IBM®

Field Engineering

Manual of Instruction

2030 Processing Unit

System/360 Model 30

PREFACE

This manual contains information about the IBM 2030 Processing Unit. A companion manual on input/output control should be obtained for information pertaining to the attachment of I/O devices to the IBM System/360 Model 30. The companion manual is the _IBM 2030 I/O Control Field Engineering Manual of Instruction_, Form 225-3362.

Minor Revision, August 1965

This edition, 225-3360, is identical in content to the previous edition, Z25-3360. The IBM Confidential classification has been removed and the form number has been changed to allow free access to the manual.

Address comments concerning the content of this publication to IBM Product Publications, Endicott, New York 13764.

CONTENTS

SYSTEM CONFIGURATIONS

- Single system concept.
- Different models provide a variety of processing speeds and storage sizes.
- Broad range of input/output devices.
- Uses an 8-bit coding structure to represent data.
- Program compatibility throughout system models.

Behind the decision to design the IBM System/360 - a single system which encompasses all areas of data processing lies the awareness that apparently unrelated applications have more similarities than differences. For example, because of teleprocessing and other factors, scientific applications require high-speed input/output similar to that required by commercial applications. In addition to this versatility, the System/360, because of its modularity, adaptability and compatibility, can handle the many kinds of growth that normally occur in computer installations.

Modularity in the System/360 is achieved through the availability of seven models. A typical Model 30 system is shown in Figure 1-1. In addition to a choice of processing speed through model selection, each model offers a choice of storage capacities.

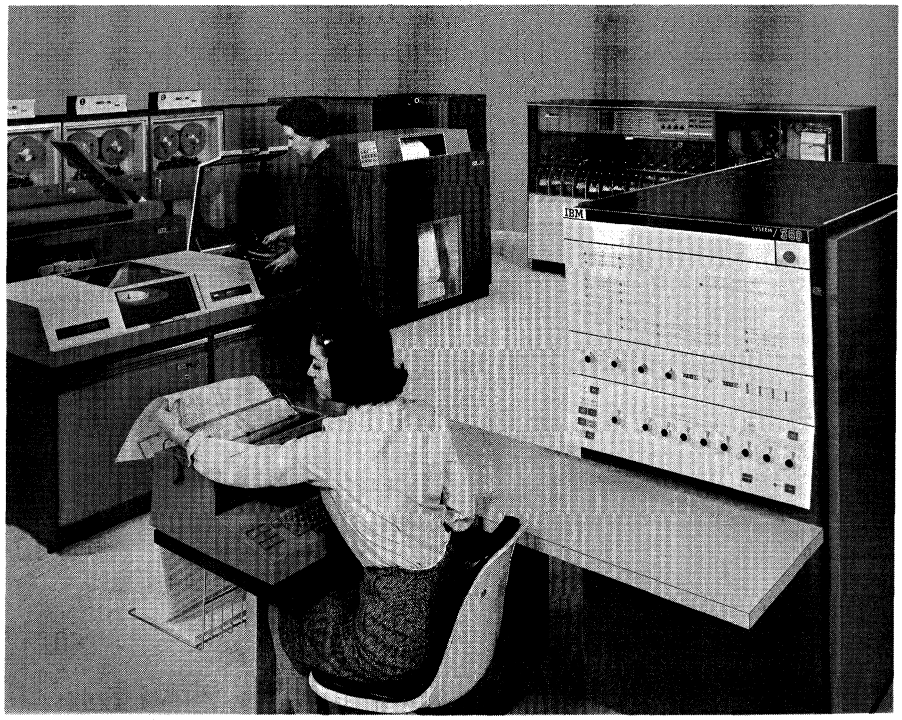


Figure 1-1. IBM System/360, Model 30

As problems and workloads grow or change, the System/360 can easily be expanded or changed to handle additional or different operations. Storage can be added and input/output and processing speeds increased -- in small increments, as needed. This adaptability makes provision for the inclusion, either initially or subsequently, of a broad range of input/output devices.

Versatile performance characteristics permit handling of data in virtually any desired character representation. Instead of the usual six-bit character, the System/360 employs a new eight-bit coding structure to represent data. An 8-bit unit of data is called a byte. This 8-bit coding structure allows 256 possible combinations for letters, digits and symbols, providing greater versatility in both binary and decimal operations. It also means that System/360 can accept a character code of fewer bits, such as a telegraph code.

More important than the modularity and adaptability of the System/360 is its compatibility. A program written for one configuration will run on any other, if there is enough memory capacity and input/output equipment, and if the program is not geared to the operating speed of any particular unit. Subject to these constraints, a program written for a smaller System/360 will run without modification on a larger one. While this "upward" compatibility is certainly an advantage, "downward" compatibility can be even more valuable; for example, a small user can utilize programs written for larger systems. This places a total library of programs at all users' disposal.

Compatibility further allows a System/360 to be tailored to fit either centralization or decentralization. That is, a company's installation can be either a large central processor or a number of smaller processors. Shifts between the extremes are possible within the same system.

Traditionally there have been constraints on computer versatility, so that one processor has lent itself to scientific and engineering application, another to commercial data processing applications, another to process control, and still another to communications.

The System/360 provides a versatile set of instructions that permit operating quickly and efficiently regardless of the system application.

MODELS AND SPEEDS

- Different processor models provide a variety of processing speeds and core storage sizes.
- The 2030 is the processor for the System/360, Model 30 (Figure 1-2).
- The 2030 is available in four core storage sizes, represented as System/360, Models C30, D30, E30, and F30.

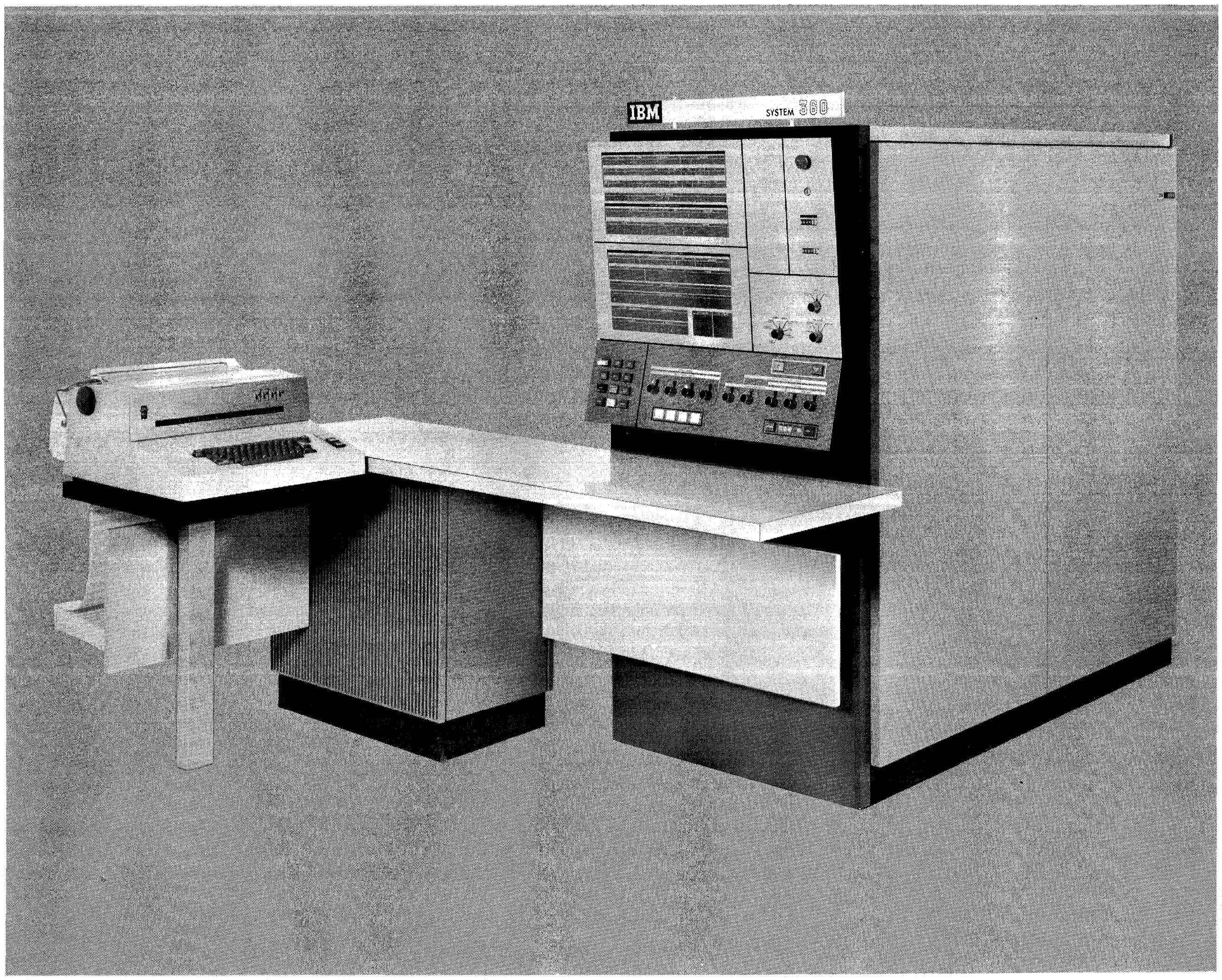


Figure 1-2. IBM 2030 Processing Unit

To fit the widely varied cost and volume needs of all computer users, the IBM System/360 is available in many different models. For instance, to fit the needs of the user who needs a minimal number of answers per month, a Model 30 is available at a minimal cost. For the user who needs a greater number of answers per month, a Model 70 is available that will give approximately 50 times as many answers per month as a Model 30. The answers will be the same; only the number of answers in a given period of time will be different. There are two basic differences between models: core storage capacity and internal processing speeds. Figure 1-3 shows the core storage capacities for the different System/360 models. Internal processing speed is largely dependent on the speed of the core storage unit and the amount of data involved on each core storage access. Core storage speed in the System/360 varies from 2 microseconds per single byte access on the Model 30's to 1 microsecond per eight-byte access on the Model 70's (Figure 1-4).

Storage Capacity in Bytes	System/360 Models					
524,288	▨	▨	▨	I60	I62	I70
262,144	▨	H40	H50	H60	H62	H70
131,072	▨	G40	G50	G60	▨	▨
65,536	F30	F40	F50	▨	▨	▨
32,768	E30	E40	▨	▨	▨	▨
16,384	D30	D40	▨	▨	▨	▨
8,192	C30	▨	▨	▨	▨	▨

Figure 1-3. System/360 Storage Sizes

Processor	System/360 Models	Bytes/ Access	Memory Speed
2030	C30, D30, E30, F30	1	2.0 microseconds
2040	D40, E40, F40, G40, H40	2	2.5 microseconds
2050	F50, G50, H50	4	2.0 microseconds
2060	G60, H60, I60	8	2.0 microseconds
2062	H62, I62	8	1.0 microseconds
2070	H70, I70	8	1.0 microseconds

Figure 1-4. System/360 Storage Access and Speed

GENERAL DATA FLOW

- The processing unit controls the system.
- Information enters the system from an input device.
- The information is manipulated by the processing unit to develop the required answers.
- The answers are sent to an output device to be stored.

In the System/360, Model 30, the 2030 processing unit provides all system control. The processing unit is given instructions by a programmer. These instructions are interpreted and executed by the processing unit. Execution of an instruction might involve adding two numbers together, or it might involve causing a printer to print a check. Regardless of the instruction, the interpretation and control lies in the 2030 processing unit.

General data flow is divided into three operations (Figure 1-5). First, information comes into the processing unit via some input device. This information is then used along with input information from other devices and constant information contained in main storage to develop the required result or output. This output is then sent to an output device where the resultant information is stored. The information storage may be a printed report, punched cards, or magnetized spots on a reel of magnetic tape.

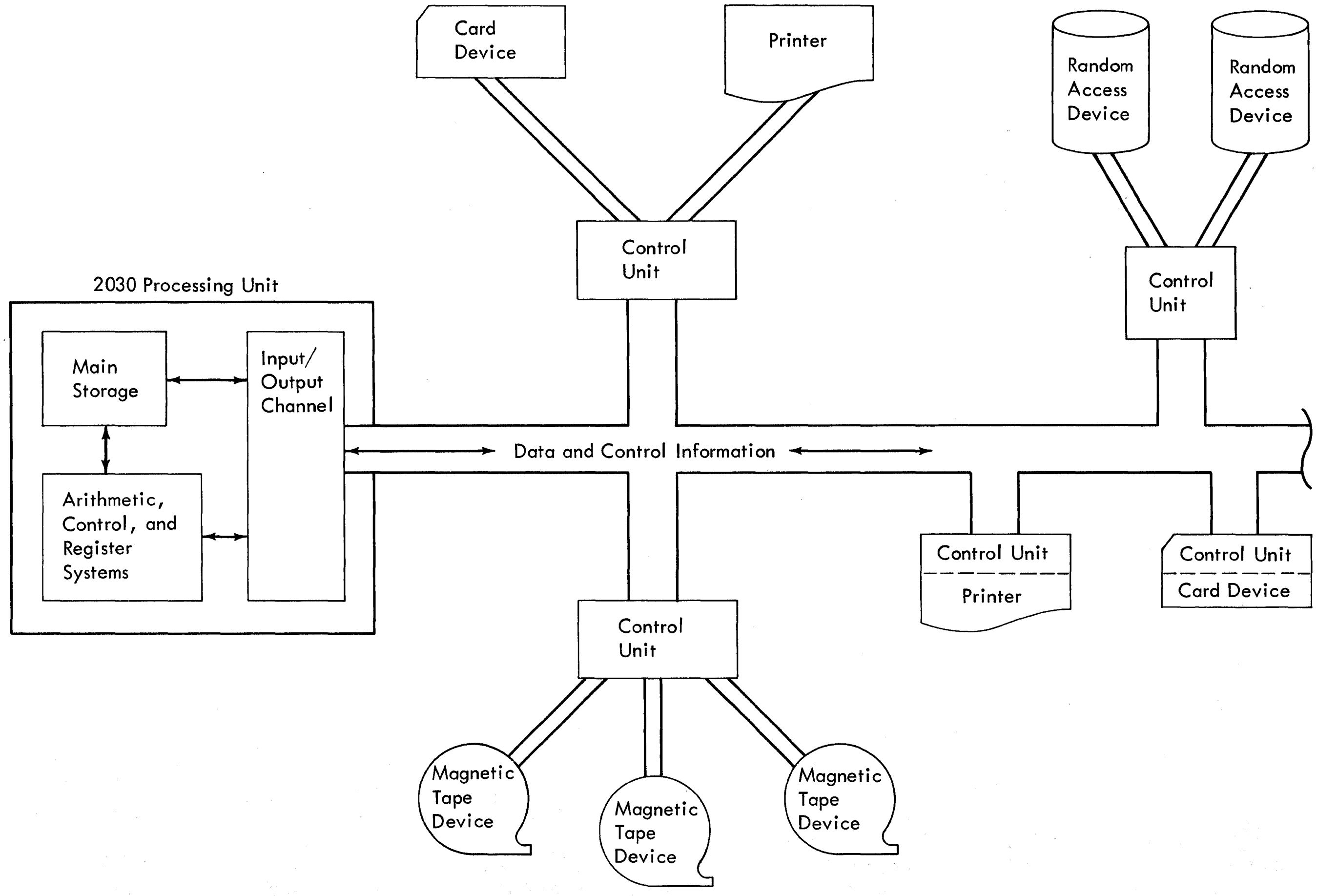


Figure 1-5. General Data Flow

SYSTEM CONCEPTS

- Programming Systems support of the IBM System/360 is called Operating System/360.
- The Operating System/360 supports the Computing System/360. Together they make up the IBM System/360.

The introduction of the IBM System/360 marks the achievement of a truly all-purpose computer that can solve any type of data-handling problem with greater speed and efficiency than ever before. This opens up greatly increased computer potential in every area.

In order to realize this potential, it was apparent to the designers that the programming support needed to be as powerful and as extensive as the computer. In fact, programming support should make the IBM System/360 an even more powerful system.

This lead to the concept of the Computing System/360 (hardware) being supported by the Operating System/360 (programming), together making up the System/360.

```
┌──────────────────────────┐
│                          │
│ ┌──────────────────────┐ │
│ │Computing System/360  │ │
│ └──────────────────────┘ │
│                          │
│  Operating System/360    │
└──────────────────────────┘
       IBM System/360
```

This means that a customer is getting a system that is powerful and flexible, yet, due to extensive programming support, he can easily apply his problem to the System. For one thing, he can write problem solving programs without the necessity of translating them into a language understandable by the machine. Cnce written, the operation of his program is controlled or supervised by Cperating System/360, relieving the operator of many tasks and increasing the utilization of the Computing System/360.

CPERATING SYSTEM/360 CCNCEPTS

- Control programs allowing monitored operation of a system have been proven by experience to produce optimum computer utilization.
- Cperating System/360 includes both control programs, and IBM and user-written processing programs.
- Basic Programming Support programs will be provided for System/360 systems with 8K bytes of storage.

As stated previously, IBM's single system approach with the System/360 recognizes that computing systems and programming systems should be integrated and not developed independently. Experience in the past decade has proved that the optimum method of producing this result is with monitored operation.

Early monitors were designed to minimize human intervention. The new and sophisticated control techniques included in programming systems with the System/360 extend its capabilities so that the monitor and control functions make up what is called an operating system.

The basic purpose of Operating System/360 is to permit the user to solve problems and process information effectively. Included in Operating System/360 are both processing and control programs. Processing programs include all application-oriented programs, including both IBM and user-written.

For systems having 16K bytes of main storage, basic control program functions will be supplied with magnetic tapes or direct access devices. Additional capability can be utilized as more main storage is added.

For systems having 8K bytes of main storage, programming systems is supplying basic programming support programs, which perform many of the functions of operating systems, including control functions.

Control Programs

- Control programs perform functions such as control of administrative operations, job flow control, Input/Cutput control, and program execution control.

A basic key to achievement of high operating efficiency in a computing or data processing installation is a good control procedure. This procedure must include many functions: administrative control of job schedules, workflow, and computer usage records; control over data and program libraries; control over computer operations; and control over the flow of programs and data within the computing system during job runs.

The control programs for the IBM System/360 set up a comprehensive control framework to assist the user in satisfying the above objectives. The control programs operate at various levels of concept. For example:

1. Operations control of installation and administration and workflow, including instructions from and to the computer operator, administrative records, logs of system operation, and control over library programs.
2. Job flow control, including I/C transition between jobs and job segments, unit assignments, initial loading and initialization when the computer is first turned on, control between jobs, and control over the type of operation mode, ranging from simple stacked jobs through teleprocessing systems performing concurrent operations.
3. Input/Output control, including physical and logical control over I/C records, files and units; buffer control; teleprocessing terminal and message handling; random access I/C

control; labeling of files, and error recovery procedures.

4. Program execution control that manages the flow of program instructions from one routine to another, including the instantaneous transitions that take place when any interrupt occurs, the decisions that control the next program series to be executed, the return of control to an interrupted program, storage allocation and protection, diagnostic programs, program loading, and management of the interval timer.

Processing Programs

- Processing programs function under control of operating system control programs.
- Some IBM supplied processing programs are, the System/360 Assembler, FORTRAN, new programming language, COBOL, report program generator, utility programs, and sort/merge programs.

Complementing the control programs and functioning under them are those programs necessary to handle users' specific data processing needs. These programs, known collectively as processing programs, include application programs both IBM and user written, compilers, Report Program Generators, sort/merge, and utility programs.

Symbolic programming languages and the programs that translate them (assemblers and compilers) offer valuable aids to the programmer in solving data processing problems.

The System/360 Assembler language is a symbolic language that permits the coding of source programs in convenient, report program generators, sort/merge, specialized language, it can be used in all kinds of applications, including both commercial and scientific.

The FORTRAN language allows the programmer to code a mathematical or scientific problem in terms closely resembling those he uses in stating the problem mathematically.

The new programming language has some features that are characteristic of FORTRAN and incorporates some of the best features of other languages, such as extensive editing capabilities, to take advantage of recent developments in computer technology.

The COBOL language provides a convenient method of coding programs in a form closely resembling the English language, using the method sponsored by the Conference on Data Systems Languages (CODASYL), a cooperative effort by a number of computer manufacturers and users.

The report program generator (RPG) provides a convenient method for producing a wide variety of reports, using IBM-provided coding forms.

Utility programs provide the user with standard, efficient handling of routine operations involving data transfer between I/O devices. These include such operations such as card to printer, card to punch, card to tape, tape to tape, tape to punch, tape to disk, and many others.

The sort/merge program is designed to satisfy the sorting and merging requirements of all tape-oriented or random storage-oriented IBM System/360 installations. It is a generalized program that can produce many different sorting/merging programs in accordance with control information specified by the user.

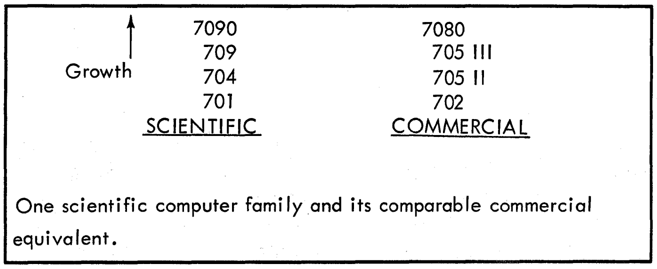
COMPUTING SYSTEM/360 CONCEPTS

- System/360 uses binary and BCD.

- System/360 uses variable and fixed length fields.

- System/360 uses a new technology called solid logic technology.

The System/360 is a general purpose computer system. By this we mean it is designated to be used for commercial, scientific, and communications applications. In the past, these applications were handled by separate computer families (Figure 1-6).



↑	7090	7080
Growth	709	705 III
	704	705 II
	701	702
	SCIENTIFIC	COMMERCIAL

One scientific computer family and its comparable commercial equivalent.

Figure 1-6. Commercial vs. Scientific Computers

The scientific computers were usually fixed word length machines and used a pure binary form of coding. On the other hand, the commercial computers were usually variable word length (character oriented) machines and used a binary coded representation of decimal information. The System/360 uses binary as well as BCD and has both fixed and variable length fields.

The System/360 also uses a new technology known as solid logic technology (SLT). It consists of printed circuitry instead of physical wiring on the back panel. It also uses packaged logic circuits. This new technology reduces manufacturing costs, increases reliability and reduces maintenance time.

In Figure 1-7, you can see the components that make up a data processing system.

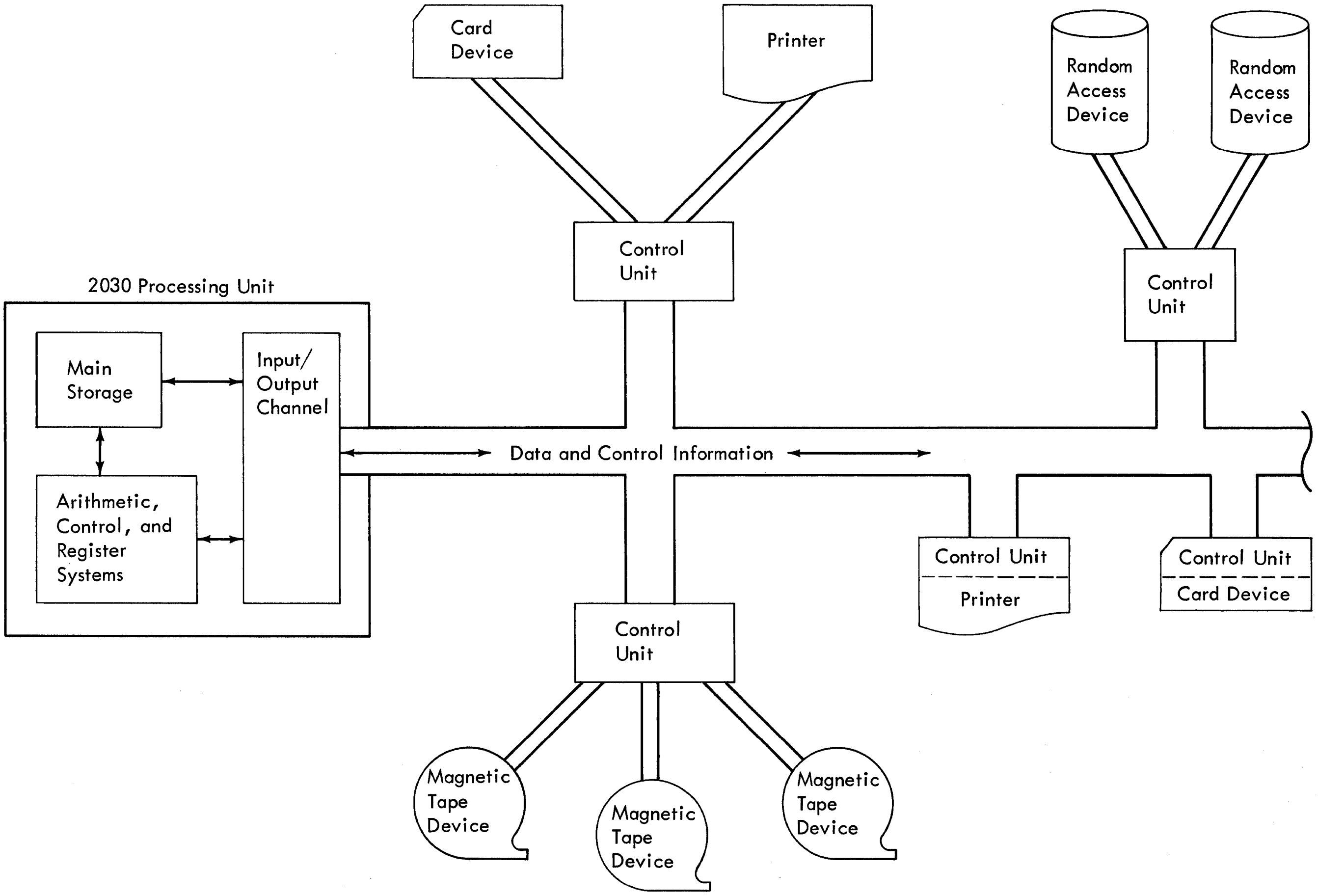


Figure 1-7. Typical Data Processing System

Primary Storage

- The smallest main storage addressable unit is called the byte.

- A byte consists of 8 data bits and 1 parity bit. Cdd parity is maintained.

- The System/360 uses a 24 bit binary address.

- A byte can represent characters, binary numbers, or many different codes.

- A half word is 2 bytes.

- A word is 4 bytes.

- A double word is 8 bytes.

- Data can be fixed length (2, 4, or 8 bytes) or variable length (up to 256 bytes).

- Fixed length data must reside on the correct boundaries in main storage.

- A program check occurs if the boundary restriction is violated.

The primary storage is that section of a computing system that contains the program to be executed as well as the data to be processed. All data entering the system goes into the primary storage before it can be processed. After processing, the data must be placed back into primary storage before it can be sent to an output device.

Primary storage is sometimes referred to as main storage. The System/360 also uses ferrite cores for its main storage.

The smallest addressable unit of main storage in the System/360 is called the byte. The byte consists of eight data bits and one parity bit.

P	0	1	2	3	4	5	6	7

The Byte

As can be seen in the preceding example, the left-most bit of a byte is the parity bit. The IBM System/360 maintains odd parity for all bytes in main storage. The parity bit is added or removed to make the total bit-count odd for any byte. This method of coding provides convenient error checking: an even number of bits indicates an error condition.

One thing you should get clear is that the byte is the smallest addressable unit of main storage. This means that each and every byte of main storage is individually addressable. To read out the first eight bytes of main storage, the Model 30 takes eight storage cycles. For each cycle, the Model 30 changes its storage address by one, using addresses 0-7.

Main storage addresses start with 00000 for the first byte and increase by one for each byte in the particular main storage unit. Valid storage addresses for a Model 30 would start with 00000 and continue up to 65535. To allow for program compatibility as well as for future growth, the System/360 uses a 24 bit binary address to address main storage. age. A 24-bit binary number allows us to go as high as 16777215 for an address. You can see the future growth that is possible here! A binary rather than a binary coded decimal address is used because it is more efficient with large addresses.

The 24-bit binary address that would be used to address byte location 0007 is written 000000000000000000000111.

You are probably a little perplexed about this byte by now. You know that a byte consists of eight data bits and a parity bit. You know that each byte is individually addressable by a 24 bit binary address! You know that main storage size can vary from approximately 8K bytes on a Model 30 to over 500K bytes on a Model 70. You know that the Model 30 has access to one byte per storage cycle.

However, you are probably asking yourself:

Is the byte a character?

Is it a binary number?

Just what is it?

The answer to these questions is simple. The eight data bits of a byte can be coded to represent characters, binary numbers, or anything you want it to be. The instructions of the System/360 are many and varied. Some of the instructions treat bytes as characters. Some instructions treat bytes as part of a binary number. So the answer to the question, "What does a byte represent?" is: It depends on the particular instruction being executed at the time. This question will be answered more to your satisfaction after you study the data formats and some of the instructions.

As was previously stated, the System/360 is a general purpose data processing system. As such it is designed to operate with fixed length as well as variable length data. The byte as you have already learned is a very versatile unit. It is individually addressable. By further specifying the number of desired bytes, we can have a variable length field in main storage starting and ending at any byte address.

To be truly general purpose, the System/360 must also be capable of operating with fixed length data. Whereas variable length data has a variable number of bytes, fixed length data always has a fixed number of bytes. Let's go on and define these fixed length fields.

A half word is two bytes in length (Figure 1-8a).

The data bit positions of a half word are numbered 0-15 from left to right (Figure 1-8b).

Notice that the parity bits are not shown. They will not be shown form here on, since they do not represent data. Remember, however, that every byte does contain a parity bit for checking purposes.

A word is 4 bytes long (Figure 1-8c).

The data bit positions of a word are numbered 0-31 from left to right (Figure 1-8d).

A double word is 8 bytes long (Figure 1-8e).

The data bit positions of a double word are numbered 0-63 from left to right (Figure 1-8f).

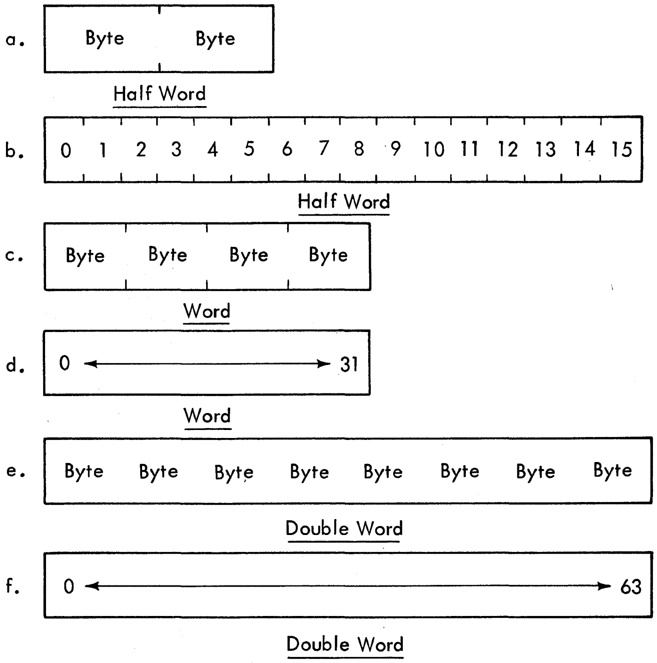


Figure 1-8. Word Formats

Remember that each byte of a half word, word, or double word carries its own parity bit.

Remember also that it is the instruction being executed that determines whether to consider data as variable or fixed. The instruction also determines in the case of fixed length data whether it is a half word, word, or double word.

Before leaving the definitions of fixed length data, you must learn the restrictions placed on the use of fixed length data.

<table>
<tr><td>Byte 0000</td><td>Byte 0001</td><td>Byte 0002</td><td>Byte 0003</td><td>Byte 0004</td><td>Byte 0005</td><td>Byte 0006</td><td>Byte 0007</td><td>Byte 0008</td></tr>
<tr><td colspan="2">Half Word</td><td colspan="2">Half Word</td><td colspan="2">Half Word</td><td colspan="2">Half Word</td><td></td></tr>
<tr><td colspan="4">Word</td><td colspan="4">Word</td><td></td></tr>
<tr><td colspan="8">Double Word</td><td></td></tr>
</table>

Figure 1-9. Boundary Restrictions

The rule is that fixed length data must reside on the correct boundaries in main storage (Figure 1-9).

Fixed length data is addressed by the high order (left-most byte) of the field.

For half words this address must be divisible by two.

For words this address must be divisible by four.

For double words this address must be divisible by eight.

Another way of stating this rule is to say the 24 bit binary address:

1. Of a half word must have one low-order zero bit.

2. Of a word must have two low-order zero bits.

3. Of a double word must have three low- order zero bits.

The boundary restriction placed on the use of fixed length fields is a restriction placed on the user. Violation of these rules does not produce a machine check. Instead, violation of these rules is considered a program check.

Because there are other restrictions placed on the programmer, you should be able to identify program checks by type. The type of program check caused by a violation of fixed length boundaries is known as a specification exception.

Another exception to valid programming is addressing a byte location that is not available on your particular model of System/360. The largest size main storage that is available on the Model 30 is 65,536 bytes. Any address other than 00000-65535 results in a program check. This type of check is known as an addressing exception.

Central Processing Unit (CPU)

- The two main sections of the CPU are (1) the control section and (2) the arithmetic and logical unit (ALU).
- The CPU uses variable field length instructions for storage to storage operations.
- Variable fields can be up to 256 bytes long.
- The CPU uses fixed length instructors for storage to register or register to register operations.
- Fixed length fields can be half-word, word, or double-word fields.
- Register-to-register or storage to register operations use any of 16 general purpose registers.
- Floating point operations use any of 4 double-word floating-point registers

In Figure 1-10, you can see the logical structure of the CPU for the System/360 and its relationship to the main storage.

There are two main sections in CPU. They are (1) the control section and (2) the arithmetic and logical unit (ALU).

From Figure 1-10, you should be able to see some of the functions of the control section. They are:

1. All references to main storage, whether for instructions or for data, are made by the control section.
2. During the first part of any instruction, the control section addresses main storage and causes the instruction to be fetched and sent to the control section. The instruction is then decoded by the control section and executed during the latter part.

The ALU contains the circuits necessary for adding and comparing data fields as well as the other circuits necessary for operating on data fields.

As can be seen from Figure 1-10, the ALU can do:

1. Variable field length operations.
2. Fixed-point operations involving fixed-length fields.
3. Floating point operations.

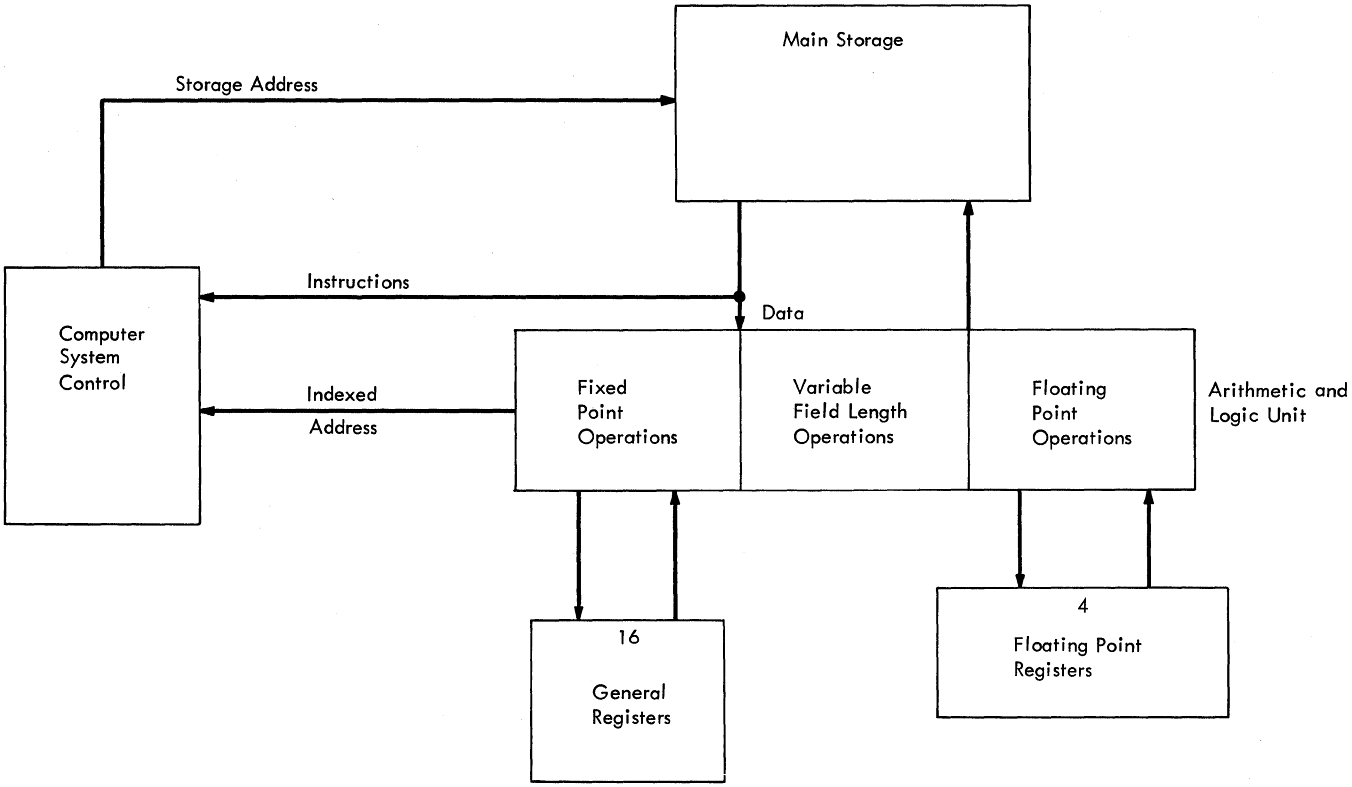


Figure 1-10. Processing Unit Logic Flow

Variable Field Length Operations: In looking at the ALU, let us first consider variable length fields as used in many commercial computers of the past. Two main concepts were used. The storage-to-storage concept was used by computers of the IBM 1401 family. In it the data fields were brought out of main storage, operated upon, and the results went back into main storage (Figure 1-11).

Other computers such as those of the IBM 702-705 family used a storage-to-accumulator concept. The accumulator was a small storage device. The storage medium could be core

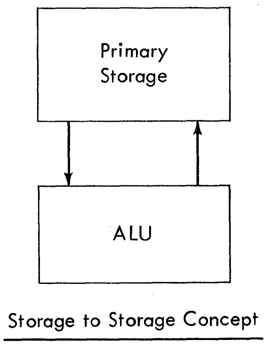


Figure 1-11. Storage to Storage Concept

storage, vacuum tubes or transistorized registers. In the storage to accumulator concept one of the data fields would be in main storage and the other would be in an accumulator. Both fields would be brought out to the ALU, operated upon, and the result would go back into the accumulator (Figure 1-12).

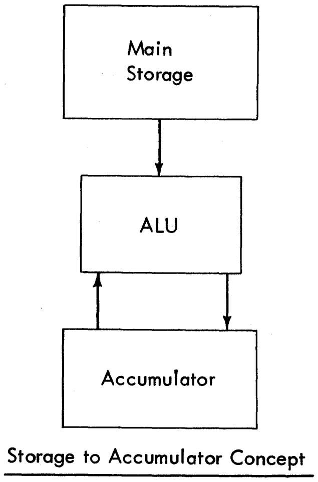


Figure 1-12. Storage to Accumulator Concept

For its variable length operations the System/360 uses the storage-to-storage storage concept (Figure 1-13).

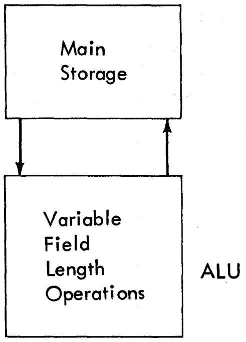


Figure 1-13. System/360 Storage to Storage Operations

As you have previously learned, variable length fields can start at any byte location in main storage. They are not restricted by storage boundaries as are fixed length operands. However, there fixed length operands. (Data fields are sometimes referred to as operands.) However, there must be some way of indicating to the system the length of the fields. In computers of the past, this was done several ways. The 1401 used a special word mark bit over the high-order position of the data. The IBM 705-II used zone bits. In the System/360 variable length operations use binary and decimal operands. In order to be code independent, System/360 specifies the length of these fields by a length code in the instruction.

The length code can be either 4 or 8 bits long, depending on the instruction. The length code is in binary. As a result the maximum length can be either 16 or 256 bytes. The values of the code is one less than the total number of bytes.

Length code of 0000 = 1 Byte

Length code of 1111 = 16 Bytes

Length code of 11111111 = 256 Bytes

Fixed-Length Operations: When operating on fixed-length fields (such as half words, words, or double words), the System/360 uses the storage-to-accumulator concept. These fixed-length operations use binary operands. For use as accumulators, the System/360 has 16 registers available to the programmer. As these registers can be used for purposes other than accumulating, they are called general registers (Figure 1-14).

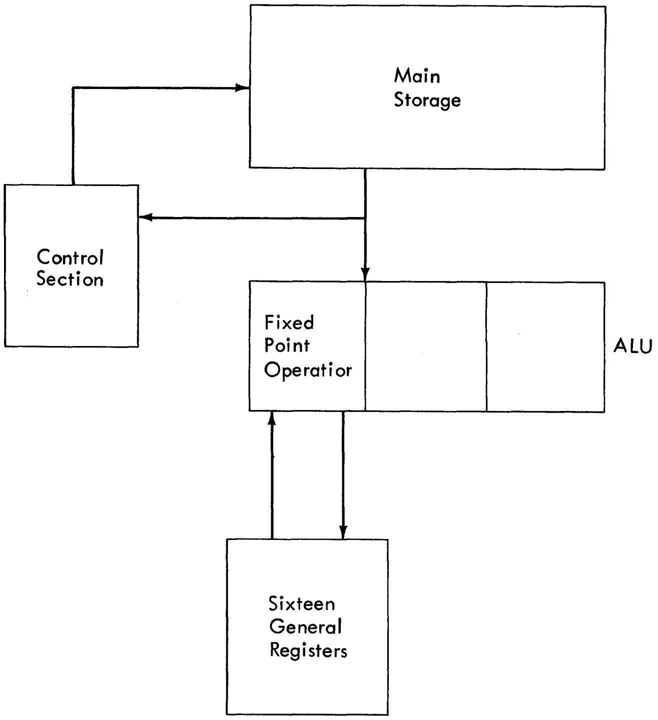


Figure 1-14. 16 General Registers

These registers are numbered 0-15 and are addressed in an instruction by a 4-bit binary address field.

Being a word in length, a general register can easily contain a half word data field.

As can be seen in Figure 1-15, the bits of a general register are numbered left to right starting with the number 0. Also we can see that a half word operand is placed in the low-order half (bits 16-31) of a General Register.

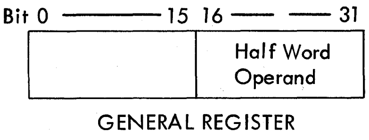


Figure 1-15. General Register

None of the General Registers 0-15 can contain a double word. For those operations that use a double word operand, such as fixed length divide, a pair of adjacent registers are used. In these cases, an even-odd pair of registers (such as 0-1 or 6-7) are used, and the even register is addressed.

In this case bits 0-63 of the double word would be in the registers as shown in Figure 1-16.

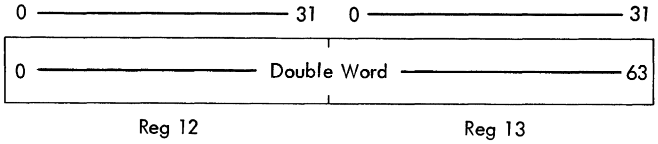


Figure 1-16. Using Two General Registers

Fixed-length operands in main storage must be on integral boundaries or a program check will occur indicating a specification exception.

The general registers are also used for purposes other than accumulating. For example, a general purpose register can be used as an index register. Indexing is a form of indirect addressing. An increment contained in an index register is added to the data address in the instruction to form an effective main storage address. Neither the index register nor the instruction in storage is changed by indexing.

Register-to-Register: With sixteen general registers, sometimes both fixed length binary operands will be in general registers. In these cases, another data flow concept is used (register-to-register operation, Figure 1-17).

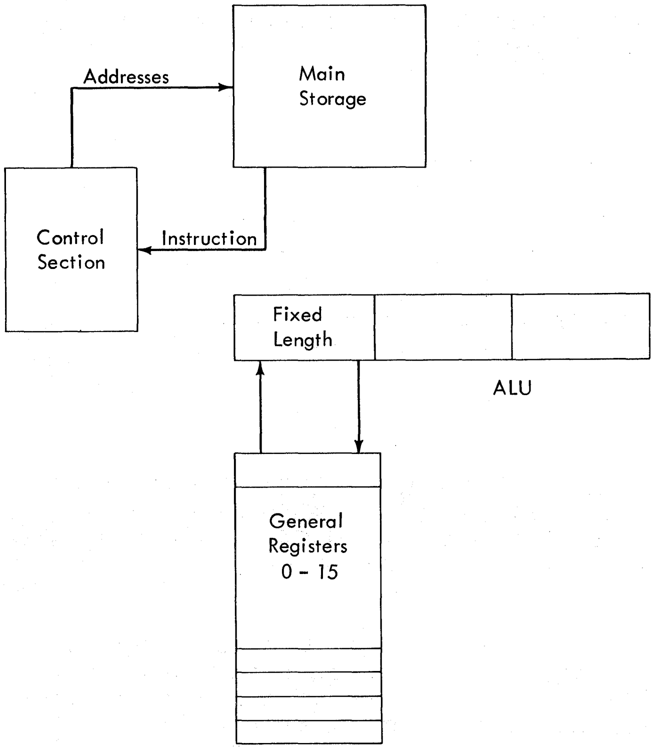


Figure 1-17. System/360 Register to Register Operations

Floating-Point Operation: Floating point is the term given to arithmetic operations involving a fraction and an exponent. For instance:

217,000 can be expressed as:
$.217 \times 10^6$

296,000 can be expressed as:
$.296 \times 10^6$

Fixed point arithmetic would add the numbers thusly:

$$\begin{array}{r} 217{,}000 \\ +\ \underline{296{,}000} \\ 513{,}000 \end{array}$$

Floating point arithmetic would do it like this:

$$\begin{array}{r} .217 \times 10^6 \\ +\ \underline{.296 \times 10^6} \\ .513 \times 10^6 \end{array}$$

Floating-point arithemetic is most useful for expressing very large numbers and operating on them with much precision. To do floating point arithmetic the System/360 has four floating point registers (Figure 1-18).

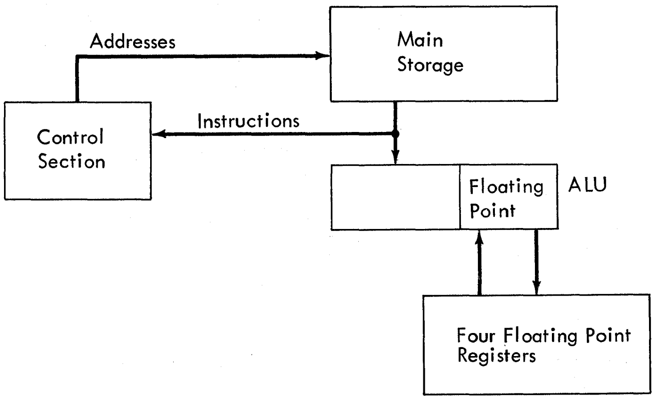


Figure 1-18. Floating Point Registers

The four floating point registers are numbered 0, 2, 4, 6. These are not the same as general registers 0, 2, 4, 6. The floating point registers are separate registers used only as accumulators during floating point operations.

The floating point registers are double word registers and are addressed by a four bit binary address in floating point instructions.

```
Bits 0                    63
     ┌────────────────────┐
     │   F.P. Reg 6       │
     └────────────────────┘
        F.P. Reg Address = 0110
```

Logical vs Hardware Structure of System/360: The structure of the System/360 which you have been learning is its logical structure: By this we mean that this is the way the System/360 appears to the programmer. The manner in which this logical structure is implemented will vary between the different models of the System/360.

For example:

1. In Models 60, 62, and 70 of the System/360, the general and floating point registers are transistor registers.

2. In Models 40 and 50 a core array is used. This array is similar to main storage but is a separate physical entity.

3. In Model 30 the general and floating point registers are located in the main storage unit. However, they do not use any of the available main storage addresses. The area of the main storage unit used for registers is called auxiliary storage.

Another example of hardware differences is in the control section of the System/360. In the Model 70, the control section is made up of high speed transistorized circuits. However, other models of System/360 use a capacitor- or a transformer-storage device for most of their control functions. This device is called Read Only Storage (ROS). The ROS is a storage device and cannot be changed by the programmer. It is strictly a control device.

Channels (Figure 1-19)

- Channels handle data transfers between main storage and I/O devices.
- All I/O devices are connected to channels via a standard interface.
- There are 2 types of channels: selector and multiplexor.
- Selector channels are designed for high data rates.
- Multiplexor channels are designed for low data rates.

One of the main functions of a channel is to handle I/O requests for a main storage cycle. The channel receives data from the I/O devices one byte at a time. When enough data has been received to justify the use of main storage, the channel requests a storage cycle. The amount of data handled varies depending on the particular model of System/360. After the data has been placed in main storage, the channel waits for additional information from the input device. For an output device the procedure reverses. The channel requests a main storage cycle and brings out data. It passes this data to the output device one byte at a time.

Since the channel is taking care of main storage cycles for the I/O device, the central processing unit is now logically free to continue processing instructions. We say that processing is overlapped with the I/O operation.

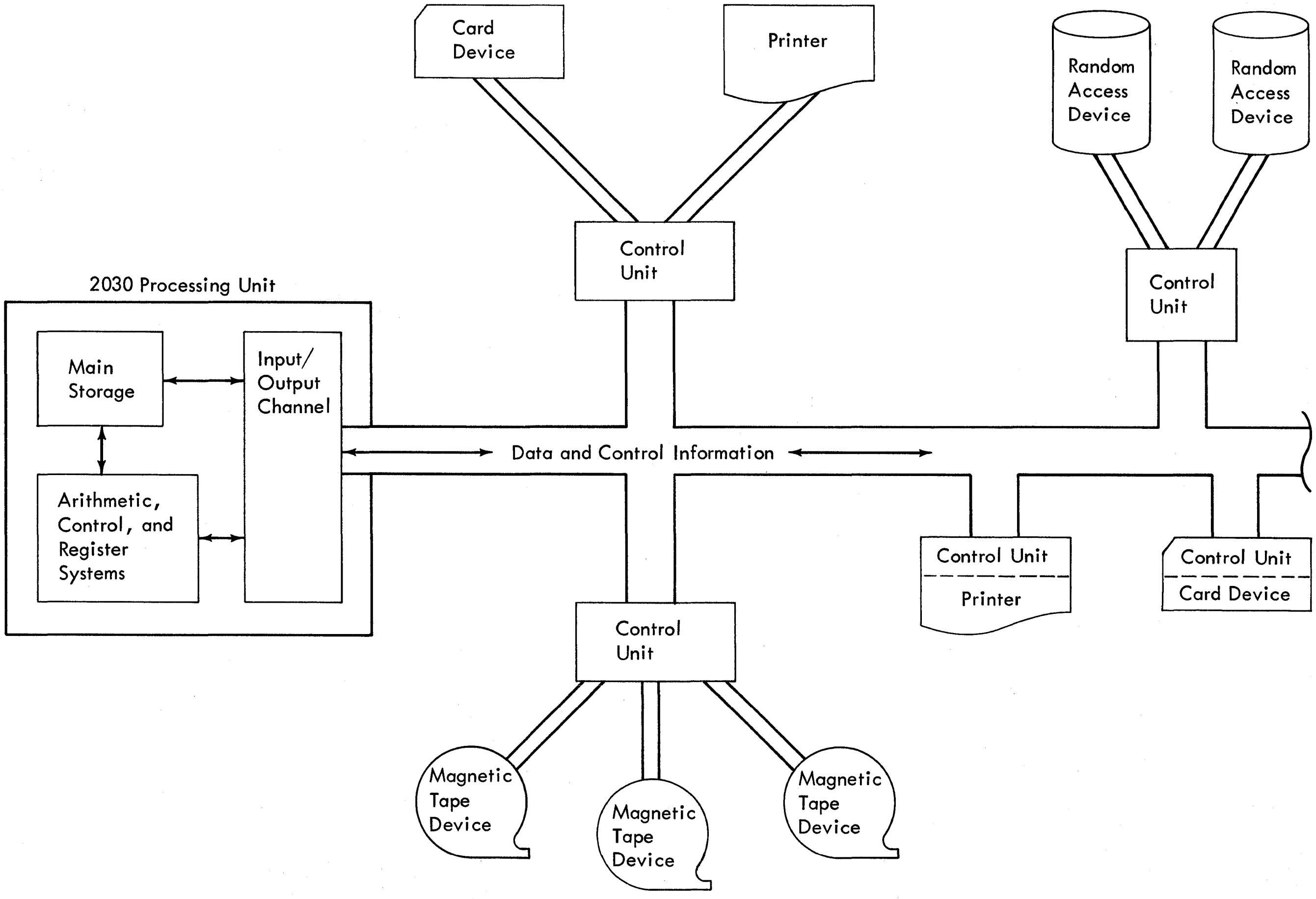


Figure 1-19. Channel Operations

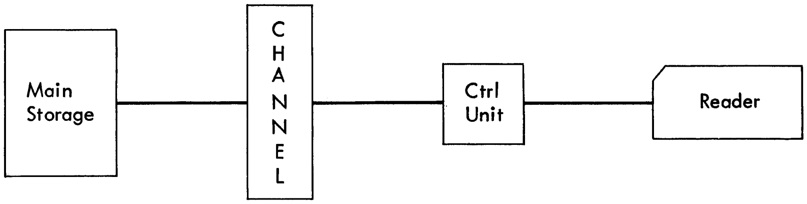


Figure 1-20. From Main Storage to I/O Device

On some models of the System/360, overlapping channel with CPU operations is allowed only at certain times. Once the CPU has started a channel operation, it has to wait for the channel operation to finish before it can continue processing instructions.

Each I/O device ties into the channel through a control unit (Figure 1-20).

Another name for a control unit is adapter. For some I/O devices the control unit or adapter is built into the device. For other devices, the control unit is external to the device.

Some adapters can control only one I/O device while others can control a number of similar I/O devices. The IBM 1443 Printer Model N1 is an example of an I/O device with a self-contained adapter. The IBM 2803 tape control is an example of a stand-alone adapter which can control up to eight IBM 2401 magnetic tape units (Figure 1-21).

Selector Channels: Selector channels are available on all models of the System/360. The maximum number per model varies from two for a Model 30 to six for a Model 70. The Selector channel is so named because only one I/O device can be selected on the channel at any one time. Once selected, a complete record is transferred over the channel one byte at a time.

Once the record has been transferred, the channel is free to select another I/O device. When a channel is transferring an entire record between main storage and an I/O device, it is said to be operating in burst mode. Because a selector channel always transfers an

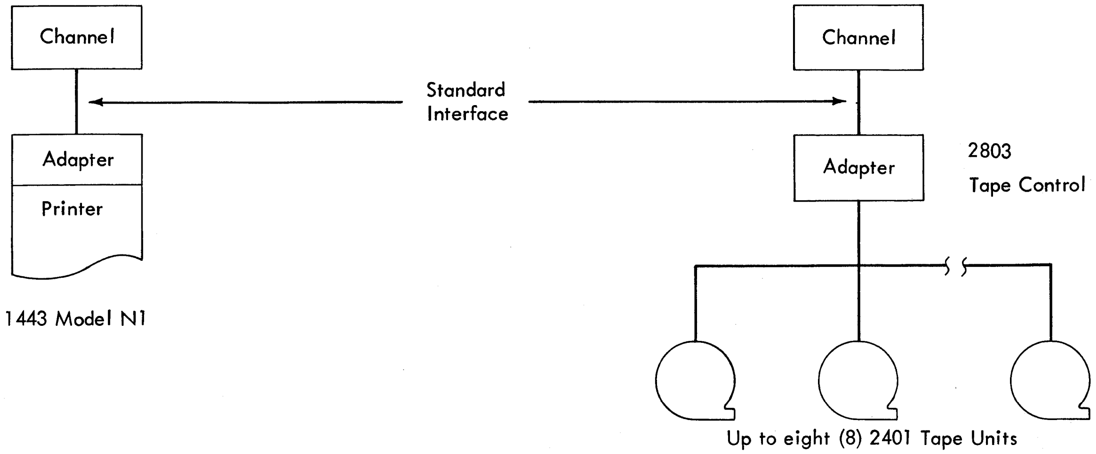


Figure 1-21. Control Units and Adapters

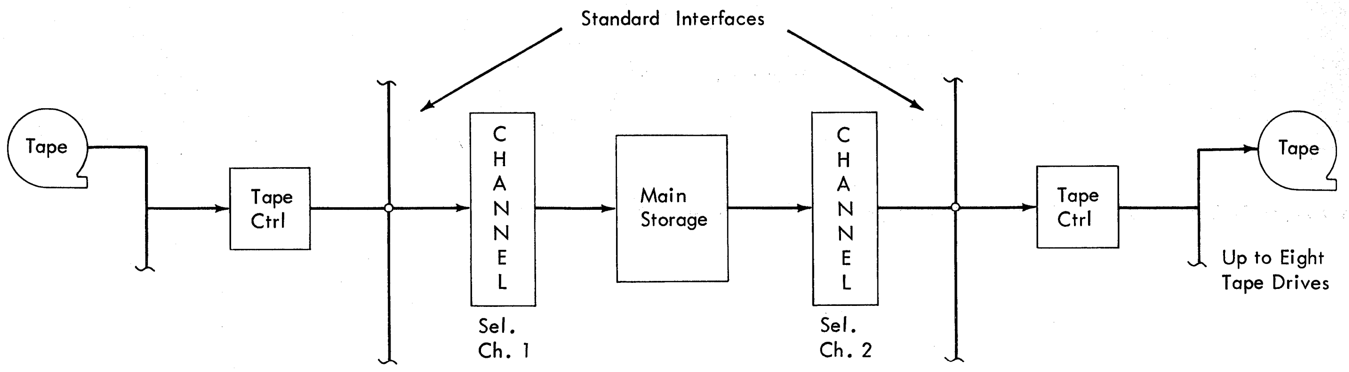


Figure 1-22. Dual Channel Operation

entire record, it can only operate in burst mode.

Although only one I/O device can be operating on a selector channel at any one time, multiple selector channels can be in operation simultaneously. Figure 1-22 shows an input record being read in from tape over selector channel 1 at the same time an output record is being transferred over selector channel 2.

Selector channels are designed to operate with high data rates. I/O devices such as magnetic tape, disk units, drums, and buffered card devices are the devices most likely to be connected to a selector channel.

Multiplexor Channels: A multiplexor channel is designed to operate with a number of I/O devices simultaneously on a byte basis. That is, several I/O devices can be transferring a record over the multiplexor channel, time-sharing it on a byte basis. When the multiplexor channel is being time shared by several devices, it is said to be operating in data-interleave mode. The multiplexor channel can also operate in burst mode for higher-speed units (Figure 1-23).

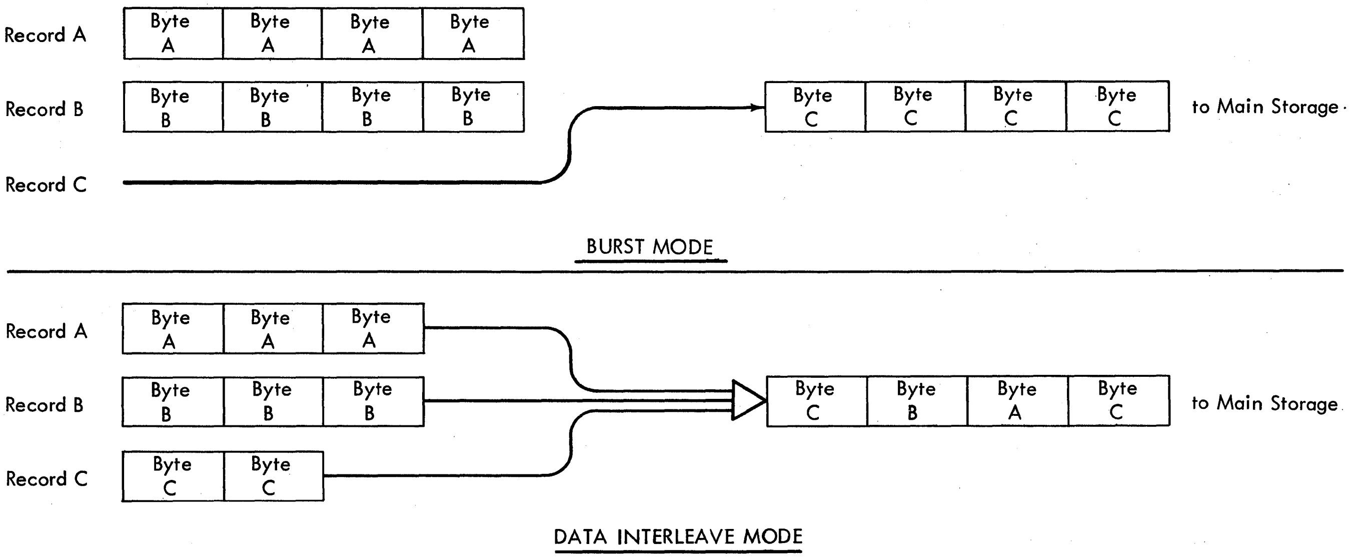


Figure 1-23. Burst Mode vs. Multiplex Mode

A comparison of burst versus multiplex mode can be seen in Figure 1-23.

To handle data flow from an I/O device, the channel needs to know certain information such as:

1. In which direction must data flow (input or output)?

2. Where in main storage should data be placed or taken from?

3. How many bytes should be sent to an output device or accepted from an input device?

Information of this type is contained in the I/O command addressed to a particular I/O device. For a selector channel which operates with only one I/O device at time, this information may be placed in the channel registers and left there

On a multiplexor channel it is possible to have I/O devices operating simultaneously. To have all this information in the multiplexor channel's registers would require a set of registers for each I/O device. Therefore, the multiplexor channel keeps this information in a compact storage area known as auxiliary storage. Auxiliary storage is part of the physical core array used for the main storage unit.

Auxiliary storage does not use any of the main storage addresses (Figure 1-24).

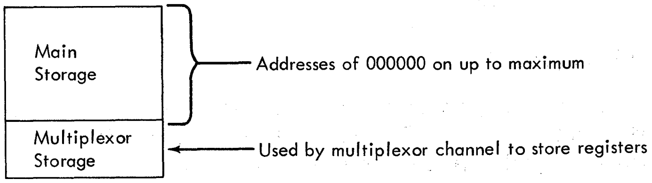

Main Storage
Addresses of 000000 on up to maximum
Multiplexor Storage
Used by multiplexor channel to store registers

Figure 1-24. Local Storage

It is a physical part of the core array used by the main storage unit. However, logically auxiliary storage is separate from local storage. On the Model 30, part of the auxiliary storage is used to contain the sixteen general registers mentioned previously.

NUMBERING SYSTEMS

Numbering systems were developed by man so that he could count. A number basically is a group of symbols. Each symbol in a number has a definite place and value.

The place value of the digit symbol is some power of the base. The power of the base starts with zero and increases by one from right to left. The base of the decimal numbering system in common usage is ten (10).

Example:

$$444 = 4x10^2+4x10^1+4x10^0 \quad (10^0 = 1)$$

Digit Base Place

$$444 = 400+40+4$$

The maximum value of a symbol is always one less than the base. In our above example, the base is ten (10) so the maximum value is nine (9). If when adding two symbols the total exceeds the base, a carry (value of one) is added to the next higher place.

```
Example:        6
               +6
        Carry  12
```

BASIC BINARY

Units of Binary

This is a numbering system using a base of two. All of the rules which we discussed for a numbering system apply here. Let us check.

The base is two so the maximum symbol value is one. The binary numbering system has only two valid symbols, 0 and 1.

Example:

Digit Base Place

$$0101 = 0x2^3+1x2^2+0x2^1+1x2^0$$

decimal 5 0 4 0 1

Binary-Decimal Conversion

The decimal place values of a six position binary number are:

32	16	8	4	2	1
$1x2^5$	$1x2^4$	$1x2^3$	$1x2^2$	$1x2^1$	$1x2^0$
1	1	1	1	1	1

The preceding binary number equals 63 when the place values are added.

The following chart shows conversion from decimal to binary:

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

ZONED DECIMAL - EXTENDED BINARY CODED DECIMAL (BCD)

Zoned decimal is a decimal number representing a BCD alphanumeric character. A zoned decimal has eight places divided into two 4-place sections. The high-order places represent the alphanumeric zone bit and the low-order places represent the alphanumeric decimal.

```
Example: 1101  0111=P

          11     7
        (zone) (digit)
```

PACKED DECIMAL

The packed decimal coding system allows two decimal digits to be stored in one byte-location:

Example:

Binary Position	8421	8421
Binary Value	1001	0011
Decimal Equivalent	9	3

HEXADECIMAL

The hexadecimal system uses the decimal value of sixteen as its base.

16^3	16^2	16^1	16^0
4096	256	16	1

In the hexadecimal system you can count to fifteen before a carry occurs.

```
 14     0     15
+ 1    +0    + 1
---    --    ---
 15     1      0
```

To express the values ten to fifteen, the symbols A-F are used. Thus, the sixteen hexadecimal symbols are:
0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F.
The following charts lists some sample decimal to binary to hexadecimal conversion.

Decimal	Binary	Hexadecimal
1	0001	1
5	0101	5
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F
16	0001 0000	10
17	0001 0001	11
18	0001 0010	12
19	0001 0011	13
20	0001 0100	14
30	0001 1110	1E
40	0010 1000	28
50	0011 0010	32
60	0011 1100	3C

Examples:
Express the hexadecimal (hex) numbers as a sum of terms in decimal.

$$\begin{aligned} \text{hex: } 482 &= 4\text{x}16^2+8\text{x}16^1+2\text{x}16^0 \\ &= 4\text{x}256+8\text{x}16+2\text{x}1 \\ &= 1024+128+2 \\ &= 1154 \text{ decimal} \end{aligned}$$

$$\begin{aligned} \text{hex: } 8C6 &= 8\text{x}16^2+12\text{x}16^1+6\text{x}16^0 \\ &= 8\text{x}256+12\text{x}16+6\text{x}1 \\ &= 2048 +192 \ +6 \\ &= 2246 \text{ decimal} \end{aligned}$$

Rules for Converting from Decimal to Hexadecimal

1. Divide decimal number by the new base (16).
2. Remainder becomes low order of new number.
3. Divide quotient by the new base (16).
4. Remainder becomes next digit of new number.
5. Repeat steps 3 and 4 until a quotient of zero is obtained.

Example: Decimal 456 to hexadecimal 1C8

```
                                                1C8
                                                ↑↑↑
                                                |||
                                                |||
          0                                     |||
         r--                                    |||
       16|1   Remainder of 1--------------------┘||
                                                 ||
        r---                                     ||
     16|28    Remainder of 12 or C---------------┘|
                                                  |
      r----                                       |
   16|456     Remainder of 8----------------------┘
```

Binary - Hexadecimal Conversion

The base of the binary system is 2^1 while the hexadecimal system uses a base of 2^4 (16). There is a direct 4 to 1 relationship (2^4 to 2^1) between the exponents of the two bases. Every hexadecimal digit becomes four binary digits. Every four binary digits in turn can be converted to a single hexadecimal digit.

Hexadecimal	8	3	4	9	E
Binary	1000	1011	0100	1001	1110

FLOATING POINT

Description

Various computer computations frequently involve lengthy and complex calculations in which it is necessary to manipulate numbers that may vary widely in magnitude. To obtain a meaningful answer, problems of this type usually require that as many significant digits as possible be retained during calculation and that the decimal point always be properly located. When applying such problems to a computer, several factors must be taken into consideration, the most important of which is the decimal point location.

Generally speaking, a computer does not recognize the decimal point present in any quantity used during the calculation. Thus, a product of 414154 will result regardless of whether the factors are 9.37 X 44.2, 93.7 X 0.442, or 937 X 4.42, etc. A system must be employed in which information regarding the magnitude of all numbers accompanies the quantities in the calculation. Thus, if all numbers are represented in some standard, predetermined format which instructs the computer in an orderly and simple fashion concerning the location of the decimal point, and if this representation is acceptable to the routine doing the calculation, then quantities which range from minute fractions having many decimal places to large whole numbers having many integer places can be handled. The arithmetic system used, in which all numbers are expressed in a format having the feature just described, is called floating point arithmetic.

The notation used in floating point arithmetic is basically an adaptation of the scientific notation widely used today. In scientific work, very large or very small numbers are expressed as a number, between one and ten, times a power of ten. Thus 427.93456 is written as 4.2793456 X 10^2 and 0.0009762 as 9.762 X 10^{-4}.

The System/360 uses hexadecimal floating point and the hexadecimal point of all numbers is placed to the left of the high-order (leftmost) nonzero digit. Hence, all quantities may be thought of as a hexadecimal fraction times a power of 16. In addition to the advantages inherent in scientific notation, the use of floating point numbers during processing eliminates the necessity of analyzing the operations to determine the positioning of the decimal point in intermediate and final results since the decimal point is always immediately to the left of the high-order nonzero digit in the fraction.

Floating-point arithmetic simplifies programming by automatically maintaining hexadecimal point placement (scaling) during computations in which the range of values used varies widely or is unpredictable.

The key to floating-point data representation is the separation of the significant digits of a number from the size (scale) of the number. Thus, the number is expressed as a fraction times a power of 16.

A floating-point number has two associated sets of values. One set represents the significant digits of the number and is called the fraction. The second set specifies the power (exponent) to which 16 is raised and indicates the location of the binary point of the number. The term Mantissa is often used instead of fraction. Characteristic is another term for exponent.

Sign

These two numbers (the fraction and exponent) are recorded in a single word or double-word.

Since each of these two numbers is signed, some method must be employed to express two signs in an area that provides for a single sign. This is accomplished by having the fraction sign use the sign associated with the word (or double word) and expressing the exponent in excess 64 arithmetic. That is, the exponent is added as a signed number to 64. Above 64 is a positive exponent and below 64 is a negative exponent. The resulting number is called the characteristic. The characteristic can vary from 0 to 127, permitting the exponent to vary from -64 through 0 to +63. This provides a decimal range of 7.2 X 10^{75} to 2.4 X 10^{-78}.

Floating-point data in the System/360 may be recorded in short or long formats, depending on the precision required. Both formats use a sign bit in bit position 0, followed by a characteristic in bit positions 1-7. Short-precision floating-point data operands contain the fraction in bit positions 8-31; long-precision operands have the fraction in bit positions 8-63.

Short-Precision Floating-Point Format

```
┌──┬───────────┬──────────────────────┐
│S │ Exponent  │ Fraction             │
└──┴───────────┴──────────────────────┘
0  1          7 8                   31
```

Long-Precision Floating Point Format

```
┌──┬───────────┬──────────────────────┐
│S │ Exponent  │ Fraction             │
└──┴───────────┴──────────────────────┘
0  1          7 8                   63
```

The sign of the fraction is indicated by a zero or one bit in bit position 0 to denote a positive or negative fraction, respectively.

ARITHMETIC PRINCIPLES

BINARY FIXED POINT

Positive and Negative Numbers

A fixed point number has a signed value recorded as a binary integer. The first bit position (high order) holds the sign of the number, with the remaining bit positions used to designate the magnitude of the number.

Positive fixed-point numbers are represented in true binary form with a zero sign bit. Negative fixed-point numbers are represented in two's complement notation with a one bit in the sign position. In all cases, the bits between the sign bit and the leftmost significant bit of the integer are the same as the sign bit (i.e., all zeros for positive numbers, all ones for negative numbers).

Negative fixed-point numbers are formed in two's complement notation by inverting each bit of the positive binary number and adding one. For example, the true binary form of the decimal value (plus 26) is made negative (minus 26) in the following manner:

```
        S      INTEGER

   +26  0 0000000 00011010
Invert  1 1111111 11100101
 Add 1  __________________1
   -26  1 1111111 11100110  (Two's com-
                             plement form)
```

This is equivalent to subtracting the number from 1 00000000 00000000.

Binary Addition

```
    0 + 0 = 0
    1 + 0 = 1
    0 + 1 = 1
    1 + 1 = 0  with a carry of 1
1 + 1 + 1 = 1  with a carry of 1.
```

The following addition examples illustrate two's complement arithmetic. Only eight bit positions are used. The left-most position is the sign position, the next position to the right is the high-order position.All negative numbers are in two's complement form.

An overflow condition occurs when carries out of the sign position and high order numeric position disagree. When overflow occurs the sign of the result is incorrect. This condition must be noted and properly used by the processor.

When there is no carry out of the high order position, the answer is in complement form. This is the final answer in fixed point binary subtract. The model 30 does not recomplement.

Examples:

		S	COMMENTS
+57	=	00111001	No high-order carry,
+35	=	00100011	no sign-position carry,
+92	=	01011100	no overflow. Result is in true form.
+57	=	00111001	High-order carry,
-35	=	11011101	sign-position carry,
+22	=	00010110	no overflow. Result is in true form.
+35	=	00100011	No high-order carry,
-57	=	11000111	no sign-position carry,
-22	=	11101010	Bit in sign position indicates result is in complement form.
-57	=	11000111	High-order carry,
-35	=	11011101	sign-position carry,
-92	=	10100100	no overflow. Bit in sign position indicates result is in complement form.
-57	=	11000111	High-order carry,
-92	=	10100100	sign-position carry,
-149	=	~~10010101~~	overflow. No bit in sign-position indicates result is in true form, but overflow sign is incorrect.
+57	=	00111001	High-order carry, no
+92	=	01011100	sign-position carry,
+142	=	10010101	overflow. Sign-position indicates a complement result, but overflow signals are incorrect sign indication.

Binary Subtraction

```
 678   001010100110
-456  +111000111000
 222   000011011110
```

Notice that binary subtraction is simply adding the two's complement of the number to be subtracted.

Examples of Fixed Point Numbers:

The following are 16-bit fixed-point numbers. The first is the largest positive number and the last, the largest negative number.

2^{15} -1	= 32,737	=0	11111111	11111111
2^{0}	= 1	=0	00000000	00000001
0	= 0	=0	00000000	00000000
-2^{0}	= -1	=1	11111111	11111111
-2^{15}	=-32,768	=1	00000000	00000000

Table 1 shows 32-bit fixed-point numbers. The first is the largest positive number that can be represented by 32 bits, and the last is the largest negative number.

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

Table 1. Fixed Point Numbers

The following is an example of hexadecimal complement addition. The general method is the same as binary but the base is sixteen.

```
Problem:  E7A4
         -A48E
```

First, convert the number to be subtracted to its sixteen- complement.

```
 FFFF
-A48E
 5B71
   +1
 5B72
```

Now, add the sixteen's complement of A48E to E7A4.

```
        E7A4
       +5B72
 Carry←4316
```

Presence of the high-order carry means the answer is in true form. If there had been no carry, the result would have been in complement form.

PACKED DECIMAL

Sign-Operation Analysis

Before any packed decimal arithmetic can take place, a sign- operation analysis must be performed. The result of this analysis determines whether the operation is to be a true add or a complement add. There are three conditions that are analyzed to determine the type of operation. These are:

1. type of operation (add or subtract) : + or -

2. sign of first operand: + or -

3. sign of second operand: + or -

An even number of minus signs calls for a true-add operation to combine the two operands. An odd number of minus signs calls for a complement-add operation to combine the two operands. Thus, eight conditions can occur. These eight are analyzed as follows:

First Operand	Operation Sign	Second Operand	True or Complement
+X	+	+Y	True
-X	+	+Y	Complement
-X	+	-Y	True
+X	+	-Y	Complement
+X	-	+Y	Complement
-X	-	+Y	True
-X	-	-Y	Complement
+X	-	-Y	True

The signs of the operands are maintained in the low-order half byte (four bit positions). The actual bit-coding of the sign depends on the data-coding scheme being used.

Packed-Decimal True Add

In the packed-decimal data format, we want each four bits to represent one decimal digit. The range of this decimal digit is, of course, 0 - 9. However, four bits with the packed-decimal value of 1, 2, 4, and 8 can represent any decimal number from 00 to

15. If we are simply storing packed-decimal data, no problem exists because we make sure that the digit we store at each packed decimal location is in the range 0 to 9. However, if we add two packed-decimal digits together, and their total exceeds nine, we are in trouble. Remember that in a decimal system, we wish to carry from the units position to the tens position when the units total exceeds nine. Using the packed-decimal system just described, if we added two decimal digits together, and if their total exceeded nine, the result would be in hexadecimal form. For example, if we added nine and four, the result would be 13, instead of three with a carry. To compensate for this discrepancy between the actual capacity of the 4-bit packed-decimal digit (00-15) and the desired capacity of the 4-bit packed decimal digit (0-9), we add a correction factor of six to our rules for true odd.

Packed-Decimal True Add

The following general rules for packed-decimal true add provide decimal correction as mentioned previously.

1. Add six to each packed-decimal digit (4-bit group) of the first operand.

2. Add the result to the second operand. This should be a conventional binary add, allowing carries from one 4-bit group to the next.

3. Now, considering each 4-bit group (packed-decimal digit) separately, subtract six from each group where no high-order carry occurred. If a high-order carry <u>did</u> occur, add 0000 to that 4-bit group. Remember, that to subtract six really means complement add six. The result will be the correct decimal number in true form.

```
Example:      16 = First operand
              18 = Second operand
              34 = Result of true add


   8421 8421  Packed-decimal value
   0001 0110  First operand
   0110 0110  Add six per Rule 1
   0111 1100
   0001 1000  Second operand
  NC    C
   1001 0100
   1010 0000  Subtract six per Rule 3
   0011 0100  Result of true add
```

Packed-Decimal Complement Add

The following general rules are for packed-decimal complement add. Notice that the necessary correction factor of six is employed differently than in the previous example of true add.

1. Develop the binary complement of the second operand.

2. Add the complement of the second operand to the first operand.

3. Considering each 4-bit group (packed-decimal digit) separately, subtract six from each group where no high-order carry occurred. If a high-order carry occurred, add 0000 to that 4-bit group. Remember, that to subtract six, you must add the binary complement of six (1010).

4. If there is a carry out of the high-order group of four bits at the completion of the original add (Rule 2), the answer will be in true form. If there is no carry out of the high-order group at the completion of the original add, the answer will be in complement form and will have to be recomplemented. To obtain the correct result when recomplementing, we must subtract six from each 4-bit group of the recomplemented number. To condition this correction operation, we must first add the recomplemented number to zero to give us the no-carry indication and apply Rule 3.

   <u>Note</u>: When subtracting six from each 4-bit group, any carry from a group is ignored.

Example: Packed-Decimal Complement Add

1st Cperand +49
2nd Cperand -92

```
   +49 =  0100 1001
   -92 =  1001 0010


          0110 1110 Complement 2nd Oper.
          0100 1001 Add 1st Cper.
        NC    C     NC From High order
          1011 0111 Position indicates
                    Recomplement Rule 4
          1011 0111
          1010 0000 Rule 3
          0101 0111
          1010 1001 Recomplement
          0000 0000 Add to Zero Rule 4
        NC   NC
          1010 1001
          1010 1010 Rule 3
   -43 =  0100 0011 Result
```

FLCATING-POINT ARITHMETIC

Description.

Floating-point is basically a mathematical shorthand by which numbers are expressed as a fraction and an exponent. This notation is an adaptation of the scientific notation used today. In scientific work, very large or very small numbers are expressed as some number one through ten times a power of ten.

Example   427.93456 = 4.2793456 X 10^2
           0.0009762 = 9.762 X 10^{-4}

System/360 uses hexadecimal floating-point, and the decimal point is always to the left of the high-order digit. The hexadecimal fraction may be thought of as a number (base 16) X a power of 16.

Decimal numbers must first be converted to numbers the system may use in floating-point operations. The sign of the fraction will be noted in the sign position of the floating-point word, a 0 for plus a 1 for minus. The sign of the exponent will be contained in the characteristic which will represent the exponent and its sign.

To develop the characteristic, the exponent is expressed in excess 64 arithmetic. The signed exponent is algebraically added to 64. A result above 64 indicates a positive exponent; a result below 64 indicates a negative expcnent.

Conversion Example:

1. The decimal number must be separated into a decimal integer and a decimal fraction.
   123.725 = 123+.725

2. The decimal integer is converted to its hexadecimal representation.
   decimal 123 = hexadecimal 7B

3. The decimal fraction is converted to its hexadecimal representation.
   decimal .725 = hexadecimal .2D5

4. Combine the integer and the fraction and express as a fraction times a power of 16.
   7B.2D5 = .7B2D5 X 162

5. Develop the characteristics and convert to binary
   64+2 = 66 = 1000010

6. The fraction is converted to binary and grouped hexadecimally.
   .7B2D5 = 0111 1011 0010 1101 0101

7. The characteristic and the fraction are stored in floating-point format. The sign position contains the sign of the fraction

```
 Sign    Characteristic
  0      1000010

         Fraction
 0111 1011 0010 1101 0101 00
```

System/360 provides four floating-point registers, each eight bytes in length, for use in floating-point operations. The first operand of any floating-point instruction is always contained in one of these registers. The result of a floating-point operation replaces the first operand except in a store operation.

Normalization

A quantity can be represented with the greatest precision by a floating-point number of given fraction length when that number is normalized. A normalized floating-point number has a nonzero high-order hexadecimal fraction digit.

The process of normalization consists of shifting the fraction left until the high-order hexadecimal digit is nonzero and reducing the characteristic by the number of hexadecimal digits shifted. A zero fraction cannot be normalized, and its associated characteristic therefore remains unchanged when normalization is called for.

Normalization usually takes place when the intermediate arithmetic result is changed to the final result. This function is called postnormalization. In performing multiplication and division, the operands are normalized before the arithmetic process. This function is called prenormalization.

Floating-point addition and subtraction may be performed with or without normalization.

When an operation is performed without normalization, high-order zeros in the result fraction are not eliminated. The result may or may not be normalized, depending on the original operands.

In both normalized and unnormalized operations, the initial operands need not be in normalized form.

Floating-Point Addition and Subtraction.

Addition or subtraction of two floating-point numbers consists of characteristic comparison and fraction addition. The characteristics of the two operands are compared, and the fraction with the smaller characteristic is right shifted; its characteristic is increased by one for every hexadecimal digit shift (four binary places) , until the two characteristics agree. The fractions are then added algebraically to form an intermediate sum. Subtraction differs here only in that the sign of the second operand is inverted before fraction addition.

Normalization may be called for in either addition or subtraction. Normalization consists of shifting the fraction left and decreasing the characteristic one for every hexadecimal digit shift.

Floating-Point Multiply and Divide

The multiplication of two floating-point numbers consists of a characteristic addition and a fraction multiplication. The sum of the characteristics less 64 is used as the characteristic of an intermediate product.

The product fraction is normalized by prenormalizing the operands before the operation, and postnormalizing the intermediate result if necessary.

Division of two floating-point numbers consists of subtracting the characteristics and dividing the fractions. The difference between the dividend and divisor characteristics plus 64 is used as an intermediate characteristic.

The quotient fraction is normalized by prenormalizing the operands before the operation. Postnormalizing the intermediate quotient is never necessary.

Instruction Format

Floating-point instructions use the following two formats:

RR Format

Op Code	R_1	R_2
0 7	8 11	12 15

RX Format

Op Code	R_1	X_2	B_2	D_2
0 7	8 11	12 15	16 19	20 31

In these formats, R1 designates the address of a floating-point register. The contents of this register will be called the first operand. The second operand location is defined differently for two formats.

In the RR format, the R_2 field specifies the address of a floating-point register containing the second operand. The same register may be specified for the first and second operand.

In the RX format, the contents of the general register specified by X_2 and B_2 are added to the content of the D_2 field to form an address designating the location of the second operand.

A zero in an X_2 or B_2 field indicates the absence of the corresponding address component.

The register address specified by the R_1 and R_2 fields should be 0, 2, 4, or 6. Otherwise, a specification exception is recognized, and a program interruption is caused.

The storage address of the second operand should designate word boundaries for short operands and doubleword boundaries for long operands. Otherwise, a specification exception is recognized, and a program interruption is caused.

Results replace the first operand, except for the storing operations, where the second operand is replaced.

Except for the storing of the final result, the contents of all floating-point or general registers and storage locations participating in the addressing or execution part of an operation remain unchanged.

The floating-point instructions are the only instructions using the floating-point registers.

DATA FLOW

SYSTEM CONTROL

- Read Only Storage (ROS) controls data movement.

- Capacitor Cards determine ROS output.

The System/360 Model 30 uses a Read Only Storage unit to control the movement of data throughout the system, Figure 1-25. This control includes the actual circuit components for addressing the ROS and for sensing and decoding the output. Just as you would follow a sequence of operations to perform a mathematical computation, the ROS steps through a series of micro-instructions to perform a machine language instruction. These micro-instructions consist of storage control, Arithmetic Logic Unit (ALU) control, register input and output controls, machine status control, ROS sequencing control, and input-output controls.

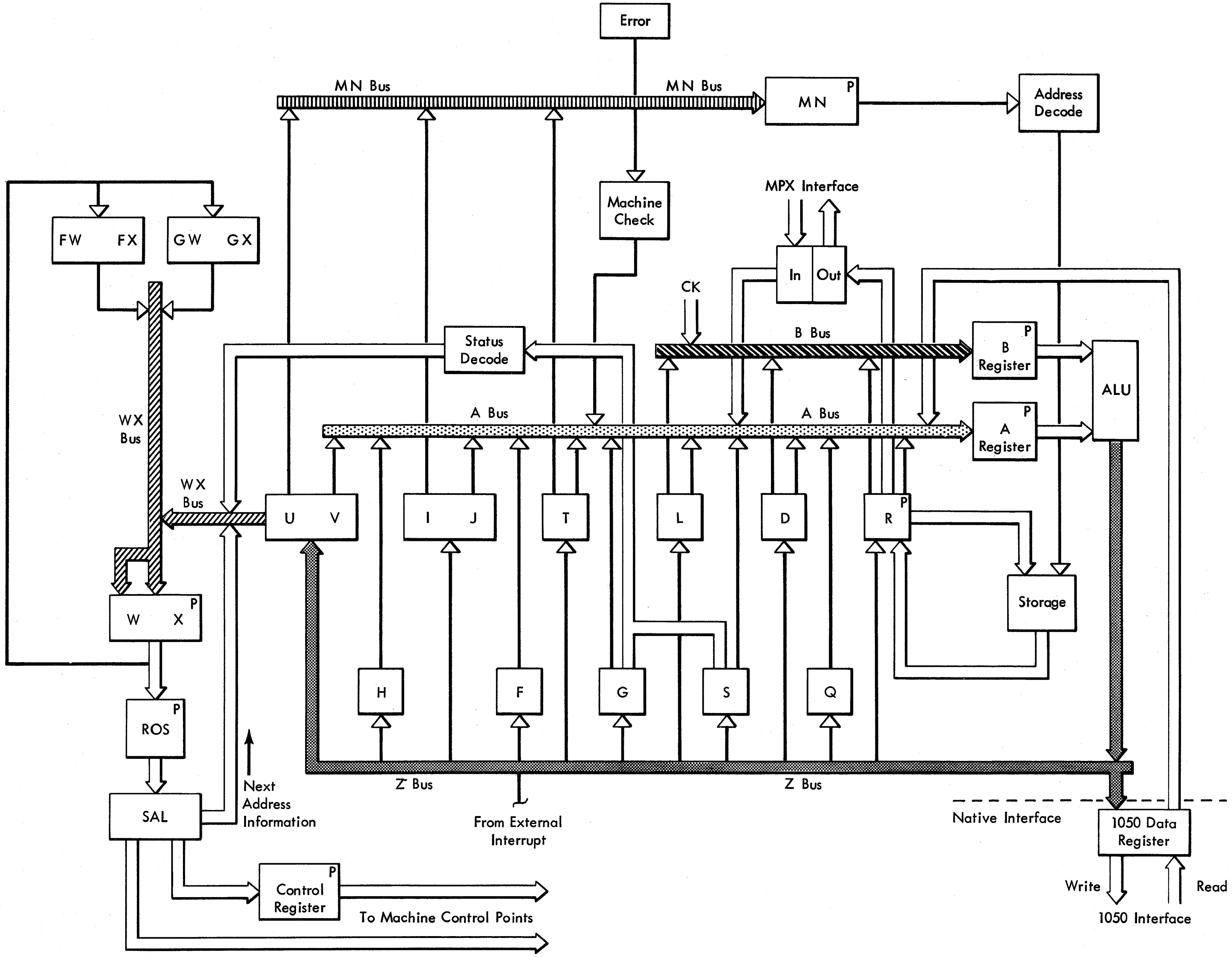


Figure 1-25. IBM 2030 Data Flow

RCS is a capacitor storage device which contains capacitors printed on a card. The output of RCS, as determined by the capacitor cards, is latched in sense amplifier latches (SAL'S). Definite fields in the cards are used to control entry and exit of data to the various registers and buses, Figure 1-25.

CORE STORAGE

- Core storage is made of small doughnut-shaped rings (magnetic cores).
- Capacities vary from 8,192 bytes to 65,536 bytes.

The basic storage unit of the Model 30 is made of small doughnut-shaped rings which are used to store information. This read-write storage is an eight bit (plus parity) wide binary addressed storage unit. The size of the core array can be expanded from the basic 8,192 bytes to a total of 65,536 bytes.

In addition to main storage, there are 512 positions of storage in the basic machine which are unaddressable as far as the programmer is concerned. These positions can only be used by internal machine micro-programming. These added positions give 256 positions for CPU use and 256 positions for I/O use.

The data flow chart (Figure 1-25) shows that there are two input lines to storage and one output. To locate data in storage the address of the data must be placed in the MN-register and decoded by circuits which actually address the proper byte in the core planes. The data "reads out" to the R-register which is the storage data register. Notice also that in order to write into core the data will come from the R-register and enter the location in core addressed by the MN register. These three operations are instructed by micro-programming through the use of RCS control fields.

DATA BUSES

- There are 7 general data buses which move information throughout the Control Processing Unit.
- All buses are 9 bits wide except the W-bus, which is 6 bits wide, including the parity bit.

In Figure 1-25 the general data buses are shown as wide data paths, only to indicate that they receive or send data to more than one register. The actual bit width of all buses is 8 (plus parity) with the exception of the W bus which is 5 bits, plus parity (bits 3 through 7 plus P).

The MN and WX buses are 4 physical buses feeding individual registers. As indicated in Figure 1-25, the UV and IJ registers are larger than the T-register and yet they feed the same buses. The T-register is programmed by ROS to transfer data to the N-register while at the same time the high order 8 bits of the MN register are set to zero. The A-bus supplies data from all registers to the A-register, while the B-bus accepts data from only 3 registers and then transfers it to the B-register.

The Z-bus is used to transfer data back to various registers from ALU. The output of the ALU becomes the Z-bus. The Z-bus negative-powered bus enters data to polarity hold latches of the registers. The Z-bus positive-powered bus enters data to the R and S registers.

ALU AND SOURCE REGISTERS

- The A and B registers supply data to ALU.
- ALU performs arithmetic and logical operations.
- The ALU output is corrected for parity, if necessary.

The data which is supplied to the A and the B registers by their respective buses, is fed to the ALU where arithmetic or logical functions are performed. ALU is only 8 bits wide as the parity bit is not used in this circuitry. Information enters ALU by the byte (8 bits) or by the half byte (4 bits), depending again on the controls generated by the ROS fields. The A-register output data before entry into ALU, passes through the straight-crossed switch and high-low gates.

The B-register output data is gated by high-low circuitry before it enters the true-complement side of ALU.

ALU is able to process 8-bit wide data, as a parity bit is not used. On the output of ALU, therefore, a parity bit is inserted, if needed. At this point the output has been corrected and becomes the Z-bus.

CTHER REGISTERS

- Ten 8-bit registers are connected to the A-bus.
- Three of these registers also supply the B-bus.
- All registers are given one character alphabetic names.

The basic unit of data is 9 bits wide (8 bits plus parity). The 9-bit registers in the data flow are given alphabetic names of one character, so that any bit position within the register can be referred to by a letter plus a number. Bits are numbered zero to seven, left to right, with the high order position being the zero. Reference to a particular bit in a register then, can be "S7", which is the low order position of the S-register, etc.

In addition to the ten registers, there are other registers shown in Figure 1. All of the registers in the data flow will have, in general, the following functions:

Register Name	Function
D	Temporary Data Register
F	For External Interrupt
FW-FX	Back-up address registers for ROS
GW-GX	Back-up address registers for ROS
G	General purpose status register
H	Priority Control Register
I	Address register, high 8 bits
J	Address registers, low 8 bits
L	Length of data field
M	Main Storage address register, 8 high bits
N	Main storage address register, 8 low bits
Q	Storage Protect Key register
R	Storage Data Register
S	Special purpose status register
T	Partial Address Register
U	Address register, 8 high bits
V	Address register, low 8 bits
W	Read only address register, high 5 bits
X	Read only address register, low bits

The gating in and out of all registers is controlled by ROS fields. A series of micro-instructions for example, is needed to move data from the D-register through ALU to the G-register. Micro-programming will move data from one location in main storage through the R-register and back to storage, using the address stored in the UV, IJ or T-registers to address core through the MN register. All of the operation again, is controlled by ROS fields and micro-programming.

BASIC PROGRAMMING

SYSTEM/360 INSTRUCTIONS

- Instructions are 1, 2 or 3 halfwords in length.
- The Op code is the first byte of an instruction and specifies instruction length and format, general data location, type of data, and operation to be performed.
- General Registers have a 4 bit address and main storage has a 24 bit address.
- The five instruction formats are called the RR, RX, RS, SI and SS formats.

INSTRUCTION LENGTHS

- System/360 instructions are 1, 2, or 3 Halfwords in length depending on the location of data.
- Register-to-register instructions are 1 halfword long.
- Storage-to-register instructions are 2 half words long.
- Storage-to-storage instructions are 3 halfwords long.
- Instructions must reside on halfword boundaries (low order address bit = 0). Let's learn about the instruction formats of the System/360. As you know, instructions specify the operation to be done and the location of data. Data may be located in main storage, in general registers or in floating point registers. Main storage is addressed with a 24 bit binary address while the general registers and floating point registers are addressed with a 4 bit binary address. As a result, instructions are of different lengths depending on the location of data. System/360 instructions may be one, two, or three <u>halfwords</u> in length.

When both operands or data are in general registers or floating point registers, only 8 binary bits are needed for addresses. As a result the shortest instruction, one halfword in length, is used.

When one of the operands is in main storage, this short instruction cannot be used. Remember that main storage requires a longer address. For these operations, instructions that are 2 halfwords in length must be used.

When both operands are in main storage, a total of 48 bits are needed for the addresses. Accordingly, the longest instruction format, 3 halfwords in length, is used.

Because instructions are a multiple of halfwords in length, they are considered as fixed length information as far as storage boundaries are concerned. If the address of an instruction has a low order 1 bit, a specification exception will occur.

OPERATION CODES

- The Op Code (Operation Code) is the first byte of each instruction.
- Bits 0 and 1 specify instruction length and data location (main storage or general registers).
- Bits 2 and 3 specify type of data (fixed or variable length, decimal, binary, or floating point).
- Bits 4 through 7 specify what to do with data.

Register to Register:

```
┌────┬───┐
│ Cp │   │
│Code│   │
└────┴───┘
```

Storage to Register:

```
┌────┬───┬────────┐
│ Cp │   │        │
│Code│   │        │
└────┴───┴────────┘
```

Storage to Storage:

```
┌────┬───┬────────┬────┐
│Cp  │   │        │    │
│Code│   │        │    │
└───┬┴───┴────────┴────┘
│   │   │        │
└───┘   └────────┘
 Byte     Halfword
```

Above you can see the three instruction lengths which are used depending on the location of data. One thing to note is that the first or high order byte of each instruction contains the Cp Code.

Cp codes in the System/360 give specific information:

1. The Op Code specifies the operation such as Add, Subtract or Branch.

2. The Cp Code specifies whether the data is variable or fixed in length.

3. The Op Code specifies whether the data is in binary or decimal format.

4. The Op Code specifies whether the operands are in main storage or general registers. The Cp Code does not give the address of data. It only indicates the data is in main storage, in general registers or in floating point registers.

5. The Op Code specifies the length of the instruction.

As you can see, there is much information in the 8 bits which make up the Cp Code. So let's break the Cp Code Byte down and see how it gives us this information.

The Op Code

```
      -----One Byte----

   ┌─────────────────┐
   │0 1 2 3 4 5 6 7│
   └─────────────────┘
```

Bits 0 and 1 of the Op Code specifies whether data is in main storage, in the general registers, or in floating point registers. Because the instruction length depends on the location of the data, the instruction length is also specified by bits 0 + 1 of the Cp Code.

There are four possible combinations of bits 0, 1 of the Op Code: 00, 01, 10, 11. If both bits are zero (00), both operands are in general registers or floating point registers and the instruction is one halfword in length. If bits 0, 1 of the Op Code are either 01 or 10, only one of the operands may be in main storage and the instruction is 2 halfwords in length. If both bits are one (11), both operands are in main storage and the instruction is 3 halfwords words in length.

Quite often the Op Code Byte is referred to as two hexadecimal digits rather than eight binary bits. Some examples of hexadecimal Cp Codes, their binary equivalent, and their instruction length, are shown below.

Hex.	Actual Op Code	Length in Halfwords
5A	01011010	Two Halfwords
4A	01001010	Two Halfwords
FA	11111010	Three Halfwords
18	00011000	Cne Halfword
58	01011000	Two Halfwords
94	10010100	Two Halfwords

GENERAL REGISTER ADDRESSING

- A general register has a 4 bit address.

- To address a general register, its address (0000 to 1111) is placed in the correct position of the instruction.

At this point, you should have a good idea of how the Op Code of an instruction specifies its length, the type of data, and what to do with the data. Before taking a further look at instruction format, let's examine how to address main storage and the general registers. We'll take a look at general register addressing first because of its simplicity. But first let's review general registers.

The programmer has sixteen general registers available for use as accumulators when working with fixed length binary operands. The general registers are numbered 0 to 15. A general register is one word in length. Besides being used as accumulators, the general registers can also be used as base registers and index registers.

As you learned when you were first introduced to the general registers, they are addressed by a 4 bit binary address. For example, the address of general register 7 is 0111. When both operands are in general registers, the instruction is 1 halfword in length. The first byte of an instruction is the Cp Code. When an instruction is one halfword in length the 2nd byte contains the addresses of the two general registers.

The following instruction shows how the general registers are addressed.

```
┌───────┬───────┬───────┬───────┐
│0 0 0 1│1 0 1 0│0 1 0 0│0 0 1 1│
└───────┴───────┴───────┴───────┘
 ─────── ─────── ─────── ───────
    ↑       ↑       ↑       ↑
    |       |       |       |
    |       |       |       |
 Reg. to    |       |       |
 Reg. Full  |       |       |
 Word       |       |       |
            |       |       |
   Add---- -┘       |       |
                    |       |
                    |       |
                 Reg. #4    |
                            |
                         Reg. #3
```

MAIN STORAGE ACDRESSING

- Storage addresses are generated by adding a displacement value to a base address.
- The instruction contains the displacement value as well as the address of the general register containing the base address.
- The general register that contains the base address is called the base register.
- 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.
- Some instructions may be indexed.
- The index value is contained in a general register.
- An indexed instruction contains the displacement value, and the address of the base register and index register.

Main storage addressing is a little more difficult. 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 length addresses. So we must look for another solution that will reduce the length of the instructions and still maintain complete campatibility.

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.

To see how main storage is addressed in the System/360 we must make some assumptions.

The first assumption is that System/360 programs will be written in sections. Each section will be 4096 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 base address for that section.

Consider the case of a program that required 12,000 bytes. By sectioning it into 4096 byte groups, we would have three sections of our program with a base address for each. The program could start anywhere, so for the example shown below, the program starts at Byte location 2,048.

```
    --┬----------┬----------┬----------┬---
      Section 1 Section 2 Section 3|
    --┴----------┴----------┴----------┴--
      ↑          ↑          ↑          ↑
      |          |          |          |
      |          |          |          |
      |          |          |          |
      |          |          |          |
      |          |          |          |
      ├----------┴----------┘          |
      |        Base                    |
      |        Addresses               |
      |                                |
      |                                |
      |                                |
Location                          Location
   2048                             14,047
```

Sectionalized Program in Main Storage

As can be seen in the above example our 12,000 byte program starts a location 2,048 and runs through location 14,047. We have divided the program into three sections. The first two sections are 4096 bytes each while the remainder of the program (the last 3,808 bytes) are in section 3.

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

Since 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 its displacement. That is, each byte is displaced from the base address by from 0 to 4095 places.

```
     Displacement          Base
0----from                  Address-----4095

┌-----------------\ \------------------┐
|    Section of    \ \Program          |
└------------------ \ \----------------┘
↑                    \ \
|
|
|
└----Base Address
```

Supposing that the program that we have been using as an example was moved so that it started at location 8192.

```
--┬----------┬----------┬----------┬---
  |Section 1|Section 2|Section 3|
  |          |          |          |
--┴----------┴----------┴----------┴----
  ↑          ↑          ↑
  |          |          |
  |          |          |
  |          |          |
 Location   Location   Location
   8192      12288      16384
```

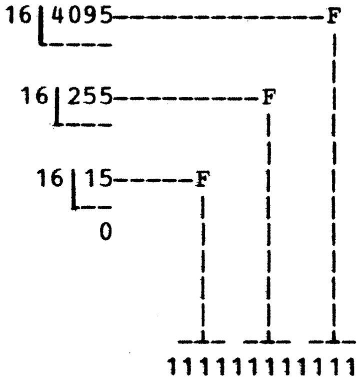
The base address for Section 1 is now 8192. The base addresses for Sections 2 and 3 are now 12,288 and 16,384. The displacement for each byte in the program has not changed. The last Byte of section 1 is still displaced from its base address by 4095.

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

As you know, main storage addresses are 24 bits long. This allows for compatibility throughout the range of

System/360 as well as for addressing up to 16 million bytes. Since a program can start anywhere in main storage, this means that the base addresses for the program must be 24 bits long.

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 certainly makes it easy to relocate a program each time it is loaded into the computer. However, we also wanted 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 base address, it is referred to as a base register. The address of the base register and the 12 bit displacement is 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 2 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       |  Distination
           Register      |  Operand
           Address  ---┘  Address

                      ---┐
3.   4 bit Base          |  Source
                         |  Operand
                         |
4.  12 bit Displace-     |  Address
           ment      ---┘
```

The instruction format for this type of operation would look like this:

```
┌─────────┬──────┬───────┬───────┬──────────┐
|0       7|8   11|12   15|16   19|20      31|
|   Op    |Gen   |       |Base   |Displace- |
|  Code   |Reg.  |       |Reg.   |ment      |
|         |Addr. |       |Addr.  |          |
└─────────┴──────┴───────┴───────┴──────────┘
```

Bits 12-15 of this instruction are used for further modification of main storage address. We will ignore them for the present and cover them later.

Given a displacement of 100110110010 and base register 11 whose contents are shown below, the effective storage binary address would be 010010001001110100100001.

```
 3                               31

┌────────────────────────────────┐
|00000000100100010010011011101111|
└────────────────────────────────┘
```

General Register 11

Remember that you add the 12 binary bit 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 remains unchanged.

As we said previously, 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.

```
┌───┬─────────┐
│ 0 │  1022   │
└───┴─────────┘
  ↑     ↑
  │     │
  │     │
  │     │
 Base  Displacement
 Reg.
```

```
                      ┌────┐
Contents of Reg. 0 is │2048│
                      └────┘
```

Given the above address portion in the instruction and the contents of Register 0, the effective storage address would be 1022. Because register 0 was specified as the Base Register, a Base Address of 0 is used. The contents of Reg. 0 is 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 also contained in a General Register.

In those instructions that include an indexing factor, the address portion looks like this:

```
┌─────┬─────┬────────────┐
│Index│Base │Displacement│
│ Reg.│Reg. │            │
│Addr.│Addr.│            │
└─────┴─────┴────────────┘
   4     4      12 bits
```

The effective storage address would be generated by adding:
1. Displacement.
2. Contents of Base Register.
3. Contents of Index Register.

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

```
┌───┬───┬─────────────┐
│ 6 │ 7 │    1012     │
└───┴───┴─────────────┘
```

Register 6 contains the value 2048, and register 7 contains the value 6024. We can derive the following values:

1. The effective storage address is 9084.

2. The address portion of the <u>instruction</u> is unchanged.

3. The values in the base and index registers are unchanged.

Thus the only thing we did was generate a storage address 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 remain unchanged, as does the displacement value in the instruction remain unchanged.

INSTRUCTION FORMATS

- There are 5 basic instruction formats.

- They are RR, RX, RS, SI and SS

```
   ┌────────┬──┬──┐
RR │Op Code │R1│R2│
   └────────┴──┴──┘

   ┌────────┬──┬──┬──┬──┐
RX │Op Code │R1│X2│B2│D2│
   └────────┴──┴──┴──┴──┘

   ┌────────┬──┬──┬──┬──┐
RS │Op Code │R1│R3│B2│D2│
   └────────┴──┴──┴──┴──┘

   ┌────────┬──────┬──┬──┐
SI │Op Code │  12  │B1│D1│
   └────────┴──────┴──┴──┘

   ┌────────┬──┬──┬──┬──┬──┬──┐
SS │Op Code │L1│L2│B1│D1│B2│D2│
   └────────┴──┴──┴──┴──┴──┴──┘
```

- In most operations, the first operand (R1 or B1, D1) is replaced by the results or the contents of the second operand (R2, I2 or B2, D2).

- The number in the length code in the SS format is always one less than the true length of the data field.

Let's take a look at the instruction formats of the System/360. As you know, the instructions are of 3 lengths: 1, 2 or 3 halfwords depending on the location of the operands.

RR FORMAT: A 1--halfword instruction is used when both operands are in two general registers or in two floating point registers. What is required is:

1. An 8 bit Op Code.

2. A 4 bit register address for 1st operand. (Destination)

3. A 4 bit register address for 2nd operand. (Source)

Instructions that involve register to register operations are considered to be of the RR format.

RR Format

Op Code	R1	R2

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 00.

The 2nd 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 2nd operand. The numbers in the address fields of the RR formats (and all other formats) indicate whether the operand is the 1st or 2nd (and is in some cases, the 3rd) operand. For most operations, the results replace the 1st operand.

RX FORMAT: Instructions which are two halfwords in length may have 3 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 is 01, it indicates a specific format known as the RX format.

RX Format

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

In the RX format, the effective address 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 above RX-type instruction, the storage address is generated by adding the low order 24 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 4 bits normally used for the X2 field are used for a 3rd Operand.

Op Code	R1	R3	B2	D2

RS Format

The RS Format 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 3rd operand. In some RS instructions, the R3 field is ignored. An example of an instruction which uses the R3 field is an instruction called <u>Load Multiple</u>. In the Load Multiple instruction, the data in main storage is loaded (or placed) into the 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

Load Multiple

In the above example the effective storage address is 0100. This is because register 0 is specified as the base register and its contents are ignored.

In the above example, registers 4 through 7 will be loaded with the data in main storage. As each register can hold one full word, registers 4-7 will be loaded with the data in storage location 0100 through 0115. (Each storage address represents a byte of data.)

SI FORMAT: There is another instruction format that is two halfwords in length. It is called the SI Format. 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 looks like this:

```
┌─────────┬───────┬───────┬───────┐
│ 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. In the SI Format the immediate operand is fixed in length and is one byte long.

An example of an SI Format is an instruction called Move Immediate. This instruction moves the immediate operand (I2) in the instruction to the storage location.

```
┌─────────┬───────┬───────┬───────┐
│ Op Code │  I2   │   0   │ 1000  │
└─────────┴───────┴───────┴───────┘
```

Move Immediate

In the above instruction, the contents of the I2 Field will be placed in storage location 1000. The SI Format is also identified by a 10 in bits 0 and 1 of the Op Code, just like the RS Format.

Since bits 0 and 1 of the Cp Code are the same for both the RS and SI formats, the remaining bits of the Op Code have to tell the computer whether it is the RS or SI Format.

SS FORMAT: In the four previous formats, the operands were of fixed length. Now let's take a look at the instruction format for variable length operations.

Variable length operation use a storage to storage concept. The instruction format is called the SS Format and looks like this:

```
┌──────────┬────┬────┬──────┬────┬────┐
│ Cp Code  L    │ B1 │  D1  │ B2 │ D2 │
└──────────┴────┐┴┌──┴──────┐┴┌──┴────┘
            │   │ │         │ │       │
            └───┘ └─────────┘ └───────┘
          Length  Location    Location
          Code    of 1st      of 2nd
                  Cperand     Operand
                  (Destina-   (Source)
                  tion)
```

The SS Format, because it must address two storage operands, is 3 halfwords in length.

Because both operands are in storage and the instruction is 3 halfwords in length, the SS Format 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 2nd 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.

Because all operands are at least one byte long, the length code is used to tell how many <u>additional</u> bytes are needed. For instance, a length code of 15 would tell us that the operand is 16 bytes long. If an operand is to be one byte long, the length code would be 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. It depends 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 type SS operations, the length code is split in two:

```
┌────────┬─────┬─────┬────┬─────┬─────┬─────┐
│Cp Code │ L1  │ L2  │B1  │ D1  │ B2  │ D2  │
└────────┴─────┴─────┴────┴─────┴─────┴─────┘
           ↑     ↑
           │     │
           │     │
Length of──┘   Length of
1st Oper-      2nd Oper-
and            and
```

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.

REVIEW QUESTICNS CN SYSTEM/360 INSTRUCTICNS

1. Instructions are a multiple of_________ in length.

2. Instruction addresses must be divisible by ____ or a __________ exception will occur.

3. The first byte of every instruction is the _____ _____.

4. For the following Op Codes (in hexadecimal code), indicate the binary bit structure of the Op Code and its length in halfwords.

	Op Code	Binary	Length
a.	1A	__________	__________
b.	56	__________	__________
c.	9C	__________	__________
d.	FD	__________	__________

5. All effective storage addresses are generated by adding a 12 bit____to a 24 bit_____ _____in one of the general registers.

6. Some effective storage addresses are generated by also including a (n)_____factor in one of the general registers.

7. Address generation (does/does not) change the contents of the general registers or the instruction in storage.

8. A program can be relocated in storage by changing the contents of the_____ _____.

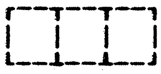
9. The displacement has a range of 0 to_____bytes.

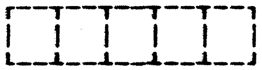
10. Cnly general registers_____to_____can be used as Base or index registers.

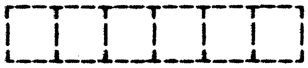
11. What happens if register 0 is specified as a base or index register?

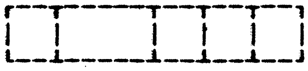
____

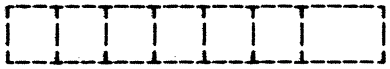
12. Label the fields of the following formats.

a. RR 

b. RX 

c. RS 

d. SI 

e. SS 

13. For most operations, the results replace the (1st/2nd) operand.

14. Given the following RR type instruction:

ADD	7	4

The result of the addition will replace the contents of register_____.

15. In the SI format, the 2nd operand is located in_____and is one_____long.

16. Only the_____format uses an index register for address generation.

17. Only the_____format involves variable length data.

18. What is the relaticnship between the number in the length code of the SS format and the number of bytes in the data field?

INSTRUCTICN SEQUENCING AND BRANCHING

- Unless otherwise specified, instructions are handled sequentially.
- Instructions are divided into two parts: Cp Code and address.
- Instructions are fetched from main storage during I-time and executed during E-time.

Coded information which causes a computer to perform a specific task (such as Add or Subtract) is called an instruction. A series of instructions used to solve a problem on a computer is called a program. A program is sometimes referred to as a stored program because of the fact that it is kept in main storage when it is executed. The instructions of the stored program are read out of main storage, 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 then executed in the Arithmetic Logic Unit (ALU) section of the CPU. For every instruction, there are two periods of time. The time during which the instruction is read out (fetched) from main storage and interpreted is known as I-time. The operation specified by the instruction is performed during E-time (or execution time). Data is the name generally given to information read out of main storage during E-time. Instructions are information 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.

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.

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 purpose of our discussion, we can think of I-time as being separate from E-time.

Instructions are generally thought of as having two parts. One part of the instruction is used to tell the computer what to do (such as Add or Branch). The portion of the instruction that tells the computer what to do is known as the Cp Code. The other portion of the instruction generally tells the computer where the data is located. For this reason it is called the address portion.

The address portion of an instruction may contain other information besides data addresses. In a branch instruction, it would give the address of the next instruction to be executed. In some instructions, the data to be operated on may be contained in the address portion. Let's continue now with the study of the System/360 and its Program Status Word (PSW).

INSTRUCTION ADDRESS FIELD

- The Program Status Word (PSW) is a double-word containing 8 bytes, or 64 bits of control and status information.

- The PSW is maintained as part of the internal 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). 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 PSW is a doubleword and contains 8 bytes of information. Like all doublewords, the bits of the PSW are numbered 0 to 63 from left to right.

```
     0-------------------63

  ┌─────────────────────┐
  │PROGRAM STATUS WORD  │
  └─────────────────────┘
```

For right now let us examine only one portion of the PSW.

The location of the next instruction to be fetched from main storage is indicated by bits 40-63 of the PSW. As you learned earlier, main storage requires binary addresses. Bits 40-63 of the PSW contain the 24 bit binary main storage address of the next sequential instruction.

```
0       39 40                          63
┌─────────┬─────────────────────────────┐
│         │                             │
└─────────┴─────────────────────────────┘
          24-Bit Instruction Address
```

The PSW is a doubleword which reflects the status and controls the program <u>currently</u> being executed. For this reason, it is often referred to as the <u>current</u> PSW.

Before examining more of the current PSW, you may be wondering where this doubleword is kept. For one thing, the current PSW does not use any of the 16 general registers or addressable locations in main storage. It is kept in some internal area or 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 doubleword of information.

The instruction address portion of the current PSW must be 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 must be updated. Since an RR type instruction is one halfword in length, the location of the next sequential instruction would be 1002. Thus the instruction address portion of the PSW must be updated to contain 1002.

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

Since instruction length is always a multiple of halfwords, the instruction address portion of the current PSW is always updated by some multiple of 2. 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 will be 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.

You should be familiar with the use of flow charts in writing a program. Decision blocks in a program are represented by a diamond shaped symbol.

The use of this symbol in a program represents a decision as to what to do next. Should the program continue with its present sequence of instructions, or should it branch out to another sequence of instructions? Sometimes a decision block represents leaving a sequence of instructions. In this case 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. What happens to the instruction address portion of the current PSW when a branch is taken? 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 would normally be changed to 1004. If however, the instruction at 1000 says to branch to location 2000, the instruction address portion of the current PSW will be changed to 2000.

In the above example bits 40-63 (the instruction address) of the current PSW might actually be updated to 1004 and then changed to 2000. This depends on the particular branch-type instruction. However, at the time the system decides that it will 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 instructions such as add, subtract, compare, etc.
- Not all instructions affect the condition code.

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

```
          34,35
           | |
           | |
           | |
0 ------33 | |    40-----------------38
┌─────────┬─┴─┬─┬──────────────────────┐
|         | C | | Instruction          |
|         | C | |   Address            |
└─────────┴───┴─┴──────────────────────┘
        Condition Code
```

The condition code can have four possible bit combinations:

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

The condition code is set to one of its four possible 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 4 possible results of an algebraic add or subtract. The result could be a 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 in the PSW retains this setting until the end of the next instruction that can change the condition code. Remember that not all instructions affect the condition code.

Another use of the condition code is to indicate the result of a compare operation. A compare operation consists of comparing the 1st operand to the 2nd operand. The condition code is set to indicate the result. Neither operand is changed. The condition code is set and indicates whether the 1st operand is equal to, less than, or greater than the 2nd operand as follows:

Condition Code	Comparison
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. Actual meaning of the condition code depends on the instruction it represents.

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 its 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.

Op Codes in Hexadecimal			
	07	R1	R2

"Branch to" location is in general register specified by R2 field.

47	R1	X2	B2	D2
Branch on Condition	Mask Field		Effective Address is Branch to location	

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.

As you know, the condition code can mean many things. For instance, it could indicate a low or equal compare, a negative arithmetic result, an over-flow and so forth. However, it can have only one setting (00, 01, 10, 11) at any one time. This setting can represent only one thing depending on the last instruction that affected the condition code.

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

As you can see from the above 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 of condition code contain 1000, a branch would occur only if the condition code had a setting of 00. If the condition code were 01 and the mask field were 0010, a branch would not occur.

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

Condition Code	Referred To As
00	0
01	1
10	2
11	3

The bits of the branch on condition mask field correspond to the condition code settings in a left to right fashion.

```
8----------11

┌─────────────┐
|1   1   1   1|      Mask Field
└─────────────┘

0,  1,  2,  3        Condition Code
```

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 will be tested. Therefore, a branch never occurs if the mask field contains 0000.

If the mask field contains 1111, all or any of the possible condition code settings is tested. Because the condition code always contains one of its four possible settings, a mask field of 1111 always results in a branch.

