IBM Customer Engineering

Manual of Instruction

Preliminary Edition

IBM 7040-7044 Data Processing Systems
Central Processing Unit

R23-2651

IBM®

Customer Engineering
Manual of Instruction

Preliminary Edition

IBM 7040-7044 Data Processing Systems
Central Processing Unit

R23-2651

PREFACE

This manual contains instructional information on the 7106 and 7107 Central Processing Unit (CPU), covering such items as CPU registers, timing and basic cycles, CPU instructions including floating point, memory protection, trapping, and the operator's console. Channel A, power supply, and the memory units are covered in separate manuals.

It is assumed that the student is familiar with the binary and octal numbering systems, and is proficient in converting numbers between the decimal, binary, and octal systems. The numbering systems and number conversions are described in the IBM 7040 and 7044 Student Text manual, form C22-6732.

The 7040-7044 Data Processing System is a medium-size scientific machine. High flexibility is realized in this system through the use of an integrated central processing unit (CPU) and core storage combined with a highly elastic input-output (I-O) capability. Two basic system concepts are employed:

1. A basic system using a nonoverlapped* I-O channel to which buffered card equipment and magnetic tape may be attached.
2. An overlapped* system which can accommodate up to four overlapped I-O channels to which tape adapters, direct data, and corporate interface devices may be attached.

Available with the basic instruction set are the following options:

1. An extended performance instruction group which includes indexing, logical, and character handling operations.
2. Single-precision floating-point arithmetic class.
3. A double-precision floating-point arithmetic class.
4. A memory protection arrangement.
5. An interval timer option.

DATA PROCESSING

Data processing is the execution of sequential operations on facts to realize a desired result. Two elements constitute the foundation of all data processing: procedures to follow and devices to perform the procedures. Procedures are constant, and devices are variable; e.g., arithmetic is the same whether performed in Europe, Asia, Africa, or America. Symbols may change, but the concepts remain the same. Devices, however, are varied: the pencil and paper of the student, the slide rule of an engineer, the calculator of a clerk, the machine of the business man. These, theoretically, have equal potential. The difference is time, for as the slide rule and calculator are faster than pencil and paper, the machine is faster than either the slide rule or the calculator.

There are many types of data processing systems, varying in size and complexity. However, regardless of the data to be processed or the devices to be used, four basic requirements must be satisfied:

1. A means of entering source data and procedures in the system.
2. A means of storing the source data and procedures until they are needed.
3. A means of processing the source data.
4. A means of converting the processing result into a form useful for human handling.

Input devices sense coded data recorded on a prescribed medium. The prescribed medium can be a card, a paper tape, or a magnetic tape. The code can be a configuration of punched holes or magnetic spots. Paper documents containing characters printed in magnetic ink may also be used.

Storage devices hold source data to be processed and the series of operations used to direct processing. In early data processing machines, storage devices consisted of interchangeable panels, relays, cards, or paper tapes. Instructions and data had to be wired or read into the machine in small batches. Processing was therefore limited in both volume and speed. A substitute for the early storage devices is the magnetic core. This core is a small ring of ferro-magnetic material easily magnetized in either of two polarities to represent a digit or symbol. A related group of digits or symbols represents a word; therefore, a related group of ferromagnetic cores can store a word.

Another type of storage device is the magnetic drum. A magnetic drum is a steel cyclinder enclosed in a copper sleeve, which is plated with a cobalt and nickel alloy to form the actual storage medium. In this device, a magnetized spot represents a digit, and a group of magnetized spots represents a word. Although the time necessary to place information on a drum and to take information off a drum exceeds that for ferromagnetic cores, the magnetic drum greatly surpasses both the speed and the capacity of the early semimanual devices.

A storage device similar to the magnetic drum is the magnetic disk. The magnetic disk is a thin metal disk, coated on both sides with a ferrous oxide recording material. Information is placed on a disk as magnetized spots located in concentric tracks. The time required to enter data on and to take data off a disk exceeds that for the magnetic drum.

The key ingredient of a data processing system is the processing device, the nerve center of the entire system. It has two basic areas: the arithmetic-logical area and the control area. The arithmetic-logical area performs arithmetic, number comparisons, shifting, etc.; the control area directs and coordinates the entire system, including the input, storage, and output devices.

Output devices record the results of processing operations on cards or paper and magnetic tapes. Printed information is also available from output

* overlapped and non-overlapped operations refer to simultaneous and non-simultaneous I-O and CPU operations, respectively.

devices. In addition, the product of an output device can take the form of electrical signals for transmission to other data processing centers.

ELEMENT FUNCTIONS

Each element of a data processing system has definite functions. These functions are defined in the following paragraphs.

Storage

Storage is the space provided in a data processing machine for the safekeeping of information. Three types of storage devices are used: core storage, magnetic drum storage, and magnetic disk storage. In each type, information can be placed in, held in, or removed, as needed. The information involved can be:

1. Instructions to direct the processing sequence.
2. Data to be processed or to reflect the results of processing.
3. Reference data necessary for processing (tables, arithmetic constants, etc.).

Storage is generally categorized as either main or auxiliary. Main storage accepts data from the input units, supplies instructions to the CPU, exchanges data with the CPU, and furnishes data to an output unit. All instructions to direct processing and all data to be processed pass through main storage to the CPU. Magnetic core storage generally serves as the main storage device.

Auxiliary storage augments the capacity of main storage and houses all reference data associated with processing. Magnetic drum storage and magnetic disk storage are examples of auxiliary storage devices. Generally, auxiliary storage is not directly accessible to input devices. Input information is usually routed through main storage to auxiliary storage. The CPU cannot reference auxiliary storage for either instructions or operands. When auxiliary storage information is needed, that information is written into main storage. The CPU then accesses main storage for the desired information. Similarly, output devices generally cannot access auxiliary storage. When output information is in auxiliary storage, that data is first written into main storage and then read out of main storage to the output device.

Information written into a storage location destroys the original contents of that location. Information read out of a storage location, however, does not affect the original contents of that location. This is called nondestructive readout. Nondestructive readout applies directly to drum and disk storage. When dealing with core storage, the actual readout is destructive; however, the end result is nondestructive. In any case, by preserving the original contents of a storage location after readout, the same information may be used many times.

Any storage operation requires identification of the desired location and transfer of information either into or out of that location. The time involved to realize these two actions is called access time. The more access time needed, the slower the device. Core storage is the fastest device, followed by magnetic drums and magnetic disks.

Core storage is very often referred to as memory because memory is the function of reproducing what has been learned. A computer, in effect, learns when information is written into storage and remembers when storage is accessed for information. In data processing, the terms storage and memory are synonymous.

Central Processing Unit

The CPU is responsible for almost all the processing in the data processing system. To satisfy this function, the CPU must be able to determine the type of processing desired, must have access to all source data, must be able to establish the necessary transfer paths for a specific operation, and must be able to perform the specified operation.

The type of operation to be performed is specified by an instruction. Instructions are stored in core storage in predetermined locations. The CPU makes all instruction fetches by referencing these predetermined core storage locations. The instruction contained in the referenced location is transferred into the CPU, where it is decoded. The result of the decoding tells the CPU precisely what type of processing to perform and the location of an operand to be processed. The CPU then references core storage for the desired operand. When the operand is transferred to the CPU, circuits necessary to accomplish the operation are established, and the specified processing is performed.

The CPU also controls I-O operations. Initially, an instruction is fetched from core storage. The instruction is decoded, and the necessary transfer paths are established. If additional information is required, the decoded instruction tells the CPU to fetch it. The CPU then fetches the additional information from core storage and sends it to the proper control circuits. With this action completed, a transfer is effected either from core storage to an output device or from an input device to core storage.

The following are CPU operations:

1. Instruction fetching.
2. Instruction decoding.
3. Operand fetching.
4. Circuit setup.

5. Processing.
6. Information exchange.

Of these six functions, all but processing are accomplished in the CPU control area; processing is accomplished in the arithmetic and logical area.

Peripheral Equipment

Peripheral equipment is that equipment operationally removed from the CPU. This I-O equipment includes card readers, magnetic tape units, paper tape readers, punches, and printers, etc.

Card readers enter punched card data in main storage. The punched information on a card is converted into an electronic form and rearranged into machine words. These words are then transferred into main storage. Generally, a program is initially entered in a machine via the card reader.

Magnetic tape units can serve as either an input or an output device. A tape unit initially receives information from main storage and writes this information on the magnetic tape. When the information is needed by the CPU, the magnetic tape is read and the information is transferred into main storage. Once a program is entered in main storage, it can be transferred to a magnetic tape. The magnetic tape can then serve to enter the program in the machine on subsequent runs.

Paper tape readers are similar to card readers. Data is represented on paper tape by means of punched holes. These holes are converted into electronic impulses, which are assembled into machine words. The machine words are then transferred into main storage.

Punches convert electronic impulses into punched holes. Two types of punches are available: card punches and paper tape punches. Information punched out on either paper tapes or cards is received from main storage. A punch is an output device.

Printers provide a visual record of processing results. A printer receives data in the form of electronic pulses from main storage. The electronic pulses drive circuits which, in turn, actuate printing elements. All printing devices have a paper transport which automatically moves the paper as printing progresses.

7040-7044 CONFIGURATIONS AND OPTIONAL PACKAGES

Configurations

For small applications, necessitating only card and printer equipment, the configuration shown in Figure 1 could be used. The CPU is used as a processing unit and as a transfer path for I-O operations. A single 1414 I-O synchronizer * services a 1403 printer and a 1402 card read punch. This synchronizer connects these devices with the CPU. In addition, a console printer, which is included as standard equipment, is available for use as an output device. An operator's console is also standard equipment. This console is physically part of the CPU, but is useful in both CPU and I-O operations.

For installations requiring a higher I-O speed, magnetic tape units can be installed to operate with a separate I-O synchronizer but using the CPU as a transfer path. Figure 2 shows this kind of arrangement.

The basic 7040-7044 configuration, then, includes a single I-O channel, called data channel A. Data channel A is realized by using the CPU circuits for I-O operations. Since the same circuits are used for both processing and I-O operations, either operation can be performed at a time, but not both. Consequently, data channel A is a non-overlapped channel: that is, processing must stop while an I-O transfer is in progress. Only two examples of the possible configurations are shown in these figures. Potentially, a great many more configurations are possible using data channel A.

The I-O capability can be further expanded and its speed increased with the incorporation of the 7904 data channel (Figure 3). With this configuration, an alternate path is provided to core storage which bypasses the CPU entirely. The 7904 is called data channel B; the 7040-7044 can accommodate as many as four. Each additional 7904 data channel is given a different alphabetical label; thus, with a maximum configuration, 7904 data channels are identified as data channel B, data channel C, data channel D, and data channel E, all identical.

The 7904 type data channel is intended to service high-speed I-O devices and auxiliary storage devices. Consequently, the figure illustrates tape and disk systems using this type of data channel. Notice that a synchronizer connects the tapes to the data channel, whereas a file control connects the disk storage to the data channel. The data channel, in turn, provides a common connection for both to core storage. Again, not all possible configurations are shown.

Optional Packages

Instructions

Optional packages in a data processing machine basically are measured in terms of instructions that

* several models of the 1414 I-O synchronizer exist. The model chosen depends on the I-O equipment controlled.

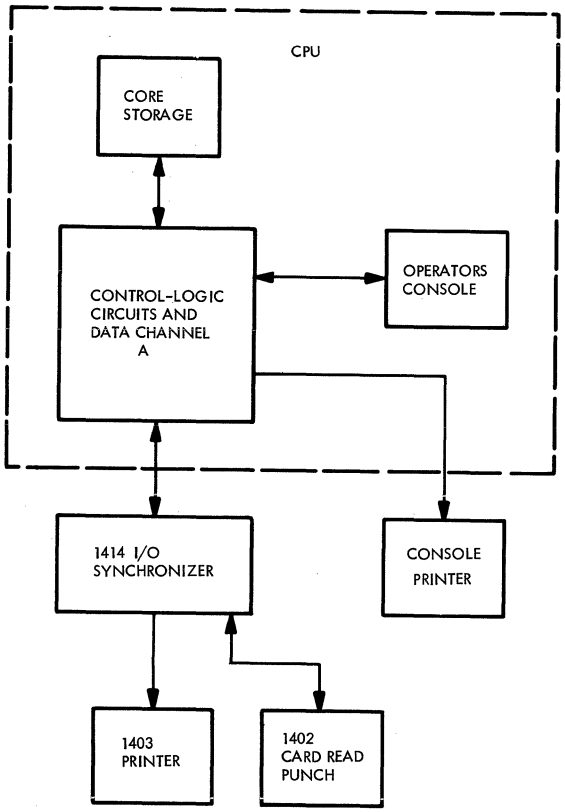


FIGURE 1. 7040-7044 WITH CARD SYSTEM CONFIGURATION, NON- OVERLAPPED OPERATION

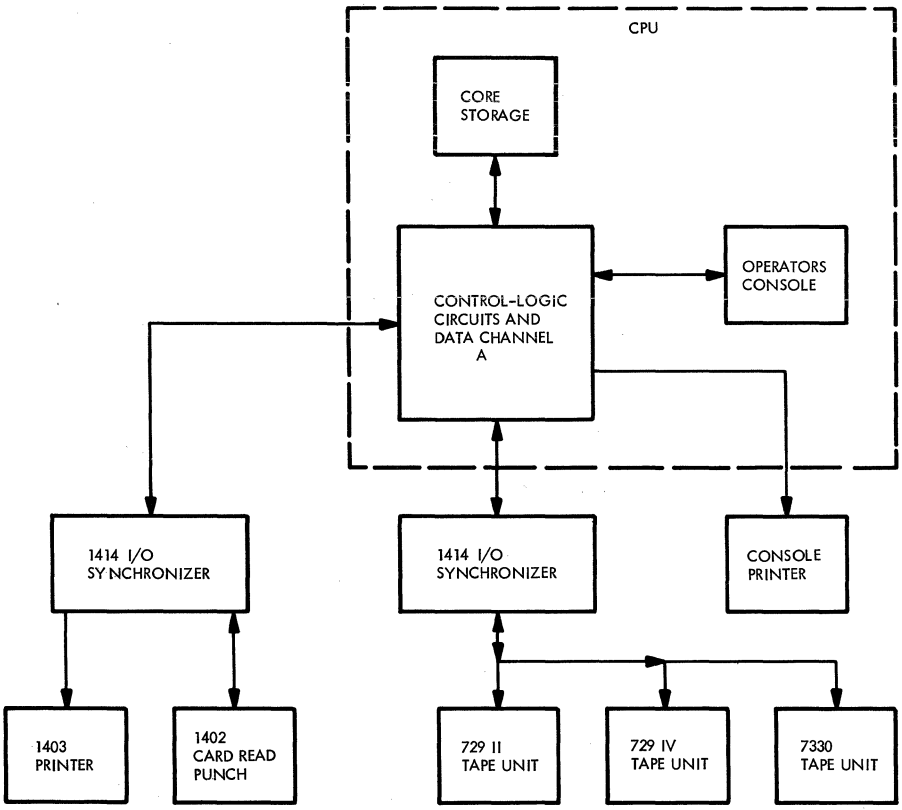


FIGURE 2. 7040-7044 WITH CARD AND TAPE SYSTEMS CONFIGURATION, NON-OVERLAPPED OPERATION

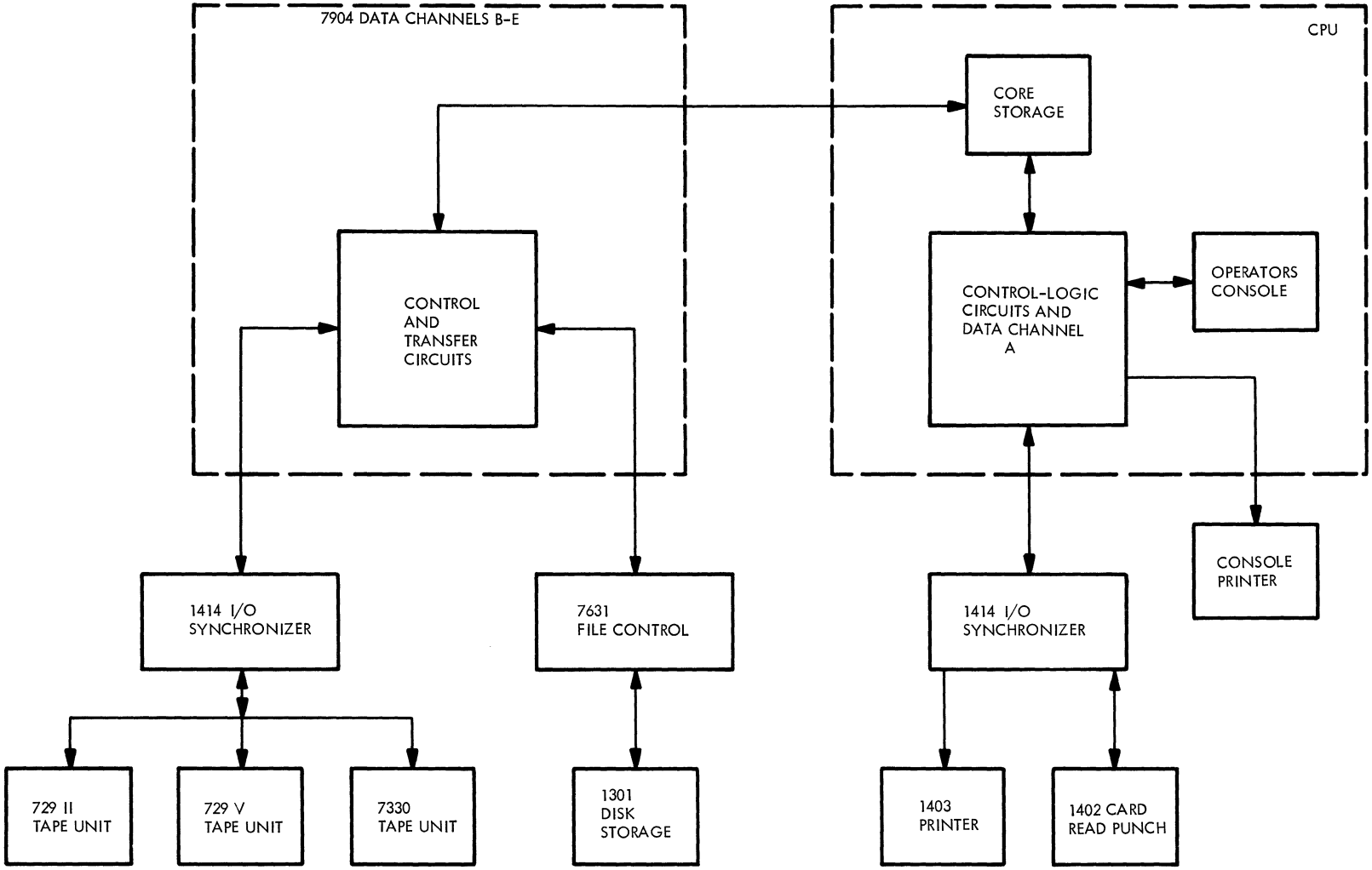


FIGURE 3. 7040-7044 WITH CARD, TAPE, AND DISK SYSTEM CONFIGURATION, OVERLAPPED OPERATION

can be executed by the machine. To realize the merits of the optional instructions requires familiarity with the basic instruction set. The instructions forming the basic set and each optional package are given in the IBM Reference Card 7040-7044 Codes, Form X22-6696, and in the Appendix B of the IBM 7040 and 7044 Data Processing Systems Student Text, Form C22-6732.

Memory Protect Option

This option permits the programmer to protect a portion of memory against alteration by storing. Two instructions are associated with this option, Set Protect Mode (SPM, -1160), and Release Protect Mode (RPM, -1004). The SPM instruction puts the machine in a memory protect mode and specifies the memory block to be protected. The RPM instruction takes the machine out of memory protect mode. Attempts to store in a protected area can result in "trapping", a feature discussed in detail later in this manual.

Interval Timer Option

Unlike the other options, no specific instructions are associated with the interval timer option. When this option is incorporated into a machine, circuits are provided which generate a C cycle. During a C cycle, core storage location 00005_8 is periodically incremented whenever system power is on. Normal processing is interrupted for two cycles in a 7040 machine and for three cycles in a 7044 machine to read out location 00005_8, increment it by 1, and return the incremented value to location 00005_8. The sign position of this location is not used, but, if it is negative initially, it will be made positive during the first increment cycle.

Location 00005 may be set to any value under program control and thus serve as a real-time clock. If an overflow occurs while it is being incremented, the contents of the instruction counter at the time of the overflow are stored in location 00006_8 and the next instruction for execution is fetched from location 00007_8. Consequently, location 00005_8 can also serve as an interval timer. The stepping rate of location 00005$_8$ is once every 1/60 second, thereby maintaining 7090 compatibility.

INSTRUCTIONS AND OPERANDS

A data processing machine is directed to perform each of its operations by an instruction. An instruction, then, is a master command telling the machine what to do, what logic circuits to establish, what information is needed, and where to get it. Processing is a series of actions leading to an end. Since an instruction effects a single action, it is the basic element of processing.

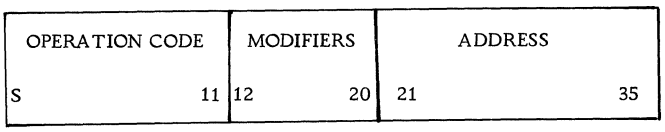
Because instructions direct the operations of a data processing machine, the entire collection of instructions associated with a particular machine is known as the instruction set. An instruction set is divided into logical groups of instructions possessing common characteristics. These logical groups are called classes. For example, all instructions that deal with arithmetic fall within the arithmetic class, all instructions that place information in main storage from the CPU fall within the store class, and instructions which effect I-O operations fall within the I-O class. Auxiliary storage is generally read and written into with I-O instructions.

The general makeup of an instruction is known as the instruction word format. The term format embraces:

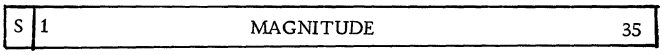
1. the length of an instruction.
2. the fields of an instruction.

The length of an instruction is the number of binary digits (bits) needed to code the entire instruction in the machine. In a particular machine, the instruction word length generally is fixed. In the 7040-7044, for example, the instruction word length is 36 bits: 36 binary digits are required in the 7040-7044 to define each operation accommodated by the instruction set.

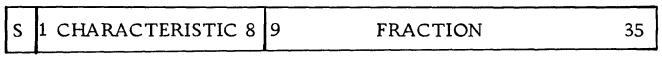
Instructions may be divided into three general fields: operation code, address, and modifier fields. The operation code field defines the general action to be performed. The address field, in most applications, identifies the storage location of the desired operand. The modifier field complements the operation and address fields. For the 7040-7044, the basic instruction word format is as follows:



An operand is a magnitude or quantity upon which a mathematical operation is performed. In a word, an operand is data. More specifically, an operand is a unit of data: one quantity. The format of an operand or data word is determined by the machine involved and the type of arithmetic employed. The machine involved determines the length of an operand, and the type of arithmetic determines field definition, if any. In the 7040-7044, the basic data word format consists of a sign bit and 35 magnitude bits:



The 7040-7044, however, also has provision for floating-point operations. A floating-point data word is the same length as the basic operand, but the word is broken into three fields:



The time involved in a CPU operation may be divided into two periods, known as instruction time and execution time. Generally, the only real distinction between instructions and data is the time when they are brought into the CPU. If a data word is brought into the CPU during instruction time, the CPU interprets the data word like an instruction. Conversely, if an instruction word is brought into the CPU during execution time, the instruction word is treated like data. Consequently, the CPU can operate on its own instructions.

ADDRESSING

In a data processing system, an address is a place where a unit of data may be communicated with. Each unit of data is placed in a register for safekeeping until needed for machine operations. A register located in main storage consists of ferrite cores. A register located in the CPU may consist of tubes or transistors in auxiliary memory, magnetic spots on a smooth surface. Wherever a register which serves to store data is located, an address is assigned to identify it. The address is nothing more than a group of numbers, unfolding sequentially; the concept is identical with that of locating a particular dwelling on a street.

Each instruction in a given set has an address field. The address contained in this field identifies some register or location in the data processing system whose contents are needed for processing or whose contents are to be replaced. This sort of addressing is explicit; that is, the desired address is specifically stated. In the 7040-7044, the category of explicit addressing may be divided into two types: direct addressing and indirect addressing.

Direct addressing is the straightforward expression of a desired location; that is, the address stated by the instruction word is the real address of the desired location. However, an instruction word address field can be modified by arithmetic; such action is called address modification. In the modifier field of an instruction word, provision is made to specify the location of a register whose contents can be algebraically added to or subtracted from the instruction word address field. The result of this arithmetic is the effective address. The register whose contents are added to or subtracted from the instruction word address field is known as an index register. Another name for the instruction word address field is the base address. Consequently, an effective address is obtained by adding to or subtracting from the base address the contents of the specified index register. If no index register is specified, or if the index register specified contains all 0's, the base address becomes the effective address. For direct addressing, then, the effective address specifies the desired location.

Indirect addressing is the roundabout expression of a desired location - it is not straight to the point. For indirect addressing, the effective address, as defined above, specifies a location whose address field specifies the real address of the desired location. Address modification can also be applied to the location whose contents specify the real address. For example, assume the instruction word effective address specifies location 100 as containing the real address. Only the address portion of address 100 is used. If an index register is specified by the address 100 modifier field, the contents of that index register are fetched and added to or subtracted from the address 100 address field. The resultant effective address becomes the real address of the desired location.

In summary, the category of explicit addressing is divided into direct addressing and indirect addressing. Each of these types can employ address modification. Address modification involves changing a base address to realize an effective address. An effective address is the usable address.

Another category of addressing is implicit addressing. An implicit address is an address understood but not expressed. Implicit addresses are stated by the instruction word operation code field. For example, the Add instruction tells the CPU to add the contents of the effective address to the contents of the accumulator. The accumulator, then, is the implied address, and the accumulator contents form the implied operand. No group of numbers is associated with an implied address. Necessary transfer circuits connecting to the implied address are formed as a result of decoding the instruction word operation code field. Address modification does not, therefore, apply to implicit addressing.

PROGRAMMING

A program is a series of instructions coded in a form recognized by the processing unit and calling for operations to be performed by the processing unit in an order necessary to solve a given problem. For example, the solution of a simple arithmetic problem requires a program, whether solved by a data processing machine or by a man with a pencil and paper. The man can recognize the necessary steps

in a program, but the machine must be given step-by-step directions for the solution of any problem. Without this series of instructions, the machine cannot perform any type of operation.

The necessity for programming is apparent: it is needed to initiate and exercise control over the operations of the processing unit. This control may be predetermined through the use of a specific instruction or may depend on the value of the numbers being manipulated at any particular point. In addition to controlling arithmetical operations, programs are used for various other functions, such as mainteance and monitoring.

A program is designed after obtaining a statement of the problem to be analyzed. With this initial requirement satisfied, four subsequent phases are required to produce a finished program:

1. Problem analysis.
2. Program organization.
3. Program coding.
4. Program testing.

Generally, the first phase is handled by mathematicians, and the other three by programmers. Frequently, however, problem analysis determines the organization of the program and is therefore done either by a mathematician-programmer or by a mathematician and a programmer working together.

After a statement of the problem is obtained, all factors that may be encountered have to be examined and arranged in a mathematical expression, which must represent the problem as simply as possible. This expression is usually complex at this point and must be reduced to simpler terms (addition, subtraction, etc.) by a mathematical technique known as numerical analysis.

Numerical analysis involves reducing complex mathematical operations to arithmetic operations within the capabilities of the machine being programmed. Examples are calculus operations reduced to simpler arithmetic operations, such as changing integration to an approximate summation operation and changing differentiation to an approximate difference-quotient operation. These changes result in approximations which can be as exact as desired.

Given a method of approximation, the programmer must then determine the program to obtain the result. The first step involves organizing a program to solve the problem, using the arithmetic methods outlined in the numerical analysis. Program organization involves sequencing the operations to be performed so as to simplify coding and to minimize execution time and, if possible, the number of storage locations required.

At this point, a flow chart is useful, both to keep the entire program in view and to develop the sequence of operations in the proper order. Figure 3-A is a flow chart of the structure of the program. An exact flow chart of a complex problem will necessarily start out in rough form and become finalized only after considerable thought and reworking. Once a tentative flow chart has been prepared, the program can be coded. The coding operation is ofter performed block by block by block from the flow chart. Both the data provided by the preceding block and the data required by the following block must be considered. Coding, then, is the selection of the precise instruction(s) to accomplish the action in a given block. The product of the coding phase is a mnemonically coded program, ready for testing.

Once a program is completely coded, it is tested to ensure its proper operation in solving the given problem. The logical design of the program is tested and revised until it correctly performs its intended function. During testing, modifications of both the program organization and the coding may be required to get the program into proper operation.

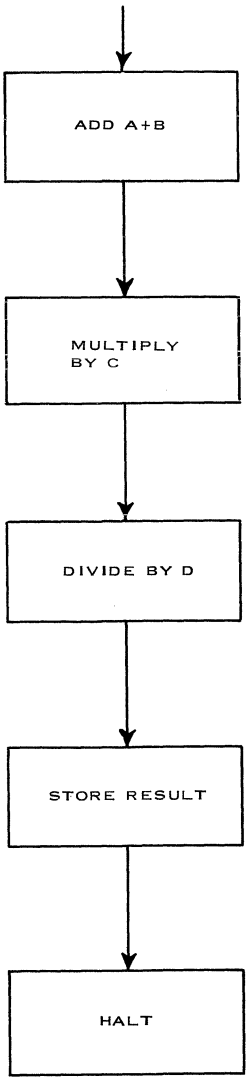


FIGURE 3A. BASIC FLOW CHART

INTERRELATION OF SYSTEM AREAS

Each system basically contains four operational areas: core storage, the central processor, data channel A, and the 7904 data channels. Figure 4, a register level diagram of a 7044 data processing system, shows that a single core storage services the central processor, data channel A, and the 7904 data channel. Data is transferred into and out of core storage via a single storage bus which connects to the other three operational areas. The desired core storage location is identified by the use of a memory address register (MAR). This register receives two inputs: one from the address register in the central processor and the other from the 7904 data channel address register. Associated control circuits determine which input is accepted into the MAR and where the information in the corresponding location is transferred from the storage bus. Core storage is physically part of the central processing unit (CPU).

The central processor satisfies the operational control and processing requirements of the system. One register, the storage register, is used both as an input path from core storage to the central processor and as an output path from the central processor to core storage. When satisfying the input function, the storage register receives instructions in accordance with the program being executed and any operands called for by those instructions. Since a single register handles both instructions and operands, distinction between the two is made with the timing scheme. Simply stated, storage register contents are treated as an instruction at one time and as an operand at another time. When instructions are received from core storage via the storage bus, the entire instruction enters the storage register. However, parts of the instruction also enter other registers associated with the decoding function of the central processor. These other registers basically serve to identify the type of instruction and to indicate whether the instruction word base address is to be modified and, in character handling operations, which character is involved. Only the address portion of the instruction word is significant to the storage register. When operands are received from core storage via the storage bus, the entire operand enters the storage register and is significant only to the storage register; that is, the other registers serviced by the storage bus do not act on operands.

The output function of the storage register is used only by store type operations.

Arithmetic is accomplished in the adder and is binary. Three types of arithmetic are available: fixed point, variable length, and floating point. In fixed-point and floating-point arithmetic, addition, subtraction, multiplication, and division can be performed. In variable length, however, only multiplication, division, and a combined operation of multiplication and addition can be performed.

Data channel A serves as a transfer path through the central processor for I-O operations. The selection function of a data channel A operation is an extension of central processor instruction decoding, and the control function of a data channel A operation is a combination of central processor operand fetching and a specialized use of the accumulator. The transfer function is satisfied, in part, by circuits unique to I-O operations and by circuits normally used in central processor operations. Note that data enters and leaves data channel A on the core storage side via the storage register and on the I-O device side via an interface. Data channel A is physically part of the CPU.

The 7904 data channel is also used for I-O transfers. It provides a second path between core storage and I-O devices. This data channel is a physical unit apart from the CPU. A 7904 data channel operation is selected similarly to a data channel A operation; that is, the 7904 data channel operation select function is also an extension of central processor instruction decoding. The control function, again, is identical with central processor operand fetching. However, the control word for a 7904 data channel operation goes to this data channel, as opposed to a data channel A operation, which uses the accumulator to hold the control word. Once the control word is fetched, a 7904 data channel operation is independent of central processor circuits. Because of this characteristic, the 7904 data channel is known as an overlapped data channel.

Both the data channel A and the 7904 data channel are discussed in separate manuals.

The above discussion applies equally to the 7040 data processing system. Though Figure 4 illustrates a 7044 system, the central processor, data channel A, and the 7904 data channel are identical for a 7040 system. The only configuration difference between the two systems is found in core storage: a 7040 system uses a 7106 core storage; a 7044 system uses a 7107 core storage. If the former were inserted in place of the latter in Figure 4, a 7040 data processing system would be illustrated.

DESCRIPTION OF CPU REGISTERS

The 7040 data processing system uses a 7106 CPU, whereas the 7044 data processing system uses a 7107 CPU. Each CPU is an integrated unit combining an

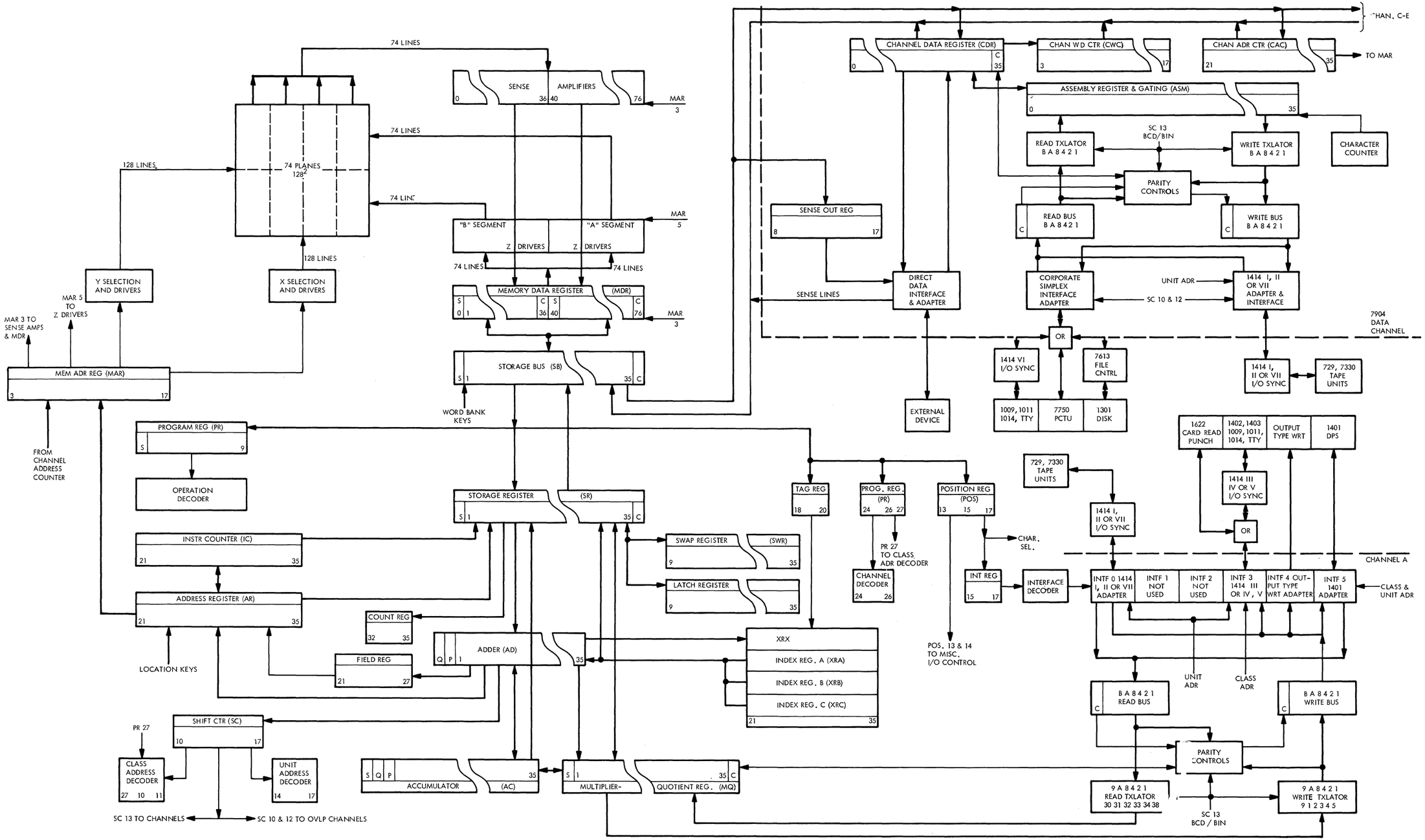


FIGURE 4. 7044 DATA PROCESSING SYSTEM, BLOCK DIAGRAM

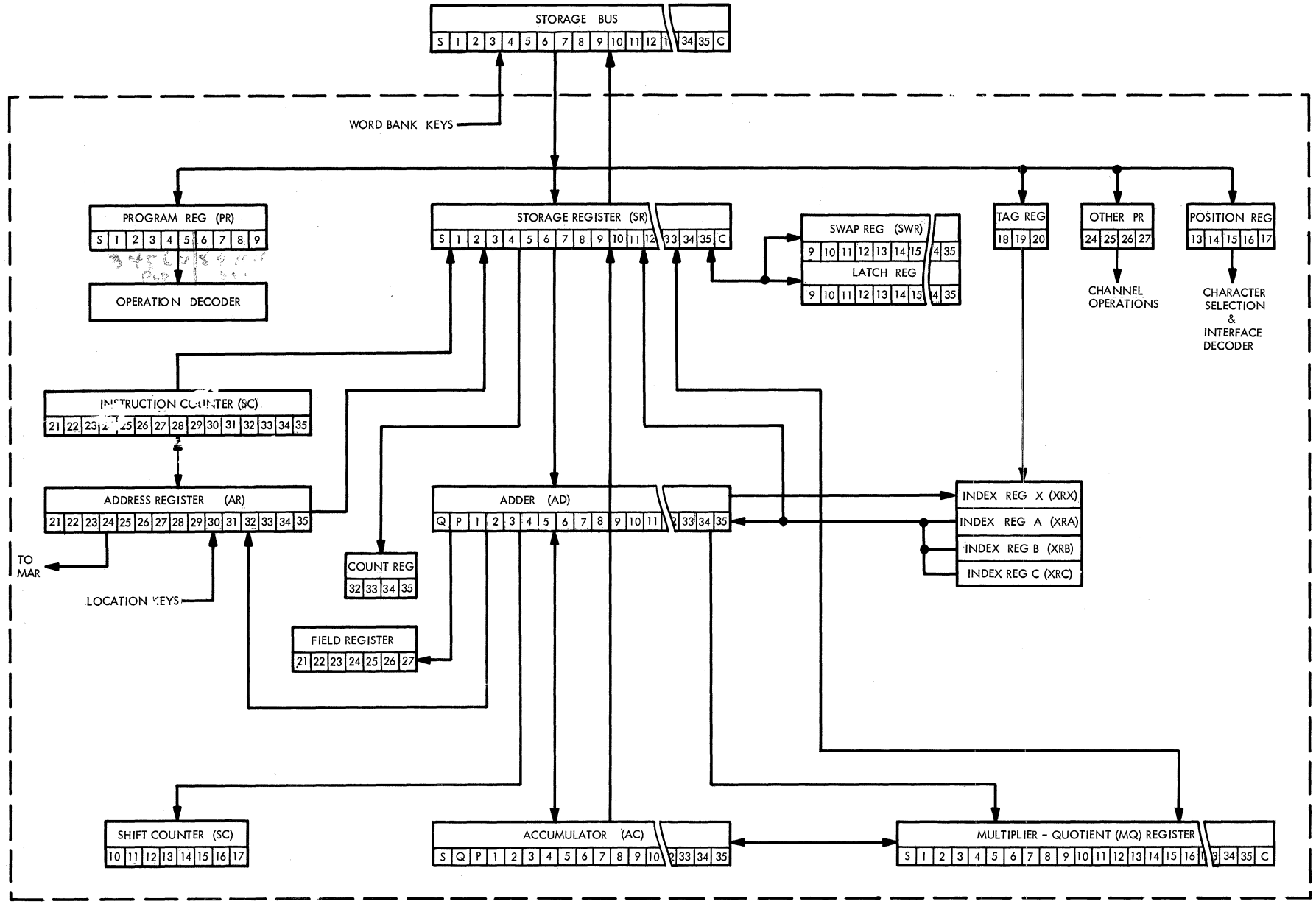


FIGURE 5. 7040-7044 CPU, SIMPLIFIED BLOCK DIAGRAM

arithmetic and a control area, a core storage, an operator's console, and a nonoverlapped data channel. Operationally, the CPU may be considered a configuration of arithmetic and control circuits and is treated as such in this section. Although the other elements are physically part of the CPU, their operations are distinctly different from CPU operations.

