to form 256 byte-positions for CPU local storage, and 256 byte-positions for multiplexor storage. In the 8K unit, these 8 auxiliary storage windings intersect with the 64 upper X-windings to form the 5-, 6-, 7-, and P-bit positions (see Figure 2-89). The eight auxiliary storage windings intersect with the bottom 14 lines to form 512-byte positions for lower bits. This would correspond to bits 0, 1, 2, 3, or 4.

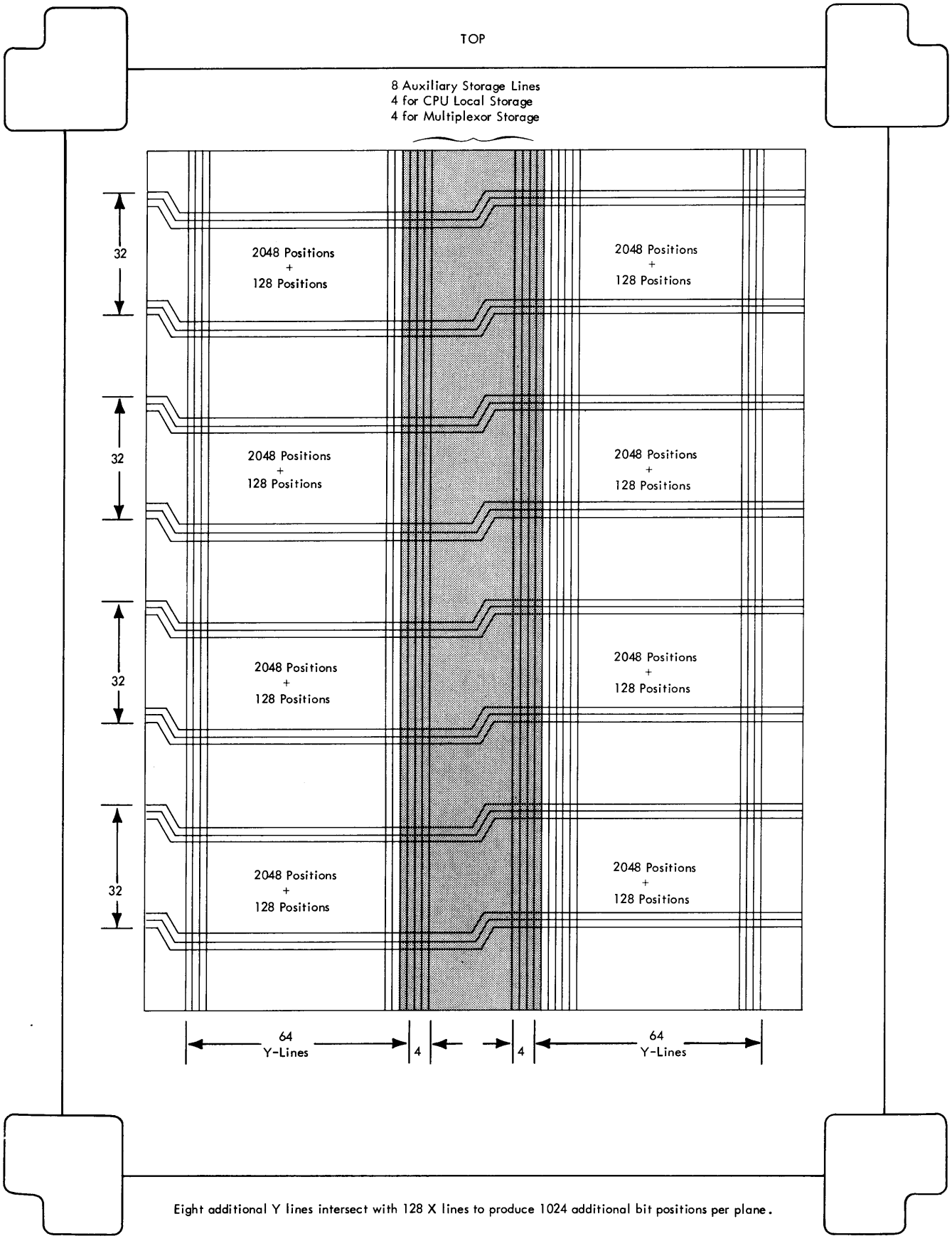


Figure 2-100. Auxiliary Storage Core Plane Windings

In a 16K core array, four 256-byte auxiliary storage areas are available. The same scheme is used to create the extra storage positions: eight auxiliary storage lines in the Y-direction intersect with 128 X-lines to produce 1,024 additional byte positions of auxiliary storage. The auxiliary storage areas are labeled MPX 0, MPX 1, MPX 2, and local storage.

A 32K storage unit provides the maximum amount of auxiliary storage. In this unit, up to eight 256-byte auxiliary storage areas are available. These areas are MPX 0, MPX 1, MPX 2, MPX 3, MPX 4, MPX 5, MPX 6, and local storage.

Expansion beyond 32K does not yield additional auxiliary storage. Therefore, auxiliary storage is always located in the lower 32K.

8K STORAGE OPERATION

- A complete storage cycle consists of a read cycle and a write cycle.
- In a given storage cycle, drive current flows through the selected drive lines in one direction for read, and in the opposite direction for write.
- At the end of the read cycle, all cores at the addressed position are logical 0.
- An interlock in the 2030 ensures that a write cycle occurs between read cycles so a storage position is not left blank.
- The inhibit drivers turn on for those bits where the core is to be left at logical 0.

Description (Figure 2-101)

When the 2030 places an address into the M- and N-Registers and requests a read cycle, the storage clock is started. The address lines from the M- and N-Registers combine with clock timing to turn on X- and Y-read current sources X- and Y-read gates, and X- and Y-read control drivers. This causes read current to flow through one X-winding and one Y-winding. The coincident read drive currents cause all the cores at the addressed position to experience a magnetic effect great enough to switch all cores to the logical 0 magnetic state. Any cores that change magnetic state from logical 1 to logical 0 cause a current pulse to be induced onto the sense winding. The clock signals combine with the M- and N-Register bits to gate the appropriate sense amplifiers. The amplified sense bits cause data latches to set on. Toward the end of the read cycle, the 2030 is signalled that the data is ready. At this time, all cores in the addressed position are set to logical 0. This means the addressed position contains an even parity byte (000000000).

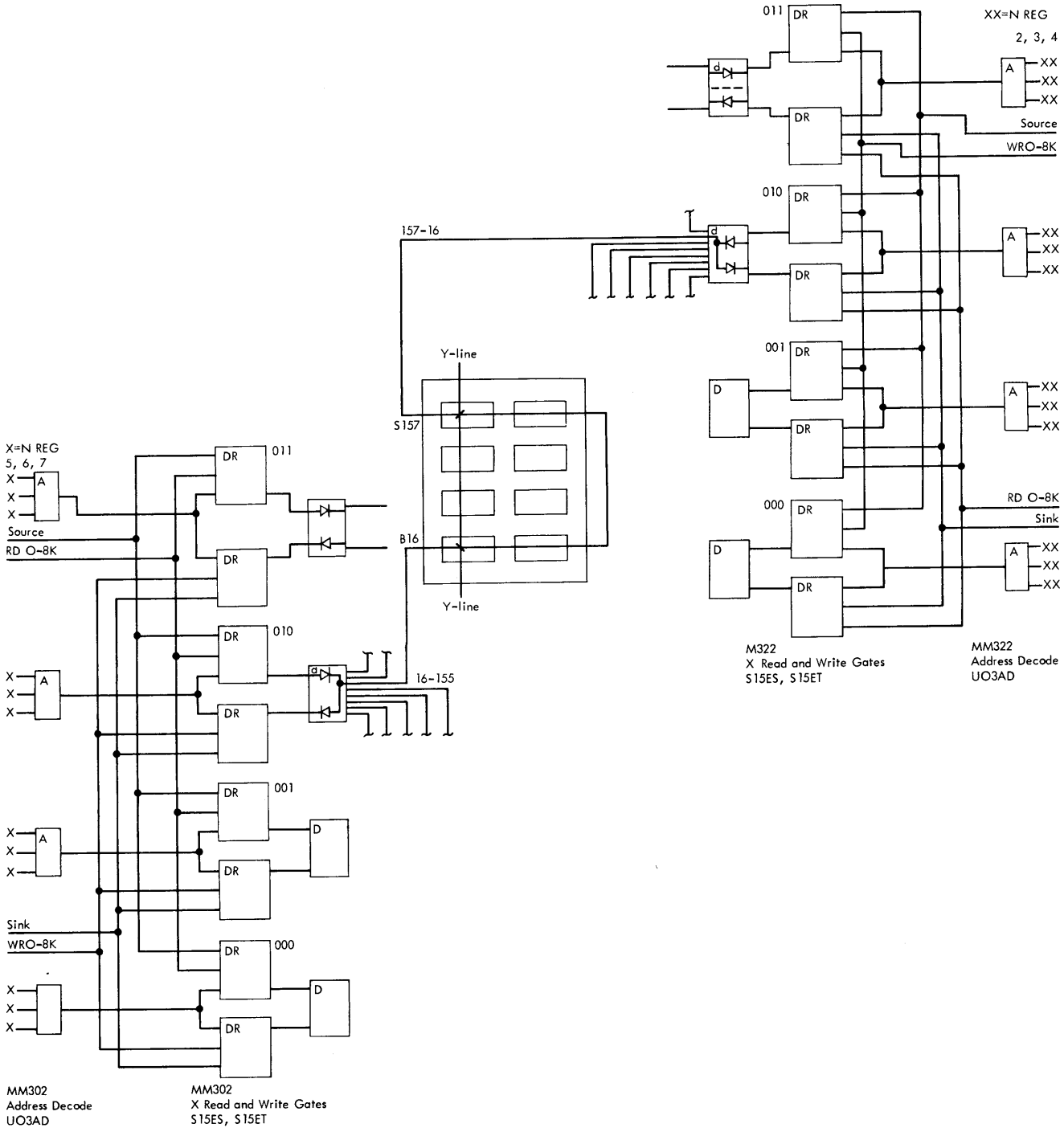


Figure 2-101. 8K Storage Operation

The write call signal from the 2030 starts the storage clock and conditions a write cycle. The M- N-Register contain the same address as during the preceding read cycle. However, the address bits now combine with write timings to turn on X- and Y-write current sources, X- and Y-write gates, and X- and Y-write control drivers. The result is that current flows in the opposite direction through the same two drive lines as during the preceding read cycle. With no further control, this would result in all cores in the addressed position being set to logical 1. However, during a write cycle, it is necessary to set some cores to logical 1 while leaving the other cores at logical 0. The byte of information to be stored in core storage was placed in the R-Register by the 2030 before the storage write cycle was initiated. To store the correct byte, the byte in the R-Register controls the appropriate set of inhibit drivers so inhibit current will flow in the bit sections where the core is to remain logical 0, and inhibit current is blocked in the bit sections where the core is to be flipped to logical 1. Thus, if the R-Register contains the byte P00101101, the 0-, 1-, 3-, and 6-bit inhibit drivers are turned on while the P-, 2-, 4-, 5-, and 7-bit inhibit drivers are blocked from turning on. The result is that although coincident write current flows through all cores in the addressed position, only those cores that experience no inhibit current are set to logical 1. This causes the byte that was in the R-Register to be stored in the addressed storage location.

Circuit Objectives

Assume the binary address 0000 0010 1001 0010 is in the M- and N-Registers and that the 2030 calls first for a read cycle, and then a write cycle. The byte read out is to be regenerated (placed back into the addressed position) on the write cycle.

1. Start the storage clock (MM122).

   Read Call

2. Turn on the read set latch to enable a read cycle (MM102).
   Read Call
   (not) Delay Tap 200 ns

3. Set the main/local storage latch to define the area of storage to be addressed (MM212).
   Read Call
   (not) Early Local Storage

4. Select and drive one X-line with read current. This involves turning on two read control drivers (one for each end of the X-line), two address decode switches (one for each end of the X-line), two read gates (one for each address decode switch), and the X-read current source.

   a. Turn on Decode Switch---010 (MM302). This is the X decode switch for the source side of the X-line.
      N Reg 6 Bit
      (not) N Reg 5 Bit
      (not) N Reg 7 Bit

   b. Turn on Decode Switch 010--- (MM322). This is the sink side of the X line.
      (not) N Reg 2 Bit
      N Reg 3 Bit
      (not) N Reg 4 Bit

   c. Turn on Rd1 0-8K Wr 8-16KB and Rd1 0-8 16-24 Wr 8-16 24-32
      These are the X control drivers and they condition the X read gates at the decoded address 010010 in the first 8K section (MM232).
      RD1 (from clock)
      (not) M Reg 1 Bit
      (not) M Reg 2 Bit

   d. An X-line has been decoded and a read gate has been conditioned at each and of the X-line. Now the read current source and sink must be turned on to cause current to flow (MM252).
      X Source Read
      Go (not M Reg 0 Bit)
      (not) M Reg 2 Bit

5. Select and drive one Y-line with read current. This requires turning on two read control drivers (one for each end of the Y-line), two address decode switches (one for each end of the Y-line), two read gates (one for each decode switch), and the Y read current source.

   a. Turn on Decode Switch - 0 - - 010 (MM402). This is a Y-decode switch for the source side of the Y-line. The gates are on the same logic page and are fed by the decode switches and the control driver.
      N Reg 0 Bit
      (not) N Reg 1 Bit
      (not) M Reg 7-Bit Ctrl
      (not) M Reg 4 Bit Ctrl

   b. Turn on Decode Switch 0001---(MM442). This is the Y-decode switch for the sink side of the Y-line.
      M Reg Not 3 and Not 4 Bits
      (not) M Reg 5 Bit
      M Reg 6 Bit

   c. Turn on Read 2 Control 0-32KA and Read 2 Control 0-32KB.
      These are the read control drivers that condition the Y read gates at

the decoded Y-address 0001010 (M-Register 3 thru 7, N-Register 0, 1).
Use Main Storage
Read 2 (from storage clock)

d. A Y-line has now been decoded and a read gate has been conditioned at each end of the Y-line. Now the read current source and sink must be turned on to cause current to flow (MM252).
Y Source Read (from storage clock)
Go

6. Develop the sense amplifier gate so the appropriate sense windings are gated to their respective sense amplifiers. The gate for this address is SA gate 0-8KA (MM692).
Not M Reg 1 Bit Cont
(not) M Reg 2 Bit
(not) N Reg 7 Bit

7. Gate the sense pulses to the sense amplifier, strobe the detector and latch the detector output. (MM512 through MM592).
SA Gate 0-8KA
SA In___Bit 0-8KA
Strobe 0-16K (from clock)

8. After the SA detector latches are set, the storage unit signals the 2030 CPU that the read data is ready (MM002).
Data Ready (from clock)

9. Without changing the address in the M- and N-Registers, the 2030 CPU requests a storage write cycle and starts the storage clock (MM122).
Write Call

10. Set up the storage clock for a write cycle by turning on the write set latch (MM113).
Write Call
Go

11. For the write cycle, it is necessary to select and drive the same X- and Y-drive lines as were driven on the read cycle. However, now they are driven with current in the opposite direction. This is done by conditioning the write gates instead of the read gates at this address. Consider the Y-line first. For this, it is necessary to turn on two control drivers (one for each end of the Y-line), two address decode switches (one for each end of the Y-line), two address gates (one for each decode switch), and the Y-write current source.

a. Decode switch-0--010 is still on because the M- and N-Registers have not been changed. This is the Y-decode switch for the sink end of the Y-line.
(not) M Reg 7 Bit Ctrl
(not) M Reg 4 Bit Ctrl
N Reg 0 Bit
(not) N Reg 1 Bit

b. Decode switch 0001---is still on because the M- and N-Registers have not been changed. This is the Y-decode switch for the source end of the Y-line.
M Reg Not 3 and Not 4 Bits
(not) M Reg 5 Bit
M Reg 6 Bit

c. Turn on the Y control-drivers, Write Control 0-32KB and Write Control 0-32KA (MM222). These will turn on the write gates at Y address 0001010.
Write B (from clock)
Use Main Storage

d. A Y-line has been decoded and a write gate has been conditioned at each end of that line. Now the write current source and sink must be turned on to cause write current to flow (MM252).
Y Source Write (from clock)
Go

12. Select and drive the same X-line with write current. This requires two control drivers (one for each end of the X-line), two address decode switches (one for each end of the X-line) and two address gates (one for each decode switch), and the X-write current source.

a. Decode Switch---010 is still on because the M- and N-Registers have not changed since the read cycle. This is the X-decode switch for the sink end of the X-line (MM302).
(not) N Reg 5 Bit
N Reg 6 Bit
(not) N Reg 7 Bit

b. Decode Switch 010---is still on because the M- and N-Registers have not changed since the read cycle, this is the X-decode switch for the source end of the X-line (MM322).
(not) N Reg 2 Bit
N Reg 3 Bit
(not) N Reg 4 Bit

c. Turn on X-control drivers Wr0-8 16-24 and Wr0-8K Rd1 8-16K.
These in turn condition the X-write gates at the decoded X-line 010010 (MM232).
Write A (from clock)
(not) M Reg 2 Bit
(not) M Reg 1 Bit

d. An X-line has now been selected and a

write gate has been conditioned at each end of that line. Now the write current source and sink must be turned on to cause write current to flow (MM252).
X Source Write
(not) M Reg 2 Bit

13. The appropriate set of inhibit drivers must be gated so that only one set of these drivers turns on. For this address, Inhibit 0-8KA must be turned on (MM502). The terminology 0-8KA denotes all even addresses in 0-8K of storage, similarly, 0-8KB denotes all odd addresses in 0-8K of storage.
(not) N Reg 7 Bit
(not) M Reg 1 Bit
(not) M Reg 2 Bit
Inhibit (from clock)

14. For those bits that are to be set ON, the inhibit driver must be blocked from turning on (MM732 through MM772). The store lines block their respective inhibit drivers.

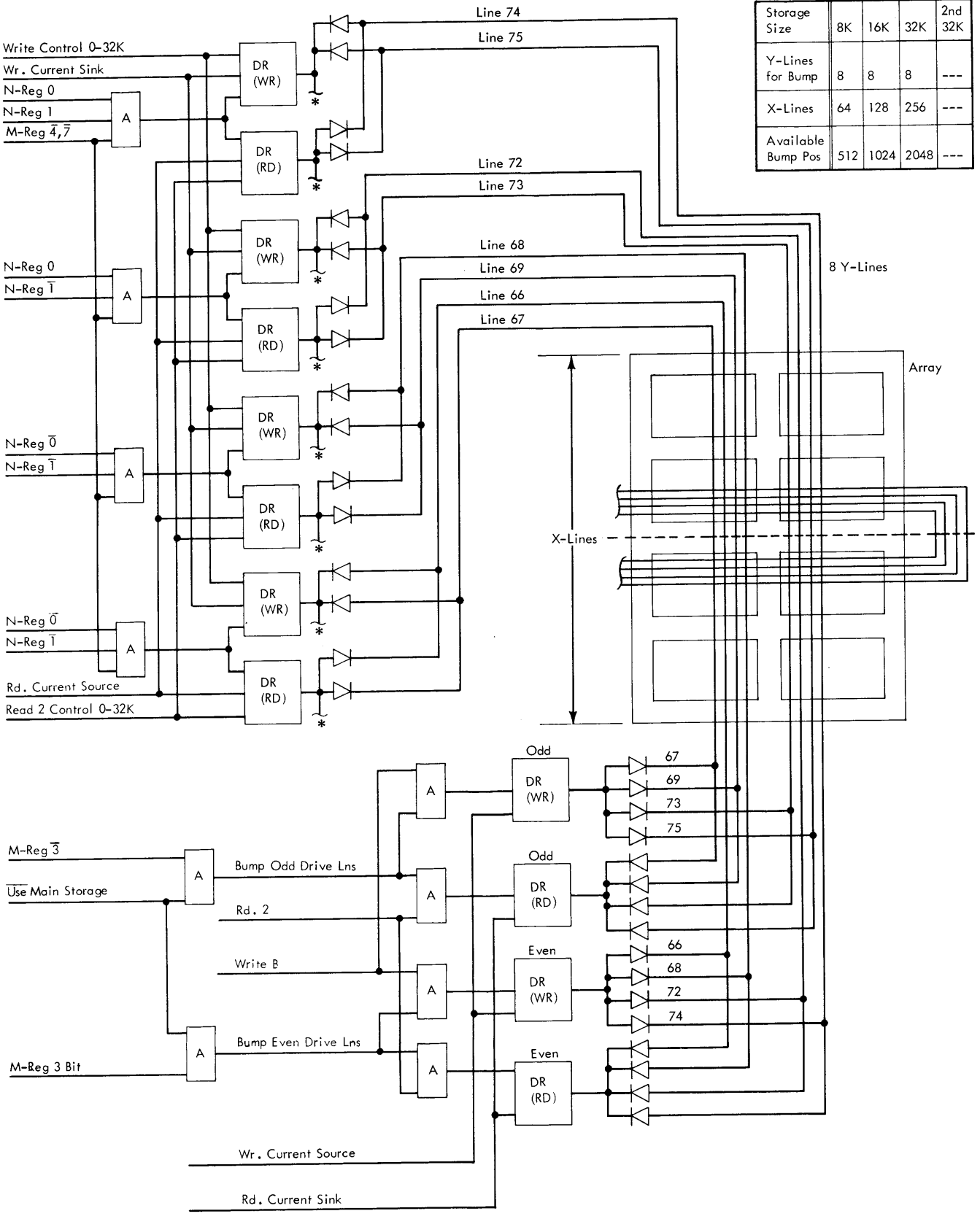
15. For those bits that are to be blocked from setting, the appropriate inhibit drivers are turned on by the (not) store lines. Inhibit current opposes the affect of the X-drive current and the core is not set (MM732 through MM772).

AUXILIARY STORAGE FOR 8K

- Auxiliary storage in the 8K unit consists of two 256-byte storage areas.
- Eight additional Y-lines intersect with 64 X-lines to produce 512 additional storage positions.
- Two additional Y-read gates and two additional Y-write gates provide control of the extra Y-lines.

Circuit control of auxiliary storage for the 8K unit requires two additional sets of Y-line bump decode read-write gates (Figure 2-102). These gates are controlled by the M-Register 3-bit (MM152). They control one end of the eight additional drive lines. The other ends are connected to four sets of read-write gates used to address main storage. X-line decode and drive is no different than for main storage.

Storage Size	8K	16K	32K	2nd 32K
Y-Lines for Bump	8	8	8	---
X-Lines	64	128	256	---
Available Bump Pos	512	1024	2048	---

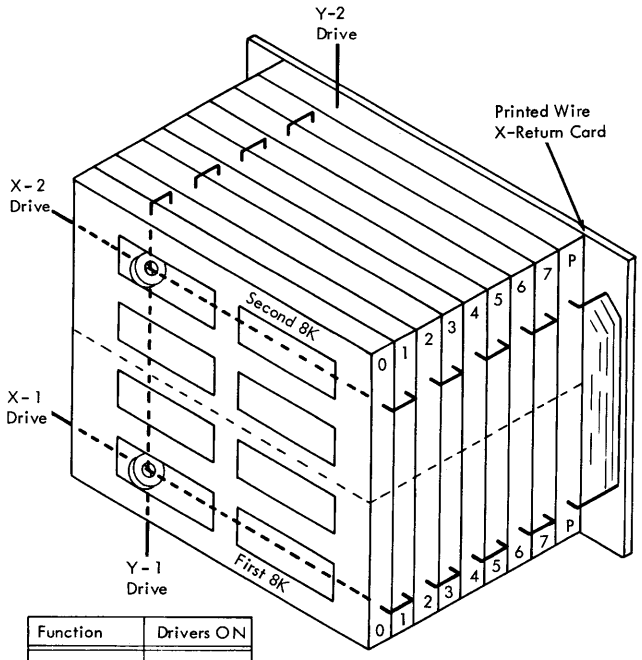


*Each Y-line driver is connected to 8 additional main storage Y-lines. The 4 RD and 4 WR drivers shown are the only ones that drive 10 lines.

Figure 2-102. Auxiliary Storage Drive Scheme

16K STORAGE OPERATION

- The 16K storage unit is composed of nine core planes.
- The X-return wires connect the two 8K units in the X-direction.
- Phase reversal takes place between the two 8K units so only one unit is addressed at a time.



Function	Drivers ON
Read 1st 8K	Y-1 X-1
Read 2nd 8K	Y-1 X-2
Write 1st 8K	Y-2 X-2
Write 2nd 8K	Y-2 X-1

Figure 2-103. 16K Storage Operation

Description

The 16K storage unit contains nine core planes. The planes are wound so that two 8K storage units are produced (Figure 2-103). The X windings thread through all nine core planes, cross over to the other half of the array, then thread back through the upper halves of these same core planes. The Y-windings thread through all nine core planes once. The result is that if one X-winding and one Y-winding are driven with drive current, nine cores experience coincident drive current. Because the X-winding undergoes a phase reversal between 8K units, the respective cores in the other half of the array do not experience coincident drive current. To address the similar position in the second 8K block of storage, the drive current must be reversed (Figure 2-104). This reversal takes place at the X-control drivers and at the X-source drivers.

Circuit Objectives

Circuit control for the 16K unit is exactly the same as for the 8K unit with the exception of the X-control driver and X-source driver. The X-control driver determines the direction of current flow in the X-winding by switching on the proper X-gate, while the X-source drivers turn on the actual source current in the appropriate direction.

1. The M-Register 2-bit combined with the function read or write, controls the X-control drivers (MM232).
   M Reg 2 Bit

Rd 1
Write A

2. The M-Register 2-bit, combined with the function read or write, controls the X-source drivers (MM242).
   M Reg 2 Bit
   Read 1
   Wr

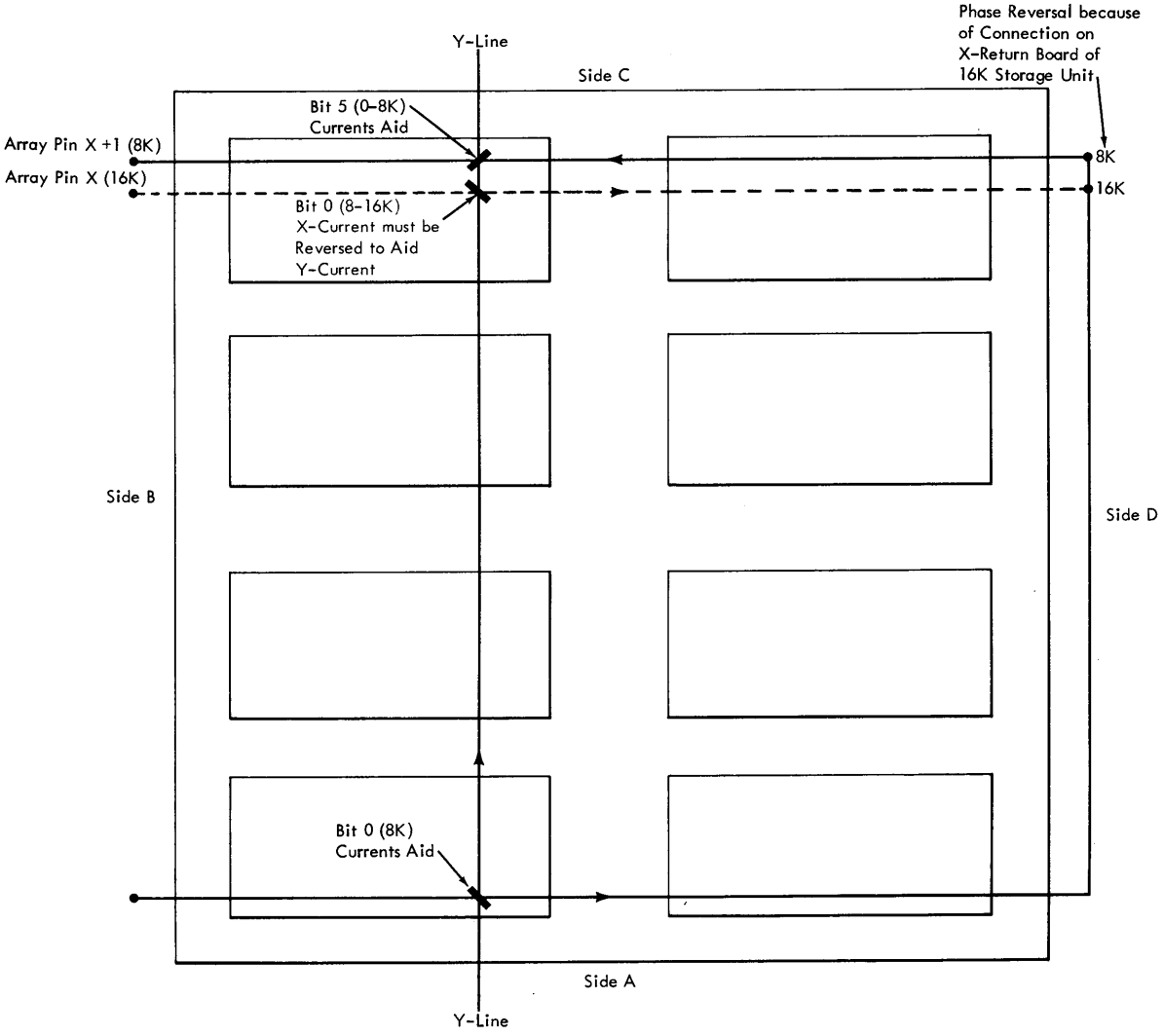


Figure 2-104. 16K Phase Reversal Wiring

AUXILIARY STORAGE FOR 16K

- Auxiliary storage in the 16K storage unit consists of four 256-byte storage areas.
- The eight additional Y-lines intersect with 128 X-lines in each plane to produce 1024 additional storage postions.
- Two additional Y-read gates and two additional Y-write gates control the additional Y-lines.

The 16K auxiliary storage unit uses the same additional Y-line bump decode gates shown in Figure 2-102. These gates are controlled by the M-Register 3-bit and the main/local latch (MM142). The X-drive lines for bump storage are the same X-lines that are used in main storage. Phase reversal is also used in bump storage to determine which 8K block is being exercised. The controls for this phase reversal are the same controls used in main storage.

32K STORAGE OPERATION

- One single set of Y-lines drives all 18 core planes.
- Two sets of X-lines drive 18 core planes; each set drives a 16K unit.
- Phase reversal takes place between the first and second 8K on the first set of X-lines, and between the third and fourth 8K on the second set of X-lines.

Description

The 32K storage unit consists of 18 core planes. The Y-windings go through all 18 planes in a serial manner (see Figure 2-91). There are two sets of X-windings; one for the first 16K, and one for the second 16K. In each 16K, the X-winding undergoes a phase reversal between 8K units. Selection of a single core storage position requires control of drive current direction for the phase reversal. This control is obtained by using the M-Register 2- and 1-bits to determine which X-control drivers and X-source drivers are turned on. Circuit control for 32K takes place at the X-control drivers and X-source drivers. The M-Register 2- and 1-bits select the appropriate set of control drivers (0-16K, 16-32K), and determine the direction of current flow by controlling the X-source drivers.

Circuit Objectives

1. Select the appropriate set of X-control drivers. The M-Register 1-bit determines which set (first 16K or second 16K) of X-control drivers is used (MM232, MM242). The M-Register 2-bit determines which half of the 16K section will be activated.
   M Reg 1 Bit
   M Reg 2 Bit
   Rd 1
   Write A
2. Allow current to flow in the proper direction according to the 8K unit being selected and whether the operation is read or write (MM252).
   Go
   X-Source Write
   M Reg 2 Bit
   X-Source Read

AUXILIARY STORAGE FOR 32K

- Auxiliary storage for the 32K storage unit provides up to eight 256-byte storage areas.
- One set of eight additional Y-lines goes through all core planes.
- There are two sets of X-lines: one is for the first 16K, one is for the second 16K.

A 32K storage unit can have up to 2048 auxiliary storage positions in the form of eight 256-byte bumps. Two sets of Y-bump-gates combine with the existing Y-decode gates to select a single Y-line. This selected Y-line goes through all four 8K storage units. However, only one 8K unit is selected because of X-line phase reversal, and because there is a separate set of X-lines and X-control drivers for each 16K of storage (MM232, MM242).

65K STORAGE OPERATION

- The 65K core storage unit consists of two 32,768-byte storage units.
- Each 32K unit contains all the necessary circuitry to address all positions in that unit.
- The M-Register 0-bit determines which 32K unit is used.

A 2030 with 65K core storage capacity has two separate core storage units. Each is mounted on hinges in the lower-left side of the 2030. The first 32K is the one located nearest the 2030 console. The second 32K is between the first 32K and the power supply tower. Each is a self-contained unit containing address decode and drive circuitry, and sense and inhibit circuitry. A single set of logics is provided to cover addresses up to 32,767. These logics contain appropriate notes to make one set of logics applicable for both units. All pin numbers and other locations are the same for both units. The only difference is the gate designation: the first unit is called gate C1; the second unit is called gate C2.
15 bits of the M- and N-Registers (Figure 2-105). The high order address bit (M-Register 0) forms the Go signal that determines which unit is to be addressed and blocks drive current in the unit not being addressed. Thus, the M-Register 0-bit can be thought of as having the value of 32,768. For example, if the binary address 00000000 00000000 is placed into the M- and N-Registers, and a read call is issued, both storage units begin addressing the low-order core storage position. Because the high-order address bit is logical 0 (not M-Register 0 bit), the low-order 32K unit receives the Go signal and drive current in the second 32K unit is blocked.

If the binary address 10000000 00000000 is placed into the M- and N-Registers, and a read call is issued, both storage units begin addressing the low-order core storage position. Because the high-order address bit is logical 1 (M-Register 0 bit), the address desired is 32,768, and the high-

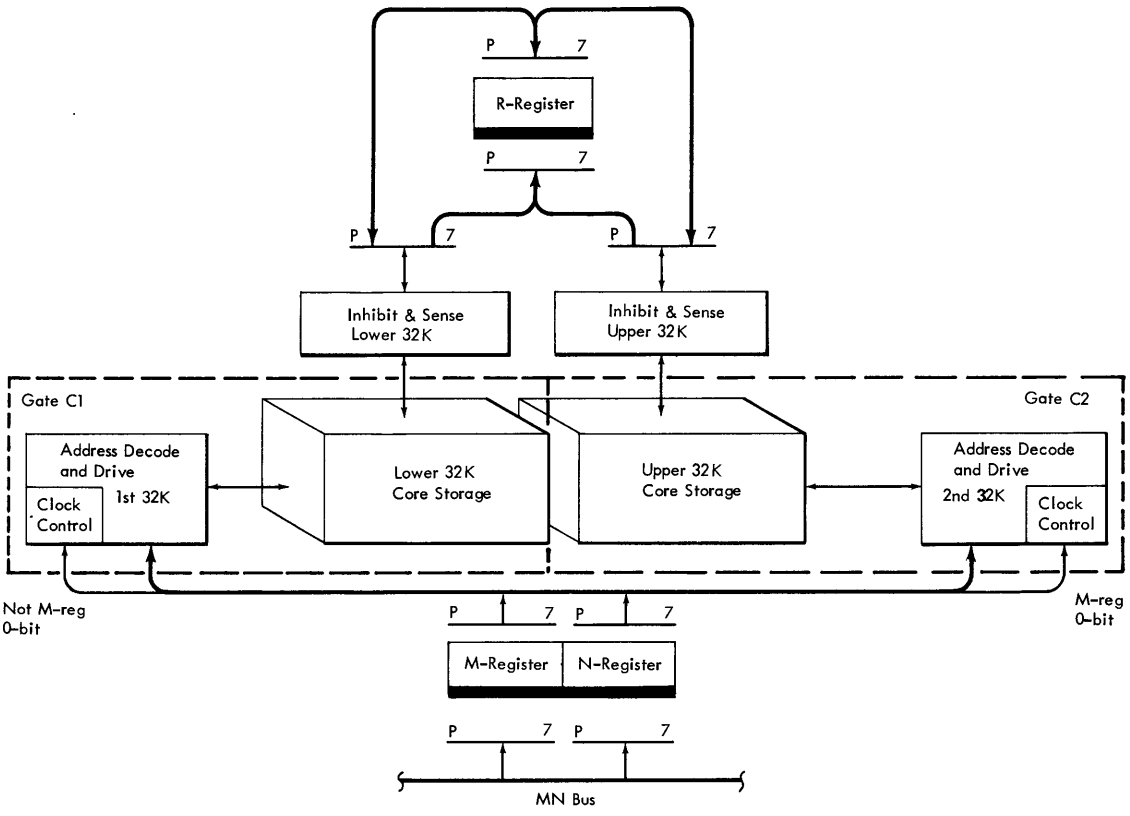


Figure 2-105. 65K Operation

order 32K unit receives the go signal. Drive current is blocked in the first 32K. The address read out is 00,000 + 32,768 which is 32,768.

Circuit Objectives

Circuit control for the 65K storage unit is dependent on the Go signal, developed from the M-Register 0-bit on logic page MM142. This same page applies to both the first 32K and the second 32K. For the first 32K, the M-Register 0-bit is inverted to produce the Go signal. In the second 32K, the M-Register 0-bit is not inverted since the M-Register 0-bit is required to produce Go for this unit. Thus, Go will be active for either one unit or the other, but never both. In the unit where the Go signal is not active, the following functions are blocked:

Data Ready on Read Cycle (MM113).
X- and Y-Source Drivers on Read Cycle (MM252).
Strobe on Read Cycle (MM692).
Write Set Latch on Write Cycle (MM113).

CSU INTERFACE

Each 32K M2-I core storage unit communicates with the 2030 over a series of signal lines known as the CSU Interface. All addresses, data, and control signals are transmitted over this interface. A brief description of the interface signals follows.

M- and N-Register Bit Lines (Logic Page MM001)

Sixteen bit-lines carry the address in the MN-Register to the core-storage addressing circuitry. The address is set into the MN-Register at the T1 time of the CPU clock cycle following the cycle when a CPU read-in is decoded by the control circuitry. The address does not change until the necessary CPU-compute and core-storage write cycles are taken. The MN-Register bit lines in order from the high-order position of the address to the low-order position of the address are:

M Reg 0 Bit
M Reg 1 Bit
M Reg 2 Bit
M Reg 3 Bit
M Reg 4 Bit
M Reg 5 Bit
M Reg 6 Bit
M Reg 7 Bit
N Reg 0 Bit
N Reg 1 Bit
N Reg 2 Bit
N Reg 3 Bit
N Reg 4 Bit
N Reg 5 Bit
N Reg 6 Bit
N Reg 7 Bit

The M Reg 0 Bit line serves an additional function on a 65K machine. If there is an M-Register 0 bit present, the desired address falls in the second 32K. If there is no M-Register 0 bit, the desired address falls in the first 32K. Read Call occurs at around T1 CPU-time. This is before the address in the M- and N-Registers is valid. Therefore, a read cycle is started in both M2-I units (on a 65K machine). Final selection of M2-I units occurs later in the read cycle. If there is no M-Register 0 bit, the X- and Y-source drives are blocked in the second M2-I (second 32K). If there is an M-Register 0 bit, the X- and Y-source drivers are blocked for the first M2-I (first 32K). In addition, the M-Register 0 bit line controls the data ready pulse to the 2030 and the strobe pulse in the appropriate 32K.

For the write cycle, the M-Register 0 bit simply blocks the Write Set latch in the low order 32K unit. This prevents any of the write latches from being set in the unselected 32K unit. This is possibe because the M-Register is not changed between read and write cycles and therefore, the M-Register 0 bit line is valid when the Write Call signal occurs. Thus if there is no M-Register 0 bit, the write cycle is blocked in the second 32K. If there is an M-Register 0 bit, the write cycle is blocked for the first 32K.

Unlike the M2, the 65K M2-I requires only one MN-Register for address drive. An intermemory cable supplies addresses from the first 32K to the second 32K.

Store Bit Lines (Logic Page MM001)

The nine store bit-lines provide the data input to the core-storage unit. These lines are direct outputs of the R-Register, and they go to the core storage inhibit drivers. The nine store bit-lines are:

Store P Bit
Store 0 Bit
Store 1 Bit
Store 2 Bit
Store 3 Bit
Store 4 Bit
Store 5 Bit
Store 6 Bit
Store 7 Bit

Read Call to Memory (Logic Page MM001)

Read Call to Memory signals the M2-I that the 2030 control circuitry has decoded a read operation. It occurs at T1-time of the cycle when a memory read cycle is to occur. Read call starts the memory clock and sets up a read cycle by turning on the Read Set latch (Logic Page MM102). Regardless of which 32K is being addressed, both clocks are started for Read Call. The M-Reg 0 bit line blocks the actual drive current for the 32K not being addressed.

Write Call to Memory (Logic Page MM001)

Write Call to Memory signals the M2-I that the 2030 control circuitry has decoded a write operation. It occurs at about T1 CPU-time of the cycle in which a write cycle is to occur. Write Call combines with the M-Register 0 bit line to determine which 32K storage clock is to run for a write cycle. If there is no M-Register 0 bit, the desired address is located in the first 32K and the first 32K clock is started. If there is an M-Register 0 bit, the desired address falls in the second 32K, and the second 32K clock is started.

Mach Reset Sw (Logic Page MM001)

The machine reset switch signal line blocks the advance of the memory clock. Machine reset turns off the Read Set control latch (Logic Page MM102), the Write Set control latch (Logic Page 113), and sends a pulse down the delay lines to reset the rest of the read and write latches (MM122).

Early Local Storage (Logic Page MM001)

The early local storage occurs before Read Call to allow setting the main/local storage latch to the local position (Logic Page MM212). When set to the local position, the main/local storage latch blocks the Y-control drivers, (Logic Page MM222), and allows the local storage control drivers to turn on (Logic Page MM142). The next time Read Call occurs and there is no request for local storage, the main/local storage latch is reset to the main storage state.

Read Echo (Logic Page MM002)

Read echo is a signal required by the 2030 in manual store operations. It follows a Read Call, and indicates that the Read Call was received, the memory clock is running, and that a read cycle is in process. Its purpose is to interlock the 2030 until the data is read out of the addressed position. The read echo results when the delay line pulses set and reset the read echo latch (Logic Page MM113).

Write Echo (Logic Page MM002)

Write echo is a signal required by the 2030 in manual store operations. It follows a Write Call, and indicates that the Write Call has been received, that the memory clock is running, and that a write cycle is in process. The write echo occurs when the delay line pulses set and reset the write echo latch (Logic Page MM112).

Memory Sense Bit Lines (Logic Page MM002)

These nine lines represent the data output of the core storage unit. They are active at memory strobe time. The core storage unit identifies the data with the data ready pulse to the 2030. If the 2030 wishes to use this data, the data ready pulse is allowed to set the data into the R-Register. The nine sense lines presented to the 2030 in order from high order to low order are:

Mem Sense P Bit
Mem Sense 0 Bit
Mem Sense 1 Bit
Mem Sense 2 Bit
Mem Sense 3 Bit
Mem Sense 4 Bit
Mem Sense 5 Bit
Mem Sense 6 Bit
Mem Sense 7 Bit

Data Ready (Logic Page MM002)

Data ready is the data strobe pulse to the 2030. The M2-I uses this signal to notify the 2030 that the read data is available on the memory sense lines. If the data is to be used by the 2030, the data ready pulse is allowed to set the memory sense data into the R-Register. The M-Register 0-bit gates data ready from the second 32K, no M-Register 0-bit gates data ready from the first 32K. This selection is necessary because both clocks are started for a Read Call.

CHAPTER 3. PRINCIPLES OF OPERATION

INSTRUCTION READ-IN

- All operations start with the entry of the op-code portion of the instruction.
- The address of the op-code byte is in the instruction counter (I and J registers).
- The op-code in the first byte is decoded to determine the type of operation and the size of the instruction.
- The I-cycle routine is included as part of each operation described in the following sections.
- CAS logics are used to illustrate the first of these operations and the CLF charts for a second group.

All operations are started in a common micro-routine called I-cycle start. The address of the new instruction is in the I and J registers either from the previous operation or from the IC restore routine at the end of the operation. The latter case results when the I and J registers are required to perform the operation. The sequence is described at the end of the pack operation. For the purpose of illustrating the I-cycle, all instructions are started through the normal I-cycle start entry at address 0100.

The I-cycle start routine reads in the first byte of the new instruction from the specified address. This byte contains the operation code. By progressively testing the bits of the two character code, the routine is branched to the exit for a specific operation. The decode indicates the type of instruction, and thus, the number of bytes to be taken. The branch on condition instruction and the binary add instruction in the RR format and the pack instruction in the SS format are discussed to illustrate the use of the CAS logic.

The CAS logic illustrations used for the following discussions are composites of many logic pages. They should not be used for servicing. The entry and exit points used for discussion can be traced from sheet to sheet. Other entry and exit paths are terminated in a box showing the conditions. The note blocks found on the individual CAS logic sheets are not shown on the illustrations but are included as part of the text.

A second group of operations are discussed with reference to the condensed logic flow (CLF) charts supplied with the system.

ROS TIMING TO CORE STORAGE TIMING

- The information read out of core storage on one ROS cycle is used on the next ROS cycle.

From the timing of a ROS cycle, we know that the SALs are set by T4 time of the ROS cycle that the ROS word is read out on. When ROS reads out a word that requests core storage operation, the memory delay clock is signalled to start at the beginning of the next CPU cycle. In Figure 3-1 we can see that during the first ROS cycle shown, a ROS word is read out requesting (for the next clock cycle) a read from main storage at the address in the I and J-register, add 1 to the contents of the J-register, and reset position 7 of S-register to ZERO.

Note: Figure 3-1 assumes that previous ROS cycles have been done.

During the second cycle, read call starts the memory clock, and the information in core storage at a given location is read out and set into the R-register. At

the same time, the information in the J-register is set in the A-register and routed to the ALU. The output of the B-register is blocked and eight ZERO's are routed to the ALU instead. A carry is forced into the ALU so the result is the contents of the J-register is increased by ONE. The output is routed to the J-register. During this time a new ROS word has been addressed and read out, which will cause the information in the R-register to be written back into core storage at the same location it was read from during the next cycle. Also, the information in the R-register is routed through the ALU and out on the Z-bus to the G-register. At the same time, the high four bits on the Z-bus are checked to see if they are all zeros; if so, the 4th position of the S-register is set to ONE.

This example shows the timing relationship between the ROS word read time and the execution of the same word.

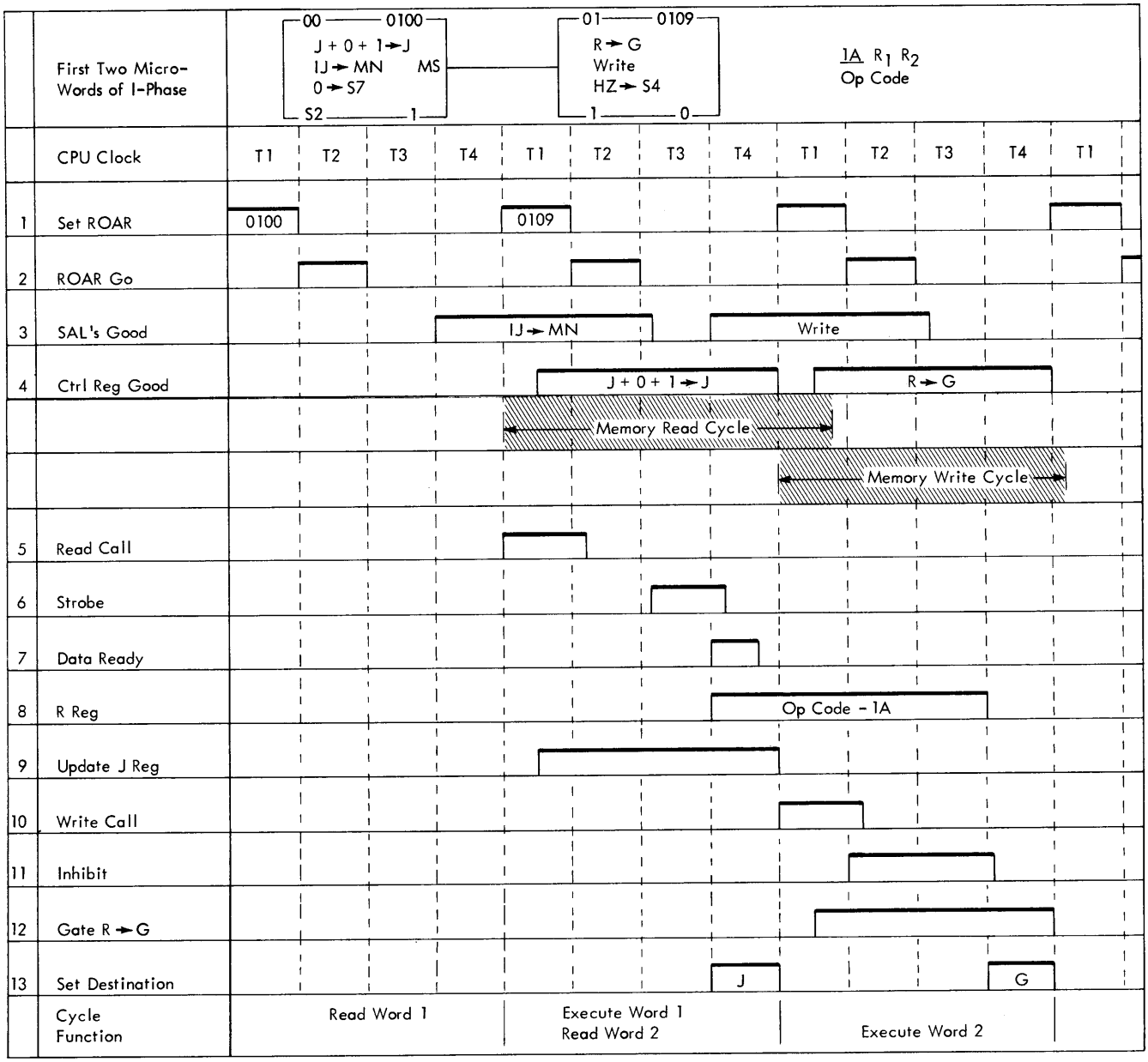


Figure 3-1. ROS to Memory Timings

BREAK-IN-TIMINGS

- A microprogram break-in, channel request, requires a dead cycle to keep the operation in step.

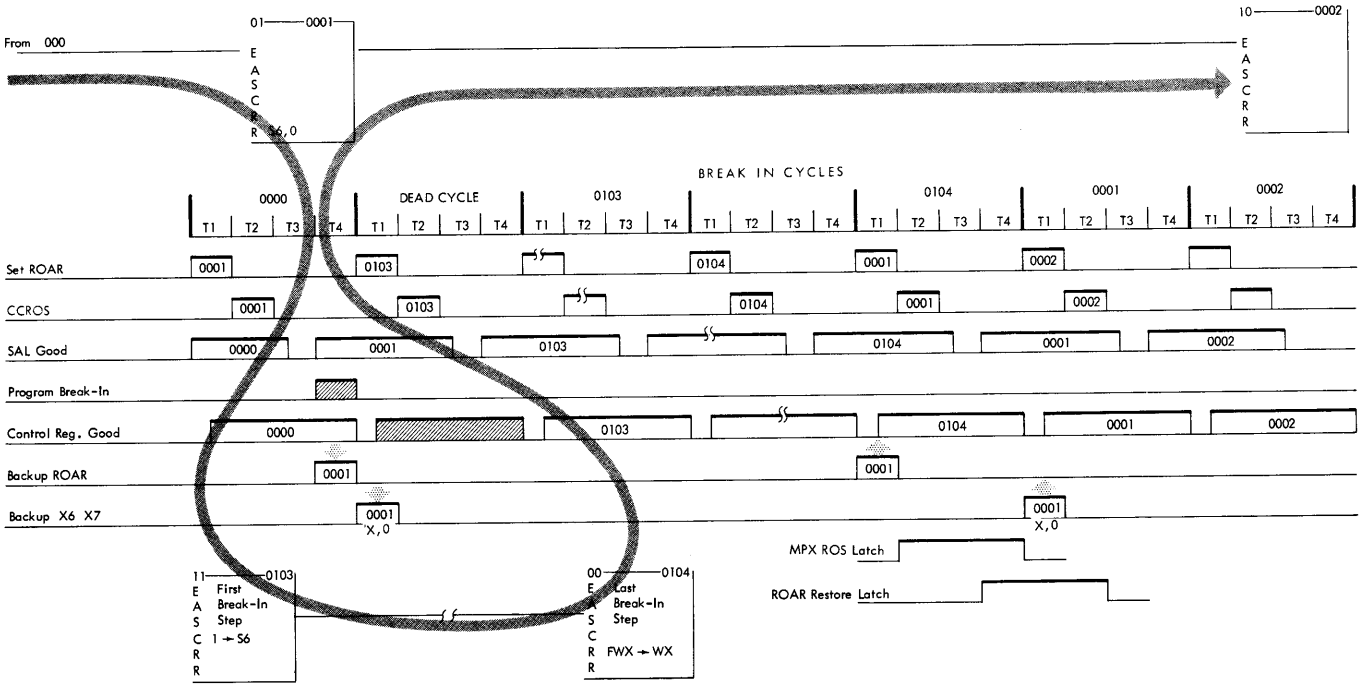


Figure 3-2. Microprogram Break-In

To help explain the timing during a microprogram break-in, let's use the example shown in Figure 3-2. The main microprogram uses ROS words at address 0001 and 0002. A break-in causes the main program to stop and a branch to the ROS word at 0103. The sequence of operation is shown by the darkened arrow. Let's examine the operation cycle by cycle. The cycles are labeled according to the time that the control-registers latches are good for that address. Remember the ROS word 0000 is read out 1-cycle prior to this time.

CYCLE 0000

At T1 time of this cycle, the address of the next microprogram step, 0001, is set into ROAR. At T2 time, the CCROS GO pulse starts the ROS delay clock and causes the data at address 0001 to be read out to the SAL's. The SAL's are good by T4 time. Now assume that during T2 time, a microprogram break-in was called for. Because of this break-in the address in ROAR (0001), is stored in a backup ROAR at T4 time.

DEAD CYCLE

This is called a dead cycle, because no control register latches are set. The set pulse to the latches is blocked for the first cycle of the break-in. At T1 time of this cycle, a new address (0103) is forced into ROAR. This address is the first step of the alternate microprogram.

Also at this time, the branch conditions for address 0001 are set into X6 and X7 buffer latches. The branch conditions have been tested at this time by the SAL's that are good for address 0001. The status of the branch condition must be stored since the condition of the latch might change during the alternate microprogram routines. In our example, we tested bit-6 of the S-register for the X6 position, and X7 is automatically set to ONE.

CYCLE 0103

This is a normal cycle that executes the ROS word at address 0103.

BREAK-IN CYCLES

These are the normal cycles of the alternate program.

CYCLE 0104

This is the last cycle of the alternate program. During this cycle, the control-register latches are good for address 0104. At T1 time, it is necessary to set ROAR with the next address to be executed. Since this is the last step of the alternate routine, the address where the main program was when the break-in occurred is needed. Therefore, the mnemonic FWX->WX causes the backup ROAR to be gated to ROAR.

MPX ROS LATCH OR ROAR RESTORE BUFFER LATCH SX

This latch is turned on at T2 time when a statement in the microprogram specifies that ROAR must be changed using the backup ROAR. In our example, the statement is in word 0104, FWX->WX. One purpose of this latch is to allow the ROAR Restore Latch to be set at T4 time of the cycle in which the ROS word at 0001 is read out.

CYCLE 0001

The control-register latches are good for address 0001 during this cycle. Even though we have addressed this ROS word before, this is the first time that it is executed. The first time that 0001 was addressed, the control registers were not set. At T1 of this cycle, the branch portion of address 0002 is set up by using the backup X6 and X7 latches.

ROAR RESTORE LATCH OR ROAR RESTORE LATCH SX

This latch provides a gate to set X6 and X7 positions of ROAR from X6 and X7 buffer latches.

CYCLE 0002

This is the normal execution of the ROS word at address 0002.

BINARY ADD

- The binary add instruction is in the RR format with an op-code of 1A.
- The second byte of the two-byte instruction contains the addresses of two of the general purpose registers located in local storage.
- The sum is stored in the first of the two general registers specified.
- The value in the second register remains intact.
- The adding routine loops while progressing across the register values.
- After adding the last bits, the operation moves into a set condition routine to indicate overflow and sign conditions.

You have see the many parts that, put together, make up the microprogram. To tie these pieces together, let's work our way through a microprogram for a fixed point binary add.

The instruction for a binary add is written in RR format. The Op code for Fixed-Point Binary Add is 1A in hexadecimal. RR format, if you will remember, is two bytes in length. The first byte is the Op code. The second byte of the instruction consists of two general purpose register addresses in hexadecimal.

In the example you will be working through, assume that the data in general purpose register 5 must be added to the data in general purpose register 7. The instruction to accomplish this becomes

1 A 7 5

0001 1010 0111 0101.

The first byte is the Op code 1A. The last byte represents the addresses of the two registers.

Let's briefly review the addressing of a general purpose register. A register contains four bytes of data. Since only one byte of data is addressable at a time, the N-register address must be constructed by

the microprogram. As an example: To address the units position of general purpose register 7, the N-register must be set to 0111 0111. The first four bits specify the register to use. The last four bits specify a particular byte of the register.

Before starting into the program itself, you should realize some functions that must be performed by the microprogram. The program must:

1. Read the instruction, analyze the format, determine the Op code.
2. Construct addresses to set the N-register starting with the units byte of each register.
3. Add four bytes of data from register 5 to the data in register 7.
4. Check for overflow conditions after the data has been added.
5. Set the condition register to indicate the status of the resultant answer (greater than, equal to, less than zero).
6. On overflow conditions, test program masks to determine if the condition should be ignored or not.
7. Branch to I-cycles, or to another microprogram if an overflow is unmasked.

The CAS sheets as written by a microprogrammer might appear as shown in Figures 3-3, 3-6, and 3-7. The description of each ROS word that is used to execute the instruction will be made in reference to the actual address which appears in the upper right corner of each block. These facts will be assumed before starting the example.

1. The instruction is

   1 A 7 5

   0001 1010 0111 0101

2. The address of the instruction is in the IJ registers.
3. The data in register 5 is:

   Byte 0 Byte 1 Byte 2 Byte 3

   00000000 00000000 00000000 01011101

4. The data in register 7 is:

   Byte 0 Byte 1 Byte 2 Byte 3

   00000000 00000000 00000000 10011001

5. The L and S registers are zero.

Objective: The answer in register 7 as a result of the addition should be: 00000000 00000000 00000000 11110110. Using Figure 3-3 let's determine how the first function, the reading and decoding of the instruction, is accomplished. The first ROS word to be executed is at address 0100 at figure location A2. Had it been necessary to change the instruction counter or test for interrupts, a ROS word at address 0101, 0102, or 0103 would have been executed.

ADDRESS 0100 (FIGURE 3-3): The expression IJ->MN MS on the S line brings up control lines to read the first byte of the instruction from core storage. The address in the I- and J-registers is set into the MN register. Main storage is specified by MS on this line. Once core has been addressed to read out the first byte, the address in IJ can be updated for reading the next byte of the instruction. To be more explicit, only the J-register need be increased by 1 because all instructions start at an even address.

Assume that byte 01FE in main storage is to be addressed by the I- and J-registers. The I- and J-registers will then contain the address

I J

00000001 11111110.

To address byte 01FF it is only necessary to add the value of 1 to the J-register. The registers now contain:

I J

00000001 11111111.

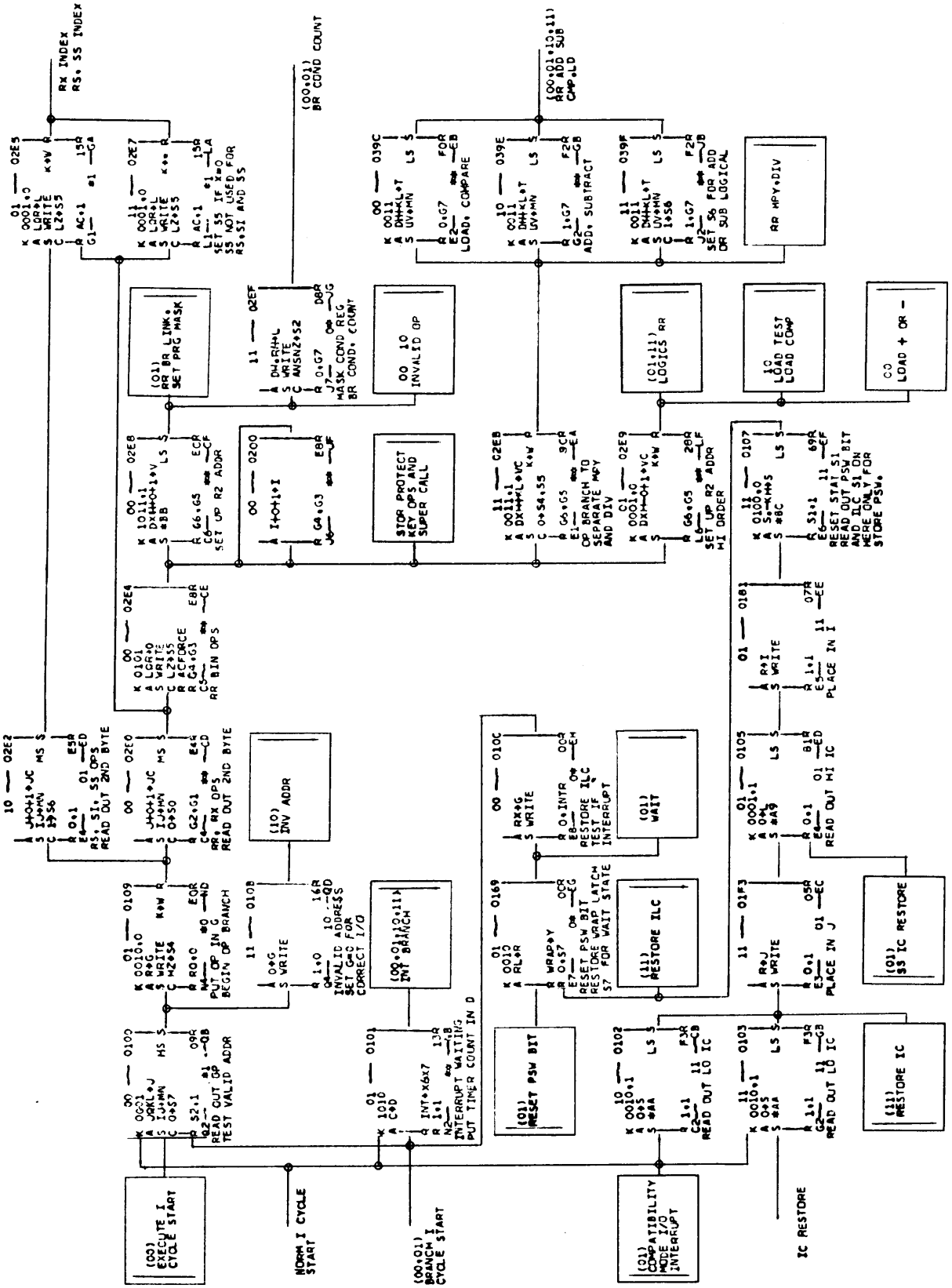


Figure 3-3. I-cycle Start (Binary Add)

Should the value of 1 be added to the J-register again, the resultant address in I and J is 00000001 00000000, or 0100. Thus, if the value 1 is added to the J-register when it is odd, it is necessary to take into account the possibility of a carryout which might affect the I-register address. Ther can be no carryout when adding 1 to the J-register address when the J-address is an even number.

The statement in the ROS word to update the J-register is J KL->J, where K has a value of one. The A-register input of ALU is the data in the J-register, and 0001 0001 is set into the B-register from the K-field. The A-register is gated directly to the ALU. Only the lower half of the B-register is gated to the ALU. The inputs are OR'ed, and the result is a 1 bit in the low-order position of the ALU, which is then gated to the J-register.

The expression on the C line is 0->S7. This statement brings up control lines to set position 7 of the S register to 0. The function performed by this statement has little bearing on our operation. It is used in an indexing routine for RX format. This brings up an important point. In any ROS word, a statement such as 0->S7 may be used that seems to have no relation to what is being done. However, it may be used further in the microprogram and should not be ignored.

The expression on the R line is S2,1. Remember that when the box format was discussed, this line was used for branching. If you look at the output line from this box, you will see that there are two ROS words that may be executed next. They are the ROS words at addresses 0109 or 010B. The expression S2,1 must somehow control a decision circuit. The convenient place to make this decision is the ROAR address itself. The two low-order positions of ROAR, X6 and X7, are controlled for branching purposes. To see how this is done, first convert the addresses of the two ROS words to binary.

```
                                        XX
                                        67
Address 0109 in binary is 0001 0000 1001
Address 010B in binary is 0001 0000 1011
```

On the R line, the left portion of the expression controls the X6 position of ROAR. To carry this one step further, the S2 portion of the expression will determine the status of X6. For this example, position 2 of the S register is zero, so the X6 position of ROAR will be set to 0. The 1 in the expression (S2,1) forces the X7 position of ROAR to a one. A 01 branch is taken to address 0109. Notice that on the top line of the box for address 0109 you see 01. These are the two low-order bits of the actual address. Had position 2 of the S register been set, X6 would be set to a one and a 1,1 branch would be executed to address 010B.

ADDRESS 0109: The first byte which was read and set into the R-register is transferred to the G-register by the expression R->G. The G-register is interrogated later in the program to determine the Op code. The data movement from the R-register to the G-register is through ALU. The output of ALU feeds the Z bus.

On line C, the expression, HZ->S4, brings up control lines that check the four high bits of the Z bus for zero. Position 4 of the S-register is set to a one if the high bits are zero. Since the data on the Z bus is:

```
          1     A

       0001  1010
```

S4 is not set. The S4 bit is interrogated in a Branch and Link routine and has no bearing on our example.

Because core readout is destructive, the information in the R-register is returned to core by the statement, WRITE.

The expression on the S line is K->W. The W-register, remember, controls the high-order positions of the ROAR address. The CK control field (K) value sets the W-register to the value shown on the K line of this block.

On the lower R line, an R0,0 branch is executed. Since R0 is off in our example, the branch is to address 02E0. This branch determines that the RR or RX format will be used.

All the ROS words until now have been at addresses 01XX. When the second high-order position of the ROS word address changes value, the W-register must be set to a new value. Since we are at address 01XX and must step to 02XX, the expression K->W is used. Notice, the K line specifies the binary value of 2. The high six bits of E0 came from the CN field of address 0109. X6 and X7 controlled the two low bits.

ADDRESS 02E0: The next byte of the instruction is read from core by the expression IJ->MN MS. This is the byte that contains the addresses of the two general purpose registers.

Once the MN registers have been set, the J-register is again updated by the expres-

sion J + 0 + 1->JC. Notice that a new element has been added to the arithmetic statement. The C to the right of the arrow allows a carryout (if there is one) to set the third position of the S-register. If no carryout results, the S3 position is set to zero. This is necessary because, should a carryout result, the I-register address portion must also be updated. At this point there is a carryout, and S3 is set to a one.

The C line of the ROS word causes the control lines to set S0 to zero. The 0 position of the S-register is a control for true or complement add when the arithmetic operation is undetermined (±). If S0 is zero, the arithmetic operation is a true add. If S0 is a 1, the operation is complement.

The G-register positions 2 and 1 are interrogated by the expression G2, G1. The data in the G-register is:

```
1     A

0001  1010
```

Finding that G1 = 0, and knowing that G0 = 0, we find that this Op code must be in RR format. RX Op codes begin with 01, RS with 10, and SS with 11.

Because G2 and G1 are both zero, a 00 branch is taken to address 02E4.

ADDRESS 02E4: The data in the R-register is again returned to core by the WRITE expression.

The arithmetic expression LΩR->D will OR the data in the L- and R-registers and transfer the resultant answer to the D-register. The symbol for the OR function is the omega. The L-register is always zero on entering I-phase except for the EXECUTE Op code. Since the L-register is zero and the R-register contains the second byte of the instruction, the D register is set 01110101.

The low-order four bits of the Z bus are checked for a zero condition by the expression LZ->S5. S5 is set to a 1 if the data on the low portion of the Z bus is zero. Because the data on the low portion of the Z bus is 0101, S5 is not set to one.

A test is made on other positions of the G-register to further decode the instruction. Looking at Figure 3-4, we see that by checking G4 and G3 our Op code must now be add, subtract, multiply, or divide. The G4 and G3 positions should set X6 and X7 to 11, the low order bits of address 02EB, but the AC-force condition overrides this and forces a branch to address 0200.

ADDRESS 0200: The only function performed by this word is I+0+1->I. This function updates the I register at the times when there is an address carry from the J-register. After updating the I-register, the microprogram again branches on the G4 and G3 positions to address 02EB.

At this time, check the data in the registers. The D-register contains 0111 0101, which is the specification for registers 7 and 5. The G-register contains 0001 1010, the Op code. The S3 bit is set to a 1, all other positions of the S register are still zero.

ADDRESS 02EB: Before any data from the general purpose registers can be added, the microprogram must set up the address of each register. The address for the low-order byte of general purpose register 5 is set up by the expression DXH + KL->VC. See Figure 3-5. To address the low-order byte, the N-register must be set to:

```
Reg 5 Byte 3
0101  0011.
```

In the expression DXH + KL->VC, consider the DXH portion first. The A-register input of ALU is set with the data from the D-register. The data in the A-register is now:

```
High  Low
0111  0101.
```

Next, the output from the A-register is crossed (X) so that the data is:

```
High  Low
0101  0111
```

The data is further controlled by the H. The H specifies that only the high portion of the data is to be used as A source data. The A source data to ALU then becomes, 0101 0000.

The B source input to ALU is controlled by the KL portion of the expression. K represents a value in the CK ROS control field. The constant is 3 and is shown in binary form on the K line of this CAS block. The B-register is set with the data 0011 0011. Only the low portion (L) of the B-register data is gated to ALU. The B source data is 0000 0011.

		RR				RX				RS		INV		SS			
		FIXED		FLT. PT.		FIXED		FLT. PT.						HI OPS		LO OPS	
	BITS	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	↓ → 0123																
	4567	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0	0000		Load Positive	Load Positive	Load Positive	Half Store	Store	Store D	Store S	Set Sys Mask	Store Multiple						
1	0001		Load Negative	Load Negative	Load Negative	Load Address					Test Under Mask				Move Numeric		Move With Offset
2	0010		Load & Test	Load & Test	Load & Test	Store Char.				Load PSW	Move Char.				Move		Pack
3	0011		Load Complement	Load Complement	Load Complement	Insert Char.				Diagnose					Move Zone		Unpack
4	0100	Set Prog Mask	AND	Halve	Halve	Execute	AND			Present	AND				AND		
5	0101	Branch & Link	Compare Logical			Branch & Link	Compare Logicql			Accept	Compare Logical				Compare Logical		
6	0110	Branch on Count	OR			Branch on Count	OR			Branch X HI	OR				OR		
7	0111	Branch on Cond	XOR			Branch on Cond	XOR			Branch X LO-EQ	XOR				XOR		
8	1000	Set Tag	Load	Load	Load	Half Load	Load	Load D	Load S	Shift RT S L	Load Multiple						Zero and Add
9	1001	Insert Tag	Compare	Compare	Compare	Half Compare	Compare	Compare	Compare	Shift Left S L							Compare
A	1010	Monitor Call	Add	Add N D	Add N S	Half Add	Add	Add N D	Add N S	Shift RT S A							Add
B	1011		Subtract	Sub N D	Sub N S	Half Sub	Sub	Sub N D	Sub N S	Shift Left S A							Subtract
C	1100		Multiply	Mult D	Mult S	Half Multiply	Mult	Mult D	Mult S	Shift RT D L	Start I-O				Translate		Multiply
D	1101		Divide	Divide D	Divide S		Divide	Divide D	Divide S	Shift Left D L	Test I-O				Translate and Test		Divide
E	1110		Add	Add U D	Add U S	Convert to Dec	Add Logical	Add U D	Add U S	Shift RT D A	Halt I-O				Edit		
F	1111		Subtract Logical	Sub U D	Sub U S	Convert to Bin	Subtract Logical	Sub U D	Sub U S	Shift Left D A	Test Channel				Edit and Mark		

Figure 3-4. Op Codes

The result of adding the B source data to the A source data is set into the V-register.

```
  A source data    = 0101  0000
+ B source data    = 0000  0011
  ------------------------------
                     Reg 5 Byte 3
  V-register data  = 0101  0011
```

The C-line of the block insures that the S-register S4 and S5 positions are blank (0) before proceeding.

The S-line in this block has the statement K->W. Since the K-field contains a three, the next address to be used will be 03XX.

The R-line is a branch on G6 and G5. This branch further breaks down the Op code and for this operation, the branch is to address 039E indicating an add or subtract Op.

ADDRESS 039E: The arithmetic statement DH+KL->T sets up the units address of register 7 in the T-register. Again consider the first portion of this expression, DH.

The A-register is set with the data in the D-register:

```
        High  Low
        0111  0101.
```

Only the high portion (H) is presented to ALU. A-source data is therefore 0111 0000. Again, the expression KL brings up the control lines to use the CK field constant of 3. The B source data is 0000 0011 because only the low portion (L) is gated to ALU.

```
  A source data    = 0111  0000
+ B source data    = 0000  0011
  ------------------------------
                     Reg 7 Byte 3
  T-register data  = 0111  0011
```

The expression UV->MN LS addresses core to read out the first byte from general purpose register 5. The data read out is 01011101. Local storage, rather than main storage, is specified by the LS portion of the expression.