SUMMARY:

With its 4 bit mask field, the branch on condition instruction has many uses:

a. It can be used as a no-op instruction, by having a mask field of 0000.

b. It can test for a specific result (such as an equal compare) by setting <u>one</u> of the bits of the mask field (1000).

c. It can test for a multiple result (such as an equal or low compare) by setting two or more bits of the mask field (1100).

d. It can be used as an unconditional branch by having a mask field of 1111.

<u>REVIEW QUESTIONS ON INSTRUCTION SEQUENCING AND BRANCHING</u>

1. PSW is short for _____ _____ _____.

2. The PSW is _____ bits long.

3. The address of the next instruction to be fetched is contained in bits _____ to _____ of the PSW.

4. After an instruction has been fetched (read out) the instruction address portion of the PSW is usually incremented by _____, _____, or _____.

5. The amount to increase the instruction address in the PSW is determined by bits _____, _____ of the instructions __________.

6. The PSW which is being used to fetch instructions is sometimes referred to as the "_______" PSW.

7. This PSW is located in (main storage/ a general register/some type of internal register or storage area).

8. Branching is accomplished by replacing the _____ _____ in the PSW with the "Branch to" ___.

9. The condition code is in bits ___, _____ of the PSW.

10. The condition code has _____ possible settings.

11. The condition code in the PSW is changed by (all/some) instructions that are executed.

12. Indicate the condition code setting after the following results:

Algebraic Add	Compare	Condition Code
zero	Equal	----------
<Zero	Low	----------
>zero	High	----------
overflow		----------

13. Following an algebraic add instruction, the following "Branch on condition" instruction would test for what result?

Branch on Condition	07	4	5

ANSWER: ________________

14. Following a compare instruction, the following instruction would test for what result?

Branch on Condition	07	C	8

ANSWER: ________________

15. The following instruction would (always/never) result in a branch.

Branch on Condition	07	F	7

ANSWER: ________________

16. If the following instruction resulted in a branch, the instruction address in the PSW would be replaced by __________.

Branch on Condition	07	8	5

17. If the following instruction resulted in a branch, the instruction address in the PSW would be replaced by __________.

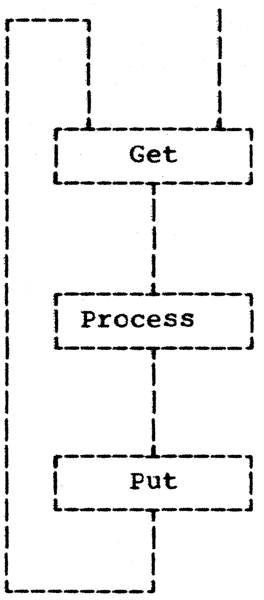
Branch on Condition	47	8	4	7	000

THE SYSTEM/360 AND INTERRUPTS

THE SUPERVISOR CONCEPTS

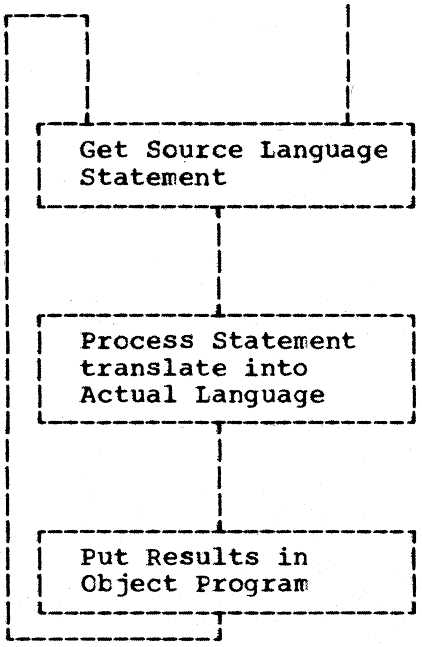
- Extensive programming features dictate the need for an operating system.
- In an operating system environment, control programs perform such functions as program loading, storage protection, I/O operations, interrupt handling, job flow, and operator communications.
- One control program (the supervisor), remains in core storage at all times.
- Basic functions of the supervisor program are I/O control and interrupt handling.

A program is a sequence of instructions designed to solve a problem. A problem typical of those solved by a stored program is a payroll application. A payroll problem would consist of (1) getting an employee's record, (2) calculating gross and net pay, and (3) putting the results out in the form of a pay check. The payroll problem would then get the next employee's record and repeat the process. This sequence of instructions would continue until all employee's records had been processed. Admittedly, this is a gross simplification of a payroll problem. However, most programs can be broken down into the three operations of (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 operations are basically 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).


Get Source Language Statement
Process Statement translate into Actual Language
Put Results in Object Program

During the past years, data processing machines have been developed with faster and faster internal processing speeds. As a result, the execution times for these problem programs have been continually reduced with no corresponding reduction in the time it took for an operator to load in the next problem program and manually set up its input data. In some data processing installations, the average "set up" time was about equal to the average "execution" time. In other words, the data processing system was idle about half the time while the operator was "Setting up" for the next problem program. Clearly this was an ineffecient way to control an installation. In an attempt to reduce this idle time and keep the system running, installation began to use stored programs to control the execution of problem programs. These programs in turn were called Control Programs. Other names used were "monitors" of "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 then tailored to the requirements of each installation. The simplest type of control program would be used to supervise the loading of problem programs. It would operate like this:

1. An input tape would be prepared containing the problem programs and their associated data. See Figure 1-26.

2. The operator would load the control program into main storage.

3. The control program would load in the 1st problem program and then pass control (via a branch) to the problem program.

4. The problem program would read in its data and perform its assigned task.

5. When the problem program is finished, it would not issue a halt instruction. Instead it would pass control (by branching) back to the control program.

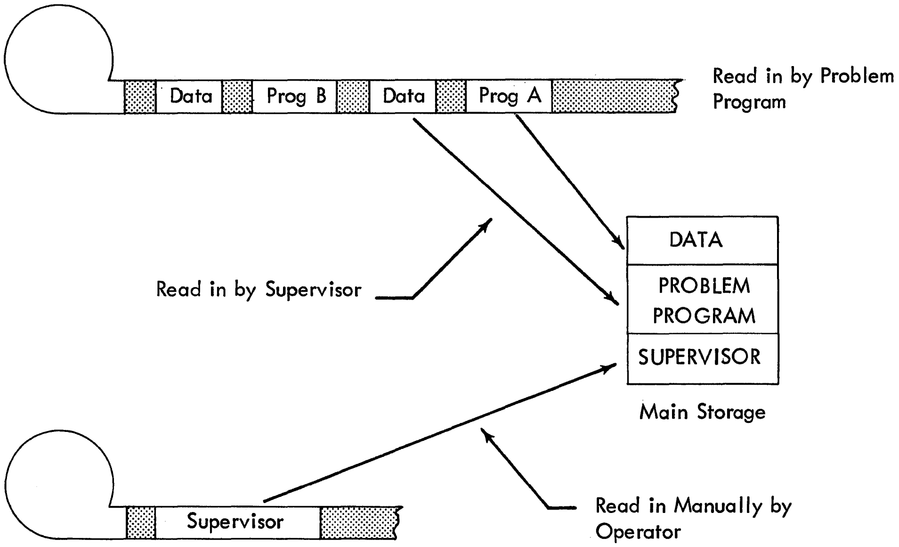


Figure 1-26. Sample Control Program

6. The control program would then load in the next problem program and pass control to it.

7. This operation would continue until all problem programs had been executed.

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

1. The system never halted between jobs.

2. The control program remained in main storage as the problem programs were executed.

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

In the simplest use of the control program, it was used only to bring in the next problem program. The problem programs handled their own input-output operations (Figure 1-27).

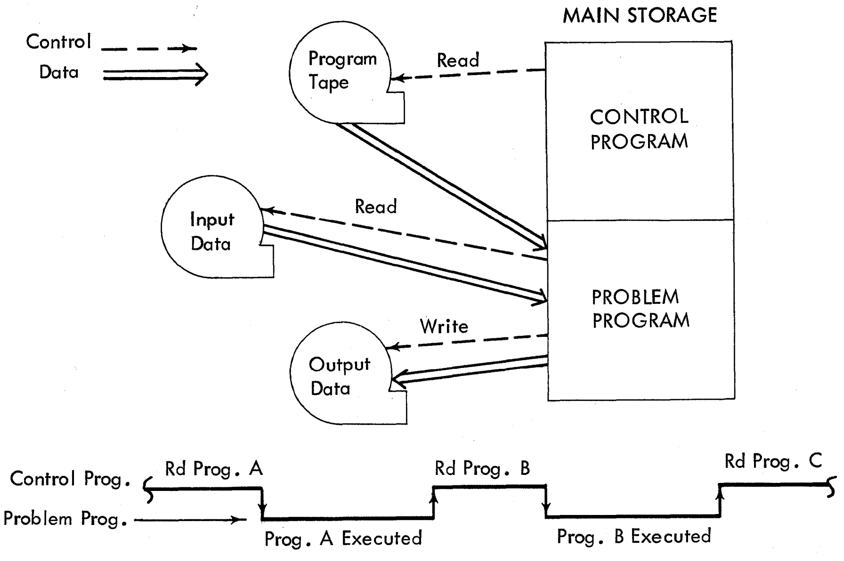


Figure 1-27. Basic Control Program

This is one example of the use of a control program. Its functions were limited. As such the entire control program could be left in main storage. Other functions 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 the input-output devices and main storage can be handled by the control program. (Figure 1-28).

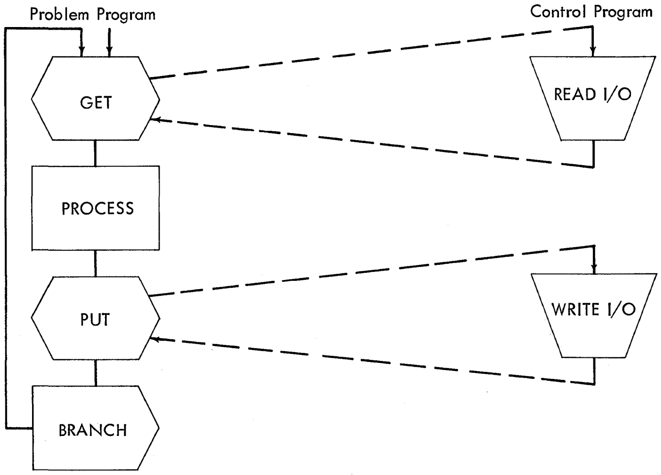


Figure 1-28. Control Program Handling I/C

In this function of a control program, control will pass back and forth between the problem and control programs during the execution of the problem program.

This differs from the original example of using the control program just to load in new problem programs.

In that example, the only time the control program was in control was between jobs. In our new example, the control program will not only read in new problem programs when necessary, it will also during the execution of the problem program, be used to start the necessary I/C units for input-output data (Figure 1-29). So in the control program concept, there are always two programs in main storage: the control program and a problem program.

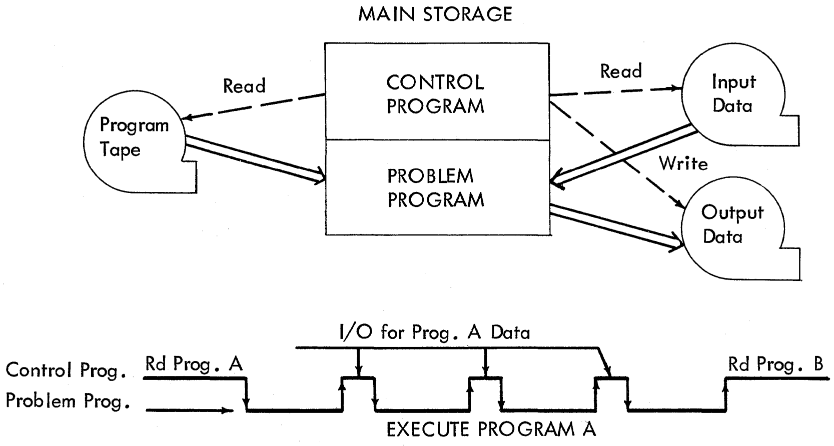


Figure 1-29. Control Program Sequencing

The control program can be given other functions as well. In fact, some control programs have reached a very high degree of sophistication. Of course, the more functions that a control program has, the more main storage space it requires. This leaves less storage for problem programs. This problem is somewhat solved by placing those sections of the control program that have infrequent usage 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 other sections of the control program when necessary.

In the simplest use of the control program, it was used only to bring in the next problem program. The problem programs handled their own input-output operations (Figure 1-27).

In review then, control programs have come in 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.

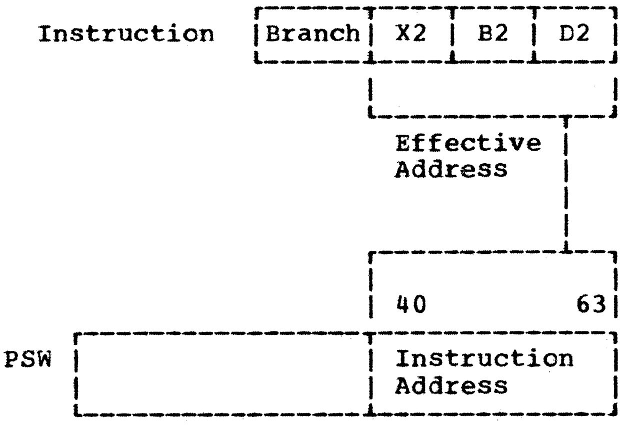
INTERRUPTS AND THE PSW

- An interrupt terminates the current sequence of instructions and causes a machine forced "branch" to the supervisor program.
- An interrupt stores the current PSW in main storage, and fetches a new PSW from main storage.
- Processing resumes at the instruction address specified by the instruction address portion of the current PSW just loaded.
- There are five classes of interrupts. Each has an old and new PSW locations in main storage.

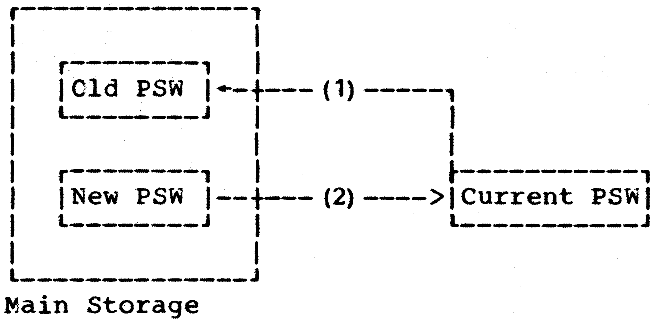
So far we have discussed the use of a control program to bring in new problem programs when the old ones are finished. Because there is no halt instruction in System/360, a problem program when finished must be able to somehow branch into the supervisor. Also when a machine or program check occurs, an automatic branch to the supervisor usually occur.

These automatic branches into the supervisor are called interrupts. 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 interrupts. When a problem program is finished, it signals the supervisor via an interrupt.

An interrupt is quite similar to a branch. However, it does much more than a simple branch instruction. A branch instruction only replaces the instruction address portion of the current PSW.



When an interrupt occurs: (1) the current PSW is placed in main storage where it's called old PSW and (2) a new PSW is brought out of main storage and becomes the current PSW.



Assuming that the instruction address portion of a new PSW contains 1096, the 1st instruction after an interrupt would be at location 1096.

By now you should realize these new and old PSW's are in fixed doubleword locations in main storage. Just where are these locations? The answer will depend on just what class of interrupt it is. There are five distinct classes of interrupts:

1. External — Can be caused by pressing an interrupt key on the operator's console.

2. Supervisor — Caused by an instruction known as supervisor call.

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 above five classes of interrupts has its own main storage locations for new and old PSW's as follows:

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

For example a machine check causes the PSW to be placed in location 0048 and a new PSW to be brought out from location 0112. Notice that these locations are all divisible by eight since they contain doublewords, and must reside on doubleword boundaries. Also, each new

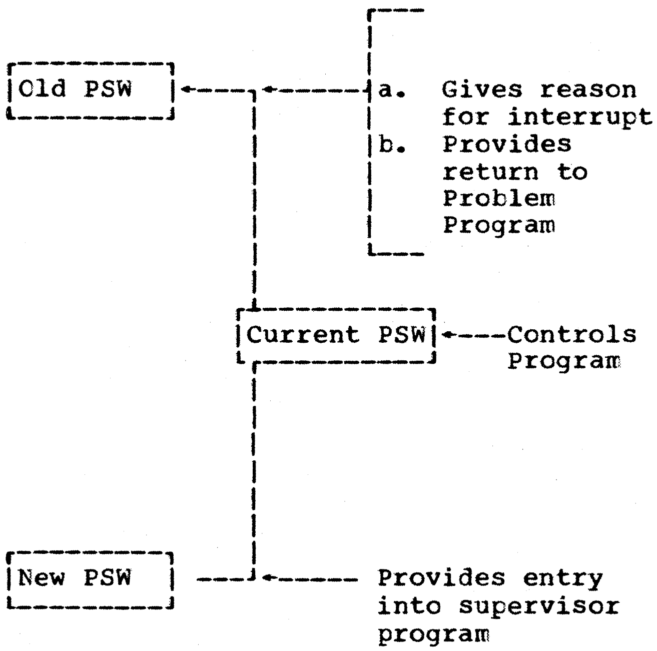
Interruption Source Identification	Interruption Code PSW Bits 16-31		Mask Bits	ILC Set	Instruction Execution
Input/Output (Old PSW 56, New PSW 120)					
Multiplex Channel	00000000	aaaaaaaa	0	x	Complete
Selector Channel 1	00000001	aaaaaaaa	1	x	Complete
Selector Channel 2	00000010	aaaaaaaa	2	x	Complete
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)				a = I/O Device Address	
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
				r = Bits 8-15 of Supervisor Call Instruction	
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)				x = Unpredictable	
Machine Malfunction	00000000	00000000	13	x	Terminate

Figure 1-30. Interruption Code and Action Chart

PSW is located 64 higher than the corresponding old PSW.

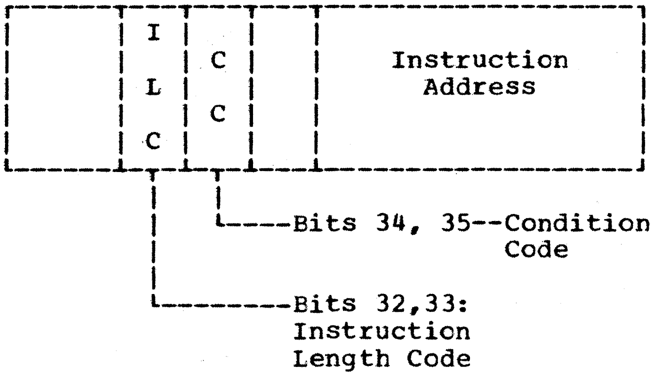
Although an interrupt may be initiated by an instruction (such as when the instruction supervisor call initiates a supervisor interrupt), the actual storing and loading of the PSW is done automatically by the internal circuitry of the System/360.

Although we state that interrupts occur only at the end of an instruction and never in the middle of one, this might be a little misleading. It is definitely and absolutely true that the current instruction will be completed before an I/O, external, or supervisor call interrupt is taken. In the case of program and machine interrupts (which indicate programming and hardware errors, respectively), the interrupt still occurs at the end of the instruction. However, in these two cases, the end may be forced by suppressing the instruction's execution where a programming error is detected during instruction fetch time or by terminating its execution when a programming or machine error is detected during execution time (Figure 1-30). The branch is effected automatically by the internal circuitry of System/360. 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 interrupt 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.


Old PSW
a. Gives reason for interrupt
b. Provides return to Problem Program
Current PSW
Controls Program
New PSW
Provides entry into supervisor program

PSW INSTRUCTION LENGTH FIELD: As you will recall from our earlier discussion of the PSW, the instruction address portion of the PSW is used to read out an instruction. Cnce the instruction has been read out, the instruction address portion of the PSW is updated so as to point to the next instruction. Interrupts can only occur 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 will contain the address of the next instruction that would have been executed if the interrupt had not occurred.

When the interrupt is complete, the supervisor may elect to return to the point of departure from the machine language program. It does this by looking at the old PSW from the last instruction executed before the interrupt occurred. In some cases, the supervisor may have to do that last instruction over. Because the instruction address portion of the old PSW had been updated before the interrupt occurred, 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 PSW and is called the instruction length code.


I
L
C
C
C
Instruction Address
Bits 34, 35--Condition Code
Bits 32,33: Instruction Length Code

When the supervisor retrieves the old PSW to determine where to re-enter the machine language program the instruction length code indicates what value must be subtracted from the old PSW instruction address field to produce the address of the last op code executed before the interrupt occurred. The instruction length code is valid only on certain types of interrupts. It is the responsibility of the supervisor to determine if this information is to be used.

Bits 32-33 of the PSW are set to 1, 2, or 3 depending on 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, an RX-format instruction would cause the instruction length code in the PSW to be set to 10 (2).

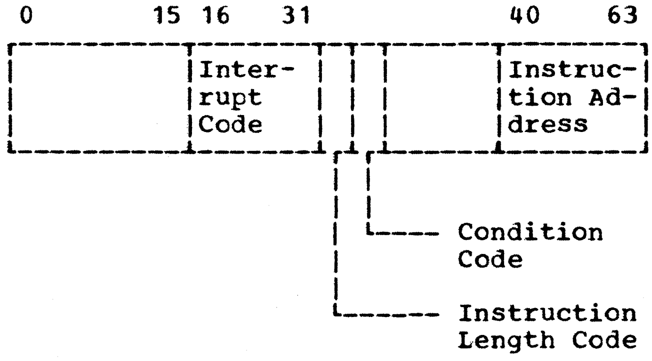
If the instruction address portion of the old PSW contains 4000 and the instruction length code contains 3, the op code of the last instruction prior to the interrupt is located at 3994.

There are five classes of interrupts. Each class of interrupt has its own handling routine. Each of these interrupt handling routines handle the interrupts in a different way. Not all of them are interested in what the last instruction executed was. In the case of program, machine or supervisor interrupts, an instruction in the problem program caused the interrupt.

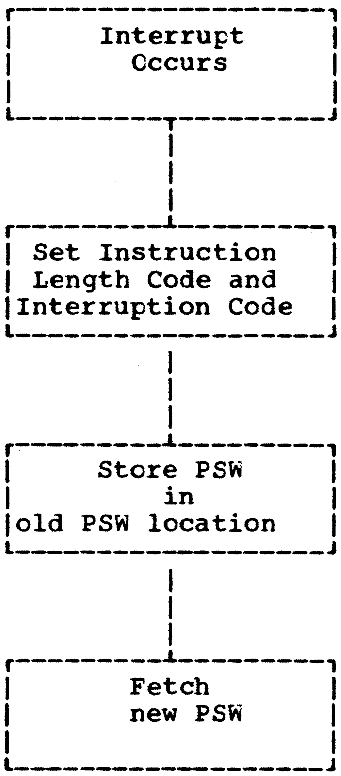
In the case of external and I/O interrupts, the problem program did not cause the interrupts. As a result, the supervisor could care less what instruction was last executed in the problem program. It would only want to be able to return to the next instruction.

PSW INTERRUPTION CODE FIELD:

Another field in the PSW that is of value to the supervisor is the interruption code field. It appears in bits 16-31 of the PSW.



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


Interrupt Occurs
Set Instruction Length Code and Interruption Code
Store PSW in old PSW location
Fetch new PSW

The five classes of interrupts tell the supervisor only the general reason for the interrupt. For instance, the fact that the new PSW was brought out of location 0040 tells the supervisor that the interrupt 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 type of exception. In the case of I/O interrupts, the interruption code tells the supervisor what channel and I/O unit caused the I/O interrupt. (Figure 1-30).

For example, when a program interrupt is caused by a fixed point overflow, the interruption code of the old PSW contains 0000000000001000. (Refer to Figure 1-30.)

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 of course, that each of five new PSW's point to different routines in the supervisor. These routines in turn use the old PSW that corresponds to the particular class of interrupt. For instance; the program check routine in the supervisor uses the old PSW at location 0040 while the supervisor call routine will use the old PSW at location 0032.

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

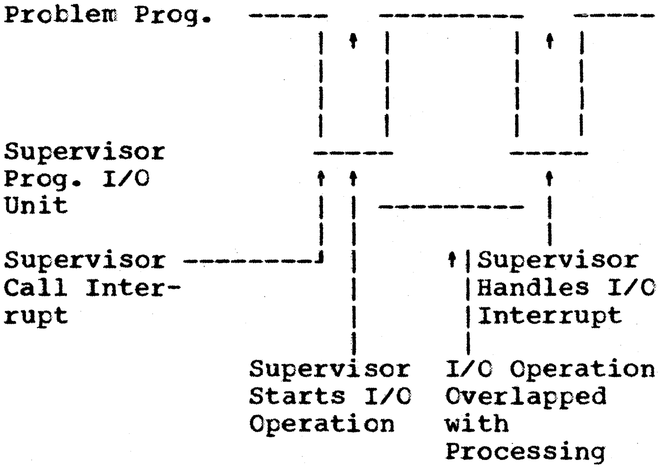
In the case of an interrupt caused by a machine check, the PSW that was controlling the program prior to the interrupt is stored automatically in location 0048. Then the doubleword at location 0112 is brought out and becomes the current PSW. This PSW directs the system to that area of the supervisor 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 interrupt caused by a program check, the current PSW at the time the interrupt occurs is stored automatically as the old PSW at location 0040. Then the doubleword at location 0140 is brought out and becomes the current PSW. This PSW directs the system to that area of the supervisor that handles program checks. The program check handling routine of the supervisor is written so that the doubleword at location 0040 will be processed as the old PSW.

In the case of an interrupt caused by the instruction supervisor call, the current PSW (prior to the interrupt) 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 which 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 operator's console is pressed, an external interrupt occurs. In this case, the current PSW is automatically stored at location 0024. On an external interrupt, the doubleword at location 0088 is brought out and becomes the new current PSW.

An interrupt may also be caused by the end of an I/O operation. I/C interrupts generally occur at the end of an I/C operation. Most I/C operations are overlapped with processing. The I/C interrupt is an efficient way of signalling the supervisor that the I/C operation is finished. An I/O interrupt causes the 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 interrupts.


Problem Prog.
Supervisor Prog. I/O Unit
Supervisor Call Interrupt
Supervisor Handles I/C Interrupt
Supervisor Starts I/C Operation
I/C Operation Overlapped with Processing

At the end of the I/C operation, the I/C interrupt causes the PSW used in the processing of the problem program to be stored in location 0056. The doubleword at location 0120 is brought out of main storage and used as the PSW to control the processing of the I/C interrupt routine in the supervisor program.

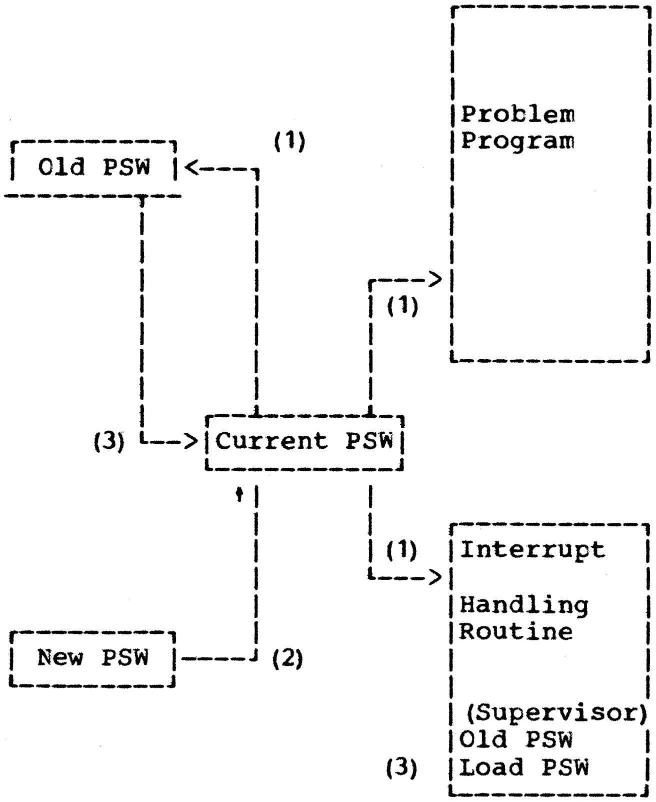
LOAD PSW INSTRUCTION

- The Load PSW (LPSW) instruction is USED TO RETURN TO THE PROBLEM PROGRAM AFTER AN INTERRUPT.

- The Load PSW is in the SI format. The I2 field is ignored.

- A doubleword is loaded into the current PSW from an address in main storage by the Load PSW instruction.

After the end of the I/O interrupt 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 effects 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 interrupt routine would not necessarily be the same as it was before the I/O interrupt 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 interrupt 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 interrupt handling routine. Note that this return to the problem program by replacing the PSW is done by means of an instruction (load PSW) and is not automatic as was an interrupt.



As can be seen from the above diagram, interrupt action is as follows:

1. At the time of the interrupt, the current PSW which is controlling the problem program is stored in the Old PSW location. The Old PSW gives the reason for the interrupt. The instruction address portion of the old PSW indicates the point at which we left the problem program. This is done automatically by machine circuits.

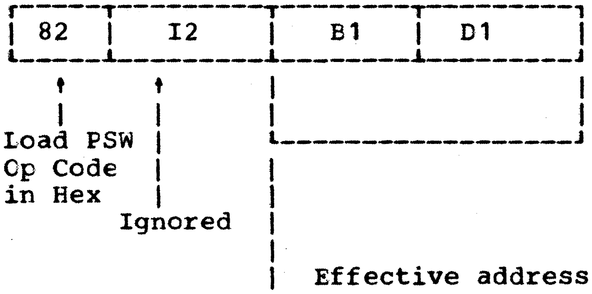
2. A new PSW is then brought out of storage and becomes the current PSW. This new PSW points to the first instruction of the interrupt handling routine which is part of the supervisor program.

3. After the interrupt has been taken care of, the last instruction of the interrupt handling routine will be load PSW. This causes the old PSW to become the current PSW and we are back in the problem program.

The Load PSW instruction is of the SI format:

Op Code	I2	B1	D1

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



Effective address of double word that is to be loaded as the PSW. Note that the current PSW at the time this instruction was fetched is not stored anywhere and is therfore lost.

The Load PSW instruction can be used by a supervisor program any time it wants to change the current PSW. The main uses is to return to the problem program after an I/O, supervisor call, or external interrupt has been serviced. It could also be used to load the PSW for a new problem program after it has been read into the machine by the supervisor program.

To return to a problem program after an I/O interrupt has been serviced, the effective address generated by the B1 and D1 fields of a Load PSW instruction should be 0056 (Figure 1-30).

SUPERVISOR CALL INSTRUCTION

- The Supervisor Call (SVC) is used by the problem program to pass control to the supervisor program by causing a Supervisor Call interruption.

- The Supervisor Call instruction is of the RR format.

- The R1 and R2 fields of a Supervisor Call instruction are placed in the interruption code field of the SVC old PSW.

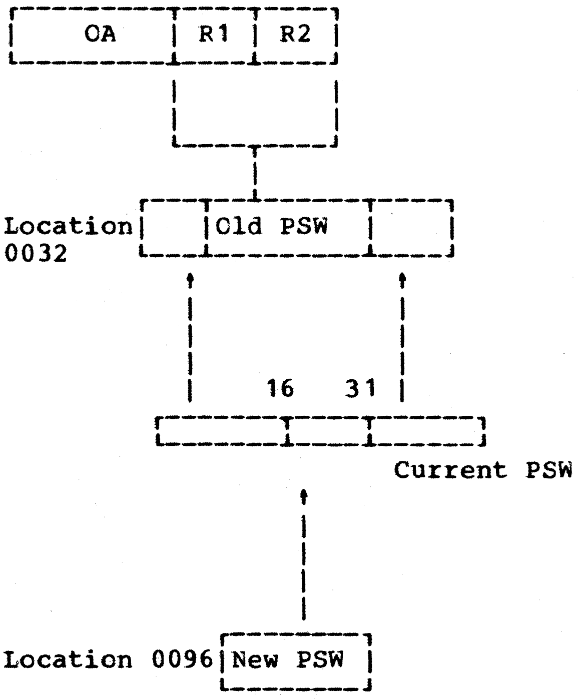
The supervisor call interrupt is used by the problem program to pass control to the supervisor program. There are a number of reasons why the problem program might want to call the supervisor program. Two of the major reasons are:

1. To tell the supervisor program that it (the problem program) is done. The supervisor could then read in a new problem program and load its PSW.

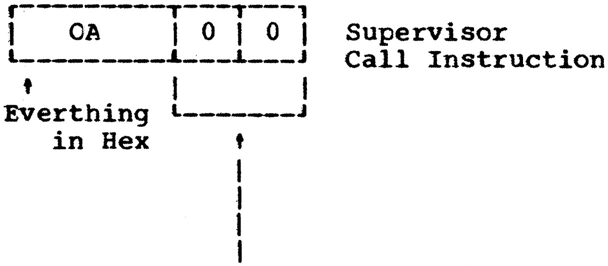
2. To request.the supervisor program to start an I/O operation for the problem program. The supervisor call instruction is of the RR format.

Op Code	R1	R2

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

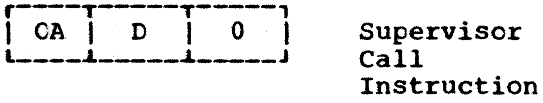


Because the bits of R1 and R2 field are stored as the interruption code, they can be used as pre-arranged signals to tell the supervisor program the reason for the interrupt. These pre-arranged signals vary depending on who wrote the supervisor program. For instance:



This interruption code of 00 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 placed in the interruption code of the old PSW is 11010000 (bits 16-31 of the old PSW in location 0032).



MASKING INTERRUPTS

- Interrupts are prevented from occurring by mask bits in the current PSW.

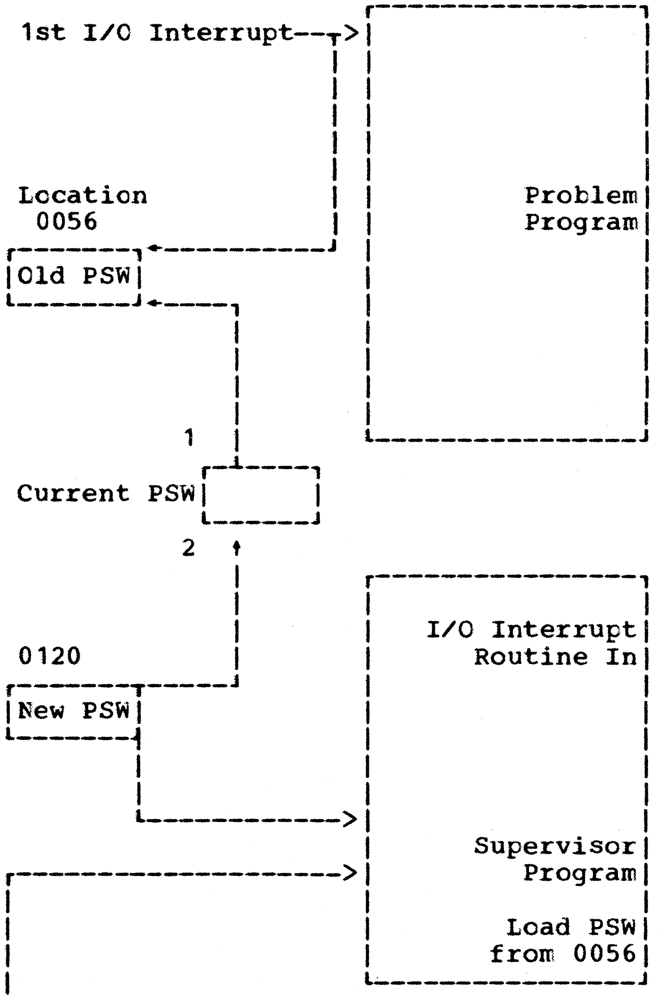
- The system mask is in bits 0 to 7 of the PSW. These are used to prevent (mask) external and I/O interrupts from interrupting faster than the supervisor can handle them.

- The machine check mask is bit 13 of the PSW. It allows machine errors to be ignored. Machine checks are not normally masked off except as a diagnostic aid.

- The program mask is bits 36 to 39 of the PSW. They are used to prevent 4 of the 16 program checks from causing interrupts.

- 11 program check interrupts, and the supervisor call interrupt, cannot be masked.

Sometimes in a program, it is not desirable to allow an interrupt. This is most apparent when we consider the I/O interrupt. 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 interrupt will usually occur. The PSW will be stored to give the supervisor program the reason (which I/O unit) for the interrupt. This old PSW also gives the supervisor program a way in which to return to the interrupted problem program. If we were to allow another I/O interrupt before the first one had been completely handled, the old PSW (from the problem program) would be lost. The supervisor program would not be able to return to the problem program via the Load PSW instruction as shown below.



If a 2nd I/O interrupt were allowed to occur here, the current PSW at this point would be stored in location 0056. This would cause the Old PSW from the problem program to be destroyed.

SYSTEM MASK: How does the supervisor program prevent this 2nd and undesirable I/O interrupt until it has processed the first one? It does this by proper usage of mask bits in the PSW.

```
 0    7    13  16   31      36   40        63
┌────┬────┬───────────┬─┬─┬──────┬───────────┐
│    │    │Inter-     │I│ │      │Instruc-   │
│    │    │ruption    │L│C│      │tion       │
│    │    │Code       │C│C│      │Address    │
└────┴────┴───────────┴─┴─┴──────┴───────────┘
              ↑                ↑
System   Machine           Program
Mask    Check Mask         Mask
```

As can be seen above:

1. Bits 0-7 are known as the system mask bits
2. Bit 13 is the machine check mask bit
3. Bits 36-39 are known as the program mask bits.

When these mask bits are set to zero, the corresponding interrupts are masked or prevented. Let's first consider the system mask bits. These eight bits can be used selectively or collectively to mask all I/O and external interrupts as follows:

PSW Bits 0-7	System Mask
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

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

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

A system mask of 00111110 would mask some I/O and all external interrupts. A system mask of 10000001 would prevent I/O interrupts by all selector channels. To prevent all I/O and external interrupts, the system mask must contain all zeros.

The system mask that determines whether or not to mask I/O and external interrupts is in the current PSW. In the case of an I/O interrupt, the channel causing the interrupt will be stored in the interruption code of the old PSW.

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

```
 0  7           From Problem Program
┌────┬────┐         │
│ FF │    ├─────────┴────────────┐
└────┴────┘                      │
 ↑  Old PSW        1.            │
 │                     ┌────┬────┴─┐
System Mask in Hex─────┤ FF │      │ Current
 │                     └────┴──┬───┘ PSW
0│  7              2.          │
┌┴───┬────┐                    │
│00  │    ├────────────────────┘
└────┴────┘        ↑
    New PSW     To Supervisor Program
```

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

The last instruction in the I-O interrupt routine of the supervisor program would be Load PSW. The old PSW in main storage would be brought out and placed back in action as the current PSW. Once this is done I/O interrupt can once more occur. This is because the system mask of the problem program's PSW would probably contain all ones (FF). Of course, a system mask of all ones would allow not only I/O interrupts but also external interrupts.

MACHINE CHECK MASK:

```
┌────┬────┬───────────┬─┬─┬──────┬───────────┐
│    │    │Inter-     │I│ │      │Instruc-   │
│    │    │ruption    │L│C│      │tion       │
│    │    │Code       │C│C│      │Address    │
└────┴────┴───────────┴─┴─┴──────┴───────────┘
              ↑                ↑
System   Machine           Program
Mask    Check Mask         Mask
```

A machine check interrupt can be masked by means of bit 13 of the PSW. If this bit contains a zero, machine checks will be ignored, and no machine interrupt will occur. Of course, this is not the usual state of the machine check mask bit. It is usually set to one, so that machine checks will cause an interrupt. In addition there is a switch on the CE section of the system control panel that can be used to cause an error stop rather than have an interrupt when a machine check occurs. The usual mode of operation is to have this switch off 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 a machine interrupt occurs.

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

1. A machine interrupt; the PSW is stored in location 0048 and a new PSW is fetched from location 0112.

2. An error halt.

3. It is ignored if PSW bit 13 is zero.

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. This log out occurs prior to the machine interrupt or error stop.

Just how much information is contained in a log out and what it means will depend on the particular model of System/360. However, log out always occurs prior to a machine interrupt and places information in storage starting at location 0128. The size of this log out area varies from 4 bytes to 256 bytes, depending on the System/360 model. This 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.

PRCGRAM MASK: Program checks (such as a specification exception) also can cause an interrupt. While machine checks cause machine interrupts, program checks will cause a program interrupt. Cn a program interrupt, the PSW is stored in location 0040 and a new PSW: is fetched from location 0040 and a new PSW: is fetched from location 0104. Program interrupts can also be masked by use of bits 36-39 of the PSW.

```
┌────┬────┬─────────┬─┬─┬────┬───────────┐
│    │    │Inter-   │I│ │    │Instruc-   │
│    │    │ruption  │L│C│    │tion       │
│    │    │Code     │C│C│    │Address    │
└────┴────┴─────────┴─┴─┴────┴───────────┘
            ↑                ↑
System  Machine          Program
Mask   Check Mack        Mask
```

There are 15 possible exceptions which can cause a program check (Figure 1-30).

On occasion, four of these may not be considered as program checks. These four exceptions are:

1. Fixed Point Cverflow

2. Decimal Cverflow

3. Exponent Underflow⎫Concerned with

4. Significance ⎭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 interrupts caused by the four exceptions mentioned earlier as follows:

```
             36      39
            ┌───────┐
            │0 0 0 0│
            └───────┘
Fixed Point  ↑ ↑ ↑ ↑  Program Mask
Cverflow ------- │ │  ---Significance
                 │ │
                 │ │
Decimal----------┘ └------Exponent
Cverflow                  Underflow
```

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

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

SYSTEM/360 STATUS BITS

- 3 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 will be 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 interrupt 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 are only allowed in the Supervisor state. A Program interrupt will occur in the Problem state, if execution of a privileged instruction is attempted.

```
0                                         63
┌─────────┬────┬──────────────────────────┐
│         │    │                          │
└─────────┴────┴──────────────────────────┘
          │    │
          │    │
    ┌─────┘    └───┐
    │              │
    │              │
    ├──────────────┤
    │ A  M  W  P   │
    └──────────────┘
     12        15
```

ASCII MODE BIT: Of bits 12-15, you are already familiar with bit 13. It is the machine check mask 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 will be done; EBCDIC or ASCII mode. A key difference is in the representation of sign values. If bit 12 of the PSW contains a one, the ASCII sign codes will be internally generated rather than the extended BCD codes for signs. For example:

The number 1 in EBCDIC looks like this:

```
1 1 1 1   0 0 0 1
Zone      Numerics
(bits)
```

The number 1 in ASCII looks like this:

```
0 1 0 1   0 0 0 1
Zone      Numerics
(bits)
```

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

```
110    =  Plus
                    EBCDIC
1101   =  Minus
```

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

```
1010   =  Plus
                    ASCII
1011   =  Minus
```

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

```
          ┌────┬────┬────┬────┐
Packed    │0001│0000│0111│1100│
   │      └────┴────┴────┴────┘
   │
   │
   │      ┌────┬────┬────┬────┬────────┐
Unpacked  │1111│0001│1111│0000│11000111│
          └────┴────┴────┴────┴────────┘
            ↑         ↑
       Zones inserted if PSW bit
       12 is 0 (EBCDIC Mode)
```

The remainder of the packed fields used by the decimal feature are the same.

For instance, a +107 would look like this:

```
If PSW      D    D    D    S
bit 12    ┌────┬────┬────┬────┐
          │0001│0000│0111│1010│  EBCDIC
is 0      └────┴────┴────┴────┘

if PSW
bit 12    ┌────┬────┬────┬────┐
is 1      │0001│0000│0111│1010│  ACCII
          └────┴────┴────┴────┘
```

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

```
          ┌────┬────┬────┬────┐
Packed    │0001│0000│0111│1010│
  │       └────┴────┴────┴────┘
  │
  │
  │       ┌────┬────┬────┬────┬────┬────┐
Unpacked  │0101│0001│0101│0000│1010│0111│
          └────┴────┴────┴────┴────┴────┘
                ↑         ↑
                │         │
                │         │
           Zones│inserted if PSW
             12 bit is 1 (ASCII)
```

WAIT STATE BIT:

```
     12   15
    ┌───────┐
    │A M W P│   PSW
    └───────┘
         ↑
         │
         │
         └───── Wait Bit
```

The wait bit contains a one, instructions are no longer fetched and executed. Instead the System/360 will wait until an interrupt occurs and changes the PSW. Of course, the "new" PSW would contain a zero in bit position 14.

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

PROBLEM STATE BIT:

As we said earlier, the System/360 may be executing either the supervisor program or the problem program. Accordingly, the System/360 is said to be in either the supervisor state or the problem state.

All instructions may 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 given by the supervisor program.

Bit 15 of the PSW is called the problem state bit. It is used to indicate the state of the instruction associated with that PSW.

```
     12   15
    ┌───────┐
    │A M W P│
    └───────┘
            ↑ 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 we are using, bit 15 identifies the state of the System/360.

Knowing the program state allows the System/360 to be sure that those instructions reserved for the supervisor state are executed only when in the supervisor state. If the problem program attempts to execute an instruction reserved for the supervisor state, a program interruption occurs.

We would expect that bit 15 of any of the old PSW's in main storage would contain ones.

We would expect that bit 15 of the five new PSW's in main storage would contain zeros.

This is because the old PSW indicates where we left the problem program while the new PSW indicates where we are entering the supervisor program.

PRIVILEGED INSTRUCTIONS

- Privileged instructions are those which can be executed only in the Supervisor State (bit 15 of PSW is 0).
- Trying to execute a privileged instruction in the Problem state (bit 15 set to 1) will cause a privileged operation program check interruption.

What instructions are considered privileged? We do not intend at this time to list all privileged instructions. However, you should be aware of the considerations which determine which instructions are privileged.

First of all, it would be expected that the supervisor program should be able to change any part of the PSW that should be changed by the problem program. Let's take a look at the fields of the PSW:

Bits	Field	Changed By
0-7	System Mask	An instruction called Set System Mask
8-11	(We'll examine this field later)	
12-15	(AMWP)	
16-31	Interruption Code	An Interrupt
32-33	Instruction Length	An Interrupt
34-35	Condition Code	Many Instructions
36-39	Program Mask	An Instruction Called Set Program Mask
40-63	Instruction Address	Execution of Program

From the above we can see that some of the PSW fields can be changed by a special instruction. The 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 interrupt. The other is by way of the instruction Load PSW. It would not be desirable to allow the problem programmer to use the Load PSW instruction since this instruction changes all parts of the PSW. You would not want the problem program to have that much control over the machine. The supervisor program only should retain this control. As a result, the Load PSW is a privileged instruction. It can only be used by the supervisor program (indicated by bit 15 of the PSW). The supervisor programmer could use the Load PSW to change any part of the PSW. It would also use this instruction to return to the problem program after an interrupt has been serviced.

```
┌──────────┐              ┌───────┐
│Supervisor│              │Problem│
│          │  Load PSW    │       │
│          ├──────────────>│       │
│Program   │              │Program│
└──────────┘              └───────┘
```

The problem program would enter the supervisor program by way of an interrupt. This interrupt would normally be result of the instruction Supervisor Call.

```
┌──────────┐                    ┌───────┐
│Supervisor│                    │Problem│
│          │  Supervisor Call   │       │
│          │<───────────────────┤       │
│          │     Interrupt      │       │
│          │                    │       │
│Program   │                    │Program│
└──────────┘                    └───────┘
```

Notice that a branch instruction is not used in either example above. This is because a branch instruction does not change the problem state bit (bit 15) in the PSW.

The supervisor program can change the state of the machine anytime it wants to by use of the Load PSW instruction. 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 interrupts) has a zero in bit 15.

Besides the Load PSW instruction, there are two other instructions which can change the PSW. They are Set System Mask and Set Program Mask. The Set Program Mask is not a privileged instruction. As such, 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. This means that the condition code is also changed.

- The Set System Mask instruction is used by the supervisor program to change the PSW system mask field.

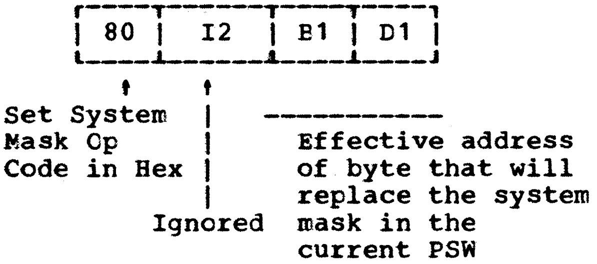
- Set System Mask is a privileged instruction.

- Set System Mask is of the SI format. The I/O field is ignored.

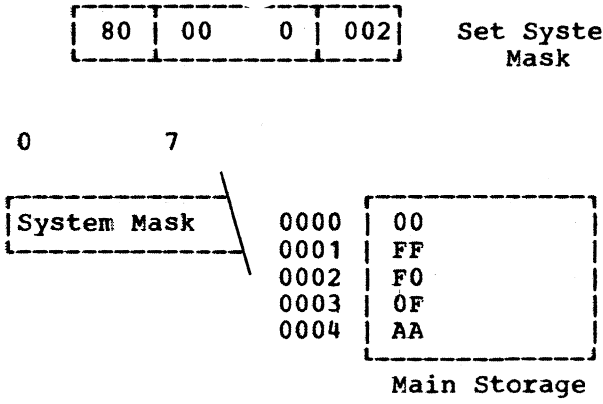
The Set System Mask instruction is a privileged instruction. This is because the system mask affects I/O interrupts. The System/360 is designed to have the supervisor handle all I/O operations. For this reason, the Set System Mask instruction and the four I/O instructions are privileged operations. The Set System Mask instruction is of the SI format.

Op Code	I2	B1	D1

This Set System Mask instruction is similar to Load PSW instruction in that the I/O field is ignored.


80
I2
B1
D1
Set System Mask Op Code in Hex
Ignored
Effective address of byte that will replace the system mask in the current PSW

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

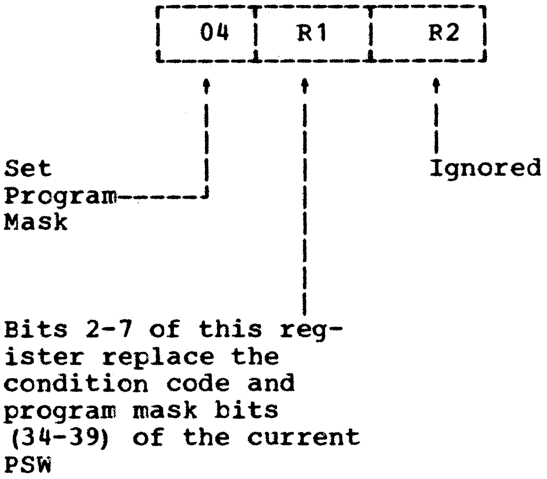


SET PROGRAM MASK INSTRUCTION

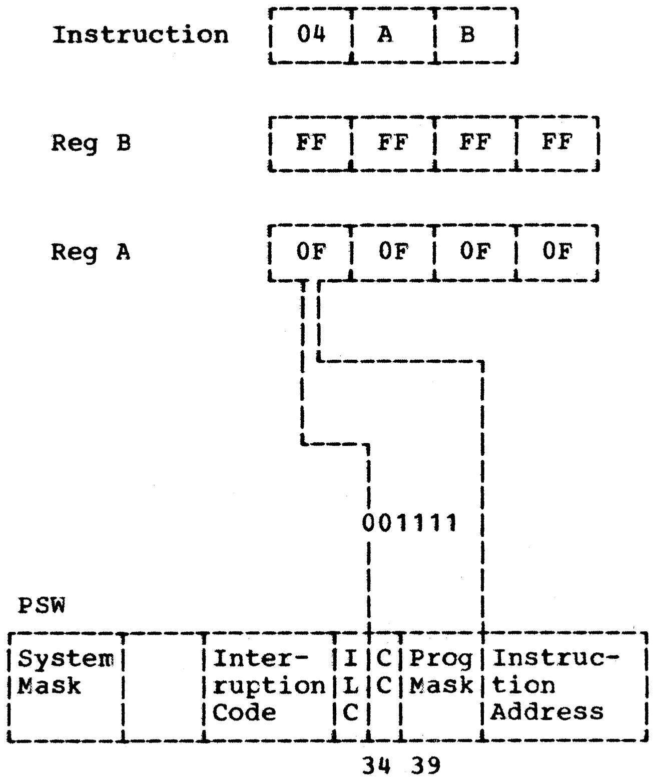
- The Set Program Mask instruction is used to change the setting of the condition code and program mask in the current PSW.

- Set Program Mask is of the RR format. The R2 field is ignored.

The Set Program Mask instruction is of the RR format.


04
R1
R2
Set Program Mask
Ignored
Bits 2-7 of this register replace the condition code and program mask bits (34-39) of the current PSW

Example:



As can be seen in the above example, reg B was ignored. Bits 2-7 (001111) of reg A were placed in positions 34-39 of the PSW. This action replaced the condition code and program mask. With a program mask of all ones, any fixed point and decimal overflows would be treated as errors and a program interrupt 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 would be as shown. Bits 32-33 are the instruction length code.

```
                   ┌────┬────┬────┐
In   ---Instruction│ 04 │ B  │ A  │
Hex                └────┴────┴────┘

                   ┌──────────────────┐
     Reg A         │ F0   F0   F0   F0│
                   └──────────────────┘

                   ┌──────────────────┐
     Reg B         │ 0F   0F   0F   0F│
                   └──────────────────┘

Bits 32-39 of      ┌────────────────┐
PSW before         │ 0 1 0 1 0 1 0 1│
                   └────────────────┘

                    32            39
Bits 32-39 of      ┌────────────────┐
PSW after          │ 0 1 0 1 0 1 0 1│
                   └────────────────┘
```

Remember, the program mask is used to determine which program checks can cause interrupts. For example, with a program mask of all zeros, a fixed point or decimal overflow will not be treated as a programming error and no program interrupt will occur. Instead an overflow will set the condition code to 11 (3). This is normal regardless of the program mask. But now no interrupt occurs and the problem programmer can use the branch on condition instruction to test for overflow.

REVIEW QUESTIONS ON SYSTEM/360 AND INTERRUPTS

1. List the five classes of interrupts.

   a. ______________________

   b. ______________________

   c. ______________________

   d. ______________________

   e. ______________________

2. Define:

   a. Current PSW ______________________
   ____
   ____

   b. Old PSW ______________________
   ____
   ____

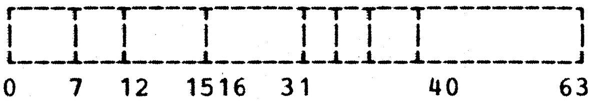
   c. New PSW ______________________
   ____
   ____

3. The area of main storage reserved for old PSW's is from 0024 to__________.

4. The area of main storage reserved for new PSW's is from _____ to 0127.

5. The area of main storage reserved for machine check log out's starts at __________.

6. Label the fields of the PSW.



7. Which interrupt cannot be masked?

8. To prevent an interrupt, a mask bit must be (0,1).

9. How can a System/360 be taken out of the wait state?

10. What can switch the system from a problem state to a supervisor state?

11. What is placed in the interruption code on an I/O interrupt?

12. What is placed in the interruption code on a supervisor call interrupt?

13. What is placed in the instruction length code when an RX type instruction was the last instruction executed prior to an I/O interrupt?

14. In the problem state _____ instructions cannot be used or a ________ will occur.

15. After handling an I/C interrupt, how does the machine return to the interrupt program?

16. Which of the following may not be given by a problem program?

   a. Set System Mask

   b. Set Program Mask

   c. Load PSW

   d. Supervisor Call

   e. Any I/C Instruction

17. The Set System instruction causes the system mask to be replaced by__________.

18. The Set Program Mask instruction replaces the __________ and _____ with__________.

19. The Load PSW instruction replaces: (Choose the most correct answer.)

   a. The current PSW with an old PSW

   b. The current PSW with the contents of a general register

   c. A new PSW with a doubleword from main storage

   d. The current PSW with a doubleword from main storage.

STORAGE PROTECTION

- There is a 4 bit storage key associated with each main storage block of 2048 bytes.
- There is a protection key 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 one is not zero, a protection exception will occur, causing a program interrupt. A storage modification cycle will not be taken.

The problem programmer may not be able to change the current PSW easily because of the concept of privileged instructions. However, what is to prevent the problem programmer from modifying the New PSW's which are in main storage? After all, any information in main storage can be treated as data and modified. The five New PSW's in storage locations 0088-0127 are no different in this respect. In fact, we would want the supervisor program to be able to modify this area of storage. However, we would not want the problem program to be able to modify this same area. It is undesirable to have any part of the supervisor program changeable by the problem program. What is needed here is some means by which the supervisor program can change any area of main storage while the problem program can only change its own assigned area. The System/360 has available a tamper-proof storage protection feature. It is optional on Models 30, 40 and is standard equipment on Models 50-70.

STCRAGE KEYS

To implement the storage protection feature, each main storage block of 2048 bytes has a key associated with it. This key is four bits long and may contain any number from 0 to 15. These numbers are referred to as Storage Keys. They need not be assigned in any order. Any of the possible 16 keys can be used regardless of storage size. For instance, the 85K storage unit below has had a key assigned for each block of 2048 bytes of 8K main storage.

```
┌───────────┐
│           ├────┐
│6144 - 8191│  13│
├───────────┼────┘  │
│           │       │
│           ├────┐  │
│4096 - 6143│   5│ ←
├───────────┼────┘  │
│           │       │ Storage Keys
│           ├────┐  │
│2048 - 4095│  11│ ←
├───────────┼────┘  │
│           │       │
│           ├────┐  │
│   0 - 2047│   0│ ←
└───────────┴────┘  │
```

A 16K main storage unit would need 8 storage keys. The hardware necessary for the storage keys is part of the Storage Protection feature.

PROTECTION KEY

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

```
┌──────┬─┬──┬────────┬───┬────┬──────────┐
│System│ │AM│Inter-  │I C│Prog│Instruc-  │
│Mask  │ │WP│ruption │L C│Mask│tion      │
│      │ │  │Code    │C  │    │Address   │
└──────┴─┴──┴────────┴───┴────┴──────────┘
        │ │
        │ │
 ┌──────┘ └─────┐
 │              │
 │              │
 ├──────────────┤
 │  Protection  │
 │     Key      │
 └──────────────┘
```

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 main storage. The fetching of instruction is not an example of 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 that block of main storage.

2. A protection exception will result in a program interrupt if the two keys are not identical. If one of the keys contains a zero, the keys are said to match and the protection exception 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 interrupt occurs.

If the key in the PSW is zero and the storage key is six, a program interrupt will not occur. As long as either key is zero, the storage modification cycle is allowed. Remember now, the storage protection feature only applies to storage modify type cycles.

PROTECTION EXCEPTION

Whenever a program interrupt occurs, the interruption code placed in the old PSW indicates the reason for the interrupt. When storage protection is violated, a protection exception is indicated in the interruption code of the old PSW (Figure 1-30).

Assuming the PSW has a protection key of six, the 2K blocks of main storage labeled A, C and D can be successfully modified.

```
┌──────────┬─┐
│A         │6│←
├──────────┼─┤
│B         │5│←
├──────────┼─┤   Storage Keys
│C         │0│←
├──────────┼─┤
│D         │6│←
└──────────┴─┘
```

Blocks A and D have a storage key of six to match the key in the PSW. Block C has a key of zero which means it can be modified by any program.

Likewise if the PSW protection key is zero, 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 the PSW used by a supervisor program.

SET STORAGE KEY INSTRUCTION

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

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 changeable only by the Load PSW instruction or by an interrupt. However, the storage keys for each block of 2048 bytes can be changed by an instruction known as Set Storage Key. 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 8 executions of the Set Storage Key instruction. The set storage key instruction is of the RR format.

```
┌─────────┬────┬────┐
│ Op Code │ R1 │ R2 │
└─────────┴────┴────┘
```

The desired storage key (0-15) is in bits 24-27 of the general register specified by the R1 field. The remainder of the register is ignored.

Given register 4 (as shown below in "hex"), the storage key of the 2048 byte block will be set to 5.

```
                  24              31
               ──┬───┬───┬───┬───┬─────
Register 3       │ 0 │ 1 │ 0 │ 0 │
                 │   │   │   │   │
               ──┴───┴───┴───┴───┴─────
```

The question now arises: "Which 2048 byte block will have its storage key set?" This is determined by the address in the general register specified by the R2 field.

```
             ┌────┬───┬───┐
             │ 08 │ 3 │ 5 │
             └────┴───┴───┘
                ↑    ↑   ↑
                │    │   │
                │    │   │
Set             │    │   │    This register
Storage─────────┘    │   └────has the address
Key                  │        of the 2k block
                     │
                     │
           Key is in this
                register
```

Storage addresses in the System/360 are 24 bits long. General register capacity is 32 bits. As we discussed previously, storage addresses are placed in the low-order 24 bit-positions in a general register (but 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. Bits 8-20 of 8K of storage addresses would appear as follows:

```
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 6143-8191
```

Notice that we didn't have to look at the register positions 0-7 or 21-31. Bits 8-20 are sufficient to determine which block of 2048 addresses is to be used.

A specification is given to the programmer that requires the four low-order 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:

```
 0      7 8             20 21    27 28 31
┌────────┬───────────────┬────────┬────┐
│Ignored │Which 2k block │Ignored │0000│
└────────┴───────────────┴────────┴────┘
```

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 will be set to 1.

```
Instruction    |0|8|3|5| (hex)

Register 3     |0|0|0|0|0|4|1|0|    (hex)

Register 5     |0|0|0|0|0|1|4|0|    (hex)
```

	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.

USING STORAGE PROTECTION

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 keys 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 1-31).

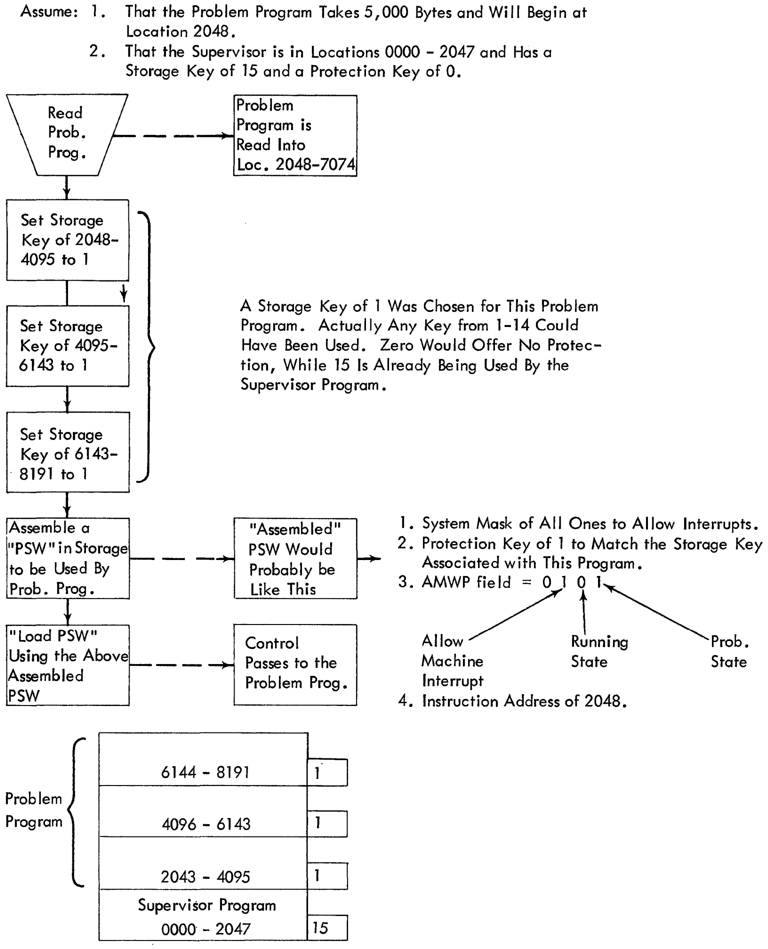

Assume: 1. That the Problem Program Takes 5,000 Bytes and Will Begin at Location 2048.
2. That the Supervisor is in Locations 0000 - 2047 and Has a Storage Key of 15 and a Protection Key of 0.
Read Prob. Prog.
Problem Program is Read Into Loc. 2048-7074
Set Storage Key of 2048-4095 to 1
Set Storage Key of 4095-6143 to 1
Set Storage Key of 6143-8191 to 1
A Storage Key of 1 Was Chosen for This Problem Program. Actually Any Key from 1-14 Could Have Been Used. Zero Would Offer No Protection, While 15 Is Already Being Used By the Supervisor Program.
Assemble a "PSW" in Storage to be Used By Prob. Prog.
"Assembled" PSW Would Probably be Like This
1. System Mask of All Ones to Allow Interrupts.
2. Protection Key of 1 to Match the Storage Key Associated with This Program.
3. AMWP field = 0 1 0 1
Allow Machine Interrupt
Running State
Prob. State
4. Instruction Address of 2048.
"Load PSW" Using the Above Assembled PSW
Control Passes to the Problem Prog.
Problem Program
6144 - 8191
1
4096 - 6143
1
2043 - 4095
1
Supervisor Program 0000 - 2047
15

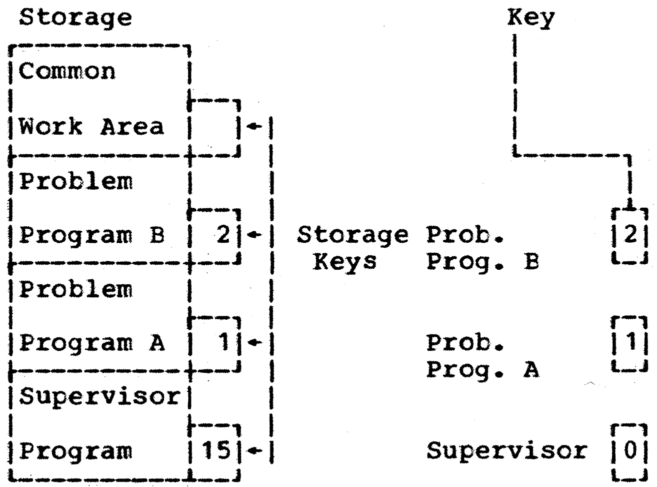
Figure 1-31. Using Storage Protection

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 15. This means that unless a problem program has a key in its PSW of 0 or 15, it will not be able to modify or change information in the area being used by the supervisor program. This is unlikely because it is the supervisor program that assigns storage and protection keys to the problem 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 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 15. 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 (as a result of a section of the supervisor program). This program takes up 3 blocks of 2048 bytes. Each of these 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 keys. This allows the problem program to alter itself if necessary, but prevents it from altering another program.

MULTIPROGRAMMING

So far, we have only discussed the concept of two programs in the computer: a supervisor program and problem program. There are two or more problem programs in the machine. Of course, just as in the supervisor-controlled concept, only one program is being executed at any one time.

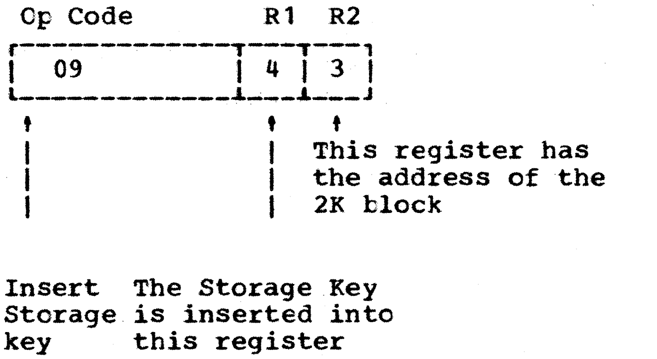


In the above example, each problem program has a different storage key. The protection keys used by each program are also different. Each matches the respective storage keys. Notice that the supervisor's protection key does not match its storage key. Because the supervisor's protection key (in its PSW) is zero, it does not have to match. It can unlock any area of main storage and alter its contents if necessary. Also note the common work area with a storage key of zero. This area is not protected and can be used by any of the programs.

INSERT STORAGE KEY INSTRUCTION

- This instruction is used to find the current value of a storage key.
- The instruction is of the RR format.
- This is a privileged instruction.

Besides the Set Storage Key instruction, there is another instruction to help a supervisor program assign storage keys. It is called Insert Storage Key. This instruction does not change any storage keys. Its purpose is to inspect or examine a storage key. The Insert Storage Key instruction is also of the RR format.


Op Code
R1
R2
09
4
3
This register has the address of the 2K block
Insert Storage key
The Storage Key is inserted into this register

Notice that this instruction works just opposite to the "Set Storage Key" instruction. 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 is 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 the storage key (1) of block 2048-4095 was inserted into bits 24-27 of register 4 while bits 28-31 were made zero. The remainder of the register was unchanged. The storage keys themselves were unchanged.

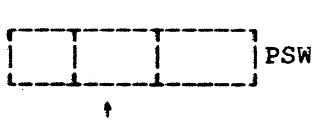
REVIEW QUESTIONS ON STORAGE PROTECTION

1.

Storage Block	Key
6144 - 8191	2
4096 - 6143	1
2048 - 4095	1
0000 - 2047	A

These are referred to as ____ keys.

2.



This field is referred to as the ------ ---------.

3. The keys in question 1 can be changed only by __________.

4. Bits 8-11 of the PSW can be changed only by (1) __________. or by (2) __________.

5. Storage protection applies to:

a. Modify-type main storage cycles

b. Fetch-type main storage cycles

c. All main storage cycles

6.

Storage Block	Key
A	0
B	1
C	2
D	3

Assuming a PSW key of 2:

a. From which blocks may information be fetched?__________

b. In which blocks may information be altered?__________

7.

Storage Block	Key
A	0
B	1
C	2
D	3

Assuming a PSW key of 0:

a. From which blocks may information be read out?

____
____

b. In which blocks may information be altered?

____
____

8.

Storage Block	Key
A	0
B	1
C	2
D	3

If an instruction with a PSW key of 3 attempts to alter data in block C:

a. The data will be stored and the program will continue.

b. The data will be stored and the program will be interrupted.

c. The data will not be stored and the program will continue.

d. The data will not be stored and the program will be interrupted.

9. In the following example, the (storage/protection) key of block (A,B,C,D) will be set to __________.

Storage Block	Key
A 6144-8191	0
B 4096-6143	1
C 2048-4095	2
D 0000-2047	3

Instruction 0 8 4 6

Register 4 0 0 0 0 2 1 4 0

Register 6 0 0 0 0 0 E F 2

10. Show the register contents after executing the insert storage key instruction.

Storage Blank	Key
A 6144-8191	9
B 4096-6143	8
C 2048-4095	7
D 0000-2047	A

Instruction 0 9 2 3

Register 2 before 0 0 0 0 0 F 0 0

Register 2 after _______ _______

Register 3 before 0 0 0 0 0 F 0 0

Register 3 after _______ _______

11. A violation of storage protection results in a:

a. Machine interrupt with a protection violation indicated in the interruption code of the new PSW.

b. Program interrupt with a protection violation indicated in the interruption code of the new PSW.

c. Program interrupt with a protection violation indicated in the interruption code of the old PSW.

d. Machine interrupt with a protection violation indicated in the interruption code of the old PSW.

A brief review of the material contained in pre-school manuals will be presented here as they pertain to the IBM 2030 Processing Unit. The intention is not to delve deeply into transistor theory, but to present a CE's view of logic blocks and circuitry as they appear in this manual and on the ALD's.

SLT CIRCUITRY

- Basic SLT circuitry contains AND, OR and inverter functions.
- A straight line into or out of a logic block indicates the more positive level. A wedge on the line indicates the least positive level associated with that circuitry.
- +L (+3 volts) and -L (0 volts) levels are used throughout the IBM 2030 Processing Unit.
- Voltage levels are 0v, 9v, 3v, and 12v.
- Different circuit speeds are 700 nanoseconds, and 5-10 nanoseconds.

+AND CIRCUIT

A basic +AND circuit consists of input diodes in parallel, tied through a resistor to a positive voltage source. The anodes of the diodes face in the direction of the positive source (Figure 2-1). While any of the input lines are negative, current flows through the resistor, keeping the output at a minus level. When all inputs are positive, current is cut off and the positive voltage level is felt at the output.

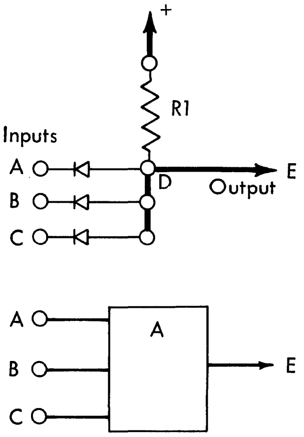


Figure 2-1. +AND Circuit

This same basic circuit is sometimes labeled a -OR circuit when the logic level needed is minus (Figure 2-2). One or more inputs must be negative to satisfy this logic function.

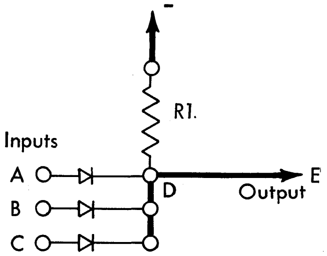


Figure 2-2. -OR Circuit

+OR CIRCUIT

A basic +OR circuit consists of input diodes in parallel, tied through a resistor to a negative voltage source. The anodes of the diodes face the input lines (Figure 2-3). When any one or all of the inputs is plus current will flow in the circuit and the output will be positive. All inputs being negative causes the circuit to be cut off and the output will be negative.

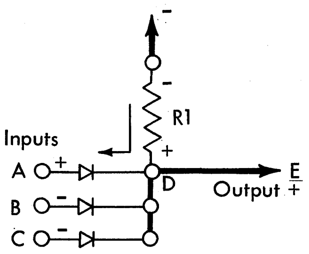


Figure 2-3. + OR Circuit

This circuit is sometimes labeled -AND when the logic output needed is a negative level (Figure 2-4). All inputs must be minus to satisfy this logic function.

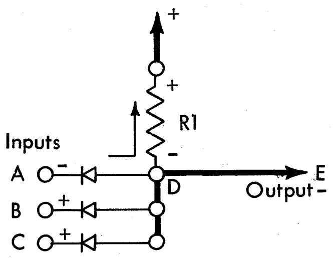


Figure 2-4. -AND Circuit

THE INVERTER

An inverter is labeled "N" and is an NPN transistor with the input line tied to the base and the output line tied to the collector and a voltage source. (This is only one type of inverter. Input lines are connected differently on other circuits but the principle of operation is the same). The inverter has only one input, and the output is always of opposite polarity (Figure 2-5).

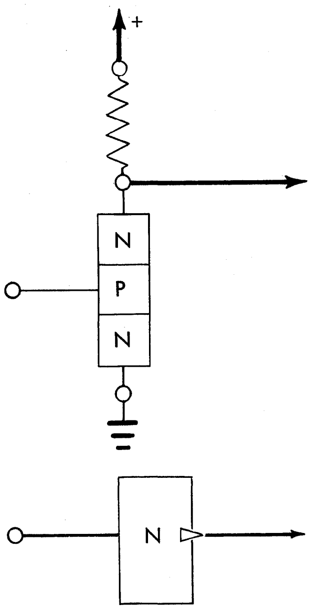


Figure 2-5. Inverter

THE EXCLUSIVE OR

The exclusive OR is labeled "OE" and has its inputs and outputs as noted on the block. The output of the block will be at its indicated polarity when one <u>and only one</u> input is at its indicated polarity (Figure 2-6).

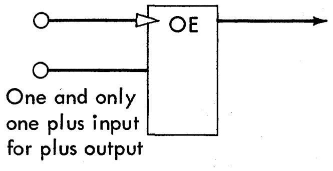


Figure 2-6. Exclusive OR

THE SINGLE SHOT

The single-shot block is labeled "SS". When an input polarity, as indicated on the block, is applied, the output will change temporarily to the polarity indicated. Each SS has its time indicated on the top of the block. The output will remain at this level for the time specified. Figure 2-7 shows positive input and output lines. Depending on the type of SS, these polarities can be negative. When these polarities are negative, they are indicated as such.

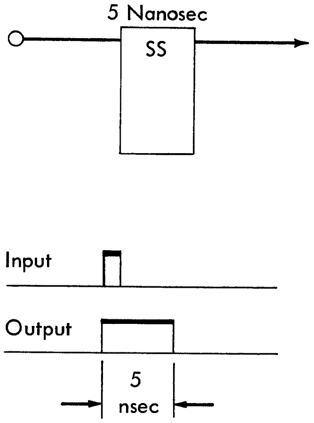


Figure 2-7. Single Shot

TIME DELAY.

Another block with an indicated time is the time-delay block which is used to delay a signal. This block is shown in Figure 2-8, and can again have different input and output polarities depending on the type of TD.

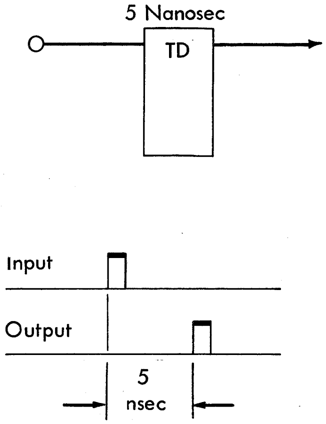


Figure 2-8. Time Delay

LINE SENSE AMPLIFIER

The line-sense-amplifier is used to tap a transmission line at two or more locations. It is an emitter follower, and therefore presents a high impedance to the line which prevents signal reflections and noise. The logic block is labeled "DLR" and is of the 5-10 nanosecond family of circuits.

REMEMBERING DEVICES

Remembering devices can be shown as single-unit logic blocks or as multiple blocks arranged to form one circuit. In the IBM 2030 a polarity-hold (PH) block and a trigger (FF) block are generally shown as single blocks. A latch (FL) is generally shown as a combination of AND and OR blocks.

THE FLIP FLOP

The flip-flop, sometimes called a trigger, has three inputs and two outputs, as shown in Figure 2-9.

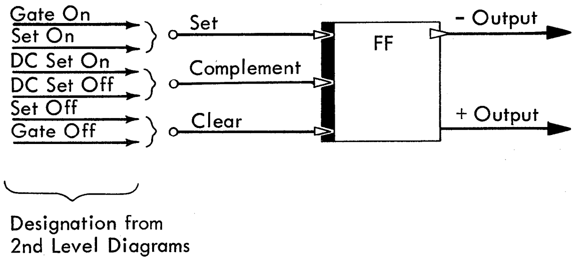


Figure 2-9. Flip Flop

1. The two inputs, set and clear, are always opposite in polarity.

2. A pulse of the indicated polarity on the set line cause the flip-flop to assume the indicated output polarities.

3. These outputs remain at their indicated polarities until the flip-flop is reset.

4. When the clear line is activated,

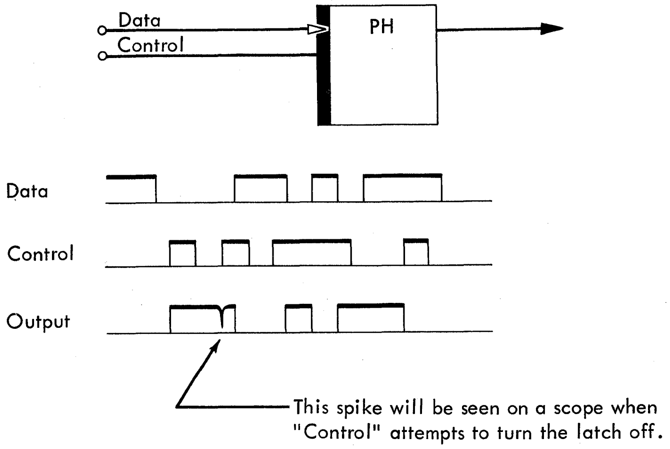
the outputs change to the opposite polarity of that indicated.

5. The third input, complement reverses the status of the flip-flop whenever complement is activated.

6. The input and output lines are shown as indicated in Figure 2-9.

THE POLARITY - HOLD LATCH

The polarity-hold latch is used to retain information sampled from a data bus. Figure 2-10 illustrates a polarity-hold latch with a negative data bus in. Whenever this data bus is negative, and the control line is active (plus) the latch is turned on (plus output). The latch remains on until the data line and the control line are positive at the same time.



This spike will be seen on a scope when "Control" attempts to turn the latch off.

Figure 2-10. Polarity Hold Latch

The polarity-hold latch may be the type that requires a positive data bus in. When the data bus and the control line are positive the output will be positive. If the data line drops while the control line is active, the output will drop. The output line will always follow the data line when the control line is active.

THE FLIP LATCH

Combinations of AND and OR logic blocks are used to allow several set lines and one or more reset lines to "flip" the latch. The flip latch will stay on or off depending on the active lines applied. Generally the latch back is combined with "not" reset lines to form the normal latch back which keeps the latch in its turned on state. The flip latch can have two output lines which are of opposite polarity.

COMBINED COMPONENTS

Combinations of the resistors, diodes and transistors allow for varied uses of the basic AND and OR circuits. Figures 2-11, 2-12, and 2-13 show some of these combinations. The basic AND circuit can become an AND inverter (AI) with the addition of a transistor. The AOI is a combined AND, OR and transistor circuit. These functions and others used on the IBM 2030 Processing Unit are:

1. AI - AND Inverter

2. AOI - AND OR Inverter

3. API - AND Power Inverter

4. DCI - Direct Coupled Inverter

5. HPD - High-Power Driver

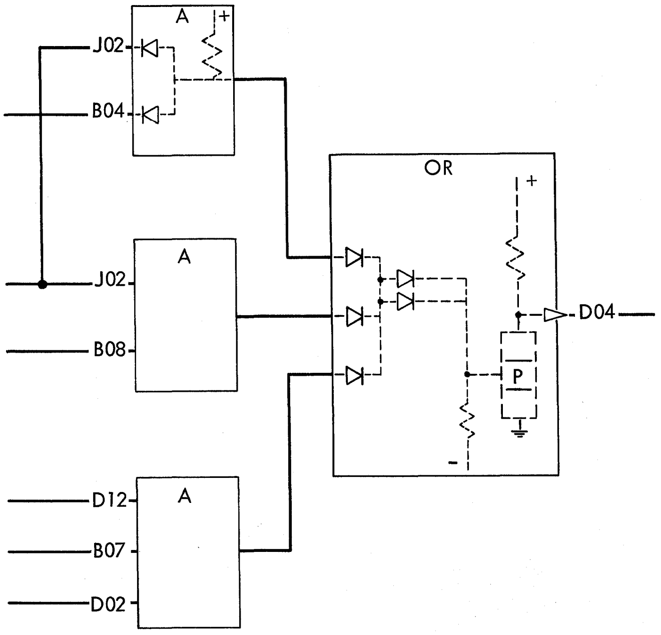


Figure 2-11. +AOI

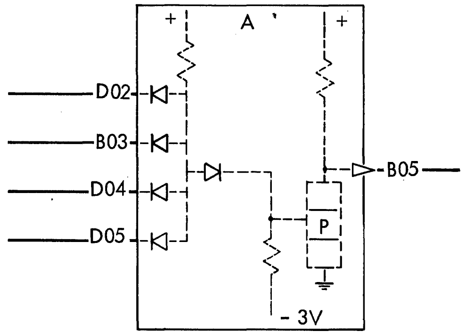


Figure 2-12. +AI

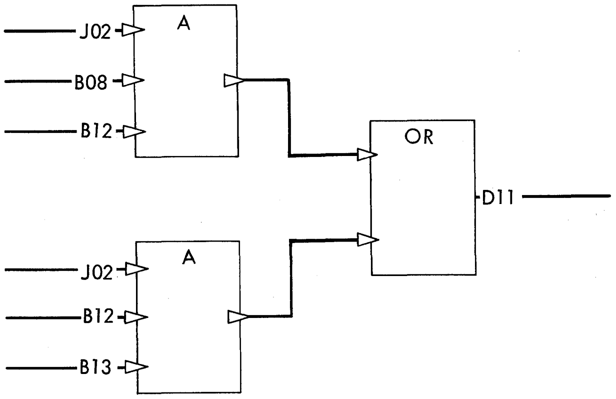


Figure 2-13. -AOI

REVIEW QUESTIONS - BASIC COMPONENTS

1. A wedge on an output line designates the active output as ________.

2. A plus OR is the same as a ________ AND.

3. Exclusive OR circuits have ________ input lines.

4. The Exclusive OR is active when the input levels are ________.

5. The time designated on an SS block is the ________ that the ________ is at its designated level.

6. The time on a TD block is the ________ difference between the ________ of the input pulse and the ________ of the designated output level.

7. Three types of remembering devices are the ________ ________ latch, the ________ latch and the ________.

8. The Control line of a Polarity Hold latch does not need to be active to deactivate the output. (True/False)

9. The output of a Polarity Hold latch will always follow the Data line. (True/False).

10. The ________ ________ requires a latch back line.

CENTRAL PROCESSING UNIT (CPU) CLOCK

- A 2-megacycle clock circuit produces clock pulses P1, P2, P3, and P4.

- The overlapping of 500 nanosecond P pulses switch to form 250 nanosecond T timing pulses of T1, T2, T3, and T4.

- P pulses are distributed to all large-boards, where the switching is done near the boards that use the particular T pulse.

The timing pulses for the IBM 2030 Processing Unit, are derived from a 4-stage overlapped latch-ring that uses five nanosecond SLT circuitry. (All other portions of the machine use medium speed 30 nanosecond circuits.) A 2-megacycle crystal oscillator provides the basic pulses for the latch ring. Looking at Figure 2-14, these oscillator pulses are fed to the clock ring by two line names: oscillator (Osc) and delayed-oscillator (Dly Osc). Each of these lines have two levels, plus and minus, which means that there are a total of four lines from the basic oscillator driving the clock.

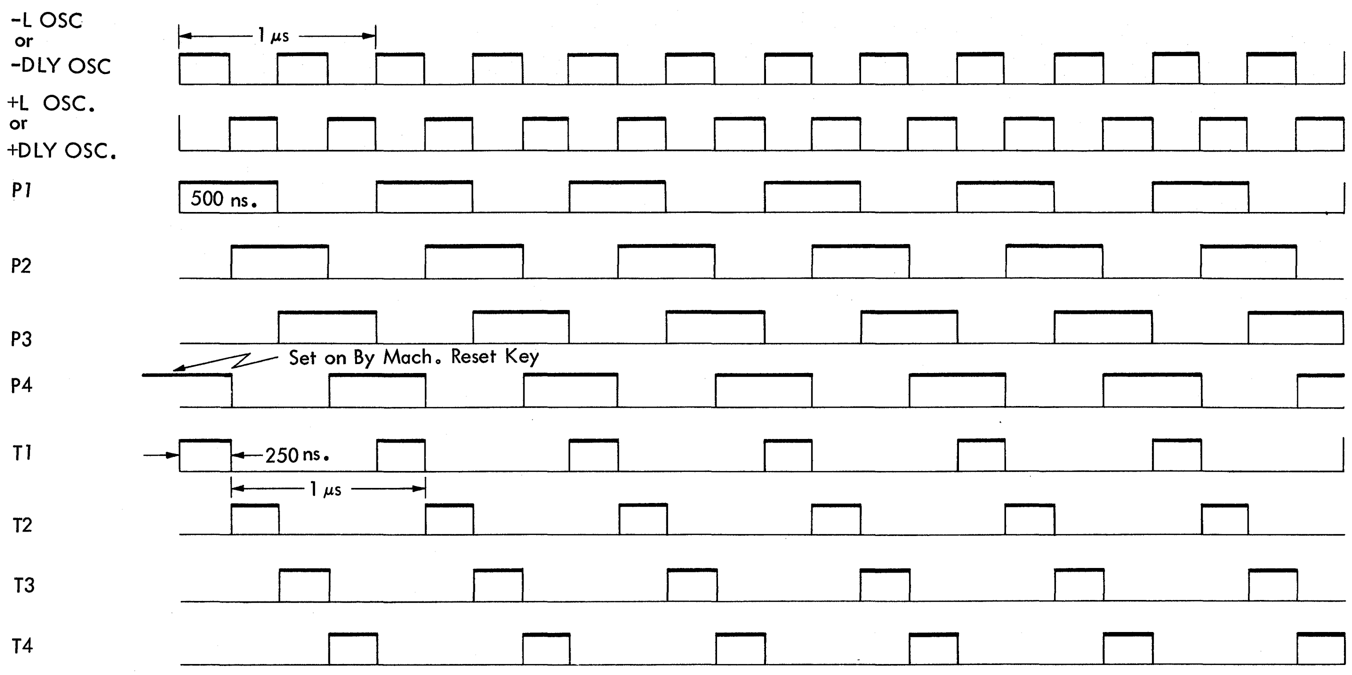


Figure 2-14. Developing Clock Pulses

As can be seen from Figure 2-14, clock latch 1 is turned on by clock-start, Dly Osc, not-clock-3, and clock-4. The initial turn-on of clock-4 is by clock-reset, which is initiated by either the machine-reset switch, or by power turn-on reset circuits. Clock latches 1, 2, and 3 turn off with the initial turn on of clock-4, so that normal progression of clock latches can be turned on. When the clock-start line is activated and the - Dly Osc becomes plus, clock-1 will turn on.

Figure 2-15 shows the relationship of the turn-on and turn-off of the clock latches. You will note that the lines + Dly Osc and - Dly Osc are not shown on the timing chart. They are essentially the same as the + Osc and - Osc lines and are used in the same manner to turn on and turn off the clock latches. This timing chart shows that all clock latches are on for 500 nanoseconds and that they successively overlap each other. Clock-1 is latched ON with the positive-going -Dly Osc pulse. On the following plus shift of the + Dly Osc line, clock-2 turns on. The second plus shift of - Dly Osc and the fact that clock-2 is on, resets the clock-1 latch. The binary-connected latches continue in this manner, and as shown in Figure 2-14 and 2-15, form separate and distinct pulses.

CLOCK DISTRIBUTION

The clock serves as a central source of four clock pulses which are put on four transmission lines. The transmission lines deliver the pulses to subdistribution points throughout the machine. At these subdistribution points, line-sense amplifiers tap off from the transmission line and feed a two-input, 30-nanosecond AOI. This brings together two clock pulses to form one timing pulse. For example, P1 and P4 are switched to develop T1. These pulses are then powered and used by the decision blocks as their basic timing pulses.

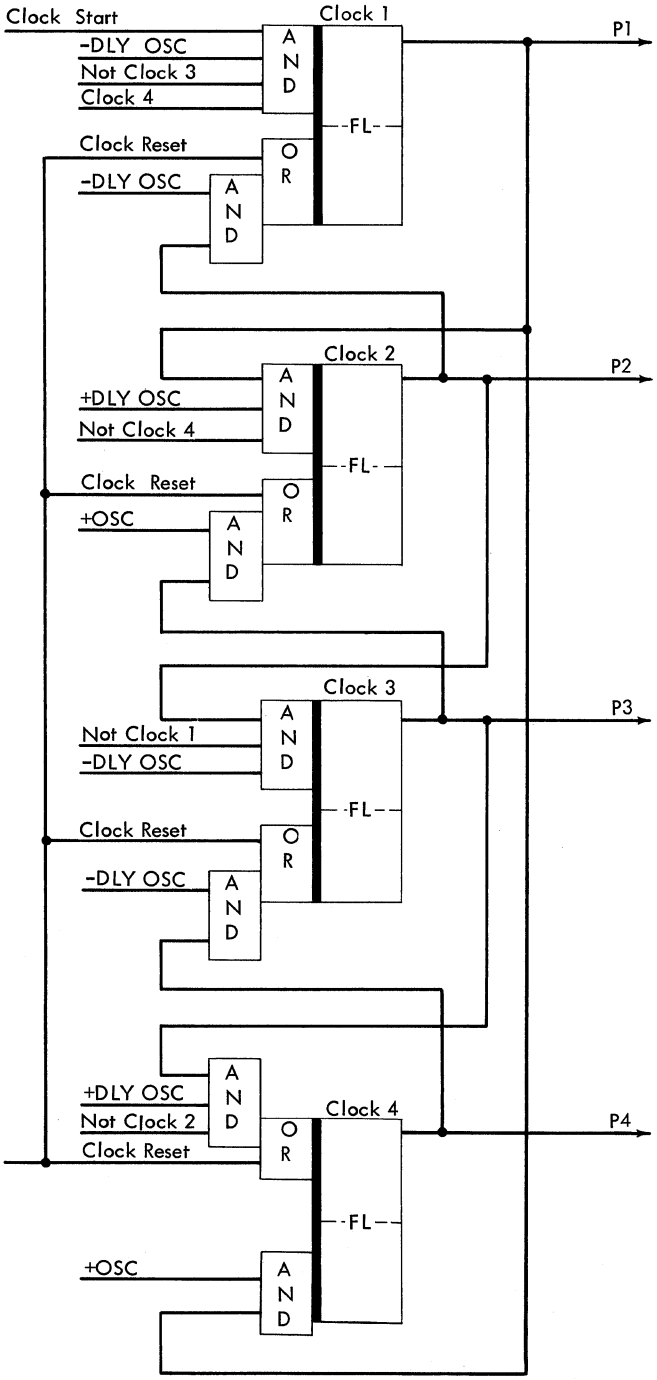


Figure 2-15. CPU Clock Timings

This distribution system sends out 500 nanosecond pulses to each large card that needs the particular pulses. The pulses are then switched to provide 250 nanosecond timing pulses. The effect of this system then allows the subdistribution centers to be close to the load centers, and ringing and noise are kept to a minimum. See Figures 2-14 and 2-16 for the development of T pulses from P pulses.

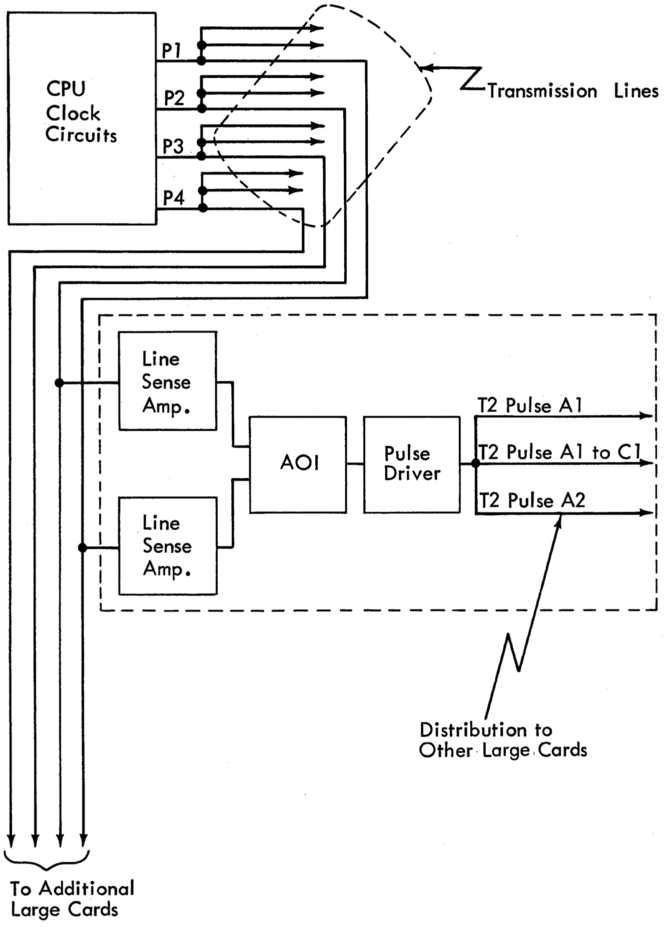


Figure 2-16. CPU Timing Pulses

CLOCK CONTROL

- The line, clock-start, must be developed and maintained in order for the clock to start and run.
- The following will de-condition Clock-Start:
  1. A hard-stop (This will also immediately reset clock latches 1, 2, and 3.)
  2. A selector-share hold
  3. A power-off condition
  4. A reset of the clock-start latch.
- The clock-start latch remains on as long as the latch-back and not-clock-stop or not-clock-reset are maintained.

The CPU clock must be capable of originating timing pulses for internal operations of gates, pulses, and reset lines as well as providing timing pulses for the multiplexor channel to which I/O equipment is attached. The clock must be told to start and stop, depending on the use and request of its generated pulses.

Initially, the clock is started by setting the clock-start latch. It is set by:

1. the start key, which turns on the clock-start-control latch and the E-cycle Stop-Sample latch,
2. the load key, which turns on the load latch, and
3. the instruction-address-load key, which sets the force-IJ request latch.

The line, clock start, is also developed by other conditions, which are ANDed with the output of the clock-start latch. These other lines are shown as "not" inputs and, in effect, prevent the restart of the clock if any one of these plus lines become minus. When the clock-start latch is reset or the clock-start line drops, the clock will stop after it reaches T4 time. The clock would run for a short period after the

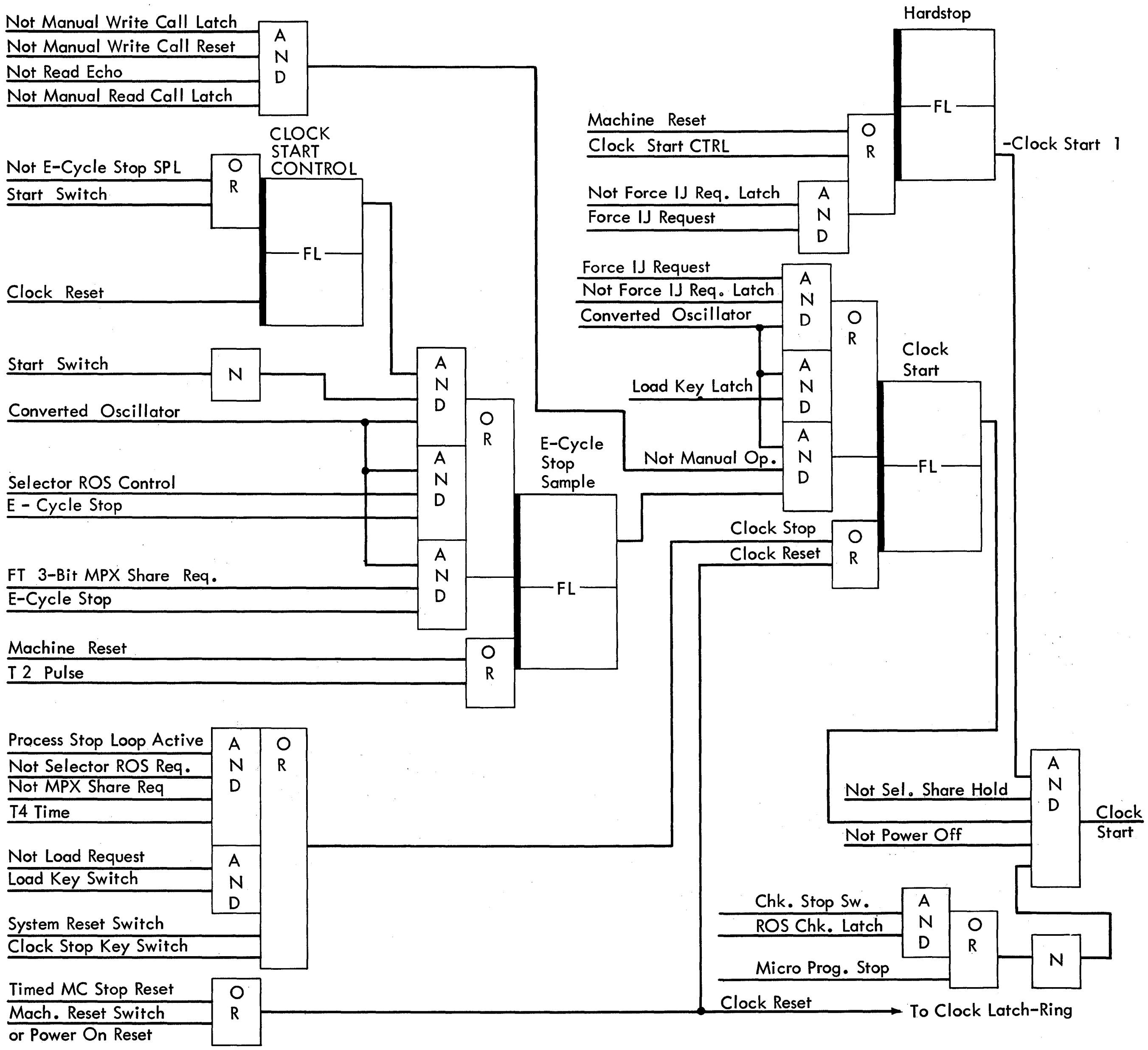


Figure 2-17. CPU Clock Control

clock-start latch is reset if it is required by some I/O device or attachment.

The clock-start latch is reset by clock-stop and clock-reset, as shown in Figure 2-17. These two lines are used to stop the clock by manual means, by programmed stops, by error conditions, or by a special CE diagnostic test. Either of these normally plus levels dropping will reset the clock-start latch.

REVIEW QUESTICNS - C.P.U. CLOCK

1. The width of a P pulse is ________ nanoseconds.

2.The circuit components of the clock are of the ________ nanosecond family.

3. The reset condition of the clock is with trigger _____ on and triggers _____, _____ and _____ off.

4. Clock trigger 4 is initially turned on by ________ ________ line.

5. T pulses are ________ nanoseconds wide.

6. Pulses on the transmission line are ________ pulses.

7. Referring to Figure 2-17, what latches are set to initially bring up "clock start" using the start key?

8. When does the clock normally stop?

9. What P pulses develop T3? Refer to Figure 2-14.

10. P pulses are distributed to every large card. (True/False)

ARITHMETIC LOGICAL UNIT (ALU)

- Binary or decimal operations as well as logical operations are performed by ALU.

- The 8-bit ALU circuitry is controlled by ROS fields.

- An excess 6 circuit adds 6 to each four bits of all decimal-add operations.

- A decimal corrector subtracts 6 from each four bits of decimal operations when necessary.

- ALU is a two-wire 8-bit parallel adder using + L and - L levels.

- ALU does not contain a parity line.

The data flow chart in Figure 2-18 shows that two buses, A-bus and B-bus, feed data into the A and B registers. The data is then gated to the ALU in different combinations. Any register can be gated individually onto the A bus and to the A register under the control of the ROS field, CA. The A-register output is further gated by the CF field to ALU. The CF field is used in microprogramming to gate the four high bits, the four low bits, or both or none to ALU. This field can also gate the eight bits straight into ALU or can interchange the four high bits with the four low bits. In Figure 2-19 you can see the various combinations of entries to ALU with their controlling ROS fields indicated.

Not all registers are gated to the B-register. Looking again at the data flow chart, Figure 2-18, only the L, D, and R registers are gated individually by the ROS field, CB. The output of the B-register is gated under control of the CG field which instructs either the high four bits, the low four bits, all bits or no bits to be sent to ALU. The ROS CV field determines whether the operation is to be true or complement binary, or true or complement decimal addition.

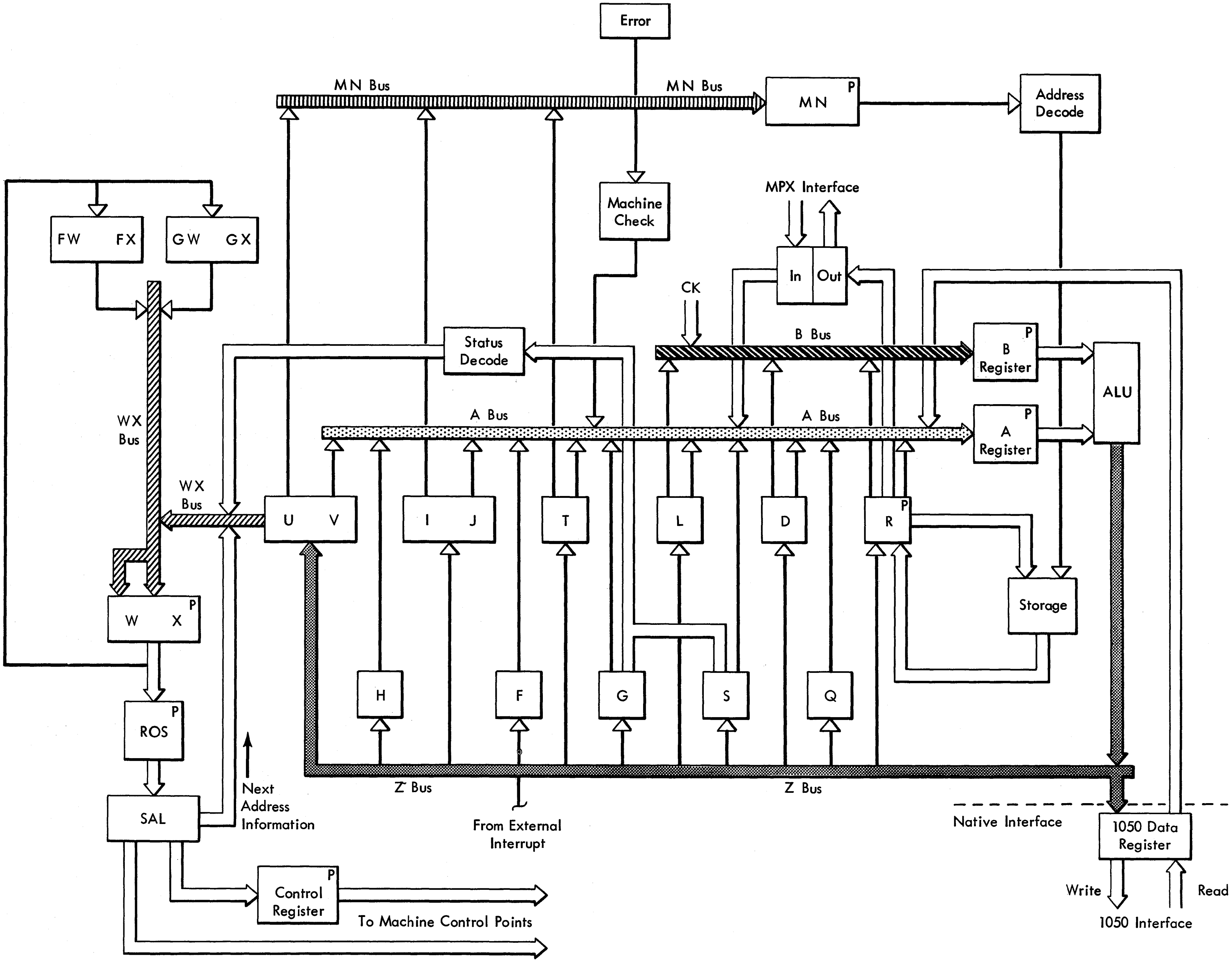


Figure 2-18. Data Flow

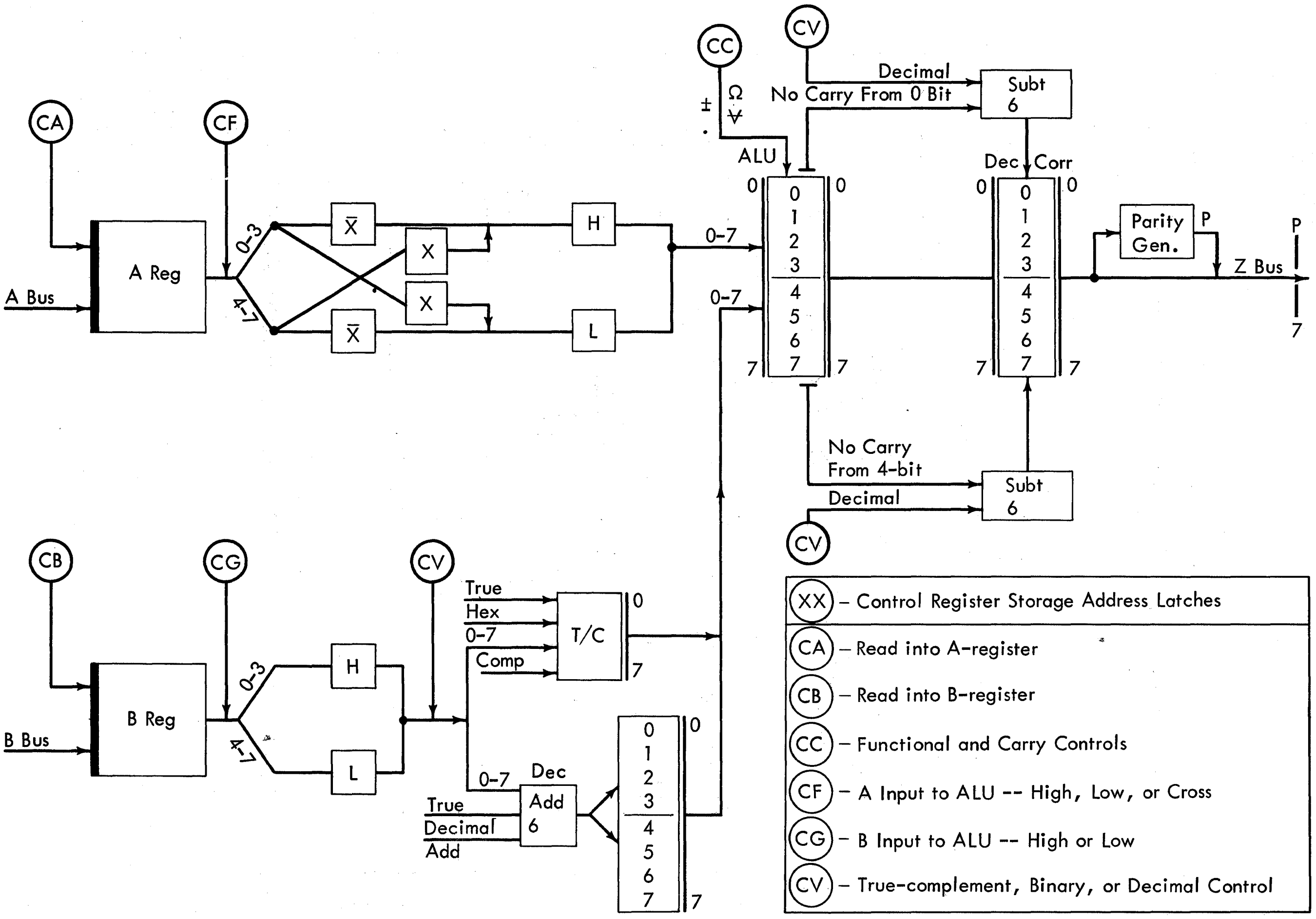
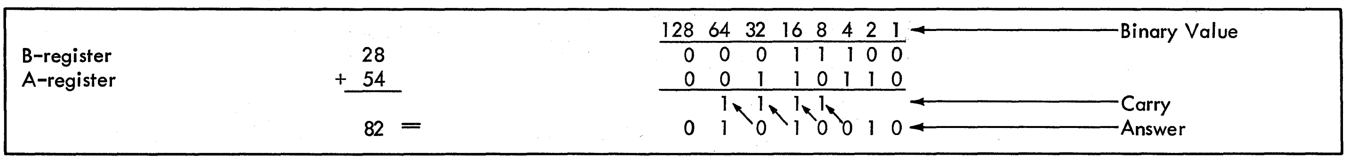

CC
CV
Decimal
No Carry From 0 Bit
Subt 6
± Ω ∀
ALU
Dec Corr
CA
CF
A Reg
A Bus
0-3
4-7
X̄
X
X
X̄
H
L
0-7
0-7
0 1 2 3 4 5 6 7
Parity Gen.
P
Z Bus
P
7
No Carry From 4-bit
Decimal
Subt 6
CV
CB
CG
CV
B Reg
B Bus
H
L
True
Hex
0-7
Comp
T/C
0
7
0-7
Dec Add 6
True
Decimal
Add
XX – Control Register Storage Address Latches
CA – Read into A-register
CB – Read into B-register
CC – Functional and Carry Controls
CF – A Input to ALU -- High, Low, or Cross
CG – B Input to ALU -- High or Low
CV – True-complement, Binary, or Decimal Control

Figure 2-19. Arithmetic Operation