With respect to register configuration and data flow transfer paths, the 7106 CPU and the 7107 CPU are identical. This discussion of Figure 5, a block diagram of the 7040-7044 CPU, applies to both equipments.

Storage Register (02.01.00-35)

The 37-bit storage register (SR) (a sign bit, 35 information bits, and a parity bit) is composed of shift cells and serves as the input to the CPU from core storage for both instructions and data. Parity bit C provides an indication of memory word parity on all CPU storage readout operations. The storage register is fed directly from the storage bus (SB) on all instruction word and operand fetches and, in each case, receives a full 37-bit word. On an instruction fetch, however, only bits 21-35 or 28-35 (depending on the instruction) are used in the storage register. On an operand fetch, the entire word is used.

Program Register (02.04.00-09)

The program register (PR), a 10-bit latch register, receives the operation code portion of an instruction from the storage bus on an instruction fetch. Program register contents are decoded to generate control signals for instruction execution; these control signals are called primary operation decoder (POD) signals and secondary operation decoder (SOD) signals.

Other Program Register (02.04.47)

The other program register (OPR), also a latch register, is actually an extension of the program register. However, the OPR is significant only to data channel operations and is discussed under that topic.

Tag Register (02.04.20)

This is a three position latch register used to select the three index registers in address modifications and other operations involving index registers. The tag register latches are set by instruction word bits 18, 19, and 20 directly from the storage bus to select the index registers C, B, and A, respectively. In any index operation, when more than one tag register latch is set, the corresponding index registers will be selected to produce multiple outputs; when corresponding bit positions in two selected index registers contain 011 and 010, for example, their combined outputs will be 011. In operations where an index register is to be loaded, all selected index registers will receive the same input.

Adder (02.02.00-37)

The adder is a 37-bit binary adder used for performing all binary arithmetic as well as address modification through indexing. The adder is discussed in detail in the next section.

Accumulator (02.02.00-37)

The 38-bit shift cell accumulator register holds one factor during arithmetic operations and receives the result from adders. In addition, data can be shifted one bit at a time to the left or right in the accumulator. The accumulator contains two overflow bit positions, Q and P, in addition to a sign bit position and 35 data bit positions. Bit positions Q and P retain adder overflows for control purposes. The P bit position is also used as the highest order bit in logical operations.

MQ Register (02.01.00-35)

The MQ register is a 36-bit shift cell register (S, 1-35) which, during a multiply operation, initially holds the multiplier and, finally, the low-order portion of the product. During a division operation, the MQ register initially holds the low-order portion of the dividend and, finally, the quotient. The MQ register can also be shifted in conjunction with the accumulator or rotated within itself.

Swap Register (02.30.82-90)

The swap register is a 27-bit auxiliary register (shift cells) used to temporarily hold fractions during double-precision floating-point operations. Its only input is from SR bit positions 9-35, and its only output is to SR bit positions 9-35.

Latch Register (02.30.82-90)

The latch register is a 27-bit auxiliary register used to temporarily hold fractions during double-precision floating-point operations. Its only input is from SR bit positions 9-35, and its only output is to SR bit positions 9-35.

Index Registers (02.03.21-35)

Three index registers (latch type) are included in the CPU register configuration: index register A (XRA), index register B (XRB), and index register C (XRC). These registers are specified by tag register contents. A 1 in tag register bit 20 specifies XRA; a 1 in tag register bit 19, XRB; a 1 in tag register bit 18, XRC. Thus, XRA is also known as index register 1; XRB, as index register 2; XRB, as index register 4. The only input to an index register is a 15-bit index value from index register X (XRX), a buffer used in loading the index registers. XRX is loaded from adder bit positions 21-35. Outputs from the index registers are a 15-bit value in 2's complement form to adder bit positions 21-35 and a 15-bit value in true form to SR bit positions 21-35 or 3-17.

Instruction Counter (02.04.21-35)

The 15-bit (shift cells) instruction counter (IC) provides for sequential instruction execution. The only input to the instruction counter is from the address register. Outputs from the instruction counter are to the address register and to SR bit positions 21-35.

Address Register (02.04.21-35)

The 15-bit (latches) address register (AR) is used to transmit instructions and operand addresses to the core storage MAR. Inputs to the address register are from adder bit positions 21-35, from the instruction counter, and from the operator's console location keys.

Outputs from the address register are to MAR in core storage, to the instruction counter, and to storage register bit positions 3-17 (trap operation).

Shift Counter (02.04.10-17)

The 8-bit (shift cells) shift counter (SC) is used to count the number of shifts specified in shifting operations. The shift counter also holds part of the instruction in I-O select type operations. Further, in transmit operations, the shift counter controls the number of words transmitted.

Position Register (02.04.18-19)

The 5-bit position register (latches) receives instruction word bits 13-17 directly from the storage bus. Position register contents are used only for character-handling operations. In these cases, position register bit positions 15, 16, and 17 specify which character in the word specified by the effective address is to be used in the operation. The only outputs from this register are gating signals which cause the specified character to be gated from its word position in the storage bus to SR bit positions 30-35 or which cause accumulator bits 30-35 to be gated into the specified character position in the storage register.

Field and Count Registers (02.16.01-03)

The 7-bit field register is used only in the memory-protection mode. This latch register is loaded with bits 21-27 of the Set Protect Mode (SPM) instruction word effective address. Working with the field register is the count register, also latch-type, which receives SPM instruction word bits 32-35. Field register contents form the pattern with which subsequent memory references are compared. Count register contents are used as follows:

1. Bit 32 specifies whether an equal or an unequal compare represents a protected area violation.
2. Bits 33-35 specify the number of high-order address bits to be examined on subsequent memory references. The field register has one input from adder bit positions 21-27, whereas the only count register input is from SR bits 32-35. Both the field register and the count register have one output going to control circuits, where the signals are combined to produce resultant signals which feed compare circuits.

This chapter provides a detailed analysis of the control and transfer circuits within the CPU.

The term CPU is often used instead of central processor. Strictly, the terms are applied more accurately when restricted to the physical and functional sense, respectively. However, since maintenance personnel commonly use CPU when referring to central processor operations, the terms are used interchangeably. In areas where confusion might result, a distinction is made between physical confines and functional operations.

Although Trapping and Memory Protection would logically be taken up in this section, they are treated separately in the last two sections of this manual because of their complexity, and because the rest of the CPU functions can be learned independent of these two subjects. Channel A, the IO function of CPU, is described in a separate manual: IBM 7040-7044 Data Processing System, Channel A CEMI, Form R23-2652.

SHIFT CELLS AND LATCHES

The CPU uses two types of circuits for register formation, shift cells and latches. Before studying the actual CPU controls, it is important to become familiar with these circuits.

Figure 6 shows a logic representation of the shift cell, a circuit used to make up most of the CPU registers. The shift cell is a bistable device very similar to a trigger. It has on and off outputs, and it stores one of two conditions. Instead of separate AC set and reset inputs, however, the single shift input is connected to both sides of the shift cell the same as a trigger is connected for binary operation. The single gate input determines which side of the shift cell is affected by the shift input. If the gate is -B, the shift input turns the cell on; if the gate is +B, the shift input turns the cell off. The gate (positive or negative) must be active for at least 200ns before the shift pulse. This characteristic of the shift cell prevents shifting a bit more than one position with a single set pulse when the shift cells are arranged in series, as in a register.

Figure 7 is the logic equivalent of a latch. Although this circuit appears more complicated than the shift cell, the latch is also very simple and straightforward in concept: +A1 is the set device, and +A2 is the reset device. Assume the entire circuit is void of input circuits. +A1 needs two positive signals to be satisfied. Assume these signals are now present. Their presence causes the +A1 output, an in-phase output, to go positive. With a positive output from +A1, the OR circuit is satisfied, causing the OR circuit out-of-phase output to go negative. The negative output of the OR circuit feeds an inverter (I) which inverts the signal, making it positive. The positive output of the inverter serves two functions:

1. It is the output of the latch representing a binary 1.
2. It is used as a feedback signal to realize the latch function.

Note that, following the feedback loop, the positive output signal from the inverter conditions one input to the +A2 reset. The other line is the reset line, which is negative only when a reset signal is generated. Consequently, this reset line is most often positive. Since it is positive, the +A2 input is satisfied, causing the +A2 in-phase output to go positive. This positive output feeds the OR circuit, thereby maintaining the negative output from the OR circuit and, by extension, the positive latch output and feedback signal. With this arrangement, the +A1 inputs can be removed (made negative) without affecting the latch status. The only way to alter the latch is to generate the negative reset signal and apply it to +A2. This negative input to +A2 deconditions +A2, causing the +A2 output to go negative. A negative output from +A2 deconditions one of the OR inputs. The other OR input is deconditioned by the absence of the +A1 inputs. Logically, a +OR is also a -A. At this point, two negative inputs are present at the OR circuits; thus, the -A requirements are satisfied. The +OR, which is now functioning as a -A out-of-phase output, goes positive, causing the I input to go positive. Consequently, the I output goes negative. The negative output signal represents:

1. A binary zero.
2. The feedback signal of the latch, because this negative signal to +A2 causes the +A2 output to remain negative.

With the +A2 output negative, the +OR circuit cannot function unless the +A1 circuit receives new inputs to condition it. Therefore, with a negative output signal, the latch is functioning logically, because the reset input to +A2 can again go positive without any effect on the latch.

Comparing the shift cell and the latch reveals that each is a binary bistable device. However, that is where the comparison ends. The shift cell inverts and retains the gate input, whereas the latch, in effect, passes the set input. A shift cell does not have to be cleared before data can be placed in it, but a latch does. Lastly, the set pulse used in the shift cell is the same pulse that changes the CPU clock and is called a cell-driver-output pulse. Therefore, the

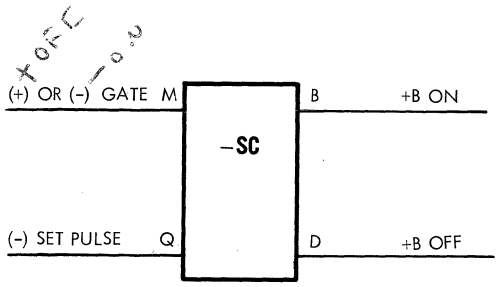


FIGURE 6. SHIFT CELL

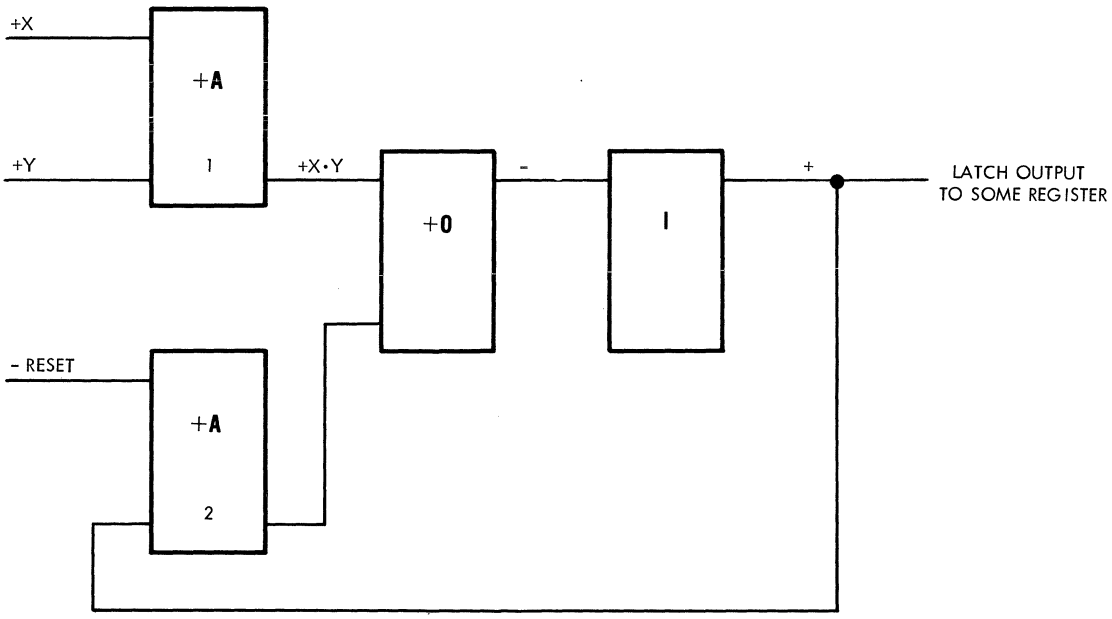


FIGURE 7. LATCH

actual information is not set into the shift cell until the pulse time following the set pulse, but the latch input information is set into the latch immediately upon receipt.

The CPU latch register, the program register, the address register, index register X, index registers A, B, and C, the tag register, and the position register are formed by latches. All other CPU registers are formed by shift cells.

PULSE GENERATION

General

The complex - though not complicated - job of keeping the computer running smoothly and properly is the function of a multitude of timing pulses. These timing pulses all originate from a single, free-running oscillator; the oscillator's output is transformed - via a timing ring and a series of AND circuits - into all the raw timing pulses necessary for computer operation.

The raw timing pulses are refined into working pulses (or levels) by AND'ing them together with various other conditions to obtain the proper pulse at the proper time. For instance, before an E cycle can start, a raw timing pulse is AND'ed with a level indicating that it is all right, with regard to the machine, to enter an E cycle; this level, of course, is contingent on many factors; e.g., a higher priority cycle has not been requested, or the previous cycle is finished. The raw timing pulses, then, do no actual machine work but check on various contingencies for proper machine operation; the pulses (or levels) resulting from the "contingency checks" are the actual working pulses.

Both the 7040 and 7044 CPU's have a machine cycle of 2.5 microseconds (usec). Since the 7044 uses the 7107 memory, which has a 2.5 -usec memory cycle, the two units are compatible without any special circuit innovations. However, the 7040 uses the 7106 memory, whose memory cycle is 8.0 usec. In the 7040, then, circuitry is used to effectively - though not actually - slow its machine cycle down to be compatible with the 7106 memory cycle; this effective slowing down is required only when the 7040 is actively using the memory. When the 7040 is performing a function independently of memory - for example, a shift operation - it will operate at its 2.5-usec machine cycle rate. So, in the case of the 7040, the timing circuitry has the additional function of buffering the CPU to the memory.

Pulse Generation

All timing pulses originate from a free-running oscillator that begins generating pulses the instant power is brought up in the CPU and continues generating them until power is dropped. The oscillator and its associated circuitry are shown in Figure 8, A. The oscillator feeds a pulse generator whose output is a continuous string of narrow positive pulses every 416 nanoseconds.

Since a machine cycle is 2.5 usec long, and since the pulse generator yields one pulse every 416 nanoseconds, there are six oscillator pulses in a machine cycle. For ease of reference, the first raw timing pulse(oscillator pulse) of a machine cycle is called A0; the second, A1, etc., through A5. The next pulse after A5 is A0 of the following machine cycle. The output, then, of the oscillator and pulse generator is thought of as being in groups of six pulses, numbered A0 through A5.

These pulses are fed to a clock timing ring, where they are developed into useful pulses.

Clock Timing Ring

The function of the clock timing ring (shown simplified in Figure 8, B) is to transform pulses from the oscillator and pulse generator into levels that can be used in the system.

The clock timing ring keeps running as long as power is up. The result is the output waveshapes shown in Figure 8, C. Each third pulse causes a given shift cell to switch, so the output of each cell is +B for a duration of three pulses and -B for a duration of three pulses. Since oscillator pulses are 416 nanoseconds apart, a duration of three pulses is 3 times 416, or 1248, nanoseconds, one-half of a machine cycle.

Both the in-phase and the out-of-phase outputs of the shift cells are used (not shown in Figure 8 for simplicity) and are fed through inverters for powering. The result is the six levels shown in Figure 8, C.

The levels are labeled to indicate exactly what kind of level (pulse) it is. For instance, in the pulse labeled -B A1 D3, -B tells the polarity. A indicates a raw timing pulse (a refined pulse would carry a different letter to designate the kind of cycle; e.g., an E cycle pulse would have an E instead of the A. The 1 tells at what time the pulse goes to a -B in this case. If the pulse were a +B pulse, the 1 would indicate at what time the pulse became +B. The 1, of course, indicates the second oscillator pulse of a machine cycle, so its time is 1 times 416 nanoseconds after pulse A0. The times of the other pulses are calculated the same way; the D3 means duration of the three pulses. This

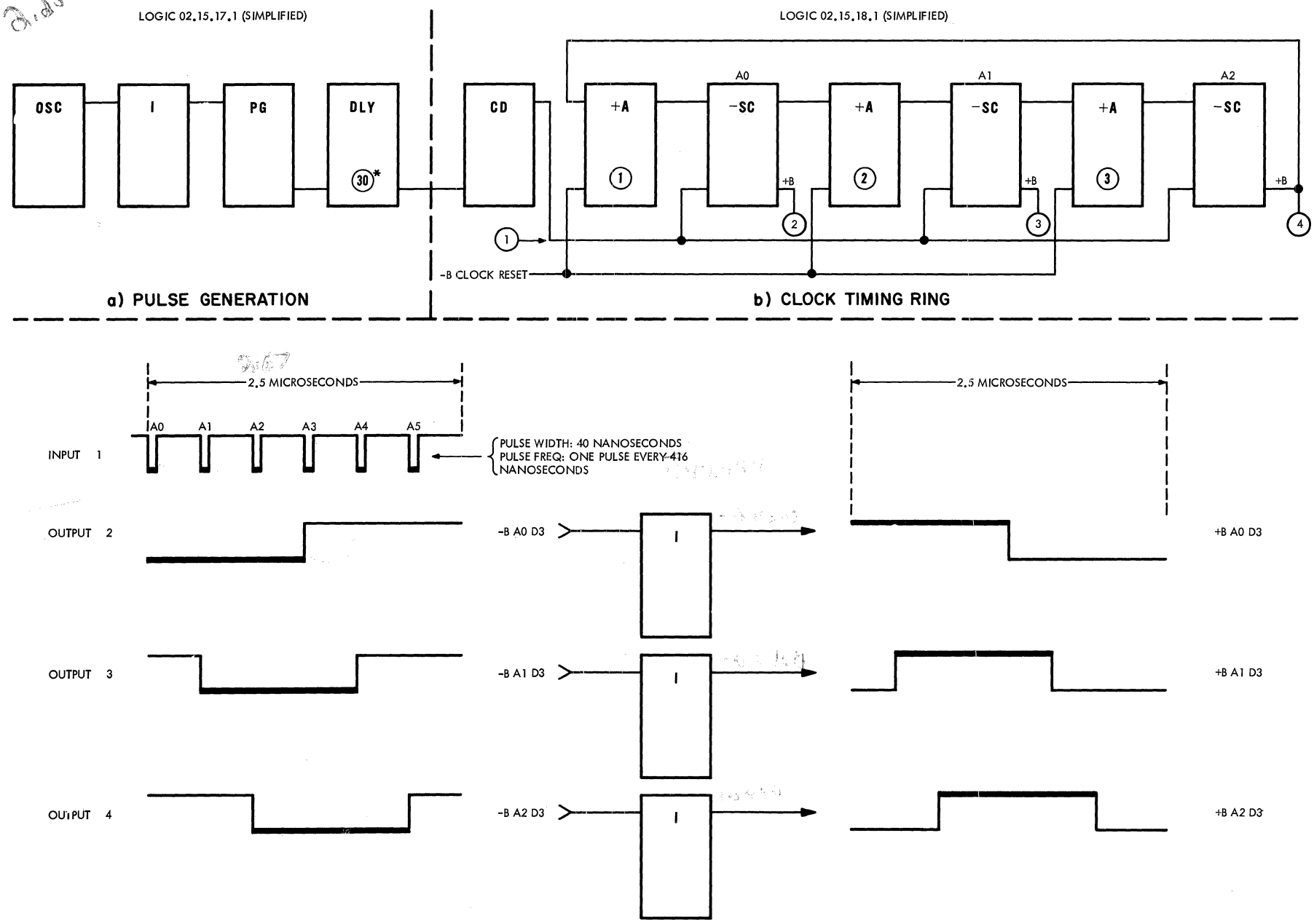


FIGURE 8. CLOCK TIMING RING

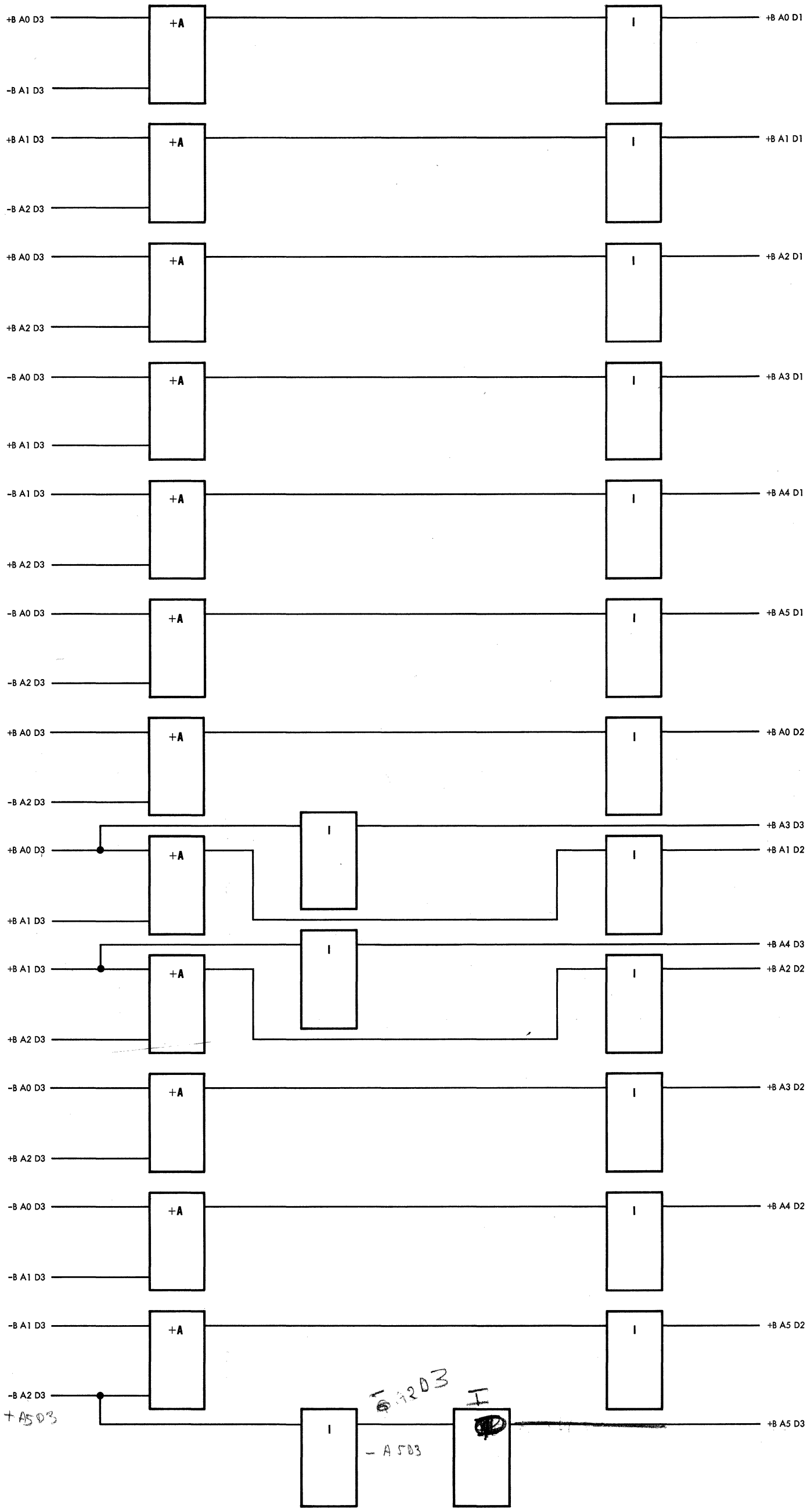


FIGURE 9. CLOCK

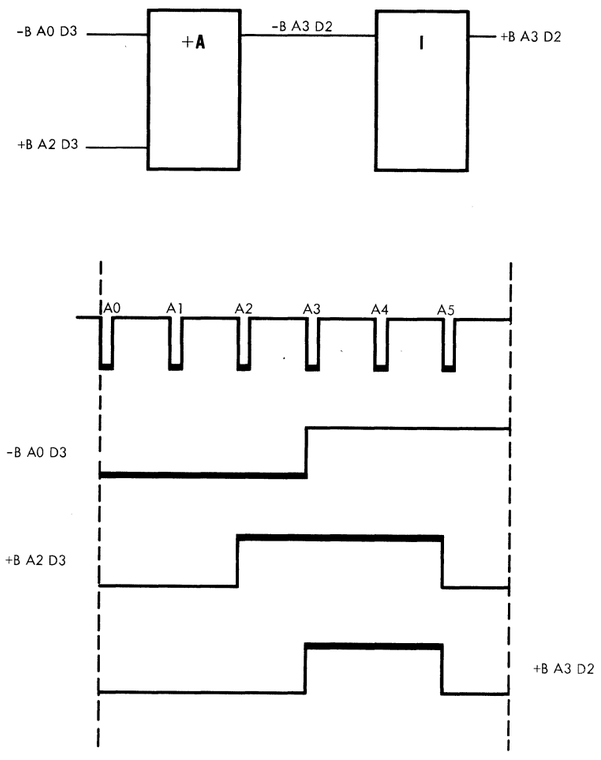


FIGURE 10. SINGLE CLOCK STAGE

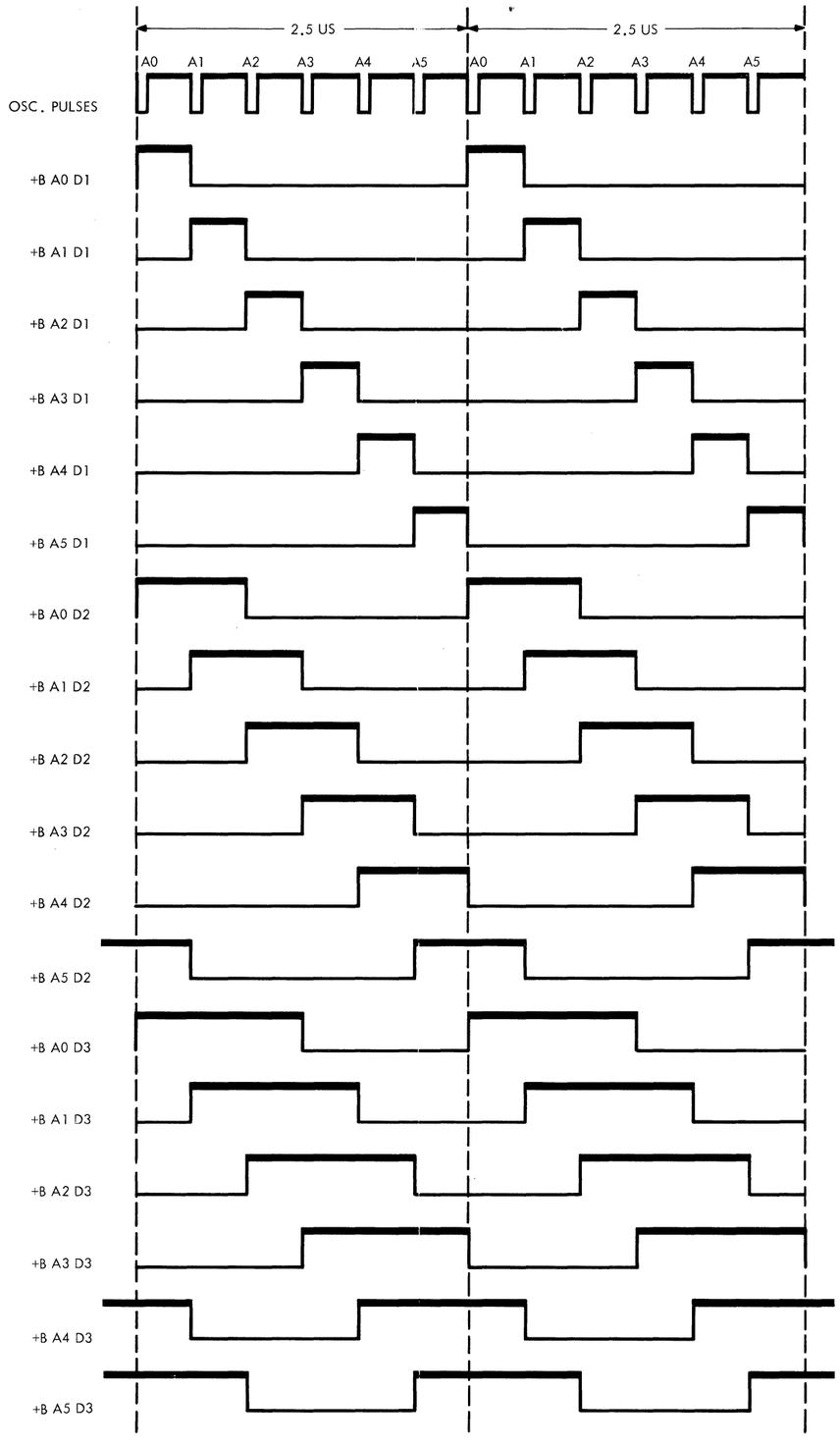
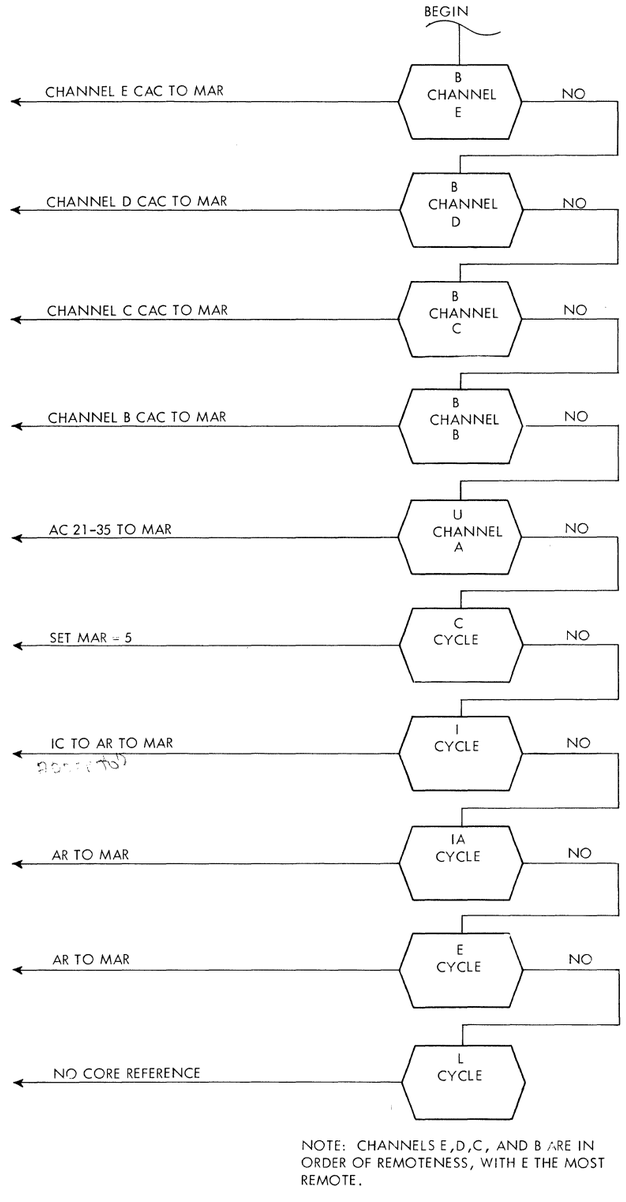


FIGURE 11. TIMING PULSES



NOTE: CHANNELS E,D,C, AND B ARE IN ORDER OF REMOTENESS, WITH E THE MOST REMOTE.