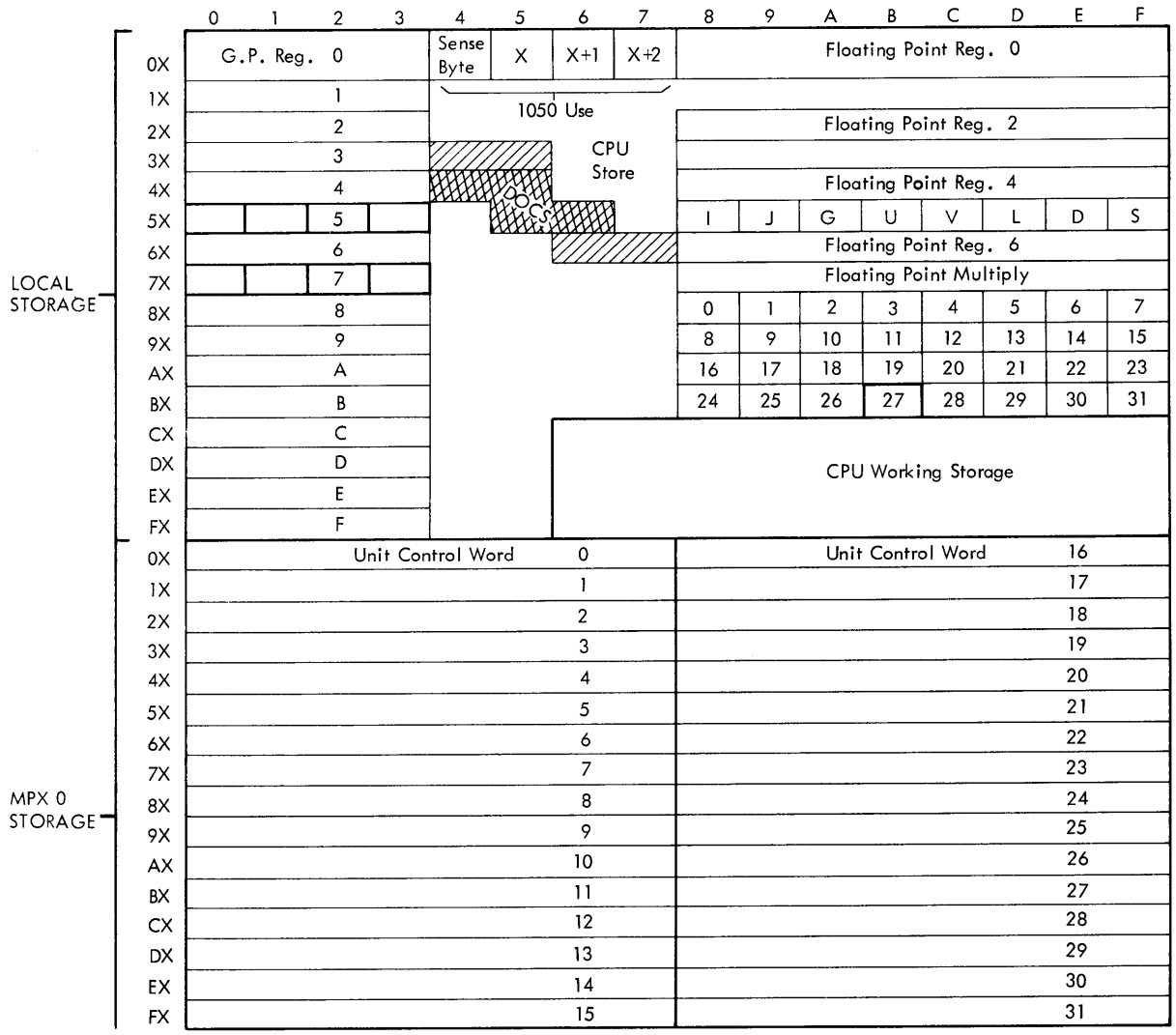


Figure 3-5. Auxiliary Storage Map

The last bit of the Op code is checked by the expression 1, G7. Since G7 is a zero, a branch is executed to address 03F2. The microprogram has fully decoded the G-register data to determine that the Op code must be a Fixed-Point Binary Add in RR Format. While this was being done, we have been setting up register addresses and even read our first byte of data from register 5.

ADDRESS 03F2 (FIGURE 3-6): The first byte of data from general purpose register 5 must be stored before reading any data from register 7. This is done by the expression R->DC. The information in the D-register is no longer needed and it is replaced by the data from register 5. The first byte of data is regenerated by the expression WRITE. The "C" following the "D" will preserve a carry out of the ALU in S3 if there is one. Here there cannot be a carry so the result is a reset of S3 for future use.

The branch condition on the R-line forces a branch to address 03A3.

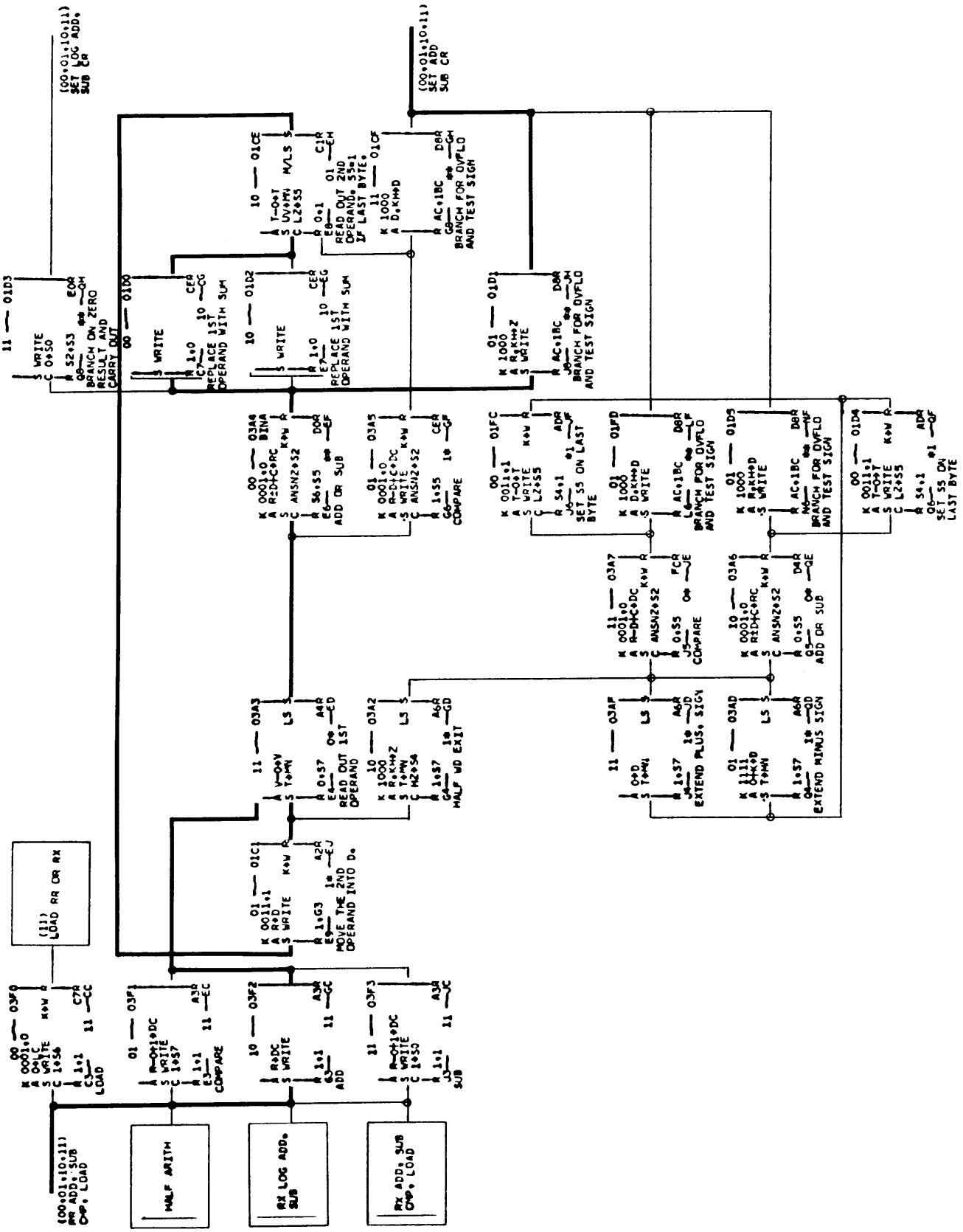


Figure 3-6. Binary Add

ADDRESS 03A3: The first byte of data from register 7 is read from core by the expression T->MN LS. The N-register is set by the data in the T-register. LS defines the core area addressed as local storage.

As this is being done, the expression V-0->V causes the value of one to be subtracted from the V-register. The V-register contains the address of byte 3 and must be changed before the next byte of data for this register is read. The example shows how ONE is subtracted by the expression V-0.

```
                      Reg 5 Byte 3
  V-register data = 0101  0011
  minus 0         = 1111  1111
  ----------------------------
                      Reg 5 Byte 2
V-register result = 0101  0010
```

As you can see, some artithmetic statements should be worked out in detail. If not, the wrong impression might be assumed from just reading the statement.

A 0,0 branch to address 03A4 is taken because S7 is still a zero.

ADDRESS 03A4: The first byte of data from register 5 and register 7 is added together by the expression R±D+C->RC. The C to the left of the arrow is a conditional carry insert. If the third position of the S-register is set to a 1, then a carry is inserted. The C to the right of the arrow allows a carry out that may result from the addition of the R and D data to set S3. The arithmetic operation, ±, is determined to be an add because the S0 position of the S-register is not set to a 1. Had it been the Subtract Op code, S0 would have been set to a 1 at ROS word address 03F3. S0 is the true or complement control position of the S-register. We know that it is a binary add rather than a decimal add because binary is specified on the K-line of the CAS block. The result of the addition is:

```
  D-register       = +01011101
  Reg 7 byte 3     = +10011001
  ----------------------------
R-register result =  11110110
```

The expression ANSNZ->S2 sets S2 to a one because the Z bus has on it the data 11110110. ANSNZ means Answer Non Zero. S2 is tested further in the program to determine whether our answer is plus, minus, or zero.

Since positions 5 and 6 of the S-register are zero, a 0,0 branch is executed to address 01D0. Because we are again changing the second high digit of our address, the expression K-->W is used. This time the value of K is 1 as shown on the K-line.

ADDRESS 01D0: The sum of the first byte from each register is regenerated (WRITE). The data in byte 3 of register 7 is now 11110110. An unconditional 1,0 branch is executed to address 01CE.

At this time, again review the location of the data and addresses in the registers.

1. The V-register contains Reg 5 Byte 2
   0101 0010

2. The T-register contains Reg 7 Byte 3
   0111 0011

3. Register 7 byte 3 data is 1111 0110.

4. The S2 position of the S-register is set to one.

5. The G-register contains 0001 1010

ADDRESS 01CE: The second byte of data from register 5 is read by the expression, UV->MN M/LS. M/LS can be either main core (M) or local storage (LS). This portion of the expression further checks the G- register. Since the G-register determines that the Op code is in RR format, only the control lines for local storage are brought up. The second byte of data read from register 5 is 000000000.

While register 5, byte 2 is read, there is no reason why the address of the next byte from register 7 cannot be set up. This is done by T-0->T, which subtracts one from the data in the T-register. The resultant answer in the T-register is:

```
Reg 7 byte 2
0111  0010.
```

The expression LZ->S5 does not set S5 to a one at this time. LZ is a check for zero on the four lower bits on the Z bus as a result of the arithmetic statement T-0->T. These four lower bits will not be zero until the last address of register 7 is obtained.

```
Reg 7 Byte 0
0111  0000.
```

A 0, 1 branch is taken to address 01C1.

ADDRESS 01C1: The byte of data just read is regenerated (WRITE). This data is also stored in the D-register, R->D. The D-register now contains 000000000, or byte 2 of register 7.

A 1,1 branch is executed to address 03A3. Position 3 of the G-register is a 1 because the Op code stored there is Add.

ADDRESS 03A3: Entering this address for the second time starts a loop in the microprogram. The loop continues until the four bytes of data from the two registers are added together.

The second byte from register 7 is read, (T->N LS).

The V-register address is changed to:

```
Reg 5 Byte 1
0101  0001.
```

A 0, 0 branch is executed to address 03A4 because S7 is still zero.

ADDRESS 03A4: The second byte of data from both registers is added and the result is stored in the R-register (R±D+C->RC). The result of this second addition is 000000000. S2 is not set to a zero (ANSNZ->S2) even though there is nothing on the Z bus because the S-register is not made up of polarity hold latches. It takes a definite reset expression to clear an S-register position to zero (0->S0).

S6 and S5 are again tested to determine the branch set up. Neither position has been set to one, therefore, the 0, 0 branch is again taken to address 01D0. K->W sets the W-register of ROAR to the value of one because of the high-order address change.

ADDRESS 01D0: The second byte of the added data is regenerated (WRITE). A 1, 0 branch is taken to address 01CE.

ADDRESS 01CE: Byte 1 of general purpose register 5 is addressed (UV->MN M/LS). The address for byte 1 of register 7 is set up (T-0->T).

S5 is still not set to a one because the data on the Z bus is:

```
Reg 7 Byte 1
0111  0001
```

Take the 0,1 branch to 01C1.

ADDRESS 01C1: Byte 1 from register 5 is stored in the D-register (R->D). It is also regenerated (WRITE). K->W is again used for the address change. Take the 1,1 branch to address 03A3.

ADDRESS 03A3: Byte 1 from register 7 is read (T->N LS). The address for general purpose register 5, byte 0 is obtained (V-0->V). Branch 0,0 to address 03A4 because S7 is still zero.

ADDRESS 03A4: Add byte-1 data from both registers (R±D+C->RC). S2 is still set to 1 and cannot be reset by the expression ANSNZ->S2. S6 and S5 are still zero. Branch to Address 01D0.

ADDRESS 01D0: Regenerate (WRITE) the sum to core. Branch 1,0 to address 01CE.

ADDRESS 01CE: Read the last byte of data from register 5 (UV->MN M/LS).

Change the address in the T-register (T-0->T).

```
                          Reg 7    Byte 1
Old T-register address  = 0111   0001
minus 0                 = 1111   1111
-----------------------------------------
                            Reg 7  Byte 0
New T-register address  = 0111   0000
```

The information on the Z bus as a result of the arithmetic statement is 0111 0000. The low-order four bits are 0000. The C-line of the box has the expression LZ->S5. S5 is now set to a 1 because the low position of the Z bus is zero (LZ). Advance to address 01C1.

ADDRESS 01C1: Store the last byte of data that came from register 5 (R->D). Regenerate this data (WRITE). Control the address change (K->W). Again check G3, set up a 1,1 branch to address 03A3.

ADDRESS 03A3: Address core and read the last byte of data from register 7 (T->N LS). Subtract one from the data in the V-register. This address, 0101 1111, is invalid for register 5 but it will not be used as we are in this loop for the last time. Check S7, which is still zero, and branch 0,0 to address 03A4.

ADDRESS 03A4: Add the last byte of data from both registers and store the result in the R-register (R±D+C->RC).

S2 is still set and cannot be reset by the expression ANSNZ->S2. K->W sets up an address change. Positions 6 and 5 of the S-register are tested. S5 had been set to a 1, therefore, a 0, 1 branch is taken to address 01D1.

ADDRESS 01D1: The last byte is stored in core (WRITE). Register 7 now contains the answer: 00000000 00000000 00000000 11110110. The data in the R-register for the last sum is 00000000. In the expression R • KH->Z, this data in the R-register is ANDed (•) with the K source high (H) and gated to the Z bus. The value of K on the K-line is eight (1000).

```
R data                  = 00000000
Constant                = 10000000
-------------------------------
ANDed result            = 00000000
```

On the branch line, R, you see the two mnemonics AC and 1BC. AC (ALU carry) brings up control lines to test for a carryout condition of ALU as a result of the arithmetic expression executed in the previous ROS word (Address 03A4, expression R±D+C->RC). 1BC (one-bit carry) brings up the control lines to test for a carry into the highest position of ALU as a result of the previous arithmetic statement. To show the positions of ALU effected, assume this data:

```
A-register  =  0100        0000

B-register  =  1100        0000
-------------------------------
               11 carries

ALU output  =  0000        0000

AC
1BC
```

The previous expression R±D+C->RC added the last two bytes of data from both registers. Both bytes of data were zero, therefore the output from ALU was zero. We had neither an ALU carry nor a 1-bit carry. A 0,0 branch is taken to address 01D8 which is in Figure 3-7.

The AC and 1BC mnemonics test to determine overflow conditions. An overflow condition would have caused branching to either address 01D9 or 01DA.

ADDRESS 01D8: The expression * BB LS addresses a byte of local storage. See Figure 3-5. This byte contains the condition code and program mask bits. Certain bits are set according to whether the answer is equal to, greater, or less than zero, plus overflow conditions.

Program masks are checked further in the program.

To address a byte in local storage, the N-register format is:

```
N0  N1  N2   N3  N4  N5  N6  N7
1,  0,  CN0, K0, 1,  K1, K2, K3.
```

The N0 and N1 positions are set 1, 0 respectively. The 2 position of the N-register is set by the CN ROS control field, 0 bit position. Position N3 is set by the CK ROS control field 0 position. N4 is set to a 1 unconditionally. N5, N6, and N7 are set by the remaining positions of the CK field. Figure 3-5 shows that the coordinates BB address byte 27. BB in binary is:

```
  B      B
1011   1011
```

Match this with the address format, and you see that the CK field must be coded 1011. This is the value that appears on the K line of the CAS block.

```
Format = 1, 0, CN0, K0, 1, K1, K2, K3,
Value BB = 1011  1011
```

The control line expression 0->S0 sets S0 to zero in case we had been in a complement operation.

The branch tests are S2 and Z = 0.

S2 was set to a 1 because we had significant data. The expression Z = 0 checks the Z bus for a zero as a result of the previous expression on the arithmetic line (R•KH->Z). Had the resultant answer been minus, the expression R•KH->Z would have provided a 1-bit output for the highest position of ALU. Because our answer is positive, the Z bus is zero and a 1,1 branch is executed to address 01EB. Had our answer been minus, the Z = 0 expression would have set X7 to a 0. A 1,0 branch would have taken us to address 01EA.

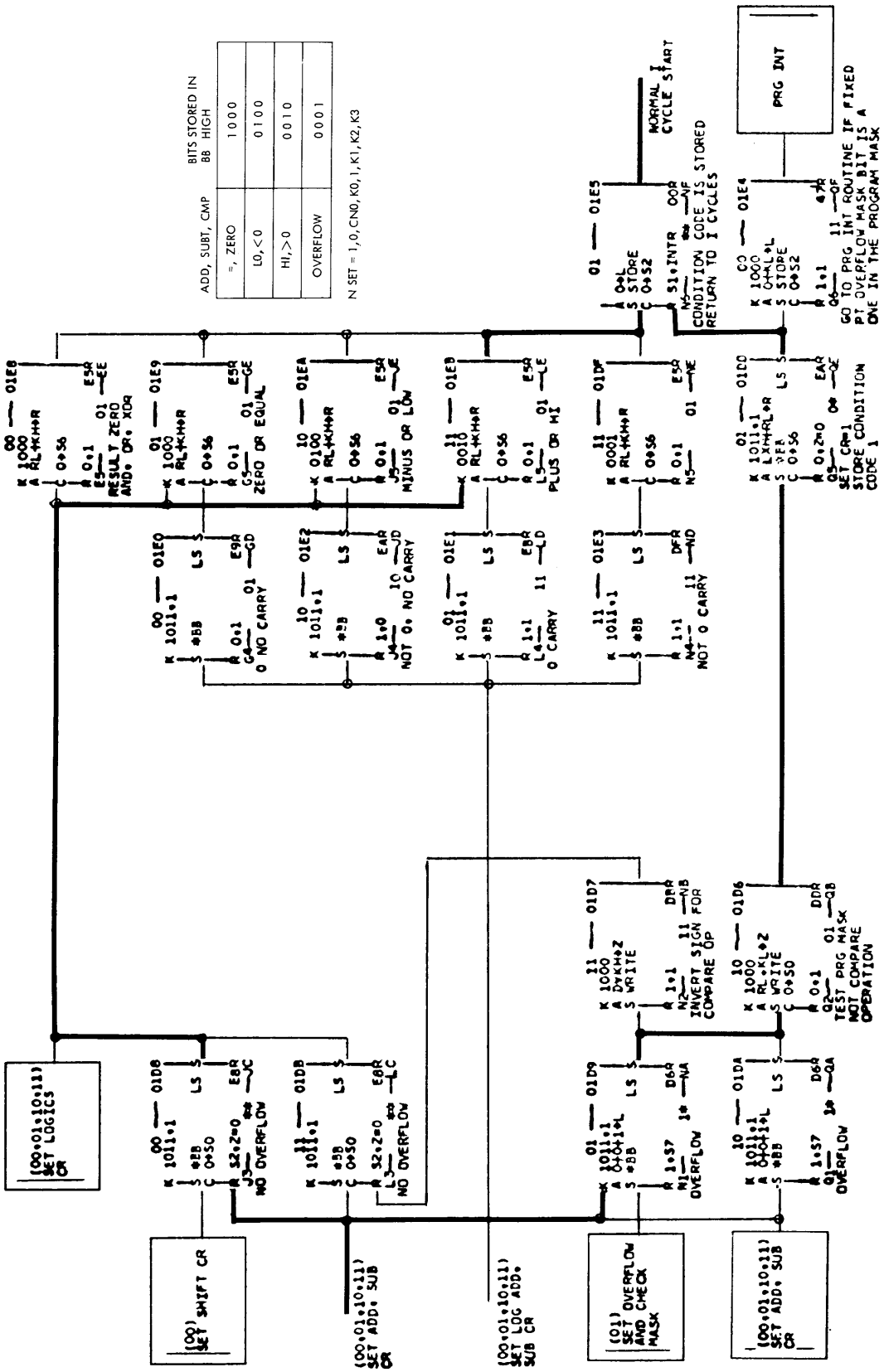


Figure 3-7. Set Condition Register

ADDRESS 01EB: The byte just read from core contains information pertaining to condition codes and program masks. The high 4-bit positions are for the condition code settings according to the answer of the problem. The expression RL + KH->R presents the data previously read to the A-register. A constant (K) sets the B-register to the value of 2 as specified by line K. The low portion (L) of the data in the A-register is used as the A source for ALU. The high position (H) of the B-register data is used as the B source for ALU. The data is then:

```
A source        0000  xxxx  x are the
                            program
                            mask bits
B source        0010  0000
--------------------------------

R-register set  0010  xxxx
```

The four high bits (0010) when returned to core signify that our resultant answer was greater than zero. Note the table in Figure 3-7.

The expression 0->S6 sets S6 to a zero.

A 0,1 branch is taken to address 01E5.

ADDRESS 01E5: The mnemonic, STORE, returns to core the information that is in the R-register. This consists of a new condition code, and the original program mask bits.

The L-register is set to zero by the expression 0->L.

Position S2 of the S-register is set to zero (0->S2).

The branch mnemonics test S1 and interrupt. S1 is not set. If there is an interrupt, a 0, 1 branch is taken to address 0101, Figure 3-3. If no interrupt exists, a 0,0 branch is executed to address 0100, Figure 3-3. Address 0100 is the ROS word where this operation began. Address 0101 is the beginning of a microprogram to test the interrupt and determine what it is (selector channel, multiplex, etc.).

This completes the Add operation. However, to carry the microprogram one step further, assume that the result of adding the two registers produced an overflow condition. An overflow makes it necessary to set a different condition code before returning to I-cycles (see chart insert on Figure 3-7). The problem program can, at some later time, branch on the condition code set.

Start at address 01D9 in Figure 3-7.

ADDRESS 01D9: The local storage byte is read out by *BB LS. The L-register is set to 00000001 (0+0+1->L).

Since S7 is not set to a 1 a 1,0 branch is taken to address 01D6.

ADDRESS 01D6: The data just read consists of the condition code and program mask bits. It is regenerated to core (WRITE). The arithmetic expression RL•KL->Z tests the program mask bits by allowing or preventing a bit on the Z bus. Assume the data in R is xxxx 0yyy. The x positions are those for the condition code. The 0 means that this position is not set. The y positions are the remaining program mask bits.

The data for this expression is:

```
A source data (RL) =   0000  0000
B source data (KL) =   0000  1000
--------------------------------

ANDed ALU output   =   0000  0000
```

Position zero of the S-register is set to zero (0->S0).

ADDRESS 01DD: Again, the same byte of information is read by the expression: *BB LS.

The condition code is set by the expression LXH+RL->R. The data in the L-register sets the A-register. The A-register now contains:

```
            High   Low

            0000   0001
```

This is crossed (X)

```
            High   Low

            0001   0000
```

and only the high (H) portion is used for the A source to ALU. The data in the B-register is xxxx 0yyy. Only the low portion is presented to ALU (RL). The result of the addition becomes:

```
A source =     0001   0000
B source =     0000   0yyy
--------------------------

R-reg set =    0001   0yyy
```

The C-line statement, 0->S6, sets position 6 of the S-register to a zero.

The branch conditions are 0 and Z = 0. Z = 0 brings up control lines to check the Z bus as a result of the arithmetic statement executed in the previous ROS word. This is how the program mask condition is checked. Our output was 0000 0000 as a result of the expression RL • KL->Z. Therefore, Z = 0 sets X7 to a 1. A 0,1 branch is taken to address 01E5 because the overflow was masked off.

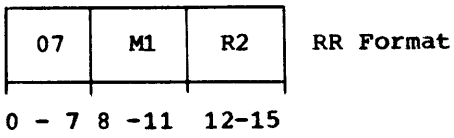
ADDRESS 01E5: The data in the R-register is returned to core (STORE). The four highest bits are the new condition code:

High	Low
0001	0yyy

0001 is the coding for an overflow. The L-register is set 0000 0000 by the expression 0->L. The S2 position is set to zero and the branch conditions are S1 set to zero and no pending interrupt. This branches the microprogram to address 0100 on Figure 3-3.

BRANCH ON CONDITION (RR FORMAT)

- The updated instruction address is replaced by the branch address (R2) if the bit that is on in the condition code is also on in the mask.
- The M1 field is used as a four-bit mask.
- Normal instruction sequencing proceeds with the updated instruction address if the condition is not met.



The branch will be successful whenever the condition code has a corresponding mask bit of one. The four bits of the mask correspond with the condition code as follows:

Condition Code Bit	Op Code Bit (Mask)
0	8
1	9
2	10
3	11

Description of a Branch-on condition instruction, RR format, with reference directly to the microprogram follows. Assume the previous instruction was a fixed point add which set bit 3 of the condition code, indicating an overflow. A Branch-on condition is issued with bit 11 of the Mask field at one.

Starting at ROS address 0100 Figure 3-8, the Branch-on condition Op code is read out. The Op code is placed in the G-register at ROS address 0109.

ADDRESS 02E0: The second byte of the instruction is read out and a branch on G-register bits G2 and G1 is taken. This determines that the operation is in RR format.

The second byte is placed in the D-register and S5 is set to 0 if the R2 field is not zero. If the R2 field had been zero, S5 would have been set to one and a no-branch condition would have resulted at address 02D9 (Figure 3-9).

ADDRESS 02E8: The R2 portion of the instruction is placed in the V-register in the event the branch address must be fetched. The condition code is read from local storage. A branch to 02EF is made by testing G-register bits 6 and 5.

ADDRESS 02EF: The mask portion of the instruction is ANDed with the condition code and the result is placed in the L-register. S2 is set to one if the result was not zero. A branch is made on G-register bit 7 which results in going to address 02D9 (Figure 3-9) .

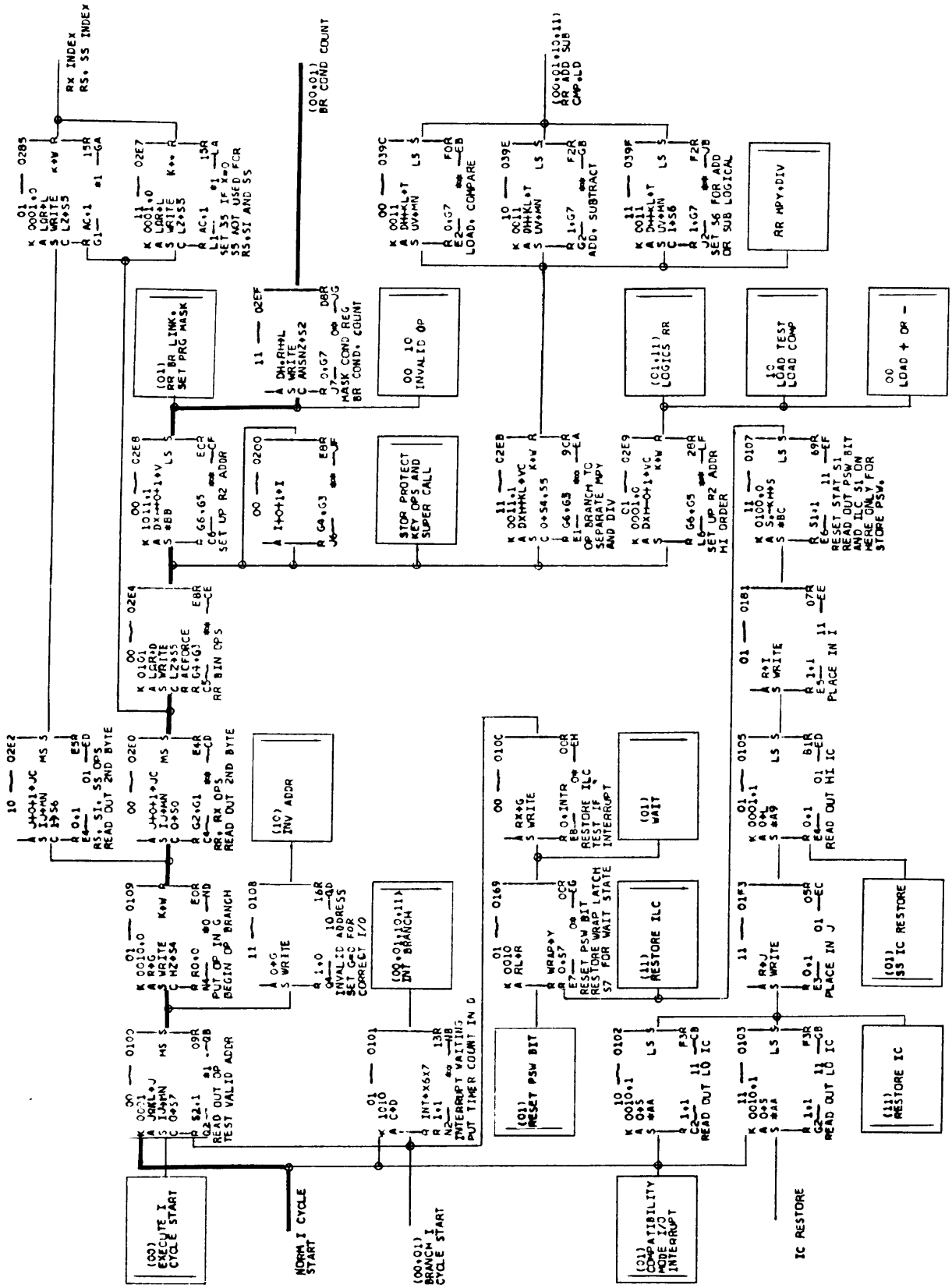


Figure 3-8. I Cycle Start (Branch On Condition)

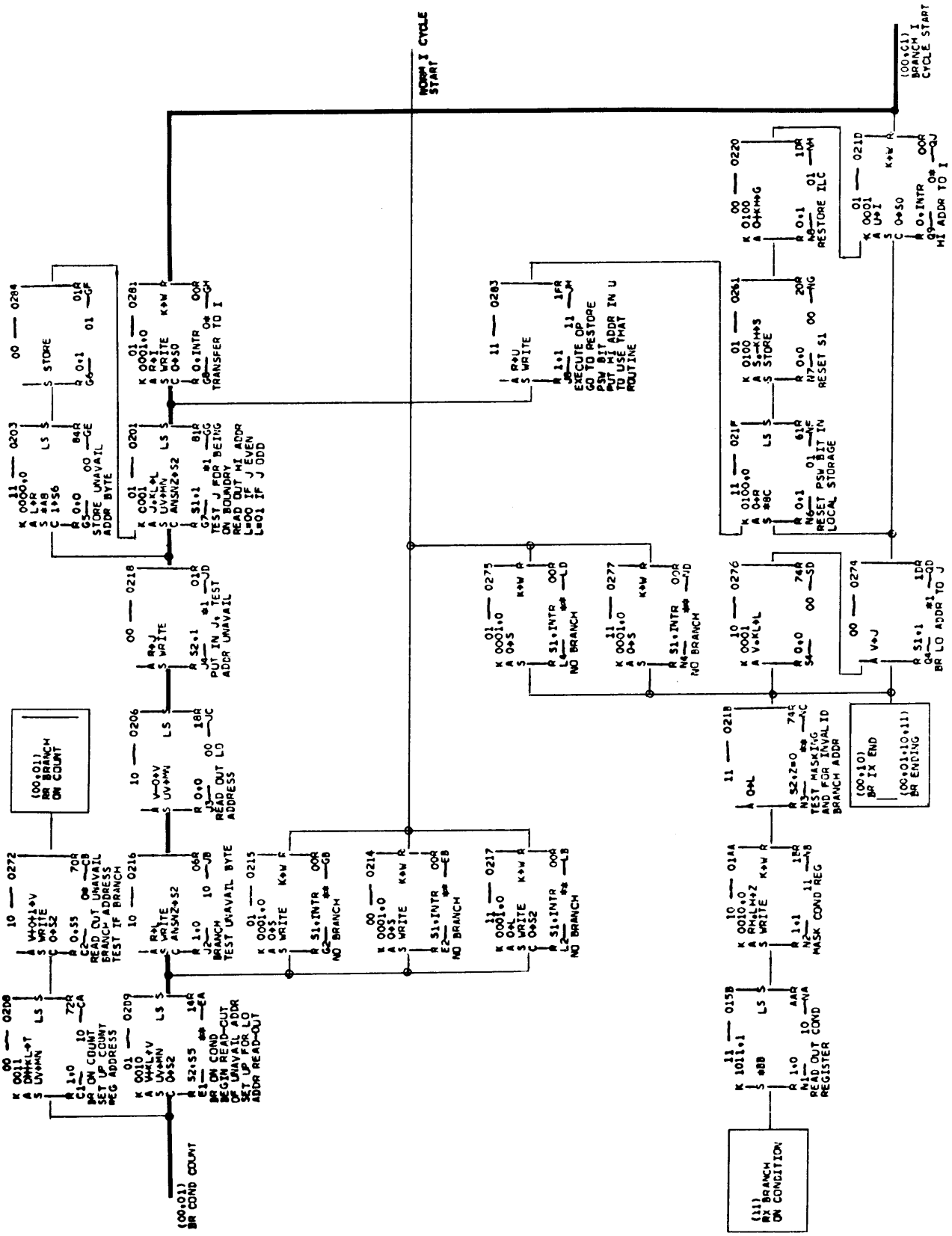


Figure 3-9. Branch On Condition

ADDRESS 02D9: S2 and S5 are tested to determine if a branch will be taken. If S2 is one (indicating condition code and mask have matched) and S5 is zero (indicating R2 not zero), the conditions for branch have been satisfied. At address 02D9, the second byte of the specified general register is read out, and the V-register is set to address the low-order byte of the specified general register.

ADDRESS 0216: The 2nd byte of the general register is set into the L-register and the S2 bit is set to one if this byte was not zero. This is done to check for an unavailable address request.

ADDRESS 0206: The low byte of the general register is read out, this is the low-order branch address.

ADDRESS 0218: The low-order branch address is placed in the J-register. A test of S2 is made here for a possible unavailable address. In this example correct operation is assumed.

ADDRESS 0201: The third byte of the specified general register is read out. This is the high-order branch address. The J-register is tested for being on boundary, and the S2 bit is set to one if not on boundary, or zero if on boundary.

ADDRESS 0281: The high-order branch address is placed in the I-register. The branch is made back to I-cycle start where the next instruction read out of storage will be taken from the branch address that the IJ registers now contains.

PACK WITH INDEXING

You have gone through the I-cycle section for a binary add and a branch on condition operation. These two instructions were of the RR format and did not require any indexing. So, let's take an instruction which requires modification of the main storage address in order to set up the address of the data field. For an example, the PACK instruction (F2) of the SS format is used.

The pack instruction format has 6 bytes;

byte 1 - Op code

byte 2 - field length count for first and second operand

byte 3 - the general register number containing the base address of the first operand and the high four bits of the first operand displacement address byte 4 - eight bits of first operand displacement address

byte 5 - the general register number containing the base address of the second operand and the high four bits of the second operand displacement address

byte 6 - eight bits of the second operand displacement address.

The Op code for the pack instruction is F2. For the example to explain the instruction, the first operand field length is 3 and the second operand field length is 4. The base register for the first operand is general register 2 and the second operand is general register 4. The displacement for the first operand is 060 and for the second operand is 040. The instruction put together is: F2 34 20 60 40 40. The first operand base register contains 00 00 0A 48 and the second operand base register contains 00 00 15 BB. The first operand data is DB F4 F2 C0 and the second operand data is F7 F8 F2 F6 C3.

The objective of the I-cycles, in addition to decoding the Op code, is to set up the low-order main storage addresses of the first and second operand bytes. The objective of the execute-cycle is to remove the zone bits from each byte in the second operand and pack the numeric bits of the bytes. The pack bytes are set into the first operand locations with the low-order byte containing the sign in the low four bits. When this example is finished, the result 0078 263C will be in main storage starting at address 0AA8.

Starting into I-cycles the S- and L-Registers have been set to 00. The I-Register is A0 and the J-Register is 40, the location in main storage where the Op code is stored. To start the microprogram start with word 0100 (Figure 3-10).

ADDRESS 0100: Set the MN-registers to A040 and read out the Op code (F2) from main storage. Increase the J-register by one to 41. The R-register is set to F2; the output of main storage. The branch statement is S2, 1 with S2 at a 0 condition giving the next ROS address of 0109.

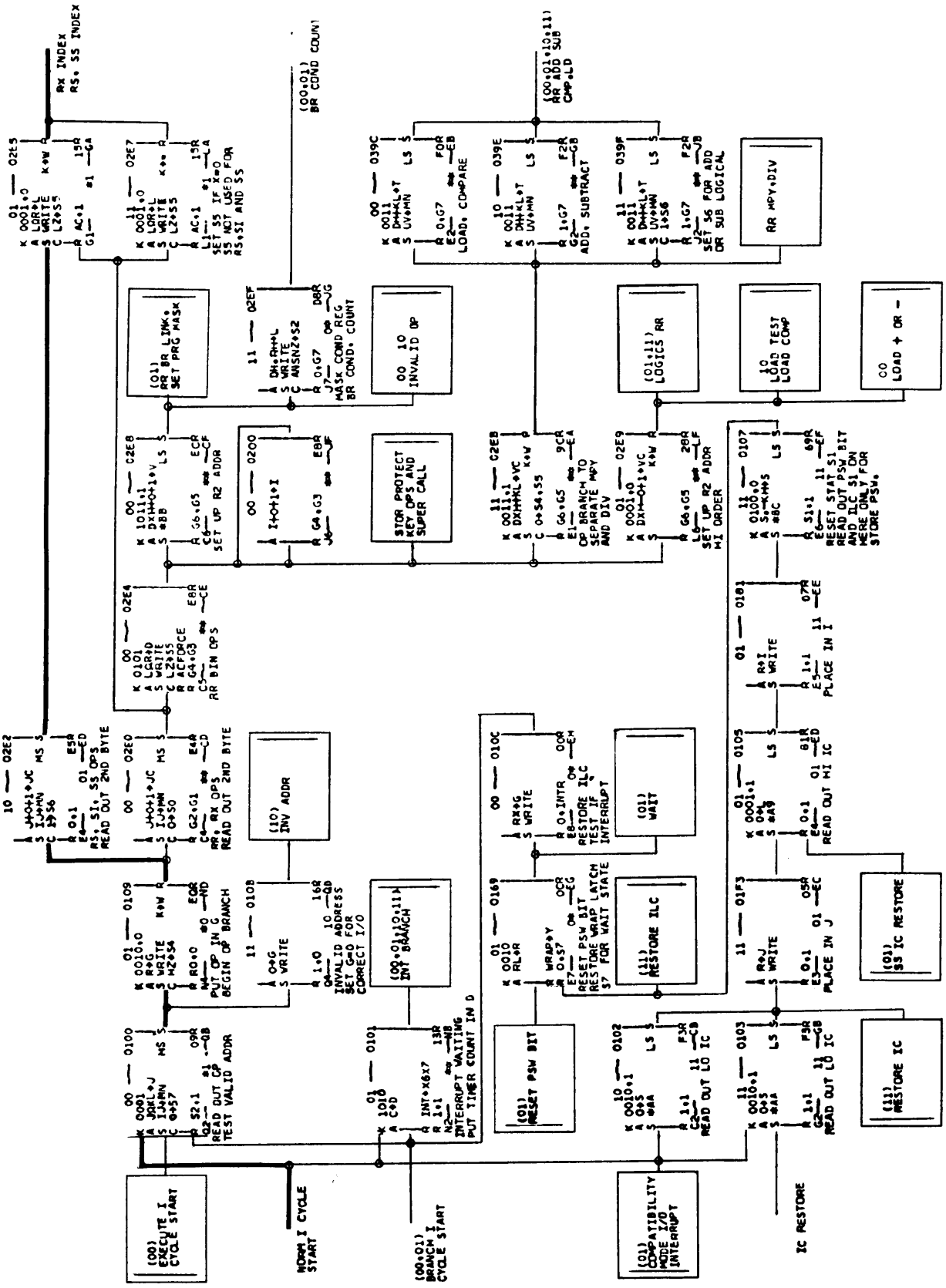


Figure 3-10. I Cycle Start (Index/Pack)

ADDRESS 0109: Set the G-Register to the value in the R-Register (F2) via the Z-bus. Write into main storage the contents of the R-Register at the same location it was read from; address A040. Test the high four bits of the Z-bus for all zeros and set S4 to a 1 if they are. The Z-bus at this time has F2 on it, so S4 is left at a 0 condition. The branch statement is R0, 0; the R-Register contains F2 so the R0 bit is a 1. This sets the next ROS address to 02E2. Remember the K->W statement set the W-Register to the value of the CK field which is two in this word.

ADDRESS 02E2: Increase the J-Register by 1 to 42; no carry occurs from the update so S3 is left at 0 condition. Set the MN-Registers to A041 and read out the second byte (34) of the instruction from main storage. Set the R-Register to the output of main storage (34). Set the S-Register bit 6, S6, to a 1 condition. Branch condition is 0, 1 so the next ROS address is 02E5.

ADDRESS 02E5: The contents of the L-Register is set into the A-Register and the contents of the R-Register is set into the B-Register. The outputs of the A- and B-Registers are ORed together in ALU. The L-Register was 00 and the R-Register has been set to 34, so the result (34) is set into the L-Register via the Z-bus. The low four bits of the Z-bus are tested for a zero condition. S5 is set to a 1 if the four bits are all zero. In the example S5 is left at a 0 condition since the Z-bus is set to 34. The branch statement is AC,1; the AC is a check of the carry latch setting from the last microprogram word. Since a carry did not occur in the last word, the condition of AC is 0. The next ROS address is 0115 (Figure 3-11 Part 1).

ADDRESS 0115: Increase the J-Register by 1 to 43. Set the MN-Register to A042 and read out the third instruction byte (20) from main storage. Set the R-Register to the output of main storage (20). Set the S-Register bit 0 to a 1 condition. The branch statement is S2,0 with S2 at a 0 condition giving the next ROS address of 0184.

ADDRESS 0184: Set the low-order byte address of the first operand base register by setting the R-Register contents into the A-Register and routing the four high bits (2) to the ALU. The B-Register is set to 33 from the CK field, but only the four low bits are routed to the ALU. The ALU inputs are 20 and 03 giving an output of 23 on the Z-bus. The T-Register is set to the output of the add (23). The four high bits of the Z-bus are tested for all zeros. If the high four bits are not all zeros, S4 is left at a 0 condition. This test was made to determine if the base register was general register 0. If general register 0 is selected as a base register, the base register amount is considered to be zero and is not added to the displacement. The branch statement is 0, S7 with S7 at a 0 condition giving a next ROS address of 012C.

ADDRESS 012C: Set the four high bits of the displacement address from the R-Register into the U-Register (00). Set the MN-Registers to A043 and read out the fourth byte of the instruction (60) and set it into the R-Register. What was in the R-Register at the beginning of a word can be used in the add statement because the output of core storage does not enter the R-Register until the end of the cycle. Set S0 to 0. The branch statement is S4,1 and S4 was left at a 0 condition giving the next ROS address as 0131.

ADDRESS 0131: Increase the J-Register by 1 to 44 leaving the S3 bit at a 0 condition. Write the fourth byte (60) into main storage at the same address it was read from. The branch statement is 10 giving the next ROS address of 012E.

ADDRESS 012E: Set the low eight bits (60) of the displacement address into the V-Register from the R-Register. Set the MN-Registers from the T-Register; the M-Register setting will change depending on the size of core storage in order to select the correct local storage area. The T-Register has been set to 23. This addresses the low-order byte of general register 2 in local storage and sets the byte (48) into the R-Register. The branch statement is 0, S3 with S3 at a 0 condition giving a next ROS address of 0138.

ADDRESS 0138: Decrement the T-Register by 1 to 22. Write the low-order byte (byte 3) into the same location it was read from. The branch statement is 1,0 giving a next ROS address of 010A.

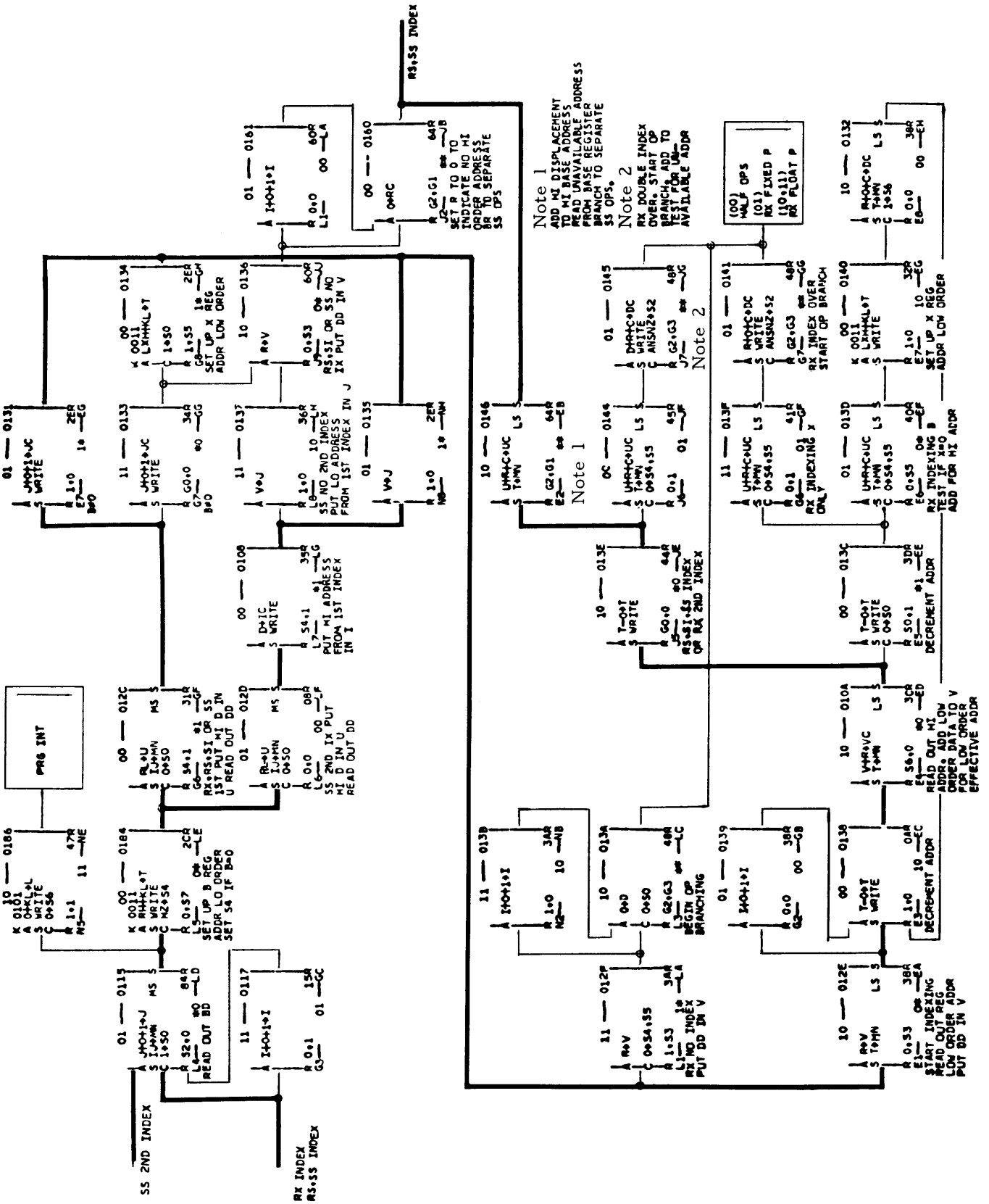


Figure 3-11. Indexing (Part 1 of 2)

ADDRESS 010A: Add the contents of the V-Register (the A-Register input) to the contents of the R-Register (the B-Register input) and set the results into the V-Register. The V-Register contained 60 and the R-Register contained 48 giving an ALU output of A8 without carry to the U-Register. Set the MN-Register from the T-Register; MN is set to XX22 where XX is determined by core storage size. Byte 2 of the general register 2 is read out from local storage and is set into the R-Register (0A). The branch statement is S6,0 with S6 at a 1 condition, giving a next ROS address of 013E.

ADDRESS 013E: Decrement the T-Register by 1 to 21. Write the second byte into local storage at the same address it was read from. The branch statement is G0,0. The G-Register has the Op code of F2, so G0 is a 1 giving the next ROS address of 0146.

ADDRESS 0146: Add the contents of the U-Register (00), (the A-Register input) to the contents of the R-Register (0A), (the B-Register input) and set the results (0A) into the U-Register. If a carry had existed from the previous add, it would be added in at this time. The MN-Registers are set from the T-Register to xx21. Byte 1 (00) is read out of general register 2 and set into the R-Register. The branch statement is G2, G1 which are at a 1 condition at this time giving a ROS address of 0167 (Figure 3-11 Part 2).

ADDRESS 0167: Add the contents of R-Register (00), (the A-Register input) to a forced output of 00 from the B-Register and a carry if one had occurred during the last add. The result of this add is set on the Z-bus to see if the result is non-zero; if result is non-zero, an address too large for a 2030 has been developed. Write byte 1 into local storage at the same location it was read from. Set S2 to a 1 condition if the result of the add was non-zero; for the example S2 stays at a 0 condition. The branch statement is S1,S7 with both bits at a 0 condition at this time giving a next ROS address of 0370.

ADDRESS 0370: Add two to the amount in the J-Register, this gives the low-order part of the main storage location of the next Op code. Set the results (46) into the R-Register. Set the MN-Registers to xxAA and read out the byte from local storage but do not set the output into the R-Register. This is blocked because the next word has a storage statement of STORE. The branch statement is 0,0 giving a next ROS address of 0384.

ADDRESS 0384: Set the contents of the U-Register (0A) into the D-Register via the Z-bus. Store the contents of the R-Register (46) into local storage location AA, byte 18. Check the WRAP latch. If it is on, set the sixth position of the X-Register to a 0. For the example, the WRAP latch is not on. The branch statement is 1,1 and the WRAP latch is not on, therefore, the next ROS address is 035F.

ADDRESS 035F: Add one to the I-Register if a carry occurred when the J-Register was increased by two. For the example a carry did not occur. Set the MN-Register to xxA9 and read out the byte in local storage. Set the core storage output into the R-Register. The branch statement is 0, S3 with S3 at a 0 condition giving a next ROS address as 03A0.

ADDRESS 03A0: Set the contents of the I-Register (A0) into the R-Register. If the WRAP latch is set now, transfer the WRAP-latch-on condition to the WRAP BUFFER latch. The branch condition is 1,0 giving a ROS address of 0306.

ADDRESS 0306: Set the 1 bit of the S-Register by use of the add statement. Set the contents of the R-Register (A0) into local storage location A9, byte 17. The branch statement is 1,0 giving the next ROS address of 032A.

ADDRESS 032A: Set the G-Register into the A-Register (F2) and set the B-Register to 44 from the CK field. Cross the output of the A-Register but only route the four low bits after the cross with the four high bits set to zeros. Only route the four high bits of the B-Register and force the four low bits to zeros. The input to the ALU is 0F from the A-Register and 40 from the B-Register. This output of 4F from the ALU is set into the R-Register. The location in local storage containing the instruction-length count and the PSW-bit-condition address is set into the MN-Registers xxBC. The information located at this position is read out but does not enter the R-Register because the next word storage statement is STORE and the R-Register is the destination register in this word add statement. The branch statement is 1,0 giving a next ROS address of 0376.

ADDRESS 0376: The statement I->I will reset the WRAP latch if on. The contents of the R-Register (4F) is set into local storage at location BC to be used later if necessary. The S-Register bit 7 is set to

a 1 condition to indicate the first operand address has been indexed. The branch statement is 0,1 giving a ROS address of 0115 (Figure 3-11 Part 1).

ADDRESS 0115: Increase the J-Register by 1 to 45. Set the MN-Register to A044 and read out the fifth byte of the instruction from main storage. Set the R-Register to the output of main storage (40). Set the S-Register bit 0 to a 1 condition. The branch statement is S2,0 with S2 at a 0 condition giving the next ROS address of 0184.

ADDRESS 0184: Set the low-order byte address of the second operand base register into the T-Register (43). Write the fifth byte into main storage at the same location it was read from. Test the Z-bus four high bits for all zeros. Since the Z-bus has 43 on it, S4 is left at a 0 condition. The branch statement is 0,S7. S7, at a 1 condition, now gives the next ROS address of 012D.

ADDRESS 012D: Set the low portion of the R-Register (the four high bits of the second operand displacement) into the U-Register. Set the MN-Registers to A045 and read out the sixth byte of the instruction (40). Set the output into the R-Register. Set the S-Register bit 0 to a 0 condition. The branch statement is 0,0 giving a next ROS address of 0108.

ADDRESS 0108: Set the I-Register to the contents of the D-Register (0A) and ensure the S3 bit is at a 0 condition. Since the last byte of the instruction has been read out, the address in the IJ-Registers at this time is not needed and the first operand address can be transferred to the IJ-Registers. The address of the next sequential Op code has already been stored. Write the sixth byte into main storage at the same location (A045) it was read from. The branch statement is S4,1 with S4 at a 0 condition because the second operand base register number is not 0. This gives the next ROS address of 0135.

ADDRESS 0135: Set the contents of the U-Register (A8) into the J-Register. The branch statement is 1,0 giving a ROS address of 012E.

ADDRESS 012E: Set the low eight bits (40) of the displacement into the V-Register from the R-Register. Set the MN-Registers from the T-Register (xx43) and read out byte 3 of the second operand base register. Set the R-Register (BB) to the output of the core storage. The branch statement is 0,S3 with S3 at a 0 condition giving a next ROS address of 0138.

ADDRESS 0138: Decrement the T-Register by 1 to 42. Write byte 3 into the same location that if was read from. The branch statement is 1,0 giving a next ROS address of 010A.

ADDRESS 010A: Add the contents of the V-Register (40) to the contents of the R-Register (BB) and set the result (FB) into the V-Register. Set the MN-Register from the T-Register (xx42) and read out byte 2 (15) from local storage. Set the output into the R-Register. The branch statement is S6, 0 with S6 at a 1 condition giving the next ROS address of 013E.

ADDRESS 013E: Decrement the T-Register by 1 to 41. Write the second byte into local storage at the same location it was read from. The branch statement is G0,0. The G-Register contains the Op code (F2) so G0 is a 1 giving a next ROS address of 0146.

ADDRESS 0146: Add the contents of the U-Register (00) to the contents of the R-Register (15) and set the result (15) into the U-Register. The MN-Register is set from the T-Register to xx41. Byte 1 (00) is read out from general register 4 and set into the R-Register. The branch statement is G2, G1. Both at a 1 condition gives a next ROS address of 0167 (Figure 3-11 Part 2).

ADDRESS 0167: Add the contents of the R-Register (00) to the forced 00 output of the B-Register and test the result on the Z-bus for an address too large for a 2030. Write byte 1 into local storage at the same location it was read from. The branch statement is S1, S7. Both at a 1 condition gives a next ROS address of 0373.

The conditions at this time are the next sequential Op code address stored in local storage. The IJ-Registers contain the address of the high-order byte for the first operand (0AA8). The UV-Registers contain the address of the high-order byte for the second operand (15FB). The S-Register bits 1, 6, and 7 are set to a 1 condition. The L-Register contains the field length count of the first and second operand.

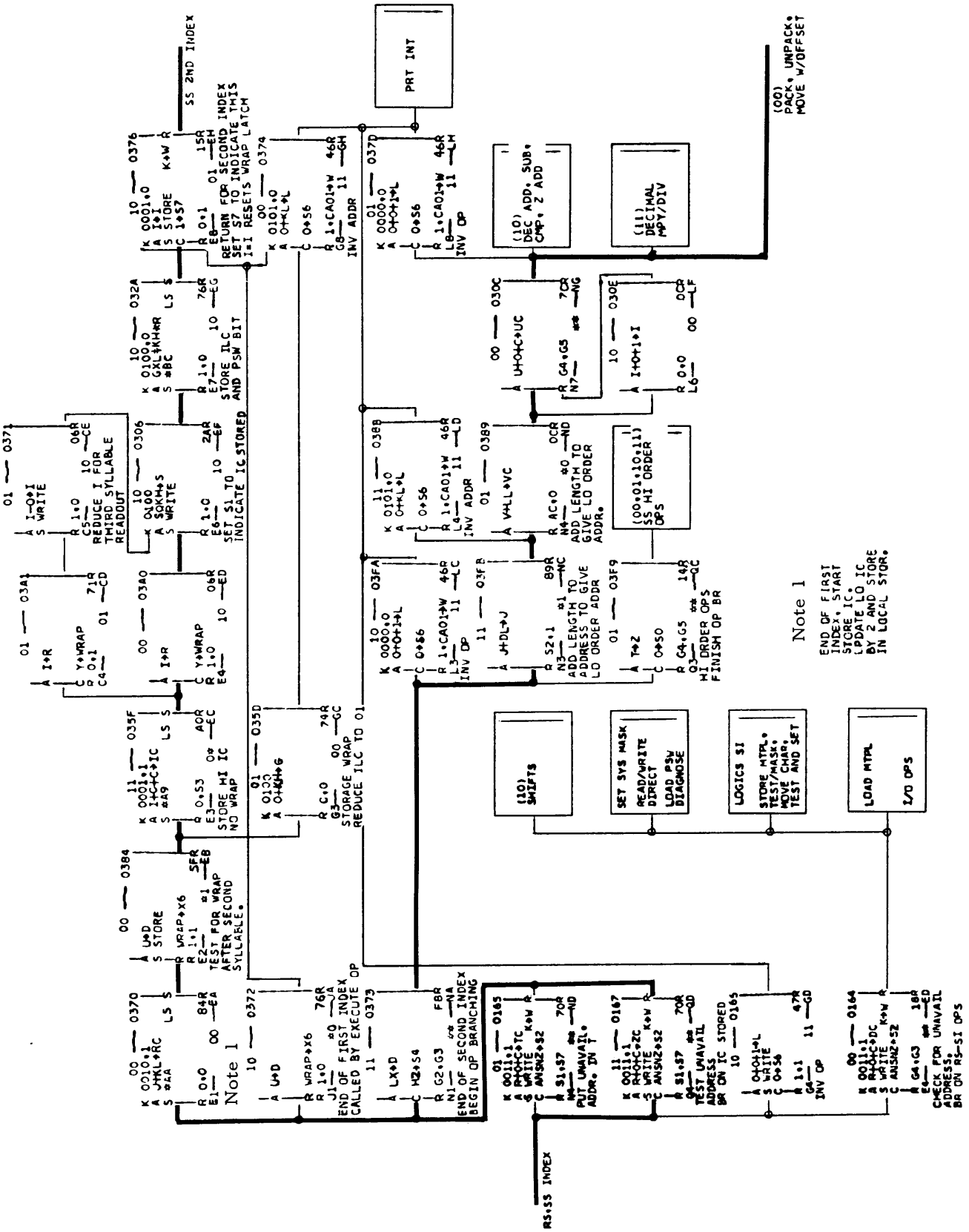


Figure 3-11. Indexing (Part 2 of 2)

ADDRESS 0373: Take the contents of the L-Register (34), cross it (43), and set the crossed amount into the D-Register. Check the four high bits of the Z-bus for all zeros. In the example S4 remains at a 0 condition. The branch statement is G2, G3. Both at a 1 condition gives a next ROS address of 03FB.

ADDRESS 03FB: Add the contents of the J-Register (A8) to the low four bits of the D-Register (03) and set the result of the add (AB) into the J-Register. The branch statement is S2,1 with S2 at a 0 condition giving a next ROS address of 0389.

ADDRESS 0389: Add the contents of the V-Register (EB) to the four low bits of the L-Register (04) and set the result (FF) into the V-Register. The branch statement AC,1 tests to see if a carry occurred in the previous word. In the example, a carry did not occur giving a next ROS address of 030C.

ADDRESS 030C: If a carry occurred in word 0389, add one to the U-Register. For the example, the U-Register remains the same. The branch statement is G4, G5 with both at a 0 condition giving a next ROS address of 037C (Figure 3-12).

ADDRESS 037C: Decrement the V-Register by 1 to FE causing a carry to occur and setting S3 to 1. Set the MN-Register to 15FF and read out the low-order byte of the second operand (C3). Set the output of storage into the R-Register. The branch statement is 1,0 giving a next ROS address of 035E.

ADDRESS 035E: Set the contents of the L-Register (34) on the Z-bus and test both the high and low four bit combinations for all zeros to determine if either field has ended. For the example here, neither have ended. Write into main storage into the same location the same information read out on the last word (C3). The branch statement is G6, G7. With G6 a 1 and with G7 a 0 giving a next ROS address of 0422.

ADDRESS 0422: Take the contents of the R-Register (C3), cross it (3C), and set the crossed number into the R-Register. This is done because the sign of the number in the unpacked field is the low-order byte zone digit and the sign is maintained. Only the sign becomes the low four bits or digit of the low-order byte in the packed field. Set the S-Register bit 7 to a 0 condition. The branch statement is 0,0 giving a next ROS address of 0438.

ADDRESS 0438: Set the D-Register to 00. Set the MN-Registers to 0AAB and read out the contents of main storage at that address but do not set the information into the R-Register. The branch statement is S4, S5 with both at a 0 condition giving a next ROS address of 0428.

ADDRESS 0428: Decrement the J-Register by 1 to AA. Store the byte in the R-Register (3C) into main storage location 0AAB. The branch statement is 0, S3 with S3 at a 1 condition from word 037C giving a ROS address of 0431.

ADDRESS 0431: Decrement both field lengths in the L-Register (34) by one and set the results (23) into the L-Register. Set MN-Registers to 15EE. Read out the next byte of the second operand (F6) and set it into the R-Register. Test the Z-bus, both the high and low four bit combination, for all zeros. In the example, the Z-bus has 23 on it at this time so S4 and S5 remain at a 0 condition. The branch statement is 1,S3 with S3 at a 1 condition from word 0428 giving a next ROS address of 0433.

ADDRESS 0433: Set the contents of the R-Register (F6) into the T-Register. Write the contents of the R-Register into main storage at the same location the information was read from. The branch statement is G6, G7 with G6 being a 1 and G7 a 0. This sets the next ROS address to 042E.

ADDRESS 042E: Decrement the V-Register by 1 to FD. The branch statement is 0, S5 with S5 at a 0 condition giving a next ROS address of 0434.

ADDRESS 0434: Decrement the second operand field length count by 1 to the number 2. Set the MN-Register to 15FD, and read out the next byte of the second operand (F2) and set it into the R-Register. Test the four low bits of the Z-bus for all zeros; the Z-bus has the L-Register input (22) on it. This leaves S5 at a 0 condition. The branch statement is AC,0; there was a carry in the last word so AC is a 1 condition giving a next ROS address of 0486.

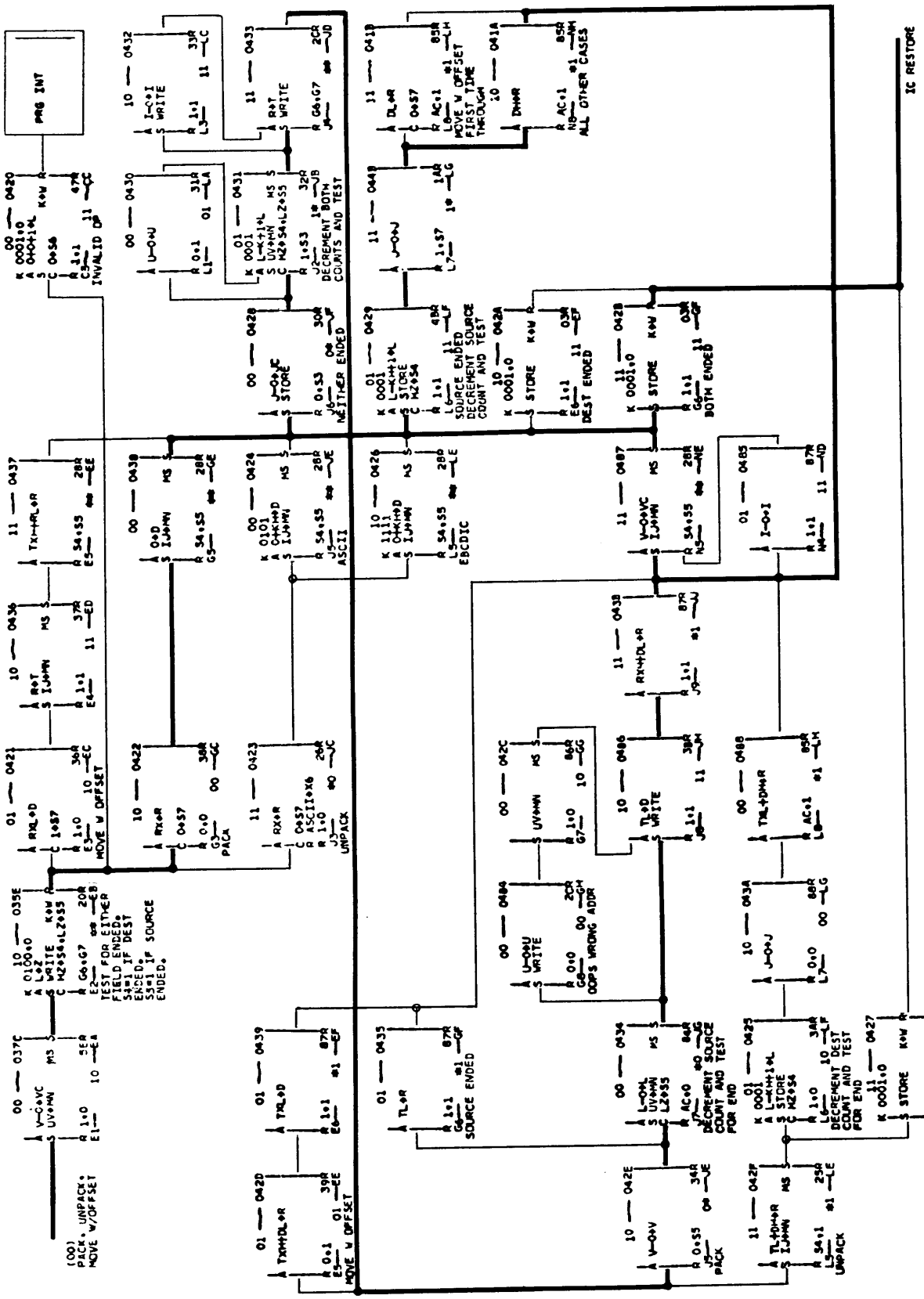


Figure 3-12. Pack

ADDRESS 0486: Set the four low bits of the T-Register (6) into the D-Register (06). The branch statement is 1,1 giving a next ROS address of 043B.

ADDRESS 043B: Set the contents of the R-Register (F2) into the A-Register and the contents of the D-Register (06) into the B-Register. Cross the output of the A-Register (2F) and route the four high crossed bits (20) to the ALU. Route the four low bits of the B-Register (06) to the ALU. Set the output of ALU (26) into the R-Register. The branch statement is 1,1 giving a ROS address of 0487.

ADDRESS 0487: Decrement the V-Register by 1 to FC. Set the MN-Registers to 0AAA and read out that main storage byte but do not set the byte into the R-Register. The branch statement is S4, S5 with both at a 0 condition giving a ROS address of 0428.

ADDRESS 0428: Decrement the J-Register by 1 to A9. Store the byte in the R-Register (26) into main storage location 0AAA. The branch statement is 0, S3 with S3 at a 1 condition from word 0487 giving a ROS address of 0431.

ADDRESS 0431: Decrement both field length counts in the L-Register (22) by one to 11 and set into the L-Register. Set MN-Register to 15FC and read out the next byte of the second operand (F8) and set it into the R-Register. Test the Z-bus, both high and low four bits, for all zeros. In the example, the Z-bus has 11 on it at this time so S4 and S5 remain at a 0 condition. The branch statement is 1,S3 with S3 at a 1 condition in word 0428 giving a ROS address of 0433.

ADDRESS 0433: Set the contents of the R-Register (F8) into the T-Register. Write the contents of the R-Register into main storage at the same location it was read from. The branch statement is G6, G7 with G6 being a 1 and G7 being a 0. This sets the next ROS address to 042E.

ADDRESS 042E: Decrement the V-Register by 1 to EB. The branch statement is 0, S5 with S5 at a 0 condition giving a ROS address of 0434.

ADDRESS 0434: Decrement the second operand field length count by 1 to 0. This leaves the L-Register at 10. Set the MN-Register to 15EB and read out the next byte of the second operand (F7) and set it into the R-Register. Test the four low bits of the Z-bus for all zeros; the Z-bus has the L-Register input (10) on it. This sets S5 to a 1 condition. The branch statement is AC,0; there was a carry in the last word so AC is a 1 condition giving a ROS address of 0486.

ADDRESS 0486: Set the four low bits of the T-Register (8) into the D-Register (08). The branch statement is 1,1 giving a ROS address of 043B.

ADDRESS 043B: Set the contents of the R-Register (F7) into the A-Register and the contents of the D-Register (08) into the B-Register. Cross the output of the A-Register (7F) and route the high four crossed bits (70) to ALU. Route the low four bits of the B-Register (08) to the ALU. Set the output of ALU (78) into the R-Register. The branch statement is 1,1 giving a ROS address of 0487.

ADDRESS 0487: Decrement the V-Register by 1 to FA. Set the MN-Register to 0AA9 and read out that main storage byte but do not set the byte into the R-Register. The branch statement is S4, S5 with S4 at a 0 condition and S5 at a 1 condition giving a ROS address of 0429.

ADDRESS 0429: Decrement the first operand field length count in the L-Register by 1 and set the result into the L-Register. The L-Register now equals 00. Store the contents of the R-Register (78) into main storage location 0AA9. Test the high four bits of the Z-bus for all zeros; the Z-bus has the L-Register input on it (00) so S4 is set to a 1 condition. The branch statement is 1,1 giving a ROS address of 044B.

ADDRESS 044B: Decrement the J-Register by 1 to A8. The branch statement is 1,S7 with S7 at a 0 condition giving a ROS address of 041A.

ADDRESS 041A: Set the high four bits of the D-Register (0) into the R-Register (00). The branch statement is AC,1; there was a carry in the last word so AC is at a 1 condition giving a ROS address of 0487.

ADDRESS 0487: Decrement the V-Register by 1 to E9. Set the MN-Registers to 0AA8 and read out that main storage byte but do not set the byte into the R-Register. The branch statement is S4, S5 with both at a 1 condition giving a ROS address of 042B.

ADDRESS 042B: Store the contents of the R-Register (00) in main storage location 0AA8. The branch statement is 1,1 giving a ROS address of 0103 (Figure 3-13).

Both fields have ended and the second operand has been set into a packed format in the first operand location. The first operand locations have been set to 00 78 26 3C. Remember, the next Op code address (A046) has been stored in local storage locations A9 and AA. In order to read out the next Op code, it is necessary to read out these two locations and set them into the I- and J-Registers. This routine is called IC (Instruction Counter) Restore.

ADDRESS 0103: Set the S-Register to 00. Set the MN-Register to xxAA and read out the byte from local storage (46). Set the byte into the R-Register. The branch statement is 1,1 giving a ROS address of 01F3.

ADDRESS 01F3: Set the contents of the R-Register (46) into the J-Register. Write the contents of the R-Register into local storage at the same location it was read from. The branch statement is 0,1 giving a ROS address of 0105.

ADDRESS 0105: Set the L-Register to 00. Set the MN-Registers to xxA9 and read out the byte from local storage (A0). Set the byte into the R-Register. The branch statement is 0,1 giving a ROS address of 0181.

ADDRESS 0181: Set the contents of the R-Register (A0) into the I-Register. Write into local storage the contents of the R-Register at the same location it was read from. The branch statement is 1,1 giving a ROS address of 0107.

ADDRESS 0107: The add statement would set the S-Register bit 1 to a 0 condition if it had been at a 1 condition. Set MN-Registers to xxBC and read out the byte from local storage (4F) and set it into the R-Register. The branch statement is S1,1 with S1 at a 0 condition this time giving a ROS address of 0169.

ADDRESS 0169: Set the R-Register into the A-Register (4F). Route the four low bits through the ALU on to Z-bus (0F) and into the R-Register. Set the condition of the WRAP BUFFER latch into the WRAP latch. The branch statement is 0,S7 with S7 at a 0 condition giving a ROS address of 010C.

ADDRESS 010C: Cross the contents of the R-Register (0F) through the ALU and set the crossed byte (F0) into the G-Register. Write into local storage location BC the contents of the R-Register (0F); this resets the PSW bit. The branch statement is 0, INTR without an interrupt pending at this time. The INTR condition is a 0 giving the next ROS address of 0100 where the next Op code is read out.