```
                                128 64 32 16 8 4 2 1 ◄──────────Binary Value
B-register            28         0  0  0  1 1 1 0 0
A-register          + 54         0  0  1  1 0 1 1 0
                                    1  1   1 1       ◄──────────Carry
                      82 =       0  1  0  1 0 0 1 0  ◄──────────Answer
```

Figure 2-20. Binary Add

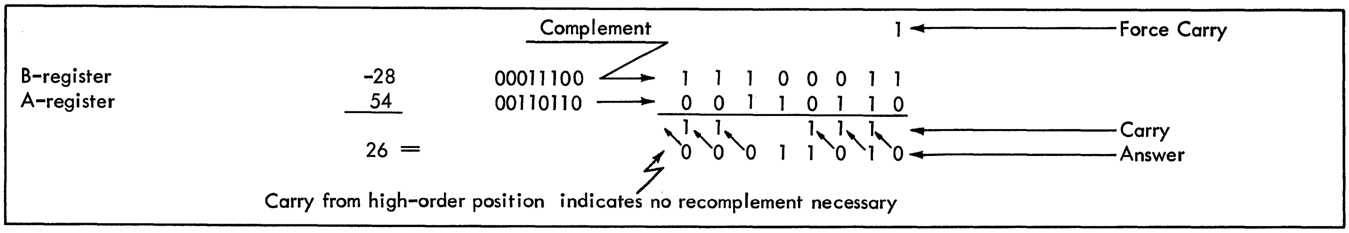


Figure 2-21. Binary Complement Add

Looking again at Figure 2-19, after gating high, low or both from the B-register, the eight data lines, 0 through 7, go to either a true-complement binary circuit or to an excess six circuit. If the CV field calls for a True Binary Add or a Complement Binary Add operation, a line labeled Hex is brought up, along with either True or Complement. Using an example of True Add and the data 54 and 28, the following operation would take place. In Figure 2-20 the binary breakdown of the decimal number 54 into eight bits is 00110110 and 28 is 00011100. Adding the two numbers and using the carries, the answer is 01010010 or 82.

Figure 2-21 is the result of a complement add of binary data. The controls were decoded again from the ROS CV field which indicated a complement binary add. The CC field forces a carry into position 7 of ALU.

PACKED DECIMAL ADDITION

Two conditions must be considered to do decimal addition. First, a "pack" instruction must be given. Second, if a decimal addition is called for, an excess six circuit will be used. The operation of the excess six circuit is used only for decimal add. A decimal add would work correctly for totals not exceeding 9. However binary figures which add up to greater than 9 will not produce the correct answer. As can be seen in Figure 2-22, 15 and 14 added will result in a binary 29. The addition of 15 and 16 will not result in a correct answer as can be seen in Figure 2-23. The excess six circuit adds six (0110) to the four high positions and six to the four low positions on all decimal add operations. See Figure 2-19. The excess six circuit is not necessary on a decimal-complement add. This data as well as any data other than a decimal add will pass through the circuit undisturbed.

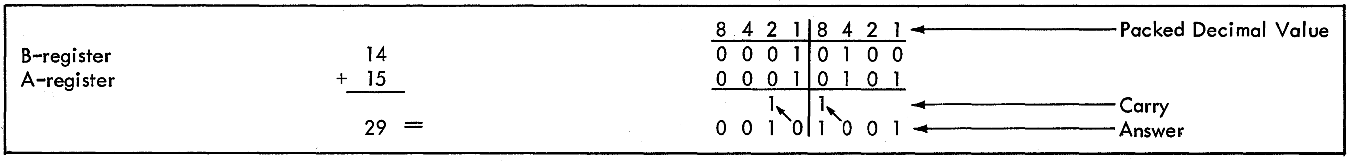


Figure 2-22. Packed Decimal Addition--No Correction

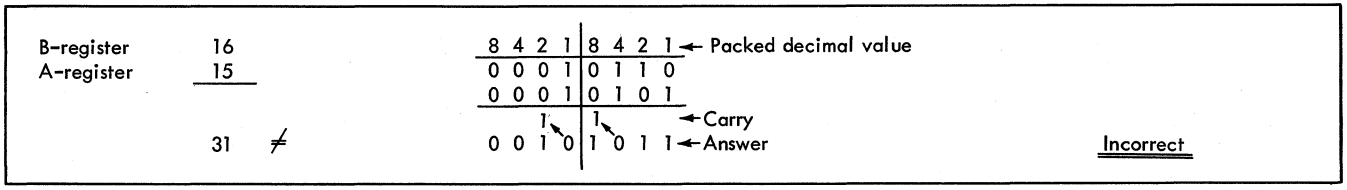


Figure 2-23. Packed Decimal Addition--No Correction

The output from the A-reg cross-high-low circuit is switched by AND/OR circuitry with the output from the T/C-excess 6 circuit under control of the ROS field, CC. The operation to be performed is dependent on the control activated. When the add control is activated, the bit entries from the A-register and the T/C + 6 circuit are added together, as described above. When controls Or, And, or Exclusive Or are activated, the logical operation indicated by the CC field is performed on the bit entries from the A-register and the T/C + 6 circuitry.

The output of the adder is fed to the decimal corrector circuit. Binary data from an add or logical operation is passed through this circuit undisturbed. For decimal operations the high-order carryout of each four bit group is investigated and if no carry exists, six is subtracted from each four bit group. Data from the decimal corrector circuit is then put on the Z bus.

An example of decimal add is shown in Figures 2-24 and 2-25. The general rules for decimal adding are as follows. First add six to each 4-bit group as it comes from the B-register. Then add the entire eight bits together as if they were binary numbers. After addition, operate only on the successive groups of 4 bits independently. Next, subtract 6 if no carry from this group occurred. For our purposes, now, keep in mind that to subtract a 6 (0110) you add the two's complement of 6 (1010). If a carry did occur from the 4-bit group, add 0000 to that group.

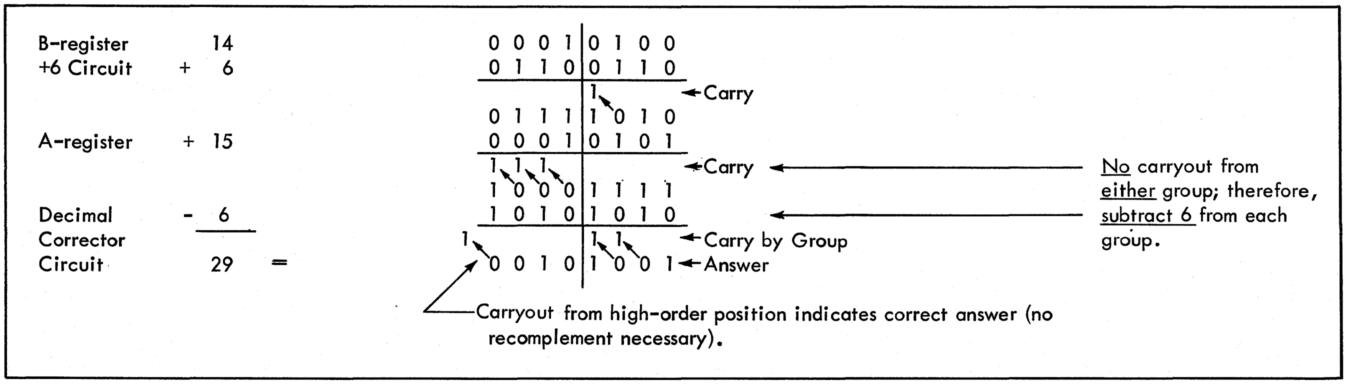

B-register 14
+6 Circuit + 6
A-register + 15
Decimal Corrector Circuit - 6
29 =
0 0 0 1 | 0 1 0 0
0 1 1 0 | 0 1 1 0
1 ← Carry
0 1 1 1 | 1 0 1 0
0 0 0 1 | 0 1 0 1
1 1 1 ← Carry
1 0 0 0 | 1 1 1 1
1 0 1 0 | 1 0 1 0
1 | 1 1 ← Carry by Group
0 0 1 0 | 1 0 0 1 ← Answer
No carryout from either group; therefore, subtract 6 from each group.
Carryout from high-order position indicates correct answer (no recomplement necessary).

Figure 2-24. Packed Decimal Addition--No Correction

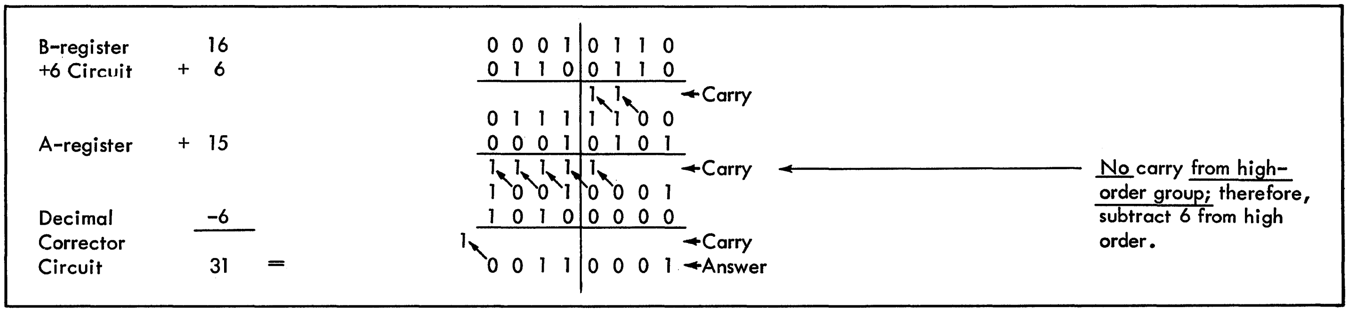


Figure 2-25. Packed Decimal Addition with Correction

Correct parity has not been mentioned up to this point as a parity line does not exist in ALU. ALU uses a two wire approach where both logical levels are used for every signal. Either an up (+L) or a down (-L) level will exist for every bit position throughout ALU. Both outputs of each signal are then run through an "exclusive or" unit to check for the presence of one and only one up level.

ALU DETAILED DESCRIPTION

The ALU unit, through the combinations of three control lines can perform the following functions:

- Adding (and subtracting by complement entry)
- ANDing (output only if both inputs are present)
- ORing (output if either input is present)
- Exclusive OR (output if only one input is present)

The inputs and outputs are divided into four-bit groups with external controls to determine if the low order group will carry into the high order group. Operation can thus be in BCD form or in pure binary.

Field 0 1 2	Value	Mnemonic	Function
0 0 0	0	0	Add Function or Block Carry
0 0 1	1	1	Force a Carry
0 1 0	2	•	AND Function
0 1 1	3	Ω	OR Function
1 0 0	4	0C	Set Carry Latch (S3) if a Carryout of ALU Occurred.
1 0 1	5	1C	Force a Carry and Set S3 if Carryout of ALU.
1 1 0	6	CC	If Carry-In, Set S3 if Carryout of ALU.
1 1 1	7	∀	Exclusive OR

Figure 2-26. Read Only Storage CC Control Field

To review briefly the control of ALU which is decoded from the CC field of RCS, the three-column field can contain a binary zero through seven, depending on the punches in the sense card. Figure 2-26 shows a breakdown of this punching and of the use of the decoded mnemonics. In addition, Figure 2-27 shows that three control lines can be obtained from various combinations of mnemonics and that a further decoding will develop definite functions which will control the operation of ALU (Figure 2-28).

Line Name	CC Field	Possible Decodes	Numeric Decode	Mnemonic
+L Control LM	Not CC2 Not CC0 CC1	0 1 0	2	.
+L Control N	Not CC1 Or CC0	X 0 X 1 X X	0 1 4 5 4 5 6 7	0 1 0C 1C 0C 1C CC ∀
+L Connect	Not CC0 CC1 or CC1 CC2	0 1 X X 11	2 3 3 7	. Ω Ω ∀

Figure 2-27. Decode of ALU Control Lines

Add Operation (also Exclusive Or)

From table 1, in Figure 2-28, it can be seen that control N is the only control line activated for A + B. Thus, lines N, LM will be up and lines N, LM and Connect will be down during add. With these controls, ACI 1 will accept an input (and its output will be down) if neither A nor B inputs have a bit or if both A and B inputs have bits. The output of ACI 2 will be down under the other two conditions, a bit on A only or a bit on B only. Assuming a bit on A only (or B) the output of ACI 1 will be up. This condition will switch with a carry at ACI 3 or a no carry at ACI 4 to provide a Sum not 1 or a Sum 1, respectively.

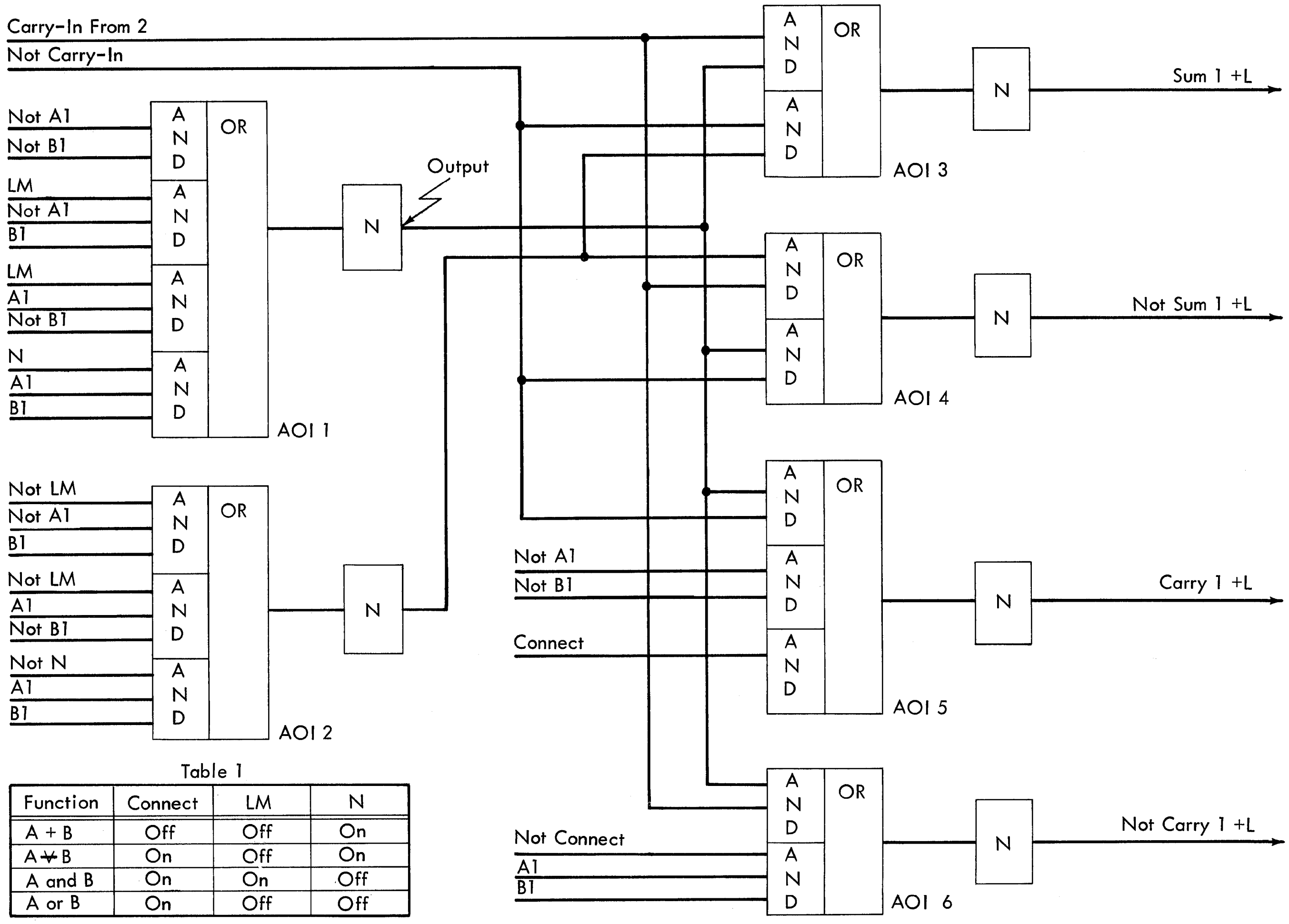


Table 1

Function	Connect	LM	N
A + B	Off	Off	On
A⊻B	On	Off	On
A and B	On	On	Off
A or B	On	Off	Off

Figure 2-28. ALU 1-Bit

If both (or neither) A and B inputs have bits, AOI 2 will provide the up level to switch with Carry or No Carry at AOI 4 or AOI 3 to provide the Sum 1 or not Sum 1 signals. The above description of this part of the add operation also applies to the Exclusive Or function.

The carry or not carry from a bit position (which will be used in the creation of the next bit sum) is created in the following manner:

Assuming a carry in condition (Carry In line up and Not Carry In line down) means that no inputs can be active to AOI 5 and that an input must be active at AOI 6. The condition at AOI 5 can be satisfied by a down level from AOI 1 (neither or both bits present at input) or by a carry into the bit one position (in this case the Not Carry In line will be down). The Not Carry 1 line is brought down through AOI 6 by either the presence of bits on both A and B or by a bit on either A or B (up level from AOI 1) together with a carry in. For the Exclusive Or operation, the Connect line is activated which effectively eliminates the carry circuit by bringing down the carry 1 line through AOI 5 and bringing up the Not Carry 1 line through AOI 6 (all inputs down).

AND Operation

The function is identical to the logical AND circuit requiring the presence of all inputs in order to create an output

= 1. This means that any output position will have a bit present only when both A and B input positions have bits. Table 1, in Figure 2- 28, shows that LM and Connect controls are ON resulting in these control lines, plus N being up. Any of the three conditions, A without B, B without A, or neither A nor B will cause the output of AOI 1 to be down and will allow the output of ACI 2 to be up. The other condition, both A and B will cause the opposite outputs with AOI 1 up and AOI 2 down. For the logical AND operation, the Not Carry In line is always up causing Sum 1 output if AOI 1 is up and a Not Sum 1 output if AOI 2 is up. The Connect control is effective in preventing a carry output as explained under Exclusive Or operation.

CR Operation

The logical "CR" function will cause a bit in any output position in which there is a bit in either input A or input B or both. The control lines up for the operation will be Connect with LM and N. Thus the output of AOI 1 will be down only if no bits are present in A or B. The output of AOI 2 will be down under the three other possible conditions, a bit in either A or B or both. As all logic operations use Not Carry In, the up output of AOI 1 will switch at AOI 4 to create a Sum, and the up output of AOI 2 will switch at AOI 3 to create the Not Sum. Again, the correct control eliminates the carry as previously described.

CARRY OPERATION

- A Carry from one position to another occurs only in ALU circuits during Hex or Decimal Add operations.

- Carries do not occur in the Decimal Corrector or Excess-6 circuits. Line Levels are "switched" for correct answers.

- The Carry Latch is the third position of the S-Register (S-3).

Carries into and out of ALU involve several circuits. As shown in the examples previously given, a carry can occur from any position of the byte, zero through seven. Within ALU, carries only occur into positions six through zero. (Carries into position seven will be confirmed later). The vehicle for transferring the carry from one position to the next is simply a line labelled Carry Bit. The line Carry 7 Bit, for example, is made active in the ALU 7 Bit circuitry and ties back to the input of ALU 6 Bit to switch with Control Lines, A-register inputs and True-Complement inputs. Developed by this circuit are the lines +L Sum 6 Bit, -L Sum 6 Bit, +L Carry 6 Bit and -L Carry 6 Bit. See Figure 2-29.

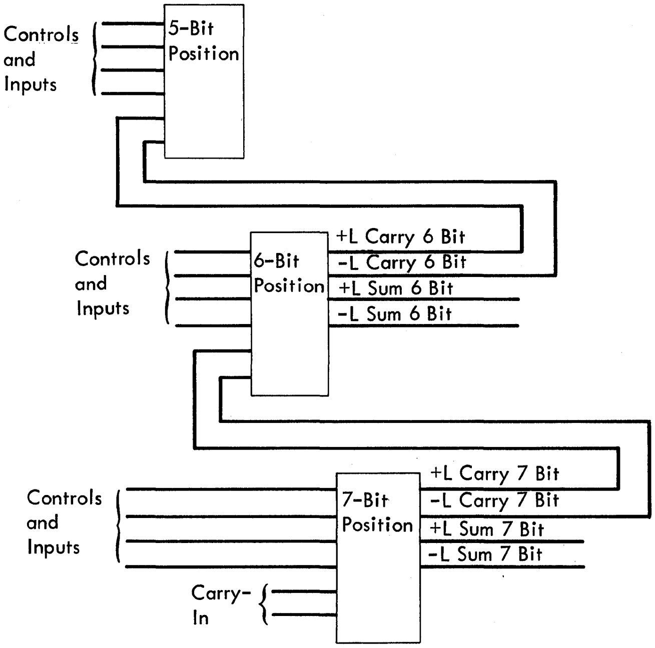


Figure 2-29. ALU Carry

ALU is the only circuit where a carry can occur from one position to another. Although carries were shown in the excess-6 and the decimal corrector operations of Figures 2-24 and 2-25, a carry line does not actually exist. The switching of various lines and levels result in the desired output. For example in Figure 2-24, Bit position 4, the final answer shows a 4 Bit ON. The result was obtained by investigating the 4, 5, and 6 positions as shown in Figure 2-30. Other outputs from these correction circuits are determined by the input lines of Carry 4. The same type of switching is done in the excess-6 circuitry. In this manner most of the arithmetic operations require just one pass through ALU. A recomplement cycle will be the next micro-program step however, if there was no carry out of the high order position.

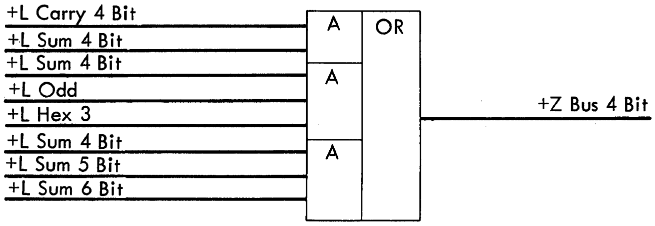


Figure 2-30. Decimal Corrector (4-Bit)

Figure 2-31 shows a composite of the carry operations including some that we have already discussed. The normal carry out and back into ALU is shown along with the carries from the zero and the four positions to the decimal corrector. In addition to these operations the Carry Zero line from the high order position is used to set the Carry Latch, which is position 3 of the S-register (S3). You will notice from Figure 2-31 that in order to set S3 a carry out of the zero position of ALU had to be switched with Gate Carry to S3, and T4 Time. Gate Carry to S3 is the result of a micro-program step which has decoded the ROS CC field. (Figure 2-26 shows that a CC 4, 5 or 6 activates the line Gate Carry to S3).

Another means of setting S3 is by decoding the ROS destination field CD. A CD 6 tells the micro-program to gate the Z-bus to the S-register. This line, switching with a Z-bus 3-bit will turn on S3.

Micro-programming can also use the S-register as a destination register. This is done by a ROS CD 6 ANDed with a Z-bus 3-bit.

To carry into the 7 position of ALU, a Carry-in latch is used. You will notice from Figure 2-31 that two AND

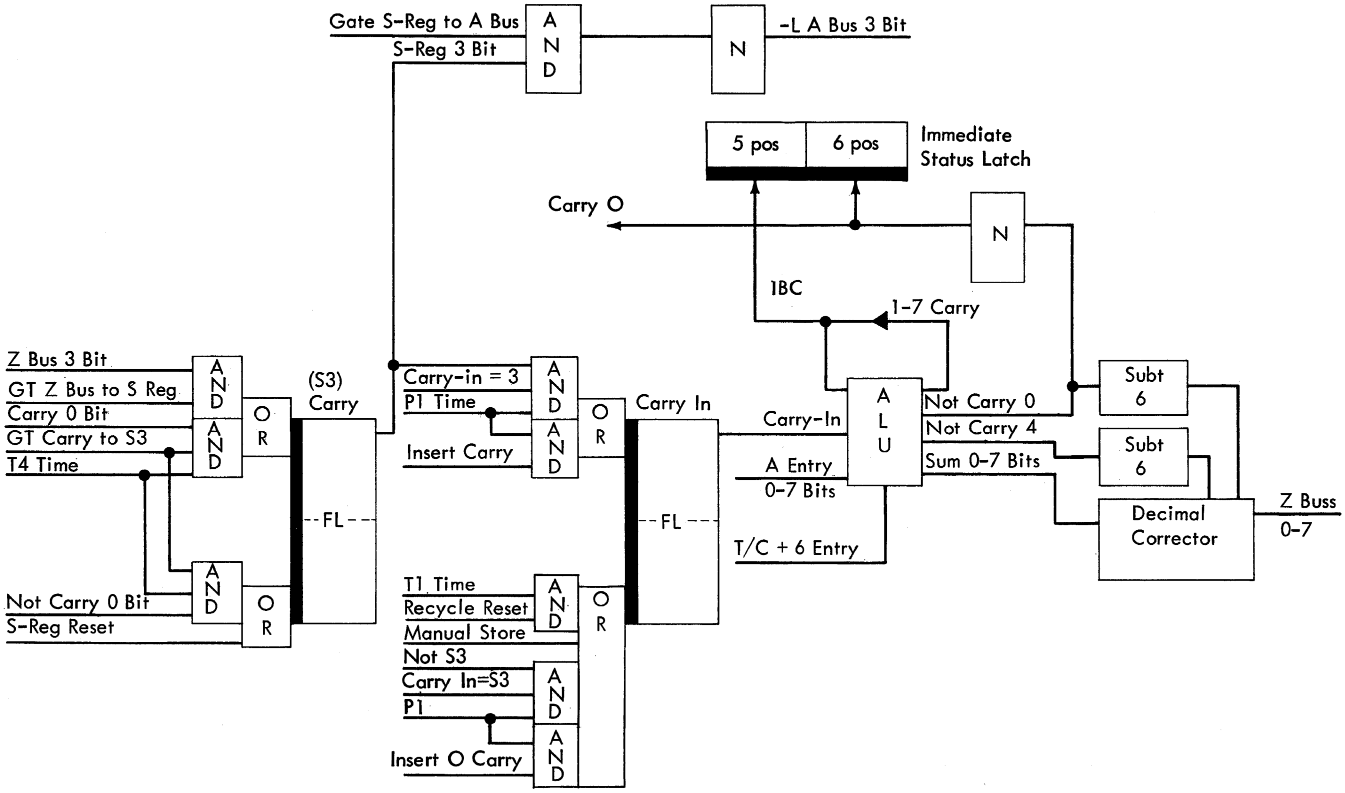


Figure 2-31. Carry Circuits

circuits can be used to set the Carry-in latch. The first allows a carry in to ALU if a Carry Out had occurred, plus resetting of S3 by a ROS field CC 6. The second set condition is an insert Carry (Force Carry), which is decoded as a 1 or a 5 in the CC field.

ALU CHECK (SEE FIGURE 2-32)

- 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.

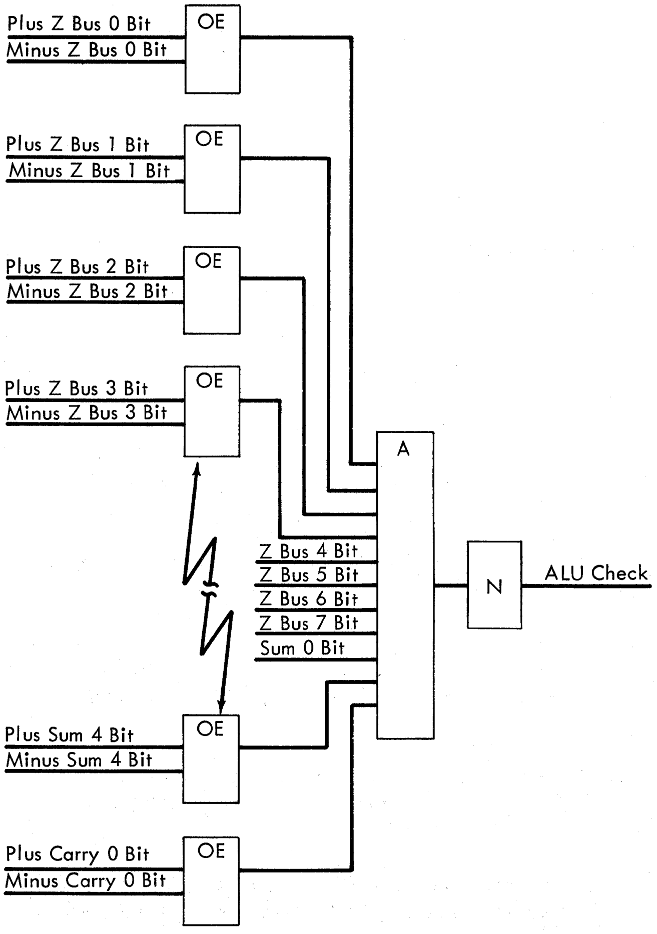


Figure 2-32. ALU Check

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 is 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 2-33 will produce a minus level out.

The output of the ALU check circuits is used several ways. See Figure 2-33.

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.

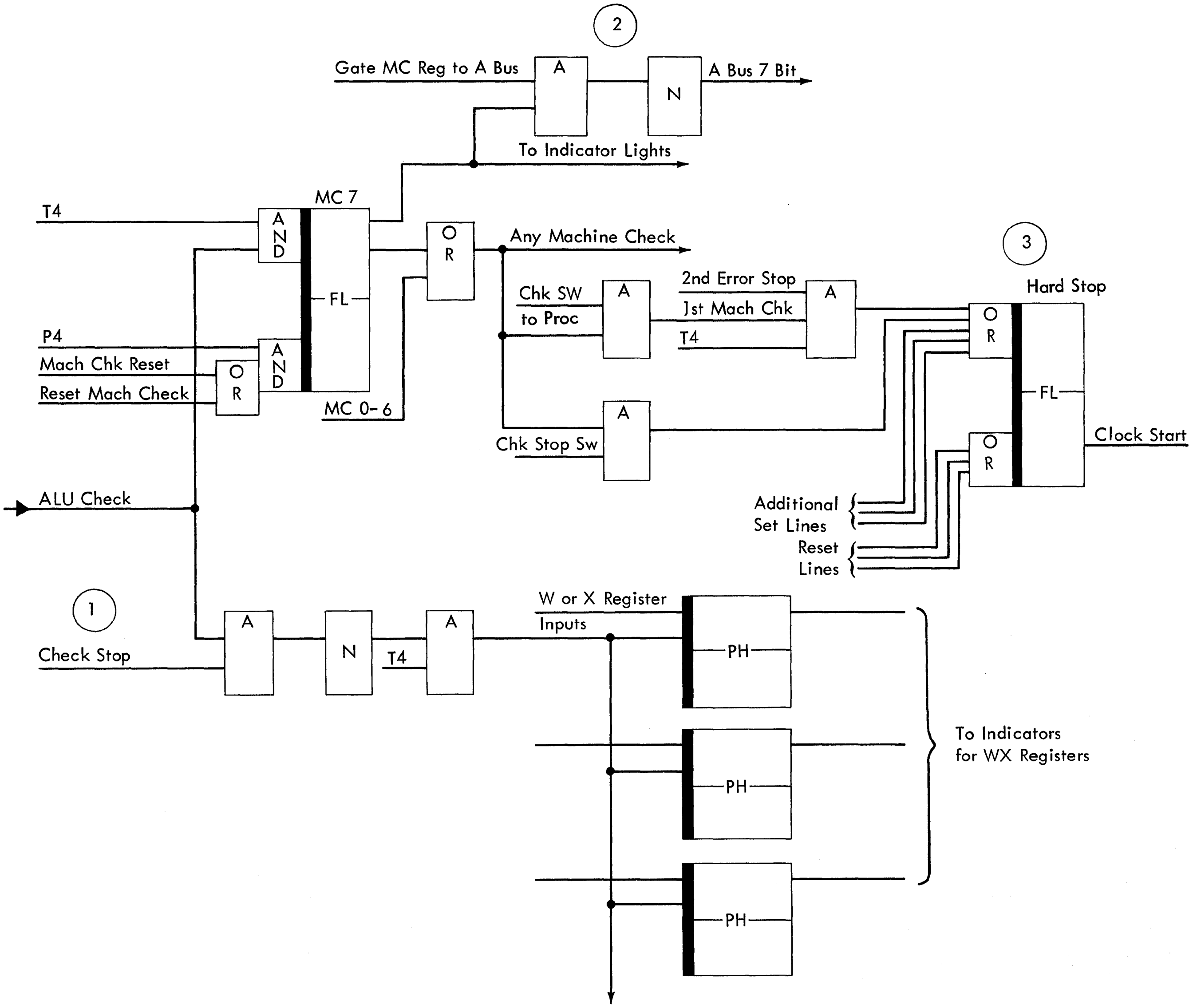


Figure 2-33. ALU check Stop Controls

REVIEW QUESTIONS - ARITHMETIC OPERATION

1. Data to ALU is supplied from ________.

2. ALU lines are ________ bits wide.

3. The control of data moving through ALU is by ________ ________.

4. List the four basic ALU operations.

5. Data can be gated to ALU from the A-register six ways. Name each.

6. Data can be gated to ALU from the B-register three ways. Name each.

7. The ________ ________ circuit, adds 6 on every _____ _____ operation.

8. A decision to subtract six is caused by ________.

9. Parity is checked by a special circuit in ALU. (True/False)

10. The Carry Latch is position ________of the ________ register.

11. What two latches must be set to carry into position seven of ALU?

REGISTERS

- The registers of the CPU are storage latches.
- Polarity Hold latches and AOI latches are used.
- Register input and output lines are controlled by ROS.

Registers have been described in the section on Data Flow as address registers, data registers, status registers and other general use registers. These registers are storage latches which can accept information, store the information and then read out the information, without destroying the data (nondestructive read out). All of these functions are under control of the ROS which supplies gates and pulses to allow movement of the data from one location to another.

There are two types of latches used in the 2030 Processor. The first and most frequently used is the Polarity Hold latch. Reviewing briefly the PH latch, the output line follows the data line when the control line is active. This means then, that the information on the data lines will be set into the register when the control line is brought up.

One example of register operation, which is similar to most registers using a PH circuit, is the R-register. This register has two input sources and three major destinations. (See Figure 2-18) One input comes from the main storage unit, consists of 9 bit lines (8 + parity), and is labeled Storage Data out_______ bit. This input is gated into the latches by memory set R. The control line is Set R-register (Figure 2-34).

The second input is from the Z-bus and is labeled accordingly. The bits on this bus are gated into the R-register by Z-bus set R and the control line Set R-register.

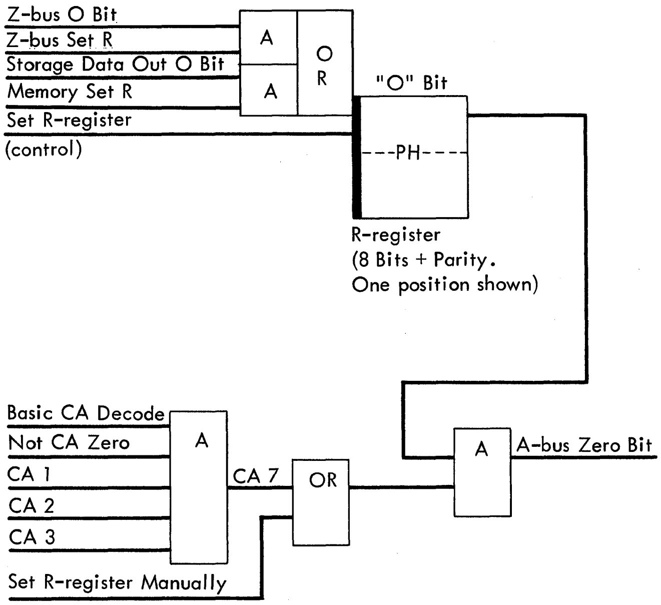
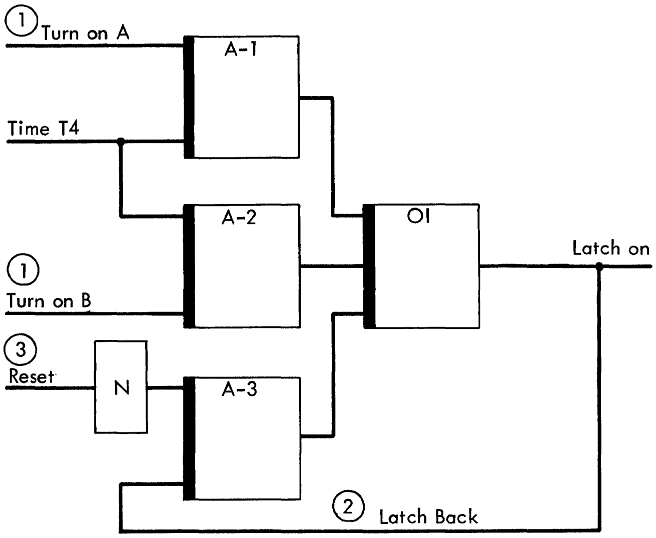


Figure 2-34. R-Register Zero-Bit Latch

All of the nine PH latches in the R-register turn on as described above. However, the parity latch has one additional turn on, which would not be present on other registers and is therefore not shown in Figure 2-34. In order to prevent an R-register parity check at T2 time, the parity latch is turned on by a line which comes from the machine reset circuitry. This prevents a completely blank R-register from turning on the Machine Check 6 latch at T2 time or after a machine reset.



(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 2-35. One Latch Position of a Register

The second type of latch used for registers is one which is made of several block functions tied together. They can be called ACI latches as they use AND functions and OR inverter functions. A latch back line from the output OI keeps the latch on, as shown in Figure 2-35. The latch can have many turn-on lines and will stay turned on until reset. This allows greater flexibility, as the output lines can be tested and switched with other conditions to control operations several cycles later. This ACI latch is used for:

1. the F-register

2. the S-register

3. the control register

4. the machine check register

REVIEW QUESTIONS - REGISTERS

1. Registers are used as ________ latches.

2. Registers are controlled by ________.

3. The two types of hardware used for latches are ________ ________ latches and ________ latches.

4. Register read out is non-destructive. (True/False)

5. All registers read out to the B-bus but only three registers read out to the A-bus. (True/False)

CORE STORAGE

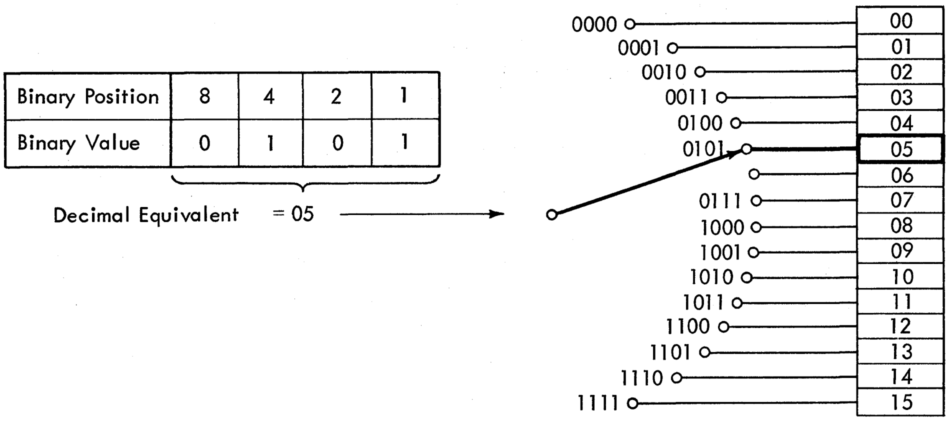
ADDRESSING THEORY

Four-Digit 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.

Using four binary digits, 16 storage locations can be assigned addresses. All items numbered 0000 (decimal 00) that we wish to store are placed in storage location 0000 (decimal 00); all items numbered 0001 (decimal 01) that we wish to store are placed in storage location 0001 (decimal 01); etc. It is now possible to find any item by selecting the storage location with the proper number. In Figure 2-36 storage location 0101 (decimal 05) has been selected by the combination of binary digits that represent the decimal number 05.



Binary Position	8	4	2	1
Binary Value	0	1	0	1

Figure 2-36. Four-Digit Addressing

Six-Digit Addressing

- Six binary digits form 64 different storage addresses.

- Address range from 000000 to 111111 (00-63 decimal).

If the original four binary digits 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 (00 - 63 decimal).

There are several ways to apply the six binary digits to an addressing scheme. For the purpose of this discussion, it is most convenient to expand the original 4-digit addressing scheme shown in Figure 2-36. Thus, in Figure 2-37, the four low-order binary digits describe some number in the range 0000-1111 (00 to 15 decimal), 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 (15 decimal) combine with the two high-order digits 10 (32 decimal) to select storage location 101111 (47 decimal).

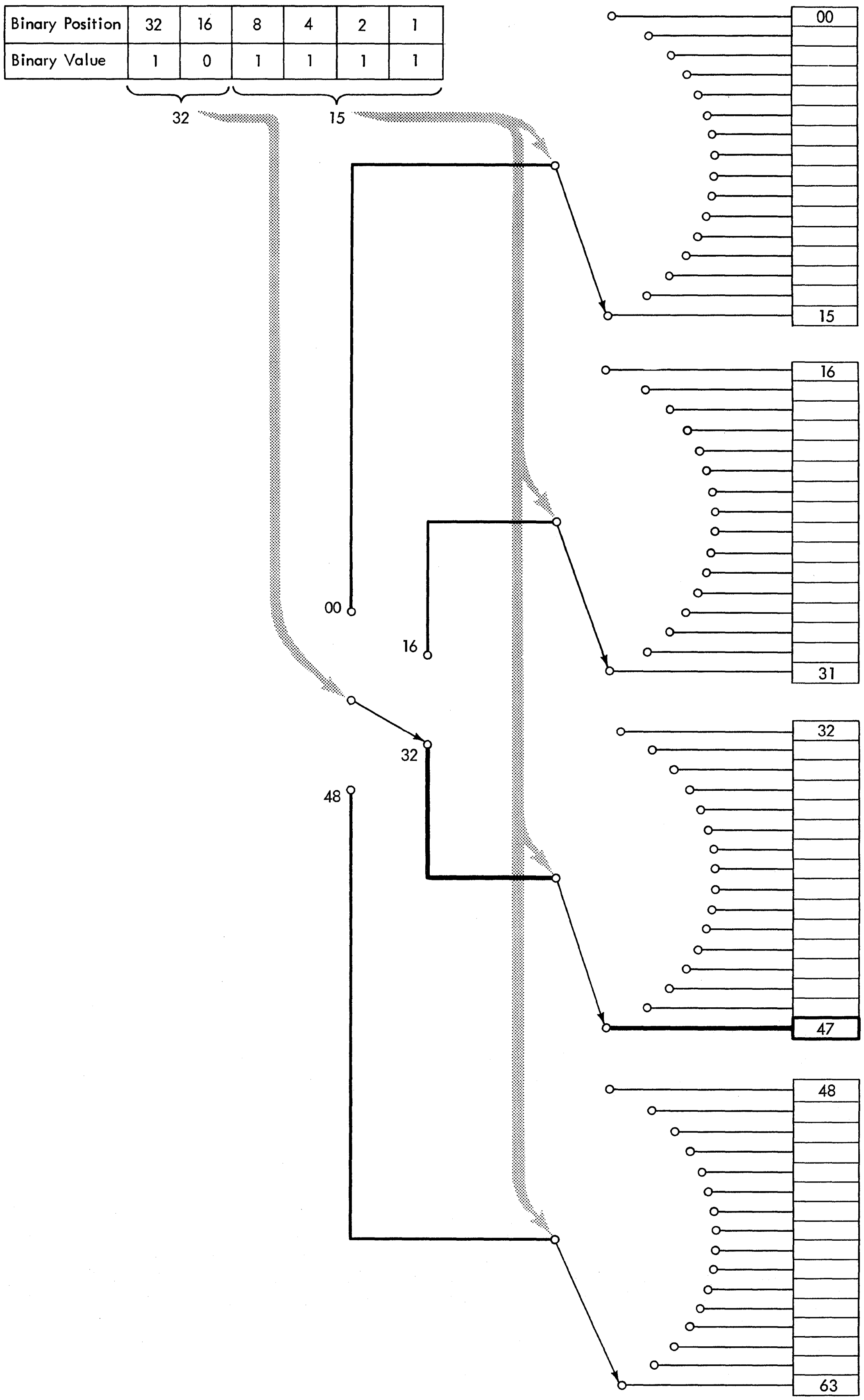
Ten-Digit Addressing

- Ten binary digits form 1,024 different storage addresses.

- Address range from 0000000000 to 1111111111 (0000-1023 decimal).

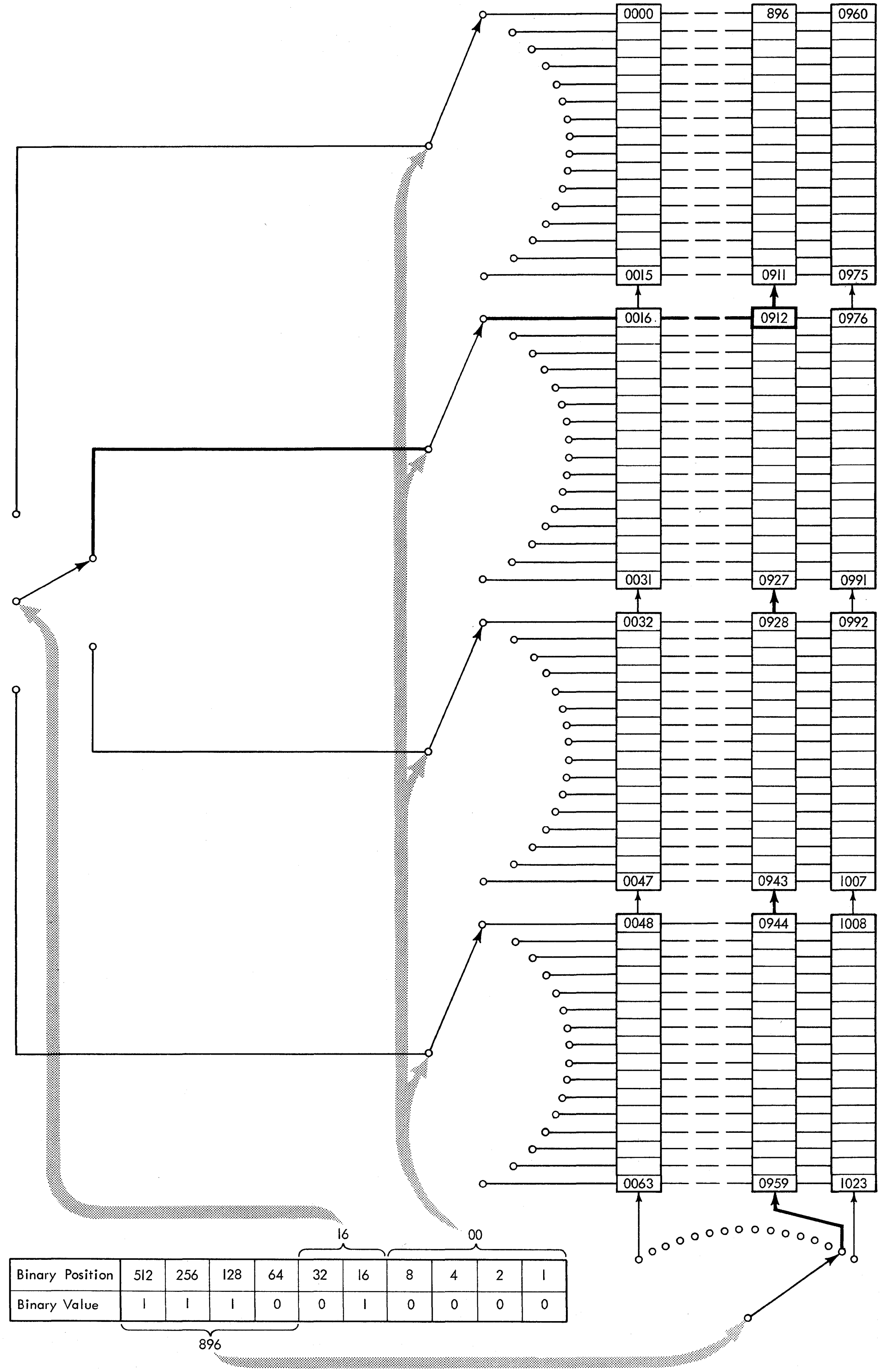
If four additional binary digits are added to the 6 digit addressing scheme, it is possible to define 1,024 storage locations (2^{10}=1,024). To accommodate the extra bits in the addressing scheme, it is necessary to add another dimension (Figure 2-38).

The four low-order binary digits 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 digits describe one of four blocks of 256 storage locations. Each of these blocks is made up of 16 blocks of 16 storage locations each. The six low-order digits have narrowed the selection to 16 storage locations. With four high-order digits, it is possible to make a final selection of one of these 16. In the example shown, the four low-order digits 0000 (00 decimal), plus the next two digits 01 (16 decimal), plus the four high-order digits 1110 (896 decimal), combine to form 1110010000 (912 decimal).



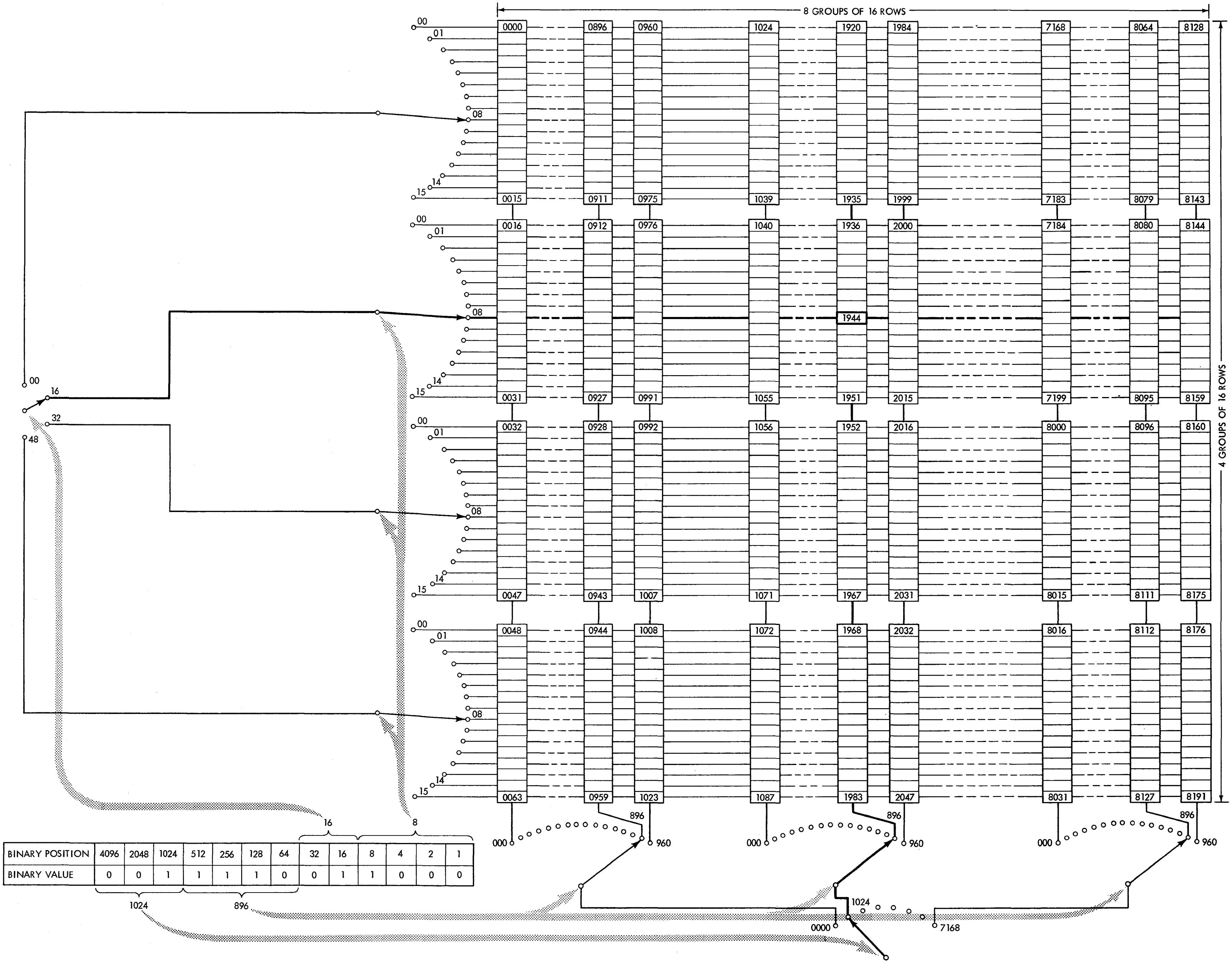
Binary Position	32	16	8	4	2	1
Binary Value	1	0	1	1	1	1

Figure 2-37. Six-Digit Addressing



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 2-38. Ten-Digit Addressing



BINARY POSITION	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
BINARY VALUE	0	0	1	1	1	1	0	0	1	1	0	0	0

Figure 2-39. Thirteen-Digit Addressing

Thirteen-Digit Addressing

- Thirteen binary digits address 8,192 storage locations.
- Address range 0000000000000 to 1111111111111 (0000-8191 decimal).

In the first example of addressing (Figure 2-36), selection depended on one group of binary digits. This was expanded to selection by three groups of binary digits (Figure 2-38). 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 digits 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 2-39). These additional three binary digits provide a fourth direction to the addressing. Basic addressing is the same as shown in Figure 2-38 except that now there are eight groups of 1,024 storage locations. The three additional digits 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 made 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 2-39 are not satisfactory. However, it is possible to apply the same addressing scheme to faster storage devices. An investigation into the properties 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.

The basic core storage size is 8,192 positions. Using the 13-digit addressing scheme described, a unique binary address can be assigned to each of the 8,192 positions.

MAGNETIC CORE THEORY

- A magnetic core is a small, doughnut-shaped object.
- Made of ferromagnetic material.
- 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.
- External magnetizing force 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 ferromagnetic 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 2-40). 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.

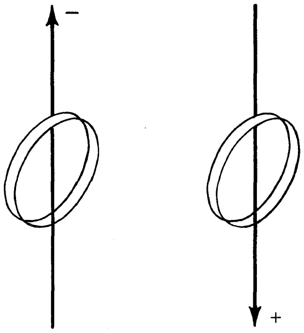


Figure 2-40. Magnetic Core

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.

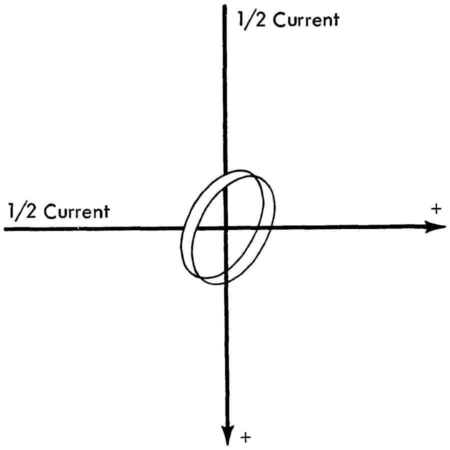


Figure 2-41. Half-Current Principle

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 2-41).

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 2-42). 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.

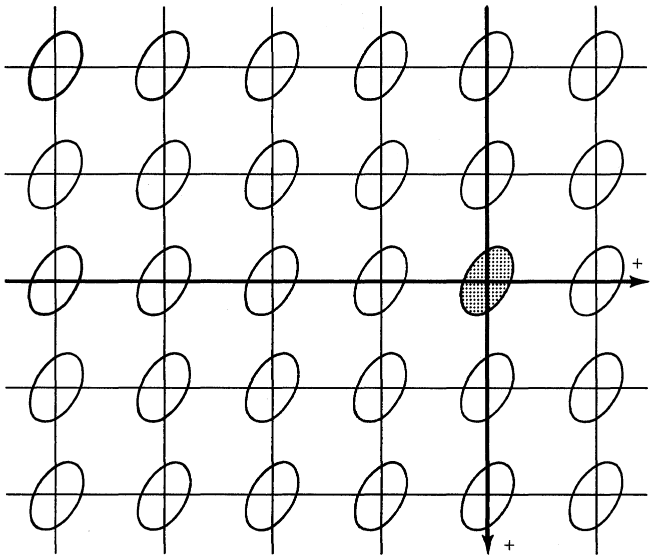


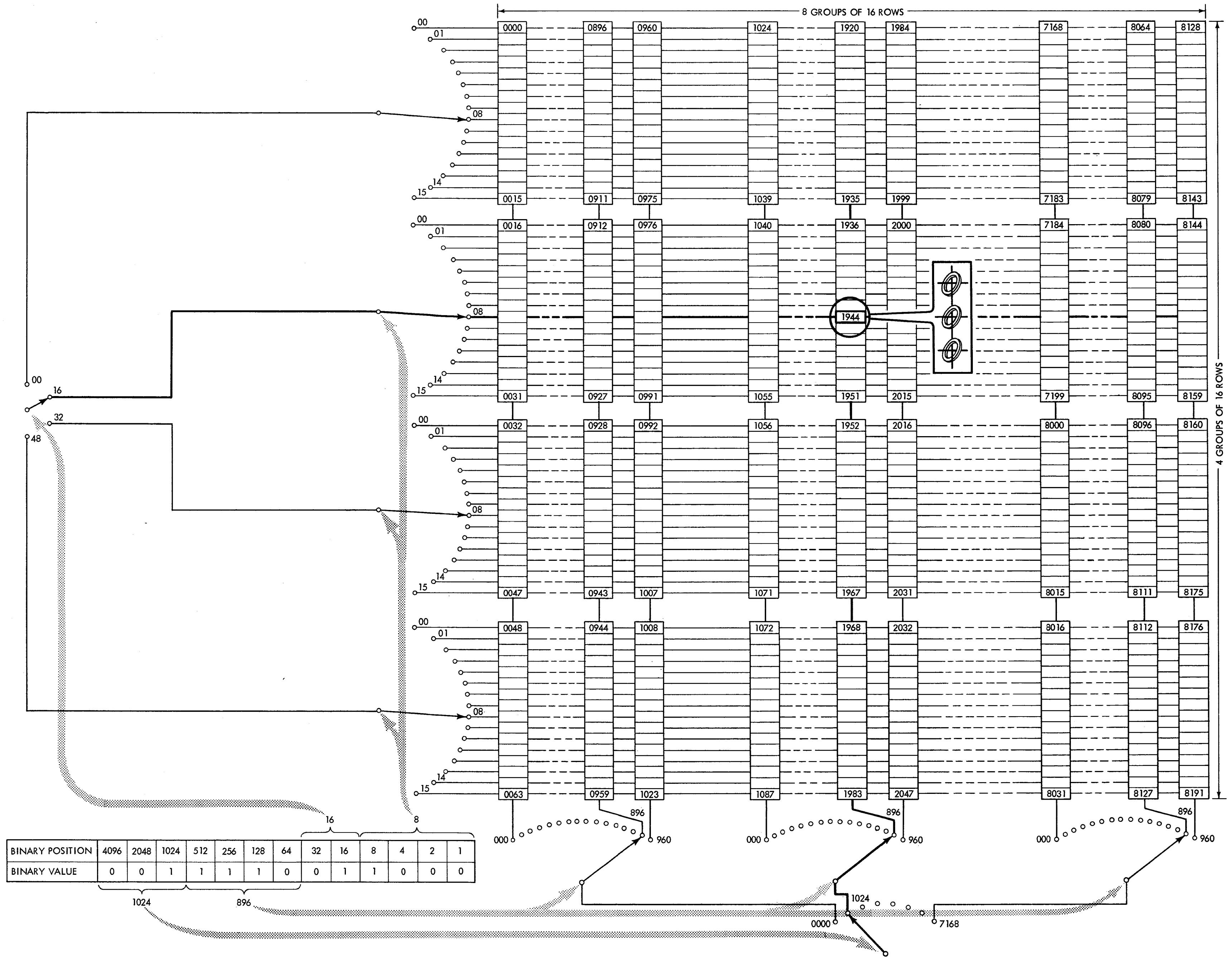
Figure 2-42. Coincident Current Addressing

Core Storage Addressing

- Transistors select address and provide selection current.
- Current flow in drive line determines core magnetic polarity.

It is possible to apply the 2-wire, coincident-current addressing scheme to a larger number of cores. In Figure 2-43, magnetic cores have been added to the 8,192-position, 13-digit storage addressing scheme explained previously. Each box 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 (Figure 2-44). The core shown can be flipped to the opposite state by changing the position of the battery control switch. The four address selection switches shown correspond to the address selection switches in Figure 2-43.

The use of switches for address selection and polarity control 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 2-45 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.



BINARY POSITION	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
BINARY VALUE	0	0	1	1	1	1	0	0	1	1	0	0	0

Figure 2-43. 8K Storage

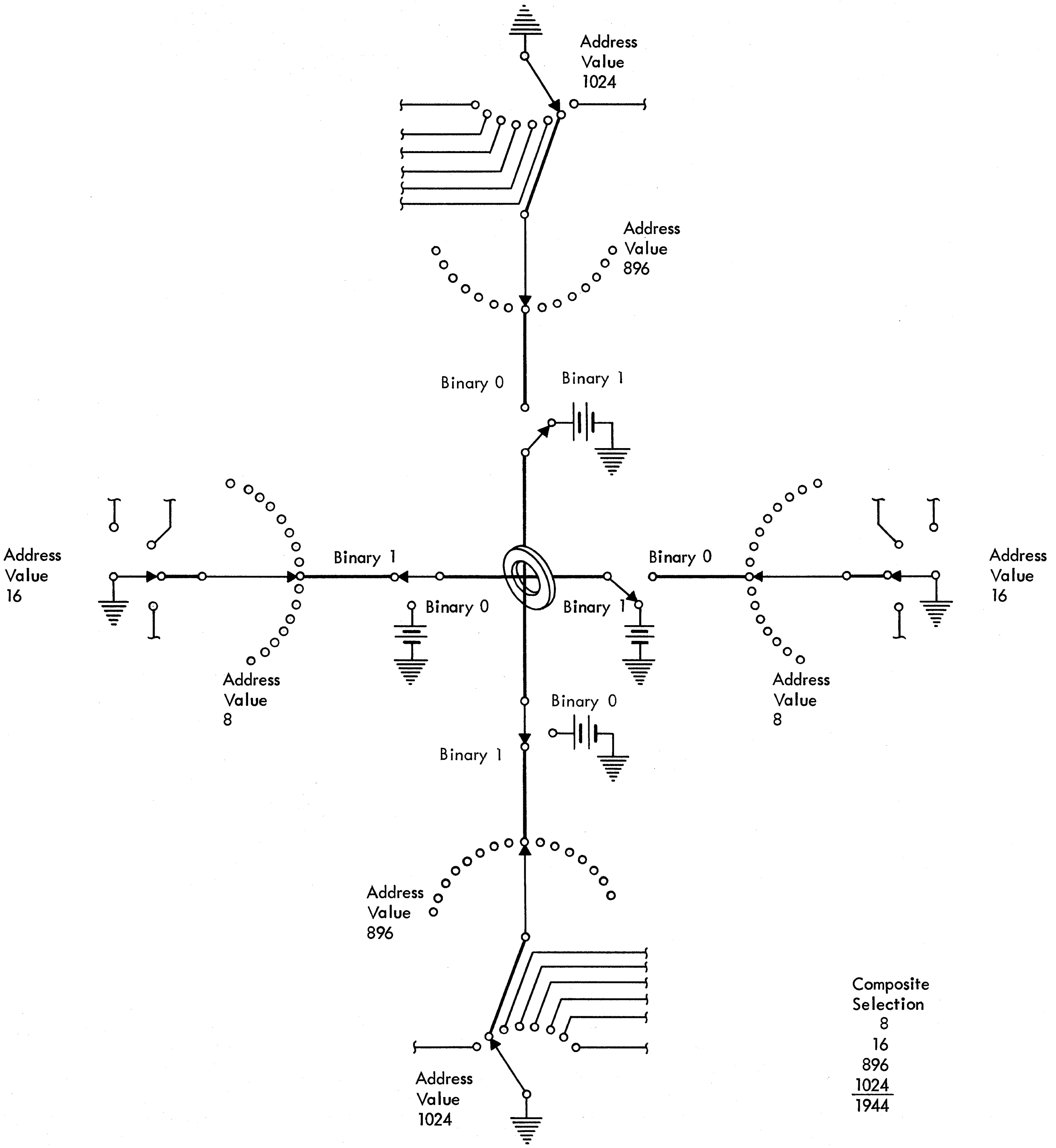


Figure 2-44. Magnetic Core Drive

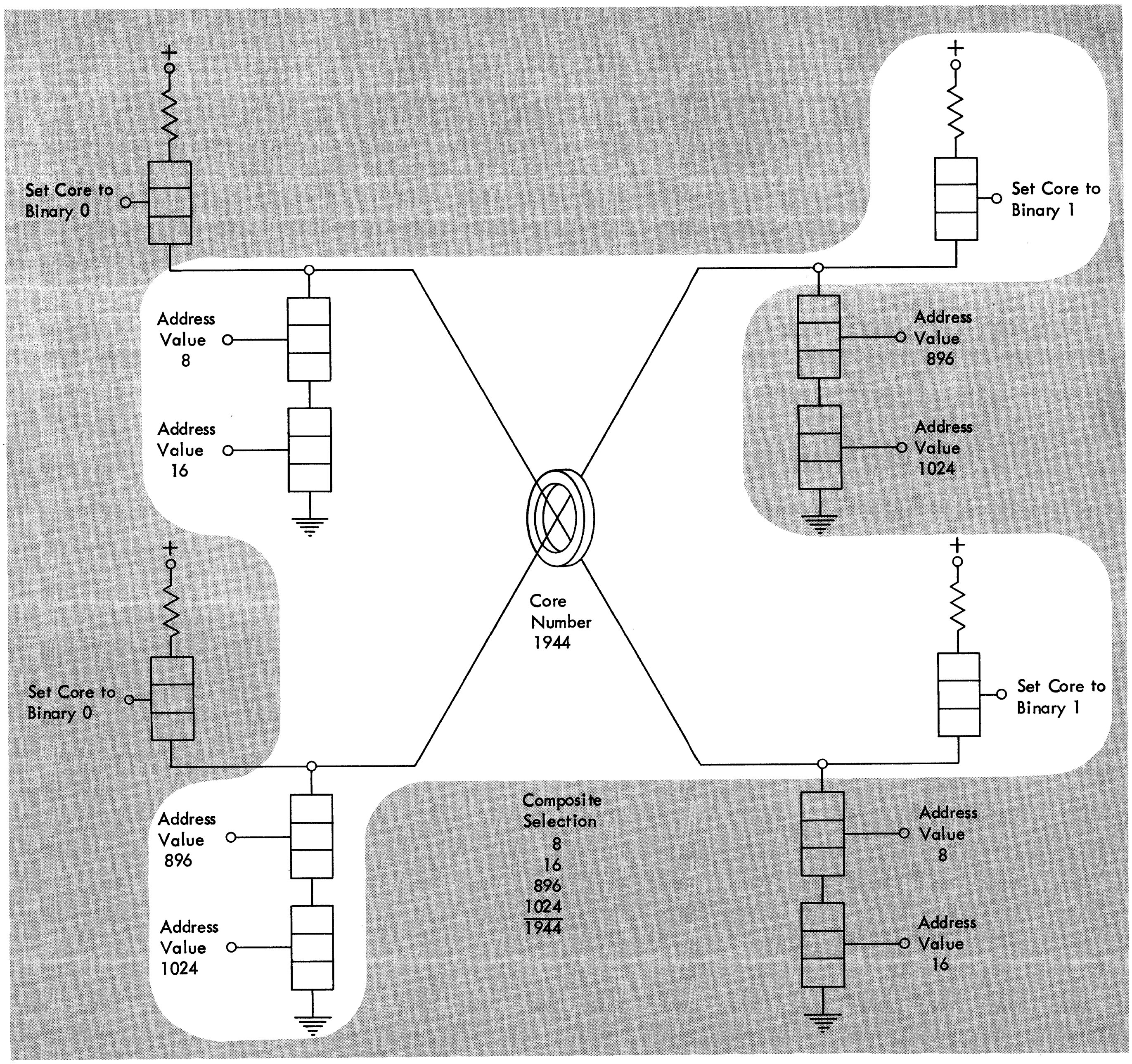


Figure 2-45. Core Storage Drive

Sense

- Core magnetized to either of two polarites, 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.

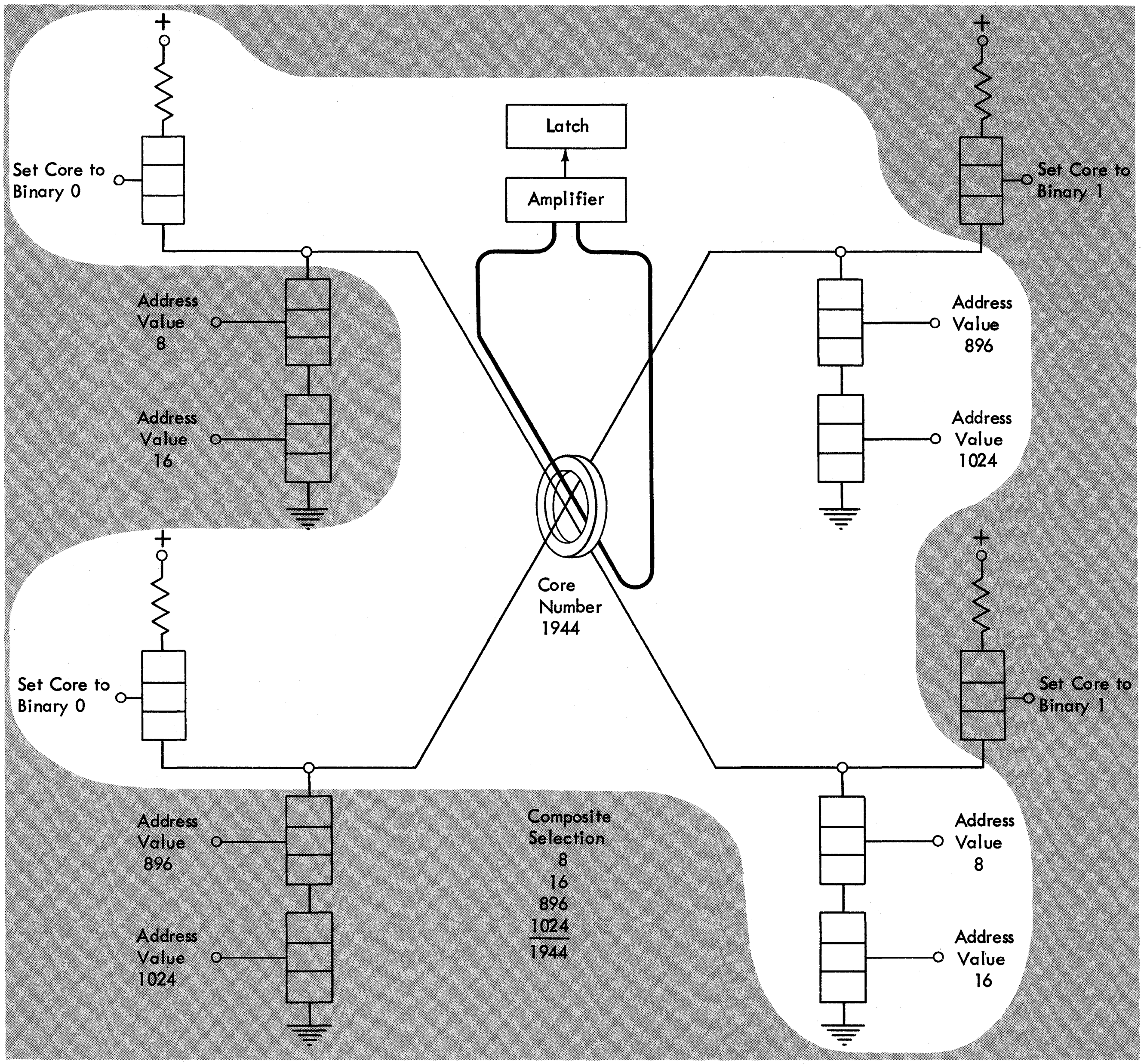


Figure 2-46. Core Read

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 <u>writing</u>, and the coincident current that causes writing is called <u>write current</u>. When information is to be retrieved form 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 2-46). Retrieving this stored information from core storage is called reading, and the coincident current that causes reading is called <u>read current</u>. 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.

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.

<u>STORAGE ADDRESS REGISTER</u>

- 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 2-47). Used together, these registers provide 65,536 different numbers, ranging from 00,000 to 65,535. These numbers are the core-storage addresses for the core-storage unit in the 2030.

Name	M-Register								N-Register							
Position	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
Binary Value	32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1

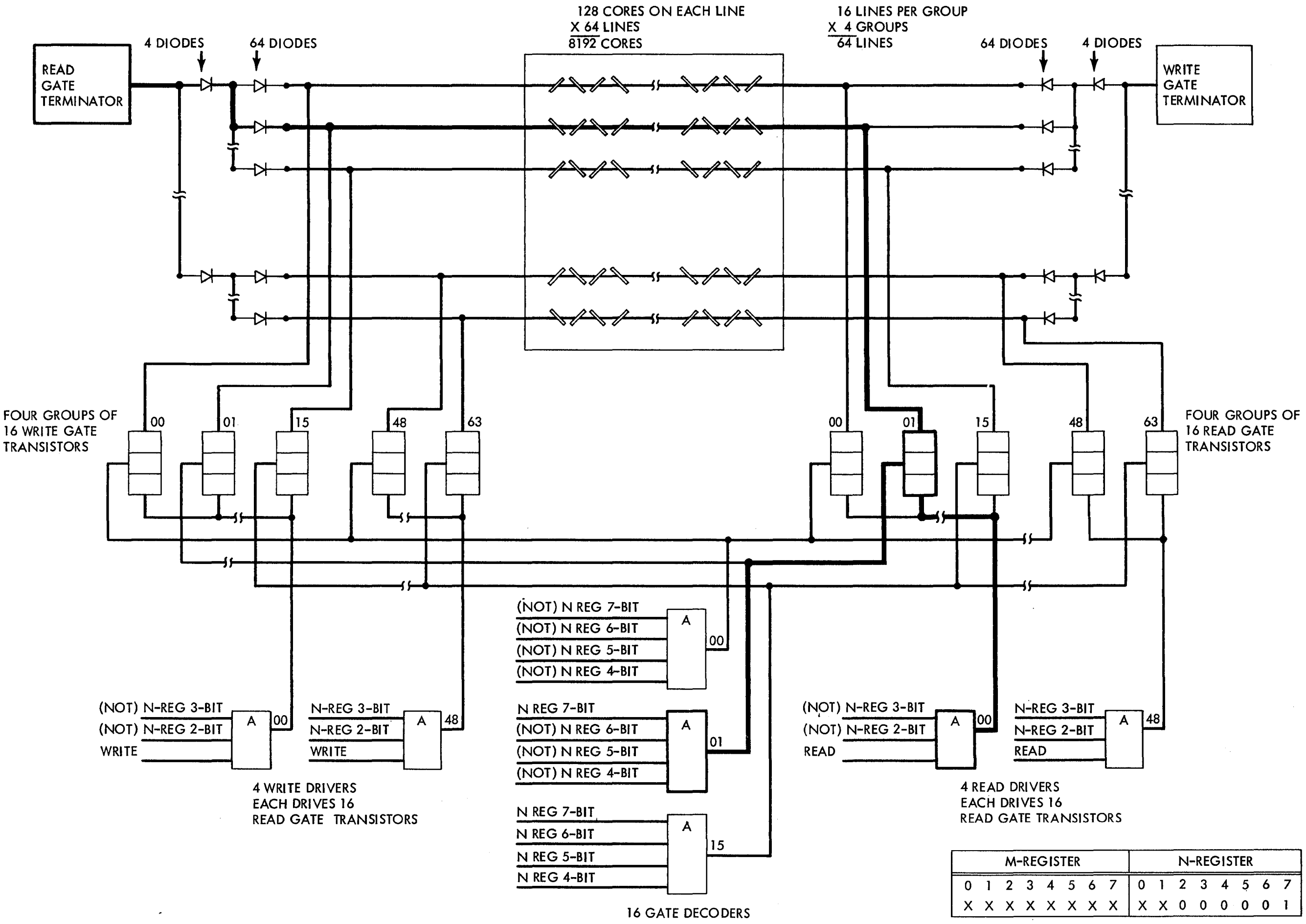
Figure 2-47. Storage Address Register

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. The remaining 3 high-order bits are used to complete the addressing scheme up to the maximum core storage size available (65,536 bytes).

Address Decode

- 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.

The examples of Figures 2-45 and 2-46 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. min 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 2-48.



M-REGISTER								N-REGISTER							
0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
X	X	X	X	X	X	X	X	X	X	0	0	0	0	0	1

Figure 2-48. X-Drive Address Decode

Four drivers and 16 gate decode switches define which gate transistor is to conduct. In Figure 2-48, a single X-line has been selected to read from the binary value shown as follows: The 1-, 2-, 4-, and 8-bits (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 16- and 32-bits (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 64-, 128-, 256-, and 512-bits (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 1,024-; 2,048-; and 4,096-bits (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.

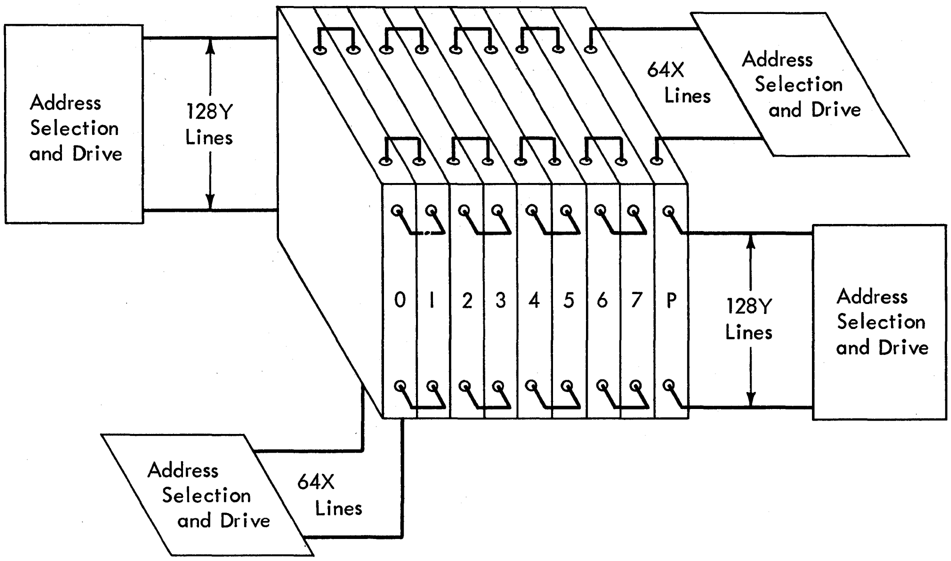


Figure 2-49. Core Plane Stacking

- 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 informaion cores plus a parity core). In Figure 2-49, 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.

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 2-49 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 2-46, 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 2-50). 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 combination sense-inhibit winding is threaded through all the cores in each of 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 (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

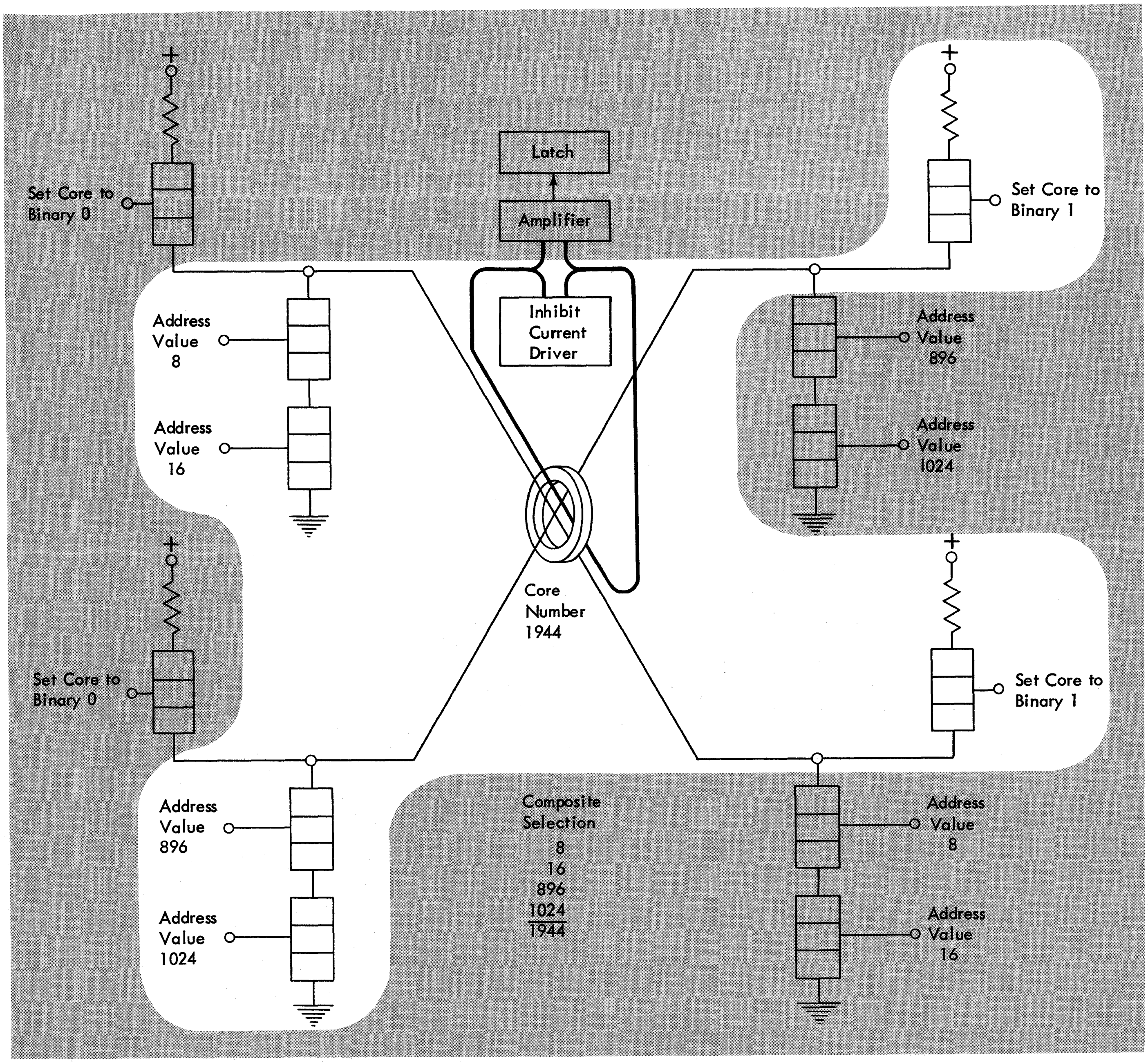


Figure 2-50. Inhibit

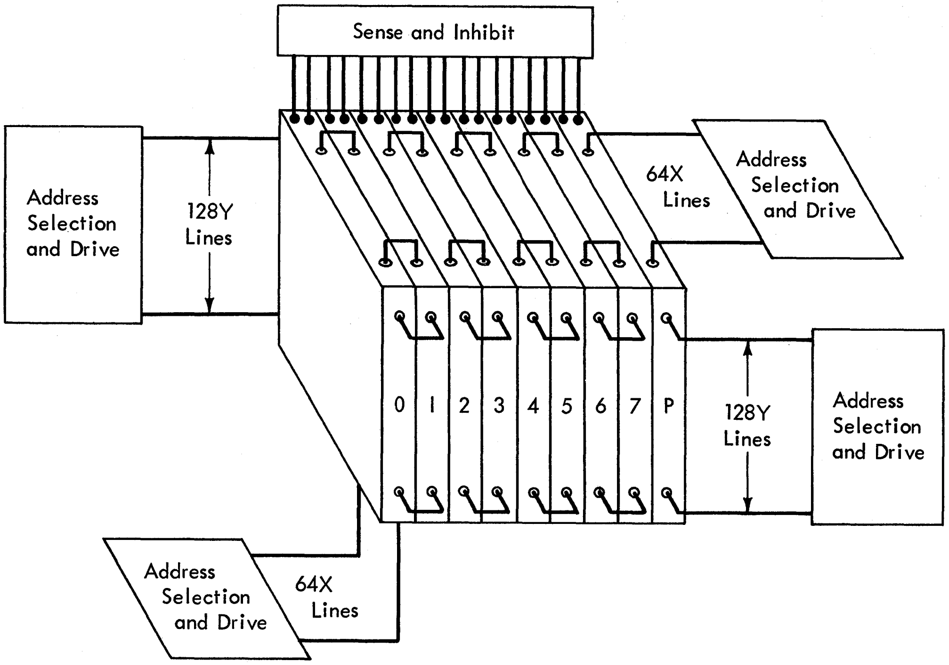

Sense and Inhibit
Address Selection and Drive
128Y Lines
64X Lines
Address Selection and Drive
0
1
2
3
4
5
6
7
P
128Y Lines
Address Selection and Drive
Address Selection and Drive
64X Lines

Figure 2-51. Composite Core Layout

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 2-51).

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.

8K AUXILIARY STORAGE

- 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 multiplexor channel. The other 256 positions of local storage are used by the CPU for special and general purpose registers (Figure 2-52).

The additional 512 storage positions are formed by adding eight lines in the Y-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 000-255 is placed in the MN-register, it refers to one of three storage positions. The desired position may be in main storage, CPU local storage, or 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.

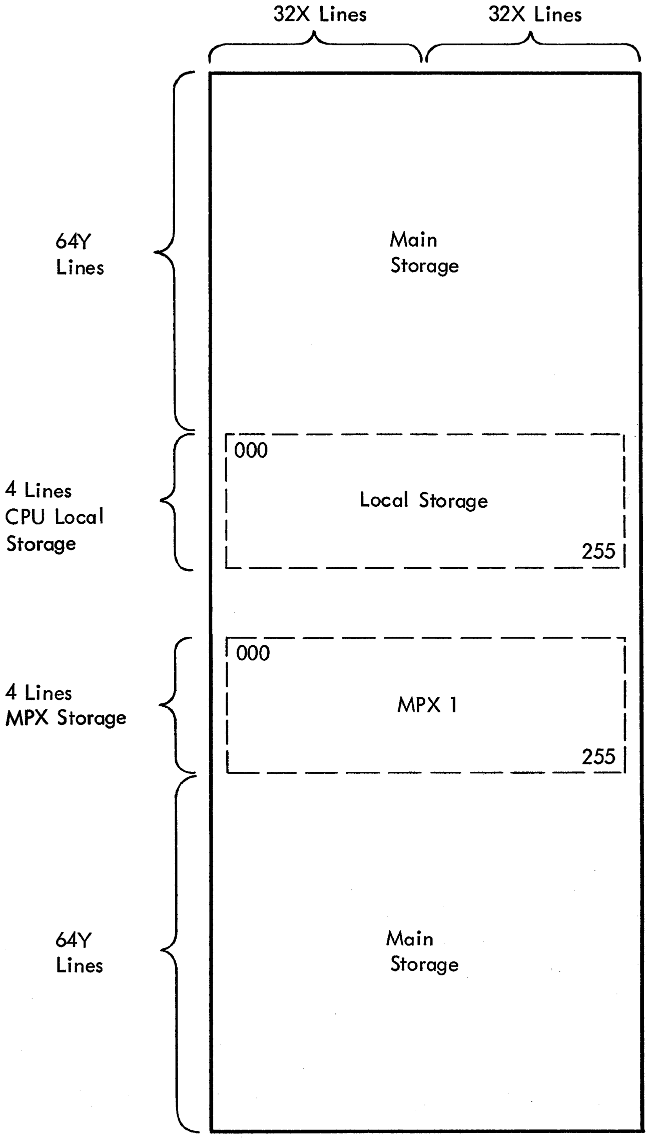


Figure 2-52. Local Storage

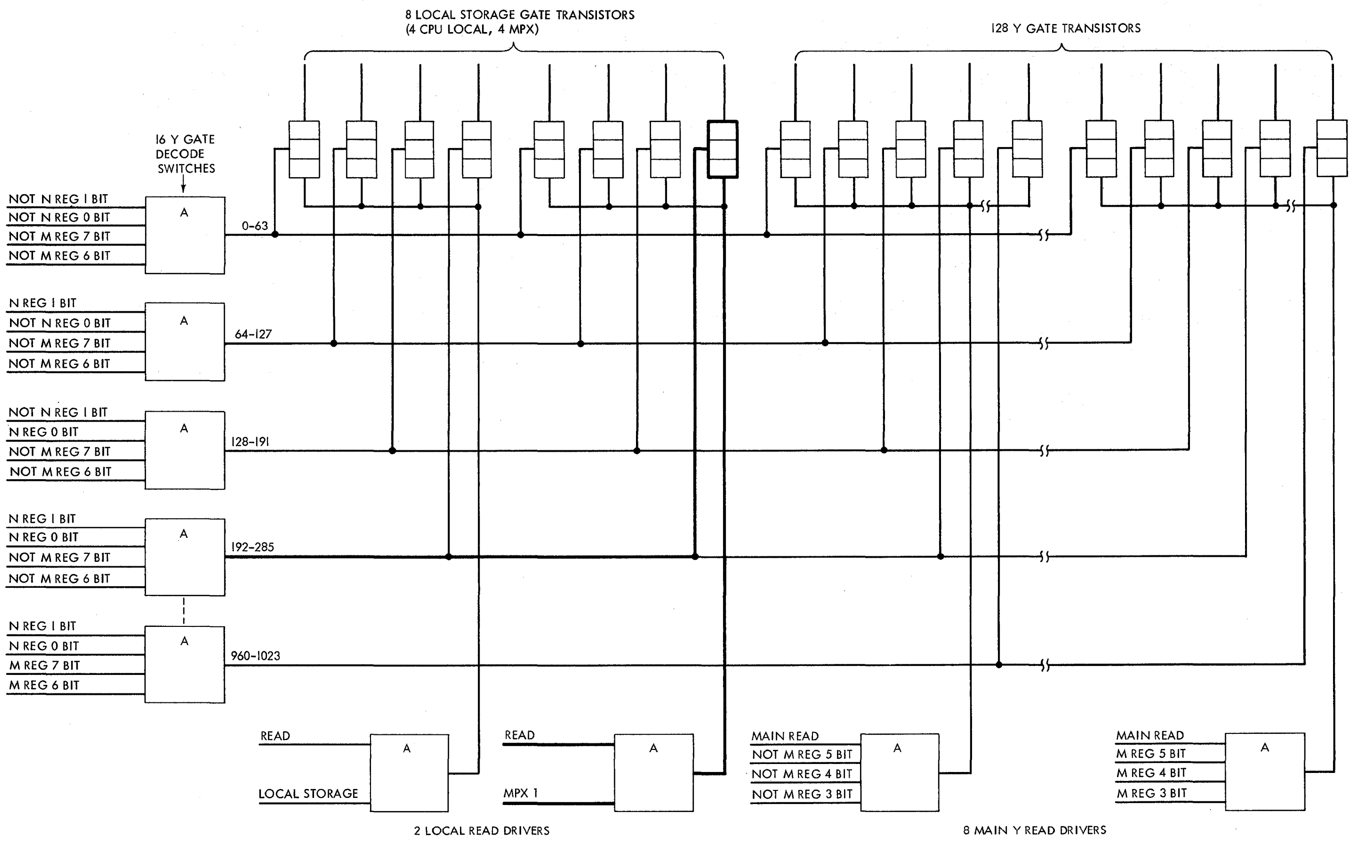


Figure 2-53. Local Storage Gate Decode

Figure 2-53 shows auxiliary Y read-gate selection when the address 174 is placed in the MN-register. The Y gate-decode switch that is turned on by the MN-register contents conditions the bases of 18 Y gate-transistors (16 main Y gate-transistors and two auxiliary 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.

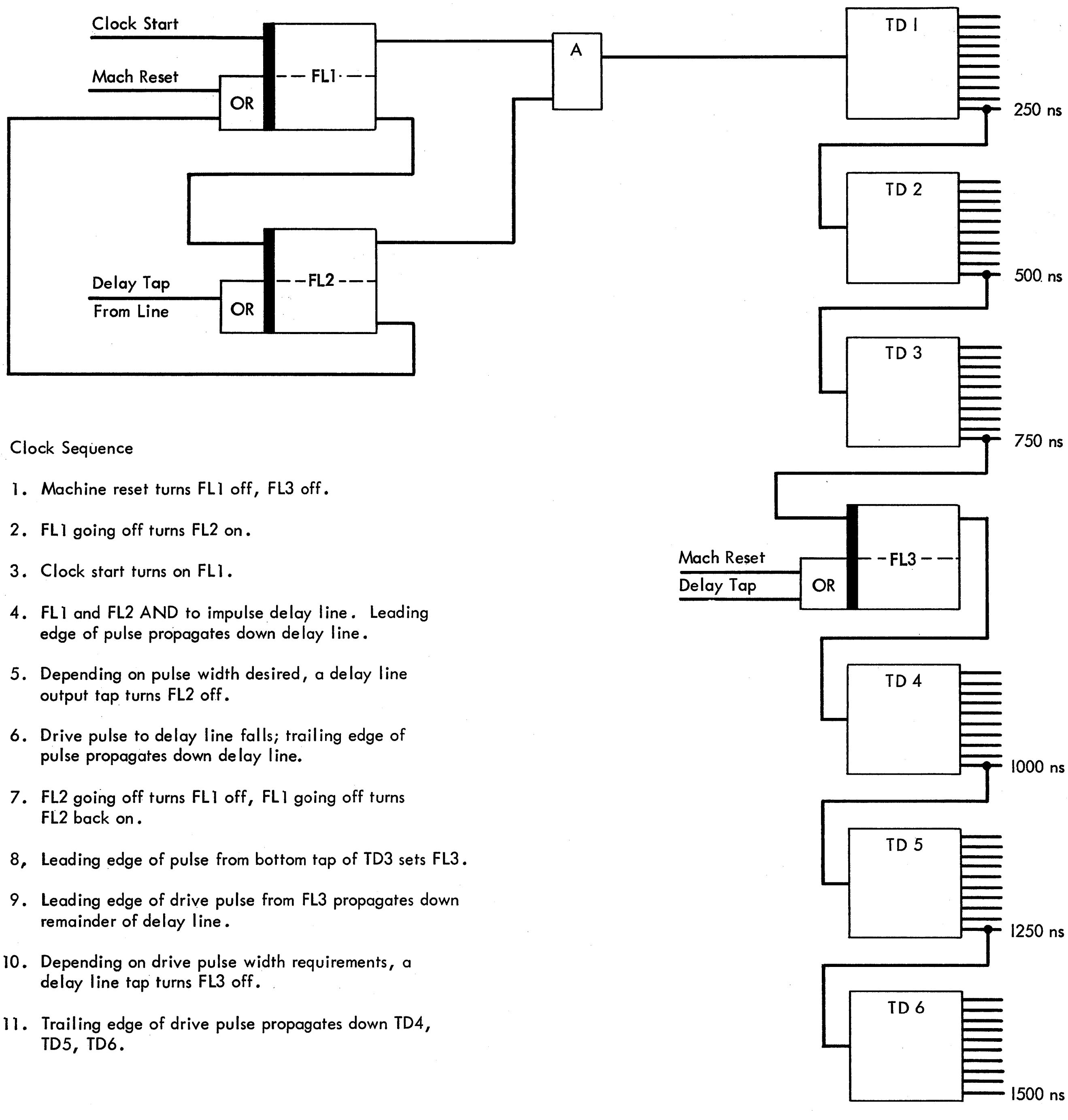
STORAGE CLOCK

- 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.
- Clock started by signal from CPU.
- Once started, clock operates for 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 is 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 form 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 MN-register. The storage circuitry is signaled to read and the storage clock is started (Figure 2-54). 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 2-54. 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 2-55 and 2-56).

8K 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 condition 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 2-48).

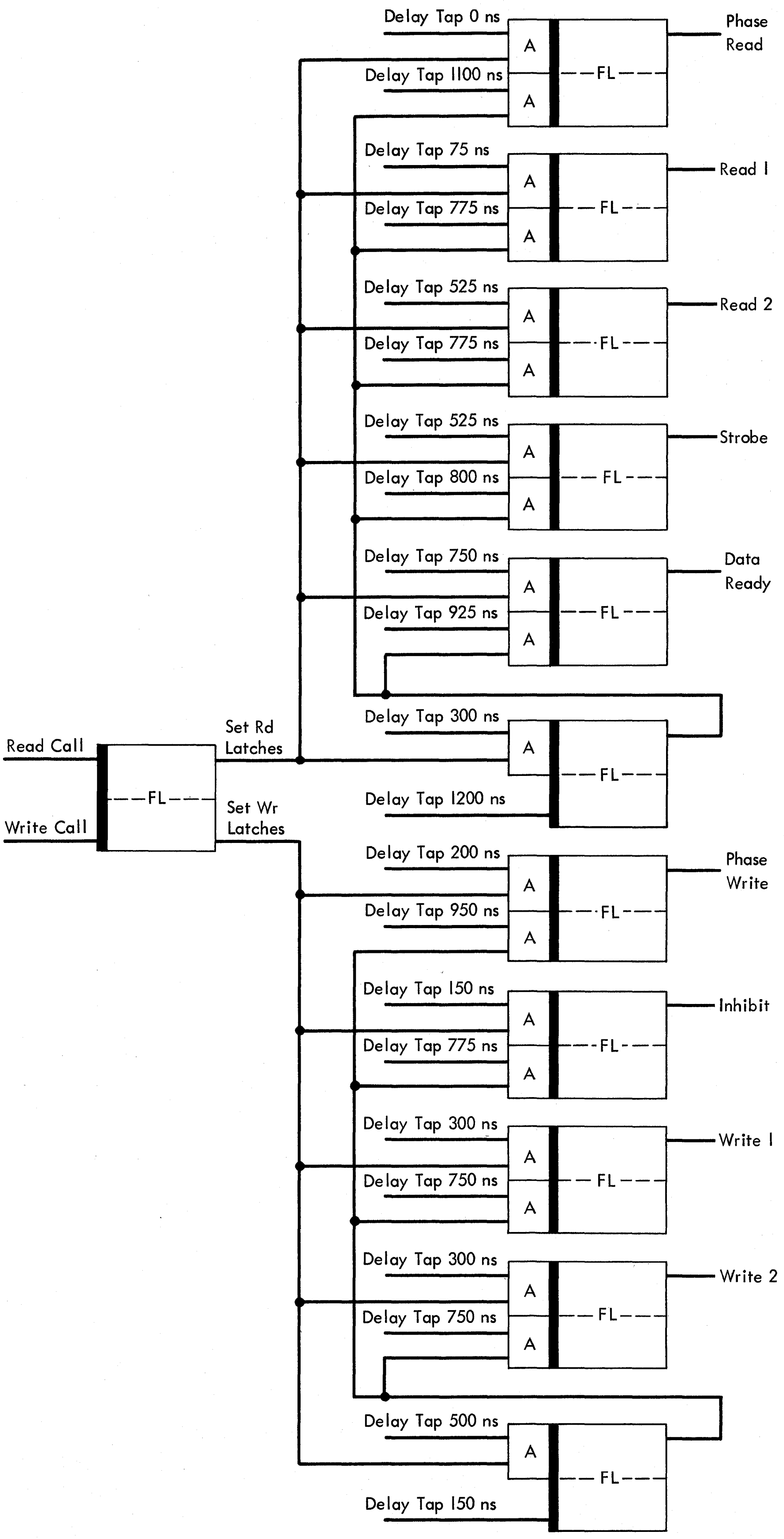


Figure 2-55. Memory Clock

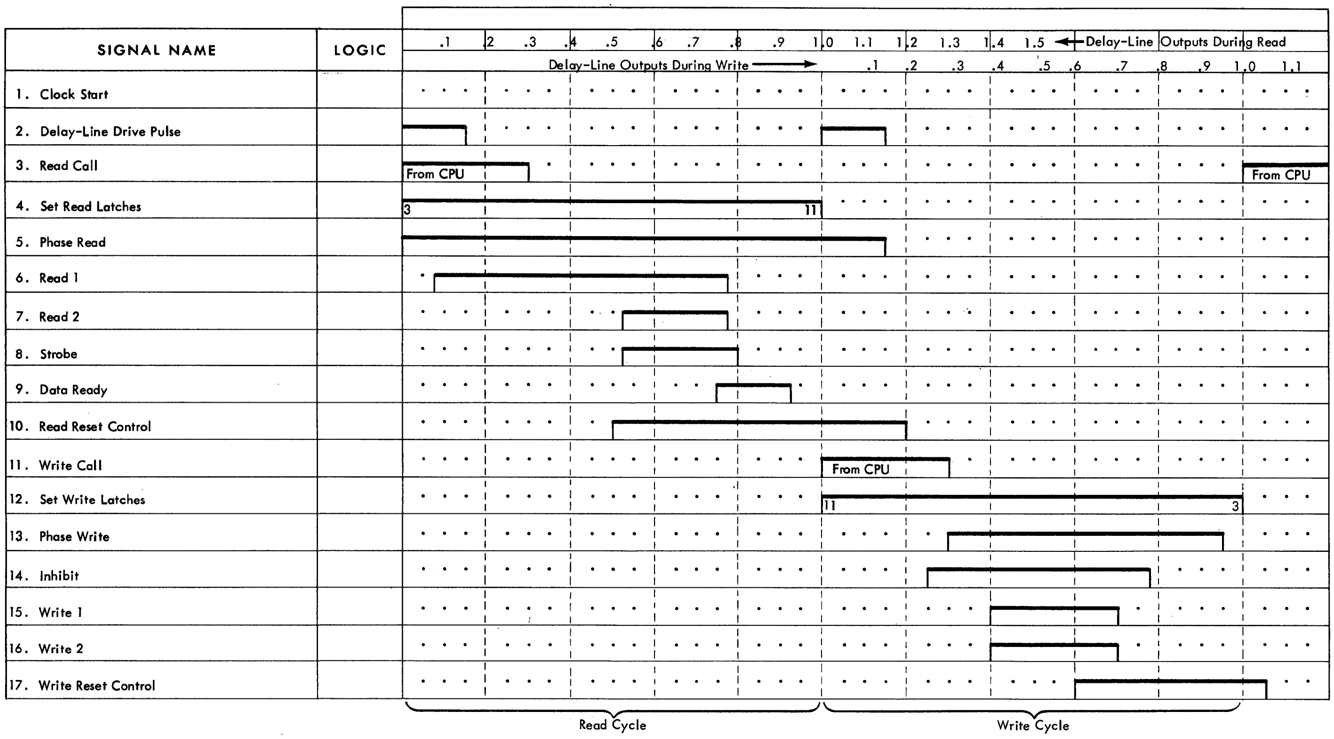


Figure 2-56. Memory Clock Timings

PHASE REVERSAL ADDRESSING (16K)

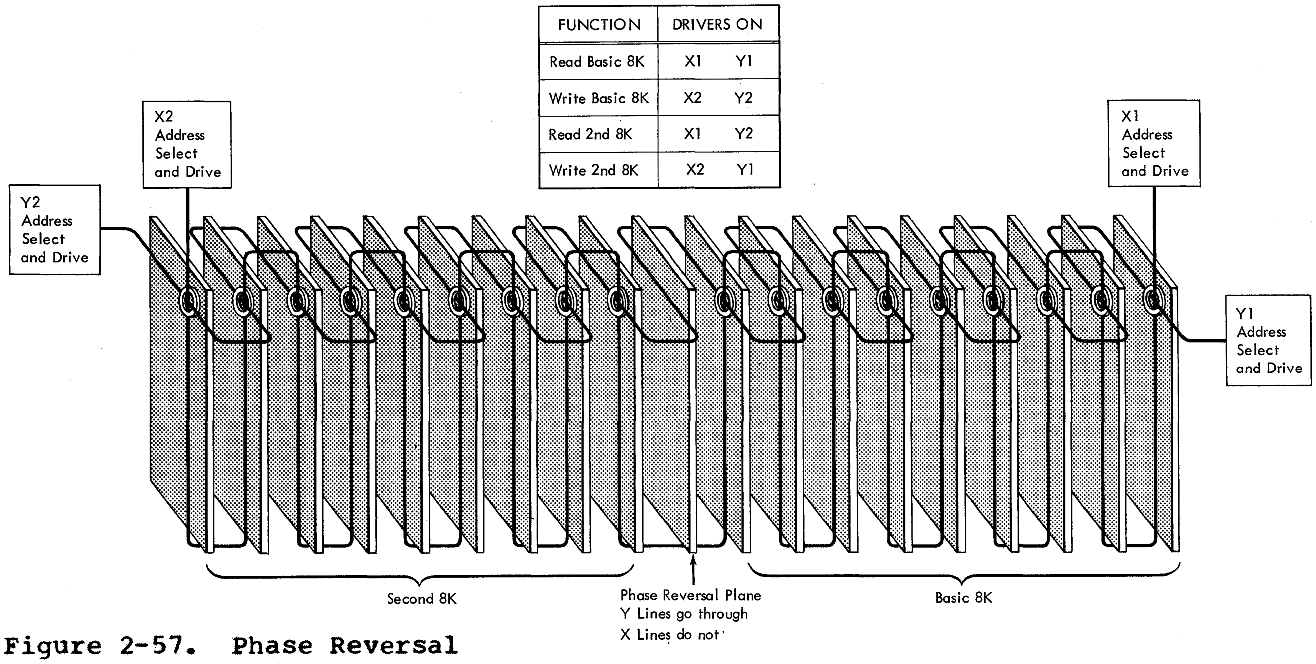
- Phase reversal principle allows twice as many storage positions to be addressed with the same drive circuitry.

- 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.

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 2-57).



FUNCTION	DRIVERS ON
Read Basic 8K	X1 Y1
Write Basic 8K	X2 Y2
Read 2nd 8K	X1 Y2
Write 2nd 8K	X2 Y1

X2 Address Select and Drive
Y2 Address Select and Drive
X1 Address Select and Drive
Y1 Address Select and Drive
Second 8K
Phase Reversal Plane
Y Lines go through
X Lines do not
Basic 8K

Figure 2-57. Phase Reversal

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 2-57. 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 2-57 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, and 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.

16K AUXILIARY STORAGE

- Four 256-byte auxiliary storage areas included in a 16K storage unit.
- M-register 2-bit and 3-bit determine auxiliary storage to be addressed
- N-register determines specific address from 000-255
- CPU local storage 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 storage, MPX 1 storage, MPX 2, and CPU 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	CPU local

For example, if the CPU wishes to address a byte of information in the CPU local 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 16K auxiliary storage unit 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 local 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 2-57.

16K 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)

- 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 2-58). 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).

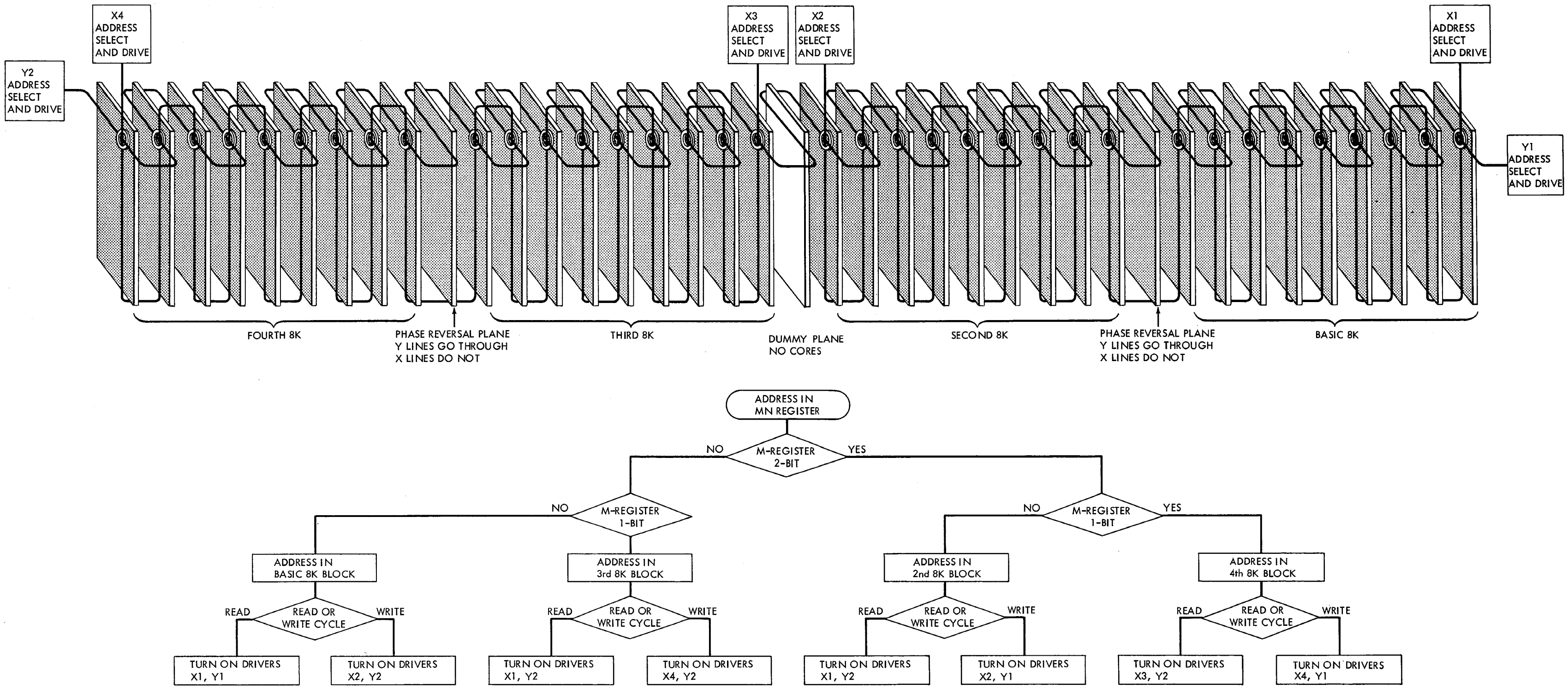


Figure 2-58. 32K Phase Reversal

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 0000-8191, 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 8192-16383. This selects the second 8K block. The third 8K block has the address range of 16384-24575, 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 24576-32767 is selected. The drivers are controlled by the M-register 2- and 1-bits and the read or write signal.

32K AUXILIARY STORAGE

- 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 a 32K storage unit is four 256-byte areas (MPX 0, MPX 1, MPX 2, and CPU local storage). These four areas are located in the first 16K of storage, and are addressed as described under <u>16K Auxiliary Storage</u>. A special 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 CPU 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	CPU Local

CLOCK CONTROL ADDRESSING (64K)

- 64K core-storage unit composed of two separate 32K core storage units.
- Each 32K unit completely independent of the other.
- 0-bit position of M-register determines which storage clock is started

Each 32K core-storage unit is a complete package that cannot be 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 2-59).

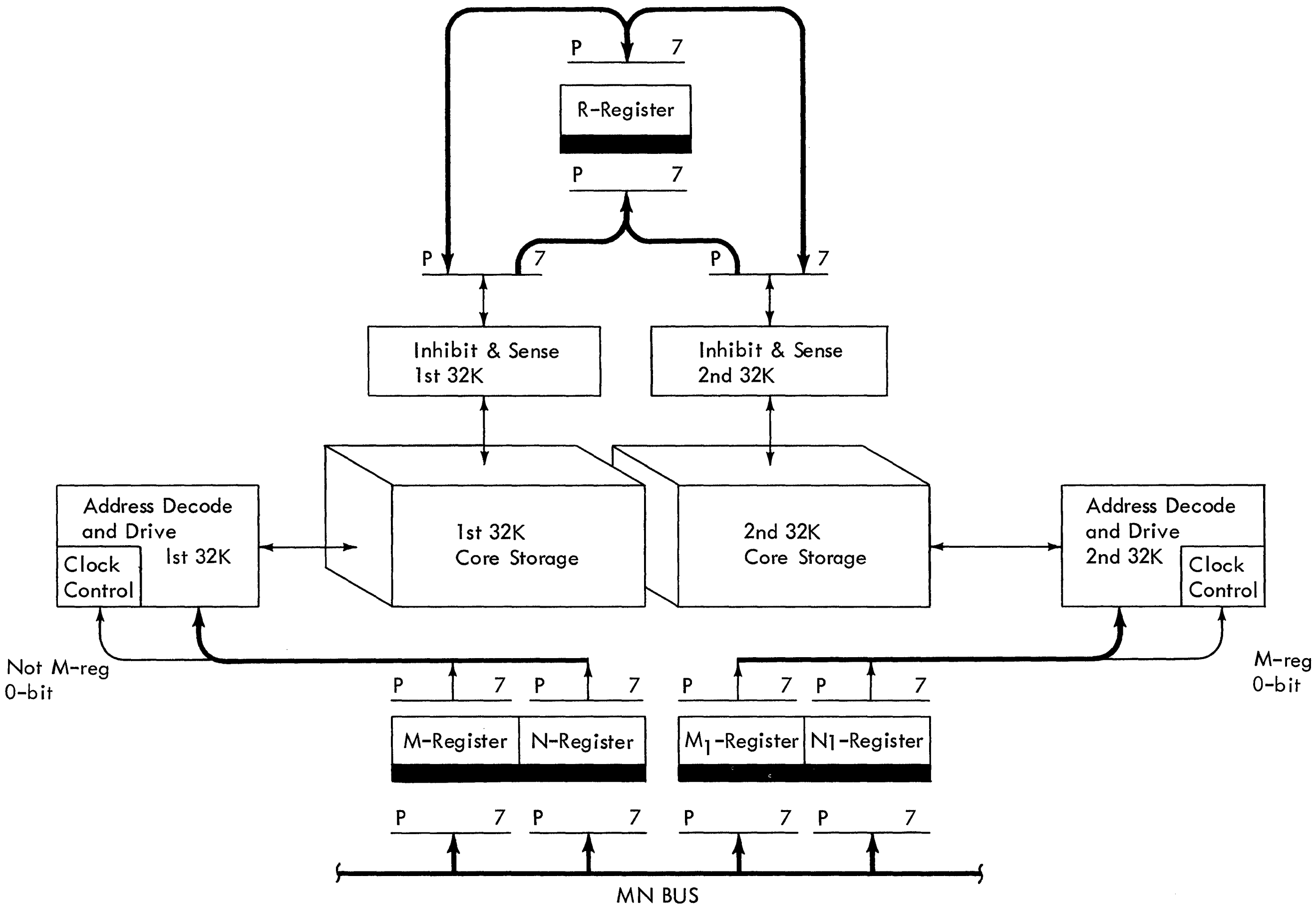


Figure 2-59. Clock Control Addressing

When the CPU requires information from the 65K storage unit, a 2-byte address is placed into both MN-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 00000-32767, 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.

MN Register Bit Lines

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 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 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
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 7 Bit
Store 6 Bit
Store 5 Bit
Store 4 Bit
Store 3 Bit
Store 2 Bit
Store 1 Bit
Store 0 Bit

Memory Sense Bit-Lines

These nine lines represent the core-storage data output. They are active at memory strobe time, which is 525-800 nanoseconds of the memory read cycle. If the data on the memory sense bit-lines is to be used by the CPU, the memory clock data ready pulse (750-900) 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 Parity 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

Early M0

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 MN-register set pulse is at T1 time, and it takes approximately 50 nanoseconds to set the MN-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 early-M0 pulse occurs <u>before</u> the M-register has set, and is actually before zero time in the CPU clock cycle. If the early-M0 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 early-M0 is not present when the CPU read-call signal arrives, the first 32K clock is started.

Early-M0 is not brought up for a write cycle because the MN-registers are not changed for a memory write cycle. For a write cycle, the M-register 0-bit line 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 early-M0: 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 clock 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 2-55).

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 2-55).

Data Ready

Data-ready is the memory data strobe pulse to the CPU. At 750-900 nanoseconds of the memory clock read cycle, the data- ready signal sets the data or the memory sense bit lines into the R-register, providing the CPU has specified memory as the source for the R-register.

Machine Reset Switch

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 selected by MN-register bits.
- Clock selected by high-order bit position of MN-register, and started by read-call from CPU.
- Resultant data byte placed into CPU R-register.

To read out a byte of data, the core-storage unit must interpret the bits in the CPU MN-register 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 MN-register (Figure 2-60). To follow a storage read example through the core-storage circuitry, assume the address 0100110110001011 is in the MN-register.

	M-REGISTER								N-REGISTER							
Position	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
Binary Value	32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	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
Decimal Equivalent	19,851															

Figure 2-60. MN Decode

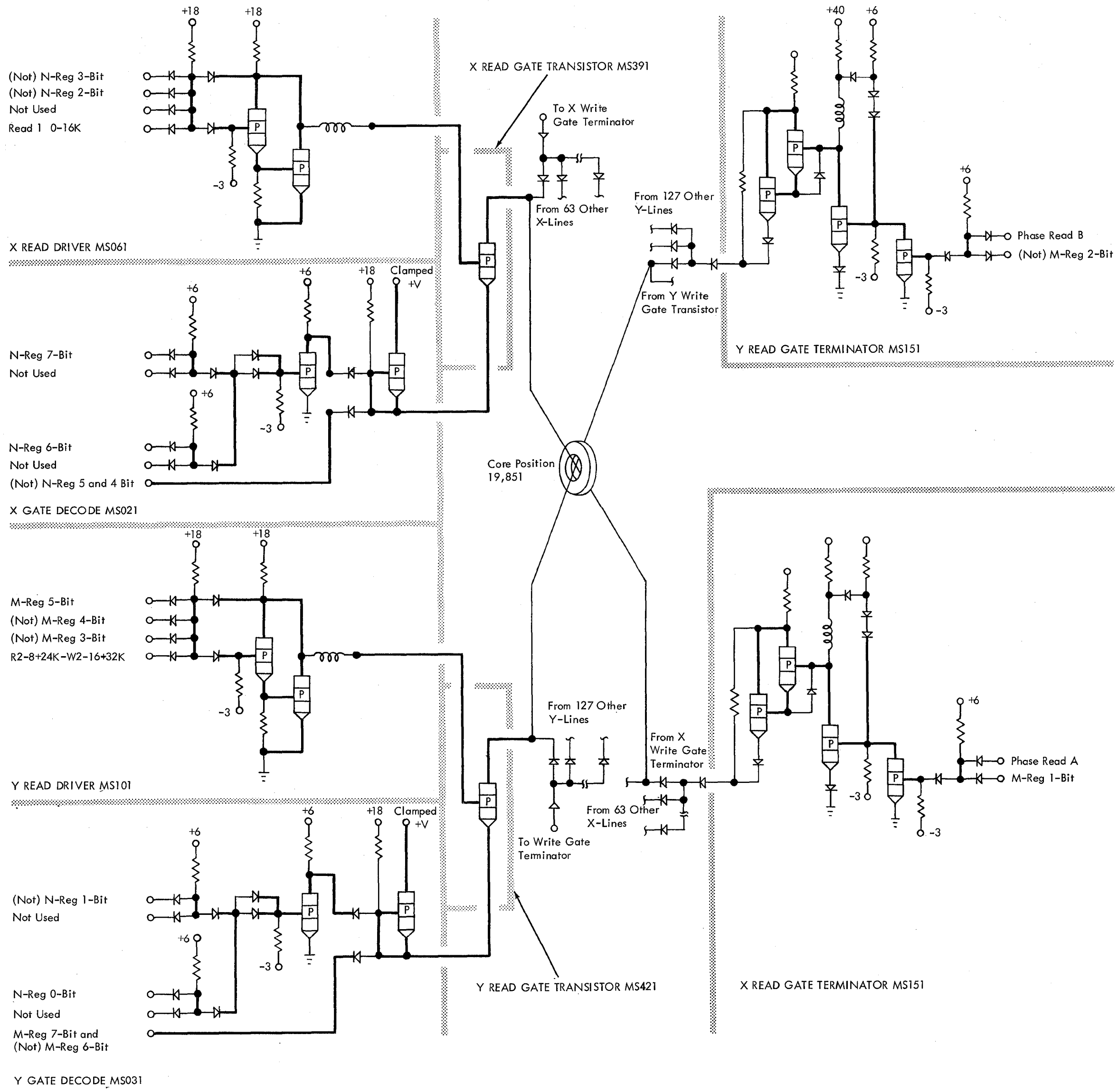


Figure 2-61. Storage Read Example

Circuit Objectives (Figure 2-61)

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
     (Not) Local St or MPX
     Read Call

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 (switched with
       Use Main Mem)

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
    Read 2 (from clock)
    Use Main Mem

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 emitter of read gate transistor)
    R-8+24K-W-16+32K 3072 - 4095 (conditions base 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)
    M-Reg Not 2-Bit controlled

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 8K block and noise from the other 8K blocks, a strobe pulse is developed for each 8K block. Because the M-register 2- and 1-bit positions determine which 8K block is addressed, these bit positions gate the memory-clock strobe pulse to the addressed 8K block. In the example given, there is an M-register 1-bit and no M-register 2-bit, indicating the desired byte is in the third 8K block. The output of the third 8K block sense amplifiers 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 (MS181).
    Strobe (from clock)
    (Not) M-Reg 2-Bit Controlled
    (not) Not M-Reg 1-Bit Controlled

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

c. SA 6-Bit 16-24K B. This is the second half (4,096 positions) of the sense circuitry for the third 8K block (MS241).
    SA - Inh Line 6-Bit B1
    SA - Inh Line 6-Bit B2
    Strobe 6-Bit 16-24K

d. Mem Sense 6-Bit 16-32K. (Final amplifier output MS281).
   SA 6-Bit 16-24K A
   - or -
   SA 6-Bit 16-24K B

STORAGE WRITE EXAMPLE

- X- and Y- drive lines selected by MN-register bits.
- Clock selected by high-order bit position of M-register and started by write call from CPU.
- Data from R-register activates appropriate store circuits, 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 MN-register does not change between read and write cycles. Therefore, we can assume the same address (19,851) is in the MN-register 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.

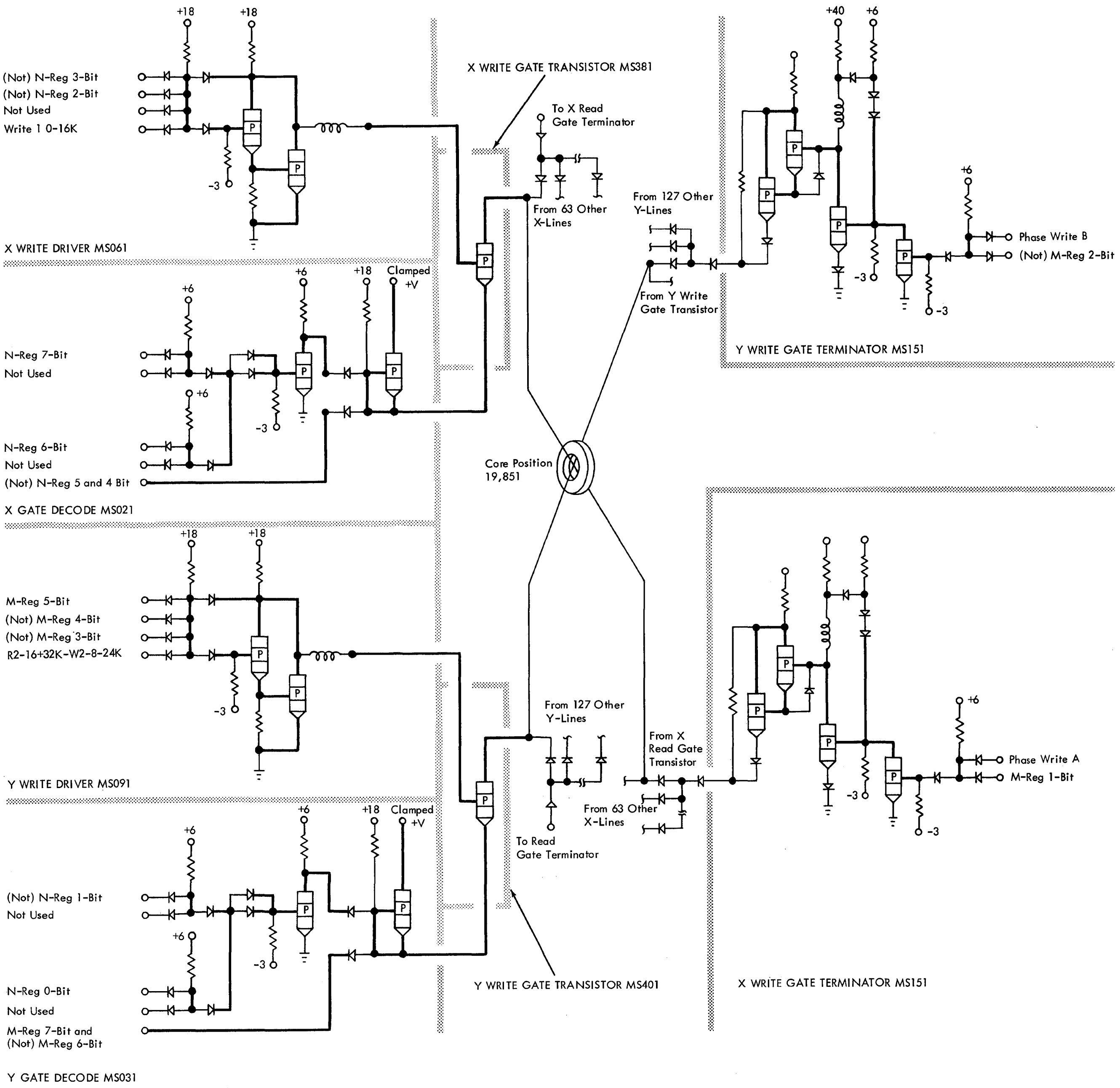


Figure 2-62. Storage Write Example

Circuit Objectives (Figure 2-62)

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 (MS 161).
      Write 1 (from clock)
      M-Reg 1-bit (switched with
        Use Main Mem)

   b. 11 Gate TX 16-32K. This is the X Gate decode (MS021).
      N-Reg 6-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 (MS 151).
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 <u>conditions</u> the proper Y-drivers as required by the phase reversal addressing scheme (MS161).
(not) M-Reg 2-Bit
Write 2 (from clock)
Use Main Mem

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 emitter of write gate transistor)
R-16+32K-W-8+24K-3072-4095 (conditions base 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)
M-Reg Not 2-Bit controlled

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 2-63). 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 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.

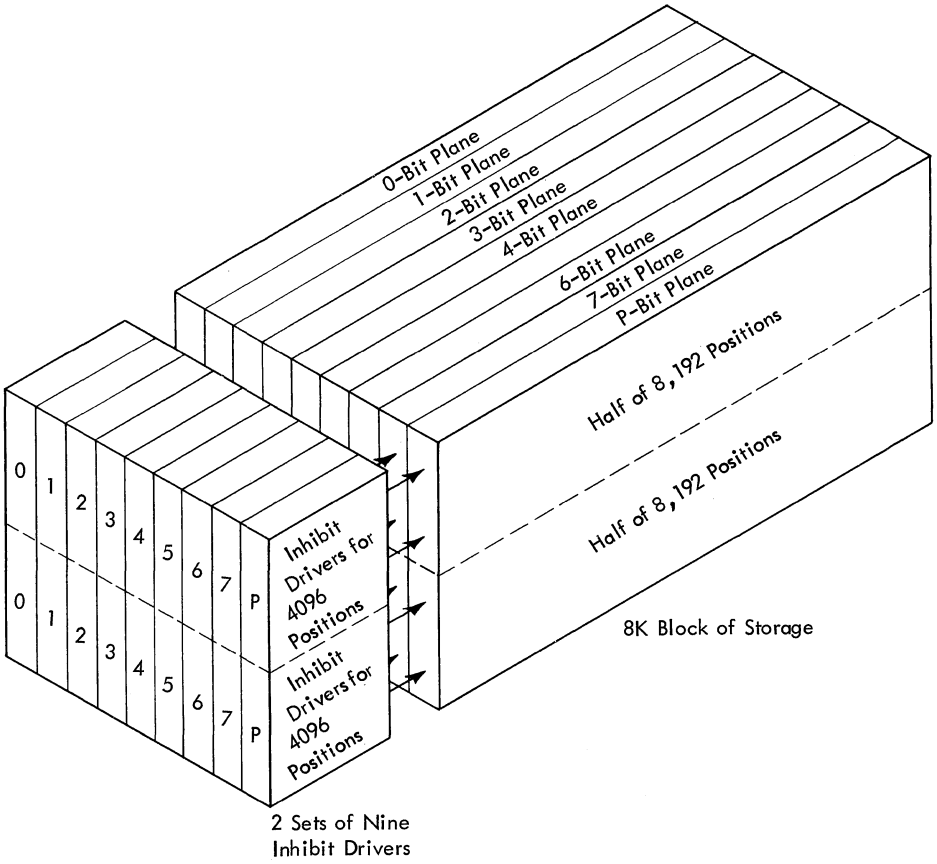


Figure 2-63. Inhibit Driver Control

To effect inhibit driver selection, the M-register 2- and 1-bits define in which 8K block inhibit current is to flow. Then the N-register 2-bit further selects either the nine low-order-half inhibit drivers or the nine high-order-half inhibit drivers.

a. 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).
   M-Reg Not 2-Bit Controlled
   M-Reg 1-Bit Controlled
   (not) N-Reg 2-Bit
   Inhibit (from storage clock)

b. SA-Inh
   SA-Inh Line 6-Bit A1. 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 <u>no</u> 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
   (not) Store 6-Bit

QUESTIONS ON CORE STORAGE

1. How many wires pass through each core? List them.

2. Addressing of a location in core

storage consists of selecting the _____ and _____ lines from the address in _____.

3. An X-line is conditioned by what bits in the address?

4. A Y-line is conditioned by what bits in the address?

5. What must be accomplished to read out a core location?

6. A 32K storage array has _____ set (s) of X_____lines and _____set (s) of Y lines, and_____phase reversal planes.

7. Data coming from storage is read into the _____ register.

8. When is current induced in the inhibit-sense line during read time?

9. Noise is eliminated from the data pulse during reading through use of a _____ pulse.

10. To write a bit in core, inhibit current must_ (flow, not flow).

11. What is the function of the dummy planes placed between each 8K of the 32K storage array?

12. The purpose of the early-M0 pulse is to allow correct _____ selection before the storage cycle starts.

13. To select the upper 32K of a 64K storage array, the _____ bit of the address must be _____ .

14. How many gate decode switches are needed to control X-Line selection in a 16K array?

15. How many gate decode switches are needed to control Y-Line selection in a 16K array?

16. A read gate transistor must have both the _____ and _____ conditioned to conduct.

17. What is the purpose of an inhibit driver?

18. Where can a CE find troubleshooting aids for fixing core-storage troubles?

19. Read 1 conditions the _____ lines.
Read 2 conditions the _____ lines.

MEMORY CONTROL

MN-REGISTER

The main MN-register function is to supply a two-byte address to the core storage unit. The MN-register consists of two separate 9-bit registers (8 bits plus parity bit). The parallel output of these registers is fed to the core storage addressing circuitry over the CPU to memory interface. There is no gate at the output of the MN-register. Whatever is stored in the MN-register is present at the input of the memory addressing circuitry. Therefore, to control the address presented to the core storage unit, the CPU controls the address in MN by controlling the data at the input of MN, and by controlling the time when MN-data can be changed.

The input to the MN-register is the 18-bit wide MN-register assembler bus. Feeding this bus are seven possible sources for MN data. These are:

IJ-registers
UV-registers
LT-registers
GUV-registers
HUV-registers
Console switches A, B, C, and D
Next-address information from the read-only storage unit.

When the CPU wishes to change an address in MN, one of the previously mentioned sources is gated onto the MN assembler bus. The MN-registers are polarity hold latches. To change the information in these latches, the CPU must activate the control line during the time when the MN assembler bus is active. If the CPU clock is running, the control line is activated at T1-time of a read cycle. The control line is not activated during a write or store cycle. This is because core storage read-out is destructive: all cores must be set to logical zero to read out information. Thus, if we were to read out a position and change the MN address before writing something back, the position read out would have all cores set to zero, and an R-register parity error would occur the next time that position is read out. If the CPU clock is not running, the MN-register can be set from switches A, B, C, and D on the operator's console.

If the 2030 is equipped with the additional 32K core storage unit, the CPU has an additional MN-register connected to the MN assembler bus. This additional MN-register supplies address information to the additional 32K storage unit (see Figure 2-59).

The control line to set the additional MN-register polarity hold latches is developed the same way as the control for the standard MN-register polarity hold latches. Thus, the additional MN-register is set only at T1 time of a read cycle, and is not set during a write or store cycle. The second MN-register can also be set during a manual store or display operation using console switches A, B, C, and D.

There is an exclusive OR parity check circuit tied to each 9-bit register in the MN-register scheme. Thus, a total of four exclusive OR parity check circuits assure correct parity of all addresses presented to the core storage addressing circuitry. An even parity condition in any of the four check circuits activates the MN-register check line on the MC-register input bus. This line then sets a latch in the MC-register. The parity of the M-register is not guaranteed when working with a local storage address. Therefore, if local storage is being addressed, the MN-register check is blocked from getting on to the MC input bus.

MEMORY WRAP

Binary storage addresses are presented to the core storage unit over the memory/CPU interface. It is possible for the CPU to generate a binary storage address that is larger than the address range of the core storage unit on that particular machine. For example, if the address 11111111 11111111 is contained in the MN-register, an 8K storage unit would interpret this as the address 00011111 11111111, because to the 8K storage unit, the largest address is 00011111 11111111. Thus, the storage unit has no way of identifying an address as being outside its address range. Therefore, the CPU must check each address placed into the MN-register to insure that a legitimate address is being used. The condition where the address constructed in MN is outside the address range of the storage unit is called memory wrap. Sensing this condition simply involves checking the M-register 0-, 1-, and 2-bits. For an 8K storage unit, any bit in the M-register 0-, 1-, and 2-bit positions causes a memory wrap. For a 16K storage unit, the M-register 0-, and 1-bit positions are checked. A 32K storage unit requires an additional bit for addressing so only the M-register 0-bit position need be considered. For a 65K storage unit, all possible bit configurations in the M-register are legitimate, and therefore this type of wrap is not checked for a 65K storage unit. These variations are accounted for by tying down different M-register bit-lines at the input of the memory wrap detection circuitry (logic page MW031).

During address arithmetic, the CPU increases or decreases the value of the storage address as required by the program. For example, it may be necessary to read out a multiple-byte number starting at the low order position of that number. For each byte that must be read out, the CPU must increase the address value by a value of 1. If increasing the address value by 1 causes the address to go to zero, a memory wrap has occurred. Thus, if a 65K storage unit address is increased from 11111111 11111111 to 00000000 00000000, then an address wrap has occurred even though the new address (00000000 00000000) is legitimate for this size storage unit. Similarly, if an 8K storage address is modified from 00011111 11111111 to 00100000 00000000, a memory wrap occurs because this address is outside the range of addresses of that particular storage unit. This type of wrap is detected by the fact that there is an M-register 2-bit.

To detect the wrap that occurs when a 65K address goes from all one's to all zero's, the CPU looks at the M-register source data to see if address arithmetic has changed the M-register data from all one's to all zero's. M-register source data could come from the I, U, GU, or HU registers. The CPU need only follow the arithmetic used with the contents of these registers to see if a wrap occurs.

A memory wrap sets the memory wrap request latch. This error is stacked as part of the machine error priority stack system. This is explained in more detail under Machine Check Handling.

ALLOW READ, ALLOW WRITE

Earlier, when we discussed the MN-register, we discussed the importance of not changing the MN-register between read and write cycles. A latch in the CPU provides the necessary interlock to prevent changing the MN-register and to prevent taking successive memory read cycles with no intervening write cycles. The allow write latch is turned on with a CPU read cycle and turned off with a CPU write cycle to provide this interlock. The output of this allow write latch controls the set of the MN-register as well as the read call and write call signals that define core storage cycles.

R-REGISTER

The R-register acts as a single-byte buffer for the transfer of information between core storage and the CPU. It is both the source and data register for the core storage unit. Information to be placed into the core storage unit must first be gated through the ALU to the R-register. Once in the R-register, this byte is available to the core storage inhibit drivers via the memory/CPU interface. Likewise, information read out of storage to be used by the CPU must first be gated to the R-register. From the R-register the information can be gated to one of a number of registers via the A-bus and the B-bus.

There are two levels of control for placing data into the R-register. First, the desired R-register source is selected and gated to the R-register polarity hold latch data inputs. This source can be either Z-bus data (CPU) or storage data (core storage). The second control is the R-register polarity hold latch control line. It is here that the R-register set timing is determined. If the data source is core storage, the R-register is set by data ready pulse from core storage. This occurs at approximately T4-time. If the R-register source is the Z-bus, the R-register is set with the T4 pulse.

COMPREHENSIVE INTRODUCTION

- 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.

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 2-64). If the 2030 requires the full 65K of storage, two separate M2-I units are installed in the 2030.

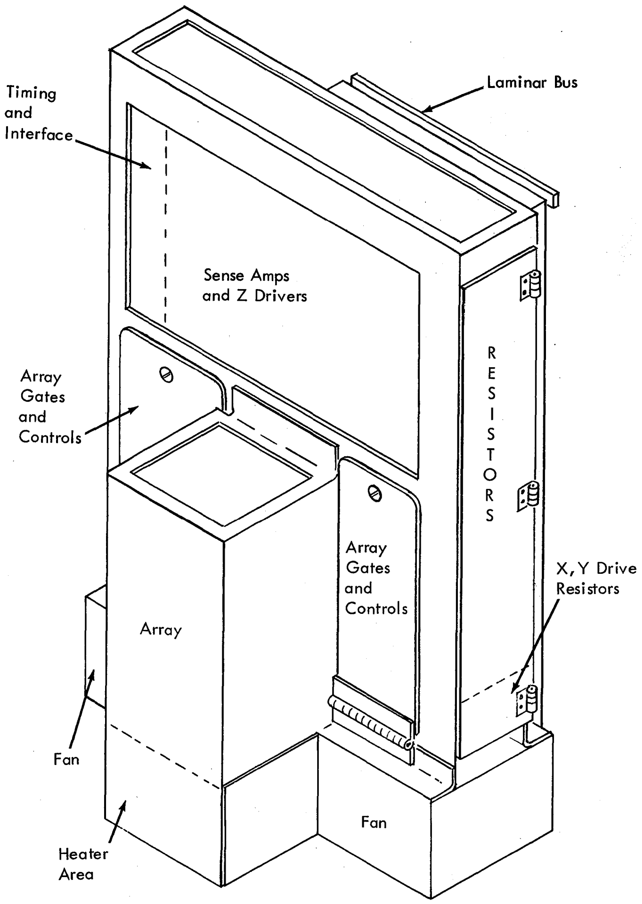


Figure 2-64. M2-I Storage Unit (8K, 16K, 32K)

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 2-65).

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 a different byte may be placed into that storage position. The byte that is to be placed into storage 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.

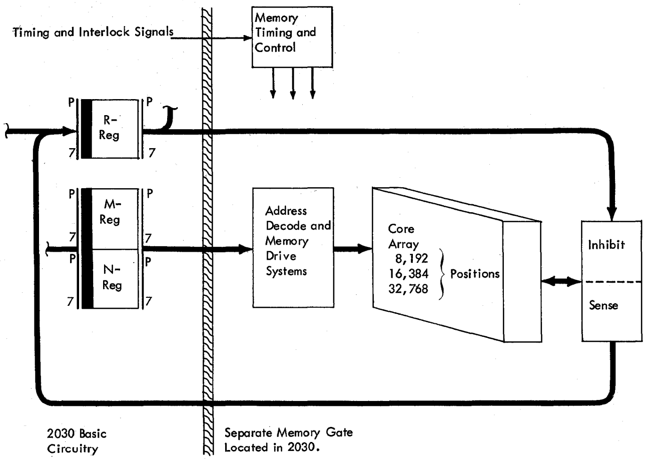

Timing and Interlock Signals
Memory Timing and Control
P
R-Reg
7
P
7
P
M-Reg
7
P
7
P
N-Reg
7
P
7
Address Decode and Memory Drive Systems
Core Array
8,192
16,384
32,768
Positions
Inhibit
Sense
2030 Basic Circuitry
Separate Memory Gate Located in 2030.

Figure 2-65. Memory-to-CPU Data Flow

FUNCTIONAL UNITS

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 combination 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 2-66). The horizontal drive lines are called X-drive lines, whereas the vertical drive lines are called Y-drive lines. The M2-I uses a combination 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.

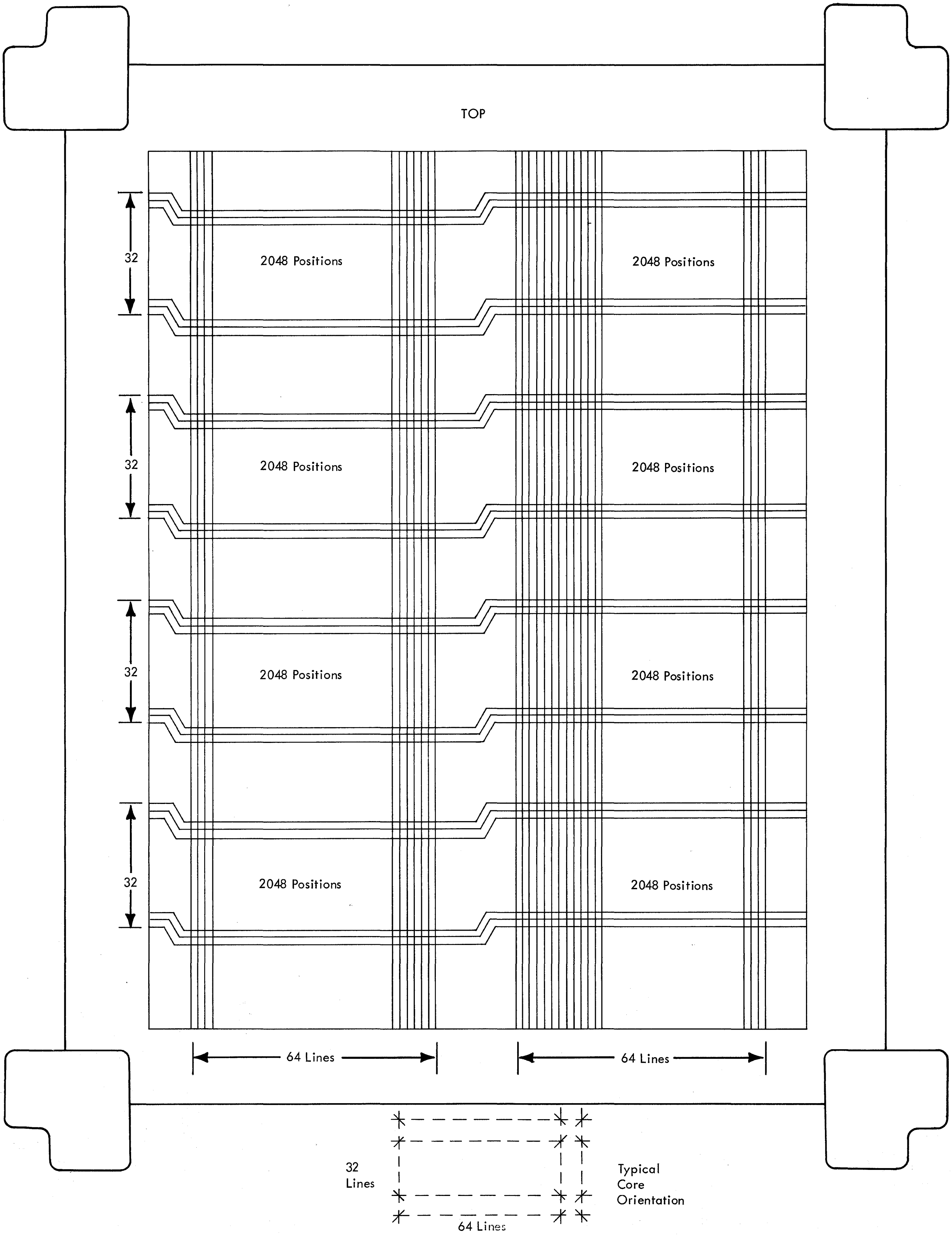


Figure 2-66. Core Plane Layout

The 8K core array is composed of five 16,384-core planes (Figure 2-67). 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 wires that carry X-drive current from one half of the X-winding to the other half of the X-winding. The winding pattern of the cores is such that alternate X-drive lines start at opposite ends of the core plane (see Figures 2-66 and 2-67 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 2-67). 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 switched to read or switched to write.

An 8K storage unit, such as shown in Figure 2-67, has eighteen sense/inhibit windings (two per bit plane). There are two windings per bit plane. Each winding serves 4,096 cores. The sense/inhibit winding is wound parallel to the X-winding.

The 16K core array consists of nine 16,384-core planes. The first plane forms bit-0 in both first and second 8K units (Figure 2-68). The second core plane forms bit-1 in both first and second 8K units. This same scheme continues through the ninth core plane which forms bit-P for both 8K units. 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.

The Y-winding goes through both 8K bit-planes 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-windings are in-phase in only one 8K unit at a time.

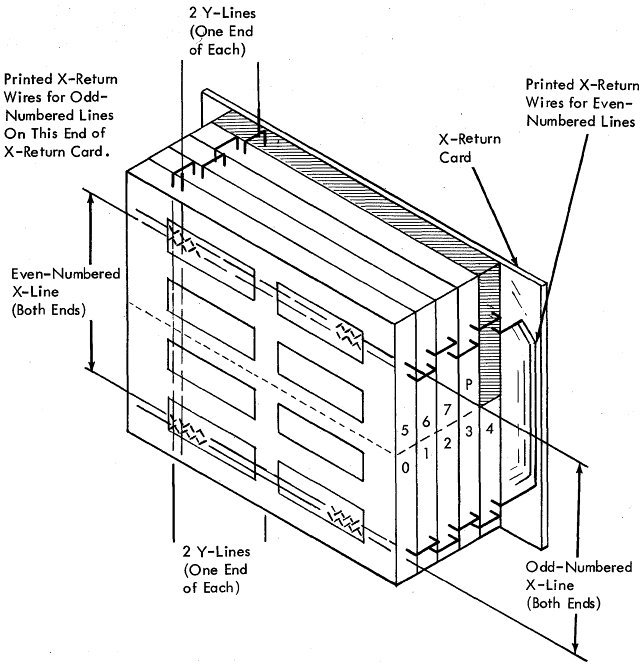

2 Y-Lines (One End of Each)
Printed X-Return Wires for Odd-Numbered Lines On This End of X-Return Card.
Printed X-Return Wires for Even-Numbered Lines
X-Return Card
Even-Numbered X-Line (Both Ends)
P
5
6
7
0
1
2
3
4
2 Y-Lines (One End of Each)
Odd-Numbered X-Line (Both Ends)

Figure 2-67. 8K Storage Winding

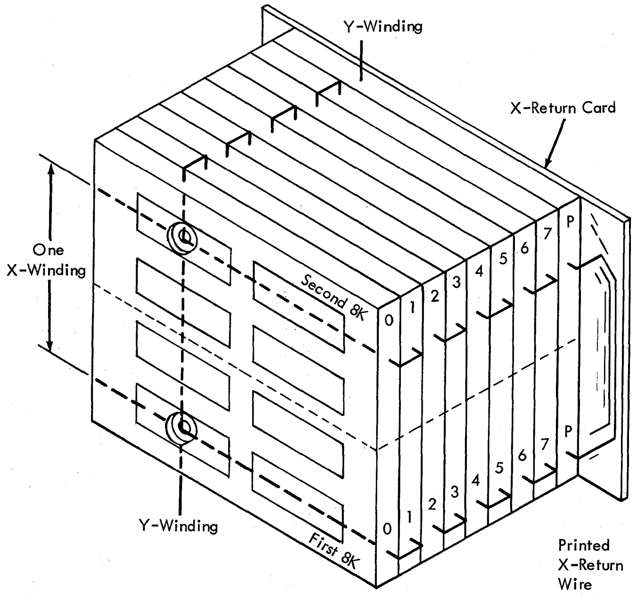


Figure 2-68. 16K Storage Winding

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 2-69). 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-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.

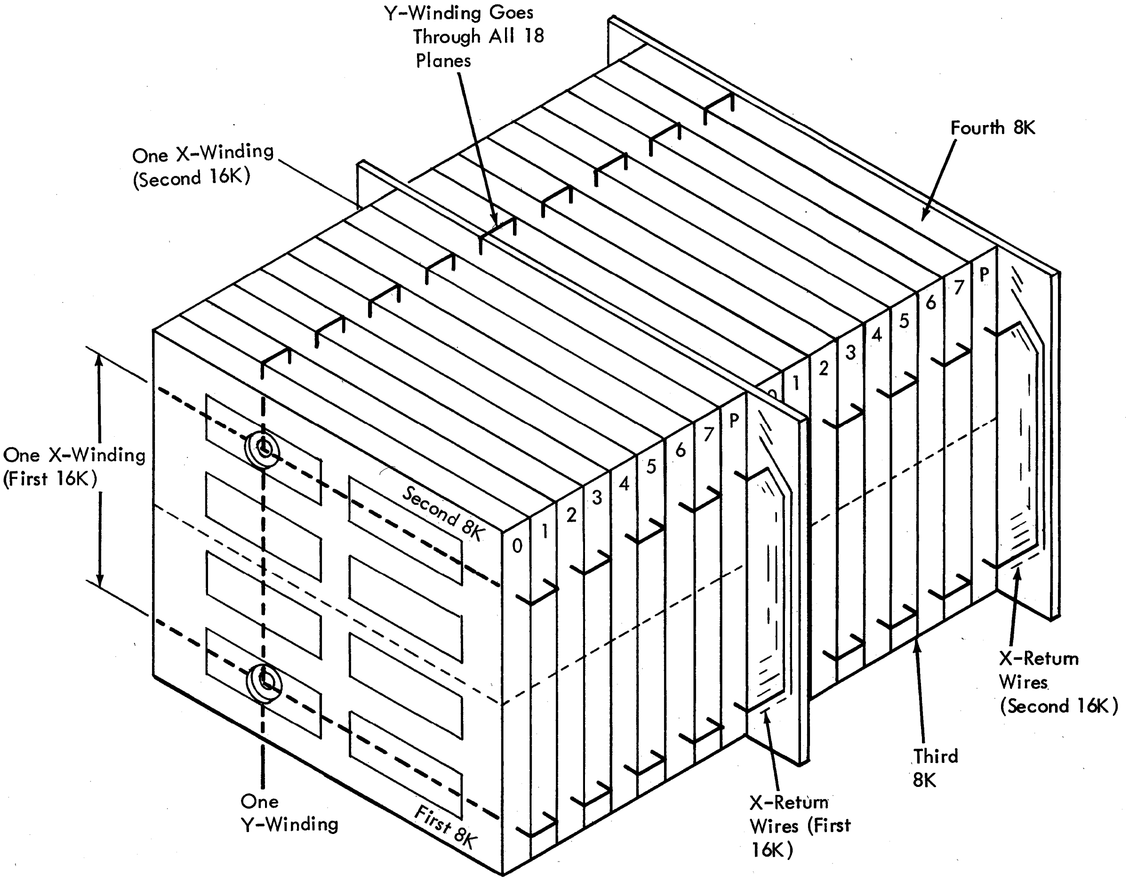

Y-Winding Goes Through All 18 Planes
One X-Winding (Second 16K)
Fourth 8K
One X-Winding (First 16K)
Second 8K
0
1
2
3
4
5
6
7
P
0
1
2
3
4
5
6
7
P
First 8K
One Y-Winding
X-Return Wires (First 16K)
Third 8K
X-Return Wires (Second 16K)

Figure 2-69. 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 2-69). The units are separate physically, and each mounts on a separate hinged gate in the 2030 (Figure 2-70).

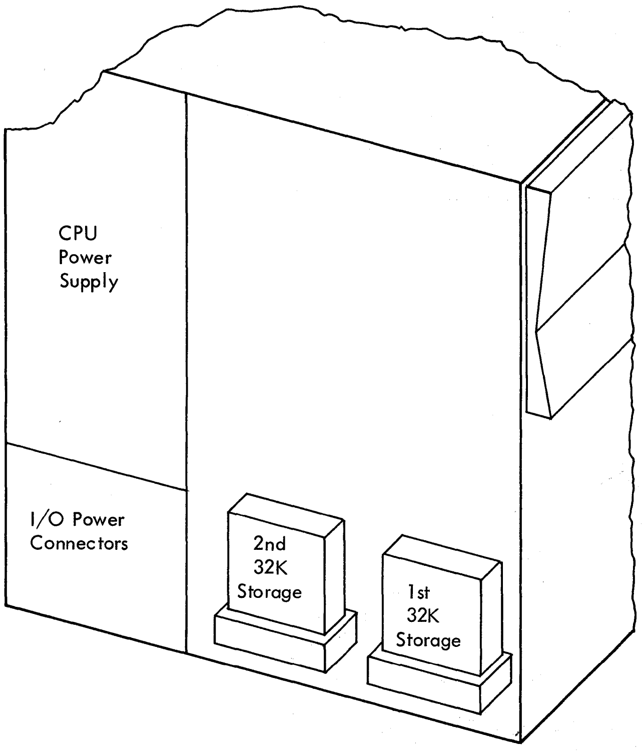


Figure 2-70. IBM 2030 Showing Two 32K Storage Units

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.

The M2-I core storage unit contains a timing generator referred to as the storage clock. Having a clock separate 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 2-71).

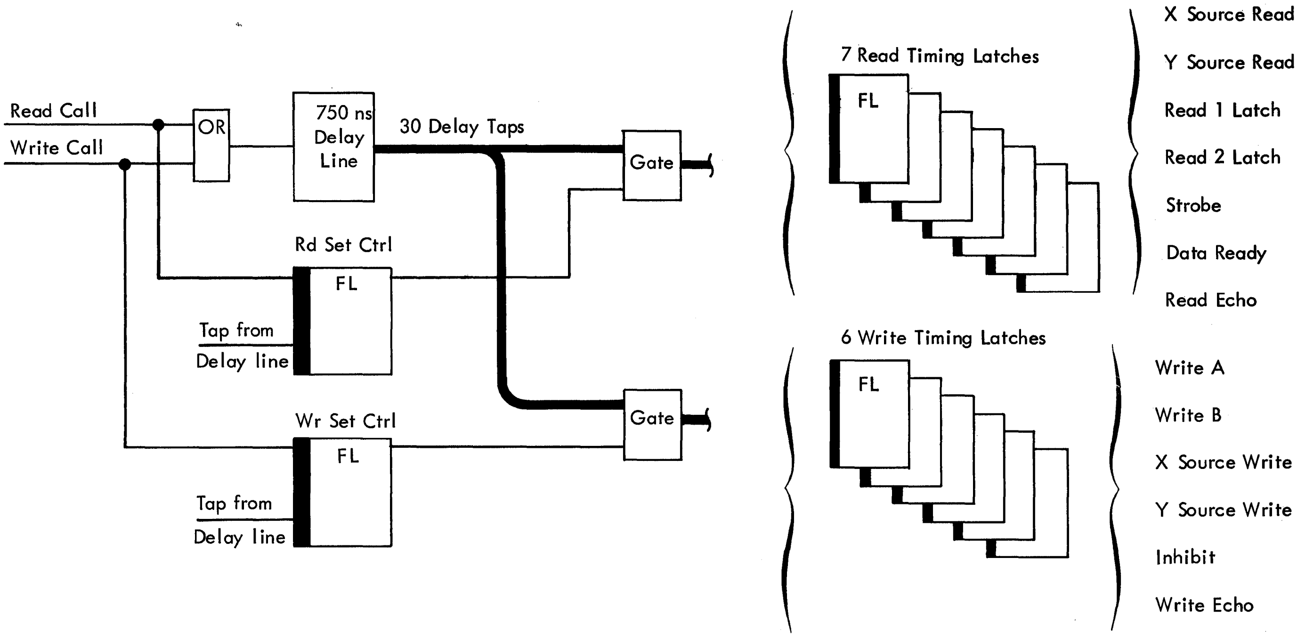


Figure 2-71. Simplified Clock Logic

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 (Figure 2-72). A read cycle is completed in 750 nanoseconds--the same time as one basic 2030 machine cycle.

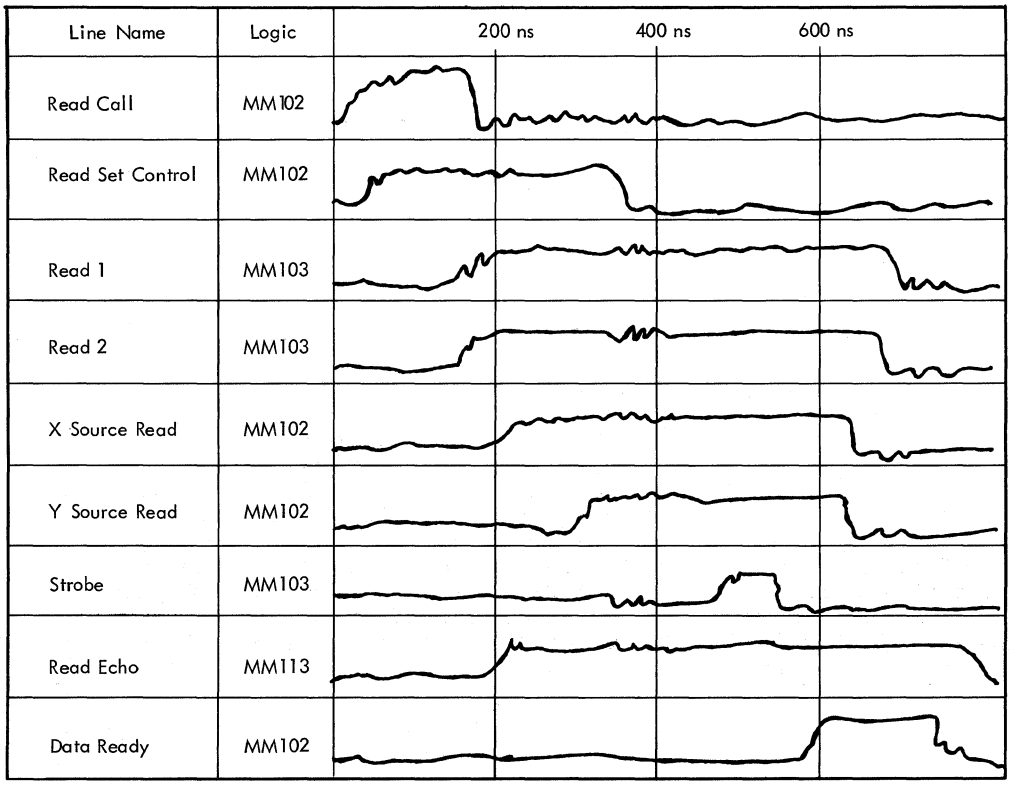


Figure 2-72. Core Storage Read Timings

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 2-73). A write cycle is completed in 750 nanoseconds--the same time required for one 2030 clock cycle.

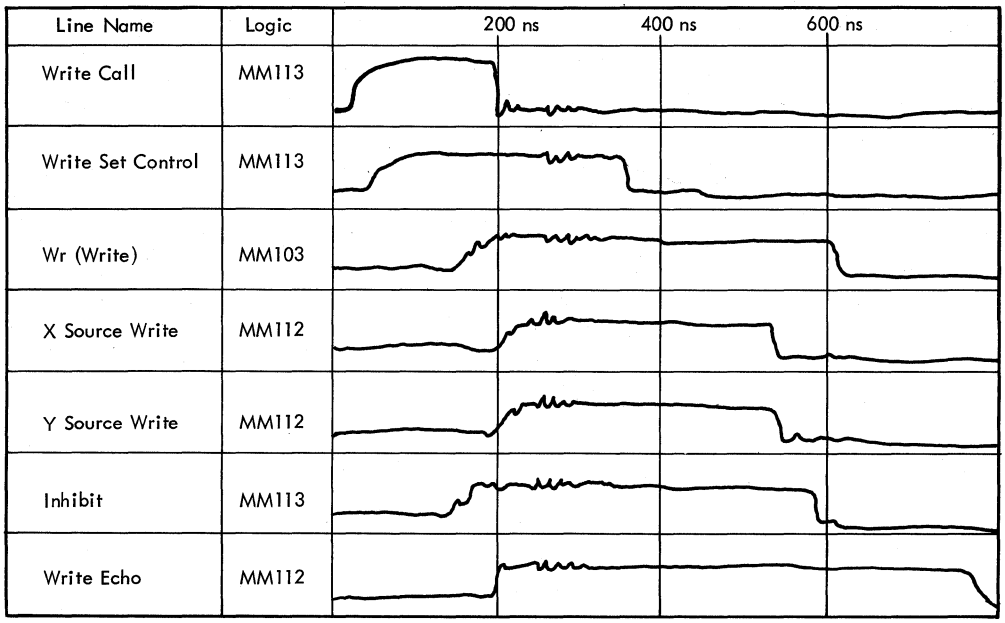


Figure 2-73. Core Storage Write Timings

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.

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 2-74). When the clock signals 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.

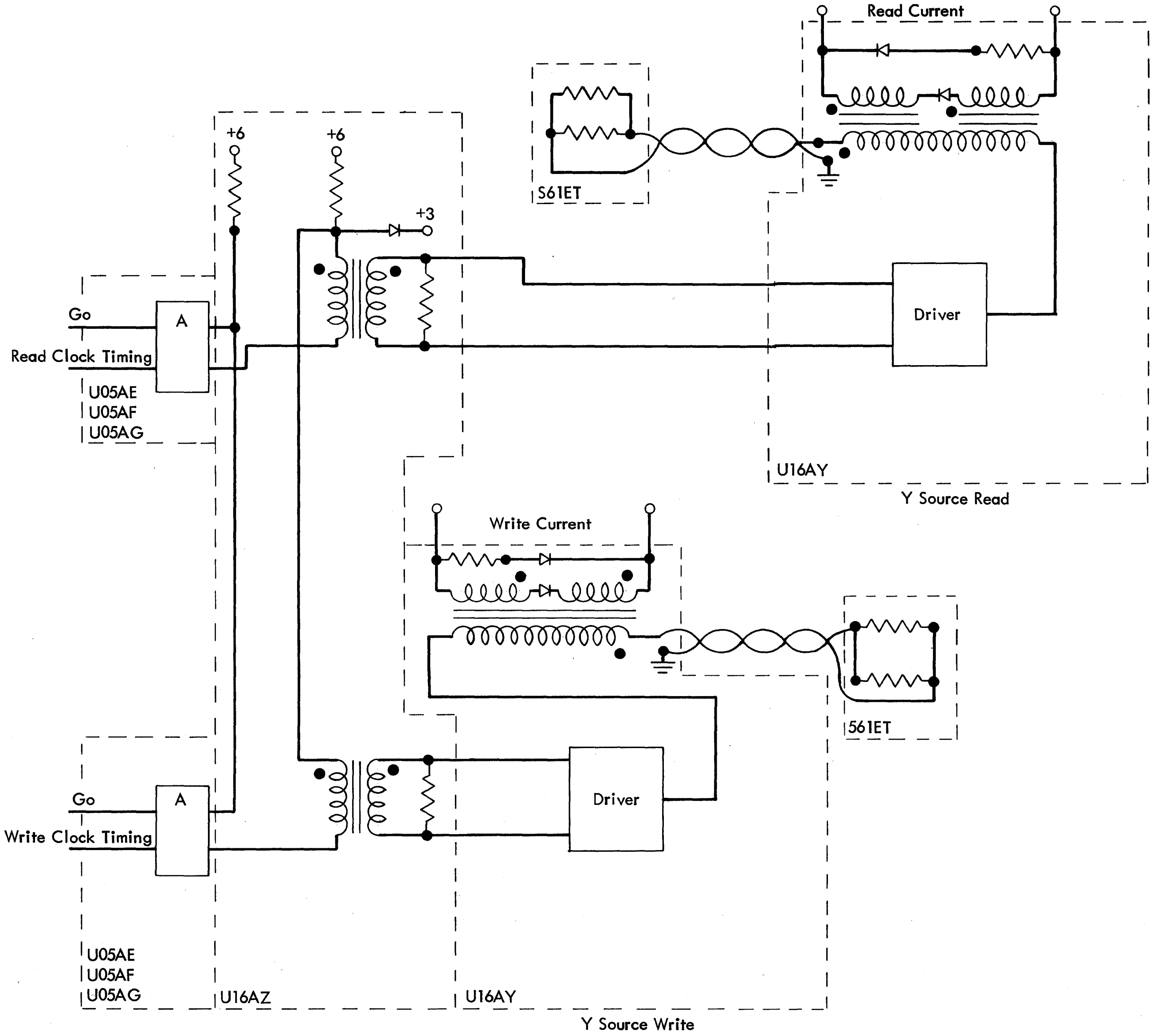


Figure 2-74. Current Sources

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 2-75). 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 (US3AD), and gates (SI5ES, SI5ET). In Figure 2-76, 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.

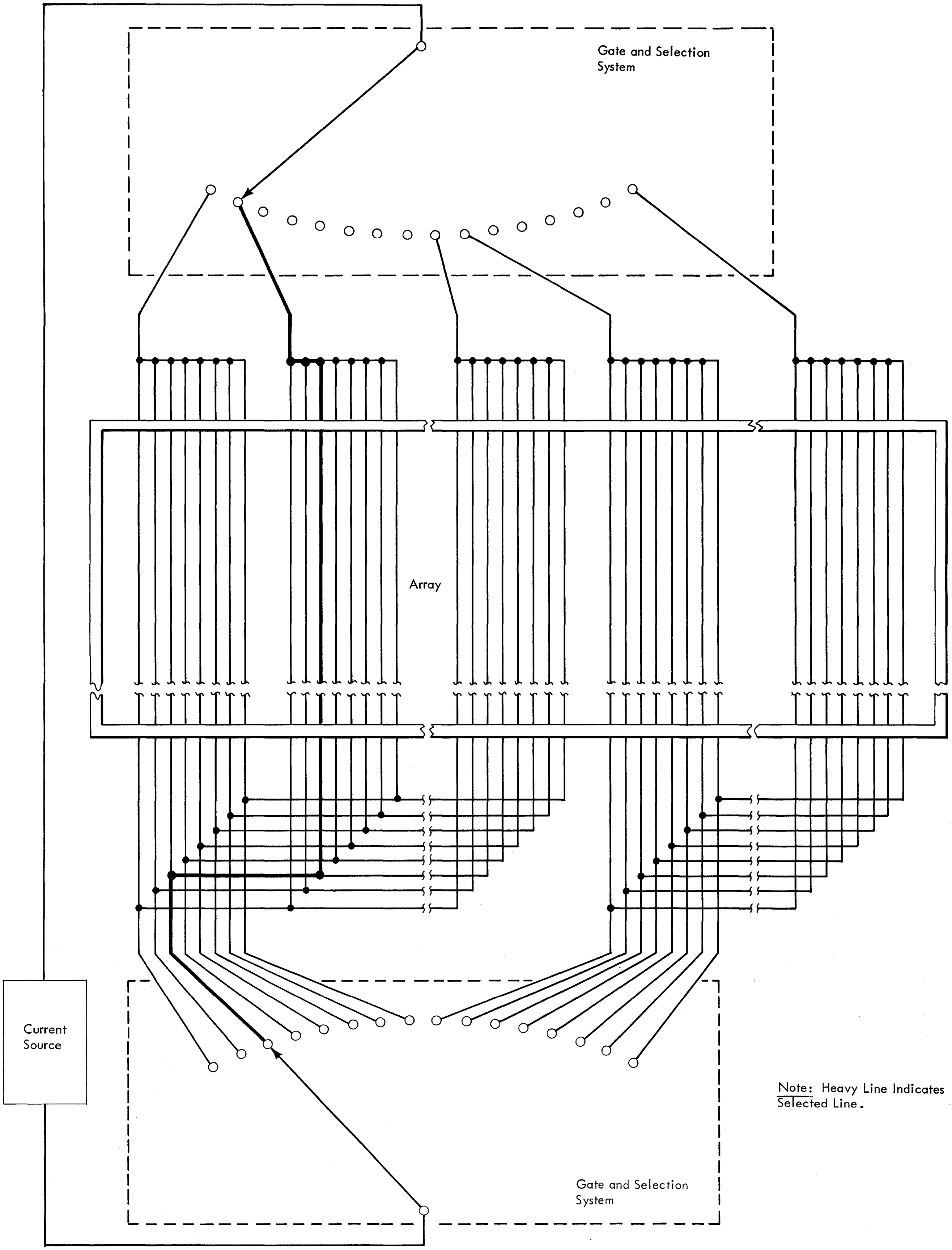

Gate and Selection System
Array
Current Source
Note: Heavy Line Indicates Selected Line.
Gate and Selection System

Figure 2-75. Gate and Selection System

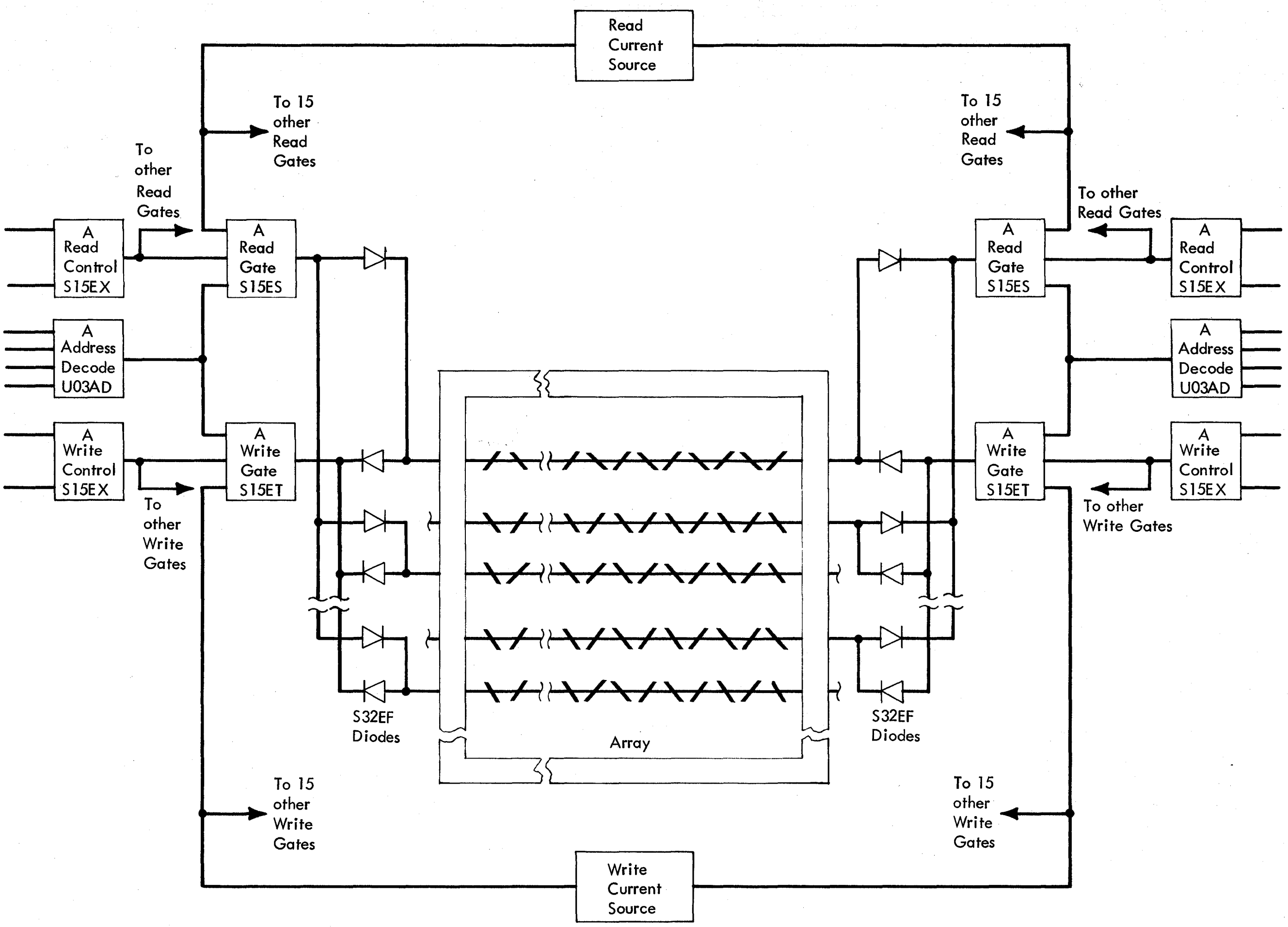


Figure 2-76. Gate and Selection Logic

SENSE/INHIBIT SYSTEM

- A combination sense/inhibit winding is used.
- Each sense/inhibit winding goes through 4096 cores, parallel to the X-drive lines.
- During a read cycle, the sense/inhibit winding senses pulses caused by cores that flip.
- During a write cycle, the sense/inhibit blocks cores from flipping.

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 4096 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 2-77). 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 reach sense amplifiers. 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 2-77). 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-winding. With no bit present at the inhibit drive 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.

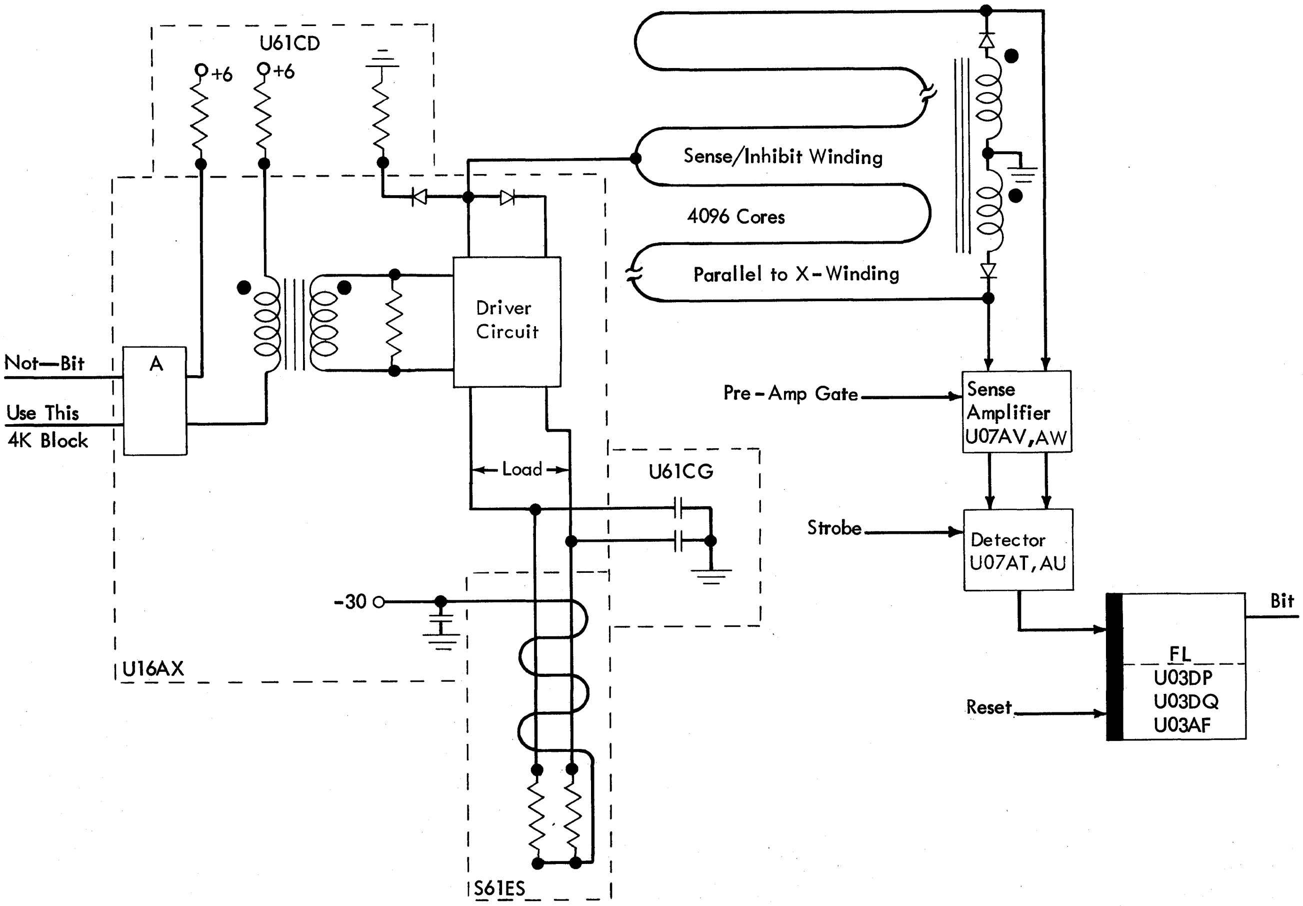


Figure 2-77. Sense and Inhibit Logic

POWER SUPPLY AND TEMPERATURE COMPENSATION

- Four power supply voltages are required for operation of the M2-I: +6, +3, -3, -30.
- The 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

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° 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 storage bump.
- 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 2-78). 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 intersect with the top 64 lines to form 256 bit-positions for CPU local storage, and 256 bit-positions for multiplexor storage. In the 8K unit, these top positions would correspond to the 5-, 6-, 7-, or P-bit positions (see figure 2-67). The eight auxiliary storage windings intersect with the bottom 64 lines to form 512-bit positions for lower bits. This would correspond to bits 0, 1, 2, 3, or 4.

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 1024 additional byte positions of auxiliary storage. The auxiliary storage areas are labled MPX 0, MPX 1, MPX 2, and CPU 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 CPU local storage, and MPX 0, MPX 1, MPX 2, MPX 3, MPX 4, MPX 5, and MPX 6.

Expansion beyond 32K does not yield additional auxiliary storage. Therefore, auxiliary storage is always located in the first M2-I.

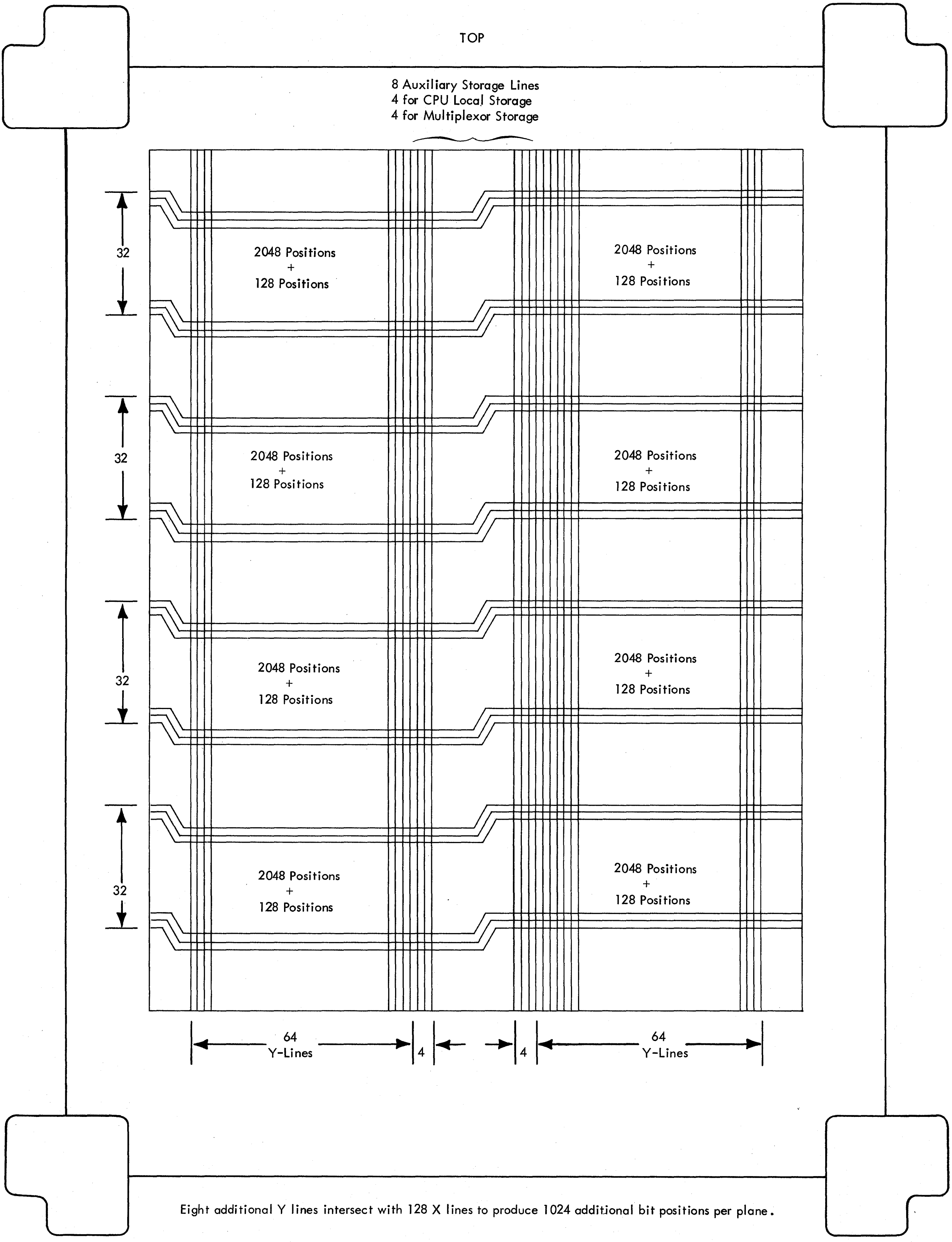


Figure 2-78. Auxiliary Storage Core Plane Windings

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 planes where the core is to be left at logical 0.

Description (Figure 2-79)

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 flip 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).

The write call signal from the 2030 starts the storage clock and conditions a write cycle. The M- and 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 planes where the core is to remain logical 0, and inhibit current is blocked in the bit-planes 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 turn ing 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.

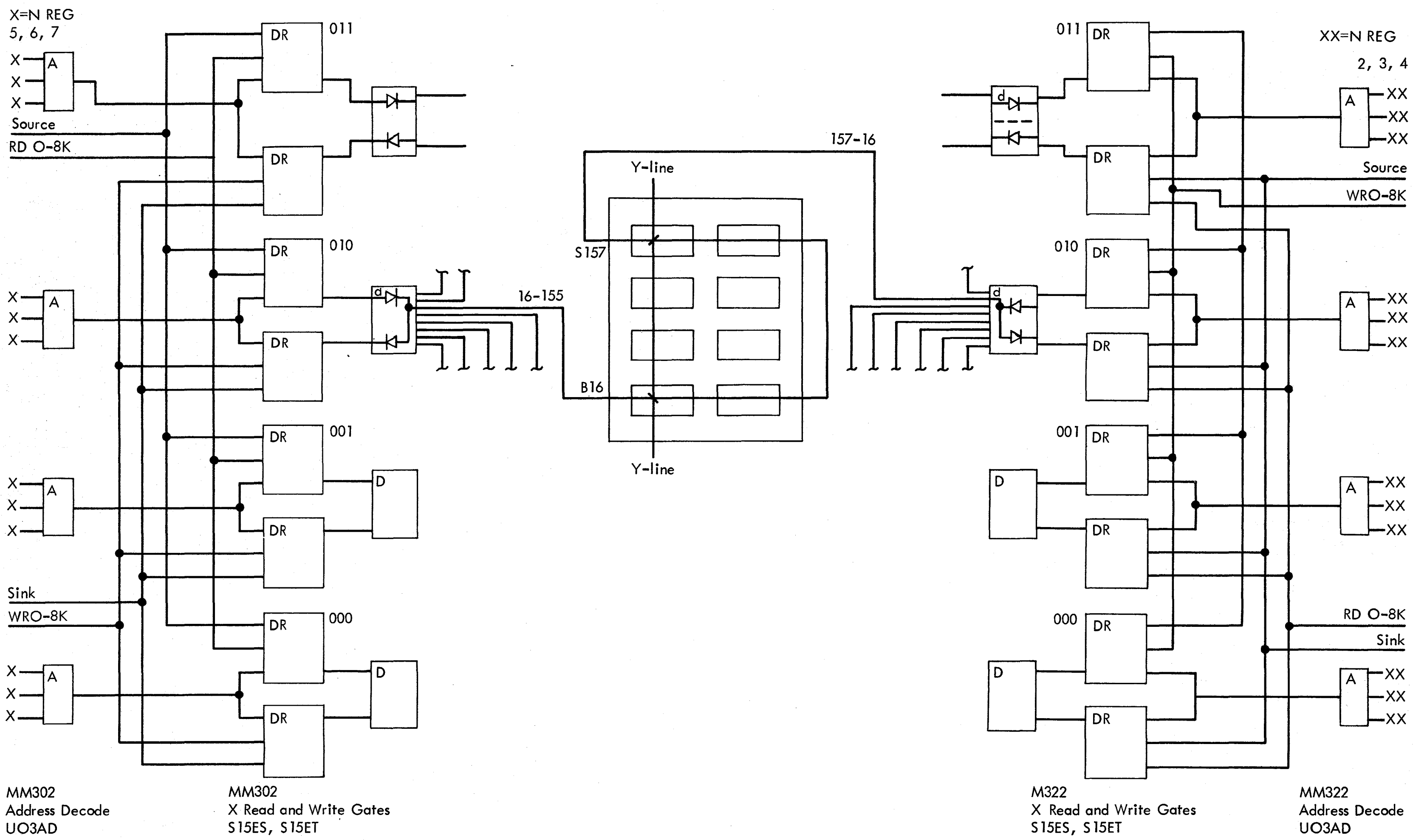


Figure 2-79. 8K Storage Operations

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 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

4. 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 Read 2 control 0-32KB (MM222). These are the read control drivers.

      Use Main Storage

      Read 2 (from storage clock)

   b. Turn on the Y read current source (MM252).

      Y Source Read (from storage clock)

      Go (not M-register 0 bit)

   c. Turn on RD - 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 feed the decode switches directly.

      Y Rd Current Source

      Read 2 Control 0-32KA

      N Reg 0 Bit

      (not) N Reg 1 Bit

      (not) M Reg 7- Bit Ctrl

      (not) M Reg 4 Bit Ctrl

   d. Turn on RD 00--001 (MM442). This is the Y-decode switch for the sink side of the Y-line.

      Rd Current Sink

      M Reg Not 3 and Not 4 Bits

      (not) M Reg 5 Bit

      M Reg 6 Bit

5. 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 Rdl 0-8K WR 8-16KA and
      Rdl 0-8K WR 8-16KB.

      These are the X control drivers (MM232).

      RDl (from clock)

      (not) M Reg 1 Bit

      (not) M Reg 2 Bit

   b. Turn on the X-read current source (MM252).

      X Source Read

      Go (not M Reg 0 Bit)

      (not) M Reg 2 Bit

c. Turn on RD---010 (MM302). This is the X decode switch for the source side of the X-line.

X RD 0-8K WR 8-16K Source

RD1 0-8K WR 8-16KA

(not) N Reg 5 Bit

(not) N Reg 7 Bit

d. Turn on RD010--- (MM322). This is the sink side of the X line.

X RD 0-8K WR8-16K Sink

RD1 0-8K WR 8-16KB

(not) N Reg 2 Bit

N Reg 3 Bit

(not) N Reg 4 Bit

6. Develop the sense amplifier gate so the appropriate sense windings are gated to their respective sense amplifiers. The gate for this address is 5A gate 0-8KA (MM692).

Not M Reg 1 Bit Cont

(not) M Reg 2 Bit

(not) N Reg 7 Bit

7. Amplify and gate the sense pulses to the sense amplifier detector latches (MM512 through MM592).

SA Gate 0-8KA

SA In___Bit 0-8K

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 (not M Reg 0 Bit).

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. 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), 5 two address gates (one for each decode switch), and the Y-write current source.

a. Turn on the Y control-drivers, Write Control 0-32KB and Write Control 0-32KA (MM222).

Write B (from clock)

Use Main Storage ,

b. Turn on the Y-write current source and sink (MM252).

Y Source Write (from clock)

Go (not M Reg 0 Bit)

c. Turn on the Wr-0--010 write address decode driver (MM402). This includes the address gate, and is on the sink end of the X line.

(not) M Reg 7 Bit Ctrl

(not) M Reg 4 Bit Ctrl

N Reg 0 Bit

Y Wr Current Sink

Write Control 0-32KA

d. Turn on the WR 0001---write address decode driver (MM442). This includes the address gate, and is on the source end of the X-line.

M Reg Not 3 and Not 4 Bits

(not) M Reg 5 Bit

M Reg 6 Bit

Write Control 0-32KB

Y Wr Current Source

12. Select and drive the same X-line in the opposite direction. 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. Turn on X-control drivers Wr 0-8K Rd 1 8-16K, and Wr 0-8K Rd 1 8-16 KB (MM232).

Write A (from clock)

(not) M Reg 2 Bit

(not) M Reg 1 Bit

b. Turn on the X-write current source (MM252).

X Source Write

(not) M Reg 2 Bit

c. Turn on the X decode driver for the sink end of the X-line. This is Wr---010 (MM302).

(not) N Reg 5 Bit

N Reg 6 Bit

(not) N Reg 7 Bit

X Wr 0-8K Rd 8-16K Sink

Wr 0-8K Rd 1 8-16KA

d. Turn on the X decode driver for the source end of the X-line. This is Wr 010--- (MM322).

(not) N Reg 2 Bit

N Reg 3 Bit

(not) N Reg 4 Bit

X Wr 0-8K Rd 8-16K Source

Wr 0-8K Rd 1 8-16KB

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).

(not) N Reg 7 Bit

(not) M Reg 1 Bit

(not) M Reg 2 Bit

Inhibit (from clock)

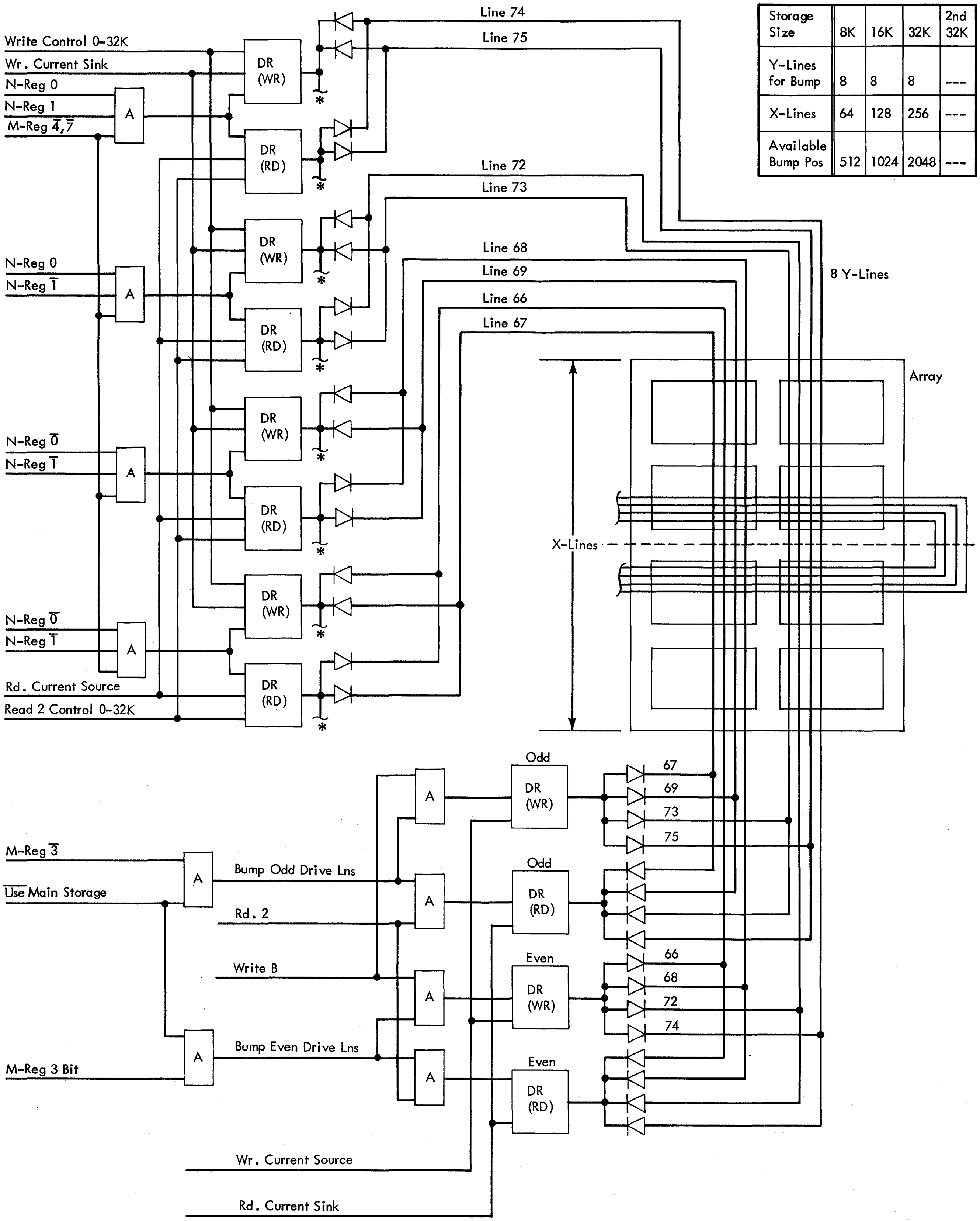
14. For those bits that are to be set ON, the inhibit driver must be blocked from turning on (MM732 through MM772). The store lines block their respective inhibit drivers.

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).

8K AUXILIARY STORAGE

- 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 four additional Y-line bump decode drivers (Figure 2-80). These drivers are controlled by the M-register 3-bit (MM152). These drivers control one end of the eight additional drive lines. The other ends are connected to 4 of the read and 4 of the write decode drivers 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.

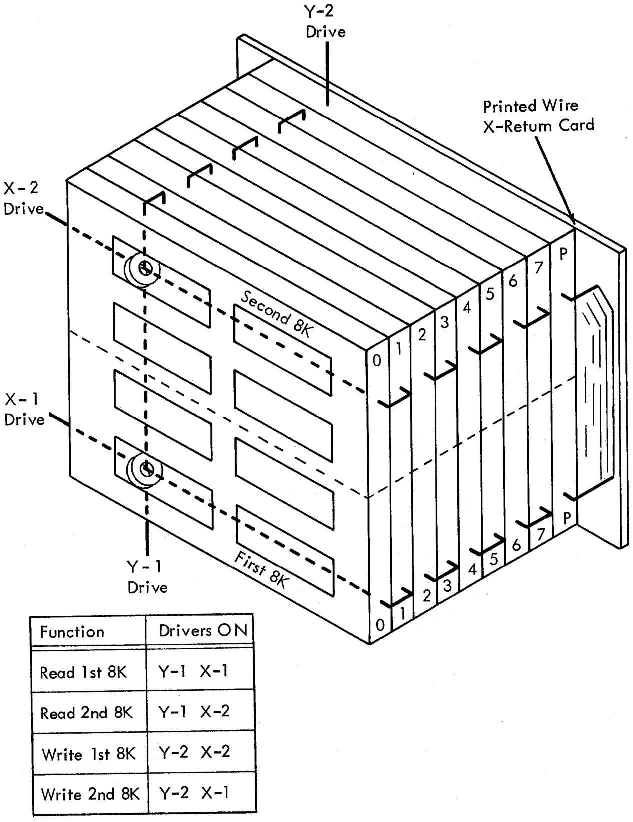
Figure 2-80. Auxiliary Storage Drive Scheme

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.

Description

The 16K storage unit contains nine core planes. The planes are wound so that two 8K storage units are produced (Figure 2-81). 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-winding is one set of 128 lines threading through all core planes. 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 2-82). This control takes place at the X-control drivers and at the X-source drivers.



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 2-81. 16K Storage Phase Reversal

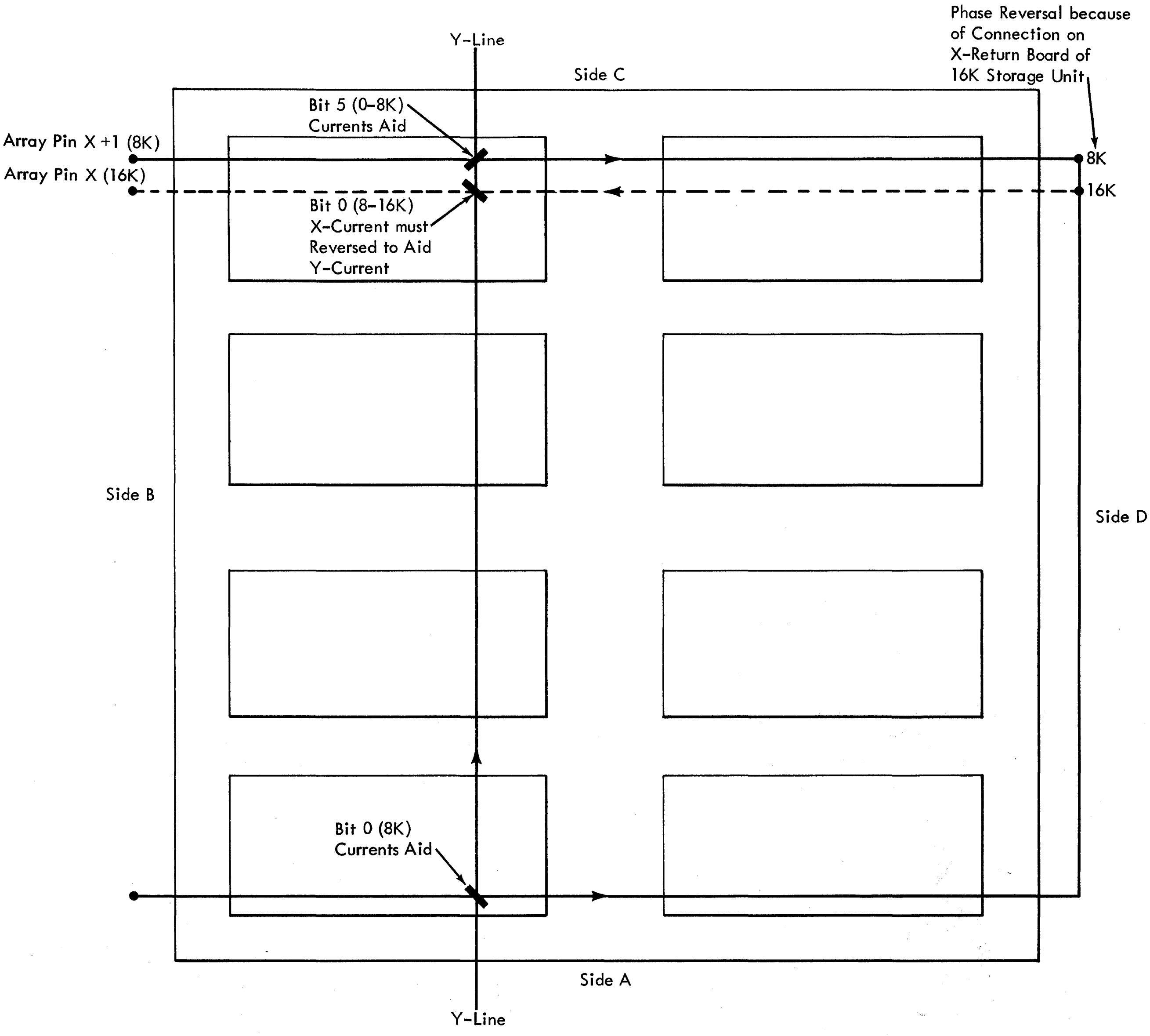

Phase Reversal because of Connection on X-Return Board of 16K Storage Unit
Y-Line
Side C
Bit 5 (0-8K) Currents Aid
Array Pin X +1 (8K)
Array Pin X (16K)
8K
16K
Bit 0 (8-16K) X-Current must Reversed to Aid Y-Current
Side B
Side D
Bit 0 (8K) Currents Aid
Side A
Y-Line

Figure 2-82. Phase Reversal Wiring for 16K

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

   (not) M Reg 2 Bit

   Read 1

   Wr

16K AUXILIARY STORAGE

- 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 positions.
- 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 drivers shown in Figure 2-80. These drivers are controlled by the M-register 3-bit (MM152). In the 16K storage unit the X-lines undergo a phase reversal between 8K units. Because of this phase reversal, selection of an auxiliary storage position requires X-line phase reversal control using the M-register 2-bit (MM232).

32K STORAGE OPERATION

- One single set of Y-lines drives all 18 core planes.
- Two sets of X-lines drive 18 core planes.
- 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 2-69). 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 uses the M-register 2- and 1-bits to determine which X-control drivers and X-source drivers are turned on.

Circuit Objectives

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.

1. Select the appropriate set of X-control drivers. The M-register 2-bit and 1-bit determines which set (first 16K or second 16K) of X-control drivers is used (MM232, MM242).

   M Reg 1 Bit

   M Reg 2 Bit

   Rd 1

   Write A

2. Control the direction of current flow 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

32K AUXILIARY STORAGE

- 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. Four additional Y-line decode drivers combine with the existing Y-decode drivers 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 tc 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 curcuitry, 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.

Address decode and drive circuitry is the same regardless of which unit is addressed. When a read call is issued and an address is placed into the M- and N-registers, both units begin to address and drive the same relative position as defined by the low 15 bits of the M- and N-registers (Figure 2-83). The high order address bit (M-register 0) 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), 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-orger core storage position. Because the high-order address bit is logical 1 (M-register 0 bit), the address desired is 32,768, and drive current is blocked in the first 32K. The address read out is 00,000 + 32,768 which is 32,768.

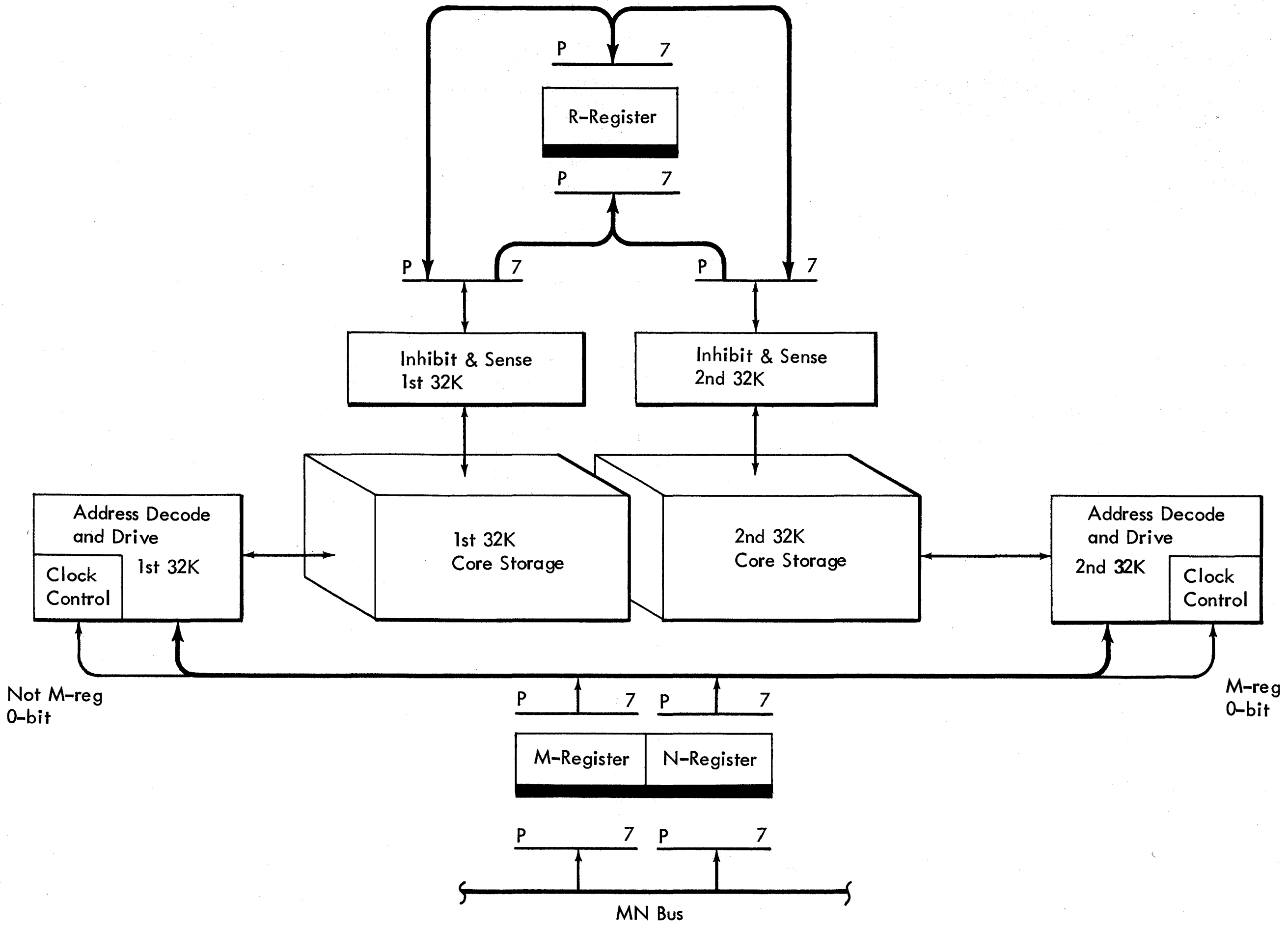


Figure 2-83. 65K Addressing

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. The M-register 0-bit is required to produce Go for the second 32K. Thus, Go will be active for either one unit or the other, but never both. In the unit where the Go signal is down, 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)
Clock Start 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 and the drive pulse to the delay line when Write Call occurs. This is possible 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 MMOO1)

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 blocks the delay line drive pulse (Logic Page MM122).

Early Local Storage (Logic Page MMOO1)

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 when the early local storage line is down, the main/local storage latch is reset to main to enable the Y-control drivers and block the local storage control drivers.

Read Echo 1 (Logic Page MMOO2)

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 1 (Logic Page MMOO2)

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 MMOO2)

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 MMOO2)

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. Data ready is the output of the data ready latch, and is gated by the M-register 0 bit. An 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.

CONCEPTS OF CAPACITOR READ ONLY STORAGE

- The capacitor is the important component of the read only storage unit.
- A line driver impulses many capacitors.
- Each control point in the data flow is controlled by a SAL.
- The logical statement is the ROS word.

PRINCIPLES

In the section titled Data Flow, the need for control points was indicated. Now let's use the "big picture" approach and build up a simplified block diagram of data flow (Figure 3-1).

On this block diagram, the control points have been numbered. For example, the in-gate control point for the G-register is numbered 3. To help illustrate again the need for control points we will use the logic statement READ OUT R, GATE THE OUTPUT THROUGH THE LOGIC UNIT, AND STORE IT IN S .

In our development we will show:

1. A control point source.
2. A selection device for the source.
3. Basic operation of ROS (Read-Only Storage).

Consider the first portion of the logic statement READ OUT R and, along with this, consider the control point for the R-register out-gate (2 in Figure 3-1). By adding a latch called a SAL (Sense Amplifier Latch) to this point, we have a device to gate R's output to this in-bus (Figure 3-2).

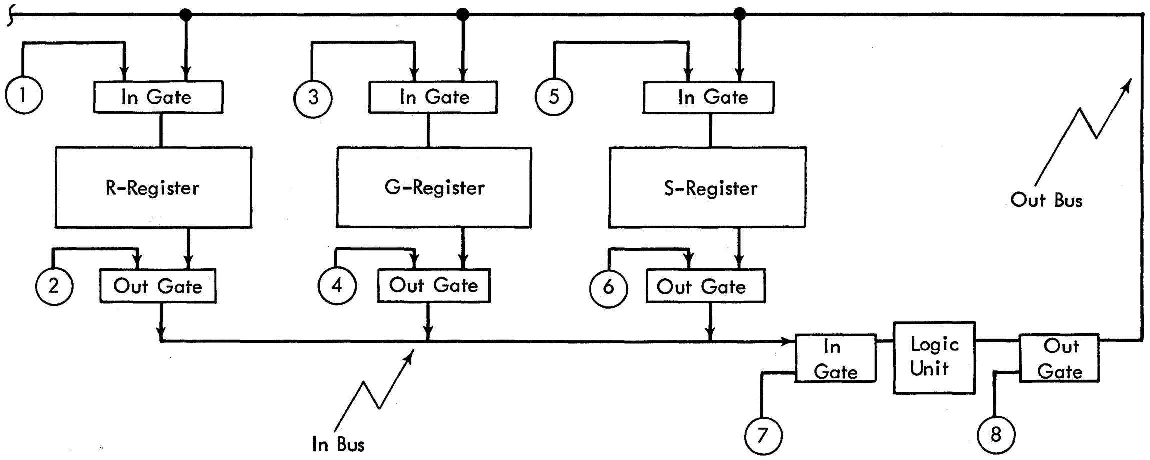


Figure 3-1. Control Points

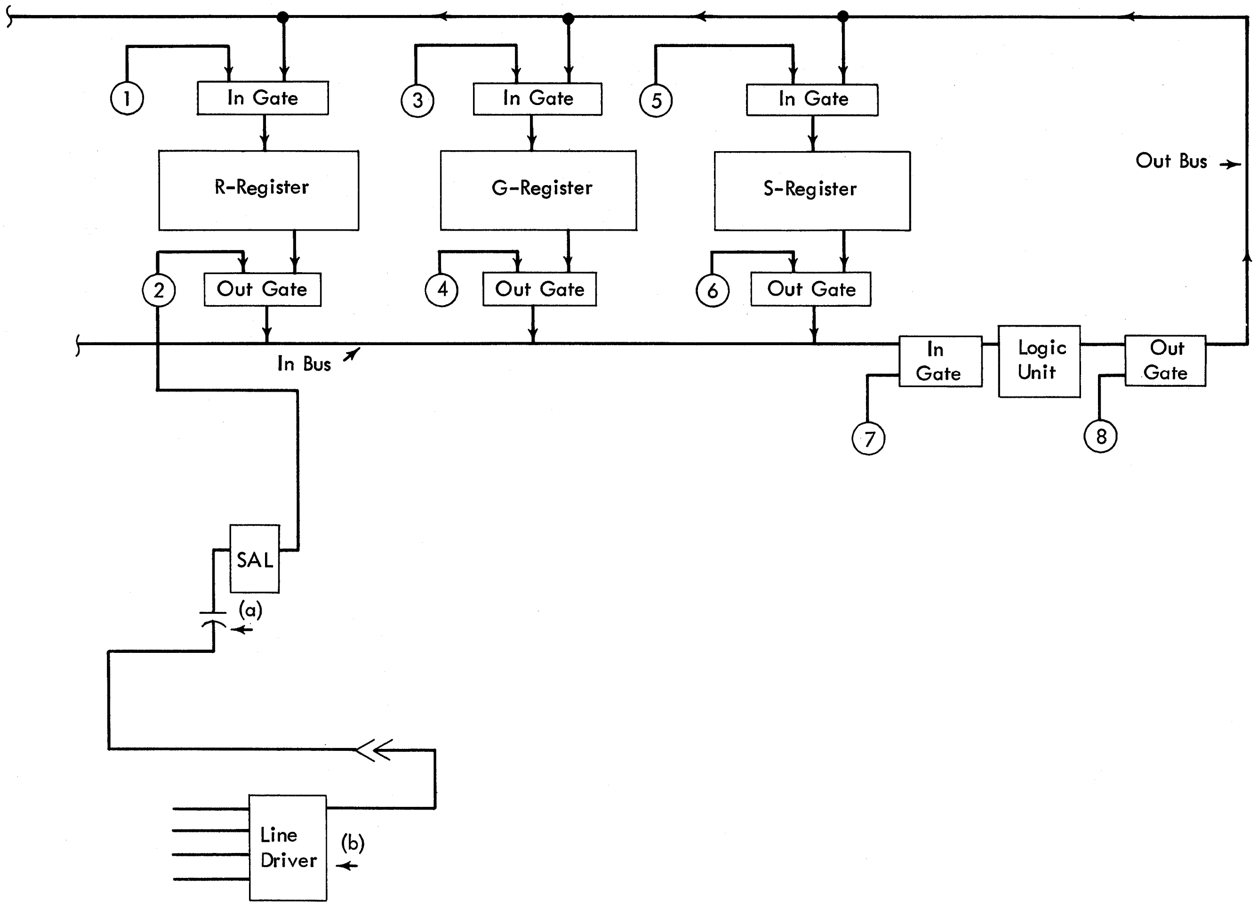


Figure 3-2. SAL Control

To control the turn-on of the SAL, we can use capacitive coupling to another device called a line driver (a and b in Figure 3-2).

So, if we impulse the driver, a pulse through the capacitor turns the SAL on, and its output can gate the contents of R to the in-bus.

Now let's build a little. The rest of the statement says in effect, read in and out of the logic unit, and read in to the S-register. To do this we need three more SAL's: one for control point 7, one for control point 8, and one for control point 5 (Figure 3-3). We'll couple them to the same line-driver.

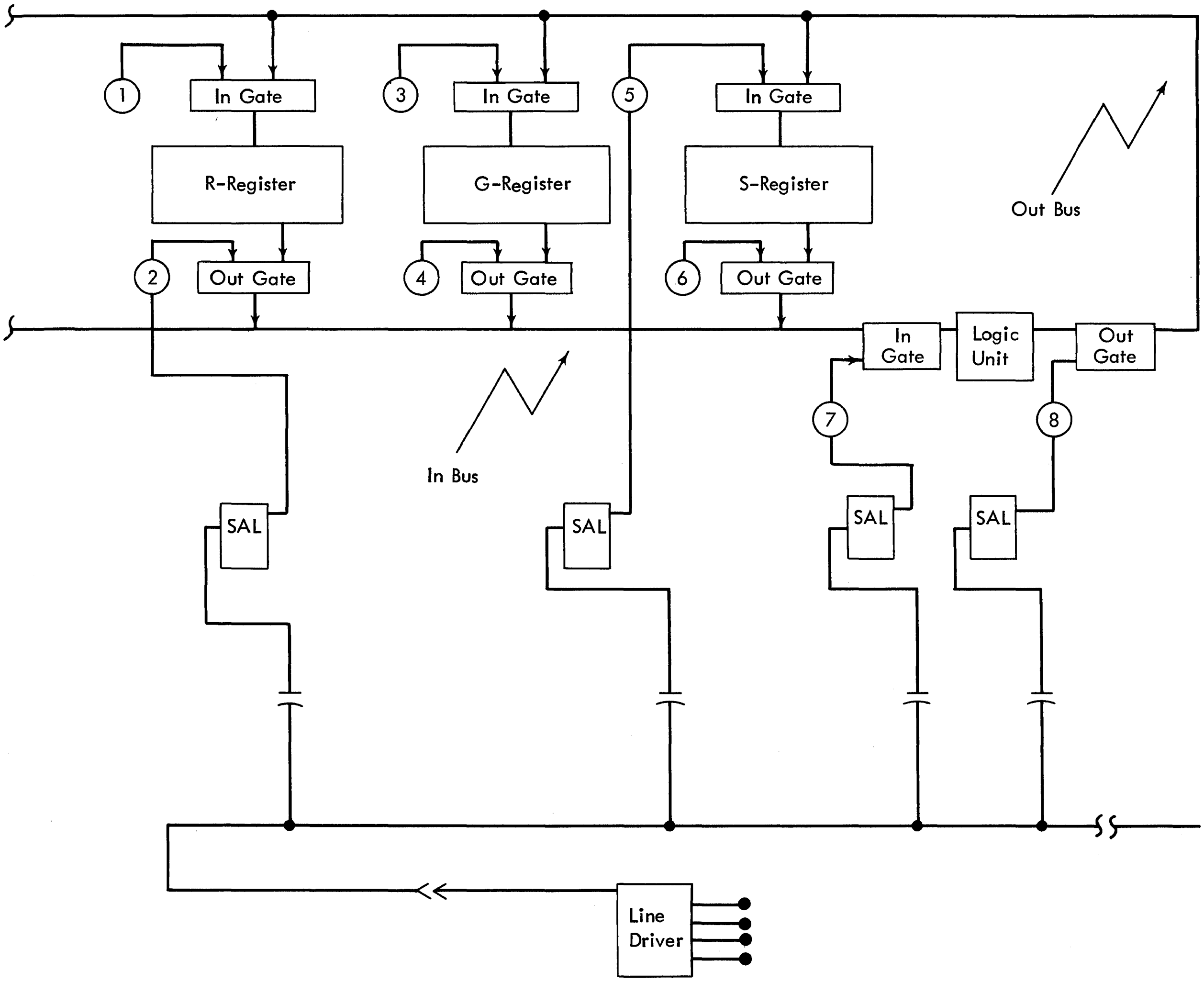


Figure 3-3. Multiple SAL's

Figure 3-3 shows that with one impulse from the line-driver, we can perform all the functions of our logic statement READ CUT R (control point 2), GATE THE CUTPUT THROUGH THE LCGIC UNIT (ccntrol points 7 and 8), AND STCRE IT IN S (control point 5). 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).

In our block diagram thus far, we have only shown SAL's that were active for our specific logic statement. There are a total of eight control points on our dataflow, and logically they all must have a source and a selection device.

Let's re-examine just one register in our dataflow, the R-register (Figure 3-4). Two control points are associated with this register: control point 1 for the in-gate, and control point 2 for the out-gate.

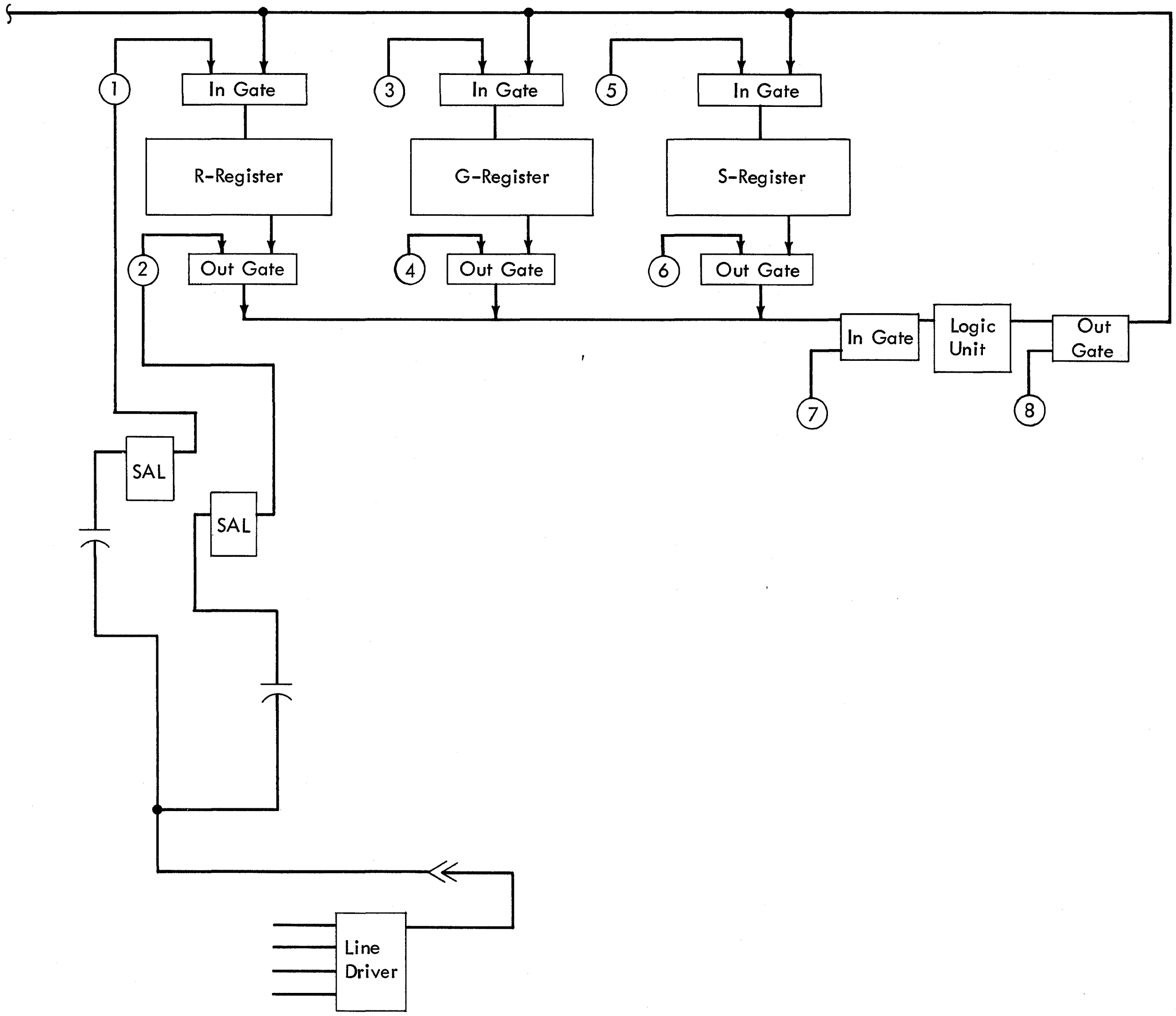


Figure 3-4. SAL Gate

If we add the source devices (SAL's) and couple them to the same line-driver we can't perform the first function of our logic statement (READ OUT R) (Figure 3-4). If we impulse the line driver now, we will read out R and also read in R. Although this is right electrically, and is a legitimate operation, it doesn't satisfy the function of our logic statement.

We must in some way modify the circuitry. Assume for the moment we can cut off one of the plates on the coupling capacitor for the SAL that conditions control point 1. (Figure 3-5). We can again perform the function READ OUT R.

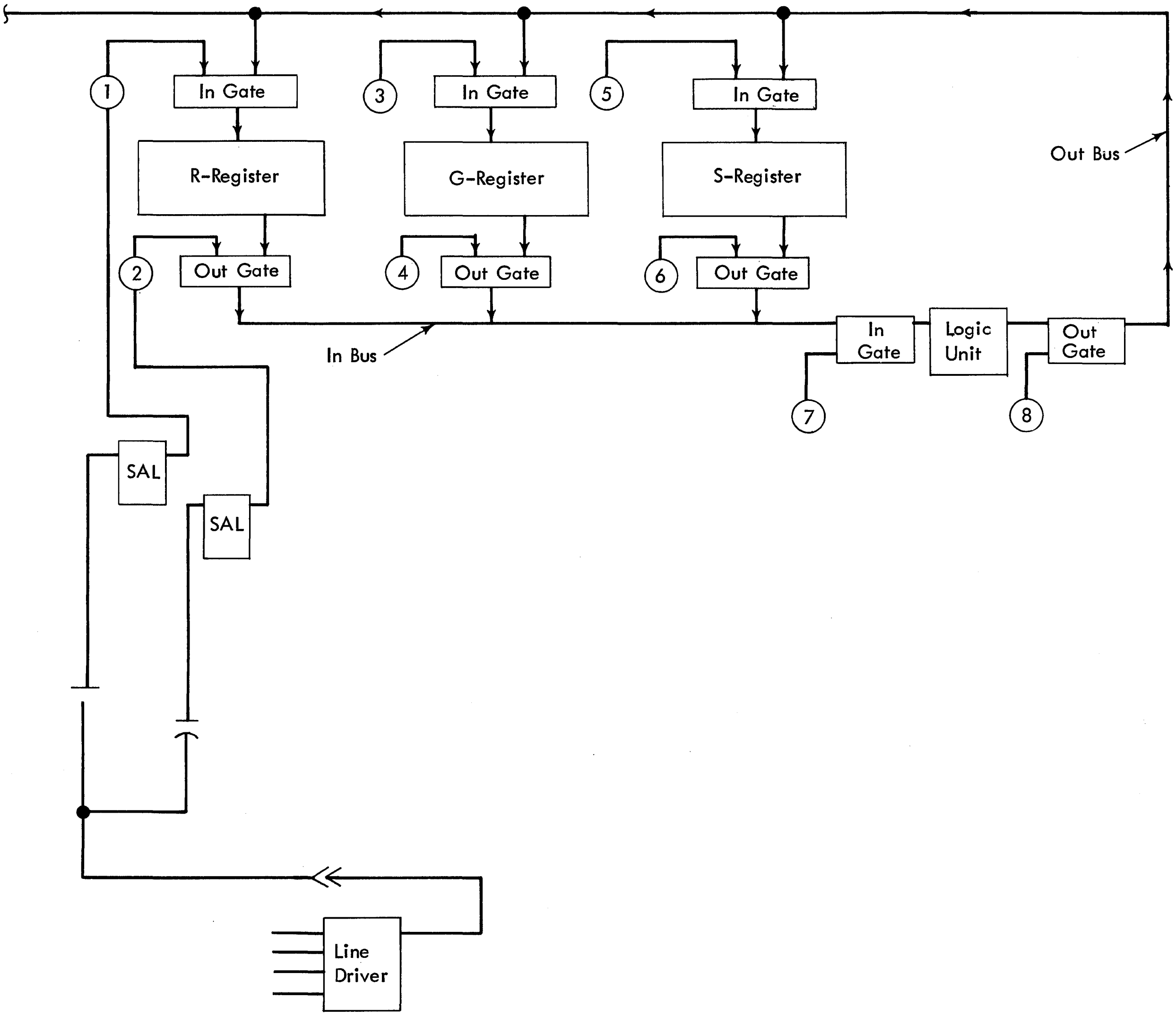


Figure 3-5. SAL Selection

Now we'll expand. Connect all the SAL's to their control points, but we will qualify them by having only one plate on any coupling capacitor we're not going to use (Figure 3-6). If we impulse the line driver, we can perform all the functions of our logic statement READ OUT R, PUT THE OUTPUT THROUGH THE LOGIC UNIT, AND READ IN S. Check, using Figure 3-6. Now we have a storage device, and every time we impulse the line driver, it performs the same function or logic statement. Let's state it a little more simply. If we impulse the line driver, we read out the same control gates every time. We have in reality a read only storage device. The device is made of capacitors with either one or two plates using a common drive line. The device is called a Read Only Storage, or ROS.

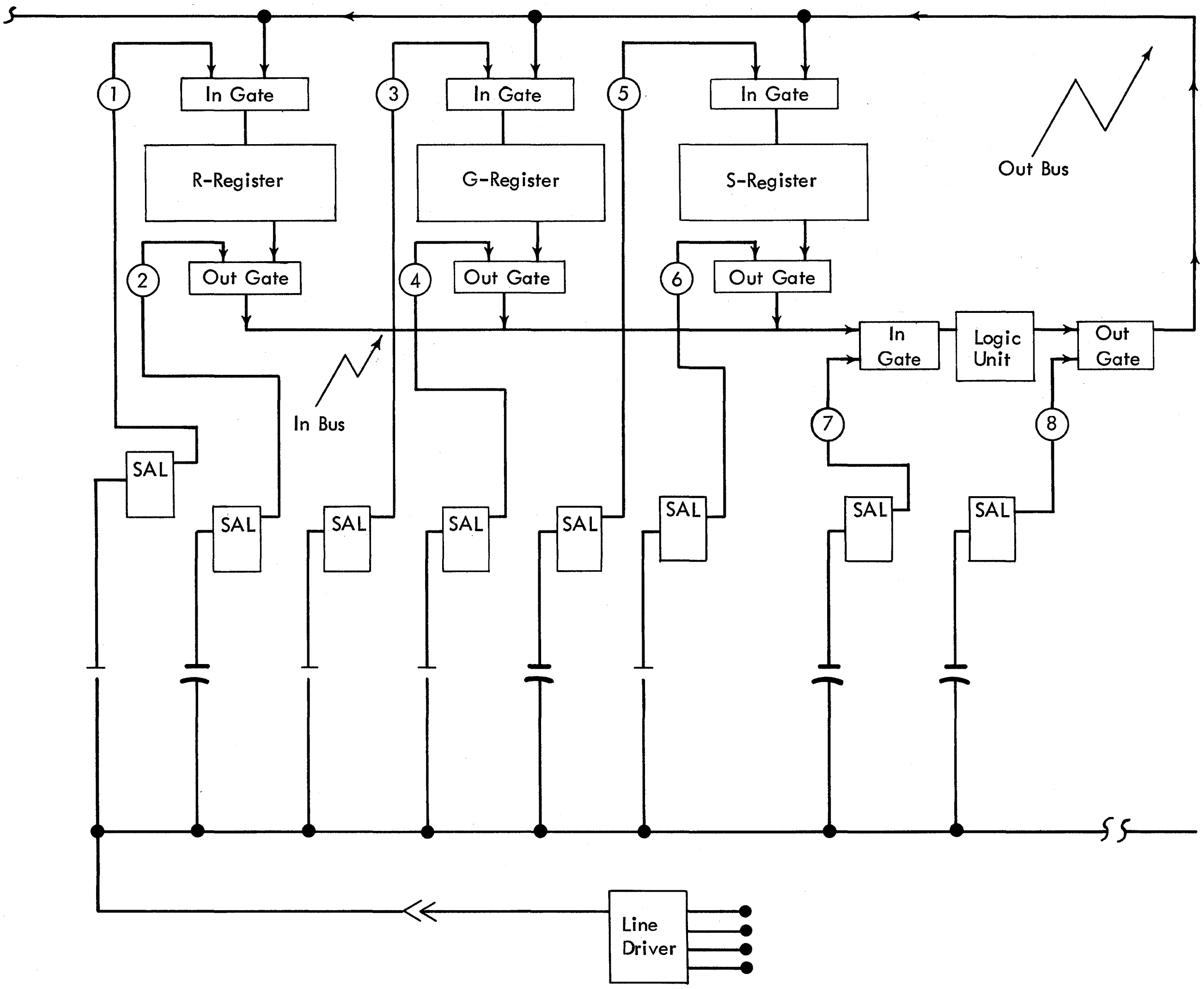


Figure 3-6. Multiple SAL Selection

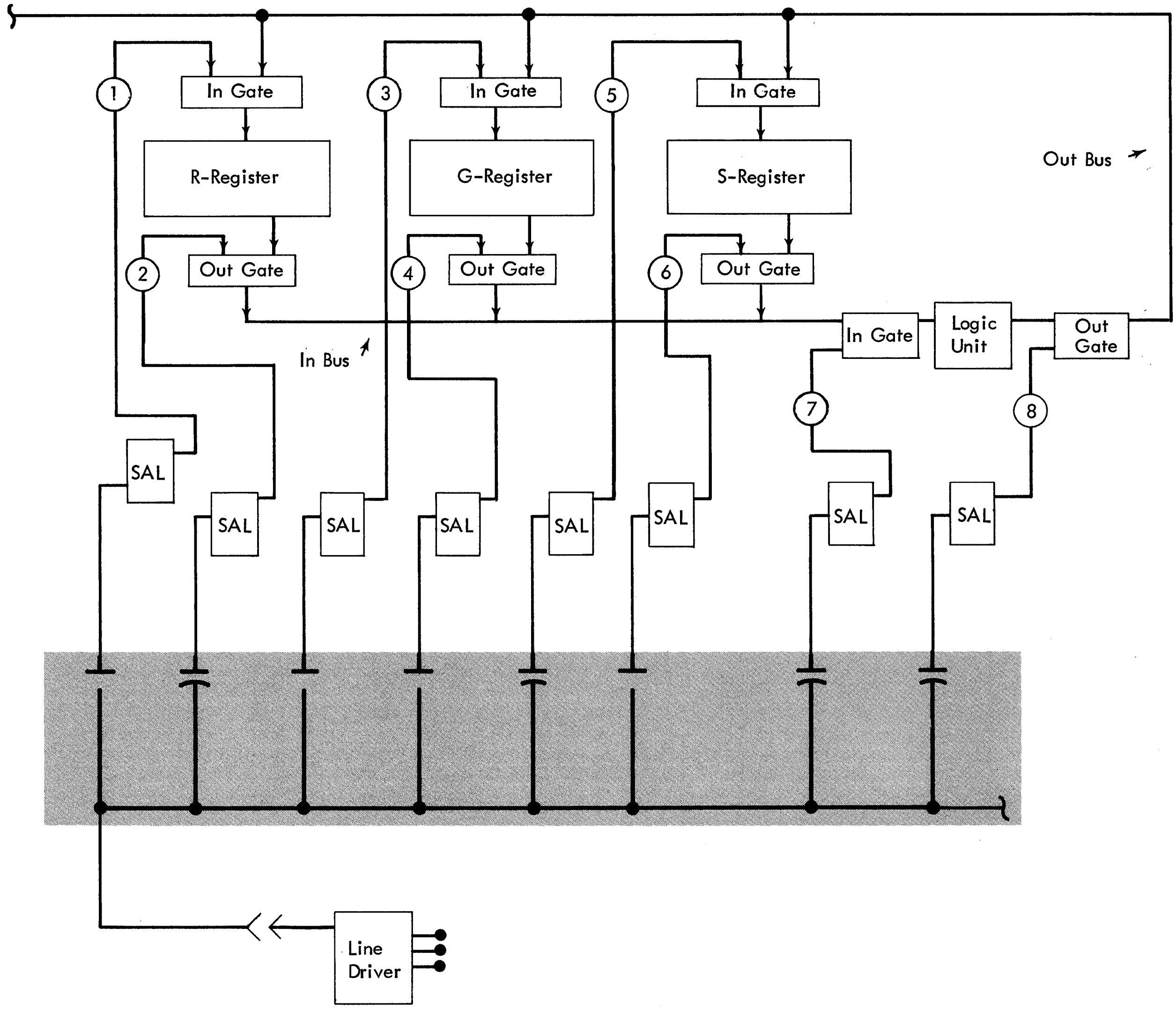


Figure 3-7. SAL Capacitors

RCS WORD

In our development let's point up the specific control used to accomplish the functions of our logic statement.

Figure 3-7 shows the SAL's were selected by capacitors commoned to a line driver. The string of capacitor plates having a common line feeding them is called a RCS-word.

If we want to expand the logic functions of our machine, we must be able to change this word or to add another. We would like to continue to perform the first function we developed. Let's add another RCS-word and another driver (Figure 3-8). This new word can perform the function of the logic statement <u>READ OUT R, TAKE THE OUTPUT THROUGH THE LOGIC UNIT, AND READ IN G</u>. Check, using Figure 3-8. Now, if we impulse line driver one, we take what's in R and put it in S. If we impulse line driver two, we take what's in R and put it in G. We can keep adding plates to the SAL's and more ROS-words until we can perform any function our data flow can handle.

Now all we need is any easy way to do this.

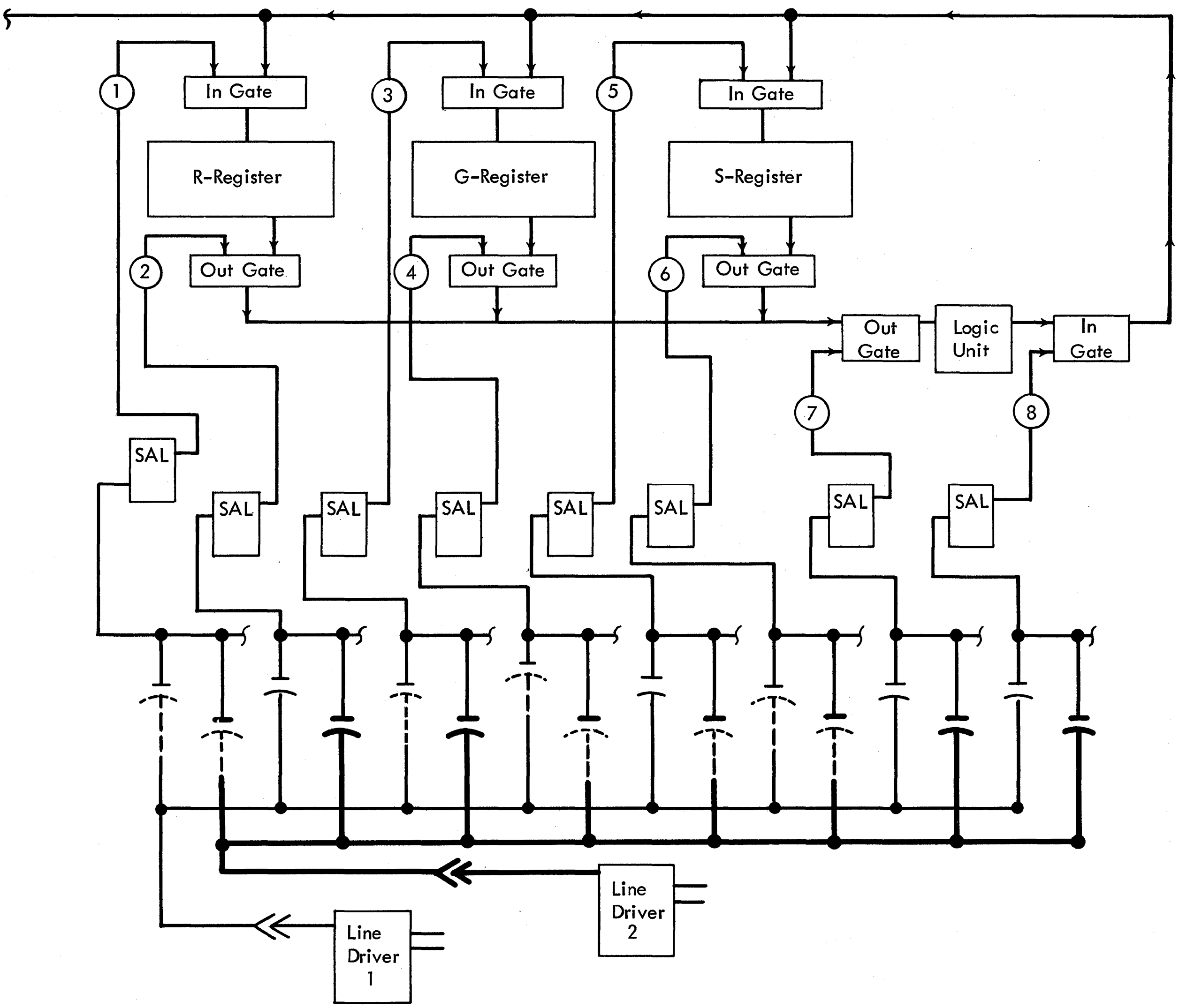


Figure 3-8. Multiple Drivers

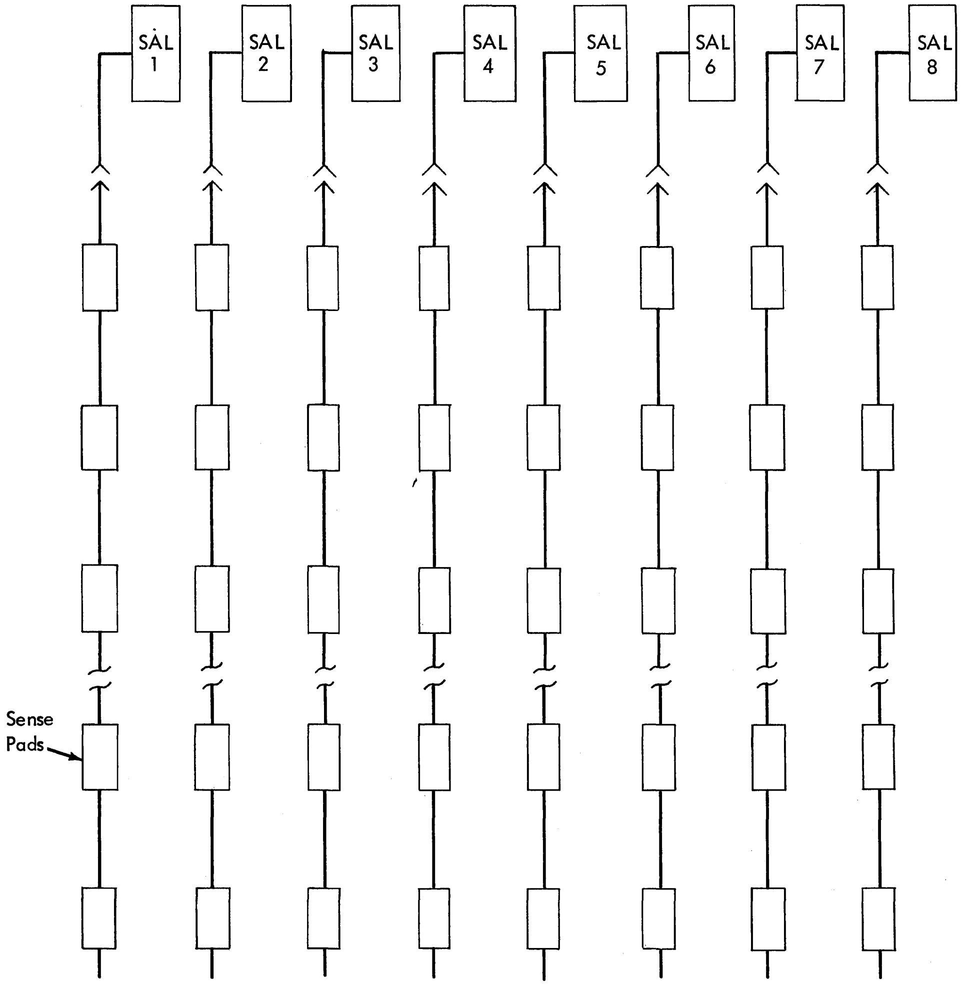
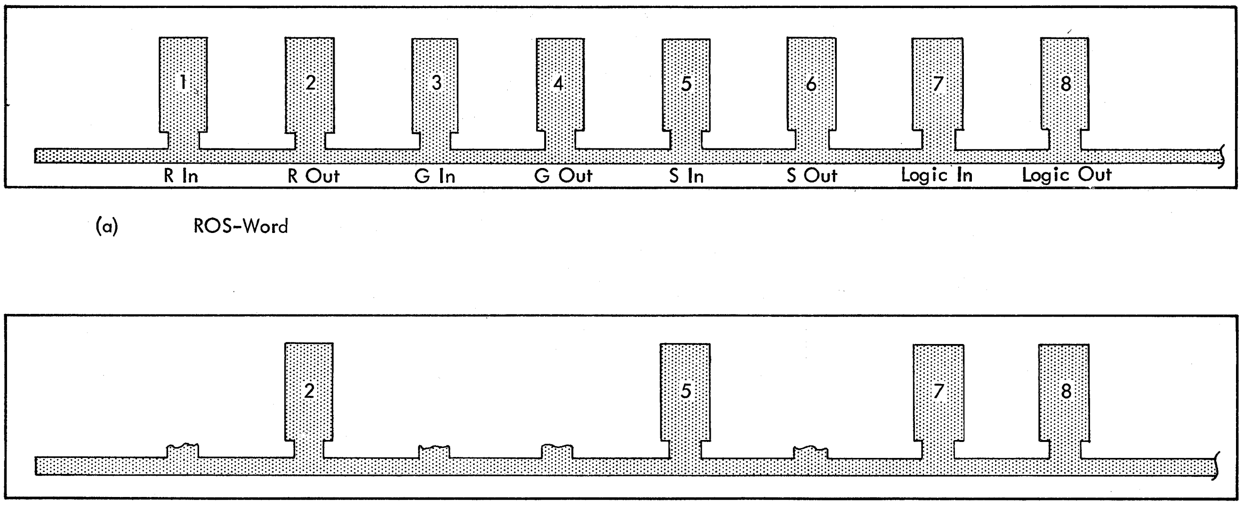


Figure 3-9. Sense Pads

First of all, let's look at the plate connections to the SAL's. In Figure 3-9 the capacitor plates are shown connected serially to a SAL. They are copper plates, laid out on a board of laminated fiber board. These strings of plates are called sense pads.

The plates connected to a line driver (RCS-word) are connected differently. They are laid out in parallel on a Mylar* strip. Figure 3-10a shows a RCS-word before it has been programmed. Figure 3-10b shows a RCS-word programmed to perform the first logic statement we used in our data flow development.

*Trademark of E. I. DuPont de Nemours



(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 3-10. Programmed ROS Word

In Figure 3-11 the Mylar strip with a programmed ROS word is shown placed over the sense-pads. If the line driver is impulsed, it will activate the SAL's required for our first logic statement. In Figure 3-12, we add the second ROS-word, and impulsing its driver activates the SAL's for statement two. By adding more ROS-words and more drivers, we can continue to build.

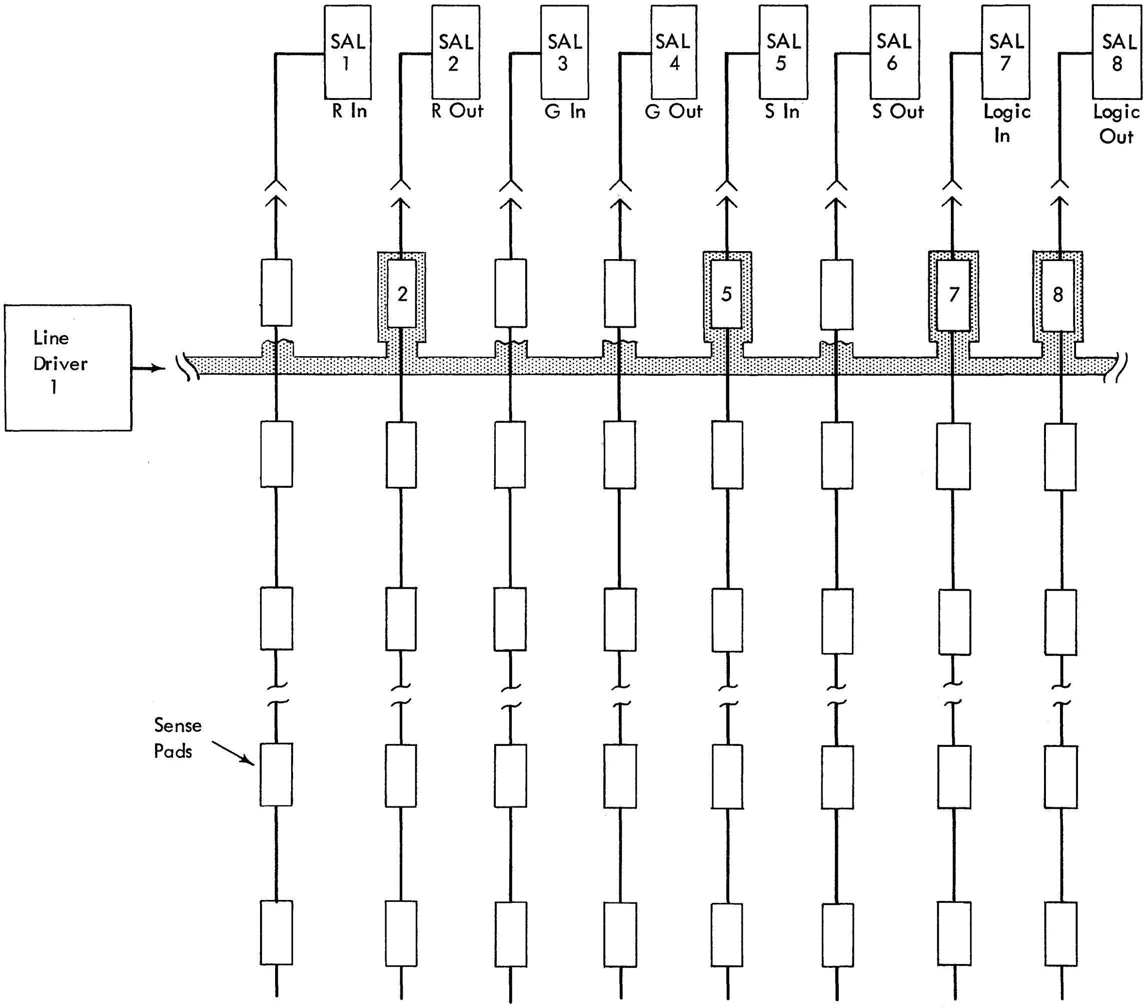


Figure 3-11. Functional Selection

In our actual CPU, the data flow is a bit more complex than the one we developed. It follows then, that our choice of control point selection must be a little more sophisticated. Our first consideration will be given to the physical layout and size of the ROS word.

The primary unit of capacitor read only storage is the ROS word. In the System/360 Model 30, the ROS word is 60 bits wide. This means there are 60 capacitor plates on one line driver. The words are packed on a sheet of Mylar exactly the same size as an 80-column punched card. The plates are positioned so as to coincide with the normal punching positions of a card. They are connected to a line running from the column-1 end to the column-80 end. This allows us to place 12 words on a ROS document (see Figure 3-13). This also gives us the capability of programming our words on existing punched-card equipment, such as the IBM 24, 514, or 1402.

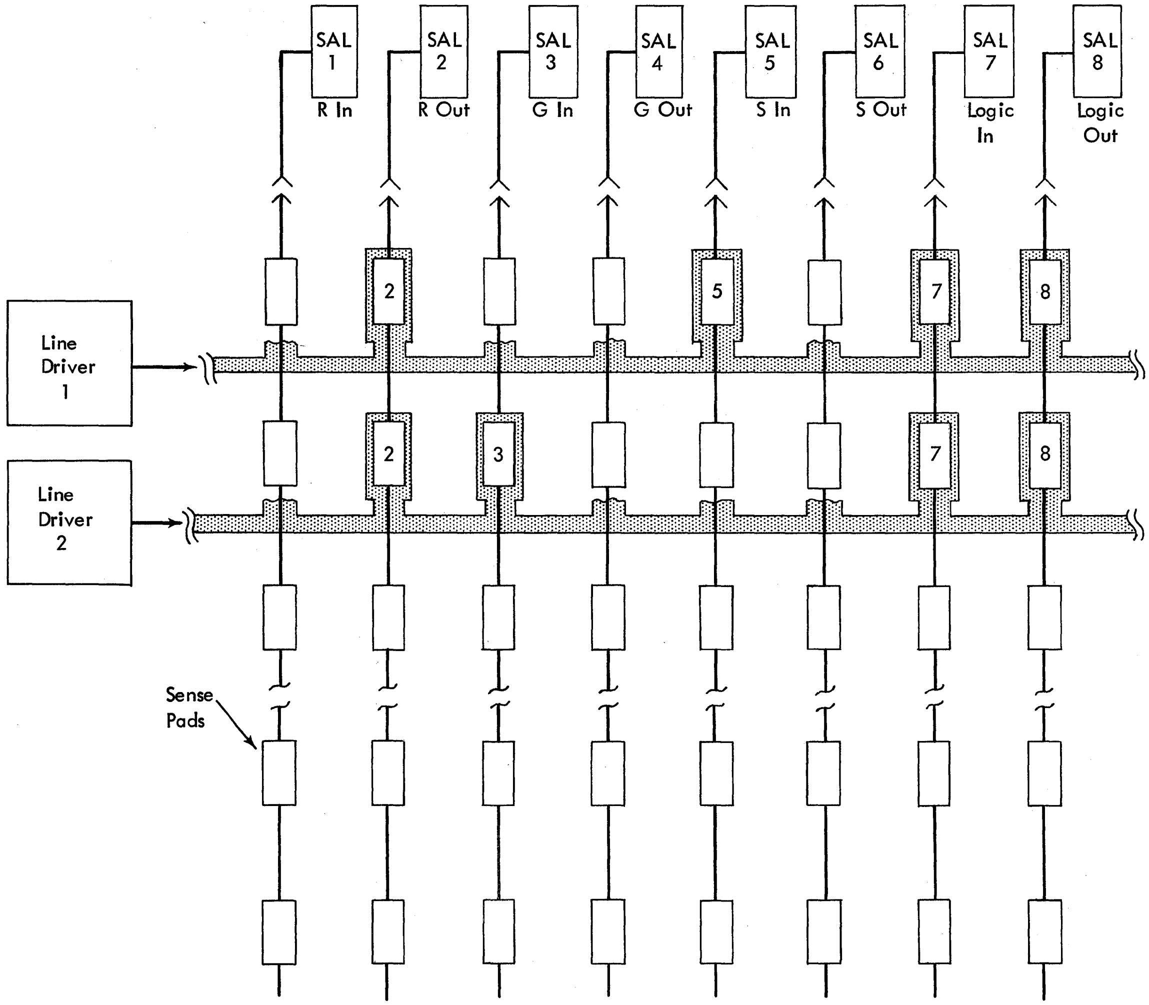


Figure 3-12. Two ROS Words

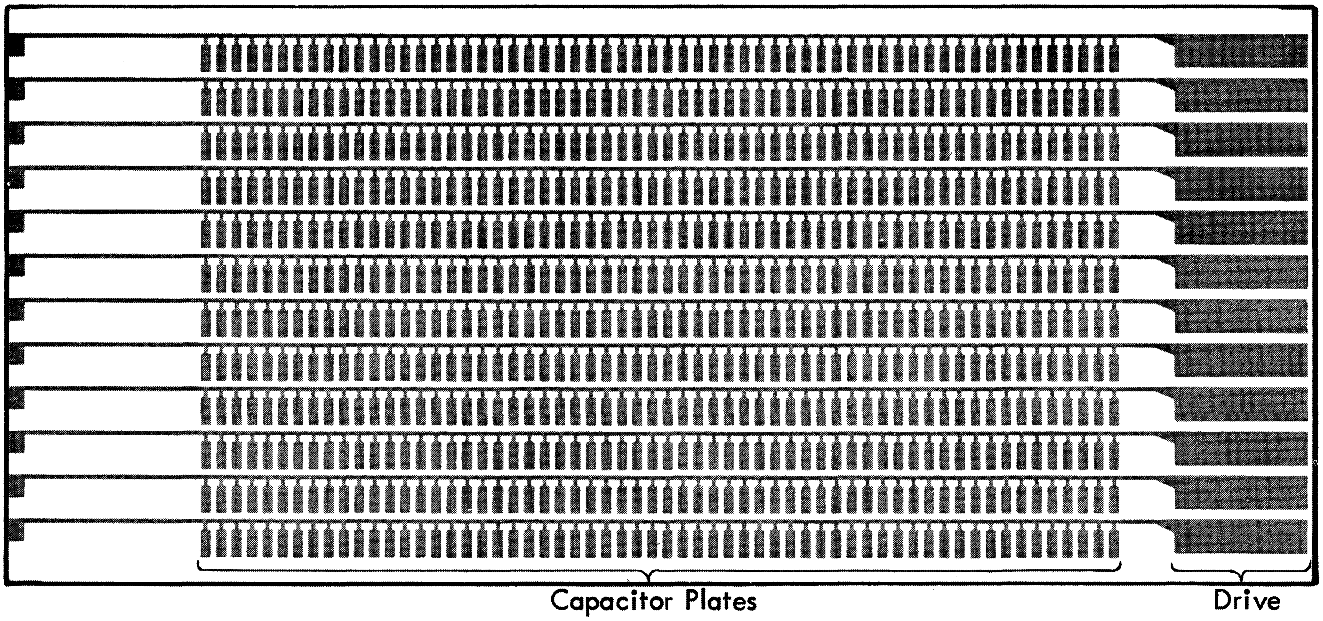


Figure 3-13. RCS Document

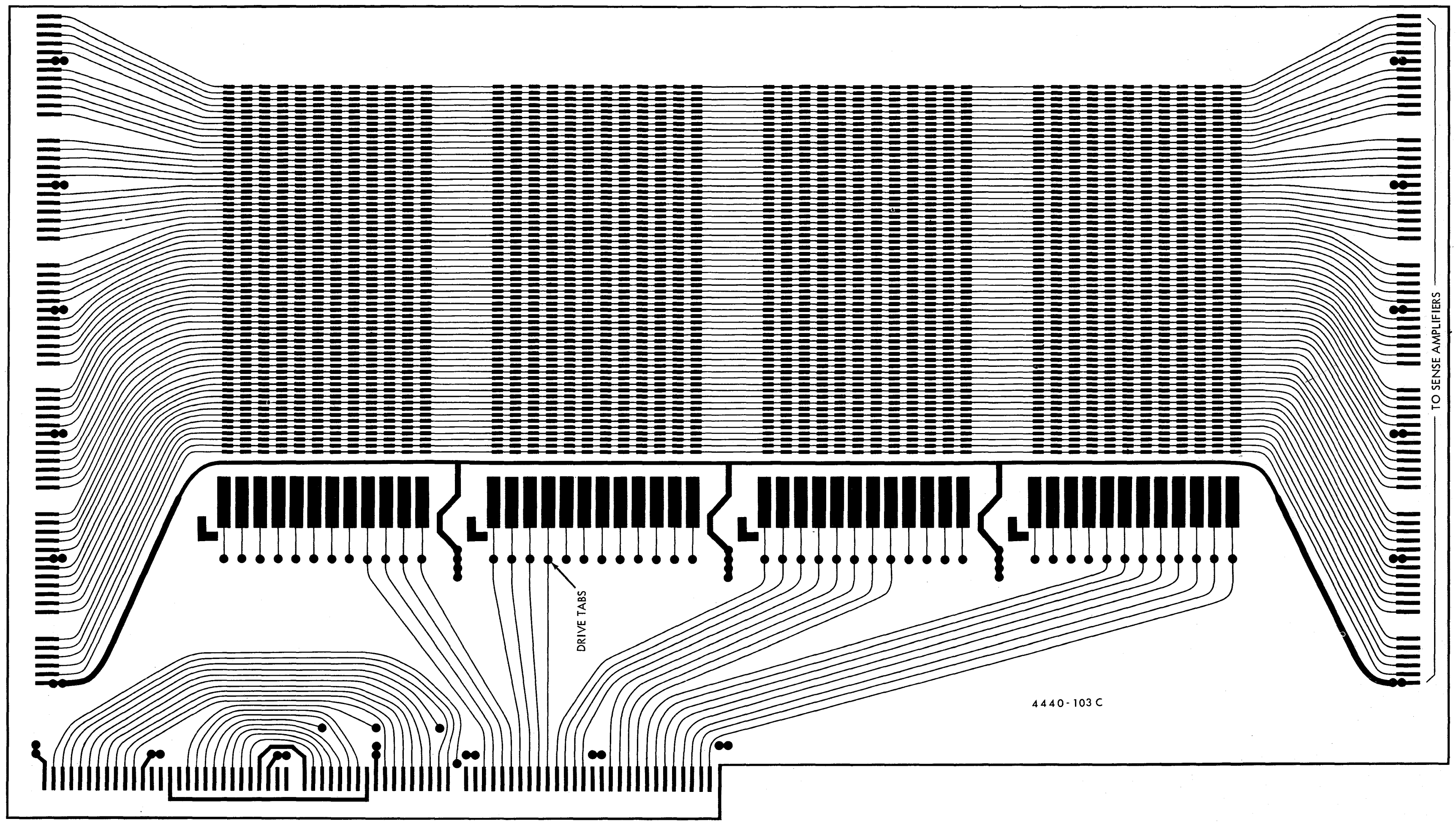


Figure 3-14. ROS Board

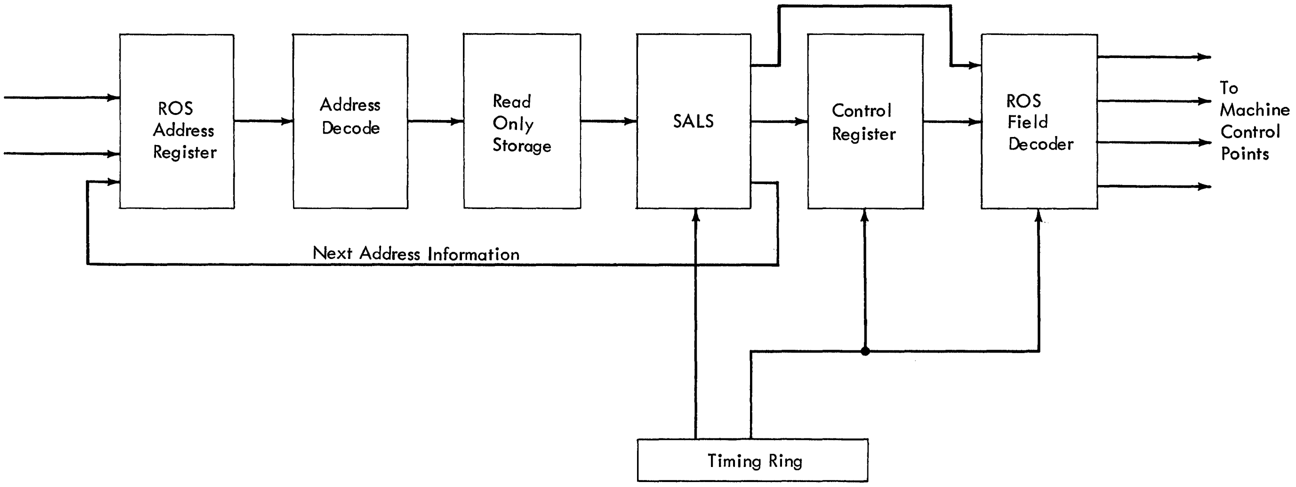


Figure 3-15. RCS Data Flow

The SAL sense pads are laid out on a RCS board as in Figure 3-14. The serial sense pad connections run vertically on the card. Both sides of the RCS board have identical sense pad patterns, so we can accommodate 8 RCS documents of 12 words apiece, for a total of 96 RCS words per RCS board. The 4K RCS module is made up of 42 RCS boards for a total of 4,032 ROS words.

RCS DATA FLCW

The control for the System/360 Model 30 processor is designed around a RCS device. It includes hardware for:
1. Addressing the RCS
2. Sensing and decoding the output of RCS
3. The basic clock

Schematically, it is depicted in Figure 3-15. The address register is decoded to select a word from RCS. This word is read out and set into the SAL's. Some of this information is set into additional latches, control-register latches, because it is not used until the following clock cycle. The ROS fields are decoded to activate the machine control points.

REVIEW QUESTIONS-RCS CCNCEPTS

1. Impulsing a __________ reads out the same control gates every time.
2. An impulsed line driver provides a pulse through a capacitor to turn on a __________.
3. A string of capacitor plates having a common line feeding them is called a __________ word
4. #__________ line driver(s) may be impulsed at one time.
5. The capacitor plates feeding a SAL are connected __________.
6. The capacitor plates that are fed from a line driver are connected in __________.
7. The RCS word is __________ bits wide.
8. There are __________ RCS words on one RCS document.
9. One RCS board can accommodate __________ RCS cards.
10. A 4K module is made up of __________ ROS boards.

MICRC PROGRAMMING INTRCDUCTION

Briefly, to perform the functions of an Cp code, such as move, add or branch, requires that the CPU initiate a sequence of logical steps. This group of steps in the System/360 Model 30, is called a micro program. Within the micro program, the ROS word is the functional statement. An Cp code is read from main storage by an address initiated in the ROAR (Read Only Storage Address Register). When the addressed RCS word is read, its contents are decoded to activate control points in the system. The RCS word consists of specific fields selected or programmed to perform a logic statement. The activated word also sends back next address information to the ROAR. Coupled with branch control (machine status tests) it forms the address of the next ROS word. To understand ROS words we must know what the ROS word contains, and what format is used to write the word.

ROS CONTROL FIELDS

- The 60-bit ROS word is divided into 14 control fields.

In Figure 3-16 notice the control fields vary in numbers of bit positions. If the field is two positions wide, we can set and decode four combinations: 00, 01, 10, 11. If the field is three positions wide, it can be set and decoded to eight combinations, 000 through 111. If it is four bit positions wide, it can be set and decoded to 16 combinations, 0000 through 1111.

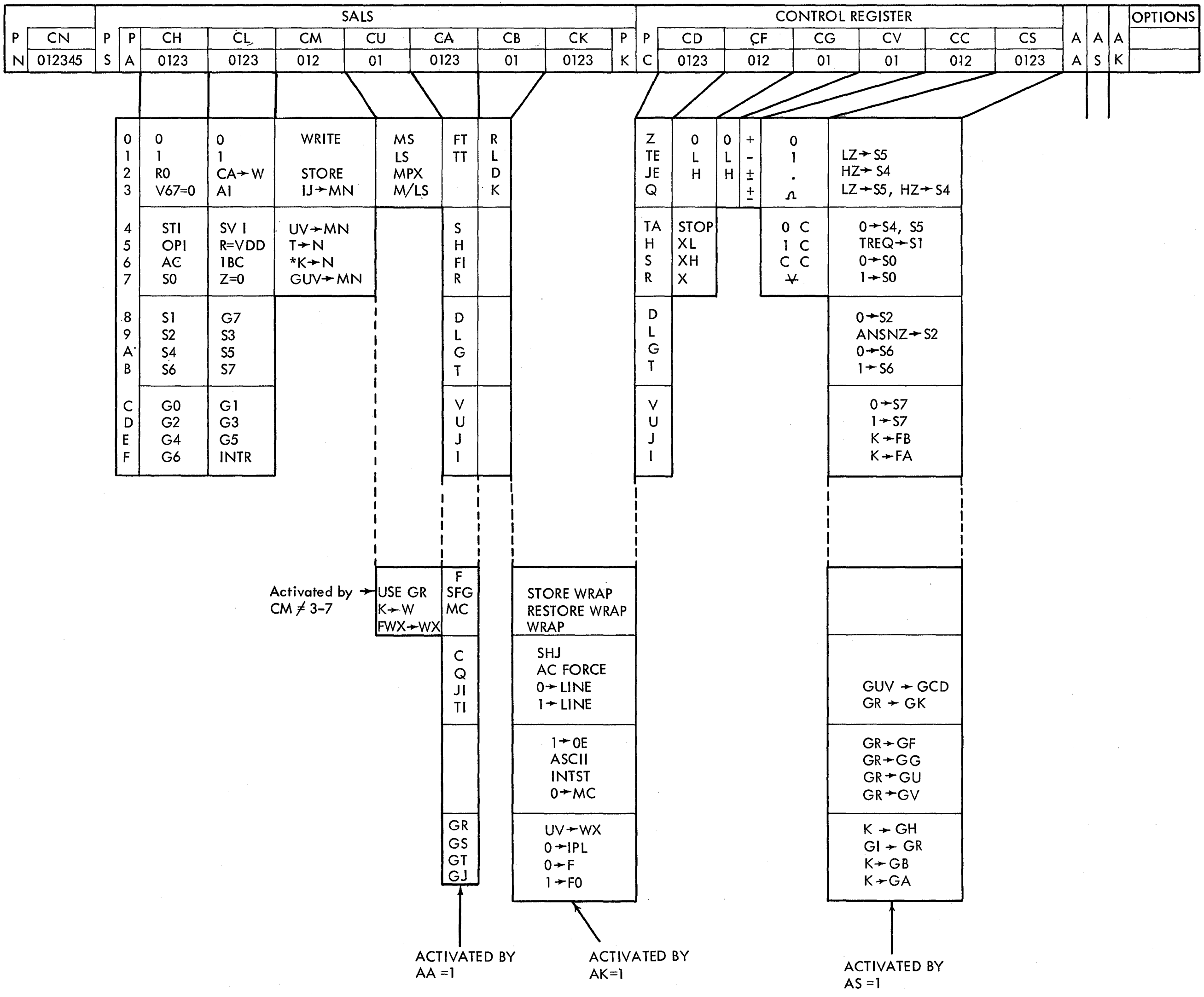


Figure 3-16. ROS Control Fields

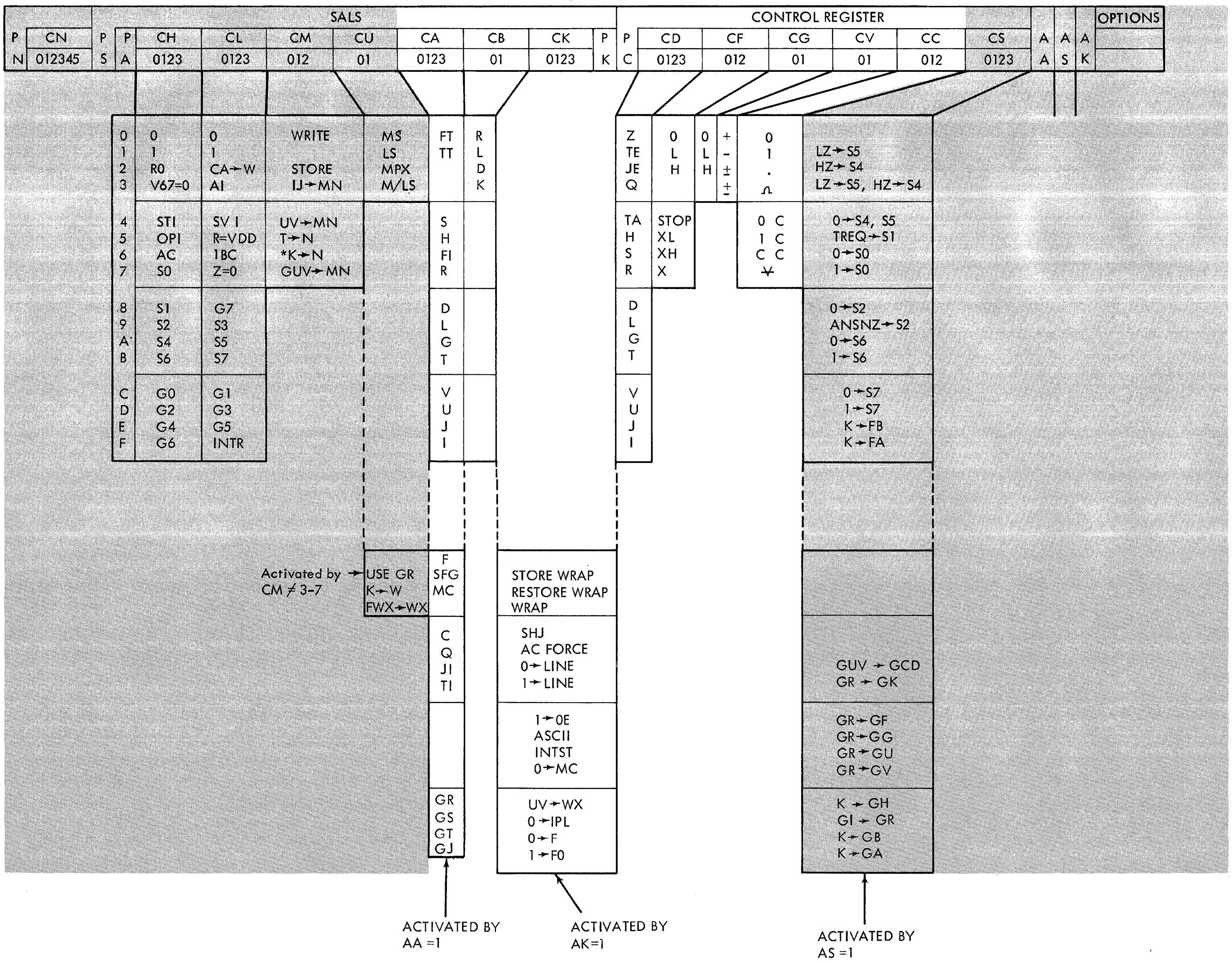


Figure 3-17. RCS Function Control Fields

These 14 fields can be separated into three broad groups:

1. Function Control (8 fields)
   CA, CF, CB, CG, CC, CV, CD, CK
2. Storage Control (2 fields)
   CM, CU
3. Branching and Address (4 fields)
   CN, CH, CL, CS

Function Control

The function control fields are subdivided into four groups. Figure 3-17. These fields are used to control all data movement within the CPU. ALL DATA MOVEMENT IS THROUGH THE ALU (Arithmetic Logic Unit). We will discuss the function control fields and the way they control data presented to either the A-register or B-register sides of ALU.

The function control groups provide:

1. The source and control for the A-register-CA, CF
2. The source and control for the B-register CB, CK, CG
3. ALU control-CV, CC
4. The destination-CD

"A" SOURCE (CA): The 4 bit CA field selects the source of data for the A register. It can be decoded to 32 combinations, 16 primary and 16 alternate.

"B" SOURCE (CB): The 2-bit CB field selects one of three registers, or the emit field (CK) to be presented to the B register.

EMIT FIELD (CK): The 4-bit CK field allows the micro programmer to use constants from the ROS. When CK is used as a B input, its 4-bit configuration is presented to both the high order and the low order 4 bits of the B-register.

<u>Example</u>: If the value in the CK field is 0110, the value of 6 is presented to both high and low positions of the B-register.

DESTINATION (CD): The 4-bit CD field selects the destination of the ALU output.

<u>Note</u>: A given register may be used as both the source and as the destination during a single ROS cycle.

"A" CONTROL (CF): The 3-bit CF field controls the data bit presentation from the A-register to ALU. This field is essentially bit significant, with a bit for the high gate, a bit for the low gate, and a bit for the straight or cross switches. The machine stop function is also found here.

"B" CONTROL (CG): The 2-bit CG field controls the high/low 4-bit gating from the B-register to ALU. There is no straight/cross function for the B register.

ARITHMETIC FUNCTION (CV): The 2-bit CV field is the true/complement and binary/decimal control. The B-Register input is the true/complement side of ALU.

ARITHMETIC CONTROL (CC): The 3-bit CC field controls carry-in, carryout, AND, OR, and exclusive OR functions. This field also permits the setting of a carryout into the carry latch, if desired.

<u>Storage Control (CM, CU)</u>

- Storage control fields (CM and CU) will provide control for storage by:
  1. Section Selection (Main Storage, local storage, or MPX 0-6).
  2. Operation-Read, Write, Compute, Store.
  3. Storage address source (MN source).

The CM and CU fields work in conjunction with each other and their control is best described considering both fields, and some basic core-storage cycling. (Figure 3-18).

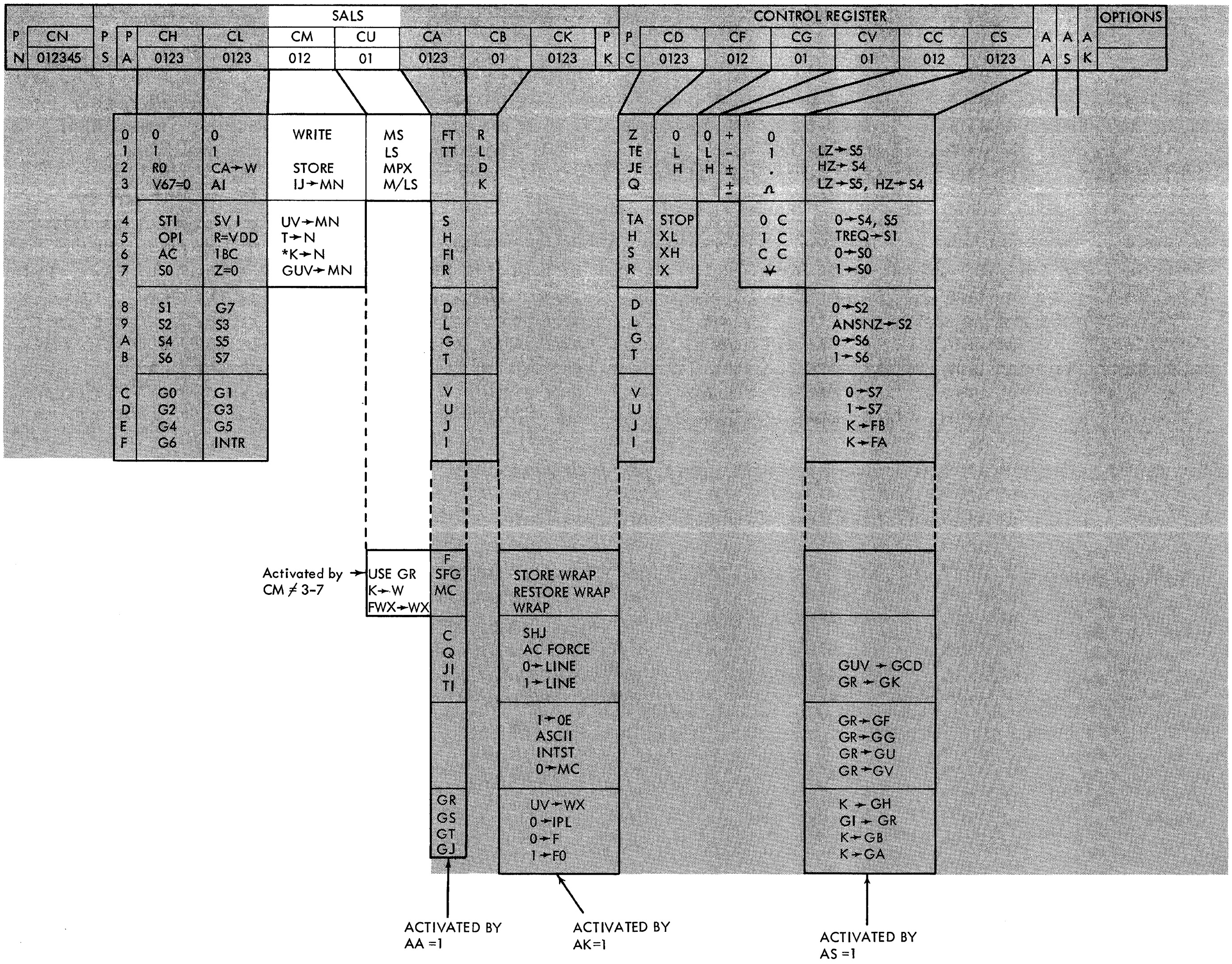


Figure 3-18. RCS Storage Control Fields

In the 2030 there are four basic cycles for core storage.

Read, Write (R, W)
Read, Compute, Write (R, C, w)
Read, Store (R, S)
Read, Compute, Store (R, C, S)

When a read call is given, the data from core storage is not ready for use until the beginning of the next ROS cycle. The next cycle then must be a write, a store, or a compute cycle.

If the read cycle is followed by a write cycle, the data will be set in the storage data register (R) and sent back to core from R. If followed by a store cycle the output from core is not used but lost. The new information for core is the data already in R, and this is gated back to core.

If the read cycle is followed by a compute cycle, the output from core is taken to R and must be written back to core on the cycle following the compute cycle with either a write or store.

READ, WRITE (CM): The 3-bit CM field controls the operation to be performed. A read call is sent to core storage using one of the CM field combinations 3 through 7. A write call is specified by the codings 0 or 2.

Note: If the CM field is combination 000 (Write), 001, or 010 (Store), the alternate coding for CU is used. A 001 allows the information to be set in R but it is not regenerated. Essentially, this is the compute cycle.

CORE CONTROL (CU): The 2-bit CU field controls which area of core is addressed. Its alternate codings have special uses.

Note: The CU coding of 11 is further decoded by the Op register (G-reg). If the high-order two bits of the Op register are 00, the instruction to be performed is in RR format and this combination specifies local storage. If the high-order two bits are anything other than 00, then this combination specifies Main Memory.

Branching Control (CN, CH, CL, CS)

- The branching control fields, CN (6 bits), CH (4 bits), CL (4 bits), and CS (4 bits), provide an address for the next ROS word to be executed (Figure 3-19).

A ROS address is a 15-bit binary number. The W-register holds the high-order five bits, and the X-register contains the low-order eight bits. Two positions are used for parity. One position for the W-register, the other for the X-register. Normally only the eight bits in the X-register are provided by the branch control fields. Of these eight bits, six bit positions (0-5) of the X-register are called next

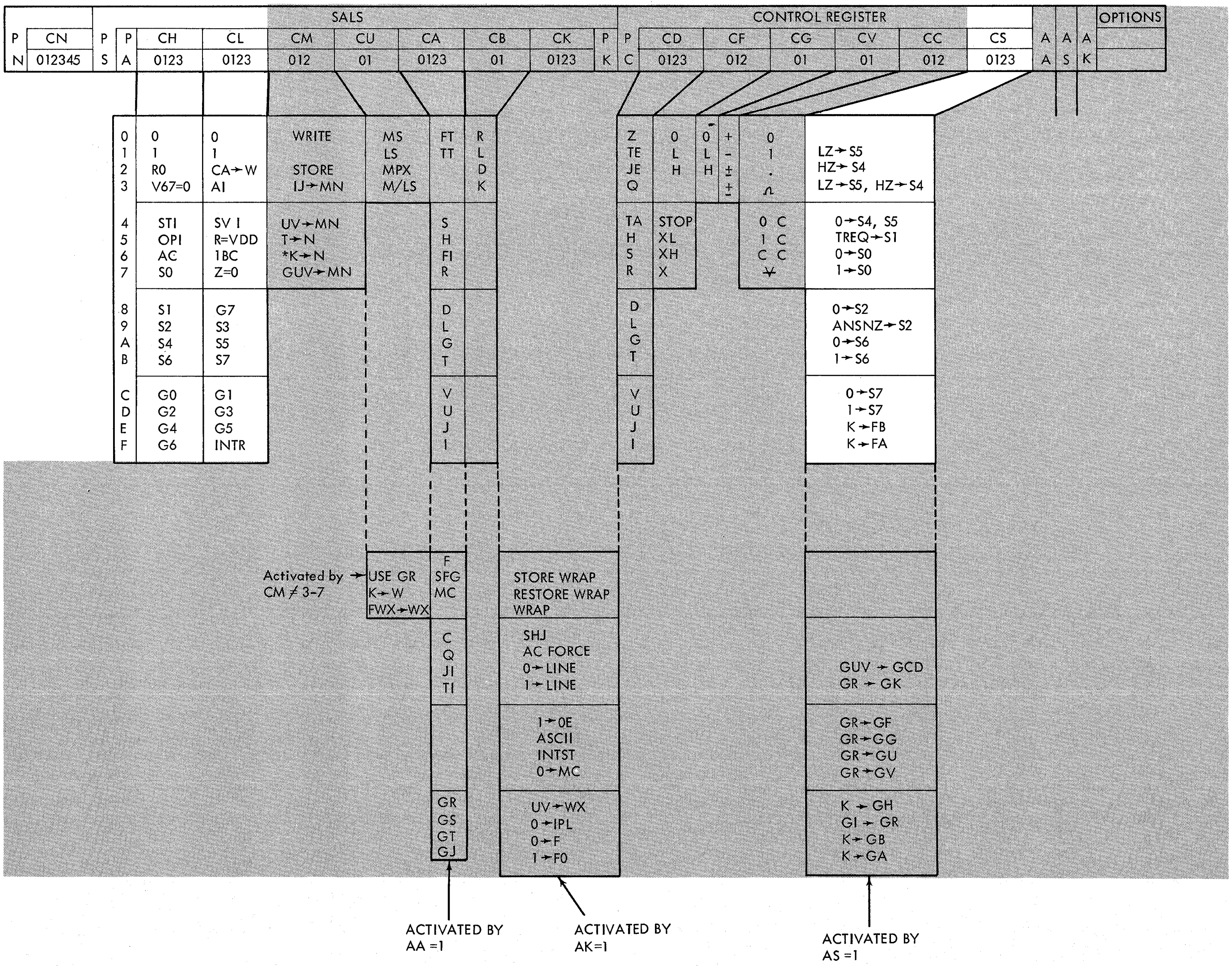


Figure 3-19. ROS Branch Control Fields

address bits, and their value is specified directly by the 6-bit CN field in the ROS word. The two low-order bit positions of the X register, X6 and X7, are called the branch bits, and their settings are controlled by the CH and CL fields, respectively.

The 4-bit CH and CL fields are decoded and switched against machine status conditions and/or the status of latches in the data flow registers G and S, to provide the two low- order bits of the address. The CS field controls the S-register. This field has alternate codings which control the selector channel registers.

AA, AS# Fields

- The 1-bit AA field if a 1 activates the alternate coding of the CA field.

- The 1-bit AS field if a 1 activates the alternate coding of the CS field.

The AA bit is a one, in conjunction with the mnemonic CA->W, to set the RCS address when in 1401 compatibility mode.

REVIEW QUESTICNS - MICRC PROGRAM

1. The series of logical steps executed by the CPU is called the __________.
2. The functional statement in the micro program is the __________.
3. The RCS word is divided into fourteen __________.
4. The fields may be divided into three groups: __________, __________, __________.
5. If a field is three positions wide, it can be decoded to __________ combinations.
6. The _____ controls for ALU are provided by the fields.
7. The RCS address is contained in the __________ and __________ registers.
8. Branching is controlled by the __________ and __________ positions of the RCS address.
9. The __________ field is often used as the source for constants.
10. The CM and CU fields provide the gates for __________ control.

RCS WORD FORMAT

Micro programs are written on CAS (Corporate Automation System) sheets. A micro program, made up of many RCS Words, is the cycle-by-cycle logical execution of a particular function. Each RCS Word is executed in a one microsecond ROS (Read Cnly Storage) cycle.

- ROS Words are written as an 8-line statement in box form (Figure 3-20).

- The top line of the box is the partial address of the RCS Word in bits and the actual address of the ROS Word in a 4-digit hexadecimal number.

- The second line (E) has an emitted value in bit form and the mode of data. The PK bit status may be indicated on this line.

- The third line (A) contains the arithmetic statement.

- The fourth line (S) is the core-control line.

- The fifth line (C) contains the status set or test statements.

- The sixth line (R) contains the controls for the RCS address.

- The seventh line (R) controls the ROS addressing branches.

- The eight line contains the page print position and the block serial number of the word.