FIGURE 11A. CYCLE REQUEST PRIORITY

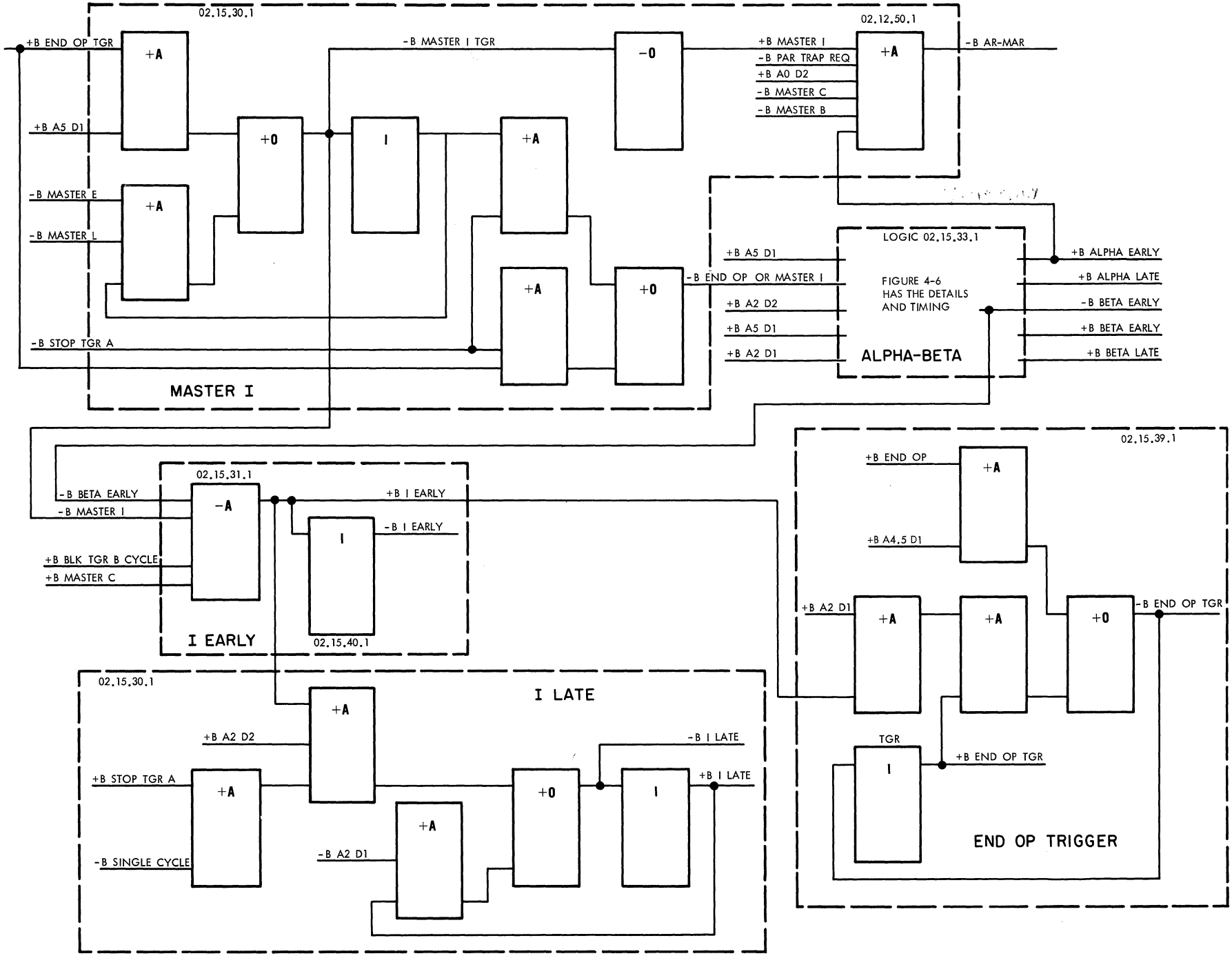


FIGURE 12. I CYCLE TIMING

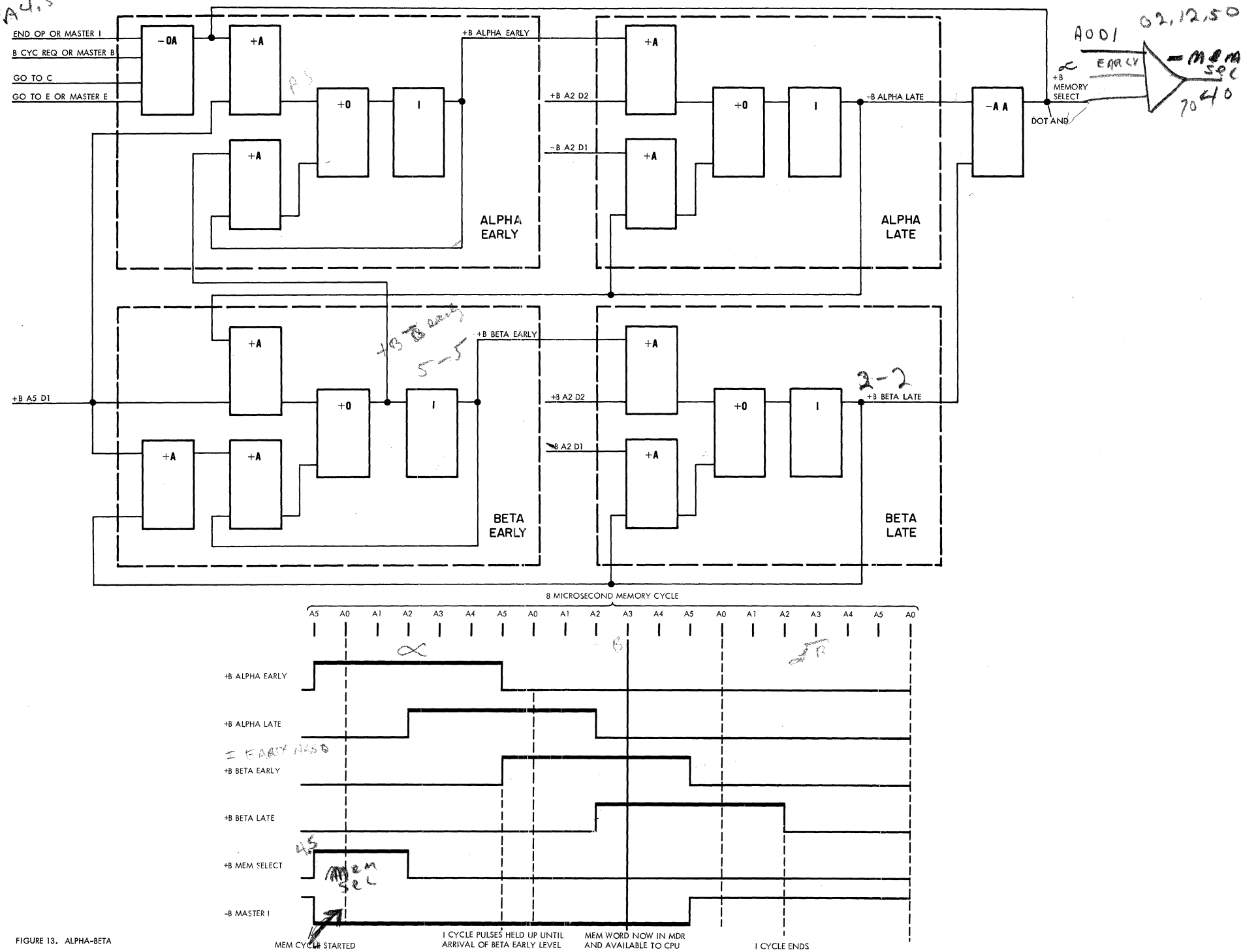


FIGURE 13. ALPHA-BETA

indicates the time the pulse remains -B in this case. So, -B A1 D3 indicates a negative pulse starting 416 nanoseconds after the start of a machine cycle, remaining negative for 1248 nanoseconds, and then returning to a positive level. By comparing waveshape 1 in Figure 8, C, with the other waveshapes, the various pulse designations should become clear.

Clock

The clock (shown simplified in Figure 9) receives the six pulses developed by the clock timing ring and AND's each of them with each of the other pulses to obtain 12 +B pulses of varying starting times and durations. It also inverts three of the input pulses without AND'ing to obtain three new pulses. All the pulses developed by the clock, plus all the pulses developed by the timing ring, make up the raw timing pulses for the 7040 and 7044 CPU. Though Figure 9 shows only +B pulses out of the clock, it should be realized that the complement (-B) of each pulse is also available: the -B is available directly off the AND output; the +B is available after the AND output is inverted.

All the AND's in the clock function identically, so by looking at one in detail the entire clock should be understood. Figure 10 shows one of the clock's AND circuits selected at random. The two AND inputs are -B A0 D3 and +B A2 D3. The only time the AND is conditioned is when both inputs are +B at the same time; the timing relationship shown in Figure 10 indicates that this time is from A3 time to A5 time; in other words, A3 for a duration of 2 (D2). The AND output, then, is -B A3 D2, which is run through an inverter to obtain +B A3 D2.

One thing may not be immediately clear in Figure 9: How can +B A0 D3 be fed directly through an inverter and come out as +B A3 D3? The same question applies to +B A1 D3 and -B A2 D3 becoming +B A4 D3 and +B A5 D3, respectively. These are the three pulses shown in the lower portion of Figure 9 that do not go through inverters. The answer is simple: +B A0 D3, after going through an inverter, becomes -B A0 D3, but a -B A0 D3 is identical with +B A3 D3. The relationship of these pulses can be readily seen in the timing ring pulses in Figure 8, C.

The final result of the pulse generator, the clock timing ring, and the clock is the 18 pulses shown in Figure 11. Only the +B pulses are shown, but the -B pulses are also available.

Two machine cycles are shown in Figure 11 to illustrate that some pulses originate in one cycle and end in the following cycle. The need for this is to allow the CPU to prepare for a subsequent cycle ahead of time, thus saving time overall.

All the raw timing pulses just discussed go out to a multitude of places in the CPU to control the operations.

MACHINE CYCLES

Though the CPU performs its operations by executing a series of machine cycles, the machine cycles are not the same. The only thing all machine cycles have in common is that they are 2.5 usec in duration and they are all controlled by the raw timing pulses.

There are six basic types of machine cycles:

1. I (instruction cycle): Fetches instruction words from core storage and decodes them.
2. E (execution cycle): Fetches operands from core storage and stores operands in core storage.
3. L (logic cycle): Performs CPU functions that do not require reference to core storage; for example, shift operations are performed during a logic cycle.
4. U (unoverlapped cycle): Allows channel A to access core storage via the CPU. All channel A data transfers occur during U cycles; at this time, the CPU is used exclusively for channel A operations.
5. B (buffered cycle): Allows channels B through E to access core storage. These channels can access storage without going through the CPU, so it is possible for the CPU to perform L cycles while a B cycle is being performed.
6. C (clock cycle): Allows the interval timer (real time clock), which is in storage location 00005, to be brought from storage, incremented by 1 in the adder, and then replaced in location 00005; this operation occurs every 60th of a second.

A seventh cycle - an IA cycle (indirect addressing) - could be included, though it really is a variation of an E cycle. Figure 11A shows the priority of these cycles.

I Cycle

The I cycle is the time during which an instruction is fetched from memory and decoded. If specified, address modification by indexing is performed during the I cycle. Further, some instructions can be completely executed during an I cycle.

The sequence of events during an I cycle is as follows:

1. Reset address register.
2. Load AR with IC contents. The signals $\overline{\text{XEC}}$, $\overline{\text{trap}}$, $\overline{\text{transfer successful}}$, and end-operation must be present before this action can be effected.
3. MDR is conditioned to transfer information to the SB.
4. Select memory (reset MAR and MDR).
5. Transfer AR contents to MAR.

6. Transfer AR contents to IC.
7. Transfer MDR contents to SB.
8. Transfer SB data into SR, PR, tag register, and position register.
9. Decode instruction in operation decoders.
10. Step IC.
11. Perform address modification by indexing, if specified:
    a. Transfer SR bit positions 21-35 to adder positions 21-35.
    b. Transfer 2's complement of contents of specified index register to adder positions 21-35.
12. Transfer adder bits 21-35 to AR. If the instruction in progress is a transfer instruction and the attempted transfer is unsuccessful, the IC contents are transferred to the AR and the transfer of adder bits 21-35 to AR is blocked. If the instruction in progress is an I cycle index register load type instruction, adder bits 21-35 are transferred to the XRX rather than to the AR.

Indirect-Addressing Cycle

The IA cycle occurs only if indirect addressing is specified in the instruction word, and, when it is specified, the IA cycle follows the I cycle. The sequence of events during an IA cycle is as follows:
1. Select memory.
2. Transfer AR contents to MAR.
3. Transfer MDR contents to SB.
4. Transfer SB data to SR and tag register.
5. Perform address modification by indexing, if specified.
6. Transfer adder bits 21-35 to AR.

I Cycle Timing

In order for an I cycle to perform its function of fetching and decoding instruction words, several working levels must be developed specifically for the I cycle: master I, I early, I late, and (in the 7040 only) alpha and beta levels. Figure 13 (simplified logic) illustrates the development of these levels.

Master I

The master I level, which is developed first, has four requirements:
1. An end-op trigger, signifying that the instruction currently in progress is in its final machine cycle (almost finished).
2. An A5D1 raw timing pulse.
3. No master E level, which will be present only if an E cycle has been initiated.
4. No master L level, which will be present only if an L cycle has been initiated.

A latch arrangement is used to hold the master I level up after the timing pulses and end-op-trigger disappear. Note that master I will be discontinued only by the advent of either an E cycle or an L cycle.

After being AND'ed to be sure that a parity trap request, a C cycle, or a B cycle has not been initiated, the master I goes out at A0 D2 time to initiate the transfer of the address register to the memory address register (AR-MAR). This, of course, starts the instruction fetch.

Alpha-Beta

As mentioned earlier in this chapter, the 7040, because of the use of the slow memory, must buffer the 2.5-usec CPU to the 8.0-usec memory. This buffering is accomplished by alpha and beta levels. The alpha-beta circuitry is not present in the 7044.

Any one of the following levels will cause alpha and beta levels to be generated:
1. End-op or master I
2. B-cycle-req or master B
3. Go to C
4. Go to E or master E

Regardless of which level initiates the level generation, the alpha-beta circuitry functions the same. Therefore, though this discussion is in reference to an I cycle, it should be understood that the same holds true for a B, C, or E cycle.

The alpha-beta block in Figure 12 shows its functional relationship to the other blocks in the I cycle; Figure 13 shows the details of alpha-beta level development. Both figures should be referred to during this discussion.

The end-op or master-I level initiates all the alpha and beta levels during an I cycle; the alpha-early comes up immediately and allows the master-I to effect the AR-MAR transfer. Also, the memory-select level is generated to ultimately start the memory clock at A0D1 time. Once the memory clock starts, it takes approximately 4 usec before the memory word is read out into the MDR and available to the CPU; an additional 4 usec are required to complete the memory cycle. Since a machine cycle is only 2.5 usec, the I cycle would be all over before the instruction word was ever read out of memory unless some method of delaying I cycle pulses was employed. The absence of a beta-early level delays the I cycle pulses until approximately 1.6 usec before the memory word is due to be put into the memory data register (MDR). Once in the MDR, the word is available to CPU. By the time the I cycle pulses are ready to use the word from memory, the word will be ready for use. Even so, the I cycle pulses will take the word from the MDR, decode it, and be completed long before the memory cycle is completed.

Subsequent memory cycles, however, cannot start until the current memory cycle is completed; this is because the memory select level depends on the absence of alpha and beta late.

I Early

The development of the I-early level is contingent on:

1. Having a master-I level (both 7040 and 7044).
2. Not having a master-B level (both 7040 and 7044).
3. Not having a master-C level (both 7040 and 7044).
4. Having a beta-early level (7040 only).

The I cycle cannot commence until the I-early level is developed. In the 7044, it is developed immediately after the master I comes up, but in the 7040, because of the beta-early contingency, the I early is delayed from being developed until a full machine cycle after the master I is developed.

The I-early level goes out to initiate the I-late level and also out to turn off the end-op trigger.

Like the other levels being discussed, the I Early goes out to various places in the CPU to perform functions in connection with the I cycles. The purpose of this section is to show that levels are developed, how they are developed, and why they are developed; the actual work done by the pulses is discussed as the individual CPU sections that utilize the pulses are discussed.

End-Operation Trigger and I Late

The I-early turns off the end-op triggers, allowing subsequent cycles (E or L) to be initiated.

The end-op trigger is turned on at A4.5D1 time if a +B end-op level is present. The +B-end-op level comes up only if the CPU is nearing the end of an instruction. The +B-end-op then is, in effect, a go to I cycle command from the CPU. There are many sets of circumstances which will cause a +B-end-op to be generated (02.15.35.1) and an I cycle to be initiated. Though the end-op is necessary for I cycle initiation, it does not force an I cycle to be initiated. For example, if a C cycle had been requested, it would be executed in preference to the I cycle.

The I late is initiated by the I early and goes to many places in the CPU to finish executing the I cycle.

E Cycle

The E cycle accommodates the transfer of an operand either from or to memory. The sequence of events during an E cycle for a from-memory operation is as follows:

1. Select memory.
2. Transfer AR contents to MAR.
3. Transfer MDR contents to SB.
4. Transfer SB data into SR.
5. Transfer SR contents to adder. (Many transfers are possible at this point; however, the SR-to-adder transfer is most probable).
6. Perform some specified action; i.e., add.

During a to-memory operation, the sequence of events during an E cycle is as follows:

1. Select memory.
2. Transfer AR contents to MAR.
3. Inhibit transfer of data from specified memory location to MDR.
4. Transfer data to be stored to SR.
5. Transfer SR contents to SB.
6. Transfer SB data into MDR.
7. Transfer MDR contents into specified core location.

E Cycle Timing

In many respects, the E cycle resembles an I cycle because both types of cycles reference memory, causing information to be read out for CPU use. The difference is that the I cycle brings out an instruction word and the E cycle brings out a data word (or stores a data word).

Figure 14 shows the functional blocks necessary for E cycle pulse generation, which is almost identical with I cycle generation (Figure 12). The E-cycle levels (E early, late, etc.) are powered through inverters (not shown) and sent to the CPU to execute an E cycle. The beta-early pulse (in the 7040 only) will delay the E cycle operation just as it did the I cycle operation.

The difference between E cycle and I cycle pulse generation is the go-to-E level, which sets the master-E trigger. Figure 15 shows the conditions necessary for a go-to-E level. A -B out of block 1, plus a +B out of any of the other blocks, brings up the go-to-E level. Block 1 will have a -B output when the CPU is not completing an instruction. The other blocks will have +B output in accordance with the conditions shown in each block.

L Cycle

The L cycle is a non-core-storage reference cycle in which logical and shifting operations are performed. This does not mean, however, that L cycles are employed only to accommodate logical and shifting type instructions. Other types of instructions use L cycles; the criterion for entry into an L cycle for these cases is time. Generally, if not enough time is available in a given cycle to perform the

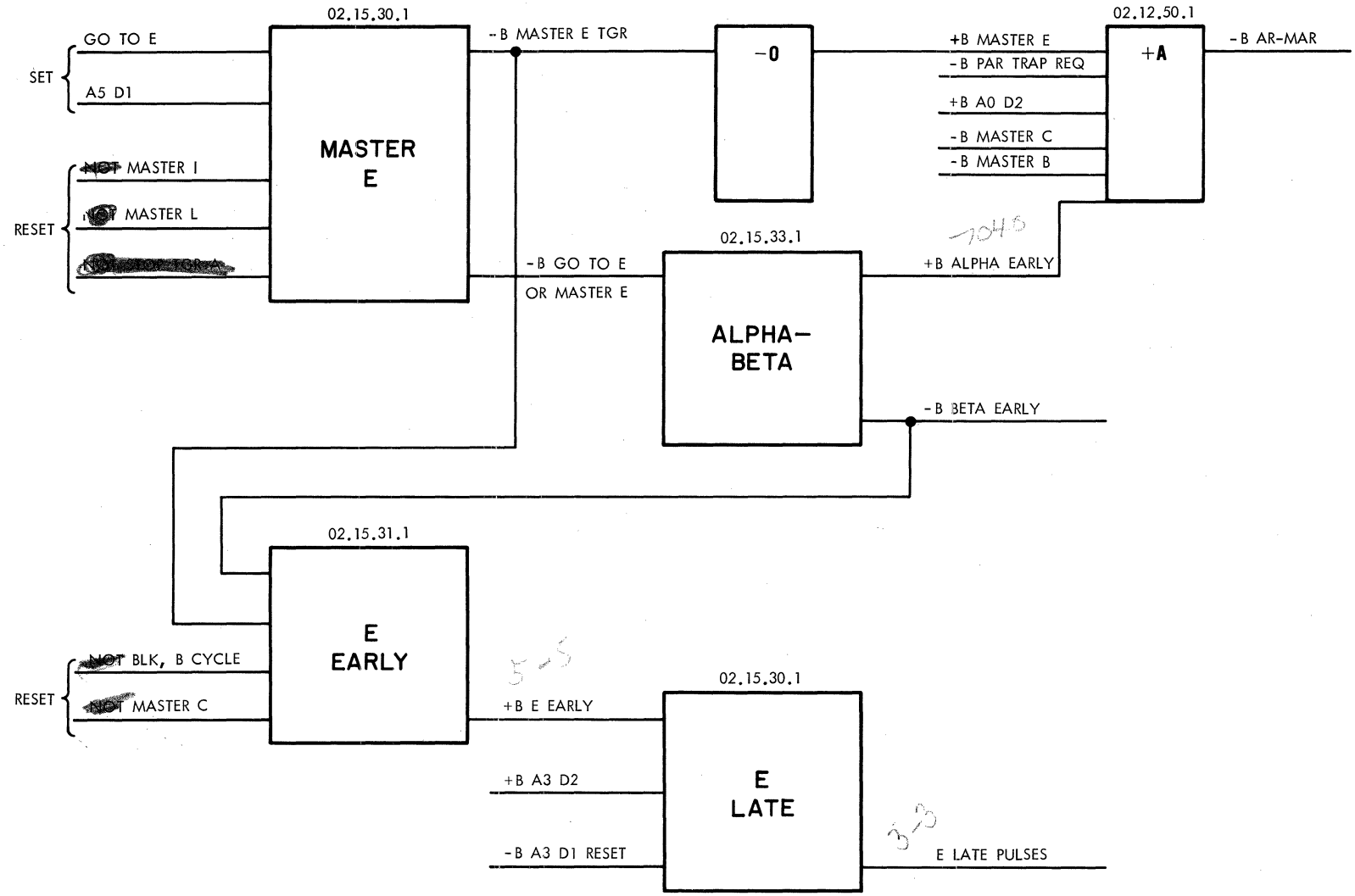


FIGURE 14. E CYCLE TIMING

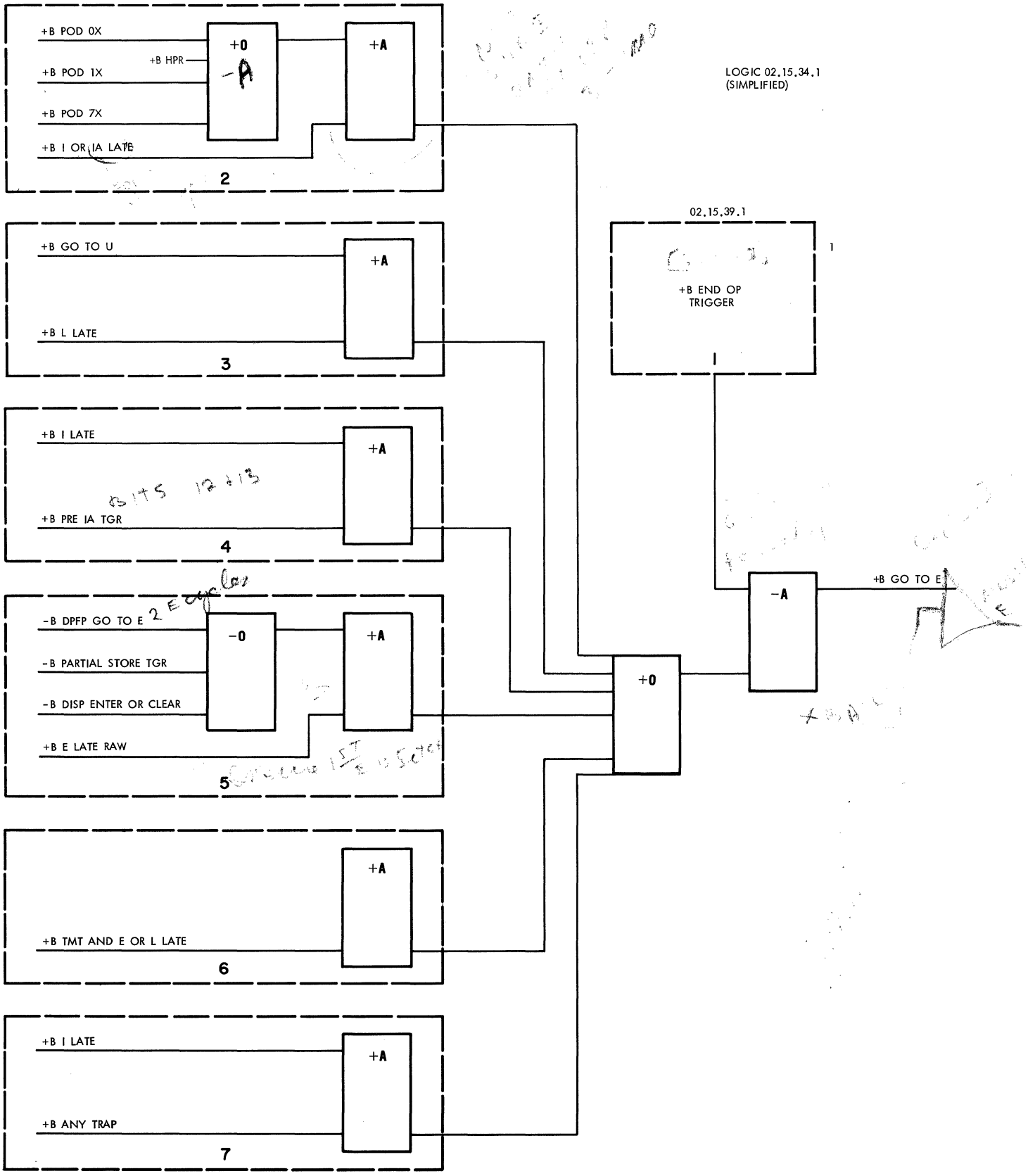


FIGURE 15. GO TO E

operation at hand, an L cycle is taken to complete the action. For example, two CPU cycles are necessary to load an index register with the contents of a defined accumulator field. The first cycle occurs because of the instruction and is therefore an I cycle. However, because of the time required to complete the necessary transfers as opposed to the duration of an I cycle, not the entire operation as specified by the instruction can be performed during an I cycle. Since the instruction, in this case, specifies the accumulator as containing the desired information, there is no need for an IA cycle. Further, since the accumulator is internal to the CPU, there is no need to communicate with memory and, therefore, no need for an E cycle. Thus, any part of the operation that is not realized during I cycle time is realized by entering an L cycle.

Another example of an instruction other than a logical or shift type instruction using an L cycle is the multiply instruction. In performing multiplication, repetitive steps are executed during which addition and shifting or only shifting are effected. For a repetitive step that requires both, the addition phase and the shifting phase are effected during L cycles.

For a shift type instruction, the shift counter generally controls the number of individual operations performed during an L cycle. For example, a single L cycle can accommodate approximately six 1-position shifts. A shift count value greater than 6, then, requires more than one L cycle; a shift count value greater than 12_{10} requires more than two L cycles; etc.

L Cycle Timing

The logic cycle is the simplest of all cycles as far as pulse generation is concerned. The L cycle does not use core storage or (in the 7040) the alpha-beta circuitry. To perform its various functions (shifting, for example), the L cycle requires an L-early and an L-late level. Development of these two levels is shown in Figure 16, A. (Note the similarity to I and E level development.)

The go-to-L level is the key to L cycle timing, and its development is shown in Figure 16, B. An L cycle follows any I cycle that is not followed by an E cycle or any E cycle that is not followed by an I cycle or another E cycle.

B Cycle

A B cycle is generated as the result of a B cycle request being sent to the CPU from one of I-O channels B through E. The B cycle request indicates that the requesting channel wants to either read from or store into memory (core storage). It is the function of the B cycle to control the memory for the I-O channel and to prevent the CPU from using memory at the same time that the I-O channel is using it; this task is fairly simple because channels B-E enter memory directly through the storage bus (SB) without using any portion of the CPU.

Figure 17 shows the B cycle timing and how it controls the use of core storage.

Cycle Generation

At A5 D2 time, a B cycle request effects generation of the master-B level. The master-B level is regenerated every A5 D2 time as long as the B cycle request is present; however, the first A5 D1 that appears turns the master-B off if a B cycle request is not present. This means that the I-O channel, not the CPU, determines the number of B cycles.

The master-B originates two other levels, a B-early and a B-cycle-requested-or-master-B. The former is used in parity checking circuitry. The latter performs two important functions:

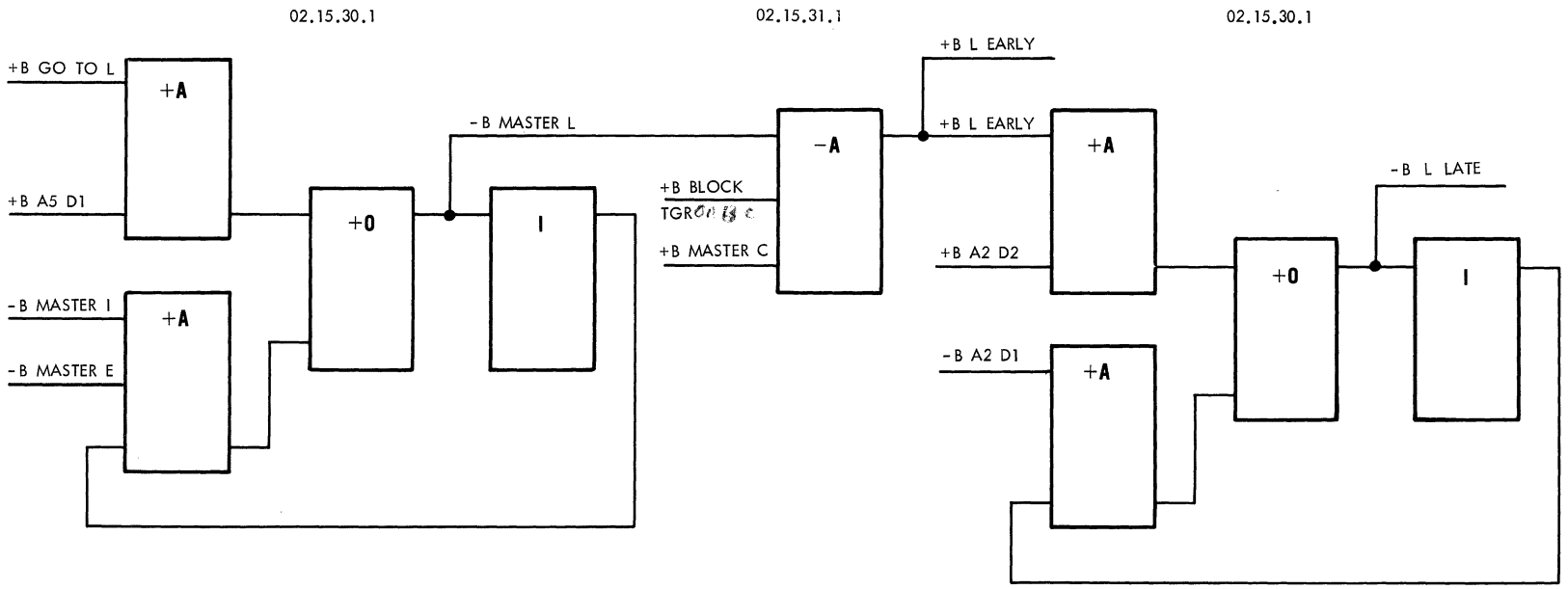
1. It starts the alpha-beta level operation (Figure 13), an identical function with the I and E cycles.
2. It prevents an AR-MAR transfer (Figures 12 and 14). It can now be seen just how a B cycle breaks into CPU operation. The B cycle allows the I and E cycles to generate a master level but prevents them from generating early or late levels. This action hangs up CPU operations (except for any L cycle operations being executed) but does not alter anything. As soon as the B cycle is finished, the I or E cycle is allowed to bring up early and late levels as though nothing had happened. Of course, the B cycle has no effect on an L cycle because the L cycle never uses memory and there can be no conflict.

Memory Control

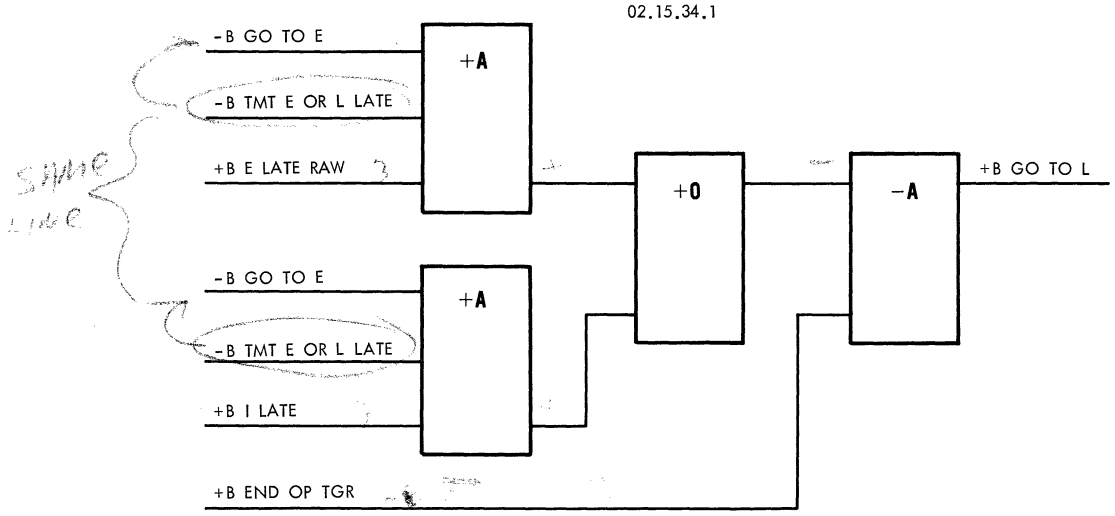
Master B goes out to control the memory lines by being AND'ed with a write-memory level from the I-O channel. This write-memory line is down when the I-O channel wants to read from memory instead of storing into it.

If a write is requested, the read-memory line is a -B. This causes the +B store cycle to come up, thus generating a store cycle. In addition, the -B causes the storage bus (which contains the information from the I-O channel) to be transferred to the MDR, where it is automatically written into memory during the second half of a memory cycle.

If a read is requested, the read-memory line is up and the contents of the MDR are gated out to the



a) L CYCLE LEVELS



b) GO TO L

FIGURE 16. L CYCLE TIMING

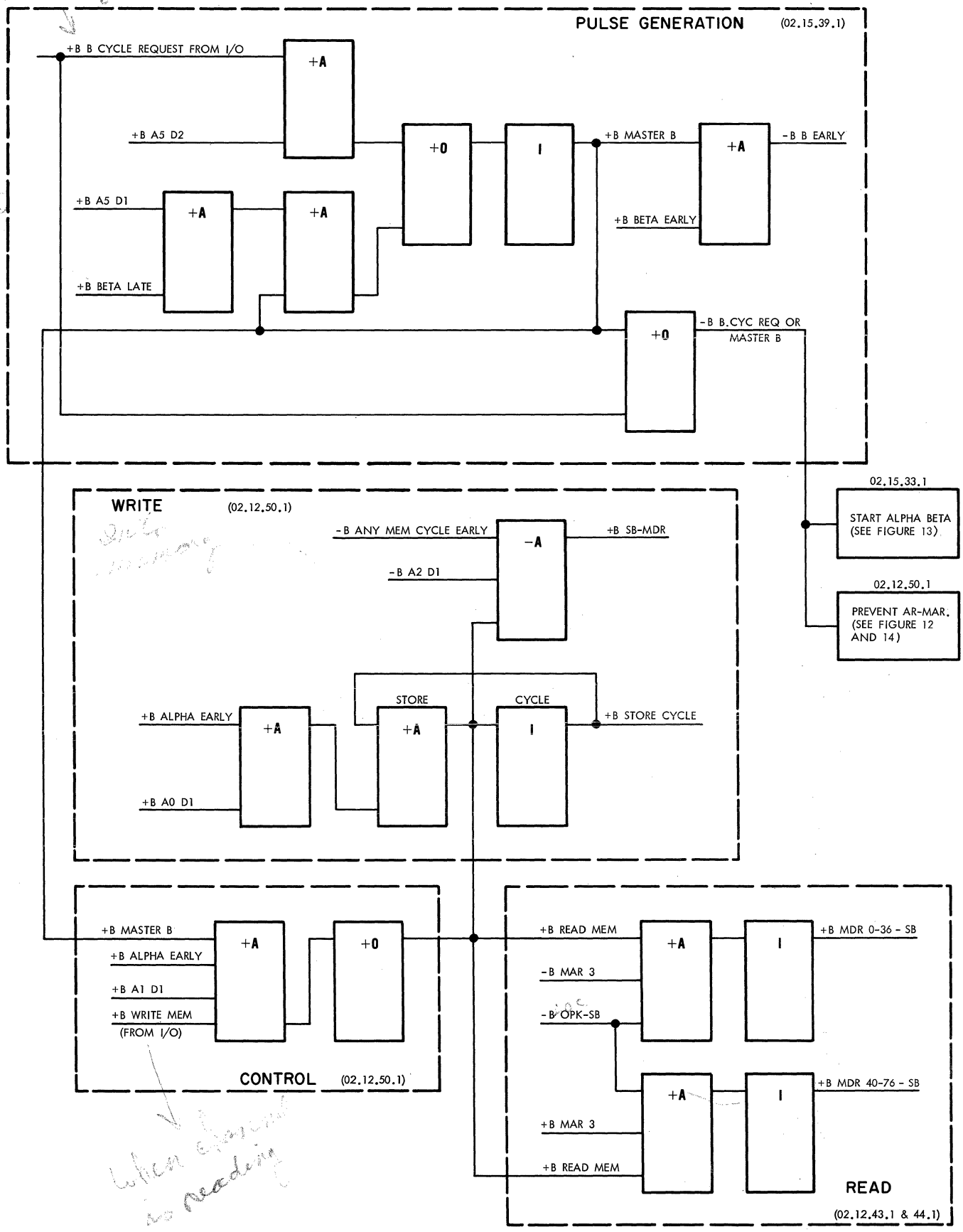


FIGURE 17. B CYCLE

storage bus. The MDR contains, at this time, the information from core storage that was requested by the I-O channel.