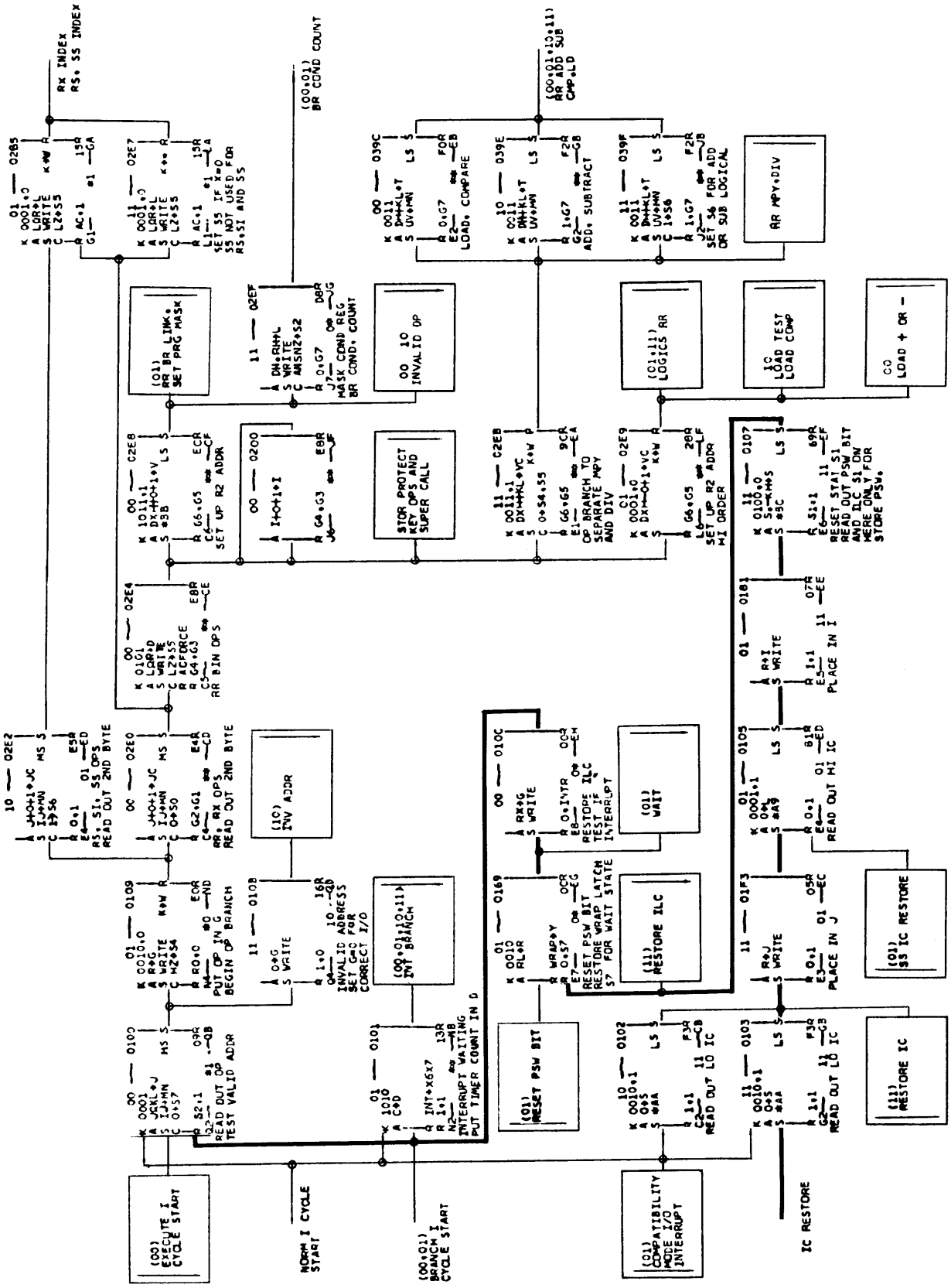


Figure 3-13. IC Restore

SHIFTS

- The description of the shift operation is explained using the CAS diagrams and the CLF charts.
- The CLF charts are found in the Maintenance Diagram Manual, Form 225-3466.

For this example, the instruction is 88200105. The number to be shifted is 0009E1A9. The objective of the instruction is to shift 5 bits to the right the number found in General Register 2. In the shift instructions, the first operand address indicates the General Register where the number to be shifted is located. The second operand address is used to determine the number of digits to be shifted; only the low six bits of the address are used. The number in this example after being shifted will be 00004F0D. Use Figure 3-14 to follow the operation step by step for register contents.

First, the Op code has to be decoded. Using CAS diagrams, start on QA001 at word 0100. For our example the S-Register and L-Register equal 00 at this time. In word 0100, the Op code is read from main storage and set into the R-Register. The instruction counter, IJ is increased by one. Since the S-Register is 00, the branch statement S2,1 will select address 0109.

ADDRESS 0109: The Op code is written into the same location in main storage from which it was read, and it is set from the R-Register to the G-Register. To set the Op code from R to G, the Op code is set on the Z-bus and the High 4 bits are tested to see if they are zeros. In this example, the Op code is 88 (10001000) so the high 4 bits are not all zeros; therefore, S4 is set to 0. The branch statement checks R0 and, in this case it is a 1, so the next address selected is 02E2 on QA011.

ADDRESS 02E2: In this word the second byte of the instruction (20) is read from main storage to the R-Register. One is added to the low portion of the instructon counter and S6 is set to 1. The next word to be read out is 02E5 on QA021.

ADDRESS 02E5: In this word the second byte is written into main storage at the location it was read from. Remember, this example was entered with the L-Register equal to 00 so the statement LΩR->I will set the second byte (20) into the L-Register. Also the low 4-bits of the Z-bus are tested for all zeros. The Z-bus has the second byte on it, so S5 is set to 1.

The branch statement checks whether an adder carry occurred when the low portion of the instruction counter was increased by one in the previous word. In this example we will assume a carry out did occur so the next address read out will be 0117.

ADDRESS 0117: In this word the high portion of the instruction counter is increased by one. The next address is 0115.

ADDRESS 0115: In this word the low portion of the instruction counter is increased by one. The third byte (01) of the instruction is read out of main storage and set into the R-Register. The 0 bit of the S-Register is set to 1. The branch statement checks the 2 bit of the S-Register which is zero at this time, so the next address is 0184.

ADDRESS 0184: In this word the third byte is written into main storage at the location it was read from. The high four bits of the third byte, the General Register number containing the base amount, is combined with an emitted three and set into the T-Register. The high 4 bits of the Z-bus are tested for all zeros. In this example the high 4 bits are zero, so S4 is set to 1; this is an indicator that the Base General Register is 0, so ignore any base amount. In the branch statement, S7 is 0 so the next address is 012C.

ADDRESS 012C: In this word the low four bits of the third byte (1) is set into the U-Register (01). The fourth byte is read out from main storage and set into the R-Register. The 0 bit of the S-Register is set to 0. In the branch statement, S4 is checked. In this example S4 equals a 1 at this time because General Register 0 was used for the base register. The next address is 0133.

Step	Comments	R-Register	G-Register	S-Register	L-Register	D-Register	T-Register	U-Register	V-Register
1	Readout Op Code	1000 1010	— — — —	0000 0000	0000 0000	— — — —	— — — —	— — — —	— — — —
2	Set Op Code in G	1000 1010	1000 1000	0000 0000	0000 0000	— — — —	— — — —	— — — —	— — — —
3	Readout 2nd Inst. Byte	0010 0000	1000 1000	0001 0010	0000 0000	— — — —	— — — —	— — — —	— — — —
4	Recognize Adder Carry	0010 0000	1000 1000	0001 0110	0010 0000	— — — —	— — — —	— — — —	— — — —
5	Add 1 to High Inst. Counter	0010 0000	1000 1000	0001 0110	0010 0000	— — — —	— — — —	— — — —	— — — —
6	Readout 3rd Inst. Byte	0000 0001	1000 1000	1001 0110	0010 0000	— — — —	— — — —	— — — —	— — — —
7	Determine Base is 0	0000 0001	1000 1000	1001 1110	0010 0000	— — — —	0010 0011	— — — —	— — — —
8	Readout 4th Inst. Byte	0000 0101	1000 1000	0001 1110	0010 0000	— — — —	0010 0011	0000 0001	— — — —
9	Check G0	0000 0101	1000 1000	0000 1110	0010 0000	— — — —	0010 0011	0000 0001	— — — —
10		0000 0101	1000 1000	0000 1110	0010 0000	— — — —	0010 0011	0000 0001	0000 0101
11	Branch on G2, G1	0000 0000	1000 1000	0000 1110	0010 0000	— — — —	0010 0011	0000 0001	0000 0101
12	Branch on G4, G3	0000 0000	1000 1000	0000 1110	0010 0000	0000 0000	0010 0011	0000 0001	0000 0101
13	Set Shift Amount (6 Bits)	0000 0000	1000 1000	0000 1110	0010 0000	0000 0101	0010 0011	0000 0001	0000 0101
14	Setup Address of 1st Byte	0000 0000	1000 1000	0001 1110	0010 0000	0000 0101	0010 0011	0000 0001	0010 0000
15	Readout High Order Byte	0000 0000	1000 1000	0001 1110	0010 0000	0000 0101	0010 0011	0000 0001	0010 0000
16	Setup Work Area Address	0000 0000	1000 1000	0001 1110	0010 0000	0000 0101	1111 1100	0000 0001	0010 0000
17	Check for Right or Left	0000 0000	1000 1000	0001 1110	0010 0000	0000 0101	1111 1100	0000 0001	0010 0000
18	Setup Left Shift Amount	0000 0000	1000 1000	0001 1110	0010 0000	0000 0101	1111 1100	1111 1011	0010 0000
19		0000 0000	1000 1000	0001 1110	0000 0011	0000 0101	1111 1100	1111 1011	0010 0000
20	Set Left Shift in G	0000 0000	1011 1000	0001 1110	0000 0011	0000 0101	1111 1100	1111 1011	0010 0000
21	Branch G6 (Logical)	0000 0000	1011 1000	0000 1010	0000 1010	0000 0101	1111 1100	1111 1012	0010 0000
22	Branch Less than 8 (S4)	0000 0000	1011 1000	0000 1010	0000 1010	0000 0101	1111 1100	1111 1011	0010 0000
23	No Adder Carry	0000 0000	1011 1000	0000 1010	0000 1010	0000 0101	1111 1100	1111 1011	0010 0000
24	Adder Carry, S5=0	0000 0000	1011 1000	0000 1010	0000 0000	0000 0101	1111 1100	1111 1011	0010 0000
25		0000 0000	1011 1000	0000 1011	0000 0000	0000 0101	1111 1100	1111 1011	0010 0000
26	Readout Work Area FC	0000 0000	1011 1000	0000 1011	0000 0000	0000 0101	1111 1100	1111 1011	0010 0000
27	Store Byte in Work Area	0000 0000	1011 1000	0000 1011	0000 0000	0000 0101	1111 1101	1111 1011	0010 0000
28	Readout Byte 0 of Number	0000 0000	1011 1000	0000 1011	0000 0000	0000 0101	1111 1101	1111 1011	0010 0001
29	Not Last Byte, Not Skewed	0000 0000	1011 1000	0000 1011	0000 0000	0000 0101	1111 1101	1111 1011	0010 0001
30	Readout Work Area FD	0000 0000	1011 1000	0000 1011	0000 0000	0000 0101	1111 1101	1111 1011	0010 0001
31	Store Byte in Work Area	0000 0000	1011 1000	0000 1011	0000 0000	0000 0101	1111 1110	1111 1011	0010 0001
32	Readout Byte 1 of Number	0000 1001	1011 1000	0000 1011	0000 0000	0000 0101	1111 1110	1111 1011	0010 0010
33	Not Last Byte, Not Skewed	0000 1001	1011 1000	0000 1011	0000 1001	0000 0101	1111 1110	1111 1011	0010 0010
34	Readout Work Area FE	0000 1001	1011 1000	0000 1011	0000 1001	0000 0101	1111 1110	1111 1011	0010 0010
35	Store Byte in Work Area	0000 1001	1011 1000	0000 1011	0000 1001	0000 0101	1111 1111	1111 1011	0010 0010
36	Readout Byte 2 of Number	1110 0001	1011 1000	0000 1011	0000 1001	0000 0101	1111 1111	1111 1011	0010 0011
37	Not Last Byte, Not Skewed	1110 0001	1011 1000	0000 1011	1001 0001	0000 0101	1111 1111	1111 1011	0010 0011
38	Readout Work Area FF	1110 0001	1011 1000	0000 1011	1001 0001	0000 0101	1111 1111	1111 1011	0010 0011
39	Store Byte in Work Area	1110 0001	1011 1000	0000 1011	1001 0001	0000 0101	0000 0000	1111 1011	0010 0011
40	Readout Byte 3, End Wk A	1010 1001	1011 1000	0000 1011	1001 0001	0000 0101	0000 0000	1111 1011	0010 0100
41	Not Skewed	1010 1001	1011 1000	0000 1011	1001 0001	0000 0101	0000 0000	1111 1011	0010 0100
42	Restore Work Area Address	1020 1001	1011 1000	0000 1011	1002 0001	0000 0101	1111 1111	1111 1011	0010 0100
43	Set S3 to Value of 1st Bit	1010 1001	1011 1000	0001 0011	0101 0010	0000 0101	1111 1111	1111 1011	0010 0100
44	Setup for 2nd Bit Value	1010 1001	1011 1000	0001 0011	1010 0100	0000 0101	1111 1111	1111 1011	0010 0100
45	Setup for 3rd Bit Value	1010 1001	1011 1000	0001 0011	0100 1000	0000 0101	1111 1111	1111 1011	0010 0100
46	Restore Low Byte Address	1010 1001	1011 1000	0001 0001	0100 1000	0000 0101	1111 1111	1111 1011	0010 0011
47	Readout Low Work Area FF	1110 0001	1011 1000	0001 0001	0100 1000	0000 0101	1111 1110	1111 1011	0010 0011
48	Shift Left 1st Bit	1110 0001	1011 1000	0001 0001	1100 0011	0000 0101	1111 1110	1111 1011	0010 0011
49	Shift Left 2nd Bit	1110 0001	1011 1000	0001 0001	1000 0110	0000 0101	1111 1110	1111 1011	0010 0011
50	Shift Left 3rd Bit	0000 1101	1011 1000	0011 0001	1000 0110	0000 0101	1111 1110	1111 1011	0010 0011
51	Store New Byte 3 In G.R. 2	0000 1101	1011 1000	0011 0011	1000 0110	0000 0101	1111 1110	1111 1011	0010 0010
52	Readout 2nd Byte, Work Area FE	0000 1001	1011 1000	0011 0011	1000 0110	0000 0101	1111 1101	1111 1011	0010 0010
53	Shift Left 1st Bit	0000 1001	1011 1000	0010 0011	0001 0011	0000 0101	1111 1101	1111 1011	0010 0010
54	Shift Left 2nd Bit	0000 1011	1011 1000	0010 0011	0010 0111	0000 0101	1111 1101	1111 1011	0010 0010
55	Shift Left 3rd Bit	0100 1111	1011 1000	0010 0011	0010 0111	0000 0101	1111 1101	1111 1011	0010 0010
56	Store New Byte 2 on G.R. 2	0100 1111	1011 1000	0010 0001	0010 0111	0000 0101	1111 1101	1111 1011	0010 0001
57	Readout 1st Byte Work Area FD	0000 0000	1011 1000	0010 0000	0010 0111	0000 0101	1111 1100	1111 1011	0010 0001
58	Shift Left 1st Bit	0000 0000	1011 1000	0010 0000	0000 0000	0000 0101	1111 1100	1111 1011	0010 0001
59	Shift Left 2nd Bit	0000 0000	1011 1000	0010 0000	0000 0000	0000 0101	1111 1100	1111 1011	0010 0001
60	Shift Left 3rd Bit	0000 0000	1011 1000	0010 0000	0000 0000	0000 0101	1111 1100	1111 1011	0010 0001
61	Store New Byte 1 in G.R. 2	0000 0000	1011 1000	0010 0000	0000 0000	0000 0101	1111 1100	1111 1011	0010 0000
62	Readout 0 Byte, Work Area FC	0000 0000	1011 1000	0010 0000	0000 0000	0000 0101	1111 1011	1111 1011	0010 0000
63	Shift Left 1st Bit	0000 0000	0100 1000	0000 0000	0000 0000	0000 0101	1111 1011	1111 1011	0010 0000
64	Shift Left 2nd Bit, End	0000 0000	1011 1000	0010 0000	0000 0000	0000 0101	1111 1011	1111 1011	0010 0000
65	Shift Left 3rd Bit, Single	0000 0000	1011 1000	0010 0000	0000 0000	0000 0101	1111 1011	1111 1011	0010 0000
66	Store New Byte 0 in G.R. 2	0000 0000	1000 0000	0010 0000	0000 0000	0000 0101	1111 1011	1111 1011	0010 0000
67	Logical End	0000 0000	1000 0000	0000 0000	0000 0000	0000 0101	1111 1011	1111 1011	0010 0000

Figure 3-14. Shift Example

ADDRESS 0133: This word writes into storage the fourth byte of the instruction at the location from which the word was read. Add one to the low portion of the instruction counter. The branch statement checks G0; the G-Register contains the Op code 88 (10001000), so the 0 bit is a 1. The next address is 0136.

ADDRESS 0136: In this word the fourth byte (05) is set into the V-Register. The branch statement checks S3. If there was an adder carry when the low portion of the instruction was increased during the last word, S3 will be a 1. In this example, a carry would not exist so the next address is 0160 on QA071. Register to 00 and ensure the carry latch and S3 are off. The branch statement checks the G-Register bits 2 and 1 (Op Code Bits 2 and 1). In this case both G2 and G1 are zeros, so the next address is 0164.

ADDRESS 0164: This word uses only the branch statement. The add statement will set the D-Register to 00 since the previous word set the R-Register to 00, and reset the carry latch. The output of the adder is 00. Therefore, S2 is left at a 0 state. The branch statement checks the 4 and 3 bits of the G-Register which are 1 and 0 respectively. This branch will take the flow out of I-cycle and read out word 031A on QA311. At this time the T-Register equals 03, the R-Register 00, the L-Register 20, the U-Register 01, the V-Register 05, and the G-Register 88.

The S-Register Bits 4, 5, and 6 are set to 1.

By this time you have been through enough microprograms so that it is not necessary to take each word and explain the operation in detail. The execution of the example will be explained using the CLF diagram 13 on page 16 of the Maintenance Diagram Manual with cross reference to micro words and Figure 3-14 when needed.

Once again what we want to do is shift the number 0009E1A9 right five positions and replace the high-order bits with zeros.

On the flow chart, the first block statement is; setup the amount of the shift and address of the high-order byte to be shifted. So in the D-register the amount 05 is set, making certain the two high order bits are zeros. Also, in the V-Register, 20 is set as the address of the high-order byte of the number.

The next statement in the flow chart is to read out the high-order byte to be shifted. This is followed by a decision to see if the number is a single or double word. In the microprogram, this is tested by checking G5, which is a 0 in our example. So from the decision block we take the NO line.

A work area is needed to shift the number into, so an address is set up for the local storage working area. The address of FC is set into the T-Register. The next block in the flow chart checks for the direction of the shift and what the sign of the number is. The direction is checked by testing G-7, which is 0 in this example and indicates a right shift. The sign is checked by testing the high-order bit of the number; in this example it is a 0, so S5 is set to a 1. Next the amount of the shift in the D-Register is complemented. The low-order 2 bits are then set into the G-Register bits 2 and 3. This sets up the amount of left-shift necessary after the shift-right is finished; a left-shift may be necessary as a right-shift may shift too far. In this case we shift 8-bits to the right as the data is moved to local storage. Then the data is shifted 3-bits to the left as it is moved back into register 2.

The next flow chart block is a decision block for direction of shift. In this example the right leg is taken. Then the amount of the shift is determined to be less than or greater than 8. In this example it is less than 8. The next CAS page is QA341.

The next block in the flow chart is a decision block for "shift amount exceeds work area". In this example it does not; therefore, take the NO leg. Now it is determined if the shift is zero, less than 5, or greater than 5. If the shift is 1 through 4, the number shifted must be skewed. This means a byte is read out and four bits are shifted right 4 positions. The next byte is then read out, and the other four bits of the first byte are shifted with 4 bits of the second byte. In this example the shift is 5; thus, on the flow chart the NO leg is taken and conditions are set to enter the work area direct. This will result in the number being shifted too far to the right and requires a left-shift when moving the number back to the General Register.

Since the shift will shift 8 bits at a time and this is a logical shift, the high-order byte is forced to 00. The next CAS page is QA351.

The next three blocks on the flow chart form a loop to shift the number 8 bits to the right and store the shifted number 1 byte at a time in the work area. When the end of the work area is found, the last

byte of the number will be in the R-Register. The address of the lowest byte in the work area is setup (FF). Using QA351 and QA361, the value (0 or 1) of the bits to be left-shifted is determined and S-Register bits are set accordingly. In this example S3 is set to 1, S6 is set to 0, and S7 is set to a 1. The low-order byte address for the General Register is set up in the V-Register. The next CAS page is QA371.

The next two blocks on the flow chart form a loop to shift the number stored in the work area the correct number of positions to the left. In this example, it shifts three positions. The bit value of the high-order bits that are shifted out are retained by setting S-Register bits and then set in the low-order bits of the next byte when it is shifted. After each byte is shifted, it is stored in the corrected position of the General Register. When the high-order byte of the General Register is found, the number has been completely shifted. Note: If this is a double word, the high-order byte would cause the next General Register low-order byte address to be set up and the loop is continued. The next CAS page is QA361. Since this is a logical shift, the operation is completed. If it had been an algebraic shift, then the correct sign would be set for the high-order bit.

We have accomplished what we wanted; the number 0009E1A9 has shifted right 5 positions. The end result is the number 00004F0D.

FLOATING POINT

- In explaining the floating point instruction, the RR format of a normal add is used.
- Refer to the Maintenance Diagram Manual, Form 225-3466, Figures CLF 28 and CLF 29 for the flow of the operation and the example being discussed.
- Refer to the IBM System/360 Principles of Operation, Form A22-6821, section on Floating Point Arithmetic for a description of the data format.
- The operation explained here should be followed by using the flow chart, the example, and CAS diagrams.

The instruction for the example is 3A02. This is a single precision normalized floating point add using floating point register 0 for the first operand and floating point register 2 for the second operand. The first operand is sometimes called the destination and the second operand is called the source. The results of the add replaces the contents of register 0. Refer to CAS diagram QA000 page 2, Local Storage Map, to see the locations of the floating point registers.

During the I-cycle portion of the instruction, the Op code is set in the G-Register (3A) and the register numbers into the D-Register (02). The register numbers are checked to determine if they are even and less than 8. If they are not, a program interrupt occurs because of an invalid specification. The address of the high-order byte of the first operand is set up in the T-Register, and the address of the high-order byte of the second operand is set up in the V-Register. In the example, the T-Register is set to 08 and the V-Register is set to 28.

Referring to the flow chart and the CAS diagrams, start the execution of the instruction. The second operand characteristic is read out and shifted left one bit position. The S-3 bit is set to 1 if the sign is minus. Set up the address of the low-order fraction byte (2B) in the V-Register. In the example, S3 is a 0 because the sign is plus.

Read out the first operand characteristic and shift the byte left one bit position. If the sign is minus, set S0 to a 1. In the example S0 stays a 0 because the sign is plus. Set up the T-Register to the low-order byte address of the fraction (0B). Subtract the second operand characteristic from the first operand characteristic to find the characteristic difference. The S2 bit is set to 1 if the characteristics are not equal, and S3 is set to 1 if the first operand characteristic is

larger than the second operand. In the example, both S2 and S3 are set to 1.

The characteristic difference is checked to see if it is too large to be handled; first for double precision and then for single precision. The example has a difference of four; over seven is too large for single precision. Also, the number of bytes to be shifted if the shift is odd or even, and which operand is to be shifted is decided by combining the S-Register and the characteristic difference. The result is set in the S-Register for branching functions. For example, the S-Register is set to 00000101. From this we know the second operand is to be shifted (S7=1) and the shift is an even shift (S6=0). The S4 and S5 bits determine the number of bytes to be shifted. In the example, S4, S5 equal 01 for one byte or 2 hexadecimal digits.

Set up to read out the byte containing the guard digit. The guard digit is used as an extra digit for the fraction on intermediate results to increase the precision of the final result. Read out the byte and take the high digit as the guard digit. Set the guard digit in the D-Register and set the L-Register to the value of the number of bytes to be shifted. Store the guard digit in byte 10 of local storage, address 9A, in true or complement form. In the example a D0 (11010000) is stored.

Read out the next byte of the second operand (E5) and set into the D-Register. Read out the low-order byte of the first operand fraction (62). Add second operand byte to the first operand and set results (47) in first operand low-order byte position (0B address). Set S3 to remember carry out. Verify whether the second operand high-order fraction byte has been read; if not, read out the next byte from each operand. In the example, read out first operand (38), second operand (42), and add. Set the result (7B) in the first operand location; the add resets S3. This time, the second operand field has ended, so force a 00 byte to be added to the next first operand byte (0B). The result (0B) is set into the same location it was read from. Verify whether the first operand high-order byte has been read out. In the example it has been.

The first operand, floating point register 0, now looks like this: 7A0B7B47. The fractions have been added together, and now the characteristic has to be corrected.

Read out the first operand characteristic byte (7A) and set into the D-Register. A test is made on S0 and S3; in the example they are both 0. The test determines if a fraction overflow has occurred or if a recomplement of the fraction is needed. For example, the fraction is OK. The address of the first operand characteristic byte is set into the V-Register (08). The G6 bit is tested; G6 would be a 0 if this was a compare operation. Then S2 and G5 are tested; S2 if a 0, indicates a zero fraction (lost significance) and G5 is a 0 for a normalize operation. In the example, S2 is a 1 and G5 is a 0, so the number should be normalized if needed. The T-Register is set to the address of the high-order byte of the fraction (09). Read out the high-order byte of the fraction and set into the L-Register. Check for high-order zero digits. In the example, the high-order digit is a 0 but the second digit is not; so do a shift of 1 hex digit to the left.

First, subtract one from the characteristic in the D-register and set the result in the characteristic position of the first operand. In the example, 7A minus 01 gives a result of 79. Check the result by testing for a 1BC, one bit carry, to see if a characteristic underflow has occurred. If a 1BC equals a 1, the characteristic is OK; if a 1BC equals a 0, then a characteristic underflow has occurred. In the example, the 1BC is a 1. Also, set S6 to a 1 to control the shift of the digits-skewed or direct.

Read out the next byte of the fraction. Using the L- and R-Registers, shift the two bytes one digit to the left and store the shifted byte in the correct position of the fraction. The remaining digit is left in the L-Register and is used with the next digit. This continues until the end of the fraction is found. When the end of the fraction is found, the guard digit is read from local storage where it was stored and set into the next digit position of the fraction. If there were more than 2 high-order zeros, the remaining positions of the fraction would be filled with zeros. The number in floating point register 0 now is 79B7B47D, which is the final answer.

When the end of the fraction field is found, the sign of the number is tested. In the example, the sign is plus, and the high bit of the characteristic is 0. The condition register is read from local storage location BB and set to the correct condition. In the example it would be set to 0010, because the number is not zero and the sign is plus.

The S- and L-Registers are set to 00 and a branch to I-cycles is taken.

MACHINE CHECK HANDLING

CPU ERRORS

- Error conditions may be highly intermittent.
- The CPU clock circuits are so designed that CPU errors do not necessarily stop the clock.
- Each type of error sets a particular postion of the machine-check register.

If the check-control switch is in the process position, an address is set in ROAR that is the start of a microprogram to handle machine checks. This microprogram stores the status of the machine-check register, sets registers to correct parity, and initiates a PSW store and load routine which causes a branch to a control program. The control program handles all machine checks. If a second error should occur before the control program can clear the first error with a load PSW command, the CPU clock stops. Should errors occur during a selector-channel ROS request or in a multiplexor-share request, further testing must be done before the control program is executed. Four main functions to consider are:

1. The setting of the machine-check register.
2. The start of the MC microprogram.
3. The objectives of the MC microprogram.
4. Stop on error conditions.

MACHINE CHECK REGISTER

- The MC register consists of eight latches. Consider the setting of each latch (Figure 3-15).

Pos. 0 This position is set at T3 time if there is an A-register check and the allow A-register-check latch is ON. The allow A-register-check latch is set on at P1 time with certain decodings of the CA control field. This latch is set off at T1 time if the suppress A-register-check latch is on. Remember, when there is an A-register check, a machine check microprogram may be entered. It is conceivable that the register that caused the error may be used in this microprogram. If further A-register checks were not blocked, a second error would occur which would stop the CPU clock. The suppress A-register-check latch blocks further A-register checks until the D-register is gated to the A bus. This does not occur in the microprogram until the registers used in the MC microprogram have been set to good parity.

Pos. 1 This position is set on at T3 time if there is a B-register check. Note that the failure may be due to the B-register, the B-register controls, or to the register gated to the B-register. If a B-register check occurs during a cycle between a read and a write, and if the data source is the R-register, the failure can be caused by a read/write storage failure.

Pos. 2 An MN register check sets position 2 at T3 time if the allow-write line is active. This line is active early during the read cycle when MN has been set.

Pos. 3 A control-register check sets position 3 at T2 time.

Pos. 4 MC-4 is set on at T2 time with a parity check in either the SALS or CN field of the ROS output.

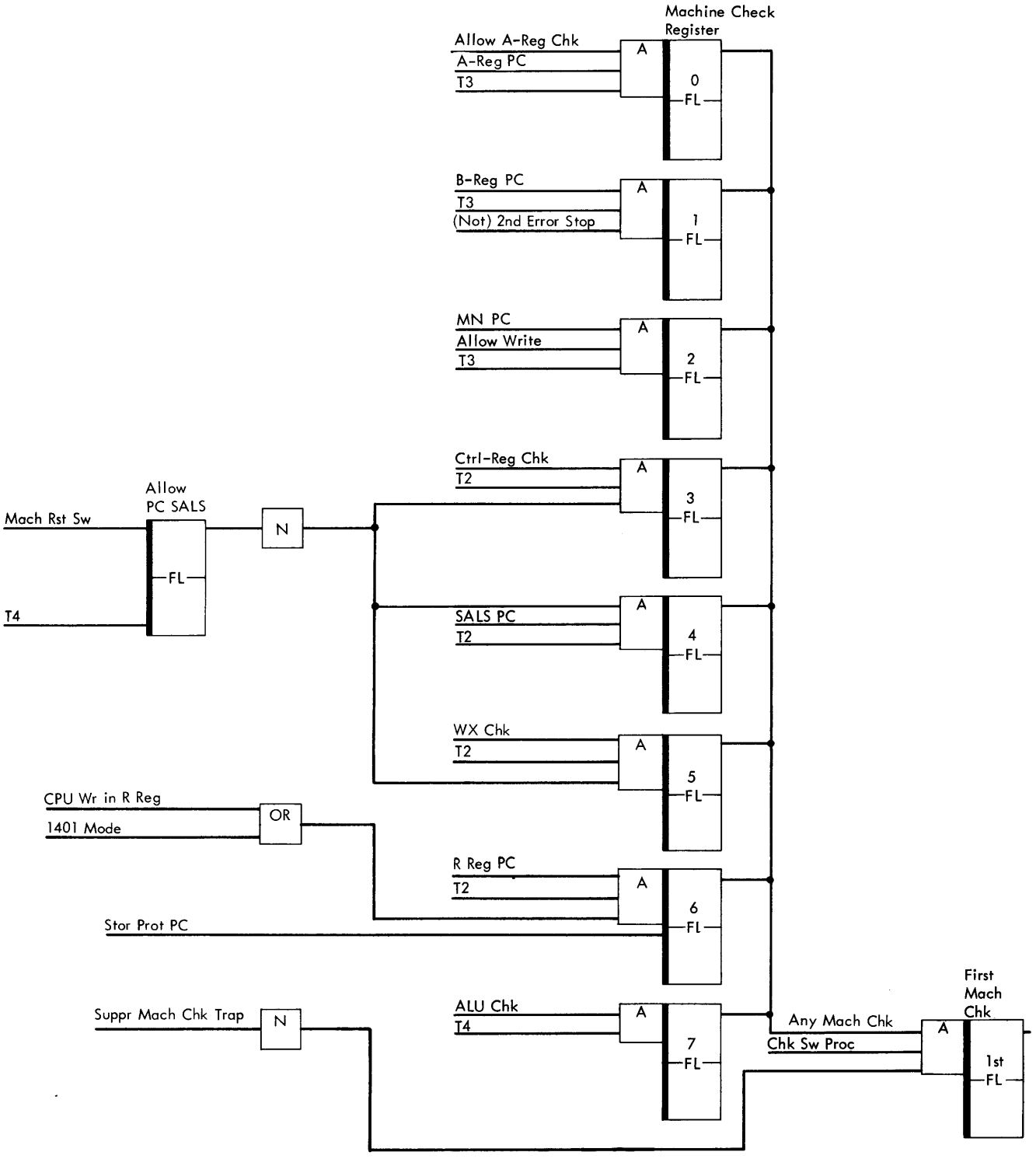


Figure 3-15. Machine Check Register

Pos. 5 Position 5 is set on to indicate a ROAR check.

Pos. 6 An R-register check sets position 6. If the Storage Protect Feature is present, this position will also be set if bad parity is read out of the storage protect stack.

Pos. 7 Set by an ALU check at T4 time.

Any MC register latch (0-7) that is on, sets the first-machine-check latch if the Console-check switch is set to PROCESS.

The next objective is to enter the MC microprogram. Figure 3-16 shows the machine-check-pulse line. This line brings up controls to enter the MC microprogram. The line is active when:

1. The priority latch is off.
2. Switches are not being used to set W and X.
3. The first-machine-check latch is on.
4. The suppress-malfunction latch is off. This latch determines whether errors are to be recognized. If this latch is on, the line to enter the MC microprogram (machine-check pulse) is not active.

Note: The machine-check latches are reset at P4 time with a reset line.

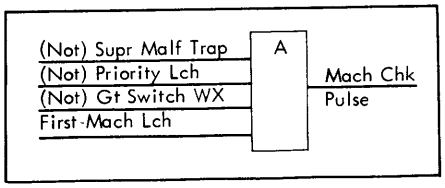


Figure 3-16. Priority Pulse

MACHINE CHECK MICROPROGRAM

The starting address for this program is 0004 (Figure 3-17). Position 5 of the H-Register is checked to see if it is set. If this position is set, it means that the error is to be charged to the selector channel. The next decision is made by testing H-6 to see if the error occurred during a multiplexor channel share-request. Position H1 is then tested. Since this is the first time through this flow chart, H1 is not set. The next step in the program is to set position 1 of the H-register. Should another error occur before the control program resets H1 to a zero with a load-PSW command, a branch would again be taken to the start of this routine. The micro order that tests H1 sets up a branch to a STOP if H1 is set on. Assume there are no previous errors and continue with the flow chart. The information in the MC register is stored in location 0080 (HEX). The Old Psw IS STORED. Some hardware registers are set to good parity. The last register set to good parity is the D-Register. This allows further A-Register checking by turning off the suppress A-Register-check latch. A new machine-check PSW is loaded to handle the different checks. During this control program, a ROS word in the load PSW operation will cause H1 to be reset. The instructions that follow depend on the control program.

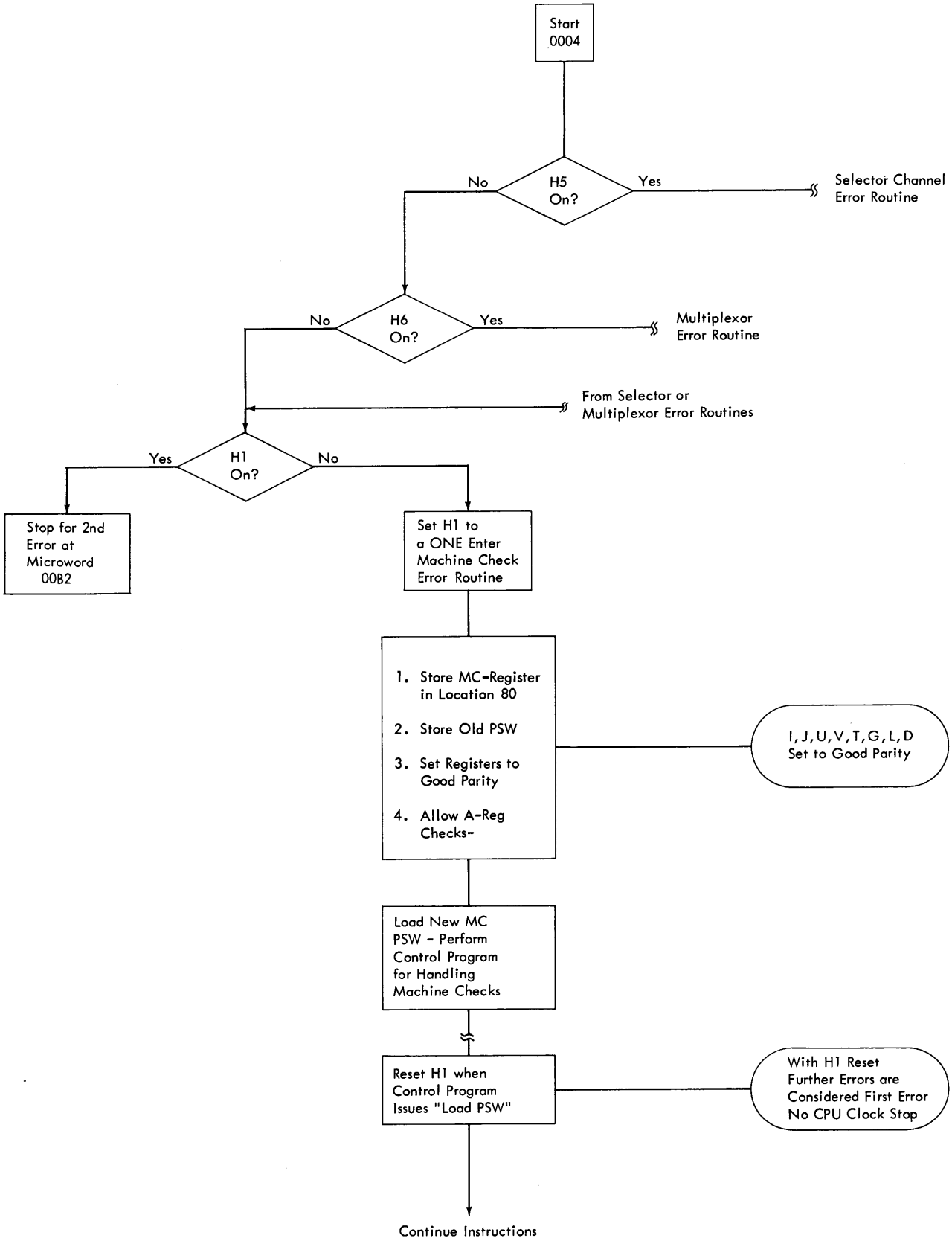


Figure 3-17. Machine Check Microprogram

Errors could occur that would cause the CPU to execute a tight loop of ROS words without stopping. Consider what would happen if a second error occurred before H1 is set on. A continuous branch would be forced to address 0004, the start of the machine-check micro-program. To overcome this condition, there is a hardware circuit which turns on the hard-stop latch (Figure 3-18). The first error that brings up the machine-check line turns on the second-error-stop latch. If this latch is still on when the next error occurs, a circuit is active to turn on the hardstop latch. The second-error-stop latch is turned off when position 1 of the H-Register is set. This gives the control program an opportunity to handle the error condition.

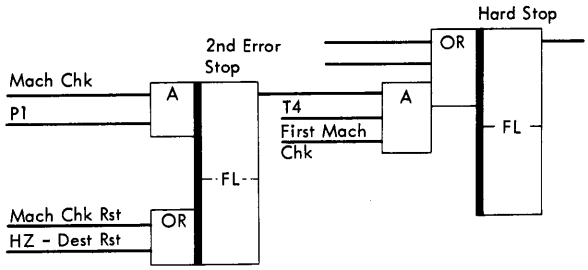


Figure 3-18. Second Error Stop Latch

FORCED MICROPROGRAM ENTRIES

- Ten ROAR addresses may be forced.
- Priorities are executed in order of importance.
- Waiting priorities are stacked.

There are nine orders of priority (Figure 3-19). Each order of priority will set a specific address in ROAR. This is done by setting a particular bit of the X-Register. The exception to this is a selector channel ROS request, which sets positions 4, 6, and 7 of the X-Register. There is one function that has priority over all others. This is machine reset. The machine-reset function sets the P bit of the X-Register. The ROAR address, therefore, is 0000 (HEX). This is the starting address of a microprogram routine to loop the microdiagnostic, zero the UCW's clear 1050 locations in local storage, set PSW bit to zero, etc.

The other priorities are shown in order of importance. As an example: AND number four must be satisfied to enter the micro-program that handles a memory wrap condition (X2 wrap). The X-Register is set, 0010000. The inputs to this AND are:

1. not priority latch. This line blocks the AND if there is another priority in process.
2. not gate switches to WX.
3. not PP 1, 2, 3. This input assures that no higher priority must be taken first. PP1 is an output from AND circuit number 1. PP2 is an output from AND number 2, etc.
4. memory-wrap-request latch. This line is developed from a priority-stacking latch, which was set because of a memory-wrap condition.
5. not H-Register 2. There are times when a memory wrap can occur but may be ignored. The microprogram can set position 2 of the H-Register. When this position is set, memory wraps are ignored.

Two other lines are developed when there is a priority entry. They are:

ALLOW LOW PRIORITY: Active on (not) priority PP1 2, 3, or 4. Used to satisfy lower-order priorities.

ANY PRIORITY PULSE: Active when any priority, PP1 through 8, is active. This line is used to set a latch which blocks further priorities, and is discussed later.

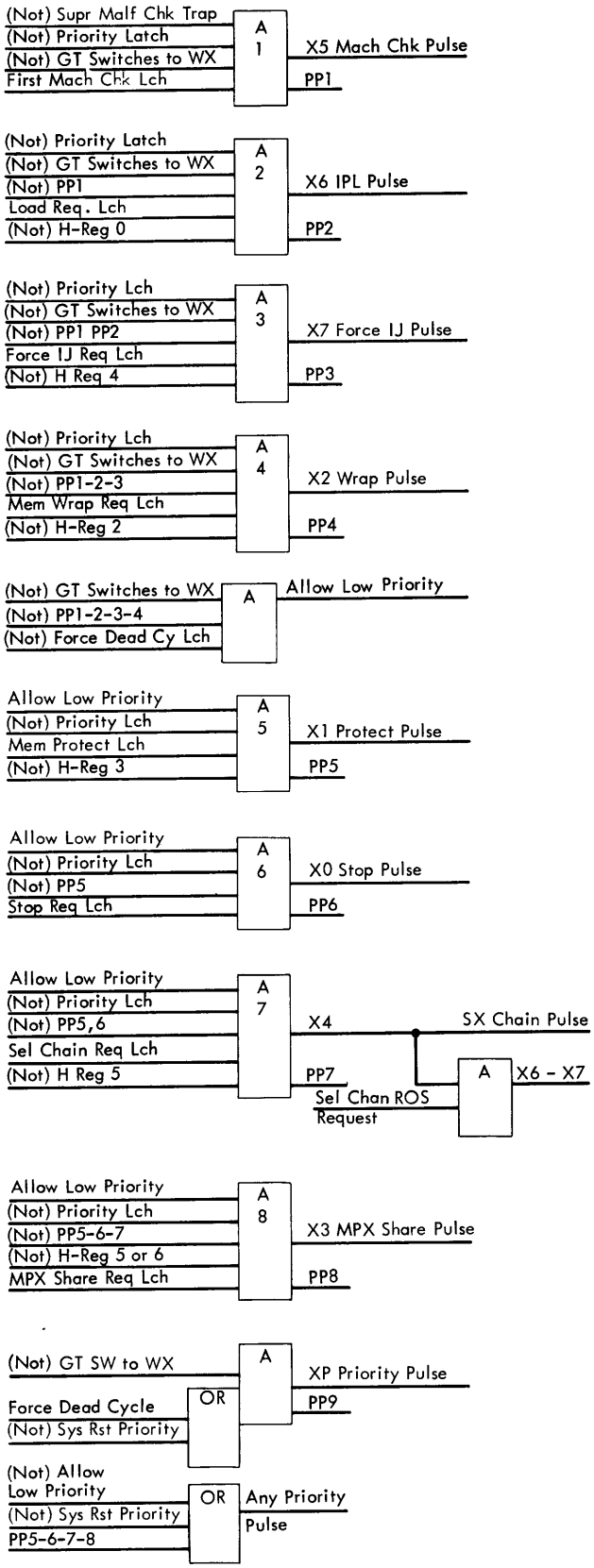


Figure 3-19. Priority Microprogram Entries

PRIORITY STACK LATCHES AND CONTROLS

The AND's that develop the priority pulses each have an input that is satisfied by a stacking latch (Figure 3-20). These latches are needed because several priorities can occur at one time, but only one can be handled. Notice that to set the stacking latch for a MPX-share request (PH8), position 6 of the H-Register must not be set. Early in the MPX-share request microprogram, this position of the H-Register is set. Further MPX-share requests are then blocked from setting the stacking latch. When there is a selector-channel-ROS request, the microprogram that handles the request sets position 5 of the H-Register. This not only blocks further selector-channel ROS-requests, but also MPX-channel-share requests.

While one priority is being handled, others must be temporarily blocked. Remember, if a priority pulse is developed, the line any-priority-pulse is active. This line turns on the any-priority latch at T1 time. With this latch on, a T3 pulse turns on the priority latch. The priority latch, when it is on, blocks the AND circuits that develop further priority pulses until the latch is reset. Some of the ways to reset the priority latch are:

1. At P4 time, with the WX SABC latch on. This latch is turned ON at T1 time if WX must be set manually.

2. At T3 time, if the priority-reset-control latch is on. This latch is set on when the H-register is specified as the destination of data (eg. A + B->H).

PARITY CHECK TIMINGS

- A parity check is made on the SAL's CN field, control registers, and ROAR.

The SAL outputs (including the control register SAL's) are checked at T2 time if the allow-PC-SALS latch is on. Machine check register position 4 is set if there is an even number of SAL outputs or an even number of CN field bits. A control register check, in turn, sets position 3 of the machine check, register. Any of these ROS parity checks block the set of the indicating ROAR if the Check-stop switch is set to the stop position.

The combination of the parity bits for the W- and X- indicating ROAR and the PA bit must be odd (all three or any one). If not, the WX check line is up, and position 5 of the machine check register is set at T2 time.

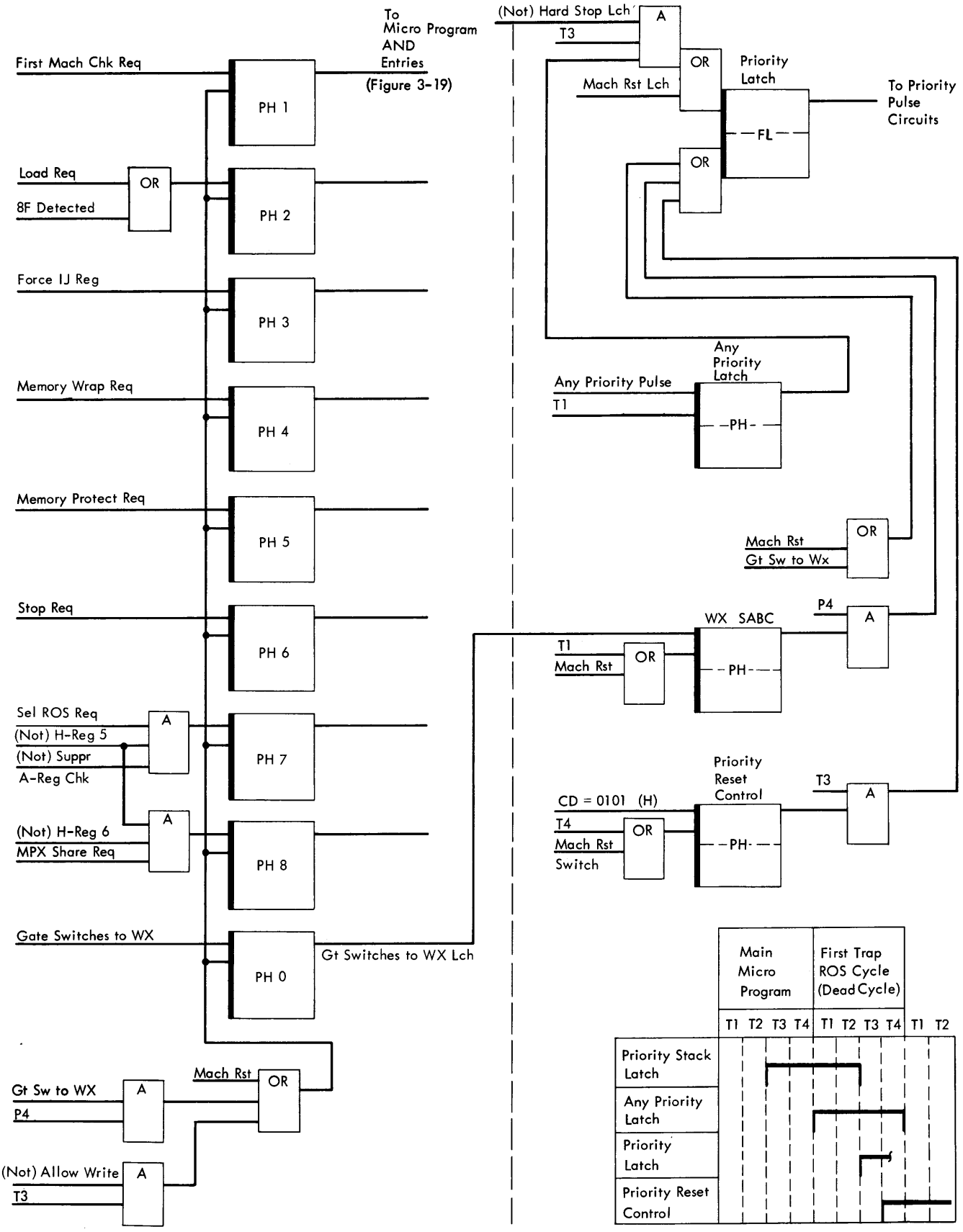


Figure 3-20. Priority Stack Registers and Controls

CHAPTER 4. FEATURES

STORAGE PROTECTION

- The storage protection feature protects main storage positions assigned to a program from being changed.
- For storage protection purposes, main storage is divided into blocks of 2,048 bytes.
- A 4-bit key (0-F) is assigned to each block by a privileged instruction.
- There can be sixteen different keys assigned at one time.
- More than one block of main storage can be assigned the same key. Blocks with the same key do not need to be consecutive.
- The storage key is matched with the protection key in the PSW or CAW for an equal compare.
- An additional 256 position 6-bit core array is added when storage protection is installed on the IBM 2030.
- The storage keys and the UCW protection keys are stored in the added core array.

Storage protection is a feature available for System/360 Model 30. The storage protection feature protects a block of main storage assigned to one program from being changed.

There are two keys used with the storage protection feature: the storage key and the protection key. The storage key is the key assigned to each 2,048 byte block of storage and is stored in a special core storage area. The protection key is the key in the program status word or command address word and is compared to the storage key to determine if the area is protected. Any time a byte in a block of main storage is read out, the protection key in the PSW (Program Status Word), bits 8-11, is compared to the storage key assigned that block of main storage. When the read is during an I/O operation, the storage key is compared to the protection key of the CAW (Channel Address Word) bits 0-3. The keys match (are equal) when both the storage key and protection key are the same, or if the protection key in the PSW or CAW has a 4-bit code of 0. Only if the information in the block is to be changed do we use the result of the compare. If a mismatch (unequal compare) occurs at this time, the information read out of storage is regenerated at write time and an interrupt occurs so corrective action can be taken.

The storage key is not part of addressable main storage. A 256 position, 6-bit core array and controls are added to the 2030 when storage protection is installed. The core array is called the storage protection stack. Only five bits are used in the 2030: 4-bit key plus a parity bit. Parity is odd.

Thirty-two positions of the stack are reserved for storing the storage key associated with each 2048 byte block of main storage. The rest of the positions (224) store the protection key used with each possible UCW. This key is obtained from the CAW during the I/O start routine.

When main storage is addressed, the five high-order bits of the M-bus are used to set the five low-order positions of the SA protection stock. (Figure 4-1).

When auxiliary storage is addressed during a multiplexor operation, the five high-order bits of the N-bus are used to set the five low-order positions of the SA-register. The three high-order positions of the SA-register are set from the XXH, XH, and XL latches in the CPU. This allows the byte in the stack (associated with the particular MPX storage block we are using) to be addressed.

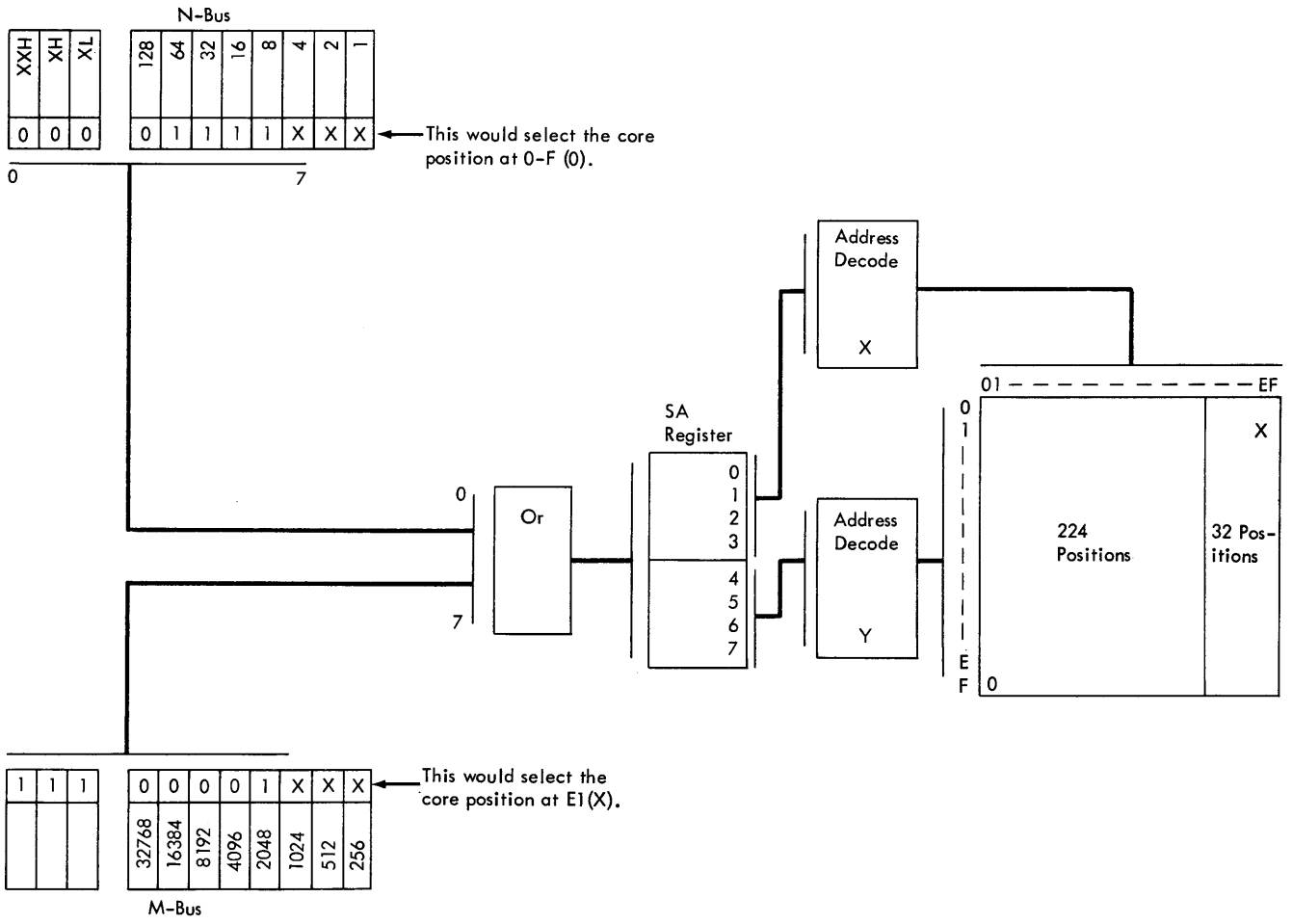


Figure 4-1. SA Register Addressing

STORAGE KEY

- A storage key is a 4-bit number assigned to a 2048 block of storage.
- There are 16 different keys, 0-F.

To implement the storage protection feature, main storage is divided into blocks of 2048 bytes. A processor with a main storage of 8192 bytes has 4 blocks and a processor with 65536 bytes has 32 blocks. Each storage block of 2048 bytes has a key associated with it. This key is four bits long and may contain any number from 0 through F. These numbers are referred to as storage keys. They can be assigned in any order and any of the possible 16 keys can be used regardless of storage size (Figure 4-2). Blocks of storage with the same key do not need to be consecutive blocks.

The storage keys are stored in the storage protection stack by the SET STORAGE KEY instruction (see Set Storage Key). There are 32 positions in the stack to store the storage keys: address E0 to FF of the SA register. Main storage block 0 to 2047 key is at location E0, block 2048 to 4095 storage key is at E1, etc.

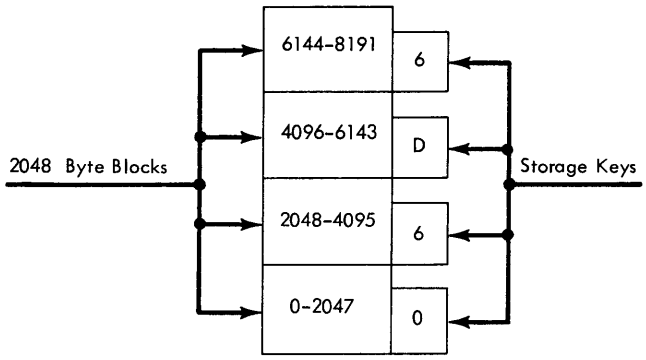


Figure 4-2. Storage Keys

PROTECTION KEY

- The protection key is a 4-bit number found in a PSW or a CAW.
- The protection key is compared to the storage key.
- The result of the compare is used only for the storage modification cycle.

The protection key is in bit positions 8-11 of the PSW and bit positions 0-3 of the CAW. The PSW or CAW protection key is compared to the storage key each time main storage is accessed. The result of the match is used only when the information in storage is to be modified. A read-write cycle, even if the keys are mismatched, is performed without an interrupt because storage is not modified.

For an I/O selector channel operation, the protection key in bits 0-3 of the CAW is placed in a register (GK or HK). When data transfer occurs for the channel, the storage key is matched to the protection key. A mismatch prevents storage from being changed, and sets a bit in the CSW (Channel Status Word) to indicate a protection exception.

For an I/O multiplexor channel operation, the protection key in bits 0-3 of the CAW is placed in a position of the storage protect stack. When data transfer occurs on the multiplexor channel, the protection key is read out of the stack and compared with the storage key. If a mismatch occurs, a bit in the CSW is set to indicate a protection exception, and storage remains the same. Sometimes the bit that is set in the CSW is temporarily stored in the UCW.

In effect the protection key, stored in the protect-stack, extends the UCW by 4 bits. This allows a record of the protection key for each I/O unit on the multiplexor channel to be maintained.

If the storage protection feature is not installed, the protection key must be zero.

PROTECTION EXCEPTION

- When storage protection is violated, the protection exception is indicated in the PSW or CSW.

Whenever a program interrupt occurs, the interruption code (004) is placed in the old PSW. During an I/O operation, a bit is set in the CSW if the protection violation occurs to remember what the channel status is.

SETTING UP STORAGE PROTECTION

- The supervisor program assigns the storage keys to each block of storage.
- After the problem programs are loaded and the protection keys set, the supervisor program transfers control to a problem program.

The SET STORAGE KEY is a privileged instruction. It may be issued only when bit 15 (problem state bit) of the PSW is zero. In a typical supervisor controlled operation, the supervisor causes a problem program to read into main storage. The supervisor sets the storage keys for the area of storage used by the problem program. The PSW used by the problem program is assembled by the supervisor program. This assembled PSW has a protection key that matches the storage key associated with the problem program. Once the function of loading a problem program into main storage and assigning the keys for storage protection is done, the supervisor passes control to the problem program. This is done with the LOAD PSW instruction which specifies the assembled PSW (Figure 4-2). The LOAD PSW instruction causes the protection key to be stored at K25 (B9) of the local store and in the four high-order positions of the Q-register.

The protection key in the PSW used by the supervisor program is generally zero. This allows the supervisor program to modify data anywhere in main storage. The main storage area occupied by the supervisor program has a storage key of F (Figure 4-3). This means that unless a problem program has a key in its PSW of 0 or F, it cannot modify information in the area used by the supervisor program. This is unlikely because the supervisor program assigns the storage and protection keys.

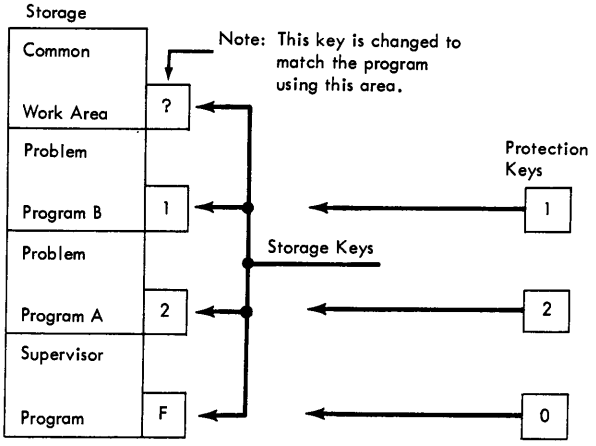


Figure 4-4. Storage and Protection Keys

The same storage key number can be set for more than one block of 2048 bytes. However, each program in main storage should have a different storage key assigned to protect one program from another. For instance, the supervisor program may take one block of 2048 bytes, which is

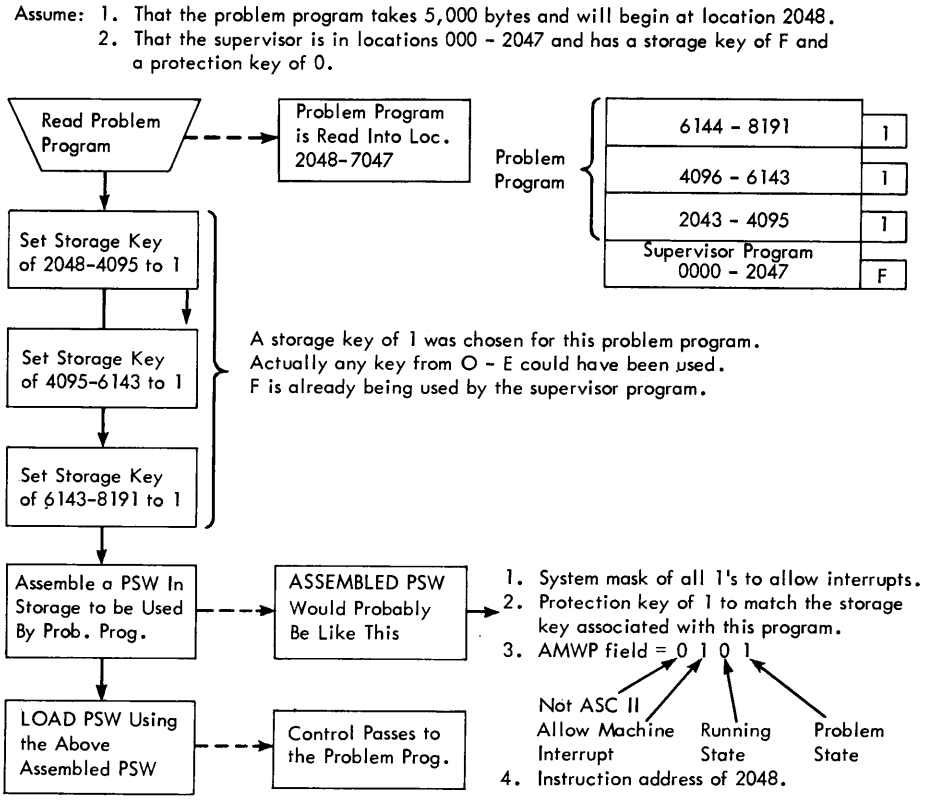


Figure 4-3. Using Storage Protection

assigned a storage key of F. This storage key would most likely be assigned by the supervisor program just after it is read into the system. The problem program is then read into the processor (as a result of a section of the supervisor program). This problem program takes up 3 blocks of 2048 bytes. Each of the three blocks is assigned the same storage key (1, for example) by the supervisor program. The PSW for the problem program is given a protection key that matches its storage key. This allows the problem program to alter itself if necessary, but prevents it from altering another program.

It is possible to have two or more problem programs in main storage at once. Of course, just as in the supervisor controlled concept, only one program is being executed at any one time. From Figure 4-4, we can see that each problem program has a different storage key. The protection key used by each program is also different; each matches the respective storage key.

Notice in Figure 4-4, the protection key of the supervisor program does not match its storage key. Since the protection key is zero, it does not have to match. A protection key of zero can unlock any area of main storage and alter its contents if necessary.

SET STORAGE KEY

- SET STORAGE KEY is a privileged instruction.
- The instruction is of the RR format.
- Used to set the storage key into the storage protection stack.

The key of the storage block addressed by the register designated by R2 is set according to the key in the register designated by R1.

The storage block of 2048 bytes is addressed by bits 8-20 of the register designated by the R2 field. Bits 0-7 and 21-27 are ignored. Bits 28-31 must be zero. Otherwise, a specification exception causes a program interrupt.

The 4-bit storage key is obtained from bits 24-27 of the register designated by the R1 field. Bits 0-23 and 28-31 are ignored.

INSERT STORAGE KEY

- INSERT STORAGE KEY is a privileged instruction.
- The instruction is of the RR format.
- Used to check what key is assigned to a given block.

The key of the storage block addressed by the register designated by R2 is inserted in the register designated by R1.

A storage block is addressed by bits 8-20 of the register designated by the R2 field. Bits 0-7 and 21-27 are ignored. Bits 28-31 must be zero. Otherwise, a specification exception causes a program interruption. The 4-bit storage key is inserted in bits 24-27 of the register specified by the R1 field. Bits 0-23 of this register remain unchanged, and bits 28-31 are set to zero.

FUNCTIONAL UNITS

PHYSICAL DESCRIPTION

- The storage protection stack consists of a 256 character core array; each character has 6 bits.
- The 2030 uses only 5 of the 6 bits.

Storage protection in the 2030 has a capacity of 256 characters of six bits each. Only five bits are used. The core array, 32 steering diodes, and the temperature sensing thermistor are packaged on a 2-high SLT card, four sockets long (Figure 4-5). All five bits are stored in one physical plane. The X and Y drive lines are shown in Figure 4-6, sense line windings in Figure 4-7, and inhibit windings in Figure 4-8. Storage protection array and circuits are located on gate 01B at board E3 or 01A at board E3 in the 2030.

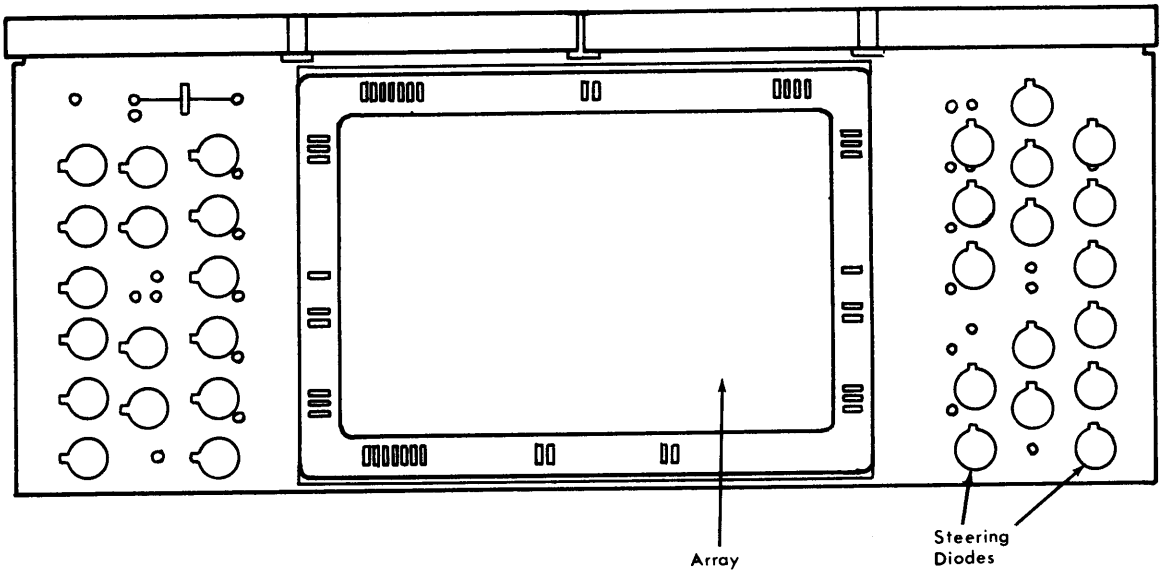


Figure 4-5. Storage Protection Array and Diode Card

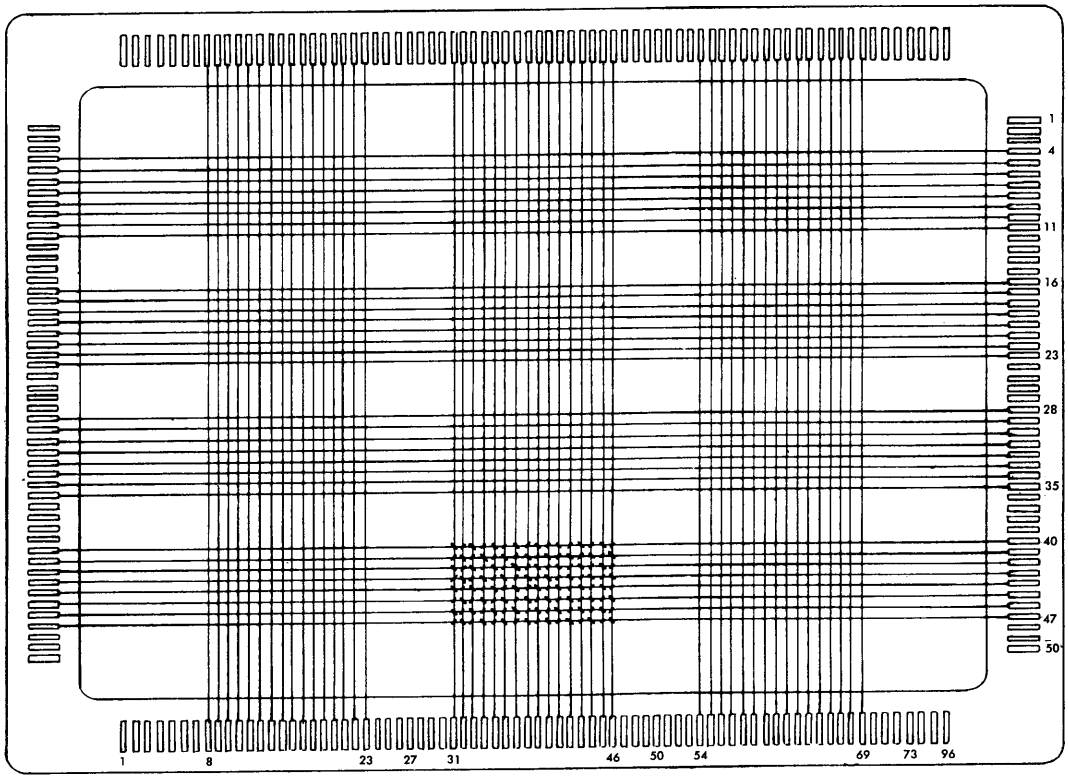


Figure 4-6. Storage Protection X- and Y- Driver Lines

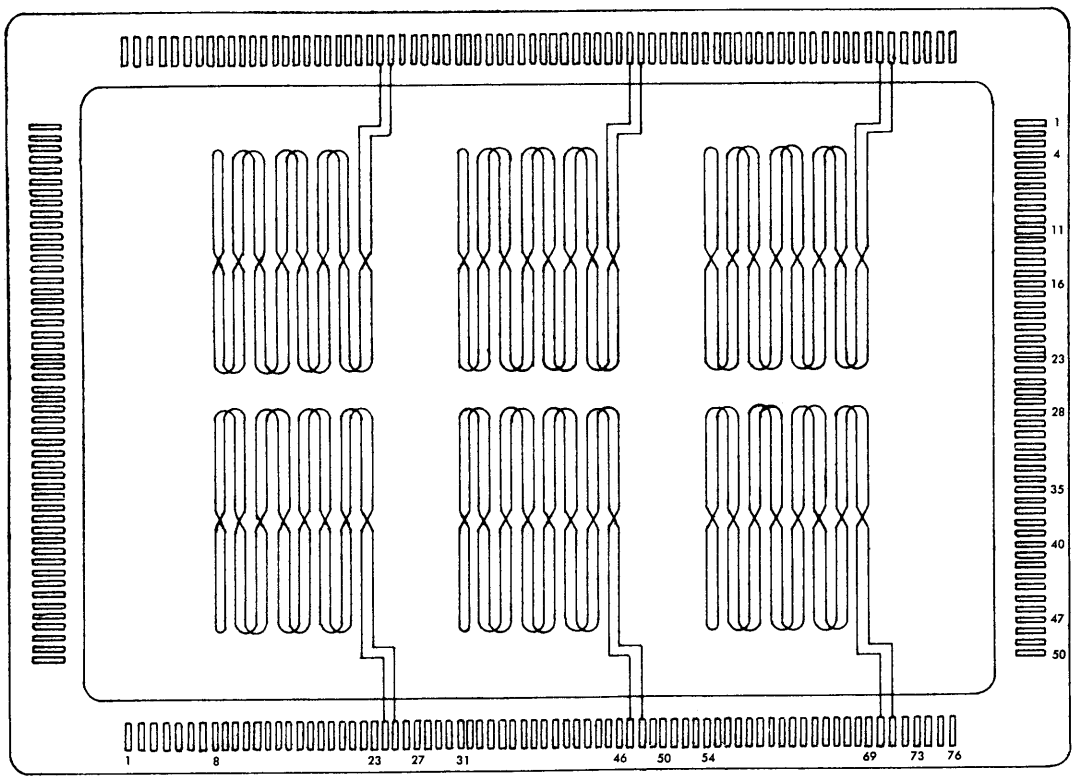


Figure 4-7. Storage Protection Sense Windings

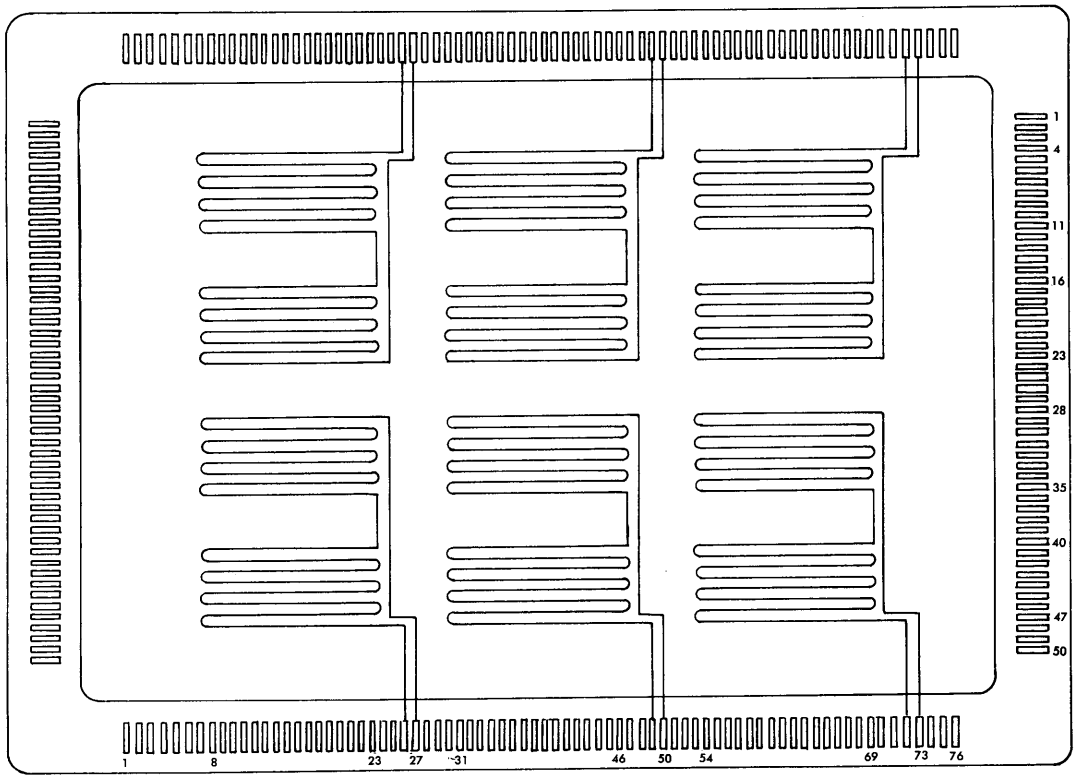


Figure 4-8. Storage Protection Inhibit Windings

TIMING

- Timing for storage protection stack is provided by a delay clock.