```
  11-----------08A3|
E K=0101       DEC.|
A R+D→L            |
S IJ→MN        MS  |
C 0→S4, S5         |
R K→W              |
R G0           G1  |
                   |
  A5-----------BB  |
```

Figure 3-20. CLD Box Format

Line Designations

Each line of a RCS word block has its own function and will perform its function independent of the others. It is not necessary to have an entry on every line of the block, so each line is identified by alpha characters on the left (Figure 3-20).

Line One-Partial and Actual: The actual address is in 4-digit hexadecimal notation. The partial address, 11, is the low-order two bits of the actual address,08A3.

Line Two (E)-Emitted Value and Mode: This line is used if the CK field is used to emit a constant. The mode of the data is shown. Figure 3-20 shows the CK field to have the value of 5 and the mode of the data specified is decimal.

Line Three (A)-The Arithmetic Statement: This line is used for all arithmetic statements. An example is R + D->L. This statement adds the contents of the D-register to the R-register data and the result is stored in the L-register.

Line Four (S)-Storage Control: This line specifies the core-storage area being used and the operation desired. The expression IJ->MN MS gives a read call to Main Storage (MS) addressed by IJ.

Line Five (C)-Status or Test Controls: This line is mainly used for setting and resetting the bits of the S-register. The expression 0->S4, S5 resets both S4 and S5 to zero.

Line Six (R Top)-ROS Controls: This line has control of statements pertaining to the W-register and some test statements. The expression K->W sets the W-register to a value determined by the CK field. This value is specified on the E line of the CLD box.

Line Seven (R)-ROS Branch: This line checks conditions which determine the X6 and X7 positions for branching. Branching occurs, dependent on G0 and G1. If G0 and G1 are both set to one, a 1, 1 branch is taken. Line Eight (Print Position and Block Serial Number): The location (A5), on a CAS sheet, of this specific ROS word is shown on the bottom left. The block serial number (BB) is on the right. This number (BB) remains the same regardless of the location of the word on the page.

A group of ROS words in box form written on one sheet is commonly called a CAS logic diagram (CLD).

Symbols

Some common symbols used within a block are:

Symbol	Definition	Example
+	plus, add, true	A + B
-	minus, complement	A - B
=	equal	A = B
->	is set into	A ->B
●	And FUNCTION	A ● B->C
Ω	OR function	A Ω B ->C
V	Exclusive OR	A ∀ B ->C
,	and	A->B , C

Figure 3-21 contains the complete symbol list.

Symbol	Definition	
+	True Add	A + B
-	Complement	A - B
=	Equal	A = B
≠	Unequal	A ≠ B
→	Is set into	A → B
.	AND function	A · B → C
Ω	OR function	A Ω B → C
∀	Exclusive OR	A ∀ B → C
±	True-Complement	A ± B → C
,	and	A → B, C
/	or	A / B → C
<	Is less than	A<B
┐	not	
:	Is compared to	
?	Function not controlled by the micro program.	
( )	Used for normal English punctuation or to enclose an expression within a statement.	
*	Special - controlled by the user. A footnote at the bottom of the page explains the function.	

Figure 3-21. CLD Box Symbols

REVIEW QUESTIONS -ROS WORD FORMAT

1. A CLD box consists of __________ line statements.
2. If the CK field is used to emit a constant, the value is shown on the __________ line.
3. The A line contains the __________ statements.

4. The ROS branches are indicated on the __________ R line.
5. The symbol ± may be a __________ add or a __________ add.
6. The operation that is executed by question 5 is shown on the arithmetic line and is further controlled by the statement on the __________ line.
7. -> means __________.
8. The symbol for the exclusive OR function is __________.
9. Each ROS word is executed in a __________ ROS cycle.
10. __________ statement(s) can be executed during one cycle.

MICRO-PROGRAM EXAMPLES

BINARY ADD

You have seen the many parts that, put together, make up the micro-program. To tie these pieces together, lets's work our way through a micro program for a fixed point binary add.

The instruction for a binary add is written in RR format. Figure 3-22 shows the Op code for Fixed-Point Binary Add to be 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

		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 3-22. Op Codes

must be constructed by the micro program. As an example: To address the units position of general purpose register 7, the N-register must be set to 0111 0011. 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 micro-program. 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 micro program if an overflow is unmasked.

The CAS sheets as written by a micro programmer might appear as shown in Figures 3-23, 3-25, and 3-26. 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	=93

4. The data in register 7 is:

Byte 0	Byte 1	Byte 2	Byte 3	
00000000	00000000	00000000	10011001	=153

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 = 246. Using Figure 3-23 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.

Note: It will be wise to keep track of the data and addresses used during the micro program.

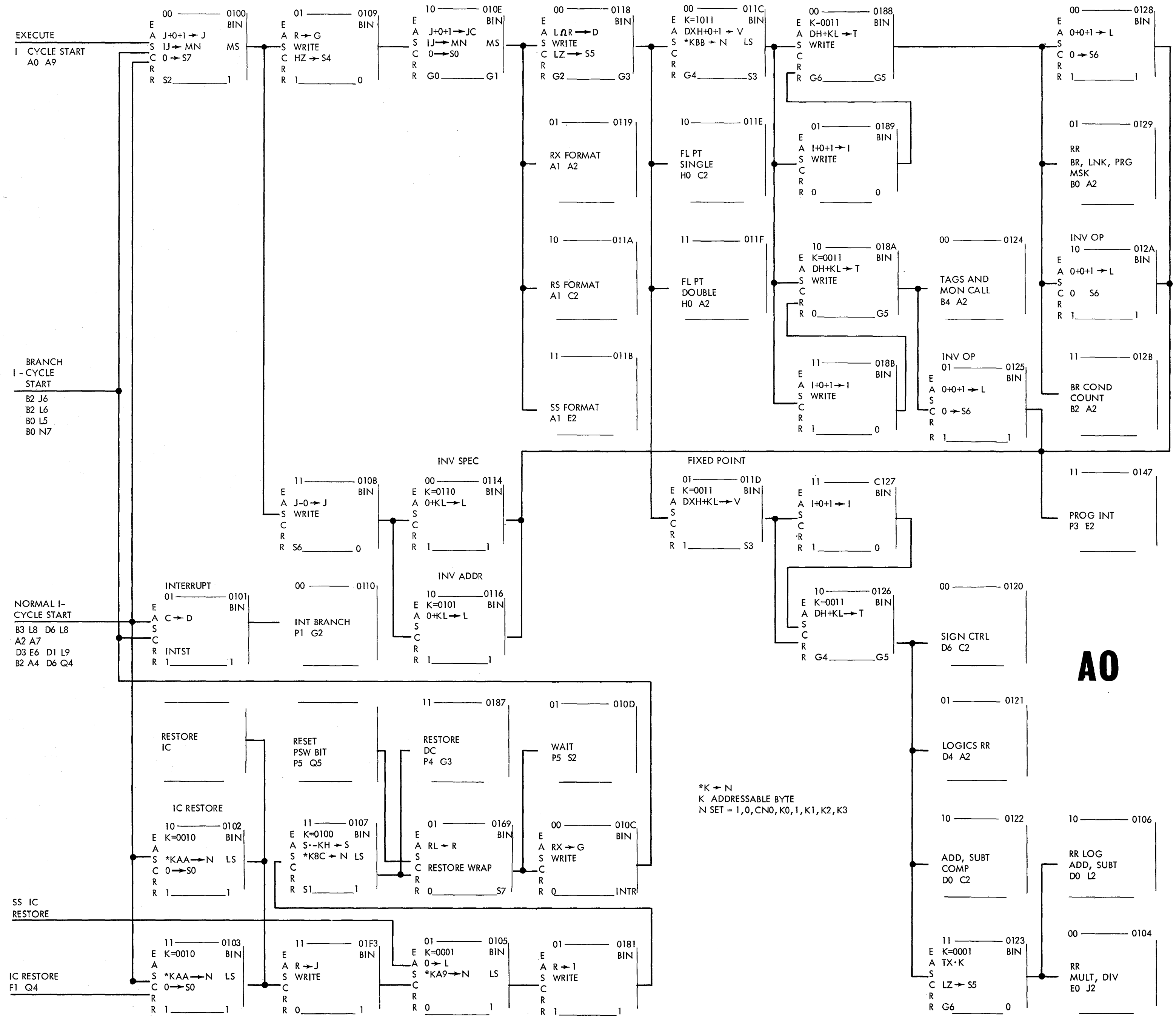


Figure 3-23. I-Cycle Start

ADDRESS 0100: The expression IJ->MN MS on the S line will bring 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 the value of one because all instructions start at an even address.

Assume that byte 510 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 511 it is only necessary to add the value of 1 to the J-register. The registers now contain