U Cycle

The U cycle allows I-O equipment to read from or write into CPU core storage via channel A. The U cycle request initiates an E cycle (Figure15) because most of the E cycle hardware and functions are the same as those required by a U cycle. Any E cycle function not required for a U cycle is inhibited by a U cycle. Conditions that start a U cycle are discussed in the channel A manual.

C Cycle

C cycles are used only to update the interval timer (real time clock), which is memory location 00005. Once every 60th of a second, memory location 00005 is read out, incremented by 1 in the adder, and replaced in the same memory location. The prime power 60-cycle source is used to originate C cycle requests. Every time the 60-cycle power reaches a peak (once every 60th of a second), a pulse is generated which causes a pair of C cycles to be initiated. These two C cycles, which access core storage, have a third cycle sandwiched in between them. This third cycle, called a 1st-C-cycle-delayed cycle, does not access core storage. The complete process of updating the interval timer then consists of three machine cycles: a 1st C cycle, a 1st C cycle delayed, and another C cycle.

During the first C cycle, the following occurs:

1. Force 00005 into MAR.
2. Select memory.
3. Transfer contents of location 00005 to MDR.
4. Transfer MDR contents to storage bus.
5. Transfer SB information into SR.
6. Initiate first C cycle delayed.

The first C cycle delayed cycle accommodates the following:

1. Transfer SR contents to adder.
2. Add 1 to low-order adder bit 35.
3. Transfer accumulator contents to SR to keep them from being destroyed.
4. Transfer adder contents to accumulator.
5. Transfer SR contents to adder.
6. Transfer accumulator contents to SR.
7. Transfer adder contents to accumulator, restoring it.

During the last of the three cycles, which is the second C cycle, the following occurs:

1. Force 00005 into MAR.
2. Select memory.
3. Transfer SR contents to SB.
4. Transfer SB information into MDR.
5. Transfer MDR contents into location 00005.

C Cycle Timing

Figure 18, the simplified C cycle logic, shows the basic functions and timing. The first A4 D1 pulse after each 60-cycle voltage peak generates a C cycle request which will, in turn, initiate a go-to-C level. The go-to-C level is contingent on not having a B cycle or U cycle request. Conditions satisfied, the go-to-C goes out to start the alpha-beta level generation and to initiate a master-C level. The master-C goes out to prevent execution of other cycles and to start the memory readout. The memory readout occurs because a 1 bit is forced directly into MAR positions 15 and 17 (which will decode as storage location 00005), and a read cycle is started by the master-C ultimately starting the memory clock.

The C-early level initiates a 1st-C-late level (Figure 18). The C late transfers the storage bus (which has the contents of location 00005) to the storage register and initiates a 1st-C-cycle delayed.

The 1st-C-cycle delayed transfers the storage register to the adder and adds 1 to the adder. The incremented location 0005 is then transferred from the adder to the accumulator. At the same time, the accumulator is sent to the storage register to preserve valid data the accumulator may have contained. The storage register is now transferred to the accumulator, through the adder, restoring the accumulator content, while the accumulator is transferred to the storage register. The up-dated timer in the storage register is now ready to be stored back in location 0005. This action completes the first two machine cycles of the three required.

The end of the second machine cycle causes a C-cycle-complete level to be generated. This level causes the second C cycle, which:

1. Transfers the storage register to the storage bus and causes it to be written back into memory.
2. Turns off various C cycle controls so the cycle will stop.

Figure 19 shows the timing of the C cycles. Figure 20 shows how a C cycle request is generated. The 60-cycle (prime power) is fed into a clock clamp whose two outputs, C and D, are as shown; C output is up twice as long as D, and both are in the order of milliseconds. A plus D output sets the 60-cycle buffer latch at A2 D1; this latch is reset by a minus C output at A4 D1. However, the latch is not reset by the A4 D1 immediately following the A2 D1 that set it because the long C output is still up (plus) when the A4 D1 appears. The 60-cycle buffer remains latched up until an A4 D1 appears coincident with a down C output; this coincidence may be a matter of milliseconds.

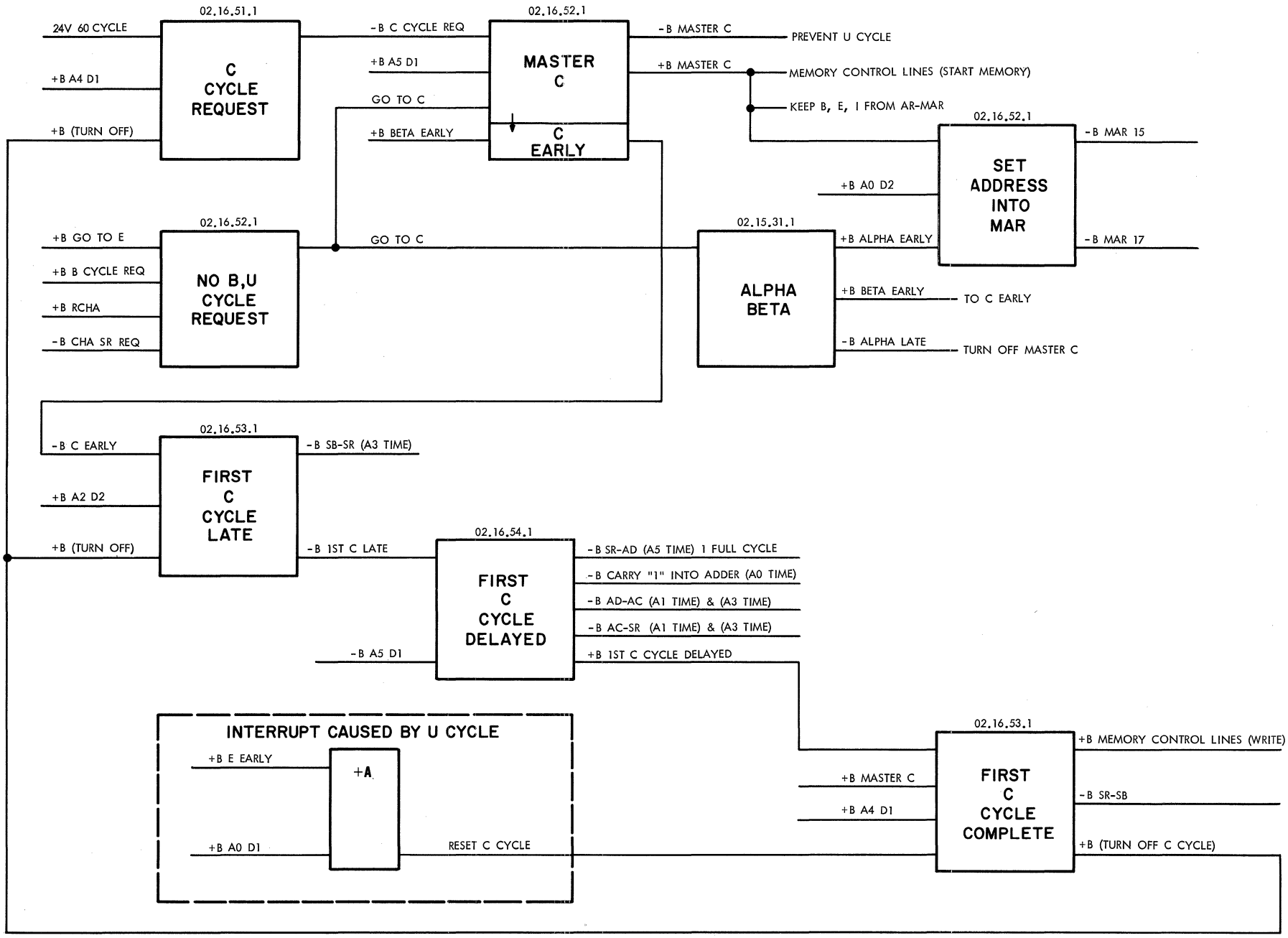


FIGURE 18. C CYCLE

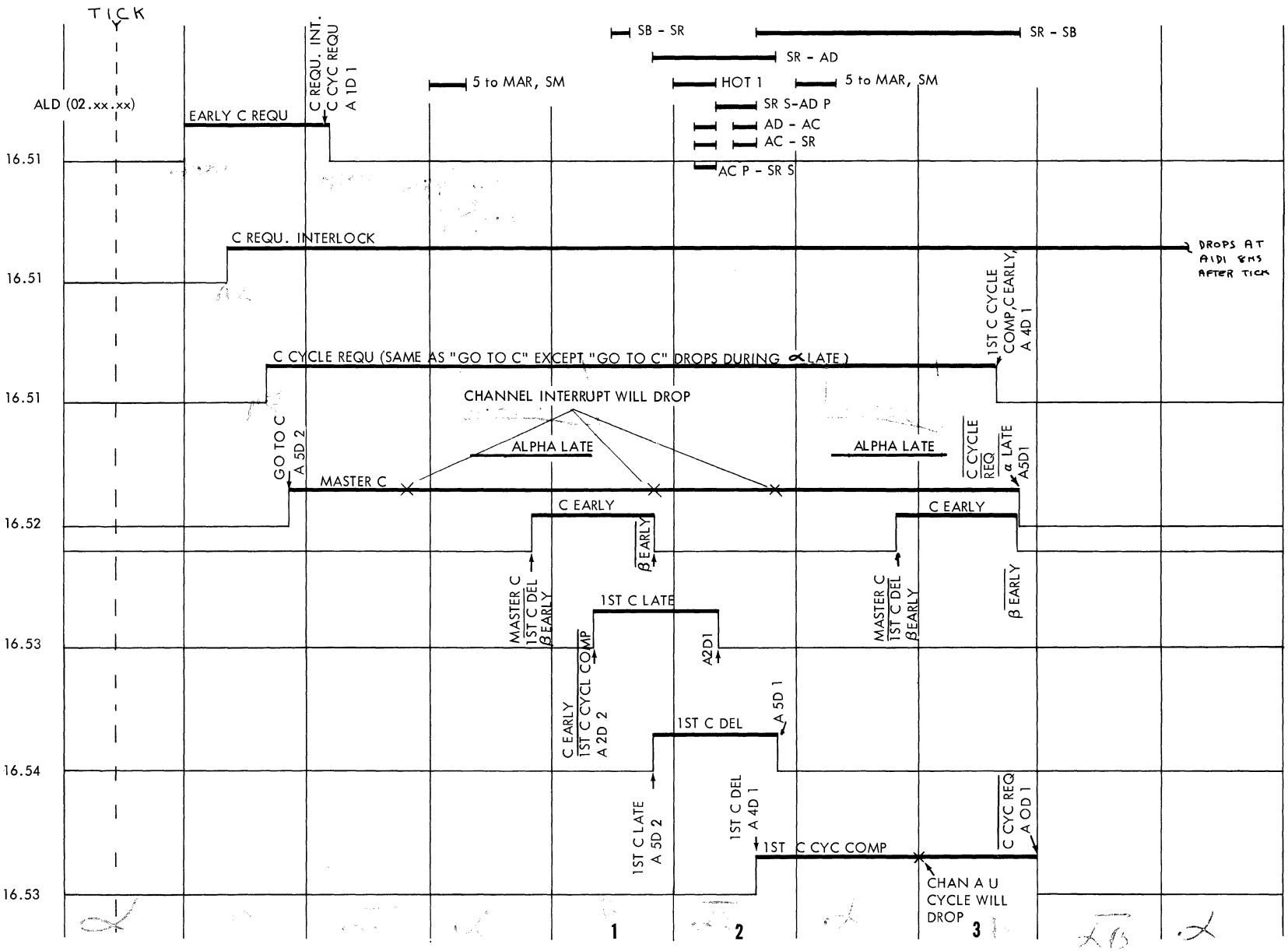


FIGURE 19. C CYCLE TIMING

LOGIC 02.16.51.1 (SIMPLIFIED)

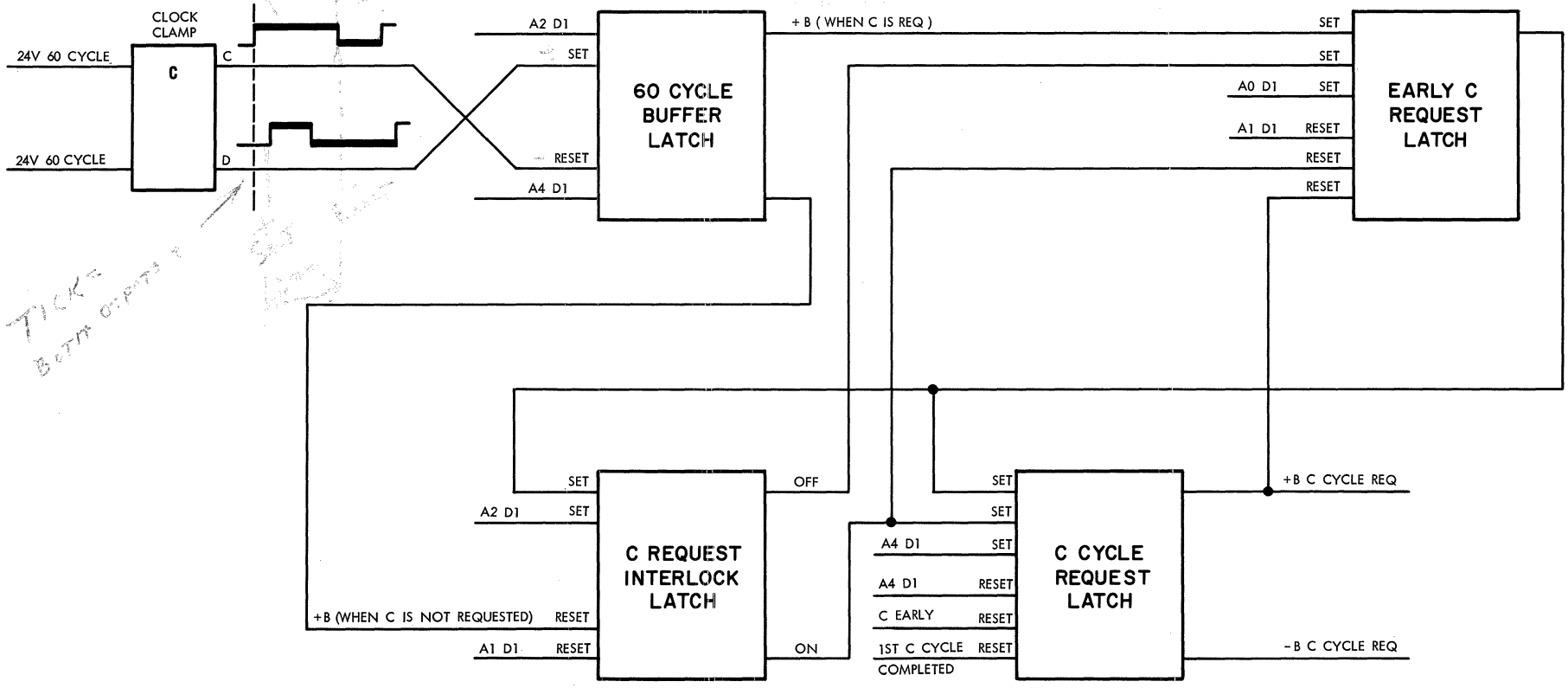


FIGURE 20. C CYCLE REQUEST

The 60-cycle buffer output sets (at A0 D1) the early C request latch, whose output, in turn, sets (at A2 D1) the C request interlock and (at A4 D1) the C cycle request latch.

The C request interlock allows the early C request to be reset and prevents it (the early C) from being set again as long as an IT operation is in process.

The 1st-C-cycle-complete level (at A4 D1 time of the final IT machine cycle) resets the C-cycle-request latch, and the IT operation is over until another C cycle request is automatically initiated (approximately 1/60 second later).

Interval Timer Interruption

The interval timer (IT) can be updated between U cycles. If a U cycle request occurs before the IT operation is finished, the U cycle takes precedence; this priority is necessary because the I/O devices are mechanical (they cannot wait) and must therefore be serviced immediately. The U cycle clears the storage register, thus destroying the IT word placed there during the C cycles.

Note, in Figure 18, that the U cycle breaks into the IT operation by resetting the 1st-C-cycle-complete latch with an E-early level at A0 D1; this action prevents the incremented IT word from being written into memory and also prevents the C-cycle-request from being dropped. Because the C cycle request is still up, the entire IT operation will recur as soon as the U cycle is finished.

Note also (in Figure 18) that the 1st-C-cycle-delayed level causes the AD-AC transfer and the AC-SR transfer at A1 time and again two pulses later at A3 time. Since the IT operation can be sandwiched in between U cycles, the preceding U cycle would have information (a word count and an address count) stored in the accumulator. At A1 time, the 1st-C-cycle-delayed cycle starts to use the accumulator. The accumulator contents (needed for the U cycles) are transferred to the storage register to prevent their loss. At A3 time, the IT operation needs to use the storage register, so its contents are put back in the accumulator and the accumulator goes to the storage register. Though this may sound confusing, it is really an accumulator/storage-register swap to preserve information for the U cycle while the IT is being updated.

INSTRUCTION DECODING

All instruction words read from core storage and placed on the storage bus are just so many random bits and have no particular significance unless they are interpreted at a specific time, in accordance with certain rules, by an interpreting device designed specifically for the application. During I cycle time, all words on the storage bus are treated like instruction words; the rules for interpretation are the various instruction word configurations, and the interpreting device is the CPU hardware in the form of registers and decoders.

Figure 21 shows the routing of bits out of the storage bus for decoding during I cycle time. The AND conditions reveal that not all bits are always decoded. In fact, only two actions occur on every I cycle:

1. Bits 18, 19, and 20 are sent to the tag register.
2. The entire storage bus is transferred broadside to the storage register and then to the adder.

Figure 21 shows the transfer of bits concerned only with decoding. The remaining storage bus bits may or may not be decoded, depending on the particular instruction.

To better point up the varying decoding requirements of instructions, Figure 22 shows three typical instructions as they appear on the storage bus. These three instructions collectively utilize all the CPU decoding facilities. By carefully comparing the instruction decoding needs with Figure 21, it can be seen when and where they will be decoded. Just how the decoding takes place is the prime function of this section and is treated in detail; the <u>when</u> and <u>where</u> are treated only to the extent necessary for understanding the <u>how</u>. Also, decoding channel instructions, which goes further than CPU instruction decoding is described in the channel A instruction manual.

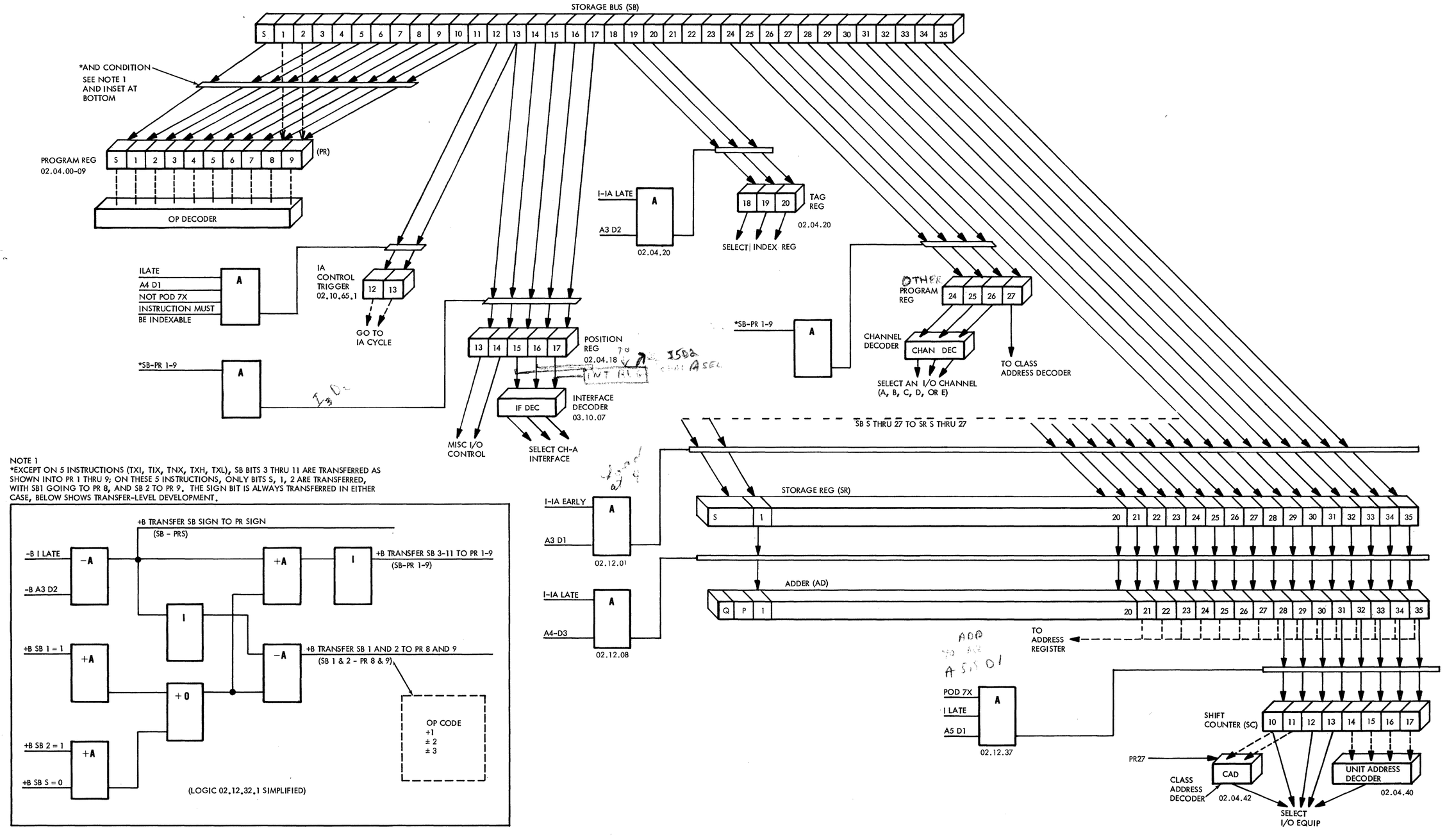
Bits S, 1, And 2

Before any other bit transfers or decoding can be started, SB bits S, 1, and 2 must be decoded (except for the transfer to the tag register, storage register, and adder as already mentioned). When these three bits decode to be +1, $\pm$2, or $\pm$3, the SB S bit goes to the PR S bit position, the SB 1 bit to the PR 8 bit position, and the SB 2 bit to the PR 9 bit position; in this case, no other bits are transferred into the program register and no bits are transferred into the position register, the IA control trigger, or the shift counter. Only five instructions, all transfer type, can cause this action: TIX (Figure 22), TXI, TNX, TXH, and TXL.

The simplified logic for decoding these three bits is shown in the inset in Figure 21. The S bit equal to 1 is minus, and equal to 0 is plus.

Operation Decoding

More than 150 different instructions can be executed by the 7040/7044; for the CPU to know exactly which instruction to perform, bits S through 11 of the instruction word must be decoded (except for TIX, TXI, TNX, TXH, and TXL).



NOTE 1
*EXCEPT ON 5 INSTRUCTIONS (TXI, TIX, TNX, TXH, TXL), SB BITS 3 THRU 11 ARE TRANSFERRED AS SHOWN INTO PR 1 THRU 9; ON THESE 5 INSTRUCTIONS, ONLY BITS S, 1, 2 ARE TRANSFERRED, WITH SB1 GOING TO PR 8, AND SB 2 TO PR 9. THE SIGN BIT IS ALWAYS TRANSFERRED IN EITHER CASE, BELOW SHOWS TRANSFER-LEVEL DEVELOPMENT.

FIGURE 21. INSTRUCTION DECODING

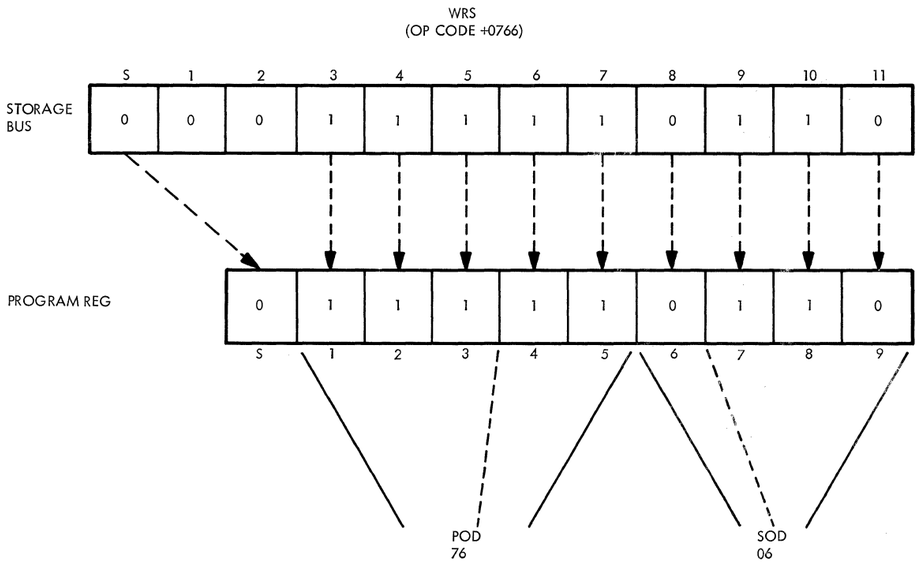
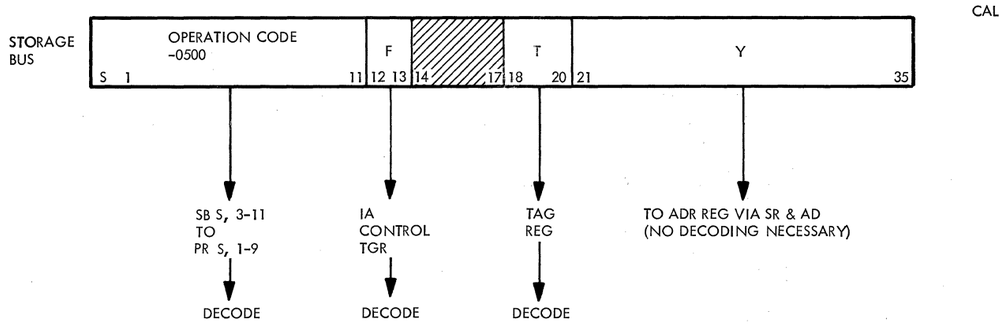
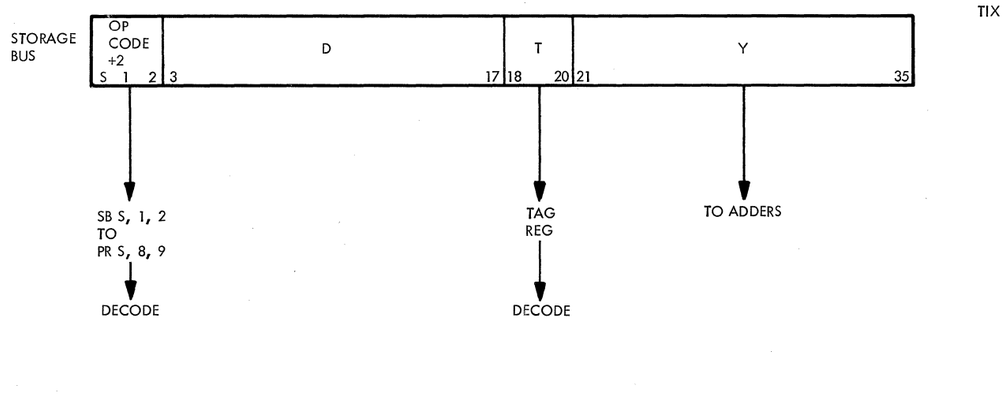


FIGURE 22. INSTRUCTION CONFIGURATIONS





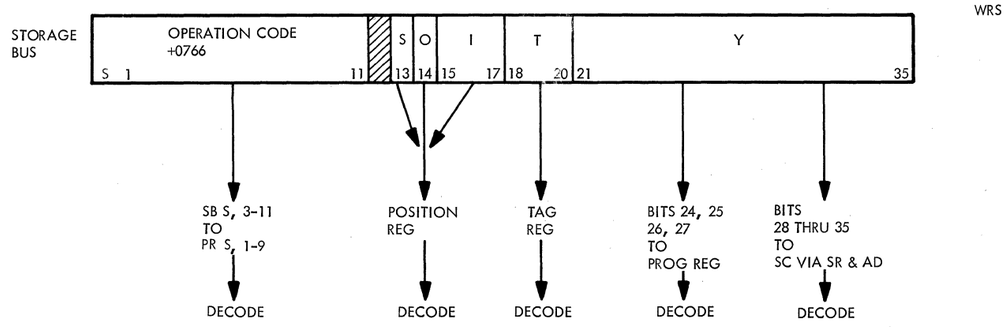


FIGURE 23. BIT LAYOUT

Bits S through 11 of the instruction word constitute the operation code and can usually be completely decoded in the primary operation decoder (POD). One class of instructions is an exception; this class requires a secondary operation decoder (SOD) to identify the instruction. This type is discussed after POD.

Figure 23 shows a WRS operation code as it appears in the storage bus, the program register, and the operation decoders.

POD

Figure 24 shows the primary operation decoder in simplified form. The primary decoding is accomplished in two distinct steps:

1. The outputs of program register positions 1, 2, 3, 4, and 5 are AND'ed together to give 12 outputs, each active for a number of different instructions.
2. The 12 outputs from the first stage are AND'ed in a manner that will give outputs for specific instructions only.

For example, assume program register positions 1 through 5 contain 001 10. This means the following program register outputs are active:

a. $\overline{\text{PR 1}}$
b. $\overline{\text{PR 2}}$
c. PR 3
d. PR 4
e. $\overline{\text{PR 5}}$
f. PR 6 assumed to be zero

The first AND'ing stage has two AND circuits active: POD 1X and POD X4. In the second stage, POD 1X and POD X4 are themselves AND'ed together to give a POD 14; the only instruction with a POD 14 is TOV.

Not all instructions are completely identified by a POD output as is the TOV instruction. For example, POD 40 can be either an ADD or SUB instruction. The only difference between an ADD and an SUB, with respect to the operation code, is the sign bit; so by AND'ing POD 40 with the signals PRS (PR S bit position contains a 1) or $\overline{\text{PRS}}$ (PR S bit position contains a 0), an ADD or SUB is determined. In the case of other instructions, the difference might be some bit other than the sign bit, but the same logic holds true: by AND'ing the second stage POD output with the particular bits involved (they do not necessarily have to be PR bits), the specific instruction is identified. Because of the number involved, not all the AND circuits used to resolve each specific instruction are illustrated. However, the illustrations do provide sufficient detail for a thorough understanding of decoding. If the development of the various POD levels is understood, understanding the breakdown within a particular POD is just a matter of noting the bit differences between the instructions and realizing that these differences are AND'ed with the POD to obtain the specific instruction.

Note that the PR 6 and $\overline{\text{PR 6}}$ signals are used in the second stage to differentiate between X2's and X3's. An example of this use is the generation of POD 52 and POD 53. The PR sign bit and bits 7, 8, and 9 are used to identify specific instructions within a POD in the manner discussed in the preceding paragraph.

SOD

Because POD 76 embraces so many different instructions, a secondary operation decoder is necessary to completely identify a particular POD 76 instruction. Figure 25 shows a simplified version of the SOD.

The SOD is much like the POD, the main difference being the PR bits used. In the SOD, PR bits 6, 7, 8, and 9 are AND'ed with a POD 76 to obtain the SOD levels. As with the POD, some SOD levels completely identify a given POD 76 instruction, while others must be further AND'ed to realize identification. Because of the many circuits involved, only the basic SOD levels are illustrated.

Flag Bit Decoding

Bits 12 and 13 of the instruction word are flag bits and are used to specify indirect addressing. These two bits are flag bits only in those instructions that can be indirectly addressed; otherwise, they may serve other purposes and are not decoded for indirect addressing.

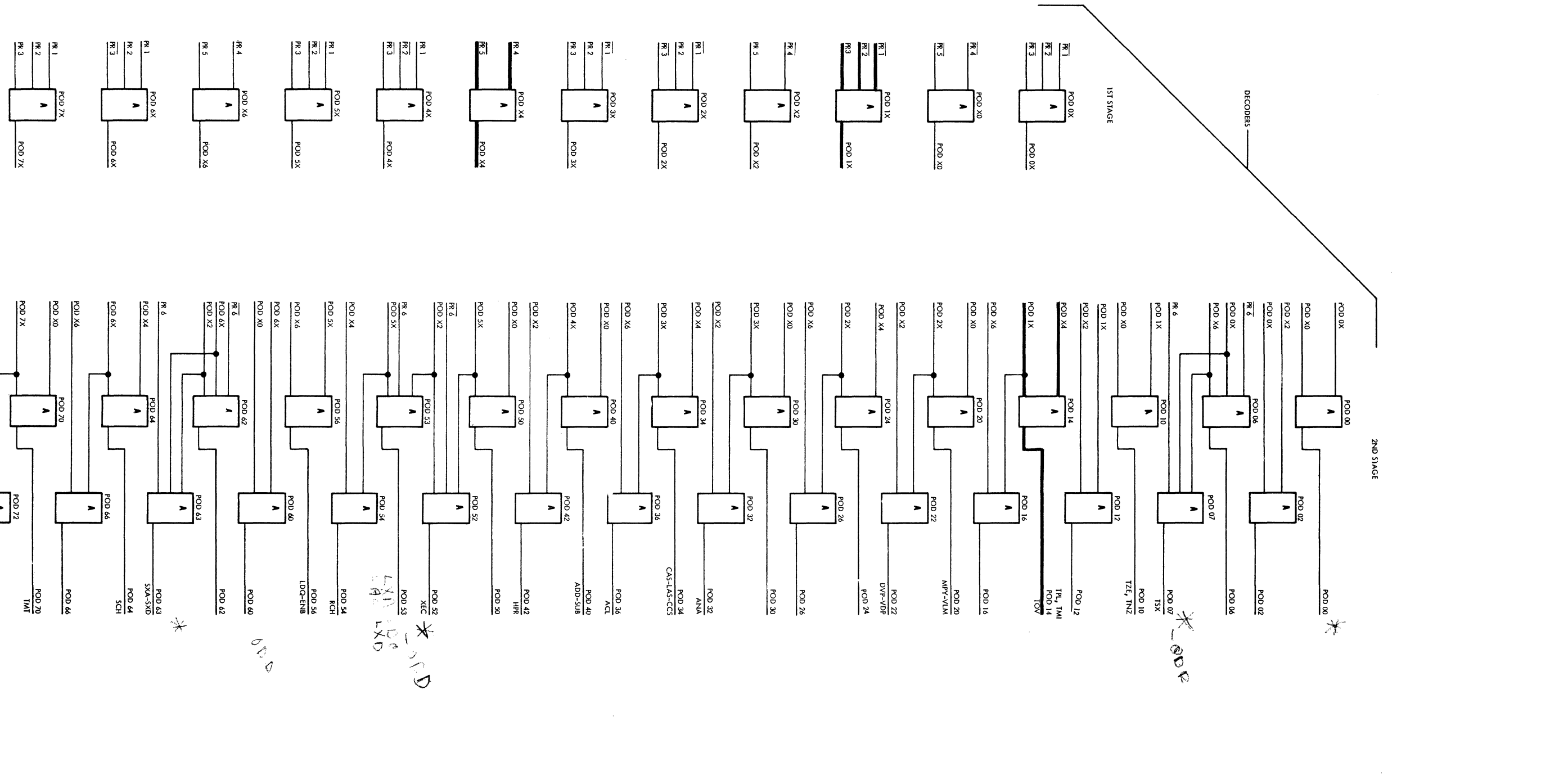
Flag bit decoding is simple (Figure 26). If bits 12 and 13 of the instruction word are both 1's, and if the instruction is indexable and not a POD 7X instruction, a PRE IA trigger is initiated at I late time by an A4 D1. The PRE IA trigger ultimately causes an IA cycle (a type of E cycle), which is necessary for indirect addressing to be initiated.

Tag Bit Decoding

The tag bits are 18, 19, and 20 of every instruction word, and they denote which index register (if any) may be used. (Some instructions do not employ XR outputs, although the bits select an XR.) There are three index registers (A, B, and C), and they are usually used for address modification. Figure 27 shows the tag bit decoding and index register selection.