Timing for storage protection stack and controls is provided by a delay line clock (Figure 4-9). The stack storage select pulse turns on a latch. The same pulse, after going through the tapped delay line, turns the latch off. Output of the latch is ANDed with read/write control to generate either read-gate or write-gate and inhibit timing. A delayed stack-storage-select is ANDed with the read gate to generate a strobe pulse. The output of the latch is delayed by three inverters to supply proper write timing. The timing sequence for reading out from the storage protection array and for writing into the array is shown in Figure 4-10.

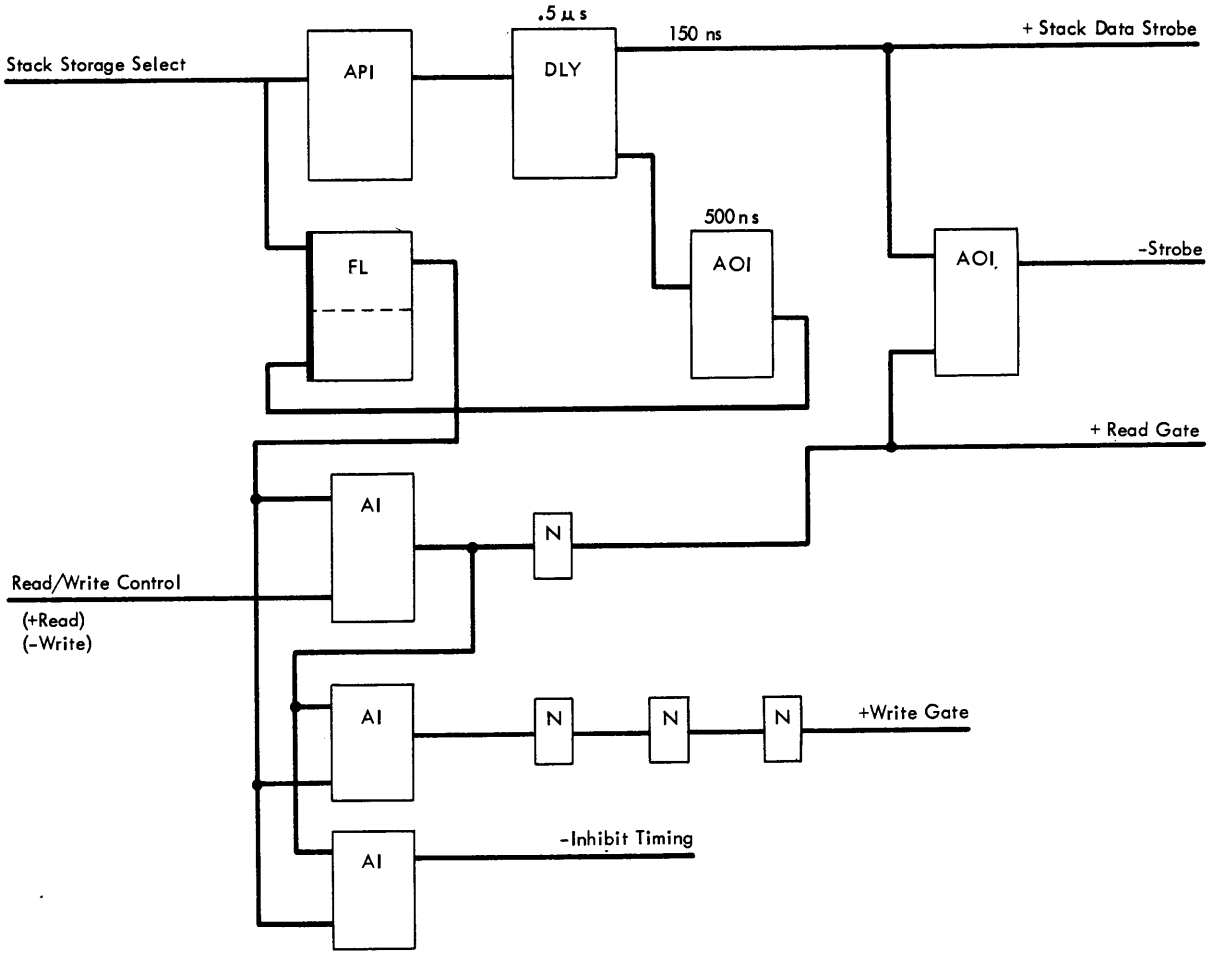


Figure 4-9. Storage Protect Clock

STACK ADDRESS REGISTER

- The SA (Stack Address) register is an 8-position register.
- The 3 high-order positions are set to 1's for main storage operation and to the XL, XH, and XXH latches for auxiliary or local store.
- The 5 low-order positions are set to the 5 high-order bits of the M-bus for main storage operation and the N-bus for auxiliary or local store.

The SA-register consists of eight polarity-hold latches (Figure 4-11). The three high-order positions (0, 1, and 2) are set in two ways. If the operation is main storage, the three positions are forced to 1 by ANDing main storage with +3 volts. The three positions are set by ANDing auxiliary storage and the output of the XXH, XH, and XL latches when the UCW is addressed.

The five low-order positions of the SA register are set either from the M-bus or the N-bus, For an operation using main storage, the five high-order positions of the M-bus are routed to set positions 4-7 of the SA register. During an MPX channel operation, the five high-order positions of the N-bus are routed to set positions 4-7 of the SA register.

The SA register is set at T1 time of the CPU clock (Figure 4-10).

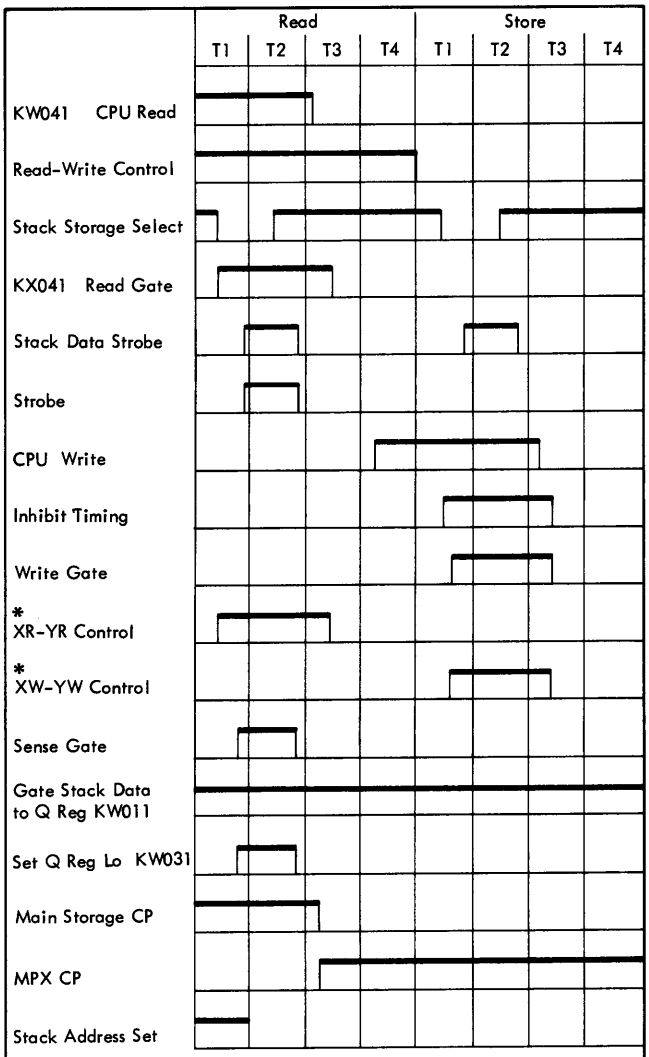


Figure 4-10. Storage Protect Stack Timing

ADDRESS DECODE AND DIODE MATRIX

- The true and complement outputs of the SA register latches are ANDed in the proper combinations to form X and Y select lines.
- The X and Y select lines are ANDed with the read and write control lines to address one position (5 bits) of the protect stack.

The eight output lines of the SA register are routed to eight inverters (Figure 4-11). The outputs of the inverters are ANDed with the outputs of the SA register to generate select lines to the core array. Positions 0-3 of the SH-register and inverted lines 0-3 are used to select the X lines. Positions 4-7 and the inverted lines 4-7 are used to select the Y lines. This gives us a 16x16 matrix and allows addressing of 256 positions.

The X and Y select lines AND with the read and write control lines so only one position of the protect stack is addressed at one time. Also the direction of current flow is determined by read or write through a driver-gate control circuit.

Figure 4-12 shows the diode matrix for selections of the X-lines. A similar circuit is used for the Y-lines. Any combination of the 0, 1, 2, or 3 positions of the SA register selects one of the X-drive lines.

SENSE AMPLIFIER

- There are 5 sense amplifiers associated with the protect stack.

The sense input amplifier consists of a differential amplifier, which gives good common-mode noise rejection while amplifying the bipolar sense line signal. Only negative-going signals from the first stage are fed through. During read time, a strobe pulse conditions the amplifier and if there is a negative signal, indicating a bit, it is sent on as an output.

INHIBIT DRIVER

- There are 5 inhibit drivers associated with the protect stack.

The inhibit driver is activated at write time. When a ZERO is to be placed in the core, a negative-going pulse is applied to the input of the inhibit driver. This causes inhibit current to flow in the inhibit winding, and prevents the core from changing states. The inhibit normally comes up before write current and drops at the same time, or a little later. The amplitude of the inhibit current is about the same as half-select write current.

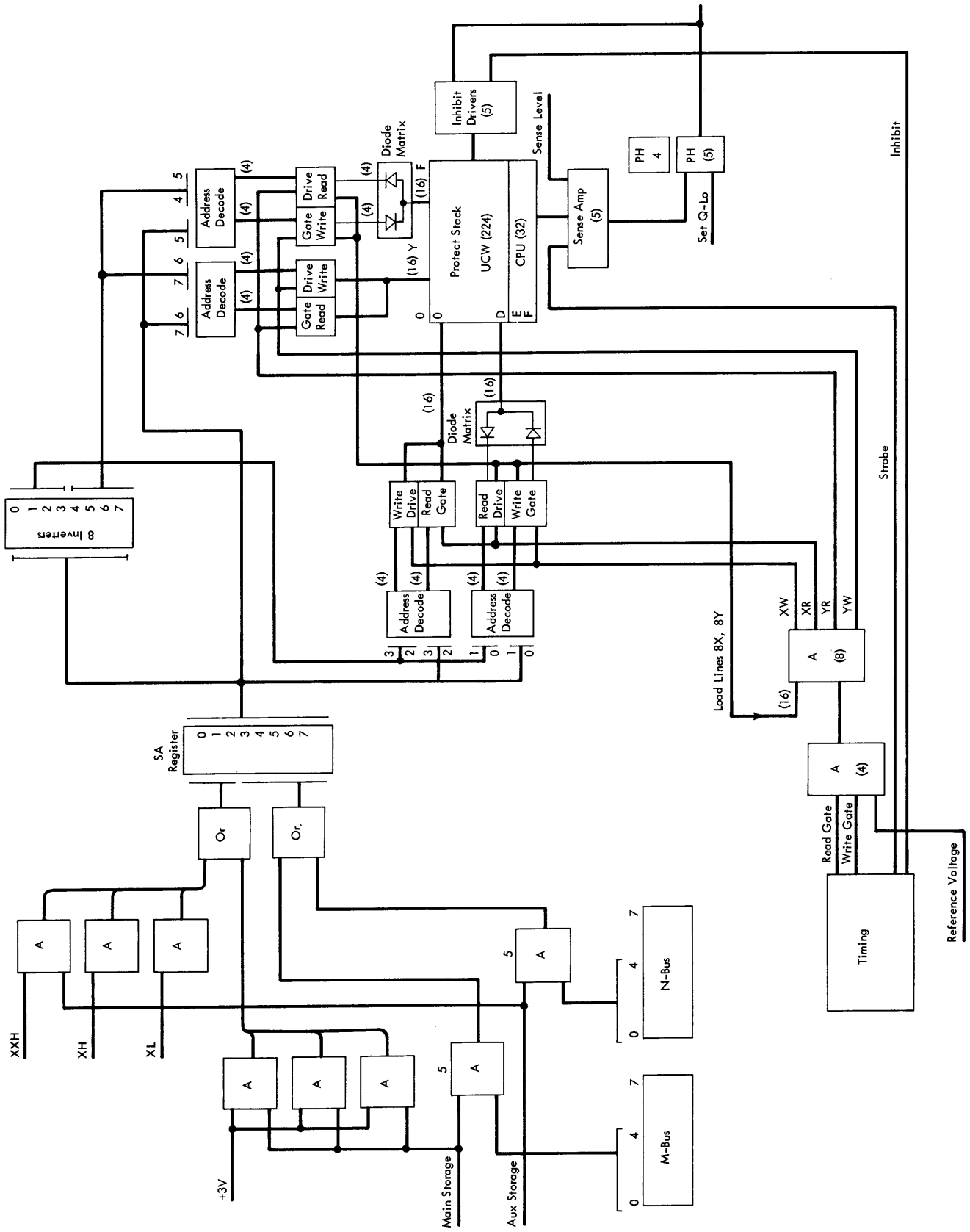


Figure 4-11. Protect Stack Block Diagram

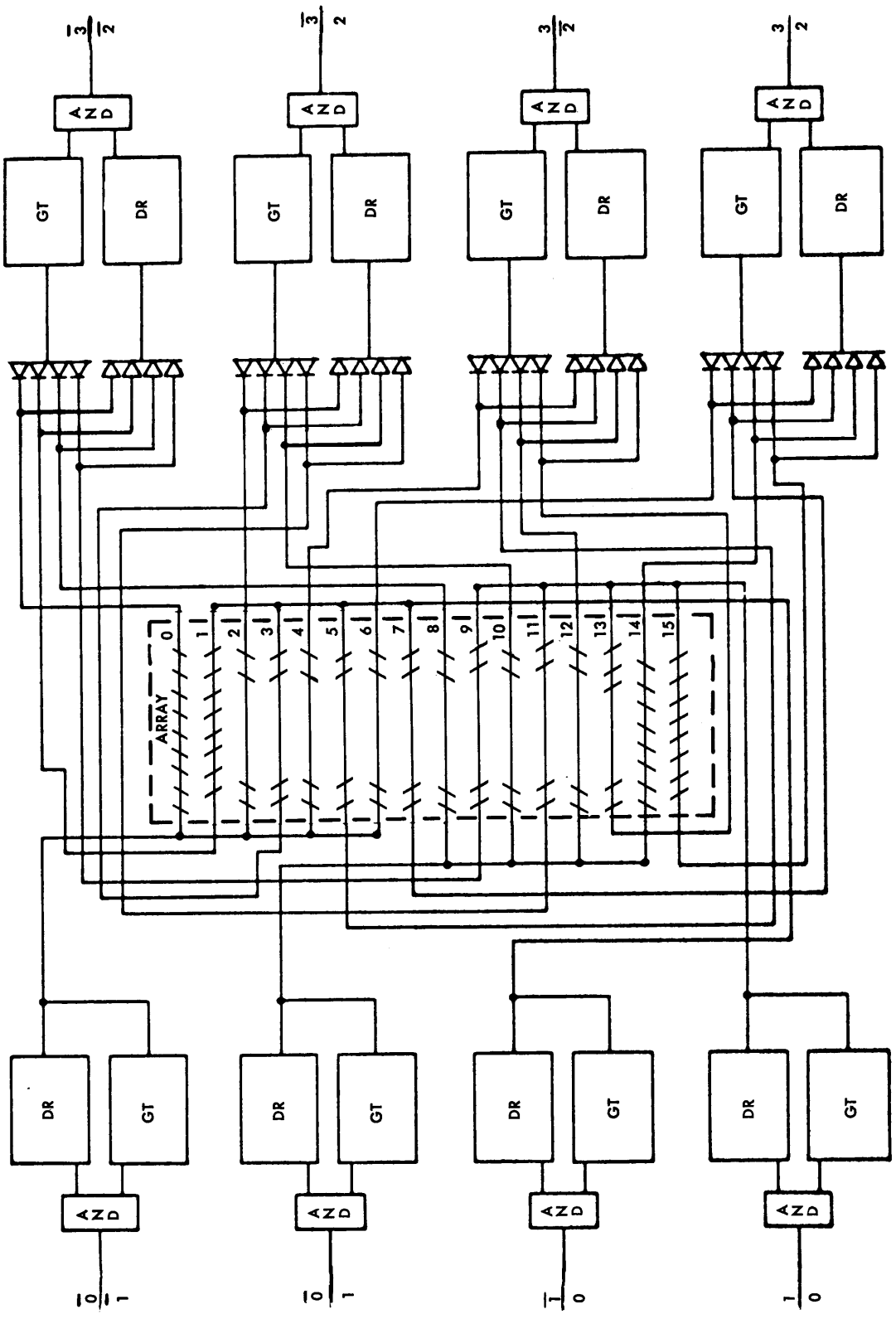


Figure 4-12. X-Diode Matrix Line Selection

The lines that feed the inhibit drivers are controlled by the data to be written and the inhibit gate (Figure 4-13).

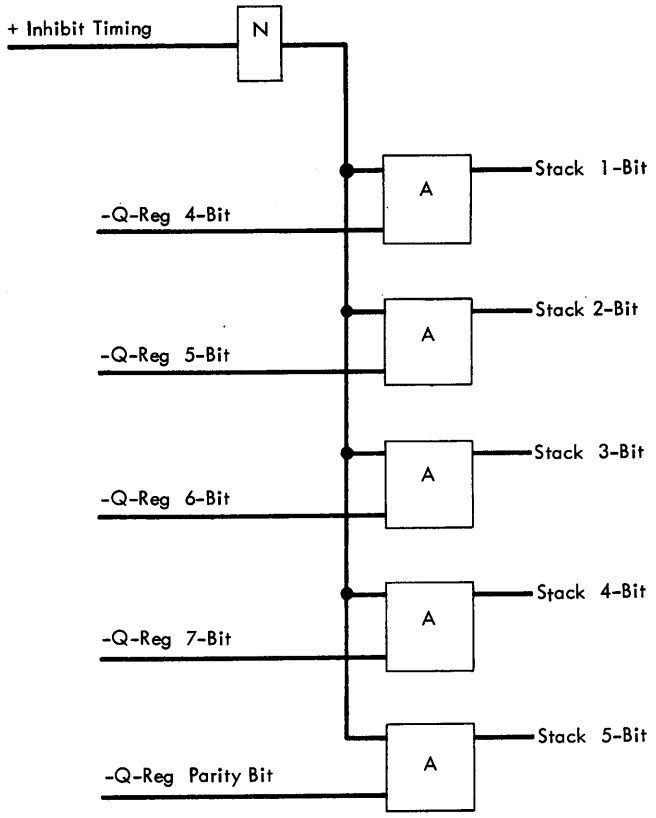


Figure 4-13. Inhibit Control

SPECIAL VOLTAGES

- There are two special voltages: reference voltage to the driver-gate lines and sense level voltage to the sense amplifiers.

The reference voltage (Vref) provides a temperature compensated output voltage to the driver-gate control card, which in turn regulates the amplitude of the driver gate output current.

The sense level circuit provides a voltage to the second stage of the sense amplifier. This voltage is adjusted to provide the best discrimination between a maximum ZERO and a minimum ONE.

THEORY OF OPERATION

DATA FLOW

- The SA register address is decoded to select the desired position.
- The position is read from core and set in Q-Lo.

- The output of Q-Lo is routed to the inhibit drivers, and during write time the key is written in the core position.

The address set in the SA register is routed and decoded so one position of the protect stack is read out. The output of the stack is sent to the sense amplifiers and then set in Q-Lo. During the write cycle, the output of Q-Lo and parity bit is conditioned by inhibit timing and the key in Q-Lo is written back in the protect stack (Figure 4-14).

STORAGE PROTECTION TIMING

- The timing of the storage protection feature depends on the CPU operation.

Figure 4-15 is a timing chart for storage protection. The dotted lines indicate the status of the line if a protection check is recognized. Notice that even if a protection check occurs it is not allowed unless main storage is to be changed.

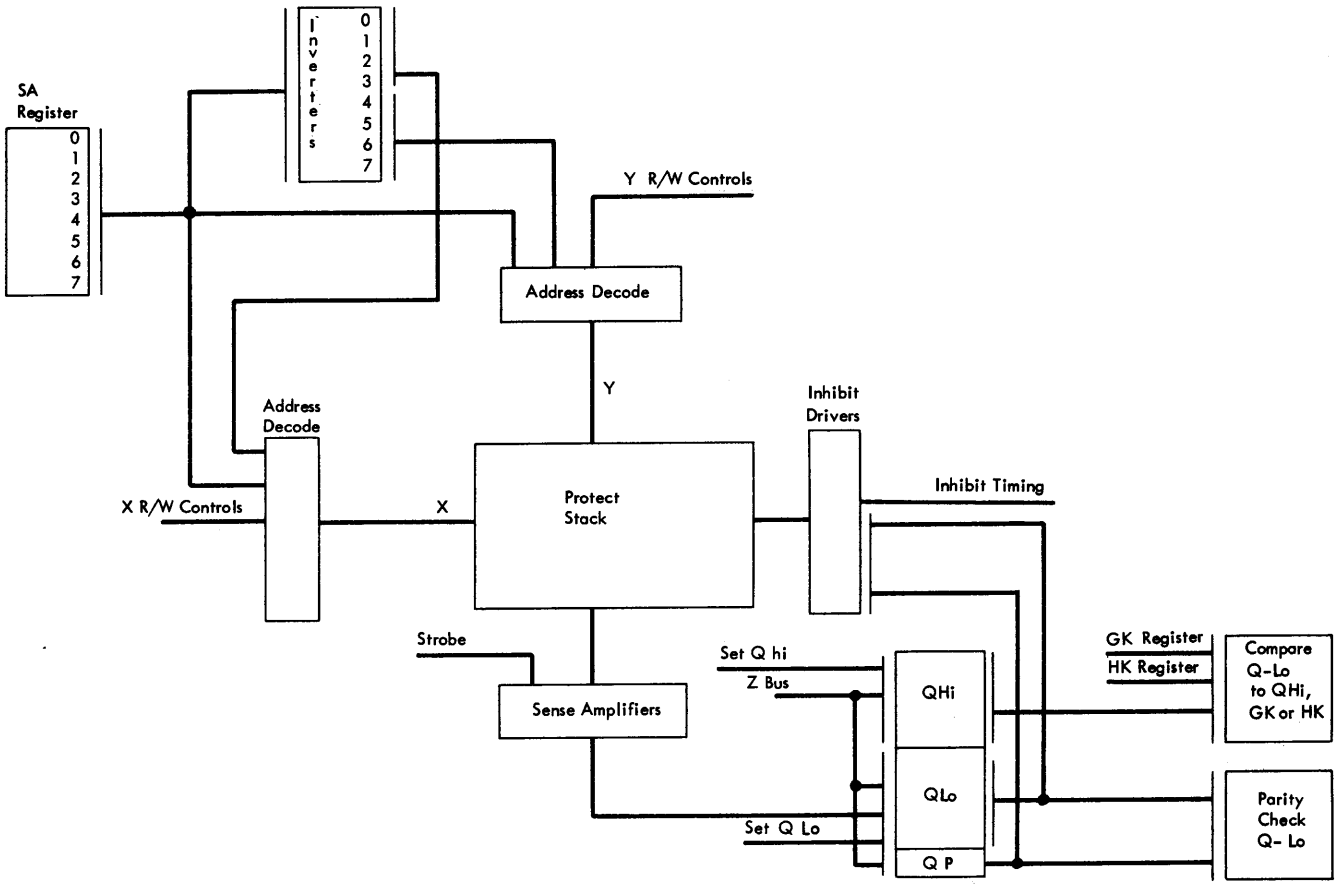


Figure 4-14. Data Flow

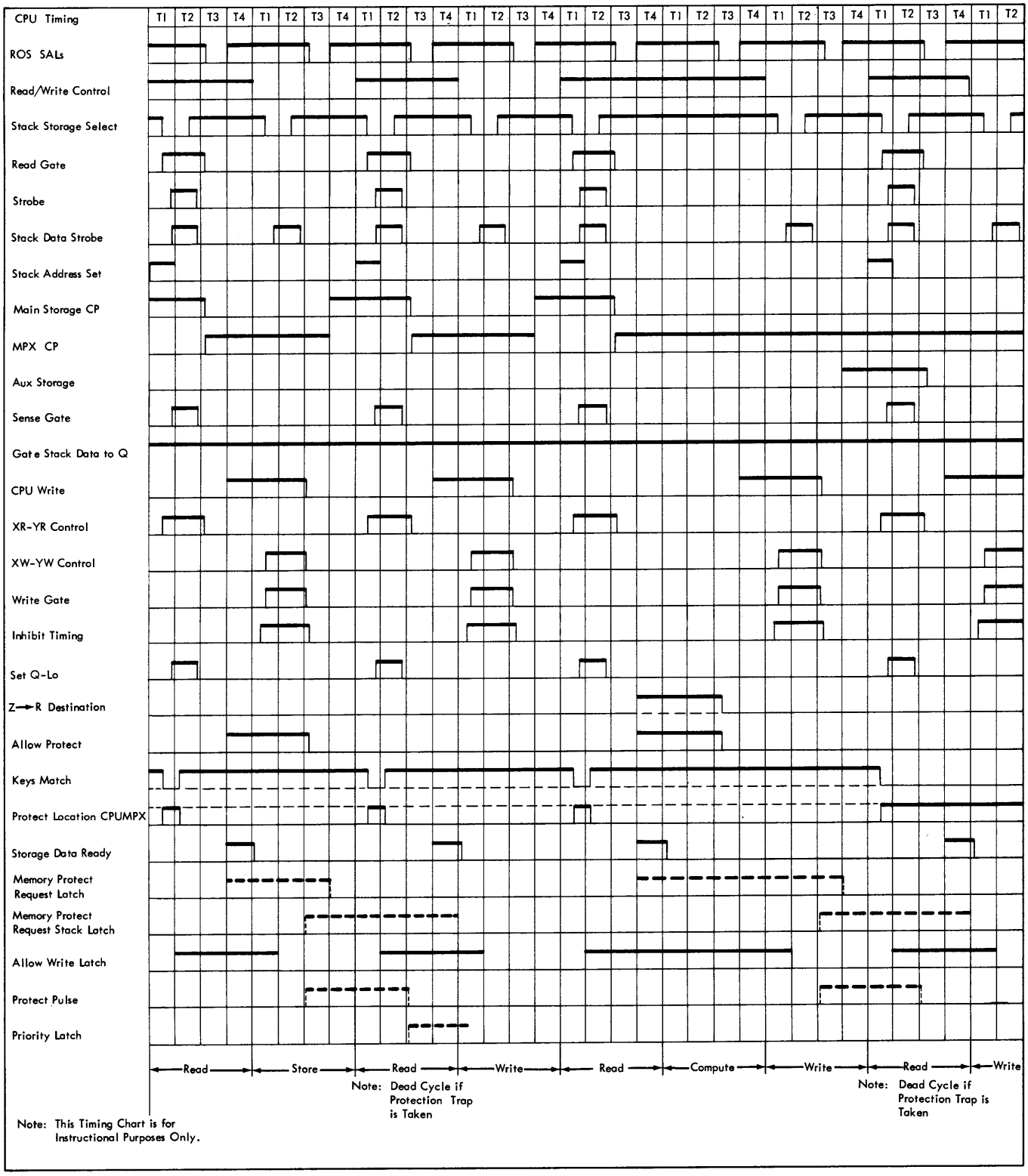


Figure 4-15. Storage Protection Timing

STORAGE KEY

- The storage key is set into the protect stack by the privileged instruction SET STORAGE KEY.

The storage key assigned to each 2,048 byte block of main storage is usually set in the protect stack during the first part of the supervisor program.

The desired storage key is in bits 24-27 of the general register specified by the R1 field of the instruction (Figure 4-16). The block of storage (to which the key is assigned) is determined by the address in the general register specified by the R2 field.

The key is read from R1 and set in the D-register. Main storage is addressed by the address set into the UV registers from R2. The Q-register is set from the D-register so the storage key is in Q-Lo. The information read from memory is written back and at the same time the storage key in Q-Lo is written in the protect stack at the correct location corresponding to the main-storage block.

If it is necessary to find out what number has been assigned to a block of storage, the privileged instruction, INSERT STORAGE KEY, is used (Figure 4-17). This instruction takes the address in R2 and sets the address in UV, main memory is read out and the storage key is set in Q-Lo. The Q-register is then crossed high and set in the D-register. The third byte of the register in R1 is addressed and the D-

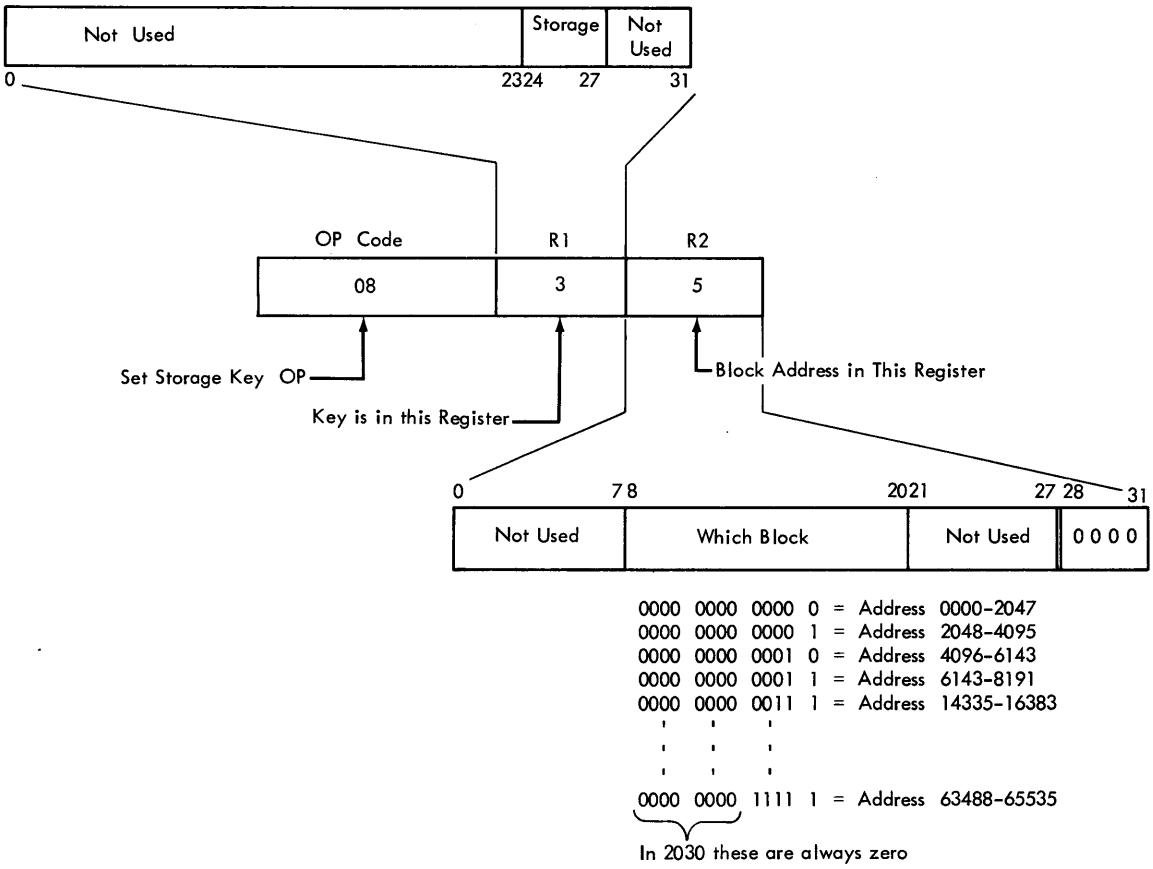
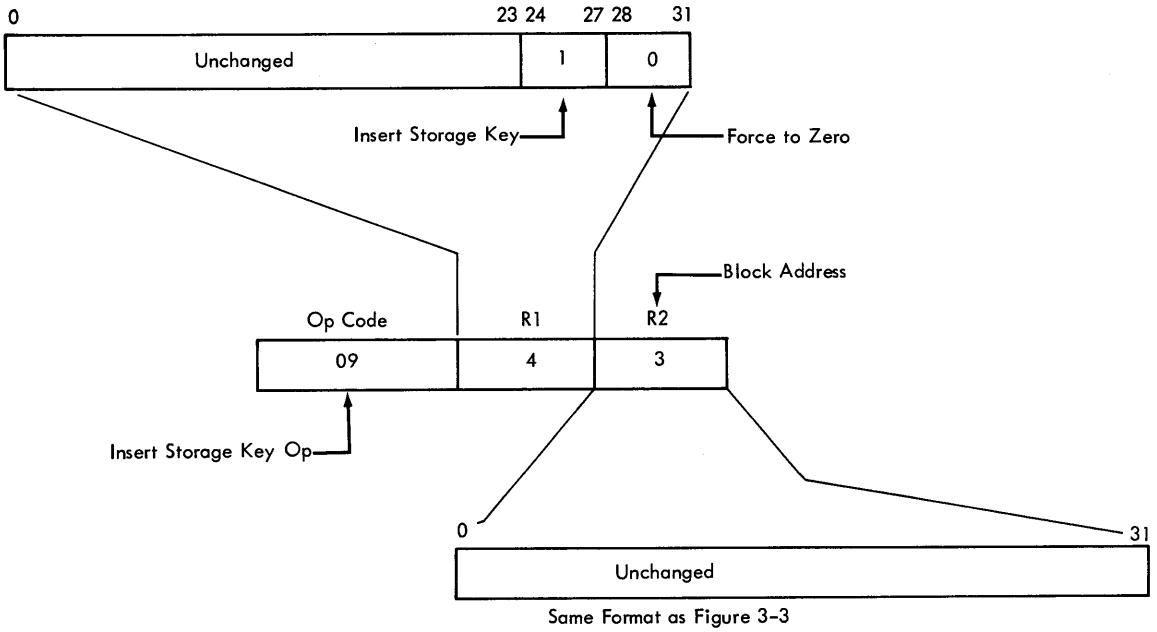


Figure 4-16. SET STORAGE KEY Instruction Word Format

register set into the R-register. The next ROS cycle stores R-register in the third byte of R1. A record has now been made of the storage key assigned to that block of storage.



Storage Block	Key	Register 4	Register 3
2048-4095	1	87004321 ← Before →	00000E00
0000-2047	F	87004310 ← After →	00000E00

Figure 4-17. INSERT STORAGE KEY Instruction Word Format

PROTECTION KEY

- The protection key for the CPU is found in bits 8-11 of the PSW.
- The protection key for the PSW set by the last LOAD PSW instruction is found in Q-Hi and at local store location K25 (address B9).
- The protection key for a multiplexor channel is read from the CAW and set in the protect stack as an extension of the UCW for that subchannel.

The LOAD-PSW routine stores the protection key in bits 8-11 of the PSW in local store at location K25 (address B-9). This serves as a permanent record of the PSW protection key. It is then set in Q-Hi from the Z-bus, and is used for matching against the storage key assigned to main storage locations accessed during that program.

When a START I/O instruction for the selector channel is performed, the protection key is read from bits 0-3 of the CAW.

The protection key is then set into the GK or HK register, depending on the channel selected. The output of GK or HK is matched against the storage key read from the protect stack when main storage is accessed. A mismatch is recognized only when the operation is an input to main storage.

When the START I/O instruction for the multiplexor channel is performed, the protection key is read from bits 0-3 of the CAW. The protection key is then temporarily stored in the U-register. As soon as the subchannel address (UCW- address) has been established, the protection key is set into Q-Lo, and then written in the protect stack. The same position of the protect stack is read out each time the same UCW is read out. In effect, the 4-bit protection key is an extension of the UCW in MPX auxiliary store. Now, each time that I/O device is selected on the MPX channel, the protection key is read out.

Since the protection key for the CPU is in Q-Hi, it is necessary to replace it with the protection key in MPX local store whenever the I/O unit is selected. The UCW protection key is read from the MPX store and set into Q-Lo. It is then transferred to the I-register for temporary storage. This is done because Q-Lo can be changed on the next cycle under certain conditions and the protection key lost. From the I-register, the protection key is transferred to Q-Hi. This destroys the CPU protection key and requires at the end of the multiplexor channel operation that the CPU protection key in local store K25 (B9) be read out and set in Q-Hi for correct operation. During the multiplexor channel operation when main storage is accessed, the storage key is read from the protect stack and set in Q-Lo. Only if main storage is to be changed is a mismatch of Q-Hi and Q-Lo recognized.

PROTECTION EXCEPTION

- A protection exception occurs for a CPU, or multiplex channel operation when a mismatch between Q-Hi and Q-Lo occurs or a parity error on Q-Lo is found, and:

  1. The protection key is other than 0.

  2. Main storage is to be changed during write time.

- A protection exception occurs for a selector channel operation when a mismatch between GK or HK and Q-Lo occurs or a parity error on Q-Lo is found and:

  1. The protection key is other than 0.

  2. The operation is an input cycle.

For a CPU or multiplexor operation, a protection exception is allowed if the operation is a store or the R-register is the destination of the Z-bus. A parity error on Q-Lo or a mismatch on Q-Lo and Q-Hi is recognized if main storage is addressed. If local store is accessed, any protection error is prevented from being recognized. If the protection exception is allowed, an interrupt occurs and the protection exception code (0004) is placed in the old PSW for CPU operation or a bit set in the CSW and/or the UCW for I/O operation (Figure 4-18).

During a selector channel operation if a protection exception is allowed and the operation is an input cycle, the protection check latch is turned on. An interrupt occurs and bit 43 of the CSW is set to 1.

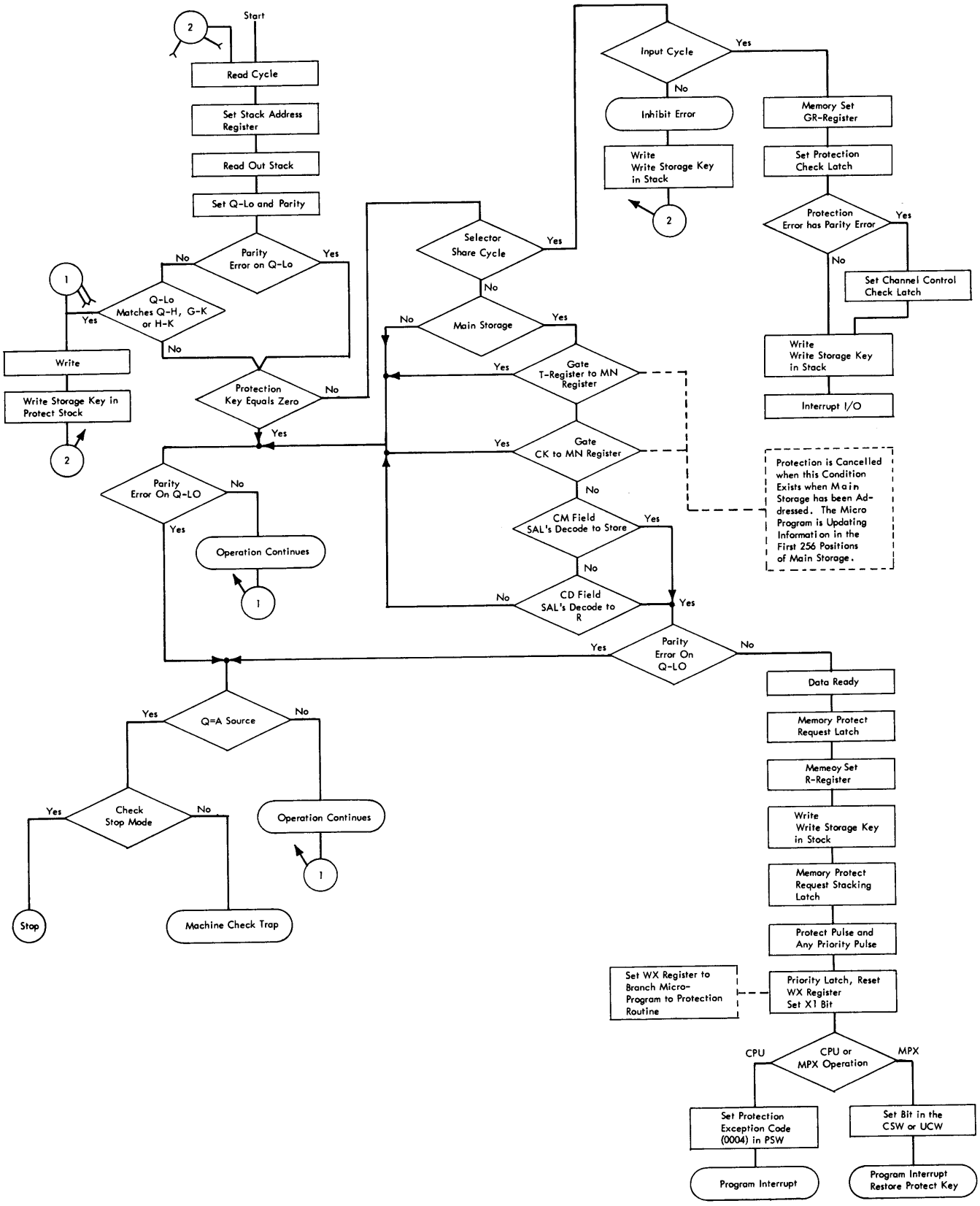


Figure 4-18. Protection Exception

MAIN STORAGE OPERATION

- The 5 bits read from the stack (storage key and parity) are set into Q-Lo.
- A parity check is made on Q-Lo.
- A parity error can cause a protection check and/or a machine check unless the protection key in Q-Hi is 0.
- A comparison between Q-Hi (protection key) and Q-Lo (storage key) is made.
- An unequal comparison is ignored if:

  1. Q-Hi is set to 0, or
  2. Storage is to stay the same

The protection key has been stored in location K25 (B9) of local store and in Q-Hi (Figure 4-19). When the ROS micro program requests a read cycle in main storage, the storage key assigned to that main storage block is read out of the protect stack. The key is set into Q-Lo and the stack-parity bit into the parity bit position of the Q-register.

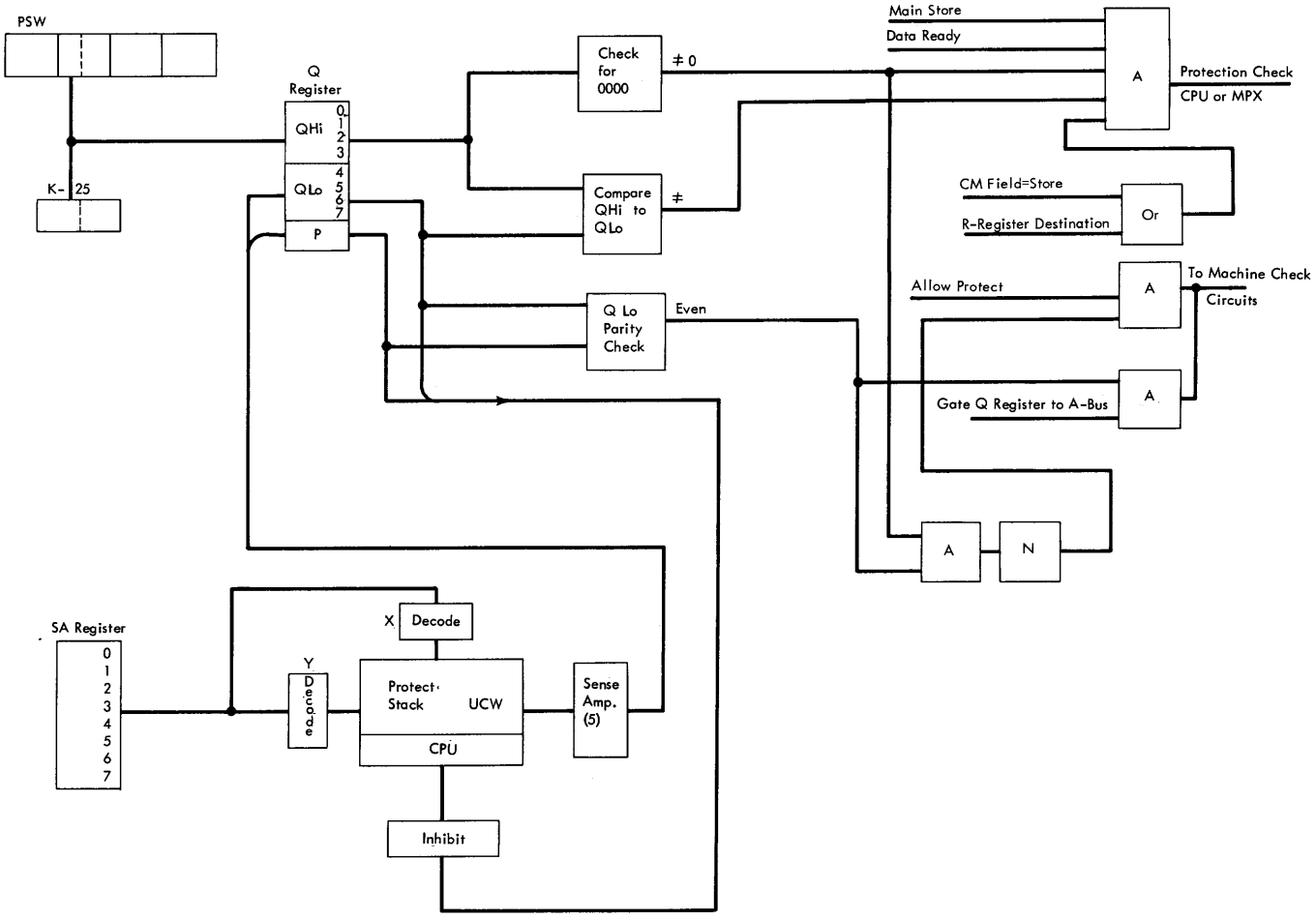


Figure 4-19. CPU Operation (Logical Diagram)

A parity check is made on Q-Lo and the parity bit. If the parity is even, a store data will occur if Q-Hi is not 0 and allow protect is up. If even parity exists when the Q-register is used as an A-register source, and a storage data check is forced to prevent the incorrect transfer of keys.

If a storage data check and protection check occur at the same time, the machine check trap takes priority over the storage protection trap.

A comparison is made between Q-Hi and Q-Lo. An unequal comparison causes a protection check; unless the protection key in Q-Hi is 0 or the information read out of main storage is to be written back.

If a mismatch or parity error occurs and it is allowed, the information read out of main storage is set into the R-register and written back into main storage during the write cycle. An interrupt is taken, and the protection exception code (0004) is placed in the old PSW to identify the program interrupt.

The storage key is regenerated in the protect stack from Q-Lo and parity bit during the write cycle.

SELECTOR CHANNEL OPERATION

- The protection key is read from the 0-3 bits of the CAW and set into GK or HK.
- When main storage is addressed for a selector-share read cycle, the storage key is read from the protect stack and set into Q-Lo plus parity bit.
- A parity check is made on Q-Lo. Even parity can cause a selector channel protection check and a channel control check unless the protection key is GK or HK is 0.
- A comparison is made between GK or HK (protection key) and Q-Lo (storage key). If unequal, a protection error is recognized.
- An unequal comparison is ignored if:

  1. GK or HK is set to 0, or
  2. Storage is to stay the same.

When the START I/O instruction is recognized, the protection key (0-3 bits) of the CAW is set into the GR or HR register depending which channel is being used (Figure 4-20). Assume selector channel 1 is being used. The protection key is then set into the GK register from GR, and retained there until the next time the ROS microprogram has the statement GR->GK.

On the cycle that main storage is addressed for a selector share read cycle, the protect stack is read out and set in Q-Lo. A parity check is made on Q-Lo and the parity bit. If parity is even, a selector channel protection check and a channel control check occurs unless the protection key in the GK's is 0000.

If a protection check occurs and is allowed, the recognition of the selector channel protection check is further conditioned that the operation is an input cycle. When all conditions are satisfied to cause a protection check, the information read from storage is placed in the GR-register (channel data register) and is regenerated in main storage during write time.

When a protection check occurs, the protection check latch and possibly the channel control check latch in the selector channel are set. The operation on the channel is terminated, and an interrupt is indicated. When the CSW is stored, a protection check is indicated by bit 43 set to 1 and a channel control check, if it exists, by bit 45 set to a 1. These latches are reset when the CSW is stored.

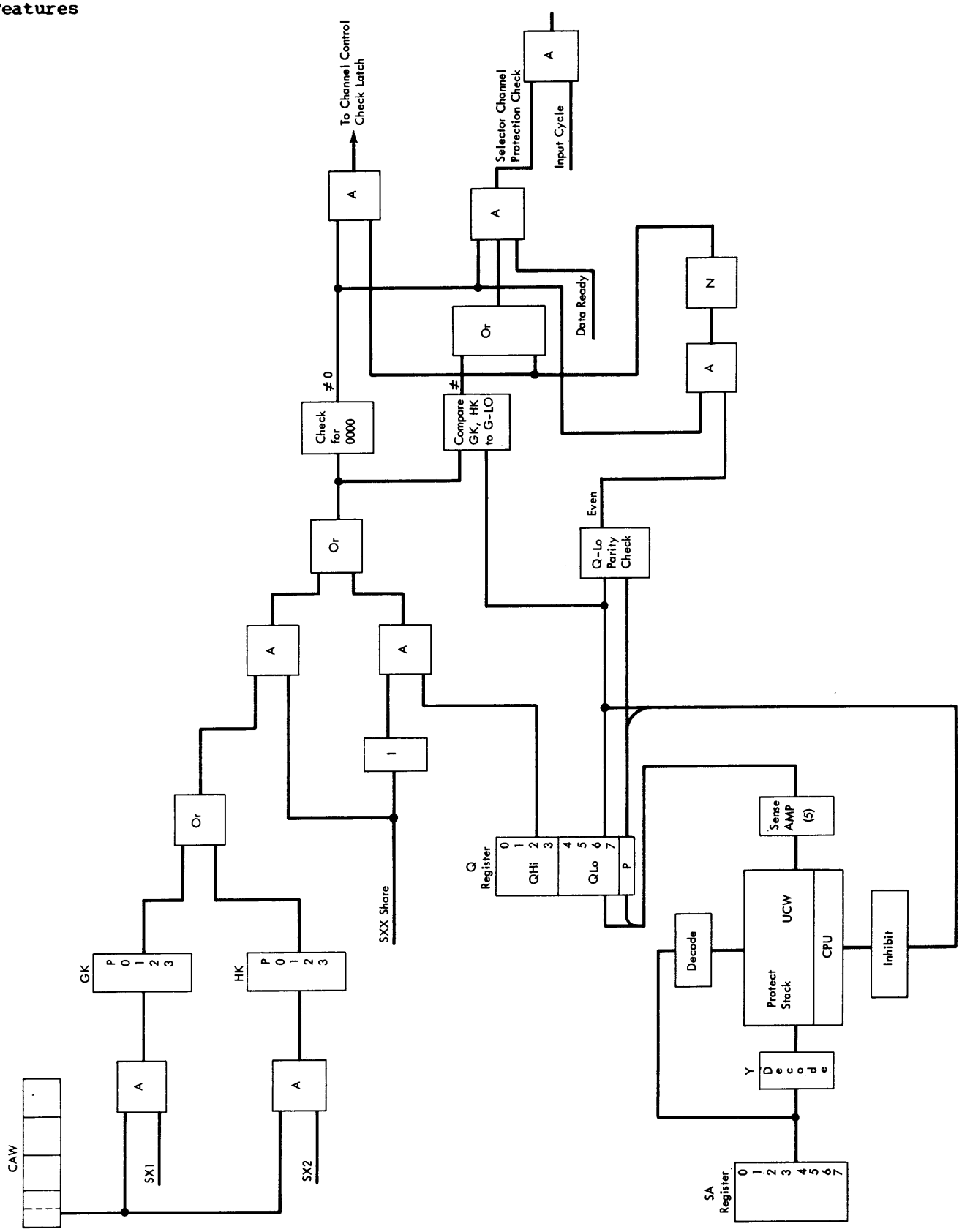


Figure 4-20. Selector Channel (Logical Diagram)

Multiplexor-Channel Operation

- The MPX-channel can have more than one I/O device running at one time. Thus, it is possible to have more than one protection key to remember.
- The I/O protection key is read out and set in Q-Lo when the I/O unit is initially selected, and then transferred to the protect stack.
- The protection key for each I/O device on the MPX channel is retained in the protect stack as an extension of the UCW.
- Each multiplexor share operation causes the protection key to be read from the UCW portion of the protect stack and set in Q-Lo and then transferred to Q-Hi.
- The storage key is read out when main storage is addressed for a read cycle, and set in Q-Lo.
- A parity check is made on Q-Lo. If even, machine check will occur if Q-Hi is not 0 and allow protect is up. A machine check will also occur if a Q-Register is gated to the A-Register and Q-Lo is even parity.
- A comparison is made between Q-Hi and Q-Lo. If unequal, a protection error is recognized.
- The protection check is ignored if:

  1. Q-Hi is set to 0, or
  2. Storage is to stay the same.

Because the MPX channel can have more than one I/O device running simultaneously, it might be possible to have more than one protection key. The protection key for each I/O unit has been placed in the stack at a location that is addressed by the I/O unit UCW address.

Whenever an I/O operation is initiated on the MPX channel which requires reading the CAW, the associated protection key is temporarily stored in the U-register. The key is then set into Q-Lo, and then in the protect stack (Figure 4-21).

The MPX channel operates in two modes: burst and interleave. The procedure used to read the protection key from the protect stack and set it in Q-Hi is similar in each case, but is found in different areas of the microprogram. The I/O device is selected and the UA, (Unit Address) for the device provides the address for the SA register. The protection key is set in Q-Lo, and then in the I-register for temporary storage. This is done because Q-Lo is changed each read cycle, and in some cases the correct protection key could be lost. The key is then set into Q-Hi from the I-register. Now, when main storage is addressed for a read cycle, the storage key is read from the protect stack and set in Q-Lo.

A parity check is made on Q-Lo and the parity bit. If parity is even, a protection check and/or a machine check can occur, unless the protection key in Q-Hi is 0.

A comparison is made between Q-Hi and Q-Lo. An unequal comparison causes a protection check; unless the protection key in Q-Hi is 0 or the information read out of main storage is to be written back.

If a protection check occurs and it is allowed, the information read out of main storage is set into the R-register and written back into main storage during the write cycle. An interrupt is taken and the protection error sets bit 43 in the CSW.

When the MPX operation ends, the protection key for the CPU is read from local store K25 (B9) and set in Q-Hi. This happens at the same time that the rest of the registers are being restored to the information held prior to the MPX share operation.

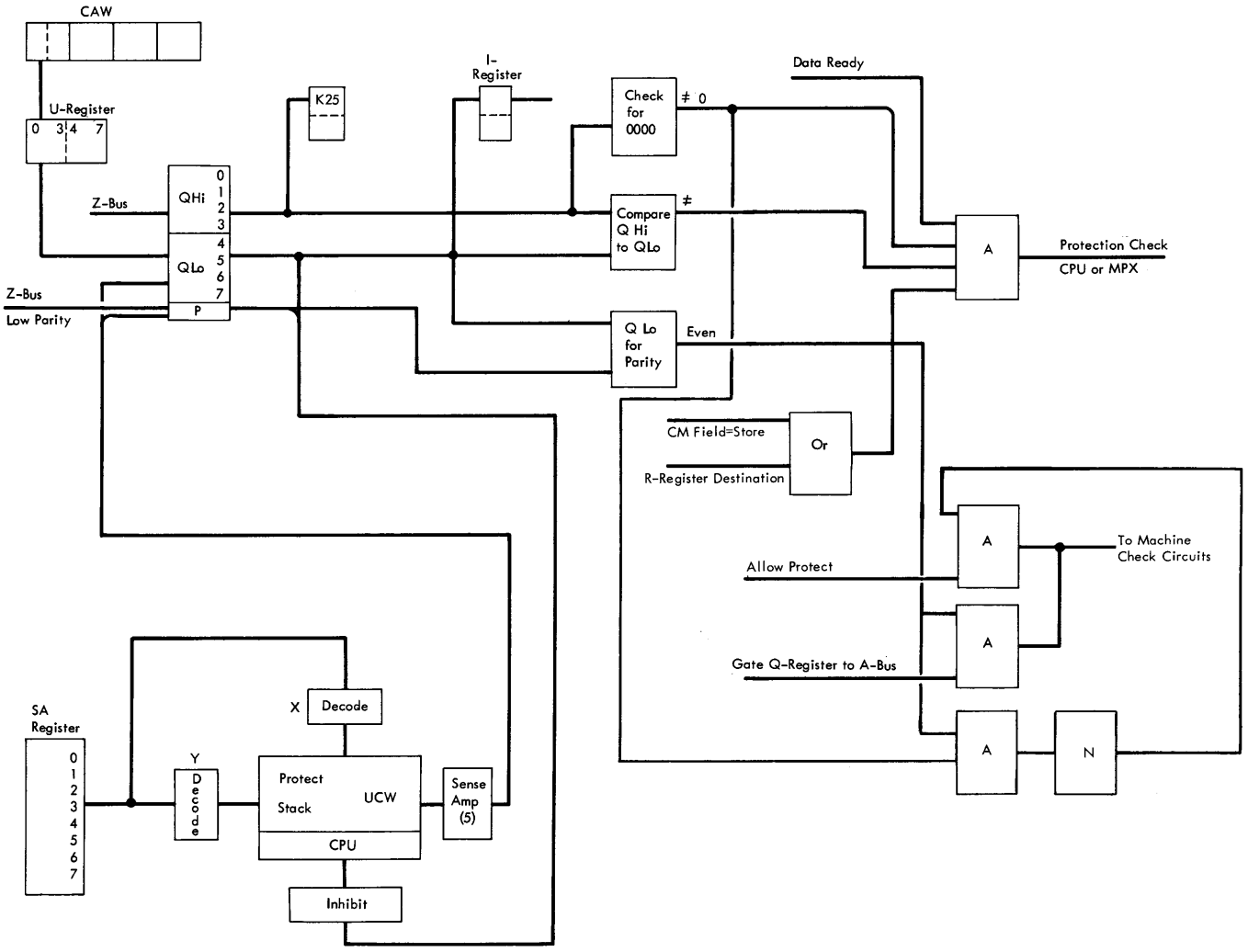


Figure 4-21. Multiplexor Channel (Logical Diagram)

INTERVAL TIMER

- The interval timer feature consists of three bytes in main storage locations 50, 51, and 52.
- The value in the timer is decreased for intervals of time.
- An external interrupt is signaled when the timer goes from a positive to a negative value.

Let's consider an application of the interval timer feature. Assume that a customer must run two jobs during the day. Job #1 takes seven hours. The information to run Job #2 is not available until 2 P.M. By using the interval timer feature, the customer, in effect, can instruct the System/360 to stop working on Job #1 and start on Job #2 at 2 P.M. If the customer knows that Job #2 is usually completed in 15 minutes, he could set the timer for 17 minutes. This would allow a 2 minute safety margin. At the end of 17 minutes the work on Job #2 is halted (whether completed or not) and processing of Job #1 is resumed.

To use this feature, a certain value is set in main storage locations 50, 51, and 52. This starts a counter that keeps track of time. The value that is set in main storage represents total elapsed time. When the counter value is subtracted from the timer value often enough, the timer value goes from a positive to a negative value. At this time, an external interrupt is taken to whatever has been previously set up by the customer. In our example, it would be the start of a routine to handle Job #2.

Let us examine the computation of timer values. The high-order bit of location 50 is reserved for sign control. This leaves 23 bit positions free for data. A value of over 16,700,000 can be set with 23 bit positions.

A microprogram routine subtracts 300 from the timer during each second of elapsed time. Thus, the full cycle time of the timer is about 15.5 hours. Because 300 is subtracted from the timer for an elapsed time of one second, then the timer must be set to the value 1,080,000 for each hour (300/sec x 60 sec x 60 min) of elapsed time that is desired.

60 CYCLE OPERATION

- The value that is subtracted from the timer is determined by the setting of the 4 position binary connected C counter.
- The C counter is driven at a 60 cycle rate.
- The C counter is FULL every .25 seconds.
- A latch ON in the C counter causes a timer update at the end of E-phase.

The C counter keeps track of actual time. A 60 cycle pulse provides the drive. Though this 4-position counter is FULL with only 15 impulses (.25 sec), any position of this counter that is set at the end of E-phase causes a timer update.

The update routine takes the value in the C-counter, multiplies it by 5 and subtracts the product from the timer value. If there is a sign change as a result of the subtraction, an external interrupt is taken. If no sign change occurs, the update routine exits to I-phase for the next instruction.

For .25 seconds, the value 75 must be subtracted from the timer. The C counter has four positions. Therefore, the highest value it can contain is 15 (1111, all positions set). All positions of the C counter are set in .25 seconds. Therefore, the full value in the counter (15) multiplied by 5 gives the value (75) that must be subtracted from the timer for this (.25 sec) elapsed time.

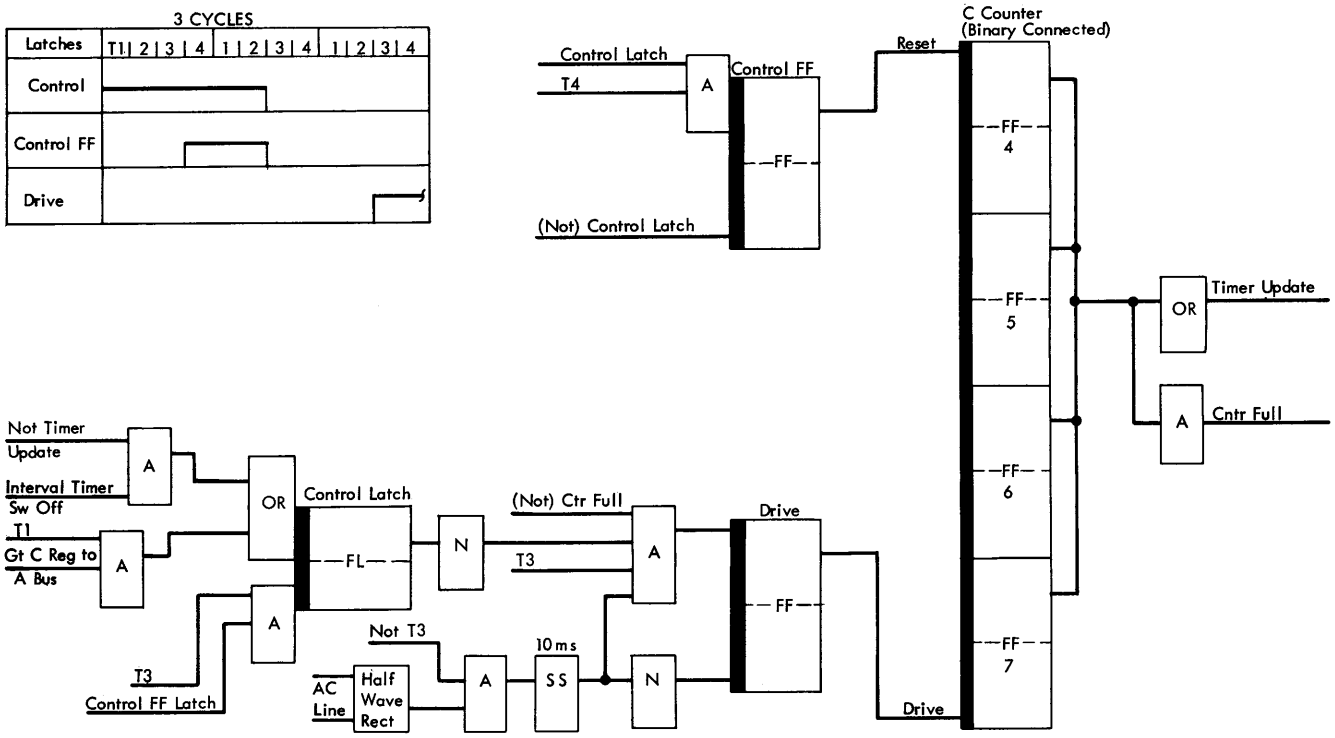


Figure 4-22. Interval Timer Controls

The controls for the C counter are shown in Figure 4-22. The governing latch is the control latch. This latch must be off to allow the C-counter to run. If the disable-timer switch is off, the control FF latch is set at 4 TIME. This latch controls the lines to reset the C-counter latches. The control FF latch turns the control latch off at T3 time. The control latch going off resets the Control FF latch.

With the control latch off and the C-counter empty, the 60 cycle time pulse sets the drive latch at T3 time. The drive latch provides one pulse at a time to the C-counter. Two lines are developed from the C-counter:

1. Counter Full--Blocks further drive pulses to the C-counter by not allowing the drive latch to be set.

2. Timer Update--If any position of the C-counter is set, this is active and a timer update is signaled.

During the update routine, the setting in the C-counter is set into the D-register (C->D) for multiplication by 5. This causes the control latch to be set ON again. The C-counter is reset so that it may again start counting the timed pulses. A flow chart of the timer update routine is shown in Figure 4-23.

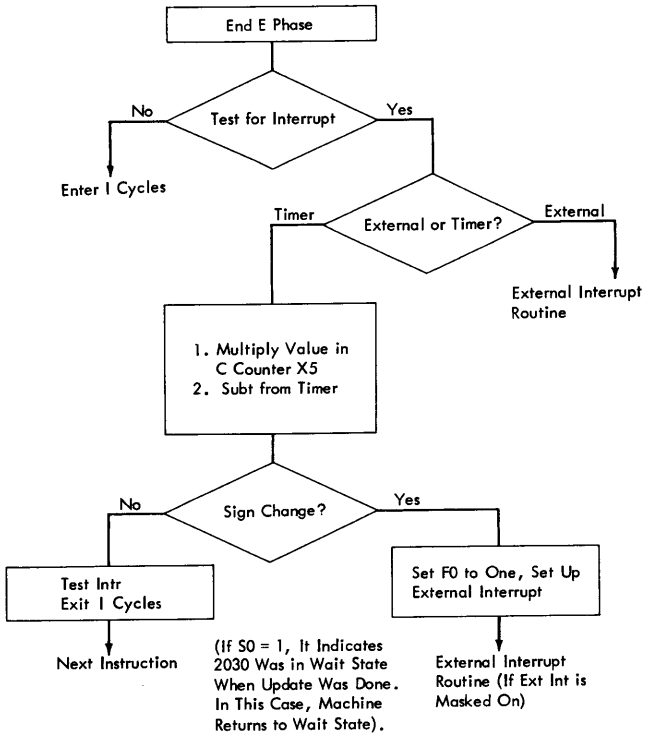


Figure 4-23. Timer Update Microprogram

50 CYCLE OPERATION

- The C-counter is driven at a 50 cycle rate.
- The C-counter is full every .3 seconds.
- Counter value multiplied by 6 is subtracted from the timer during timer update.

If the 2030 is operating on 50 cycle alternating current, a slight change in the timer-update-controls is necessary. Instead of multiplying the counter value by 5 to get the correct number to subtract from the timer, the 50-cycle-machine-timer-update-routine multiplies the counter value by 6. This is necessary because on a 50 cycle machine, the counter is full every .3 seconds instead of every .25 seconds.

By using a multiplier of 6 for 50 cycle machines, and a multiplier of 5 for 60 cycle machines, the same value (300) is subtracted from the timer on all machines. This makes timer programming compatible for all machines.

1401/1440/1460 COMPATIBILITY

- The compatibility feature and associated subfeatures enable a rapid and simplified transfer from 1400 applications to the IBM System/360 Model 30.
- 1400 programming systems operate without change using this feature with appropriate I/O devices.
- I/O Subfeatures are microprogram routines that operate particular I/O devices.

The 1401/1440/1460 Compatibility Features consist of the 1401/1440/1460 Basic Compatibility Feature and the appropriate subfeatures required to provide compatibility with an existing 1401, 1440, or 1460 system configuration. Compatibility features permit 1401/1440 and 1460 object programs, using comparable I/O devices included for the feature, to be executed on a System/360, Model 30, without modification or additional storage requirements.

The 1400 object program is stored in upper 2030 storage. Address conversion and character conversion occurs as needed. The original 1400 program sequence and character configuration is not disturbed. 1400 Mode microprograms perform the same functions in 1400 Compatibility Mode as hardware performed in 1400 systems.

A System/360, Model 30 with at least equivalent core storage and comparable I/O devices, can assume the functions of any 1401, 1440, or 1460 system having the following IBM I/O units:

1402 Card Read-Punch

1442 Card Read-Punch

1403 Printer

1404 Printer (Continuous Forms Operations)

1443 Printer

1407 Console Inquiry Station

1447 Console Inquiry (Models 2 or 3)

1311 Disk Storage Drive

729 or 7330 Magnetic Tape Units

7335 Magnetic Tape Unit

The Compatibility subfeatures that are available for emulating specific 1400 configurations are as follows:

Column Binary

1402/1403 Attachment

1442/1443 Attachment

Console Inquiry Station

Disk Storage Drives

Magnetic Tapes

The programmed mode switch subfeature (PMS) is also available. No other compatibility subfeature is either required or pre-empted by this subfeature. PMS provides the ability to switch the 2030 processor from compatibility mode to 2030 mode, and vice versa under 2030 program control. This permits the use of System/360 capabilities and devices that are not otherwise available in compatibility-mode operations. The PMS feature must be factory installed and is only available on a 2030 having 16,384 or more positions of core storage (Models D, E, or F).

IMPLEMENTATION

- A separate and distinct (4K) 1400 Read Only Storage (ROS) control is incorporated to control data flow in a manner that emulates the 1400 systems.
- The standard System/360 ROS is not used when in compatibility mode. The W3 bit on in the WX Reg causes the added 4K ROS to be addressed.
- IJ, LT and UV perform the functions of I-Star, A-Star, and B-Star, respectively.
- Auxiliary storage is loaded with conversion tables, constants, and other control factors required by ROS to absorb the differences in code structure and storage addressing between the 1400 Systems and the System/360.

The 1401/1440/1460 compatibility feature consists physically of a second 4K ROS unit and 5 SLT Cards (Figure 4-24) that are added to the 2030 processing unit. This additional ROS contains the general microprogramming necessary to process 1400 system instructions and the specific subfeatures for controlling the various I/O devices. The 2030 is put into compatibility mode by turning on the W3 bit of the W-register. This bit causes the added 4K ROS to be addressed and controls all mode dependent functions. The standard System/360 ROS is not used when in compatibility mode.

W3 can be turned on (placing the 2030 in 1400 compatibility mode) in three ways:

1. Console switches. Console switch F set to an odd hex digit turns on W3.
2. Micro program control UV->WX. The status of U3 determines W3 when the micro program statement UV->WX is used.
3. Micro program control CA->W. The status of AA determines W3 when the microprogram statement CA->W is used.

Any system reset turns off W3 and causes the 2030 to leave compatibility mode. Recycle reset, the microprogram control K->W, and priority trapping do not affect the setting of W3 when the compatibility feature is present on the machine.

When operating in compatibility mode, IJ performs the functions of the I-Star, LT performs the functions of the A-Star and UV performs the functions of the B-Star. Circuitry is provided so that the L and T registers (though normally separate) may be gated as a pair into the MN register when the T register is named as the source (i.e.: T->N MS causes the address contained in LT to gate to MN).

ROS Field	Decode	Normal Function	1400 Compatibility Function
CH	0011 1000	V00 S1	GMWM R2
CL	0110 1100	1BC G1	R1 R3
CM	0101	T	LT

Note: The above control field functions changes are accomplished by five additional SLT cards as follows:

Frame - Gate	Board	Socket
01A	A1	F5
01A	A1	B7
01A	B2	C2
01A	B2	B3
01A	A2	J2

Figure 4-24. ROS Control Field Changes; Additional SLT Cards

Auxiliary storage, which normally provides residence for general purpose, floating-point, condition registers, and multiplexor channel Unit Control Words (UCW's), is also loaded with conversion tables, constants, and other control factors required by ROS control during compatibility mode operation. In addition to constants and conversion tables, considera-

ble information such as storage size, tape densities, unit addresses and other characteristics unique to a particular 1400/System/360 emulation are entered into auxiliary storage.

Auxiliary storage also provides variable data that is used by microprogramming. Some examples follow:

Status of 1400 instruction counter, A-Address, B-address and A register.

The 1400 condition bits (high-low-equal, tape error, overflow, end of file, etc.)

Sense-switch and check-stop switch settings.

Programmed mode switch control information.

Refer to Auxiliary Storage for information on contents and function of specific areas of auxiliary storage.