```
           I          J

        00000001   11111111.
```

Notice that, should the value of 1 be added to the J-register again, the resultant address in I and J is 00000001 00000000, or 256. 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. There 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 + C + 1 ->J. The A-register input of ALU IS SET WITH the data in the J-register. Zeros are set into the B-register. The 1 in the expression is a carry insert. The addition becomes:

```
     J data = A-register
   00000000 = B-register
  +       1 = Carry insert
  ----------
 J data + 1 = ALU output
```

The arithmetic statement can be thought of as being in the format: A-input + B-input + Carry is set into Destination. It is not necessary to have all portions of this format. The expression G->J, for example, brings up the control lines to set the G-register data to the A-register input. The B-register input and carry insert positions are zero.

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, which is many ROS words away. 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 micro program 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 it self. 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. Since position 2 of the S register is zero, 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, if you will recall, 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
```

the first byte of the instruction, S4 is not set. The four high bits are 0001. The S4 bit is interrogated in a Branch and Link routine and has no bearing on our example.

Because core readout is destructive readout, the information in the R-register is returned to core by the statement, WRITE.

A 1,0 branch is forced by the R line expression. If you look at the next address 010E, and convert the E to binary, 1110, you'll notice that the two low-order bits are 10.

ADDRESS 010E: Because the Op code is stored in the G-register, 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 expression 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, as a result of adding a one to the data in J, 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. Assume at this point there is a carryout, that S3 is set to a one, and let's see when and how this is handled by the micro program.

The C line of this 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 0 and 1 are interrogated by the expression G0, G1. See Figure 3-22. The data in the G-register is

```
 1     A

0001  1010
```

testing these positions tells us that our Op code must be in RR format. RX Op codes begin with 01, RS with 10, and SS with 11.

Because G0 and G1 are both zero, a 00 branch is taken to address 0118.

ADDRESS 0118: 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 3-22, we see that by checking G2 and G3 our Op code must now be in the right-hand column under FIXED POINT. The G2 and G3 positions set X6 and X7 to 01, which are the low-order bits of the next address to be executed, 011D.

Let's pause for a moment to 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 011D: Before any data from the general purpose registers can be added, the micro program must set up the units address of each register. The address for the low-order byte of general purpose register 5 is set up by the expression DXH + KL->V. See Figure 3-24. To address the low-order byte, the N-register must be set to:

```
Reg 5  Byte 3
0101   0011.
```

In the expression DXH + KL->V, 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:

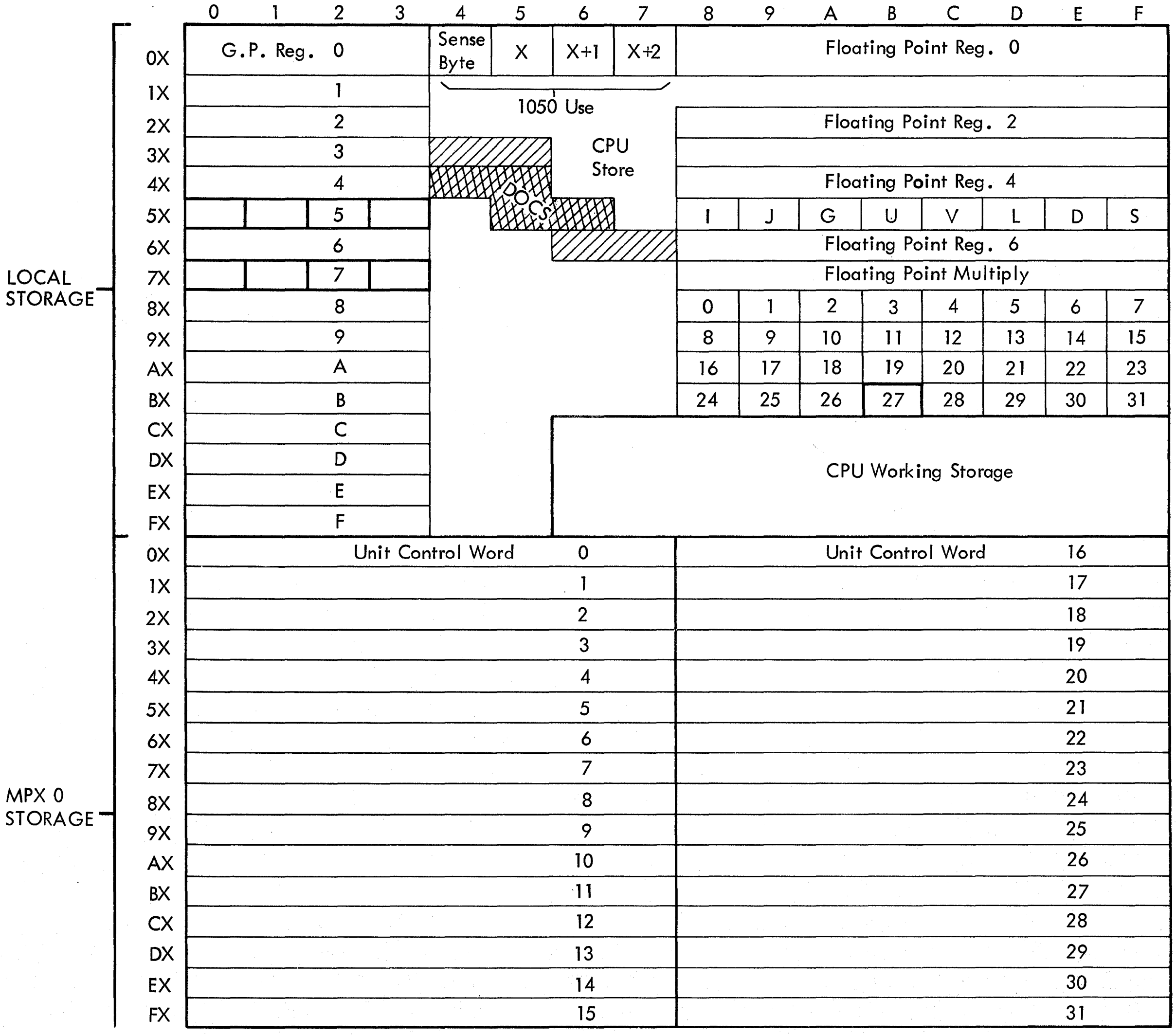


Figure 3-24. Auxiliary Storage Map

```
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 E 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.

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
```

On the R line the branch expressions are 1,S3. Remember, S3 was set to a one when the J-register was updated to indicate a carryout condition. This expression sets X6 and X7 to 1,1 and a branch is executed to address 0127.

ADDRESS 0127: The only function performed by this ROS word is I + 0 + 1->I Didn't we say earlier that when the J-register data had the value of one added to it and a carryout resulted, that the I-register must also be updated?

On the branch line, an unconditional 1,0 branch is taken to address 0126.

ADDRESS 0126: We have seen how the branch to this address from address 0127 was set up. We could have branched to this step from address 011D. Had S3 at address 011D not been set, a 1,0 branch would have been set up to this address.

The arithmetic statement DH + KL->T sets up the units address of register 7 in the T-register. Again let's 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 G-register positions 4 and 5 are now tested for further decoding of the Op code. G 4 is set to a 1. G 5 is set to a 0. Therefore, 1,0 branch is executed to address 0122. This address is in Figure 3-25 at coordinates C2. Our Op code, as a result of checking G 4 and G 5, must either be LOAD, COMPARE, ADD, or SUBTRACT.

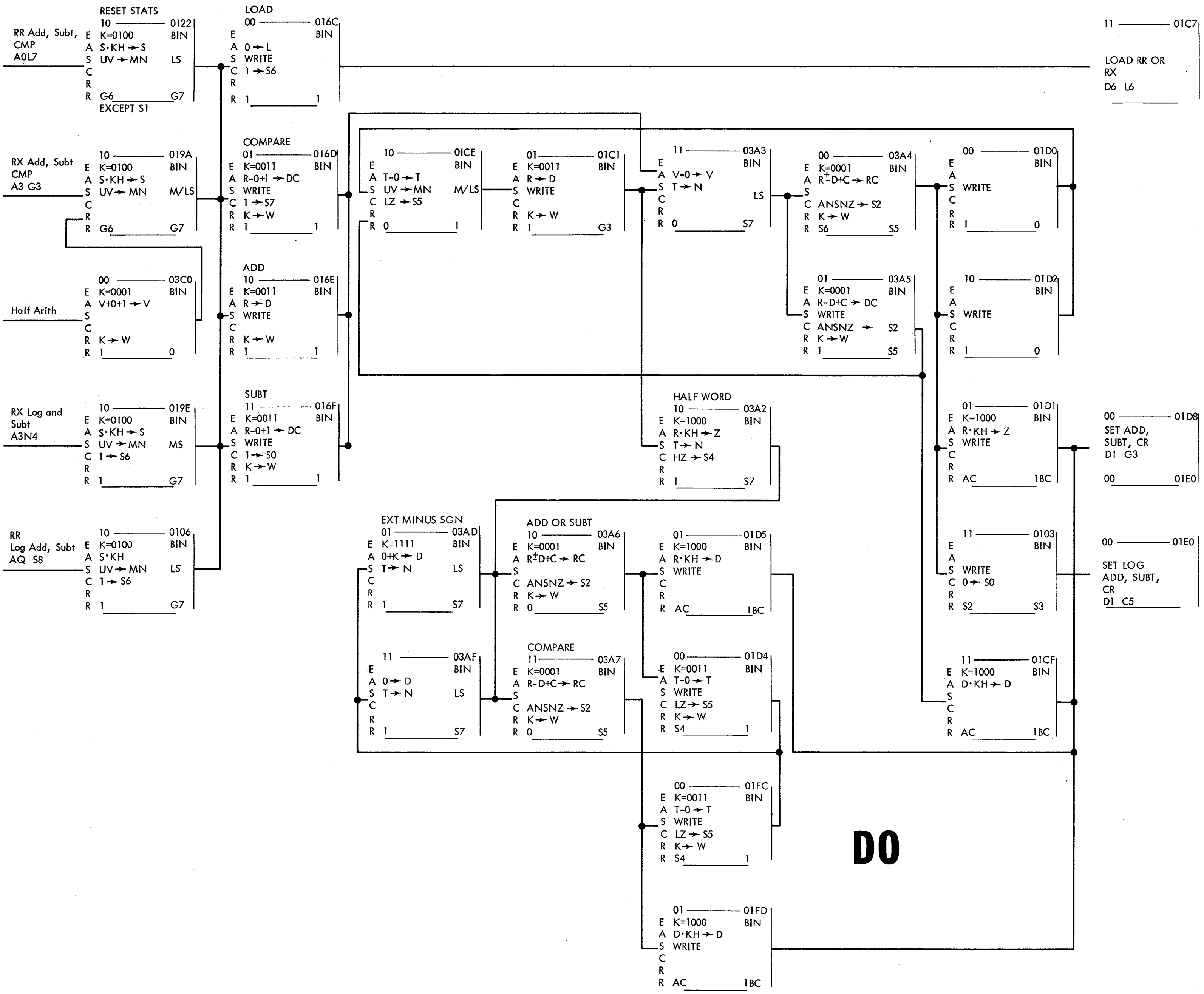


Figure 3-25. Fixed Point Add, Subtract, and Compare

ADDRESS 0122: The expression UV->MN LS addresses core to read out the first byte from general purpose register 5. The data read out is 01011101 (93). Local storage, rather than main storage, is specified by the LS portion of the expression.

The arithmetic S•KH->S sets to zero all positions of the S register except S1. To see how this is possible, let's assume for the moment that, in addition to the S3 position that has been set, the S1 position is also set to a 1. The • in the expression is the symbol for the AND function. The A source data (01010000) is set from the S-register. KH brings up control points to gate the high (H) portion of the B register data to ALU. Remember, the B register is set by the CK ROS control field.

```
S-reg. position  01234567

     S reg Set   01010000 =A source data
                 01000000 =B source data
-----------------------------------------
  Result of AND  01000000
```

Therefore, the result of the AND function is to allow S1 to remain set and to reset the other positions of the S-register.

The last two bits of the Op code are checked by the expression G6 G7. These last positions are set 1,0 and a branch is executed to address 016E. The micro program 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 016E: 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->D. 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 expression on the top R line is K->W. The W-register, if you will 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 E line of this block. To explain why, let's look at the branch line.

On the lower R line, an unconditional 1,1 branch is executed to address 03A3. 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 03XX, the expression K->W is used. Notice, the E line specifies the binary value of 3.

ADDRESS 03A3: The first byte of data from register 7 is read from core by the expression T->N 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. Let's see how ONE is subtracted by the expression V-0.

```
                         Reg 7  Byte 3
   V-register data     = 0111   0011

   minus 0           + = 1111   1111
----------------------------------------
                         Reg 7  Byte 2

   V-register result = 0111   0010
```

As you can see, some arithmetic statements should be worked out in detail less the wrong impression be assumed from just the reading of 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 RCS word address 016F figure location J3. S0, if you recall, 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 E line of the CAS block.

The result of the addition is:

```
                          R = +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 E 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 location E4.

Again, let's pause for a moment to review where the data is and the addresses in our 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 or 246

4. The S1 and S2 positions of the S register are 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 G-register determines that our 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. As you can see, 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 our Op code is still stored there.

ADDRESS 03A3:
We have been in this block before. You should begin to realize that we are in a small loop and will exit from it once all four bytes of data from the two registers have been added together.

The second byte from register 7 is read, (T->N LS).

The V-register address is changed to

Reg 7 Byte 1

0111 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 RCAR to the value of one because of the high-order address change.

ADDRESS 01D0: The second byte of 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 our 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 changed 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 changed 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 our answer: 000000000 000000000 000000000 11110110. The data in the R-register for our last sum is 000000000. 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 E line is shown to be 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
                   ↑↑
                   ||
                   ||
                   ||
                   |└---1BC
                   |
                   |
                   └----AC
```

Let's continue with our example. 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 3-26, location G3.

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 *KBB->N LS addresses a K addressable byte of local storage. See Figure 3-24. This byte contains the condition codes and program mask bits. Certain bits are set according to whether the answer is equal to, greater, or less than zero.

Program masks are checked further in the program.

A K-addressable byte sets the N-register to the format:

```
   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. You can see by looking at Figure 3-24 that the coordinates BB address byte 27. BB in binary is:

```
   B     B
   1011  1011
```

Match this up with the address format and you see that the CK field must be coded 1011. This is the value that appears on the E line of the CAS block.

```
  Format = 1, 0, CN0, K0, 1, K1, K2, K3,
  Value BB = 1  0  1  1  1  0  1  1
```

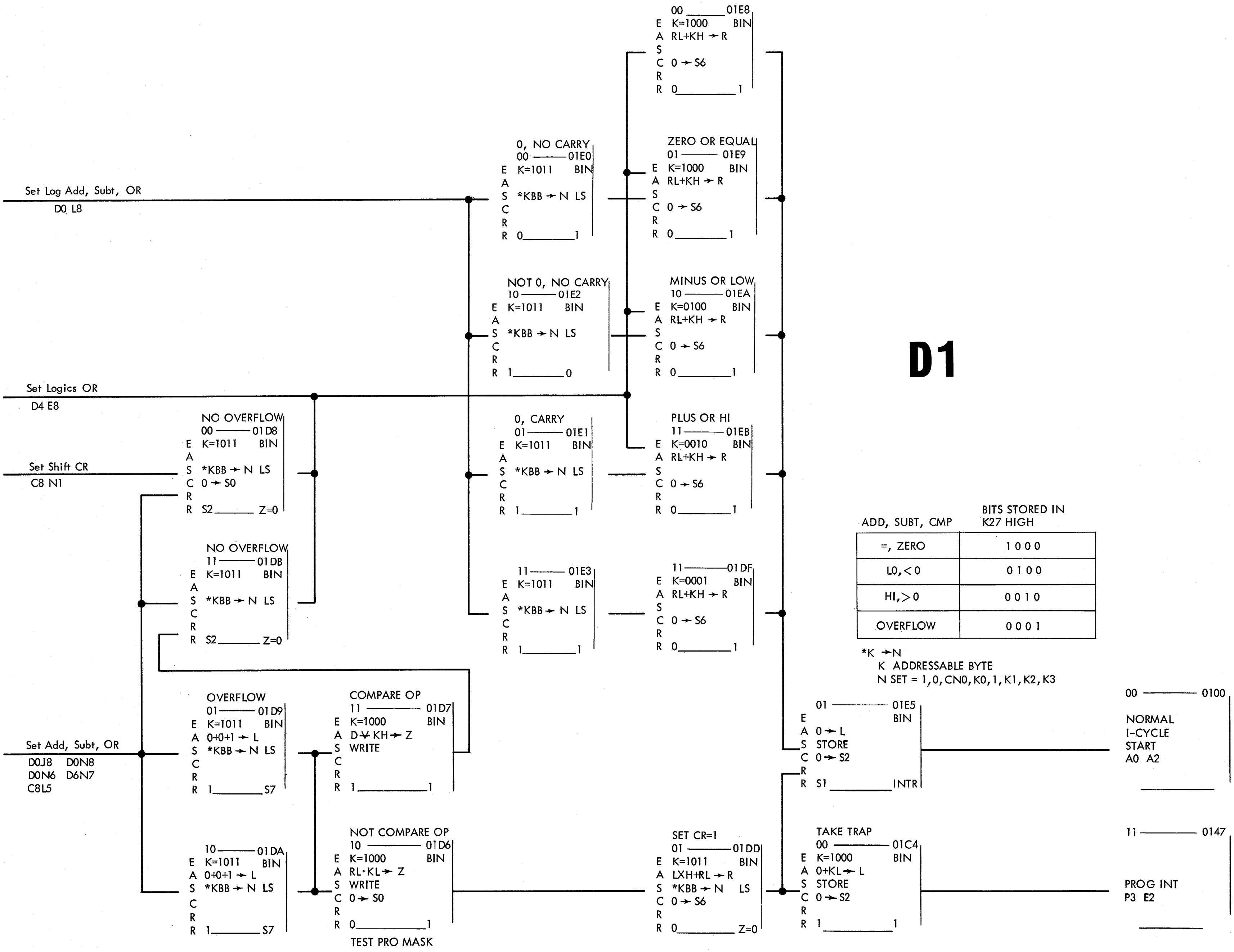
The control line expression 0->S0 sets S0 to zero in case we had been in a complement operation.

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 therefore, 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.

ADDRESS 01EB: The byte that was 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 E. 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. Notice the table in Figure 3-26.



ADD, SUBT, CMP	BITS STORED IN K27 HIGH
=, ZERO	1 0 0 0
L0, < 0	0 1 0 0
HI, > 0	0 0 1 0
OVERFLOW	0 0 0 1

*K → N
K ADDRESSABLE BYTE
N SET = 1, 0, CN0, K0, 1, K1, K2, K3

Figure 3-26. Set Condition Register

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 which consists of our new condition code and 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 INTR, interrupt. S1 is not set. If there is an interrupt a 0, 1 branch is taken to address 0101, Figure 3-23 location L2. If no interrupt exists a 0,0 branch is executed to address 0100, Figure 3-23 location A2. Address 0100 is the ROS word where we began. Address 0101 is the beginning of a micro program to test the interrupt and determine what it is (selector channel, multiplex, etc.).

Our problem is finished, but to gain more experience in micro programming let's assume that the result of adding the data of the two registers <u>did</u> produce an overflow condition.

Start at address 01D9 in Figure 3-26.

ADDRESS 01D9: The K addressable byte is read (*KBB->N 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. Let's assume the data in R is xxxx 1 yyy. The x positions are those for the condition code. The 1 means that this position is set. The y positions are the remaining program mask bits.

The data for this expression is:

```
 A source data (RL) =   0000  1000
 B source data (KL) =   0000  1000
----------------------------------------

   ANDed ALU output =   0000  1000
```

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: *KBB->N 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 1 yyy. Cnly the low portion is presented to ALU (RL). The result of the addition becomes:

```
A source =           0001  0000
B source =           0000  1yyy
----------------------------------------
R-register set =     0001  1yyy
```

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. Cur output was 0000 1000 as a result of the expression RL • KL->Z. Therefore, Z=0 does not set X7 to a 1. A 0,0 branch is taken to address 01C4 because the overflow was masked on.

ADDRESS 01C4: The data in the R-register is returned to core (STORE). The four highest bits are the new condition code:

```
High    Low

0001    1yyy.
```

0001 is the coding for an overflow. The L-register is set 0000 1000 by the expression 0 + KL->L. The S2 position is set to zero and a 1,1 branch is taken to a program interrupt routine. This completes the branch on overflow ROS words.

INDEXING

Let's try a final example. Assume that the Op code is in this format:

```
Op code  R1   X2   B2        D2
   X    0011 0100 1111 0000 0110 0100
```

The objective in working through the ROS words in Figure 3-27 is to arrive at an address in the UV-registers. This is done by adding the data in the general purpose registers, as specified by X2 and B2, to the binary number specified by the D2 field. The facts that you will need before starting are these:

1. General purpose register 4 (X2) contains the value of 40 in binary.

2. Register 15 (B2) contains the value of 1860 in binary.

3. The L- and S-registers are zero.

4. The R-register contains

```
R1      X2

0011    0100
```

5. The IJ-registers contain the address of the third byte of the instruction, B2 and the high four bits of D2.

Note: While working this problem it will be helpful if you keep track of the data in the R,T,U,V, and S-registers. All reference is made to the actual address of the ROS word which appears in the upper right corner of each block. To begin, see Figure 3-27 address 0119: at A2.

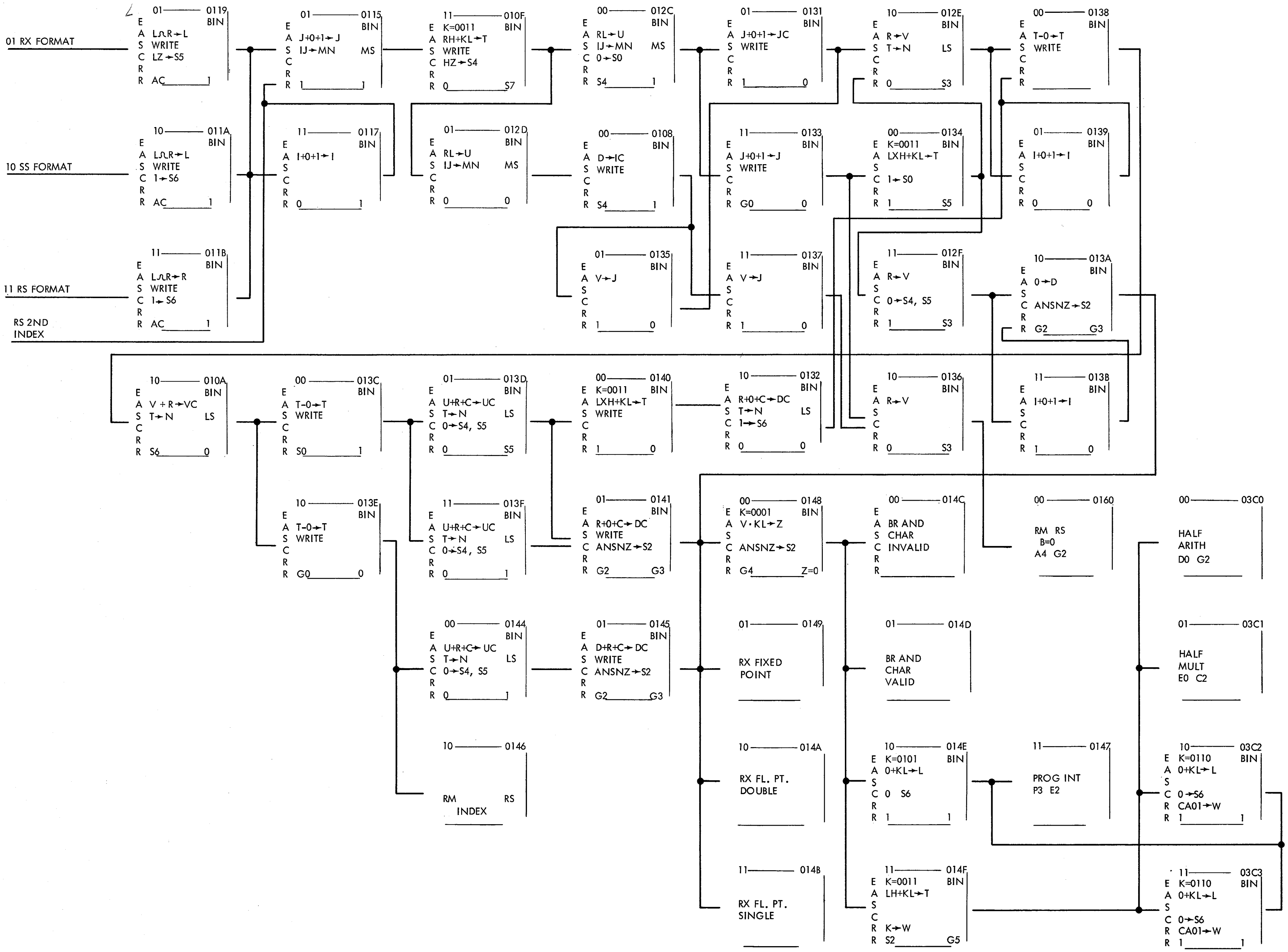

01 RX FORMAT
10 SS FORMAT
11 RS FORMAT
RS 2ND INDEX
01 0119 BIN L⊓R→L WRITE LZ→S5 AC 1
10 011A BIN L⊓R→L WRITE 1→S6 AC 1
11 011B BIN L⊓R→R WRITE 1→S6 AC 1
01 0115 BIN J+0+1→J IJ→MN MS 1 1
11 0117 BIN I+0+1→I 0 1
11 010F BIN K=0011 RH+KL→T WRITE HZ→S4 0 S7
01 012D BIN RL→U IJ→MN MS 0 0
00 012C BIN RL→U IJ→MN MS 0→S0 S4 1
00 0108 BIN D→IC WRITE S4 1
01 0135 BIN V→J 1 0
01 0131 BIN J+0+1→JC WRITE 1 0
11 0133 BIN J+0+1→J WRITE G0 0
11 0137 BIN V→J 1 0
10 012E BIN R→V T→N LS 0 S3
00 0134 BIN K=0011 LXH+KL→T 1→S0 1 S5
11 012F BIN R→V 0→S4, S5 1 S3
00 0138 BIN T-0→T WRITE
01 0139 BIN I+0+1→I 0 0
10 013A BIN 0→D ANSNZ→S2 G2 G3
10 010A BIN V + R→VC T→N LS S6 0
00 013C BIN T-0→T WRITE S0 1
10 013E BIN T-0→T WRITE G0 0
01 013D BIN U+R+C→UC T→N LS 0→S4, S5 0 S5
11 013F BIN U+R+C→UC T→N LS 0→S4, S5 0 1
00 0144 BIN U+R+C→UC T→N LS 0→S4, S5 0 1
10 0146 RM RS INDEX
00 0140 BIN K=0011 LXH+KL→T WRITE 1 0
01 0141 BIN R+0+C→DC WRITE ANSNZ→S2 G2 G3
01 0145 BIN D+R+C→DC WRITE ANSNZ→S2 G2 G3
10 0132 BIN R+0+C→DC T→N LS 1→S6 0 0
10 0136 BIN R→V 0 S3
11 013B BIN I+0+1→I 1 0
00 0148 BIN K=0001 V·KL→Z ANSNZ→S2 G4 Z=0
01 0149 RX FIXED POINT
10 014A RX FL. PT. DOUBLE
11 014B RX FL. PT. SINGLE
00 014C BR AND CHAR INVALID
01 014D BR AND CHAR VALID
10 014E BIN K=0101 0+KL→L 0 S6 1 1
11 014F BIN K=0011 LH+KL→T K→W S2 G5
00 0160 RM RS B=0 A4 G2
11 0147 PROG INT P3 E2
00 03C0 HALF ARITH D0 G2
01 03C1 HALF MULT E0 C2
10 03C2 BIN K=0110 0+KL→L 0→S6 CA01→W 1 1
11 03C3 BIN K=0110 0+KL→L 0→S6 CA01→W 1 1

Figure 3-27. Indexing

ADDRESS 0119: We began the problem at this particular ROS word because a look at the construction of our Op code shows us that it is in RX format. The second byte of data which is contained in the R-register

```
        R1      X2

        0011    0100
```

is regenerated by the WRITE expression. This data is also stored in the L-register by the expression LΩR->L. Ω The data in the L register, zeros, is ORed (Ω) with the data in the R-register,

```
        R1      X2

        0011    0100
```

And the result is set into (->) the L-Register.

The expression LZ->S5 does not set position 5 of the S-register to a 1 because the low portion of the Z bus$_{(LZ)}$ has the data:

```
                X2

                0100
```

The branch controls are AC and 1. AC, ALU carry, is the mnemonic that brings up control lines to check for an ALU carry that may have occurred as a result of the arithmetic statement in the previous ROS word. Had there been an ALU carry, a 1,1 branch would be taken to address 0117 where the only function performed is to update the I-register, I+0+1->I. Assume that there was not an ALU carry and that the next address in sequence is address 0115.

ADDRESS 0115: The third byte of the instruction, B2 and the high-order four bits of D2, is read by the expression IJ->MN MS. The MN register is set by the I- and J-registers. MS specifies main storage rather than local storage.

The J-register is updated by the expression J+0+1->J. The data in the J-register sets the A-register input of ALU. The B-register input to ALU is set to zero. A carry is inserted. The result of the addition is set into the J-register (->J)

```
A-Register = xxxx xxxx (J-Reg data)
B-Register = 0000 0000
Carry Insert         1
ALU Output = xxxx xxxx +1
```

An unconditional 1,1 branch is taken to address 010F.

ADDRESS 010F: The third byte of data is regenerated (WRITE). The data in the R-register is

```
          B2      D2 High

          1111    0000.
```

Because our objective is to arrive at an address using data in general purpose registers, it is necessary to obtain the units address of each register. The expression RH + KL->T constructs the units address of register 15, specified by the B2 portion of our instruction. The first portion of this expression, RH, sets the A-register input to ALU from the data in the R-register, but gates as A source data only the high portion (H). The B-register is set by the value in the CK ROS control field (K). This value is 0011, as shown on the E line of this box. The B-register is set 0011 0011. However, only the low portion (L) is presented to ALU as B source data. The data is:

```
                    B2
   A source     = 1111 0000
 + B source     = 0000 0011
----------------------------------------
               Reg 15 Byte 3

   T-register   = 1111 0011
```

The expression HZ->S4 does not set position 4 of the S register to a one because the high portion of the data on the Z bus (HZ) is not all zeros.

The branch controls are 0,S7. One of the facts assumed in starting this micro program was that the S- register was zero. Therefore, a 0,0 branch is taken to address 012C.

ADDRESS 012C: The fourth and last byte of the instruction is read by the expression IJ->MN MS.

The data in the low portion of the R-register is set into the U-register by the expression RL->U. The U-register now contains

D2	High
0000	0000

Position zero of the S-register is set to zero (0->S0).

Position 4 of the S-register was not set. The branch controls set up a 0,1 branch to address 0131.

ADDRESS 0131: The last byte read,

D2	Low
0110	0100

is regenerated (WRITE). The J-register is again updated (J+0+1->JC). The C to the right of the arrow allows a carry out, as a result of adding the value of one to the data in J, to set position 3 of the S-register to a one. S3 is the carry control position of the S-register. If there is a carry out as a result of this arithmetic statement it indicates that the I register must also be updated.

An unconditional 1, 0 branch is executed to address 012E.

ADDRESS 012E: The data in the R-register,

D2	Low
0110	0100

is stored in the V-register by the expression R->V.

The T-register is used to set the N-register and the first byte from general purpose register 15 is read by the expression T->N LS. LS specifies local storage.

The branch controls are 0, S3. Had there been a carryout as a result of the arithmetic expression in the previous ROS word, a 0,1 branch would be taken to address 0139 wherein the I-register is up dated (I+0+1->I). Assume that S3 is a zero so that a 0,0 branch is taken to address 0138.

ADDRESS 0138: The first byte of data read from general register 15 is regenerated (WRITE). The R-register contains 0100 0100 as data. One of the facts given was that register 15 contained the value of 1860 in binary this is:

Byte 0	Byte 1	Byte 2	Byte 3
00000000	00000000	00000111	01000100

The T-register data is changed to:

Reg 1	Byte 2
1111	0010 (T-0->T).

An unconditional 1,0 branch is taken to address 010A location G2.

ADDRESS 010A: The data from byte 2 of general purpose register 15 is read by the expression T->N LS. This data is 0000 0111.

The data in the V-register is added to the data in the R-register, with the result stored in the V-register by the arithmetic expression V+R->VC. The C to the right of the arrow allows a carryout as a result of adding V+R to set S3 to a one. The addition of data is:

V-reg = 0110 0100 = D2 low
R-reg = 0100 0100 = Reg 15 Byte 3 data.

V-Reg result 1010 1000

There is no carryout as a result of this addition. Therefore, S3 is not set to a one.

On the branch line a test is made on S6. S6 has never been set to one, thus a 0,0 branch is taken to address013C.

ADDRESS 013C: The data from GP-register 15, byte 2, is regenerated (WRITE). Again, the data in the T-register is changed so that the T-register now contains

Reg 15	Byte 1
1111	0001. (T-0->T).

S0 has not been set to a 1. Branch to address 013D.

ADDRESS 013D: The expression U+R+C->UC causes the data in the U-register to be added to the data in the R-register, with a carry insert (C) under control of S3. The resultant answer will be stored in the U-register and a carryout, if there is one is allowed to set S3. S3 has not been set previously, therefore there is no carry insert. The data is:

```
                D2 High

 U-reg  = 0000 0000
 R-reg  = 0000 0111 = Reg 15 Byte 2 data
----------------------------------------

 U-reg

 result = 0000 0111
```

S3 will remain 0 as there is no carryout.

Local storage is again addressed by the expression T->N LS. The data read is 00000000. This is the data from byte 1 of register 15.

Positions 4 and 5 of the S-register are set to zero (0->S4, S5).

S5 has never been set to one, so the 0,0 branch is taken to address 0140.

ADDRESS 0140: The data from byte 1 of register 15 is regenerated (WRITE).

The expression LXH + KL->T sets up the units byte address of general purpose register 4 in the T-register. Register 4 is specified by the X2 portion of our instruction. Remember, the L-register contains

```
R1      X2

0011    0100.
```

The A-register input to ALU is set from the L-register. The data in the A-register is

```
High    Low

0011    0100.
```

This data is crossed (X) and only the high portion (H) is used as A source data, 0100 0000.

In the portion of the expression, KL, the B-register input to ALU is set from the CK ROS control field (K). The value in the CK field is specified on the E line. The data in the B-register is therefore, 0011 0011. Only the low portion (L) is used as B source data. The data becomes:

```
 A source data    = 0100    0000
 B source data    = 0000    0011
----------------------------------------
                    Reg 4   Byte 3

 T-register data  = 0100    0011
```

The next ROS word is at address 0132.

ADDRESS 0132: The expression R+0+C->DC transfers the data in the R-register to the D-register. The D-register data is 0000 0000, which comes from register 15 byte 1. No carries are involved.

Again, core is read (T->N LS). This causes the first byte of data to be read from register 4. This data is 00101000 because one of the facts of our problem specified that this register contained the value of 40 in binary.

The sixth position of the S-register is set to 1 (1->S6).

The next step is address 0138 at location A8.

ADDRESS 0138: The data just read is regenerated (WRITE). The T-register is changed to obtain the address of the next byte of information.

```
Reg 4    Byte 2

0100     0010
```

is the data in the T-register as a result of the expression T-0->T.

The next ROS word in sequence is at address 010A.

ADDRESS 010A: The data in the V-register is added to the data in the R-register. If a carryout occurs as a result of the addition, it is allowed to set S3 (V + R->VC). The data added is:

```
      V-register data    = 10101000
      R-register data    = 00101000
----------------------------------------
      V-register result = 11010000
```

There is no carryout. S3 remains zero.

The reading of core is controlled by T->N LS. This causes the data from register 4 byte 2 to be read. This data is 0000 0000.

The branch controls are S6 and 0. S6, if you will recall, was set to a one at the ROS word at address 0132. Because of this, a 1, 0 branch is taken to address 013E.

ADDRESS 013E: The last byte of data that was read is regenerated (WRITE).

The T-register data is again changed (T-0->T). The result in the T-register is now

Reg 4	Byte 1
0100	0001.

Since we did not assign a particular Op code to our problem, the G-register settings are unknown. Assume that G0 is a zero and advance to the ROS word at address 0144.

ADDRESS 0144: The expression U + R + C->UC brings up control lines to add the data in the R-register to the data in the U-register with a carry insert. Also, the result of the addition is set in the U-register and a carryout, if there is one, is allowed to set S3. The data for this operation is:

```
U-reg  = 0000 0111 D2 high plus B2
R-reg  = 0000 0000 X2 byte 2
-----------------------------------------
Result = 0000 0111
```

After the completion of this arithmetic statement, the U and V- contain an address that was developed by adding a binary value, as specified by D2 in our instruction, to the data contained in two general purpose registers, as specified by X2 and B2 in our instruction. The U-register and V-register data is

U-reg	V-reg
00000111	11010000

or 2000. The value of the D2 portion of our instruction is 100. To this was added the value in register 4 (X2) and register 15 (B2). Register 4 contained the value of 40, register 15 the value of 1860.

As you have noticed, each function performed in a ROS word can be readily determined. To work through any given problem you must keep track of the data and addresses involved. The most important register to be aware of is the S-register.

SPECIAL STATEMENTS

0 ±->Z: Decimal mode is specified on the E line. 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.

J ±->Z: Decimal mode is specified on the E line. The wait latch is set on and the ROS word that contains this statement is continually executed until an interrupt occurs.

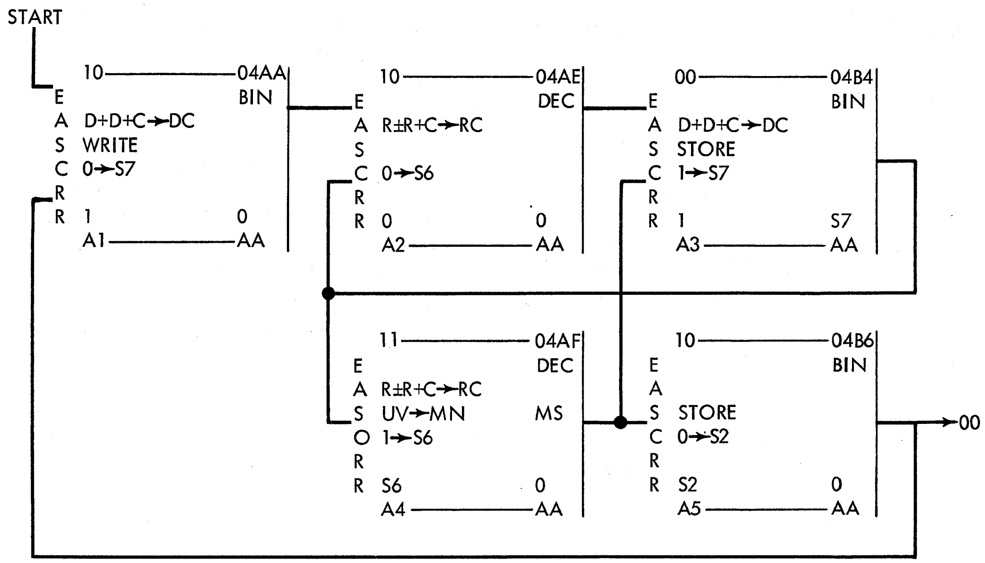


Figure 3-28. Micro Program Review

1. Using Figure 3-28 determine the data in the R-register after leaving block 04B6 the second time. The conditions to start are:

   a. The UV registers contain the address XXXX.

   b. The D-register contains the value of 17 in binary 0001 0001.

   c. The S-register is set to zero except for S2, which is set to a 1.

   d. The R-register data is zero.

CONTROL FIELD MNEMONICS

FUNCTION CONTROL MNEMONICS

The mnemonics that provide the source data and control for ALU are shown in Figure 3-29. The numbers and letters to the left in the figure are the coding designations for each mnemonic. As an example: The coding for the mnemonic FI in the CA field is 0110, 6. The CA field would also be punched 0110, where 0 is the punched-out capacitor plate and 1 is the representation of the plate left intact.

Some of these mnemonics need further explanation.

ROS CONTROL FIELD MNEMONICS

	CA		CB		CK		CD		CF		CG		CV		CC	
	MENMONIC	DESCRIPTION	MNEMONIC	DESCRIPTION	MNEMONIC	DESCRIPTION	MNEMONIC	DESCRIPTION	MNEMONIC	DESCRIPTION	MNEMONIC	DESCRIPTION	MNEMONIC	DESCRIPTION	MNEMONIC	DESCRIPTION
0	FT	MX TAGS IN	R				Z	Z BUS	0	BLOCK GATE	0	BLOCK GATE	+	ADD	0	BLOCK CARRY
1	TT	1050 TAGS IN	L				TE	1050 BUS OUT	L	GATE LOW	L	GATE LOW	-	SUBTRACT	1	INSERT CARRY
2			D				JE	DIRECT DATA CHANNEL BUS OUT	H	GATE HIGH	H	GATE HIGH	±	BINARY ADD SUBTRACT	•	AND FUNCTION
3			K				Q	MEMORY PROTECT REGISTER		GATE ALL DATA		GATE ALL DATA	±	DECIMAL ADD SUBTRACT	Ω	OR FUNCTION
4	S						TA	1050 TAGS OUT	● STOP	MACHINE STOP					0 C	NO CARRY INSERT ALLOW CARRY OUT
5	H						● H		XL	CROSS LOW					1 C	INSERT CARRY ALLOW CARRY OUT
6	FI	MX BUS IN					S		XH	CROSS HIGH					C C	CONDITIONAL CARRY IN ALLOW CARRY OUT
7	R						R		X	CROSS					∀	EXCLUSIVE OR
8	D						D									
9	L						L									
A	G						G									
B	T						T									
C	V						V									
D	U						U									
E	J						J									
F	I						I									

ALTERNATE CODINGS

	MNEMONIC	DESCRIPTION	MNEMONIC	DESCRIPTION
0	F	F REGISTER EXTERNAL INTR.		
1	SFG	SWITCH F and G	●STORE WRAP	GATE SET OF WRAP BUFFER LATCH
2	MC	MACHINE CHECK REGISTER	● RESTORE WRAP	GATE WRAP LATCH
3			● WRAP	TEST WRAP LATCH
4	C	INTERVAL TIMER COUNTER	SHJ	SWITCHES H and J
5	Q	MEMORY PROTECT REGISTER	● AC FORCE	FORCE X-REGISTER TO ZERO ON ALU CARRY
6	JI	DIRECT DATA CHANNEL BUS IN	0 → LINE	RESET LINE LATCH (1050)
7	TI	1050 BUS IN	1 → LINE	SET LINE LATCH (1050)
8			● 1 → 0E	FORCE ALU CHECK
9			● ASCII	CHECK ASCII LATCH
A			● INTST	TEST INTERRUPT
B			0 → MC	RESET MACHINE CHECK
C	GR	SELECTOR CHANNEL R-REGISTER	UV → WX	GATE UV REGISTER TO WX REGISTER
D	GS	SELECTOR CHANNEL STATUS REGISTER	0 → IPL	RESET LOAD LATCH
E	GT	SELECTOR CHANNEL TAGS IN	0 → F	RESET F REGISTER
F	GJ	SELECTOR CHANNEL GJ ASSEMBLER	1 → F0	SET POSITION 0 OF F REGISTER

ACTIVATED BY THE AA BIT BEING A ONE

ACTIVATED BY THE AK BIT BEING A ONE

● EXPLAINED IN THE TEXT

Figure 3-29. Function Control Mnemonics

AC Force: If there was a carryout of ALU on the previous ROS word, the X-register is forced to zero, which causes a branch-to-location 00, regardless of the normal next address determined by the CN, CH, and CL fields.

ASCII: The CH and CL fields normally control the X6 and X7 positions of ROAR. The ASCII statement tests the ASCII latch and if this latch is on, which specifies ASCII mode, the CH field loses control of the X6 position and instead a 0X7 branch is taken. X7 is still controlled by the CL field. If the ASCII latch is off, the X6 branching remains under control of the CH field.

The ASCII latch is set by a particular statement. Refer to the section for SPECIAL Statements.

H: When the H-Register is specified by the CD field coding 0101, the priority-reset-control latch is set on. This latch, when on at T3 time, turns the priority latch off so that priorities may be recognized.

INTST: This mnemonic, alternate CK decode 1010, tests for interrupts and sets X6 and X7 according to the interrupt that has been stacked.

Type of Interrupt	X6	X7
Timer or External	0	0
Selector Channel 1	1	0
Selector Channel 2	0	1
Multiplexor Channel	1	1

See Figure 3-30. This mnemonic is used at the same time that the CH and CL fields are coded 0001. This coding in each field tries to set X6 and X7 to one. The OR's in this figure may be thought of as inhibit control. In the case of a timer interrupt, the X6 and X7 AND's are blocked, and therefore X6 and X7 are set to zero. Should either AND be satisfied it sets its respective position of ROAR to a one. Assume that there had been a timer interrupt.

OR number 2 in the X6 circuit is not satisfied. The output from this OR then blocks the X6 AND. OR number 6 in the X7 circuit is also dissatisfied and blocks the X7 AND. With both AND's blocked, a 0, 0 branch is set up.

If we now assume that there is a selector channel 1 interrupt, the X6 AND must be satisfied and the X7 AND dissatisfied. The bottom input to the X6 AND is satisfied by the CH field coded for the forced one. The next input, from OR number 4, is satisfied because we are not testing for a memory wrap. OR number 3 is satisfied because of the selector channel 1 line. OR number 2 is satisfied because we have neither a timer nor an external interrupt.

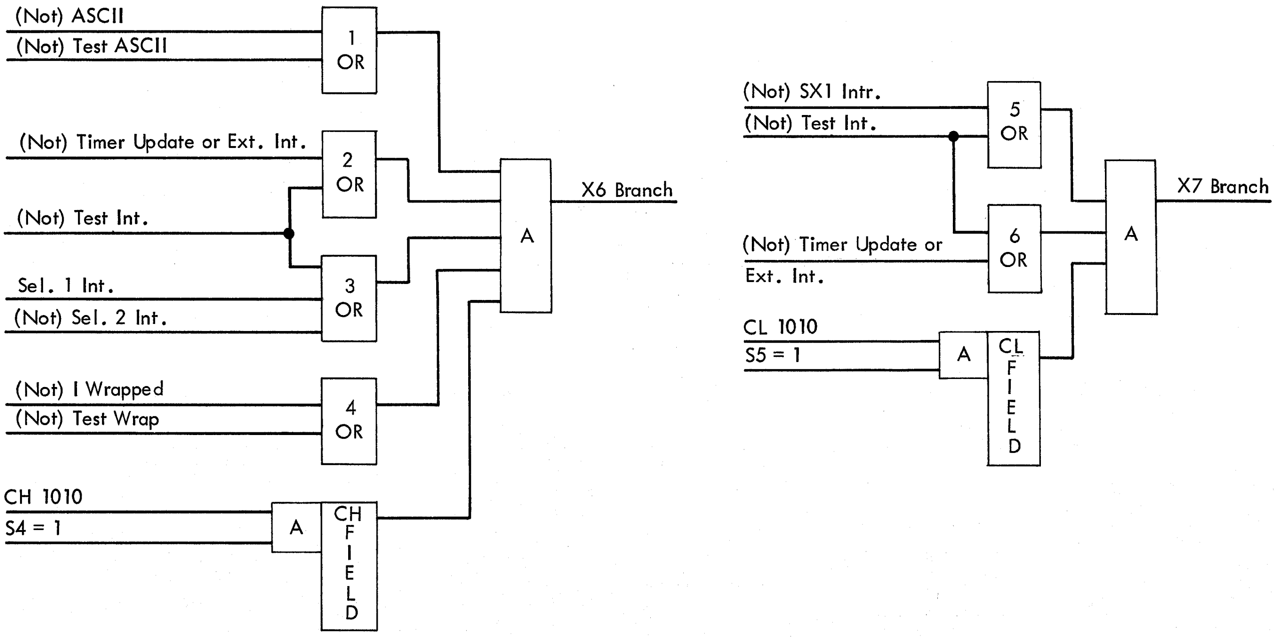


Figure 3-30. Test Interrupts

Lastly, OR number 1 is satisfied because we are not testing to see if we are in ASCII mode. The X7 AND is dissatisfied because OR number 5 is dissatisfied. We are testing interrupts, and our assumption was that we had a selector channel 1 interrupt.

RESTORE WRAP-STORE WRAP-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 I- or U-register position as a result of updating the address.

If the IJ registers are used to address core, a memory wrap condition sets the I-wrap latch. It is sometimes necessary to remember the status of this latch. The mnemonic STORE WRAP will gate the status of the I-wrap latch to the wrap-buffer latch.

When it is again necessary to determine whether there had been a wrap, the mnemonic RESTORE WRAP will gate the status of the wrap-buffer latch back into the I-wrap latch.

The mnemonic WRAP tests the status of the I-wrap latch, and if this latch is on a 0, X branch is taken. If it is off, the X6 portion of the branch is still controlled by the CH field.

A wrapped condition can also occur on a 8k, 16k, or 32k unit. This is detected by testing the high-order positions of the M-register.

1->OE: This mnemonic is used in diagnostic testing. The first time this expression appears, it forces bad parity by blocking the +L Z bus 0 and +L Z BUS 4 lines. The next time this expression is used in the program, it forces an ALU check by forcing all the minus SUM lines and the minus carry 0-bit line to plus L levels. See Figure 3-31. The odd-even-control latch is turned on (EVEN) by the decoding 1->OE, T2 and the introduce-latch set OFF. A circuit to gate the +L Z bus 0 and +L Z bus 4 lines 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 line so that the lines are all plus and an ALU check is forced. The odd-even-control latch is turned off three ways: machine reset, reset load line, (which is developed when the mnemonic LOAD 0->IPL is used), or at T2 time when the introduce-ALU-check latch is on.

STOP: This mnemonic appears in the CLD box as STOP or S STOP. The CF field coding, 100, causes the stop latch to be turned on at T4 time, Figure 3-32. The output from this latch feeds two

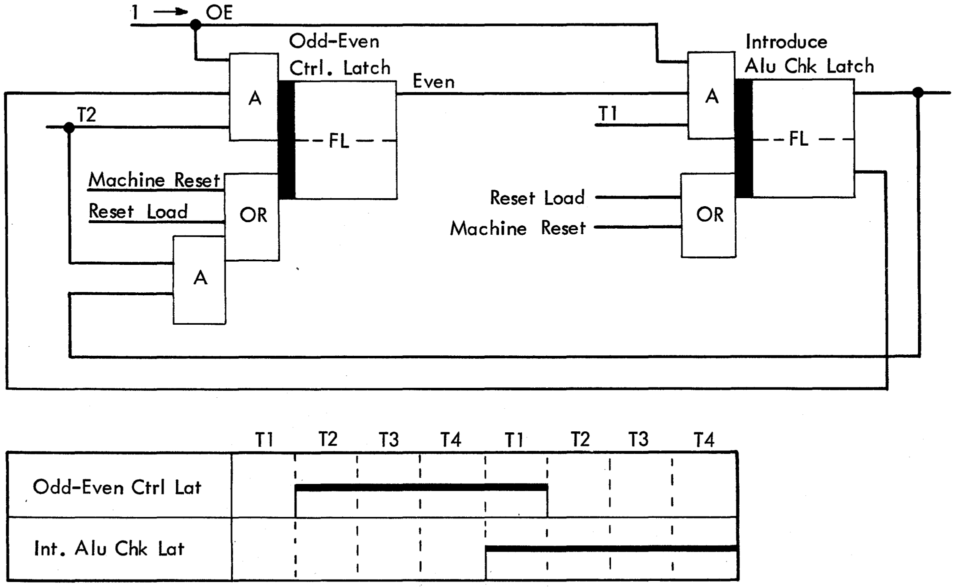


Figure 3-31. 1->OE Control

circuits. If the S-register is not specified by the CA field, one circuit causes a micro-program stop line to be active. This line stops the CPU clock by blocking the clock start circuit. If the S-register is specified by the CA field, S STOP, and if the process-stop latch is on as a result of certain switch conditions, a process-loop-stop line is made active. This line allows the CPU clock to be started 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 that the micro program returns to the address of the S STOP word after execution of any ROS share or multiplexor share request.

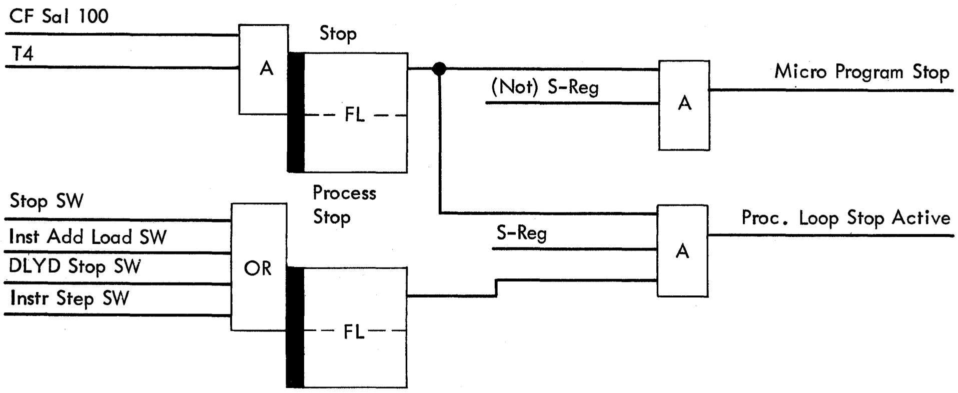


Figure 3-32. Stop Mnemonic

	ROS CONTROL FIELD MNEMONICS			
	CM		CU	
	MNEMONIC	DESCRIPTION	MNEMONIC	DESCRIPTION
0	WRITE	ALLOW DATA TO SET R REGISTER-REGENERATE	MS	MAIN STORAGE
1			LS	LOCAL STORAGE
2	STORE	BLOCK SET OF R REGISTER WRITE TO CORE	MPX	MPX STORAGE
3	IJ→MN	READ CORE	● M/LS	MAIN OR LOCAL STORAGE
4	UV→MN	READ CORE		
5	T→N	READ CORE		
6	*K→N ●	ADDRESS N-REGISTER USING CK FIELD		
7	GUV→MN	READ CORE		

● EXPLAINED IN THE TEXT

ACTIVATED BY CM ≠ 3-7

0		
1	USE GR	SELECTOR CHANNEL R-REGISTER
2	K→W	SET W-REGISTER
3	FWX→WX	MPX BACKUP ROAR

Figure 3-33. Storage Control Mnemonics

STORAGE CONTROL MNEMONICS (FIGURE 3-33)

*K->N: This mnemonic addresses a byte in the local storage. The N-register is set using the format: 1,0, CN0, K0, 1, K1, K2, K3. Let's assume byte 27 has to be read. The address in hexa- decimal for byte 27 is BB. This must appear in the N-register. BB in bit form looks like this, 1011 1011. If the designations N0 through N7 are assigned to this number let's determine how each bit is set.

N0=1	Forced
N1=0	Forced
N2=1	CN0 is set to a 1
N3=1	CK-0 a 1
N4=1	Forced
N5=0	CK-1 a 0
N6=1	CK-2 a 1
N7=1	CK-3 a 1

The CK field must be 1011. Since this is used as a constant, it must be specified on the E line of the CLD box. The expression to read byte 27 is *KBB->N LS.

M/LS: This mnemonic controls which portion of core is used. A test is made on the status of the G-register at the time it contains an Op code. If, as a result of testing the G-register, the Op code is determined to be in RR format, then the expression M/LS controls local storage. Any other format causes main storage to be selected.

ROS	CONTROL		FIELD		MNEMONICS	
CN	CH		CL		CS	
	MNEMONIC	DESCRIPTION	MNEMONIC	DESCRIPTION	MNEMONIC	DESCRIPTION
0	0		0			
1	1		1		LZ → S5	LOW Z BUS EQUAL ZERO
2	R0		CA→ W ●	GATE CA FIELD TO SET W REGISTER	HZ → S4	HIGH Z BUS EQUAL ZERO
3	V67=0 ●	V-REGISTER POS. 6 and 7	AI	ADDRESS IN	LZ→S5,HZ→S4	
4	STI	STATUS IN	SVI	SERVICE IN	0 → S4, S5	RESET S4, S5
5	OPI	OP IN	R=VDD ●	TEST R REGISTER FOR VALID DECIMAL DIGIT	TREQ → S1	1050 REQUEST
6	AC	ALU CARRY	1BC	ONE BIT CARRY	0 → S0	
7	S0		Z=0	Z BUS	1 → S0	
8	S1		G7		0 → S2	
9	S2		S3		ANSNZ → S2	ANSWER ON Z BUS NON ZERO
A	S4		S5		0 → S6	
B	S6		S7		1 → S6	
C	G0		G1		0 → S7	
D	G2		G3		1 → S7	
E	G4		G5		K→ FB●	FB REGISTER-MX TAGS OUT
F	G6		INTR	TEST FOR ANY INTERRUPT	K→ FA●	FA REGISTER-MX TAGS OUT

● EXPLAINED IN THE TEXT

ACTIVATED BY AS COL 64=1

0		
1		
2		
3		
4		
5		
6	GUV → GCD	SELECTOR CHANNEL REGISTERS
7	GR → GK	
8	GR → GF	
9	GR → GG	
A	GR → GU	
B	GR → GV	
C	K → GH	
D	GI → GR	
E	K → GB	SX TAGS OUT
F	K → GA	SET SX TAGS OUT

Figure 3-34. Branch Control Mnemonics

BRANCH CONTROL MNEMONICS

Branch mnemonics are shown in Figure 3-34. Some mnemonics require further explanation.

CA->W: The CA field is gated to set the W-register for address changing of the ROS word. Example: to change from address 01A3 to 08B6 the expression CA08->W is used. Parity is maintained for the W- register by using the PK bit. An X1 branch is taken.

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 CLD box. The PK bit is necessary and its condition is also specified on the E line. If the CK field value is 3 and the PK bit is used, on the E line you will find K=0011 P1.

K->FB: The CK field and the PK bit control the set and reset of some multiplexor or FB register latches. This mnemonic also provides a gate for the set and reset of other latches.

R = VDD: Each half of the R-register is checked for a valid decimal digit. Positions 0-3 and 4 through 7 of the R-register, if valid, contain any digits from 0 through 9.

V67=0: Positions 6 and 7 of the V-register are tested. If both positions are 0, the X6 position of ROAR is set to one.

Status: The bits of the S-register normally are set and reset to control specific functions.

S0	True or complement. If S0 is a 0 the operation to be executed is true (used when the CV field is coded±).
S1	A 1050 request will set S1 to 1.
S2	S2 is set to a 1 if there is a bit on the Z bus. ANSNZ->S2.
S3	Carryout from ALU. This bit is used with the CC field decoding 4,5, and 6.
S4	High Z bus zero
S5	Low Z bus zero
S6	Miscellaneous
S7	Miscellaneous

PARITY BITS

- The PK bit provides odd parity for the CK or the CA fields.
- The PA bit provides odd parity on the ROAR address.
- The PC bit provides odd parity on the control registers.
- The PS bit provides odd parity for the SALs.
- The PN bit provides odd parity for the CN field.

The PK bit is used with either the CA or the CK field. When the CK field is used as a constant in an arithmetic statement, the PK bit is not specified (0). If K appears in a storage statement (*KBB->N), or in a statement which changes the W-register, the PK bit is used to provide odd parity on the CK field. In the expression K->W the CK field is used to set the W-register. If the value in the CK field is 0011 (3) the PK bit must be a one to maintain odd parity of the CK field. If the CA field is used to set the W-register, (CA->W), the PK bit maintains odd parity for the CA field.

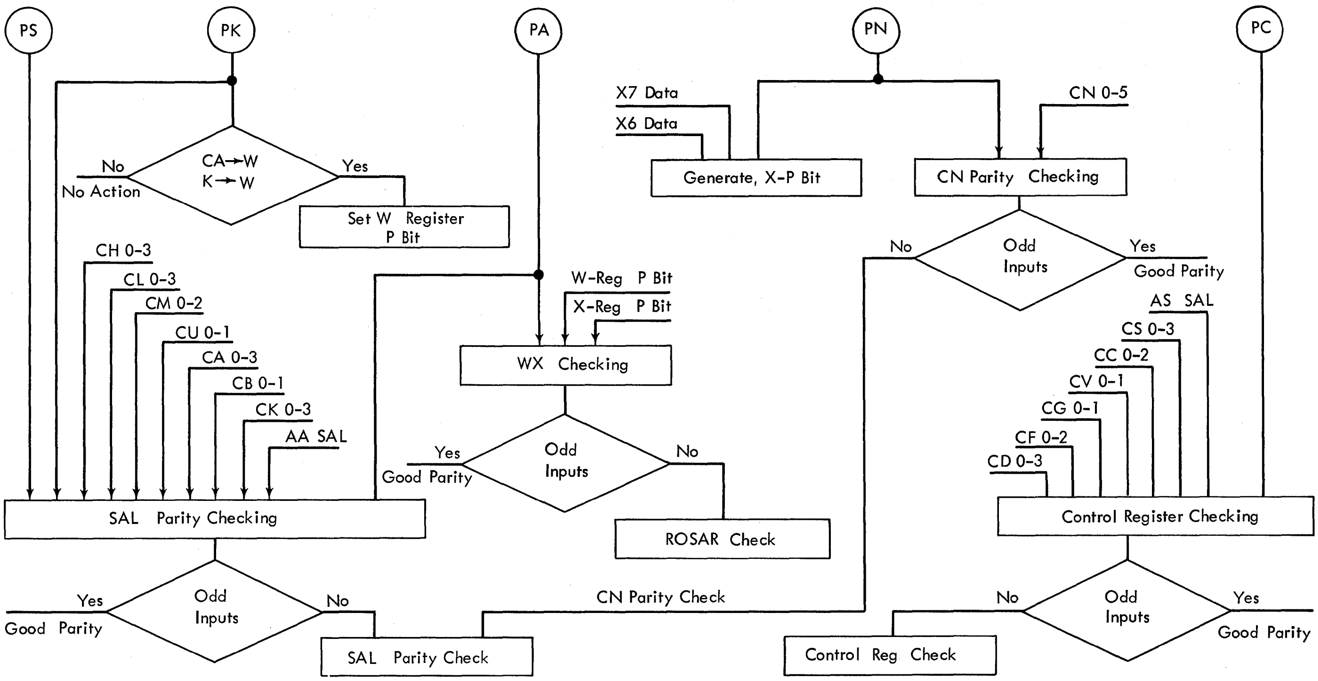


Figure 3-35. Parity Check Bits

The PA bit sets parity for the ROAR address. As an example, if the address of the ROS word is 01BF (0001 1011 1111), the PA bit must be a one to maintain odd parity.

Lines controlled by the AS bit position and the ROS control fields (CD, CF, CG, CV, CC, CS) are maintained in odd parity by use of the PC bit.

Lines controlled by the PA bit position, the AA bit position and the SAL's (CH, CL, CM, CU, CA, CB, CK) are kept in odd parity by use of the PS bit.

The PN bit is used to maintain odd parity on the CN SAL s.

Figure 3-35 shows the fields and the parity bits used for each checking circuit.

REVIEW QUESTIONS - CONTROL FIELD MNEMONICS

1. What mnemonic is used to gate the status of the I-Wrap latch into the Wrap-Buffer latch?

2. If the ASCII latch is on, the mnemonic ASCII forces a _____ branch.

3. Is the symbol ± in the CV field a binary or decimal operation ?

4. How is the CC field encoded to perform the function?

5. In reference to question four, are the capacitors for this field punched out or are they left intact?

6. Can the data in the G-register be gated to the B-register of ALU?

7. How are the CH and CL fields encoded when the mnemonic INTST IS USED?

8. A line called Reset Load may reset the Odd/Even latch and the introduce ALU Check latch. What Mnemonic activates this line?

9. How is the CK field punched when it is to be used as an emit constant of 2?

10. How are the lines for local storage made active when the mnemonic M/LS is used?

11. What affect does the CM field coding 001 have on storage?

12. If the four low bits on the Z bus are zero when the mnemonic LZ->S5, HZ->S4 is used, how is S5 set?

13. Can a four way branch be set up using the status of S0 and S1?

14. How many parity bits are there? Name them.

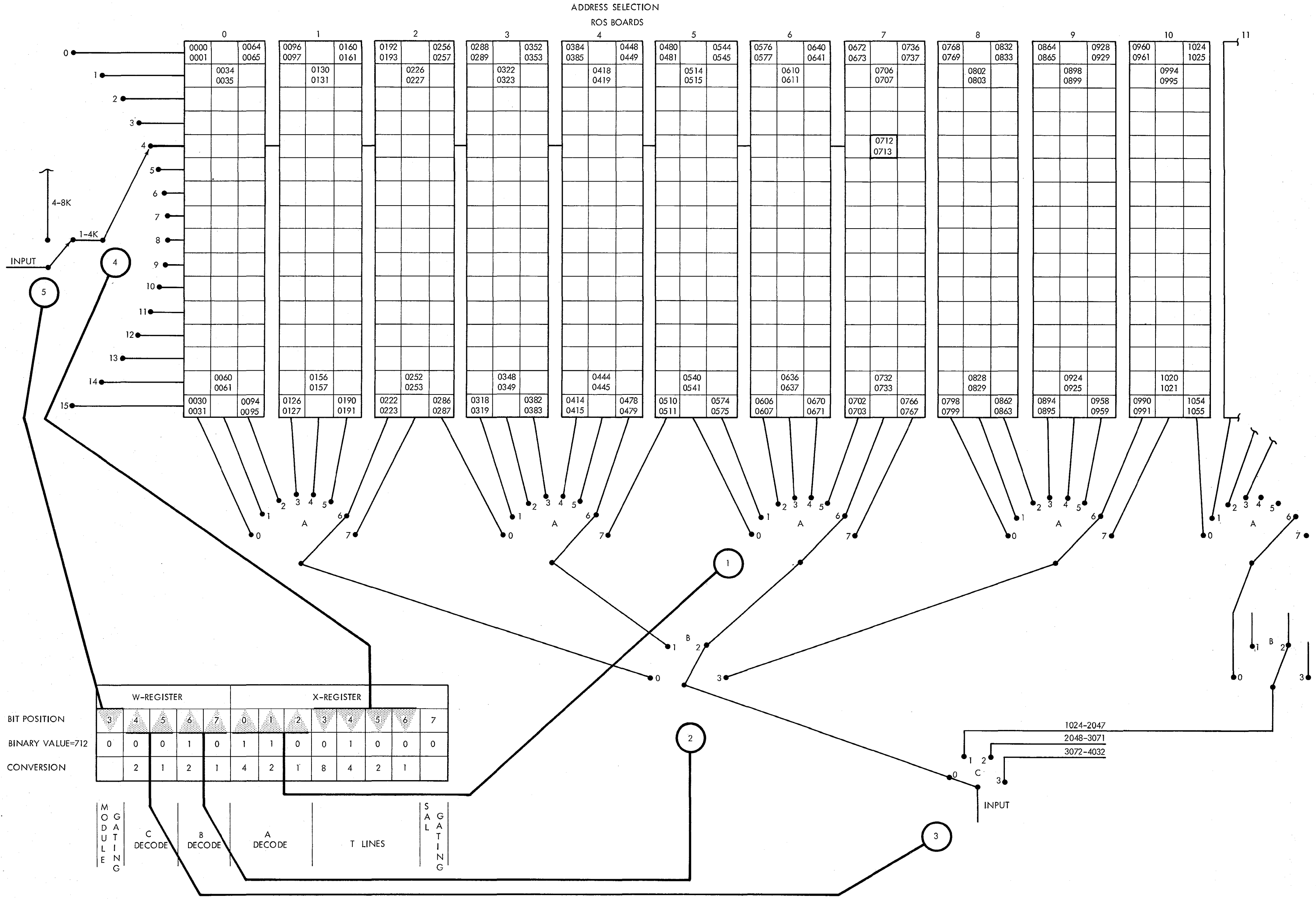
15. Is the PK bit ever used to maintain W-register parity? If it is, explain.

ROS ADDRESSING

GENERAL INTRODUCTION

- There are 96 ROS words of 60 bits each on one ROS board.
- A 4K module contains 4032 words on 42 ROS boards.
- A ROAR (Read Only Address Register) address selects two ROS words to read out.
- The address in ROAR gates the proper ROS word to the sense amplifier latches.
- Maximum storage is 8064 words.

A ROS board accommodates 8 ROS documents. Each ROS document contains 12 ROS words of 60 bits each. The total number of words on a ROS board is 96. All words on a ROS board are used. In a 4K module the addresses are sequential. Because of the electrical connections on a ROS board, a ROAR address selects two ROS words. However, only one of these words is gated to the sense amplifier latches. Since there are 96 words on a ROS board and two words are addressed at a given time, 48 drivers are needed to read out the 96 words.



	W-REGISTER					X-REGISTER							
BIT POSITION	3	4	5	6	7	0	1	2	3	4	5	6	7
BINARY VALUE=712	0	0	0	1	0	1	1	0	0	1	0	0	0
CONVERSION		2	1	2	1	4	2	1	8	4	2	1	
	MODULE GATING	C DECODE		B DECODE		A DECODE			T LINES				SAL GATING

Figure 3-36. Switch Addressing

In Figure 3-36 notice the addresses on ROS board 0. Each small block represents two ROS word addresses. The decimal addresses on this first board are 0000 through 0095. These addresses are contained in the 48 blocks which represent our units of selection, drivers. Note at this time, that a 3 by 16 matrix is used.

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 0057, this address and address 0056 are selected. However, only the ROS word at address 0057 is gated to the sense amplifier latches.

ADDRESSING PRINCIPLES

- A driver is selected by the coincidence of a T line and a driver decode line.
- There are 16 T lines, T0 through T15, in a 4K module.
- A driver decode line provides half selection for 32 addresses.
- ROAR consists of 15 latches, 13 are used for addressing and 2 for parity.

Assume that ROAR contains the address of the ROS word at location 0712. The binary value in ROAR that represents this address is: 0001011001000. This value is shown in the small table in Figure 3-36. Address 0712 is found on ROS board 7. The horizontal line to this address is developed from the T switch. The vertical line to this block is developed from switch A, position 6. See Figure 3-36 ①. The lines from switches A, B and C are the driver decode lines. Remember, both a T line and a driver decode line are necessary for full driver selection.

To see why line 6 from switch A is used, see the chart in this figure. ROAR consists of 15 latches. The W portion consists of 5 latches, the X portion 8. Two latches are used to maintain parity. The darkened portion, X0, X1, and X2 positions of ROAR provide selection for the A switches. If binary weights are given to these positions, as shown on the conversion line, then the result is:

Position		Value		Conversion
X0	=	1	=	4
X1	=	1	=	2
X2	=	0	=	+ 0
				6

Note: If a bit value is a one, use the number below it on the conversion line.

Because there is an A switch for every 256 addresses, further selection is necessary. This is done by switch B, ② in Figure 3-36. With switch B at position 2, our selection narrows to the third group of 256 address in each 1K unit of words. The B switch is controlled by positions W6 and W7 of ROAR. Again, using the bit value and conversion line, our result becomes:

Position		Value		Conversion
W6	=	1	=	2
W7	=	0	=	+ 0
				2nd position of switch B

This is all well and good, but remember that there is a switch B for each 1K unit of words. This decoding is duplicated three more times for a 4K module.

Switch C ③ in Figure 3-36 is used to select which 1K unit contains our desired address. This decoding is controlled by the W4 and W5 positions of ROAR. Since the bit values of these positions is zero, the conversion values are not used and position 0 of switch C is active. If you are wondering why there are only four positions of switch C even though the maximum amount of storage is 8K, it is because the selection that determines whether our address is in the first 4K module or the second 4K module is made elsewhere.

Let's review for a moment. The driver for address 0712 and 0713 requires 2 inputs for full selection. One of these inputs is developed through switches A, B, and C. Switch C determines that our address is in a particular block of 1024 addresses. Switch B divided this block into smaller divisions of 256 addresses. Switch A divided a block of 256 addresses into eight sections of 32 addresses each.

To get back to our address of 0712, follow the horizontal selection line to the T switch, position 4. See ④ Figure 3-36. These 16 positions of the T switch are controlled by the X3, X4, X5, and X6 positions of ROAR.

Position		Value		Conversion
X3	=	0	=	0
X4	=	1	=	4
X5	=	0	=	0
X6	=	0	=	+ 0
				4th position of switch T

We have previously mentioned the fact that selection is made in modules of 4K. Figure 3-36 ⑤ shows that the T switch is further controlled. There are two T switches, one that activates addresses 1K-4K, and the other, 4K-8K. Which 4K module is selected depends on the W3 position of ROAR.

The selection is complete. Addresses 0712 and 0713 are selected. Position X7 of ROAR was never used for address selection. This bit position provides a gate to the sense amplifier latches. If, as in the case of address 0712, position X7 is a 0, then the 60 bits stored at the even address are gated to the sense latches. Had address 0713 been desired, the X7 bit would have been a one. All other positions would be identical. The one in position X7 gates the 60 bits stored at the odd address to the sense latches.

APPLIED PRINCIPLES

- There are 48 drivers for one ROS board.
- The ROS unit uses both the bit and non-bit lines sent from the CPU.
- There are 120 sense amplifiers.
- There are 60 Sense Amplifier Latches (SAL's)

Until now, we have used switches to explain the addressing principles. Switches are slow-operating and cumbersome. Faster and more reliable addressing is done electronically.

Figure 3-37 is a representation of the electronic addressing for one ROS board. Again, let us start with the smallest unit of selection, a driver. The driver connections are shown in Figure 3-38.

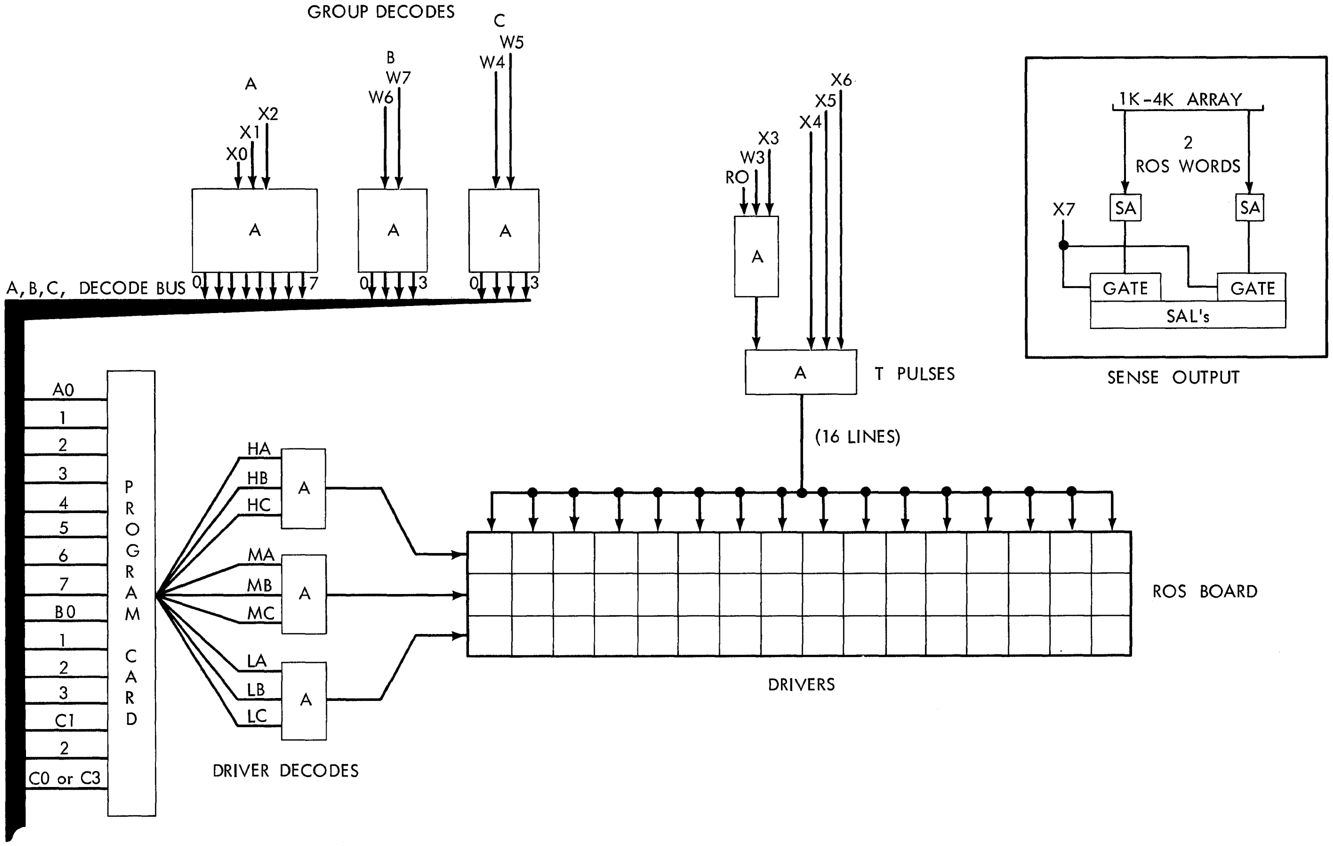


Figure 3-37. ROS Addressing

Figure 3-38 represents the 48 drivers of one board. The drivers are physically located on two small cards, driver card A and driver card B, attached to the ROS board. There are 24 electrically connected drivers on each small card. Each driver is a one transistor circuit. A transistor consists of a base, emitter, and a collector. Let us consider the connections to each.

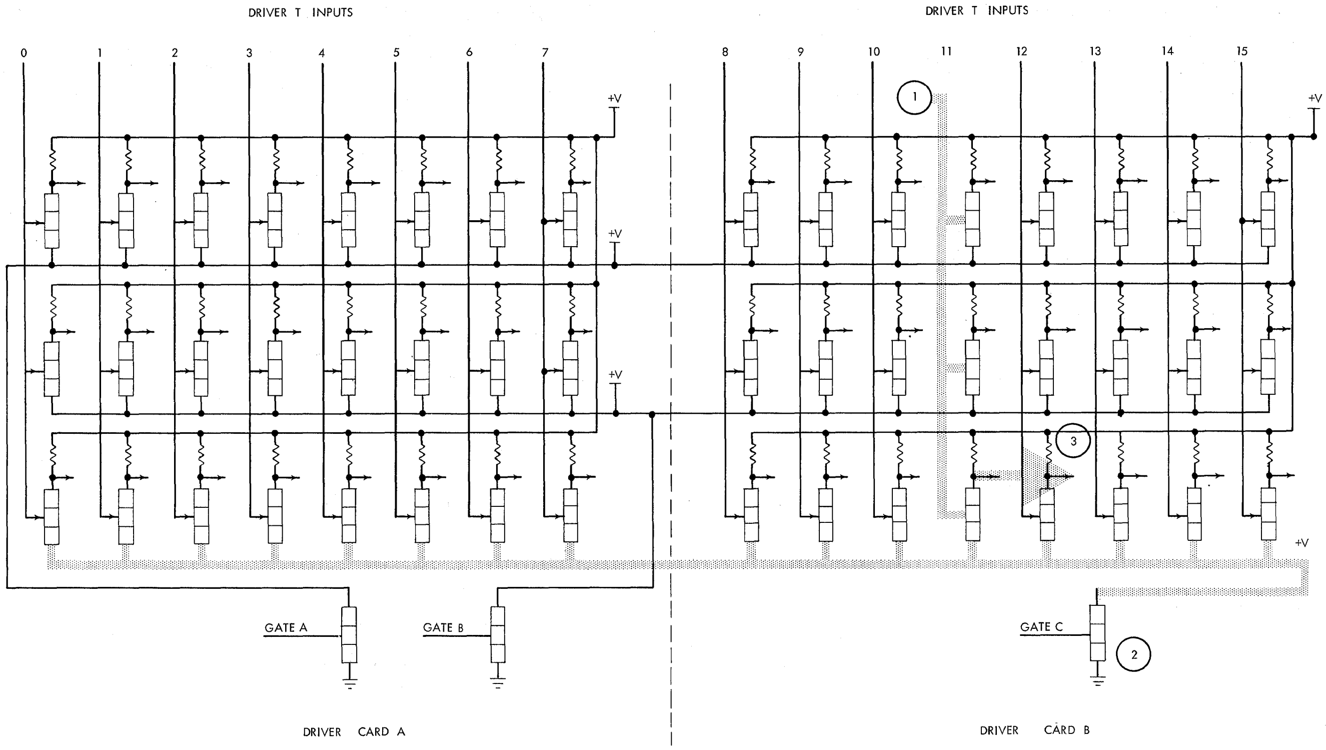


Figure 3-38. Driver Cards

The Base 1: Three driver bases are common. The darkened line that connects these three drivers is a T input line. Since one T line feeds three drivers, there must be 16 T lines going to each ROS board.

The Emitter 2 : Sixteen drivers have their emitters commoned. The darkened line that connects these 16 drivers is the driver decode line. Notice that the driver decoders are on the driver cards. Two decoders are on driver-card A, one driver decoder is on driver-card B. Since one driver decoder line controls the emitters of 16 drivers, three driver decoders are used for controlling the addresses on one ROS board.

The Collector 3 : The output from the collector drives two ROS Words.

Again, see Figure 3-37. The T lines for a 4K module are developed from 16 four-input AND circuits. Only one T output line is active at a time. The CPU provides both the bit and not-bit lines to satisfy the AND inputs. Three inputs to each AND are controlled by the six lines for the X4, X5 and X6 positions of ROAR. The other input is developed from the ANDed conditions of the X3 and W3 positions of ROAR plus the timing pulse, RO (Read Out).

In the 8K module there are two sets of 16 AND circuits. One set is used for addresses 0000 through 4031, the other set for addresses 4096 through 8127. The addresses 4032 through 4095 are non existent. The W3 bit, when set to a one, selects the second 4K module. The binary weight for this position is 4096 if you use a powers of 2 table.

The other half of the driver selection is done by the driver decode circuits. These are essentially three input AND circuits. Only one of these circuits is active at a time.

The inputs to these circuits are: HA, HB, HC, MA, etc. The A, B, C portions of these notations correspond to the previously discussed switches. H, M and L mean high, middle, and low and refer to the three address groups on a ROS board. These nine lines, which feed our driver decoder circuits, are selected by means of a program card. A program card is nothing more than a pluggable card used to join the circuits from the decode bus to the decoder input circuits. There are 15 inputs to the program card from this decode bus. They are the lines:

1. A0 through A7
2. B0 through B3
3. C1, C2, and either C0 or C3.

Six lines are sent from the CPU, bit and non-bit lines from positions X0, X1, and X2. These lines are ANDed to develop the eight bus lines A0 through A7. The B0 through B3 bus lines are developed by the four lines from the W6 and W7 positions of ROAR. Lastly, the C0 through C3 bus lines are made active by positions W4 and W5.

It should be evident that with a given setting in ROAR, one each of the A, and B, and the C decode bus line is active and the other 13 are inactive. The program card uses these lines from the deocde bus to satisfy the inputs of the three input decoder circuits. Therefore, any group of 96 ROS words can be physically located on another ROS board in the array just by changing the position of the program card. There are some physical limitations in that a program card for an even ROS board, 0 2, 4, etc., can only be put in another even location. An odd program card may only be plugged in another odd location. There is also an electrical restriction in that the C0 pulse is supplied to the first 11 boards, the C3 pulse is only supplied to boards 12 through 42.

When a driver is fully selected it provides the drive to two ROS words. See the insert for the sense output circuit in Figure 3-37. These two ROS words are read out to sense amplifiers. Each word contains 60 bits therefore, there are 120 sense amplifiers. The output from one set of 60 sense amplifiers is gated to the sense latches, depending on the setting of the X7 position of ROAR. If X7 is set to a one, the word at the odd address is set into the sense latches. The words located in the 4K-8K section of an 8K module have their own sense amplifiers. These are ORed with the amplifiers in the 1K to 4K section ahead of the gating circuit. This is possible as only one driver is active in either the 1K to 4K section or the 4K-8K section.

From Figure 3-39 you can determine the inputs to the three input AND decoder circuits for any address desired. If the console indication of the ROAR address is converted to decimal, then the three inputs for address 0712 are C0, B2, and A6. C0 is used for any address 0000 through 1023. B2 is used for addresses 0512 through 0767 within this group. A6 selects the addresses 0704 through 0735. Our selection has been narrowed to 32 addresses. The additional selection is made by the T line. Remember, two words are read out at a time, one of which is gated into the latches under control of X7.

		A0	A1	A2	A3	A4	A5	A6	A7
C0	B0	0000	0	00095		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 3-39. Address Table

Let us try, as an example, the address 2787. This address occurs in the third unit of 1024 addresses; therefore, the C2 input is needed. The address 2787 falls in the horizontal B2 row. Vertically, it is in the A7 row. The three inputs to the decoder circuit must be C2, B2, and A7. Figure 3-39 can also be used to determine the ROS board on which any given address is located. Using out last example, address 2787, you can see that it appears on ROS board 29. Our first address of 0712 is found on ROS board 7.

By doing some simple arithmetic, Figure 3-39 may be used for addresses greater than 4032. Take, for example, address 6789. Subtract 4096 from this. The result is 2693. This is the equivalent address in the first 4K module. The three inputs to the decoder circuit for this address are C2, B2, and A4. The address is located on ROS board 28 of the second module, 4K-8K.

REVIEW QUESTIONS - ROS ADDRESSING

1. How many words are read out by the address in ROAR?

2. What is the maximum amount of ROS storage?

3. A driver is selected by means of a ________ line and a ________ line.

4. ROAR consists of ________ latches.

5. How many drivers are used for each ROS board?

6. How many sense amplifiers and sense amplifier latches are there ?

7. What is a program card?

8. Are additional SALs needed for the 4K to 8K section of an 8K ROS unit? If yes, how many?

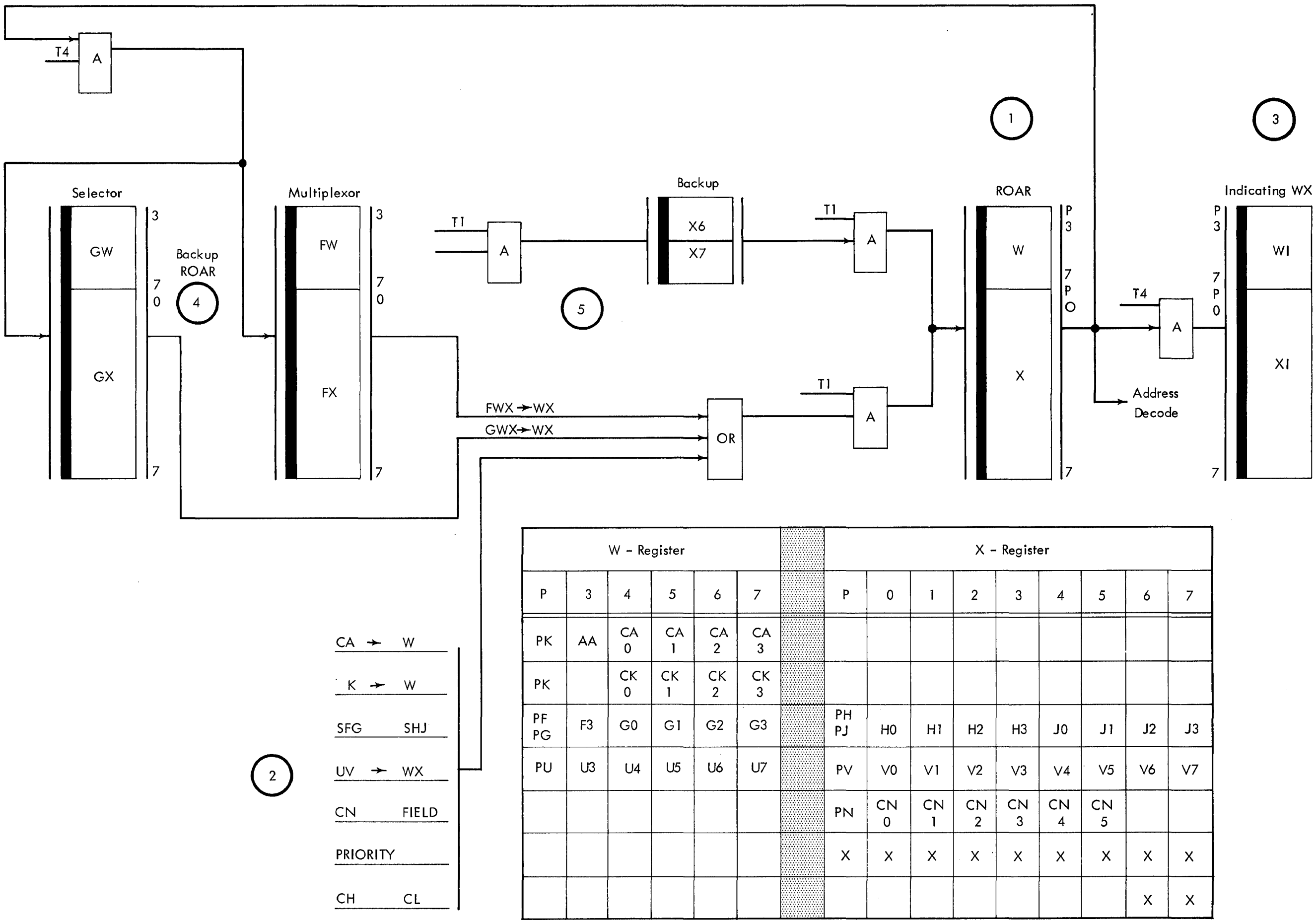
9. What position of ROAR selects the addresses in the 4K to 8K unit?

10. Which position of ROAR provides the final gate to the SAL's?

<u>ROAR CONTROLS</u>

- The console lights for a ROS word address are controlled by an indicating ROAR.
- The address in ROAR may be stored in one of two backup ROAR s.
- Two latches are used to store backup branch conditions.

Refer to Figure 3-40 ①. You have seen how an address in ROAR selects a particular ROS word. ROAR consists of 15 polarity hold latches. Thirteen are used for address decoding and two for parity.



	W - Register							X - Register									
	P	3	4	5	6	7		P	0	1	2	3	4	5	6	7
CA → W	PK	AA	CA 0	CA 1	CA 2	CA 3										
K → W	PK		CK 0	CK 1	CK 2	CK 3										
SFG SHJ	PF PG	F3	G0	G1	G2	G3		PH PJ	H0	H1	H2	H3	J0	J1	J2	J3
UV → WX	PU	U3	U4	U5	U6	U7		PV	V0	V1	V2	V3	V4	V5	V6	V7
CN FIELD								PN	CN 0	CN 1	CN 2	CN 3	CN 4	CN 5		
PRIORITY								X	X	X	X	X	X	X	X	X
CH CL															X	X

Figure 3-40. ROAR Controls

ROAR is set from many sources. Let's first consider the sources shown in Figure 3-40 ②. The sources set all or some positions of ROAR. The table in this figure will help explain the specific positions of W and X that are set from each source.

The source CA->W is an expression found in a micro program that causes control lines to be brought up and gate the CA control field SAL output to the W-register. Notice in the table that the parity bit for W, P, is set by PK. PK is the parity position for the CK field and is used because the CA field has no parity bit of its own. The W3 position is determined by the one bit AA field. Note. The W3 bit is set when in 1401 mode to address a second 4K RCS module. A complete description of 1401 mode operation is found in the Special Feature section. The remaining positions of the W-register, W4 through 7 are set from the CA 0 through 3 SAL outputs.

The K->W expression is similar to CA->W, except that the W-register is set from the CK field SAL output.

Switches F, G, H and J are used to set the ROAR also. Each position on a switch provides five output lines when decoded. The lines are 0, 1, 2, 3 and a parity line. The 0 through 3 lines can handle any value from 0 through F in hexadecimal. As an example: If switch G is set to position A, the lines 0 through 3 would contain the value 1010. 1010 is A in hexadecimal.

Refer to the table in Figure 3-40 to see the switch control of ROAR. The parity position of the W-register is set from the parity lines of switch F and switch G. The W3 position is set from line 3 of switch F. The W4 through W7 positions are set by switch G lines 0 through 3. The parity for the X-register is determined by the parity lines of both the H and J switches. Switch H lines 0 through 3 and switch J lines 0 through 3 set the X0 through X7 positions, respectively.

UV->WX is a statement in micro programming that brings up control lines to gate the U- and V-register to set ROAR. The U-register positions 3 through 7 control the W-register.

The CN control field, which is part of every ROS word, sets the X0 through X5 positions of ROAR and provides parity on the X-register.

A priority is an interruption of a micro program routine. There are nine orders of priority and each sets one specific position of the X register. The position that is set in the X-register determines the starting address of a micro program to handle that particular interruption. The priorities and the positions of ROAR that are set are listed:

Priority	ROAR Position
STOP	X0
PROTECT	X1
WRAP	X2
MPX Share	X3
SX Chain Pulse	X4 see note
Machine Check	X5
Initial Program Load	X6
Force IJ	X7
Machine Reset	XP

Note: Other positions of X may be set in combination because of specific conditions.

The CH and CL control fields are decoded to test for branch conditions. The conditions that are tested determine how the X6 and X7 positions are set.

Recall that each micro program step is addressed by a 4-digit number in hexadecimal. Let's try an example. Assume that the ROS cycle that is active is decoding the ROS word at address 0123. If the next address to be executed in the program is 0225, what positions of ROAR are changed and how might they be changed?

ADDRESS		ROAR		
		W Position	X Position	
Register Positions	= 3	4567	0123	4567
123 in hexadecimal	= 0	0001	0010	0011
225 in hexadecimal	= 0	0010	0010	0101

First: The W portion of ROAR has to be changed from 0001 to 0010. One way to set the W-register might be to use the mnemonic K->W. The CK control field would be coded 0010. Remember, the address 0225 must be set up while address 0123 is being executed.

Second: The X-register positions 0 through 5 have to be changed from 001000 to 001001. The CN field of ROS word 0123 must, therefore, be coded 001001.

Last: The X6 and X7 positions of ROAR are controlled by the CH and CL fields. The CH field must be coded 0 and the CL field must be coded 1 to satisfy the remaining ROAR positions.

Now that you have seen some of the ways that ROAR can be set, refer to Figure 3-40 ③. This portion of the figure shows another W- and X register which is used for indicating purposes. Because of timing considerations the output from ROAR is gated to the indicating ROAR to control console lights.

Refer to Figure 3-40 ④. These two registers, GWX and FWX, are backup registers for ROAR. As an example: If during a step of a micro program the selector channel has to break in, the address in the W- and X-registers is stored in the GW an GX registers. An address will be set into W and X that is the first step of another micro program to handle the interruption. After the selector channel micro program is complete, the address that was stored in GW and GX is returned to ROAR and the original program continues.

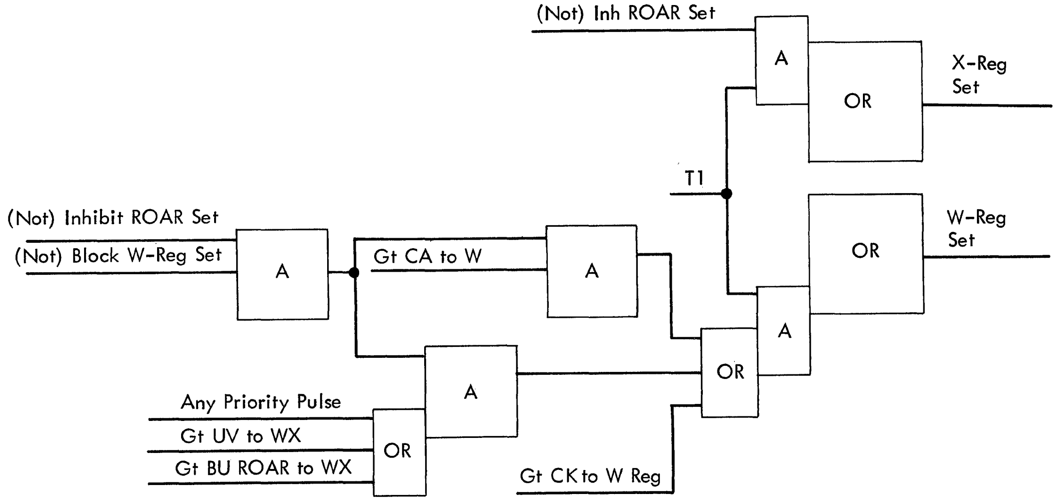


Figure 3-41. WX Register Sets

See Figure 3-40 ⑤ . There are also backup positions for branching. X6 and X7 backup latches are necessary because of timing considerations.

Specific controls for ROAR and indicating ROAR are shown in Figure 3-41 and 3-42.

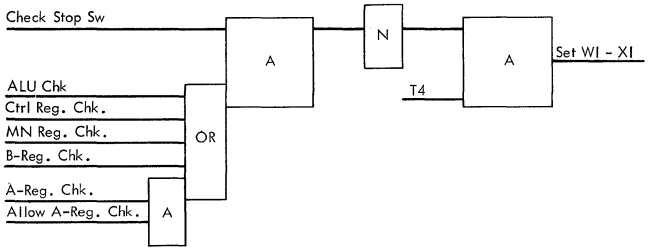


Figure 3-42. Set Indicating ROAR

REVIEW QUESTIONS - ROAR CONTROLS

1. How is position 3 of the W-register set when the mnemonic CA->W us used?

2. What console switches are used to set ROAR?

3. Which control field sets positions 0-5 of the X-register?

4. What address is set in ROAR as a result of a machine check?

5. There are console lights that indicate the status of ROAR.
   TRUE FALSE

6. An ALU check blocks the set of the indicating ROAR.
   TRUE FALSE

ROS TIMINGS

- ROAR is set at T1 time.
- The CPU GO pulse is used to develop the ROAR read out pulse.
- The SAL's are good at T4 time.
- The control registers are set at T1 time.
- Backup ROAR is set at T4 time.

The ROS timings may be divided into three groups:

1. Basic
2. Micro Program Break In
3. Parity Check

BASIC TIMINGS

Figure 3-43 shows the basic timings associated with ROAR. Two 1-microsecond ROS cycles are represented. Each cycle is divided by the CPU times T1, T2, T3, and T4.

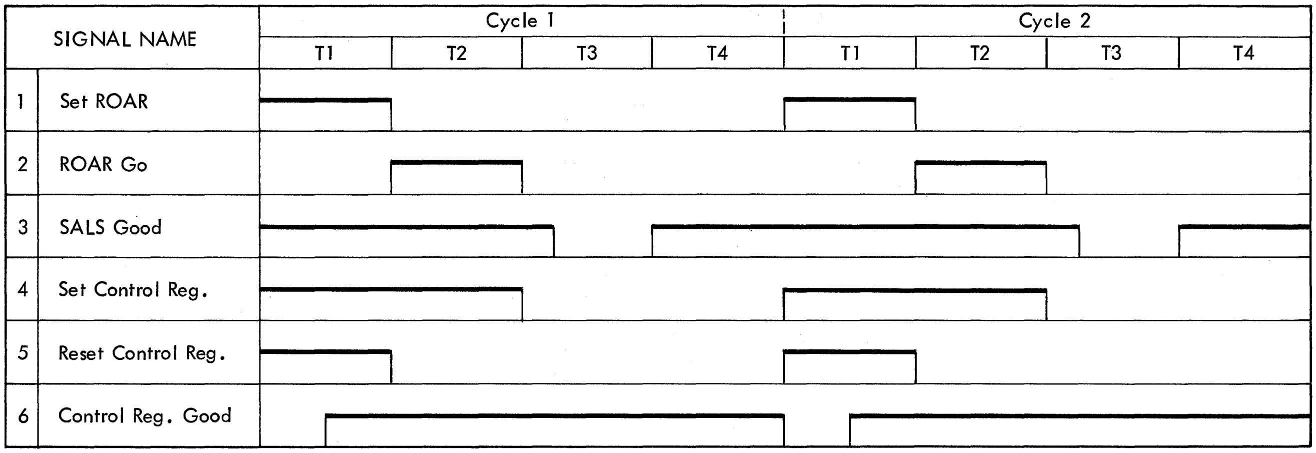


Figure 3-43. Basic Timings

Before any ROS word can be executed, there must be an address in ROAR. The pulse to set ROAR is at T1 time. Remember, ROAR consists of polarity hold latches and that to set a latch it takes a set pulse (T1) plus a bit input. If a latch is ON at set time with no-bit input, it is turned off.

Once ROAR has been set and the latches have been allowed to settle down, the latches can be used to bring up the gate and drive lines for addressing. This is done at T2 time by the GO pulse. The GO pulse is delayed to set up a readout pulse, which allows the SAL's to set at T4 time. The GO pulse is also used to reset the SAL's. The SAL's are considered good from T4 time through a portion of T3 time.

The addressed ROS word contained in the SAL's is not gated to the control-latches until T1 of the <u>following</u> cycle. If the clock were to stop at T4 time, the SAL's would contain the information of the ROS word addressed. The control-register latches would contain the information from the previous ROS word. Though the set and reset times of these latches partially overlap, the set time takes preference.

BREAK-IN TIMINGS

To help explain the timings during a micro program break-in, let's use the example shown in Figure 3-44. Consider the main micro program to be executed consists of the ROS words at addresses 0001 and 0002. A micro program break-in will cause the main program to stop while the ROS words at addresses 0103 and 0104 are executed. The sequence of operation is shown by the darkened arrow. Let's take the operation cycle by cycle. The cycles are labeled according to the time that the control-register latches are good for that address.

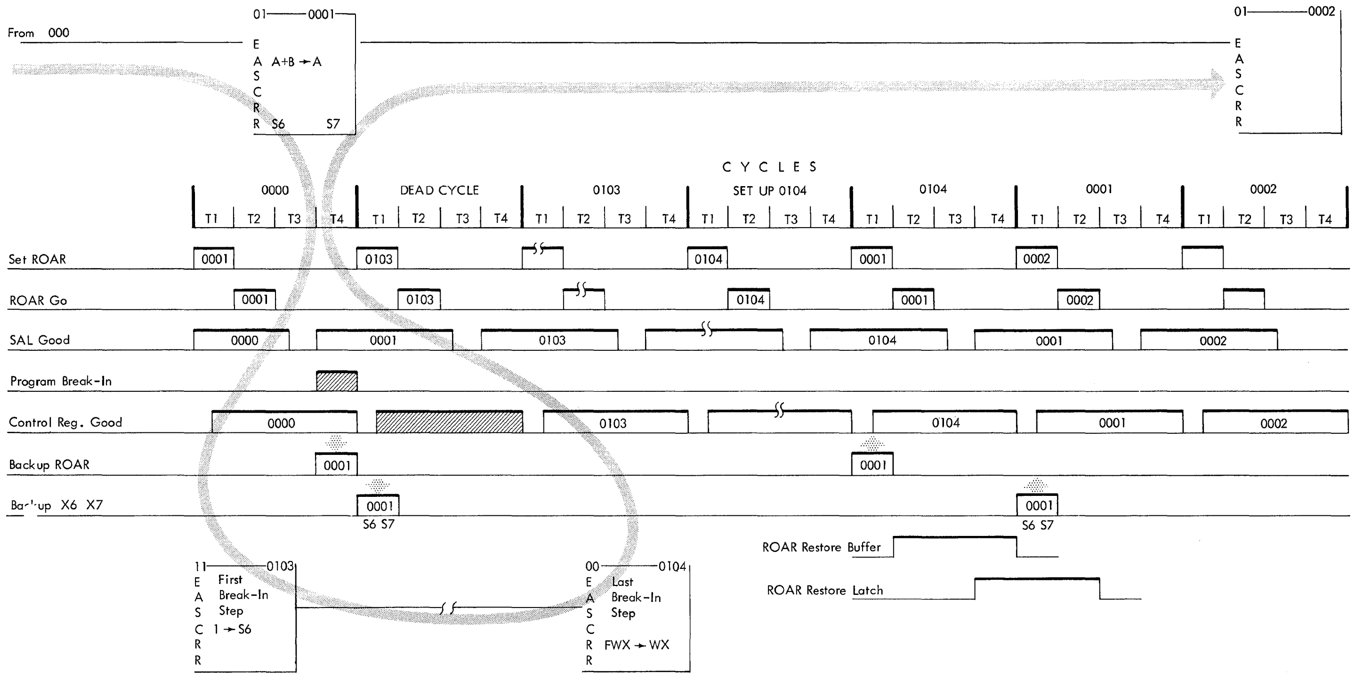


Figure 3-44. Micro Program Break-In

Cycle 0000

At T1 of this cycle, the address of the next micro program step, 0001, is set into ROAR. At T2 time, the GO pulse causes the data at address 0001 to be read out to the SAL's. The SAL's are good during T4 time. Assume that during T4 time, a micro program break-in is called for. Because of this interruption, the address in ROAR, 0001, is stored in a backup storage address register (ROBAR).

Dead Cycle

This is called a dead cycle because no control latches are set. The set pulse to the latches is blocked for the first cycle of the interruption. This should have been the time the control latches were set for address 0001. At T1 time of this cycle, a new address (0103) is forced into ROAR. This address is the first step of the alternate micro program to handle the interruption.

Also at this time, the branch conditions for address 0001 are set into backup X6 and X7 latches. The branch conditions, S6 and S7, have been tested by the SAL's that are good for address 0001. Their status must be stored because, as the alternate program is executed, the 6 and 7 positions of the S-register could be changed.

Cycle 0103

This is a normal cycle that executes the ROS word at address 0103.

Set Up Cycle

This cycle is a normal cycle before the last step in the alternate routine. Its only function in this example is to set ROAR with the address 0104 and begin execution of this ROS word.

Cycle 0104

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 in backup ROAR, 0001, is gated to ROAR so that the original micro program that was interrupted may be resumed.

ROAR Restore Buffer Latch

This latch is turned on at T2 time when a statement in the micro program specifies that ROAR must be changed. This statement appears in the last step of the alternate routine. An example: FWX->WX, which means gate the multiplexor channel backup register to ROAR.

Cycle 0001

The control-register latches are good for address 0001 during this cycle.Even though we have addressed this ROS word again, this is the first time that it is fully 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

This latch is turned on by the coincidence of a ROAR-restore-buffer latch being on and T4 time. This latch provides a gate to set the X6 and X7 position of ROAR from backup X6 and X7.

Cycle 0002

This is the normal execution of the ROS word at address 0002.

PARITY CHECK TIMINGS

A parity check is made on the SAL's, control registers, and ROAR. The SAL's are active from T4 time of one cycle through T3 time of the following cycle. The SAL outputs are sampled at T2 time for odd parity. Machine check register position 4 is set if there is an even number of SAL outputs.

The control-register latches, which are set at T1 time, have their outputs sampled at T4 time. A control-register check sets position 3 of the machine-check register. It also blocks the set of the indicator ROAR if the check-stop switch is set to STOP.

The combination of the parity bits for the W- and X-registers and the PA bit must be odd (all three or any one). If not, the WX check line is active and the ROAR check latch is set (Figure 3-45). If this latch is on at T2 time, the MC-5 latch is set.

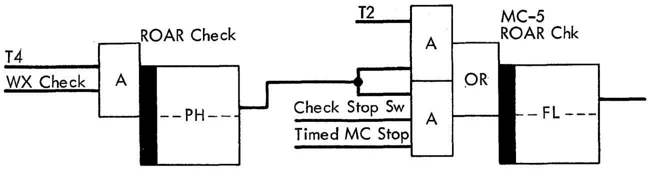


Figure 3-45. ROAR Check

If the SAL's are bad parity, the GO-pulse is blocked. The control register checking is done from the SAL's for those fields. The address in W and X on a check stop is the address of the ROS-word that failed.

REVIEW QUESTIONS - ROS TIMINGS

1. Arrange the following functions in the correct sequence of operation.
   a. Set the SALs
   b. Read out ROAR
   c. Set the control register latches
   d. CPU GO pulse
   e. Set ROAR

2. Explain what data is contained in the SALs and the control register latches if the CPU clock is stopped at T4 time.

3. What is meant by the term "dead cycle"?

4. Why are there backup positions for the X6 and X7 branch controls?

5. At what time are the control register latches checked for odd parity?

PHYSICAL DESCRIPTION

Figures 3-46 and 3-47 show a 4K ROS module as viewed from the left side of the console. The ROS documents are inserted from this side.

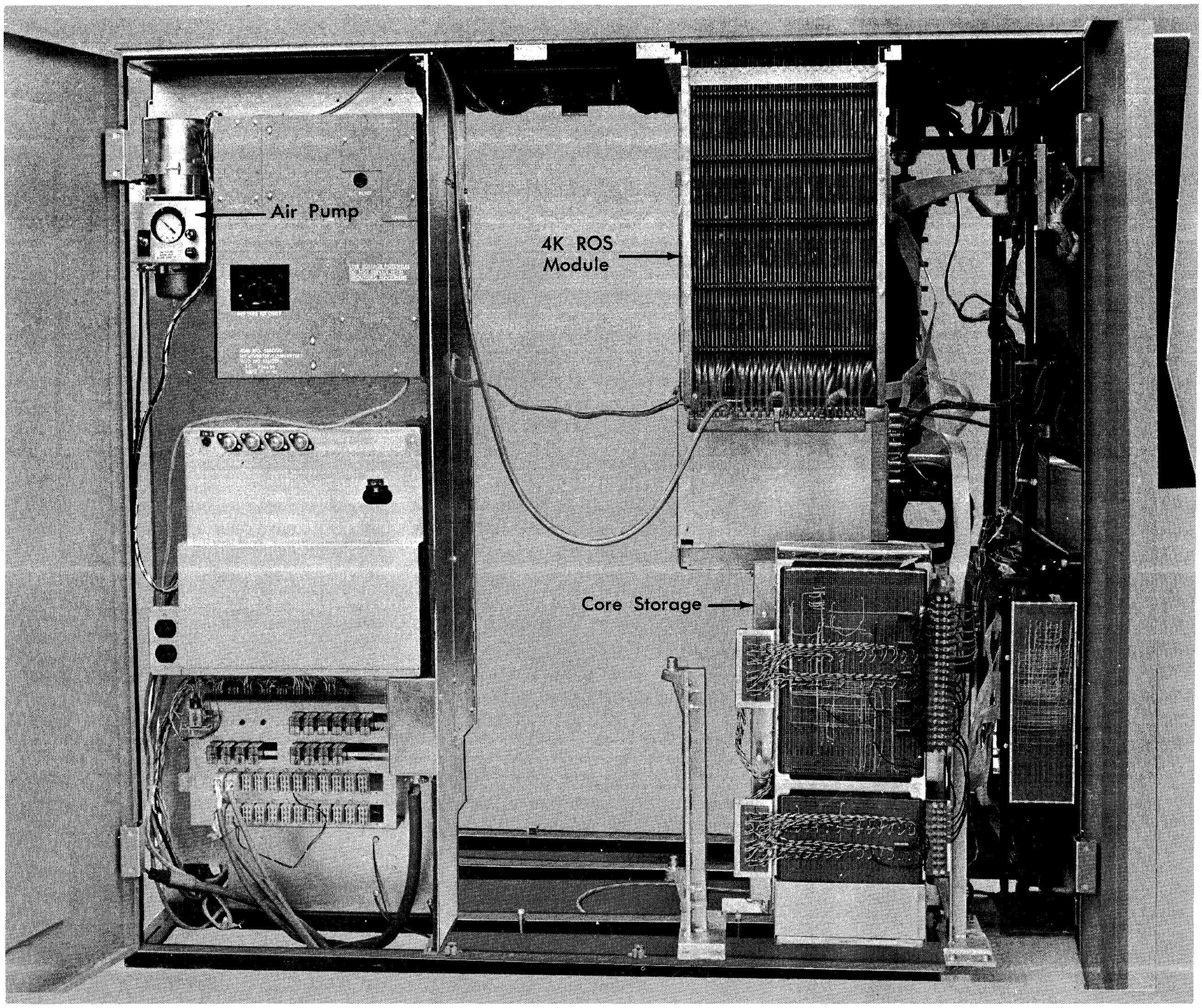


Figure 3-46. IBM 2030, Left Side

One side of a ROS board is pictured in Figure 3-48. The ROS documents are held to this board by means of an air bag. There is a ground plane between adjacent boards as well as within the ROS board itself.

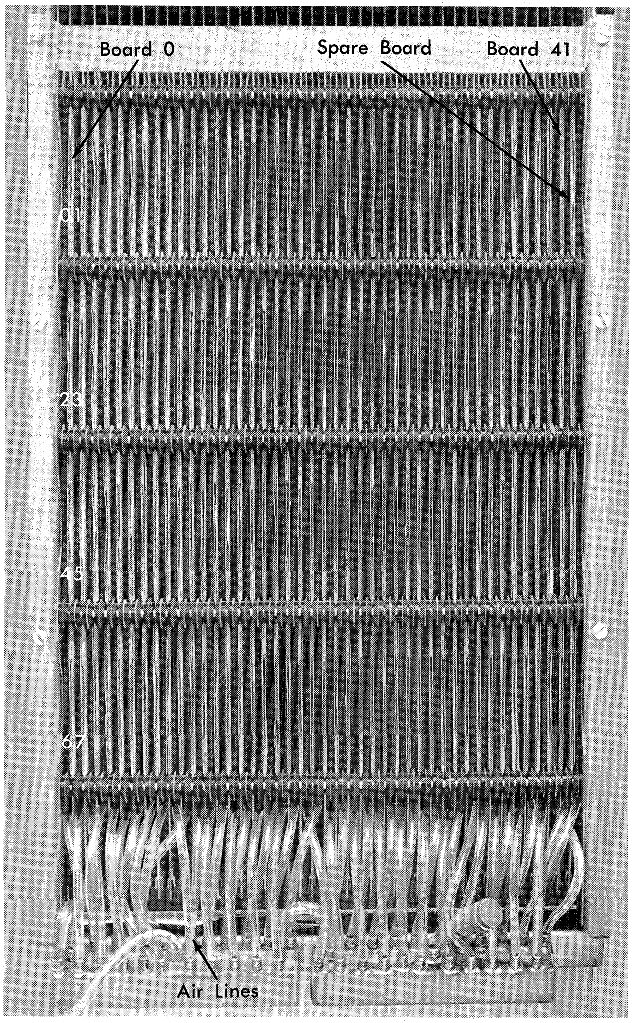


Figure 3-47. RCS Module, Front View

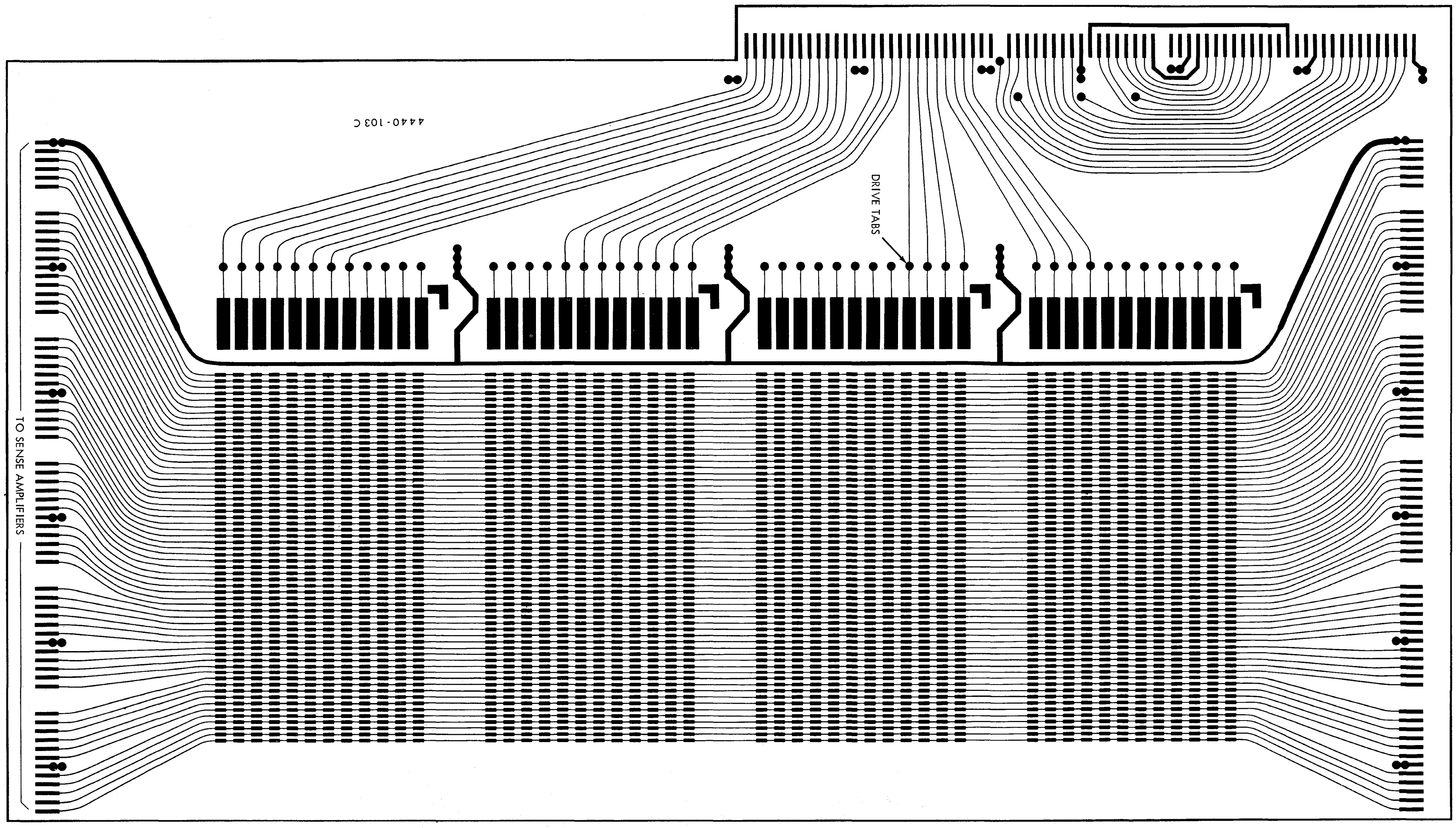


Figure 3-48. RCS Board Layout

Figure 3-49 is a rear view of the RCS module. This figure shows the location of the driver connections and the pluggable program cards. The circuits and cabling for RCS are located in the area below the ROS boards.

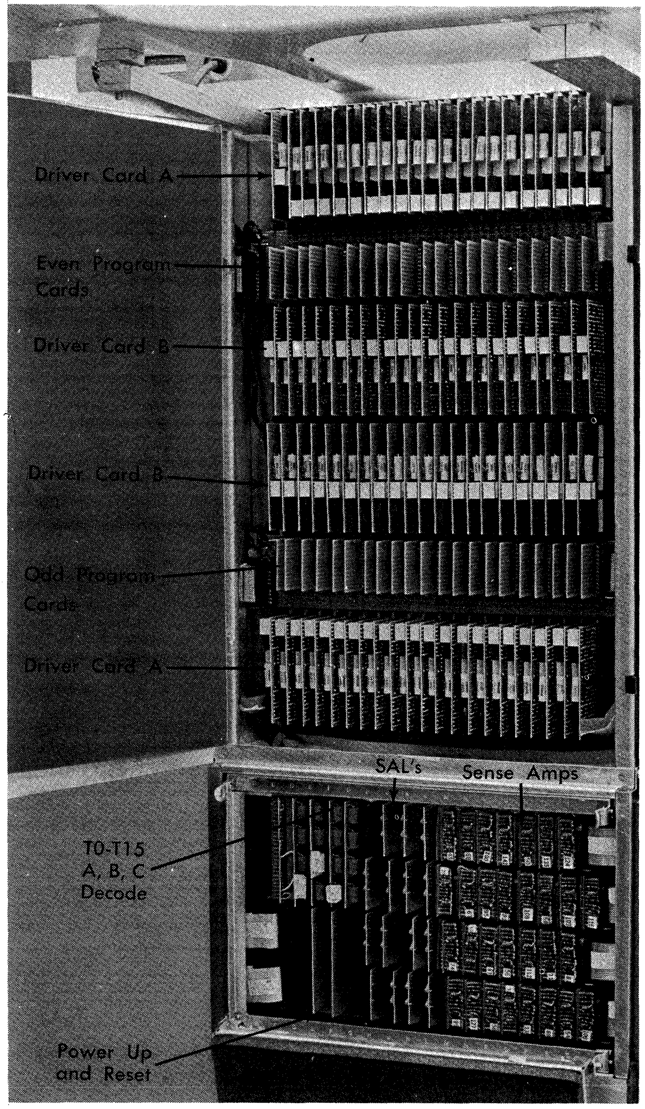


Figure 3-49. RCS Module, Rear View

Figure 3-50 shows the addresses as they appear on RCS boards 0 and 1. Even addresses are on the left side of a RCS board, odd addresses on the right. As an example: address 0017 is on RCS document number 1. Document 1 is the uppermost card on the right side of board 0.

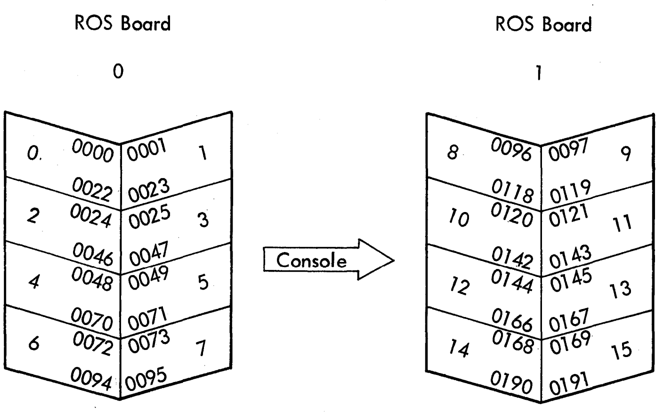


Figure 3-50. RCS Document Addresses

Figure 3-51 shows the layout of the RCS document. The drive tabs are located on the column-80 end of the card. The document is inserted into the module drive tab first. Because the capacitor plates on this document must be next to the ROS board, all the RCS documents that contain words at odd addresses must first be flipped over before insertion.

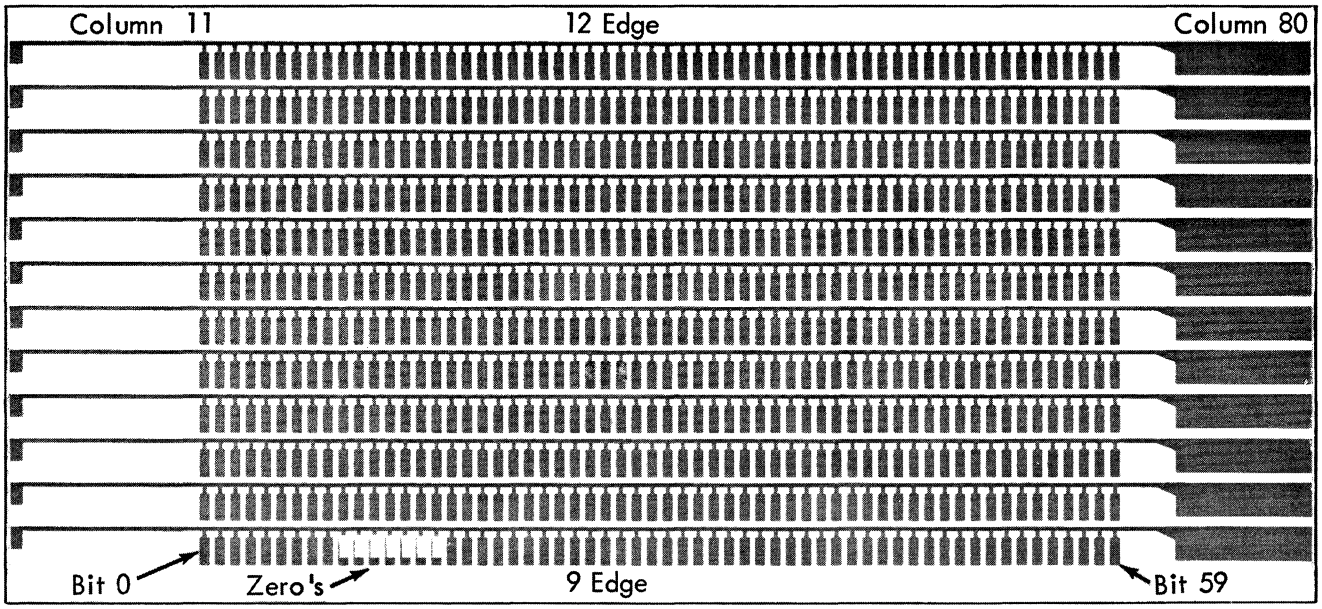


Figure 3-51. RCS Document Layout

If all the RCS documents 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 document. The documents that contain the RCS words at even addresses are numbered from top to bottom as viewed (Figure 3-52).

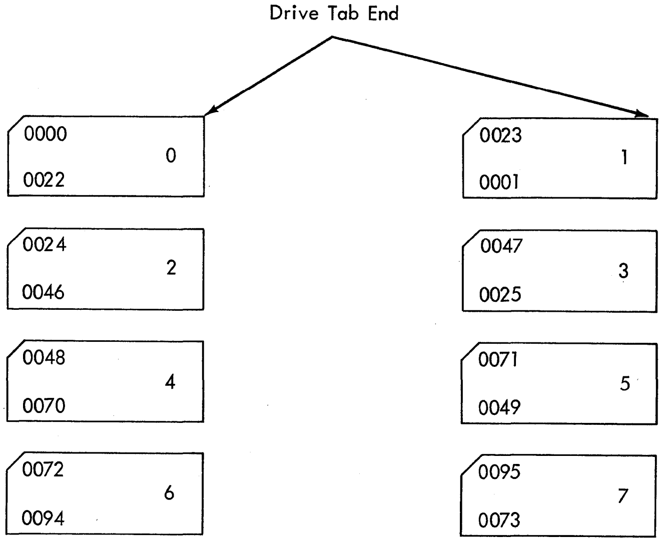


Figure 3-52. RCS Word Numbering

MACHINE CHECK HANDLING

CPU ERRORS

Error conditions (ALU check, SAL check, etc.) may be highly intermittent. The CPU clock circuits are so designed that intermittent errors do not necessarily stop the clock. Each type of error sets a particular position of the machine-check check register. If the CPU is allowed to recognize error conditions, an address is set in RCAR that is the start of a micro program to handle machine checks. This micro program 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, the CPU clock will stop. Should errors occur during a selector-channel RCS 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 micro program. 3. The objectives of the MC micro program. 4. Stop on error conditions.

MACHINE CHECK REGISTER

The MC register consists of eight latches. Consider the setting of each latch (Figure 3-53).

(0) This position is set at T4 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 micro program may be entered. It is conceivable that the register that caused the error may be used in this micro program. 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 micro program until the registers used in the MC micro program have been set to good parity.

(1) This position is set ON at T4 time if there is a B-register check.

(2) An MN register check sets position 2 at T4 time if the allow-write-line is active. This line is active early during the read cycle when MN has been set.

(3) A control-register check sets position 3 at T4 time.

(4) MC-4 is set ON at T2 time with a SAL parity check

(5) Position 5 is set ON to indicate a ROAR check.

(6) An R-register check sets position 6.

(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.

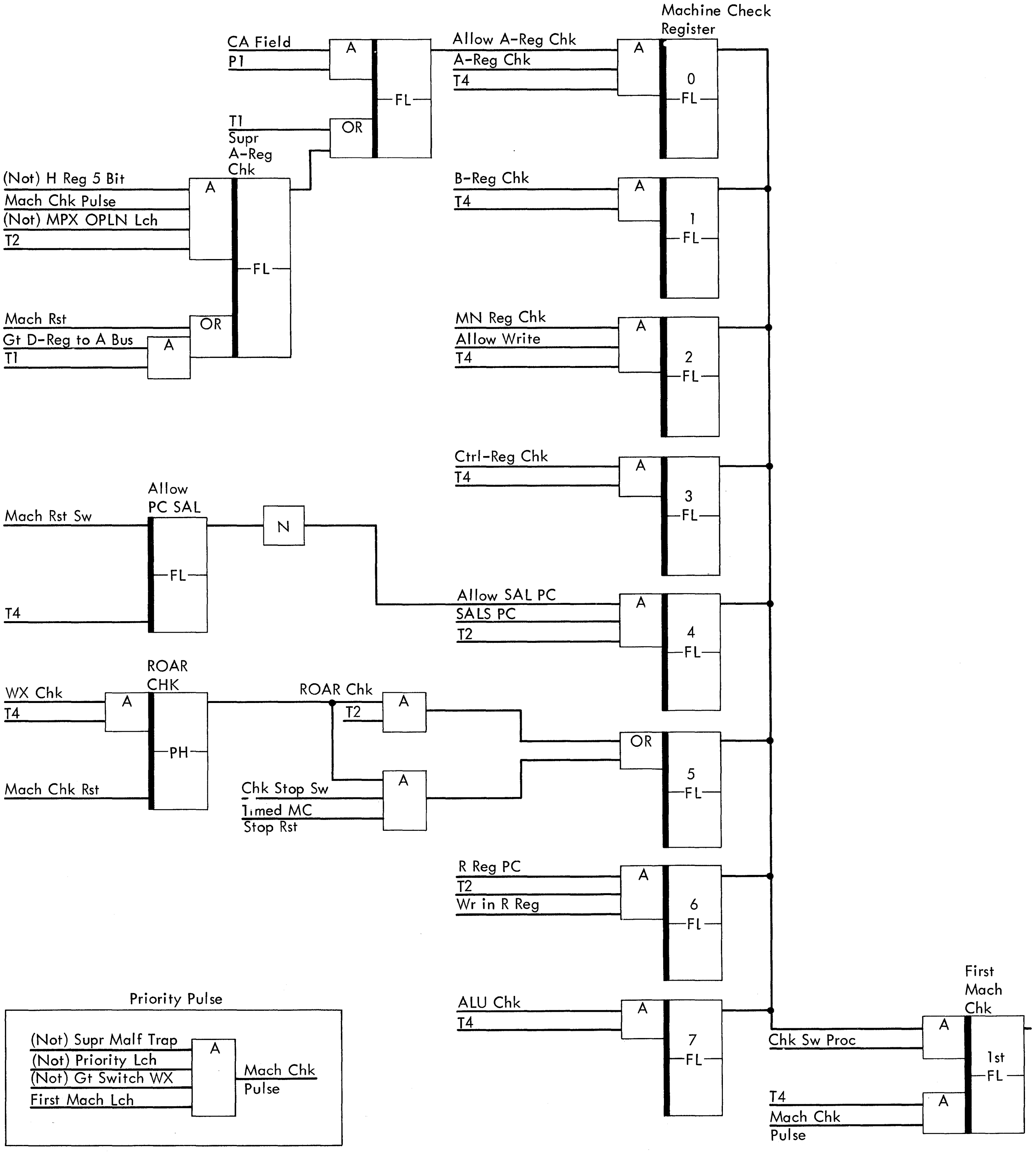


Figure 3-53. MC Register

The next objective is to enter the MC micro program. The insert in Figure 3-53 shows the machine-check-pulse line. This line brings up controls to enter the MC micro program. 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 micro program (machine -check pulse) is not active.

Note: The machine-check latches are reset at P4 time with a reset line.

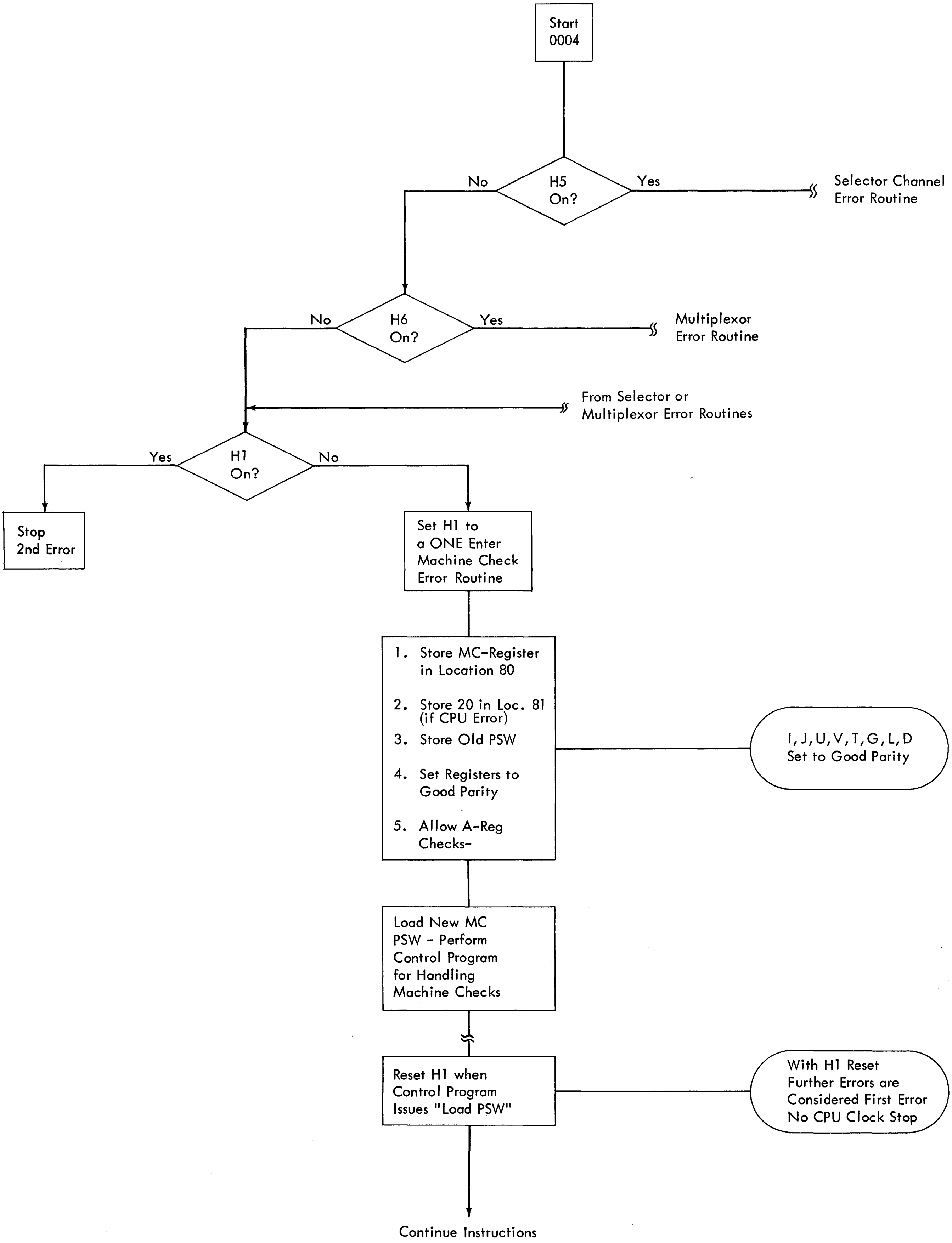


Figure 3-54. MC Micro Program

MACHINE CHECK MICRO PROGRAM

The starting address for this program is 0004 (Figure 3-54). Position 5 of the H-register is checked to see if it is set. If this position is set, it means that the error occurred during a selector-channel-ROS request and further testing is necessary to determine whether the error is to be charged to the channel or the CPU. 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, 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. Let us assume there are no further errors and continue with the flow chart. The information in the MC register is stored in location 0080 (HEX). A 20 (HEX) is stored in location 0081 (HEX) to signify that the failure is charged to the CPU. 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 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.

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 3-55). 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 hard-stop 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.

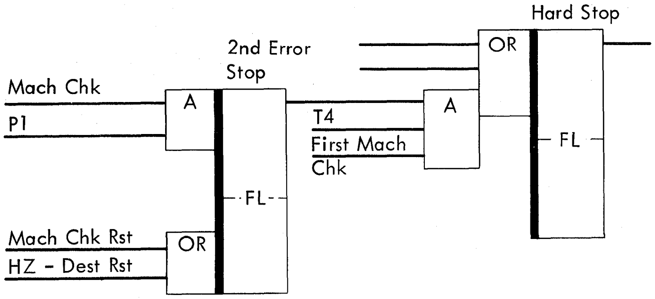


Figure 3-55. 2nd Error Stop Latch

REVIEW QUESTIONS - MACHINE CHECK HANDLING

1. Why are further A-register checks blocked when entering the machine check microprogram?

2. How is position 7 of the machine check register set?

3. There are several conditions that must be met before the machine check microprogram can be entered. Name two.

4. How are H5 and H6 used in the MC micro program?

5. How is further A-register checking allowed by the MC microprogram?

6. How is H1 used by the MC microprogram?

FORCED MICRO PROGRAM ENTRIES

- Ten ROAR addresses may be forced.
- Priorities are executed in order of importance.
- Waiting priorities are stacked.

What is a priority? By definition, a priority is that which is superior in rank, privilege, or position. As an example: Assume that an instruction micro program is being executed and a machine check occurs. The instruction micro program must be halted while a trap is taken to another micro program that will handle the error. Since the original program is interrupted, the machine check must have the higher priority.

There are nine orders of priority (Figure 3-56). Each order of priority will set a specific address in ROAR. This is done by setting a particular position 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 zero the UCW's, clear 1050 locations in local storage, set PSW bit to zero, etc.

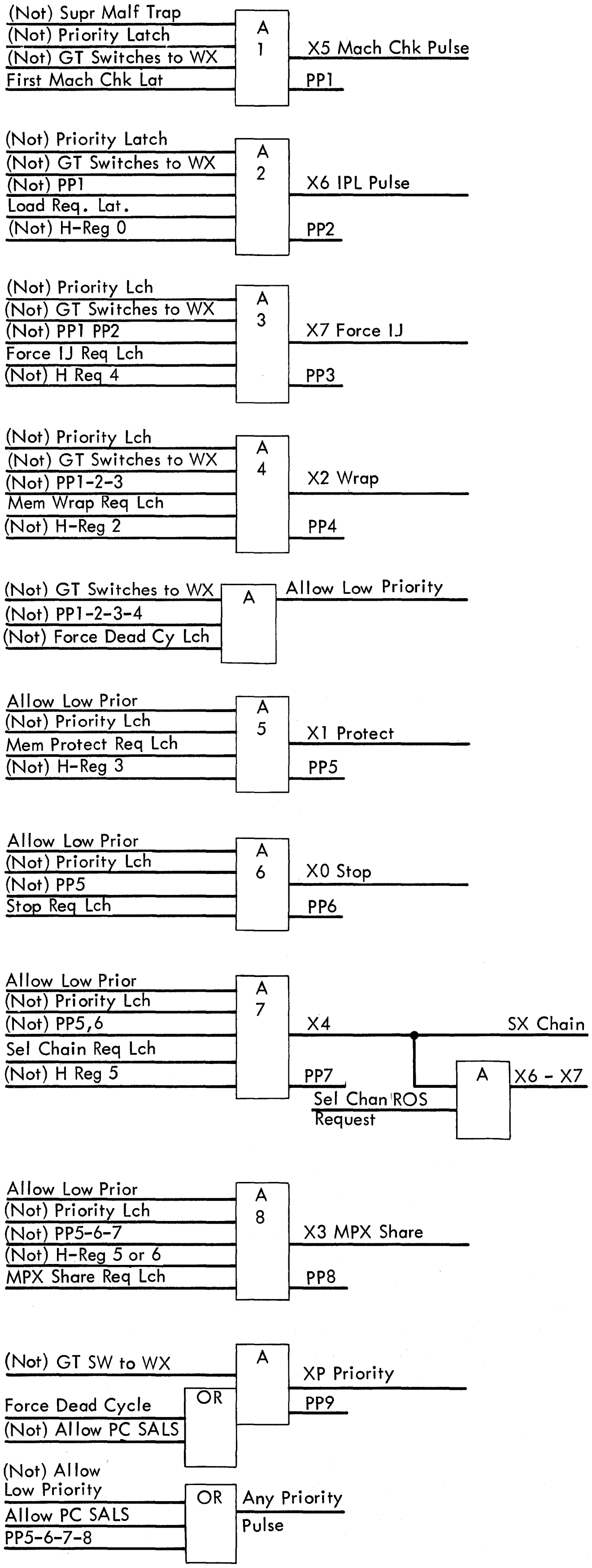


Figure 3-56. Priority Micro Program Entries

The other priorities are shown top to bottom 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 micro program 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 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 block further priorities, and is discussed later.

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 3-57). These latches are needed because several priorities may occur at one time, but only one can be handled. Notice that to set the stacking latch for a MPX-share request (PH7), position 6 of the H-register must not be set. Early in the MPX-share request micro program, 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-RCS request, the micro program that handles the request sets position 5 of the H-register. This not only blocks further selector-channel RCS-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 T1 time, with a process-stop-loop active. This line is active when the mnemonic S STOP is encountered in the micro program.

2. At P4 time, with the WX SABC latch on. This latch is turned ON at T1 time if WX must be set manually.

3. 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).

REVIEW QUESTIONS--FORCED MICROPROGRAM ENTRIES

1. What address is set in ROAR for MPX share request?

2. Which function has highest priority?

3. Priorities are stored while awaiting execution by means of _____ latches.

4. How is the priority reset control latch turned on?

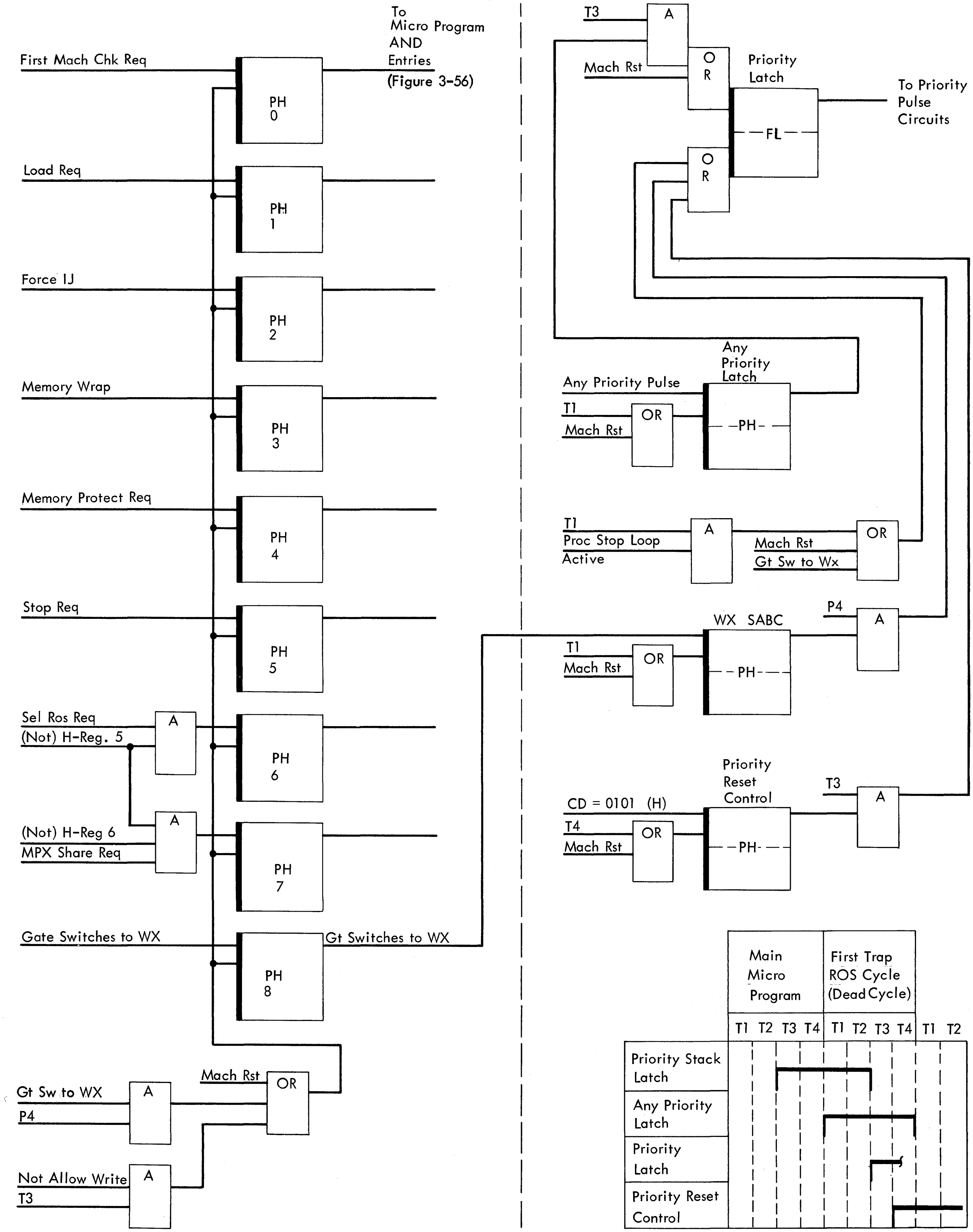


Figure 3-57. Priority Stack Registers and Controls

CVERALL TIMING RELATIONSHIPS

Earlier, wnen we discussed core storage, we emphasized the fact that it is an independent unit. It communicates with the CPU through the memory-to-CPU interface. Core storage has its own clock that is started by a signal from the CPU. This means that ROS requests core storage service on one time basis (CPU clock), whereas core storage answers this request on another time basis (memory clock). Although these two clocks operate independently, their timings are definitely related. When the ROS reads out a word that requests core storage service, the memory clock is signalled to start at the beginning of the next CPU cycle. In Figure 3-58 read call is developed and sent to the core storage unit as a result of the statement IJ->MN. Read call starts the memory dock and specifies a memory read cycle. While the core storage unit is reading out the requested byte, the ROS is reading out and executing the second micro-word. When the memory has the requested byte ready (about T4 time), it signals the CPU with the data ready signal. The CPU then accepts the byte and gates the byte to the R-register.

The second micro-word contained the statement write. Thus when the core storage unit finishes its read cycle, the write statement develops the write call signal to start the memory clock for a memory write cycle. These timing relationships are important because they point out how the core storage cycle seems to be one cycle behind the CPU cycle.

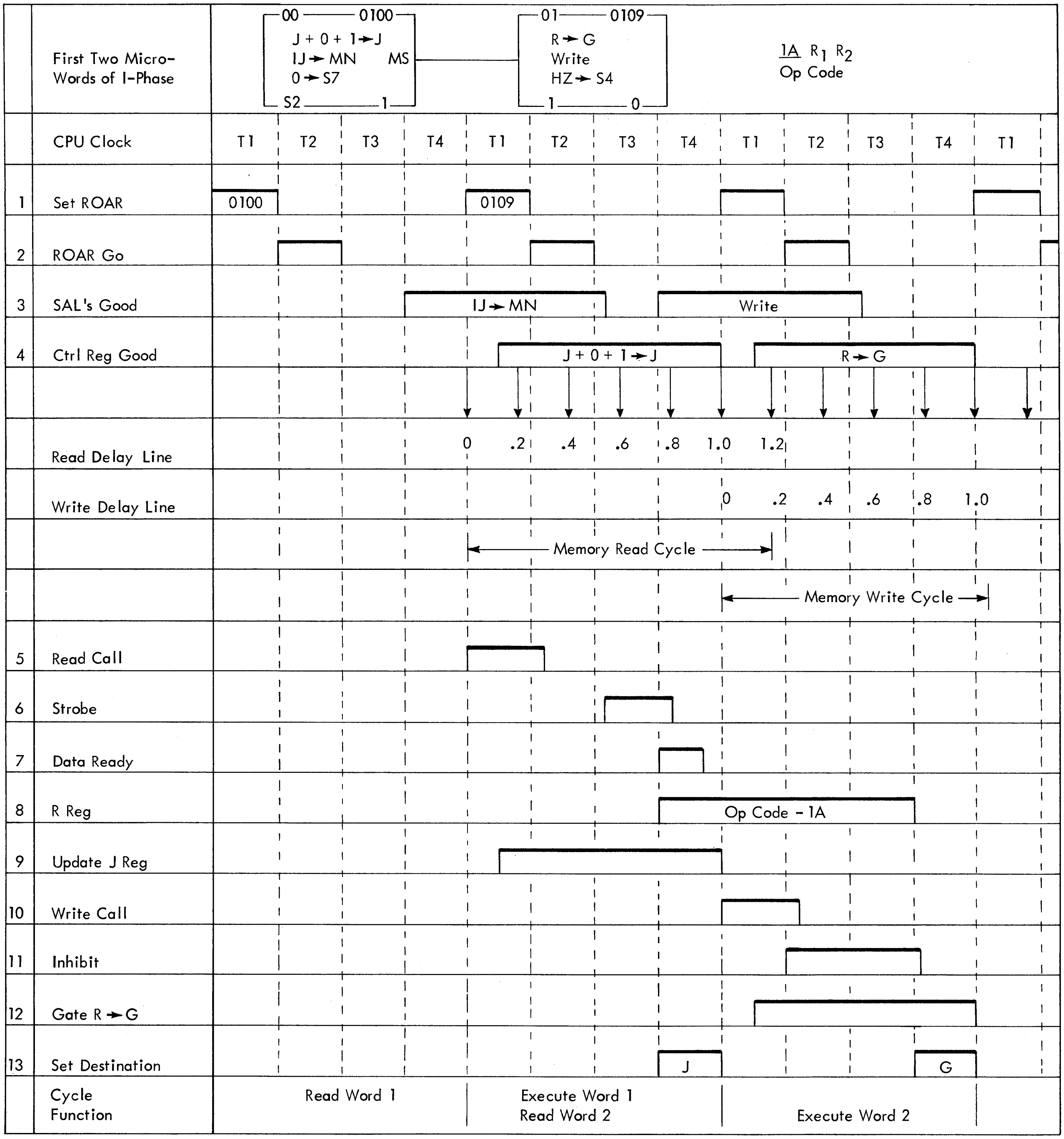


Figure 3-58. RCS to Memory Timings

POWER-ON SEQUENCE

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 switch picks RY3 to initiate the power-on sequence (Figures 4-1 and 4-2). Relay RY3 in turn picks contactor K2 applying power to the blower motors and the converter-inverter, and starts the delayed pick of TDR1 for the two second power-on-reset. By this time all DC voltages are applied to the circuits except +40 volts and the special -3 sequence voltage for Gate B (board E3 only). 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 +40 volts and the -3 volt sequence voltage to the circuits.

At the end of the two second time delay, the -3 volts is removed from the -Y reset line by the pick of TDR1 and relay RY5 picks to indicate that the processor is ready.

During this power-on sequence, each I/O control unit on the channels will power up and the last I/O control unit in the chain completes a circuit to turn on the power-on light on the console panel.

POWER-OFF SEQUENCE

Pressing the power-off switch on the console panel sequences power down in the CPU and the power-on control relays (Figures 4-1 and 4-3). 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 switch 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.

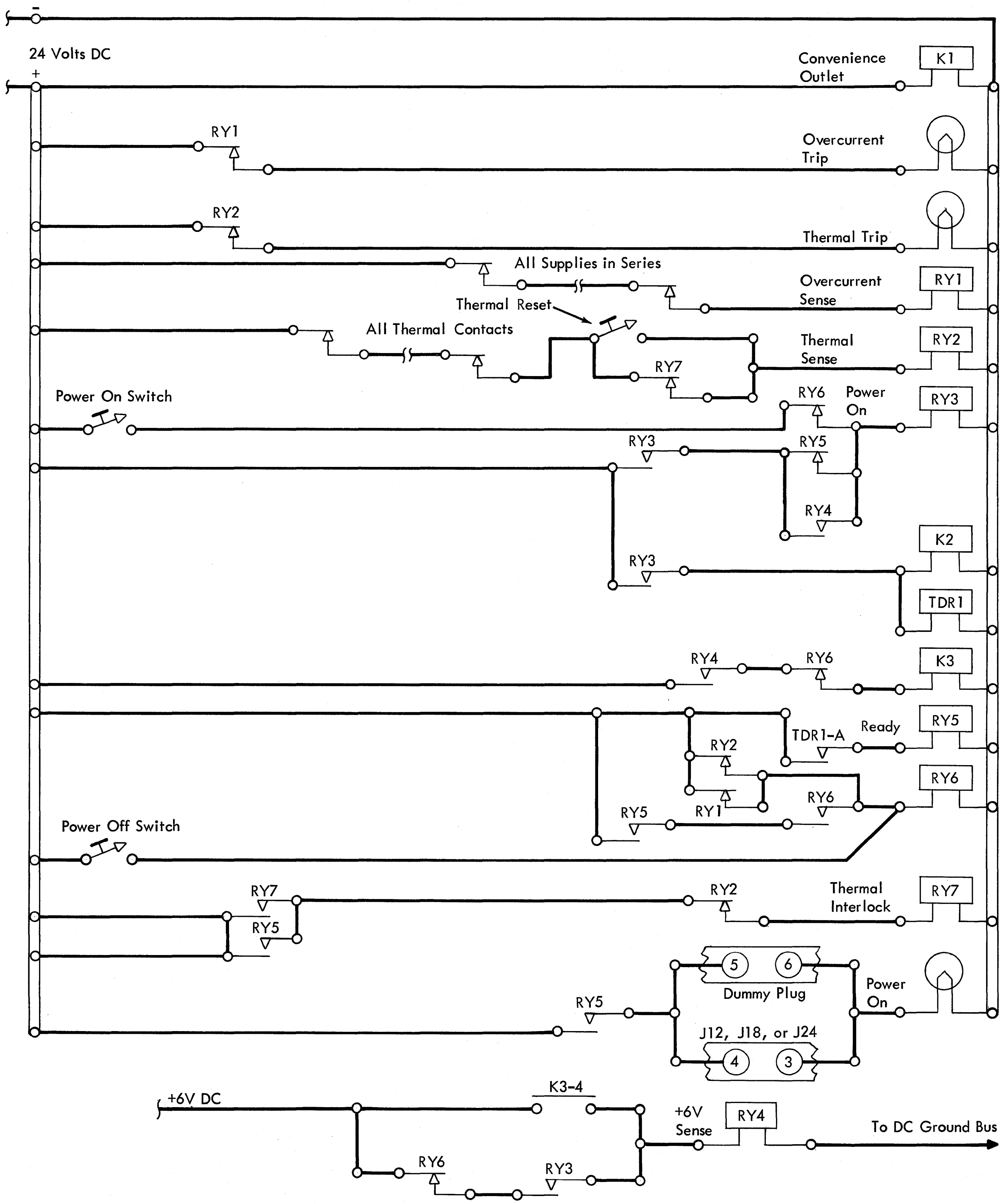
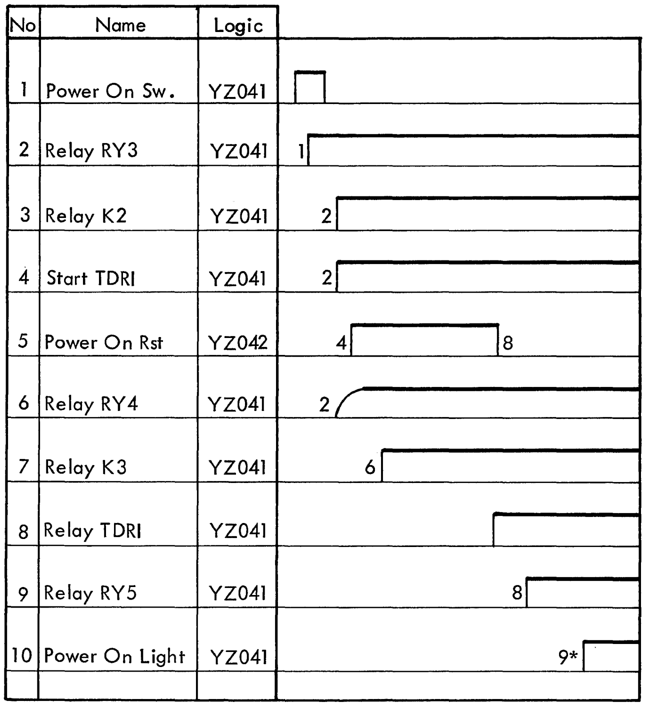


Figure 4-1. Power-On-Off Control



*Depends on number of I/O units attached.

Figure 4-2. Power-On Sequence

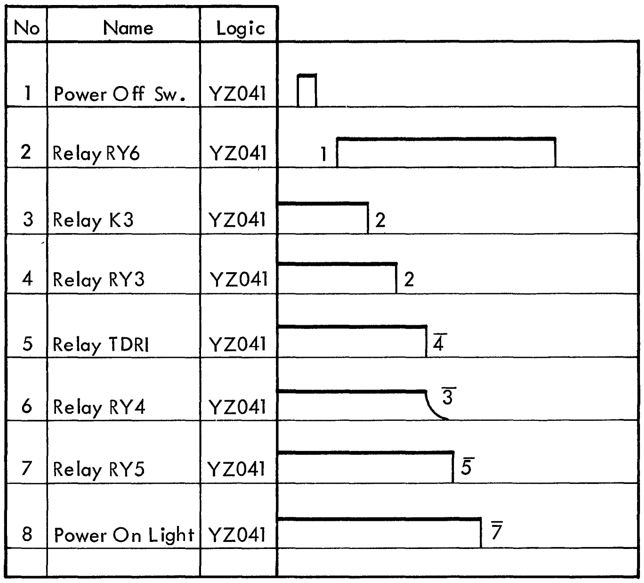


Figure 4-3. Power-Cff Sequence

MARGINAL CHECKING

Marginal checking consists of varying certain supply voltages so that intermittent failures become solid failures, permitting more rapid trouble diagnosis. Marginal checking has two basic applications:

1. Unscheduled to minimize system downtime when troubleshooting intermittent failures.

2. Scheduled maintenance: to decrease system failures by eliminating troubles that would limit the marginal check range.

The marginal-check scheme used in the 2030 is a part of the circuit design and requires no separate voltage supply. Three of the supply voltages are variable within a certain range:

The +60 volt supply is variable ± 15 percent.

The +40 volt supply is variable ± 15 percent.

The +6M volt supply is variable from 4 to 7 volts.

Each supply can be varied by means of a potentiometer located on the amplifier card in the power supply module.

POWER DISTRIBUTION

The power distribution throughout the 2030 is shown in Figure 4-4.

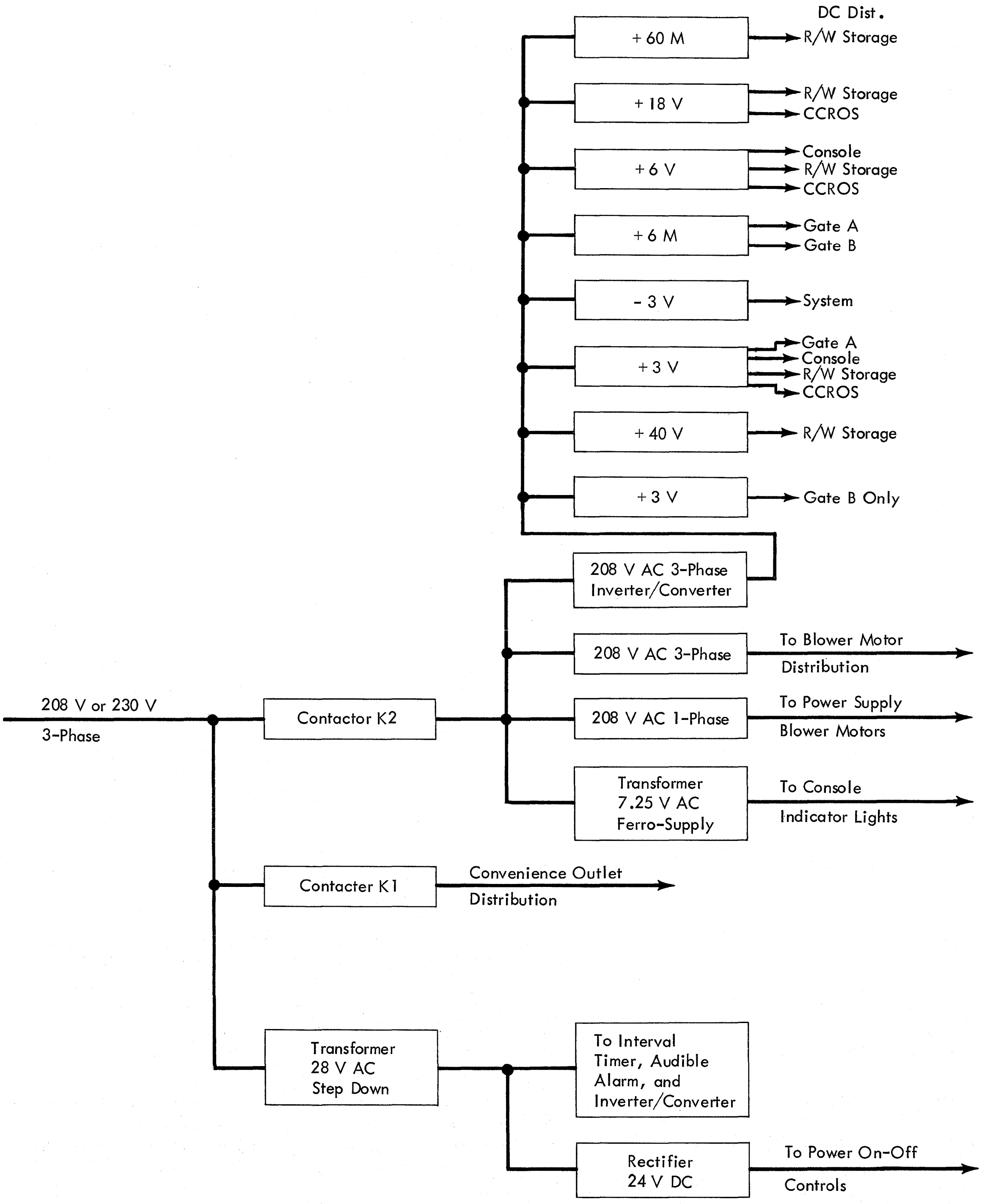


Figure 4-4. Power Distribution

1401, 1440, AND 1460 COMPATIBILITY FEATURES

- The System/360 is in compatibility mode when W3 of the WX Reg is on.
- Cnly one compatibility feature can be present on the machine.
- Any system reset causes the 2030 to leave 1401 mode.
- IJ, LT and UV perform the functions of I-Star, A-Star,and B-Star, respectively.
- Sense switch settings for 1401 object program are stored in K8, local storage.

The 1401, 1440, and 1460 Compatibility Features are special features available to the customer to allow him to run 1400 system (1401, 1440, 1460) object programs. The facilitate the processing of 1400 system programs, a second 4K RCS has been added to the 2030. This additional ROS contains the micro programming necessary to process 1400 system instructions. The 2030 can be put into compatibility mode by turning on the W3 bit of the W-register. This bit causes the added 4K RCS to be addressed, and controls all mode dependent functions. Hardware is provided to allow W3 to be sent from:

1. the console switches
2. the micro-program control UV->WX
3. the micro-program control CA->WW.

When W3 is set from the console switches, the position of rotary switch F determines W3 (ie: F at an odd hex digit will turn on W3). The status of U3 determines W3 when the micro-program statement UV->WX is used to turn on compatibility mode. When the micro-program program statement CA->W is used, the status of AA determines the status of W3. Any system reset turns off W3 and causes the 2030 to leave compatibility mode. Recycle reset, the micro-program 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, and UV performs the functions of B-star. Hardware is provided sc that the L and T registers may be gated as a pair into the MN-register when the T-register is named as the source (ie: T->N LS causes the address contained in LT to gate to MN). LT performs the functions of the A-star in compatibility mode.

To run 1401, 1440 or 1460 object programs on the 2030, an initializing program is loaded ahead of the object program. The purpose of the initializing program is to load the conversion tables and address constants necessary to perform 1401, 1440, or 1460 instructions into the MPX 1 and local storage areas of main storage. The initializing deck also defines by means of control cards, the characteristics of the 1400 system being simulated (ie: memory size, special features, and the I/O configuration.)

IBM 1400 system sense switches are set local storage by use of switch F on the console. The procedure is : set switch F to the alpha-character designation of the sense switch desired, press the interrupt key; press the start button. The sense switch has now been set. To turn off, the same procedure is used, except switch F should be dialed 180 degrees opposite the alpha character designation of the sense switch that you wish to turn off (ie, switch F to C turns on sense switch C, switch F to C (4) turns off sense switch C.) At the completion of altering sense switches, the R-register displays 01110111, and the A-register displays the whole sense switch byte, (K8 local storage) to indicate the status of the sense switches.

After altering the desired sense switches, pressing the start key causes the processor to continue.

For additional information on console operation, read the SRL publication, System/360 Model 30 1401 Compatibility Feature,Form A24-3255. The topic "Operating Considerations" covers the complete console operation for the Compatibility mode.

CHARACTER CONFIGURATION

- The internal character code for compatibility mode is EBCDI code.
- Bit 1 of the character off represents a WM associated with the character.
- Character conversion from EBCDI to BCD and vice-versa can be accomplished through a table lookup if conversion is necessary.

Characters in core storage in compatibility mode are represented in EBCDI code configuration. (See Figure 5-1.)

All 1400 systems characters without word marks have bit 1 of the byte on. If a word mark is associated with the character, it is represented by having the bit 1 of the byte off. The character A without a word mark is represented as 11000001 in EBCDI code, while the character A with a wordmark is represented as 10000001 in EBCDI code.

	WITH WORDMARK				NO WORDMARK				WITH WORDMARK				NO WORDMARK			
4567 ↓	0123→															
	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	Ƶ	9	I	R	Ƶ	9
1010				b̸				b̸								
1011	.	$	,	#	.	$	,	#								
1100	⌑	*	%	@	⌑	*	%	@								
1101	[	]	∨	:	[	]	∨	:								
1110	<	;	\	>	<	;	\	>								
1111	⯒	△	⧻	√‾	⯒	△	⧻	√‾								

Figure 5-1. 1400 Defined Characters

			0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
LOCAL STORAGE	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-LO	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	56	12	68	24	80	36	92	48	04	60	16	72	28	84	40
	BCD TO EBCDI	4X	40	F1	F2	F3	F4	F5	F6	F7	F8	F9	F0	7B	7C	7D	7E	7F
		5X	7A	61	E2	E3	E4	E5	E6	E7	E8	E9	E0	6B	6C	6D	6E	6F
		6X	60	D1	D2	D3	D4	D5	D6	D7	D8	D9	D0	5B	5C	5D	5E	5F
		7X	50	C1	C2	C3	C4	C5	C6	C7	C8	C9	C0	4B	4C	4D	4E	4F
		8X	TAPE								0	1	2	3	4	5	6	7
		9X									8	9	10	11	12	13	14	15
		AX	WORKING STORAGE								16	17	18	19	20	21	22	23
		BX	00	96	92	88					24	25	26	27	28	29	30	31
	OP CODE TABLE	CX	?)1C	A)18	B)08	C)1F	D)12	E)16	F)2A	34	H)B1	34	34	.)02	⌑)15	34	34	34
		DX	!)1D	34	K)29	L)90	M)80	N)06	34	P)1E	Q)F1	34	34	34	34	34	34	34
		EX	34	/)05	S)19	34	U)20	V)3A	W)38	34	Y)13	Z)17	34	,)04	%)1B	34	34	34
		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
MPX 1 STORAGE		0X	00	05	01	06	02	07	03	08	04	09						
		1X																
	HUNDS-HI	2X	Z	Z+00	Z+00	Z+01	Z+01	Z+01	Z+02	Z+02	Z+03	Z+03						
		3X																
	EBCDI TO BCD	4X	00	40	40	6B	6C	6D	6E	6F	80	60	40	3B	3C	3D	3E	3F
		5X	30	40	40	5B	5C	5D	5E	5F	40	40	40	2B	2C	2D	2E	2F
		6X	20	11	40	4B	4C	4D	4E	4F	90	50	40	1B	1C	1D	1E	1F
		7X	40	40	7A	7B	7C	7D	7E	7F	40	40	10	0B	0C	0D	0E	0F
		8X									0	1	2	3	4	5	6	7
		9X	FILE UNIT 0 ADDER	UNIT 0 CYL	FILE UNIT 1 ADDR	UNIT 1 CYL	FILE UNIT 2 ADDR	UNIT 2 CYL	FILE UNIT 3 ADDR	UNIT 3 CYL	8	9	10	11	12	13	14	15
		AX	CYL # 00	CYL # 32	CYL # 0A	CYL # 3C	CYL # 14	CYL # 46	CYL # 1E	CYL # 50	16	17	18	19	20	21	22	23
		BX	H	R	00	DL	DL				24	25	26	27	28	29	30	31
	EBCDI TO BCD	CX	3A	31	32	33	34	35	36	37	38	39	E0	40	40	40	45	5C
		DX	2A	21	22	23	24	25	26	27	28	29	D0	40	40	40	46	5D
		EX	1A	40	12	13	14	15	16	17	18	19	C0	40	40	40	4F	56
		FX	0A	01	02	03	04	05	06	07	08	09	F0	40	40	40	44	5F

K=ADDRESSABLE BYTE UTILIZATION	
LOCAL STORAGE	MPX 1 STORAGE
0 I BACK-UP	0 ERROR CODE
1 J " "	1
2 U " "	2 SEE SHEET BX
3 V " "	3 USED BY 1402-03
4 L " "	4 AND 1442-43
5 T " "	5
6 G " "	6 O STAR HI
7 S " "	7 O STAR LO
8 SENSE SW BYTE	8 FILE UNIT 4 ADDR
9 HI-LO-EQ BYTE	9 UNIT 4 CYLINDER
10 MEM SIZE BYTE	10 I/O ERROR
11 FILE BRANCH BYTE	11 TAPE TRK IN ERROR
12 1401 CONTROL	12
13 PMS CONTROL	13 A HUND'S BK-UP
14 D BACKUP	14 B HUND'S BK-UP
15 ALLOW I/O TRAPS	15 1050 STATUS
16 WORKING STORAGE	16 CYL # 28
17 " "	17 CYL # 5A
18 " "	18 PREVIOUS OPR
19 " "	19
20 CONSTANT-1F	20 00
21	21 00
22	22 C
23	23 H
24	24 WORKING STORAGE
25	25 " "
26 CONSTANT-0F	26 " "
27	27 " "
28	28 " "
29	29 " "
30 CONSTANT-2E	30 " "
31	31 " "

NOTES 1. X INDICATES THAT THE QUANTITY IN PARENTHESIS IS CROSSED IN THE TABLE
2. Z=MEMORY BIAS-HI
Y=MEMORY BIAS-LO
3. ENTRIES IN HUNDS-HI TABLE INCLUDE CARRY FROM HUNDS-LO TABLE

1401 CONTROL BYTE

BIT 0-I/O CHECK STOP
1-ADV PROG FEATURE
2-EXPANDED EDIT
3-MODE SW ON INVALID OPS
4-COLUMN BINARY
5-MODE SW ON HALT
6-TAPE ON S X 2
7-MODE SW ON ERROR STOPS

Figure 5-2. Auxiliary Storage Map for Character Conversion

The internal code used in the 2030 for the compatibility feature is the EBCDI code. Because the 2030 is basically a binary system, occasionally a translation of character codes from EBCDI to BCD and back again is necessary to process certain 1400 system instructions, such as bit test, in the 2030.

Most conversions are done through a table lookup procedure, utilizing tables in local storage. These tables are read into storage as part of the initializing routine that was run in ahead of the 1400 system object program. To illustrate the use of the table using the local storage map (Figure 5-2), we will convert a character from EBCDI to BCD. The character "C" in EBCDI CODE IS A C3 (HEX). By checking the EBCDI to BCD table for C3, we will pull out of MPX1 a 33, or 00110011, which is the BCD configuration of a "C"

BA	8421
0011	0011

In utilizing the conversion tables, if a WM is present with the character, the micro program eliminates it from the character before the table lookup is done to convert the character. In the EBCDI to BCD conversion table, any 01000000 configuration read out of the table is detected as an invalid BCD configuration, and will read out as a blank in BCD.

1400 SYSTEM ADDRESSING

- All 1400 systems object programs are loaded into upper storage of the 2030.
- Tens and hundreds address bytes have the upper and lower 4 bits of the byte crossed in local storage.
- Address bytes in local storage contain a bias constant.

All 1400 systems programs are loaded into upper storage in the 2030. If a 1401 program written for 4K of storage is going to be run on a 2030 with 16,384 positions of storage, the program is stored in IBM 2030 storage locations 12,384 to 16,383. This is done to allow the 2030 to detect 1401 storage wrap errors through existing circuits. As previously mentioned, the 2030 uses a conversion table in the local storage and MPX 1 areas of core storage to convert 1401 BCD addresses to 2030 binary addresses. This table also includes a storage bias constant 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 (Figure 5-3). To illustrate, assume we attempted to run 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
```

In our example of storage bias 3060, this could be further broken down to: 30 is bias for the high-order byte of the address, and 60 is bias for the low-order byte of the address. The local storage map for compatibility mode refers to the high-order bias as Z, and the low-order bias as Y.

360 / 1400	Y BIAS	Z BIAS			
		16,384	32,768	65,536	8,192
16K	80	01	41	C1	
12K	20	11	51	D1	
8K	C0	20	60	E0	00
4K	60	30	70	F0	10
2K	30	38	78	F8	18
1.4K	88	3A	7A	FA	1A

Y = Bias for Low-Order Byte in Hex.
Z = Bias for High-Order Byte in Hex.

Figure 5-3. Compatibility Mode Memory Bias

When the I-cycles micro program reads out IBM 1400 system addresses and converts them to binary addresses it interrogates the hundreds digit on two occasions, since the hundreds digit affects the value stored in both the high-order byte of the address and the low-order byte of the address. For example, hundreds 3 inserts 0000 0001 in the high-order byte, and 0010 100 in the low-order byte.

Again, assuming a 2030 with 16,384 positions of storage simulating a 1401 with 4K memory, let us examine the A-star address formed during I-cycles for 1401 instruction / 122.

The micro program reads out the hundreds position of the 1401 instruction, in this case a 1. Next, an address is formed to address local storage. The micro program uses the hunderds digit to form bits 4-7 of the address, and since this is the hundreds position, forces the bits 0-3 to a 2 (HEX). The resultant address 21 in HEX is used to address local storage. Position 21 in local storage brings out a C4 (HEX). C4 represents the Y bias 60 plus the binary equivalent of 100 (64 in HEX). If the same address, 21, is now used to address the MPX 1 portion of local storage, it will bring out the high-order byte of the address being formed. In this case, 21 brings out 30. 30 represents the Z bias plus 00 hundred. At this point the micro program has developed the address 3064 (HEX). Now the micro program forms an address of CX, where X is the units digit of the 1401 address. In this case the address is 02 (HEX). Addressing local storage with 02 brings out 02, which is added to 30C4 already stored, giving new value of 30C6. Finally, the micro program processes the tens position of the 1401 address by forcing a constant of 1 in the high order of the local storage address and inserting the tens digit of the 1401 address in the low order of the byte. In this case, 12 is formed to address the tens conversion table, bringing out a 14 (HEX). This is added to 30C6. The result, 30DA, is inserted into UV (1401 B-Star). The micro program tests to see if it is forming an A-field address, and if so, the micro program takes the contents of UV and inserts it into LT (1401 A-Star). See Figure 5-4.

	TO ADDRESS BUMP		CONVERTED 1400 ADDRESS				
	M	N					
Hrds Low	2	1	0	0	C	4	Hrds Low Bias 60 + 100
Hrds High	2	1	3	0	C	4	Hrds High Bias 30 + 0 Hrds
Units	0	2	3	0	C	6	Hrds 30 C4 + 2
Tens	1	2	3	0	D	8	30 C6 + 20

Figure 5-4. 1400 System Address Conversion

Zone bits in the hundreds and units positions of the 1401 are tested for by the micro program and are not found in the address conversion tables (Figure 5-5).

			0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
LOCAL STORAGE	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-LO	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	56	12	68	24	80	36	92	48	04	60	16	72	28	84	40
	BCD TO EBCDI	4X	40	F1	F2	F3	F4	F5	F6	F7	F8	F9	F0	7B	7C	7D	7E	7F
		5X	7A	61	E2	E3	E4	E5	E6	E7	E8	E9	E0	6B	6C	6D	6E	6F
		6X	60	D1	D2	D3	D4	D5	D6	D7	D8	D9	D0	5B	5C	5D	5E	5F
		7X	50	C1	C2	C3	C4	C5	C6	C7	C8	C9	C0	4B	4C	4D	4E	4F
		8X	TAPE								0	1	2	3	4	5	6	7
		9X									8	9	10	11	12	13	14	15
		AX	WORKING STORAGE								16	17	18	19	20	21	22	23
		BX	00	96	92	88					24	25	26	27	28	29	30	31
	OP CODE TABLE	CX	?)1C	A)18	B)08	C)1F	D)12	E)16	F)2A	34	H)B1	34	34	·)02	⌑)15	34	34	34
		DX	I)1D	34	K)29	L)90	M)80	N)06	34	P)1E	Q)F1	34	34	34	34	34	34	34
		EX	34	/)05	S)19	34	U)20	V)3A	W)3B	34	Y)13	Z)17	34	,)04	%)1B	34	34	34
		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
MPX 1 STORAGE		0X	00	05	01	06	02	07	03	08	04	09						
		1X																
	HUNDS-HI	2X	Z	Z+00	Z+00	Z+01	Z+01	Z+01	Z+02	Z+02	Z+03	Z+03						
		3X																
	EBCDI TO BCD	4X	00	40	40	6B	6C	6D	6E	6F	B0	60	40	3B	3C	3D	3E	3F
		5X	30	40	40	5B	5C	5D	5E	5F	40	40	40	2B	2C	2D	2E	2F
		6X	20	11	40	4B	4C	4D	4E	4F	90	50	40	1B	1C	1D	1E	1F
		7X	40	40	7A	7B	7C	7D	7E	7F	40	40	10	0B	0C	0D	0E	0F
		8X									0	1	2	3	4	5	6	7
		9X	FILE UNIT 0 ADDER	UNIT 0 CYL	FILE UNIT 1 ADDR	UNIT 1 CYL	FILE UNIT 2 ADDR	UNIT 2 CYL	FILE UNIT 3 ADDR	UNIT 3 CYL	8	9	10	11	12	13	14	15
		AX	CYL# 00	CYL# 32	CYL# 0A	CYL# 3C	CYL# 14	CYL# 46	CYL# 1E	CYL# 50	16	17	18	19	20	21	22	23
		BX	H	R	00	DL	DL				24	25	26	27	28	29	30	31
	EBCDI TO BCD	CX	3A	31	32	33	34	35	36	37	38	39	E0	40	40	40	45	5C
		DX	2A	21	22	23	24	25	26	27	28	29	D0	40	40	40	46	5D
		EX	1A	40	12	13	14	15	16	17	18	19	C0	40	40	40	4F	56
		FX	0A	01	02	03	04	05	06	07	08	09	F0	40	40	40	44	5F

K=ADDRESSABLE BYTE UTILIZATION	
LOCAL STORAGE	MPX 1 STORAGE
0 I BACK-UP	0 ERROR CODE
1 J " "	1
2 U " "	2 SEE SHEET BX
3 V " "	3 USED BY 1402-03
4 L " "	4 AND 1442-43
5 T " "	5
6 G " "	6 O STAR HI
7 S " "	7 O STAR LO
8 SENSE SW BYTE	8 FILE UNIT 4 ADDR
9 HI-LO-EQ BYTE	9 UNIT 4 CYLINDER
10 MEM SIZE BYTE	10 I/O ERROR
11 FILE BRANCH BYTE	11 TAPE TRK IN ERROR
12 1401 CONTROL	12
13 PMS CONTROL	13 A HUND'S BK-UP
14 D BACKUP	14 B HUND'S BK-UP
15 ALLOW I/O TRAPS	15 1050 STATUS
16 WORKING STORAGE	16 CYL # 28
17 " "	17 CYL # 5A
18 " "	18 PREVIOUS OPR
19 " "	19
20 CONSTANT-1F	20 00
21	21 00
22	22 C
23	23 H
24	24 WORKING STORAGE
25	25 " "
26 CONSTANT-0F	26 " "
27	27 " "
28	28 " "
29	29 " "
30 CONSTANT-2E	30 " "
31	31 " "

NOTES 1. X INDICATES THAT THE QUANTITY IN PARENTHESIS IS CROSSED IN THE TABLE
2. Z=MEMORY BIAS-HI
Y=MEMORY BIAS-LO
3. ENTRIES IN HUNDS-HI TABLE INCLUDE CARRY FROM HUNDS-LO TABLE

1401 CONTROL BYTE

BIT 0-I/O CHECK STOP
1-ADV PROG FEATURE
2-EXPANDED EDIT
3-MODE SW ON INVALID OPS
4-COLUMN BINARY
5-MODE SW ON HALT
6-TAPE ON S X 2
7 -MODE SW ON ERROR STOPS

Figure 5-5. Auxiliary Storage Map for Address Conversion

ADDRESS ERROR DETECTION

- An invalid 1400 character (8F) is placed in 1401 address 000 minus 1 for error detection.
- 2030 circuits detect a storage wrap error.
- R3 bit on signifies and invalid address character.

Because the high-order position of the 1400 system storage is physically the same location as the high-order position of 2030 memory, 1400 systems storage wrap errors are detected in the 2030 through existing error detection circuits. To detect an error when address 000 is modified by minus one, an invalid character has been placed in 2030 memory one core location below the address assigned to 1400 storage location 000. If this location is addressed, the invalid character causes a wrap trap that initializes the halt routine. The invalid character is placed in storage by the 1401 initializing program that loads local storage and puts the machine into compatibility mode.

When converting 1400 system addresses to 2030 addresses, another error detection device detects invalid characters in the address. When the micro program converts a 1400 system address to a binary address, it is done in three stages. First, the hundreds digit is used to address the table and a binary equivalent is extracted. Next, a binary equivalent of the tens digit is extracted and added to the hundreds value. Finally, the units digit is obtained and added to the total of the tens and hundreds already obtained. When the hundreds digit addresses local storage to obtain a binary equivalent, the tens and units of the address are disregarded and assumed to be zero. (ie: 5 in the hundreds position brings out the binary equivalent of 500 plus bias). When the tens digit addresses local storage for conversion, the units position is assumed to be zero. Since the tens and hundreds digits extract binary values that are round numbers (numbers that end in zero), the 1 bit is not turned on. To provide for error detection the hexadecimal values in the conversion table for the tens and hundreds have been crossed. An example is : the tens 2 (20) equivalent in hex is 1 4, but it is stored in local storage as 4 1.

When 4 1 is read out of local storage, the R-register 3-bit (R3) is not turned on. Note also that the units, 0-9 if not crossed, will never turn on R3. Therefore, when addresses are read out, if R3 is on, it is an invalid address. Observe the local storage map for compatibility mode and note that all invalid digit values in a 1400 system address turn on the R3 bit when they are read out (Figure 5-5).

OP CODE RECOGNITION

- 1400 system Op codes are convert- to bit significant characters.
- Op code conversion table is stored in local storage.
- 1400 system Op code character bits 0 and 1 are forced on before using the character to address the conversion table.
- Converted Op code is stored in the G-register during I cycles.