If instruction word bit 18 is a 1, the tag C latch is set at I late time by an A3 D2. The tag C latch output is AND'ed with the output of each latch position on index register C. Thus, when bit 18 is

FIGURE 24. PRIMARY OPERATION DECODING



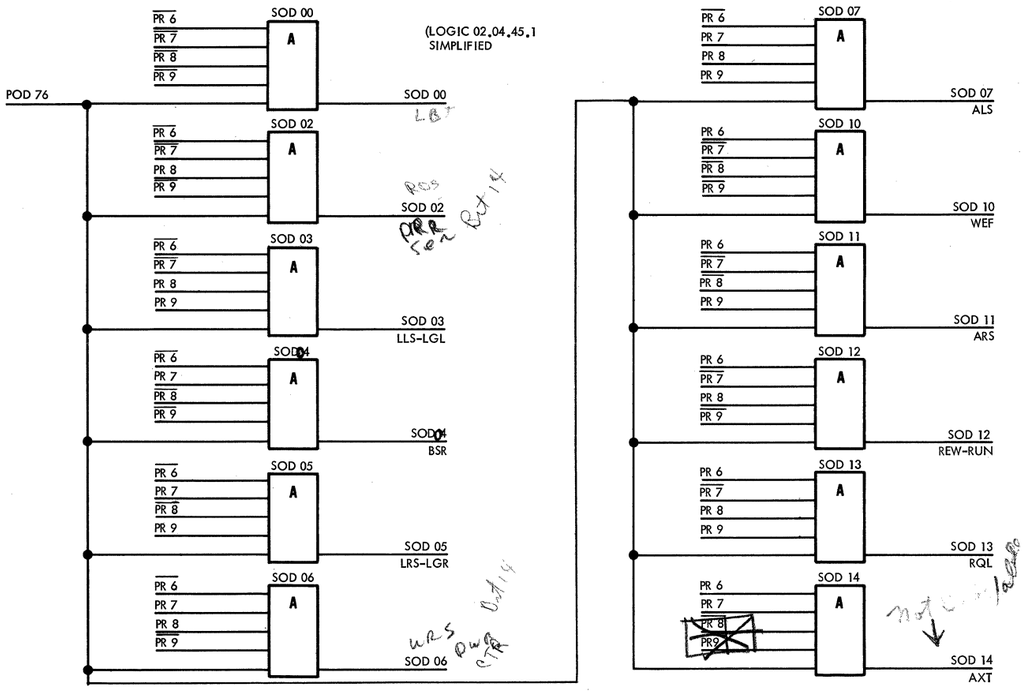


FIGURE 25. SECONDARY OPERATION DECODING

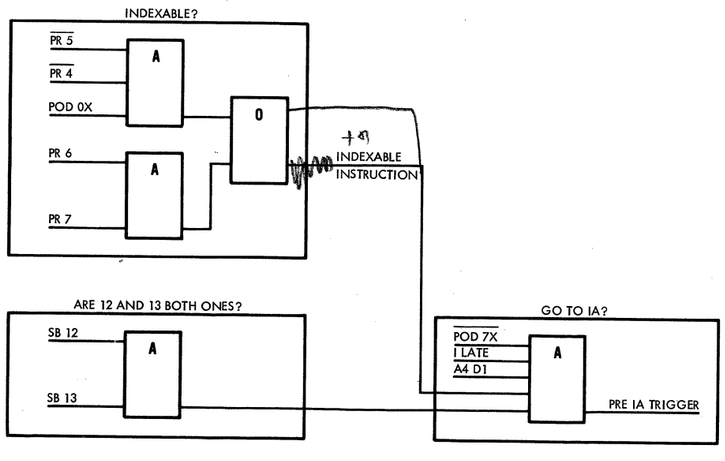


FIGURE 26. DECODE FOR INDIRECT ADDRESSING

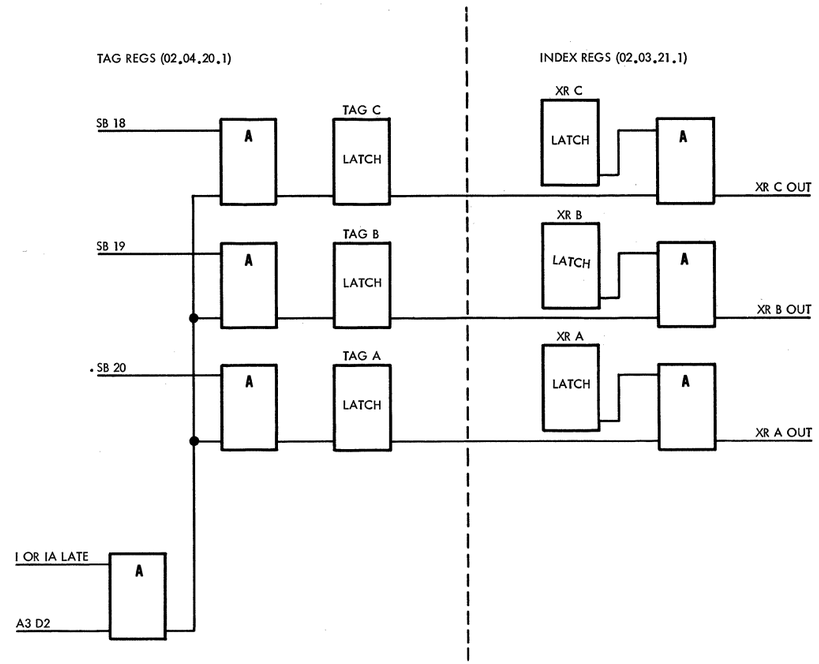


FIGURE 27. TAG DECODING

a 1, tag C is set and the index register C contents are gated out. Bit 19 sets tag B and controls index register B; bit 20 sets tag A and controls index register A. If none of the tag bits is a 1, it indicates that an index register is not required for that particular instruction, and, therefore, no index register is selected.

ADDER

The 37-bit binary adder is used in performing all binary arithmetic and address modification through indexing. Several inputs and outputs are therefore necessary for these operations. Adder position 33 and the inputs and outputs are shown in Figure 28 as a typical example. Inputs to block 4G of the adder come from the index registers (A, B, or C) or from the accumulator 33 position. Inputs to block 4H of the adder may be from storage register 33 or storage register 15. The combinations that can be added together are:

1. Any index register 33 (XR A, B, or C) to storage register 33 or storage register 15.
2. Accumulator 33 to storage register 33 or storage register 15.

The adder position 33 output may be routed to one of five places (Figure 28):

1. Accumulator 33
2. Accumulator 34
3. Index register X33 (XRX), which is then sent to XRA 33, XRB 33, or XRC 33.
4. Address register 33
5. Shift counter 15

The adder block shown in Figure 28 is identical for all adder positions. The +OR (4G) is considered the A input, and +OR (4H) the B input. The outputs from blocks 4G and 4H feed a -OR (3G) and a -AND (2I). The -OR (3G) output represents an A or B (A + B) condition, and the output from -AND (2I) represents an A and B (AB or A . B) combination. Ones and 0's may be substituted for A and B as follows:

1. A by 1
2. B by 1
3. Not A, or $\overline{A}$ by 0
4. Not B, or $\overline{B}$ by 0

An AB condition causes a generate (G) level and a propagate (P) level (Figure 28). An AB (11), $\overline{A}$B (01), or A$\overline{B}$ (10) input condition causes a propagate (P) level. These outputs are sent to the lookahead (LA) circuits and to the carry generator. The important concepts to remember about the adder block are:

1. All adder blocks are identical in operation.
2. The following conditions cause generate (G) and propagate (P) outputs: G = AB Input, and P = AB or $\overline{A}$B or A$\overline{B}$ inputs.
3. The generate and propagate levels are sent to lookahead and provide adder carries.
4. A positive output from the adder indicates a sum of 1.

Lookahead Adder

The basic principle behind connecting individual adders to make an adder unit is to take the carry-out of one adder block and connect to the carry-in of the next high-order position adder. If all adder positions contained a 1 and a 1 were added to the low-order position, a carry would have to ripple through the adder unit from the low-order position to the high-order position. As the carry ripples through the adders, each adder block introduces additional delay. The lookahead adder contains additional logic circuits that provide a means of predicting how many positions would be affected by a ripple carry and injects the carry into all affected positions almost simultaneously.

Lookahead Propagate and Generate Outputs

The 7040/7044 adder contains an adder block for each adder position, a carry generator for each position, and three levels of lookahead. Figure 29 is a block diagram of the lookahead adder. The carry generators provide the carries into the affected positions. The first level of lookahead examines the adder outputs in groups of 4 and sends the generate and propagate outputs to the second and third lookahead circuits (Figure 29). The second and third levels of lookahead determine the carry into various groups in the adder. The lookahead outputs and carries from lookahead are available almost immediately after data is sent to the adders, thereby eliminating the need for the carries to ripple through the complete adder.

As shown in Figure 29, the generate and propagate levels from adder blocks 1-32 feed the first level of lookahead. The outputs are grouped in groups of four adder blocks, and the first lookahead level propagate and generate outputs are sent to the second and third lookahead levels. The second lookahead level provides carries into adder carry generators for adder positions 20, 24, and 28; the third lookahead level provides carries for adder positions P, 4, 8, 12, and 16. The carry generators provide the carries for the remaining adder positions.

Figure 30 shows how the propagate and generate outputs and carries are caused from the three levels of lookahead. The following examples cause a carry 1 into adder position 32 (Figure 30):

Carry into 32	33	34	35	
	1	0	0	
	1	0	0	
1	0	0	0	No carry into 35
	0	1	0	
	1	1	0	No carry into 35
1	0	0	0	
	0	0	1	
	1	1	1	No carry into 35
1	0	0	0	
	0	0	0	
	1	1	1	Carry into 35
1	0	0	0	

More than one AND may be conditioned to cause a 33 carry into position 32.

	1	0	0	
	1	1	1	Carry into 35
1	1	0	0	

The above combination conditions two AND's, which provide a carry into adder 32. To illustrate this statement, apply the following expression to Figure 30: Hot 1 · 33P · 34P · 35P or 33G. If all 1's are added together, resulting in a carry into position 35, all four AND's for the 33 carry are conditioned. The remaining groups of the first level of lookahead provide no carries. These outputs are sent to the second and third levels of lookahead.

The (29-32) G output is caused similarly to the (33-35) G. The (29-32) P output shows that the propagate outputs from adder positions 29, 30, 31, and 32 must be present to the + AND (Figure 30). The remaining first-level lookahead outputs are caused the same as the (29-32) P and (29-32) G. Note that in lookahead outputs it is possible to have a generate output with no propagate output.

The second lookahead level receives inputs from the first lookahead level. As a typical example, the 29 carry output is generated if one of two or both conditions exist: a (29-32) G or (33-35) G and (29-32) P. This is similar to the first lookahead level except that groups from the first level are examined instead of individual adder outputs to determine carries.

Since floating-point arithmetic requires dividing the accumulator (Q, P, 1-8, and 9-35), the adder 9 carry may be considered a special case for carry control. The second lookahead level combines outputs from the first lookahead level to cause the (9-20) G and (9-20) P outputs. If the instruction is not a floating-point operation, the second and third lookahead levels operate normally. If a floating-point instruction is being executed, a 1 to adder 8 provides the 9 carry. The 9 carry from adder positions 9-35 is inhibited from affecting the 8 position for all floating-point instructions (Figure 30).

The third lookahead level accepts outputs from the first and second levels to provide carries to high-order bits 16, 12, 8, 4, and P (Figure 30).

Adder Operation

To obtain the correct sum, the adder must consider every possible combination that may appear at the inputs. The 7040/7044 adder may be divided into sections containing four adder blocks in each section. Figure 31 shows a 4-bit adder for adder positions 29, 30, 31, and 32. Operation of positions 1-4, 5-8, 9-12, 13-16, 17-20, 21-24, and 25-28 is the same as for adder positions 29-32 except for the differences in the generation of the carries from lookahead. The carries from lookahead can be determined from Figure 30. Adder positions 33-35 are a 3-bit adder with the carry-in from an external source. Positions P and Q are extensions of the adder with the carry into position P from the third level of lookahead. As shown in Figure 31, adder position 29 receives a not 29 carry from the second lookahead level. See Figure 30 for the levels that cause the 29 carry.

Since an adder block cannot distinguish between 0 plus 1 and 1 plus 0, three combinations are possible at the inputs (00, 10 or 01, and 11). The possibilities double when a carry-in condition exists (six total possibilities). In a 4-bit adder, 272 combinations are possible. This includes a carry-in and a no-carry-in to the low-order bit. Therefore, only a few examples of adder operation are explained in detail here.

Example 1:

0000
0000
0000 No carry-in

When adding 0's with no carry into position 32, all the AND's in the adder and carry generator are de-conditioned. The sum output from each adder position is negative, which represents 0's.

Example 2:

0000
0000
0000 Carry into position 32

If a carry-in (CI) to position 32 occurs, AND 2G in adder position 32 is conditioned, resulting in a positive output from inverter IF. A sum of 0001 is sent to the receiving register (Figure 31).

Example 3:

1010
0101
1111 No carry-in

Assume that the above numbers are added together with no carry-in to position 32. In adder position 32, AND 2F is conditioned and a 1 is sent to the receiving register (Figure 31). One input to AND 2F is conditioned by a 01 or 10 ($\overline{A} \cdot B$ or $A \cdot \overline{B}$) input to the adder.

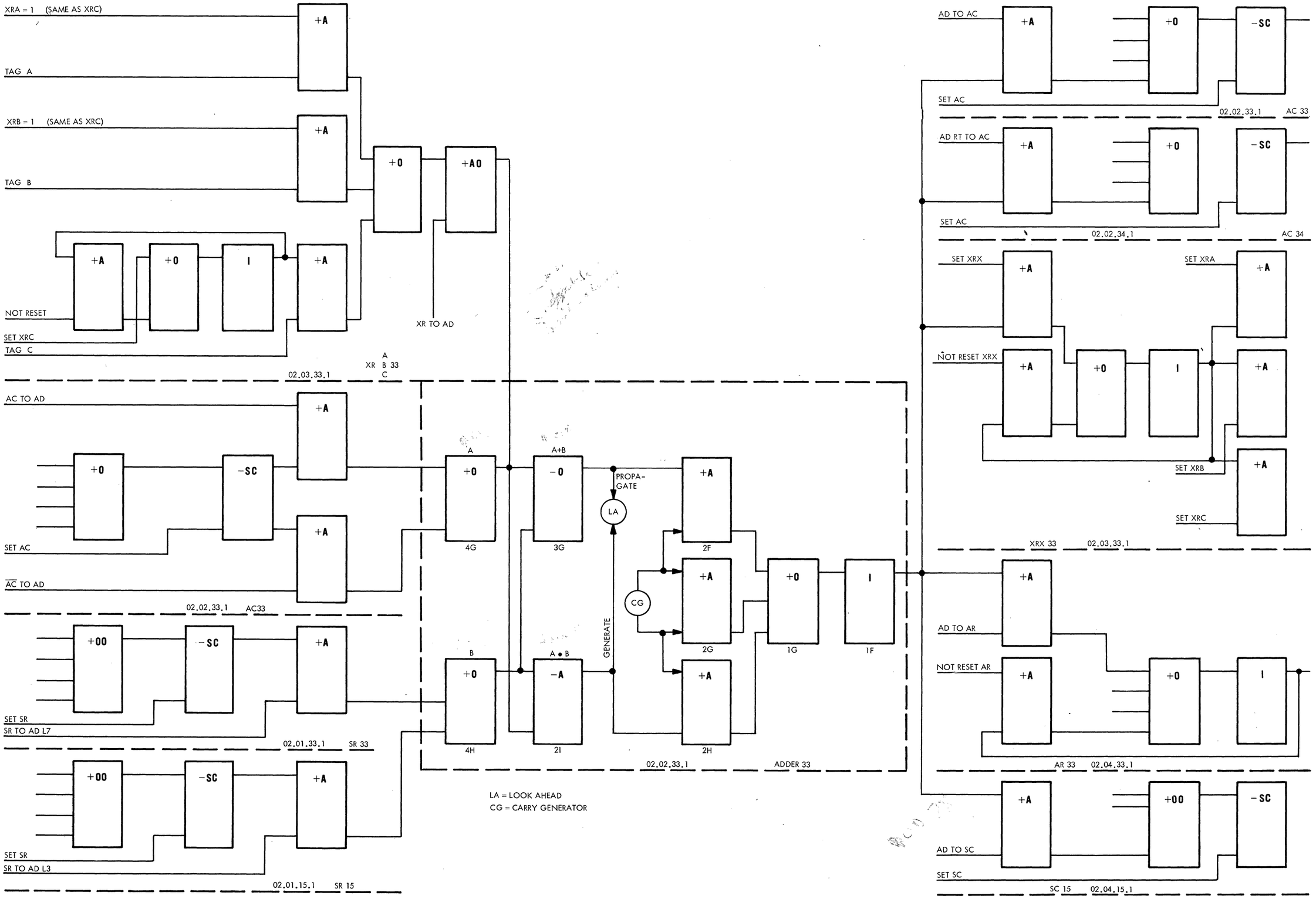


FIGURE 28. ADDER INPUTS AND OUTPUTS

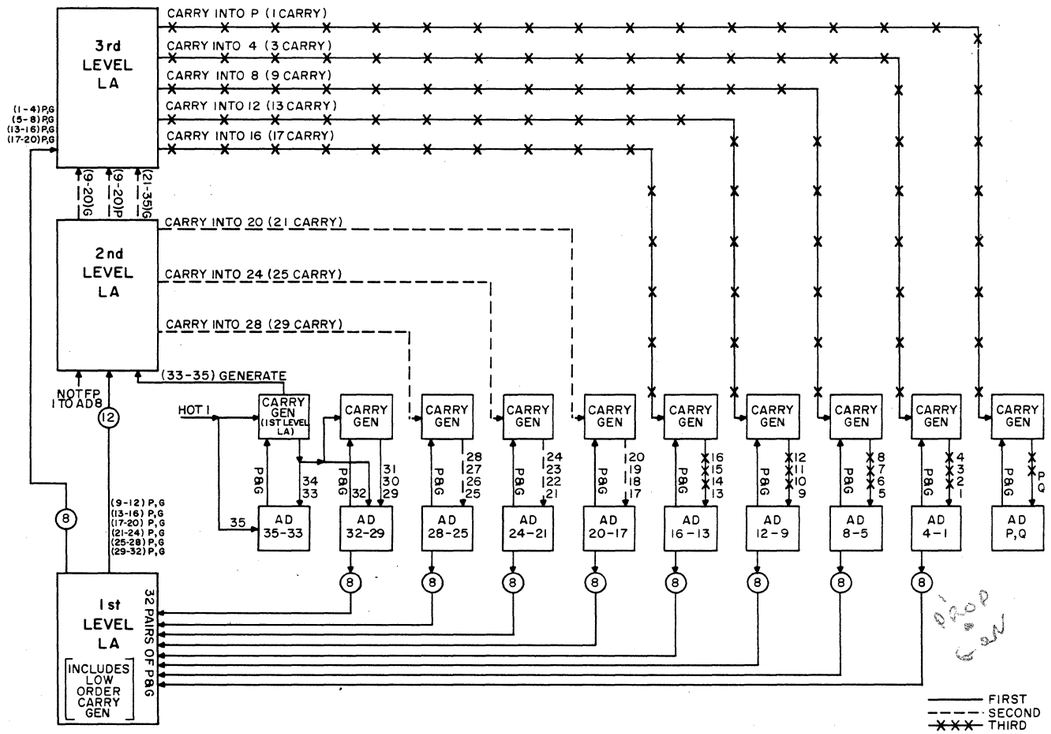
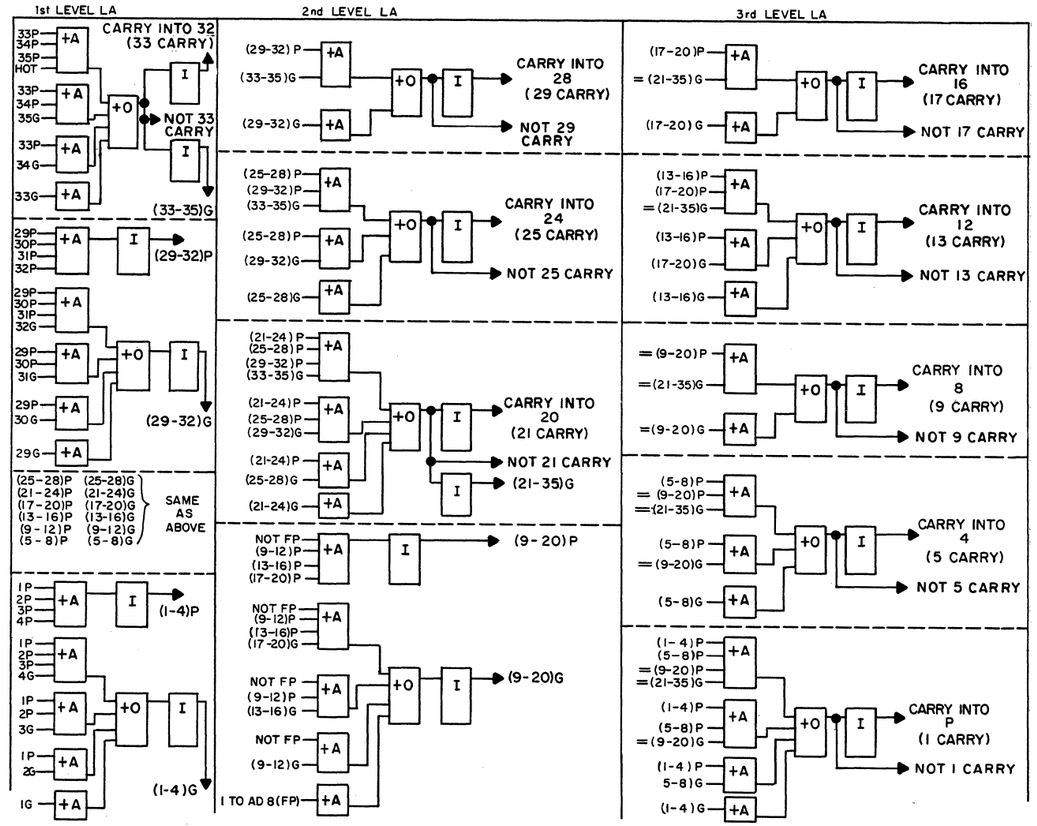


FIGURE 29. ADDER LOOKAHEAD BLOCK DIAGRAM



P = PROPAGATE , G = GENERATE , HOT = 1 TO AD 35, FP = FLOATING POINT, = 2nd LEVEL LA OUTPUTS

FIGURE 30. ADDER LOOKAHEAD CIRCUITS

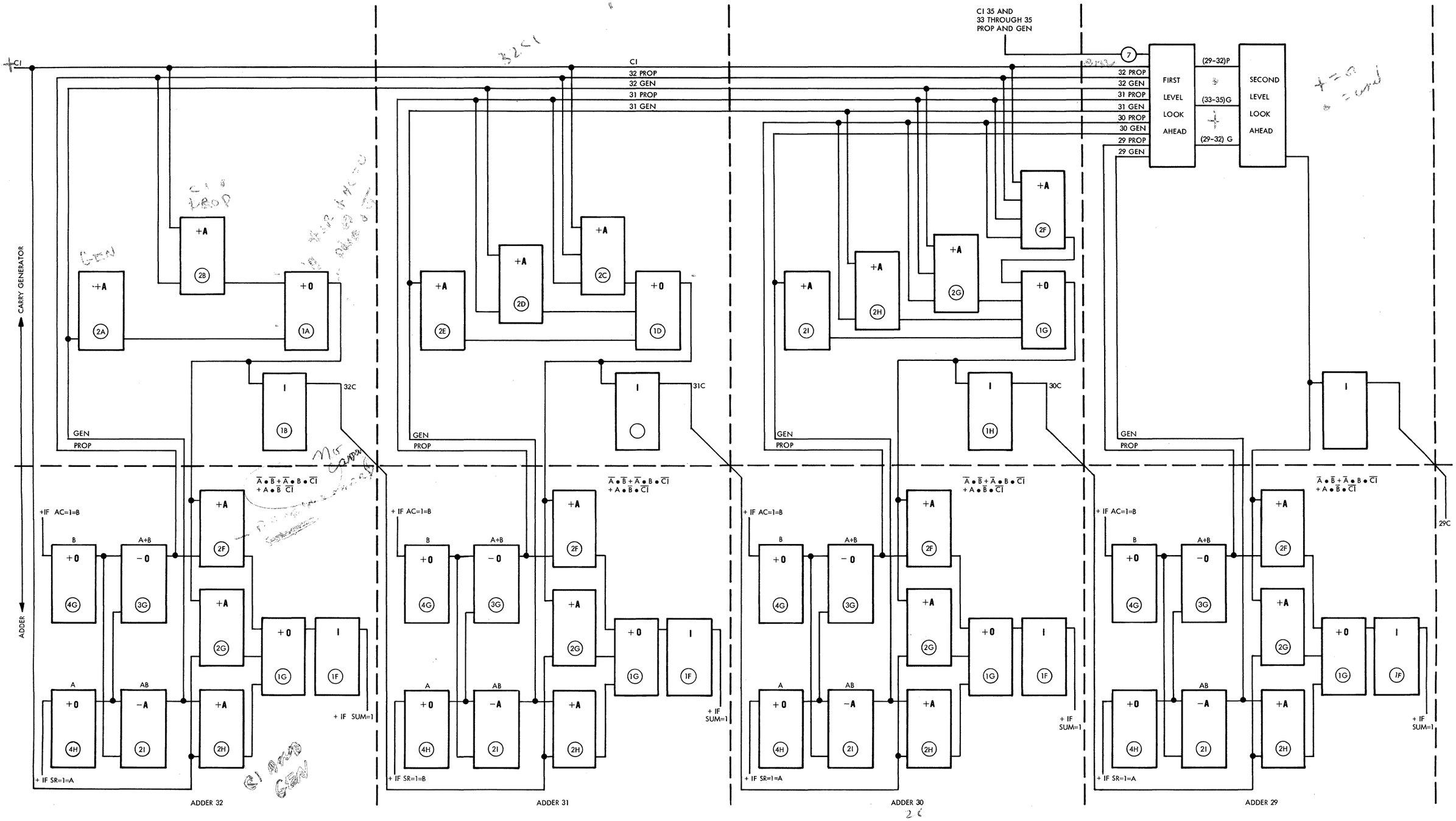


FIGURE 31. 4-BIT ADDER OPERATION (POSITIONS 29, 30, 31, AND 32)

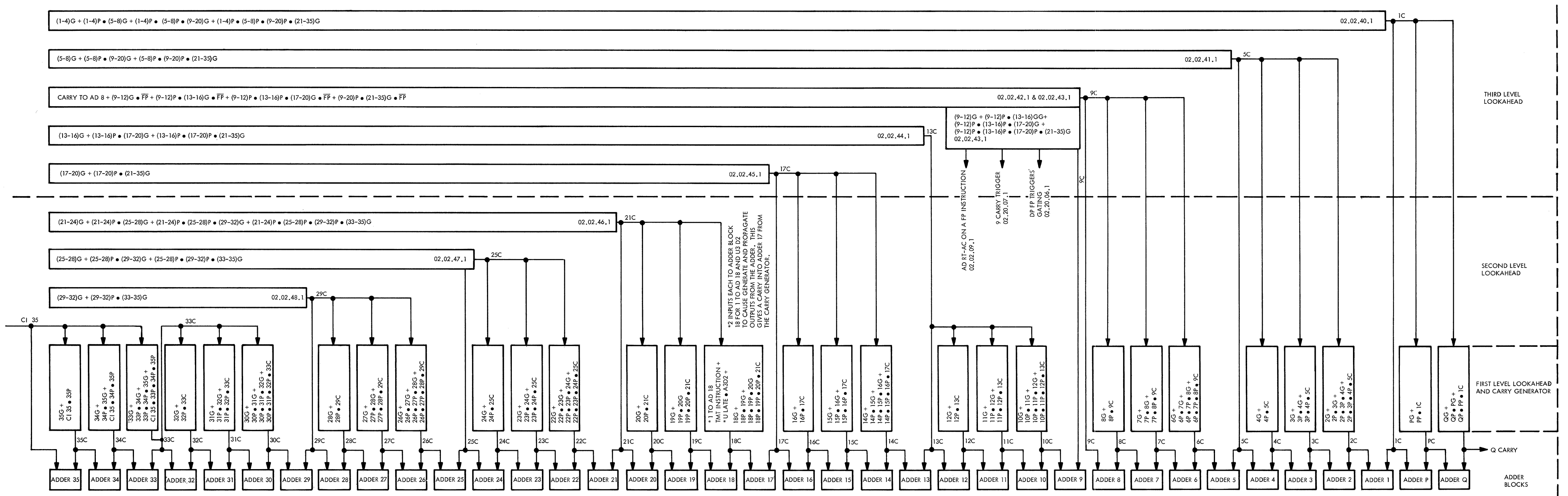


FIGURE 32. ADDER CARRY GENERATION

Both inputs to OR 1A in the carry generator are negative. The positive output of OR 1A conditions the second input of AND 2F. The +OR's (1D and 1G) in the carry generator outputs from positions 31 and 30, respectively, are positive. AND 2F in position 31 and AND 2F in position 30 are conditioned, and a sum of 1 is generated for both positions. In adder position 29, the output from the second lookahead level is positive, and an AND 2F condition is met, causing a sum of 1 for position 29.

Example 4:

```
      1010
      0101
1 Carry 0000  Carry into position 32
```

Assume that a carry-in occurs and the same two values in Example 3 are added together. A propagate output appears at all the inputs of the carry generator. In carry generator 32, AND 2B is conditioned (Figure 31). The negative output of OR 1A is inverted by inverter 1B, causing a 32 carry (32C) into position 31. Since the output of OR 1A is negative, AND's 2F and 2G (adder 32) are deconditioned and a zero sum results. The propagate levels from positions 32 and 31 and the CI-to-AND-2C (position 31) level apply a positive input to OR 1D, causing a 31 carry (31C) into position 30 and deconditioning AND's 2F and 2G in adder position 31. In position 30, AND 2F in the carry generator is conditioned, causing a carry into position 29. Since a carry into position 32 occurred, the (33-35) G level to the input of the second lookahead level is present and the (29-32) P output is generated (Figure 30). These two conditions cause a 29 carry into adder position 28. The output from the second lookahead level is negative, deconditioning AND's 2F and 2G in adder position 29. The sum is 0000 and a carry into adder position 28.

Example 5:

```
      1111
      1111
1 Carry 1110  No carry-in
```

In this example, the generate and propagate outputs appear at the inputs of all the carry generators for each adder position (Figure 31). With no carry into position 32, AND 2A in the carry generator is conditioned. The negative output of OR 1A is inverted (1B), causing a 32 carry into adder position 31. Adder position 32 AND's (2F, 2G, and 2H) are deconditioned, resulting in a sum of zero for position 32.

Carry generator position 31 and AND's 2E and 2D are conditioned by the 31G (AND 2E) and 32G and 31P (AND 2D) levels. This causes a carry into 30. The 32C and the A · B (11) inputs to the adder, via OR's 4G and 4H, condition AND 2H in adder position 31 (which results in a sum of 1).

In carry generator position 30, AND's 2I, 2H, and 2G are conditioned, causing a carry into adder position 29. AND 2H in adder position 30 is conditioned by the 31C and an A · B (11) input to the adder. The sum for position 29 is equal to 1. The second level of lookahead provides a 29 carry into adder position 28. Figure 30 shows how this carry is generated.

Example 6:

```
      1111
      1111
1 Carry 1111  Carry into position 32
```

Assume a carry into adder position 32 and a 1 plus 1 input to adder position 32. AND 2H is conditioned, causing a sum of 1 (Figure 31). The same AND's as in Example 5 are conditioned in all positions, including AND 2F in the carry generator 30 position. The following AND's are conditioned:

Position	Carry Generator	Adder
32	2A, 2B	2H
31	2C, 2D, 2E	2H
30	2F, 2G, 2H, 2I	2H
29	Second level of lookahead output provides carry into adder 28.	2H

Summary

Figure 32 summarizes the conditions that cause a carry to the next higher position for all 37 bits of the 7040/7044 adder. If the propagate and generate levels from the adder are established, the carries can be determined. The logic involved in this figure is 02.02.40.1 through 02.02.50.1. Boolean Algebra symbols are used: a plus sign (+) indicates an OR condition, and a dot (·) indicates an AND condition.

As shown in Figure 32, the 18 carry may be caused by a condition external to the adder. The Transmit and Reset and Load Channel A are the two instructions that require a carry into adder 17.

The Transmit (TMT) instruction reads one area of memory and stores the data in another section of memory. Accumulator bits 3-17 are the from address, and accumulator bits 21-35 are the to address. Accumulator bits 3-17 and 21-35 must be incremented by 1 after each word is read and stored. When the shift counter goes to 0, the instruction ends. A hot 1 into position 35 increments accumulator positions 21-35. A carry 1 into accumulator 17 is necessary to increment 3-17 by 1. To cause a carry 1 into adder position 17, two inputs to adder 18 are provided which represents a 1 plus 1 condition. These two inputs cause propagate and generate outputs to be sent to the carry generator, resulting in a carry 1 into adder 17 (logic 02.02.18.1). Figure 32 shows the conditions that cause an 18 carry into adder 17.

When a Reset and Load Channel A (RCHA) instruction is executed, the channel command word is stored in the accumulator. Accumulator bits 3-17 contain the word count, and 21-35 contain the starting address. Positions 21-35 are incremented by 1 when a hot 1 is sent to adder 35. The complement of the accumulator is sent to the adder, and a 1 is sent to adder 18 at U2 D3 time, causing a carry 1 into adder 17 and decreasing the word count by 1. Two inputs to the adder are provided for an RCHA instruction, causing generate and propagate outputs to appear at the carry generator inputs. The carry 1 into adder 17 is therefore realized (logic 02.02.18.1).

The following concepts summarize the operation of the adder:

1. Generate output: An A · B (11) input to the adder block. This provides a carry to the next higher-order position.
2. When grouping generate outputs, a carry-out from the highest order position is indicated; i.e., (25-28) G indicates a 25C.
3. From the first, second, and third lookahead levels, it is possible to have a generate output without a propagate output.
4. Propagate: Any output from the adder other than an $\overline{A} \cdot \overline{B}$ (00) at the inputs or groups of propagate outputs are AND'ed together.
5. A generate output from an individual adder block causes a propagate output.
6. First, second, and third lookahead levels cause carries into certain positions in the adder (Figure 30).

The concepts of the adder have been presented in the following order:

a. Inputs and outputs from adder position 33.
b. Generate and propagate output conditions from the adder.
c. Block diagram analysis of the adder.
d. Conditions that cause first, second, and third lookahead generate and propagate outputs and carries.
e. 4-bit adder analysis.

ADDRESSING

All information used in the 7040 and 7044 data processing systems must be placed in core storage to make it accessible for processing. The manner in which information is entered in and removed from core storage must be orderly to prevent confusion and erroneous results. Consequently, each core storage register is assigned a number which serves as its address in the core array. References to memory are made with these addresses. Thus, an item of information is specified in the machine via a core storage address. This section provides a detailed analysis of addressing as applicable to the 7040 and 7044 CPU operations.

Addressing Core Storage

Addressing is the process of referencing a specific core storage location. The reason for referencing a specific core storage location depends on the user (CPU, channel A, or an overlapped channel). When the CPU is the user, memory is referenced for instructions, operands, or storing purposes. When either channel A or an overlapped channel is the user, a memory reference is made either to transfer the contents of the referenced location to an I-O device (write operation) or to transfer data from an I-O device to the referenced memory location (read operation). In each of these cases, however, the end result of addressing is the transfer of an effective address to the memory address decoding circuits.

The memory address decoding circuits are activated when the MAR is loaded. Referencing a core address therefore involves placing the effective address in MAR. Overlap channels do this directly by transferring the contents of the channel address register to the MAR. All CPU operations and data channel A operations load the MAR from the address register (AR).

The most basic form of addressing is illustrated by the instruction counter. Initially, the instruction counter contains some value; for example, 1000_8. This value is transferred from the instruction counter to the AR, and from the AR to the MAR, as the address of the desired instruction. Note that no address modification is involved in this application of addressing. When the instruction contained in location 1000_8 is received in the CPU and decoded, the instruction counter is incremented by 1 to the value 1001_8. This value represents the address of the next instruction to be executed by the CPU.

Similar action occurs when an overlapped channel or data channel A represents the user. With an overlapped channel, the channel address register acts exactly like the instruction counter; that is, the contents of the channel address register form the address of the desired core location. When this core location is loaded during a read operation or transferred to the I-O device presently in use during a write operation, the channel address register is incremented by 1 to identify the next core location to reference. A channel A operation uses accumulator bits 21-35 in an identical manner.

From the above illustrations, it can be concluded that referencing core storage via the instruction counter, the channel address register, or the accumulator represents a basic application of the addressing concept. In addition, the effective action is identical in each case.