COMPATIBILITY INITIALIZATION

- Compatibility initialization is accomplished by Programs and information supplied by the compatibility Initialization Deck (CID).
- The CID must be modified to fit the configuration with which it will be used.
- A Diagnose (83) instruction enables the Compatibility Feature Initialization Mode (99) instruction.
- Initialization duplicates the 1400 load routine.
- Initialization loads 512 bytes of Auxiliary Storage with conversion tables, constants, etc.
- Initialization turns on W3 of the WX register to cause 1400 mode ROS to be addressed.

A compatibility initialization program executed in the 2030 ahead of the 1400 object program supplies information necessary to allow the 2030 to run in Compatibility mode. A Compatibility Initialization Deck (CID) provided by IBM and supplied with data applicable to a particular system configurations performs the functions of initialization. The deck must be modified before it can be used. The CID contains its own loading routine and consists of 40 cards (41 cards for 65K systems). The first three cards are a hexadecimal loader. The next 32 cards are the hexadecimal constants used to load 1400 auxiliary storage LS and MPX for compatibility-mode operation. The last section of the CID contains routines to clear all word marks in the 1400 core-storage area, to set the 256 high-order byte locations of 65K systems to 8F, to load auxiliary storage for 1400 mode operation, and to set the compatibility mode for 1400-series program loading.

The initialization routine is started by loading the compatibility Initialization Deck into the 2030 using the normal load operation. (Refer to Console Ops.) The program contained in the hexadecimal loader cards assembles the auxiliary storage information from the 32 data cards into 512 consecutive bytes of information in 2030 storage. (Later in the program, these 512 bytes will be loaded into auxiliary storage.)

The CID initializes one of the following instruction sequences in the location specified in auxiliary storage:

1. 1001b — Load object program from 2540
2. M%G1001RB001b — Load object program from 1442
3. L%U1001Rb — Load object program from tape.

Only one of these instructions is loaded, depending on which program load device is specified.

CID card 36 (0500, cols. 1-4) is used to clear all 1400 mode word marks. The following card in 65K systems, loads the

character 8F into the 256 high-order bytes of 2030 storage.

The CID next executes a diagnose instruction (83) that enables execution of a 99 instruction (Compatibility Feature Initialize Mode). A 99 instruction is treated as invalid when not preceded by an 83 instruction. The CFIM (99) instruction loads 1400 auxiliary storage LS and MPX with the 512 bytes of information obtained from cards 4 through 35 of the CID. The $B_1 + D_1$ field of the CFIM instruction supplies the starting main storage address of the 512 bytes of information.

With auxiliary storage loaded, the W2 field of the CFIM is tested. The W2 field defines the method to be used to load the 1400 program as follows:

W2	Initialization
0	No initialization
1	1402 card load initialization
2	Tape load initialization
9	1442 Card load initialization

The initializing routine of the CFIM instruction duplicates the load routine in the 1400. For example, consider the 1402 load key operation. In 1400 compatibility initialization, the 2030 is initialized by clearing positions 001-080 (1400 equivalent), and a word mark is inserted in position 001.

The CFIM instruction generates an invalid 1400 character (8F) and inserts this character into 1400 address 000 minus 1. The actual location of the invalid character is determined by the storage sizes supplied by the CID.

The last step in initialization turns on W3 of the WX register. This puts the 2030 processor in 1400 compatibility mode. W3 on causes the compatibility mode ROS to be addressed. At this point, the Compatibility Initialization Deck with its Compatibility Feature Initialization Mode instruction has loaded auxiliary storage and defined the method of loading to be used to load the 1400 object program. The appropriate 1400 instruction is executed to read the first card or tape record of the 1400 object program. The address of the appropriate 1400 load instruction is stored in IJ backup in auxiliary storage by the CFIM instruction as 2 of the 512 bytes. The CFIM instruction retrieves the IJ backup and uses the location specified by the contents to begin processing in 1400 compatibility mode.

FUNCTIONAL DIFFERENCES BETWEEN SYSTEM/360 MODEL 30 1400 COMPATIBILITY FEATURE AND THE 1401, 1440, OR 1460.

- The System/360 in 1400 compatibility mode, treats some conditions differently than the 1400 system does.
- Differences involve the CPU and I/O devices.
- Knowledge of the differences that exist is essential to diagnostics.
- Console operations vary considerably and will be discussed in a separate section.

CPU Differences

Memory Wrap. In 1400 Compatibility mode, memory wrap indications do not occur until the address is used to address memory or store in memory by a store Address-register instruction.

In 1400 System Operation, a process check occurs immediately when an address is incremented or decremented beyond the memory size.

d-Modifer. The d-modifier character (A-register) is set differently in 1400 compatibility mode than in the 1400 systems when it follows the recomplement operation of an add or subtract instruction. Because of this difference, a chained instruction that uses the d-modifier following these instructions will not operate as in 1400 systems.

A-address Register following I/O Ops. In 1400 compatibility mode, the A-Address Register contains a variable after an I/O instruction of the general form M/L %xx BBB d. Because of this, a Store A Address Register instruction following these I/O instructions does not store the same information as in the 1400 systems.

1050 Differences

Equivalent Functions. The following list gives the 1407/1447 operations or indications and the equivalent 1050 operations or indications:

1407/1447	1050 Equivalents
Request key	Request key
Enter light	Proceed light
Respond key	Operate alternate code key and 5-key
Type-out key	Not available
Clear-key light	During a read-into-storage operation, this function is performed by operating the alternate-code key and the 0 key. During a write-out-of-storage operation, this function is not available.
Cancel key	Cancel key
Release key	EOB key

Errors are not indicated by an underscore. The error is indicated to the program.

1401 Stage I Differences

Memory Wrap Consideration. The 1400 compatibility feature does not accomodate 1401 Stage I programs that are written to use the ability to wrap memory or that use an address that is over the memory size of the machine. For example, if the 1401 is described as a 4K machine, or less, zone bits in the units position will cause an address wrap check.

1402 Differences

Punch Feed Read. When operating Punch Feed Read, in 1400 compatibility mode, a blank card must be inserted before the deck to be read. (1C) operation in 1400 Compatibility mode with the 2540, any column binary characters that are not one of the normal 64 BCD characters entering the normal read-in area (001-080) are replaced by blanks. Recall that in 1400 operation, a column binary character enters the normal read-in area in card image, whether or not it is recognizeable as a BCD character. Binary, data does not enter the normal read-in area.

Last Card Indication. In 1400 compatibility mode operation, the last card indication is reset by the Start reset function, or by the first read instruction following the indication. In the 1400 system, the last card indication is turned off by either a Test and Branch (B xxx A) instruction, on a run in, or by the switch itself or by the start reset key.

1403 Differences

Branch on Channel 9 or 12. On unbuffered 1400-series printers, a branch on channel 9 or 12 interlocks the printer. The last line of printing occurs one line above the channel-9 or -12 punch in the carriage tape. On buffered 1400-series printers, a branch on channel 9 or 12 immediately before printing causes the last line to print one line above the channel-9 or -12 punch in the carriage tape. The same is true if the overflow branch occurs while the printer is busy (currently engaged in printing or forms movement.)

If the overflow (branch on channel-9 or -12) occurs immediately after printing, the last printed line of that form is on the line corresponding to the channel 9 or 12 carriage tape punch.

These considerations are also true for 1400 compatibility operation and should include the space immediate command. (This command is not physically interlocked as in the 1401 system). However, more than one channel-9 or -12 punch may be necessary if the program performs housekeeping routines at the end of the printed page (on and/or below the carriage-tape overflow punch) and still requires an active output from the effect of reading the overflow punch to cause, for example, skipping to the next form. The channel-9 and -12 indicators are reset by the next actual printer command rather than by any channel punch as in 1401 and 1460 systems.

When skipping is not required on or after the overflow punch, one channel-9 or -12 punch is sufficient. Multiple print and/or carriage commands without intervening channel-test branch instructions may also require multiple channel-9 and/or -12 punches.

On the 1442/1443 sub-feature, channel-9 and -12 are reset by a skip to channel-1 or by a programmed branch on channel-9 or -12.

1442 Differences

Last Card Indication. The last card indication is reset by the start reset function

or by the first read instruction following the indication.

Read Error with I/O Check-Stop Switch On. When the I/O Check-stop switch is on, a read error halt occurs at the end of the card instead of at the column in error.

File Differences

Module Number. The module number is deleted when recording addresses on file.

Switches. The Diagnostic, Write-Address, and Disk-Write switches are not provided on the 2311 Disk Storage Drive.

Write Disk Track Record Operations. A 1400 write-disk-track-record operation without address should not be attempted on a disk pack formatted in sector mode.

Tape Differences

Diagnostic Read. The Diagnostic Read instruction will detect a tape mark only if it is a single character record.

Odd Redundancy Tape Mark. A tape mark read in odd redundancy will not set the tape error indicator.

EOF Indication. An end-of-file indication from a given tape unit is issued for every Write operation following the initial detection of the reflective strip, until a rewind or backspace is performed by that same unit.

EOF followed by Manual Unload. If an EOF is sensed (tape read or write) and the tape unit is manually unloaded before a branch-if-end-of-file or rewind-unload command is given, the EOF status bit in local storage must be manually reset from the 2030 console. Under normal conditions, this situation would never occur, and the user need not be concerned about the status of the EOF bit.

CHARACTER CONFIGURATION

- The BCD characters of the 1400 system being emulated are represented in the 2030 in System/360 EBCDIC.
- Absence of a 1-bit in EBCDIC representation indicates that a word mark is associated with the character.
- Character conversion from EBCDIC to BCD and vice-versa can be accomplished through a table lookup.

The BCD characters of the 1400 system being emulated in compatibility mode are represented in the 2030 in EBCDIC. This utilizes all eight data bits plus a parity bit. See Figure 4-25. Note from the chart that all valid BCD characters represented in EBCDIC contain a bit in byte position 1. Word mark notation is achieved by manipulating bit position 1 while still maintaining EBCDIC compatibility externally. The absence of a 1-bit in the EBCDIC representation indicates that a word mark is associated with the character. Thus the character "A" without a word mark is represented as 11000001 in EBCDIC while the character A with a word mark is 10000001 in EBCDIC.

4567 ↓	WITH WORDMARK 0123→				NO WORDMARK				WITH WORDMARK				NO WORDMARK			
	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0000	Blank	&	-		Blank	&	-		?	!	≠	0	?	!	≠	0
0001			/				/		A	J		1	A	J		1
0010									B	K	S	2	B	K	S	2
0011									C	L	T	3	C	L	T	3
0100									D	M	U	4	D	M	U	4
0101									E	N	V	5	E	N	V	5
0110									F	O	W	6	F	O	W	6
0111									G	P	X	7	G	P	X	7
1000									H	Q	Y	8	H	Q	Y	8
1001									I	R	≠	9	I	R	≠	9
1010				≠				≠								
1011	·	$	,	#	·	$	,	#								
1100	¤	*	%	@	¤	*	%	@								
1101	⊏	⊐	∨	:	⊏	⊐	∨	:								
1110	<	;	\	>	<	;	\	>								
1111	≠	△	⧺	√‾	≠	△	⧺	√‾								

Figure 4-25. 1400 Defined Characters

Occasionally a translation of character codes from EBCDIC to BCD and back again is necessary to process certain 1400 system instructions, such as bit test, move zone, or move numeric. Conversion is accomplished by a table lookup procedure that uses tables stored in auxiliary storage. These tables are read into storage as part of the initializing routine. To illustrate the use of the table in auxiliary storage, we will convert a character from EBCDIC to BCD. The character "C" in EBCDIC is C3 (1100 0011). By going to aux storage MPX location, C3, we find the BCD configuration 33 or 00110011, which is the BCD configuration for a "C".

BA 8421

0011 0011

In utilizing the conversion tables, if a word mark is present with the character, the microprogram eliminates it before the table lookup is executed. It might be helpful to examine the relationship between A and B zone bits in BCD representation and EBCDIC bit structure. Bits 2 and 3 in EBCDIC represent the B and A bits in BCD configuration. The relationship is a negative one, however, in that 2 = B, and 3 = A. The four possible patterns of A and B zone bits are $\overline{B}A$, $B\overline{A}$, $\overline{BA}$, and BA; in EBCDIC, these respective values in bits 2 and 3 are as follows: 11, 10, 01, and 00. (Figure 4-26).

	Zone Bits 2 and 3			
Zone Configuration	$\overline{B}A$	$B\overline{A}$	$\overline{BA}$	BA
BCD	00	01	10	11
EBCDIC	11	10	01	00

Figure 4-26. Zone Bits; BCD vs. EBCDIC

If an invalid EBCDIC character addresses the EBCDIC to BCD conversion table, a hexadecimal 40 is read out and detected as an error by the microprogram.

1400 SYSTEM ADDRESSING

- The 1400 compatibility mode main storage area is normally located contiguously in the upper part of 2030 main storage.
- 2030 Model F30 loads the last 256 bytes (65,280 - 65,535) with the character 8F.
- Address bytes in local storage contain a bias constant.
- Tens and hundreds address bytes have upper and lower 4 bits of the byte crossed in local storage.

For 1400 compatibility mode operation, all programs are normally loaded into upper storage in the 2030 Processing Unit. If a 1401 program written for 4K of storage is to be run on a 2030 with 16,384 positions of storage, the program (and work areas, etc.) is stored in IBM 2030 storage locations 12,384 to 16,383. This allows the 2030 to detect 1401 storage wrap errors through existing circuits. It also allows a 2030 supervisory program to remain in lower storage, and facilitates programmed mode switching.

When a 2030 Model F30 (65,536 positions of storage) is being used in 1400 compatibility mode, the last 256 bytes of main storage are loaded with the character 8F. This facilitates wrap-around detection. Thus if a program written for a 4K 1400 is run on a 2030 with 65,536 positions of storage, the compatibility storage occupies IBM 2030 locations 61,280-65,279. The last 256 bytes (65,280 - 65,535) are loaded with the character 8F. For wrap-around detection, 8F in the one position would be sufficient. The additional 255 bytes are loaded with 8F to take care of any inadvertent addressing of this area.

The 2030 uses a conversion table in auxiliary storage to convert 1401 BCD addresses to 2030 binary addresses. This table also includes a storage bias constant (or offset) factor to cause IBM 1401 addresses to address upper 2030 storage. The storage bias constant is a number equal to the 2030 storage size minus the 1400 system storage size (minus 256, if there are 65,536 positions of storage). Refer to Figure 4-27.

For example, in running a 1401 program written for 4K of storage on a 2030 with 16,384 positions of storage, the storage bias constant would be:

```
16,384 (2030 storage size)
-4,000 (1401 storage size)
12,384 or  3060 in hexadecimal.
```

For running a 1401 program, written for 4K of storage on a 2030 with 65,536 positions of storage, the storage bias constant would be:

```
65,536
-4,000
  -256
61,280 or EF60 in hexadecimal
```

The storage bias (or offset) represents the 16 bit (4 hexadecimal characters) 2030 address that is equivalent to the normal location of position 0 of the 1400 core storage area. This address is referenced in the local storage map by Y and Z.

2030 / 1400	Z				Y
	8,192	16,384	32,768	65,536	
16K		01	41	C0	80
12K		11	51	D0	20
8K	00	20	60	DF	C0
4K	10	30	70	EF	60
2K	18	38	78	F7	30
1.4K	1A	3A	7A	F9	88
Byte 9A 1400 Aux Storage LS	1F	3F	7F	FE	

Figure 4-27. Storage Size Byte and Storage Offset Constants

Y = the low-order eight bits (2 hexadecimal characters)

Z = the high-order eight bits (2 hexadecimal characters)

In the two examples 3060 and EF60, Y refers to 60 in both cases. Z refers to 30 and EF respectively. These values can be read directly from Figure 4-27 for all normal combinations of 1400/360 compatibility.

The user may alter the location of the 1400 core-storage area, or the EBCDIC-to-BCD conversion table may be changed to cause printing of characters other than those normally specified by the CID. In the former case (relocation), the memory bias (offset) must be such that the ending 1400 core-storage position is a multiple of 256 bytes from the logical end of 2030 core storage. This restriction assures the correct operation of the 1400-mode clear storage and scan operations. If the ending address of 1400-mode core storage is changed to some multiple of 256 bytes from the ending 2030 core storage position (or more than 256 bytes in the case of 65K systems), some address-validity checking is lost. In addition, byte 9A of 1400 local storage A must be changed to reflect this lower memory bias. Byte 9A is the high-order 8 bits of a 16-bit address that specifies the last 1400 address in 2030 core storage. For example, this 16-bit address is normally 7FFF (32,767) for a 32,768-byte 2030.

You will note in the Auxiliary Storage Map (Figure 4-28) that some factors in the tens and hundreds-low rows have the character X after the value in parenthesis. This indicates that the quantity in parenthesis is crossed in local storage. This crossing facilitates the invalid address checking performed by the microprogram. For working out sample conversions from this chart, ignore the crossing and read the direct value of the factor in parenthesis. We will discuss the significance of crossing in the Address Error Detection section.

ADDRESS CONVERSION

- 1400 addresses are stored in EBCDIC form
- Address Conversion is accomplished by microprogramming utilizing tables in auxiliary storage.
- A 2 byte binary address is developed from 3 characters in EBCDIC code.
- Units and hundreds zone bits determine thousands; (tens zone bits provide indexing)

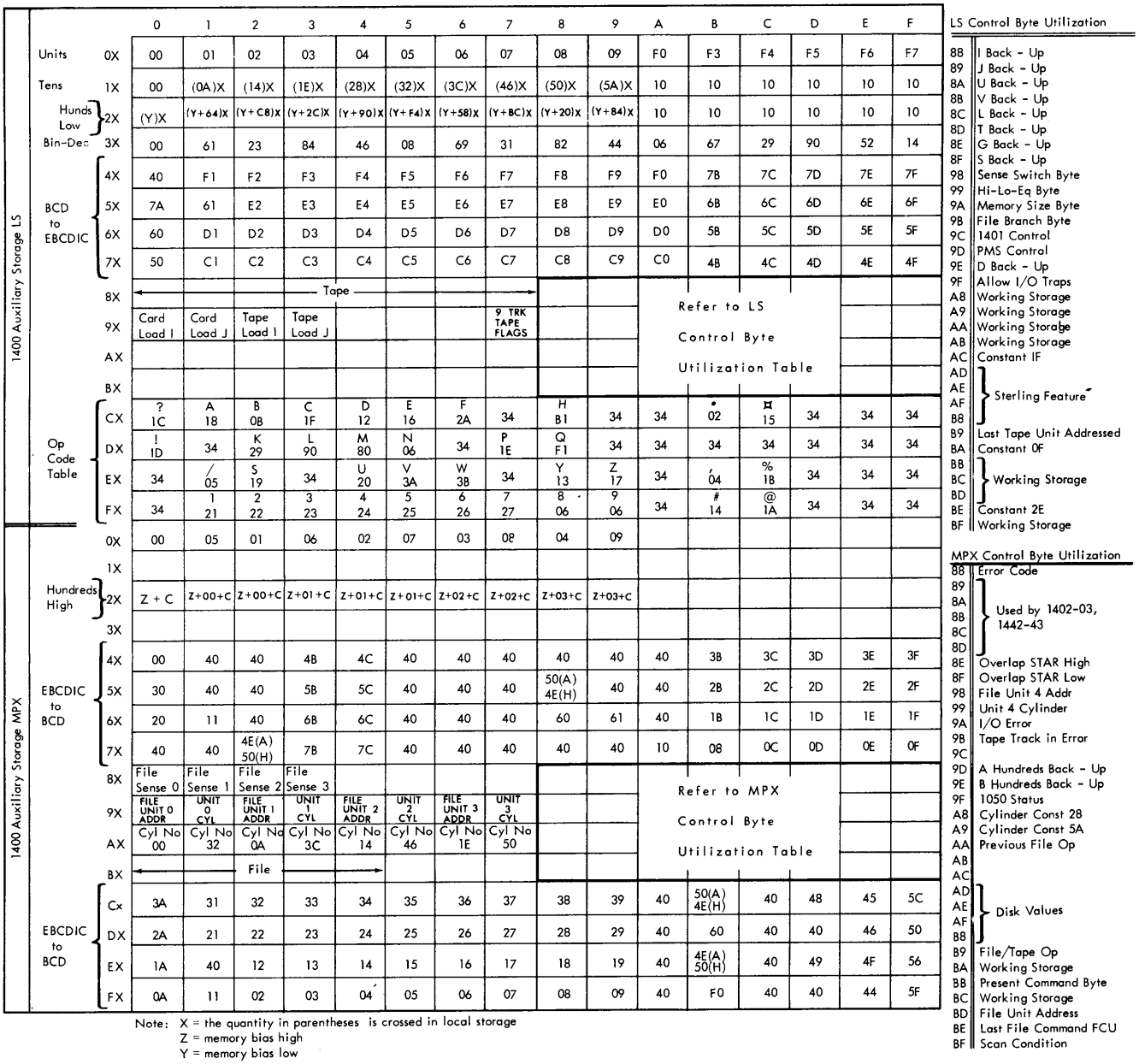
For an example of address conversion, assume a 2030 with 16,384 positions of storage is emulating a 1401 program written for a 12K 1401. From Figure 5-27 we determine that the normal bias is 1120 (Y = 20, Z = 11). We will discuss A-Star address development during I-Phase for the instruction BY14E.

Our objective in compatibility address conversion is to convert the 1400 3 EBCDIC character address that is in storage to a two byte binary address that includes the memory bias offset factor. Microprogramming and conversion tables facilitate the conversion.

When converting a decimal address to a binary value, the hundreds digit may affect the value of both the high-order byte and the low order byte of the binary address (e.g., 200 = C8 but 300 = 12C). For this reason, in converting the hundreds digit we address auxiliary storage twice -- once for hundreds low and once for hundreds high.

To convert the 1400 address to a binary address the microprogram uses digits in the 1400 address to read out tables in auxiliary storage (Figure 4-28). The digits (bits 4-7) in the 1400 address become bits 4-7 of an address generated to read out auxiliary storage tables. Bits 0-3 of the storage table address are forced by the microprogram as follows:

Bits 0-3	Position
0010	Hundreds low
0010	Hundreds high
0001	Tens
0000	Units



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Units 0X	00	01	02	03	04	05	06	07	08	09	F0	F3	F4	F5	F6	F7
Tens 1X	00	(0A)X	(14)X	(1E)X	(28)X	(32)X	(3C)X	(46)X	(50)X	(5A)X	10	10	10	10	10	10
Hunds Low 2X	(Y)X	(Y+64)X	(Y+C8)X	(Y+2C)X	(Y+90)X	(Y+F4)X	(Y+58)X	(Y+BC)X	(Y+20)X	(Y+84)X	10	10	10	10	10	10
Bin-Dec 3X	00	61	23	84	46	08	69	31	82	44	06	67	29	90	52	14
BCD to EBCDIC 4X	40	F1	F2	F3	F4	F5	F6	F7	F8	F9	F0	7B	7C	7D	7E	7F
BCD to EBCDIC 5X	7A	61	E2	E3	E4	E5	E6	E7	E8	E9	E0	6B	6C	6D	6E	6F
BCD to EBCDIC 6X	60	D1	D2	D3	D4	D5	D6	D7	D8	D9	D0	5B	5C	5D	5E	5F
BCD to EBCDIC 7X	50	C1	C2	C3	C4	C5	C6	C7	C8	C9	C0	4B	4C	4D	4E	4F
8X	← Tape →										Refer to LS Control Byte Utilization Table					
9X	Card Load I	Card Load J	Tape Load I	Tape Load J				9 TRK TAPE FLAGS								
AX																
BX																
Op Code Table CX	? 1C	A 18	B 0B	C 1F	D 12	E 16	F 2A	34	H B1	34	34	• 02	¤ 15	34	34	34
Op Code Table DX	! 1D	34	K 29	L 90	M 80	N 06	34	P 1E	Q F1	34	34	34	34	34	34	34
Op Code Table EX	34	/ 05	S 19	34	U 20	V 3A	W 3B	34	Y 13	Z 17	34	, 04	% 1B	34	34	34
Op Code Table FX	34	1 21	2 22	3 23	4 24	5 25	6 26	7 27	8 06	9 06	34	# 14	@ 1A	34	34	34
0X	00	05	01	06	02	07	03	08	04	09						
1X																
Hundreds High 2X	Z + C	Z+00+C	Z+00+C	Z+01+C	Z+01+C	Z+01+C	Z+02+C	Z+02+C	Z+03+C	Z+03+C						
3X																
EBCDIC to BCD 4X	00	40	40	4B	4C	40	40	40	40	40	40	3B	3C	3D	3E	3F
EBCDIC to BCD 5X	30	40	40	5B	5C	40	40	40	50(A) 4E(H)	40	40	2B	2C	2D	2E	2F
EBCDIC to BCD 6X	20	11	40	6B	6C	40	40	40	60	61	40	1B	1C	1D	1E	1F
EBCDIC to BCD 7X	40	40	4E(A) 50(H)	7B	7C	40	40	40	40	40	10	08	0C	0D	0E	0F
8X	File Sense 0	File Sense 1	File Sense 2	File Sense 3							Refer to MPX Control Byte Utilization Table					
9X	FILE UNIT 0 ADDR	UNIT 0 CYL	FILE UNIT 1 ADDR	UNIT 1 CYL	FILE UNIT 2 ADDR	UNIT 2 CYL	FILE UNIT 3 ADDR	UNIT 3 CYL								
AX	Cyl No 00	Cyl No 32	Cyl No 0A	Cyl No 3C	Cyl No 14	Cyl No 46	Cyl No 1E	Cyl No 50								
BX	←		File	→												
EBCDIC to BCD Cx	3A	31	32	33	34	35	36	37	38	39	40	50(A) 4E(H)	40	48	45	5C
EBCDIC to BCD DX	2A	21	22	23	24	25	26	27	28	29	40	60	40	40	46	50
EBCDIC to BCD EX	1A	40	12	13	14	15	16	17	18	19	40	4E(A) 50(H)	40	49	4F	56
EBCDIC to BCD FX	0A	11	02	03	04	05	06	07	08	09	40	F0	40	40	44	5F

Note: X = the quantity in parentheses is crossed in local storage
Z = memory bias high
Y = memory bias low

Figure 4-28. Auxiliary Storage Map for 1400 Compatibility

Thus to convert the hundreds position (example: Y = 1110 1000) of a 1400 address to a binary value, we would address auxiliary storage with 0010 1000. Observe that bits 0-3 were forced to 0010 and the digit 8 of the hundreds Y was inserted into bits 4-7.

In converting the three character 1400 address to a two byte binary address, the microprogram accumulates the binary values of the three characters in the 1400 address plus the factor for memory bias. Consider our sample address Y14. This represents an actual address of 1814 in the original 1401 program (Y = CA8; "A" zone in hundreds equals 1 thousand).

Let's first examine this address decimally, and convert it using decimal values to illustrate how our result is obtained. The desired address actually consists of the 1400 address 1814 plus the memory bias of 4384 for a total of 6198. The result of 6198 converted to hexadecimal is 1836 (binary 00011000 00110110). There are considerably more steps involved in the actual conversion, however. The I-Cycle Address Setup Flowchart in the Maintenance Diagram Manual, Form Y24-3466, illustrates the actual microprogram manipulation during the address conversion.

Let us follow this same example through conversion showing logically what happens without regard to exact sequence of microprogramming steps and hardware register involved. We'll use the term "accumulator" to refer to the place of address development.

1. Read out hundreds character Y = CA8 = 11101000. Zone bits 10 = A = 1 thousand = 3E8 entered in the accumulator.

2. Digit portion (bits 4-7) of hundreds character (1000) becomes bits 4-7 of the address for addressing local storage. The microprogram emits a 2 (0010) in bits 0-3. The resultant address (28 in hex) is used to address local storage.

3. As previously discussed, we must address auxiliary storage twice to convert the hundreds digit to binary. Hundreds low is addressed in Auxiliary Storage LS, location 28 (Figure 5-28). From this location, we read out the factor Y+20. The X indicates that the amount is crossed. This is for error detection and will be discussed later. In our example, Y = 20, therefore the actual (uncrossed) value in local storage byte 28 is 40. (There was no carry in the Y+20 addition). This is added to the low byte in the accumulator -- At this point logically, accumulator value is 3E8 (thousands) + 40 (hundreds low + bias) = 428.

4. The high bias conversion factor is taken from aux stor MPX byte 28. Note that this factor is Z + 03 + C. For the example, Z = 11, and there was no carry in low hundreds conversion therefore, the factor 14 is read from the table and added to the accumulator high byte 0428 + 1400 = 1828. We have now accumulated thousands and hundreds and have tens and units yet to add.

5. The tens byte is read out. The tens character 1 = 11110001. There are no zones. If tens byte were zoned, indexing would be required. Refer to Address Indexing in the Maintenance Diagram Manual. The table address for tens conversion is 0001 0001 because the digit portion of the tens character becomes bits 4-7 and the microprogram emits 1 in bits 0-3. Aux Stor LS, byte 11 contains the factor 0A. (Ignore for now the fact that the value is crossed). 0A is added to the accumulator. 1828 + 0A = 1832.

6. The units byte is read out. The units character 4 = 11110100. (There are no zones; if zones were present, we would add the appropriate number of thousands 4, 8, or 12 to the accumulator.) The table address for units conversion is 00000100 = 04. Aux storage LS byte 04 contains the factor 04. 04 is added to the accumulator. The accumulator now contains 1832 + 04 = 1836 -- the desired hexadecimal equivalent address in the 2030 for the address Y14 for the conditions in the example. If the instruction had a B-address, the B-star would be developed in the same manner. Figure 4-29 summarizes the example.

Factor	Action	Accumulator	Auxiliary Storage
Thousands	1000) = 3E8)	03E8	Direct Entry
Hundreds Low Bias	Y + 20 = 20 + 20 03E8 + 40	0428	LS 28
Hundreds High Bias	Z + 03 + C 11 + 03 0428 + 1400	1828	MPX 28
Tens Bias	1828 + 0A	1832	LS 10
Units Bias	1832 + 04	1836	LS 04

Figure 4-29. Address Conversion Summary

ADDRESS ERROR DETECTION

- Invalid 1400 characters (8F) are placed in the upper 256 bytes of the 65,536 position 2030 to aid in detecting high storage wrap errors.
- 8F is placed in 1400 address 000 minus one for low storage wrap error detection.
- 2030 circuits detect storage wrap errors.
- Crossing the tens and hundreds values in the compatibility conversion table facilitates error detection by utilizing the status of the R3 bit.

In all non-65K systems, the upper position of 1400 system compatibility storage is normally coincident with the upper position of 2030 storage. This enables high storage wrap error detection by the same circuitry as in 2030 mode operation.

In 65K systems, high memory wrap must be detected in a different way because all possible bit configurations in the M-register are legitimate. As an aid in detecting memory wrap in 1400 compatibility mode, the invalid character 8F is placed in the upper 256 bytes of memory. In effect, an invalid character is used to detect a memory wrap condition.

To detect an error when the equivalent 1400 address 000 is modified by minus one (low storage wrap error) an invalid character (8F) is placed in 2030 memory one core location below the address assigned to 1400 compatibility storage location 000. If this location is addressed, the invalid character causes a 1400 mode process check.

The invalid character 8F is placed in storage as required, by the Compatibility Initialization Instruction.

During the conversion of 1400 system addresses to 2030 addresses, an error detection procedure detects invalid characters. This procedure is based on the status of the R3 bit. R3 ON indicates an invalid address character.

During address conversion, the hundreds and tens digits that are used to address the conversion table assume tens, and units digits respectively are zero. Thus, a binary value ending in zero is extracted and set into the R-register. For example the tens 2 equivalent located at 12 in Aux-storage A is binary 14 (decimal 20 = hex 14). However, if 14 were set into the R-register, the bit status would be 00010100 and R3 being on would indicate an invalid character. For this reason, the value 14 is crossed by a microprogram step and set into the R-register as 41 (01000001). Because the converted values are round numbers, the normal low-order digit never turns on the low R-register one bit (R-7). Therefore when crossed, it never turns on the high R-register one bit (R-3).

The units values are not crossed in local storage. A microprogramming test determines validity.

OP CODE CONVERSION AND RECOGNITION

- 1400 system Op codes are converted to bit significant characters to facilitate recognition by the microprogram.
- An op code conversion table is stored in local storage.
- 1400 system op code character bits 0 & 1 are forced on before using the character to address the conversion table.
- The converted op code is stored in the G-register during I-cycles.

The EBCDIC bit configurations of 1400 system op codes do not readily indicate to 2030 microprogramming what type of op code is being handled. A conversion table stored in local storage groups similar op codes together and converts the bits to a configuration that is bit sensitive for easy identification by the microprogram.

When the microprogram reads a 1400 system op code from storage in EBCDIC form, it forces on the 0 and 1 bits of the op code. Refer to Figure 5-25, and observe that if the 0 and 1 bits of all 1400 system characters are forced on, there are still enough unique bit combinations for all characters except Blank, -, and &. These three are not valid 1400 system op codes and they are detected as such by a special microprogram text.

The modified op code, formed when the 0 and 1 bits are forced on, is used to address local storage and bring out the converted op code that was stored by the initialization routine. The converted op code bears no logical resemblance to the original. The new op code character has a bit configuration that is more readily tested to determine the type of operation desired (I/O CPU, Miscellaneous).

Let us examine the use of the op code conversion table. Assume the op code read out of the 1400 system program is the Edit operation. The hexadecimal bit configuration of E with a WM in EBCDIC is 85 (10000101). A microprogram step forces on bits 0 and 1. This changes the configuration to C5 (1100 0101).

C5 is used to address the op code table in local storage (Fig. 4-28). From Aux Stor LS, position C5, 16 (00010110) is read out and stored in the G-register. 16 is bit sensitive to the microprogram as an Edit op code.

Any invalid EBCDIC op code configuration that addresses the op code table brings out a byte containing 34. This is recognized by the microprogram as an error.

CONSOLE OPERATIONS

- The 2030 console bears little resemblance to a 1401/1440/1460 console.
- Programmed and error stop codes facilitate troubleshooting.
- General console operation, display, etc. is presented in SRL manuals.
- Switch F performs additional functions for compatibility mode operation.

The IBM System/360 Model 30 console differs considerably from the 1400 system being emulated. Familiarity with the console is essential and can be advantageous for analyzing malfunctions that may occur while operating in 1400 mode. Refer to the SRL publication; IBM System/360 Model 30 1401/1440/1460 Compatibility Feature, Form A24-3255, for a more complete presentation of console operations.

The console makes use of error indication codes to tell the operator the reason for a programmed or error stop. These indications are presented to assist the customer engineer in trouble analysis. In reality, there are many more aids to trouble shooting 1400 programs that are emulated on the System/360 Model 30 than for the 1400 program on a 1400 system.

Console Error Indications and Restart Procedures

- A coded display in the MSDR indicates the reason for all programmed and error stops.
- Coded error indications and console displays aid in isolating machine or programming malfunctions.

On all stops at ROS address 10FF, except for set-IC and sense switch operations, a coded digit is displayed in the main storage data register (MSDR) to indicate the reason for the stop.

The 1400 decimal instruction address is displayed in the BA Register lights and the 1400 decimal A-address is displayed in the MN register lights. The 1400 decimal B-address can also be displayed manually in the UV registers by the normal 2030 procedure as, for example, in the case of a programmed or error stop on the instruction A 500 600 at address 400. The coded byte in the R-register (MSDR lights) indicates the reason for the stop. 500 is displayed in the main storage address register. The I-address (400) is displayed in the B- and A-register, and the B-address (600) can be displayed in the UV registers. (Refer to Console Procedures in SRL, Form A24-3255.)

The stop codes are presented in numerical order as a reference aid. Most stops involve a situation such as program error or wrong operating procedure. In these cases, the corrective action is usually self explanatory. The notable exception to this are these errors involving the reader punch. For these stops (where possible) error recovery procedures are given.

Stop Code in MSDR	Reason for Stop/Recovery Procedures
00	Normal Stop. Appears when the stop is caused by pressing the Stop key, ending an instruction-execute in instruction step mode, or by getting a match in SAR delayed-stop mode.
01	Attempted to use invalid 1400 B-address.
02	Attempted to use invalid 1400 A-address.
03	Attempted to use invalid 1400 B and A address.
04	Attempted to use invalid 1400 operation code.
05	Invalid I/O operation attempted; either unit selected or unit number is invalid.
06	Storage wrap occurred when address was used that was outside of system capacity.
07	Storage protection occurred in 1400 mode
08	Attempted to switch to 2030 mode without the PMS feature.
09	Invalid source or destination address on one of the special PMS tape operations.
0A	Attempted to convert to binary an address that was less than the bias (offset) address on a clear storage or store STAR operation.
0B	Storage wrap-around 1400 address 0000.
0C	Attempted to start a 1400 I-cycle at main storage address 0000.
0E	Attempted to index without advanced programming comment in CID.
10	Read-back check-stop (Disk-File operation).
11	Some other device attempted to take a multiplexor-channel data cycle while in the data-transfer portion of a 1050 operation.
20	No channel or device ends received (Disk File Operation).
21	Word mark missing from 1400 operation code during I-Op.
22	Wrong address sent back from channel (Tape Operation, Selector or Multiplexor channel).
30	Wrong address sent back from channel (File Operation).
31	Word mark in A-address of an I/O instruction.
3F	2540 or 2501 reader error. A read check or validity check has occurred, however the error

lights at the 2540 will have been reset before the stop.

Restart procedure:

1. The last card in stacker R1 is the error card. Remove it and correct any errors in this card.
2. Open the joggler gate and remove the card from the hopper.
3. With the joggler gate open, press the Reader-start key to clear the feed.
4. Place the corrected card in front of the three cards that were run out. Place these four cards in the hopper (or ahead of the cards in the file feed magazine).
5. Close the joggler gate.
6. Press the Reader-start key.
7. Press the Start key on the CPU console.

3F 2540 Punch error, PFR operations. The Punch-check light and/or the Validity-check light or no lights may be on. The following restart procedure should be followed.

1. Remove the cards from the punch hopper.
2. Press the Punch-start key to clear the feed.
3. Remove the last three cards from stacker P1.
4. The first of these cards must be reconstructed to remove the punching for re-run and checking.
5. The second of these cards is the error card. This card may require correction.
6. Place reconstructed card #1, corrected card #2 and card #3 in front of the deck. A readily identifiable blank card should be placed in front of the three cards.
7. Reconstruct internal data in the system as necessary to restart the program at the instruction that caused card #1 to be read at the PFR station.
8. Set the Process switch to the single cycle position. Set address 10FF is switches F, G, H, and J. Press the System Reset, Roar Reset, and Start keys in that order. Set the Process switch back to the process position.
9. Press the Start key at the punch.
10. Do "set IC" function to instruction referred to in item 7.
11. Press Start key at CPU console to resume processing.
12. Remove previously inserted readily identifiable card from the stacker. (This card may be punched or blank).

40 Disk-File Stop. Unit-Check status response to seek channel.

41 An 8F character was detected at an address other than the offset address while in 1400 mode.

42 Tape stop. Selector or Multiplexor channel. Invalid channel status on data transfer.

4F 2540 Reader Intervention Required, Reader Feed Stop light on.* The reader-stop condition indicates that all cards beyond the pre-stacker station have been correctly read. Any card at the pre-stacker station or behind it in the read feed must be re-entered. Use the following restart procedure:

1. Remove the cards from stacker R1.
2. Open the hopper joggler gate and remove the cards from the hopper.
3. Remove any jammed cards from the read feed.
4. With the joggler gate open, press the Reader-start key to clear the feed. Any damaged cards must be reconstructed.

5. Remove the cards that ran into stacker R1 by the clearing operation. Place these and any reconstructed cards in the correct sequence, ahead of the cards removed from the hopper, and replace this deck in the hopper, or ahead of the cards in the file feed.

6. Close the joggler gate.

7. Press the Reader-start key.

8. Press the start key on the CPU console.

*Note: This condition could be caused by the reader running out of cards, or by the stacker being full, or by an error.

50 Disk-File Stop. Operational Interlock.

51 An I/O operation was attempted on a device for which the compatibility subfeature is not installed.

52 Tape Stop, Selector or Multiplexor channel. Device-end signal before encountering a GMWM on a tape-write operation.

55 A 1400 start-reset function was performed using the Console-interrupt key.

5F 2540 Punch Intervention Required, Punch Error light on. This indicates a card jam or a misfeed in the card transport area of the punch, hopper empty, stacker full, etc. Correct as follows (as applicable):

1. Remove the cards from stacker P1.

2. Remove the cards from the hopper.

3. Remove any jammed cards from the punch feed; run out any undamaged cards.

4. Discard the last card punched; the data for this card will be repunched on the restart.

5. Replace the blank cards in the hopper and press the Punch-start key. The card that was being punched when the error occurred will be repunched.

6. Press the Start key on the CPU console.

5F 2540 Punch Intervention Required, Punch-check light on. Use the following restart procedure.

1. Examine the last card in stacker P1. This is the card that caused the stop. Correct as necessary.

2. Place corrected card in the correct stacker.

3. Press the Start key at the CPU console to continue the program.

5F 2540 Punch Error, Punch-feed-stop Error light and Punch-check-error light on. This indicates a jam at the punch check station. Use the following recovery:

1. Remove the cards from the hopper.

2. Remove any jammed cards from the punch feed and run out any undamaged cards. Remake any damaged cards. The first two of the four cards that were run out have been punched but not checked. They can be checked manually and placed in the correct stacker.

3. Replace the blank cards in the hopper and press the Punch-start key.

4. Press the Start key on the CPU console. Alternate recovery procedure:

    1. Remove the cards from the hopper.

    2. Remove any jammed cards from the punch feed and run out any undamaged cards. The two punched cards that were in the feed transport at the time the jam occurred

have not been checked. Discard these two cards.

3. Reconstruct internal data in the system, as necessary, to restart at the instruction that punched the card that was jammed in the punch check station.

4. Replace the blank cards in the hopper and press the Punch-start key.

5. Set the instruction counter to the instruction referred to in item 3.

6. Press the Start key on the CPU console.

2540 Punch Error (PFR operation) Punch-feed-stop light on only. This indicates a card jam or misfeed. The cards that have fed past the punch check station have been read and punched correctly. Correct the error condition as follows:

1. Remove the cards from stacker P1.

2. Remove the cards from the hopper.

3. Remove any jammed cards from the feed and run out any undamaged cards.

4. Remake any damaged cards, or cards that have been punched and not punch checked.

5. Place the reconstructed cards and the unread cards from stacker P1 ahead of the cards removed from the hopper. Place a readily identifiable blank card in front of this deck and place the deck in the hopper.

6. Reconstruct internal data in the system as necessary to restart the program at the instruction that caused the first reconstructed card to be read at the PFR station.

7. Set the Process switch to the single cycle position. Set address 10FF in switches F, G, H, and J. Press the System-reset, Roar reset, and Start keys in that sequence. Set the Process switch back to the process position.

8. Press the Start key at the punch.

9. Set instruction counter (IC) to the instruction referred to in item #6.

10. Press the Start key at the CPU console to resume processing.

11. Remove previously inserted readily identifiable card from the stacker.

5F Punch Error (PFR operation) Punch-check light on, Punch-feed-stop light may or may not be on. The following recovery procedures should be followed:

1. Remove the cards from the hopper. (If the Punch-feed-stop light is on, clear the jam).

2. Press the Punch-start key to clear the feed.

3. Remove the last four cards from stacker P1. The last two cards are correct; prepunching in the first two cards must be reconstructed.

4. Place a readily identifiable blank card in front of the two reconstructed cards, the two correct cards and the cards removed from the hopper (in that sequence). Place the deck in the hopper.

5. Reconstruct internal data in the system as necessary to restart the program at the instruction that caused the first reconstructed card to be read at the PFR station.

6. Set the process switch to the single cycle position. Set address 10FF in switches F, G, H, and J. Press the CPU System-rest, Roar-reset, and Start key in that order. Set the Process switch back to the process position.

7. Press the Start key at the punch.

8. Set instruction counter (IC)

to the instruction referred to in step 5.

9. Press Start key at CPU console to resume processing.

10. Remove previously inserted readily identifiable card from stacker. (card may be punched).

61 Sterling feature process check - Marker misalignment.

62 Tape stop. Selector channel - Status-in and service-in on a tape write.

6F Printer stop. Intervention required. Correct the condition, then try the instruction that caused the stop by pressing the 2030 Start key.

71 Sterling Feature process check - Invalid character.

7F Reader Punch stop - (1402/1403 subfeature). Stacker select instruction given after maximum time-out. (6ms after a card read). Correct the condition, then restart.

80 Reader Punch stop - (1442/1443 subfeature) Wrong address sent back from the channel. 1402/1403 sub feature: No address-compare, or punch-transfer error.

81 Sterling Feature. Marked in add or subtract pence or shilling position.

82 Tape stop - Selector channel. Status-in and service-in on a read-move operation.

8F Tape stop - Selector or Multiplexor channel. Tape unit intervention required.

90 Reader Punch stop - 1402/1403 subfeature; Operational-in disconnect on 2540 or 2501 reader. 1442/1443 subfeature; Invalid d-modifier.

92 Tape stop - selector or Multiplexor channel. Tape error on a 1400 initial program load.

A0 Reader Punch stop - 1402/1403 subfeature; Operational-in disconnect on 1403 printer 1442/1443 subfeature; No GMWM in storage.

A2 Tape stop - selector channel. Invalid channel status was received on a branch-if-error operation

B0 Reader punch stop - 1402/1403 subfeature; Operational-in disconnect on 2540 or 2520 punch. 1442/1443 subfeature; 1442 error on read or punch operation.

B2 Tape Stop - selector channel. Status-in and service-in on a 1400 read-load operation.

C2 Tape stop - Multiplexor channel. Operational-in disconnect on a read operation.

CF 1050 stop - 1050 intervention required. Correct the condition and retry the instruction that caused the stop.

D2 Tape stop - Multiplexor channel. Premature end to a sense operation.

DF 1050 stop - Alter or display stop. Restart by pressing the 2030 Start key.

E2 Tape stop - Multiplexor channel. Operational-in disconnect-on-mode-set operation.

*F0 A 1400 halt instruction, with no invalid addresses.

*F1 A 1400 halt instruction with invalid B-address.

*F2 A 1400 halt instruction with invalid A-address.

*F3 A 1400 halt instruction with invalid B and A address.

FF A 1400 halt and branch instruction has been executed.

*Note. These stops indicate execution of all 1401 Halt instructions (•) except the four character Halt and Branch (•AAA). The status of the existing address in the STAR is provided, but does not indicate an error stop. (Ex: M%U1BBBW• would cause an F2 Halt.)

AUXILIARY STORAGE

The auxiliary storage of the 2030 normally provides residence for general-purpose registers, floating-point registers condition registers, and multiplexor channel Unit Control Words (UCW's). For operation in 1400 compatibility mode, auxiliary storage must be loaded with certain fixed information required by ROS to compensate for the difference in op code structure, and storage addressing between the 1400 system and system/360. Variable information such as tape densities, unit addresses storage size etc. must also be entered before the 2030 can execute 1400 instructions. Other areas of auxiliary storage are initialized (to either 0 or some required value). These areas provide status indication and control information for ROS, also I-STAR, B-STAR, and A-STAR backup, etc.

Because auxiliary storage performs such a vital role in 1400 compatibility mode operation, the function of each of the 512 bytes is quite significant. Customer modifications, such as alteration of op code tables, non-standard special-character print arrangements, relocation of the 1400 mode core storage area etc., demand an even greater involvement with details in auxiliary storage.

Placement of the 1400 compatibility requirements (1400 auxiliary storage A and B) are as shown in Figure 4-30.

In 16K and larger systems the 1400 mode auxiliary storage is assigned to the two highest numbered MPX Auxiliary Storage Areas. 1400 auxiliary storage LS is assigned to the highest MPX number (MPX-2 or MPX-6), 1400 Auxiliary storage MPX is assigned to the next highest number (MPX-1 or MPX-5.) In 8K systems, there are only 2 areas; MPX and Local Storage. Assignment of 1400 auxiliary storage is reversed, the MPX unit is used for auxiliary storage LS, and Local Storage is used for auxiliary storage MPX.

When the Programmed Mode Switching feature is used, Local Storage, MPX-0 (and MPX-1 through -4 in 32K and 64K) can be used in 2030 mode.

If the microprogram mnemonic M/LS is used in the storage Statement in 1400 compatibility mode with the Programmed Mode Switching feature installed, 2030 LS is addressed. If M/LS is used without PMS, the storage statement is interpreted according to the RR option. For example, if the operation is in RR format, 2030 is addressed. If in any other format, Main Storage is addressed.

We will examine the 512 bytes of auxiliary storage (Figure 4-31) row by row, and byte by byte or bit by bit when necessary to explain the function. Auxiliary storage consists of 256 bytes in 1400 aux stor LS and 256 bytes in 1400 aux stor MPX. The relationship of auxiliary storage to the CID card that initializes it is also shown. 32 cards of the CID deck load auxiliary storage.

Auxiliary Storage LS

Row 0X (CID card 0300). Units Conversion Constants. This row provides the units conversion factors for converting 1400 system addresses in BCD to 2030 addresses in EBCDIC. 0A through 0F contain invalid digit values that turn on the R3 bit when they are read out. (Refer to Addresses Error Detection).

Row 1X (CID card 0310). Tens conversion constants. This row provides tens conversion for converting BCD to EBCDIC. The values are crossed in storage to facilitate error detection. 1A through 1F contain invalid digit values.

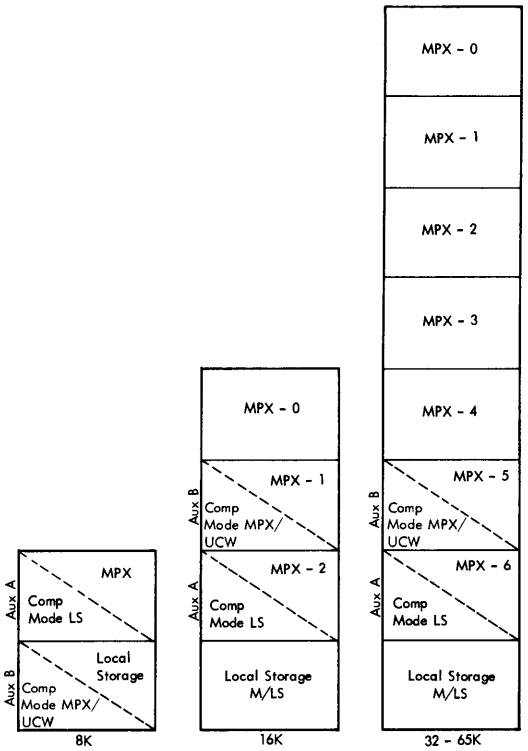
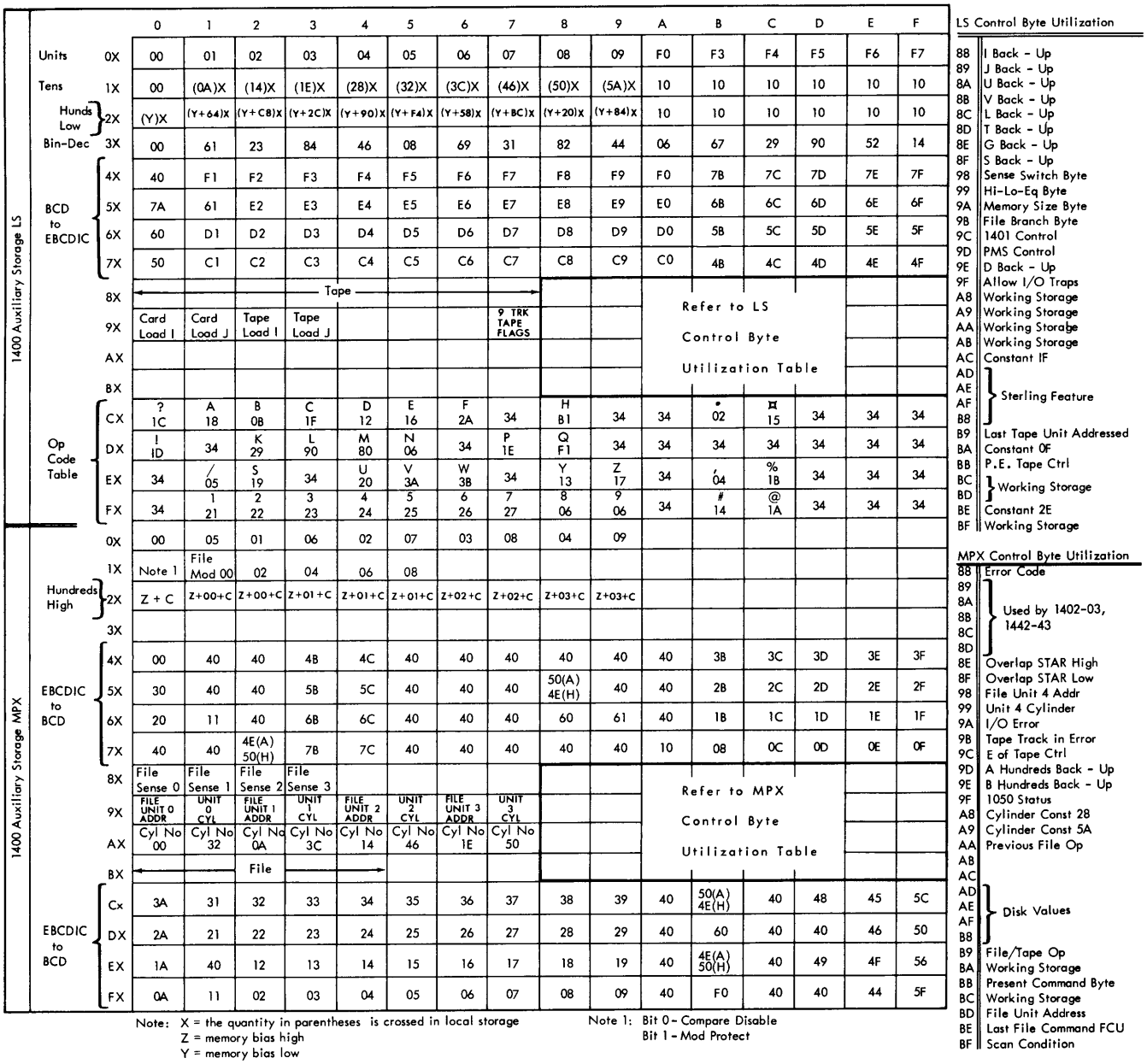


Figure 4-30. Placement of 1400 Compatibility Mode Aux Storage



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Units 0X	00	01	02	03	04	05	06	07	08	09	F0	F3	F4	F5	F6	F7
Tens 1X	00	(0A)X	(14)X	(1E)X	(28)X	(32)X	(3C)X	(46)X	(50)X	(5A)X	10	10	10	10	10	10
Hunds Low 2X	(Y)X	(Y+64)X	(Y+C8)X	(Y+2C)X	(Y+90)X	(Y+F4)X	(Y+58)X	(Y+BC)X	(Y+20)X	(Y+84)X	10	10	10	10	10	10
Bin-Dec 3X	00	61	23	84	46	08	69	31	82	44	06	67	29	90	52	14
BCD to EBCDIC 4X	40	F1	F2	F3	F4	F5	F6	F7	F8	F9	F0	7B	7C	7D	7E	7F
BCD to EBCDIC 5X	7A	61	E2	E3	E4	E5	E6	E7	E8	E9	E0	6B	6C	6D	6E	6F
BCD to EBCDIC 6X	60	D1	D2	D3	D4	D5	D6	D7	D8	D9	D0	5B	5C	5D	5E	5F
BCD to EBCDIC 7X	50	C1	C2	C3	C4	C5	C6	C7	C8	C9	C0	4B	4C	4D	4E	4F
8X	← Tape →										Refer to LS Control Byte Utilization Table					
9X	Card Load I	Card Load J	Tape Load I	Tape Load J				9 TRK TAPE FLAGS								
AX																
BX																
Op Code Table CX	? 1C	A 18	B 0B	C 1F	D 12	E 16	F 2A	34	H B1	34	34	• 02	¤ 15	34	34	34
Op Code Table DX	! 1D	34	K 29	L 90	M 80	N 06	34	P 1E	Q F1	34	34	34	34	34	34	34
Op Code Table EX	34	/ 05	S 19	34	U 20	V 3A	W 3B	34	Y 13	Z 17	34	' 04	% 1B	34	34	34
Op Code Table FX	34	1 21	2 22	3 23	4 24	5 25	6 26	7 27	8 06	9 06	34	# 14	@ 1A	34	34	34
0X	00	05	01	06	02	07	03	08	04	09						
1X	Note 1	File Mod 00	02	04	06	08										
Hundreds High 2X	Z + C	Z+00+C	Z+00+C	Z+01+C	Z+01+C	Z+01+C	Z+02+C	Z+02+C	Z+03+C	Z+03+C						
3X																
EBCDIC to BCD 4X	00	40	40	4B	4C	40	40	40	40	40	40	3B	3C	3D	3E	3F
EBCDIC to BCD 5X	30	40	40	5B	5C	40	40	40	50(A) 4E(H)	40	40	2B	2C	2D	2E	2F
EBCDIC to BCD 6X	20	11	40	6B	6C	40	40	40	60	61	40	1B	1C	1D	1E	1F
EBCDIC to BCD 7X	40	40	4E(A) 50(H)	7B	7C	40	40	40	40	40	10	08	0C	0D	0E	0F
8X	File Sense 0	File Sense 1	File Sense 2	File Sense 3							Refer to MPX Control Byte Utilization Table					
9X	FILE UNIT 0 ADDR	UNIT 0 CYL	FILE UNIT 1 ADDR	UNIT 1 CYL	FILE UNIT 2 ADDR	UNIT 2 CYL	FILE UNIT 3 ADDR	UNIT 3 CYL								
AX	Cyl No 00	Cyl No 32	Cyl No 0A	Cyl No 3C	Cyl No 14	Cyl No 46	Cyl No 1E	Cyl No 50								
BX	←		File	→												
EBCDIC to BCD Cx	3A	31	32	33	34	35	36	37	38	39	40	50(A) 4E(H)	40	48	45	5C
EBCDIC to BCD DX	2A	21	22	23	24	25	26	27	28	29	40	60	40	40	46	50
EBCDIC to BCD EX	1A	40	12	13	14	15	16	17	18	19	40	4E(A) 50(H)	40	49	4F	56
EBCDIC to BCD FX	0A	11	02	03	04	05	06	07	08	09	40	F0	40	40	44	5F

Note: X = the quantity in parentheses is crossed in local storage
Z = memory bias high
Y = memory bias low

Note 1: Bit 0 - Compare Disable
Bit 1 - Mod Protect

Figure 4-31. Auxiliary Storage Map for 1400 Compatibility

Row 2X (CID card 0320). Hundreds low conversion. The values in positions 20-29 are constant for a particular operation and are determined by the size of the 1400 being emulated. This is because the Y-bias value is a factor (Figure 4-31). This value forms part of the low order byte when 1400 addresses are converted to 2030 addresses. The value is crossed in storage. The crossed value for card 0320 bytes 20-29 for all 1400 core sizes is available from Figure 4-32. Positions 2A-2F contain invalid digit values for address error detection.

1400 Core Size	Auxiliary Storage LS Locations									
	20	21	22	23	24	25	26	27	28	29
16K	08	4E	84	CA	01	47	8D	C3	0A	40
12K	02	48	8E	C4	0B	41	87	CD	04	4A
8K	0C	42	88	CE	05	4B	81	C7	0E	44
4K	06	4C	82	C8	0F	45	8B	C1	08	4E
2K	03	49	8F	C5	0C	42	88	CE	05	4B
1.4K	88	CE	05	4B	81	C7	0E	44	8A	C0
	6 7	8 9	11 12	13 14	16 17	18 19	21 22	23 24	26 27	28 29
	CID Card 0320 Column Locations									

Figure 4-32. Hundreds Low Conversion; Auxiliary Storage LS Row 2X Values

Row 3X (CID Card 0330). Binary to Decimal Conversion Table. This row contains constants for converting binary to decimal equivalent. This table is used, for example, for Clear, Store A-star and Store B-star and for console display.

Rows 4X-7X (CID cards 0340-0370). BCD to EBCDIC conversion table. These four rows provide the constants for translating the 64 possible BCD configurations to the appropriate EBCDIC bytes.

Rows 8X and 9X, Bytes 0-7 (CID cards 0380, 0390). Magnetic Tape and Card Load controls.

Byte	Bit	Use
80	0	Initialized off. When on, do first tape-load instruction as a 1400 initial tape load.
	1-3	Initialized off. Indicates last 1400 tape unit addressed.
	4-7	Tape-Control unit address: 8 to F on multiplexor channel 0 to F on selector channel
81	0-1	Tape density for 1400 mode tape drive 1: 00 = 200 bpi on 7-track drive 01 = 556 bpi on 7-track drive 10 = 800 bpi on 7-track drive 11 = 800 bpi on 9-track
	2	Initialized off. If on; a backspace was the last operation performed on 1400-mode tape drive 1.
	3.	Initialized off. If on, an end-of-file condition is outstanding on 1400-mode tape drive 1.
	4-7	System/360 unit address assigned to be 1400-mode tape drive 1.
82-86		Same as byte 81, for 1400-mode tape drives 2 through 6.
87		Initialized to 08. Last status byte received from tape-control unit.
88-8F		Refer to Miscellaneous Control Bytes.
90-91		The sixteen bit binary address of the first 1400-mode instruction to be executed when the 1400 mode initial card program load (1400 IPL) is used: ROAR reset to ROS-address 1402 or 1442 procedure.
92-93		The sixteen-bit binary address of the first 1400-mode instruction to be executed when the 1400-mode tape IPL is used: ROAR reset to ROS-address 1729 procedure.
94-96		Not used.
97	0	If on, System/360 tape drive 0 is a 9-track unit.
	1	If on, System/360 tape drive 1 is a 9-track unit.
	2	If on, tape drive 2 is a 9-track unit.
	3	If on, tape drive 3 is a 9-track unit.
	4	If on, tape drive 4 is a 9-track unit.
	5	If on, tape drive 5 is a 9-track unit.
	6	If on, tape drive 6 is a 9-track unit.
	7	If on, tape drive 7 is a 9-track unit.

MISCELLANEOUS CONTROL BYTES
Row 8x 88-8D, Row 9x 98-9F, Row Ax A8-AF and Row Bx B8-BF (CID Cards 0380, 0390, 03A0, and 03B0) are 32 control bytes.

Byte	Bit	Use
88		I-register back-up } I-STAR in binary. Initialized to location of first
89		J-register back-up } 1400-mode instruction.
8A		U-register back-up } B-STAR in binary. Initialized
8B		V-register back-up } to 00 00.
8C		L-register back-up } A-STAR in binary. Initialized
8D		T-register back-up } to 00 00.
8E		G-register back-up
8F		S-register back-up
98		Sense Switch Byte Initialized to 00
	0	Sense switch A (last card)
	1	Sense switch B
	2	Sense switch C
	3	Sense switch D
	4	Sense switch E
	5	Sense switch F
	6	Sense switch G
	7	Not used
99		Hi-Lo-Eq Byte Initialized to 00
	0	High-compare result (U)
	1	Unequal-compare result (/)
	2	Low-compare result (T)
	3	Equal-compare result (S)
	4	Not used
	5	Overflow indicator (Z)
	6	Inquiry-request indicator (Q)
	7	Not used
9A		2030 storage size byte
	1F	8,192
	3F	16,384
	7F	32,768
	FE	65,536
9B		Disk Status indicators (Branch Byte) Initialized to 00
	0	Unequal-address compare (X)
	1	Access busy (\)
	2	Wrong-length record (W)
	3	Any-disk condition (Y)
	4	Disk error (V)
	5	Not Ready (N)
	6	Read-back-check interlock
	7	No X followed by Seek
9C		Status of Features
	0	I/O check-stop switch
	1	Advanced programming feature
	2	Expanded print edit feature
	3	Mode-switch on invalid operation codes (Initialized to 0 unless using Programmed Mode Switch)
	4	Not used (Initialized to 1)
	5	Mode-switch on halt (Intialized to 0 unless using PMS)
	6	Tape units on selector channel 2
	7	Mode-switch on error stops (Initialized to 0 unless using PMS)
9D		Mode-Switch Status
	0	Mode-switch on invalid I/O ops
	1	Mode-Switch on console ops
	2	Mode-Switch on printer ops
	3	Mode-switch on reader/punch ops
	4	Not used
	5	Mode-switch on tape ops
	6	Mode-switch on disk ops
	7	Not used

9E		D-register back-up (1400-series A-register) Initialized to 00
9F		Allow I/O Traps
	0	Not used
	1	Not used
	2	Not used
	3	Alternate 9-track-tape mode
	4	Initialized to Zero
	5	Allow I/O Traps (PMS is used)
A8 A9 AA		Working Storage. Initialized to 00.
AB		Working Storage
AC		Constant (1F)
AD AE AF		Used for sterling feature. Initialized to 00.
B8		
B9		NPL address of the last tape unit addressed
BA		Constant (0F)
BB BC BD		Working Storage. Initialized to 00.
BE		Constant (2E)
BF		Working storage Initialized to 00
Row Ax	0-3	Alternate track (cylinder to head) location. Address transfer when seeking alternate track
	4-7	(CID card 03A0) Working Storage
Row Bx	0-7	(CID card 03B0) Working Storage

Rows Cx, Dx, Ex, and Fx (CID cards 03C0, 03D0, 03E0, and 03F0). 1400 operation code decode table. These four rows (64 bytes) provide the translation of 1400 operation codes to a bit significant form that is usable by the 2030. In the chart (Figure 5-31) the 1400 series op code is shown in parenthesis. The hex values shown outside parenthesis are in storage. The code 34 is used to indicate an invalid 1400 series op code. The code 06 is used to indicate a No Op.

Notice that several 1400-series special features such as Branch if Bit Equal (W op code), Divide (% op code), etc., are standard with the 1400 compatibility feature. These 1400-series operations can be made invalid by inserting the invalid code (34) in the corresponding table location if desired. This permits the system to detect 1400-mode programming errors, such as a divide operation where none was intended.

Auxiliary Storage MPX

Rows 0x, 1x, 8x, 9x, Ax, and Bx (CID cards 0400, 0410, 0480, 0490, 04A0, and 04B0). Disk File Functions. These six partial rows while not contiguously located, can be classified as disk file functions.

Row 0x Bytes 00-09 (CID card 0400). File Units Digits Cylinder and Head decode for converting from 1401 to binary. Bytes 0A-0F: Not used.

Row 1X (CID card 0410).

Byte	Bit	Use
10	0=0	Compare Disable is inactive. A successful address compare on a 1400 mode indelible address must occur before an indelible address (I/A Read or Write can be executed (M/L%F6/@BBBR/W).
	0=1	Compare Disable is active. Read or Write with I/A operations will be executed without first doing an address compare on a 1400 mode indelible address. This bit should be set to 1 only when initializing a disk pack in 1400 mode.
1=0		Module Overflow Detection is active. The 1400 program module value, within each disk control field, is compared against a module value pre-set in auxiliary memory. If the module values don't match, a coded stop occurs with 60 displayed in the console R-register. Correct module values must be set in MPX memory locations. i.e., 11 for drive 0, 12 for drive 1, etc.
10	1	Bit 1=1, Module Overflow Detection is Inactive.
11		Module value for Drive 0. Normal value 00

12 Module value for Drive 1. Normal value 00

13 Module value for Drive 2. Normal value 00

14 Module value for Drive 3. Normal value 00

15 Module value for Drive 4. Normal value 00

If a 1400 program requires a module value other than 0, the correct module value for the drive unit must be placed in auxiliary storage.

16-1F Not used

Row 8X (CID Card 0480)

80 File sense 0
81 File sense 1
82 File sense 2
83 File sense 3
(Variable) Sense information from the file control units. Initialized to 00

84-87 Not used Initialized to 00

88-8F Refer to Miscellaneous Control Bytes

Row 9X (CID Card 0490) Disk Functions

Byte	Bit	Use
90		System/360 control and unit address for 1400 mode disk module 0
91		(Initialized 01) Cylinder number position of access mechanism for 1400 mode disk module 0.
92		System/360 control and unit address for 1400 mode disk module 1.
93		(Initialized 01) Cylinder number position of access mechanism for 1400 mode disk module 1
94		System/360 control and unit address for 1400 mode disk module 2
95		(Initialized 01) Cylinder number position of access mechanism for 1400 mode disk module 2
96		System/360 Control and unit address for 1400 mode disk module 3
97		(Initialized 01) Cylinder number position of access mechanism for 1400 mode disk module 3
98		System/360 control and unit address for 1400 mode disk module 4
99		(Initialized 01) Cylinder number position of access mechanism for 1400 mode disk module 4
9A-9F		Refer to Miscellaneous Control bytes.

Row Ax (CID Card 04A0) Disk Functions

Byte	Constant	
A0	00	File-cylinder tens digit decode constants, 1401 to binary.
A1	32	
A2	0A	
A3	3C	
A4	14	
A5	46	
A6	1E	
A7	50	
A8	28	
A9	5A	

AA-AF Refer to Miscellaneous Control bytes

Row BX (CID Card 04B0) B0-B4: Head and Record Values. Also key length and data length.

Bytes B5-B7 Not used Bytes B8-BF: Refer to Miscellaneous Control bytes.

Row 2X (CID Card 0420). Hundreds high conversion. The values in positions 20-29 are constant for a particular operation. These values are determined by the 1400/2030 core size relationship and include carries (if any) from the hundreds low table. The actual values for each possible combination are shown in Figure 4-33. Positions 2A-2F are not used. An invalid hundreds digit would be detected during hundreds low conversion.

Core Size 1400/2030	Auxiliary Storage MPX Location									
	20	21	22	23	24	25	26	27	28	29
16K/16,384	01	01	02	02	03	03	03	04	04	05
16K/32,768	41	41	42	42	43	43	43	44	44	45
16K/65,536	C0	C0	C1	C1	C2	C2	C2	C3	C3	C4
12K/16,384	11	11	11	12	12	13	13	13	14	14
12K/32,768	51	51	51	52	52	53	53	53	54	54
12K/65,536	D0	D0	D0	D1	D1	D2	D2	D2	D3	D3
8K/ 8,192	00	01	01	01	02	02	03	03	03	04
8K/16,384	20	21	21	21	22	22	23	23	23	24
8K/32,768	60	61	61	61	62	62	63	63	63	64
8K/65,536	EF	E0	E0	E0	E1	E1	E2	E2	E2	E3
4K/ 8,192	10	10	11	11	11	12	12	13	13	13
4K/16,384	30	30	31	31	31	32	32	33	33	33
4K/32,768	70	70	71	71	71	72	72	73	73	73
4K/65,536	EF	EF	F0	F0	F0	F1	F1	F2	F2	F2
2K/ 8,192	18	18	18	19	19	1A	1A	1A	1B	1B
2K/16,384	38	38	38	39	39	3A	3A	3A	3B	3B
2K/32,768	78	78	78	79	79	7A	7A	7A	7B	7B
2K/65,536	F7	F7	F7	F8	F8	F9	F9	F9	FA	FA
1.4K/ 8,192	1A	1A	1B	1B	1C	1C	1C	1D	1D	1E
1.4K/16,384	3A	3A	3B	3B	3C	3C	3C	3D	3D	3E
1.4K/32,768	7A	7A	7B	7B	7C	7C	7C	7D	7D	7E
1.4K/65,536	F9	F9	FA	FA	FB	FB	FB	FC	FC	FD

1400/2030 Core Size	CID Card 0420 Column Locations																		
	6	7	8	9	11	12	13	14	16	17	18	19	21	22	23	24	26	27	28	29

Figure 4-33. Hundreds High Conversion; Auxiliary Storage MPX Row 2X Values

Row 3X (CID Card 0430). Initialized to 00. Rows 4X-7X and Cx-Fx (CID cards 0440-0470 and 04C0-04D0) EBCDIC to BCD conversion table. These eight rows provide a 128 character table that contains a BCD code corresponding to each EBCDIC character. Some positions of the table are dependent upon the printer typebar arrangement that the system uses. Variations are shown in Figure 4-34. Figure 4-31 shows the arrangement for the 1403 Printer chain/train. EBCDIC to BCD translation is required during the execution of instructions such as Bit Test, Move Zone and Move Numeric.

System/360 Typebar

1400 Auxilliary Storage B	0	1	2	3	4	5	6	7	8	9	A
4x		61				4D	4C	4F	80		
5x						5D	5E	5F	50	50	
6x						6D	7E	4E	90	60	
7x			40			7D	6E	7F			
Cx											6F
Dx											5A
Ex											4A
Fx											70

1403 Printer with 1442/1443 Subfeature

4x		61							80		
5x									50	50	
6x									90	60	
7x			40								
Cx											4E
Dx											60
Ex											4E
Fx											F0

52-Character 1400-Series Typebar

4x		41		4C	4D				80	60	
5x				5C	5D				40		
6x				6C	7E				90	50	
7x		4E	7B	7C	7D	7E					
Cx											4B
Dx											5B
Ex											6B
Fx											79

63-Character 1400-Series Typebar

4x		41				4D	4E	4F	80	60	
5x						5D	5E	5F	40		
6x			6A			6D	6E	6F	90	50	
7x		7F	7A			7D	7E				
Cx											4A
Dx											5A
Ex											6A
Fx											79

Figure 4-34. Table-Load Constants for 1443 Graphics Variations

MISCELLANEOUS CONTROL BYTES.
Rows 8x; 88-8F, 9x; 98-9F, Ax; A8-AF and Bx; B8-BF (CID cards 0480, 0490, 04A0 & 04B0)
These 32 control byte locations are used to contain various compatibility mode data. We will define each byte (and bit) function.

Byte	Hex Char	Compatibility Mode Use General Stops
88		Error code (displayed in the Main storage Data Register) identifying a programmed or error stop.
	00	Normal stop. Appears when the stop is caused by the stop key ending an instruction-execution in instruction-stop mode or getting a match in SAR delayed-stop mode.
88	01	Attempted to use invalid 1400 B-address
	02	Attempted to use invalid 1400 A-address
	03	Attempted to use invalid 1400 A- and B-address
	04	Attempted to use invalid 1400 operation code
	05	Invalid I/O operation attempted; unit selection or unit number invalid
	06	Storage wrap occurred when address was used that was outside of system capacity
	07	Storage protection occurred in 1400 mode
	08	Attempted to switch to 2030 mode without the PMS feature
	09	Invalid source or destination address on one of the special PMS tape operations
	0A	Attempted to convert to binary an address that was less than the bias on a clear storage or store STAR operation
	0B	Storage wrap around 1400-address 000.
	0C	Attempted to start a 1400 I-cycle at main storage address 0000
	0E	Attempted to index without advanced programming comment in CID
	21	Word mark missing from 1400 operation code during I-op
	31	Word mark in A-address of an I/O instruction
	41	An 8F character was detected at an address other than the offset address while in 1400 mode
	51	An I/O operation was attempted on a device for which the compatibility feature is not installed
	55	A 1400 start-reset function was performed using the console interrupt key
	*F0	A 1400 halt instruction with no invalid addresses
	*F1	A 1400 halt instruction with invalid B-address
	*F2	A 1400 halt instruction with invalid A-address
	*F3	A 1400 halt instruction with invalid A- and B-address
	FF	A 1400 halt and branch instruction has been executed.