1400 system Op codes in their EBCDI configuration would require extensive interrogation by the micro program to determine exactly what the Op code is, since their EBCDI bit configurations would not readily indicate what type of Op code the machine is handling. To make Op codes more easily identified as to the type, a table of all 1400 system Op codes has been made and stored in local storage. This table groups similar Op codes together. The bit configurations are bit sensitive for easy identification by the micro program. (See Figure 5-6.)

4567→

	0123	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
	0000			•	N$^{8}_{9}$	'	/						B				
Cpu Ops	0001			D	Y	#	¤	E	Z	A	S	@	%	?	!	P	C
I/O Ops	0010	U	1	2	3	4	5	6	7	PFR	K	F		(I/O) U	(I/O) M	(I/O) L	
Misc Ops	0011											V	W				
	0100					Invalid Ops											
	0101																
	0110																
	0111		Column Binary														
	1000	M															
	1001	L															
	1010																
	1011		H														
	1100																
	1101																
	1110																
	1111		Q														

Figure 5-6. Op Code Conversion Table

To understand the addressing of the Op code table in the local storage, first refer back to the EBCDI code of 1400 system character configurations (Figure 5-1). Notice that if the bit 0 of all 1400 system Op codes that did not have bit 0 on are forced on, characters in EBCDI with their bit 0 off can be overlaid with the rest of the EBCDI characters with the exception of the blank, +, and -, which are not valid 1400 system Op codes. (See Figure 5-7.)

BITS 4567	BITS 0123			
	1100	1101	1110	1111
0000	?	!	ǂ RM	0
0001	A	J	/	1
0010	B	K	S	2
0011	C	L	T	3
0100	D	M	U	4
0101	E	N	V	5
0110	F	O	W	6
0111	G	P	X	7
1000	H	Q	Y	8
1001	i	R	Z	9
1010				ƀ
1011	.	$	,	#
1100	⌑	*	%	@
1101	⊏	⊐	⋎ WS	:
1110	(	;	\	)
1111	ǂ GM	△ MC	⧻ SM	√ TM

Figure 5-7. Modified 1400 System Op Codes

When the micro program reads out the Op code in EBCDI form, it turns on the 0 and 1 bits of the Op code. Next, the EBCDI character formed in the previous step is used to address local storage and remove the new character that is stored in the G-register. (In the case of a blank, +, or -, the contents of the Op code table are ignored and the G-register is forced to an invalid Op code.) The new character has a bit configuration that is more readily tested to tell the type of operation desired. To illustrate the use of the Op code table, assume that the Op code read out of the 1400 system program is an edit Op. The hexadecimal bit configuration of an E with a WM in EBCDI is 85. Forcing on the bits 0 and 1 changes the configuration to C5. Using C5 to address the Op code table in local storage (Figure 5-8) a 16 is read out and stored in the G-register. The 16 is bit sensitive to the micro program as an edit Op code. Any Invalid EBCDI Op code configuration addressing the Op code table brings out a 34 byte that is recognized by the micro program as an error. To assist in understanding the method used by the 2030 to process 1400 system instructions in compatibility mode, logic flow charts are included in Figures 5-9, 5-10, and 5-11 for I-phase of a 1401 instruction and the execute phases of a Move operation and an Add operation. It should be stressed that these flow charts are for instructional use only and may not be exactly the same as the microprograms they represent.

		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
LOCAL STORAGE	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-LO 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	56	12	68	24	80	36	92	48	04	60	16	72	28	84	40
	BCD TO EBCDI 4X	40	F1	F2	F3	F4	F5	F6	F7	F8	F9	F0	7B	7C	7D	7E	7F
	5X	7A	61	E2	E3	E4	E5	E6	E7	E8	E9	E0	6B	6C	6D	6E	6F
	6X	60	D1	D2	D3	D4	D5	D6	D7	D8	D9	D0	5B	5C	5D	5E	5F
	7X	50	C1	C2	C3	C4	C5	C6	C7	C8	C9	C0	4B	4C	4D	4E	4F
	8X	TAPE								0	1	2	3	4	5	6	7
	9X									8	9	10	11	12	13	14	15
	AX	WORKING STORAGE								16	17	18	19	20	21	22	23
	BX	00	96	92	88					24	25	26	27	28	29	30	31
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
	DX	!)1D	34	K)29	L)90	M)80	N)06	34	P)1E	Q)F1	34	34	34	34	34	34	34
	EX	34	/)05	S)19	34	U)20	V)3A	W)3B	34	Y)13	Z)17	34	,)04	%)1B	34	34	34
	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
MPX 1 STORAGE	0X	00	05	01	06	02	07	03	08	04	09						
	1X																
	HUNDS-HI 2X	Z	Z+00	Z+00	Z+01	Z+01	Z+01	Z+02	Z+02	Z+03	Z+03						
	3X																
	EBCDI TO BCD 4X	00	40	40	6B	6C	6D	6E	6F	80	60	40	3B	3C	3D	3E	3F
	5X	30	40	40	5B	5C	5D	5E	5F	40	40	40	2B	2C	2D	2E	2F
	6X	20	11	40	4B	4C	4D	4E	4F	90	50	40	1B	1C	1D	1E	1F
	7X	40	40	7A	7B	7C	7D	7E	7F	40	40	10	0B	0C	0D	0E	0F
	8X									0	1	2	3	4	5	6	7
	9X	FILE UNIT 0 ADDER	UNIT 0 CYL	FILE UNIT 1 ADDR	UNIT 1 CYL	FILE UNIT 2 ADDR	UNIT 2 CYL	FILE UNIT 3 ADDR	UNIT 3 CYL	8	9	10	11	12	13	14	15
	AX	CYL# 00	CYL# 32	CYL# 0A	CYL# 3C	CYL# 14	CYL# 46	CYL# 1E	CYL# 50	16	17	18	19	20	21	22	23
	BX	H	R	00	DL	DL				24	25	26	27	28	29	30	31
	EBCDI TO BCD CX	3A	31	32	33	34	35	36	37	38	39	E0	40	40	40	45	5C
	DX	2A	21	22	23	24	25	26	27	28	29	D0	40	40	40	46	5D
	EX	1A	40	12	13	14	15	16	17	18	19	C0	40	40	40	4F	56
	FX	0A	01	02	03	04	05	06	07	08	09	F0	40	40	40	44	5F

K=ADDRESSABLE BYTE UTILIZATION	
LOCAL STORAGE	MPX 1 STORAGE
0 I BACK-UP	0 ERROR CODE
1 J " "	1
2 U " "	2 SEE SHEET BX
3 V " "	3 USED BY 1402-03
4 L " "	4 AND 1442-43
5 T " "	5
6 G " "	6 O STAR HI
7 S " "	7 O STAR LO
8 SENSE SW BYTE	8 FILE UNIT 4 ADDR
9 HI-LO-EQ BYTE	9 UNIT 4 CYLINDER
10 MEM SIZE BYTE	10 I/O ERROR
11 FILE BRANCH BYTE	11 TAPE TRK IN ERROR
12 1401 CONTROL	12
13 PMS CONTROL	13 A HUND'S BK-UP
14 D BACKUP	14 B HUND'S BK-UP
15 ALLOW I/O TRAPS	15 1050 STATUS
16 WORKING STORAGE	16 CYL # 28
17 " "	17 CYL # 5A
18 " "	18 PREVIOUS OPR
19 " "	19
20 CONSTANT-1F	20 00
21	21 00
22	22 C
23	23 H
24	24 WORKING STORAGE
25	25 " "
26 CONSTANT-0F	26 " "
27	27 " "
28	28 " "
29	29 " "
30 CONSTANT-2E	30 " "
31	31 " "

NOTES 1. X INDICATES THAT THE QUANTITY IN PARENTHESIS IS CROSSED IN THE TABLE
2. Z=MEMORY BIAS-HI
Y=MEMORY BIAS-LO
3. ENTRIES IN HUNDS-HI TABLE INCLUDE CARRY FROM HUNDS-LO TABLE

1401 CONTROL BYTE

BIT 0-I/O CHECK STOP
1-ADV PROG FEATURE
2-EXPANDED EDIT
3-MODE SW ON INVALID OPS
4-COLUMN BINARY
5-MODE SW ON HALT
6-TAPE ON S X 2
7 -MODE SW ON ERROR STOPS

Figure 5-8. Auxiliary Storage Map for Op Code Conversion

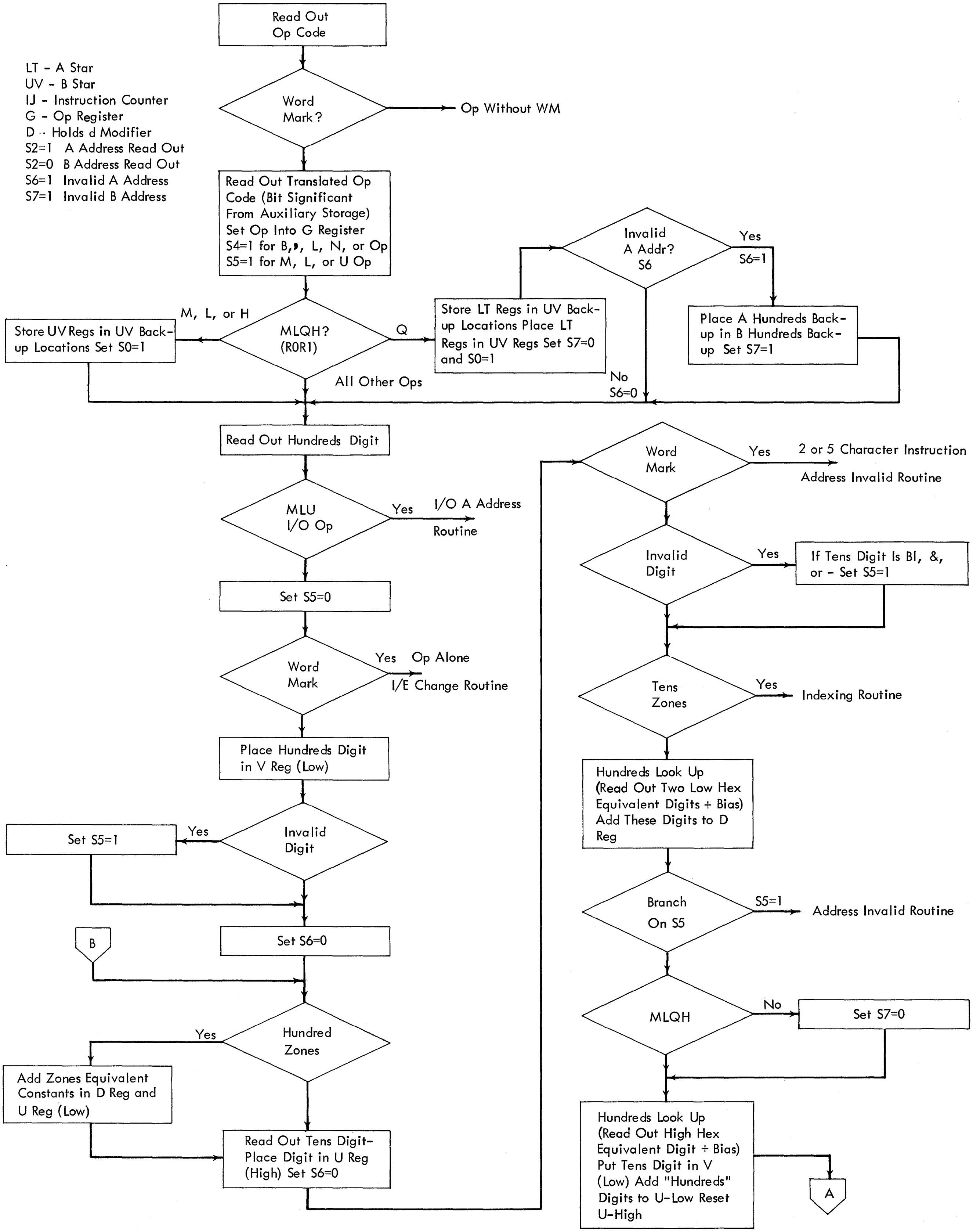

LT - A Star
UV - B Star
IJ - Instruction Counter
G - Op Register
D -- Holds d Modifier
S2=1 A Address Read Out
S2=0 B Address Read Out
S6=1 Invalid A Address
S7=1 Invalid B Address
Read Out Op Code
Word Mark?
Op Without WM
Read Out Translated Op Code (Bit Significant From Auxiliary Storage) Set Op Into G Register S4=1 for B,•, L, N, or Op S5=1 for M, L, or U Op
M, L, or H
Store UV Regs in UV Back-up Locations Set S0=1
MLQH? (ROR1)
Q
Store LT Regs in UV Back-up Locations Place LT Regs in UV Regs Set S7=0 and S0=1
Invalid A Addr? S6
Yes
S6=1
Place A Hundreds Back-up in B Hundreds Back-up Set S7=1
No
S6=0
All Other Ops
Read Out Hundreds Digit
MLU I/O Op
Yes
I/O A Address Routine
Set S5=0
Word Mark
Yes
Op Alone
I/E Change Routine
Place Hundreds Digit in V Reg (Low)
Invalid Digit
Yes
Set S5=1
Set S6=0
B
Hundred Zones
Yes
Add Zones Equivalent Constants in D Reg and U Reg (Low)
Read Out Tens Digit-Place Digit in U Reg (High) Set S6=0
Word Mark
Yes
2 or 5 Character Instruction
Address Invalid Routine
Invalid Digit
Yes
If Tens Digit Is Bl, &, or - Set S5=1
Tens Zones
Yes
Indexing Routine
Hundreds Look Up (Read Out Two Low Hex Equivalent Digits + Bias) Add These Digits to D Reg
Branch On S5
S5=1
Address Invalid Routine
MLQH
No
Set S7=0
Hundreds Look Up (Read Out High Hex Equivalent Digit + Bias) Put Tens Digit in V (Low) Add "Hundreds" Digits to U-Low Reset U-High
A

Figure 5-9. I-Cycles (Part 1 of 3)

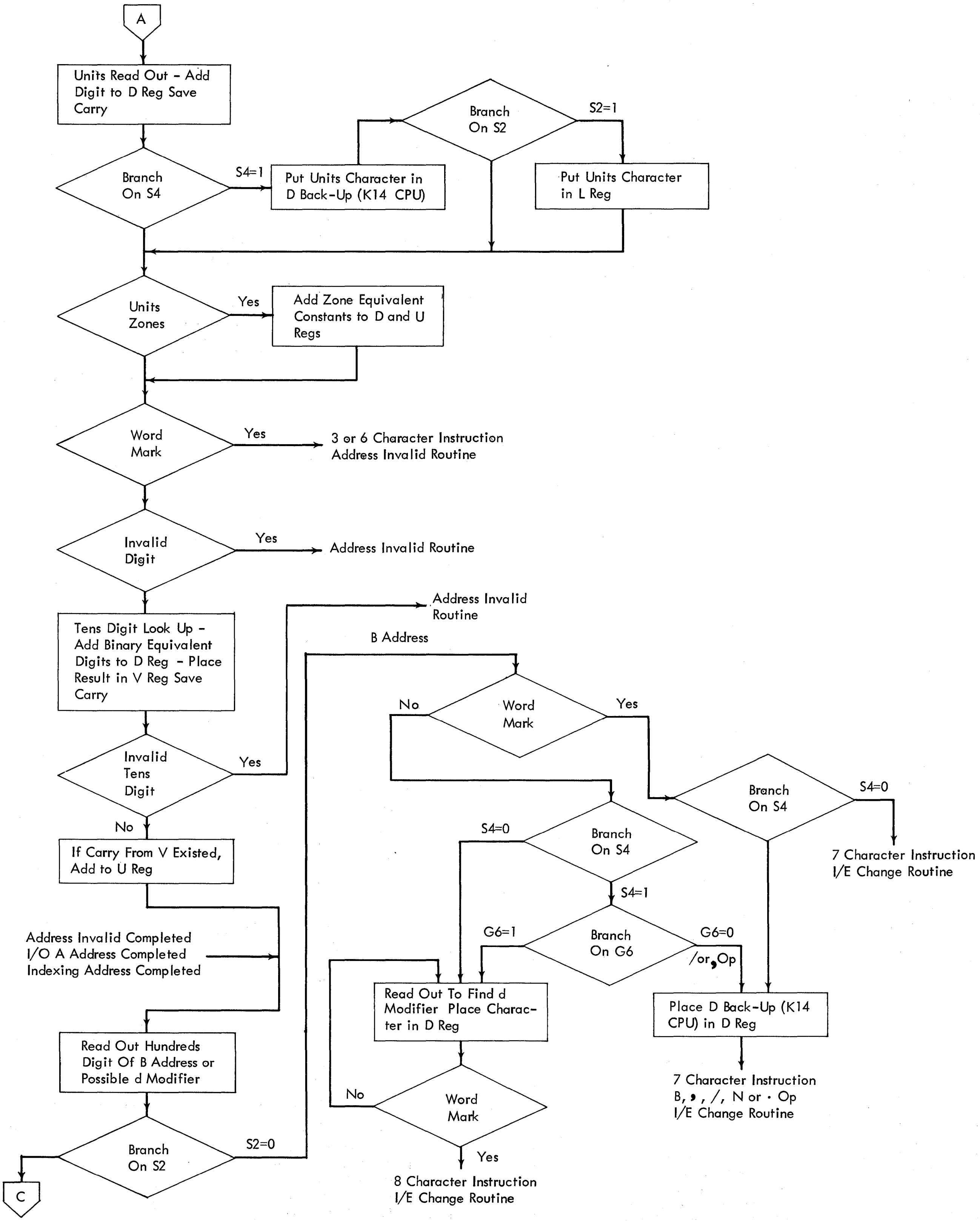

A
Units Read Out - Add Digit to D Reg Save Carry
Branch On S4
S4=1
Put Units Character in D Back-Up (K14 CPU)
Branch On S2
S2=1
Put Units Character in L Reg
Units Zones
Yes
Add Zone Equivalent Constants to D and U Regs
Word Mark
Yes
3 or 6 Character Instruction Address Invalid Routine
Invalid Digit
Yes
Address Invalid Routine
Tens Digit Look Up - Add Binary Equivalent Digits to D Reg - Place Result in V Reg Save Carry
Address Invalid Routine
B Address
Invalid Tens Digit
Yes
No
If Carry From V Existed, Add to U Reg
Address Invalid Completed
I/O A Address Completed
Indexing Address Completed
Read Out Hundreds Digit Of B Address or Possible d Modifier
Branch On S2
S2=0
C
No
Word Mark
Yes
Branch On S4
S4=0
7 Character Instruction I/E Change Routine
S4=0
Branch On S4
S4=1
G6=1
Branch On G6
G6=0
/or,Op
Read Out To Find d Modifier Place Character in D Reg
Place D Back-Up (K14 CPU) in D Reg
7 Character Instruction B, , , /, N or · Op I/E Change Routine
No
Word Mark
Yes
8 Character Instruction I/E Change Routine

Figure 5-9. I-Cycles (Part 2 of 3)

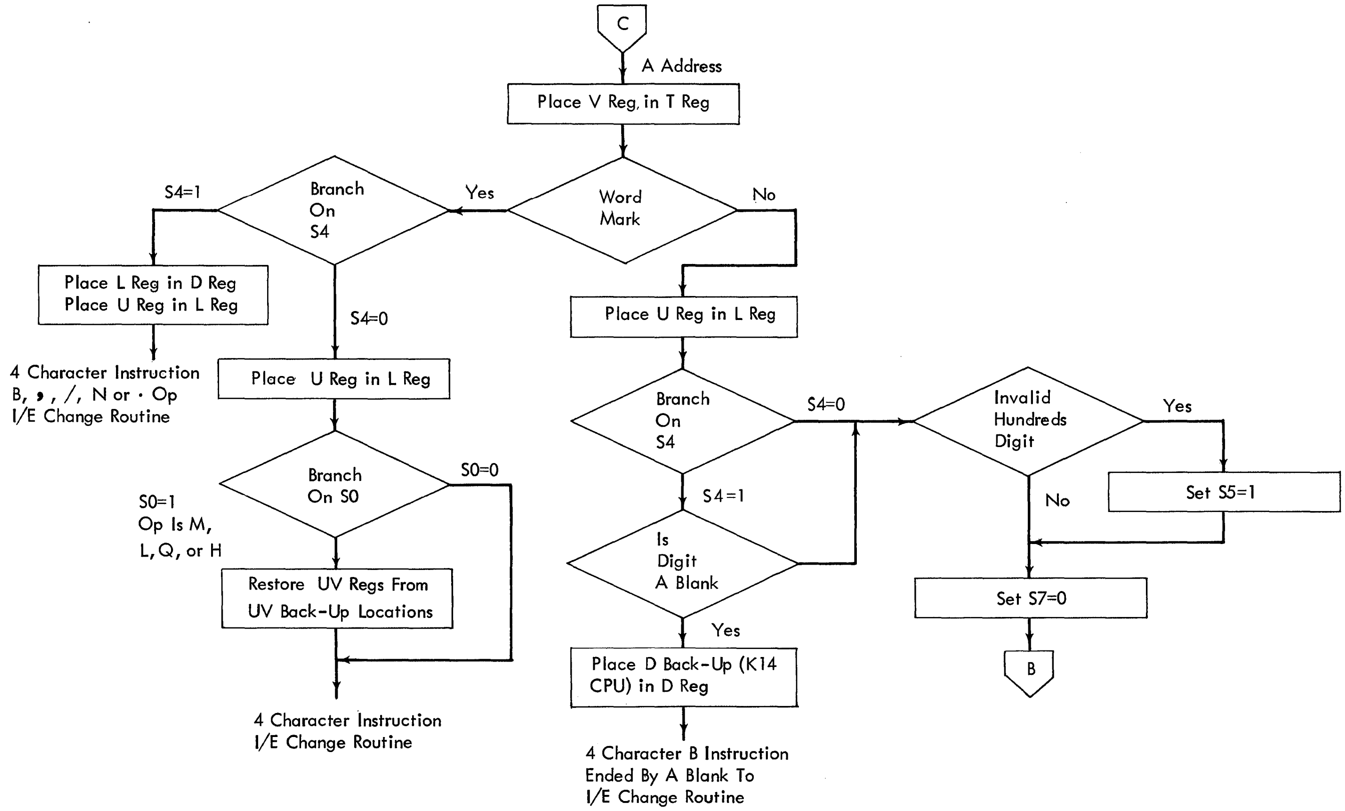


Figure 5-9. I-Cycles (Part 3 of 3)

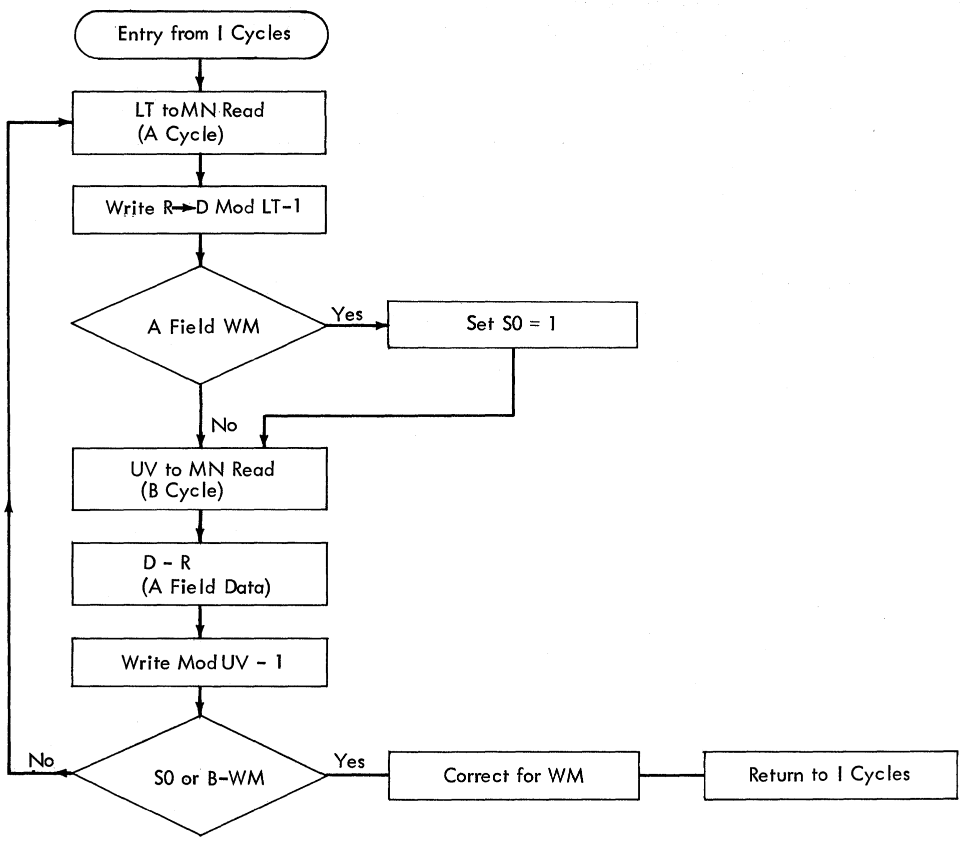


Figure 5-10. Move Op

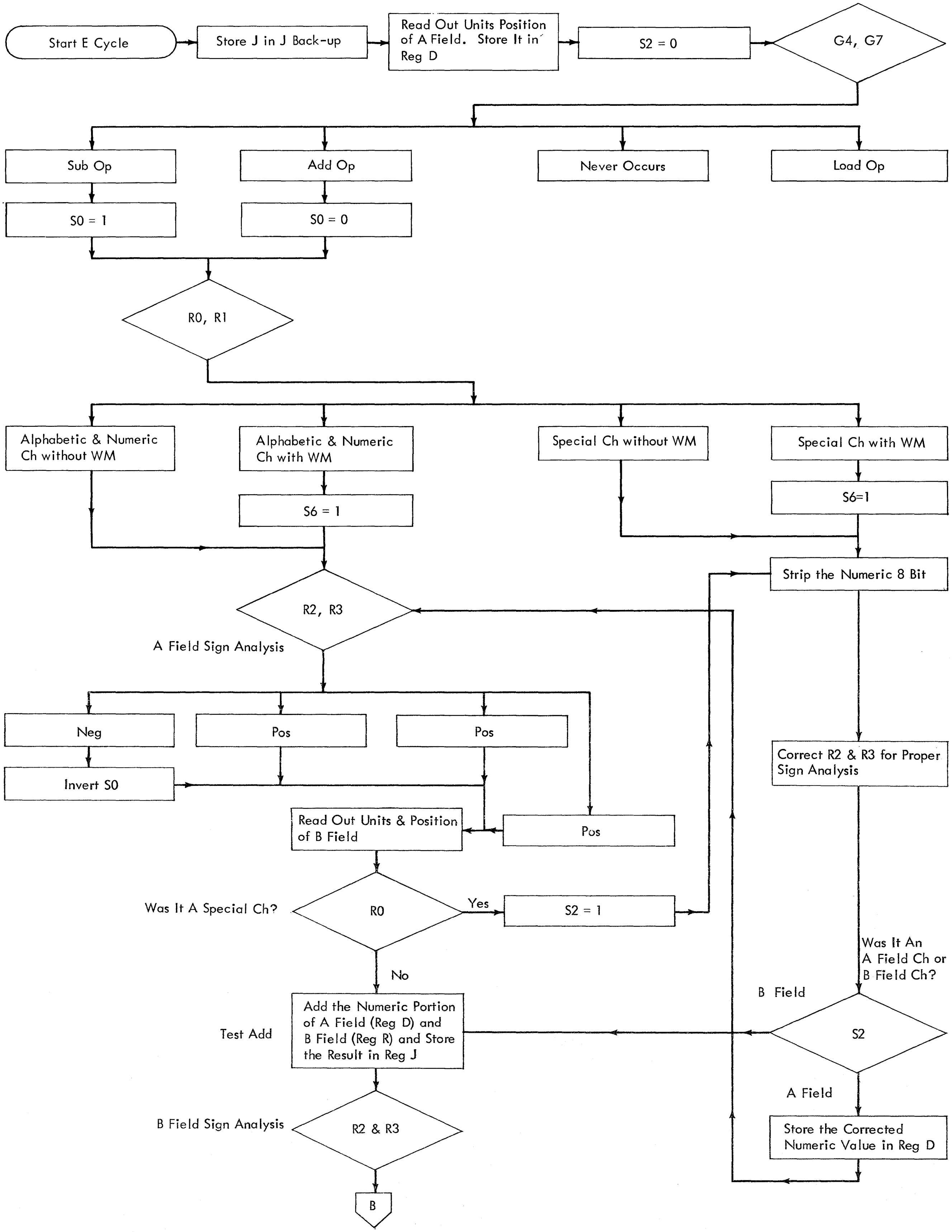

Start E Cycle
Store J in J Back-up
Read Out Units Position of A Field. Store It in Reg D
S2 = 0
G4, G7
Sub Op
Add Op
Never Occurs
Load Op
S0 = 1
S0 = 0
R0, R1
Alphabetic & Numeric Ch without WM
Alphabetic & Numeric Ch with WM
Special Ch without WM
Special Ch with WM
S6 = 1
S6=1
Strip the Numeric 8 Bit
R2, R3
A Field Sign Analysis
Neg
Pos
Pos
Invert S0
Correct R2 & R3 for Proper Sign Analysis
Read Out Units & Position of B Field
Pos
Was It A Special Ch?
R0
Yes
S2 = 1
No
Was It An A Field Ch or B Field Ch?
B Field
Test Add
Add the Numeric Portion of A Field (Reg D) and B Field (Reg R) and Store the Result in Reg J
S2
A Field
B Field Sign Analysis
R2 & R3
Store the Corrected Numeric Value in Reg D
B

Figure 5-11. Add Cp (Part 1 of 5)

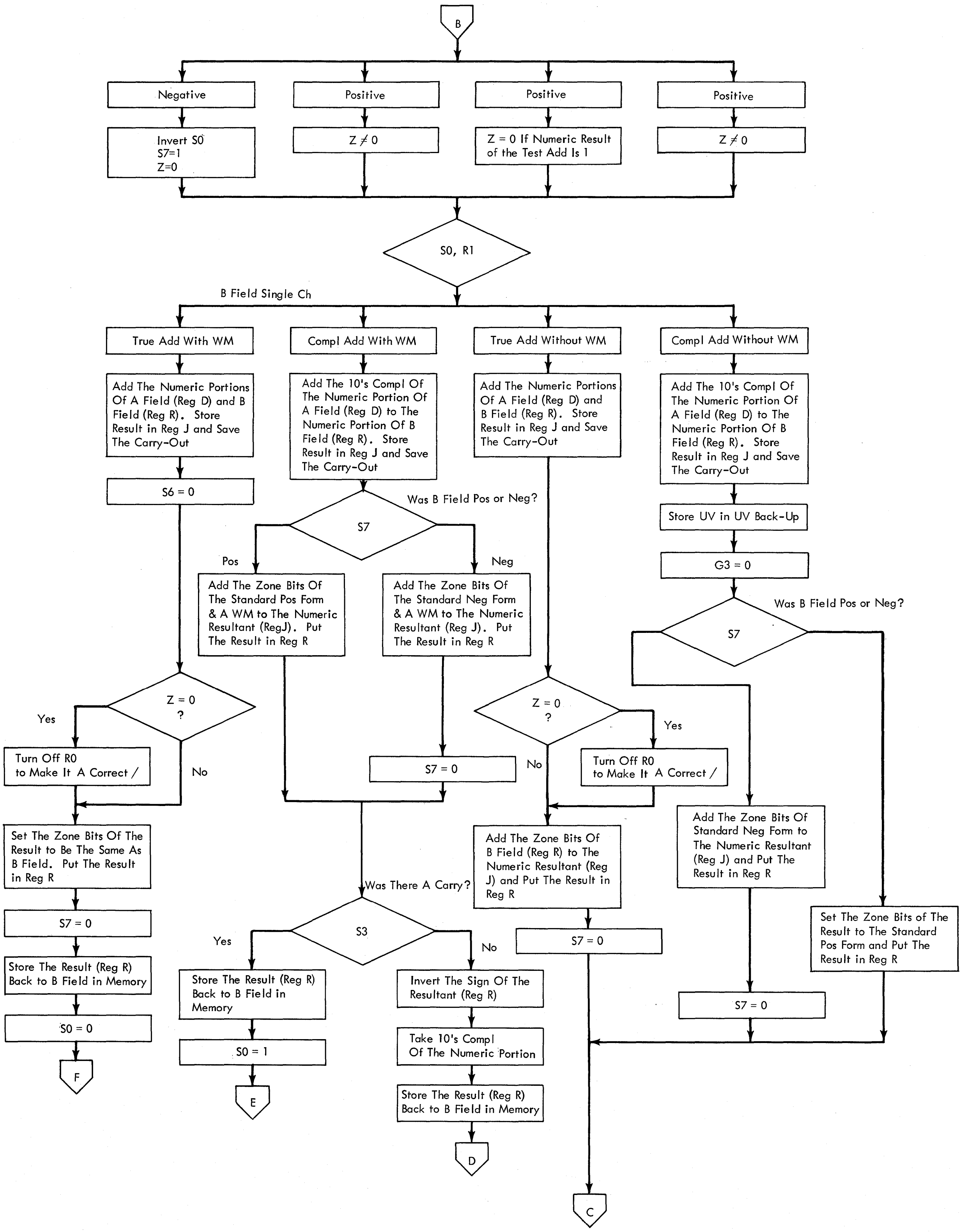
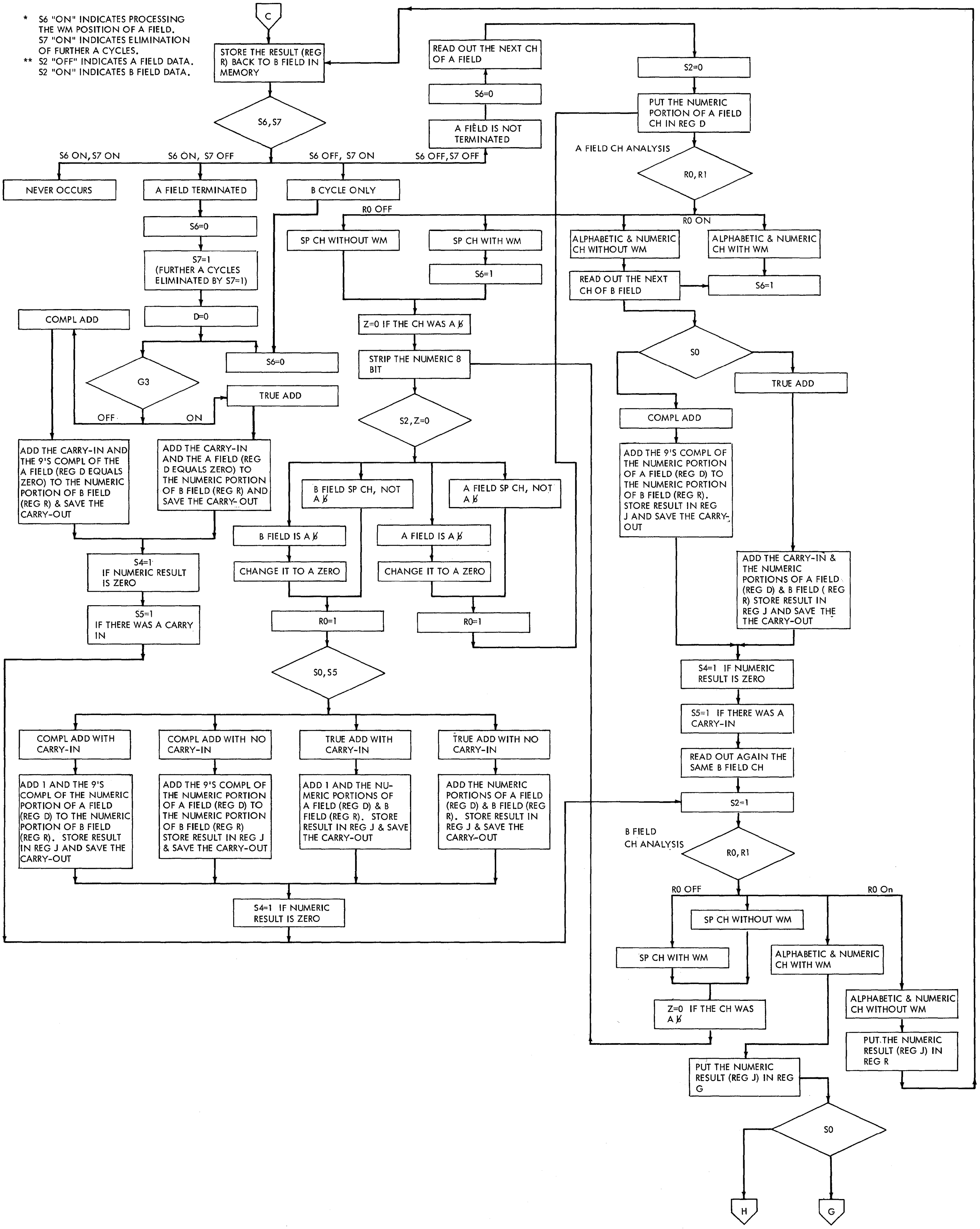

B
Negative
Positive
Positive
Positive
Invert S0
S7=1
Z=0
Z ≠ 0
Z = 0 If Numeric Result of the Test Add Is 1
Z ≠ 0
S0, R1
B Field Single Ch
True Add With WM
Compl Add With WM
True Add Without WM
Compl Add Without WM
Add The Numeric Portions Of A Field (Reg D) and B Field (Reg R). Store Result in Reg J and Save The Carry-Out
Add The 10's Compl Of The Numeric Portion Of A Field (Reg D) to The Numeric Portion Of B Field (Reg R). Store Result in Reg J and Save The Carry-Out
Add The Numeric Portions Of A Field (Reg D) and B Field (Reg R). Store Result in Reg J and Save The Carry-Out
Add The 10's Compl Of The Numeric Portion Of A Field (Reg D) to The Numeric Portion Of B Field (Reg R). Store Result in Reg J and Save The Carry-Out
S6 = 0
Was B Field Pos or Neg?
S7
Pos
Neg
Store UV in UV Back-Up
G3 = 0
Add The Zone Bits Of The Standard Pos Form & A WM to The Numeric Resultant (RegJ). Put The Result in Reg R
Add The Zone Bits Of The Standard Neg Form & A WM to The Numeric Resultant (Reg J). Put The Result in Reg R
Was B Field Pos or Neg?
S7
Z = 0 ?
Yes
No
Turn Off R0 to Make It A Correct /
S7 = 0
Z = 0 ?
No
Yes
Turn Off R0 to Make It A Correct /
Set The Zone Bits Of The Result to Be The Same As B Field. Put The Result in Reg R
Add The Zone Bits Of B Field (Reg R) to The Numeric Resultant (Reg J) and Put The Result in Reg R
Add The Zone Bits Of Standard Neg Form to The Numeric Resultant (Reg J) and Put The Result in Reg R
Was There A Carry?
S3
Yes
No
S7 = 0
S7 = 0
Set The Zone Bits of The Result to The Standard Pos Form and Put The Result in Reg R
Store The Result (Reg R) Back to B Field in Memory
Store The Result (Reg R) Back to B Field in Memory
Invert The Sign Of The Resultant (Reg R)
S7 = 0
S0 = 0
S0 = 1
Take 10's Compl Of The Numeric Portion
F
E
Store The Result (Reg R) Back to B Field in Memory
D
C

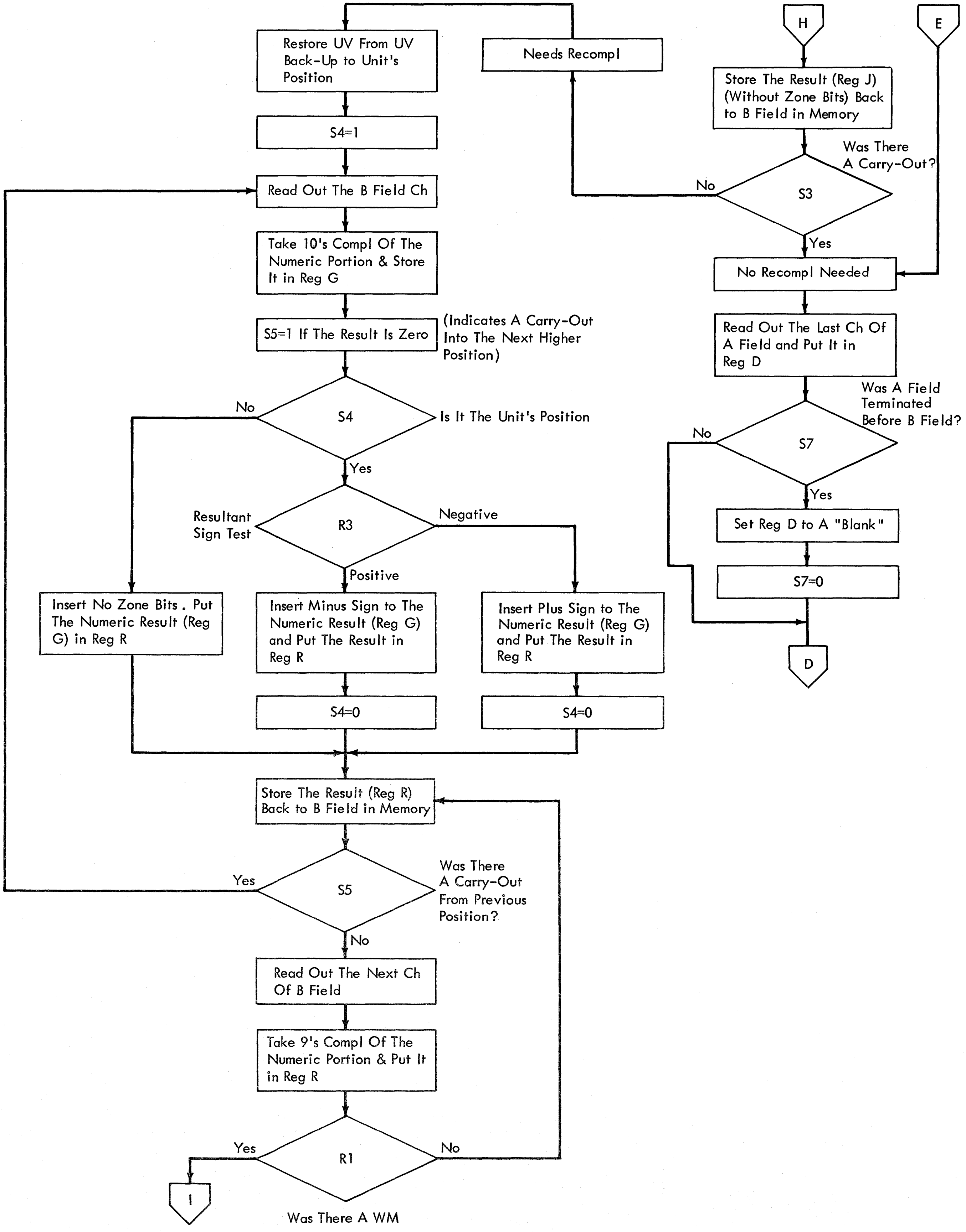
Figure 5-11. Add Op (Part 2 of 5)



* S6 "ON" INDICATES PROCESSING THE WM POSITION OF A FIELD.
S7 "ON" INDICATES ELIMINATION OF FURTHER A CYCLES.
** S2 "OFF" INDICATES A FIELD DATA.
S2 "ON" INDICATES B FIELD DATA.

C
STORE THE RESULT (REG R) BACK TO B FIELD IN MEMORY
S6, S7
S6 ON, S7 ON
S6 ON, S7 OFF
S6 OFF, S7 ON
S6 OFF, S7 OFF
NEVER OCCURS
A FIELD TERMINATED
B CYCLE ONLY
S6=0
S7=1 (FURTHER A CYCLES ELIMINATED BY S7=1)
D=0
COMPL ADD
G3
OFF
ON
S6=0
TRUE ADD
ADD THE CARRY-IN AND THE 9'S COMPL OF THE A FIELD (REG D EQUALS ZERO) TO THE NUMERIC PORTION OF B FIELD (REG R) & SAVE THE CARRY-OUT
ADD THE CARRY-IN AND THE A FIELD (REG D EQUALS ZERO) TO THE NUMERIC PORTION OF B FIELD (REG R) AND SAVE THE CARRY-OUT
S4=1 IF NUMERIC RESULT IS ZERO
S5=1 IF THERE WAS A CARRY IN
READ OUT THE NEXT CH OF A FIELD
S6=0
A FIELD IS NOT TERMINATED
R0 OFF
SP CH WITHOUT WM
SP CH WITH WM
S6=1
Z=0 IF THE CH WAS A ƀ
STRIP THE NUMERIC 8 BIT
S2, Z=0
B FIELD SP CH, NOT A ƀ
A FIELD SP CH, NOT A ƀ
B FIELD IS A ƀ
A FIELD IS A ƀ
CHANGE IT TO A ZERO
CHANGE IT TO A ZERO
R0=1
R0=1
S0, S5
COMPL ADD WITH CARRY-IN
COMPL ADD WITH NO CARRY-IN
TRUE ADD WITH CARRY-IN
TRUE ADD WITH NO CARRY-IN
ADD 1 AND THE 9'S COMPL OF THE NUMERIC PORTION OF A FIELD (REG D) TO THE NUMERIC PORTION OF B FIELD (REG R). STORE RESULT IN REG J AND SAVE THE CARRY-OUT
ADD THE 9'S COMPL OF THE NUMERIC PORTION OF A FIELD (REG D) TO THE NUMERIC PORTION OF B FIELD (REG R) STORE RESULT IN REG J & SAVE THE CARRY-OUT
ADD 1 AND THE NUMERIC PORTIONS OF A FIELD (REG D) & B FIELD (REG R). STORE RESULT IN REG J & SAVE THE CARRY-OUT
ADD THE NUMERIC PORTIONS OF A FIELD (REG D) & B FIELD (REG R). STORE RESULT IN REG J & SAVE THE CARRY-OUT
S4=1 IF NUMERIC RESULT IS ZERO
S2=0
PUT THE NUMERIC PORTION OF A FIELD CH IN REG D
A FIELD CH ANALYSIS
R0, R1
R0 ON
ALPHABETIC & NUMERIC CH WITHOUT WM
ALPHABETIC & NUMERIC CH WITH WM
READ OUT THE NEXT CH OF B FIELD
S6=1
S0
COMPL ADD
TRUE ADD
ADD THE 9'S COMPL OF THE NUMERIC PORTION OF A FIELD (REG D) TO THE NUMERIC PORTION OF B FIELD (REG R). STORE RESULT IN REG J AND SAVE THE CARRY-OUT
ADD THE CARRY-IN & THE NUMERIC PORTIONS OF A FIELD (REG D) & B FIELD (REG R) STORE RESULT IN REG J AND SAVE THE THE CARRY-OUT
S4=1 IF NUMERIC RESULT IS ZERO
S5=1 IF THERE WAS A CARRY-IN
READ OUT AGAIN THE SAME B FIELD CH
S2=1
B FIELD CH ANALYSIS
R0, R1
R0 OFF
R0 On
SP CH WITHOUT WM
SP CH WITH WM
ALPHABETIC & NUMERIC CH WITH WM
ALPHABETIC & NUMERIC CH WITHOUT WM
Z=0 IF THE CH WAS A ƀ
PUT THE NUMERIC RESULT (REG J) IN REG R
PUT THE NUMERIC RESULT (REG J) IN REG G
S0
H
G

Figure 5-11. Add Op (Part 3 of 5)



Restore UV From UV Back-Up to Unit's Position
Needs Recompl
H
E
Store The Result (Reg J) (Without Zone Bits) Back to B Field in Memory
Was There A Carry-Out?
S3
No
Yes
S4=1
Read Out The B Field Ch
Take 10's Compl Of The Numeric Portion & Store It in Reg G
No Recompl Needed
S5=1 If The Result Is Zero
(Indicates A Carry-Out Into The Next Higher Position)
Read Out The Last Ch Of A Field and Put It in Reg D
Was A Field Terminated Before B Field?
S7
No
Yes
S4
No
Yes
Is It The Unit's Position
Set Reg D to A "Blank"
S7=0
D
Resultant Sign Test
R3
Negative
Positive
Insert No Zone Bits. Put The Numeric Result (Reg G) in Reg R
Insert Minus Sign to The Numeric Result (Reg G) and Put The Result in Reg R
Insert Plus Sign to The Numeric Result (Reg G) and Put The Result in Reg R
S4=0
S4=0
Store The Result (Reg R) Back to B Field in Memory
S5
Yes
No
Was There A Carry-Out From Previous Position?
Read Out The Next Ch Of B Field
Take 9's Compl Of The Numeric Portion & Put It in Reg R
R1
Yes
No
I
Was There A WM

Figure 5-11. Add Cp (Part 4 of 5)

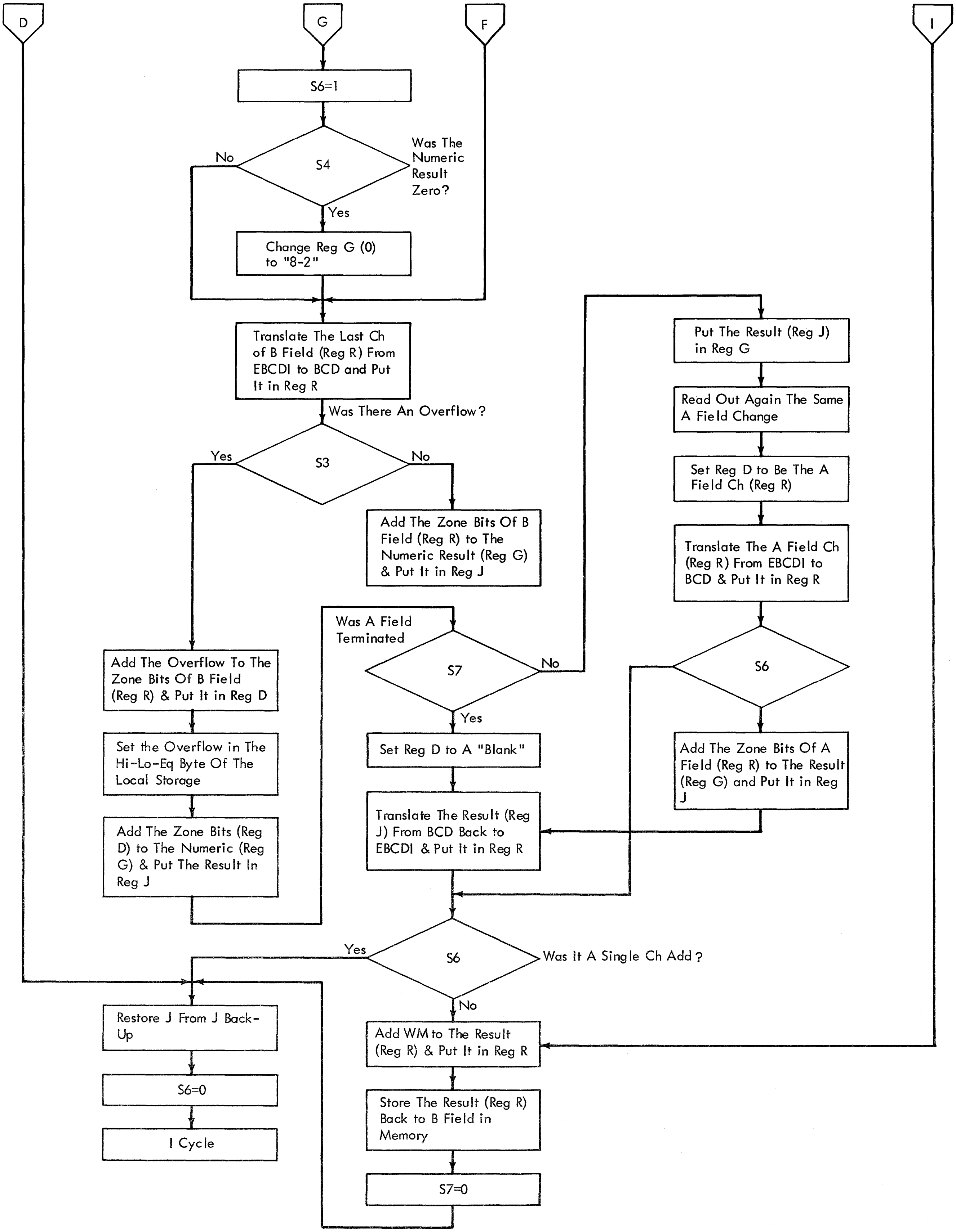

D
G
F
I
S6=1
S4
Was The Numeric Result Zero?
No
Yes
Change Reg G (0) to "8-2"
Translate The Last Ch of B Field (Reg R) From EBCDI to BCD and Put It in Reg R
Was There An Overflow?
S3
Yes
No
Add The Zone Bits Of B Field (Reg R) to The Numeric Result (Reg G) & Put It in Reg J
Add The Overflow To The Zone Bits Of B Field (Reg R) & Put It in Reg D
Set the Overflow in The Hi-Lo-Eq Byte Of The Local Storage
Add The Zone Bits (Reg D) to The Numeric (Reg G) & Put The Result In Reg J
Was A Field Terminated
S7
No
Yes
Set Reg D to A "Blank"
Translate The Result (Reg J) From BCD Back to EBCDI & Put It in Reg R
Put The Result (Reg J) in Reg G
Read Out Again The Same A Field Change
Set Reg D to Be The A Field Ch (Reg R)
Translate The A Field Ch (Reg R) From EBCDI to BCD & Put It in Reg R
S6
Add The Zone Bits Of A Field (Reg R) to The Result (Reg G) and Put It in Reg J
S6
Was It A Single Ch Add?
Yes
No
Restore J From J Back-Up
S6=0
I Cycle
Add WM to The Result (Reg R) & Put It in Reg R
Store The Result (Reg R) Back to B Field in Memory
S7=0

Figure 5-11. Add Op (Part 5 of 5)

BRANCHING

- In compatibility mode, a branch can be executed from bits R1, R2, or R3.

- A branch on GMWM can be executed.

Certain branch conditions normally available in the 2030 are replaced by different branch conditions when the 2030 is operating in compatibility mode. These changes are shown in the following table:

Compatibility Mode	2030 Mode	CH	CL
R1	1 BC		0110
R2	S1	1000	
R3	G2		1100
GMWM	V67=0	0011	

The R1, R2, and R3 branches are made from the contents of the R-register. These branches can be given in the cycle immediately following a read cycle. A GMWM latch has been added that detects a GMWM on the storage sense bus when a read cycle occurs. The GMWM latch feeds the stat use decoder and remains latched on until the next read call is given to core storage.

I/C OPERATICNS

- All I/C operations in compatibility mode are executed in burst mode.

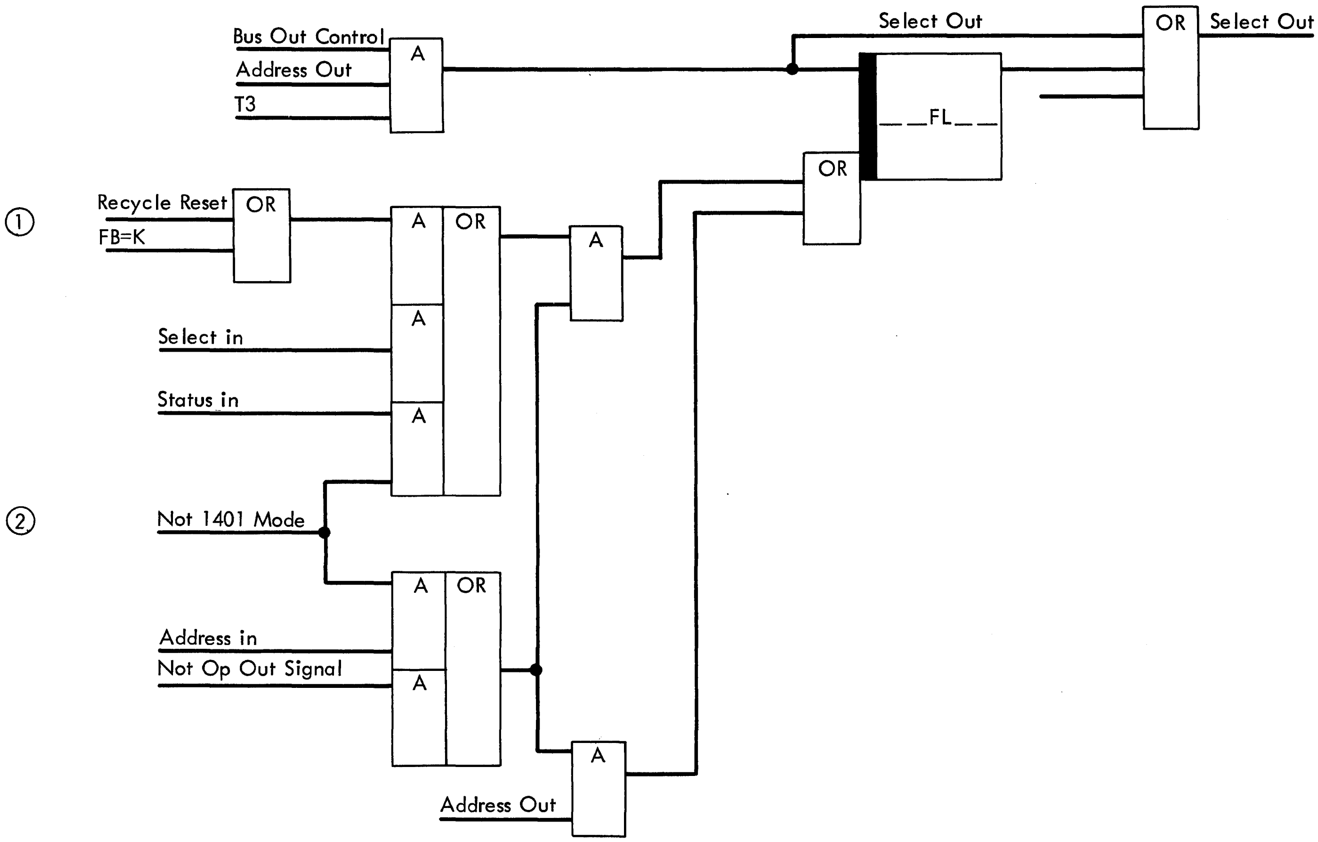
- The 1402 reader automatically feeds a card 6 milliseconds after a read command.

- 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/C apparatus is in EBCDI code.

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 5-12). 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 micro-program statement K->FB. 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 micro program statement K->FB, which is given when a Channel End is sensed by the 2030.



1.1401 Reset Of Burst Mode
2. Blocks Normal Reset to Select Out

Figure 5-12. Select Out

1402 Read Operation

With a 1402 operating with a 1401 or a 1460 system, the programmer has 10 milliseconds after a read-a-card instruction to give a stacker select instruction. The card being read then feeds past the read brushes and into the stacker selected.

When a 1402 is operating on System/360, Model 30, a read command causes the buffer to transmit data to the CPU, but no card movement takes place.

To make the 1402 on a 2030 act like a 1402 on a 1401, a circuit is added to cause a provisional feed 6 milliseconds after a read command is given. Basically this circuit accomplishes this: 6 milliseconds after the data transfer of a read command starts, an automatic feed cycle occurs and the card just read selects 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. This feed command causes the card to feed immediately and stacker select and also prevents the provisional feed from occurring (Figure 5-13).

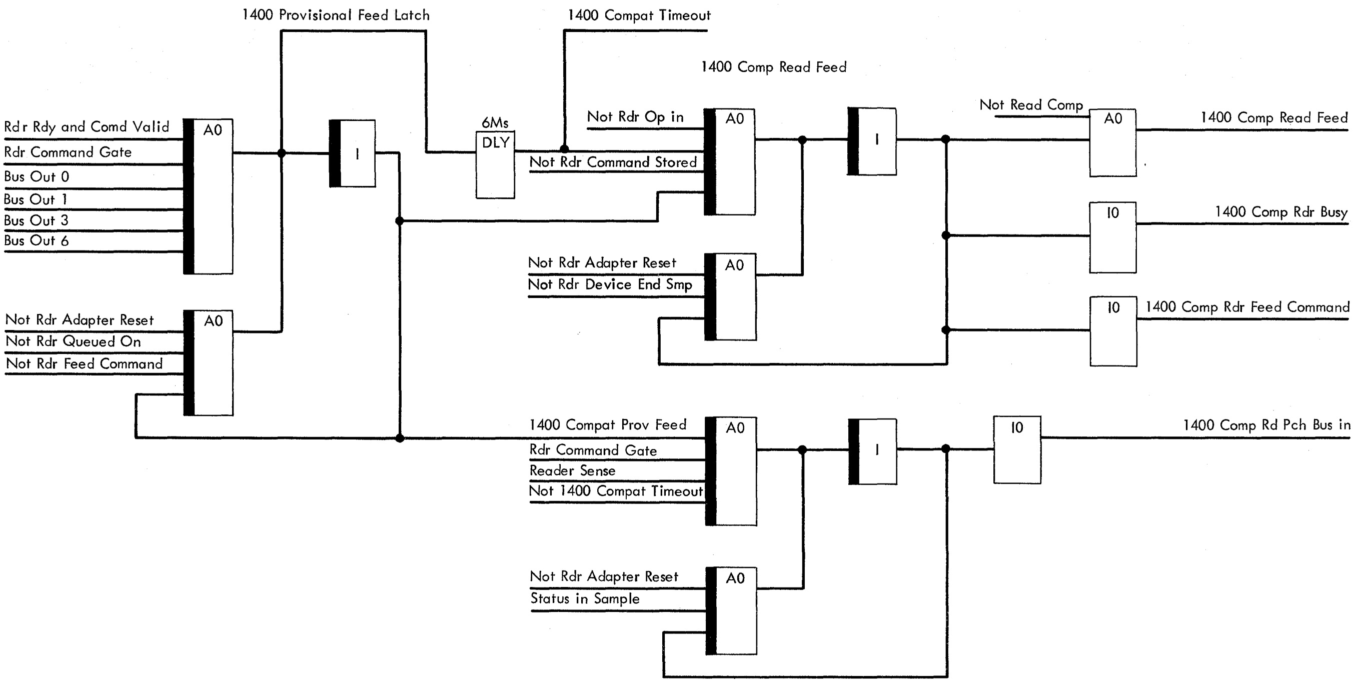


Figure 5-13. Card Read

The micro program for stacker select can detect whether or not the 6-millisecond 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 5-13). In this case, the micro program does not issue the stacker select command, but will indicate an invalid stacker select to the operator by a coded byte in the R-register.

Another modification of the 1402 circuitry on the 2030 changes the end-of-file condition. When the 1402 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 file condition to occur. In 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.

1402 Punch Operation

A stacker select command given by a 2030 to the 1402 punch causes the card about to be punched to be selected. A stacker select command given to the 1402 punch on the 1401 or 1460 systems causes the card at the punch check station to be selected. To make the stacker select command on the 2030 system apply to the card at the punch check station while operating in 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.

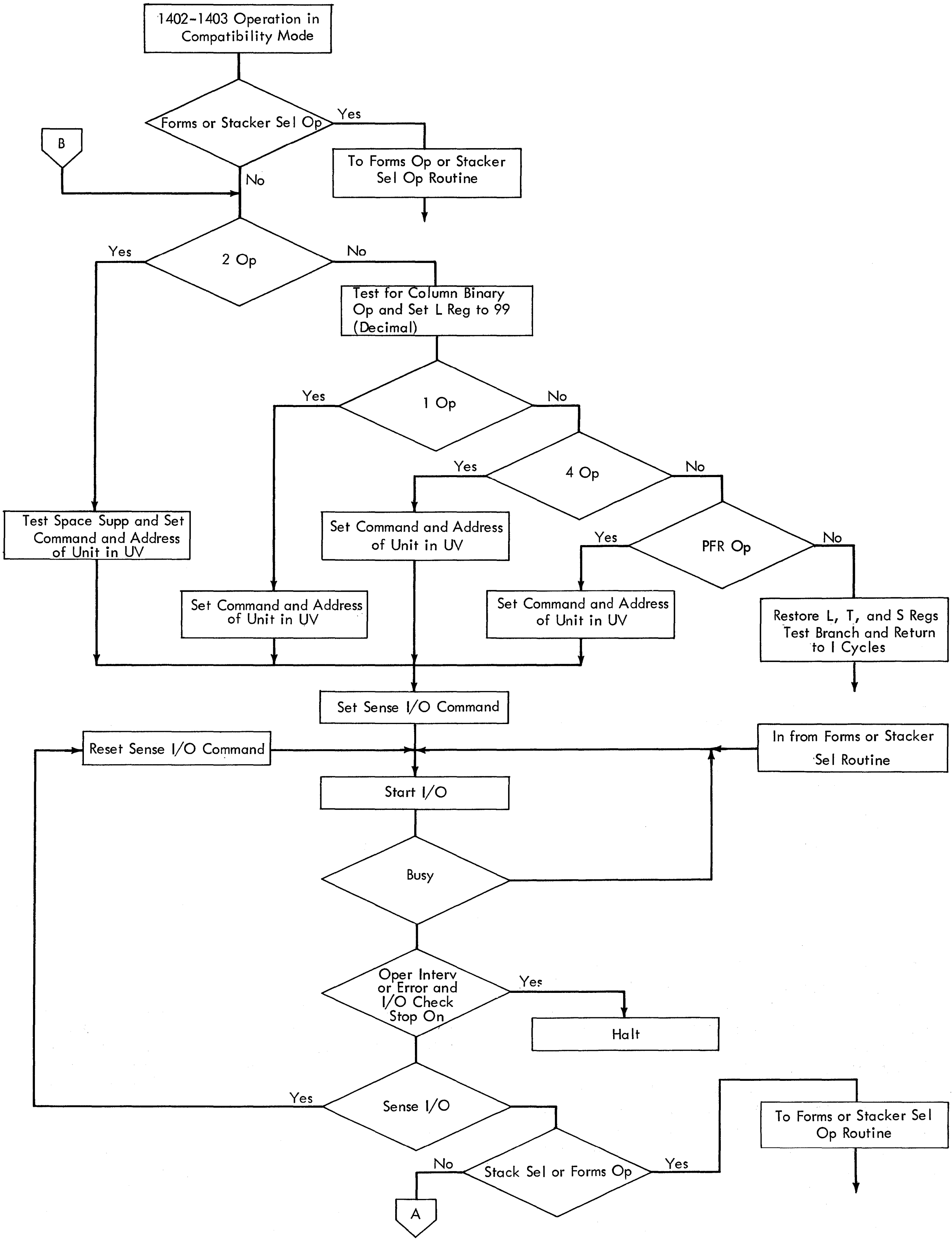

1402-1403 Operation in Compatibility Mode
Forms or Stacker Sel Op
Yes
To Forms Op or Stacker Sel Op Routine
B
No
2 Op
Yes
No
Test for Column Binary Op and Set L Reg to 99 (Decimal)
1 Op
Yes
No
4 Op
Yes
No
PFR Op
Yes
No
Test Space Supp and Set Command and Address of Unit in UV
Set Command and Address of Unit in UV
Set Command and Address of Unit in UV
Set Command and Address of Unit in UV
Restore L, T, and S Regs Test Branch and Return to I Cycles
Set Sense I/O Command
Reset Sense I/O Command
In from Forms or Stacker Sel Routine
Start I/O
Busy
Oper Interv or Error and I/O Check Stop On
Yes
Halt
Yes
Sense I/O
To Forms or Stacker Sel Op Routine
No
Stack Sel or Forms Op
Yes
A

Figure 5-14. 1402-1403 Compatibility (Part 1 of 2)

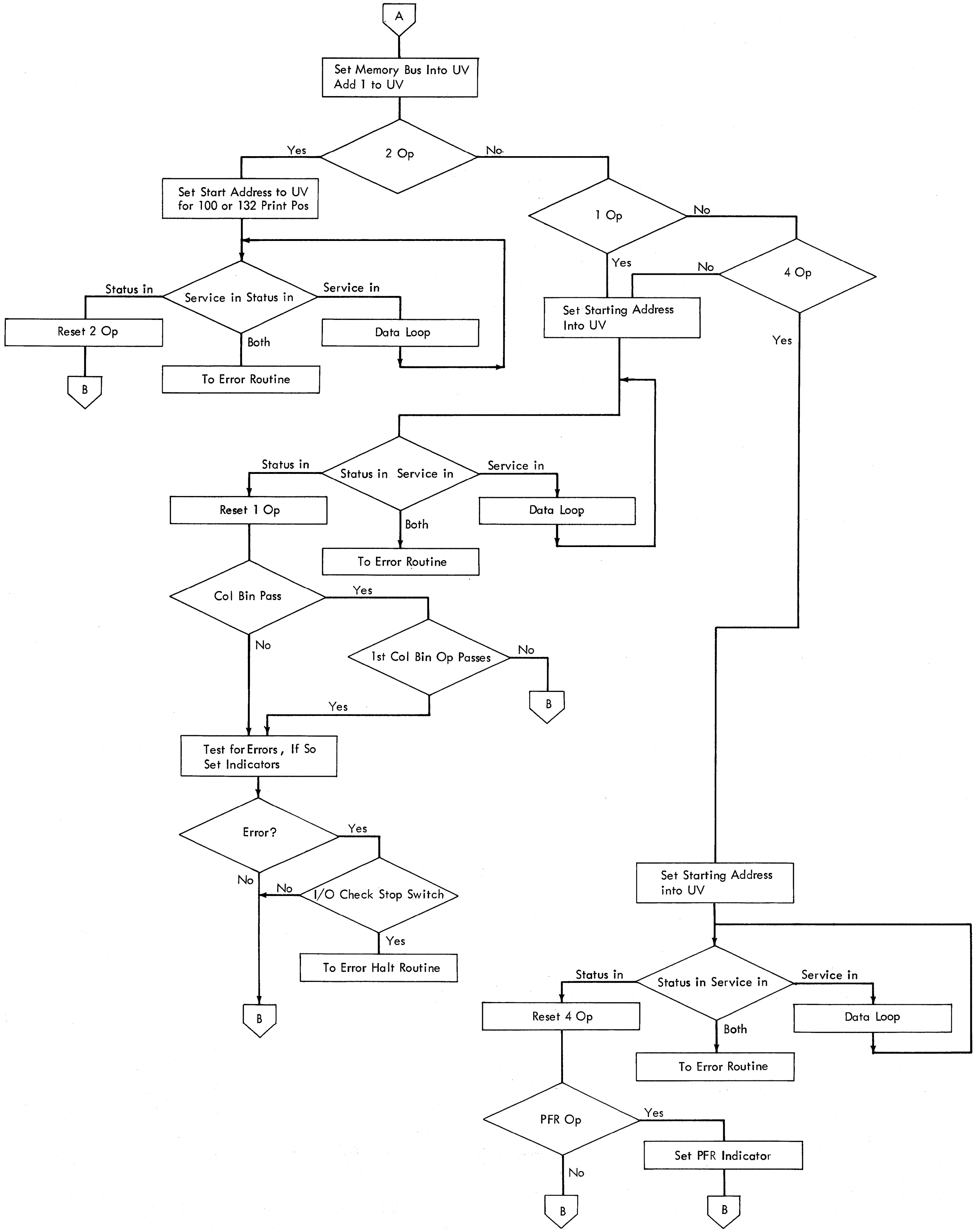


Figure 5-14. 1402-1403 Compatibility (Part 2 of 2)

See Figure 5-14 for reader/punch operational flow.

The I/O operations required by the 1400 object program are performed by their respective micro programs. The 1400 I/O commands must be able to sense status, perform the operation, and detect any errors that occur during that operation. To further illustrate this point, let us examine the 1402 operational flow chart. Notice the micro program will first define what operation is to be performed. Next, it fetches the unit address from local storage and issues a sense command. The micro program then examines the status byte coming back to make sure the 1402 can accept the command. If the 1402 status is good, the command is given and the micro program goes into a data loop. While in the data loop, the micro program is continually looking for a Channel End to occur. The operation is being done in burst mode because select out cannot be reset. When Channel End occurs, the micro program gives the statement FB->K, which resets select out and allows the 1402 to drop-off the channel. The micro program will examine the status byte that came with Channel End to make sure the operation was performed error free. If not, the micro program will go to an error routine to display a coded byte in the R register to indicate a 1402 error. If the operation did occur error free, the micro program goes back to interrogate the Op code again. This is done to determine whether or not it is a combined Op (read, punch, etc.). If it was not a combined Op, the micro program will exit to I-cycles for the next Op code. Notice the micro program does not use Device End at all. The micro program will accept Device End to get it off the line, but does not use it because Channel End indicates the end of the operation.

1442 Reader/Punch Operation

When running the 1442 in compatibility mode, there are two 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 micro program 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 micro program so that the 8-9 punches that designated the characters as MLP characters are eliminated. The characters in core storage are the EBCDI equivalent of the card code minus the 8-9 punches. For 1442 operational flow, see Figure 5-16.

CHARACTER IN STORAGE	CHARACTER TO 1443	CHARACTER IN STORAGE	CHARACTER TO1443
&	&	Q	Y
-	-	9	8
1	0	Ƶ	I
/	A ①	I	R
A	J	R	Ƶ
J	/ ②	0	9
2	1	ǂ	>
S	B	?	<
B	K	!	ǂ
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	)	-v-
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	b̸	Blank
H	Q		

① Sent As 01000001
② Sent As 11100001

Figure 5-15. 52 and 63 Character Typebar Decode

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. Since the characters on the bar are not the same as the 1443 N1 bar, the micro program must construct different characters configurations to send to the printer (Figure 5-15). If the character in storage to be printed is an A, the micro program sends a J to the 1443 N1. The 1443 N1 circuits fire the hammer when it thinks there is a J in front of the hammer. Actually, since 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. For operational flow of 1443 in compatibility mode, see Figure 5-16.

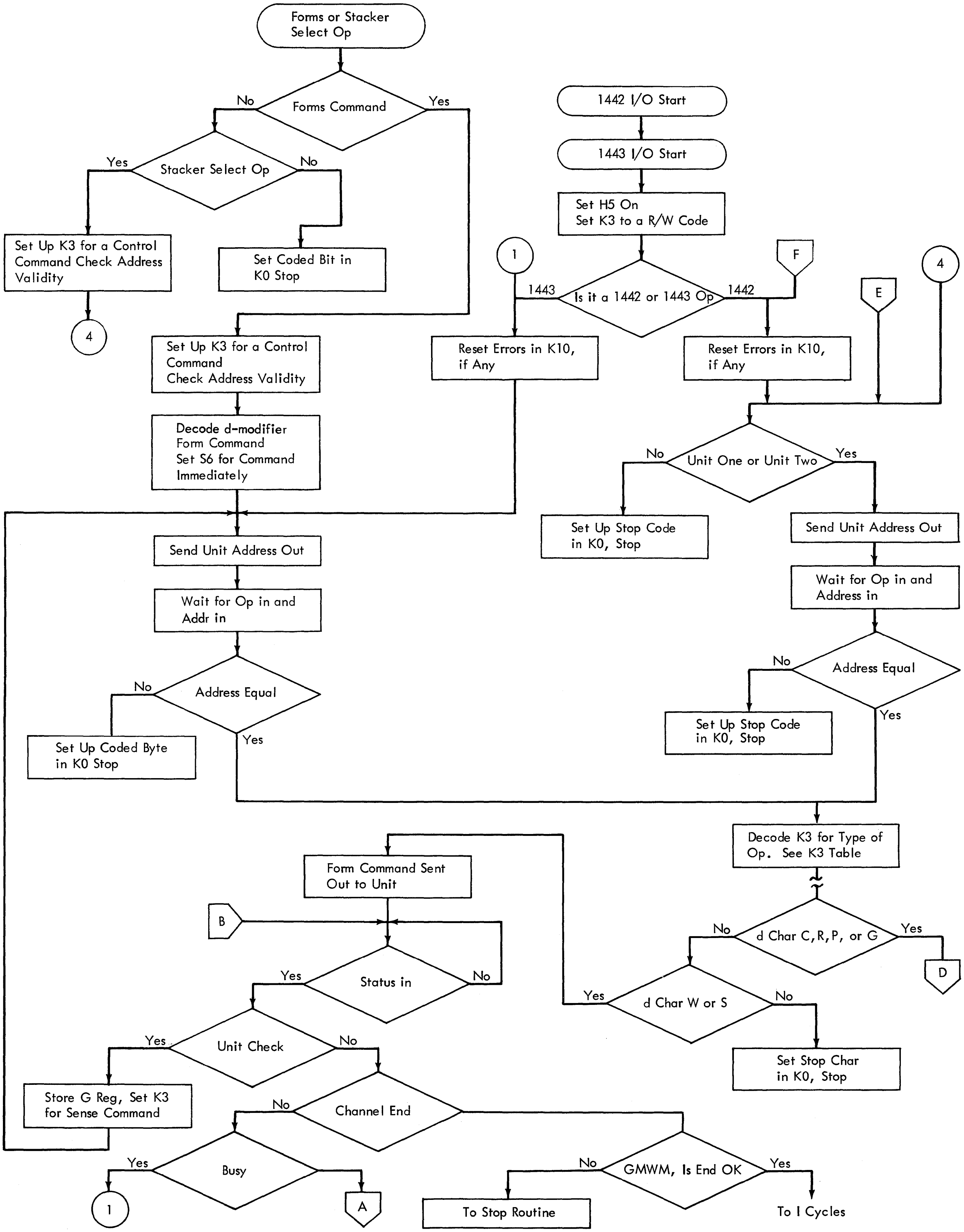

Forms or Stacker Select Op
Forms Command
No
Yes
Stacker Select Op
Yes
No
Set Up K3 for a Control Command Check Address Validity
Set Coded Bit in K0 Stop
4
Set Up K3 for a Control Command Check Address Validity
Decode d-modifier Form Command Set S6 for Command Immediately
1442 I/O Start
1443 I/O Start
Set H5 On Set K3 to a R/W Code
1
F
4
E
1443
Is it a 1442 or 1443 Op
1442
Reset Errors in K10, if Any
Reset Errors in K10, if Any
Unit One or Unit Two
No
Yes
Set Up Stop Code in K0, Stop
Send Unit Address Out
Send Unit Address Out
Wait for Op in and Addr in
Wait for Op in and Address in
Address Equal
No
Yes
Address Equal
No
Yes
Set Up Coded Byte in K0 Stop
Set Up Stop Code in K0, Stop
Decode K3 for Type of Op. See K3 Table
Form Command Sent Out to Unit
B
d Char C,R,P, or G
No
Yes
D
Status in
Yes
No
d Char W or S
Yes
No
Set Stop Char in K0, Stop
Unit Check
Yes
No
Store G Reg, Set K3 for Sense Command
Channel End
No
Busy
Yes
1
A
GMWM, Is End OK
No
Yes
To Stop Routine
To I Cycles

Figure 5-16. 1442/1443 Operational Flow (Part 1 of 3)

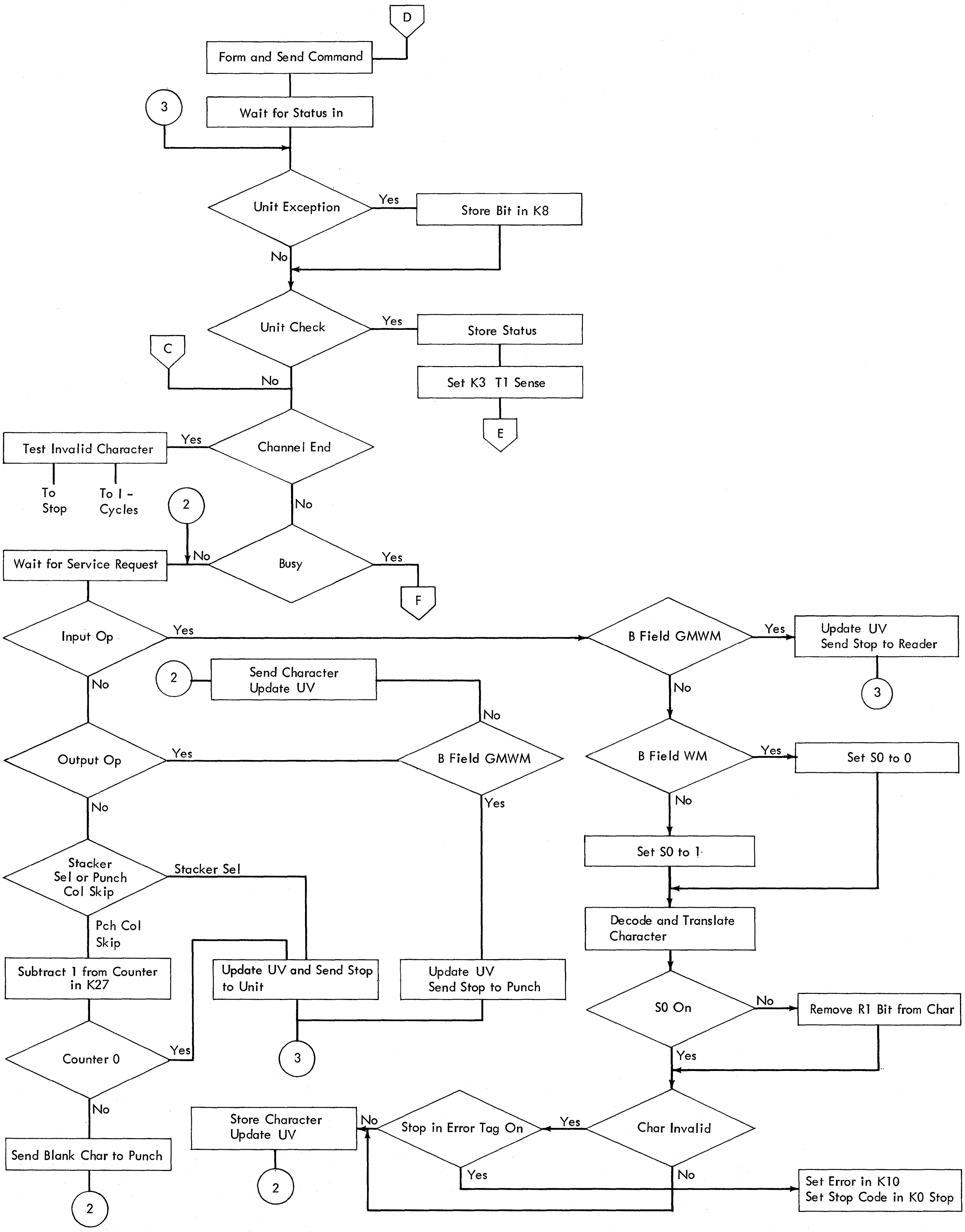
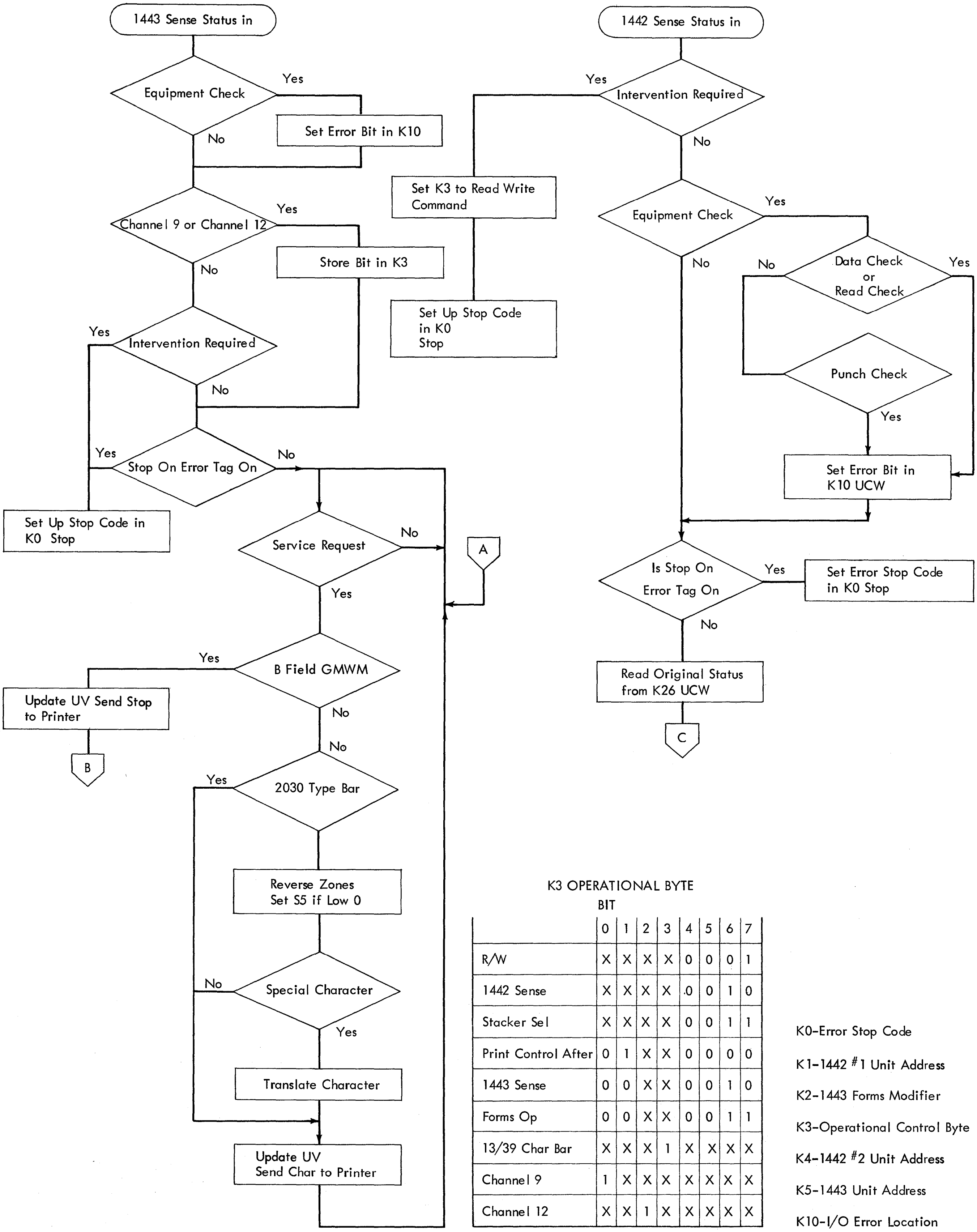


Figure 5-16. 1442/1443 Operational Flow (Part 2 of 3)



K3 OPERATIONAL BYTE

BIT

	0	1	2	3	4	5	6	7
R/W	X	X	X	X	0	0	0	1
1442 Sense	X	X	X	X	0	0	1	0
Stacker Sel	X	X	X	X	0	0	1	1
Print Control After	0	1	X	X	0	0	0	0
1443 Sense	0	0	X	X	0	0	1	0
Forms Op	0	0	X	X	0	0	1	1
13/39 Char Bar	X	X	X	1	X	X	X	X
Channel 9	1	X	X	X	X	X	X	X
Channel 12	X	X	1	X	X	X	X	X

K0-Error Stop Code

K1-1442 #1 Unit Address

K2-1443 Forms Modifier

K3-Operational Control Byte

K4-1442 #2 Unit Address

K5-1443 Unit Address

K10-I/O Error Location

Figure 5-16. 1442/1443 Operational Flow (Part 3 of 3)

Magnetic Tape Control

The following conditions should be noted in order to understand tape operations in compatibility mode on the 2030. In a 1401 or a 1460, an erase tape instruction causes a long gap to be erased when the next write command is issued. In compatibility mode on the 2030, an erase tape instruction causes a long gap to be erased immediately. 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 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 reset by any backward command. Because the bit in local storage is not reset by manually unloading the tape unit, the operator must insure that the bit is reset when reloading the tape unit to eliminate false end of file conditions.

A tape error during initial program load causes a micro-program stop with a coded byte in K0 of MPX1 storage indicating the error. The tape control word in local storage is a double word stored in local storage locations 80 through 87 (HEX) with the following format:

	0		1	
8X	TCU Control	TCU Address	TU1 Control	TU1 Address
	2		3	
	TU2 Control	TU2 Address	TU3 Control	TU3 Address
	4		5	
	TU4 Control	TU4 Address	TU5 Control	TU5 Address
	6		7	
	TU6 Control	TU6 Address	Last Status from TCU	

The TCU 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. The TCU tape control unit as provided by the initial program load deck. In the tape units 1-6 control, 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

When on, bit 2 of the TU control indicates the last operation performed on this unit was a backspace operation. When on, bit 3 of the TU control indicates an end-of-file condition outstanding on this unit. Tape unit 1-6 address contains the address assignment of the tape unit to be used as a 1401 or 1460 unit. The tape unit control bits 0 and 1 and the tape unit address are provided by the initial program load routine.

The following miscellaneous storage locations in the MPX1 storage are used for tape operations.

K6, K7 O-Star locations used during a read operation as back up for the starting address of the Read In area (B-Star)

K11 Track in Error sense byte which is stored if a read error occurs on a 9 track tape.

K25 Temporary storage of tape unit control byte address for unit being used.

K26 Storage Location used in Read Mask of setting H5 bit. The actual address of the unit being used is also stored here.

K27 Temporary storage of command byte.

K28 Temporary storage of the read status byte for 9 track operation.

For tape operation on a multiplex channel, see Figure 5-17.

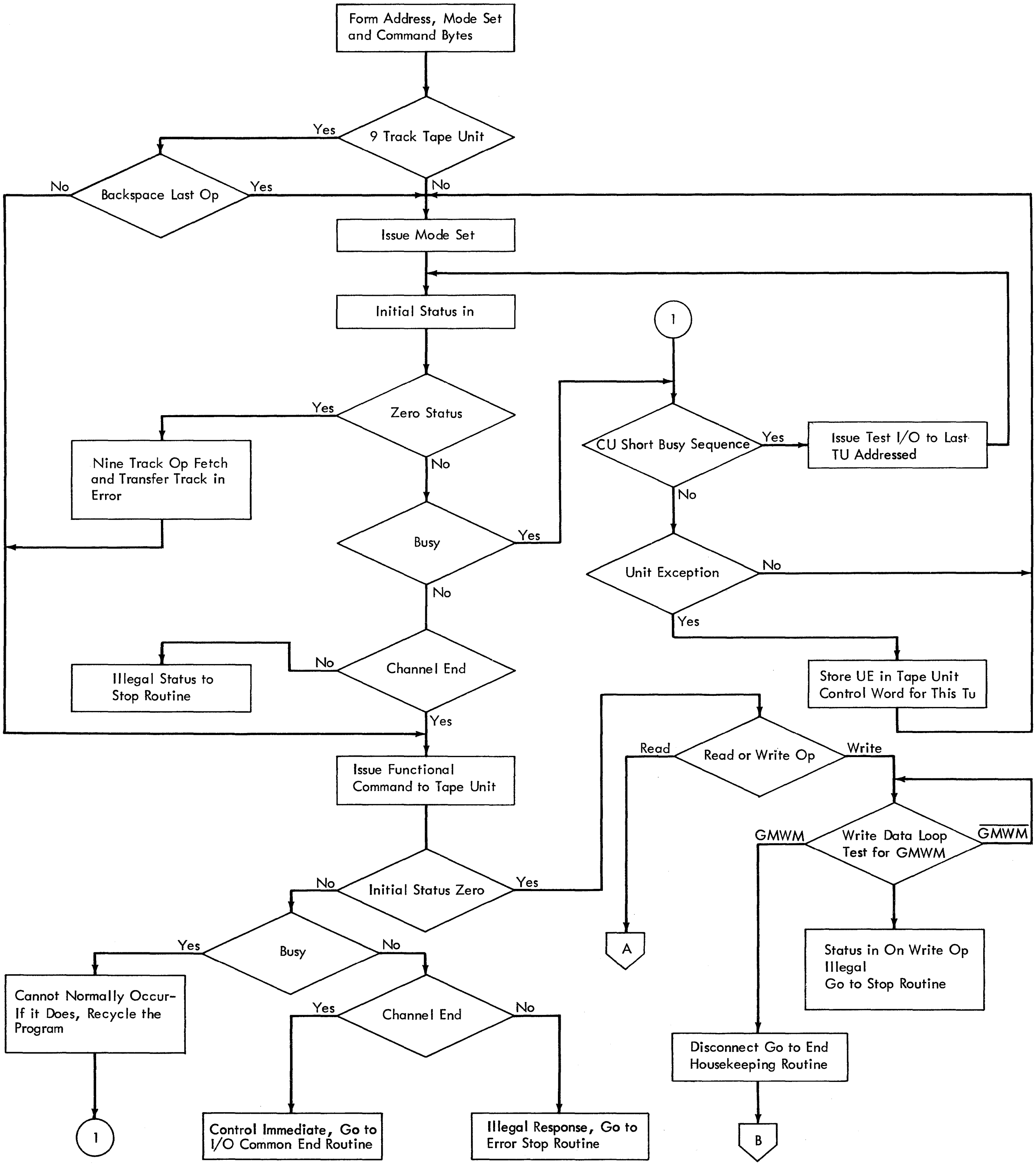

Form Address, Mode Set and Command Bytes
9 Track Tape Unit
Yes
No
Backspace Last Op
No
Yes
Issue Mode Set
Initial Status in
1
Zero Status
Yes
No
Nine Track Op Fetch and Transfer Track in Error
CU Short Busy Sequence
Yes
Issue Test I/O to Last TU Addressed
No
Busy
Yes
No
Unit Exception
No
Yes
Channel End
No
Yes
Illegal Status to Stop Routine
Store UE in Tape Unit Control Word for This Tu
Read or Write Op
Read
Write
Issue Functional Command to Tape Unit
Write Data Loop Test for GMWM
GMWM
$\overline{\text{GMWM}}$
Initial Status Zero
No
Yes
A
Busy
Yes
No
Status in On Write Op Illegal Go to Stop Routine
Cannot Normally Occur- If it Does, Recycle the Program
Channel End
Yes
No
Disconnect Go to End Housekeeping Routine
1
Control Immediate, Go to I/O Common End Routine
Illegal Response, Go to Error Stop Routine
B

Figure 5-17. Tape Compatibility (Part 1 of 2)

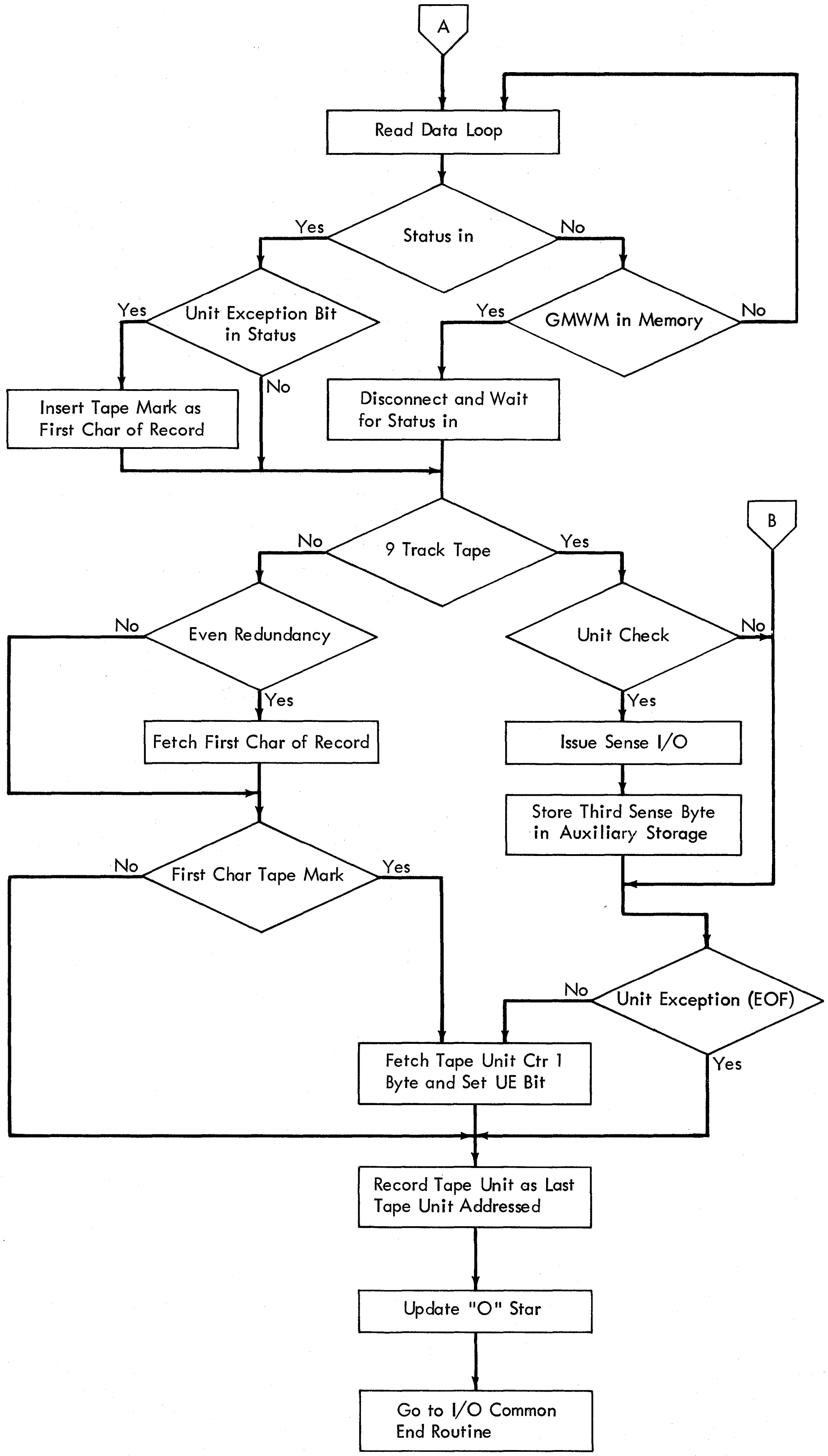

A
Read Data Loop
Status in
Yes
No
Unit Exception Bit in Status
Yes
No
GMWM in Memory
Yes
No
Insert Tape Mark as First Char of Record
Disconnect and Wait for Status in
9 Track Tape
No
Yes
B
Even Redundancy
No
Yes
Unit Check
No
Yes
Fetch First Char of Record
Issue Sense I/O
Store Third Sense Byte in Auxiliary Storage
First Char Tape Mark
No
Yes
Unit Exception (EOF)
No
Yes
Fetch Tape Unit Ctr 1 Byte and Set UE Bit
Record Tape Unit as Last Tape Unit Addressed
Update "O" Star
Go to I/O Common End Routine

Figure 5-17. Tape Compatibility (Part 2 of 2)

Disk Storage Control

All disk packs run on the 2030, including those processed in compatibility mode, must have their addresses in System/360 format. The System/360 disk address is a 5-byte binary address, where the bytes represent cylinder, cylinder, head, head, and record in that order. In compatibility mode, the data record is in EBCDI code.

Let us assume the 2030 in compatibility mode encounters the disk operation M%F1bbbR, where bbb represents the high-order position of the disk control field. To execute this instruction the micro program converts the 6-position 1401 address that is in the disk control field to the 5-position binary address that is compatible with the disk addresses. The converted address is stored in MPX1 storage in locations AD through B1. This address will be the address presented to the disk storage unit. Disk storage unit addresses are found through a table lookup method and extracted from MPX1 storage where they were stored by the initial program load. Associated with each disk unit address in a unit cylinder byte. The unit cylinder byte indicates to the micro program the cylinder number where the access arm is located for seek commands. This byte is one position displaced from the disk unit address, such as disk 0 unit address is in MPX1 storage location 9 0 and 0 cylinder is in MPX1 storage location 9 1.

As previously mentioned, the disk address in AD through B1 is the address presented to the disk storage drive. In a sector operation, as each sector is successfully completed, this address is updated to the address of the next sector. It is important to note here that the address found in the disk control field represented by the bbb address of the disk Op code will not be increased as each successful sector is read or written. Instead, the disk control field in the bbb address is updated at the successful completion of the disk operation.

To provide an address register to gate MN while executing disk operations, IJ is stored at the end if I phase into IJ backup locations 88 and 89 of Local Storage, and restored at the end of the disk operation. The four sense bytes from the disk storage drive are stored in MPX1 storage locations B9 through BC to indicate to the micro program if the operation can be continued. MPX1 storage location AA contains the previous disk operation performed. When a disk scan operation is performed, the result is stored in 99 local storage for testing. If needed, the address of the alternate track is stored in local-storage locations A0 through AB. See Figure 5-18 for the storage locations for disk operation.

			0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
LOCAL STORAGE	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-LO	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	56	12	68	24	80	36	92	48	04	60	16	72	28	84	40
	BCD TO EBCDI	4X	40	F1	F2	F3	F4	F5	F6	F7	F8	F9	F0	7B	7C	7D	7E	7F
		5X	7A	61	E2	E3	E4	E5	E6	E7	E8	E9	E0	6B	6C	6D	6E	6F
		6X	60	D1	D2	D3	D4	D5	D6	D7	D8	D9	D0	5B	5C	5D	5E	5F
		7X	50	C1	C2	C3	C4	C5	C6	C7	C8	C9	C0	4B	4C	4D	4E	4F
		8X	TAPE								0	1	2	3	4	5	6	7
		9X									8	9	10	11	12	13	14	15
		AX	WORKING STORAGE								16	17	18	19	20	21	22	23
		BX	00	96	92	88					24	25	26	27	28	29	30	31
	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
		DX	I)1D	34	K)29	L)90	M)80	N)06	34	P)1E	Q)F1	34	34	34	34	34	34	34
		EX	34	/)05	S)19	34	U)20	V)3A	W)3B	34	Y)13	Z)17	34	,)04	%)1B	34	34	34
		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
MPX 1 STORAGE		0X	00	05	01	06	02	07	03	08	04	09						
		1X																
	HUNDS-HI	2X	Z	Z+00	Z+00	Z+01	Z+01	Z+01	Z+02	Z+02	Z+03	Z+03						
		3X																
	EBCDI TO BCD	4X	00	40	40	6B	6C	6D	6E	6F	80	60	40	3B	3C	3D	3E	3F
		5X	30	40	40	5B	5C	5D	5E	5F	40	40	40	2B	2C	2D	2E	2F
		6X	20	11	40	4B	4C	4D	4E	4F	90	50	40	1B	1C	1D	1E	1F
		7X	40	40	7A	7B	7C	7D	7E	7F	40	40	10	0B	0C	0D	0E	0F
		8X									0	1	2	3	4	5	6	7
		9X	FILE UNIT 0 ADDER	UNIT 0 CYL	FILE UNIT 1 ADDR	UNIT 1 CYL	FILE UNIT 2 ADDR	UNIT 2 CYL	FILE UNIT 3 ADDR	UNIT 3 CYL	8	9	10	11	12	13	14	15
		AX	CYL # 00	CYL # 32	CYL # 0A	CYL # 3C	CYL # 14	CYL # 46	CYL # 1E	CYL # 50	16	17	18	19	20	21	22	23
		BX	H	R	00	DL	DL				24	25	26	27	28	29	30	31
	EBCDI TO BCD	CX	3A	31	32	33	34	35	36	37	38	39	E0	40	40	40	45	5C
		DX	2A	21	22	23	24	25	26	27	28	29	D0	40	40	40	46	5D
		EX	1A	40	12	13	14	15	16	17	18	19	C0	40	40	40	4F	56
		FX	0A	01	02	03	04	05	06	07	08	09	F0	40	40	40	44	5F

K=ADDRESSABLE BYTE UTILIZATION	
LOCAL STORAGE	MPX 1 STORAGE
0 I BACK-UP	0 ERROR CODE
1 J " "	1
2 U " "	2 SEE SHEET BX
3 V " "	3 USED BY 1402-03
4 L " "	4 AND 1442-43
5 T " "	5
6 G " "	6 O STAR HI
7 S " "	7 O STAR LO
8 SENSE SW BYTE	8 FILE UNIT 4 ADDR
9 HI-LO-EQ BYTE	9 UNIT 4 CYLINDER
10 MEM SIZE BYTE	10 I/O ERROR
11 FILE BRANCH BYTE	11 TAPE TRK IN ERROR
12 1401 CONTROL	12
13 PMS CONTROL	13 A HUND'S BK-UP
14 D BACKUP	14 B HUND'S BK-UP
15 ALLOW I/O TRAPS	15 1050 STATUS
16 WORKING STORAGE	16 CYL # 28
17 " "	17 CYL # 5A
18 " "	18 PREVIOUS OPR
19 " "	19
20 CONSTANT-1F	20 00
21	21 00
22	22 C
23	23 H
24	24 WORKING STORAGE
25	25 " "
26 CONSTANT-0F	26 " "
27	27 " "
28	28 " "
29	29 " "
30 CONSTANT-2E	30 " "
31	31 " "

NOTES 1. X INDICATES THAT THE QUANTITY IN PARENTHESIS IS CROSSED IN THE TABLE
2. Z=MEMORY BIAS-HI
Y=MEMORY BIAS-LO
3. ENTRIES IN HUNDS-HI TABLE INCLUDE CARRY FROM HUNDS-LO TABLE

1401 CONTROL BYTE

BIT 0-I/O CHECK STOP
1-ADV PROG FEATURE
2-EXPANDED EDIT
3-MODE SW ON INVALID OPS
4-COLUMN BINARY
5-MODE SW ON HALT
6-TAPE ON S X 2
7-MODE SW ON ERROR STOPS

Figure 5-18. Auxiliary Storage Map for Disk Operation

To assist in troubleshooting disk operations, B9 through BF of the MPX 1 storage contain information that is of value to the C E in his trouble analysis:

B9 - Sense 0 byte	
BA - Sense 1 byte	Same bit significance as 2030 errors.
BB - Sense 2 byte	
BC - Sense 3 byte	
BD - File Address	Current or last file address worked with.
BE - File Command	Last file command issued.
BF - Scan Condition	Is file scanning Hi/Lo/Eq?

See Figure 5-19 for disk operational flow.

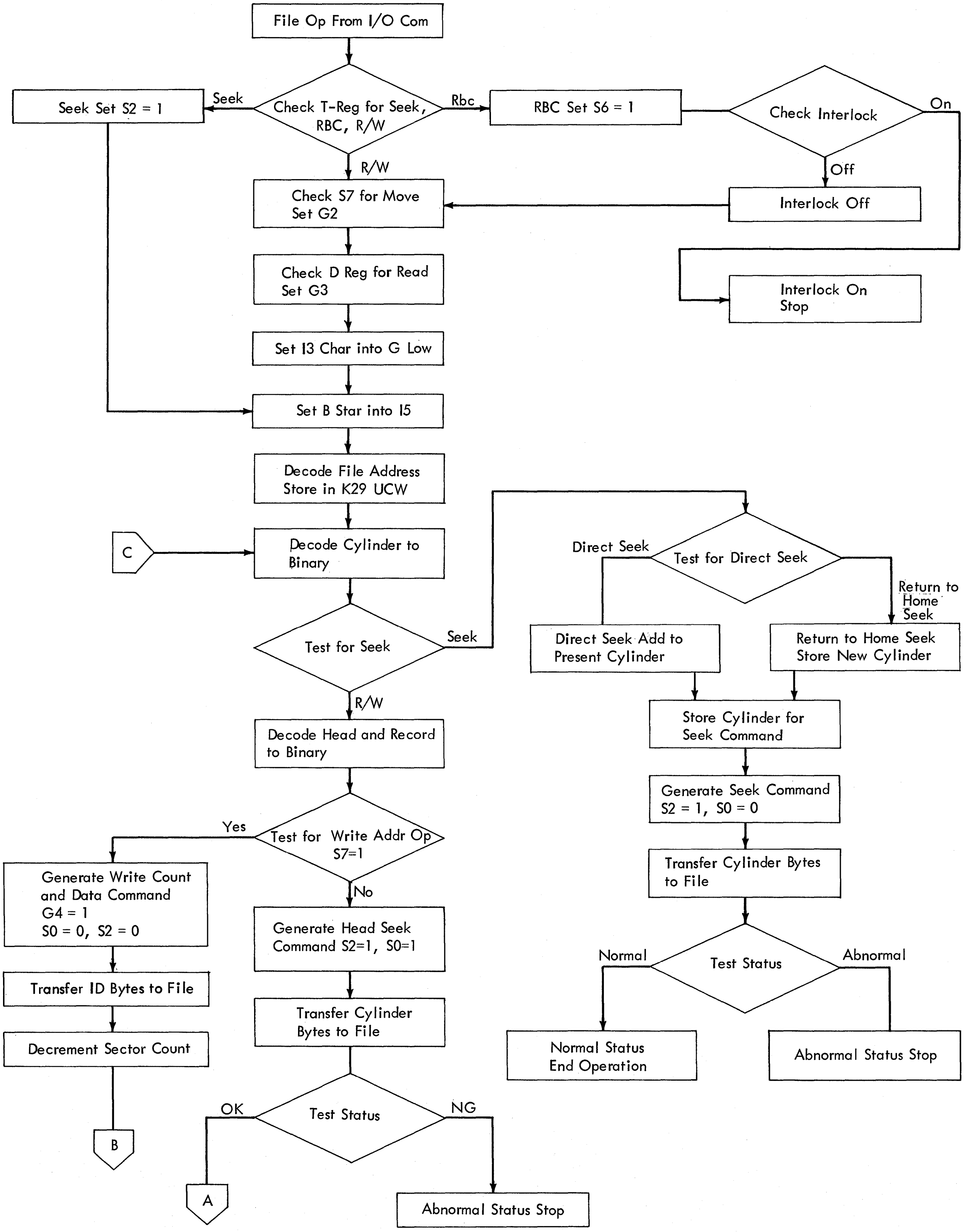

File Op From I/O Com
Check T-Reg for Seek, RBC, R/W
Seek
Seek Set S2 = 1
Rbc
RBC Set S6 = 1
Check Interlock
On
Off
Interlock Off
Interlock On Stop
R/W
Check S7 for Move Set G2
Check D Reg for Read Set G3
Set 13 Char into G Low
Set B Star into I5
Decode File Address Store in K29 UCW
C
Decode Cylinder to Binary
Test for Seek
Seek
R/W
Test for Direct Seek
Direct Seek
Return to Home Seek
Direct Seek Add to Present Cylinder
Return to Home Seek Store New Cylinder
Store Cylinder for Seek Command
Generate Seek Command S2 = 1, S0 = 0
Transfer Cylinder Bytes to File
Test Status
Normal
Abnormal
Normal Status End Operation
Abnormal Status Stop
Decode Head and Record to Binary
Test for Write Addr Op S7=1
Yes
No
Generate Write Count and Data Command G4 = 1 S0 = 0, S2 = 0
Transfer ID Bytes to File
Decrement Sector Count
B
Generate Head Seek Command S2=1, S0=1
Transfer Cylinder Bytes to File
Test Status
OK
NG
A
Abnormal Status Stop

Figure 5-19. Disk Storage Compatibility (Part 1 of 3)

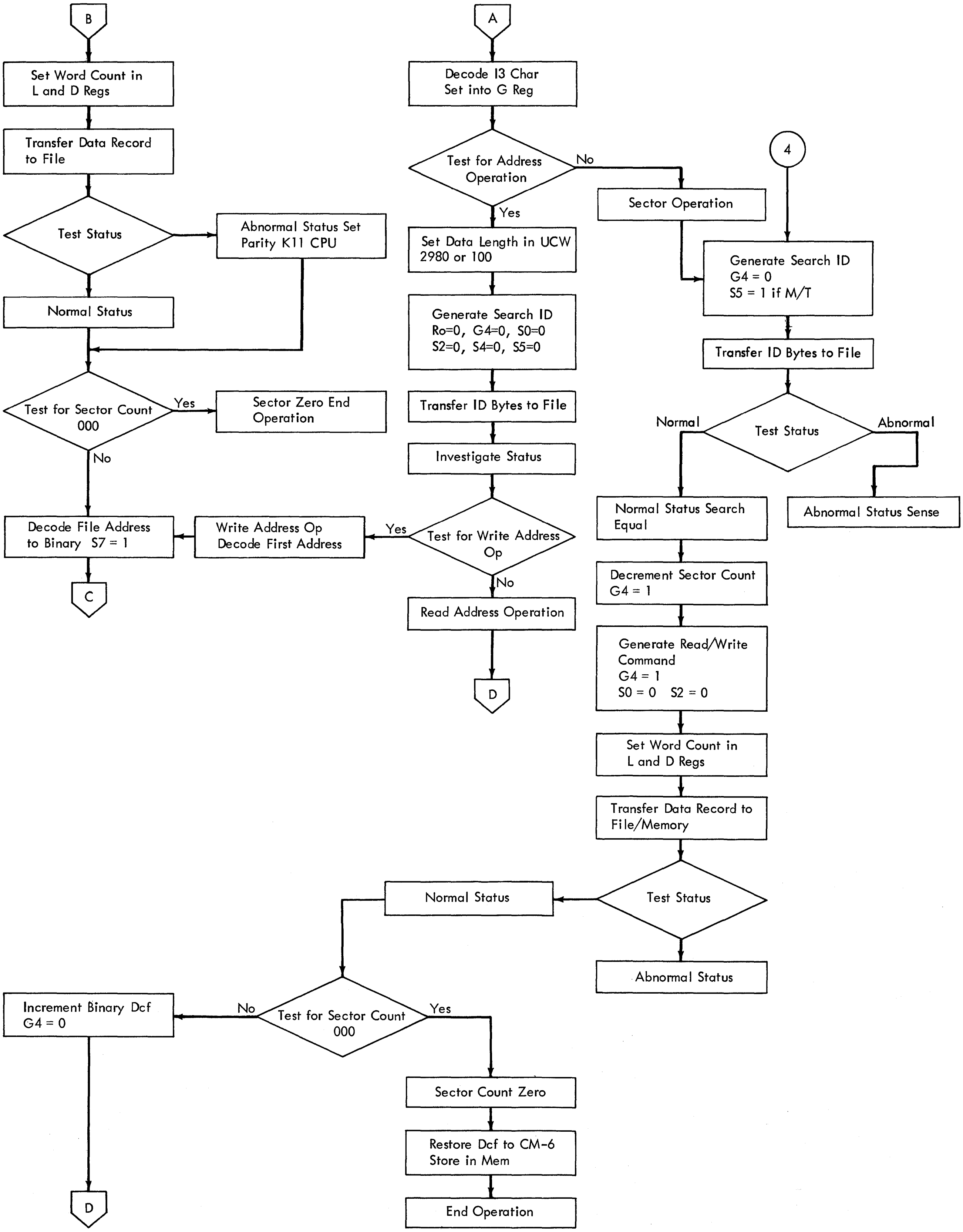
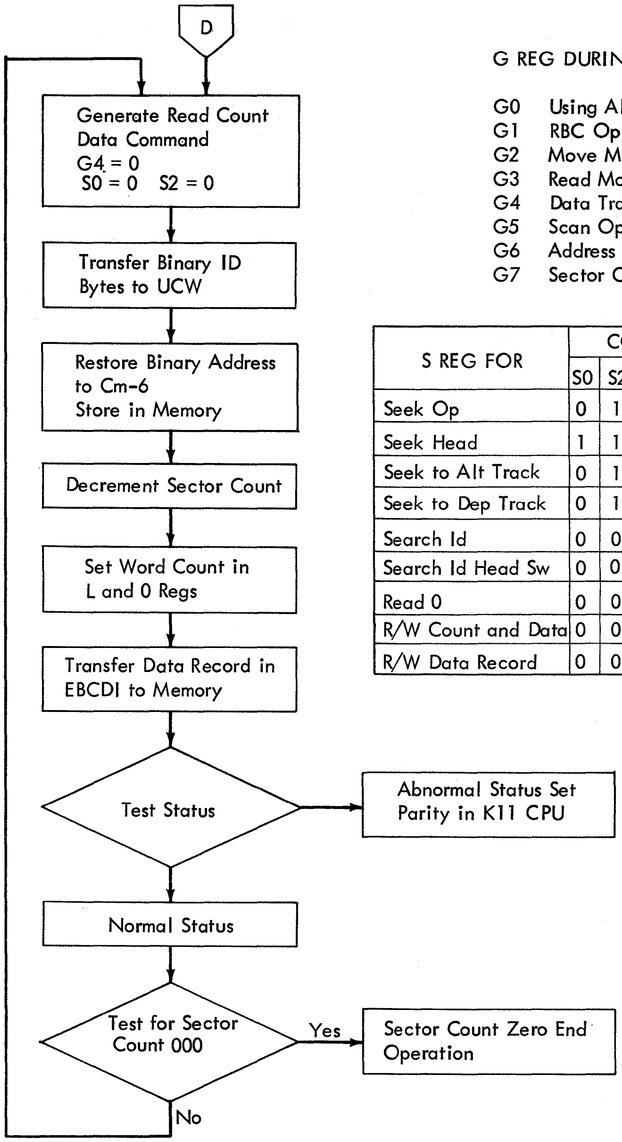

B
Set Word Count in L and D Regs
Transfer Data Record to File
Test Status
Abnormal Status Set Parity K11 CPU
Normal Status
Test for Sector Count 000
Yes
Sector Zero End Operation
No
Decode File Address to Binary S7 = 1
C
A
Decode I3 Char Set into G Reg
Test for Address Operation
No
Yes
Set Data Length in UCW 2980 or 100
Generate Search ID Ro=0, G4=0, S0=0 S2=0, S4=0, S5=0
Transfer ID Bytes to File
Investigate Status
Test for Write Address Op
Yes
Write Address Op Decode First Address
No
Read Address Operation
D
4
Sector Operation
Generate Search ID G4 = 0 S5 = 1 if M/T
Transfer ID Bytes to File
Normal
Test Status
Abnormal
Normal Status Search Equal
Abnormal Status Sense
Decrement Sector Count G4 = 1
Generate Read/Write Command G4 = 1 S0 = 0 S2 = 0
Set Word Count in L and D Regs
Transfer Data Record to File/Memory
Test Status
Normal Status
Abnormal Status
Increment Binary Dcf G4 = 0
No
Test for Sector Count 000
Yes
Sector Count Zero
Restore Dcf to CM-6 Store in Mem
End Operation
D

Figure 5-19. Disk Storage Compatibility (Part 2 of 3)


D
Generate Read Count Data Command G4 = 0 S0 = 0 S2 = 0
Transfer Binary ID Bytes to UCW
Restore Binary Address to Cm-6 Store in Memory
Decrement Sector Count
Set Word Count in L and 0 Regs
Transfer Data Record in EBCDI to Memory
Test Status
Abnormal Status Set Parity in K11 CPU
Normal Status
Test for Sector Count 000
Yes
Sector Count Zero End Operation
No

G REG DURING FILE OP

G0	Using Alt Track
G1	RBC Op
G2	Move Mode
G3	Read Modifier
G4	Data Transfer
G5	Scan Op
G6	Address Op
G7	Sector Count 000/End Op

S REG FOR	COMMAND						TRANSFER			
	S0	S2	S3	S4	S5	G4	S0	S4	S6	S7
Seek Op	0	1	0	0	0	X	0	0	0	0
Seek Head	1	1	0	0	0	X	0	0	0	0
Seek to Alt Track	0	1	0	1	1	X	1	0	0	0
Seek to Dep Track	0	1	0	1	0	X	1	0	0	0
Search Id	0	0	0	0	0	0	0	0	0	0
Search Id Head Sw	0	0	0	0	1	0	0	0	0	0
Read 0	0	0	0	1	0	1	1	0	1	0
R/W Count and Data	0	0	0	0	0	1	0	1	1/0	0
R/W Data Record	0	0	1	X	X	X	X	X	X	X

Figure 5-19. Disk Storage Compatibility (Part 3 of 3)

	00				01				10				11			
	00	01	10	11	00	01	10	11	00	01	10	11	00	01	10	11
0000					SP	&	-									0
0001							/		A	J						1
0010									B	K	S					2
0011									C	L	T					3
0100									D	M	U					4
0101									E	N	V		)		(	5
0110									F	O	W		b̸	=	v	6
0111									G	P	X		?	!	ǂ	7
1000									H	Q	Y					8
1001									I	R	Z					9
1010					±	!		b̸								
1011					.	$	,	#								
1100					⌑	*	%	@								
1101					[	]	⌒	:								
1110					<	;	\	=								
1111					⧧	△	⧻	√								

Figure 5-20. EBCDI Translation for 1050

Console Inquiry (1050)

When operating in compatibility mode, the 1050 perform the functions of the 1401 inquiry station. Since the graphic representation of 1400 system defined character is not the same as the 1050 graphic representation of EBCDI characters, some graphic conversion is necessary to obtain the correct character print out for 1400 system characters. The micro program converts the characters going to and coming from the 1050 in accordance with the chart in Figure 5-20. To illustrate this conversion, if a ? (11000000) is sent to the 1050, it is first converted by the micro program to 11000111 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 is 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 compatibility mode, to get the correct character printout. See Figure 5-21 for 1050 operational flow in compatibility mode.