When relating the concept of addressing to CPU operations, it is generally associated with the instruction word. In the 7040-7044, instruction word bits

21-35 form the address field. The value contained in these bits is the base address. Instruction word bits 18-20 form the tag field, which serves to specify address modification by indexing when dealing with instructions that do not apply to the index registers. If the tag field contains value other than 0, address modification is specified. Actually, the tag field value identifies an index register whose contents are to be used for the address modification. The contents of the specified index register are subtracted from the base address to obtain the effective address. If no address modification by indexing is specified, the instruction word base address becomes the effective address. The effective address, in any case, is loaded into the AR and goes from the AR to the MAR as the address of the desired operand.

The 7040-7044 also employs indirect addressing. Instruction word bits 12 and 13 are used to specify this type of addressing. These bits form the flag (F) field, and, when they contain a value of 11_2, specify indirect addressing. In this case, the effective address identifies a core location whose contents contain the effective address of the desired operand.

Since the actions associated with CPU addressing occur during instruction decoding and during the action taken on an operand (in indirect addressing), CPU addressing is concerned only with I and E cycles. The reasons for resolving it to these cycles are as follows:

1. B cycles are not at all concerned with the CPU.
2. C cycles force an address directly into MAR.
3. L cycles do not reference memory.
4. U cycles initiate E cycles and so, in effect, are the same as E cycles.
5. IA cycles are just a type of E cycle.
6. IC memory referencing pertains to the sequence of instruction execution rather than to addressing as the result of instruction decoding.

Figure 33 shows the general flow of information pertinent to CPU addressing. The key to this action is knowing what is loaded into the AR and when.

Instruction Counter

The 15-position (21-35) instruction counter (IC) keeps track of the program currently being executed by indicating to the CPU the address of the next instruction to be performed. Figure 34 shows the three least significant positions of the instruction counter to illustrate how it is stepped; the various conditions that will step the instruction counter are also shown. Normally, the instruction counter is stepped once during I cycle time (block 1 of Figure 34) but, under test conditions or during special instructions, can be stepped during E or L cycle time (blocks 2, 3, and 4 of Figure 34).

Each step-IC level is fed to all IC positions (block 5 of Figure 35) as is the set-IC level. Whenever present, these levels cause the least significant position (IC 35) to change its state; i.e., if IC 35 is a 1 (as shown in block 5), it changes to a 0 (-SC in-phase output will be +B; out-of-phase, -B) when the set-IC level comes in, and, if it is a 0, it changes to a 1 when the combination of set-IC and step-IC is fed to it. Instruction counter 34 changes state with every other set and/or step input; IC 33, with every fourth; IC 32 with every eighth, etc.

The OR input to the -SC of any one of the IC positions must be -B for the set level to cause the -SC to switch to, or be maintained at, a 1 state. For the OR to feed a -B to the -SC, one of the OR's input AND circuits must be conditioned; if one of the AND's is not conditioned, the -SC switches to, or remains at, a 0 state. Each IC position is AND'ed so that the only time a given position switches to a 1 state is when it is already at a 0 state and all lesser significant positions are 1's. At the time a given position switches to a 1, all lower positions switch to 0's; the given position then remains a 1 until all lower positions become all 1's again. When lower positions are all 1's, and the given position is also a 1, the given position switches to a zero when it receives the next set level. Simply, if IC 33 is a 1, it does not switch to a 0 unless both IC 34 and IC 35 are also in a 1 state; if IC 33 is a 0, it does not switch to a 1 unless both IC 34 and IC 35 are 1's. Block 5 of Figure 34 shows how the circuits are connected to accomplish this.

Address Register

The address register contains 15 latches (21-35) which may be set from the corresponding bit position of either the adder, the instruction counter, or the address keys on the operator's console. Figure 35 shows one address register position (all positions are the same) and the conditions that set and/or reset it. The most important items in Figure 35 are the conditions involved in determining which information source is transferred into the address register. A study of the figure will reveal that:

1. The IC to AR transfer takes place in bringing out the next sequential instruction, (and for manual conditions).
2. The AD to AR transfer takes place when data is to be fetched or stored, and when a 1-cycle transfer instruction has been executed in which the transfer conditions have been met.

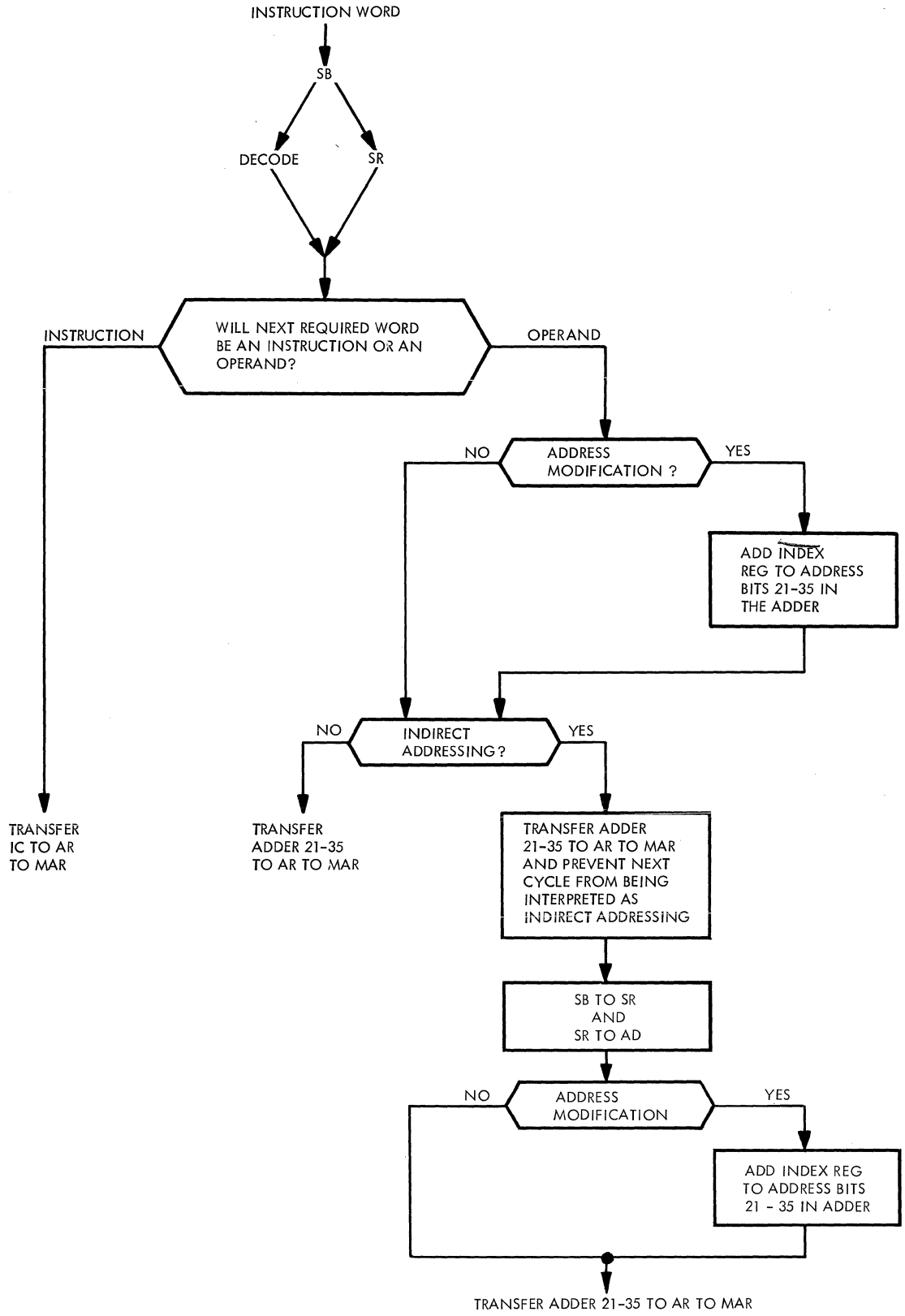


FIGURE 33. ADDRESSING

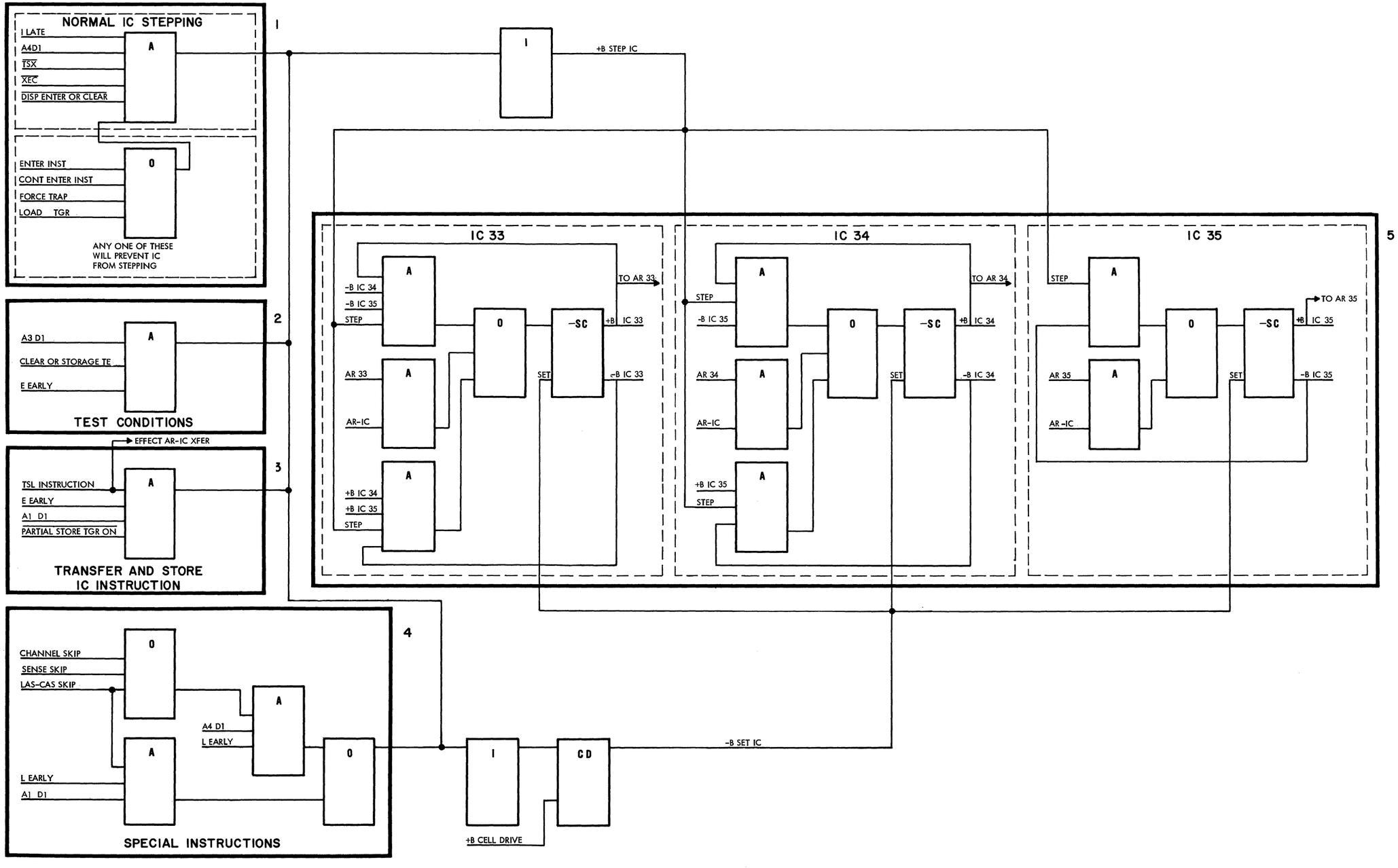


FIGURE 34. INSTRUCTION COUNTER

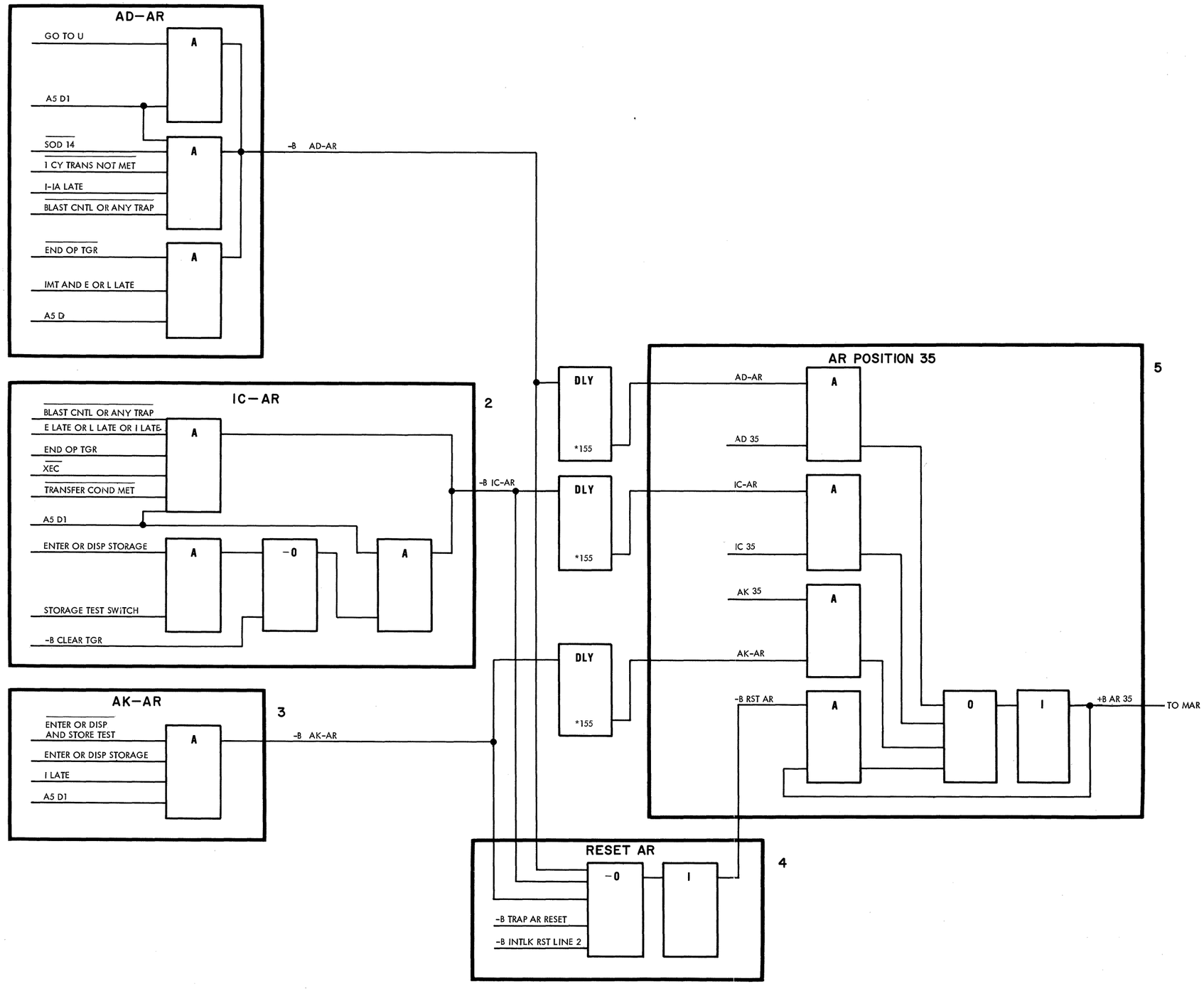


FIGURE 35. ADDRESS REGISTER

Effective Address

Effective address is the term usually applied to the actual address in core that is required by a particular instruction; it is the address that is placed in the address register from the adder to fetch an operand.

The effective address can be obtained in any of three ways:

1. The address may be taken directly from the address portion of the instruction word (Figure 36, A). When the instruction word is decoded, the tag bits (18, 19, and 20) indicate that an index register will not be selected to modify the address portion (21-35) of the instruction word; bits 12 and 13 indicate that indirect addressing will not take place. So the address is transferred, unaltered, from the storage bus to the storage register, to the adder, to the AR, and then to MAR, where an operand fetch is effected.

2. The address may be taken from the instruction word, modified by an index register, and then sent out to fetch an operand; this is called indexing or address modification (Figure 36, B). Decoding of instruction word bits 12 and 13 indicates indirect addressing will not take place; tag bits, however, indicate that index register A is to be used to modify the address of the instruction word. The instruction word goes from the storage bus to the storage register, and from the storage register to the adder; the contents of index register A also feed the adder (into positions 21-35). A 2's complement subtraction between the address of the instruction word and the contents of the index register takes place in the adder; the result is the effective address. The effective address goes to the AR and then to the MAR to effect an operand fetch.

3. The address portion of the instruction word may, instead of indicating the address of an operand, indicate the address of a word in core storage which contains the address of the operand; this technique is called indirect addressing (Figure 36, C). Decoding of bits 12 and 13 indicates that indirect addressing is to take place; the IA trigger is set, causing the next cycle to be an IA cycle. Bits 18, 19, and 20 indicate there is to be no indexing. (The address is capable of being indexed if so indicated by tag bits; for simplicity no indexing has been selected.) The address goes from the storage bus to storage register, to adder, to AR, and then to MAR to effect the readout of the word in the specified location. The word read out and placed on the storage bus is decoded. Bits 12 and 13 are not decoded this time because the CPU is in an IA cycle, and the IA trigger can be set only during I late time (Figure 21). Bits 18, 19, and 20 indicate index register B is to be used for address modification. The contents of index register B in 2's complement form are subtracted from the address of the word in the adder; the resultant is the effective address of the operand. This effective address goes to the AR and then to the MAR to fetch the operand required by the CPU.

Index Register

The 7040/7044 has three identical index registers (A, B, and C), each with 15 latch positions (21-35). Figure 37 shows one position of one index register to illustrate how and when it may be set or read out. One 15-position register, XRX, loads all three index registers. The index registers can be loaded only from the XRX; the instruction tag bits determine which index register is to be loaded. The XRX is loaded from the adder as a result of any one of the special instructions that cause an index register to have its contents changed. There are many combinations of levels and timing pulses that reset and set both the XRX and the specified index register; in general, however, it can be said that any instruction that alters the contents of an index register causes the reset and set level to be generated.

The specified index register may have its contents read out to either the adder or the storage register. In either a POD 63 or 74 class instruction, the readout is to the storage register; in all other instances, the readout is to the adder. The AND labeled 1 (Figure 37) is the one concerned with address modification. The XR-AD transfer takes place at the same time, A4D3 during I-IA Late, as does the SR-AD (Figure 37), so the address portion of the instruction word becomes the effective address the instant it is transferred into the adder.

PARITY

The 7040/7044 is an odd-parity system; i.e., each word in core storage contains an odd number of 1's. When storing a word, the CPU ascertains how many 1 bits are in the word (bits S through 35). If there are an even number of 1's, the CPU assigns an additional 1 bit into MDR position 36, making an odd total of 1 bits in the word being stored. If the word contains an odd number of 1 bits (S through 35), the CPU lets MDR position 36 remain a 0. Core storage words, then, have one more bit position than the words utilized by the CPU. This extra position, the parity bit, is used to check the validity of words read out of memory, called parity checking. The additional bit is labeled the check (C) bit but is commonly called the parity bit; the terms are synonymous.

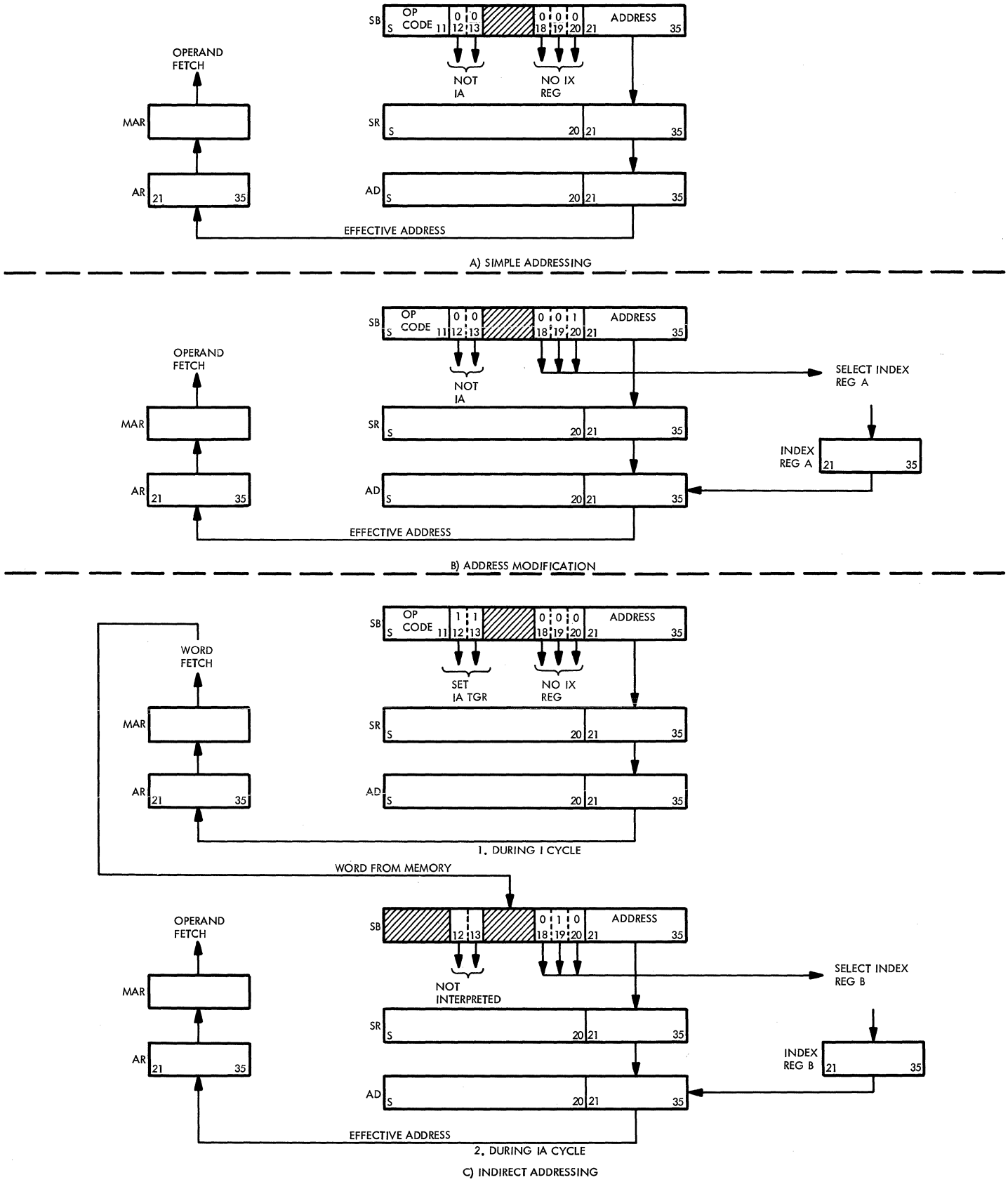


FIGURE 36. OBTAINING EFFECTIVE ADDRESS

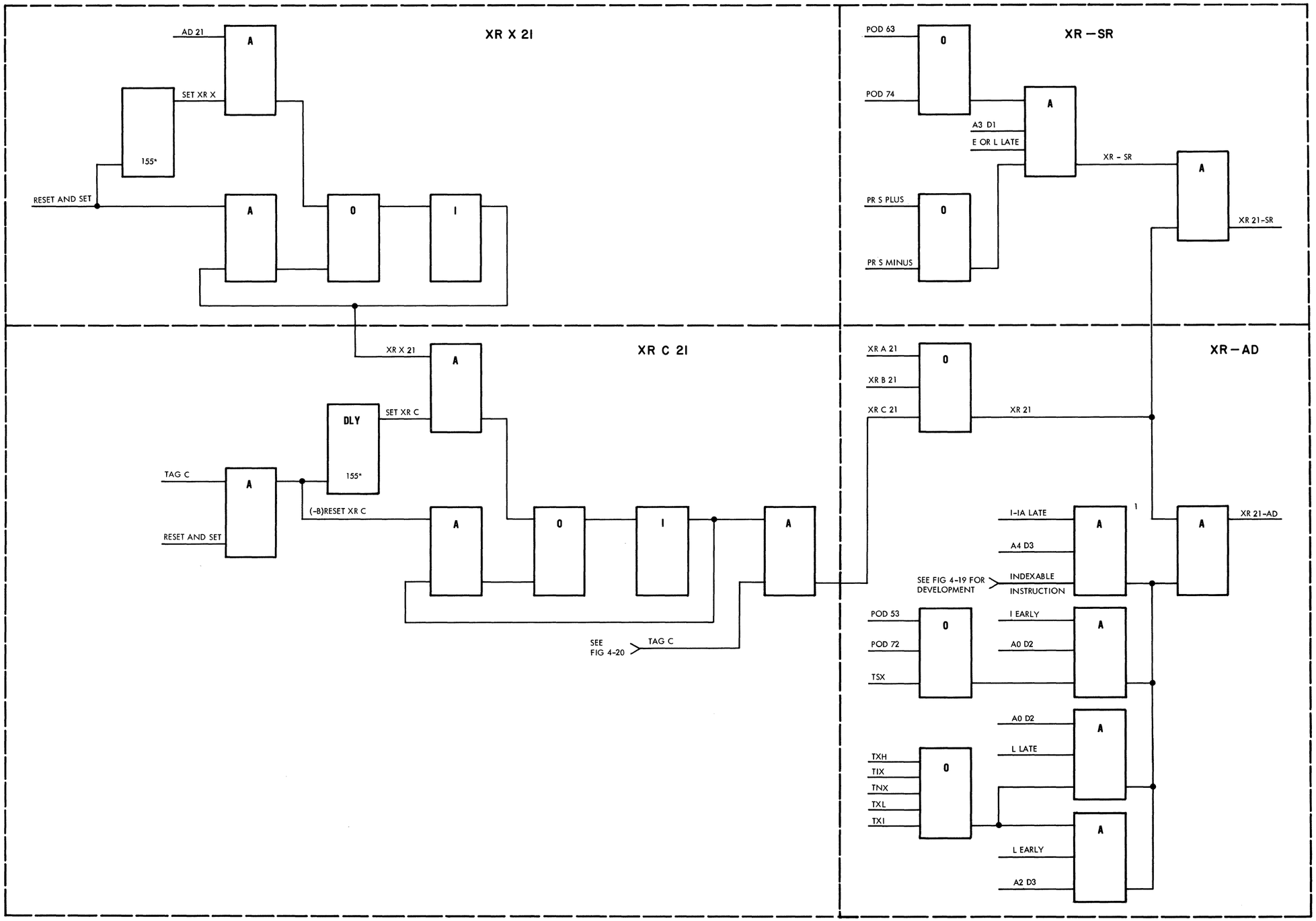


FIGURE 37. INDEX REGISTER

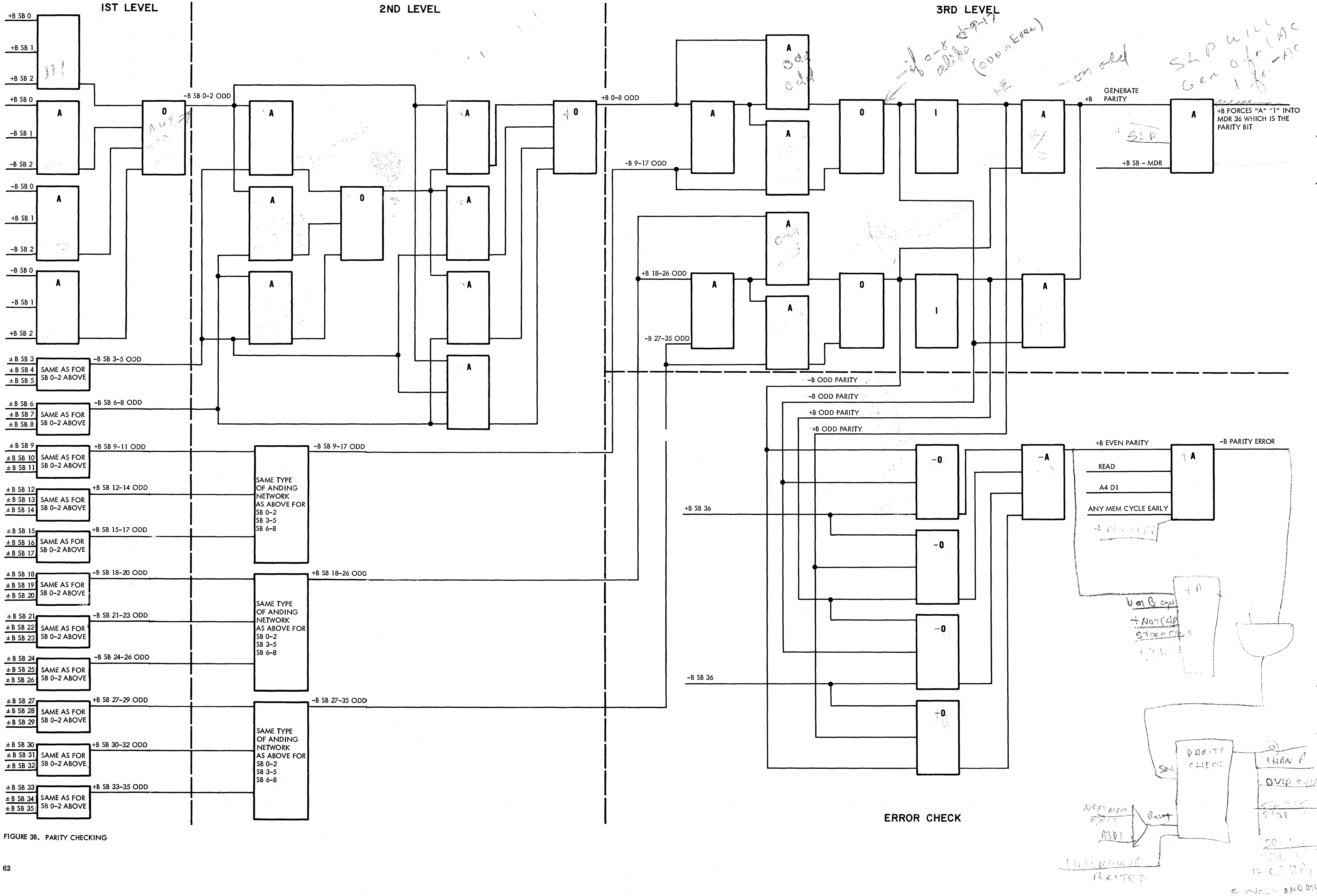


FIGURE 38. PARITY CHECKING

Parity Checking

Parity checking is necessary because bits may be lost or gained during a store and/or readout process. Many factors could cause the losing or gaining of bits: noise buildup in memory, circuit malfunctions, improper inhibiting within memory, etc.

The CPU ascertains whether all words read from memory contain an odd number of 1's. Any word whose 1-bit count is not odd is considered to have a parity error. Usually, a parity error is undesirable, and the CPU program will be written so as to cause the CPU to go into a parity trap routine. However, the presence of a parity error in itself will not cause an automatic trap.

Figure 38 shows how parity is assigned and checked. All words on the storage bus, whether going to or coming from storage, have their parity checked automatically.

Parity checking is accomplished in three stages or levels. The first level receives all 36 bits (S through 35) from the word on the storage bus and groups the bits into 12 groups of three bits each. Each of the 3-bit groups is AND'ed in a manner that will detect whether the group has an odd or even number of 1's. For any 3-bit group, there are four combinations which will produce an odd number of 1's: if all three bits are 1's or if any one (and only one) of the three bits is a 1. The first-level parity will have 12 outputs (one for each 3-bit group); each output will indicate odd parity if the outputs are as shown in Figure 38, or even parity if opposite from that shown.

The second-level parity receives the 12 outputs from the first-level parity and groups them into four groups of 3. Each of the second-level groups is AND'ed in effectively the same manner as the first level to determine whether the group has odd or even parity. The second level has four outputs; each output is representative of nine bits of the word on the storage bus.

The four outputs from the second-level parity feed into the third-level parity, where they are resolved into one output: a +B, indicating that the entire word (S-35) contained an even number of 1's, or a -B, indicating an odd number. This one output is called a generate parity output and will, on a store operation, determine the contents of MDR position 36; if the generate-parity level were +B (indicating even parity), a 1 bit would be forced into MDR 36 when the SB-MDR transfer took place.

Four inputs from the third-level parity network are fed into an error-checking network. In the error-checking network, the parity of the word from the storage bus is compared with SB 36 (parity bit) to determine whether the word has the parity count that the parity bit (SB 36) indicates it should have. For instance, if the storage bus word (S-35) contains an even number of 1's, the parity bit should be a 1. The output of the error check should be -B to indicate correct parity. If, however, the error check output is +B even parity, an error is indicated. The +B even parity is AND'ed with a read to generate a -B parity-error indication.

On write, the parity checking network assigns a parity bit to the word being stored in memory. On read, the parity-checking network determines whether the word contains the same number of 1 bits as when it was originally stored in memory; a parity error is generated if the word read out does not contain the number of 1's indicated by the parity bit.

This section describes the majority of the 7040-7044 instructions. Not included are the I-O instructions, which are covered in the Channel A instruction manual, form R23-2652, and in the Channel A reference manual, form R23-2644.

Detailed flow diagrams of the instructions are found in the CPU Logic Diagrams manual, form R23-2659; refer to them when studying this section. A figure list in the front of the CPU Logic Diagrams manual lists the instruction flow diagrams alphabetically by mnemonic code. Only three flow diagrams are included in this section, not as detailed as those in the CPU Logic Diagrams manual, that illustrate add, subtract, multiply, and divide. The purpose of the three flow diagrams is to help in understanding these basic CPU operations.

Also included in this section is a discussion of how the machine obtains the difference of two numbers using the complement method, and the significance of the Q carry.

Subtraction - Machine Method

In contrast to the direct method of subtraction, where borrowing must take place and adjustments made accordingly, the machine subtracts by adding - the only function the adder can perform. This is possible because when the complement of a number is added to another number, in any numbering system, the difference of the two numbers is obtained.

Binary

Two types of binary complementing are used, a 1's complement and a 2's complement, as illustrated in the following example:

100 100 Number
011 011 1's Complement
011 100 2's Complement

As the example shows, the 1's complement of a binary number is the number with all its bits reversed; and the 2's complement of the same number is 1 greater than its 1's complement. Both complement types are used by the machine in obtaining the difference of two numbers. The examples given below illustrate the two methods, the results obtained, and the significance of the high-order (Q) carry. In all cases, the subtrahend (number being subtracted) is complemented.

	Direct Method	1's Complement Method		2's Complement Method	
Case 1:	101 101	101 101		101 101	
Subtrahend	100 100	011 011	1's Comp	011 011	1's Comp
Smaller	001 001	001 000	(Q Carry)	1	Carry In
		1	Carry In	001 001	True Difference
		001 001	True Diff.		(Q Carry)
Case 2:	101 101	101 101		101 101	
Subtrahend	101 101	010 010	1's Comp	010 010	1's Comp
Equal	000 000	111 111	1's Comp Diff. (no Q Carry)	1	Carry In
				000 000	True Difference (Q Carry)
Case 3:	101 101	101 101		101 101	
Subtrahend	110 101	001 010	1's Comp	001 010	1's Comp
Greater	001 000	110 111	1's Comp Diff. (no Q Carry)	1	Carry In
				111 000	2's Comp Diff. (no Q Carry)

Note first that the 2's complement operation is effected by adding a 1 to the low order position along with the 1's complement of the subtrahend. In the 1's complement operation, the 1 is added in a separate addition, as a correction, when a Q carry results from the first. A Q carry is possible because leading 0's in the subtrahend are introduced as 1's in the adder.

Note also the significance of the Q carry; in both the 1's and 2's complement operations, the Q carry indicates the true difference was obtained; where no Q carry occurred, the difference is in complement form. All the machine has to do is recomplement the difference in these cases to obtain the true difference. The Q carry further indicates that the subtrahend is the smaller number in 1's complement operations; and that the subtrahend is the smaller or equal number in 2's complement operations. Here then is the real difference in the 1's and 2's complement operations: when equal numbers are subtracted, a Q carry and a true difference result in 2's complement operations; no Q carry and a complement difference result in 1's complement operations. This fact is useful in comparing two numbers. If a Q carry occurs while subtracting them in 2's complement, but does not occur when subtracting them in 1's complement, the two numbers are equal.

The following chart summarizes the Q carry indications for both types of operations. The subtrahend always refers to the complemented number.

1's Complement		2's Complement	
Q Carry	No Q Carry	Q Carry	No Q Carry
True Difference; Subtrahend Smaller	Comp (1's) Difference; Subtrahend Equal or Greater	True Difference; Subtrahend Equal or Smaller	Comp (2's) Difference; Subtrahend Greater

Octal

The 1's and 2's complement of a binary number have their equivalent in the octal numbering system, as the 7's and 8's complement, shown in the following example:

Binary		Octal	
100 100	Number	44	
011 011	1's Complement	33	7's Complement
011 100	2's Complement	34	8's Complement

Besides directly interpreting its binary equivalent, the 7's complement of an octal number is derived by taking the difference of each digit and the highest number in the octal system (7). The 8's complement is 1 greater than the 7's complement. Using another example, the 7's complement of the octal number 320 is 457; the 8's complement is 1 greater, or 460.