Byte 88	Hex Character	Reader, Punch & 1402/1403 Sub-feature	Printer Stops 1442/1443 Sub-feature
	80	Punch-transfer error or no address compare	Wrong address sent back from the channel
	90	Operational-in disconnect on 2540 or 2501 reader	Invalid d-modifier
	A0	Operational in disconnect on 1403 printer	No GMWM in storage

B0	Operational in disconnect on 2540 or 2520 punch.	1442 error on read or punch operation or 1443 error on Print operation

*Note: These stops indicate execution of all 1401 Halt instructions (•) except the four character Halt and Branch (•AAA). The status of the existing address in the STAR is provided but does not indicate an error stop. (Ex: M%U1BBBW• would cause an F2 Halt.)

3F	2540 or 2501 reader error, or invalid character occurred	
4F	2540 or 2501 reader intervention required	1442 reader intervention required
5F	2540 or 2520 punch intervention required	1442 punch intervention required
6F	1403 printer intervention required	1443 printer intervention required
7F	2540 stacker-select instruction given after maximum time-out (6ms after a card read)	

Byte 88	Hex Character	Channel S=Selector M=Multiplexor	Tape Stops
	22	SM	Wrong address sent back from channel
	42	SM	Invalid channel status on data transfer
	52	SM	Device-end signal before encountering a GMWM on a tape-write operation
	62	S	Status-in and service-in on a tape write
		M	Operational-in disconnect on a tape write
	82	S	Status-in and service-in on a read-move operation
	92	SM	Tape error on a 1400 tape initial program load
	8F	SM	Tape-unit intervention required

Byte 77	Hex Character	Channel S=Selector M=Multiplexor	Tape Stops
	A2	S	Invalid channel status was received on a branch-if-error operation
	B2	S	Status-in and service-in on a 1400 read-load operation

C2	M	Operational-in disconnect on a read operation
D2	M	Premature end to a sense operation
E2	M	Operational-in disconnect-on-mode-set operation

1050 Stops

11	Some other device attempted to take a multiplexor-channel data cycle while in the data-transfer portion of a 1050 operation
CF	1050 intervention required
DF	Alter or display stop

Disk-File Stops

10	Read-back check stop
20	No channel or device ends received
30	Wrong address sent back from the channel
40	Unit-check status response to seek command
50	Operational interlock

Meaning

Byte	Bit	1402/1403 Subfeature	1442/1443 Subfeature
89	0	On during first part of a Read Op. Stays on if there is a late Stacker Select.	Not Used
	1	51-column read feed	Not used
	2		
	3	Reader address	Reader 1 address
	4	Reader address	Reader 1 address
	5	Reader address	Reader 1 address
	6	Reader address	Reader 1 address
	7	Reader address	Reader 1 address
8A		Temporary forms control Initialized to 08	Temporary forms control Initialized to 00
8B	0	On for the read portion of a PFR Operation	Channel 9
	1		Forms After

	2	If an error occurred during the Punch Portion of PFR, this bit turns on during the read portion	Channel 12
	3	Punch address	System/360 train/chain or typebar configuration
	4		Not used
	5		Not used
	6		Command Type
	7		
8C	0	132 print positions	
	1	Not used	
	2	If a forms Op follows a print error, this bit turns on	
	3	Printer address	Reader 2 address
	4		
	5		
	6		
	7		
8D	0	Temporary stacker-select information Initialized to 1D for 2540, OD for 2501 or 2520 where all cards on 2520 go into stacker 1; 4D for 2501 or 2520 where valid cards 2520 go into stacker 2 while error cards are automatically directed into stacker 1.	Not used
	1		
	2		
	3		Printer address
	4		
	5		
	6		
	7		

Byte	Bit	Compatibility Mode Use
8E		Overlap STAR high (Initialized to Z value of storage bias)
8F		Overlap STAR low (Initialized to Y value of storage bias)
98-99		Refer to Row 9X Disk Functions
9A	0	Last 1400 tape operation was a forward-space-record
	1	This 1400 tape operation is a forward-space-record
	2	Tape erase latch is on
	3	Disk Control Field has been incremented
	4	1050 Error
	5	Card reader error
	6	Card punch error

	7	Printer error
9B		Tape track in error
9C		Tape Control Byte for EOF
9D		A-register (hundreds) backup
9E		B-register (hundreds) backup
9F		1050 status
	0	Load mode (initialized off)
	1	Type arrangement H element on 1052
	2-7	Not used
A8-A9		Refer to Row Ax, Disk Functions
AA		Previous operation on file
AB		00
AC		00
AD AE AF		00) Decoded disk control field 00 } cylinder and high order head 00) values (Refer to B0-B4).
B8		Disk Control Field Cylinder value buffer
B9		Present 2400 Tape Unit Address (in tape ops) File Unit number in 1401 form (0, 2, 4, 6, 8) in File ops
BA		Working storage
BB	0	Odd redundancy operation (Tape)
	1	This operation is using 9 track tape unit
	2-7	Present command byte with flags
BC		Working storage
BD		File Unit address for initial selection
BE		Last File Command sent to File Control Unit
BF		Scan Condition

PROGRAMMED MODE SWITCHING

- The programmed mode switching sub-feature provides the ability to switch the 2030 processor from 1400 compatibility mode to 2030 mode and vice versa under 2030 program control.

- The programmed mode switching sub-feature is available on 16K or larger System/360 Model 30's.

- A program mask in 9C and 9D LS controls execution of mode switching.

- Special 2030 move instructions are provided to facilitate programmed mode switching.

- Multiple 1400 programs may be stored and executed selectively by programmed mode switching facilities.
- Programmed mode switching is implemented by microprogramming.

The Programmed Mode Switching (PMS) subfeature enables 2030 programs and 1400 programs to reside in storage coincidentally, and to be executed in an interleaved manner by providing the capability to switch the processor between 1400 compatibility mode and 2030 mode under control of the 2030 program.

Special System/360 instructions are provided to control and facilitate communication and data movement between the 2030 program and the 1400 programs. This permits the use of System/360 capabilities and devices that are not other wise available in 1400 compatibility mode operations. For example, tape read or write operations can be executed in 2030 mode to take advantage of available simultaneous TAU capabilities.

The programmed mode switching subfeature is available on 2030 models having 16,384, 32,768, or 65,536 bytes of storage. The basic compatibility feature is a prerequisite.

If the 1400 program requires 16,000 positions of core storage, then at least a model E30 (32,768 bytes of storage) is required in order to accommodate a minimum 2030 program.

Utilizing the capabilities of PMS it is possible to operate with multiple 1400 programs in storage, within the limits of storage capacity. When switching from one such 1400 program to another, it is necessary to switch to 2030 mode to reload certain areas of 1400 Auxiliary storage with data applicable to the 1400 program to be executed. An example of such data is Rows 2X (Hundreds 1400 address conversion tables) and Rows 8X and 9X (Disk and Tape control bytes, 1400 instruction address, Tape Densities, Unit address assignments, etc.)

Special instructions facilitate saving, in main storage, the contents of auxiliary storage that are pertinent to program #1, and loading into auxiliary storage the information (previously put in main storage) that is pertinent to 1400 program #2. The process is reversed to change back to program #1.

Switching from 1400 compatibility mode to 2030 mode can be controlled by bytes 9C and 9D in 1400 Auxiliary storage LS. The conditions that can be selected to cause mode switching are as follows:

Invalid Op Code

Halt Op Code

Error Stops

Invalid I/O Ops

Console Ops

Printer Ops

Reader/Punch Ops

Tape Ops

File Ops

To switch from 1400 compatibility mode to 2030 mode, a supervisor call interrupt is executed. When mode switch interruption occurs, the new and old PSW's are used in the normal manner. A coded byte is entered in positions 24-31 of the old PSW to indicate the reason for the interruption. For additional information on Supervisor Call Codes, refer to Compatibility Mode Interruptions SRL, Form A24-3255. The location of the last accessed compatibility instruction is also stored in the old PSW except for mode switches due to certain error conditions. The current status of 1400 registers is stored in backup locations in auxiliary storage so that they may be referenced when necessary.

Return to 1400 compatibility mode is via the Mode Set (CFIM) instruction. Refer to the SRL manual for additional information (format, etc.).

Operations and Special Instructions

- Special System/360 instructions are made available by the PMS feature.
- The basic feature special instructions are utilized to manipulate auxiliary storage.

- Auxiliary storage manipulation is performed in 2030 mode.

In performing programmed mode switching applications, an important consideration is the moving of data between 1400 compatibility storage and 2030 storage.

A tape-read operation in 2030 mode could insert the data directly into the 1400 compatibility area of storage. However, the 1400 program would then have no indication of the amount of data received. The special compatibility PMS sub-feature instructions transfer data from 2030 storage to 1400 compatibility storage (and vice versa) and also updates the 1400 A-star and B-Star appropriately. This allows the next Op code after an I/O instruction (for example, Store B-Star) to function with the correct address.

Two move and two load instructions are used to emulate 1400 treatment of word spearators and word marks. For example, in emulating a 1400 tape write operation in 2030 mode, it is necessary to move or load data from 1400 compatibility storage to 2030 storage, then perform the write operation via 2030 channel. A tape read Op is performed by reading via 2030 channel to 2030 storage, then moving or loading data from 2030 storage to 1400 compatibility storage, then continuing the 1400 program.

Compatibility Feature Move To Compatibility

This operation is similar to a tape-move operation in the 1400. Data is moved from the 2030 storage area to the area reserved for the 1400 portion of the program. The binary address specified by general register R_2 is the source address in the 2030 core-storage area. Word marks in the source field are not moved; word marks in the destination field remain undisturbed. The binary address specified by general register R_1 is the destination address in the 1400 core storage area. The low-order 16 bits of the general register specified by $R_1 + 1$ is the count field.

The two addresses are incremented by 1 and the count is decremented by 1 in the specified general purpose register after each byte is moved. The count is checked for zero before each byte transfer. If it is zero, the operation is terminated and a groupmark is inserted in the destination field. The operation is also terminated on GMWM detection in the destination field, in which case no character is moved.

The destination address is always updated to 1 beyond the GMWM or GM address at the end of the move. The effective 1400 B-storage address register is set to this address.

Compatibility Feature Move From Compatibility

This instruction is identical to the Move To Compatibility instruction with the following exceptions:

1. The movement of data is from 1400 compatibility storage to 2030 storage.
2. GMWM detection is done on the source field.
3. When a GMWM terminates the instruction, the source address is incremented to one beyond the GMWM address, and the destination address remains unchanged. The effective 1400 B-storage address register is set to this (source) address.
4. No group mark is inserted in the destination field on termination by count.

Compatibility Feature Load To Compatibility

The function of this instruction is simialr to a tape-load operation. Data is loaded from 2030 storage to 1400 compatibility storage. The binary address specified by general register R_2 is the source address and the binary address specified by general register R_1 is the destination address. The low-order 16 bits in general register $R_1 + 1$ is a count of the number of bytes to be handled in the source field.

Word marks in the destination field are cleared. When a word-separator character is detected in the source field, the count is decremented by 1, the source address is incremented by 1 and no character transfer takes place. A word mark is inserted with the first character following (in the source field).

Termination on GMWM detection or count, and setting of the condition register is identical to that of the move to compatibility operation. The effective 1400 B-storage address register is set to the final destination address.

Compatibility Feature Load From Compatibility

This instruction is identical to the Load To Compatibility instruction with the following exceptions:

1. The movement of data is from 1400 storage to 2030 storage.

2. Word marks in the source field cause a word-separator character to be inserted in the destination field. The destination address is incremented by 1, the count is decremented by 1 and the source address remains unchanged.

3. GMWM detection is done on the source field.

4. When a GMWM terminates the operation, the source address is incremented to 1 beyond the GMWM, and the destination address remains unchanged. The effective 1400 B-storage address register is set to this source address.

5. No group mark is inserted on termination by count.

Auxiliary Storage Manipulation

Auxiliary Storage Manipulation is accomplished by utilizing the basic compatibility feature instructions; Compatibility Feature Store Variables, Compatibility Feature Load Variables, Compatibility Feature Store Constants, and Compatibility Feature Load Constants.

For programmed mode switching applications, these instructions are used for altering the contents of auxiliary storage, thus controlling the conditions under which mode switching occurs. This facility also makes it possible to perform multiprogramming. (Multi-1400 programs.)

Compatibility Feature Load/Store Variables

These two instructions are used to alter any of the 64 bytes of Auxiliary Storage LS rows 8 and 9 and Auxiliary Storage MPX rows 8 and 9. The Load instruction transfers four bytes from 1400 Auxiliary Storage into main storage, the Store instruction transfers four bytes from main storage into 1400 Auxiliary Storage.

Most variables contained in 1400 auxiliary storage (such as 1400 instruction address, tape densities, and unit address assignments,) can be changed with these instructions. A selected variable field can be transferred from auxiliary storage, altered as desired then transferred back into auxiliary storage.

Refer to the SRL Manual, Form A24-3255 for additional information on instruction format, byte addressing codes, etc.

Compatibility Feature Load/Store Constants

These instructions enable altering any of the 512 bytes of 1400 Compatibility auxiliary storage. Transfer from auxiliary storage to main storage is done by the Load instruction, transfer from main storage to auxiliary storage is done by the Store instruction. Information is transferred by blocks of 16 contiguous bytes (one complete row in auxiliary storage).

Multiprogramming is an application for these instructions. In switching from one 1400 program to another, various bytes of information in auxiliary storage must be changed (address conversion constants etc.). This information can be transferred out of auxiliary storage into main storage, then previously stored conversion information applicable to a second 1400 program can be transferred into storage.

LOGIC FLOW CHARTS

The logic flow charts that appeared in a previous edition of this manual have been deleted. Refer to the Maintenance Diagram Manual for revised versions of these charts. These charts can be used as instructional and maintenance aids. The charts have been revised and expanded so that a minimum amount of additional description is needed.

Two general types of charts are presented.

1. Objectives; the overall picture. Useful as an introduction and a recall device.

2. Details; the actual operations through the microprogram.

I-CYCLES

- 1400 addresses are converted to 2030 hexadecimal equivalent; address error detection is performed.
- 1400 operation codes are converted to 2030 type (bit significant) operation codes.
- Indexing is performed if designated.

The objectives of I-Cycles in 1400 Compatibility Mode are essentially the same as for I-Cycles in the 1400. There are important additional (support) objectives that are necessary as a means of accomplishing the I-phase (conversion, etc.) in compatibility mode.

Refer to the I-Cycle flow charts in the Maintenance Diagram Manual. 1401 I-cycle and indexing flow charts are included for comparison.

I-phase is of variable length, depending on the length of the 1401 instruction.

The conversion of 1400 decimal addresses to 2030 hexadecimal equivalent address is performed by table lookup as described in "Address Conversion." In a similar manner, the operation codes are converted to obtain codes that are bit significant for easy identification by the microprogram. Refer to "Op Code Conversion and Recognition."

After the 1400 operation code is converted and placed in the G-register a variety of paths are available, depending on the type of instruction, length of instruction, etc. Comments and examples that are included with the flow charts (Maintenance Diagram Manual) explain the I-phase operation. An example of indexing is presented.

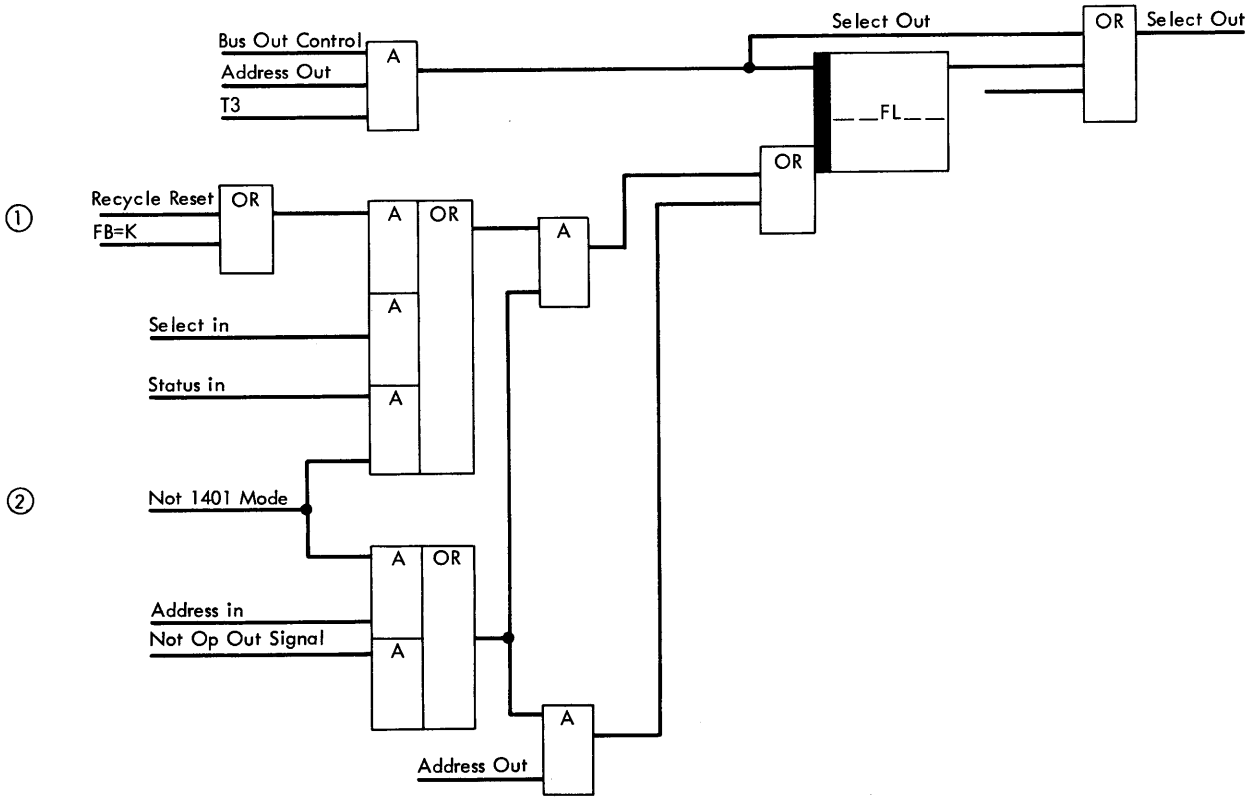
At the completion of I-phase the microprogram starts execution phase for the instruction. The G-register contains the Op Code; LT has the A-address, UV has the B-address, and IJ contains the address of the next instruction.

I/O OPERATIONS

- All I/O operations in compatibility mode are executed in burst mode.
- The 1402 reader automatically feeds a card 6 milliseconds after a read command.
- A stacker select command for the 1402 must be given within 6 milliseconds after a read command.
- End of file occurs with Channel End of the last card read.
- Character representation to and from I/O apparatus is in EBCDI code.
- Operation code bit structure changes during execution of an I/O Operation.

All 1400 systems I/O operations are executed in burst mode. Tape and file operations always force burst mode on the multiplex channel (compatibility mode included).

Burst mode for 1402 and 1403 operations is forced in the 2030 by holding up Select Out until Channel End occurs (Figure 4-35).



1. 1401 Reset Of Burst Mode
2. Blocks Normal Reset to Select Out

Figure 4-35. Select Out

The normal resets of Select Out in 2030 mode is blocked by the line "Not 1401 Mode." The only resets available to turn off Select Out are Recycle Reset, Select-In, or the microprogram statement K->FB, where no bits or only one bit of the K value is on. Example: K0, K1, K2, K4, or K8. Parity bit setting has no effect on the statement. Recycle Reset is the result of giving a system reset or a recycle reset when in CE mode.

In compatibility mode, Select In from the channel can come up only due to an abnormal condition existing, such as having ICU power turned off or a machine failure. Therefore, the only controlled reset to Select Out will be the microprogram statement K->FB, which is given when a Channel End is sensed by the 2030.

Refer to the I/O operation flowcharts in the Maintenance Diagram Manual. As I/O operations are executed, the bit significant Op code changes, indicating the status of the operation. A summary of the Op Code information appears with the I/O operation flow charts. For example, in executing the Print-Read-Punch instruction, the original converted Op Code (27) changes to 25 (Read-Punch) upon completion of the Print operation; to 24 (Punch) upon completion of the Read operation; and to 20 (indicating Operation completed) upon completion of the Punch operation.

The first section of the I/O Operations flow chart is the Op decode portion. This portion illustrates the sequence of preliminary setup that occurs prior to the start I/O command. This includes testing the Programmed Mode Switch byte, setting up registers and performing other initialization requirements of the I/O command. Various I/O operations are discussed under the appropriate headings.

1402 Read Operations

The functions of the 1402 attached to a 1400 system are performed by an IBM 2540 Card Read-Punch or an IBM 2501 Card Reader. Recall that with a 1402 operating with a 1400 system, the programmer has 10 milliseconds after a read-a-card instruction during which to give a stacker select instruction. The card being read then, feeds past the read brushes and into the selected stacker.

When a 1402, 2540 or 2501 is operating on a System/360, Model 30 in 2030 mode, a

read command causes the buffer to transmit data to the CPU, but no card movement occurs.

To cause the 1402, 2540 or 2501 on a 2030 in 1400 compatibility mode to duplicate the stacker select action of a 1402 on a 1400 system, a Provisional Feed circuit has been added. Basically, this circuit does the following: 6 milliseconds after the read command data transfer starts, a feed cycle occurs, and a card is read and selected to the normal pocket. If, during the 6-millisecond timeout period, a 1401 or 1460 stacker select command is sensed, a feed and stacker select command is issued to the 1402 (2540 or 2501). This feed command causes the card to feed immediately and select to the stacker. It also prevents the provisional feed from occurring. See Figure 4-36.

The microprogram for stacker select detects whether or not the 6 ms timeout is over by issuing a sense command prior to the feed and stacker select command. If the status byte coming back from the channel does not contain the attention bit, the 6-millisecond stacker select time has expired (Figure 4-36). In this case, the microprogram does not issue the stacker select command but indicates an invalid stacker select to the operator by a coded byte (7F) in the R-register (MSDR).

Another modification of the reader circuitry on the 2030 changes the end-of-file condition. When the reader is operating in 2030 mode, an extra Read command must be issued after the last card is read in order for the end-of-file condition to occur. In 1400 compatibility mode on the 2030, End-of-file occurs with Channel End of the last card read. This allows the branch on last card to occur without issuing an extra read command.

In the Maintenance Diagram Manual, refer to: I/O and read operation flow charts; I/O Op Decode; I/O Setup; K and F Ops; Branch Ops (Reader, Punch, Printer); Reader Data Loop; and Read Instruction Objectives.

The Read Instruction Objectives chart is largely self-explanatory. This chart presents the overall objectives. Refer to

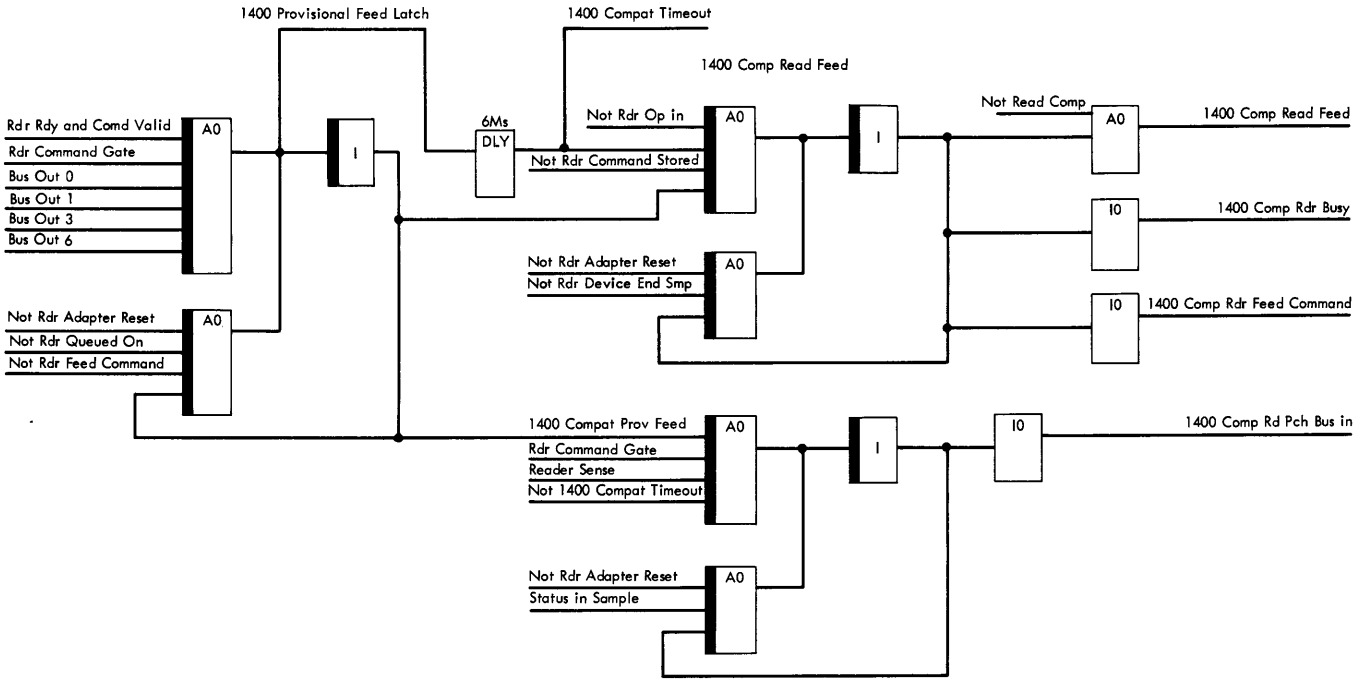


Figure 4-36. Card Read

the other charts for details of read operation execution. Read Column Binary is included.

Refer to the Reader Data Loop flow charts for details of data handling during a read operation.

1402 Punch Operations

1402 Punch Operations in 1400 compatibility mode differ from punch operations in 2030 mode as follows: A stacker select command given in 2030 mode causes the card entering the punch station to be selected. A stacker select command given for a 1402 punch in a 1400 system causes the card at the punch check station to be selected. In the 1400 compatibility feature (1402/1403 Subfeature), microprogramming causes the stacker select operation to emulate the 1401 system.

To cause the stacker select command to select the card at the punch check station while in 1400 compatibility mode, the punch 3-bit modifier latch has been added to the control circuitry. The punch 3-bit modifier latch turns on when a 3-bit is on the bus with a punch command.

When the punch 3-bit modifier latch is on, it prevents the punch stacker sequence 1 latch from turning on and causes the punch stacker sequence 2 latch to come on at punch counter F-E time.

Refer to the Maintenance Diagram Manual I/O and Punch operation Flow chart: I/O Op Decode, I/O Setup, Printer, Punch Data Loops, and Punch Objectives.

The I/O operations required by the 1400 object program are performed by their respective microprograms. The 1400 I/O commands sense status, perform the operation, and detect any errors that occur during the operation.

For a further illustration of these points, examine the 1402 punch objectives flow chart. (The device used is the 2540).

The microprogram first defines the operation to be performed. Next it fetches the unit address from local storage and issues a sense command. The microprogram then examines the status byte that is coming back to ensure that the 2540 can accept the command.

If the 2540 status is good, the command is given and the microprogram goes into a data loop. (Refer to Printer, Punch Data Loop). The operation is done in burst mode because select out cannot be reset. When Channel End occurs (but not as a direct result), the microprogram gives the statement FB->K, that resets select out and allows the 2540 to disconnect (electrically) from the channel.

The microprogram examines the status byte that comes with Channel End in order to determine whether an error occurred. If an error occurred, the microprogram executes an error routine that displays a coded byte in the R register. If there were no errors, the microprogram interrogates the Op code again to determine whether or not it is a combined operation (read, punch etc.).

If operation is not a combined operation, the microprogram exits to I-cycles for the next Op code.

1403 Printer Operations

The System/360 Model 30 with the 1400 Compatibility Feature and the 1402/1403 Subfeature can process 1400 programs that utilize the 1403 printer. The printer must be attached to the multiplexor channel through a 2821 control unit.

The printer is buffered (print buffer is standard) on the System/360. However, there are no programming differences except for channel-9 and -12 interrogation. (Refer to Functional Differences.)

Forms operations and print operations objectives are illustrated separately in the Maintenance Diagram Manual. For forms operations, a space or skip after print is conbined with the next Print command. A space or skip immediate instruction sets up and issues a sense command and proceeds with execution.

For print instructions, the microprogram analyzes the operation, sends device address, issues sense command, sends service out, and performs various other details in preparation for print instruction execution. The microprogram then enters a data loop that sends 1 character at a time to the print buffer. Table lookup is preformed, if necessary, for character conversion.

Detection of the last character to be printed terminates the data loop and performs the routine for disconnecting electrically. The Op code is changed to indicate completion of the print operation and the next instruction is read out.

1442 Reader-Punch Operation

When running the 1442 in compatibility mode, there are two main operational differences between the 1442 on the 1440 system and the 1442 on a 2030 in compatibility mode.

The 1442 when reading or punching in compatibility mode does not stop on a column in error but continues to the end of the card. The microprogram tests for errors at the end of the card operation. The second difference modifies the last card indication. Last card (end-of-file) occurs with the Channel End of the last card read.

The 34 MLP characters transmitted by the 1442 are changed by the microprogram so that the 8-9 punches that designated the characters as MLP characters are eliminated. The characters in core storage are the EBCDIC equivalent of the card code minus the 8-9 punches.

1443 Printer Operation

The character configuration of the 52- and 63-character typebar for the 1443 N1 is not the same as the 52- and 63-character bar for the 1443 on the 1440 system. To run the 1443 N1 in compatibility mode, the 52- or 63-character bar for the 1440 system must be installed in the 1443 N1.

Because the characters on the bar are not the same as the 1443 N1 bar, the microprogram must construct different character configurations to send to the printer (Figure 4-37). If the character in storage to be printed is an A, the microprogram sends a J to the 1443 N1. The 1443 N1 circuits fire the hammer when it determines there is a J in front of the hammer. Actually, because the typebar is from the 1440, there will be an "A" in front of the hammer when the print compare equal for a "J" occurred.

Magnetic Tape Operations

Some important differences between tape operations in 1440 compatibility mode on the 2030 and in the 1401 or 1460 should be noted.

The 1401 or 1460 stores an End-of-file in the tape unit as a tape indicate. Tape indicate is reset by unloading the tape unit or by branching on the End-of-file condition. In 1400 compatibility mode on the 2030, the End-of-file condition is stored as a bit in the tape unit control byte in local storage. This bit is reset by a rewind-unload instruction or a branch on End-of-file. The tape indicate in the tape unit is set only by the End-of-file reflective strip on the tape during a write instruction, and is reset by any backward command.

Because the End-of-file bit in local storage is not reset by manually unloading the tape unit, the operator must ensure that the bit is reset when reloading the tape unit to eliminate false End-of-file conditions.

CHARACTER IN STORAGE	CHARACTER TO 1443	CHARACTER IN STORAGE	CHARACTER TO1443
&	&	Q	Y
-	-	9	8
1	0	≠	I
/	A ①	I	R
A	J	R	≠
J	/ ②	0	9
2	1	≠	>
S	B	?	<
B	K	!	≠
K	S	#	:
3	2	,	.
T	C	.	$
C	L	$	,
L	T	@	#
4	3	%	←
U	D	¤	*
D	M	*	%
M	U	:	@
5	4	V	(
V	E	(	)
E	N	)	¬
N	V	>	∀
6	5	\	+
W	F	<	;
F	O	;	-
O	W	¬	=
7	6	+++	≠
X	G	≠	≠
G	P	Δ	±
P	X	Blank	¬
8	7	Blank	Blank
Y	H	≠	Blank
H	Q		

① Sent As 01000001
② Sent As 11100001

Figure 4-37. 52 and 63 Character Typebar Decode

A tape error during initial program load causes a microprogram stop with a coded byte in 88 of Auxiliary Storage MPX. The Tape Control information in Auxiliary Storage is in AUX storage LS locations 80 through 87. Refer to Figure 4-38. In byte 80 (TCU Control), bit 0 on indicates an initial program load condition. Bits 1, 2, and 3 contain the 1401 or 1460 address for the last tape unit addressed. Bits 4 through 7 contain the tape control unit number as assigned by the CID.

	0		1		2		3		4		5		6		7
BX	TCU Control	TCU Address	TU1 Control	TU1 Address	TU2 Control	TU2 Address	TU3 Control	TU3 Address	TU4 Control	TU4 Address	TU5 Control	TU5 Address	TU6 Control	TU6 Address	Last Status From TCU

Bytes 81 - 86 = 1400 Units 1 - 6 Respectively

Figure 4-38. Tape Control Information -- Bytes 80 through 87 Auxiliary Storage LS

Bytes 81 through 86 are Tape Unit 1 through 6 controls. Bits 0 and 1 provide the density or Unit identification as follows:

00 = 7 track @ 200 BPI

01 = 7 track @ 556 BPI

10 = 7 track @ 800 BPI

11 = 9 track (Density information is in LS 92 and BB)

Bit 2 on in the TU control bytes (81-86) indicates that the last operation performed on that particular unit was a backspace operation.

Bit 3 on indicates an End-of-file condition is outstanding for that unit.

Tape Unit 1 to 6 address (bits 4-7) contains the 2030 address assignment of the tape unit to be used as a 1401 or 1460 unit.

Miscellaneous bytes in AUX stor are used for tape operations as follows:

Byte		M/S	
8E&8F	MPX	MS	"O" STAR locations used during a read operation as back-up for the starting address of the read in area (B-STAR).
9B	MPX	MS	Track in Error Sense byte that is stored if a read error occurs on a 9-track tape. This byte is used for the mode set of a 9 track unit.
9A	MPX	MS	Bit 0=1; last 1400 tape operation was a forward space record Bit 1=1; Current 1400 tape operation is a forward space record Bit 2=1; the tape erase latch is on.
9F	LS	S	Bit 3=1; Alternate Redundance Mode.
B9	MPX	SM	Temporary storage of tape unit control byte address for unit being used.
B9	LS	SM	NPL address of the last tape unit addressed.
BA	MPX	M	Storage location used in Read Mask of setting H5 bit. The actual address of the unit being used is also stored here.
BB	MPX	SM	Temporary storage of command byte used to hold the command during the Mode Set routine. Bit 0 = 1 denotes Odd Redundancy Bit 1 = 1 denotes 9 track
BC	MPX	M	Temporary storage of the read status byte for 9 track operation.
Byte 97 and BB	LS LS	SM	Seven/nine track and density status (refer also to bytes LS 81-86). These bytes are required in addition to bytes 81-86 to define density when 9-track tape is used. Bit 0 of each byte has track status for tape unit 0, bit 1 has status for tape unit 1 etc. Status is as follows for associated tape unit:

Byte 97	Byte BB	
1	0	9-Track @ 800 BPI
1	1	9-Track @ 1600 BPI

Refer to the Maintenance Diagram Manual. Separate charts present Tape Selector Channel Objectives and Tape Multiplexor Channel Objectives. The two operations have much in common. The common circuit details are shown on the chart Tape-Common. From the common chart, the details are developed separately for selector and multiplexor

operations. The various tape operation charts are corss referenced for ease in following an operation through the micro-program.

Disk Compatibility Operation

- 1311 Disk Packs must be reloaded and reformatted (by utility programs) to enable operation with the 2030.
- The 1311 file must be loaded into cylinders 1 to 100 on the 2311 to provide compatibility. Microprograms increment and decrement 1400 cyl values as needed to compensate.
- Disk Compatibility microprograms are not shared with any other feature.
- RBC can be overridden for diagnostics by altering one CCROS card.

The Magnetic Disk Operations sub-feature permits processing IBM 1311 magnetic disk-file records in either track or sector mode in 1400 compatibility mode using the IBM 2311 Disk Storage Drive. The 2311 must be attached to the multiplexor channel through a 2841 file-control unit.

The 2311 uses the same disk-pack (1316) as the 1311. However, because of increased recording density and different format, existing 1311 files must be reloaded using the 2311 on the 2030. The 1311 file must be loaded into cylinders 1 to 100 on the 2311.

System/360 Model 30 file-unit addresses must be assigned to correspond with 1311 module numbers. This is done during initial program load by the CID. Ten bytes are reserved in auxiliary storage for this purpose. (90-99 in LS). Five bytes are reserved for seek control (AD-B1 in LS).

Because formatting is a major consideration, and must be done before compatibility operations can be performed, we will first examine formats before discussing actual operations.

FORMATS

Home Address

The Home Address is the binary equivalent of the cylinder and head physical location. Refer to Figure 4-39 for format. Home addresses are prewritten by a utility program.

F	C	C	H	H
	0		0	

Figure 4-39. Home Address Format (Showing Fixed Value)

Home addresses are not used directly in the execution of compatibility mode file operations. The Home Addresses serves as a reference point. The symbols F, C, C, H and H stand for Flag, Cylinder, Cylinder, Head and Head. The values of the first C and the first H (from left to right) are fixed at 0.

Record Zero (R0)

Record Zero (R0) for each track is prewritten by a utility program. Record Zero is normally the same as the physical cylinder and head address. An exception exists when an alternate track is assigned to replace a damaged or defective track. Refer to Alternate Track.

Record Zero format, count, and data fields are shown in Figure 4-40. The data field is normally 8 characters. It is used for handling alternate track situations in 2030 mode. The R0 data field is not used in 1400 compatibility mode.

R0 Count								R0 Data							
C	C	H	H	R	K_L	D_L	D_L	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8
0		0		0	0	0	8	X	X	X	X	X	X	X	X

Figure 4-40. Record Zero Format (Showing Fixed Value)

Count, Key and Data Fields (CKD)

The count field provides an indelible address (I/A) for each record. Refer to Figure 4-41. The count field for sector mode operations specifies a data length of 100. The count field for Record mode Operations specifies a data length of 2980.

Sector Operations														
	Count								Data					
C C H H R	C	C	H	H	R	K_L	D_L	D_L	D_1	D_2	D_3	⟨⟨	D_{98}	D_{99} D_{100}
Numeric Value	0		0			0	0	100						
Hexadecimal Value	00		00			00	00	64						
Full Track Record Operations														
	Count								Data					
	C	C	H	H	R	K_L	D_L	D_L	D_1	D_2	D_3	⟨⟨	D_{2978}	D_{2979} D_{2980}
Numeric Value	0		0			0	2980							
Hexadecimal Value	00		00			00	0BA4							

Figure 4-41. CKD Format (Showing Fixed Value)

When writing a data field in Load Mode, the data transfer is truncated after the 90th data character is transferred to the File Control Unit. Short records are filled with valid blanks (40 hex) through the 90th character. The File Control Unit fills the 91st through the 100th character of the record with all Zero-bit bytes.

When writing a data field in move mode and a short record is encountered, the 1400 compatibility microprogram sends valid blanks (40 hex) to the File Control Unit until a transfer of 100 or 2980 characters is completed. This is done for short records to provide correct mode detection when reading. Refer to Error Checking.

RECORD NUMBER. Record numbers in the count field are numbered 1 through 20. 1400 series record numbers are incremented by 1 before being written on a disk and before executing a compare address. Conversely, after a record number is read from a disk, it is decremented by 1 before it is stored in main memory.

This handling of record numbers permits a record zero, while maintaining a unique number for each data record.

HEAD NUMBER. The head numbers that are written on the disk (0 through 9) are the same as the head value specified by the Disk Control field.

CYLINDER NUMBER. The cylinder numbers that are written on the disk are one greater than the cylinder value specified by the disk control field. These values are 1 through 100.

MODULE OR UNIT NUMBER. In 2030 format, a module number is not recorded on the disk pack as was done in 1400 disk operations. Thus, there is no protection against accidentally using the wrong pack. Module overflow detection is provided however.

DATA FIELD. Data is written on the disk in EBCDIC code. Each record is written in the data field identified by its unique count field.

Operations

Disk operations, as in 1400 series, fall into three major groups: Seek, Read or Write. The 1400 operation code is as follows:

M/L %Fn BBB R/W

Values of n and meanings are:

0 = Seek

1 = Sector mode

2 = Track-Record mode

3 = RBC

4 = Not used

5 = Sector Control Overlay

6 = Sector mode with I/A

7 = Scan (Lo or Eq)

8 = Scan (EQ)

9 = Scan (Hi or Eq)

a = Track Record mode with I/A

SEEK COMMAND. Seek commands are either Return To Home (RTH) or Direct Seek. Direct seek, an option with 1400 series, is standard with the 1400 Compatibility Feature (Disk Sub-Feature).

RTH Seek: To execute an RTH seek, the disk Control Field is decoded to binary values and a 2030 full seek command is issued to the File Control Unit. A 6-byte address transfer follows. The 6-byte address specifies in Binary values:

B B C C H H

0 0 0 0

The 0 values are not significant, as only the cylinder value is needed to perform seek.

Direct Seek: To execute a direct seek, the Disk Control Field (DCF) is decoded to determine the difference between acutal location and desired location. A plus or minus sign denotes the direction of the difference. This value is added (algebraically) to the cylinder location stored in auxiliary storage (91, 93, 95, 97, or 99--LS) for the appropriate drive unit. The RTH seek value is sent to the File Control Unit during the 6-byte address transfer.

Note: It is necessary to ensure that the control unit is not busy before updating a cylinder value in auxilairy storage. Otherwise, the true location of an access can be lost.

RTH Seek (Recalibrate): A re-orientation/recalibrate seek command chain, is initiated by the following sequence of events:

1. A read or write operation resulting in X and not WLR branch indicators
2. Seek command
3. A second read or write resulting in X and not WLR
4. A return to home seek command

This sequence discriminates between cylinder overflows and access disorientation, and provides seek recovery without impairing thruput time.

Cylinder Displacement: During execution of a seek Op code, the cylinder value is incremented by one. Thus, a seek to cylinder 0 actually places the access at cylinder 1. This reserves cylinder 0 for the Initial Program Load (IPL) function. 1400 Series programs occupy cylinders 1 through 100.
Caution: 1311 Diagnostics (4F series) should not be run on a CE disk in compatibility mode. Adjustment data will be erased.

Direct Seek Special Case: Some existing customer programs cause a Direct Seek in the reverse direction (outward), with a number-of-cylinders value, which results in a cylinder value of less than zero. The 1311 drive will seek to minus one cylinder, turn around and finish out the seek in the forward direction. This behavior has been simulated arithmetically for compatibility operation, to provide program compatibility.

Read/Write Operations With Indelible Address

A Head Seek command must precede all Read/Write commands to ensure that the head specified by the Disk Control Field is the head selected. If a search equal ID command is issued to emulate an address compare function, two equal address compares are necessary. First, a compare equal on a 1400 mode formatted indelible address must be made. (For disk packs not formatted in 1400 mode, this first compare is bypassed by setting the auxiliary storage LS position 10 bit 0 to 1). Upon satisfying the first compare equal, the cylinder value in the compare argument is set to the seek cylinder value. The second number is set to 0 and an address compare equal on record zero must be satisfied.

This provides correct time orientation so that subsequent indelible address and data fields will be written in their correct locations.

Substitution of the seek cylinder value is necessary to provide for cases where the Disk Control Field contains an abnormal cylinder value. Cylinder overflow recognition is provided when compare disable is active.

Cylinder Values: The cylinder value written on the disk pack is one higher than the value specified by the Disk Control

Field. Because all seeks are microprogram incremented by one, the R0 cylinder value, written in 2030 mode, will be the same as the indelible address cylinder value written in 1400 mode (in most cases).

When indelible addresses are read from a disk, the cylinder value is decremented by one before being placed in main memory.

All indelible address search arguments taken from the Disk control field are microprogram incremented by one.

Head Values: The Head value during the Address Compare operation will be the same as the value specified by the DCF.

When formatting a disk in 1400 mode, the head specified by the Disk Control Field must be the same as the head which executes the command.

An exception is made for Alternate Track operation.

MODULE OVERFLOW DETECTION. This function is necessary when a 1400 program uses a change in module value in the Disk Control Field to set the No Address Compare (No-X) branch indicator and branch to a seek routine.

Module overflow detection (byte 10, bit 1 off in MPX) is accomplished by comparing the Disk control Field Module Value to a corresponding Module value that is preset in the MPX Auxiliary Storage. Module values are:

1400

Drive Number	MPX Location	Normal Value
0	11	00
2	12	00
4	13	00
6	14	00
8	15	00

The module values are normally set to 00 so that changes in module value can be detected, and abnormal module values written by the disk-label utility program will not present a problem.

The operator must set the module values into the MPX manually as required by the operation to be preformed. When a module mismatch is detected, a coded stop occurs displaying 60 in the MSDR.

Module Overflow or mismatch detection can be made inactive by setting MPX byte 10 bit 1 to 1.

Missing Address Mark Detection: The compatibility microprogram checks that the last record number read into memory on address ops is 16 or greater. If the last record number is less than 16, it is assumed possible that address marks might have been missed, and the No Address Compare (X) branch indicator is set.

Zero the DCF 6th Digit: When reading with indelible addresses into main memory, the 6th digit of the Disk Control Field is set to zero. This prevents residual characters in main memory from being transferred to tape or other permanent records.

READ BACK CHECK (RBC). Read Back Check interlock is provided to ensure that a RBC instruction follows each write instruction. If an instruction other than RBC follows a write operation, the instruction is not executed and the system returns to I cycles and stops.

It is possible to override RBC interlock for testing purposes by altering a branch condition in the microprogram. Refer to CAS logic page QH301 address 1D06. The statement Z=0 must be changed to Z=1. Remember to restore the original card upon completion of testing.

Word Marks and Zone Bits in Control Fields. Disk Control Fields: The Disk Control Field word marks, zone bits, and numeric values are unaltered during the execution of Read/Write operations with indelible addresses because no arithmetics are performed on the DCF.

Sector Count Field: The sector count is decremented each time a record is transferred to or from a disk. The sector count field is modified to conform with 1400 series arithmetic operations as follows:

Load Mode: Zone bits and word marks are removed from the sector count field.

Move Mode: Zone bits are removed, word marks are retained in the sector count field.

At the end of a Read or Write operation--(free of error conditions), the final A-Star value is BBB+9.

Read/Write Operations, Sector Mode.

This operation is a normal customer application.

As previously stated, a Head Seek command precedes all Read/Write commands. A Search Equal identification (ID) command is issued to emulate an Address Compare. The record number in the search argument will be the binary equivalent of the 1400 series record number, plus 1. Data record numbers on a disk are 1 through 20.

The head number in a search argument will be the binary equivalent of the 1400 series head number in the DCF. The cylinder value in a search argument will be the binary equivalent of the 1400 series cylinder number in DCF plus 1.

SCAN OPERATIONS. The Scan Feature is standard with the Disk Compatibility Subfeature. This scan is independent of the Scan Feature provided by the File Control Unit.

OVERFLOW CONDITIONS. This section discusses the overflow conditions affecting Head, Cylinder and Module values.

Head - When execution of a Read/Write operation requires Head switching, the multi-track bit is set on. This causes the next head to search the first record. After one search, the multi-track bit is reset.

Cylinder - When the execution of a Read/Write operation goes beyond the last record on surface 9, the head value is set to 0 and the cylinder count is incremented by 1. This forces a No-Address-Compare branch condition to be set following the next search. At this time, WLR branch condition is also set (to simulate 1400 operation).

Module - When the execution of a Read/Write operation exceeds the capacity of a drive unit or module, the Disk Control Field is updated to the first address on the next module. If the 5th drive unit gets a module overflow (exceeds 099999), the resulting Disk Control Field following incrementation is 000000.

WORD MARKS AND ZONE BIT HANDLING--SECTOR MODE. The DCF word marks and zone bits are altered whenever the original sector count is two or greater. In load mode, the zone bits and word marks are removed from the Disk Control Field in main memory. This does not include the alternate module select position which is not processed arithmetically during incrementation.

In move mode, Except for alternate module select position) the DCF Zone bits are removed and the wordmarks are saved in main memory.

The Sector count field word marks and zone bits and Final B-Star values are treated the same as for Indelible Address values.

The final A-Star value upon error free completion of a Read/Write operation is BBB+6.

ERROR CHECKING AND BRANCH CONDITIONS. Error conditions posted by the file control unit are interpreted and translated into 1400 series branch conditions as follows (Branch Byte 9B in LS).

Bit	Condition	Symbol
0	No Address Compare	X
1	Busy	\
2	Wrong Length Record	W
3	Any Disk Condition	Y
4	Parity	V
5	Not Ready	N
6	RBC Interlock is On	

A unit check posted by the file control unit as a response to a 2030 mode file command (initial status) is interpreted directly as a not ready condition and the not ready branch condition is set.

When a unit check is posted by the file control unit as a part of ending status, a sense command is issued to the file control unit, the results are examined and corresponding branch conditions are set. Sense byte interpretation is as follows:

Sense Byte	Sense Bit	1400 Equivalent
0 (MPX 80)	0	N
	1	N
	2	V
	3	N or V
	4	V
	5	V
	6	Defective Track, go to Alternate Track
	7	N
1 (MPX 81)	0	Not Checked; Duplicate Information
	1	N

	2	X, W
	3	Not Checked; Duplicate Information
	4	X
	5	Not Checked; Duplicate Information
	6	Unused
	7	Not Checked; 2030 Feature
2 (MPX 82)	0	N
	1	Unused
	2	V
	3	N
	4	N
	5	Unused
	6	Unused
	7	Unused
3 (MPX 83)	Byte not Examined	CE Use

Some branch conditions are recognized directly or evaluated during the execution of file commands. Busy can be recognized when posted during initial status in the initial selection sequence. Busy can be recognized as a short busy signal sequence when the disk is being formatted.

If Busy is caused by a microprogram decision (example: busy because of a seek to an alternate track) the microprogram will loop in the initial selection sequence until the busy is cleared, then issue the final command.

If Busy occurs under conditions that the macro-programmer would not anticipate (example: A RBC instruction is specified by the macro-programmer, and the programmer expects the file to be not busy), the microprogram must take the initiative, looping until the busy condition is cleared, then re-issue the command.

If a Busy and Device End are encountered during initial status, the unit in question is not busy as soon as the Device End is accepted. In this case, Device End is accepted and the file command is set to the File Control Unit. This is a convenient way to clear seek completes following the first seek of a file.

In situations where a macroprogrammer can anticipate a Busy condition, the Busy Branch condition is set when a Busy condition is detected.

For Wrong-Length Record, a counter is set for each data transfer and decremented during the data transfer. When the group mark with a word mark in main storage and a count of 0 do not coincide, the Wrong-Length Record (WLR) branch condition is set.

On scan operations, WLR is set only if the group mark with word mark fails to precede the end of data field by two bytes or more.

MODE CHECKING. Checking is done to ensure that records are read from a file in the same mode as they were written on the file.

Read Load Mode Check--The 91st data character is read from the disk and examined for a word mark. If the record was written in Load Mode, the 91st character should contain a word mark. If this requirement is not met, the validity condition is set to simulate 1400 series setting. Reading full track record in load mode, the 2683rd character is examined.

Read Move Mode Check--The 100th data character is examined for a word mark. The 100th character (and preceding 99 characters) should not have a word mark. If this requirement is not met, the validity condition is set. Reading full track record in move mode, the 2980th character is examined.

Alternate Track Operation

Alternate tracks can be assigned for any imperfect or damaged tracks on a disk pack. Provision is made to seek an alternate track when necessary, process, then return to the original track, either to continue processing with the next sequential head, or to end the operation (Figure 4-42).

An alternate track situation is recognized following an address compare operation (Search ID).

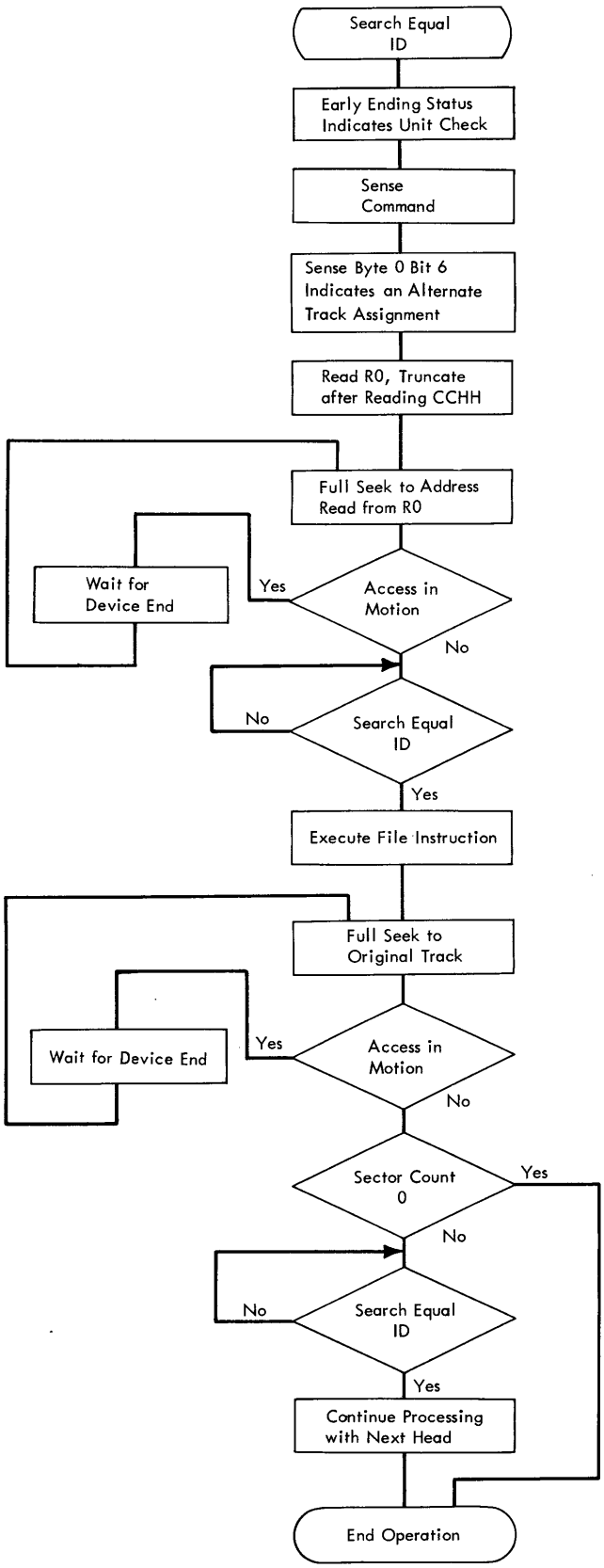


Figure 4-42. Alternate Track Operations

ALTERNATE TRACK FORMATTING. Alternate track formatting (by a formatting program in 2030 mode) is prerequisite to alternate track operations in 1400 compatibility mode.

Alternate tracks will normally be assigned to cylinders 201, 202, and 203, but this is not necessary on disks used only for 1400 file programs.

An alternate track can have a head assignment different from the head value of the original track. It is possible to write other than normal indelible address cylinder values on an alternate track, but not an abnormal head value.

A normal address is an address that could be expected to appear on the original track.

CONSOLE INQUIRY (1050)

When operating in 1400 compatibility mode, the 1050 performs the functions of the 1400 inquiry station. Because the graphic representation of 1400 system defined character is not the same as the 1050 graphic representation of EBCDIC characters, some graphic conversion is necessary to obtain the correct character printout for 1400 system characters.

The microprogram converts the characters going to and coming from the 1050 in accordance with 1400 graphic representation. For example, if a "?" (1100 0000) is sent to the 1050, it is first converted by the microprogram to 1100 0111 to comply with the special typehead.

There is one operational difference between a 1447 on a 1400 system, and a 1050 on a 2030 in compatibility mode. A character in error on a typewriter read operation prints as an underscore (_) on a 1400 system, but on the 1050, the character prints as the character it most closely resembles. However, the character in error presents an error condition to the 2030 that can be tested by the 1400 system object program.

The special typehead should be used on the 1050 when in 1400 compatibility mode to get the correct character printout.

1620 COMPATIBILITY

- The 1620 Compatibility Feature consists of:

  A 4,032 position Read Only Storage (ROS)

  A 1620 Emulator Program

  ROS control field changes

  2030 console changes

The purpose of the 1620 Compatibility Feature is to emulate the IBM 1620 system as closely as possible while processing 1620 programs. There are restrictions and limitations placed on the program being run and the 1620 system being emulated. These restrictions and limitations may be found by referring to IBM System/360 Model 30 1620 Compatibility Feature, Form A24-3365.

The 4,032 position ROS is in addition to the standard ROS and is located to the rear of and adjacent to the standard ROS. This additional ROS contains all the microprograms necessary to perform the logical, arithmetic, internal data transmission, and most of the program control instructions. The input-output instructions are handled by the compatibility microprogram up to the point of translation and format testing, the Input-Output Control Program (IOCP) takes control at this point.

The IOCP is part of the 1620 Emulator Program which is loaded into the 2030 prior to any 1620 program or group of programs. The initialization deck consists of an initialization program, the IOCP, the Disk Format program and variable data control cards.

The control cards contain information about the 1620 system being emulated and the System/360, Model 30 being used. This information, such as storage capacities, system configurations, and special feature information, is used by the 1620 compatibility microprogram to properly process and direct the 1620 programs being run.

The IOCP handles the data transmission between I/O units and the input-output areas in storage. After an input-output instruction is translated and recognized by the compatibility microprogram, control is transferred to the IOCP, the machine leaves compatibility mode, and the data is handled in 2030 mode. Translation of data is handled by the compatibility microprogram.

ROS control fields CH, CL, CM, and CS have changes made in their functions to perform specific operations needed by the 1620 compatibility feature. These changes are brought about through 6 SLT cards; these cards and the ROS field changes are shown in Figure 4-43.

SLT Cards Producing the Changes to the ROS Field

Frame	Gate	Board	Socket
01	A	A1	L4
01	A	A2	J2
01	A	A3	L7
01	A	B2	B3
01	A	B2	B4
01	A	B2	C2

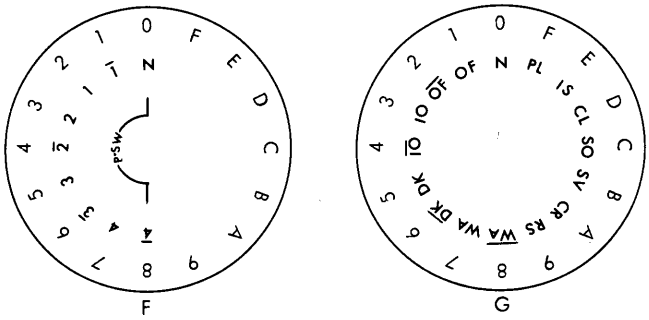
ROS Field Changes

ROS Field	Decode	Normal Function	1620 Compatibility Function
CH	3	V00	S1
	4	STI	RHVDD
	5	OPI	RLVDD
	8	SI	R2
CL	3	AI	RL = E
	4	SVI	G1
	6	IBC	R1
	C	G1	R3
CM	5	T→MN	LT→MN
CS	5	TREQ→S1	1 →S1
	F	K→FA	0 →S1

Figure 4-43. ROS Control Field Changes

The 2030 console is changed by altering the F and G rotary switches. The F and G switches allow the emulation of the 1620 program switches, machine check switches, 1311 write address switch, and control keys. The functions of the F and G switches for 1620 compatibility mode are indicated by the inner ring of labels. These compatibility functions are operative only when the 2030 is in compatibility mode and the machine is stopped (Figure 4-44). The F switch emulates the 1620 program switches 1-4 by setting or resetting bits 4-7 of 9B LS. Program switch 1 is represented by bit

7, switch 2 by bit 6, 3 by bit 5 and 4 by bit 4. 9B LS also contains the following 1620 switches; Write-Address key, Disk switch, I/O switch and the Overflow switch. Write-Address key is represented by bit 0 of 9B LS, Disk switch by bit 1, I/O switch by bit 2, and Overflow switch by bit 3. The remaining functions of switch G are not test type switches and perform a particular job immediately upon activation. Read Console Procedures in IBM 1620 Compatibility Feature, Form A24-3365, for details of F and G switch functions.



F Switch - Sets or Resets 1620 Program Switches 1 - 4 into Bits 4 - 7 of Byte at 9B LS

G Switch - Sets or Resets 1620 Switches. Write Address (W/A), Disk (DK), Input Output (I/O), and Overflow (OF), into Bits 0 - 3 of Byte at 9B LS

Bits	0	1	2	3	4	5	6	7
9B LS	W/A	DK	I/O	OF	PS-4	PS-3	PS-2	PS-1

Figure 4-44. F and G Switches

PROGRAM DATA HANDLING AND STORING

- The digit and flag portions of 1620 characters are separated and stored into specific areas of mapped storage.
- The 1620 digits are stored two per byte and the flags are stored eight per byte.

The mapped areas of 2030 storage for 1620 program data vary with the storage size of the 1620 being emulated.

1620 Storage Size	Digit Locations	Flag Locations
(Decimal)	(Hexadecimal)	(Hexadecimal)
20,000	0E00-350F	3600-3FC3
40,000	0E00-5C1F	6000-7387
60,000	0E00-832F	9000-AD4B

The 1620 digits are stored two per byte into the mapped digit area of storage. Bits 0-3 of a byte in the digit area receive the digit from an even 1620 address. Bits 4-7 of the same byte receive the 1620 digit from the next higher 1620 address which is odd. Example; the byte at 0E00 contains the digits from 1620 storage locations 00000 and 00001. The digit from 1620 address 00000 is contained in bits 0-3 and the digit from 1620 address 00001 is contained in bits 4-7. The flag portions of the 1620 characters are stored as bits 8 per byte into the mapped flag area of storage. Example; the flag from the character at 1620 address 00000 is stored in the 0 bit position of the first byte in the flag area. The 1 bit position of the first byte contains the flag from the 1620 character at 1620 address 00001. See Figure 4-45 for storage example.

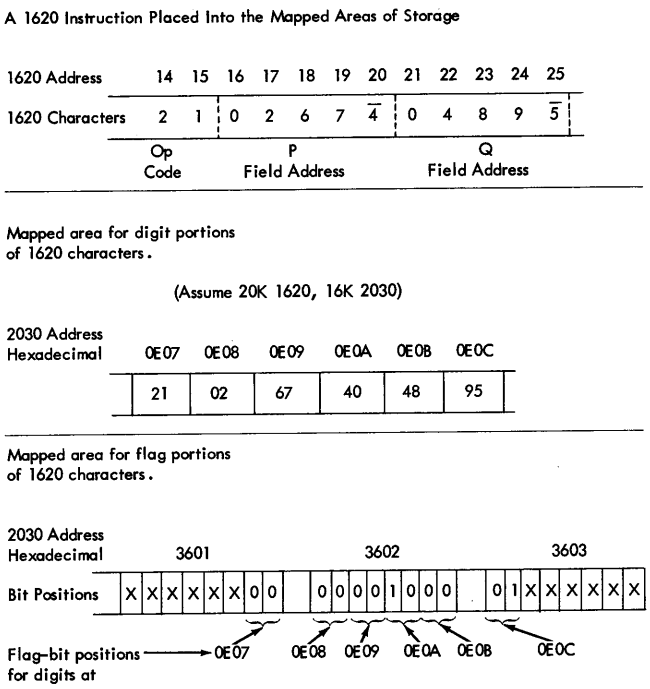


Figure 4-45. Digit and Flag Storage

MODE SWITCHING

- Setting the 3 bit of the W register to 1 allows addressing of the additional 4K ROS.
- The W3 bit on indicates the 2030 is in compatibility mode.
- Setting of the W3 bit is accomplished by issuing one of the special System/360 compatibility operation codes, or by the F switch on the console.

The W3 bit may be turned on in three ways:

1. Console switches. Setting the F switch on the console to any odd hexadecimal digit turns on W3 bit.
2. Micro program control UV->WX. The status of U3 determines W3 when the micro program statement UV->WX is used.
3. Micro program control CA->W. The status of AA determines W3 when this statement is used.

The 1620 compatibility program is properly entered after a stop of some kind, by dialing 1620 into the switches, pressing System Reset, ROAR reset, and the Start key in that order. If the machine had been properly initialized, and the LS and MPX auxiliary storage areas assigned to the compatibility feature had not been altered, the 1620 program should run. The program will start with the next sequential instruction.

SPECIAL COMPATIBILITY OP CODES

- The Special Compatibility Op Codes are enabled by the Diagnose instruction.

- There are five 99 type special op codes in the SI format, and three other special op codes in the RR format.

The Diagnose instruction in the SI format provides a means to enable and disable the special op codes. When the diagnose instruction is used, the displacement will contain the hexadecimal address 3CC and a base of 0. When the immediate operand is 80 the special op codes are enabled. When the immediate operand is 00, the special op codes are disabled and will cause a program interruption if used.

To enable the special op codes use;
838003CC

To disable the special op codes use;
830003CC

The 99 op codes in the SI format and their functions are as follows:

1. 99 00 B+D, Loads 512 bytes of information into the auxiliary storage areas listed as MPX and LS for compatibility mode. Loading takes place from 2030 main storage, starting at the address specified by the B+D effective address.

2. 99 10 0 000, branches to 1620 I/cycles and executes these instructions located at 0DE0. These three instructions are; clear mapped core, read a card, and branch to 1620 address 0 (0E00 1620 mapped digit area). This routine is the initial program load from cards.

3. 99 30 0 000, branches to 1620 I cycles and executes three instructions located at 0DF0. These three instructions are; clear mapped core, read from the typewriter, and branch to 1620 address 0 (0E00 1620 mapped digit area). This routine is the initial insert from typewriter.

4. 99 2R B+D, Reads a byte from auxiliary storage and stores it at a 2030 main storage location specified by B+D. The low digit of the second byte of this instruction designates one of the general registers. The general register contains in its low byte the address coordinator of a byte in auxiliary storage. The high byte of the general register contains a code to indicate the auxiliary storage area to be addressed; 00 = M/LS, 01 = LS, 02 = MPX, (these mnemonics are the 1620 compatibility designations).

5. 99 6R B+D, Reads a byte from 2030 main storage designated by B+D, and stores it into an auxiliary storage area. The auxiliary storage address is derived in the same manner as for the 99 2R B+D instruction.

The three special op codes in the RR format and their functions are:

1. $0CR_1R_2$, issued by the IOCP program to switch to 1620 compatibility mode and enter the move and translate microprogram. The R1, R2 field of this instruction contains the routine pointer that indicates to the move and translate microprogram the proper routine to execute (Figure 4-46).

2. $0DR_1R_2$ issued by the IOCP to enable a 1620 mode system interlock. The machine stops in 1620 compatibility mode at ROS address 10FF. The two hexadecimal characters specified in the R1, R2 fields of this instruction are displayed in the MSAR (Figure 4-46). Pressing the start button returns control to the next System/360 instruction of the IOCP.

3. $0ER_1R_2$, issued by the IOCP to switch to 1620 compatibility mode and enter some disk operation. The code determining the disk operation to be performed is in the R1, R2 field of this instruction (Figure 4-46).

0CXX	Operation Code Modifiers
Modifier XX	Routine to Perform
00	Alpha Output
20	Numeric Output
0C	Binary Input
04	Alpha Input
24	Numeric Input
64	Numeric Input (Flagged Character on Type)

0DXX	Operation Code Modifiers
Modifier XX	Causes for Stop
20	Invalid Typewriter Control
21	No Device Address
22	Machine Check
30	I/O Release
33	Program Check
44	Invalid Input Command
55	Invalid I/O Device Code
77	Error in Reading Overlay from Disk
88	Paper-Tape Overrun
F0	Reader Check Stop
F5	Paper-Tape Read
F6	Card Punch
F7	Card Read
F8	Printer
F9	Disk Drive 0
FA	Disk Drive 1
FB	Disk Drive 2
FC	Disk Drive 3
FD	Disk Track Read or Write Error

Device Not Available

0EXX		Operation Code Modifiers
Modifier XX		Routine
00		Go to 1620 I-Cycles
11		Disk Sector Mode Interrupt
20		Disk Track Mode, Data Transfer to Buffer
22		Disk Track Mode, Data Transfer to Mapped Core
28	WLRC	Disk Track Mode, Data Transfer to Buffer
2A	WLRC	Disk Track Mode, Data Transfer to Mapped Core

● Figure 4-46. Special Op Code Modifiers

AUXILIARY STORAGE 1620 COMPATIBILITY MODE

- Two 256 byte Auxiliary Storage areas are needed by the 1620 Compatibility Feature.

The Auxiliary storage areas assigned to the 1620 Compatibility Feature are designated by the mnemonics MPX and LS (Figure 4-47). The 2 auxiliary storage areas contain address and op code translation tables, special characters, index registers, instruction counters, and other pertinent data for use by the 1620 Compatibility feature.

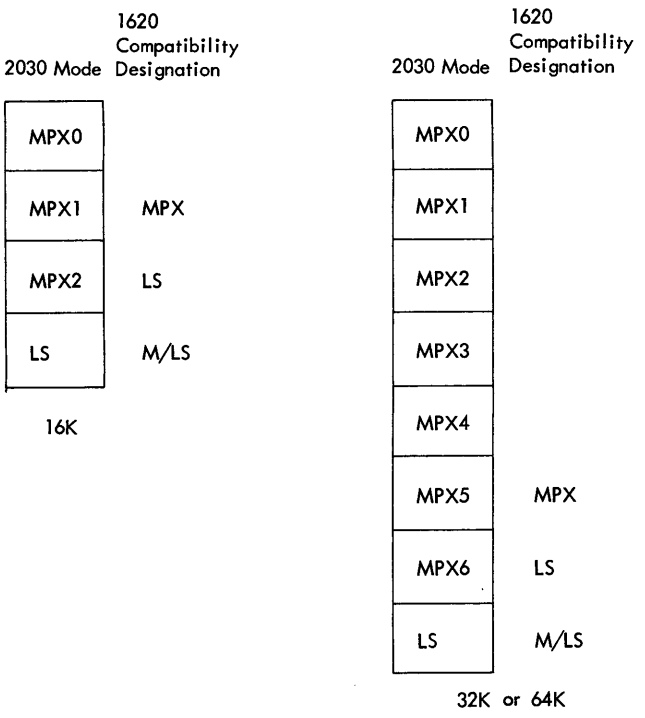


Figure 4-47. Auxiliary Storage Designations

The 2030 auxiliary storage area labeled LS is used by the 1620 compatibility feature during I/O operations. General Registers are used in the following manner:

General Register	Use
2	Contains the character count less one during data transmission in I/O operations.
3	Contains the current buffer address
4	Contains the I/O indicators
5	Contains the current flag mask and flag address
6	Contains the current digit address
7	Contains the device code
8	Contains the routine code.

Bit 1 of the byte at 8C 2030 LS when set to one indicates to the 2030 that the next instruction should be taken from the address in the bytes at A9, AA of 2030 LS. If the bit is zero the next instruction will be taken from the address in the IJ registers.

The byte at BB 2030 LS contains the condition codes as set by the Move and Translate microprogram, before it branches to the IOCP. These condition code settings in BB 2030 LS are;

Bit 0 = 1, Record Mark or Group Mark detected.

Bit 1 = 1, Wrap detected.

Bit 2 = 1, Invalid character detected.

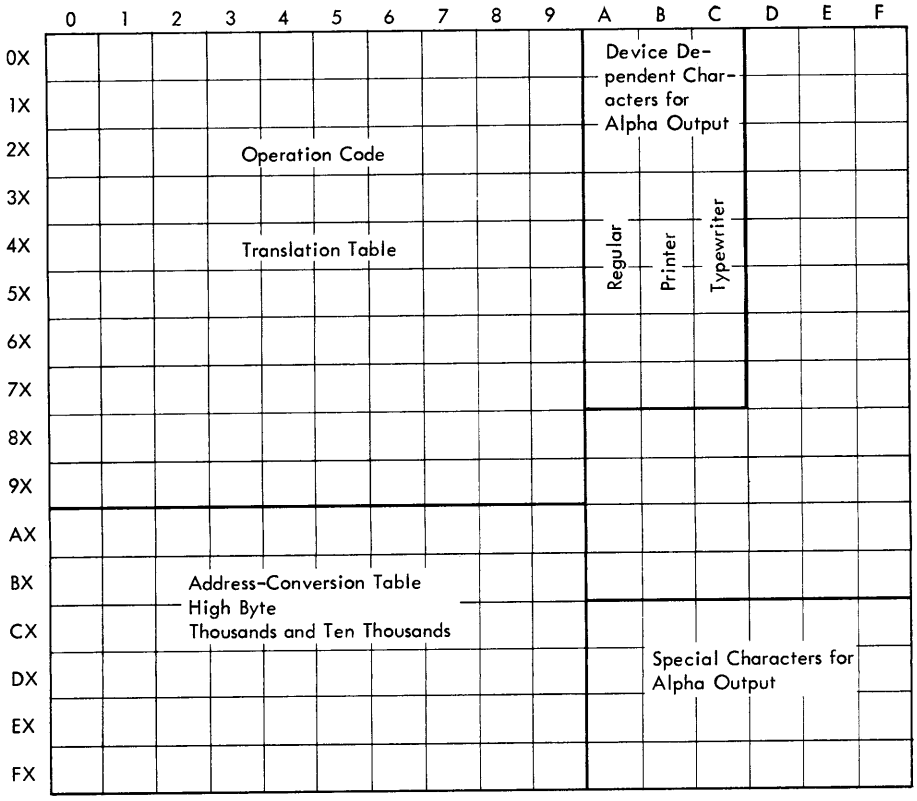
Bit 3 = 1, count equal zero.

I-CYCLES

- The 1620 operation codes are translated to bit significant by table lookup.
- Flag analysis, address conversion, and sign and field length routines are performed for most instructions.
- Indexing and or Indirect addressing is performed if designated.

The translation of the 1620 operation code is done by table lookup. The two digit 1620 operation code is used as the low byte of an address that reads out an operation code from a table located in the MPX Auxiliary Storage area (Figure 4-48). The 1620 operation codes are translated to obtain codes which may be more fully used for branching through the microprogram.

Once the 1620 operation code is translated and placed in the operation register, various paths are taken. The different types of operations require various methods of setting up data and registers for accomplishing the operation designated (Figure 4-49). See I-Cycle Flowcharts in IBM 1620 Compatibility Feature Diagram Manual, Form Y25-3478, for I-Cycle details.