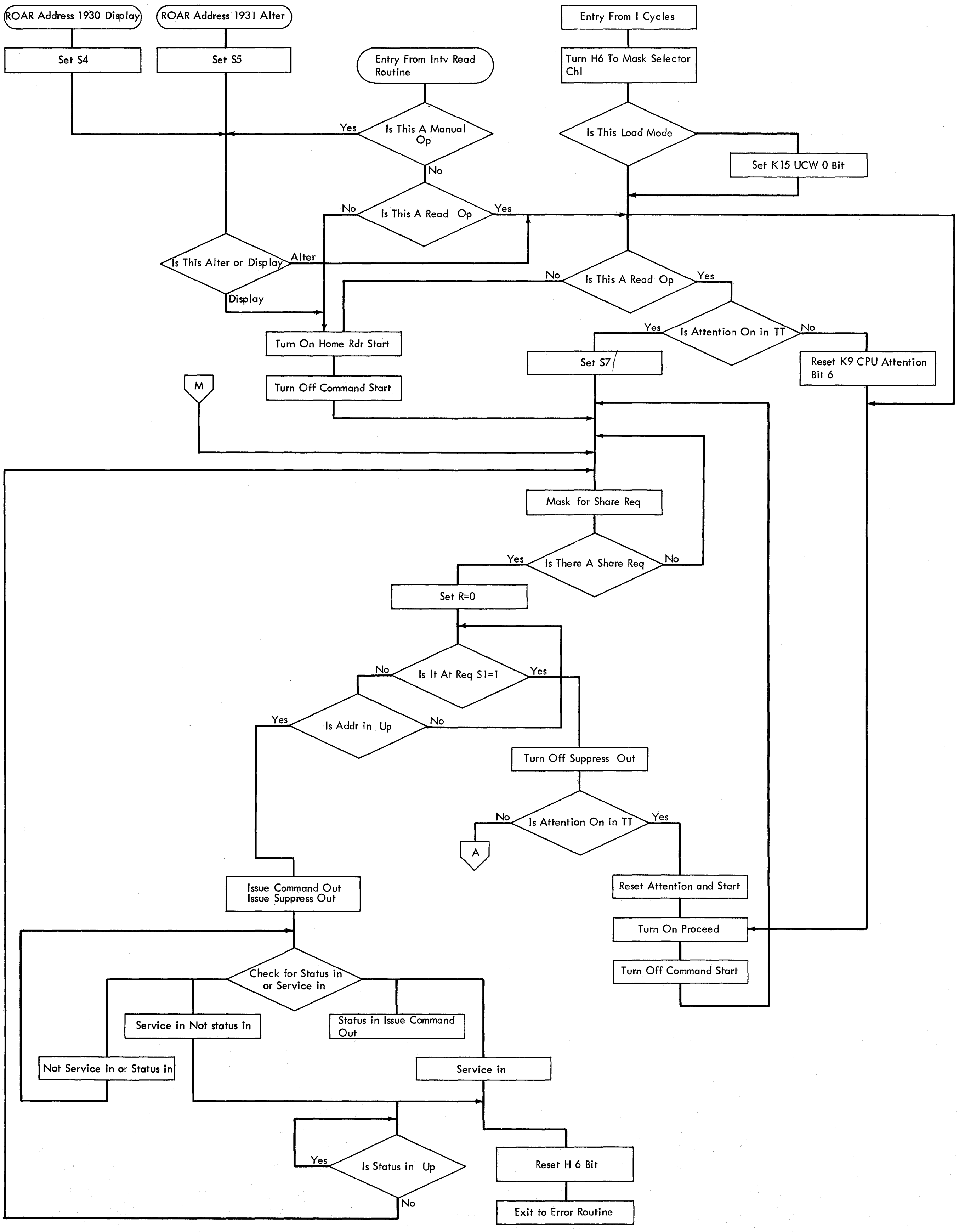


Figure 5-21. Compatibility Feature Console Inquiry (Part 1 of 5)

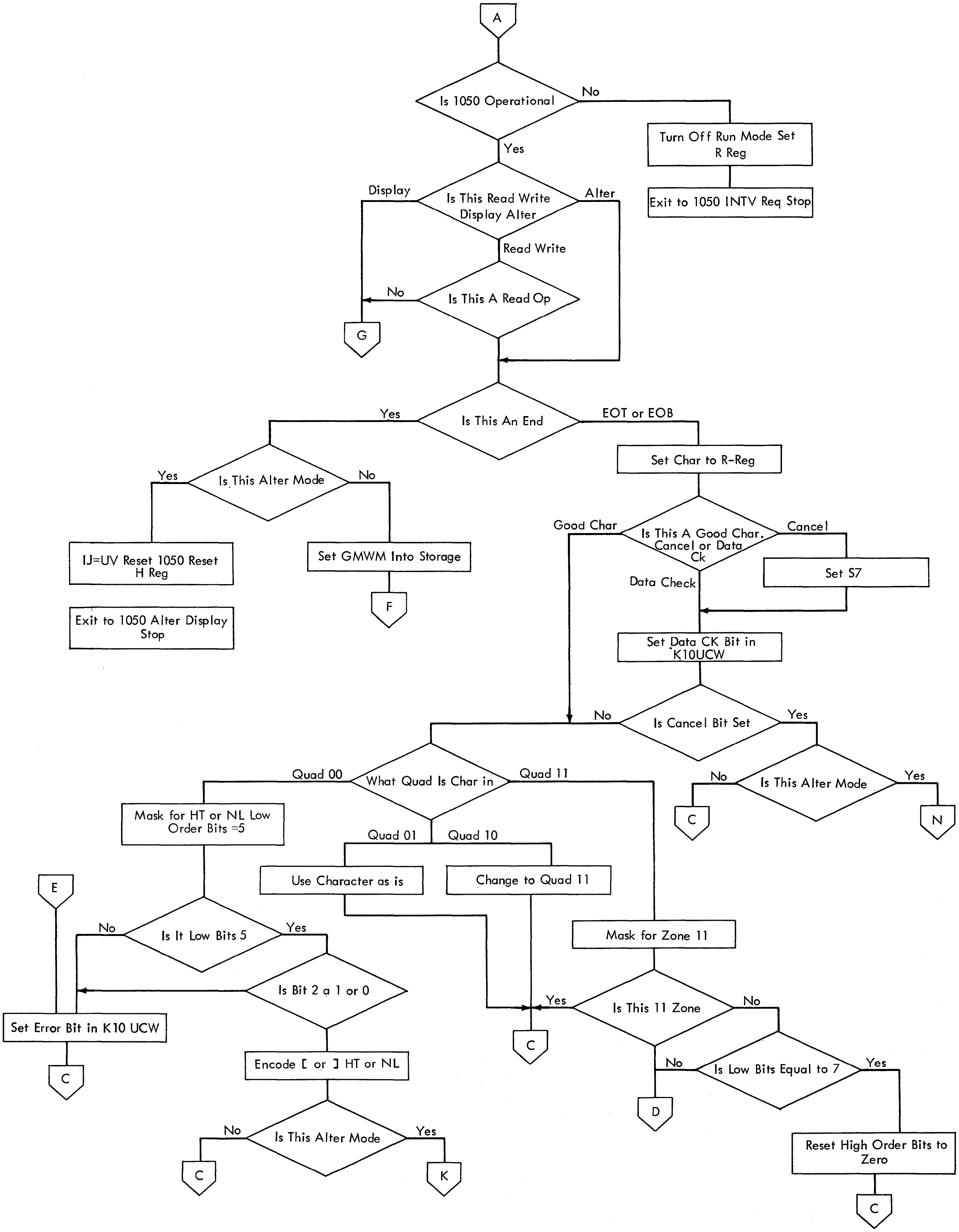


Figure 5-21. Compatibility Feature Console Inquiry (Part 2 of 5)

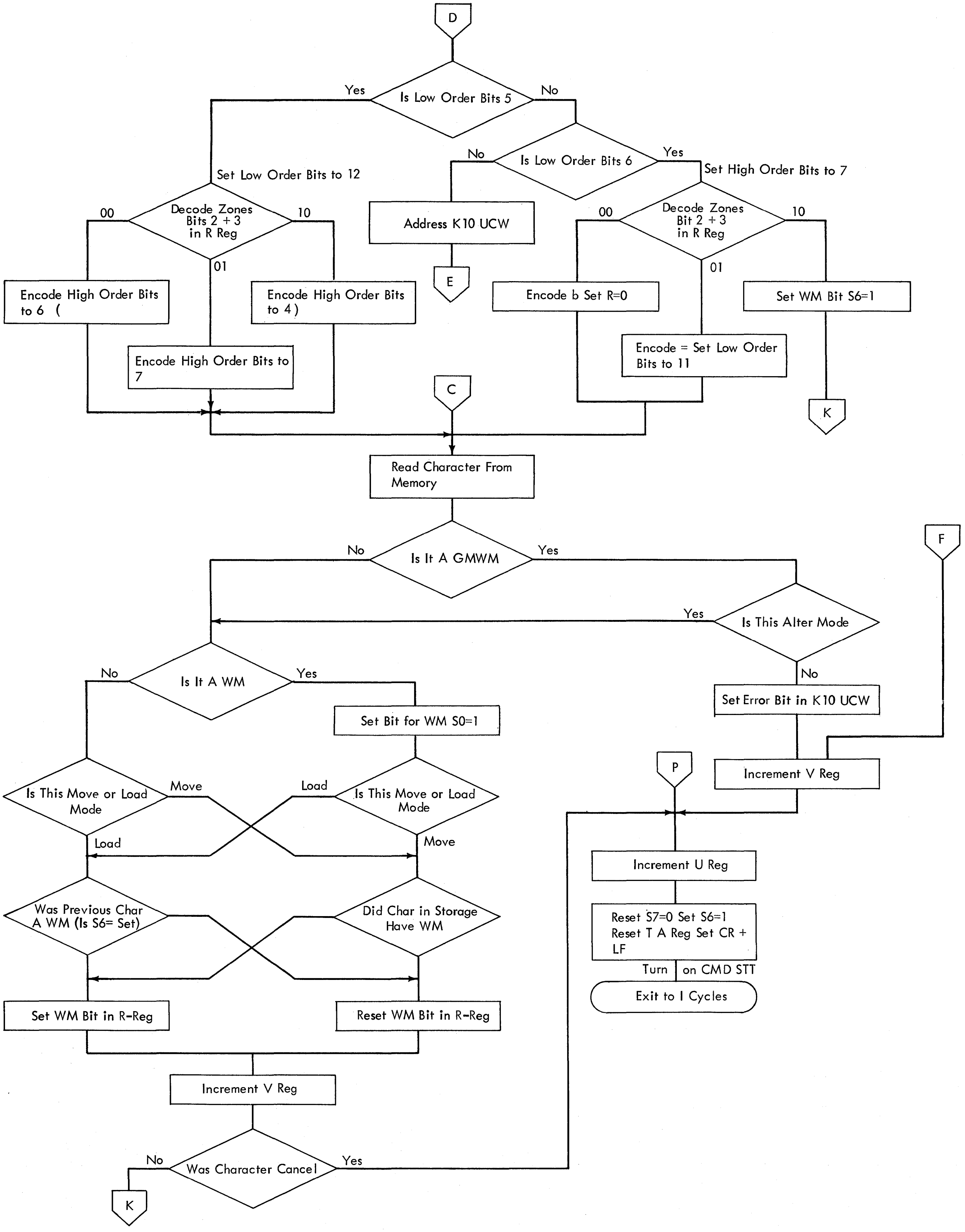


Figure 5-21. Compatibility Feature Console Inquiry (Part 3 of 5)

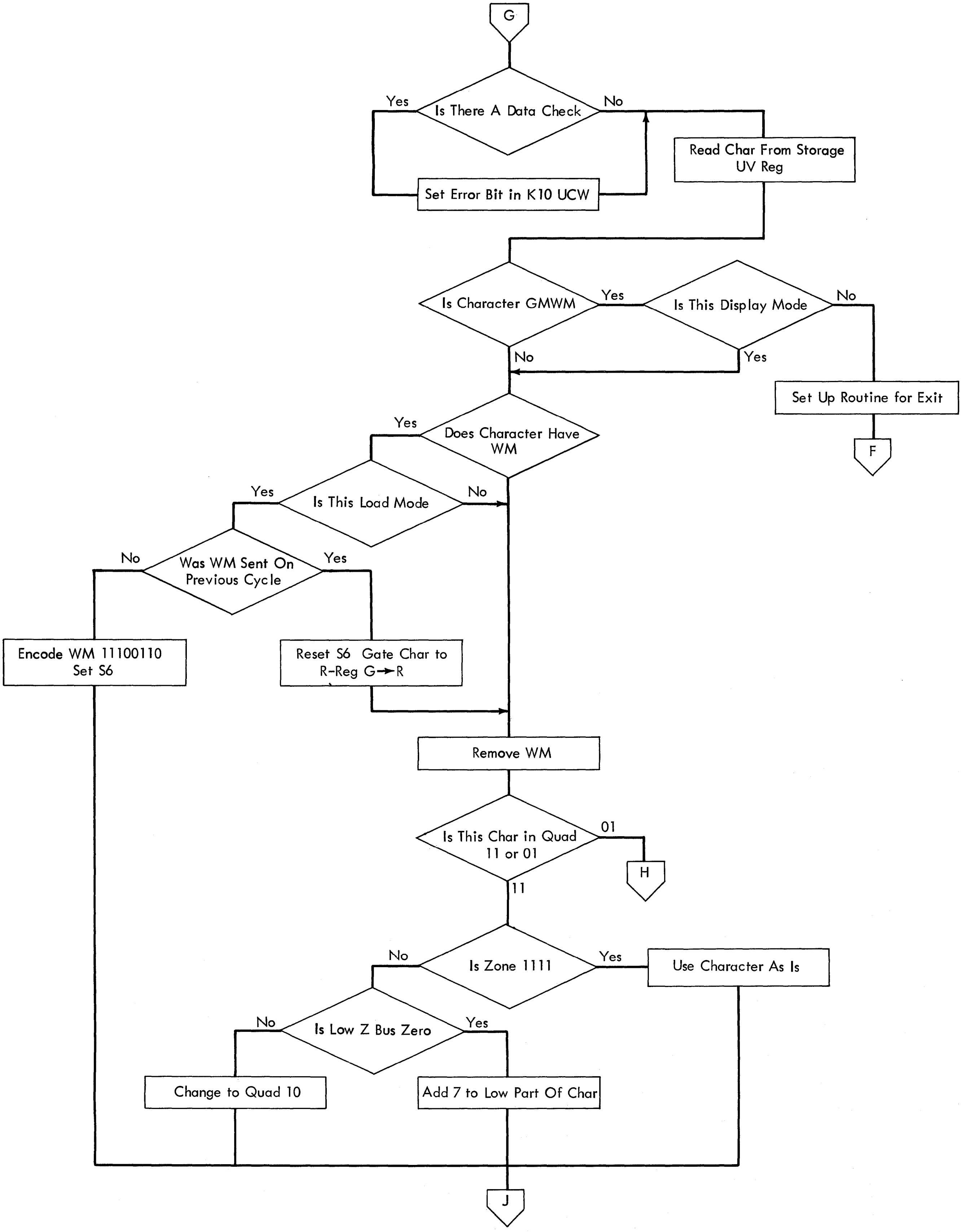


Figure 5-21. Compatibility Feature Console Inquiry (Part 4 of 5)

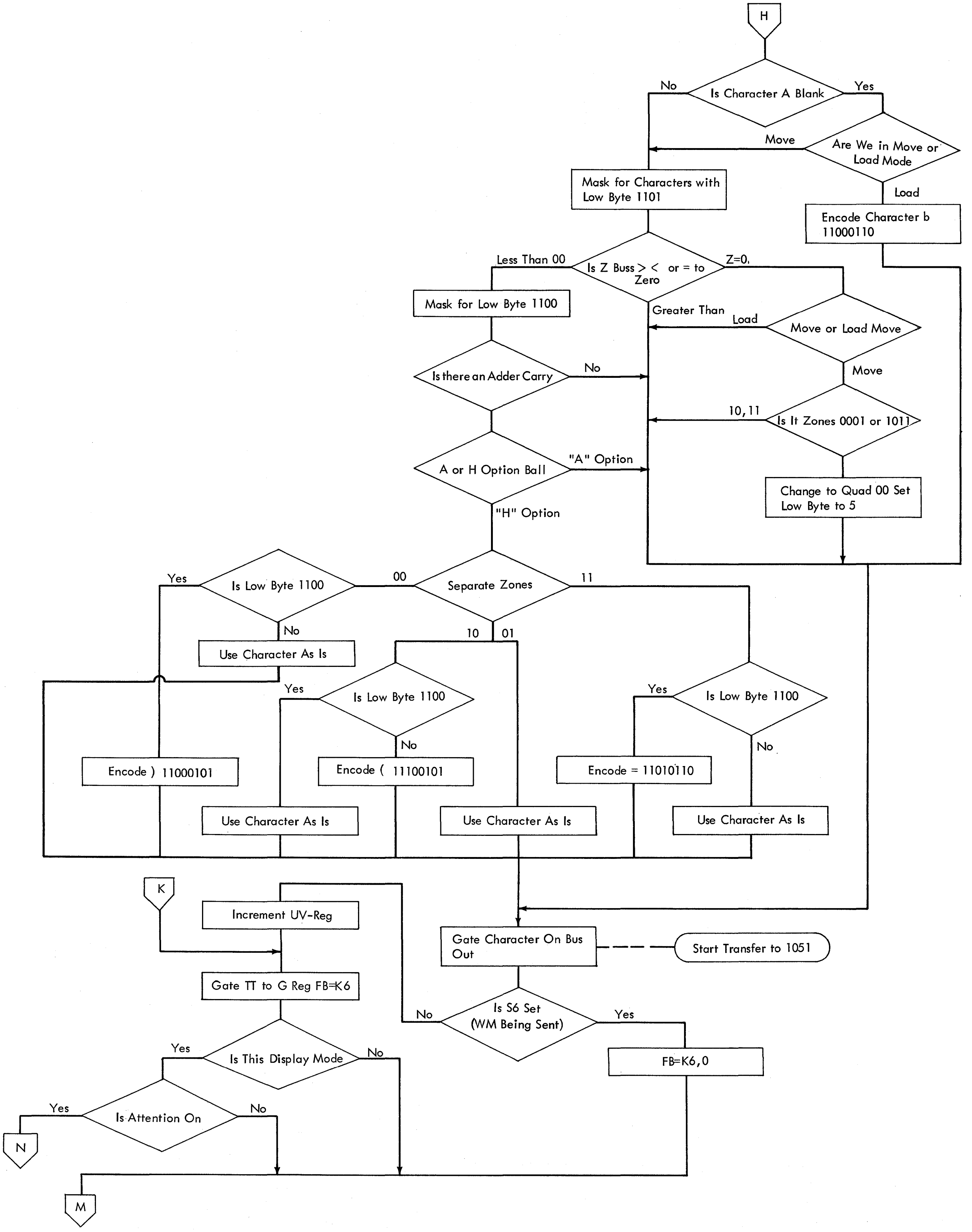


Figure 5-21. Compatibility Feature Console Inquiry (Part 5 of 5)

ERRORS AND PROGRAM HALTS IN COMPATIBILITY MODE

- A common micro-program halt routine is used to display IJ, LT and a coded byte in the R-register to indicate the cause of an error.

A common stop routine is micro programmed to handle all stops in compatibility mode that can be brought to a stop through micro programming. This routine accomplishes the following:

1. Stores the address registers along with the D-, G-, and S- register in their local storage backup locations.

2. Converts all addresses to the form:

M		N	
4bits 1	4bits 2	4bits 3	4bits 4

   Where 1 denotes the thousands posi- of a 1400 system address in binary.

   Where 2 denotes the hundreds position of a 1400 system address in decimal.

   Where 3 denotes the tens position of a 1400 system address in decimal.

   Where 4 denotes the units position of a 1400 system address in decimal.

3. Displays the converted IJ address in the MN lights.

4. Displays the converted LT address in the B- and A- register lights.

5. Displays a coded byte in the R-register lights which indicates the cause of a stop to the operator.

The stop routine does not run invalid addresses through the conversion routine. However, it does display the addresses in their binary form, so the CE can inspect their contents to diagnose the error.

The coded byte displayed in the R-register lights is backed up in byte K0 of MPX 1 storage before the 2030 stops (Figure 5-22).

	0	1	2	3	4	5	6	7
0X		INVALID B ADDRESS	INVALID A ADDRESS	INVALID A & B ADDRESS	INVALID OP CODE	INVALID I/O OP	MEMORY WRAP-HIGH	MEMORY PROTECT
1X	FILE-RBC STOP		NO ADDRESS RESPONSE SX					
2X	FILE-NO ENDS		ADDRESS MISMATCH					
3X	FILE-NO ADDR COMP		INV UNIT STATION INIT SEL-SX					
4X	FILE-SEEK UNIT CHECK	RDR INTV REQ'D	INV CHNL STATION ON INIT SEL					
5X	FILE-OPNL INTLK	PCH " "	DEV ENDING 1ST ON TAPE WR					
6X	NON-ZERO STATUS RESPONSE TO SENSE COMMAND	PRT " "	OP IN DISC ON TAPE WR-MPX STI & SVI ON TAPE NR-SX					
7X	FILE-UNIT # CK COMM OUT	LATE STACKER SELECT 1402	WS SENT LAST, NO WM THIS SX					SENSE SWITCH PERFORMED
8X	42-43-NO ADDR COMP	1050 INTV REQ'D	STI & SVI ON READ MOVE-SX					
9X	42-43-INV MODIFIER	HALT & BRANCH	TAPE ERROR ON IPL					
AX	42-43-NO GMWM	ALTER-DISPLAY STOP	INV CHNL STAT FOR BR ON SX ERR					
BX	42 ERR ON RD OR PCH	1402 READER ERROR	STI & SVI ON READ LOAD-SX					
CX			OP IN DISC ON READ-MPX					
DX			PREMATURE END ON SENSE-MPX					
EX			OP IN DISC ON MODE SET-MPX					
FX	HALT-NO INV ADDR*	HALT-INV B * ADDR*	HALT-INV A ADDR*	HALT-INV A & B ADDR*				

* THESE STOPS INDICATE EXECUTION OF THE 1401 HALT INSTRUCTION (.) WITH THE STATUS OF THE A & B ADDRESSES THEN EXISTING SHOWN

	8	9	A	B	C	D	E	F
0X	MODE SW ATTEMPT	INV ADDR ON PMS TAPE OP		MEMORY WRAP LOW	IJ=ZERO	LOAD ADDR INVALID	IMPROPER INDEXING NO ADV PROG FEATURE	NON 1050 DATA CYCLE
1X								NO OP WORD MARK
2X								WM IN I/O A ADDR
3X								
4X								
5X								
6X								
7X								
8X								
9X								
AX								
BX								
CX								
DX								
EX							START RST PERFORMED	
FX								SET IC PERFORMED

Figure 5-22. R-Register Error Bytes

If the start button is pushed after coming to a stop as just described, a restart routine is started. All registers stored by the stop routine are restored to the values then in their respective storage backup locations. A new I-cycle is started at the address in IJ, unless the stop was due to an I/O device needing operator intervention for such things as full stackers, out of paper, etc. In the latter case, the restart routine returns to a specific micro-program entry for that particular device.

The restart routine also turns on bit 0 of byte K1 in MPX 1 storage. This bit is used to interlock against giving a stacker select instruction after restarting, which applies to a card read operation given before the stop occurred. If a stacker select instruction is given while bit 0 is on, the CPU stops again and indicates with a coded byte in K0 of MPX 1 storage that the operator should check the last card stacked to be sure it is in the right stacker. A new card read or punch instruction resets bit 0 of K1 MPX 1 storage.

PROGRAMMED MODE SWITCH

- The 2030 can switch modes from compatibility mode to 2030 mode and vice-versa under program control.

- Local storage and MPX 1 for compatibility mode are relocated to MPX 2 and MPX 3 storage respectively.

- A program mask in K12 and K13 controls when a mode switch is taken. is taken.

- Additional diagnose instructions are provided to facilitate programmed mode switching.

Programmed Mode Switching is a special feature made available to the customer to allow switching the 2030 from compatibility mode to 2030 mode and vice- versa under 2030 control. It provides the customer the means to utilize some System/360 capabilities that are not available in compatibility mode. The Programmed Mode Switch option allows the customer to place IBM 2030 programs and IBM 1400 systems programs in storage at the same time. A switch from compatibility mode to 2030 mode can occur due to any of the following conditions:

Mode Switch on Invalid Op code.
Mode Switch on Halt Op.
Mode Switch on Error stops.
Mode Switch on Invalid I/O Ops.
Mode Switch on Console Ops.
Mode switch on Printer Ops.
Mode Switch on Reader/Punch Ops.
Mode Switch on Tape Ops.
Mode Switch on File Ops.

Provisions are made to allow any of the preceding conditions to cause a supervisor call interrupt. Normally, these conditions would cause the processor to come to a micro-programmed stop. Whether or not a mode switch actually occurs is controlled by the 2030 supervisor program. A program mask for mode switching is contained in local storage locations K12 and K13. If a mode switch occurs, all current 1400 system addresses and the contents of the 1400 A-Star are stored in backup locations in local storage. The Supervisor Call old PSW contains a special code to indicate the type of error. It also contains the 1400 system instruction address of the last Op code processed.

The programmed mode switch-feature is only available on a 2030 having 16,384 or more positions of core storage. When programmed mode switch is installed, local storage and MPX 1 tables and control bytes needed to run in compatibility mode are relocated to MPX storage 2 and 3, respectively. This leaves the CPU local storage and MPX 1 storage available to the 2030 mode of operation. A circuit change has been added to cause MPX 2 storage to be addressed in compatibility mode if the micro program calls for information in local storage. For example K11, local storage addresses K11 MPX 2 in the 2030. If the program desires to address local storage in compatibility mode, the micro-program statement MLS causes local storage to be addressed. In compatibility mode when the programmed mode switch is present, the compatibility micro program cannot address MPX 1 storage.

Because extra diagnose instructions are available to the programmer to allow communication between the 2030 program and the 1400 system program. Review System Reference Library publication for the compatibility and programmed mode switch features.

QUESTIONS ON COMPATIBILITY FEATURE

1. If a 2030 with 32,768 positions of storage is simulating a 1401 with 8K storage, 1401 storage location 1056 is located in storage location ______ in the 2030?

2. During I cycles, the G-register content for a 1401 clear Op instruction would be ______?

3. What is the purpose of the invalid character (8F) that is placed one position below the low-order position of storage used by the 1401 object program.

4. For what purpose are 1400 system Op codes converted?

5. How is the storage bias constant derived?

6. If system reset is pressed when operating in compatibility mode, what will happen?

7. What will happen if a 1402 stacker select command is issued 11 milliseconds after a read command?

8. If the programmed mode switch-feature is present on the machine, where will the compatibility mode conversion and control tables be found?

9. If an invalid 1400 system character addresses local storage during conversion of the character from EBCDI to BCD, what bit configuration will be read out?

10. What internal character code is used when processing data in compatibility mode?

INTERVAL TIMER

INTRODUCTION

- The interval timer consists of three bytes of data in main storage locations 50, 51, and 52 (HEX).

- 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.

What good is the interval timer? Let's make a few assumptions. Assume that a customer must run two jobs during the day. Job #1 takes seven hours of running time. The information to run Job #2 is not available until 2 P.M. By using the interval timer feature the customer, in effect, can tell 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 might then set the timer for 17 minutes. The extra 2 minutes are his safety margin. Then, whether Job #2 is completed or not, the work is halted after 17 minutes and Job #1 is automatically resumed.

How is the timer used? To use this feature, the customer will set a certain value in main storage locations 50, 51, and 52. This automatically starts a counter which keeps track of time. The value that is set in main storage by the customer represents total elapsed time. When the counter value is subtracted from the timer value enough times, the timer value will go 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.

How is the timer value computed? First, the high-order bit of location 50 is reserved for sign control. This leaves the other 23 bit positions (7 positions in byte 50 plus the two bytes 51 and 52) free for data. If you use a Powers of 2 table you can see that a value of over 16700000 can be set with 23 bit positions. A micro program subtracts the value of 300 from the timer for every second of elapsed time. As a result, the full cycle time of the timer is about 15.5 hours. If 300 is subtracted from the timer for an elapsed time of one second, then the timer must be set to the value of 1080000 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 depends on the setting of the 4 position, binary connected 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 interrupt at the end of E-phase.

- For machines operating on 60 cycle alternating current.

The C counter is used to keep track of time. A 60 cycle pulse provides the drive for this counter. 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 will cause a timer interrupt.

The interrupt routine takes the value in the C-counter, multiplies it by 5, and subtracts this 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 the interrupt routine exists to I-phase for the next instruction. The big question in your mind is probably, "multiply by 5?" Remember, the value of 300 is subtracted from the timer for one second of elapsed time. For .25 of a second, the value of 75 must be subtracted from the timer. If the C counter has only four positions, the highest value it can contain is 15 (1111, all positions set). All positions of the C-counter are set in .25 of a second. Therefore, the full value in the counter (15) multiplied by 5 gives the value (75) that must be subtracted from the timer for this elapsed time.

The controls for the C-counter are shown in Figure 5-23. 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 latch will be turned on when main storage location 51 is addressed. This location is addressed when the customer desires to set the timer to a new value. With the control latch on, the control FF latch is set at T4 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.

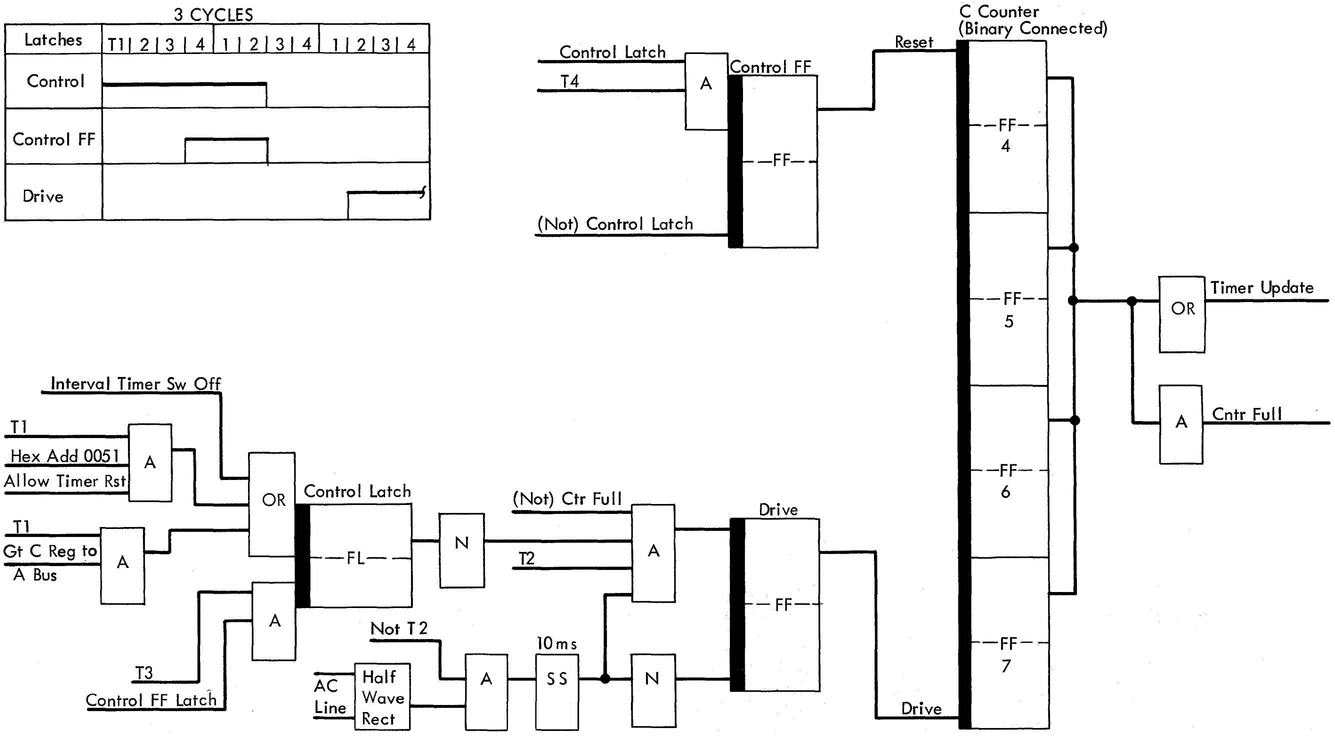


Figure 5-23. Interval Timer Controls

With the control latch off and the C counter empty, the 60 cycle time pulse will set the drive latch at T2 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 interrupt is signaled.

During the interrupt routine, the setting in the C-counter is set into the D-register (C->D) for the 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 interrupt routine is shown in Figure 5-24.

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 timer during timer update.
- For machines operating on 50 cycle alternating current.

If the 2030 is operating on 50 cycle alternating current, a slight change in

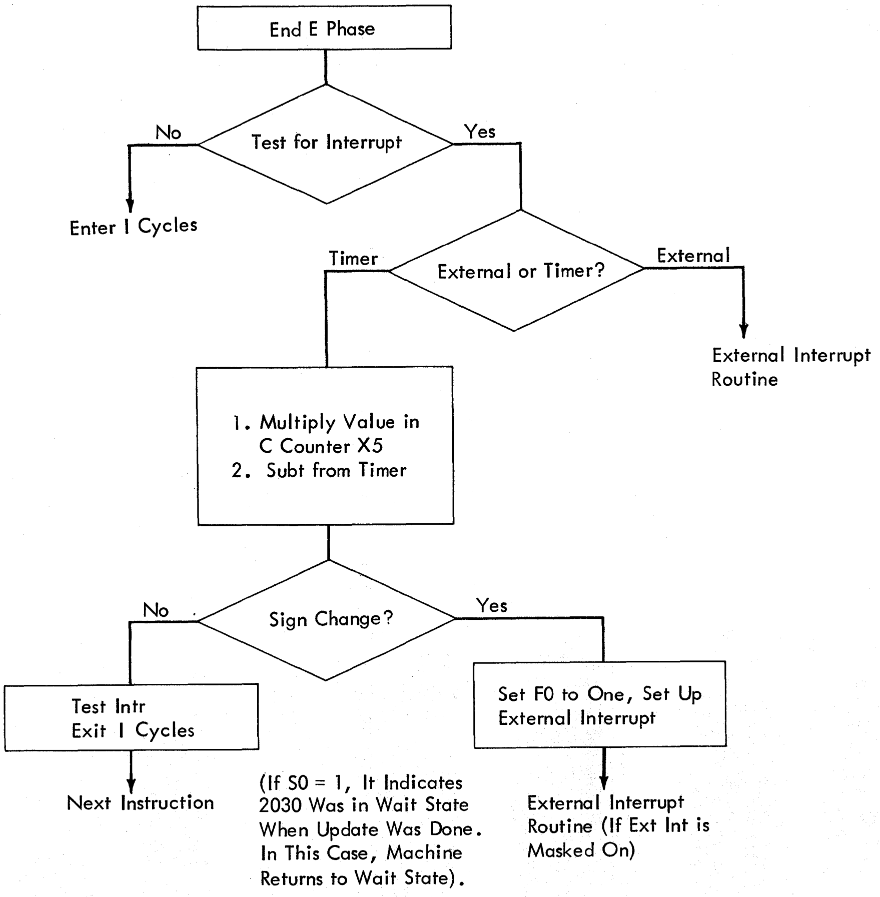

End E Phase
Test for Interrupt
No
Yes
Enter I Cycles
External or Timer?
Timer
External
External Interrupt Routine
1. Multiply Value in C Counter X5
2. Subt from Timer
Sign Change?
No
Yes
Test Intr Exit I Cycles
Set F0 to One, Set Up External Interrupt
Next Instruction
(If S0 = 1, It Indicates 2030 Was in Wait State When Update Was Done. In This Case, Machine Returns to Wait State).
External Interrupt Routine (If Ext Int is Masked On)

Figure 5-24. Timer Update Micro Program

the timer update controls takes place. Instead of multiplying the counter value by 5 to get the correct number to subtract from the timer, the 50 cycle machine timer interrupt routine multiples 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. We still want the value 300 to be subtracted from the timer each second. 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 allows timer programming to be compatible for all machines.

APPENDIX 1. SYSTEM CHARACTERISTICS

											Service				
						Conn		Dimensions (inches)			Clearances (inches)				
Type	Mod	Description	BTU/Hr	KVA	CFM	Type	Weight	H	F	S	F	R	Rt	L	Notes
360	30		10,000	3.8	900	B	1,500	60	68	84	42	18	60	30	
360	40		6,000	2.8	300	D	1,700	60	60	109	48	48	30	72	
360	50F, G		14,900	9.0	2,350	E	4,700	72-1/2							4
360	50H		18,700	10.6	2,990	E	5,350	72-1/2							4
360	60		28,000	12.4	2,840	E	2,800	72-1/2							4
360	62		12,000	6.9	2,100	E	2,400	72-1/2							4
360	70		38,700	16.2	2,700	E	4,575	72-1/2							4
1015	1	Inquiry Display Terminal	900		0		375	47	48	29	36	6	36	30	4, 11
1015	2	Inquiry Display Terminal	900		0		300	47	48	29	36	6	36	30	4, 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	36	0	30	
1052	1	Printer-Keyboard	335	0.1	0		65	9	23	19-3/4	0	0	0	0	12
1231	N1	Optical Mark Page Rdr	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	
1302	N1	Disk Storage	20,000	9.0	2,210	E	4,025	68-3/4	85-1/2	33	44	44	40	40	2
1302	N2	Disk Storage	28,000	12.0	2,210	E	4,400	68-3/4	85-1/2	33	44	44	40	40	2
1402	N1	Card Read Punch	2,600	1.2	50		1,000	45-1/4	57-1/2	29-3/4	36	36	36	36	3
1403	2	Printer	3,000	1.0	310		750	53-1/4	47-3/4	28-1/2	36	36	30	30	3
1403	3	Printer	4,600	1.4	350		750	53-1/4	47-3/4	28-1/2	36	36	30	30	3
1404	2	Printer	5,100	2.1	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	
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	
1418	2	Optical Char Rdr	8,300	3.8	575	D	2,700	60-1/4	112	41-1/4	42	48	36	36	
1419	1	Magnetic Char Rdr	8,500	3.3	400	C	2,675	60-1/4	112	41-1/2	42	48	36	36	
1428	1, 3	Alphameric Optical Rdr	10,500	4.6	575	D	2,750	60-1/4	112	41-1/4	42	48	36	36	
1428	2	Alphameric Optical Rdr	10,500	4.6	575	D	2,800	60-1/4	112	41-1/4	42	48	36	36	
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	50	
2150		Console	1,740	0.65	180	B	800	52-1/8	64	28-3/4	30	48	30	30	
2201	3	Printer	4,600	1.4	350		825	53-1/2	57-1/8	29	36	36	42	42	3
2250	1	Display Unit	7,200	2.8	480	A	590	50							4
2250	2	Display Unit	6,600	2.4	320	A	375	50	22	28	30	30	30	30	
2301		Drum Storage	3,800	1.5	320	D	850	64	34-1/2	29	48	48	42	42	
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	
2360	1, 2, 3	Core Storage	2,500		1,000		1,200	70		72					5, 12
2361	1	Core Storage	8,200	3.0	620	D	1,600	70-1/2	62-1/4	31-3/4	72	42	30	36	2
2361	2	Core Storage	8,200	3.0	620	D	1,700	70-1/2	62-1/4	31-3/4	72	42	30	36	2
2362	1, 2	Storage	33,000	12.5	2,150	E	2,560	70							5
2401	1, 2, 3	Magnetic Tape Unit	3,500	1.6	500		800	60	30	29	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	
2671		Paper Tape Reader													10
2701		Data Adapter Unit	1,200	0.3	120	A	320	40	40	25-1/2	42	42	30	42	
2702		Transmission Control	1,800	1.0	800	A	900	60	28-3/4	61-1/2	30	18	42	30	
2802		Hypertape Control	1,360	0.6	300	F	425	60	28-3/4	61-1/2	30	30	42	42	
2803		Tape Control	2,500	1.0	500	E	1,400	60	60	29	42	42	30	30	
2804		Tape Control	4,000	1.5	700	E	1,600	60	60	29	42	42	30	30	
2816	1, 2	Switching Unit	1,500	0.9	280	A	500	60	29	42	30	18	42	42	
2820		Drum Storage Control	4,000	1.5	400	C	650	60	28-3/4	61-1/2	30	30	42	42	
2821	1, 2, 3, 4	Control Unit	7,000	2.4	300	D	1,000	60	32	46	30	18	48	42	
2822		Paper Tape Rdr Ctrl Unit	1,700	2.05	150	A	400	40	30	24	30	30	30	30	
2840		Display Control	4,800	1.4	300	A	550	60	29	42	30	30	30	30	
2841		Storage Control Unit	5,500	1.9	1,000	D	750	60	31	42	30	30	48	30	
2860	1	Selector Channel	8,200	3.05	420	D	1,800	71	31	70	30	36	66	66	
2860	2	Selector Channel	10,000	3.65	740	D	1,900	71	31	70	30	36	66	66	
2860	3	Selector Channel	11,600	4.25	1,060	D	2,000	71	31	70	30	36	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
7770	3	Audio Response Unit				A		70		31-1/2					4
7772	3	Audio Response Unit				A		43		24					4

NOTES:

1. For airflow, see specifications page for 1302 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

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

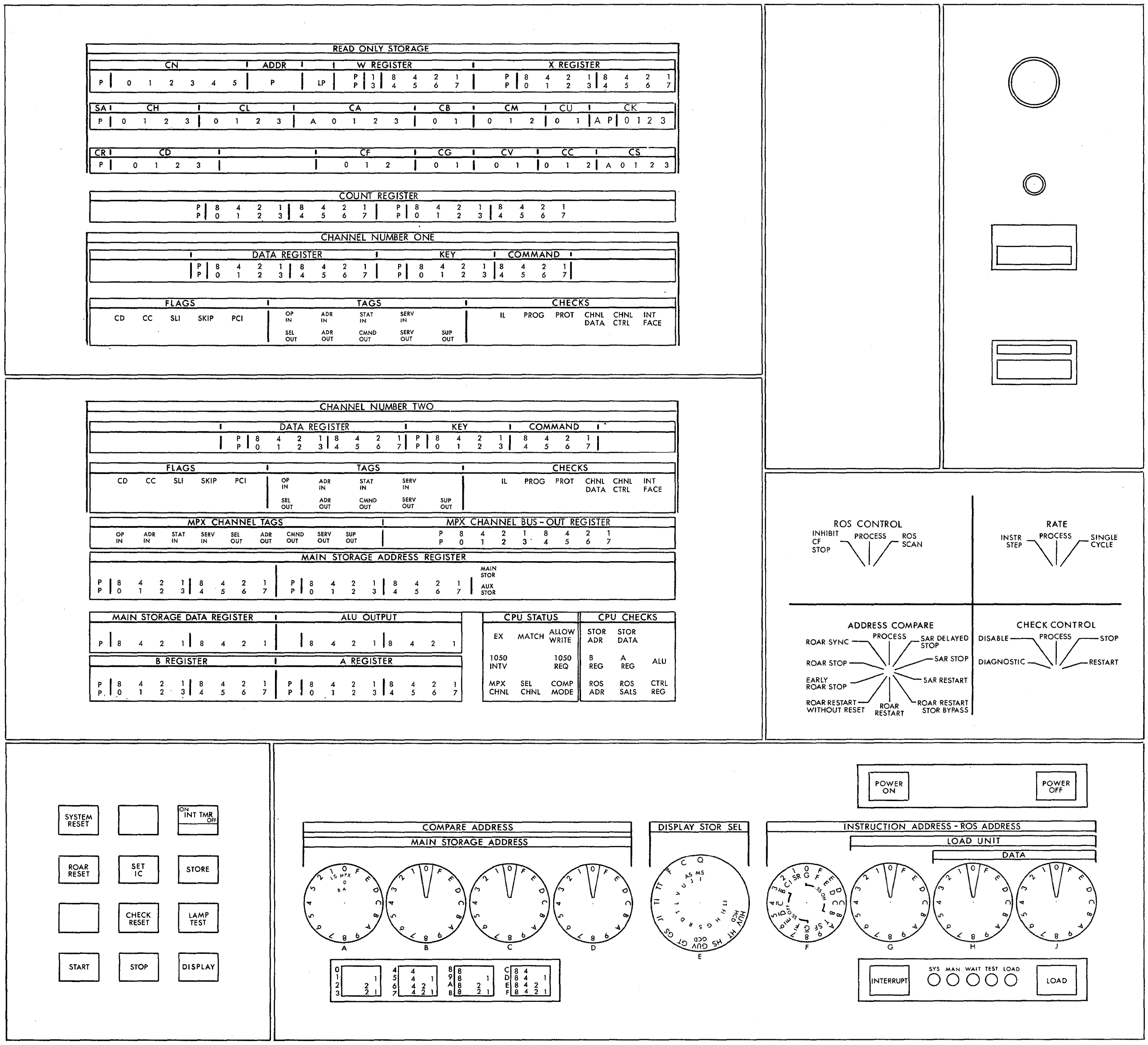


Figure 6-1. IBM 2030 Control Panel

- Controls are on IBM 2030 control panel (Figure 6-1).

- 2030 panel is divided functionally into seven sections.

- 2030 panel allows display and alteration of data, and control and status information.

UPPER INDICATOR PANEL (FIGURE 6-2)

READ ONLY STORAGE DISPLAY

- Display consists of fifteen bits of address and fifty-four output bits.

- When the system stops, the displayed address is normally the address of the displayed control word.

Description

Whenever the system stops, the displayed address is the address of the displayed control word, except when the address-compare switch is in the Early Roar Stop position and a match occurs, or the check control switch is in the Stop position: and one (or more) of the following checks occur: CTRL-REG, A-REG, B-REG, ALU, STOR DATA, or STOR ADDR.

COUNT REGISTER DISPLAY

- A group of eighteen indicators.

- Allow the contents of either the GCD or HCD register to be observed.

CHANNEL NUMBER ONE DISPLAY

- A group of indicators provided to observe Selector Channel One operation.

DATA REGISTER

- A display of the channel data register for channel one.

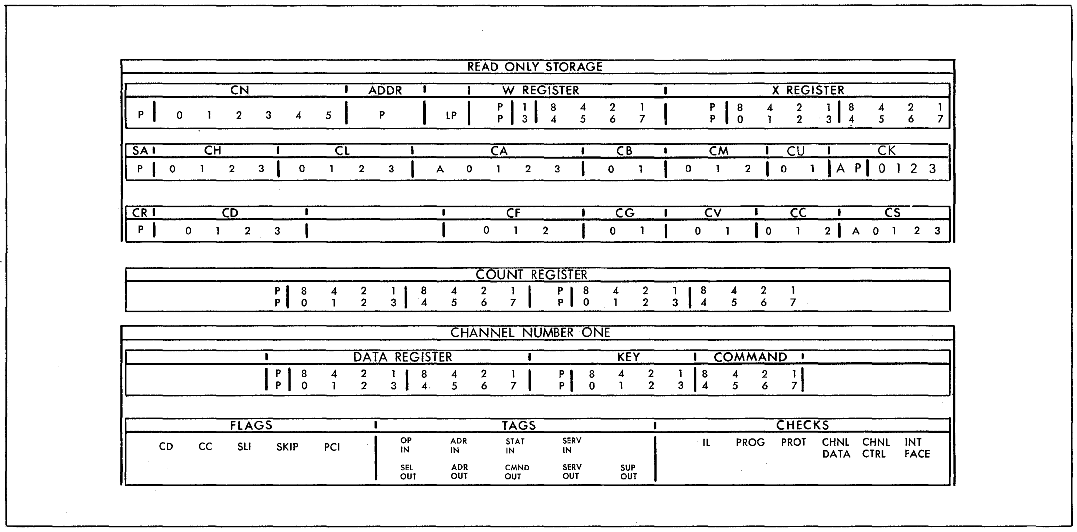


Figure 6-2. Upper Indicator Panel

KEY

- A display of four bits plus parity.
- Indicates the main storage protection key for all commands associated with the start I/O instruction.

COMMAND

- These four bits, when decoded, identify the channel command.

FLAGS

- This group of five indicators displays the manner in which a channel command is executed.
- These indicators, when on, have the following meaning:

  CD Indicates chaining of data addresses

  CC Indicates command chaining

  SLI Indicates that the program will not be notified in the event of wrong length record.

  SKIP This bit is turned on when it is desired to inhibit writing data into storage during Read, Read Backwards, or Sense operations.

  PCI This is the Program Controlled Interrupt bit, which permits the channel to generate an interrupt upon fetching the CCW.

TAGS

- Nine selector channel tags are displayed as follows:

  OP IN-Indicates that an I/O unit has been selected and is in communication with the channel.

  ADDR IN- Indicates that the address of the currently selected I/O unit is on Bus In.

  STAT IN - Indicates that the select- I/O unit has placed status information on Bus In.

  SERV IN - Indicates that the selected I/O unit is ready to transmit or receive data.

  SEL OUT - When this light is on, the several I/O units are being polled to determine which unit requested service.

  ADDR OUT - Indicates that the information on Bus Out is an address.

  CMND OUT - Indicates that the information on Bus-Out is a command.

  SERV OUT - Indicates that the CPU has accepted the information on Bus In or has provided data on Bus Out.

  SUPP OUT - This signal used alone or with other tags indicates the following functions:

  Suppress Status
  Suppress Data Transfer
  Chained Command Control
  Selective Reset.

CHECKS

- Indicates a detected malfunction during selector channel operation.
- The Check lights are turned on for the following reasons:

  IL - This light is turned on whenever 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.

  PROG - When this light is on, the channel has detected a programming error.

  PROT - This light is turned on whenever the channel attempts to violate a protected area of Main Storage during an I/O operation.

  CHNL DATA and CHNL CTRL - Any time an invalid byte of data is detected in the data register, the CHNL DATA Check light is turned on. If a control byte is contained in the data register at the time the error is detected, the CHNL CTRL Check light for the appropriate channel will be turned on. There are additional checking circuits which can turn on the CHNL CTRL light.

LOWER INDICATOR PANEL (FIGURE 6-3)

CHANNEL NUMBER TWO DISPLAY

- These indicators provide the same function as the Channel Number One display.

INT FACE

This indicator turns on when:

1. A response, from a control unit, is not given to a signalling sequence initiated by the channel.
2. A device indicates that it is busy (after device end has been given) to an initial selection sequence.
3. Either no address response or an address mismatch occurred as a result of an addressing sequence initiated by the channel.
4. A parity error was detected on status or address information sent from a control unit to the channel.

In addition to the two sets of lights for the two selector channels, a group of eighteen indicators is provided to allow the operator to observe the contents of either count register for the appropriate channel. The leftmost nine bits displayed are for channel number one, the rightmost nine for channel number two.

MPX (MULTIPLEX) CHANNEL TAGS

- Nine Multiplex channel tags are displayed as follows:

  OP IN - Indicates that an I/O unit has been selected and is in communication with the channel.

  ADDR IN - Indicates that the address of the currently selected I/O unit is on Bus In.

  STAT IN - Indicates that the address of the currently selected I/O unit has placed status information on Bus In.

  SERV IN - Indicates that the select-

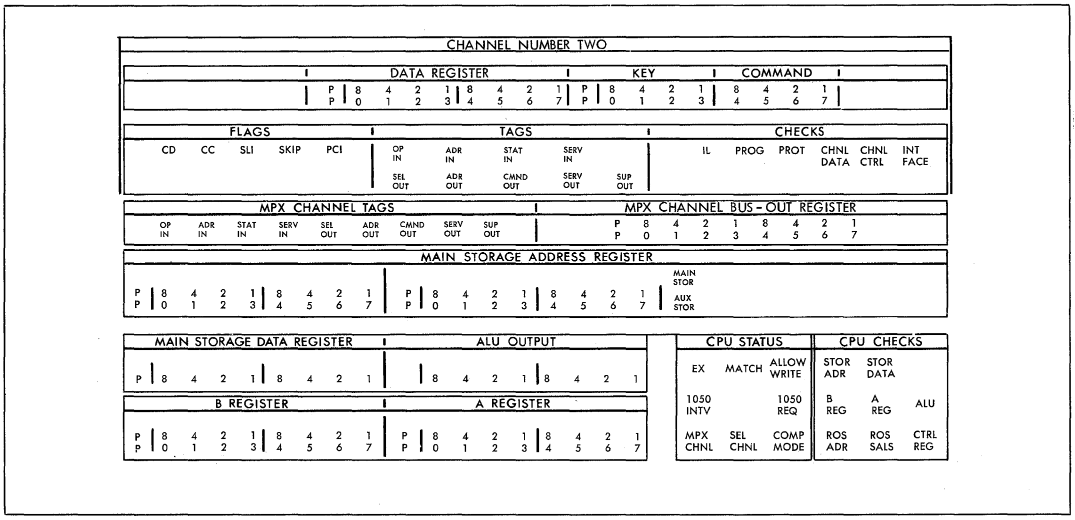

CHANNEL NUMBER TWO
DATA REGISTER
KEY
COMMAND
P P 8 0 4 1 2 2 1 3 8 4 4 5 2 6 1 7
P P 8 0 4 1 2 2 1 3
8 4 4 5 2 6 1 7
FLAGS
CD CC SLI SKIP PCI
TAGS
OP IN ADR IN STAT IN SERV IN
SEL OUT ADR OUT CMND OUT SERV OUT SUP OUT
CHECKS
IL PROG PROT CHNL DATA CHNL CTRL INT FACE
MPX CHANNEL TAGS
OP IN ADR IN STAT IN SERV IN SEL OUT ADR OUT CMND OUT SERV OUT SUP OUT
MPX CHANNEL BUS - OUT REGISTER
P P 8 0 4 1 2 2 1 3 8 4 4 5 2 6 1 7
MAIN STORAGE ADDRESS REGISTER
P P 8 0 4 1 2 2 1 3 8 4 4 5 2 6 1 7
P P 8 0 4 1 2 2 1 3 8 4 4 5 2 6 1 7
MAIN STOR
AUX STOR
MAIN STORAGE DATA REGISTER
P 8 4 2 1 8 4 2 1
ALU OUTPUT
8 4 2 1 8 4 2 1
B REGISTER
P P 8 0 4 1 2 2 1 3 8 4 4 5 2 6 1 7
A REGISTER
P P 8 0 4 1 2 2 1 3 8 4 4 5 2 6 1 7
CPU STATUS
EX MATCH ALLOW WRITE
1050 INTV 1050 REQ
MPX CHNL SEL CHNL COMP MODE
CPU CHECKS
STOR ADR STOR DATA
B REG A REG ALU
ROS ADR ROS SALS CTRL REG

Figure 6-3. Lower Indicator Panel

ed I/O unit wants to transmit or receive data.

SEL OUT - When this light is on, the several I/O units are being polled to determine whether any unit requests service.

ADDR OUT - Indicates that the information on Bus Out is an address.

CMND OUT - Indicates that the information on Bus Out is a command.

SERV OUT - Indicates that the CPU has accepted the information on Bus In or has provided data on Bus Out.

SUPP OUT - This signal used alone or with other tags provides the following functions:

Suppress Status
Suppress Data Transfer
Chained Command Control
Selective Reset

BASIC CPU DISPLAY

- A display of registers, status conditions, and checks in the CPU.
- Those registers with full time indicators are:

MPX Channel Bus-Out Register	9 bits
Main Storage Address Register	18 bits
Main Storage Data Register	9 bits
B Register	9 Bits
A Register	9 Bits
ALU Output (not a register)	9 Bits

Whenever a storage cycle is taken, an address is gated into the Storage Address Register, and one of two indicators will be turned on to indicate whether the access is in Main Storage or Auxiliary Storage (Local Storage or one of the MPX Storages). These lamps are labeled MAIN STOR and AUX STOR, respectively. If an access to Auxiliary Storage is required, the contents of the high order digit of the Main Storage Address Register will determine which part of the Auxiliary Storage is to be used.

During wait state and process stop, the instruction counter is displayed in the B and A registers. The current operation code is not displayed.

CPU STATUS

- These indicators signal the actual operating status of the CPU at any time.

Description

The indicators and their meanings are:

EX: This lamp is turned on at the end of each instruction execution, that is, whenever the micro-instruction "Branch on Interrupt" occurs. In the microinstruction word immediately following the interrupt word, the EX latch is reset. Note that if the system stops at the end of instruction execution (for example, if the Stop button has pressed), the EX lamp remains on. It is extinguished upon restarting the CPU clock.

MATCH: Some modes of operation require the use of an exclusive or match circuit, and the MATCH indicator is turned on whenever the Compare Address in switches A, B, C, and D matches the contents of either the R/W Storage Address Register or the Read/Only Storage Address Register. The position of the Address Compare Switch determines which of these registers is monitored, as well as the system response to a match.

ALLOW WRITE: Whenever the allow write indicator is on,the R/W Storage has completed a read operation, but not the corresponding write operation.

1050 INTV: This light is turned on whenever operator intervention is required at the 1050.

1050 REQ: When this light is on, the CPU has recognized a request for service that was initiated at the 1050.

MPX CHNL: Whenever a multiplex channel share request is recognized by the CPU, the MPX light is turned on. This light is turned off at the completion of the share cycle.

SEL CHNL: This lamp is lighted whenever either selector channel is using the Read-Only Storage.

COMP MODE: Whenever the system is processing a 1400 type program, the comp Mode lamp is turned on. This lamp is turned on at the same time as the W3 lamp.

CPU CHECKS

- Errors detected during CPU operation set a check latch.
- Each check latch turns on a corresponding indicator.
- These indicators are:
  STOR ADDR
  STOR DATA
  B REG
  A REG
  ALU
  ROS ADDR
  ROS SALS
  CTRL REG

Except for the ALU check, the turning on of any of these lamps is an indication of detected parity error in the associated register. In the ALU, a duplicate check is made.

<u>OPERATOR PANEL (FIGURE 6-4)</u>

SYS (SYSTEM) INDICATOR

- This indicator is on whenever the customer or CE usage meter is recording time.
- The System Indicator is located on logic page PA 061.

MAN (MANUAL) INDICATOR

- The Manual Indicator is on whenever the CPU clock is stopped.
- When on, this light indicates that manual controls requiring a stopped clock may be performed.
- The Manual Indicator is located on logic page PA 061.

WAIT INDICATOR

- The Wait Indicator turns on when the CPU is in the wait state.
- An interrupt causes a ROS branch-out of the wait state.
- The Wait Indicator is located on logic page KU 011.

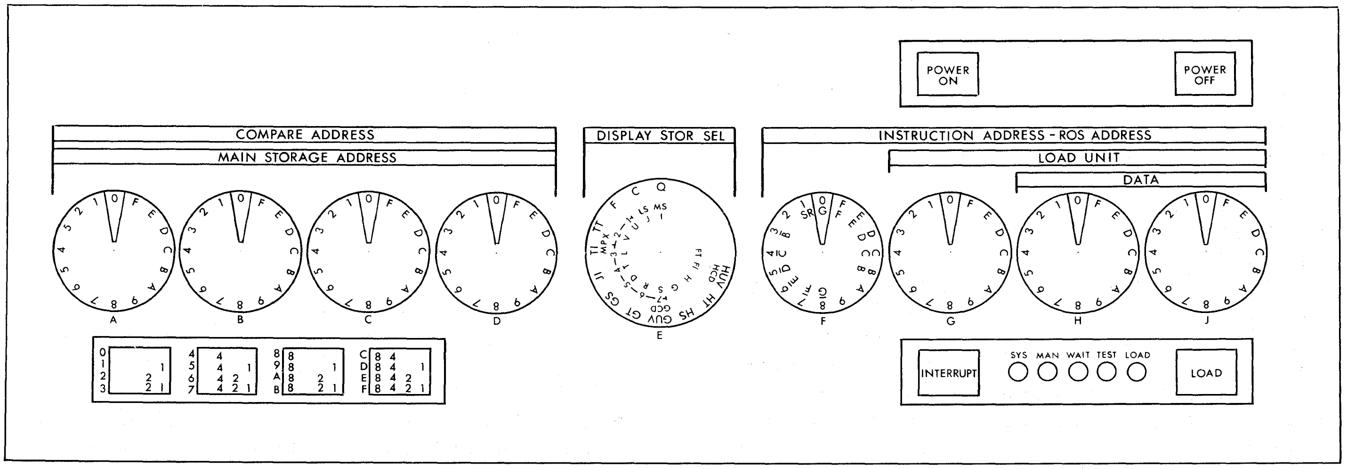


Figure 6-4. Operator Panel

TEST INDICATOR

- The Test Indicator comes on when either the Rate Switch, the Address Compare Switch, the Check Control Switch, or the ROS control switch are not in the Process position.
- When on, the Test light indicates that a test is in progress.
- The Test Indicator is located on logic page PA 061.

LOAD INDICATOR

- The Load Indicator comes on when the Load Key is pressed and turns off at the completion of the initial program load sequence.
- When the Load Indicator turns off, it means that the input device has finished reading in the new program and that the new PSW has been loaded.
- The Load Indicator is located on logic page PA 061.

DATA AND ADDRESS ENTRY SWITCHES

- Allow manual entry of data and addresses to the CPU.
- Each switch provides more than one function.
- Data and addresses are entered with correct parity.

Description

Eight rotary switches are provided for entering manual data or addresses into the system. Each of these is a sixteen-position switch which provides one hexadecimal digit, or four bits plus parity. The first four switches, labeled A, B, C, and D are used to set up the address for manual operations of the core storage, or to set up a compare address. The switches labeled F, G, H, and J are used to set up an Instruction Address, a ROS Address, a Load Unit Address, or manual data.

The following chart shows the switches used to perform the various functions.

Functions	Switches Used
Compare ROAR Stop Address	A,B,C,D
Compare ROAR Restart Address	A,B,C,D
Compare SAR Restart Address	A,B,C,D
Compare SAR Stop Address	A,B,C,D
Manual Display or Store Address	A, B, C, D
Instruction Address	F,G,H,J
ROS Address	F,G,H,J
Load Unit Address (IPL)	G,H,J
Store Data	H,J

DISPLAY STORAGE SELECTION

- Two concentric switches at one location.
- Inner switch has 3 positions; outer switch has 16 positions.
- Define register or storage area to be addressed on a display or store operation.

DESCRIPTION

The Display/Storage Selection switch (switch E) provides a means of selecting a register or one location of main storage or auxiliary storage, in order to display or store information. Selection is made according to the following table.

Outer Switch	Inner Switch Position	Inner Switch Position	Inner Switch Position
Position	1	2	3
1	Q	MS	I
2	C	AS	J
3	F	Spare	U
4	TT	Spare	V
5	TT	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

POWER ON KEY

- Lighted Key
- Initiates power on sequencing of entire system
- Power On key is located on logic page YZ041.

Description

Pressing the Power-On key starts the normal power on sequence. (Information in core storage remains unchanged.) When the system power-on sequence has been completed, the indicator bulb behind the Power On key lights. A system reset function occurs during the power-on sequence.

Power-Off Key

- The Power-Off Key provides a means of removing power from the entire system via a normal power-off sequence.
- Core storage data remains unchanged.
- R-register is written into core if the allow write latch is on.
- The Power-Off key is located on logic page PA 111.

Description

Pressing the Power-Off key on the console initiates the power-off sequence in the CPU and the remainder of the system.

INTERRUPT KEY

- Pressing the Interrupt Key generates a console interrupt which the system will recognize if programmed to do so.
- The Interrupt Key is located on logic page PA 111.

LOAD KEY

- When pressed, the Load Key sets bit 25 in the PSW and causes a system reset function to occur.
- When released, the Load Key initiates an initial program load routine.
- The Load Key is located on logic page PA 101.

Description

The Load Key causes a system reset and sets the machine reset latch. The machine reset latch establishes priority. At the end of the clear-UCW routine, the machine reset latch is reset and a load trap occurs. This load trap starts the load microprogram. This microprogram does not restore any old PSW information as part of its operation. Note: The trap is allowed because the system reset function resets the F-register to all 1's (with the exception of the 1A bit). The F-register contains the external interrupt mask, and all bits on means that all external interrupts are allowed.

CONTROL KEYS (FIGURE 6-5)

SYSTEM RESET KEY

- Resets system (CPU, channels, and control units) to its initial state.
- Sets up ROS branch to 0000.
- Active in all modes of operation.

- Error status information is reset.
- System Reset key is found on logic page PA 101.

Description

Pressing the system reset key causes a system reset function to occur. The system reset function causes all hardware registers to be set to zero with proper parity. All hardware latches are reset with the exception of the machine reset latch, which is set on. The machine reset latch disables all traps until it is reset. Pressing the start key starts the machine at a microprogram that clears all UCW's (Unit Control Words). After the UCW's are cleared, the machine reset latch is reset and traps may occur. When the machine reset latch is reset, the system reset function is over. Note: The ROAR Reset key may be pressed prior to pressing the start key to prevent clearing the UCW's (see ROAR Reset key).

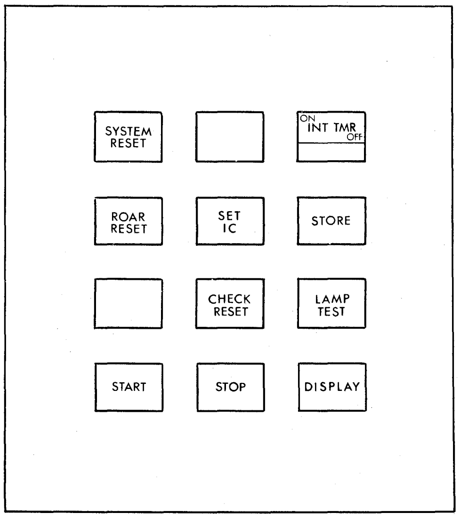


Figure 6-5. Control Keys

ROAR RESET KEY

- Allows manual change of ROAR address by gating contents of switches F, G, H, and J into ROAR.
- The ROAR reset key is effective when the clock is running.
- Can also be used to block the clear-UCW routine after a system reset.
- ROAR Reset switch is located on logic page PA 111.

Description

To manually change the address in WX, the CPU clock must be stopped. Pressing the ROAR reset key turns on the gate switches to WX stacking latch. When the clock is re-started and traps are allowed to take place according to their assigned priority, the gate switches to WX stacking latch, forces an address into WX, which locates the trap to gate the switches to WX.

The ROAR reset key also alters the events that take place after a system reset function. A system reset function resets all registers and latches except the machine reset latch, which is turned on. The machine reset latch forces an address of 0000 to ROAR. If ROAR reset is pressed after a system reset function, but before the CPU clock is started, the machine reset latch is turned off. Then, when the clock is started, the clear UCW's trap is not taken. Other traps of higher priority may take place before the gate switches to WX trap takes place.

SET IC KEY

- Allows manual setting of instruction counter from console.
- Forces ROS branch to 0001, which is the address of the Set IC Trap.
- Set IC key is inoperative if CPU clock is running.
- Set IC key is located on logic page PA 101.

Description

The Set IC key is used to change the core storage address in the instruction counter (registers I and J). Pressing this key forces the address 0001 into WX, and starts the CPU clock. Located at ROS-address 0001 is the routine that sets I and J from switches F, G, H, and J. After entering IJ, the address is also gated into the A- and B-registers so it will be displayed on completion of the Set IC trap, the CPU again enters the wait state.

If a system reset had occurred prior to pressing the Set IC key, the microprogram starts at address 0000. This causes the UCW's to be cleared prior to setting the instruction counter from the switches.

STORE KEY

- Loads the byte specified by the Data Switches (H and J) into the area specified by Display-Store Selection Switch (E).
- Store key is inoperative if the CPU clock is running.
- The Store key is located on logic page PA 111.

Description

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 memory is selected to receive the contents of H and J, the Z-bus is gated to the R-register. In the case where memory is selected by switch E, switches A, B, C, and D provide the memory address, and a manual read cycle and a manual write cycle are taken to place the data byte from switches H and J (now in the R-register) into the desired address.

The CPU clock must be stopped, and the allow write latch must be off for the store operation to take place.

CHECK RESET KEY

- Resets the machine check register as well as several machine check control latches to the no error state.
- Check reset may occur with the CPU clock running or stopped.
- Check reset key is located on logic page PA 111.

Description

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. This means that all machine check logic is reset.

LAMP TEST KEY

- Turns on all console indicator drivers.
- Can be pressed while system is running.
- The lamp test key is located on logic page PA 111.

Description

An additional input is provided on all indicator driver circuits for testing purposes. This input serves an OR function along with the normal driver input. When the lamp test key is pressed, all indicator drivers should turn on to light all console panel lights.

START KEY

- The start key starts the CPU clock.
- System operation depends on what conditions exist when the start key is pressed.
- The start key is located on logic page PA 101.

Description

If the start key is pressed after a normal stop (for example, if the stop key had been pressed), instruction processing continues as if no stop had occurred. Machine status is not affected.

If the start key is pressed after a system reset, the machine reset latch forces the address 0000 into WX. This causes the microprogram to start at the UCW clear routine. Other traps are then allowed to occur in the order of their assigned priority.

STOP KEY

- Stops the CPU clock at the end of the instruction in process.

- All pending interrupts are taken before CPU clock is stopped.

- Machine status is not effected.

- Stop key is located on logic page PA 101.

Description

The CPU proceeds to the end of the instruction in process at the time the stop key is pressed. All pending interrupts are taken before the CPU clock is stopped. Once the clock is stopped, no traps are allowed. 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, these chains are completed before the clock is stopped.

When the CPU has stopped, the storage address of the next instruction is displayed in the B and A registers.

DISPLAY KEY

- Allows a selected byte to be gated to a display register.

- Selected byte may be from any register or from any core storage area.

- CPU clock must be stopped, and allow write latch must be off.

- Display Key located on logic page PA

Description

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 to be displayed in a display register.

An additional display feature occurs when the I-, J-, U-, or V-registers are selected. If either the I- or J- register is selected, both I and•J are transferred to the M- and N-registers so the entire address is displayed by the M- and N-register indicators. Similar- if either the U- or V-registers is displayed via the A-register, then both U and V are transferred to and displayed by the M- and N-registers. 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 to permit changing the address in MN.

To use the display feature, first make sure that the CPU clock is stopped. In addition, if a storage position or the I-, J-, U- or V-registers are to be displayed, the allow write latch must be off. (The allow write latch lights the Allow Write CPU status indicator on the lower console indicator panel.) Next, set the display-store select switch, switch E, to define the register or storage area to be displayed. If a storage area is selected (main storage, or auxiliary storage), or the storage address must be set up in the main storage address switches, 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 gates the selected register and its complementing register to the MN-registers for display. No storage cycle is taken.

If a storage location is being displayed, a storage read cycle is taken followed by a storage write cycle. This retrieves the desired byte from storage and places it into the R-register for display. When a storage location is being displayed, where a program has been halted, it is a good idea to note the contents of the R-register prior to the display operation. Then, the R-register can be restored prior to reentering the program and starting the CPU clock.

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.

- The interval timer switch is located on logic page PA 111.

Description

If the interval timer feature is installed on the 2030, the interval timer switch 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.

MODE CONTROL PANEL (FIGURE 6-6)

RATE SWITCH

- Three position switch with process, instruction step, and single cycle positions.

- Lights test light on operator panel if in instruction step or single cycle position.

- The Rate Switch controls the manner the CPU processes instructions.

- The Rate Switch is located on logic page PA 081.

INSTR (INSTRUCTION) STEP POSITION: When the Rate Switch is in the Instr Step position, one complete instruction, including interrupts, 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.

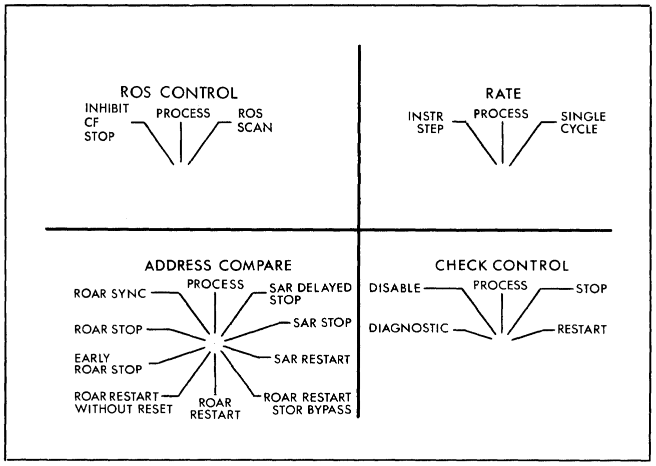


Figure 6-6. Mode Control Panel

If the instruction is an I/O operation, then the I/C operation is completed and interrupts are pending before the CPU clock is stopped.

SINGLE CYCLE POSITION: When the Rate Switch is in the Single Cycle position, the CPU advances by one RCS cycle each time the start key is pressed. Thus, the CPU processes instructions in one microsecond increments, I/O instructions included. This means that I/O data overruns may occur in this mode.

PROCESS POSITION: When the Rate Switch is set at the process position, the CPU clock is allowed to run at the one microsecond rate until some condition causes a stop. This is the position in which customer's will process data.

ADDRESS COMPARE SWITCH

- Determines function to be performed by the address match circuit.

- If the Address Compare switch is at <u>other</u> than the process position, the Test indicator on the operator panel is lighted.

- Switches A, B, C, and D are compared with either RCAR or SAR as defined by the Address Compare switch.

- The Address Compare switch is located on logic page PA 081.

PROCESS POSITION: This is the position in which customers will run their programs. A sync pulse is provided when the address specified in the address switches 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.

ROAR STOP POSITION: With this switch 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 that ROS cycle and the system stops.

EARLY ROAR STOP POSITION: With this switch 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 immediately and the system stops. This function differs from the ROAR Stop function. This reset occurs before ROAR is reset and the address displayed is the address of the ROS word just prior to the 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 are reset to zero before the ROAR is reset to the value in Switches F, G, H, and J. In the ROAR Restart Stor. 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.

SAR RESTART POSITION: When a match occurs in this mode, the CPU is reset and a fixed address is forced into the ROAR. The resulting microprogram loads the contents of Switches F, G, H, and J into the instruction counter (Registers I and J) and then 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 mode, a match causes the CPU clock to stop at the conclusion of the execution of the instruction in which the match occurs. All pending interrupts will be taken before the clock is stopped.

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: A combination of hardware control and a particular microprogram allows the ROS to be scanned sequentially for the purpose of checking. As each ROS word is scanned it is parity checked, but is not used for control. By using the ROAR RESTART operation, the scan can be started at any word and restarted at any word. Setting the ADDRESS COMPARE Switch to ROAR STOP will cause only one scan. If the RATE Switch is set to "INSTR STEP" the clock will stop with each scanned word displayed in the ROS display.

PROCESS: This position allows normal operation of the ROS. This "PROCESS" position is like the equivalent on the other three switches on this panel in that when not in this position the "TEST" indicator will be turned on.

CHECK CONTROL SWITCH

- Determines system action when an error is encountered.

- Causes the Test indicator on the operator panel to light when not in the process position.

- The Check Control Switch is located on logic page PA 081.

DISABLE POSITION: In this position any parity check causes its associated check latch to be set, but otherwise the failure will be ignored. Results 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 with micro-program words.

RESTART POSITION: Upon the 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, a system reset is initiated following which the instruction counter is loaded with the contents of the Switches F, G, H, and J and an I-cycle is started.

2. In the ROAR RESTART position or the ROAR Restart Storage Bypass position, a recycle reset is given which resets hardware 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 micro-program (with or without the op tion of Main Storage).

3. In the ROAR Restart Without Reset position, operation is identical to that in ROAR Restart position except that no reset is initiated.

4. In any other position, operation is like that in SAR Restart position except that no reset is initiated.

PROCESS POSITION: This position is the position in which problem programs are processed. Upon detection of a parity check with the switch in this position, the ROS will automatically initiate what is known as the Malfunction Trap Routine. This routine stores the contents of the Check Register in a fixed location of Main Storage, as well as the Program Status Word, and upon successful completion of this task, originates a Machine Check Interruption.

METER PANEL (FIGURE 6-7)

EMERGENCY POWER OFF SWITCH

- Causes power to turn off beyond the entry terminal of every unit on the system.

- All power in the CPU and all I/O devices is dropped immediately.

- The contents of core storage is not guaranteed.

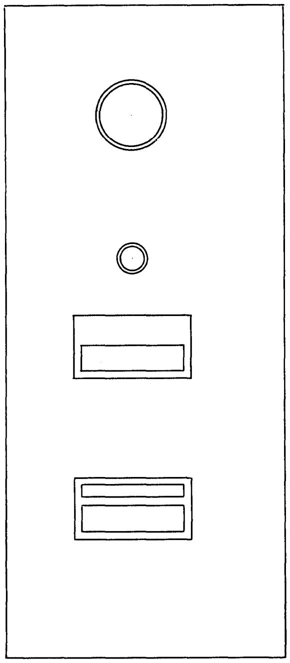


Figure 6-7. Meter Panel

METERING SWITCH

- Enables or disables the timing meters.

- Operated by a removable key.

- Two positions are:

  1. Normal -- Enable process meter, disable CE meter.

  2. CE -- Disable process meter, enable CE meter.

PROCESS METER AND CE METER

- Measure operating time when the CPU clock is running, the CPU is not in the Wait state, or no external interrupts are pending.

- The position of the key switch determines which meter is recording time.

ANSWERS -- SYSTEM/360 INSTRUCTIONS

1. Halfwords

2. Two, Specification

3. Op Code

4.

a.	1A	00011010	1
b.	56	01010110	2
c.	9C	10011100	2
d.	FD	11111101	3

5. Displacement, Base Address

6. Index

7. Does not

8. Base Registers

9. 4095

10. 1 to 15

11. The contents of Register 0 are ignored and a value of zero is used for the Base or Index factor.

12.

```
  a.  RR   ┌───────┬─────┐
           |Op Code|R1 R2|
           └───────┴─────┘

  b.  RX   ┌───────┬─────┬──┬──┐
           |Op Code|R1 X2|B2|D2|
           └───────┴─────┴──┴──┘

  c.  RS   ┌───────┬─────┬──┬──┐
           |Op Code|R1 R3|B2|D2|
           └───────┴─────┴──┴──┘

  d.  SI   ┌─────────────┬──┬──┐
           |Op Code  I2  |B1|D1|
           └─────────────┴──┴──┘

  e.  SS   ┌───────┬──┬──┬──┬──┬──┬──┐
           |Op Code|L1|L2|B1|D1|B2|D2|
           └───────┴──┴──┴──┴──┴──┴──┘
```

13. 1st

14. 7

15. The Instruction, Byte

16. RX

17. SS

18. The number of Bytes in the data field is one greater than the number in the length code.

ANSWERS -- INSTRUCTION SEQUENCING AND BRANCHING

1. Program Status Word

2. 64

3. 40 - 63

4. 2, 4, 6

5. 0-1; Op Code

6. "Current"

7. Some type of internal register or storage area

8. Instruction address, address (location)

9. 34 - 35

10. 4

11. Some

12. Condition Code

    00
    01
    10
    11

13. <zero

14. Equal or Low

15. Always

16. Bits 8 - 31 of general register 5

17. The effective address generated by adding the contents of register 4 and register 7 and a displacement factor of 0.

ANSWERS -- SYSTEM/360 AND INTERRUPTS

1. a. External
   b. Supervisor Call
   c. Program
   d. Machine
   e. I/O

2. a. Current PSW is the double-being used by CPU to control the execution of a sequence of instructions. There is only one current PSW.
   b. Old PSW is the doubleword placed in main storage as a result of an interrupt. There are five locations reserved in main storage, one for each class of interrupt.
   c. New PSW is the doubleword fetched from main storage as a result of an interrupt. It then becomes the "current" PSW. Bits 40-63 of this doubleword would switch the machine to a new sequence of instructions.

3. 0063; there are 5 old PSW'S of 8 bytes each. Each main storage address refers to an individual byte.

4. 0088

5. 0128; it uses the area of main storage just above the area for new PSW's.

6.
```
┌───────┬───┬────┬──────────────┐
|System |   |AMWP|Interruption  |
|Mask   |   |    |Code          |
|       |   |    |              |
└───────┴───┴────┴──────────────┘
 0      7  12  15 16            31

┌───┬─────┬──────┬────────────┐
| I |  C  |Prog. |Instruction |
| L |  C  |Mask  |Address     |
└───┴─────┴──────┴────────────┘
  32              40          63
```

7. Supervisor call interrupts and those program interrupts not caused by a.) fixed point overflow, b.) decimal overflow, c.) exponent underflow, or, d.) significant. The last two deal with floating point arithmetic.

8. 0

9. Only by an I/O or external interrupt.

10. Any interrupt

11. The address of the channel and I/O unit

12. The 8 bits in the R1 and R2 field of the supervisor call instruction.

13.
```
 32   33
┌──────┐
|1    0|
└──────┘
  ↑
 ILC
```

14. Privileged, Program Interrupt

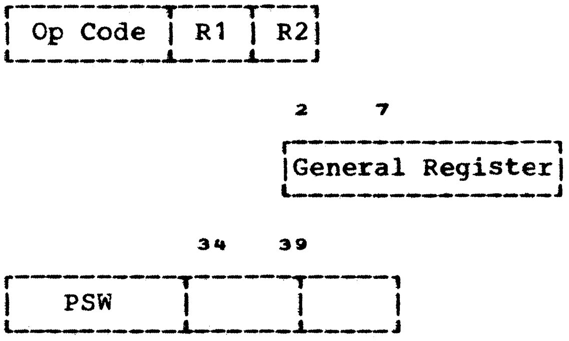
15. By issuing a "Load PSW" instruction addressing the doubleword at location 0120 (the old PSW for an I/O interrupt).

16. a, c, e

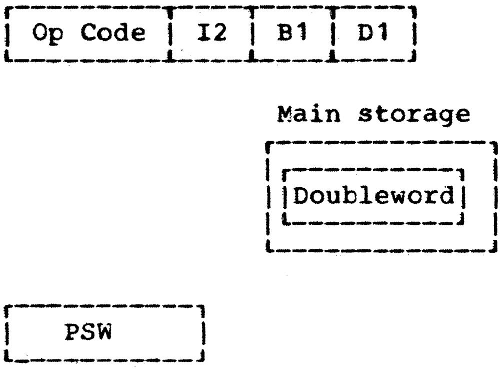
17. A byte from main storage

```
┌────┬──┬──┬──┐
|Op  |  |  |  |
|Code|12|B1|D1|
└────┴──┴──┴──┘

                 ┌────────────┐
                 |Main storage|
                 └────────────┘

 0   7
┌────┬─────┐
|    |     |
└────┴─────┘
    PSW
```

18. Condition code and program mask with bits 2-7 of the register addressed by the R1 field.



19. d. Although the "Load PSW" instruction is used to return to an interrupted program by loading the old PSW, this old PSW has to be addressed from main storage just like any other doubleword.



ANSWERS -- STORAGE PROTECTION

1. Storage Keys

2. Protection Key

3. A privileged instruction called "Set Storage Key"

4. (1) A privileged instruction called "Load PSW"

   (2) An Interrupt

5. a. Modify-type main storage cycles

6. a. All blocks; storage protection does not apply to fetch-type cycles

   b. Blocks A and C

7. a. All blocks

   b. All blocks

   Note: As long as either key (storage or protection) is zero, the store cycle is allowed and not interrupt occurs.

8. d. The data will not be stored and the program will be interrupted

9. Storage, C, 4

10.

Reg 2	Reg 3
0 0 0 0 0 F 7 0	0 0 0 0 0 F 0 0

11. c. Program interrupt with a protection violation indicated in the interruption code of the old PSW.

ANSWERS -- BASIC COMPONENTS

1. negative

2. minus

3. 2

4. opposite

5. time, output

6. time, start, start

7. Flip Flop, Polarity Hold

8. false

9. false

ANSWERS -- CPU CLOCK

1. 500

2. 5

3. 4, 1, 2 and 3

4. clock reset

5. 250

6. P

7. Clock Start Control, E-cycle Stop Sample, and Clock Start latches

8. After T4 time

9. P2 and P3

10. False

ANSWERS -- ARITHMETIC OPERATION

1. The A and the B-registers

2. 8

3. Micro-programming

4. Add, and, or, exclusive or

5. High, low, straight, cross, cross high, cross low.

6. High, low, straight

7. Excess six, decimal add

8. No carry out of the high order position of a group.

9. False

10. 3, S

11. Carry latch, carry-in latch

ANSWERS -- REGISTERS

1. Storage

2. ROS

3. Polarity hold, AOI

4. True

5. False

ANSWERS -- CORE STORAGE

1. Three, an X-line, a Y-line, and an Inhibit-Sense Line.

2. X, Y, MN-Regs.

3. The base of the decode switch is conditioned by the N-register 4, 5, 6, and 7 bits. The emitter of the decode switch is conditioned by the N-register 2, and 3 bits.

4. The base of the decode switch is conditioned by the M-register 6 and 7 bits plus the N-register 0 and 1 bits. The emitter of the decode switch is conditioned by the M-register 3, 4, and 5 bits.

5. Coincidence of current flow in an X-band and Y-line, the flipping of a core, strobing of the pulse.

6. 2 sets of X-lines, 1 set of Y-lines 3 phase reversal planes.

7. R-reg.

8. When a core flips from the logical 1 state to the logical 0 state.

9. Strobe pulse

10. Not flow.

11. To reverse the direction of current flow in the Y-lines of corresponding planes in alternate 8K arrays. (i.e., if Y-current is flowing from top to bottom is left to through the 1-bit plane of the first 8K array, it will be flowing from bottom to top through the 1-bit plane of the second 8K array.

12. Of the lower or upper 32K clock.

13. M 0 bit, on

14. 16 gate decode switches

15. 16 gate decode switches

16. Emitter and base conditioned to conduct.

17. To cause current to flow opposite the X-lines to prevent the core from flipping if the bit is not desired in storage.

18. The maintenance manual.

19. X-lines, Y-lines.

ANSWERS -- ROS CONCEPTS

1. line driver

2. SAL

3. RCS

4. one

5. serially

6. parallel

7. 60

8. 12

9. 8

10. 42

ANSWERS -- MICRO PROGRAM

1. Micro-program

2. ROS word

3. control fields

4. function control, storage control, branching and address.

5. 8

6. function control

7. W, X

8. X6, X7

9. CK

10. storage

ANSWERS -- ROS WCRD FORMAT

1. 8

2. E

3. arithmetic

4. lower

5. binary, decimal

6. E binary or decimal mode must be specified.

7. is set into

8. V

9. one microsecond

10. Many

ANSWERS -- BLOCK EXAMPLE

1. 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 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

   Position 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 E 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). Cn 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+->DC).

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.

ADDRESS 04B4: The data just read from address XXXX is lost 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 insert.

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.

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).

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).

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 (1S6) 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

carryout S3=1. 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 0000 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.

ANSWER -- CONTROL FIELD MNEMONICS

1. STORE WRAP

2. 0,X7

3. either -- The operation is specified on the E line of the CLD box.

4. 1 1 1

5. left intact

6. No, the CB control field only specifies R, L, D, and K.

7. 0001 - Force a 1, 1 branch

8. 0->IPL

9. 0010 - The zeros represent capacitor plates that are punched out.

10. G0 and G1 are interrogated. Local storage is specified if these positions are both zero.

11. This coding allows the R-register to be set from core if the previous ROS word gave a read call. This is the CM coding used for the "compute" cycle.

12. To one

13. No, the S0 and S1 mnemonics are both in the CH field.

14. 5 PK, PA, PS, PC, PN

15. Yes. When the mnemonic K->W or CA->W is used.

ANSWERS -- ROS ADDRESSING

1. Two

2. 8K unit 8064 ROS words

3. T, driver decode

4. 15

5. 48

6. 120 sense amps, 60 SALs.

7. A small pluggable card which is used to satisfy the inputs to the 3 driver decodes for each ROS board.

8. No, the SALs are shared by each 4K unit. Additional sense amplifiers are needed.

9. W3

10. X7

ANSWERS -- ROAR CONTROLS

1. By means of the AA bit. If the AA is a one then W3 is set to one.

2. F, G, H, and J

3. CN

4. 0004 (HEX)

5. False. The lights are actually tied to an indicating ROAR.

6. False. The ALU check is further controlled by the CPU Check switch.

ANSWERS -- ROS TIMINGS

1. e, d, b, a, c

2. The SALs are set by the ROS word as specified by the address in ROAR. The control register latches contain the data of the previous ROS word.

3. The first cycle of a micro program break-in is called a dead cycle because no control register latches are set.

4. The branch conditions such as Z = C, AC, and 1BC which depend on a previous ROS word would be lost due to a program break in.

5. T4

ANSWERS -- MACHINE CHECK HANDLING

1. To insure that a second error does not occur if the failing register is one which is used in the machine check microprogram.

2. By an ALU check

3. Priority latch off; First machine check latch on; Suppress malfunction trap latch off; Switches not used to set W and X.

4. If H5 is set the error occurred during a selector ROS request. If H6 is set the error occurred during a multiplexor share request.

5. A microprogram step that specifies the D-register as the destination of the arithmetic statement.

6. If H1 can be reset by the MC microprogram, further errors are considered first errors.

ANSWERS -- FORCED MICROPROGRAM ENTRIES

1. 0010 (HEX)

2. Machine reset

3. Stacking

4. By a microprogram step that specifies the H-register as the destination of the arithmetic statement.

ANSWERS -- COMPATIBILITY FEATURE

1. 25,824

2. 00000101

3. Wrap detection

4. So that they may be more easily tested by the microprogram.

5. The storage bias constant is a number equal to the 2030 storage size minus the 1400 system storage size, in HEX.

6. The 2030 will leave compatibility mode, and possibly destroy a control byte in local storage.

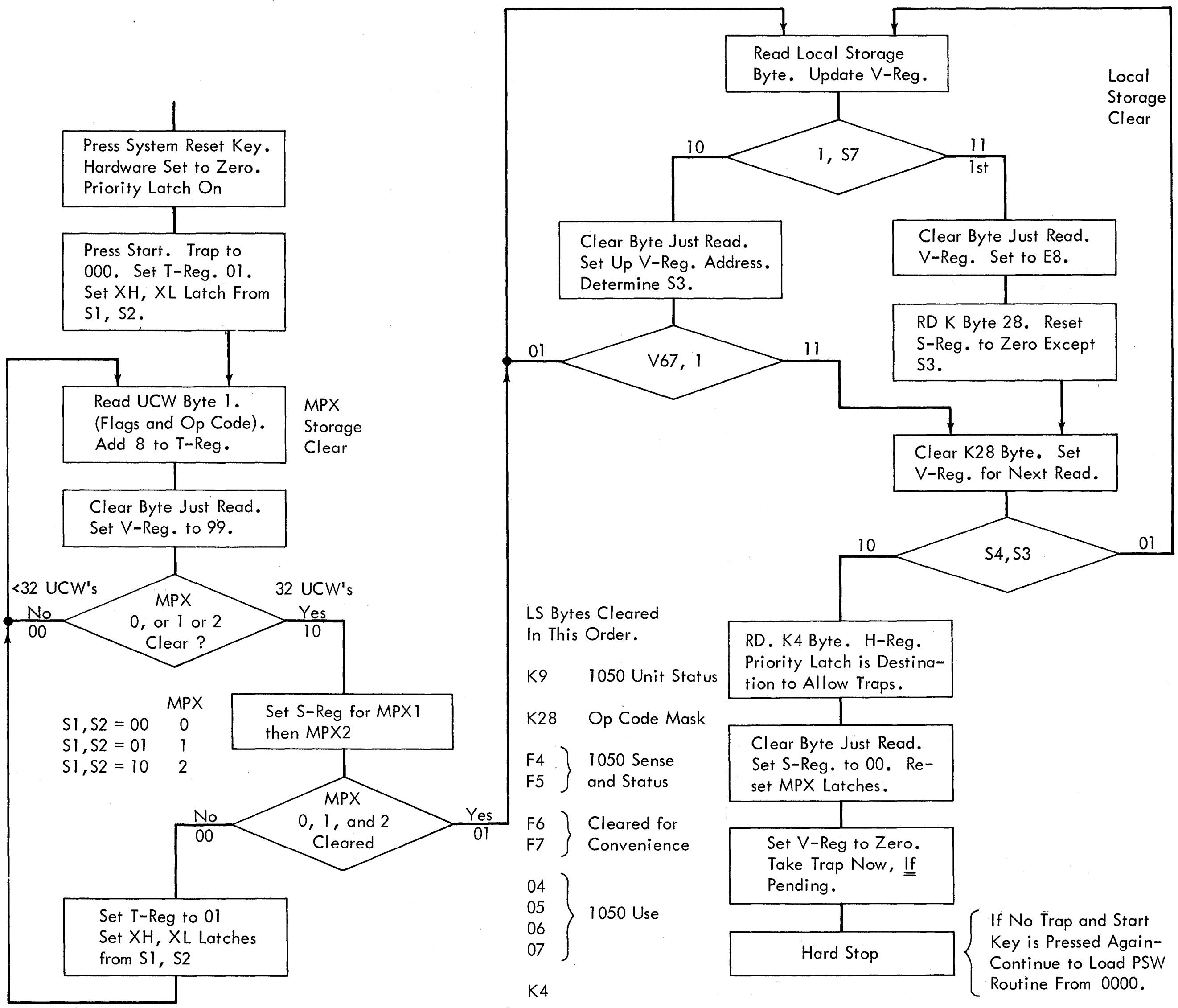
7. The 2030 comes to a program halt with a coded byte in the R-register showing an invalid stacker select command because a stacker select command must be issued within 6 milliseconds after a read command.

8. If the Programmed Mode Switch feature is present on the 2030, the compatibility features normally found in local storage will be relocated to MPX 2 storage.

9. The local storage will read out a 01000000, which is a blank.

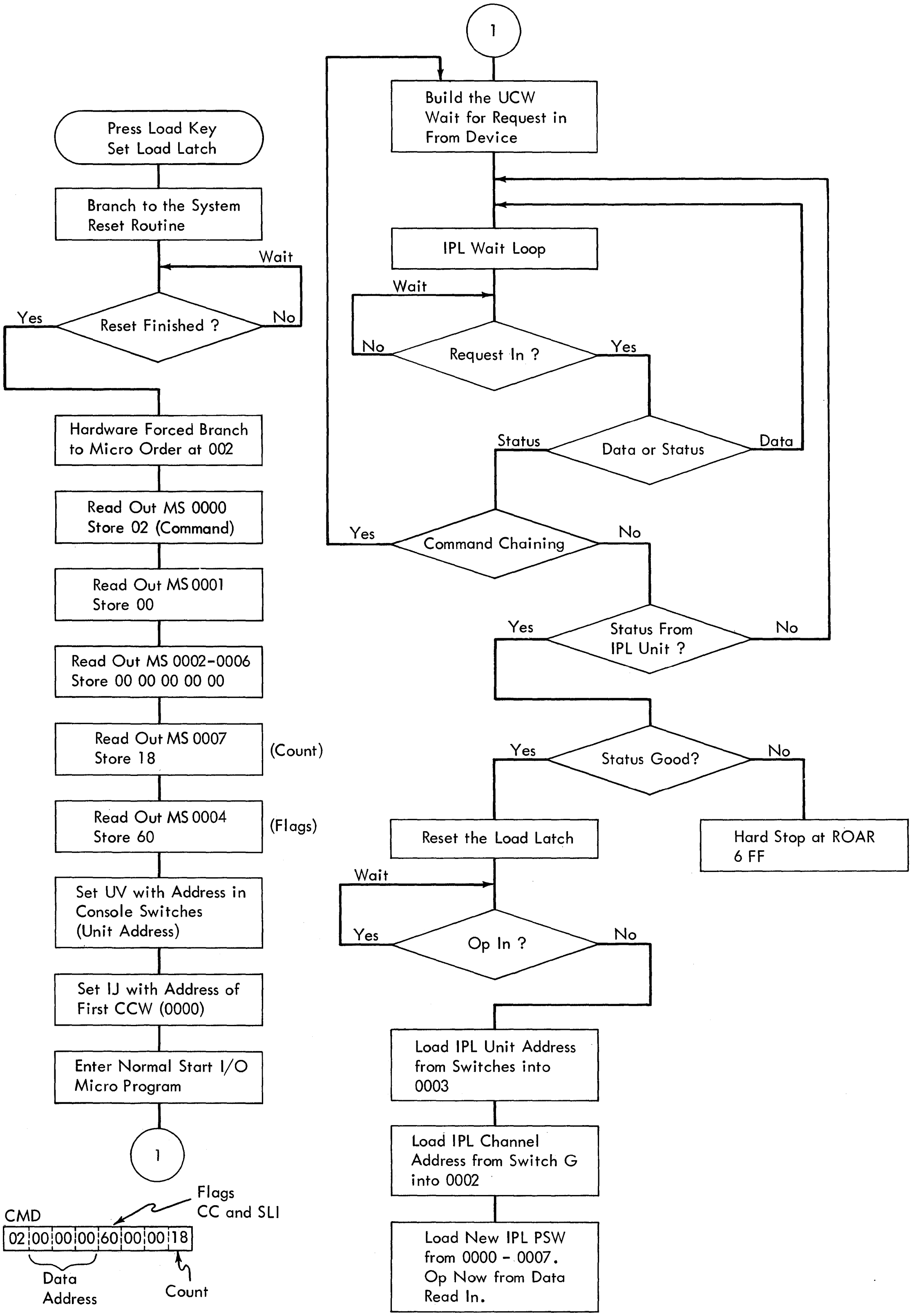
10. The internal code used for the compatibility mode is the EBCDI character representation.

Figures 6-8, 6-9, 6-10, 6-11, and 6-12 are provided to help you understand micro program concepts.



Press System Reset Key. Hardware Set to Zero. Priority Latch On
Press Start. Trap to 000. Set T-Reg. 01. Set XH, XL Latch From S1, S2.
Read UCW Byte 1. (Flags and Op Code). Add 8 to T-Reg.
MPX Storage Clear
Clear Byte Just Read. Set V-Reg. to 99.
<32 UCW's
No
00
MPX 0, or 1 or 2 Clear ?
32 UCW's
Yes
10
MPX
S1,S2 = 00 0
S1,S2 = 01 1
S1,S2 = 10 2
Set S-Reg for MPX1 then MPX2
No
00
MPX 0, 1, and 2 Cleared
Yes
01
Set T-Reg to 01 Set XH, XL Latches from S1, S2
Read Local Storage Byte. Update V-Reg.
Local Storage Clear
10
1, S7
11
1st
Clear Byte Just Read. Set Up V-Reg. Address. Determine S3.
Clear Byte Just Read. V-Reg. Set to E8.
RD K Byte 28. Reset S-Reg. to Zero Except S3.
01
V67, 1
11
Clear K28 Byte. Set V-Reg. for Next Read.
10
S4,S3
01
LS Bytes Cleared In This Order.
K9 1050 Unit Status
K28 Op Code Mask
F4 F5 1050 Sense and Status
F6 F7 Cleared for Convenience
04 05 06 07 1050 Use
K4
RD. K4 Byte. H-Reg. Priority Latch is Destination to Allow Traps.
Clear Byte Just Read. Set S-Reg. to 00. Reset MPX Latches.
Set V-Reg to Zero. Take Trap Now, If Pending.
Hard Stop
If No Trap and Start Key is Pressed Again-Continue to Load PSW Routine From 0000.

Figure 6-8. System Reset



Press Load Key
Set Load Latch
Branch to the System Reset Routine
Wait
Yes
Reset Finished ?
No
Hardware Forced Branch to Micro Order at 002
Read Out MS 0000 Store 02 (Command)
Read Out MS 0001 Store 00
Read Out MS 0002-0006 Store 00 00 00 00 00
Read Out MS 0007 Store 18
(Count)
Read Out MS 0004 Store 60
(Flags)
Set UV with Address in Console Switches (Unit Address)
Set IJ with Address of First CCW (0000)
Enter Normal Start I/O Micro Program
1
Flags
CC and SLI
CMD
02 00 00 00 60 00 00 18
Data Address
Count
1
Build the UCW Wait for Request in From Device
IPL Wait Loop
Wait
No
Request In ?
Yes
Status
Data or Status
Data
Yes
Command Chaining
No
Yes
Status From IPL Unit ?
No
Yes
Status Good?
No
Reset the Load Latch
Hard Stop at ROAR 6 FF
Wait
Yes
Op In ?
No
Load IPL Unit Address from Switches into 0003
Load IPL Channel Address from Switch G into 0002
Load New IPL PSW from 0000 - 0007. Op Now from Data Read In.

Figure 6-9. IPL

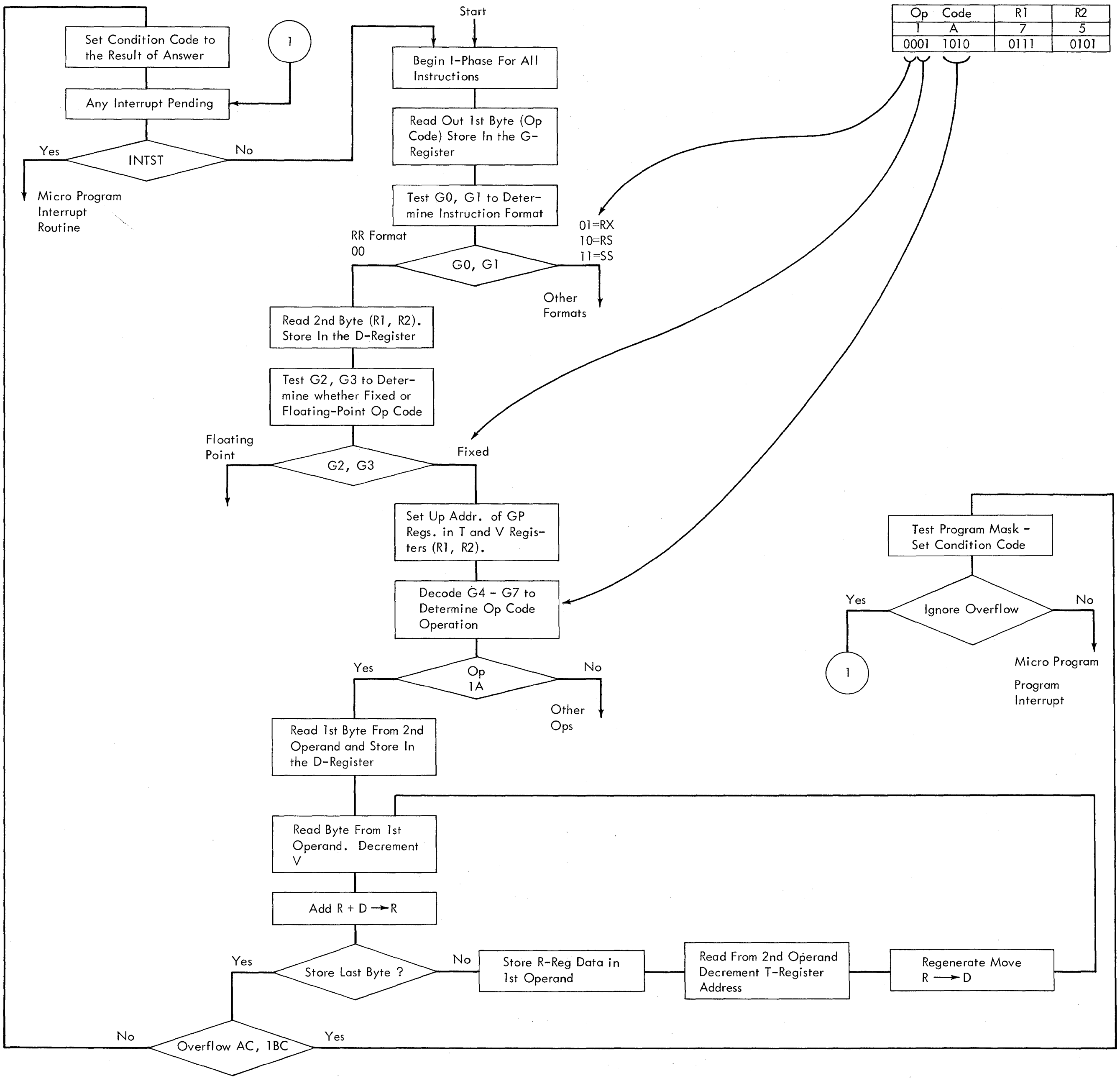
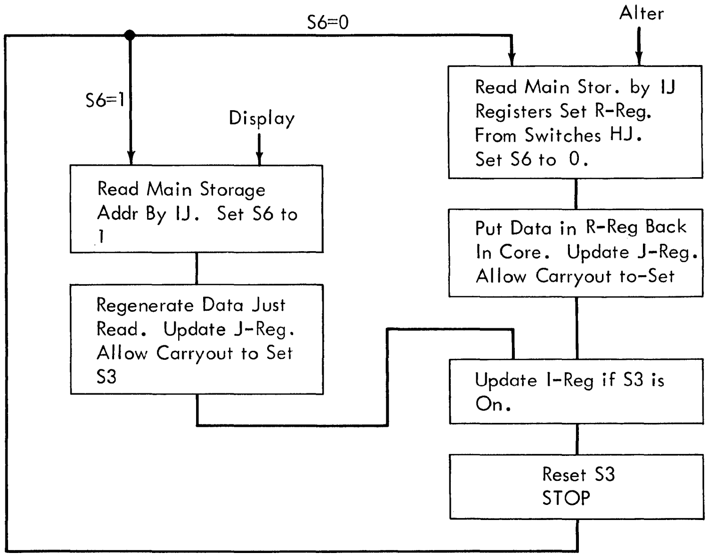

Start
Op Code
R1
R2
1
A
7
5
0001
1010
0111
0101
Set Condition Code to the Result of Answer
1
Any Interrupt Pending
Yes
INTST
No
Micro Program Interrupt Routine
Begin I-Phase For All Instructions
Read Out 1st Byte (Op Code) Store In the G-Register
Test G0, G1 to Determine Instruction Format
RR Format 00
G0, G1
01=RX
10=RS
11=SS
Other Formats
Read 2nd Byte (R1, R2). Store In the D-Register
Test G2, G3 to Determine whether Fixed or Floating-Point Op Code
Floating Point
G2, G3
Fixed
Set Up Addr. of GP Regs. in T and V Registers (R1, R2).
Decode G4 - G7 to Determine Op Code Operation
Yes
Op 1A
No
Other Ops
Read 1st Byte From 2nd Operand and Store In the D-Register
Read Byte From 1st Operand. Decrement V
Add R + D → R
Yes
Store Last Byte ?
No
Store R-Reg Data in 1st Operand
Read From 2nd Operand Decrement T-Register Address
Regenerate Move R → D
No
Overflow AC, 1BC
Yes
Test Program Mask - Set Condition Code
Yes
Ignore Overflow
No
1
Micro Program
Program Interrupt

Figure 6-10. Fixed Point

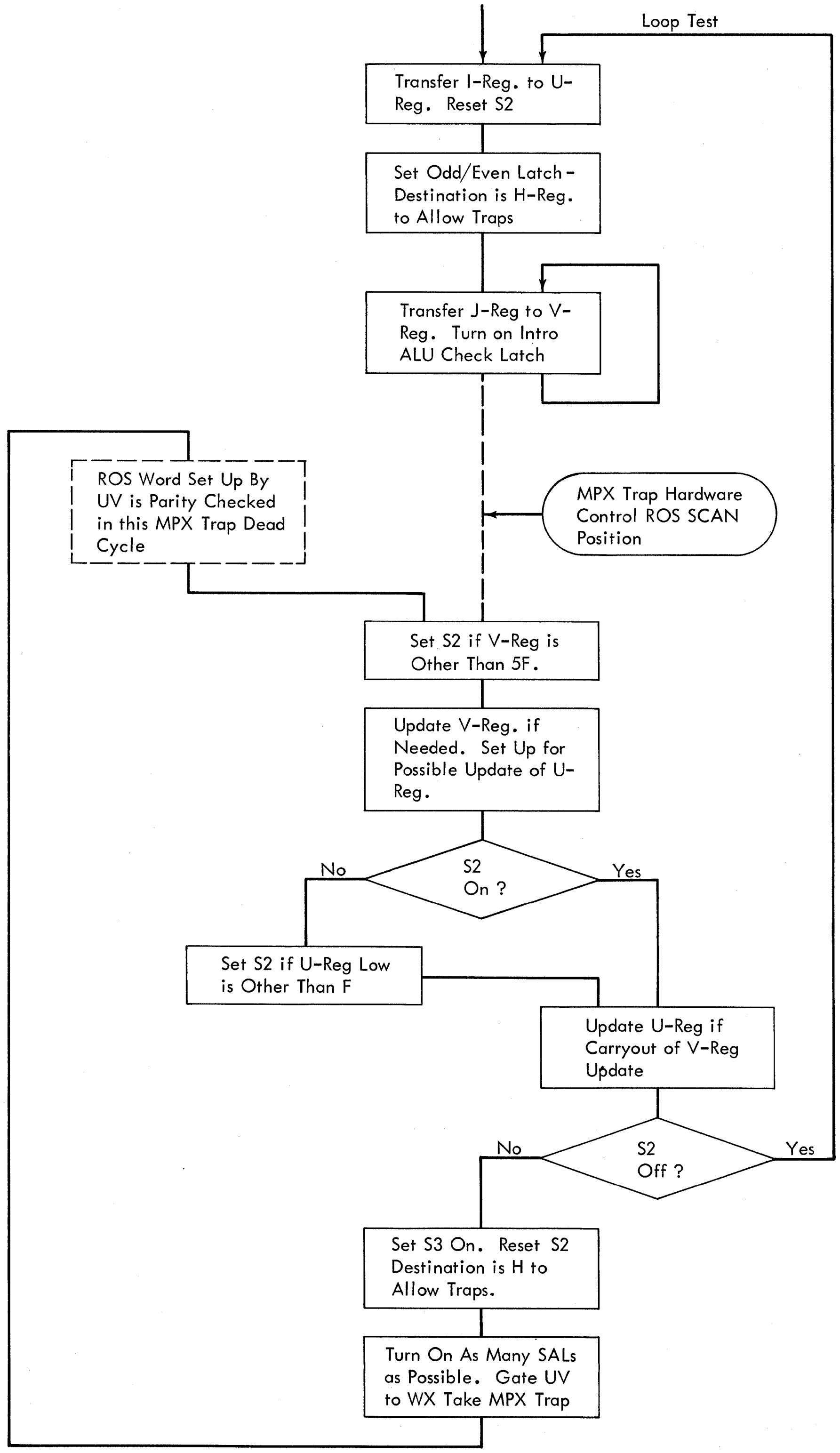

S6=0
Alter
S6=1
Display
Read Main Stor. by IJ Registers Set R-Reg. From Switches HJ. Set S6 to 0.
Read Main Storage Addr By IJ. Set S6 to 1
Put Data in R-Reg Back In Core. Update J-Reg. Allow Carryout to-Set
Regenerate Data Just Read. Update J-Reg. Allow Carryout to Set S3
Update I-Reg if S3 is On.
Reset S3
STOP

MANUAL TRAPS
to DISPLAY
or ALTER

PROCEDURE

1. System Reset
2. Store Desired Address in IJ (Set IC) Registers
3. ROAR Reset to ROS Address of Display or Alter
4. Press Start
5. To Display Successive Characters Depress Start Key.
6. To Change New Data (STORE). Set Switches H and J to Desired Character and Press START.

Figure 6-11. Manual Traps to Display or Alter


Loop Test
Transfer I-Reg. to U-Reg. Reset S2
Set Odd/Even Latch - Destination is H-Reg. to Allow Traps
Transfer J-Reg to V-Reg. Turn on Intro ALU Check Latch
ROS Word Set Up By UV is Parity Checked in this MPX Trap Dead Cycle
MPX Trap Hardware Control ROS SCAN Position
Set S2 if V-Reg is Other Than 5F.
Update V-Reg. if Needed. Set Up for Possible Update of U-Reg.
No
S2 On ?
Yes
Set S2 if U-Reg Low is Other Than F
Update U-Reg if Carryout of V-Reg Update
No
S2 Off ?
Yes
Set S3 On. Reset S2 Destination is H to Allow Traps.
Turn On As Many SALs as Possible. Gate UV to WX Take MPX Trap

PROCEDURE

1. System Reset
2. Set IJ to Address of First ROS Word To Be Scanned.
3. Rate Switch to PROCESS
4. Check Switch to STOP
5. ROAR RESET to 1st ROS Address of This Test

Note 1: If Rate Switch is Set to Instr Step, the System Stops With Each Scanned Word in the SALs.

Note 2: If Check Switch is On Disable, Scan Through Errors.

Figure 6-12. Ros Scan

READER'S SURVEY FORM

IBM 2030 Processing Unit Field Engineering Manual of Instruction
Form 225-3360-0

• Is the material:	*Yes*	*No*
Easy to read?	□	□
Well organized?	□	□
Fully covered?	□	□
Clearly explained?	□	□
Well illustrated?	□	□

• How did you use this publication?

As an introduction to the subject □
For additional knowledge of the subject □
Studying machine operation □
During an actual service call □

• Which of the following terms best describes your job?

Customer Engineer	□	Field Stock Personnel	□
Instructor	□	Manufacturing Engineering	□
Sales Representative	□	Manufacturing	□
Systems Engineer	□	Development Engineering	□
Trainee	□	Other ____________________	

• How often do you use this publication?

Often □ Occasionally □ Seldom □

• Comments:

Your comments help us produce better publications. Please include specific page references if appropriate. You may include your name and address if you wish.

Space is available on the other side of this page for additional comments.
Thank you for your cooperation.

Staple

Fold Fold

FIRST CLASS
PERMIT NO. 170
ENDICOTT, N. Y.

BUSINESS REPLY MAIL

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY . . .

IBM Corporation
P. O. Box 6
Endicott, N. Y. 13764

Attention: Product Publications, Dept. 171

Fold Fold

IBM®
International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N. Y. 10601

Additional Comments:

Cut Along Line

IBM FE Supplement

System/Unit IBM 2030 Processing Unit

Re: Form No. 225-3360

This Supplement No. S25-0030

Date: December 7, 1965

Previous Supplement Nos. None

This supplement, Form S25-0030, pertains to the 1.5 microsecond core storage unit for the IBM 2030 Processing Unit. This material should be inserted after the 2.0 microsecond storage unit write-up in the IBM 2030 Processing Unit FE Manual of Instruction, Form 225-3360.

IBM Corp., Product Publications Dept., Endicott, N. Y. 13764

PRINTED IN U.S.A.

FE
System
Maintenance
Library

System

cut here

225-3360-0

IBM

International Business Machines Corporation
Field Engineering Division
112 East Post Road, White Plains, N. Y. 10601