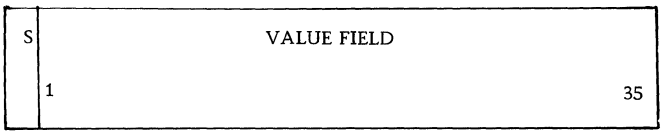
Because the 7's and 8's complement in octal is equivalent to the 1's and 2's complement in binary, high order carries occuring in a complement add operation have the same significance in either system. In the chart above, the 7's and 8's complement could be substituted for the 1's and 2's complement, respectively, and the chart would still be valid.

This means that you can predict whether or not the machine will produce a Q carry in a given problem using the octal system, by substituting the 7's complement when the machine uses the 1's complement, and the 8's complement when the machine uses the 2's complement. It's important to realize this because the machine makes many decisions based upon the presence or absence of the Q carry signal. As in case 1 of the examples above, the Q carry tells the machine that the difference produced is true, but must be increased by 1; in the 2's complement operation, the Q carry also indicates a true difference, but that no correction is necessary. The case 1 examples are reproduced below using the octal numbers to show that the high-order (Q) carry occurs the same as in binary. Do the same for the remaining examples to prove it to yourself.

	Direct	7's Complement	8's Complement
Case 1:	55	55	55
Subtrahend	44	33 7's Comp	34 8's Comp
Smaller	11	↖10 Carry	↖11 True Difference
		1 Correction (Carry In)	(Carry)
		11 True Difference	

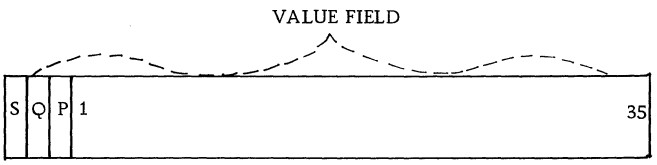
FIXED POINT ARITHMETIC

Fixed-point arithmetic is the most basic form of arithmetic. Simply stated, it is the process of computation using quantities whose magnitude is completely expressed by a single value field. The relationship of the magnitude to zero is expressed by a sign position. In fixed-point arithmetic, the length of an operand is generally determined by the smallest unit of data that can be accessed in core storage. In the 7040-7044, fixed-point arithmetic operands have the following basic format:



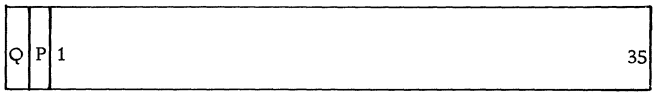
The sign bit S determines whether the magnitude is positive or negative. When S is a 0, the magnitude is positive; when S is a 1, the magnitude is negative. The value field is 35 bits long and states the magnitude of the number. A fixed-point operand can then be defined as a unit of data 36 bits long, containing a sign bit and 35 magnitude bits.

Fixed-point arithmetic in the 7040-7044 includes addition, subtraction, multiplication, and division. All operations involve only two operands: one operand is explicitly addressed; the other is implied. In addition and subtraction, the explicitly addressed operand is obtained from the core storage location specified by the instruction word effective address. The implied operand is obtained from the accumulator. The former is generally known as the addressed operand; the latter, as either the implied or accumulator operand. In the CPU, the addressed operand is placed in the storage register, which has the format shown above. The implied or accumulator operand has the following format:



The accumulator value field is 37 bits long. The adtional two bits, Q and P, are provided primarily to handle conditions which result in a carry of 1 out of position 1. Bits P and Q are therefore known as overflow bits and are treated as the two highest-order accumulator bits during the execution of fixed-point arithmetic.

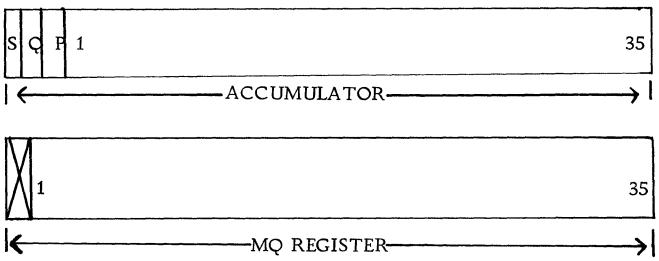
The actual arithmetic takes place in the adder, which has the following format:



Basically, the contents of the storage register are transferred simultaneously with the accumulator contents into the adders. An addition or subtraction is effected, and the result is transferred into the accumulator.

In multiplication, the addressed operand is obtained from the core storage location specified by the instruction word effective address; the implied operand is obtained from the multiplier-quotient (MQ) register. In the CPU, the addressed operand is placed in the storage register, which has the basic format of a sign bit and a 35-bit value field. Storage register contents become the multiplicand. The MQ register contents form the multiplier, which has a format identical with the multiplicand. Multiplication is effected by a combination of right shifts and simple additions. A multiplication result is placed in the combined accumulator-MQ register. with MQ register bit 35 the lowest-order bit. Multiplication is algebraic, and the result sign is placed in both the accumulator sign position and the MQ register sign position.

In division, the addressed operand is obtained from the core storage location specified by the instruction word effective address; the implied operand is obtained from the combined accumulator-MQ register. The addressed operand is placed in the CPU storage register and becomes the divisor; the combined accumulator-MQ register becomes the dividend. Divisor format is the basic single sign bit and 35 value field bits. The dividend format is a single sign bit and 72 value field bits:



The result or quotient is placed in the MQ register and has a format identical with the divisor. Remainder bits, if any, go into the accumulator, with a format of one sign bit and 37 value field bits; accumulator bit 35 is the lowest-order remainder bit. Division is effected by a combination of subtractions and left shifts.

Addition

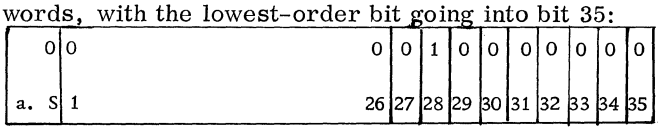
In performing addition in the 7040-7044, the general rules of algebra must first be applied to the signs of the quantities involved to determine whether the sum or difference of the quantities involved is to be obtained. Therefore, when adding two positive quantities, the result is the sum of those quantities with a positive sign. When adding a positive and a negative quantity is involved, the sum is actually the difference of the two quantities, with the result sign being the sign of the larger magnitude. Finally, when adding two negative quantities, the result is the sum of the quantities with a negative sign.

Assume the quantity $+200_8$ is to be added to the accumulator, which contains $+75_8$. The result is $+275_8$. To satisfy machine operand format, convert the quantities into their binary equivalents:

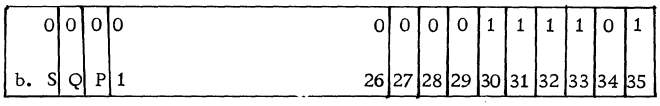
a. $+200_8 = +010\ 000\ 000$

b. $+\ 75_8 = +\ 000\ 111\ 101$

Insert these binary numbers in their respective data words, with the lowest-order bit going into bit 35:



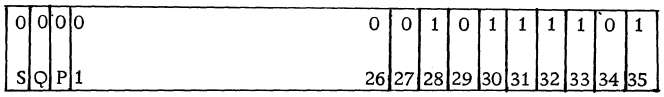
STORAGE REGISTER



ACCUMULATOR

Bits 1 through 26 are not needed to express the quantities and are therefore all 0's. Because accumulator bits Q and P are treated as part of the value field and the accumulator value is assumed as $+75_8$, bits P and Q are 0's. Since each number is positive, a 0 is placed in the respective sign bit S.

Adding the two operands produces a result magnitude of 010 111 101, with a result sign of 0. In machine operand format, the result is illustrated as follows:



ACCUMULATOR

If the same magnitudes are used but the signs changed to negative, the entire handling of the magnitude remains unchanged in performing the addition. The 7040-7044 treats the sign bits separately. To correctly represent the negative values, simply insert a 1 in the sign bit position of each of the operands and the result; this is what is done in the machine.

Since algebraic principles are employed, addition of two quantities with unlike signs is effectively a subtraction. Using the same values, but changing the sign of the accumulator operand to a minus, the problem becomes $(+200_8) + (-75_8)$. To accomplish addition, line up the octal points and subtract:

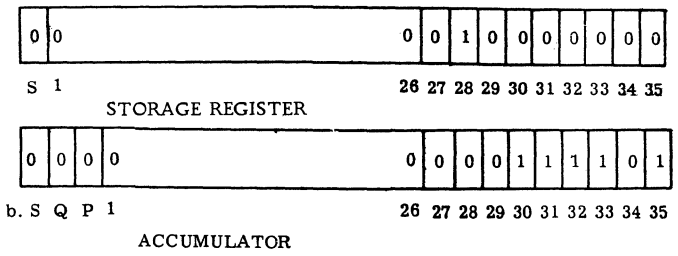
$$\begin{array}{r} +200_8 \\ + \\ -075_8 \\ \hline +103_8 \end{array}$$

To satisfy machine operand format, convert the values into their binary equivalent:

a. $+200_8 = +010\ 000\ 000$

b. $-075_8 = -000\ 111\ 101$

Insert these binary numbers into their respective data words, with the lowest-order bit in each value going into bit 35:



Bits 1 through 26 are not needed to express the quantities and are therefore all 0's. Accumulator bits Q and P are implied 0's by the assumed accumulator value.

The following describes how the machine effects addition of values having unlike signs. Refer to figure 39, a simplified flow diagram of the Add-Subtract operation, while reading this text. A detailed flow diagram is in the CPU Logic Diagrams manual.

a. Adds the complemented accumulator value field and storage register value field.

b. Places the result in the accumulator.

c. Checks for a Q carry:

(a) If there is a Q carry, adds 1 to the accumulator in the lowest-order position (bit 35), inverts the accumulator sign, and plaees the resultant operand in the accumulator.

(b) If there is no Q carry, complements the accumulator value field.

Overflow and Q Carry

The term overflow means that the capacity of the machine has been exceeded: the arithmetic result cannot be represented by the machine, because it contains more than 35 value field positions. It was previously stated that accumulator bits Q and P are called overflow bits. The name, however, only provides an easy means of identifying these bits as a pair. Because they could originally contain 00, 01, 10 or 11, their significance depends on the problem. When dealing with values having like signs, a resultant 1 in either bit or in both bits indicates an overflow. In this case, the overflow is recorded, but subsequent action depends on the program being executed. When dealing with unlike signs, the overflow bits are significant as a pair and, in this sense, either generate or do not generate a Q carry. If a Q carry is generated, it indicates (1) that the accumulator operand was the smaller operand and (2) that the number presently in the accumulator value field is a true number equal to 1 less than the correct answer. If a Q carry is not generated, it indicates (1) that the accumulator operand was the larger operand and (2) that the number presently in the accumulator value field is the correct answer in complement form.

Subtraction

Subtraction in the 7040-7044 is algebraic and is accomplished as previously described, by complementing and adding. At the start of the operation, the sign of the subtrahend (storage register operand) is inverted. Subtraction of unlike signs becomes addition, and whether the accumulator is the larger or smaller operand is insignificant. Generation of a Q carry when dealing with unlike signs before inversion represents an overflow.

The following describes how subtraction is effected. Refer to Figure 39.

1. Complement storage register sign (actually occurs as sign bit enters the storage register from the storage bus).
2. Compare accumulator and storage register signs:
   a. If alike, add accumulator and storage register.
   b. If unlike, add complemented accumulator to storage register.
3. Place addition result in accumulator.
4. a. If accumulator and storage register signs are alike, check for a carryout of adder value field position 1. The coincidence of like signs and a 1 carryout of value field position 1 indicates an overflow.
   b. If accumulator or storage register signs are unlike, check for a Q carry:

(1) If there is a Q carry, add 1 to present accumulator value field in low-order position, and invert accumulator sign.

(2) If there is no Q carry, complement accumulator value field.

The Q carry serves to indicate the accumulator was the smaller operand and that the present accumulator value field is in true form and 1 less than the correct answer, when dealing with operands having like signs. The absence of a Q carry, on the other hand, indicates the accumulator was the larger operand and the present accumulator value field is the correct answer in complement form.

Multiplication

The rules for binary multiplication are similar to those of decimal multiplication. The rules for multiplying two single digits are the same in both systems. These rules are:

$0 \times 0 = 0$

$0 \times 1 = 0$

$1 \times 0 = 0$

$1 \times 1 = 1$

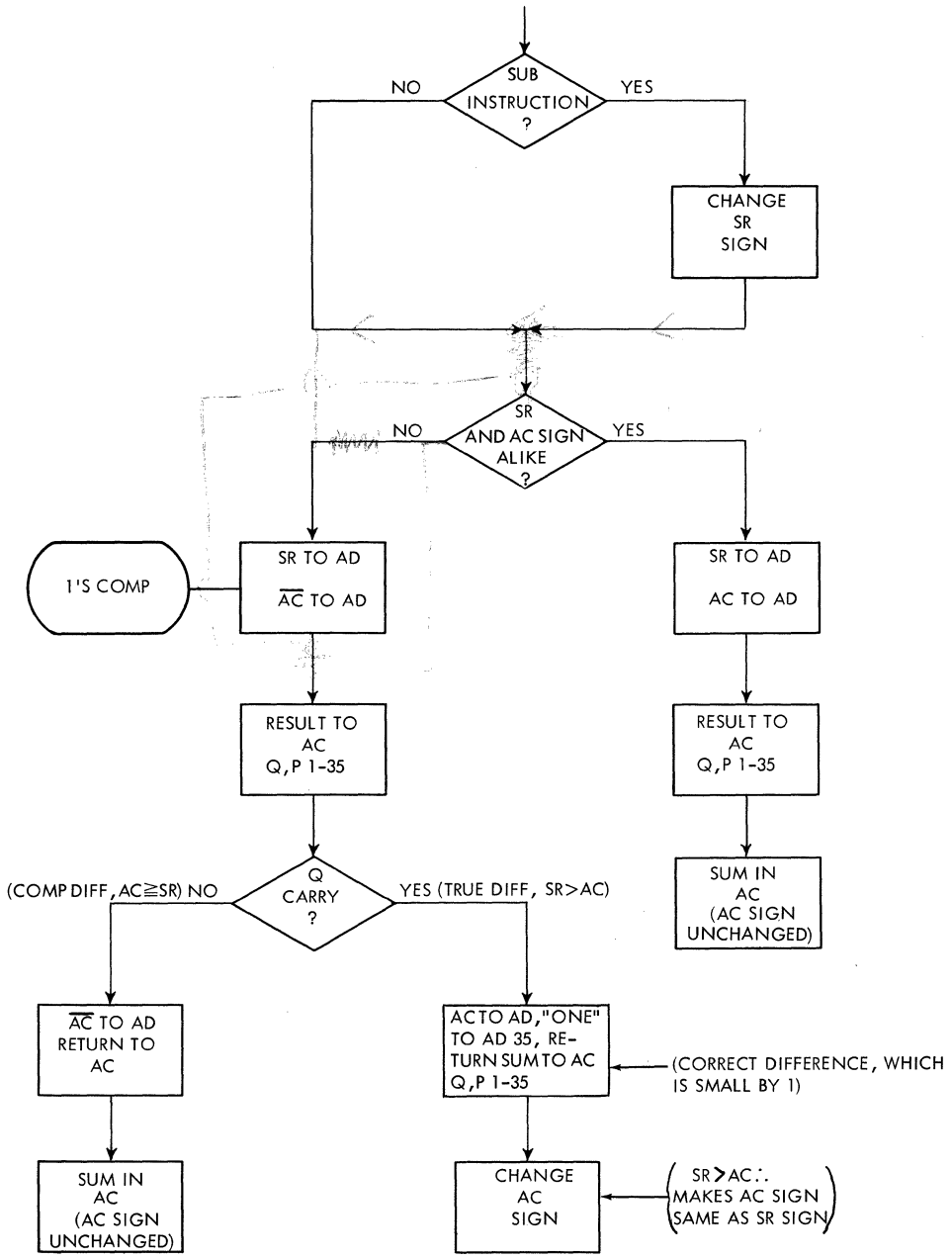


FIGURE 39. 7040/44 ADD - SUBTRACT PROCESS

The general procedure when multiplying two multiple digit binary numbers is the same as that in decimal arithmetic; that is, the multiplicand is multiplied by a digit of the multiplier, and the partial product obtained is placed so that the least significant digit is under the multiplier digit. When all the partial products have been found, they are added to find the final product. The only difference between decimal and binary multiplication, therefore, is in the summing of the partial products. In binary, the binary addition table is used; in decimal, the decimal table is used.

As can be seen from the following examples, the method of obtaining partial products and then adding them to obtain the final product is identical with that of decimal arithmetic:

Multiplicand	1010	10.11	1111
Multiplier	1101	100.1	1111
First Partial Product	1010	1 011	1111
Second Partial Product	0000	00 00	1111
Third Partial Product	1010	000 0	1111
Fourth Partial Product	1010	1011	1111
Final Product	10000010	1100.011	11100001

Note the placement of the binary point in the second example. The same rules hold for its placement as for placement of the decimal point in decimal arithmetic.

The third example also illustrates an interesting point. This is the multiplication of the two largest possible 4-bit numbers. The product is eight bits long. In other words, the largest product that can result from the multiplication of two numbers will be no longer than the sum of the number of bits in the multiplier and multiplicand.

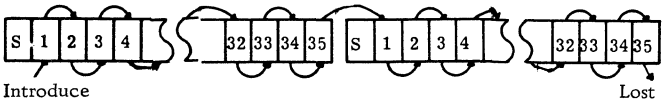
If a number is multiplied by the radix of the number system, this multiplication has the effect of shifting the number one place to the left with respect to the radix point. This is true in any number system. For example, multiply 12.51_{10} by 10 (the radix of the decimal system) and multiply the number 10.11_2 by 2 (the radix of the binary system):

Number	12.51	10.11
Number Times Radix	125.1	101.1

Binary multiplication, then, is nothing more than a series of add and shift operations.

Multiplication--Machine Operation

When performing multiplication in the 7040-7044, the contents of the storage register are multiplied by the contents of the MQ register. The actual multiplication is accomplished by repeated conditional addition of the storage register contents to the AC register contents, interspersed with a shift of the accumulator and MQ register one bit position to the right. Basically, for every multiplier bit that is a 0, the combined accumulator-MQ register value field is shifted right one position. For every multiplier bit that is a 1, the storage register contents are added to the accumulator contents, with the result going to the accumulator displaced right one position. As this result is placed in the accumulator, the MQ register value field is shifted right one position. As a result of the shifting, MQ register bit 35 is lost.



A device called the shift counter is used to determine the number of repetitive shifts executed. In the machine, upon decoding a multiply operation, the shift counter is set to 43_8 (35_{10}). Therefore, regardless of the number of high-order 0 bits, two 35-bit numbers are multiplied. For ease of discussion, the example given in this paragraph uses 6-bit numbers and, therefore, a value of 6 in the shift counter.

Multiplication in the 7040-7044 is algebraic: positive X positive = positive; negative X negative=positive; positive X negative = negative. Therefore, in every multiplication operation, the sign bits of the operands involved are treated separately from the value fields. Before any adding or shifting takes place, the sign bits are compared, and the result sign is determined.

The following describes how the machine effects multiplication. Refer to Figure 40, a simplified flow diagram of the multiply operation, while reading this text. A detailed flow diagram of the multiply code is in the CPU Logic Diagrams manual.

a. Sets the shift counter to a value of 43_8 (35_{10}) and then checks it for a value of 0.
   1. If it is =0, ends the operation.
   2. If it is ≠0, continues.

b. Tests the storage register for all 0's (storage register contents are tested in accumulator): If all 0's, clears MQ register bits S-35 and resets the shift counter.

c. Clears the accumulaotr (Q-35).

d. Compares the storage and MQ register signs:
   1. If alike, sets the accumulator and MQ register signs to 0 (positive).
   2. If unlike, sets the accumulator and MQ register signs to 1 (negative).

e. Checks the shift counter for a value of 0; if SC = 0, ends the operation, whereas if SC ≠ 0, checks MQ register bit 35:
   1. If it is a 0:
      (a) Shifts the combined accumulator-MQ register right one position.
      (b) Steps the shift counter (reduces value by 1).
      (c) Repeats step e.
   2. If it is a 1:
      (a) Adds the storage register to the accumulator, and places the result in the accumulator displaced one position to the right.
      (b) Shifts the MQ register right one position.
      (c) Steps the shift counter.
      (d) Repeats step e.

Division

Binary division is the process of counting the number of times a divisor goes into a dividend. The count of the number of times the divisor may be subtracted from the dividend before a negative remainder occurs is called the quotient.

Direct binary division is performed by a series of subtractions of the divisor (actually a multiple of the divisor), just as it is in the decimal system. For example, divide 100 011 100 by 1110:

```
              bd ehi jk
              10 100.01
          ┌──────────
1110      │ 100 011 100.00
            11 10
          ──────
          (c) 111 1
          (f) 111 0
          ──────────
          (g)     100 00
          (l)      11 10
          ──────────────
          (m)         10
```

In the example, the first step is to place the divisor below the dividend in a position which is as far removed to the left as possible (a), but which will allow a position difference to result when the divisor is subtracted from the dividend. Since the divisor will go into this many bits of the dividend once, a 1 is placed in the quotient at b in the same column as the lowest-order digit of the divisor. The divisor is then multiplied by the quotient digit, and the resulting product is subtracted from the dividend to produce the positive difference (c) called the current remainder. The next digit in the dividend is brought down to line c. Compare the divisor with line c; note that the divisor is larger than line c, or that the divisor goes into line c 0 times. Therefore, place a 0 in the quotient at the d position. The next digit of the dividend is then brought down to line c. Comparing the divisor with line c shows line c to be greater. Place a 1 in the quotient at the e position. Multiply the divisor by the last quotient bit to form line f. Subtract line f from line c to start line g. The next digit in the dividend is brought down to line g. Compare the divisor with line g; the divisor is greater, so place a 0 in the quotient at position h. Bring the next digit of the dividend down to line g; by comparison, line g is still smaller than the divisor. Place a 0 in the quotient in position i, and place the next dividend digit on line g. Line g is still smaller than the divisor, so a 0 is placed in the quotient at position j. Placing the next dividend digit on line g now makes line g greater than the divisor. Place a 1 in the quotient at position k, and multiply the divisor by this 1 to form line 1. Subtract line 1 from line k to start line m. Assuming a quotient has been developed of sufficient length, terminate the operation. The quotient is 10100.01 with a remainder of 10 (line m).

Since the quotients bit is always either 0 or 1, the division process can be reduced to a series of subtractions of the divisor, multiplied by the power of the quotient bit being sought from the dividend. Each time a subtraction results in a positive current remainder, a 1 is placed in the corresponding quotient bit position, and the process is immediately repeated for the next quotient bit. Each time the subtraction results in a negative remainder, a 0 is placed in the corresponding quotient bit. In this case, the current remainder is restored to a positive number by adding the divisor back to it. Following this, the next quotient bit is obtained by the subtraction of the divisor multiplied by the power of the next quotient bit.

Since the quotient bits are generated from left to right, the power of each quotient bit is one smaller than that of the last bit generated. This means that, as the divisor is successively subtracted from the dividend (or current remainder), the divisor is shifted to the right in relation to the binary point. The division process can therefore be reduced to a pro-

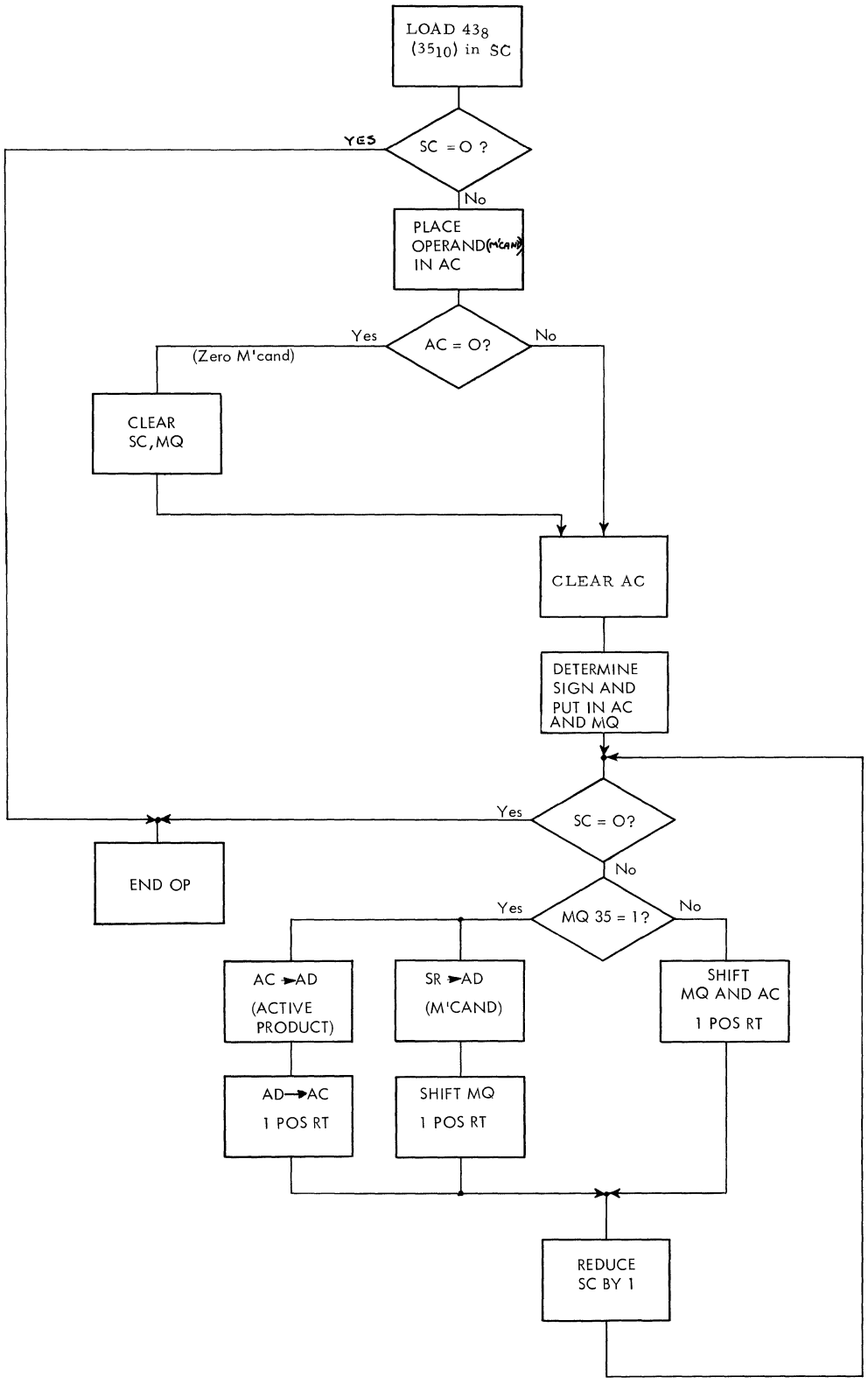


FIGURE 40. 7040/44 MULTIPLY INSTRUCTIONS

cess of successive subtract and shift steps.

Division--Machine Operation

When performing division in the 7040-7044, the contents of the combined accumulator-MQ register are divided by the contents of the storage register. MQ register bit 35 is the lowest-order dividend bit; accumulator bit 1 is the highest-order dividend bit. Although not part of this dividend, accumulator bits Q and P must be 0 at the start of division. The sign of the dividend is the sign of the accumulator; the MQ register sign is not significant to the dividend. The quotient is placed in the MQ register, and the remainder, if any, is placed in the accumulator. Division is algebraic; therefore, like signs yield a positive result, and unlike signs, a negative result. Quotient sign determination is accomplished independently of the division, with the result sign being placed in the MQ register sign position. The sign of the accumulator remains unchanged throughout the divide process and is the sign of the remainder. This sign may differ from the sign of the quotient.

The MQ register has a format of a sign bit and 35 value field bits. A quotient, therefore, can never exceed 35 bits in length; that is, it can never have a value greater than $2^{35}-1$. A maximum value of $2^{35}-1$ is established by comparing the maximum value a register can contain (its bits all 1's) against the value of 2 raised to the number of bits in the register. For example, a 2-bit register can contain a maximum value of 11_2 or 3. The number of bits in the register is 2, and 2 raised to the second power is 2^2 or 4 which is 1 more than the maximum value that the register can contain. Extending this comparison to any size register confirms its validity. Further, the comparison provides the basis for the value size limitation of the quotient.

To insure that the quotient will fall within MQ register capacity, the dividend must be less than the divisor x 2^{35}. Ascertaining whether this condition exists is quite simple. It is done by comparing accumulator bits Q-35 with storage register bits 1-35. If Q or P is a 1, a divide check will occur. If the value in the accumulator bits is smaller than the value in the storage register bits, a quotient will result that can be contained in the MQ register. The comparison is accomplished by complementing accumulator bits Q-35, and then adding the complement value to storage register bits 1-35. Generation of a Q carry indicates the value in the accumulator bits is smaller and, therefore, the quotient can be contained in the MQ register.

The following describes how the machine effects division. Refer to Figure 41, a simplified flow diagram of the divide operation, while reading this text. A detailed flow diagram of the divide code is in the CPU Logic Diagrams manual.

1. Set the shift counter to 43_8 (35_{10}).
2. Complement the accumulator (bits Q-35).
3. Test for SC = 0. If it is, end Op and re-complement the AC in the following I cycle.
4. Add the storage register and accumulator value fields. (The result of this addition is not recorded).
5. Check for a Q carry:
   a. If there is no Q carry, turn on the divide check trigger and reset the shift counter. Then complement the accumulator, and get the next instruction.
   b. If there is a Q carry:
      (1) Set the quotient sign.
      (2) Complement MQ register bit 1.
      (3) Shift accumulator bits P-35 and MQ register bits 1-35 left one position.
      (4) Step the shift counter (reduce it by 1).
6. Add the storage register and accumulator value fields.
7. Check for a Q carry:
   (a) If there is no Q carry:
      (1) Place a 1 in MQ register bit 35.
      (2) Transfer the addition result into the accumulator; go to Step 8.
   (b) If there is a Q carry, go to Step 8.
8. Check the shift counter for a value of 0:
   a. If SC $\neq$ 0:
      (1) Complement MQ register bit 1.
      (2) Shift accumulator bits P-35 and MQ register bits 1-35 left one position.
      (3) Step the shift counter.
      (4) Go back to Step 6, and repeat the action.
   b. If SC = 0:
      (1) Complement accumulator bits Q-35.
      (2) End the operation.

VARIABLE-LENGTH ARITHMETIC

Variable-length arithmetic is fixed-point arithmetic using operands of a length other than 35 bits. Variable-length operations include multiply, divide, and multiply and add. Each of these operations requires the use of the shift counter, which is the key to variable-length operations. In the variable-length instruction word, bits 12-17 form a count field.. When a variable-length operation is decoded, the value in the count field is set into the shift counter, rather than 43_8.

Although the count field can contain counts from 0 to 77_8, certain counts are impractical or of no value. For instance, the count in a multiply operation specifies the number of multiplier bits and identifies the low-order result bit. Consider the multiplier. It is contained in the MQ register, which is 35 bits long. Therefore, a multiplier exceeding 35 bits in length cannot be contained in the MQ register. This fact, however, is not important to machine operation because it bases its actions on the shift counter. As long as the shift counter does not equal 0, the machine

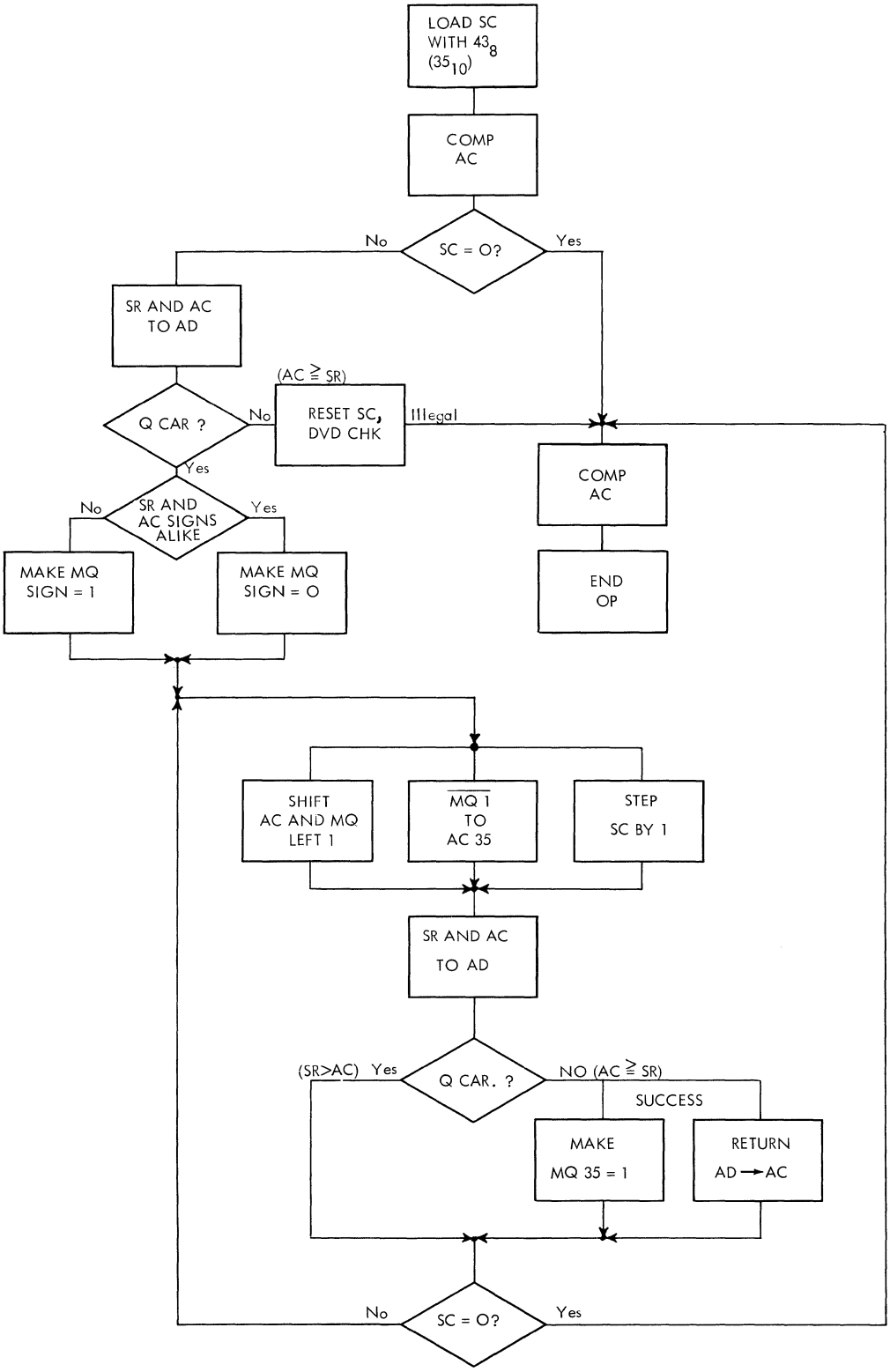


FIGURE 41. 7040/44 DIVISION PROCESS

performs its repetitive steps. Therefore, the multiplier bits needed to satisfy the number specified by the count field in excess of 35 are obtained from accumulator bit 35 with each right shift executed. Further, if a count of 43_8 is specified for a variable-length multiply, the operation performed is identical with a fixed-point multiply. Obviously then, variable-length multiplication is intended for use with multipliers of less than 35 bits in length. In this case, by setting the shift counter to the size of the multiplier, the time needed to execute the operation is reduced, thereby increasing machine efficiency.

In multiplication, the count also serves to identify the low-order result bit. The count in fixed-point multiplication is always 43_8, and the highest-order result bit is accumulator bit 1, with MQ register bit 35 the lowest-order result bit. In variable-length multiplication, accumulator bit 1 is also the highest-order result bit, but the lowest-order result bit is not necessarily MQ register bit 35. The lowest-order result bit in variable-length multiplication is the MQ register bit corresponding to the count field value up to and including a count of 43_8. For example, with a count of 1, MQ register bit 1 is the lowest-order result bit; with a count of 12_8, MQ register bit 10 is the lowest-order result bit, and any count over 43_8 causes the loss of low-order result bits, because they are shifted right, out of MQ register bit 35. In addition, when the count is stepped from 43_8 to 44_8, MQ register bit 35 is actually the lowest-order result bit developed thus far. Therefore, subsequent multiplications are performed using result bits as multiplier bits, with the final result being completely erroneous.