Other Information Contained in MPX Auxiliary Storage Area is:

Co-ordinates	Contents
8A, 8B	Instruction Counter Saved by the BT Instruction
9A, 9B	Instruction Counter Saved by the Save Operation
9C	High 2 Digits of Highest Flag Byte Address
9D	Low 2 Digits of Highest Flag-Byte Address
9E	High-Order Byte of Field Length for TF or BT Instructions Only
AB, AC	Size of Machine in Hexadecimal

Figure 4-48. MPX Auxiliary Storage

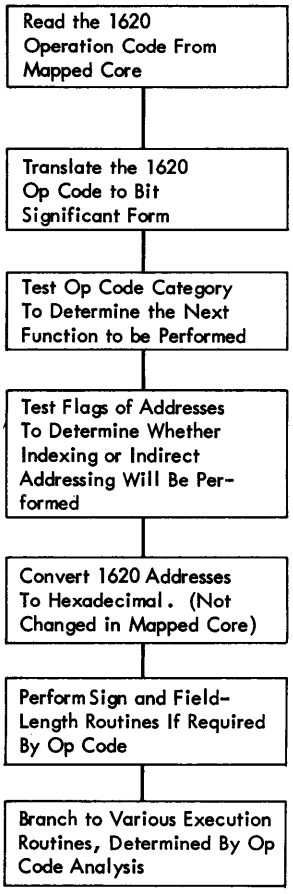


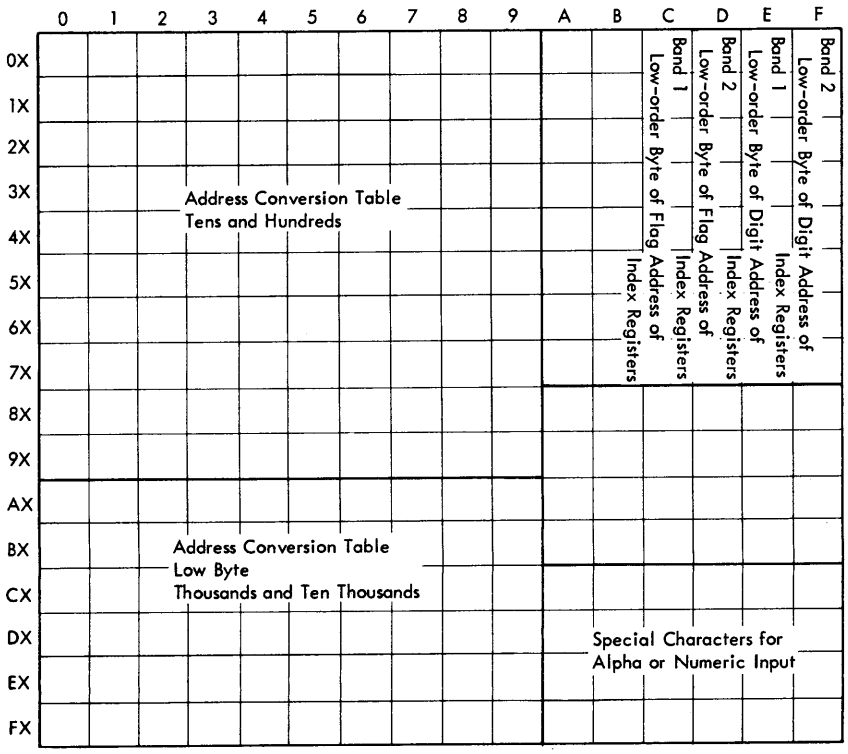
Figure 4-49. I-Cycle Objective Flow

Flag analysis is performed for Q and P addresses for most of the 1620 operations. There are certain operations however that do not require this information for both addresses.

There are two Instruction counters which are used primarily during I cycles; one contains the hexadecimal digit address of the operation code portion of the instruction to be executed. This counter is updated to address the lowest two Q-address digits once the 1620 operation code has been read out of storage. The other instruction counter is called the Flag Instruction counter and contains the address of the byte in mapped flag storage which has the flags for the P_2 and P_3 digits of the instruction that is to be executed.

The flags are tested (normally for the Q-address first) to determine if indexing and or indirect addressing is to be done. If indexing is designated and an index band has been selected, the contents of the index register called for is decimally added to the address being operated on. The result becomes the new address and is now translated to its hexadecimal equivalent by table lookup (Figures 4-48 and 4-50). This hexadecimal address may now be used to read out an indirect address if indirect addressing was called for during the flag analysis. The new address (which will be a 5 digit 1620 address) is tested for indexing, and indirect addressing, and operated on. This sequence continues until an address is obtained which has no indirect or indexing flags.

The sign and field length routine is performed for the instructions that require this information. The signs of both fields are found if required, and are stored as status bits for use in the execution phase. The field length is determined for one or both of the fields and is saved for use in the execution phase. The field length is found by computing the flag address of the low order digit of a field and searching right to left for the field limiting flag. A digit counter keeps count of the number of flag positions scanned before a limiting flag is found. This counter then contains the correct field length and is located in LS Auxiliary storage for use during instruction execution. For an example of an instruction processed in 1620 compatibility mode, see Figure 4-51.



Other Information Contained in LS Auxiliary Storage Area is:

Co-ordinates	Contents
8A, 8B	Instruction Counter
8C, 8D	Flag Instruction Counter
8E	Indirect and Index Indicators
8F	Field-length Low-order Byte for TF or BT (High Byte in 9E MPX)
9A	Arithmetic Indicators
9B	Console Switches
9C, 9D	Flag Base Address
9F	Flag Bit Mask for Sign and Field Length
AA, BB	Highest Digit Address

Figure 4-50. LS Auxiliary Storage

1620 Representation of a Transmit Field Instruction

1620 Storage Representation (Assume 1620 Specified as 20K)

1620 Address	200	201	202	203	204	205	206	207	208	209	210	211
Transmit Field Instruction	2	6	1	4	2	5	3	1	3	$\overline{2}$	6	8
	Op Code		P-Field Address					Q-Field Address				

Flag at 209 indicates a call for index register 2. Assume Band 1 selected.

1620 Address	14246	14247	14248	14249	14250	14251	14252	14253
P-Data Field	$\overline{0}$	2	4	9	2	6	8	7

1620 Address	19923	19924	19925	19926	19927	19928
Q-Data Field	$\overline{6}$	3	4	8	5	$\overline{9}$

This is Q-Field address after indexing.

1620 Address	310	311	312	313	314
Index Register 2 Band 1	0	6	6	6	0

1620 Address	14246	14247	14248	14249	14250	14251	14252	14253
P-Data Field After Execution	$\overline{0}$	2	$\overline{6}$	3	4	8	5	$\overline{9}$

Figure 4-51. Instruction Example (Part 1 of 2)

Transmit Field Instruction — Represented in 2030 Storage

Digit Address Hexadecimal

0E64	0E65	0E66	0E67	0E68	0E69
2 6	1 4	2 5	3 1	3 2	6 8

Op Code — P-Address — Q-Address

Instruction Digit Flag Address

3619		361A	
0000	0000	0100	0000

Flag From Q9 Digit

Transmit Field Procedure

1. Do table lookup using 1620 Op code as low-order MPX address, read out translated OP code.

   1620 op code 26
   Translated op code 30

2. Scan Q-field flag area for flags. Flag is found for Q9 digit; this signifies index register 2 selected.

   Hex flag addresses

3619		361A	
00	00000	0	0100 0000

   P-Flags — Q-Flags

3. Test 8E LS to determine what index band has been selected. A branch and select instruction given before this instruction selects a band, and this information is placed in 8E LS.

   8E LS 1100 0100

   Index band Indicators — Indirect Addresssing Indicators

   0 = Band 0 — 0 = Indirect
   C = Band 1 — 4 = Indirect
   D = Band 2

4. Decimally add the Q-field address and the contents of the index register, taking into account the sign, if any, of the index register. Band 1 had been selected. This addition does not affect the Q-field address of the instruction in storage.

	0E9B	0E9C	0E9D
Index 2 Band 1	0 6	6 6	0

Q-field Address	1	3	2	6	8
	1	9	9	2	8

   Effective Q-field address (1620)

   19928

5. Convert Q-field address (effective) to hexadecimal. This address points to units digit.

   19928 effective 1620 Q-field address
   34EC effective 2030 Q-field address

6. Test the P-field address for flags, none are found, convert P-field address to hexadecimal.

   14253 1620 P-field address
   29D6 2030 P-field address

Transmit Field Procedure

7. Compute Q-field flag address. Through computation the flag position for units digit is found to be bit 0 of 3FBB. Read out next lower flag byte for remaining Q-flag positions.

   Q-field flag byte which contains flag position for units digit.

   3FBB

   1000 0000

   Flag for units digit (Sign) — 3FBA

   0001 0000

   Flag for field limitation

8. Retain flag for units digit of Q-field as sign of Q-field.

   Q-Units digit flag → S2

9. Scan Q-field flag area for field limiting flag, increment digit counter for every flag position read.

3FBA	3FBB
00010000	10000000

   Q-field flag area scan

   8FLS is Digit counter = 6

10. Compute P-field flag address. Computation indicates units - digit flag position is located in bit 5 of 3CF5. This is all the information about the P-field neede for transmit field.

    P-field flag byte containing units digit flag.

    3CF5
    0 7
    00000000

    P-units digit flag position.

11. Set the flag of the P-field units digit equal to the flag of the Q-field units digit (Q field sign). Clear P-flag area according to Q-field length. Set a flag in the high-order P-field flag area according to the Q-field length.

    3CF5
    0 7
    00000100

    Sign of Q-field

    3CF5
    0 7
    00000100

    Clear flags

    3CF5
    0 7
    10000100

    High-order new P-field

12. Transmit Q-field digits to P-field until digit counter steps to zero. Q-field remains un changed.

effective Q-field	34E9	34EA	34EB	34EC
	X 6	3 4	8 5	9 X

P-field	6 3	4 8	5 9
	29D4	29D5	29D6

Figure 4-51. Instruction Example (Part 2 of 2)

DISK FILE FORMATS FOR THE 1620 COMPATIBILITY FEATURE

- The IBM 1311 Disk Storage Drive is emulated by the IBM 2311 Disk Storage Drive.
- The 2311 track format is changed to give maximum performance for the 1620 Compatibility Feature.

The format of the 2311 disk packs are changed by a disk file format program which is supplied with the 1620 compatibility feature. Each 1311 track is mapped onto the corresponding track of the corresponding cylinder on the 2311. The alternating arrangement of the records on the track and the addition of dummy records provides for maximum performance when doing multiple sector disk operations (Figure 4-52).

The 1311 cylinders 00 thru 99 are mapped onto cylinders 100-199 on the 2311. Cylinders 00 thru 98 of the 2311 are not restricted as to contents and are not used by the 1620 compatibility feature. Cylinder 99 track 0 of the 2311 contains the IOCP program which is also contained in main storage. Track 1 contains the disk track IOCP for a 16K 2030. This disk track IOCP is called into main storage whenever a disk track operation is called for (Figure 4-53). Upon completion of the disk track operation, the IOCP program is called back into main storage from Track 0 of cylinder 99 and control is given to the microprogram. Track 2 and 3 of cylinder 99 contain the disk track programs for 32K and 64K sizes of 2030. One of these routines is called in during the initialization period and remains in main storage until cleared out or written over.

The sector records (20 per track) are 127 bytes in length and contain the 1311 sector address, and the 100 digit data record with associated flag bits. The dummy records are 101 bytes in length and contain all F's for a 1311 sector address (invalid) and a data field of all zero's (Figure 4-54). The dummy records are read, only in track mode, but are ignored.

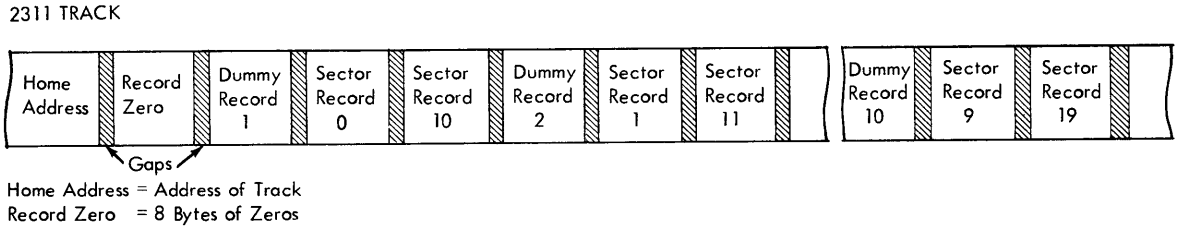


Figure 4-52. 2311 Track Format for 1620 Compatibility Feature

Map of 16K 2030 for 20K 1620

Address	Use
0000 008B	Hardware Locations
008C 0B03	Overlay Area *
0B04 0DDF	Non-Disk Track and Disk Track IOCP
0DE0 0DFF	1620 Initilization Instructions
0E00 350F	1620 Digits
3510 3515	1620 Digit Ending Characters
3516 351F	Not Used
3520 355F	Auxiliary Storage Restore
3560 35F3	Not Used
35F4 35FF	Initilization Instruction Flags
3600 3FC3	1620 Digit Flags
3FC4 3FFF	Not Used

*Overlay area for any instruction other than disk full track

008C 09DB	Non-Disk Track IOCP
09DC 0B03	Maintenance Area

*Overlay area for disk full track

008C 0A5F	Disk Track IOCP
0A60 0B0C	Maintenance Area

Map of 32K 2030 for 40K 1620

Address	Use
0000 008B	Hardware Locations
008C 09DB	Non-Disk Track IOCP
09DC 0B03	Maintenance Area
0B04 0DDF	Non-Disk Track and Disk Track IOCP
0DE0 0DFF	1620 Initilization Instructions
0E00 5C1F	1620 Digits
5C20 5C25	1620 Digit Ending Characters
5C26 5FF3	Not Used
5FF4 5FFF	Initilization Instruction Flags
6000 7387	1620 Digit Flags
7388 739F	Not Used
73A0 7D5F	Disk Track IOCP
7D60 7E0D	Maintenance Area
7E0E 7EFF	Not Used

Map of 64K 2030 for 60K 1620

Address	Use
0000 008B	Hardware Locations
008C 09DB	Non-Disk Track IOCP
09DC 0B03	Maintenance Area
0B04 0DDF	Non-Disk Track and Disk Track IOCP
0DE0 0DFF	1620 Initilization Instructions
0E00 832F	1620 Digits
8330 8335	1620 Digit Ending Characters
8336 8FF3	Not Used
8FF4 8FFF	Initilization Instruction Flags
9000 AD4B	1620 Digit Flags
AD4C AFFF	Not Used
B000 B9BF	Disk Track IOCP
B9C0 BA6D	Maintenance Area
BA6E FFFF	Not Used

● Figure 4-53. Core Storage Maps

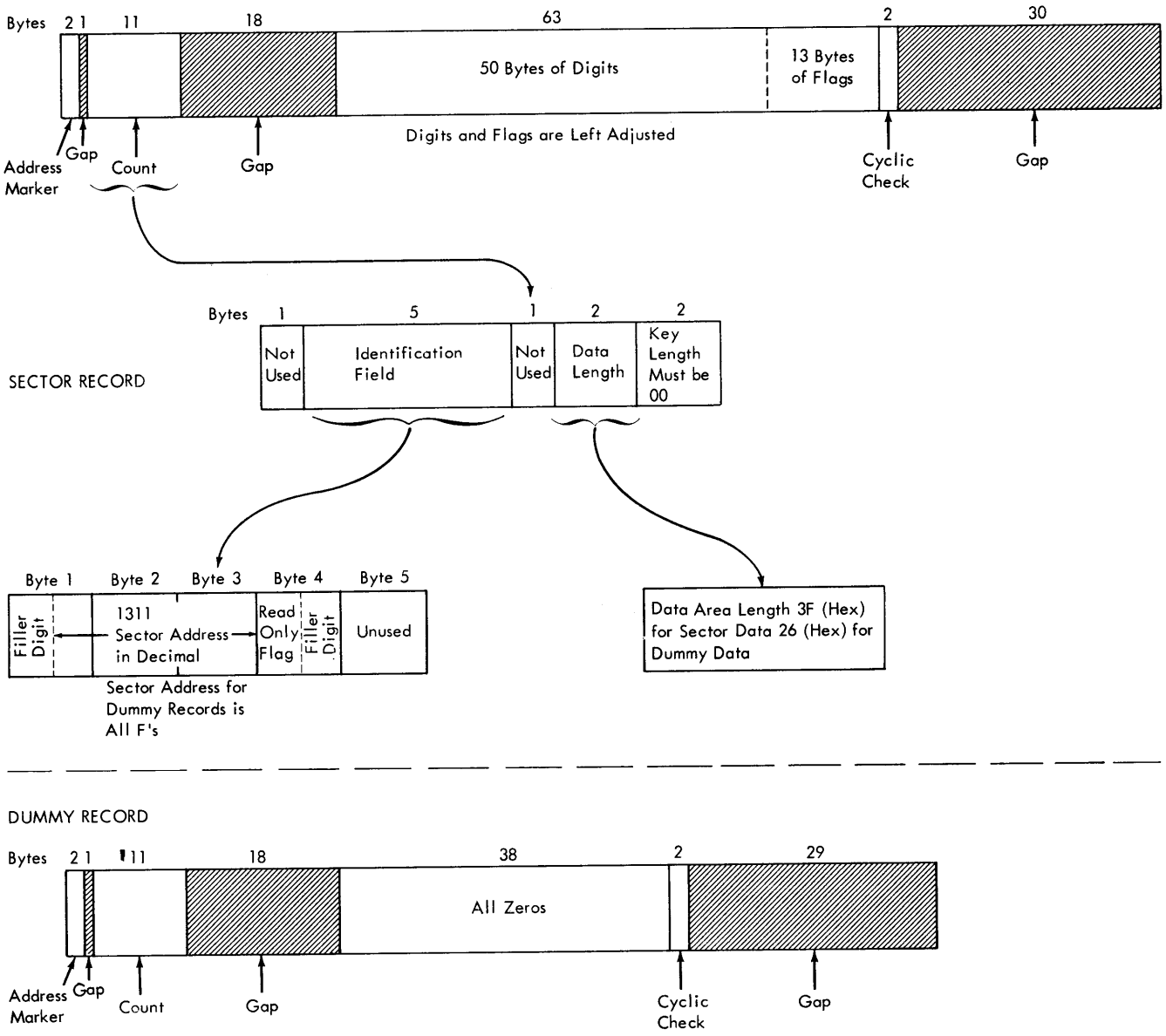


Figure 4-54. Sector and Dummy Records

DISK OPERATIONS

- Disk Operations are executed by a combination of compatibility microprogramming and the regular IOCP or disk track IOCP.
- All Seek operations are direct.
- The 1311 Write Address Switch function is performed by the 0 bit position in 9B LS Auxiliary storage.

Sector mode operations are executed by a combination of compatibility microprogramming and the regular IOCP. The microprogram controls the data flow between the buffer and mapped 1620 storage and checks the data for group marks if Wrong Length Record Check (WLRC) is specified. The microprogram also transfers the flags, properly aligning them according to the 1620 address specified. The IOCP controls the transfer of data between the disk and the buffer.

The microprogramming functions of data transfer and checking are performed while the disk is skipping over the two records adjacent to the sector record being operated on. Once these microprogramming functions have been performed, the next sequential sector record will be in position to be read or written.

A sector mode write instruction is performed only if the Write Address switch (bit 0 of 9B LS) is off. If it is on and a write sector mode is called for, the system stops and displays the stop code B0 in the Main Storage Address Register. Pressing the Start key causes the system to resume the program at the next sequential instruction.

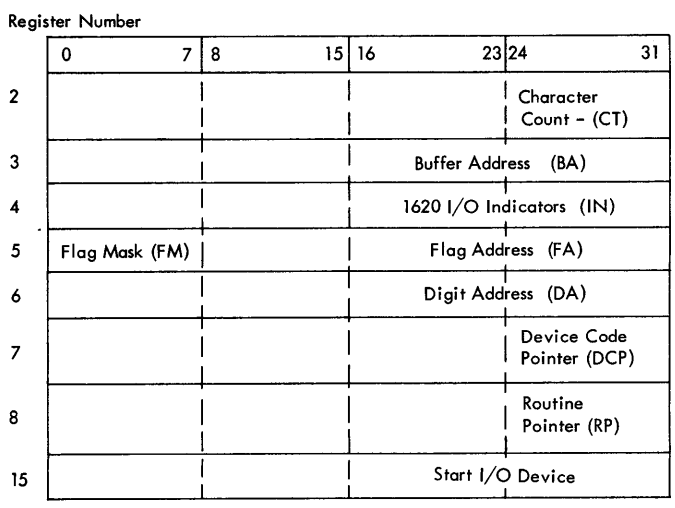
Track mode operations are executed by a combination of the compatibility microprogram and the disk track IOCP. In any read full track operation, the entire track involved is read into a buffer by the disk track IOCP. The microprogram then transfers the 20 sectors to mapped 1620 storage, checking for group marks for WLRC instructions, and aligning flags.

A track mode write operation is executed only when the Write Address switch is on (Bit 0 of 9B LS). If it is off and a write full track is called for, the system stops, displaying the stop code C0 in the main storage address register. Pressing the Start key causes the system to resume the program at the next sequential instruction.

NON DISK I/O OPERATIONS

- All data is handled by the compatibility microprogram and the IOCP.

During I phase of a 1620 I/O operation, the compatibility microprogram scans the operation fields of the instruction and places the data in the general registers (Figure 4-55). This information will be used by the IOCP, and the move and translate microprogram routines for proper data handling, data placement, device recognition, and error detection. After the compatibility microprogram has set up the general registers, external interrupt is masked off and a switch is made to 2030 mode for entering the IOCP.



CT = The count byte contains the number of data characters minus 1 to be processed by the move and translate microprogram. Set up by the IOCP.

BA = Buffer address set up by the IOCP for use by the move and translate microprogram.

IN = I/O indicators set by the IOCP for testing by the 1620 microprogram.

FM,FA = The flag address is the address of the flag byte associated with the current digit address. The flag mask, in bits 0, 6, 7 of the mask byte indicates the bit of the flag byte at which the flag corresponding to the current digit address is found.

DA = The digit address is the current digit address in 1620 mapped core.

DCP = The device code pointer is set by the 1620 microprogram before entering the IOCP.

RP = The routine pointer is set by the 1620 microprogram; indicates to the IOCP the proper routine to be performed as stated by the operation code.

Figure 4-55. General Register Assignments

The IOCP scans the contents of the general registers and forms a single CCW or chain of CCW's for the particular write or read operation to be performed.

If the operation is a write operation, the IOCP will transfer control back to the compatibility microprogram for the move and translate routine which fills the buffer. Control is now given to the IOCP and the start I/O command is issued. The 2030 enters wait state until a channel end interrupt is received, the IOCP will then branch back to 1620 compatibility mode I cycles. If the operation is a read, the IOCP will issue the start I/O command and enter wait state until channel end interrupt. When the I/O device has completed its transmission to the buffer, the IOCP gives control back to the compatibility microprogram. The proper move and translate routine is entered to read the information from the buffer into the mapped area of storage.

MOVE AND TRANSLATE ROUTINES

- The Move and Translate routines are part of 1620 compatibility microprogram.
- Entered from the IOCP by the execution of the special Operation Code OC.

The Move and Translate routines move data to and from the buffer, translating the data to the proper format dependant on the device used and the mode of operation. There are four major routines: Alpha or Numeric input, Alpha out, Numeric out and Binary input.

If the operation is an Alpha input, each character in the buffer is tested to determine if it is a special character. If the character is special, a table lookup is performed and the character found in the table is read and placed directly to the designated mapped area of storage. If the character is not special, the translate routine will translate the character to the proper 1620 coded alpha character and place it into the designated mapped area of storage.

For Numeric input, each character in the buffer is tested for special character format. If it is special, a table lookup is done and the digit portion of the character in the table is placed in mapped storage. A flag will be placed in the proper flag area if called for by the flag portion of the byte read out from the buffer. If the character were not special, the low half of the buffer byte would be placed directly into mapped storage and a flag placed in the proper mapped flag area, if designated. For either special or regular characters, the proper flag position in mapped storage will be reset if a flag is not called for by the byte in the buffer.

For Alpha output, the characters to be written out are tested to determine if they are special. If yes, the special character alpha output table in the MPX Auxiliary Storage is scanned. If the character is not present, the device dependent table is referred to and the correct EBCDIC character is written into the buffer. If the character is not found in either table, a blank is written into the buffer and the invalid character condition code is set in BB 2030 LS. For non special characters, direct translation to EBCDIC is performed and the translated character is written into the buffer.

Binary input from the Paper Tape Reader is handled almost entirely by the IOCP. The IOCP reads in the paper tape character, checks for correct parity, translates the character to 1620 binary format and places the character in the buffer. The Move and Translate routine moves the 1620 binary character from the buffer to the proper digit area in mapped core (Figure 4-56).

1620 Paper Tape Character		Buffer Bit Positions	Bits of Translated Paper Tape Character to 1620 Mapped Core
EOL	IOCP reads character, checks for parity, and translates to binary 1620 character	0*	0
X		1	X
O		2	O
C		3	8
8		4	0
4		5	4
2		6	2
1		7	1

*Set to 1 if parity bad
Set to 0 if parity good

Figure 4-56. Binary Paper Tape Input

If the IOCP detects a parity error, bit zero of the byte in error is set to one in the buffer. The move and translate routine checks the zero bits of the characters in the buffer before moving them to mapped storage. If an error is found the zero bit of the error byte is reset, the error byte is moved to mapped storage, the invalid character condition code is set in BB 2030 LS, and control is given back to the IOCP. The IOCP tests the condition code and sets the read/check indicator on. Control is returned to the move and translate routine which continues to transmit the buffer characters to mapped storage.

The Move and Translate routines are entered from the IOCP by the execution of the instruction 0CXX. The value of XX determines the major routine to be entered.

XX	Function
00	Alpha output
04	Alpha input
0C	Binary input
20	Numeric output
24	Numeric input
64	Numeric input (Flagged Character on Type)

All data, except binary, is translated between EBCDIC in the buffer and 1620 data code (separate digit and flag areas) in mapped 1620 storage.

ERROR-STOPS AND ERROR HANDLING

- All programming and operational error stops are displayed in the MSAR as a stop code.
- Other errors and conditions are displayed (Figure 4-57) on the 2030 console.

The majority of the programming error stops should be handled by reloading the IOCP and the program being run to eliminate the possibility the program had been loaded incorrectly on the initial run. All console indications should be noted before and after program is rerun, to determine if the identical stop occurs. Refer to Figure 4-58 for stop codes and stop conditions. Stops other than error stops are mainly for operational convenience.

CARD READ OPERATION

- Provisional Feed Feature must be installed on the 2821 for proper operation of card read in 1620 compatibility mode.

The Provisional Feed circuit provides an automatic feed cycle and a select to the normal pocket after a 6 millisecond timeout. The 6 millisecond timeout starts when the read-command-data transfer begins.

The IOCP issues a stacker select command during this 6 millisecond timeout to allow an immediate card feed and normal pocket select. When the last card is read and the channel-end is received, the IOCP tests the unit exception bit of the CSW and if on the IOCP sets the last card indicator (bit 23 of general register 4). The IOCP, after every stacker select, returns control to 1620 compatibility I cycles.

The last card indication being set as the last card is read, eliminates the need for an extra read command to be issued to provide this information.

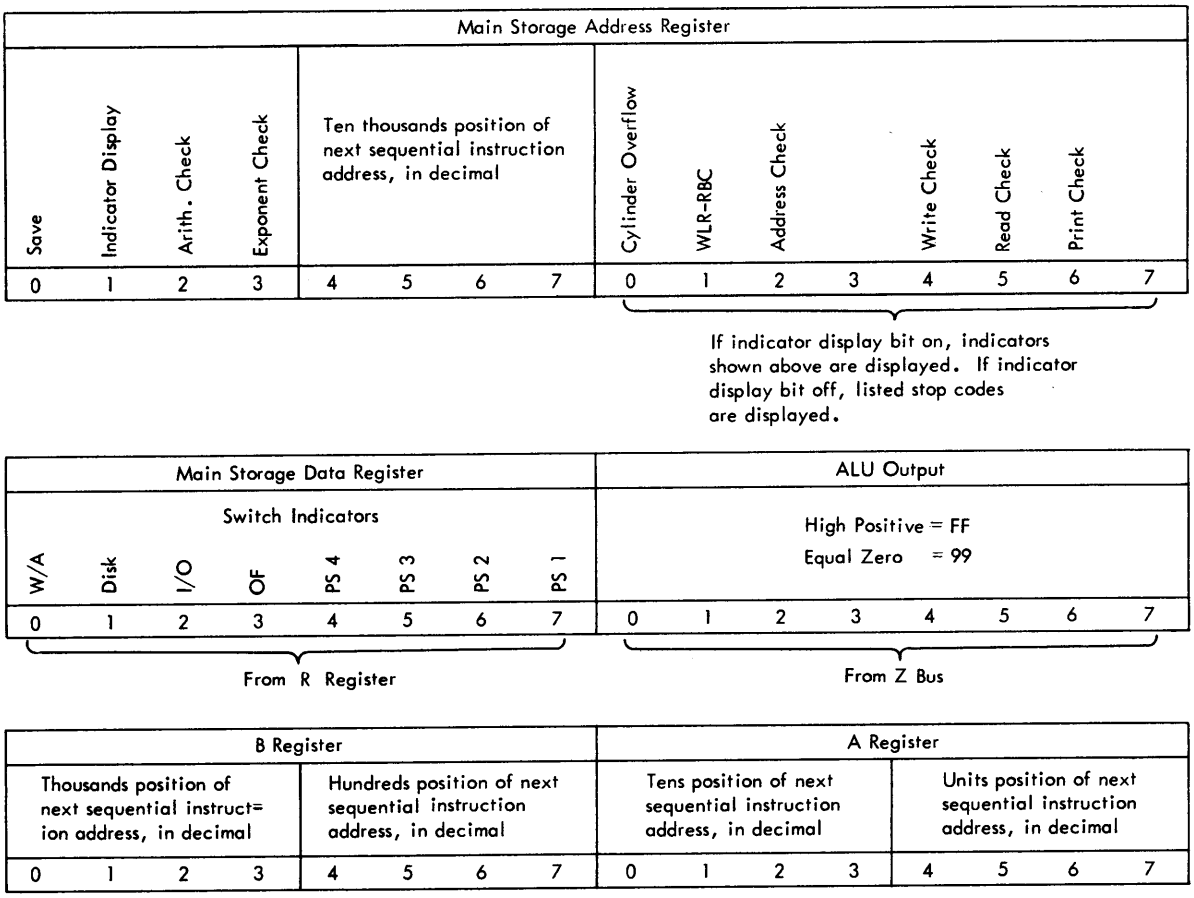


Figure 4-57. Console Displays

MSAR Tens and Units Positions Will Display Stop Codes If Indicator Display Bit Is Off

Stop Code	Condition	Stop Code	Condition
01	Correct F or G switch operation executed	AO	Specified 1620 storage address is odd
02	Incorrect F or G switch operation executed (one switch not neutral)	AF	Branch to an odd location attempted (1620 address)
03	Entry into 1620 mode executed	B0	Write-Address switch is on (Bit 0 9B LS)
07	Console stop key pressed, or rate switch set to INSTR STEP position	B1	Storage wrap detected, right to left field operation
1F	Exponent flag error (no exponent limiting flag)	BF	Invalid decimal digit used in exponent
20	Invalid typewriter control	C0	Write Address switch is off (Bit 0 9B LS)
21	No device address	C1	Invalid decimal digit used in multiplicand (P-field)
22	Machine check	CF	Addressed location not available in 2030 storage
2F	Mantissa length error (not equal, too long)	D0	Disk-control field is at an odd address
30	I/O Release (attention bit on)	D1	Invalid decimal digit used in multiplier (Q-field)
31	Invalid decimal data used in DTO and OTD or Wrap in DTO	DF	Address wrap attempted in either direction, or invalid address
33	Program check	E0	I/O operation code incorrect
3F	Floating-point address check	E1	Product Area Wrap
41	Quotient wrap	EF	Invalid op code used
44	Invalid input command	F0	Reader check stop
48	Halt instruction executed	F1	Right to left wipe-out attempted, or an even alpha address used in TNF
4F	Storage wrap in transmit field	F3	Typewriter Read
51	Invalid decimal digit used as data for add, subtract, or compare.	F6	Card Punch
55	Incorrect I/O unit address	F7	Card Read
5F	Field length greater than 255 characters in TFL, BTFL, BTA, or BTAM	F8	Printer
61	Divide P-address is not less than 99	F9	Disk Drive 0 } Device Not Available
6F	No BT (or BTM) before a BB instruction	FA	Disk Drive 1
71	Storage Wrap detected in sign and field-length routine	FB	Disk Drive 2
77	Error in overlay from disk	FC	Disk Drive 3
7F	Storage Wrap on index execute	FD	Disk Track read or write error
81	Storage Wrap detected in a logic operation	FE	Left to right wipe-out executed
88	Paper Tape Overrun		
8F	Invalid Band 0 selection		
90	Specified sector count is invalid		
91	Field longer than 255 digits detected in sign and field-length routine		
9F	Invalid modifier in Branch and Select instruction		

● Figure 4-58. Stop Condition Codes

CHAPTER 5. POWER SUPPLIES

POWER-ON SEQUENCE (STEPPER SWITCH)

- Information on I/O sequencing for processing units without the stepper switch can be found in the System/360 Model 30 Functional Maintenance Diagrams, pages 5-14A and 5-14B.
- Before power can be applied to the IBM 2030 processing unit, the over-voltage, over-current, and over-temperature conditions must be normal and the high frequency inverter/converter oscillator must be running.

Pressing the Power-on key picks RY3 to initiate the power-on sequence (Figures 5-1, 5-2, and 5-3). Relay RY3 in turn picks contactor K2 applying power to the blower motors and the converter-inverter.

The inverter/converter output brings up all dc voltages except-30 volts (+40 volts on M2 machines) and the special -3 sequence voltage for the storage-protect feature. With +6 volts up, a point of RY3 picks RY4, the 6 volt sense relay. The pick of RY4 causes contactor K3 to pick applying -30 volts and the -3 volt sequence to the circuits. A point of RY4 (N/O) applies the control voltage to the Stepping switch that will power-on each I/O unit in sequence. The Stepping switch is advanced from the home position to the first I/O control position (position 1) when RY16 picks through the N/C cam contact (Figure 5-2). With the step to position 1, the cam turns and allows the N/O cam contact to close. At the same time, the Stepping switch (decks B and C) provides a circuit to pick the power-on relay(s) in the first I/O unit. This in turn closes the N/O points designated power-on-signal in (Figure 5-2).

Now, a circuit is made from the CPU control voltage, through the power-on-signal N/O points, positon 1 of the Stepping switch (deck A), and the cam contact N/O points, to again pick RY16. This advances the stepping switch to position 2 and the cycle is repeated for the next I/O unit.

When the Stepping switch has advanced through all wired I/O units, and all I/O units have power on, the last unit to power-on advances the switch one position to the dummy plug position. This picks relay TDR1 to start a two-second time delay before removing system reset and turning on the system Power-on light. The Stepping switch remains in this position until a power-off sequence is initiated.

At the end of the two second time delay, the -3 volts is removed from the reset line by the pick of TDR1 and relay RY5 picks to indicate that the processor is ready.

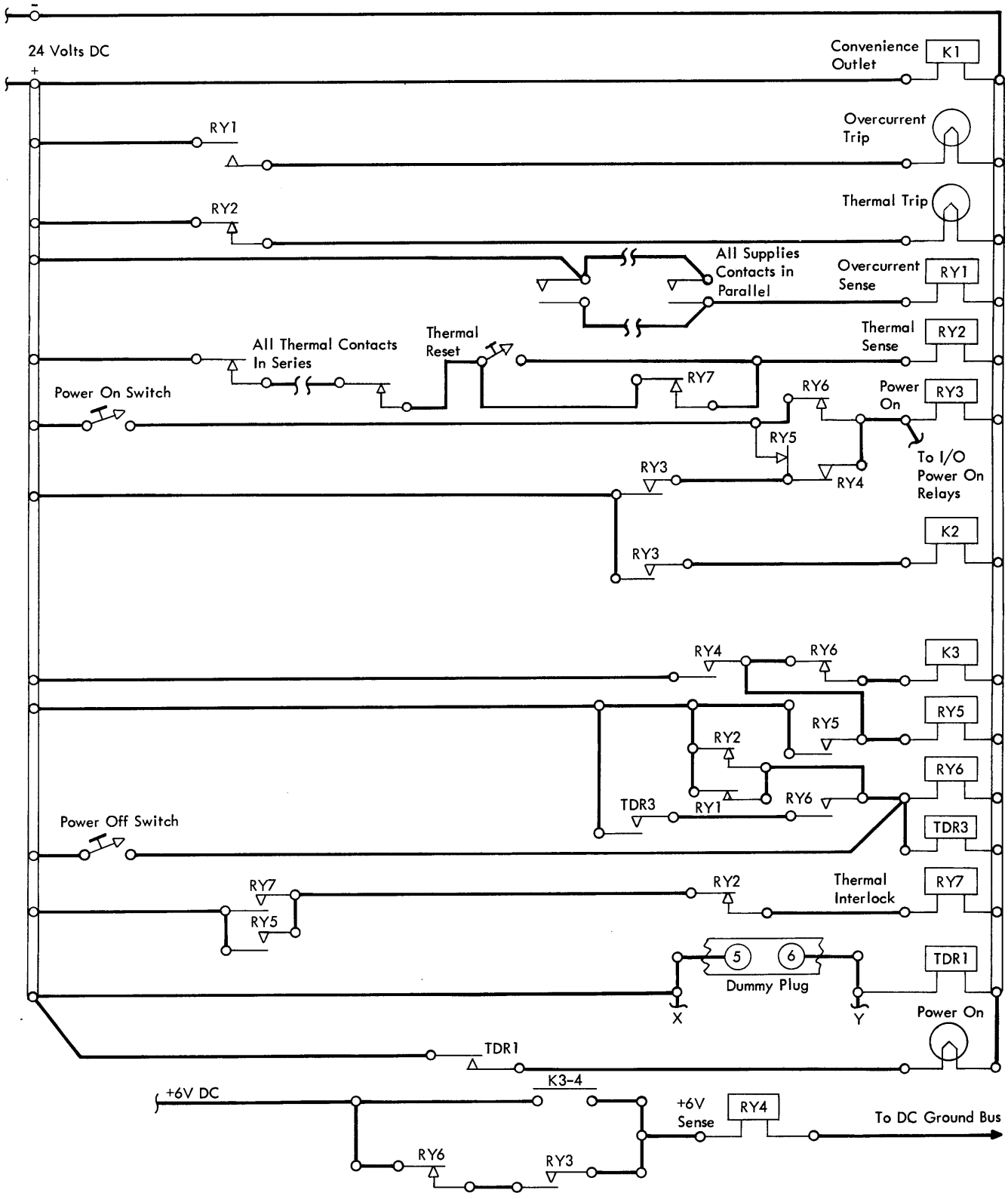


Figure 5-1. Power On-Off Control

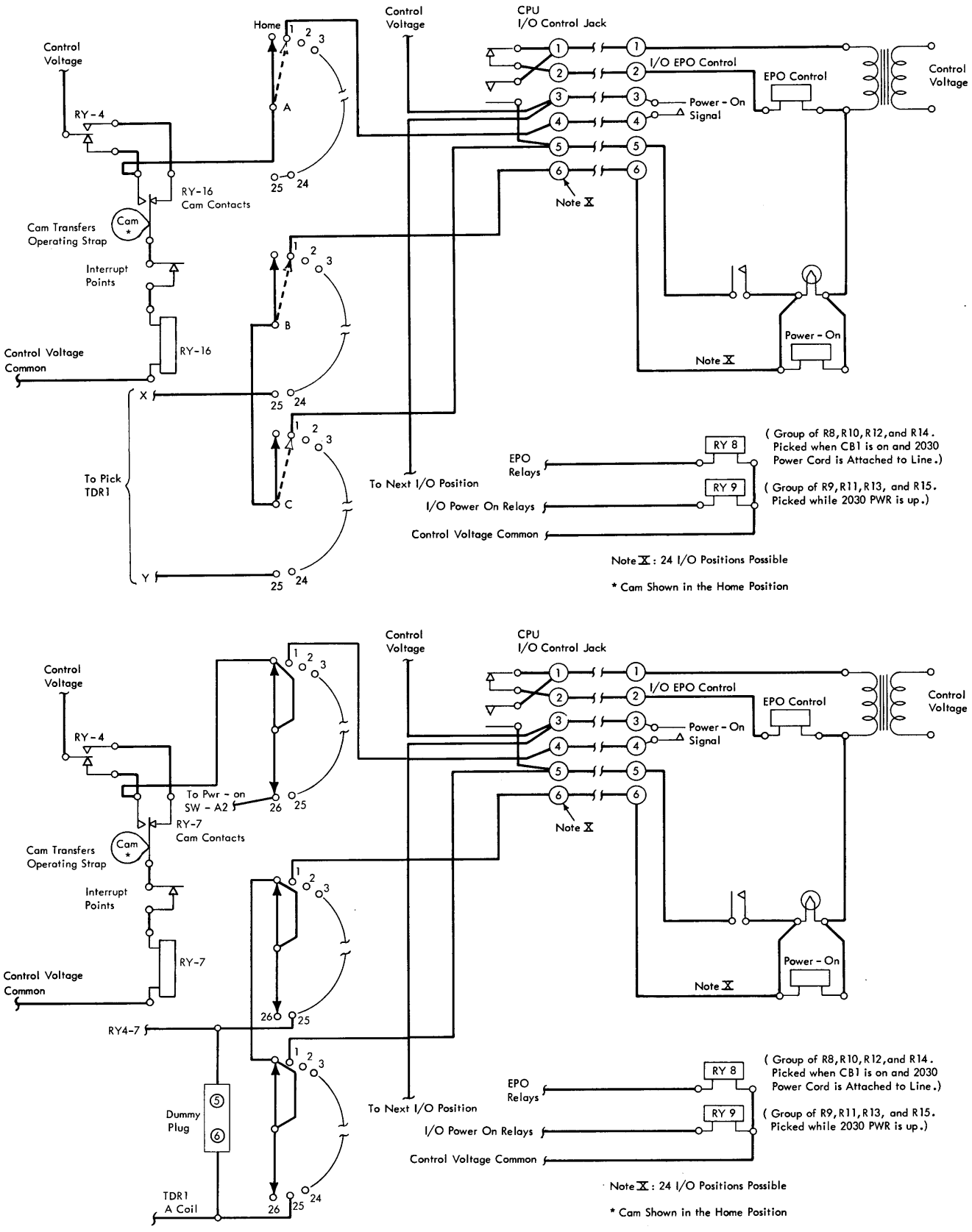
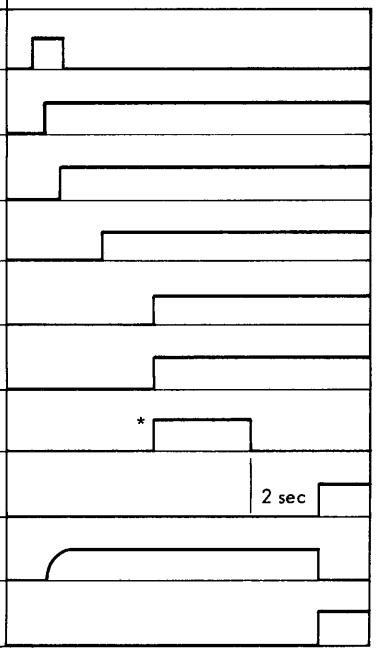


Figure 5-2. I/O Sequencing (Upper--Inv/Con, Lower--Mid-Pac)

No	Name	Logic
1	Power On Sw	YZ041
2	RY - 3	041
3	K - 2	041
4	RY - 4	041
5	RY - 5	041
6	K - 3	041
7	Ry - 16	041
8	TDR - 1	041
9	System Reset	042
10	Power On Light	041



* Time Varies with Number of I/O Units

Figure 5-3. Power On Sequence

No	Name	Logic
1	Power Off Sw	041
2	RY - 6	041
3	TDR - 3	041
4	K - 3	041
5	RY - 4	041
6	RY - 16	041
7	RY - 3	041
8	K - 2	041

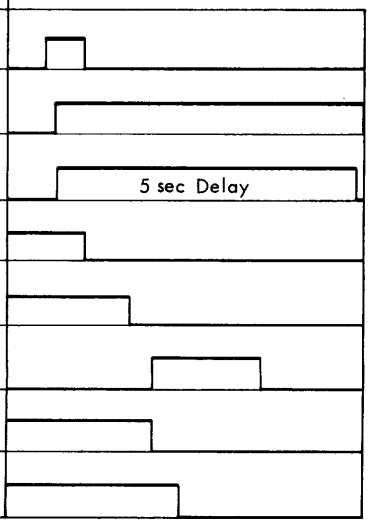


Figure 5-4. Power Off Sequence

POWER OFF SEQUENCE

- When system power is turned off, RY4 performs two major functions:

  1. It removes the -30 volts from the storage unit(s).

  2. It advances the stepping switch to the home position.

The Power-off key, besides picking RY6 to start the power-off sequence, starts a five second time delay. This delay prevents another power-on cycle for at least five seconds. Without the delay, damage to the Inverter/Converter power unit can occur if power-on is pressed immediately after power-off.

Pressing the Power-off key on the console panel sequences power down in the CPU and the power-on control relays (Figures 5-1 and 5-4). Power to all I/O units is dropped simultaneously.

All data in core storage remains unchanged. If the allow-write latch is on at the time the Power-off key is pressed, a memory write is forced and the contents of the R-Register is inhibited into core.

EMERGENCY POWER-OFF

The EPO switch removes all power except the 24 volt control power from the CPU and every I/O control unit attached to a channel simultaneously, and without sequencing.

An emergency power-off can cause the data in core storage to be lost.

OVERVOLTAGE OR OVERCURRENT SENSE

Either of these conditions remove primary power from the inverter/converter and the blowers by initiating a normal power-off sequence. An indicator lights on the power supply module affected, or on the relay and connector panel.

Power cannot be restored until the cause of the overload is corrected and the reset pushbutton, located on the high-frequency inverter, is pressed.

OVER TEMPERATURE SENSE

The thermal switches, located throughout the CPU, remove primary power from the inverter/converter and the blowers by a normal power-off sequence.

The thermal trip light on the relay and connector panel remains on, and RY7 the thermal interlock relay re-picks, even though the thermal reset switch is pressed as long as the over-temperature condition exists. Power cannot be restored while any thermal switch is open. When the over-temperature condition is corrected, power can be restored only after the thermal reset switch is pressed to pick RY2 and to drop RY7.

POWER DISTRIBUTION

Figure 5-5 shows the power distribution in the two existing models of the 2030 CPU.

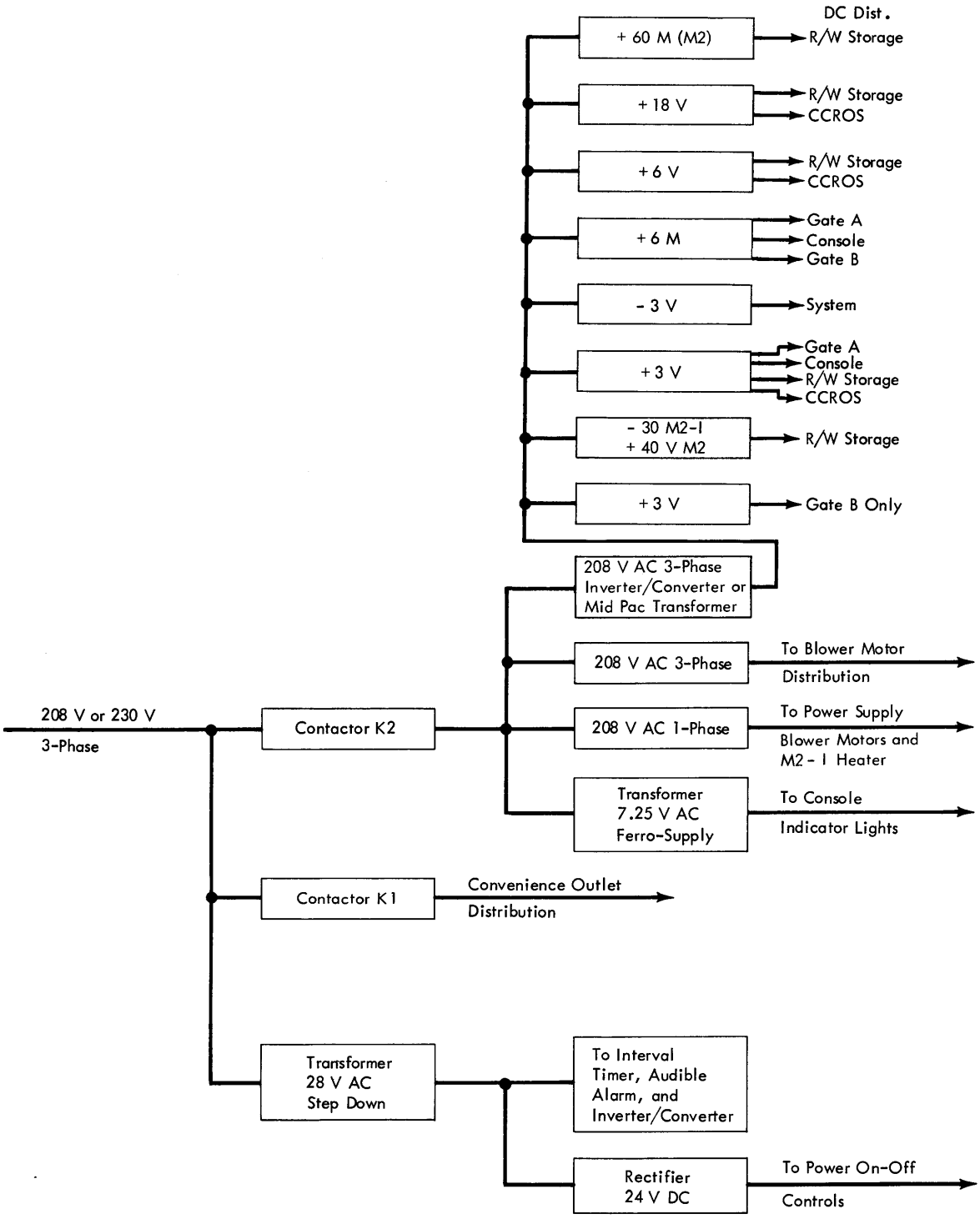


Figure 5-5. Power Distribution

POWER-ON SEQUENCE (MID-PAC)

Before power can be applied to the 2030 processing unit, the Thermal Trip Sense relay (RY1) must be energized, and the over-current sense circuitry must be normal.

Pressing the Power-on switch picks relay 3 through position 26A of RY-7 to initiate the power-on sequence (Figure 5-6 and 5-7). Voltage is fed to the Power-On key thru RY-7-26A to pick RY-3; once RY-3 is picked and latched up, RY-7 is free to move when it receives voltage via RY-4. This prevents a Power-On before RY-7 has returned to its home position after a Power-Off has been initiated.

The mid-pac blowers will run after CB-12 trips. AC power does not drop, only the dc levels do. Relay 3 picks contactor K2, which causes power to be supplied to the blower motors and the mid-pac transformer. Relay 4 (+6v Sense) is picked when PS6 supplies a +6 volts. Relay 4 and relay 3 combine to pick contactor K3 (-30v Sequence on). K3 picks the I/O power-on control relays 9, 11, 13, and 15. This prepares the sequencing of the individual I/O groups.

When relay 4 is picked, voltage is supplied to pick the stepper control relay 7 (Figure 5-2). Relay 7 causes the stepper switch to advance to the first contact position. A cam activates the stepper contacts and causes relay 7 to drop and await a power-on signal from the I/O group being sequenced. Relay 7 will be picked again by a power-on signal from the I/O group contacted, and will cause the stepper switch to advance to the next contact. This procedure continues until all I/O groups are powered up. Once all I/O power is up, the stepper switch causes relay TDR-1 (Power-on Reset) to pick, opening the -3V power-on reset line and turning on the System Power-On indicator.

POWER-OFF SEQUENCE (MID-PAC)

The normal power-off sequence is initiated by pressing the power-off switch that opens the circuit to relay 3 (Figures 5-6 and 5-7). Relay 3 opens the circuits to contactors K2 and K3. Relay 4 drops when the -6v power supply no longer is active. Relay 4 points cause the pick of relay 7 (Stepper Control), which allows the stepper contactors to return to the home position. The stepper switch returning to home will open the circuit to TDR-1 (Power-on reset relay) causing the system power-on indicator to be turned off. Contactor K3 causes power to drop to all I/O units simultaneously.

EMERGENCY POWER-OFF (MID-PAC)

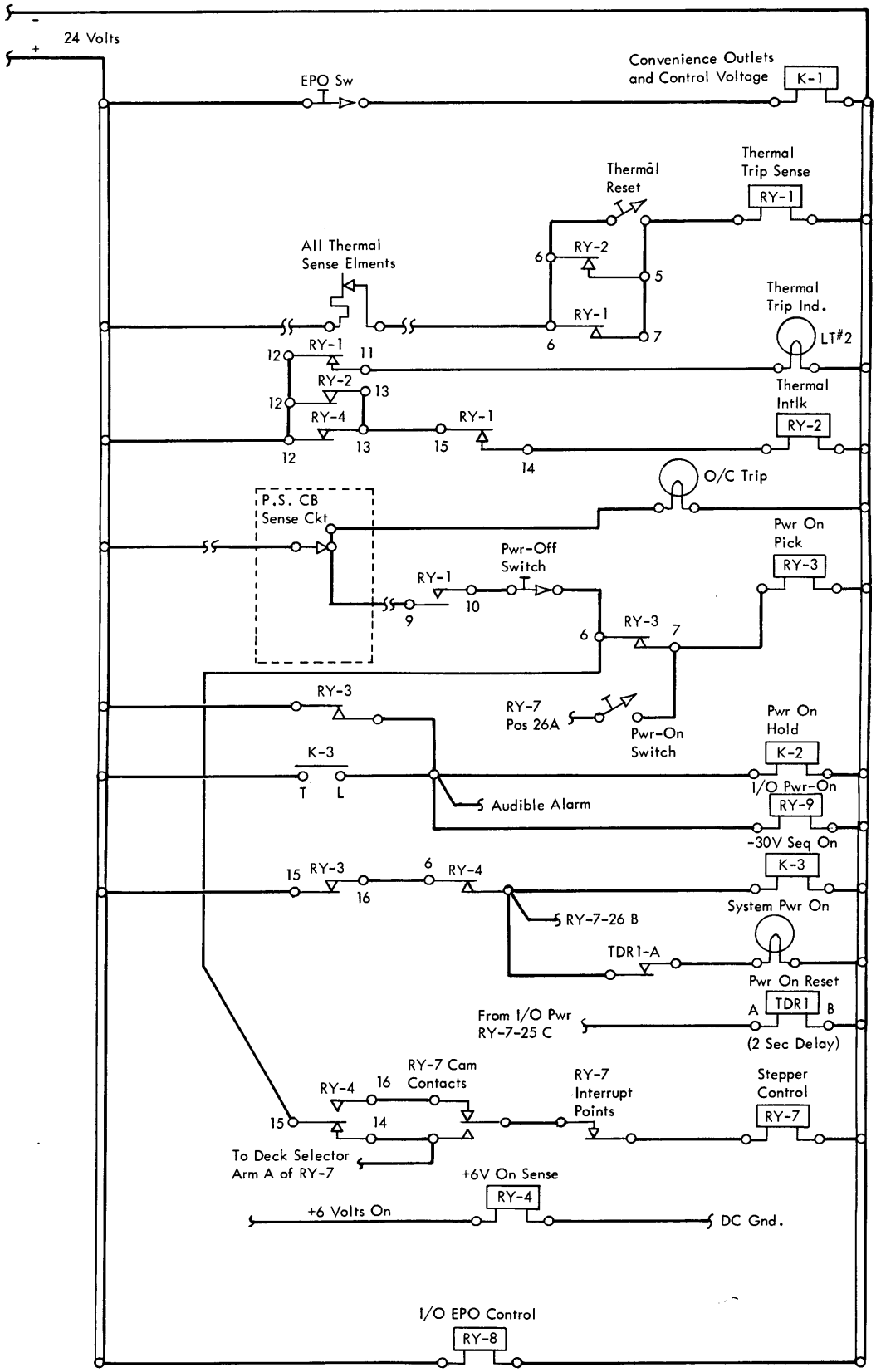
The Emergency Power-off switch opens the circuit to contactor K1 and removes all power from the system and all I/O units attached. Multi-system EPO and two-system EPO connection require special consideration because suppression diodes are added to the sequencing relays as a noise consideration. Refer to YZ045 for two system EPO connections, and to wiring diagram number 5271794 for multi-system EPO connection.

OVERVOLTAGE OR OVERCURRENT (MID-PAC)

The over-current trip sense circuitry opens the circuit to relay 3 and a normal power-off sequence occurs. The over-current indicator is turned on, and will remain on, until the circuit breaker causing the voltage trip is reset. Power may then be brought up normally.

OVER TEMPERATURE (MID-PAC)

The Thermal sense circuitry, when activated, causes the drop of relay 1 and the pick of relay 2 (Interlock relay). Picking relay 1 opens the circuit to relay 3 and a normal power-off sequence occurs. The thermal circuit may be restored only after the temperature is brought back to normal. After all thermal switches are closed, the thermal reset must be pressed to re-pick relay 1 and drop relay 2. The power-on switch may then be pressed to initiate a power-on sequence.



● Figure 5-6. Power On/Off Control (Mid-Pac)

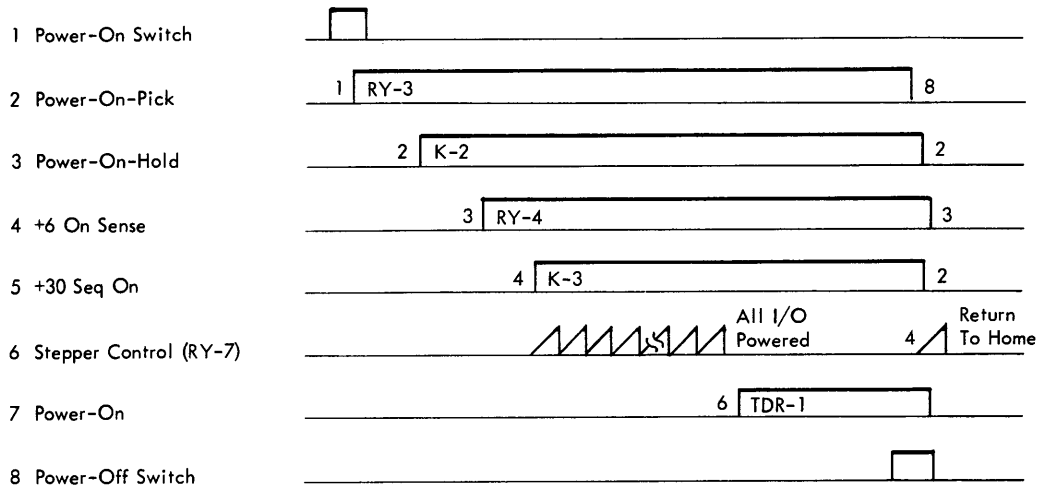


Figure 5-7. Power-On, Power-Off Sequence (Mid-Pac)

CHAPTER 6. CONSOLE AND MAINTENANCE FEATURES

2030 CONSOLE

- The system console (Figure 6-1) is divided into seven panels:

  1. ROS, selector and multiplexor channels, and CPU register, status, and check indicators are in the upper-left and left-center panels.

  2. The EPO (Emergency Power Off) switch, and use meters are in the upper-right panel.

  3. Four rotary switches, used for testing purposes, are in a center-right panel.

  4. Pushbutton switches for various functions (such as system reset) are in the lower-left corner panel.

  5. Rotary switches for storing, displaying, and matching purposes are in the lower-right corner panel. (Also in this section is the Operator's Control Panel which contains all switches and indicators needed for normal problem-program processing.)

The IBM 2030 Processing Unit has a number of indicators and manual controls that permit operation of the system in any of several modes, and observation of the results of any operation. These indicators and controls are on a panel (Figure 6-1, Part 1 of 2) that serves as both an operator's system-control-console and a customer engineer's panel. The console is divided into seven sections or panels. In one section (on the lower-right corner panel) is a portion designated as the OCP (Operator's Control Panel). The OCP contains four pushbutton switches, three sixteen-position rotary switches and six indicators. Except for the nomenclature printed on the panel, the OCP is identical on all CPU's in System/360. The OCP contains all controls and indicators necessary for normal problem-program processing. The remainder of the lower-right corner panel has six 16-position switches (A, B, C, D, E, and F). These switches are used in conjunction with other console controls to provide for such operations as manual setting of a core-storage address for storing or displaying purposes.

Just above the OCP is another section (center-right section) that has four rotary switches. These switches are all set to their process positions for normal problem-program processing. If any one or more of these switches is set to a position other than process, some kind of test is indicated. (The Test light on the OCP is then on.)

In the upper-right corner panel are the system Emergency Pull switch and the customer's and customer engineer's use-meter counters. The key-switch that determines which meter is to record time is also in this section. (The blank panel to the left of the Emergency Pull and meter panel is not used for any standard or special feature indicators or switches.)

In the lower-left corner panel are push-button keys. Each of these keys is used to initiate a particular function such as system reset, manual store, or system start.

In the upper-left panel (called the upper indicator-panel) are ROS indicators and indicators for various items associated with channel number one (selector channel one). Note, however, that the count register indicators are used for displaying the data count for either selector channel. (Selection of which selector channel's count is to be displayed is made with rotary switch E.)

The lower indicator-panel (in the middle-left portion of the console) contains indicators for channel number two (selector channel two) and multiplexor channel functions. CPU status and checks indicators and the main storage address register (MN), main storage data register (R), ALU output, and B- and A-register indicators are also in this panel.

The following descriptions relate to the circuit function(s) performed by switches and the circuits used to light indicators. For operating procedures, refer to IBM System/360 Model 30 Operator's Guide, Form A24-3373. Additional diagnostic procedures are described in diagnostic documentation shipped with each system.

Figure 6-1, Part 2 of 2 shows the ALD locations of the console indicators and controls.

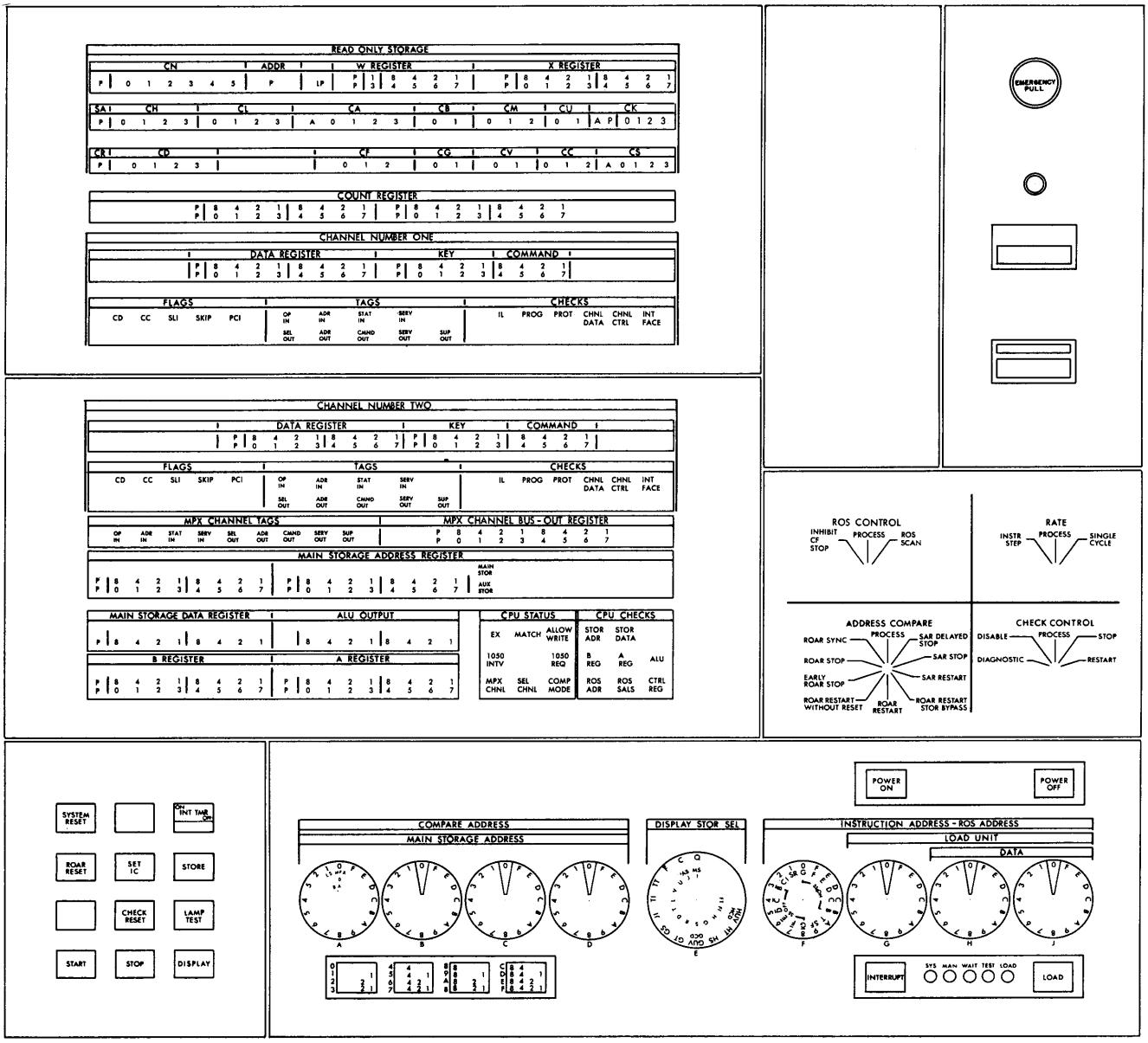


Figure 6-1. IBM 2030 Console (Part 1 of 2)

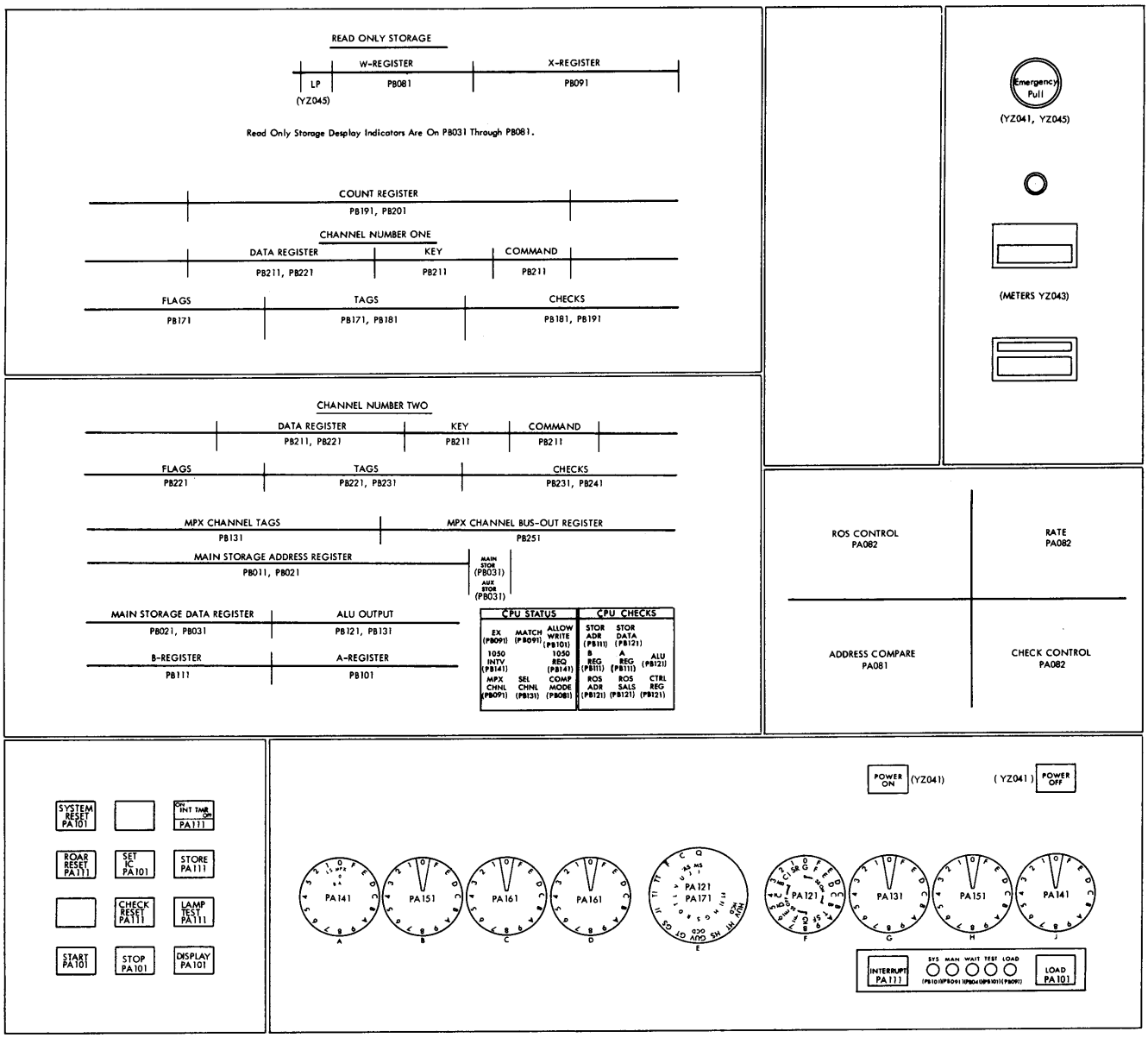


Figure 6-1. Logic Locations (Part 2 of 2)

UPPER INDICATOR-PANEL (FIGURE 6-2)

READ-ONLY-STORAGE DISPLAY AND LP INDICATOR

- ROS display consists of 55 ROS-output indicators and 15 ROS-address indicators.
- When a system stop occurs, the ROS word and ROS address (in WX indicators) displayed depends upon what caused the stop.
- The LP (Low Pressure) indicator, in the ROS display area of the console, has a dual purpose. When on, it indicates that either:
  1. The pressure to the ROS unit is too low, or
  2. The core-storage unit is below minimum operating temperature (on 2030's with a 1.5-microsecond RW cycle only).

The WX register is used to address ROS, but a separate indicating WX-register is used to light the WX indicators on the console. When the system stops, the 55 ROS-output indicators do not always display the output of the ROS word indicated by the WX address lights. The displayed word and address are dependent upon switch settings and the condition that causes the system to stop, as follows:

	Condition	Stop Caused By	Indicators
1.	Normal program processing (test light off)	Pressing system stop key	Address of displayed ROS word
2.	Address compare switch in early ROAR stop position	Match of ROAR address with setting in rotary switches A, B, C, and D	Address of the ROS word just before match occurred (ROS word displayed is at ROS address equal to setting of switches A, B, C, and D).
3.	Check control switch in stop position	ROS ADR, A REG, B REG, STOR DATA, STOR ADDR, or ALU check (i.e., one of these indicators is on in the CPU checks section of console)	Address of ROS word in process when the error occurred. (The ROS word displayed is the "next" ROS word.)
4.	Check control switch in stop position	CTRL REG, ROS SALS (indicators in CPU checks section of console)	Address of ROS word in process when the error occurred. (The ROS word displayed is the ROS word indicated by the WX display.)

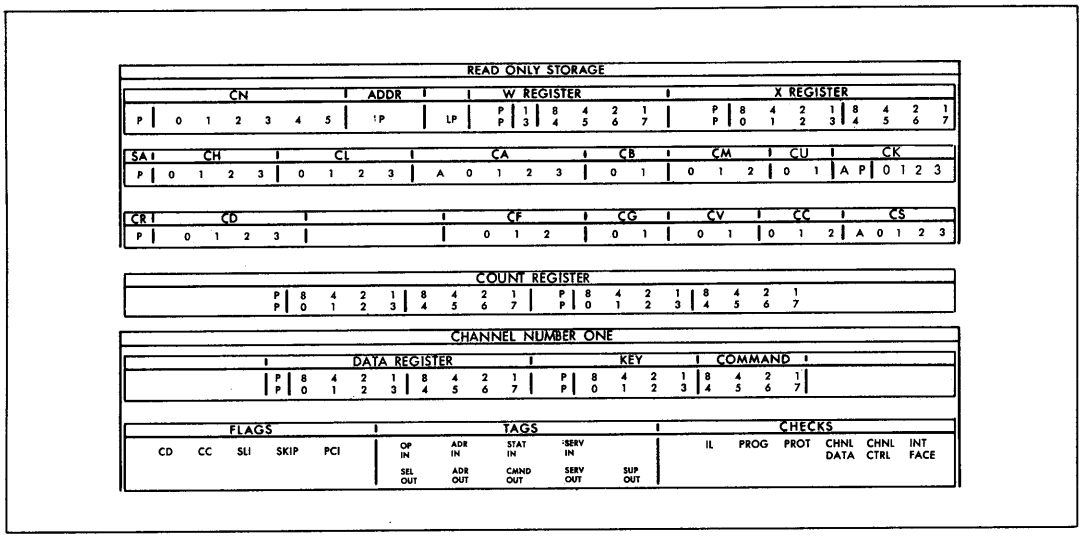


Figure 6-2. Upper Indicator Panel

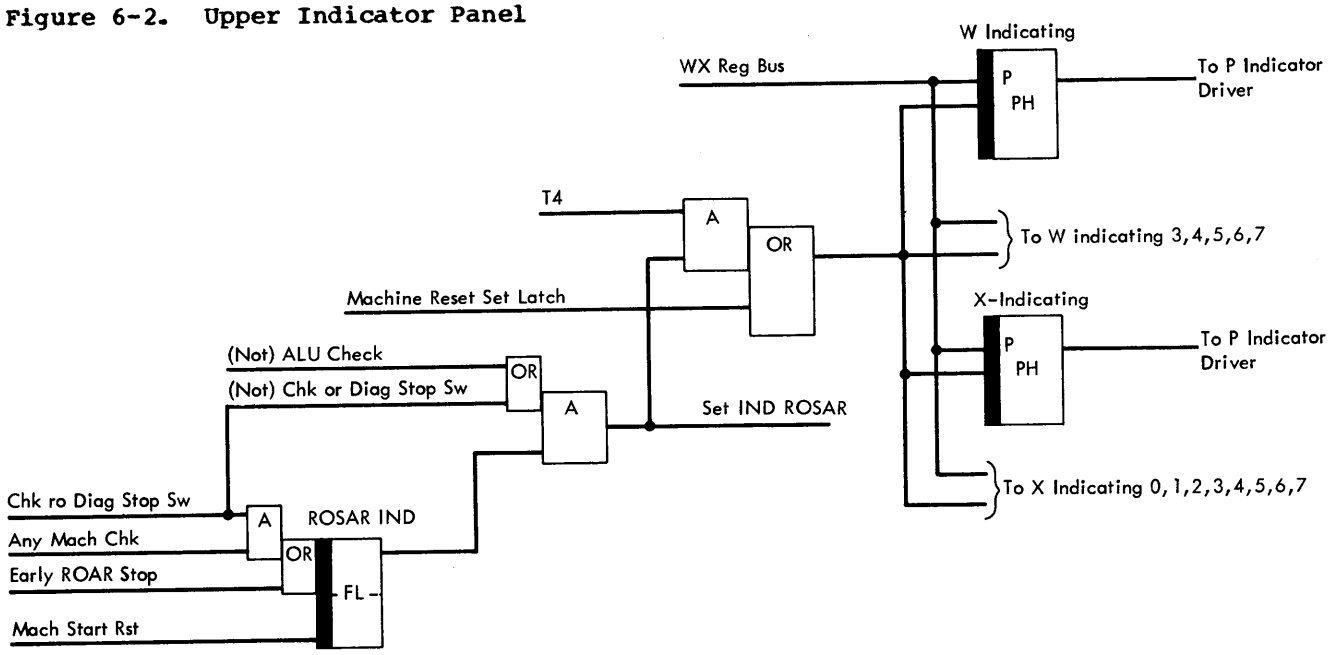


Figure 6-3. WX Indicating

Figure 6-3 shows the conditions required to set the W and X indicating registers. Notice that an ALU check line prevents setting the W and X indicating registers if the check- or diagnostic-stop-switch line is also active. This is necessary because ALU checks occur late in a cycle (T4 time).

If (as in the preceeding item 4) the check control switch is in the stop position and a control register or a ROS SALS parity check occurs, the ROS word in which the error occurred is displayed. The SALS cannot be loaded with a new word because the CROS GO pulse is blocked at T2 time (Figure 6-4).

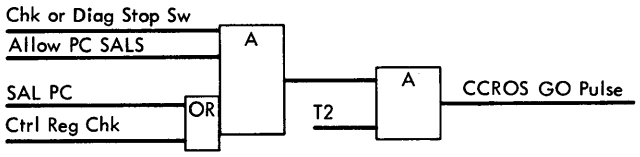


Figure 6-4. CCROS GO Pulse

The LP indicator (not a ROS word output display light) is turned on whenever the pressure to the ROS unit is too low or the core storage unit is below minimum operating temperature.

COUNT REGISTER

- The contents of the GC and GD (for selector channel one) or the HC and HD (for selector channel two) registers can be displayed in the eighteen positions of the count register.
- The GC and GD (HC and HD for selector channel two) registers contain the residual data count for an I/O operation on selector channel one.

The CCW count is loaded into the GC (high-order count bits) and GD (low-order count bits) registers at the start of an I/O operation on selector channel one. As each data byte is transferred, the count is decremented by one. Normally, it is not possible to stop a selector channel operation in order to observe an intermediate count. However, if there is a residual count (after the I/O operation is completed and the system is stopped), it can be displayed in the count register indicators.

The conditions necessary to display the GC and GD registers are shown in Figure 6-5. Notice that it is not necessary to press the Display key to gate the contents of the GC and GD registers to the count register indicators. All that is required is to set rotary switch E to the GCD-GUV position. (The HC and HD registers provide the same function, but for selector channel two).

CHANNEL NUMBER ONE DISPLAY

Channel number one indicators are used for display of selector channel one information. For detailed descriptions of selector channel one (and selector channel two and multiplexor channel) register operations, refer to the 2030 I/O Control, System/360 Model 30 Theory of Operation, Form Y24-3362.

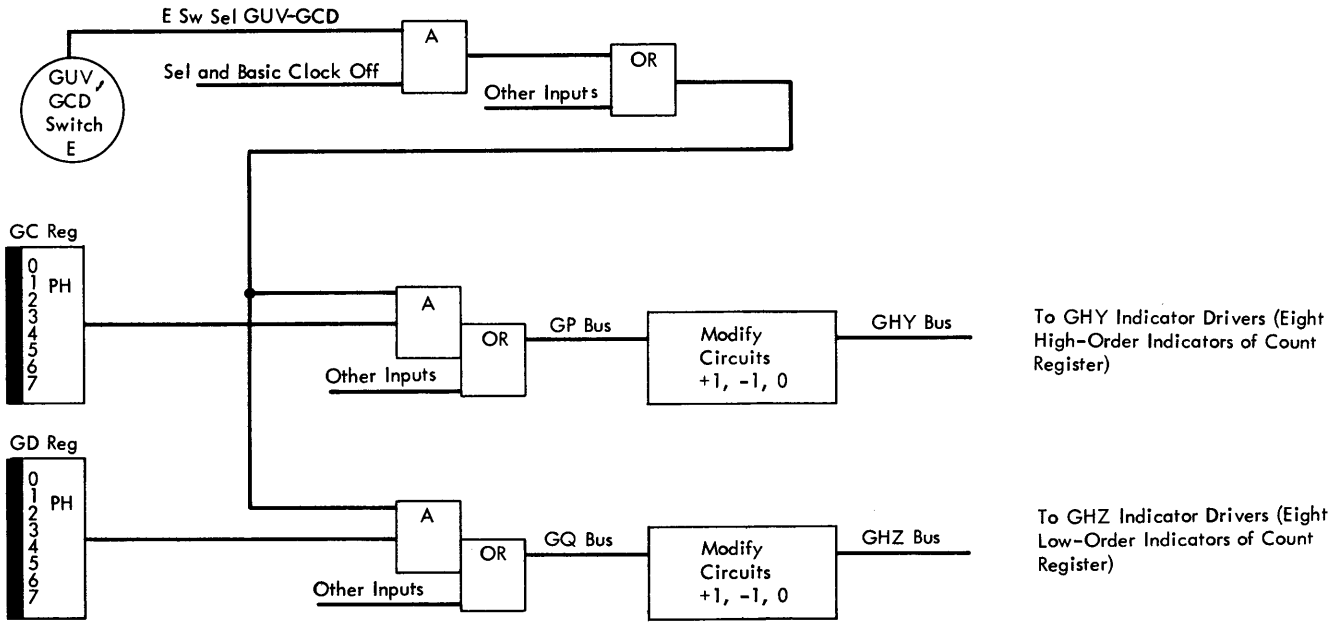


Figure 6-5. GCD to Count Register Display

Data Register

- The data register indicators display each byte set into the GR register (for selector channel one).

The data register indicators provide full time display of the GR register's contents. That is, GR register outputs are sent directly to the data register indicator drivers. Hence, it is not necessary to press or set any console switch to gate the GR register outputs to the data register indicators.

Key

- The protection key from the GK register is displayed in the key indicators.

The protection key from the CAW is set in the GK register. For any selector channel one operation during which data is sent from the channel into main storage, the GK register contents are compared to the Storage key for the main storage area affected.

The key indicators provide full time display of the GK register's contents. It is not necessary to set or press any console switches in order to gate the GK-register's outputs to the key display lights.

Command

- The command indicators display the contents (four low-order bits of the CCW operation code) of the GG register.

The four low-order bits of the CCW operation code are loaded into the GG register and displayed in the four command indicators. It is not necessary to set or press any console switches to gate the GG register outputs to the command display lights.

Flags

- The flag indicators (CD, CC, SLI, SKIP, and PCI) display the contents of the GF register.

The flag bits, when on, indicate the following functions:

GF Reg Bit	Flag	Function
0	CD	chain data
1	CC	chain command
2	SLI	suppress incorrect length
3	SKIP	suppress data transfer to main storage during read, read backwards, and sense operations.
4	PCI	program controlled interruption bit (causes channel to attempt an interruption upon fetching the CCW that contains this bit).

The flag indicators provide full-time display of the contents of the GF register. It is not necessary to set or press any console switch to gate the GF register outputs to the flag display lights.

Tags

- Indicator drivers for selector channel one outbound tags are activated by the outputs of the GA-register
- Indicator drivers for inbound tags are activated by interface terminator circuits

The outputs of the GA-register provide inputs to the indicator drivers for the outbound tags of selector channel one:

Tag	Function
1. SEL OUT (select out)	Indicates that I/O units on selector channel one are being polled to determine which unit requested service.
2. ADR OUT (address out)	Indicates that the information on bus-out is an address.
3. CMND OUT (command out)	Indicates that the information on bus-out is a command; means "proceed" in response to address-in after initial selection; or means that no more data is needed.
4. SERV OUT (service out)	Indicates that the CPU has accepted the information on bus-in or has provided data on bus-out.
5. SUPP OUT (suppress out)	Indicates by itself or with other tags. a. suppress status b. chained command control c. selective reset

Inbound tag indicator drivers are activated by the outputs of standard interface terminator circuits in the CPU:

Tag	Function
1. OP IN (operational in)	Indicates that an I/O unit is selected and is in communication with the channel.
2. ADR IN (address in)	Indicates that address of the selected I/O unit is on bus-in.
3. STAT IN (status in)	Indicates that the selected I/O unit has placed status information on bus-in.
4. SERV IN (service in)	Indicates that the selected I/O unit is ready to transmit or receive data.

It is not necessary to set or press any console switches to gate tag signals to the tag indicator drivers.

Checks

- The channel one checks lights indicate detected malfunction(s) during selector channel one operations.
- The checks indicator drivers for selector channel one are activated by outputs from the GE-register.

The outputs of the GE-register activate indicator drivers for selector channel one check lights. It is not necessary to press or set any console switch to gate the output of the GE-register to the checks indicator drivers. For detailed information about what the check conditions signify, refer to 2030 I/O Control, System/360 Model 30 Theory of Operation, Form Y24-3362. The general functions of the checks are:

Check	Function
1. IL (incorrect length)	Indicates that the number of bytes contained in the assigned storage area is not equal to the number of

bytes requested or offered by the I/O unit, provided the SLI flag is not on.

2. PROG (program) — Indicates that the channel has detected a programming error.

3. PROT (protection) — Indicates that channel has attempted to place information in a protected area of main storage. (This check can occur on read, read backwards, or sense operations.)

4. CHNL DATA (channel data) — Indicates that a data byte in the GR-register has even parity.

5. CHNL CTRL (channel control) — Indicates a control byte in GR-register has even parity. (Certain other conditions may also cause this indication.)

6. INT FACE (interface control) — Indicates that:
a. A response from a control unit is not given to a signalling sequence initiated by the channel,
b. A device is busy (after device end has been given) to an initial selection sequence,
c. Either no address response or an address mismatch occurs as a result of an addressing sequence initiated by the channel, or
d. A parity error is detected on status or address information sent from a control unit to the channel.

LOWER INDICATOR PANEL (FIGURE 6-6)

CHANNEL NUMBER TWO DISPLAY

These indicators provide the display of the same functions as the channel number one display, except that these indicators are for selector channel two.

MPX (MULTIPLEXOR) CHANNEL TAGS

The functions of the multiplexor channel tags are the same as the corresponding tags described in the Channel Number One Display section, except that they pertain to the multiplexor channel only. It is not necessary to set or press any console switch to gate the multiplexor tag lines to their corresponding indicator drivers.

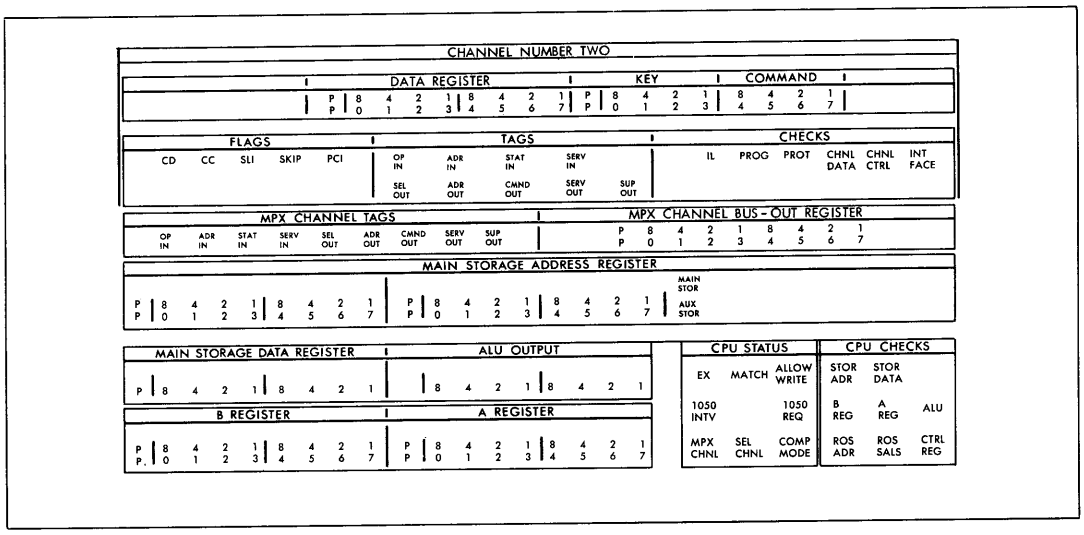


Figure 6-6. Lower Indicator Panel

MPX CHANNEL BUS-OUT REGISTER

- The outputs of the multiplexor channel bus-out register (FO) activate the MPX channel bus-out indicator drivers.

The FO-register is loaded with data from the R-register when information is to be sent to the bus-out lines of the multiplexor channel. The contents of the FO-register are indicated in the MPX channel bus-out lights. It is not necessary to press or set any console switch to gate the contents of the FO-register to these lights.

Note that if a stop occurs immediately before processing a micro-word that gates the R-register to the MPX channel bus-out register, the bus-out register may have bad parity. This results from the fact that the bus-out register has been reset, but not set (i.e., all bits, including parity bit, are off).

MAIN STORAGE ADDRESS REGISTER AND MAIN STOR AND AUX STOR INDICATORS

- The main storage address register lights display the contents of the M- and N-registers.
- The main storage address register display is for a main storage location if the MAIN STOR light is on and for an auxiliary storage location if the AUX STOR light is on.

The M-register's contents (eight high-order bits of the storage address) and the N-register's contents (eight low-order bits of the storage address) are displayed by the main storage address register indicators. For a 64K 2030 with a 2-microsecond storage cycle, there are two M and two N registers (one MN set for each 32K of storage). Only the one for the lower 32K of storage, however, causes the main storage address indicators to light. If an address that pertains to main storage is displayed, the MAIN STOR light is on; if an address that pertains to auxiliary storage is displayed, the AUX STOR light is on. The MAIN STOR and AUX STOR lights are just to the right of the main storage address register display indicators. Note that if the AUX store light is on, it remains on until an access to main storage occurs; if the MAIN STOR light is on, it remains on until an access to auxiliary storage occurs.

The M- and N-registers are set by inputs from the UV- or IJ-registers or rotary switches A, B, C, and D. Also, the T-register can provide input to set the N-register. (In 1400 compatibility mode, the L-register is gated to M at the same time that the T-register is gated to N.) During selector channel data transfers, the M- and N-registers are set with address information from the GUV-register (for selector channel one) or the HUV-register (for selector channel two).

In manual operations, the information to be displayed in the main storage address register indicators is determined by the setting of rotary switch E (and in some cases by additional settings of switches A, B, C, and D). The selections that can be made (for input to the M- and N-registers) with rotary switch E are:

Rotary switches A, B, C, D
UV-registers
IJ-registers
GUV-registers
HUV-registers

The system clock must be stopped and the allow write indicator must be off before any of these items can be manually displayed in the main storage address register indicators.

If an access to auxiliary storage is required, the contents of the high-order hexadecimal digit (four high-order bits) of the main storage address register determine which part of the auxiliary storage (i.e., local storage or one of the MPX storages) is to be addressed.

Note that during wait state and process stop, the contents of the instruction counter (I- and J-registers) are displayed in the B- and A-register indicators. The current operation code (i.e. the last processed or the next to be processed) is not displayed.

MAIN STORAGE DATA REGISTER

- The main storage data register indicators provide display for information (9-bits) in the R-register.

Information (for CPU or multiplexor channel operations) sent to or from core storage (either auxiliary or main) passes through the R-register. The main storage data register indicators provide full time display of the contents of the R-register. It is not necessary to press or set any console switch in order to gate the R-register's outputs to the main storage data register indicator drivers.

ALU DISPLAY

- The ALU indicators provide display of ALU output (including the P bit generated at the output of ALU).

The ALU indicators provide full-time display of the output of ALU. Also, the P bit generated for a result byte at the output of ALU is displayed. It is not necessary to press or set any console switch to gate the outputs of ALU to the ALU indicator drivers.

B- AND A-REGISTER DISPLAY

- The B- and A-register indicators display the outputs of the B- and A-registers.
- During wait state and process stop, the contents of the instruction counter (I- and J-registers) are displayed in the B- and A-registers.

The B- and A-register indicators provide full-time display of the contents of the B- and A-registers. It is not necessary to press or set any console switch to gate the outputs of the B- and A-registers to the B- and A-register indicator drivers.

During manual operations, the information to be displayed in the A-register depends on the console switch settings used. (Refer to the Display Key section of this manual.)

When a wait state (instruction processing stopped and program waiting for an interruption) or process stop (such as when the stop key is pressed) condition occurs, the instruction counter is displayed in the B- (I-register contents) and A-registers (J-register contents).

CPU STATUS

- The CPU status lights indicate the operating status of the CPU.

The CPU status lights provide full-time display of the CPU's status condition. It is not necessary to press or set any console switch to gate a CPU status signal to the associated indicator driver. The CPU status indicators are:

1. EX: This lamp is turned on at the end of each instruction execution (i.e., whenever the micro-instruction branch-on-interrupt occurs). During execution of the micro-instruction immediately following the branch-on-interrupt word, the EX lamp is turned off. Note that if the system stops at the end of instruction execution (for example, if the stop key is pressed), the EX lamp remains on. It is turned off when the CPU clock is restarted and processing of the next micro-instruction is started.

2. MATCH: Some modes of operation require use of an exclusive-OR match circuit. The match indicator is turned on when the compare-address (in rotary switches A, B, C, and D) matches the contents of either the main storage address register (MN) or the read-only-storage address register (WX). The position of the address-compare switch determines which of these registers is monitored, as well as the system response to a match.

3. ALLOW WRITE: Whenever the allow write indicator is on, a read operation for a storage location (auxiliary or main) is completed, but the subsequent write operation has not occurred. This indicator must be off before manual display or store operations for core storage are allowed.

4. 1050 INTV: This light is turned on whenever operator intervention is required at the 1050 Documentary Console.

5. 1050 REQ: This light turns on whenever the operator presses the Request key on the 1052. It is reset when attention status is recognized by the attachment and accepted into the unit status register.

6. MPX CHNL: Whenever a multiplexor channel share-request is recognized by the CPU, this light turns on. It is turned off at the completion of the share cycle.

7. SEL CHNL: This lamp is lighted whenever either selector channel is using ROS (such as for a selector channel chaining operation).

8. COMP MODE: Whenever the system is operating in compatibility mode, this light is on. It is turned on at the same time as the W3 lamp (i.e., the three-bit position of the W register) and turned off at T2 time of the first ROS cycle in which W3 is not on.

CPU CHECKS

- The CPU checks indicator drivers are activated by outputs from the machine check register.
- Each CPU check lamp (except the ALU) indicates, when on, that bad parity is detected.

The outputs of the machine check register activate the indicator drivers for the CPU checks lamps. The indicator turned on and the machine check register position set for each check are:

Position	Indicator	Check
0	A REG	A-register parity
1	B REG	B-register parity
2	STOR ADDR	MN-register parity
3	CTRL REG	Control-register parity
4	ROS SALS	ROS SALS parity
5	ROS ADDR	ROS address parity
6*	STOR DATA	R-register parity
7	ALU	ALU check (not a parity check)

(*If the storage protection feature is used, a parity check for the low half of the Q register can also set bit six of the MC-register and cause a STOR DATA check indication.)

In ALU, a duplicate check is made to determine that each output line at an up level has a corresponding line at a down level.

INDICATORS ON OCP (FIGURE 6-7)

- The lamps on the OCP indicate overall system state.

As already mentioned, the OCP (Operator's Control Panel) is a portion of the lower-right console panel that has all the keys and indicators needed for normal problem-program processing. The OCP indicators and their functions are:

1. SYS: This indicator is on whenever the customer or CE use-meter is recording time.

2. MAN: Whenever the CPU clock is stopped (and no selector channel transfer is in progress), this indicator is on. As the name of this light implies, several of the manual controls are operative only when the system is in this state.

3. WAIT: This light is on when the CPU is in the wait state (i.e., CPU clock running but instruction execution is not taking place). If an interruption occurs, the CPU is taken out of the wait state and processing occurs, depending upon the program directing the system.

4. TEST: This light is on whenever any one or more of the following switches is in any position other than process:

   a. ROS Control
   b. Rate
   c. Address Compare
   d. Check Control

   For normal problem-program processing, all of these switches should be in the process positions.

5. LOAD: Whenever a load microprogram is in progress, this indicator is on. It turns on after the Load key has been pressed and then released, and it turns off when the initial PSW is successfully loaded.

6. Power-On Key Indicator: A light behind the Power-on key turns on after the Power-on key is pressed, but only after the CPU and all on-line I/O units have been power-sequenced on.

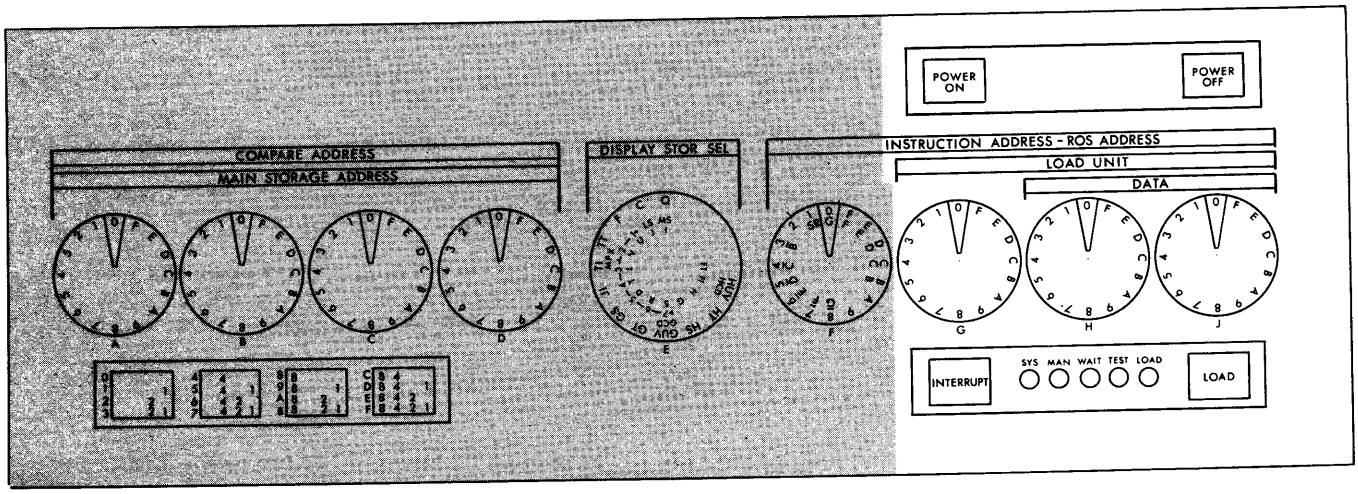


Figure 6-7. Operator Panel

PUSHBUTTON CONTROLS ON OCP (FIGURE 6-7)

POWER-ON KEY

- The Power-on key, when pressed, starts the power-on sequence for the CPU and all on-line I/O units.

The power-on sequence for the CPU and all on-line I/O units starts when the Power-on key is pressed. When the entire sequence is successfully completed, a light behind the Power-on key turns on. A system reset function (for CPU and on-line I/O units) occurs during the power-on sequence.

All data flow registers (but not the general-purpose or floating-point registers in local storage) in the CPU are reset to zero. Any priorities that happen to turn on are reset. Also, the CPU clock is reset so that a possible access to storage is prevented. Hence, information in core storage is not disturbed.

The I/O units are sequenced on, one by one, to prevent line surges. If power cannot be brought up for an on-line I/O unit, further power-on sequencing is prevented (console Power-on light remains off) until coreective action is taken for that I/O unit.

POWER-OFF KEY

- Pressing the power off key removes power from the CPU and all on-line I/O units.
- If the allow write latch is on, the contents of the R-register are written into the storage location specified by the contents of the MN-registers.

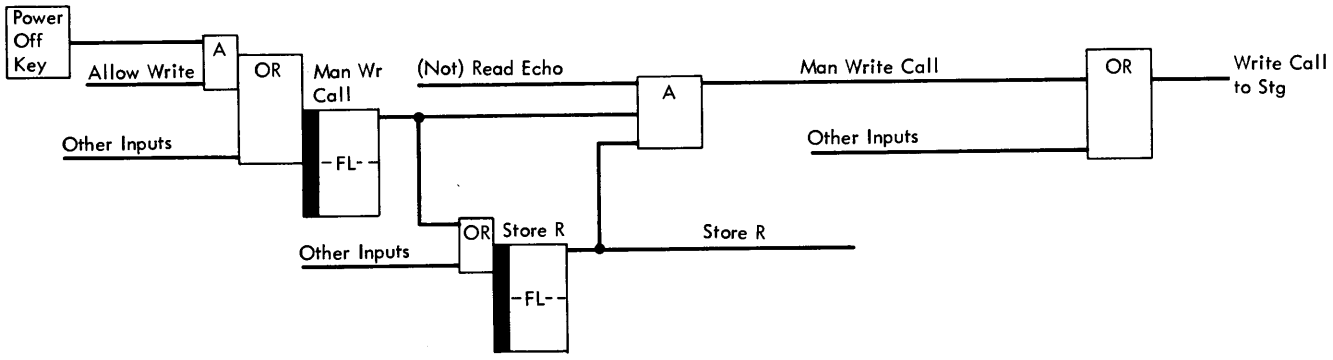


Figure 6-8. Forced Write Cycle During Power Off

The Power-off key, when pressed, drops power to the CPU and on-line I/O units. Power to the I/O units is not sequenced off, but is dropped for all I/O units simultaneously. The contents of core storage are not altered during the power-off operation. If a storage read cycle has been taken (the allow write latch is on), a manual write cycle (i.e., not a normal clock controlled write cycle) is forced during the power-off operation (Figure 6-8).

The Power-off key takes precedence over the Power-on key, such that when both are pressed simultaneously, power is dropped.

INTERRUPT KEY

- Pressing the interrupt key causes an external interruption.

The external interruption that occurs when the Interrupt key is pressed results in setting bit 25 on in the interruption code of the old PSW. (The system recognizes this interruption, however, only if programmed to do so.)

LOAD KEY

- When pressed, the Load key initiates a system reset; when released, it starts an initial program load routine.

The address of the I/O unit, from which the program is loaded, is set into rotary switches G, H, and J before the Load key is pressed. Pressing the Load key causes a system reset and sets the machine reset latch. When the Load key is released, the load indicator turns on, the CPU clock starts, the basic microdiagnostic routine is initiated, and this is followed by the clear-UCW microprogram routine. At the end of the clear-UCW microprogram, the machine reset latch is reset and a load trap occurs. The load microprogram is then started. The loading operation is completed when the IPL PSW (for the program being loaded) is successfully set up in circuitry. At this time the Load light turns off.

DATA AND ADDRESS ENTRY SWITCHES (FIGURE 6-7)

- Eight rotary switches (A, B, C, D, F, G, H and J) are used to enter addresses or data (in odd parity) into the CPU.

Rotary switches A, B, C, D, F, G, H, and J are all sixteen-position switches. Each position of each switch provides one hexadecimal digit (four bits plus parity). The names (on the console) over the switches describe the general purposes of the switches. Information in the switches can be used for :

	Functions	Switches
1.	A restart or stop address for matching against the contents of the main storage address register (MN) or the ROS address register (WX).	A,B,C,D
2.	A core storage address for manual store or display operations.	A,B,C,D
3.	Manual change of an instruction address or a ROS address.	F,G,H,J
4.	Manual set-up of the address of a load unit (from which a program is to be loaded into core storage).	G,H,J
5.	A data byte to be stored into core storage or into a data-flow register.	H,J
6.	Sense switch settings (on or off) for compatibility mode operations.	F

Switches A, B, C, and D are connected to the main storage address register and to the match (or compare) circuit, while switches F,G,H, and J are connected to the data-bus system through the A- and B-registers and to circuitry that leads to the WX-registers.

Rotary switch E is set to MS (for main storage) or AS (for auxiliary storage)

during manual store or display operations involving storage. Local storage (or one of the MPX storages) is further designated by the setting of switch A (Figure 6-9). Switches C and D are then used to specify the address of a particular local or MPX storage byte.

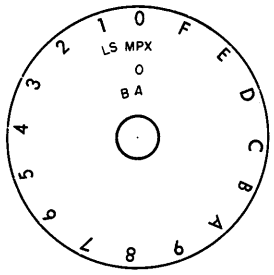
DISPLAY STORAGE SELECTION SWITCH (SWITCH E--FIGURE 6-7)

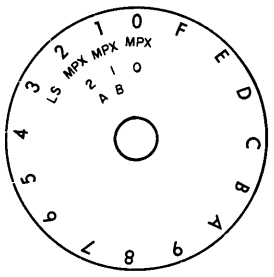
- The display storage selection switch is two concentric switches at one console location.
- The inner switch has three positions.
- The outer switch has sixteen positions, each performing one of three different functions, depending upon the position of the inner switch.
- The display storage selection switch is used to select a register or storage area for display or store purposes.

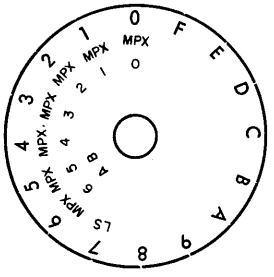
The Display Storage Selection switch (switch E) provides a means of selecting any one of a number of registers or a general storage area (main or auxiliary) for manual store or display operations. (If main or auxiliary storage is selected, the address of the specific storage location is specified by the settings in switches A, B, C, and D.)

Selections that can be made with switch E are:

Outer Switch Position	Inner Switch Position 1	Inner Switch Position 2	Inner Switch Position 3
1	Q	MS	I
2	C	AS	J
3	F	Spare	U
4	TT	Spare	V
5	TI	Spare	L
6	JI	Spare	T
7	GS	Spare	D
8	GT	Spare	R
9	GUV,GCD	Spare	S
10	HS	Spare	G
11	HT	Spare	H
12	HUV,HCD	Spare	FI
13	Spare	Spare	FT
14	Spare	Spare	Spare
15	Spare	Spare	Spare
16	Common	Common	Common







Note: The MPX 0 setting is on all 2030 consoles. The MPX 1 and MPX 2 settings are on all 2030's that have 16K or more main-storage addresses. MPX 3, MPX 4, MPX 5, and MPX 6 settings are on 2030's (with 32K of more main-storage addresses) that have the 224 subchannel special feature.

Figure 6-9. Rotary Switch A

PUSHBUTTON KEY CONTROLS (FIGURE 6-10)

SYSTEM RESET KEY

- When the System Reset key is pressed, the system (CPU, channels, and I/O control units) is reset to its initial state.
- ROS address 0000 is set into the WX-registers when the System Reset key is pressed.
- Pressing the System Reset key initiates a system reset regardless of the state (i.e., CPU clock can be running or stopped) of the system.
- Any error status information is reset when the System Reset key is pressed.

When the System Reset key is pressed, a system reset is initiated. During the reset, all registers (but not the 16 general or the 4 floating point registers) are set to zero with correct parity. Also, all latches, except the Machine Reset and the diagnostic latch are reset. The Diagnostic latch then causes the priority latch to turn on. The Machine Reset latch is set on, thereby disabling all traps until it is reset. If the Start key is then pressed (after the system reset function is completed), the machine starts at a diagnostic microprogram (unless the Check Compare switch is in the disable position) which is looped 128 times. The clear-UCW routine then clears the flag bytes of all the UCW's. After these flag bytes are reset to zero, the Machine Reset latch is reset and traps can occur. The system reset function is completed when the Machine Reset latch is reset.

Note that if the ROAR Reset key is pressed after the System Reset key is operated (and before the Start key is pressed) the flag bytes in the UCW's are not cleared when the Start key is pressed, and the microdiagnostic routine is not processed.

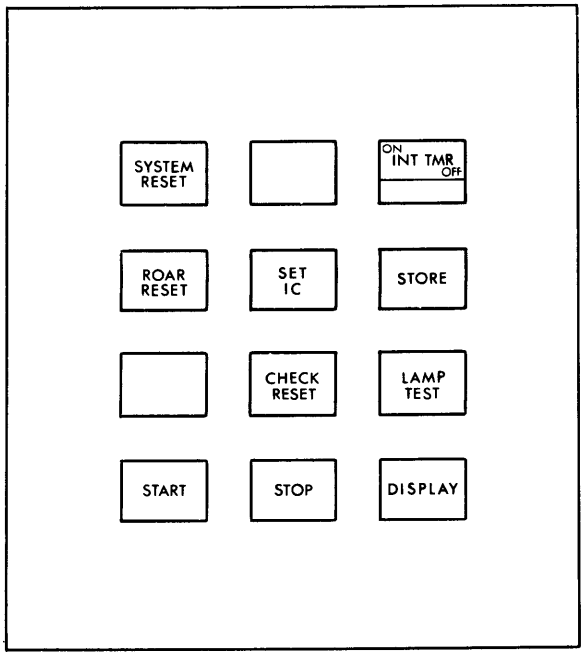


Figure 6-10. Pushbutton Controls

ROAR RESET KEY

- Pressing the ROAR Reset key allows a manual change of the ROAR address by gating the contents of switches F, G, H, and J to the WX-registers when the start key is pressed.
- The ROAR Reset key is effective only when the CPU clock is stopped. (The ROAR reset function, however, does not occur until the CPU clock is subsequently started.)
- Pressing the ROAR Reset key blocks the clear-UCW routine and the microdiagnostic routine that normally occur after a system reset.

The ROAR Reset key is used to set-up for a manual change of the ROAR address in the WX-registers. The address to be set in ROAR is manually set into switches F, G, H, and J. (The CPU clock must be stopped before the ROAR Reset key is active.) The ROAR Reset key is pressed and then the start key is pressed to start the CPU clock. As soon as the CPU clock starts, the ROAR reset function occurs.

When the ROAR Reset key is pressed, a latch is set that gates the outputs of switches F, G, H, and J to the WX-registers and blocks next-address information from being sent to the WX-registers.

If the ROAR Reset key is pressed after a system reset function, but before the CPU clock is started, the machine reset latch and the diagnostic latch are turned off. Then when the CPU clock is started, the UCW flag bytes are not reset to zero, and the microdiagnostic routine is not performed.

START KEY

- The start key starts the CPU clock; resulting system operation depends on what conditions exist when the start key is pressed.

If the start key is pressed after a normal stop (for example, after the stop key is pressed), instruction processing continues as if no stop occurred. Machine status is unaffected.

If the Start key is pressed after a system reset, the microdiagnostic routine is performed and the flag bytes of the UCW's (in MPX storages) are reset to zero by a clear-UCW microprogram routine. If the start key is pressed again, the 2030 loads a PSW from address 0000 and processing starts.

SET IC KEY

- The Set IC key is used in conjunction with switches F, G, H, and J to manually change the setting of the instruction counter (IJ-registers).
- The set IC key is operative only if the CPU clock is stopped (Manual light is on).

Before the address in the instruction counter (IJ-registers) can be changed, the CPU clock must be stopped (Manual light is on). Then the address to be set into the instruction counter is manually dialed into rotary switches F, G, H, and J and the Set IC key is pressed. When the Set IC key is released, the CPU clock is started and a forced branch (trap) is taken to ROS address 0001 (the set IC trap). The address from switches F, G, H, and J is set into the instruction counter and displayed in the B-register (I-register portion of address) and A-register (J-register portion of address). The CPU clock then stops at ROS address 0FF. Any outstanding channel share cycles are completed before the stop at address 0FF occurs.

A set IC operation after a system reset is similar to a Start key operation after a system reset in that the flag bytes of the UCW's are cleared. Then the instruction counter is set with the address in switches F, G, H, and J.

If you desire to start processing at the instruction address set up by the set IC procedure, press the Start key.

CHECK RESET KEY

- Pressing the check reset key causes the machine check register and several machine check control latches to be reset to the no-error state.
- The check reset key is operative when the CPU clock is either running or stopped.

Pressing the Check Reset key resets all positions of the machine-check register to the no-error state. In addition, the first machine check, the second error stop, and the check-restart latches are reset. In other words, all machine-check logic circuits are reset.

STOP KEY

- Pressing the stop key causes the CPU to stop (manual light turns on) at the end of execution of the instruction in progress.
- All pending interruptions are taken before the CPU clock is stopped.

The CPU proceeds to the end of execution of the instruction being processed at the time the Stop key is pressed. (This state is recognized by a branch on interrupt.) All pending interruptions are taken before the CPU clock is stopped. Any I/O operation in process at the time the stop key is pressed is allowed to finish before the CPU clock is stopped. If an I/O device is involved in command or data chaining, then the chaining is completed before the clock is stopped. The Manual light turns on when the clock stops. At this point, the 2030 is in a stopped state that permits the multiplexor channel to take share-request traps and the selector channels to take data cycles. When the CPU clock stops, the address of the next instruction is displayed in the B- and A-registers.

INTERVAL TIMER SWITCH

- When on, the Interval Timer switch allows the interval timer to advance.
- When off, the Interval Timer switch prevents interval timer advance.

If the interval timer special feature is installed in the 2030, the Interval-Timer-Toggle switch (not a pushbutton) controls its operation. If the Interval Timer switch is off, the timer control latch is held on to block C-counter drive pulses, thus preventing timer advance.

LAMP TEST KEY

- When the Lamp Test key is pressed, all console indicators should light.
- The Lamp Test key can be pressed at any time and does not affect any system operation.

An input is provided to each indicator driver for testing purposes. This input is one leg of an OR function; the second leg is the functional indicator-driver input. When the Lamp Test key is pressed, all indicator drivers should light all console panel lights. (The console lamps do not light as brightly when this test is made as they do when the lamps are lighted by their normal functional inputs.)

STORE KEY

- During a Store key operation the byte specified by the data switches (H and J) is loaded into the area specified by the Display-store Selection switch (E).
- The Store key is inoperative if the CPU clock is running.
- The clock is not used during a Store key operation.
- ROS is not used during a Store key operation.

Pressing the Store key gates the contents of switches H and J into the B-register. The B-register is gated high and low through ALU. The resultant data byte appears on the Z-bus and is gated to the area selected by switch E. If a register is selected to receive the data byte, the Z-bus is gated directly to the selected register. If a storage location is selected to receive the contents of H and J, the Z-bus is gated to the R-register. The selected register is then gated to the A-register. Therefore, at the end of a store operation, the A- and B-registers should contain the data specified by the settings of switches H and J.

In the case where storage is selected by switch E, switches A, B, C, and D provide the storage address, and a manual read-cycle and a manual write-cycle are taken (the clock is not used) to place the data byte from switches H and J (then in the R-register) into the desired location.

The CPU clock must be stopped for the store operation to take place. The allow write latch (allow write indicator) must be off if the store operation is to core storage.

Data cannot be manually stored in all registers selected by switch E. The registers into which data cannot be manually stored are designated by an asterisk (*) in Figure 6-11.

Register to be Displayed	Usual Function	Where Displayed
I	Instruction Address (high-order bits)	A-register (also the high-order eight bits of the main-storage address register if the allow-write indicator is off)
J	Instruction Address (low-order bits)	A-register (also the low-order eight bits of the main-storage address register if the allow-write indicator is off)
U	Data Address (high-order bits)	A-register (also the high-order eight bits of the main-storage address register if the allow-write indicator is off)
V	Data Address (low-order bits)	A-register (also the low-order eight bits of the main-storage address register if the allow-write indicator is off)
L	Data Length	A-register
T	Auxiliary Storage Address	A-register
D	General Purpose Data Register	A-register
R	Storage Data Register	A-register (Also has own display in main-storage data-register indicators)
S	Status (CPU)	A-register
G	Instruction Operation Code	A-register
H	Priority Status Register	A-register
*FI	Multiplexor Channel Bus-In	A-register
*FT	Multiplexor Channel Tags	A-register
Q	Storage-Protection key in PSW (High 4-bits) Storage-Protectection key of block of storage just used (low 4-bits)	A-register
*C	Interval Timer Count	A-register
*F	External Interrupt: Interval Timer (bit 0) Console (bit 1) Six direct-control interrupts (bits 2 through 7)	A-register
*TT	1050 Documentary Console Tags	A-register
*TI	1050 Documentary Console Bus-In	A-register
*JI	Direct Control Bus-In	A-register
*GS	Selector Channel One Status	A-register
*GT	Selector Channel One Tags	A-register
*GUV-GCD	GUV contains storage address for data for selector-channel one. GCD contains the current byte count for selector-channel one	GUV in main-storage address register. GCD in count register (18 bits each).
*HS	Selector Channel Two Status	A-register
*HT	Selector Channel Two Tags	A-register
*HUV-HCD	HUV contains storage address for data for selector-channel two. HCD contains the current byte count for selector-channel two.	HUV in main-storage address register. HCD in count register (18 bits each)

Note: * Indicates that you cannot manually store data in the designated register

Figure 6-11. Display

DISPLAY KEY

- A Display key operation allows selected information to be gated to a display register.
- The selected information can be from any register or from any core storage location selectable by rotary switch E.
- The CPU clock must be stopped for all display operations.
- The allow write latch must be off for display of any storage location or display of the contents of the I, J, U, or V registers in the MN register indicators. (If allow write is on, I, J, U, or V can be displayed in the A-register.)
- ROS is not used during a Display key operation.

Because certain registers in the 2030 do not have their own console indicators, provision has been made to display these registers in another way. With the CPU clock off, pressing the Display key causes the contents of the register or storage location specified by console switch E (and switches A, B, C, and D if a storage location is specified) to be displayed in a display register.

An additional display function occurs when the I-, J-, U-, or V-registers are selected. If either the I- or J-register is selected and displayed in the A-register, the contents of both the I- and J-registers are transferred to the M- and N-registers so that the entire address is displayed by the main storage address register (M- and N-register) indicators. Similarly, if either the U- or V-registers is displayed via the A-register, then the contents of both U and V are transferred to and displayed by the main storage address register indicators. (Note: This transfer of IJ or UV to MN during display takes place only if the allow write latch is off and the CPU clock is stopped. Allow write must be off in order to change the address in MN).

To use the display feature, first make sure that the CPU clock is stopped. In addition, if a storage position is to be displayed, the allow write latch must be off. (The allow write latch, when on, causes the allow write CPU status indicator, on the lower console indicator panel to light.) Next, set the Display-Store Select switch (switch E) to the register or storage area to be displayed. If a storage area is selected (main storage or auxiliary storage), the storage address must be set up in the main storage address switches (A, B, C, and D). If a register is being displayed, pressing the Display key gates the selected register into the A-register for display. No storage cycle is taken. If the I-, J-, U-, or V-register is being displayed, pressing the Display key gates the selected register to the A-register, and also gates the selected register and its complementing register to the MN-registers for display (if allow write is off). No storage cycle is taken.

If a storage location is displayed, a storage read cycle and a storage write cycle occur. The desired byte is retrieved from storage and placed into the R-register for display (main storage data register indicators). When a storage location is displayed (when a program has been halted), it is a good idea to record the contents of the R-register prior to the display operation. Then, the R-register can, if necessary, be restored before reentering the program and starting the CPU clock.

The selections made with switch E, for display, are listed in Figure 6-11.

ROTARY CONTROL TEST SWITCHES (FIGURE 6-12)

RATE SWITCH

- The Rate switch is a three-position switch with process, instruction step, and single cycle positions.
- If the Rate switch is in the instruction step or single cycle position, the Test light (on OCP) is on.
- The Rate switch controls the rate that the CPU processes instructions.

INSTR (INSTRUCTION) STEP POSITION: When the Rate switch is in the instr step position, one complete instruction, (including all unmasked, pending interruptions) is executed each time the Start key is pressed. When the clock stops after executing an instruction, the B and A-register lights display the address of the next instruction.

If the instruction is an I/O instruction, then the I/O operation (including all associated chaining) is completed before the CPU clock is stopped. This stop is identical to the stop that occurs when the Stop key is pressed.

SINGLE CYCLE POSITION: When the Rate switch is in the single cycle position, the CPU advances by one ROS cycle each time the Start key is pressed. Thus, the CPU processes instructions in .75-microsecond (or 1-microsecond depending on cycle rate of 2030) increments. I/O data-overruns may occur in this mode.

PROCESS POSITION: When the Rate switch is set to the process position, the CPU clock is allowed to run until some condition causes a stop. This is the position in which customers process problem programs.

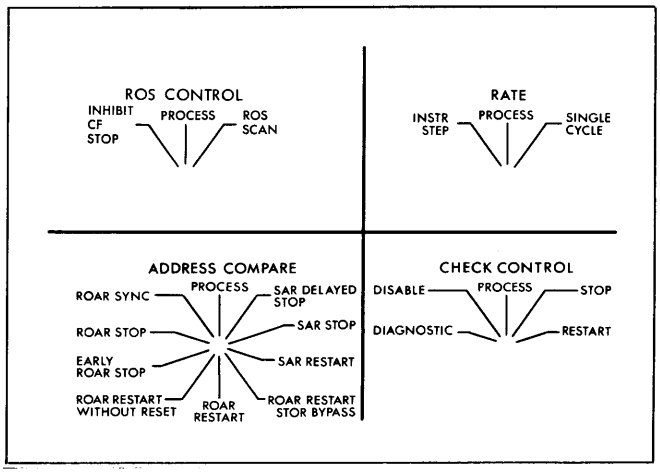


Figure 6-12. Rotary Control Test Switches

ADDRESS COMPARE SWITCH

- The Address Compare switch determines the function to be performed by the address match circuit.
- If the Address Compare switch is at any position other than process, the test indicator (on the OCP) is lighted.
- Switches A, B, C, and D outputs are compared with the contents of either ROAR or SAR as defined by the Address Compare switch.

PROCESS POSITION: This is the position in which customers process problem programs. A sync pulse is generated when the address specified in the address switches (A,B,C, and D) matches an address in SAR.

ROAR SYNC POSITION: This position provides a sync pulse when the address specified in switches A, B, C, and D matches the contents of the read-only-storage address register (WX).

ROAR STOP POSITION: With this setting, the operation proceeds until the contents of the ROAR match the contents of switches A, B, C, and D. When this match occurs, the clock is turned off at the end of the current ROS cycle and the system stops.

EARLY ROAR STOP POSITION: With this setting, processing proceeds until the contents of the ROAR match the contents of switches A, B, C, and D. When the match occurs, the clock is turned off at the end of the current ROS cycle, and the system stops. This function differs from the ROAR stop function in that the indicating ROAR is not set by the contents of WX at T4 time, and the address displayed is the address of the ROS word just prior to the ROS word-address set in switches A, B, C, and D.

ROAR RESTART WITHOUT RESET, ROAR RESTART, AND ROAR RESTART STOR (STORAGE) BYPASS POSITIONS: These three positions are similar in that the occurrence of a match between the ROAR and the switches A, B, C, and D cause the ROAR to be reset to the value set in switches F, G, H, and J. In the case of the ROAR restart position, the CPU hardware registers (except the R-register) are reset to zero before the ROAR is reset to the value in switches F, G, H, and J. In the ROAR restart storage bypass position, operation is similar to that in the ROAR restart position except that main storage is not permitted to operate. Note that a normal problem program cannot be processed if the main storage does not operate. ROAR restart without reset is similar to ROAR restart except that the reset function is blocked.

SAR RESTART POSITION: When a match occurs in this mode, the CPU is reset and a fixed address is forced into the ROAR. The basic microdiagnostic routine and then the clear UCW routine are performed. Then the microprogram loads the contents of switches F, G, H, and J into the instruction counter (registers I and J) and starts an instruction cycle.

SAR STOP POSITION: In this position a match between switches A, B, C, and D and the address in SAR causes the CPU clock to stop at the end of the write cycle in which the match occurs.

SAR DELAYED STOP POSITION: In this position, a match causes the CPU clock to stop at the conclusion of execution of the instruction in which the match occurs. All pending interruptions are taken before the clock is stopped.

ROS CONTROL SWITCH

- The ROS control switch is used for certain FE diagnostic procedures.

INHIBIT CF STOP: In this position, processing occurs in the normal fashion except that microprogram stops (a particular pattern of bits in the CF field) are ignored.

ROS SCAN: This position is used when the microprograms that scan the R/W storage or the ROS are run. The switch performs the following functions (SLD references are in parentheses):

1. Modifies all ROS trap addresses from 0XX to 3XX (01B-C4).
2. Provides a constant reset (04B-E5) to the diagnostic latch. (However, the diagnostic position of the check control switch provides a constant set to the diagnostic latch.)
3. Inhibits certain functions of the introduce-ALU-check latch (06B-E5).
4. Inhibits normal machine reactions to selector channel checks (11A-D6).
5. Resets the introduce-ALU-check latch (06B-E5) when switches F, G, H, and J are gated to the WX register.
6. Turns on the hard-stop latch (03C-E4) with a priority pulse when the rate switch is set to the instruction step position.
7. Prevents the W-register 3-bit from placing the CPU in 1400 compatibility mode (05A-D5).
8. Forces continual multiplexor channel share requests (08D-D2).
9. Inhibits A-register checks (07A-A4).
10. Blocks machine check stop when the suppress-malfunction-trap latch is off (03A-A3).

PROCESS: This position allows normal operation of the ROS. This position is like the equivalent on the other three switches on this panel in that, when not in this position, the test indicator is turned on.

CHECK CONTROL SWITCH

- The Check Control switch determines system action when an error is encountered.
- The test indicator (on the OCP) lights when the Check Control switch is not in the process position.

DISABLE POSITION: In this position, any parity check causes its associated check latch to be set, but otherwise the failure is ignored. Results of program processing may be wrong when updating in this mode.

STOP POSITION: Detection of a parity error in the stop position causes an immediate, unconditional clock stop.

DIAGNOSTIC POSITION: In this position, stopping or ignoring of machine checks is under the control of a latch that can be turned on or off under microprogram control.

RESTART POSITION: Upon detection of an error, action is conditioned by the setting of the Address Compare switch, as follows:

1. With the Address Compare switch in the SAR restart position, a system reset is initiated, the basic microdiagnostic and clear-UCW routines are executed and then the instruction counter is set by the outputs of switches F, G, H, and J, and an I-cycle is started.
2. In the ROAR restart or the ROAR restart storage bypass position, a recycle reset is given which resets circuitry registers only (not the UCW's) and then gates the contents of switches F, G, H, and J to the read-only-storage address register and starts the resulting microprogram (with or without the operation of main storage).
3. In the ROAR restart without reset position, operation is identical to that in the ROAR restart position except that no reset is initiated.
4. In any other position of the Address Compare switch, operation is like that in the SAR restart position except that no reset, no basic microdiagnostic routine, and no clear-UCW routine are initiated.

PROCESS position: This is the position in which problem programs are processed. Upon detection of a parity check with the switch in this position, the ROS automatically initiates what is known as the malfunction trap routine. This routine stores the contents of the machine check register in a fixed location (80, in hexadecimal) of main storage, stores the current program status word, and upon successful completion of these tasks, originates a machine check interruption.

METER PANEL (FIGURE 6-13)

EMERGENCY PULL SWITCH

- When the emergency pull switch is operated, all power in the CPU and all I/O devices is dropped immediately.
- The contents of core storage may be altered if the emergency pull switch is operated.

When the emergency pull switch is pulled, all system power (including that to all on-line I/O units) is dropped without regard to sequencing. Therefore, the contents of main storage may be partially destroyed during an emergency power-off operation.

As the name of the switch implies, emergency power-off should be initiated only under unusual circumstances. Once the Emergency Pull switch is pulled, it is mechanically locked so that system power cannot be brought up again until the Customer Engineer has reset this switch.

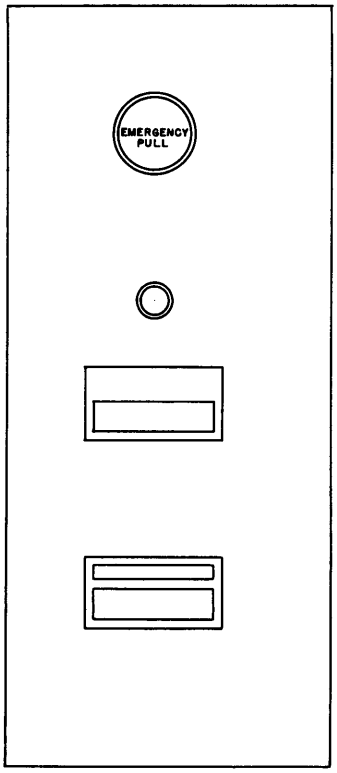


Figure 6-13. Meter Panel

METERING SWITCH

- The metering switch enables one and disables the other use meter.
- The metering switch is operated by a removable key.
- Two positions of the metering switch are:
  1. Normal -- Enable process meter, disable CE meter.
  2. CE -- Disable process meter, enable CE meter.

The 2030 console is provided with two direct-reading meter counters that record operating time: a customer's meter and a Customer Engineer's meter. The position of a Key switch determines whether the customer's meter or the CE meter is operating. The Customer Engineer holds the key for this switch, and whenever he is performing either scheduled or unscheduled maintenance in the CPU, he sets the switch to cause the CE meter to operate. One of these meters (determined by Key switch setting) operates whenever:

1. The CPU clock is running and the CPU is not in the wait state.
2. The metering-in signal is up on an I/O channel.
3. Selector share cycles (selector hold latch) occur.
4. Manual store or display operations are in progress.

The meter, when started is forced to operate for a minimum of 400 milliseconds.

The system indicator is on whenever either meter is running.

APPENDIX A. SYSTEM CHARACTERISTICS

Type	Mod	Description	BTU/Hr	KVA	CFM	Conn Type	Weight	Dimensions (inches) H	F	S	Service Clearances (inches) F	R	Rt	L	Notes
360	30		10,000	3.8	900	B	1,500	60	68	84	42	18	60	30	2
360	40		7,000	2.5	300	D	1,700	60	60	109	48	48	30	72	2
360	44		10,000	4.0	1,600		1,800	67-1/2	102-1/2	72	48	30	30	72	
360	50F,G		14,900	9.0	2,350	E	4,700	72-1/2							4,2
360	50H		18,700	10.6	2,990	E	5,350	72-1/2							4,2
360	50I			13.8		E		72-1/2							4,2
360	65		12,000	6.9	2,100	E	2,400	72-1/2							4
360	75		43,000	12.6	3,350	E		72-1/2							4,2
1015	1	Inquiry Display Terminal	900	0.28	0		375	47	48	29	36	6	36	30	4,6,11
1015	2	Inquiry Display Terminal	900	0.28	0		300	47	48	29	36	6	36	30	4,6,11
1016		Control Unit	1,600	0.5	50	A	200	29	15	37	30	30	30	30	
1051	N1	Control Unit	670	0.2	0	A	195	27	26	15	0	30	36	0	
1052	1	Printer-Keyboard	335	0.1	0		65	9	23	19-3/4	0	0	0	0	12
1231	N1	Optical Mark Page Reader	3,700	1.2	300	A	620	44-3/4	43-1/2	24	42	42	30	36	
1285		Optical Reader	5,000	2.0	600	D	850	60	71-1/4	35-3/4	36	48	42	48	
1403	2,7	Printer	2,500	1.0	310		750	47-3/4	28-1/2	53-1/4	36	36	30	30	3
1403	3	Printer	3,000	1.2	350		750	47-3/4	28-1/2	53-1/4	36	36	30	30	3
1403	N1	Printer	3,000	1.2	350		825	53-1/2	57-1/2	29	36	36	42	42	3
1404	2	Printer	3,800	1.5	280		1,600	53-1/2	67-1/8	31-3/4	36	36	48	42	3
1412	1	Magnetic Char Rdr	6,300	2.7	320	C	2,475	60-1/4	112	41-1/4	42	48	36	36	13
1418	1,3	Optical Char Rdr	8,300	3.8	575	D	2,650	60-1/4	112	41-1/4	42	48	36	36	13
1418	2	Optical Char Rdr	8,300	3.8	575	D	2,700	60-1/4	112	41-1/4	42	48	36	36	13
1419	1	Magnetic Char Rdr	8,500	3.3	400	C	2,675	60-1/4	112	41-1/2	42	48	36	36	13
1428	1,3	Alphameric Optical Reader	10,500	4.6	575	D	2,750	60-1/4	112	41-1/4	42	48	36	36	13
1428	2	Alphameric Optical Reader	10,500	4.6	575	D	2,800	60-1/4	112	41-1/4	42	48	36	36	13
1442	N1	Card Read Punch	1,500	0.7	0	A	575	49	43	24	36	42	0	0	
1443	N1	Printer	3,200	1.1	50	A	800	46	55-7/8	43	36	36	48	30	
1445	N1	Printer	3,200	1.1	50	A	825	46	55-7/8	43	36	36	48	30	
1801		Processor-Controller	8,900	5.2	650	E	1,200	72	57	28	30	30	0	0	
1802															
1827		Data Control Unit	2,000	0.69	490	A	800	72	29	28	30	30	0	0	
2067	1,2	Processing Unit	15,700	8.0	2,700	E	2,992								4
2150	-	Console	1,740	0.65	180	B	800	52-1/4	64	28-3/4	30	48	30	30	
2167	1 to 6	Configuration Unit	8,792	3.5	500	A	583								4
2250	1	Display Unit	7,200	2.8	480	A	590	50							4,6
2250	2	Display Unit	6,600	2.4	320	A	375	50	22	28	30	30	30	30	6
2260	1,2	Display Station	477				45	16	13-1/4	21	0	0	0	0	11
2260	3	Display Station	477				25	16	13-1/4	21	0	0	0	0	11
2280		Film Recorder	36,500	13.3	1,405	E	1,900	70	111		69	48	36	54	4
2281		Film Scanner	36,500	13.3	1,405	E	1,900	70	111		69	48	36	54	4
2282		Film Recorder Scan	36,500	13.3	1,405	E	1,900	70	111		69	48	36	54	4
2301		Drum Storage	3,800	1.5	320		850	64	34-1/2	29	48	48	42	42	7
2302	3	Disk Storage	20,000	9.0	2,210	E	4,025	68-3/4	85-1/2	33	60	60	60	60	2
2302	4	Disk Storage	28,000	12.6	2,210	E	4,425	68-3/4	85-1/2	33	60	60	60	60	2
2311		Disk Storage Drive	2,000	0.75	100		390	38	30	24	36	36	30	30	7
2321	1	Data Cell Drive	19,500	8.7	850	D	1,950	60	68-1/2	50-1/2	30	30	34	30	
2361	1,2	Core Storage	24,600	9.0	1,095	E	2,125	70-1/2							2,4
2365	1	Processor Storage	21,840	8.0	1,055	E	2,200								4
2365	1,2	Storage	33,000	12.5	2,150	E	2,560	72-1/2							5,4
2365	2,3	Processor Storage	34,130	12.5	1,495	E	2,500								4
2365	12	Processor Storage	50,268	18.5	2,345	E	3,950								4
2401	1,2,3	Magnetic Tape Unit	3,500	1.6	500		800	60	30	20	36	36	30	30	7
2402	1,2,3	Magnetic Tape Unit	7,000	3.2	1,000		1,600	60	60	29	36	36	30	30	7
2403	1,2,3	Magnetic Tape Unit and Ctrl	5,500	2.1	1,000	E	2,000	60	60	29	42	42	30	30	
2404	1,2,3	Magnetic Tape Unit and Ctrl	6,300	2.4	1,200	E	2,000	60	60	29	42	42	30	30	
2415	1	Magnetic Tape Unit and Ctrl	10,000	3.25	1,250	D	1,800	60	30	70	36	36	36	36	
2415	2	Magnetic Tape Unit and Ctrl	12,500	4.1	1,500	D	2,300	120	30	70	36	36	36	36	
2415	3	Magnetic Tape Unit and Ctrl	15,000	4.9	1,750	D	2,800	180	30	70	36	36	36	36	
2501	B1,B2	Card Reader	2,700	0.5	0	A	425	44-1/2	30	24	36	42	24	6	

Appendix A

Type	Mod	Description	BTU/Hr	KVA	CFM	Conn Type	Weight	Dimensions (inches) H	F	S	Service Clearances (inches) F	R	Rt	L	Notes
2520	B2, B3	Card Read Punch	6,350	1.85	75	A	660	50	43	24	48	36	18	36	
2540		Card Read Punch	3,000	1.2	50		1,050	45-1/4	57-1/2	29-1/4	36	36	36	36	3
2671		Paper Tape Reader													10
2701		Data Adapter Unit	1,200	0.3	120	A	320	40	40	25-1/2	42	42	42	0	
2702		Transmission Ctrl	1,800	2.0	800	A	900	60	28-3/4	61-1/2	30	18	42	30	
2703		Transmission Ctrl				E		32	68	71	30	36	66	66	4
2802		Hypertape Ctrl	1,360	0.6	300	F	928	60	28-3/4	61-1/2	30	30	42	42	2
2803		Tape Ctrl	2,500	1.0	500	E	1,400	60	60	29	42	42	30	30	
2804		Tape Ctrl	4,000	1.5	700	E	1,600	60	60	29	42	42	30	30	
2814	1,2	Switching Unit	750	0.15	120	A	320	40	40	25-1/2	42	42	30	0	
2816	1,2	Switching Unit	1,500	1.2	280	A	500	60	29	42	30	18	30	42	
2820		Drum Storage Ctrl	4,000	1.5	550	D	750	60	28-3/4	61-1/2	30	30	30	42	2
2821	1,2,4	Control Unit	7,000	2.4	300	D	1,000	60	32	46	30	18	48	48	
2821	3,5	Control Unit	14,000	4.8	600	E	2,000	60	32	93	30	30	48	48	
2822		Paper Tape Rdr Ctrl Unit	1,700	2.05	150	A	400	40	30	26-1/4	30	30	30	30	
2840		Display Control	4,800	1.4	300	D	550	60	32	61-1/2	30	30	30	30	
2841		Storage Control	5,500	1.9	1,000	D	750	60	32	45-1/2	30	30	30	40	
2846		Channel Controller	5,200	1.5	900	A	2,000								4
2848		Display Control	3,542	1.5		C	1,000	71-1/4	32-1/4	61-1/4	30	30	48	48	
2860	1	Selector Channel	8,200	3.05	420	B	1,150	71	32-1/4	67-3/4	30	51	66	66	
2860	2	Selector Channel	10,000	3.65	740	B	1,450	71	32-1/4	67-3/4	30	51	66	66	
2860	3	Selector Channel	11,600	4.25	1,060	B	1,750	71	32-1/4	67-3/4	30	51	66	66	
2870		Multiplexor Channel	11,600	4.25	1,060	B	1,450	71	32-1/4	67-3/4	30	51	66	66	
7320		Drum Storage	2,800	1.1	320	D	850	60	30	29	40	40	42	42	
7340	3	Hypertape Drive	12,000	4.0	700		1,500	48	29	60	46	52			7,8
7404		Graphic Output Unit	3,000		200		800	81	50	18	42	30	30	30	7
7634		Graphic Control Unit	6,000	2.5	500	D	540	70	37-1/2	31-1/2	42	36	30	30	
7770	3	Audio Response Unit				C		70	37-1/2	31-1/2	42	36	30	30	4
7772	3	Audio Response Unit				A		70	37-1/2	31-1/2	42	36	30	30	4

NOTES:

1. For airflow, see specifications page for 2302 Disk Storage.
2. This unit is equipped with radio interference control circuitry and requires a good wired earth or building ground. Total resistance of the ground conductor, measured between the receptacle and the building grounding point, may not exceed 3 ohms. For proper operation, all components of the system or systems to which this unit is attached must have the same ground reference. Conduit is not a satisfactory means of grounding.
3. Powered from 2821.
4. For data, see specifications page for that item.
5. See System/360 specifications page for this data.
6. It is recommended that in the area immediately surrounding this unit provision be made for lowering the lighting level to provide good image resolution.
7. Powered from control unit.
8. Minimum clearance for two 7340 units is 7 inches; clearances should alternate: 7, 22, 7, and 22 inches. Clearance between 7340 and any other unit or structure is 30 inches.
9. Shipped in two sections, 50-1/8 inches and 35-3/8 inches long.
10. Included in specifications for 2822.
11. Available for remote installation only.
12. Powered from System/360.
13. Shipped in two sections 40 and 72 inches long.

Type	Plug	Connector	Receptacle	Rating
A	Russell & Stoll, FS3720	FS3913	FS3743	15 amp, 1 phase, 3 wire
B	Russell & Stoll, FS3730	FS3914	FS3744	15 amp, 3 phase, 4 wire
C	Russell & Stoll, FS3750	FS3933	FS3753	30 amp, 1 phase, 3 wire
D	Russell & Stoll, FS3760	FS3934	FS3754	30 amp, 3 phase, 4 wire
E	Russell & Stoll, SC7328	SC7428	SC7324	60 amp, 3 phase, 4 wire
F	Russell & Stoll, JPS1034H	JCS1034H	JRS1034H	100 amp, 3 phase, 4 wire

For additional information regarding physical characteristics, refer to IBM System/360 Installation Manual--Physical Planning, form C22-6820.

APPENDIX B. SPECIAL CIRCUITS

The information contained in this section is intended to aid the CE in understanding those special circuits that appear in the 2030 ALD's.

These special circuits are non-standard logic blocks with unique input or output configurations that require further explanation.

M2-I VOLTAGE REGULATOR CARDS (U25AH, U25AG)

- Two cards, voltage regulator card 1 and voltage regulator card 2, are used together to develop a regulated -18 volt supply from the -30 volt power supply.
- A potentiometer on voltage regulator card 2 provides adjustment of output voltage.

The sense amplifiers for the M2-I storage require a well-regulated, -18 volt supply voltage. This voltage is derived from the -30 volt supply in the 2030 by means of two voltage regulation cards (Figure B-1). The output of the cards is -18 volts ±1% regardless of wide swings in input supply voltage.

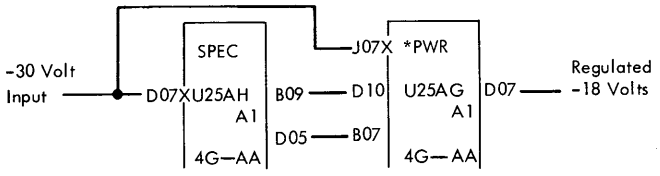


Figure B-1. M2-I Voltage Regulator

Voltage regulator card #2 (U25AG) is the basic card containing the regulator circuit and the adjustment potentiometer. For storage units above 16K in size, an additional card (U25AH) is necessary to carry the current required for the additional sense amplifiers. Voltage regulator card #1 adds two transistors in parallel with the two controlled transistors on card #2.

The potentiometer on card #2 is part of a voltage divider that sets a referance voltage for the base of a control transistor. Changing the potentiometer setting changes the referance voltage, and thus changes the current through the control transistor.

The sense amplifier voltage is set to -18 volts, ± .01 volts accuracy, under controlled conditions at the factory before shipment. The potentiometer must not be adjusted in the field unless the card (U25AG) is replaced. If it is necessary to verify or set this voltage, a very accurate meter (such as the Weston 901) must be used. An example of these two cards may be seen on logic page ZZ502.

CCROS DRIVER DECODE (T11EE)

- The T11EE block contains eight drive transistors.
- Each drive transistor drives two ROS words.

The T11EE block is a multiple input/multiple output logic block containing eight CCROS driver transistors (Figure B-2). Eight of the input lines connect to the bases of the drive transistors and feed address information to the drive transistors. A ninth address line input connects to the emitters of all eight transistors to form a matrix with the other eight address lines. An additional input line provides the transistors with a special +12 volt collector voltage supply.

The eight outputs are tapped from the eight drive transistor collector load resistors. Given an address that falls within a group of eight drive transistors, one output line activates to provide a

drive pulse to complementary ROS word positions on a ROS board.

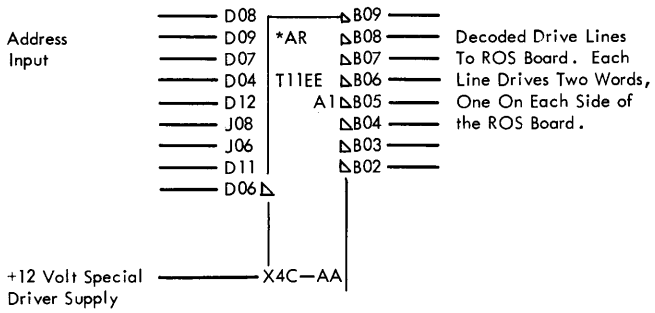


Figure B-2. CCROS Driver Decode

M2-I Z-CURRENT DRIVER (U61AX)

- This block provides inhibit drive current for one 4K inhibit winding.
- This block has two standard-level logic inputs.

The inhibit current driver provides inhibit current for one inhibit winding. (One winding goes through 4096 cores.) There are two logic inputs to the U61AX block; the remaining inputs provide resistive and capacitive load characteristics necessary for the inhibit winding (Figure 3-3). These non-logic inputs are identified by the X in the side of the block. One input provides connection to the special -30 volt power supply.

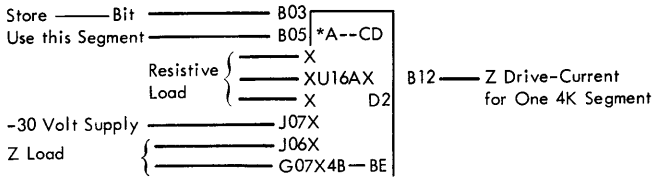


Figure B-3. Z-Current Driver

CCROS SENSE AMPLIFIER (S07EG)

- This block provides the first level of amplification for the CCROS-output voltage-pulse.
- One input is the logic voltage input; the remaining inputs provide reference voltage and impedance matching.

This block represents a 5-transistor voltage amplifier that senses the voltage pulse at the output of a CCROS sense line. The bit input to the block is the only logic input. The other inputs are non-logic inputs, identified by the X in the side of the block (Figure B-4). A sample of this block may be seen on logic page ED521.

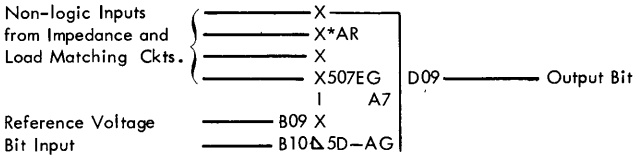


Figure B-4. CCROS Sense Amplifier

M2 GATE TRANSISTOR (S32EB)

- These blocks are used to select one core storage drive line and to permit drive current to flow through the selected line.

- Similar circuits using the same gate block are used for both X- and Y-drive lines.

The S32EB gate block serves as a multiple AND block by decoding address information. This address information conditions the bases and emitters of the gate transistors which are connected to form a matrix. The eight plus-level logic inputs condition the bases of eight NPN gate transistors, while the minus logic input connects to the emitters of all eight gate transistors. Thus the minus input combined with any plus input selects one line (Figure B-5).

An additional input, through a diode (S25EE), provides a path for current when a line is driven from a gate transistor at the other end of the lines. A similar gate circuit (S32EC) is used for the auxiliary storage areas. This gate circuit selects one of two lines instead of one of eight lines. A sample of the S32EB gate may be seen on logic page MS411, while the S32EC may be seen on logic page MS441.

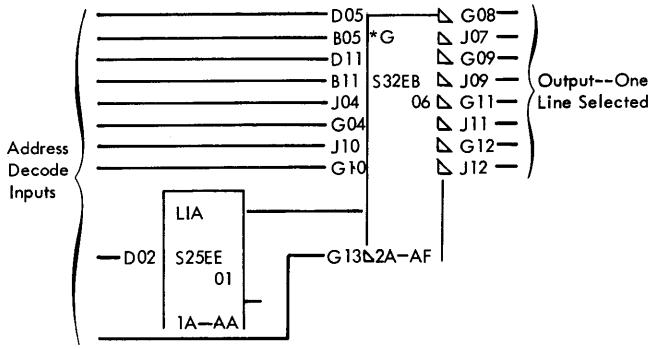


Figure B-5. M2 Gate Transistor

M2 GATE DECODE (S32AD)

- This block provides one level of decode in the M2 storage unit.
- A non-logic output line provides a clamped resistive load.

This block performs a standard AND function of the decoded bit inputs (Figure B-6). A unique non-logic output pin provides a path to the plus voltage supply through a resistive load. The output voltage swing is controlled by a diode in the LIM block.

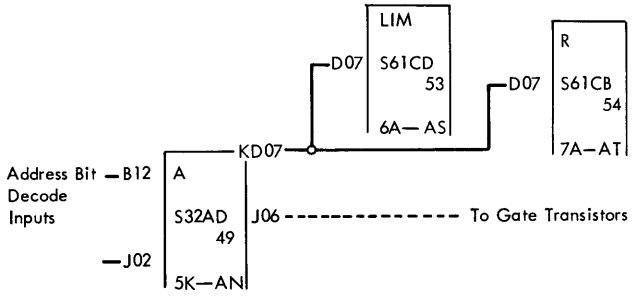
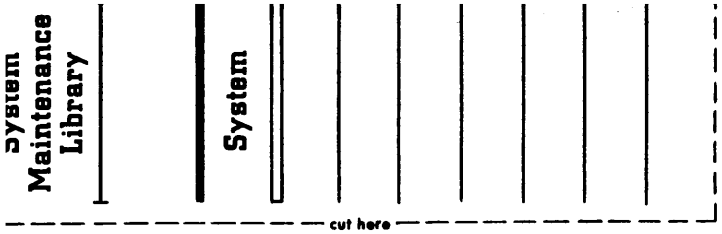


Figure B-6. M2 Gate Decode



Y24-3360-1

IBM 2030 Printed in U. S. A. Y24-3360-1

IBM

International Business Machines Corporation

Field Engineering Division

112 East Post Road, White Plains, N. Y. 10601