The count in a division operation specifies (1) the number of significant dividend bits in the MQ register and (2) the number of quotient bits to be developed. For the moment, ignore the fact that the machine cannot execute a divide operation with a count of 0. With a decimal count of 35 in the shift counter, the dividend is 70 bits long, extending from accumulator bit 1 through MQ register bit 35. Subtracting 35 from the shift counter makes it 0, and subtracting 35 from the dividend makes it 35. If a divide could be performed with a count of 0, the smallest size dividend possible would be realized, which would be 35 bits long and extend from accumulator bit 1 through accumulator bit 35. For each increment of the count, therefore, the dividend picks up one more significant bit, starting at MQ register bit 1. Thus, a count of 1 makes the dividend accumulator 1-35 and MQ 1; a count of 14_8 makes the dividend accumulator 1 through 35 and MQ 1 through 12; a count of 30_8 makes the dividend accumulator 1-35 and MQ 1 through 24. When the count is 43_8, the variable-length divide is identical with the fixed-point divide. A count exceeding 43_8, in effect, is specifying a dividend beyond the capacity of the machine. However, this condition poses no problem to the machine; as long as a value greater than 0 is in the shift counter, the repetitive steps are performed, causing needed bits to be shifted left from MQ register bit 1 into accumulator bit 35.

In the case of a count greater than 43_8, consider the contents of the accumulator and MQ register as the count goes from 43_8 to 44_8. At the lesser count, the contents of the accumulator form the current remainder, and the contents of the MQ register form the quotient developed thus far. Each, however, is 35 bits long. When the count value goes to 44_8, the highest-order quotient bit is left-shifted into the accumulator. From this point on, division is performed using quotient bits as dividend bits; the final answer is meaningless.

In fixed-point division, the count is always 43_8 and the quotient developed is always 35_{10} bits long. Subtracting 43_8 from the count makes it 0, and subtracting 35_{10} from the quotient makes it 0 bit long. Thus, with every increment of the count, an additional binary position is developed in the quotient: a count of 15_8 results in the development of 13 quotient bits; a count of 34_8, 28 quotient bits, etc. In each case, MQ bit 35 is the lowest-order quotient bit, with the higher-order bits being developed in left adjacent positions. At a count of 43_8, a 35-bit quotient is in the MQ register, with the highest-order bit being MQ 1. When the count goes to 44_8, the highest-order quotient bit is lost; at 45_8, the next highest; etc. It is apparent that a count exceeding 43_8 is of no value, and a count of 43_8 is the same as fixed-point division. Variable-length division, therefore, is best employed using dividends of less than 70 bits or, said another way, using a count of less than 43_8.

In the above discussion, the count is used in an ascending fashion. The machine does not use the count in this fashion. Rather, the machines decrements the count from its maximum value to 0, one decrement at a time. An ascending count was used for ease of description. The effect, however, is the same. Saying the count is stepped from 43_8 to 44_8 is the same as saying the shift counter is stepped for the 44_8 time.

In the 7040-7044, indirect addressing can be used with arithmetic class instructions. Indirect addressing is specified by a 11 configuration in instruction word bits 12 and 13. Variable field length operations, however, also use these bits as part of the count field. Therefore, any count of 60_8 or above causes the instruction word effective address to be interpreted as an indirect address. The contents of the location specified by the instruction word indirect address are referenced, and an effective address is formed in the normal manner. Further, the contents

of bits 12 through 17 in the indirect address (instruction word effective address) are used to set the shift counter. The contents of the effective address are used as the multiplicand or divisor, depending on the operation.

VLM, VMA, and VDP

The variable length multiply, variable length multiply and accumulate, and variable length divide or proceed instructions differ in their execution from their fixed-point counterparts in the value loaded in the shift counter. As already mentioned, the fixed-point multiply and divide instructions load the shift counter with 43_8 (35_{10}), while the variable length instructions load the shift counter with the instruction word count field (bits 12 to 17). The VMA instruction further differs from fixed point multiply by not checking for a zero multiplicand at the beginning of the operation. The check is avoided because to do so would destroy whatever is in the accumulator, where the zero check takes place; also, the accumulator is not cleared as it is in fixed-point multiply.

Aside from these differences, the VLM, VMA and VDP instructions operate the same as the fixed-point operations. Refer to the detailed flow diagrams in the CPU Logic Diagrams manual to verify the differences and note the similarity between the fixed-point and variable-length instructions.

FLOATING-POINT ARITHMETIC

The range of numbers anticipated during a calculation may be extremely large, extremely small or, in some cases, unpredictable. Such situations make fixed-point arithmetic difficult to work with for two reasons:

1. The size of the number is limited by the size of the register (35 binary bits or 10 decimal digits).
2. The programmer must keep track of the point in all numbers throughout the calculation.

To meet the needs of large numbers and to automatically keep track of the point, an alternative set of arithmetic instructions, called floating-point arithmetic instructions, are available.

As the name "floating-point" implies, the binary point does not have to be lined up before each operation or remain in the same position at the end of the operation. Instead, it "floats" or is re-positioned during calculations in much the same manner as the decimal point is repositioned when calculating with a pencil and paper. Floating-point arithmetic instructions automatically position the operands to be used and deliver the result in correct form.

Scientific Notation

The principle on which floating-point arithmetic works is basic in mathematics and is called scientific notation. Before floating-point operations are described, a review of scientific notation may help toward a thorough understanding of how floating-point arithmetic is performed in the 7040-7044.

Principles: When a quantity is measured, the number generated is the number of units contained in the quantity. If the quantity is small, it is usually expressed directly; e.g., something "is 4 feet high" or "weighs 100 pounds." When dealing with large values, however, direct expression is often cumbersome. For example, the value which constitutes one coulomb, or the unit of static charge, is approximately 6, 300, 000, 000, 000, 000, 000 free electrons. This value is so large that it is seldom expressed in this manner, not only because it is cumbersome, but because it may very easily be expressed incorrectly by dropping one or more of the trailing zeros. To avoid direct expression of this quantity, a coulomb is usually defined as the unit of static charge present when 6.3×10^{18} free electrons are collected on a single body. The expression 6.3×10^{18} denotes exactly the same value as the number written out with all the trailing zeros, but it is much easier to state and not so susceptible of error.

Representation of value in the manner shown is referred to as scientific notation. This method of notation is arrived at by taking the scientific digits (coefficient) of a particular value and multiplying them by the radix of the number system being used, raised to a power (exponent) which will correctly express the magnitude of the number. Other examples of scientific notation are the velocity of light, expressed as 2.998×10^8 meters per second, and the angstrom unit, expressed as 1×10^8 centimeters. All these notations, if multiplied by the indicated power of 10, will give the value commonly associated with the measurement of the given quantity.

If the significant digits of a value expressed by scientific notation are shifted so that the decimal point falls in a different place, the accuracy of the expression can still be maintained by a corresponding change in the power to which the radix is raised. For example, all the notations below will yield exactly the same result if multiplied out:

2.998×10^8	$.2998 \times 10^9$
29.98×10^7	$.02998 \times 10^{10}$
299.8×10^6	$.002998 \times 10^{11}$
2998×10^5	$.0002998 \times 10^{12}$

It can be seen that, for each shift left of the number (assuming that the decimal point stays in a fixed position), the power of 10 must be reduced by 1 to maintain the equality of the expression. Similarly, for

every shift to the right, the value of the exponent is increased by 1. Shifting the significant digits of a value back and forth and making the corresponding changes in the power of the radix can be utilized to perform addition or any other arithmetic function. For example, assume that the following expressions are to be added:

$$3.75 \times 10^3$$
$$+\ 445 \times 10^2$$

Because the exponents of the radix terms differ, a direct addition cannot be performed. However, one of the terms can be shifted until the exponents are of the same value; then the significant digits may be added, and the radix term may be carried to the sum. If the first expression is shifted, the result is as shown below:

3.75×10^3 shifted right one place = 37.5×10^2

$$\begin{array}{l} 37.5 \times 10^2 \\ \underline{+445 \times 10^2} \\ 482.5 \times 10^2 \end{array}$$

Multiplying this notation out yields a result of 48,250, the same as would be obtained by obtaining the true value of each expression separately and then adding them. If the second expression were shifted, the result would be:

445×10^2 shifted left one place = 44.5×10^3:

$$\begin{array}{l} 44.5 \times 10^3 \\ \underline{+3.75 \times 10^3} \\ 48.25 \times 10^3 \end{array}$$

Multiplying 48.25×10^3 out also yields 48,250, the correct result. From this simple example, it can be seen that it is necessary only to make the exponents of the radixes the same value by shifting the significant digits one way or the other and then performing the desired arithmetic operation.

The principle of scientific notation can be summarized by stating that it uses two factors to indicate the magnitude of a measured value. One factor is the radix raised to a power (either positive or negative), and the second factor is the significant digits of the value. Changing one of these factors requires a corresponding change in the other to maintain the validity of the expression. These same rules may be applied to the binary number system.

Notation with Binary System: Because the binary system uses a radix of 2, all forms of scientific notation are expressed in terms of powers of 2. In addition, since only symbols of 0 and 1 are employed in this system, the significant part of the notation will consist of a combination of 0's and 1's. For example, the value 256_{10} expressed in binary is 100 000 000(400_8). If this binary number is considered to be an integer, scientific notation of the value would be as shown below:

$$100\ 000\ 000 \times 2^0$$

Of course, this expression could be given in many forms, all equal in value, by shifting the bits of the expression and changing the exponent of the radix 2. For example, all the following expressions are equal to 100 000 000:

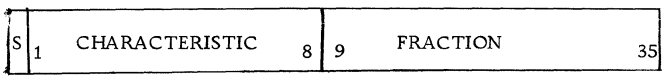
$$010\ 000\ 000 \times 2^1$$
$$000\ 001\ 000 \times 2^5$$
$$000\ 000\ 00.1 \times 2^9$$

The last expression has shifted the number so that it becomes a fraction, but no difficulty is encountered, since a binary fraction of this magnitude equals $\cdot 5_{10}$, or 1/2. The 9th power of 2 equals 512_{10}, and 1/2 x 512 yields a result of 256_{10}, the original value.

This type of notation is adequate for paper and pencil, but in a computer a different way of expressing the power of the radix is necessary. Since the 7040-7044 is a binary machine, the power of the radix must also be expressed in binary. This 2^9 power will appear in some other form inside the machine, although the power indicated is still 2^9.

Floating-Point Data

It has already been stated that floating-point arithmetic in the 7040-7044 uses operands expressed by scientific notation. Also, scientific notation has been defined as the significant digits of a value multiplied by the power of the radix. All that remains now is to show exactly how the power of the radix and the significant digits (or bits in binary) are expressed in the 7040-7044. The format for a floating-point data word is as follows:



Bit positions 1-8, referred to as the characteristic of the word, indicate the power of 2 to which the significant bits are raised. It can be assumed that 2 is the radix involved, since the binary system is being employed. If a characteristic of all zeros is arbitrarily chosen to represent 2^0, the range of exponents possible with eight bit positions would be 2^0 - 2^{377}. However, this arrangement is impractical because it allows only positive exponents to be expressed, and it is desirable to express negative exponents as well. Therefore, the midpoint between the total number of exponents that can be expressed (400_8) has been arbitrarily chosen to represent 2^0. This value is 200_8. Thus, a positive power of 2 will be between the values 200_8 and 377_8, and a negative exponent will be between 0_8 and 177_8. For example, to express 2^9 as it is done in the machine, the exponent must first be changed from decimal to octal form. Thus, 2^9 in the decimal system equals 2^{11} in the octal system. The radix is understood to be 2, so only the power (11) need be expressed. If 2^0 equals 200_8,

then 211_8 equals 2^{11}. The actual appearance of the characteristic (in binary) which indicates 2^9 is as follows:

10 001 001

The characteristic part of the floating-point data word thus constitutes one of the two factors employed in scientific notation.

Bits 9-35 of the data word, called the fraction, constitute the significant bits of the value, or magnitude. The term fraction is used because the data contained in this part of the word is considered to be in fractional form; that is, a binary point is effectively located between bit positions 8 and 9, making all bits to the right of this point represent a value somewhere between -1 and +1. This fraction should not be confused with the fraction represented by a fixed-point data word. It is true that the numerical significance of these fractions is the same in that each position represents a power of 2, but the actual magnitude of the floating-point data word can be determined only after the fraction has been multiplied by the power of 2 indicated by the characteristic. In fixed-point data words, the magnitude of a number can be determined after the various powers of 2 present in a given word are added together.

The sign bit position of a floating-point data word represents the sign of the fraction; that is, if the sign bit is 0, the fraction portion is positive, and if the sign bit is 1, the fraction portion is negative.

A characteristic in the range 0_8 to 177_8 does not in itself indicate that the quantity is negative. To express a very small quantity may require a negative power of 2, but the quantity may still be positive. For example, to express the value of 1/8 in floating-point form requires a negative exponent (assuming that the fraction is 1/2). The smallest position exponent that can be expressed is 200_8 or 2^0, and multiplying this by 1/2 still yields a result of 1/2. To obtain the quantity 1/8, a characteristic of 176_8 is required when the fractional part of the data word is 1/2. The characteristic of 176_8 represents 2^{-2}, and multiplying this by 1/2 yields the desired quantity:

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

Similarly, a value of -1/8 would be shown as follows:

S	CHARACTERISTIC	FRACTION
1	176	400000000_8

This value is known to be negative because the sign bit is a 1. Thus, it is the sign bit, and only the sign bit, which determines the polarity of the value expressed.

As an example of a floating-point data word, assume that it is desired to express the value 256_{10}. This value may be represented in octal by 400 or in binary by 100 000 000. This term may be expressed in octal by 400 or in binary by 100 000 000. This term may be expressed in scientific notation by 100 000 000 x 2^0 or 000 000 000.1 x 2^9. Taking the latter case and placing 2^9 (2^{11} in octal) into the characteristic bit positions yields a result of 211_8. The fraction remains as is, and the sign bit is cleared; so the floating-point form of 256_{10} is as follows:

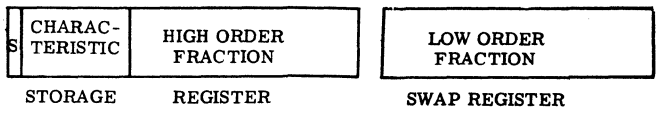
S	CHARACTERISTIC	FRACTION
0	211	.400000000

Arithmetic operations with floating-point data words are performed in much the same manner as the addition of terms in scientific notation. The characteristics are made the same by shifting one of the fractions and making the corresponding change in the value of the characteristic. The two fractions are then added (assuming addition is the operation called for), and the characteristic is assigned to the sum. Though a certain amount of "lining up" may be necessary before a floating-point operation may take place (the characteristics must be made equal), this process is performed automatically by the machine and is not the concern of the programmer. Also, the result of the operation will be a value which does not require further manipulation before another arithmetic operation can take place. Thus, the floating operation which occurs in floating-point arithmetic is really nothing more than an adjustment of the characteristic to keep the value being expressed in the proper order of magnitude.

Double Precision

When a fixed-point fraction is changed to floating-point form, the resulting characteristic and fraction may exceed 35 bits. In this case, additional hardware is available to accommodate the longer length fraction. The addressed operand is always placed in the storage register. When a double-precision addressed operand is required, the low-order fraction bits are housed in the swap register which is associated only with the storage register. In the case of the implied operand, the accumulator and the MQ register combine to house the double-precision number. The MQ register is assigned a characteristic 27_{10} less than that of the accumulator because the fraction contained in the MQ register in bits 9-35 is displaced 27 positions to the right of the accumulator binary point - the point just to the left of accumulator bit 9. No characteristic is assigned to the swap register because it serves to hold either the low-order addressed operand until it can be operated on or the partial result developed during an arithmetic process. However, the MQ register always reflects the result of an operation; a separate characteristic must be assigned because it is required to be very accurate in a floating-point operation, and these low-order bits must be dealt with separately.

The double-precision operand contained in the combined storage-swap register is as follows:



Bit S, the sign bit, is the sign of the entire fraction. When bit S is 0, the fraction is positive; when bit S is 1, the fraction is negative. The characteristic indicates the power of 2 to which the fraction is raised. In double-precision, the fraction is 54 bits in length; storage register bits 9-35, on the high-order fraction bits, and the swap register, which is only 27 bits long, form the low-order fraction. The accumulator-MQ register double-precision operand is shown as follows:

The only difference between the accumulator-MQ register operand and the storage-swap register operand is the existence of a characteristic for the low-order fraction. Note that the MQ register sign bit is not used.

Floating-Point Spill

During the execution of a floating-point operation, the resultant characteristic in either the accumulator or MQ register may exceed eight bit positions in length. The existence of such a condition means that machine capacity has been exceeded: machine capacity is exceeded when the exponent goes beyond 377_8 or below 0_8. When the characteristic goes beyond 377_8, a condition known as floating-point overflow is said to exist. Similarly, if the characteristic tries to go below 0_8, a condition known as floating-point underflow exists. These conditions are referred to collectively as floating-point spill.

Overflow and underflow may occur in either the accumulator or the MQ register. Upon sensing the existence of either condition, the processing unit places the address of the instruction causing the condition plus 1 into bits 21-35 of location 00000. In addition, one of bits 14-17 of location 00000 is set to record the cause of the spill.

Normalizing

When a floating-point data word is being dealt with, it may be in one of two forms, normalized or unnormalized. A normalized number is one that contains the binary point of the fraction just to the left of the most significant bit. Since the binary point of the fraction is considered to be just to the left of accumulator bit 9 in a floating-point data word, bit 9 must contain a significant bit if the number is to be in normalized form; that is, bit 9 must contain a 1. Therefore, the absolute magnitude of the fractional part of a floating-point data word must be greater than or equal to 1/2, but less than 1 if the number is in normalized form. If the most significant bit is not contained in bit 9, the number is said to be unnormalized. Normalizing can be thought of as eliminating leading zeros from a fraction.

At the completion of an arithmetic operation, the result may be in either normalized or unnormalized form. Certain instructions in the floating-point arithmetic class of the 7040-7044 contain the option to normalize the result if so desired. When this is done, the fraction is shifted left until a significant bit is contained in the accumulator bit 9 position. However, to maintain the value of the expression, the characteristic must be reduced by 1 for each shift to the left that occurs. As an example, assume the result of an arithmetic operation appeared in the combined accumulator-MQ register as shown:

$$0.\ 10\ 001\ 011\ 000\ 111\ \ldots\ldots\ldots 0_2$$

The first eight bits of the accumulator contain the characteristic, 213_8; bits 9-35 of the accumulator and 9-35 of the MQ register contain the fraction, 070000000000000000_8. This expression is in unnormalized form because the fraction contains leading zeros. To normalize the fraction, the fraction is shifted left three places, with the bits leaving accumulator bit 9 being lost, and to maintain the equality of the expression, the characteristic is reduced by 3. The normalized number becomes:

$$0.210.700000000000000000_8$$

The value of the expression is maintained in both cases; however, the leading zeros have been eliminated from the fraction in the normalized form. When the result of an arithmetic operation is to be normalized, the normalizing process takes place automatically after the final result has been computed. Normalization is specified by a positive sign (S bit is 0) in the floating-point instruction word.

At this point, it may seem desirable to always have results appear in the normalized form. This would seem true because, as leading zeros are shifted out of the fraction, low-order bits enter the accumulator from the MQ register, thus increasing the accuracy of the answer. However, there are instances when it is desirable to perform an unnormalized operation. For example, if the values being dealt with contained very small characteristics (large negative powers of 2), a series of operations could cause accumulator underflow when normalizing takes place. If the magnitudes of the numbers are known to be very small, accumulator underflow may be avoided by leaving the answer in unnormalized form.

Consider the MQ register after a floating-point operation. It may contain an expression whose characteristic is always 27_{10} less than the accumulator characteristic. To maintain the difference in characteristics between the low-order MQ register fraction bits and the high-order accumulator fraction bits, normalizing is performed before the MQ register characteristic is computed.

Zero Fraction

A floating-point number having a zero fraction can be treated in a variety of ways because the significance of a zero fraction operand depends on the arithmetic process to be performed. In addition and subtraction, a zero fraction operand just means that the fraction portion of the answer is identical with the non-zero fraction operand. The result of the arithmetic is meaningful. In the machine, a zero fraction operand has no effect on the operation; the airthmetic is performed, allowing normalization of the non-zero operand fraction if specified. Naturally, if both operands contain a zero fraction, the answer has no meaning and can never be normalized. Such a situation, however, is highly improbable.

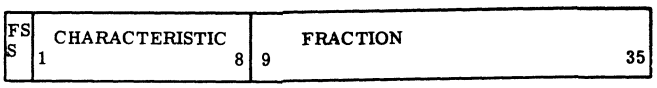
In multiplication, a zero fraction has a vastly different meaning and is therefore treated quite differently. In multiplication, a zero fraction multiplier results in a product containing a zero fraction: anything times zero equals zero. Likewise, a zero raised to some power is still zero. It serves no purpose to perform the operation because the result will be meaningless. Also, a zero fraction can never be normalized. Consequently, in single-precision multiplication, a zero fraction multiplier causes the operation to be terminated and the characteristic portion of the accumulator and MQ registers, which receive the result, to be cleared. In double - precision multiplication, the multiplicand is checked for a zero fraction. Effectively, a multiplicand with a zero fraction has the same meaning as a multiplier with a zero fraction: the result fraction will be zero. Consequently, a zero multiplicand fraction in double-precision multiplication causes the operation to be terminated and the accumulator and MQ register characteristic portions to be cleared. In addition, a better use of hardware is realized by checking the multiplier in one case and the multiplicand in the other. The end result is a shortening of the time necessary to accomplish floating-point multiplication.

When dealing with division, the divisor or the dividend could contain a zero fraction. Each case has a different meaning and is therefore treated differently. The treatment, however, applies to both single-and double-precision operations. If the divisor has a zero fraction, the quotient cannot be determined; a divide-check condition results, and the operation is ended. The dividend, however, remains unaltered in this case. When the dividend contains a zero fraction, the quotient will be zero. Since the quotient will have no meaning, the operation is ended. However, in this case, the associated characteristic positions of the accumulator and MQ registers, which hold the result of a division, are cleared.

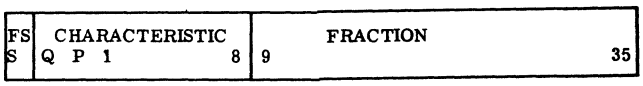
The above discussion pertains only to zero fraction operands. There remains the condition of the result of a floating-point operation containing a zero fraction. Since in multiplication and division the result is also influenced by zero fraction operands, these cases have already been covered. Only addition and subtraction, then, have not been discussed. In either of these operations, a zero fraction result causes the associated characteristic to be cleared and the operation terminated.

Single-Precision Floating-Point Addition and Subtraction

For single-precision floating-point addition in the 7040-7044, the addressed operand is placed in the storage register, which has the following format:



The sign bit serves as the fraction sign; the characteristic, to indicate the power of 2 to which the fraction is raised; the fraction, to express the significant bits of the quantity. The second implied operand is the accumulator, which has the following format:



In the accumulator, bits Q and P are interpreted as part of the characteristic. Field meaning is identical with that of the storage register.

Single-precision floating-point addition is accomplished by adding the storage register contents to the accumulator contents. The result is placed in the accumulator. Addition is algebraic; sign determination is independent of the actual addition. The result sign is the sign of the largest operand.

In the addition process, initial action involves determining which operand is larger; the larger operand is placed in the storage register. Therefore, whenever the accumulator is the larger operand, it is placed in the storage register, and the storage register operand is placed in the accumulator. Although the original value of accumulator bits Q and P are used in determining which operand is larger, they are not transferred to the storage register when the accumulator is found larger. In addition, in this case, bits Q and P are cleared when the storage register contents are placed in the accumulator.

After the larger operand is in the storage register, the characteristics are equalized. This action simply involves subtracting the accumulator characteristic from the storage register characteristic and placing the difference in the shift counter. The accumulator fraction, which will always be the smaller operand at this point in the addition, is shifted right the number of times specified by the shift counter . High-order fraction zeros are introduced into bit 9 with each right shift. Bits shifted out of accumulator bit position 35 enter MQ register bit position 9. During equalization then, the MQ register can become part of the smaller fraction. Bits shifted out of MQ register bit position 35 are lost.

When the characteristics are equalized, addition of the fractions can begin. However, two types of addition are possible: true addition and complement addition. A comparison of the storage register and accumulator signs determines which type to perform. If the signs are alike, true addition is performed; if the signs are unlike, complement addition is performed. With the latter type, the 1's complement of the accumulator fraction is added to the storage register fraction. Whichever type is performed, the result is placed in the accumulator.

The true addition of two 27-bit numbers can result in a 28-bit sum: the true addition of two fractions can yield a mixed number. However, only a fraction can be expressed in the result. In such a case, the result must be shifted right one position, and the characteristic must be updated. To simplify the equalizing action necessary when a 28-bit sum results during true addition, the storage register characteristic, which is the result characteristic, is transferred to the adder with the storage register fraction at the time of addition. A carryout of bit 9 (a 9 carry) is allowed to propagate to characteristic bit 8, thereby automatically updating the characteristic. The result fraction is then shifted right one position; shifting at this point involves the combined accumulator MQ register fraction. Accumulator bit 9 is then set to 1 to complete the equalizing action.

Complement addition is really subtraction. In this case, a 9 carry indicates that the storage register fraction was larger than the accumulator fraction and that the value in the accumulator is only a partial result. A 9 carry in complement addition must therefore be added to the partial result to get the true result. It is possible at this time for the MQ register to contain significant bits of the partial result because of the equalizing action taken before the fraction addition was initiated. Therefore, the MQ register is checked for a value of zero. If it does equal zero, the 9 carry is added to accumulator bit 35; if it does not equal zero, the 9 carry is added to MQ register bit 35.

No matter which type of arithmetic is performed, the sum of a floating-point addition is in the combined accumulator-MQ register. The low-order fraction bits or least significant portion of the sum appear in the MQ register fraction, and the high-order fraction bits or most significant portion of the sum appear in the accumulator. The result characteristic appears in the accumulator, and the MQ register is assigned a characteristic 27_{10} less than the accumulator characteristic. In addition, the MQ register sign is set to the accumulator sign.

The option to normalize the result is provided in single-precision addition. Normalization is specified by a positive instruction word sign. When normalization is specified, accumulator bit 9 is inspected. If it is a 0, the combined accumulator-MQ register fraction is shifted left one position. A zero is introduced into MQ register bit 35, and the characteristic is reduced by 1. This action is repeated until a 1 enters accumulator bit 9. Note that normalization, if specified, is completed before the MQ characteristic is computed.

Single-precision subtraction is identical with single-precision addition, except the sign of the addressed operand is inverted before it enters the storage register.

Machine Action

The action taken in the 7040-7044 to perform single-precision addition and subtraction is as follows:

1. Determine which operand is larger:
   a. Reset MQ register.
   b. Transfer storage register bits 1-35 to adder.
   c. Transfer complement of accumulator characteristic (Q-8) to adder.
   d. Add characteristics by generating a 1 carry to adder position 8.
   e. Place result in accumulator bits Q-8.
   f. Check for Q carry:
      (1) If a Q carry is present, the accumulator is equal to or smaller than the storage register; therefore, no operand interchange is necessary, so proceed to step 3, c (true difference in characteristics is in the adder at this point).
      (2) If no Q carry is present, the accumulator is greater than the storage register; proceed to step 2.
2. Operand interchange action (larger number to SR):
   a. Simultaneously transfer accumulator sign to storage register sign position, and storage register sign to accumulator sign position.

b. Transfer accumulator bits 1-35 to storage register.
c. Transfer adder bits Q-35 to accumulator.

3. Determine characteristic difference:
   a. Transfer complement of accumulator bits 1-8 to adder.
   b. Generate a 1 carry to adder position 8.
   c. Check adder bits 1 and 2:
      (1) If either bit is a 1, the characteristic difference is equal to or greater than 100_8, so accumulator bits Q-35 are cleared; proceed to step 4.
      (2) If neither bit is a 1, the characteristic difference is less than 100_8, and adder bits 3-8 are transferred into the shift counter.
4. Equalize characteristics:
   a. Check shift counter for a value of 0:
      (1) If SC = 0, go to step 5.
      (2) If SC $\neq$ 0, go to step 4, b.
   b. Shift MQ register bits 9-35 right one position.
   c. Shift accumulator bit 35 right one position into MQ register bit 9.
   d. Shift accumulator bits 9-35 right one position, and introduce a 0 into vacated accumulator position 9.
   e. Step shift counter.
   f. Return to step 4, a.
5. Compare operand signs:
   a. Compare accumulator and storage register signs:
      (1) If alike, perform true addition (step 6, a).
      (2) If unlike, perform complement addition (step 6, b).
6. Addition:
   a. True addition:
      (1) Transfer storage register bits 1-35 and accumulator bits 9-35 to adder, and add.
      (2) Place result in accumulator bits Q-35.
      (3) Check for a carryout of adder bit 9:
         (a) If no carry is present, normalize fraction if specified (step 7); if normalization is not specified, end operation (step 8).
         (b) If a carry is present:
            1. Let carry propagate to adder bit 8.
            2. Shift right one position MQ register bits 9-35.
            3. Shift right one position accumulator bit 35 into MQ register bit 9.
            4. Shift right one position accumulator bits 9-35.
            5. Place a 1 in accumulator bit 9.
            6. Normalize fraction if specified (step 7); if normalization is not specified, end operation (step 8).
   b. Complement addition:
      (1) Transfer storage register bits 1-35 and complement of accumulator bits 9-35 to adder.
      (2) Place result in accumulator bits Q-35.
      (3) Check for a carryout of adder bit 9:
         (a) If no carry is present, complement accumulator bits 9-35; then normalize fraction if specified (step 7); if normalization is not specified, end operation (step 8).
         (b) If a carry is present, transfer storage register sign to accumulator sign position (SR > AC).
      (4) Check MQ fraction for a value of 0:
         (a) If MQ = 0, transfer accumulator bits 9-35 to adder, generate a carry to adder bit 35, and place result in accumulator; then either normalize fraction (step 7) or end operation (step 8).
         (b) If MQ $\neq$ 0 (subtract MQ from imaginary extension of SR):
            1. Transfer MQ register bits 9-35 to storage register.
            2. Transfer storage register bits 9-35 to adder, thereby placing MQ register fraction in adder.
            3. Transfer accumulator bits 9-35 to storage register.
            4. Transfer adder bits 9-35 to accumulator (MQ register fraction is now in accumulator).
            5. Transfer complement of accumulator bits 9-35 to adder.
            6. Generate a 1 carry to adder bit 35, and place result in accumulator.
            7. Transfer storage register bits 9-35 to adder.
            8. Transfer accumulator bits 9-35 to storage register.
            9. Transfer adder bits 9-35 to accumulator.
            10. Transfer storage register bits 9-35 to MQ register bits 9-35.
            11. Normalize fraction (step 7) if specified; if not, end operation (step 8).

7. Result normalization:
   a. Inspect accumulator bit 9.
   b. If accumulator bit 9 = 1, proceed to step 8.
   c. If accumulator bit 9 = 0:
      (1) Shift accumulator bits 9-35 left one position, and document shift counter, making it all 1's.
      (2) Shift MQ register bit 9 left one position into accumulator bit 35.
      (3) Shift MQ register bits 10-35 left one position.
      (4) Place a 0 in MQ register bit 35.
      (5) Transfer accumulator bits Q-8 to adder.
      (6) Add 1's to adder bits Q, P, 1, and 2.
      (7) Transfer shift counter contents to adders.
      (8) Transfer adder bits Q-8 to accumulator bits Q-8.
      (9) Repeat step 7, a.
8. End operation:
   a. Transfer accumulator bits Q-8 to adder.
   b. Add 1's to adder bits 0-3 and 6 and 8.
   c. Transfer accumulator sign to MQ register sign.
   d. Transfer adder bits 1-8 to MQ register positions 1-8.
   e. Check for accumulator and MQ fraction = 0:
      (1) If both = 0 and normalization is specified, reset accumulator bits Q-35 and MQ register bits 1-35.
      (2) If either = 0, or if neither = 0, the operation is complete.

Single-Precision Multiplication

In single-precision multiplication in the 7040-7044, the addressed operand is the multiplicand and is placed in the storage register. The format of the multiplicand is identical with the storage register operand in single-precision addition. The implied operand is the multiplier and is in the MQ register, which has a format identical with the storage register. Single-precision multiplication, then, is accomplished by multiplying the storage register contents by the MQ register contents.

The product of a single-precision multiplication appears in the combined accumulator-MQ register. The high-order fraction bits appear in the accumulator; the low-order fraction bits, in the MQ register. The product or result characteristic appears in the accumulator, and the MQ register is assigned a characteristic 27_{10} less than the accumulator characteristic. Multiplication is algebraic; therefore, like signs yield a positive result sign, and unlike signs, a negative result sign. The signs of the accumulator and the MQ register are set to the algebraic sign of the result.

Initial action in multiplication involves determining whether the operation is normalized or unnormalized. If the operation is unnormalized, machine circuits are automatically set up for the action and the product sign is determined. However, if the operation is normalized, the combined accumulator and MQ register fraction is checked for a zero value. The presence of a zero value causes the operation to be ended and the product sign to be determined. The absence of a zero value results in setting up the machine circuits to accomplish the multiply and in determining the product sign.

Once the decision is made to perform the multiplication, the result characteristic is computed. When multiplying floating-point numbers, the characteristics are added. Therefore, the MQ register characteristic is added to the storage register characteristic. Since 200_8 is used to represent 2°, the sum of the characteristics is the result characteristic plus 200_8. Consequently, 200_8 is subtracted from the sum of the characteristics to get the real result characteristic. This value is placed in the accumulator.

After the result characteristic is computed and in accumulator bit positions Q-8, the fractions are multiplied. Multiplication in this case is identical with fixed-point multiplication in that it is a series of right shifts or additions and right shifts. The only difference between single-precision multiplication and fixed-point multiplication is the setting of the shift count. In the former, two 27_{10} bit fractions are involved and, therefore, the shift counter is set to 33_8; in the latter, two 35-bit operands are involved and, therefore, the shift counter is set to 43_8.

When the repetitive steps that constitute multiplication are repeated and the result is in the combined accumulator-MQ register, the result is normalized if normalization is specified. Normalization is specified when the instruction word sign bit is positive(0). However, in single-precision multiplication, provision is made to normalize only one position. Therefore, if accumulator bit 9 is a 0, the combined accumulator-MQ register fraction is shifted left one position. A trailing 0 is introduced into MQ register bit 35, and the characteristic is reduced by 1. When this action is completed, accumulator bit 9 is not inspected further.

The final action taken in single-precision multiplication is the computation of the MQ characteristic. At this time, the result characteristic is in the accumulator. Accumulator bits Q-8 are therefore transferred to the adder, where $27_{10}(33_8)$ is subtracted from the Q-8 value. The result is placed in MQ register bits 1-8. Completion of this transfer ends the operation.

Machine Action

The action taken by the 7040-7044 during the execution of a single-precision multiplication is as follows:

1. Determine whether a multiply is possible and, if so, initiate the action:
   a. Check to determine whether operation is normalized or unnormalized.
   b. If an unnormalized operation is specified, go to step e, but ignore not-zero-value contingency.
   c. If a normalized operation is specified, check combined accumulator and MQ register fraction for zero value.
   d. If a zero value is found, end operation and go to step f.
   e. If a zero value is not found:
      (1) Set SC to 33_8.
      (2) Transfer SR positions 1-35 to adder positions 1-35.
      (3) Transfer adder positions Q-35 to accumulator positions Q-35.
   f. Compare SR sign with MQ register sign.
   g. If alike, make accumulator sign positive.
   h. If unlike, make accumulator sign negative.
   i. Check accumulator positions Q-35 for zero value.
   j. If a zero value is found, make PR sign position positive.
2. Compute result characteristic:
   a. Transfer storage register bits 1-8 to adder.
   b. Transfer adder bits Q-8 to accumulator (positions Q-8).
   c. Transfer MQ register bits 1-8 to storage register positions 1-8.
   d. Simultaneously transfer storage register bits 1-8 and accumulator bits Q-8 to adder.
   e. Transfer result, adder bits Q-8, to accumulator bits Q-8; the accumulator now contains the result characteristic plus 200.
   f. Subtract 200 from accumulator characteristic:
      (1) Transfer accumulator bits Q-8 to adder.
      (2) Add 1 to adder bits Q, P, and 1.
      (3) Transfer result to accumulator positions Q-8.
3. Multiply fractions:
   Test MQ register bit 35:
   (1) If MQ 35 = 0:
      (a) Shift combined accumulator-MQ register fraction (accumulator bits 9-35 and MQ register bits 9-35) right one position.
      (b) Step shift counter (reduce SC value by 1).
      (c) Test shift counter for a value of 0.
      (d) If SC = 0, proceed to step 4; if SC $\neq$ 0, repeat step 3, a.
   (2) If MQ 35 = 1:
      (a) Add storage register fraction to accumulator fraction, and place result in accumulator.
      (b) Shift combined accumulator-MQ register fraction right one position.
      (c) Step shift counter.
      (d) Test shift counter for a value of 0.
      (e) If SC = 0, proceed to step 4; if SC $\neq$ 0, repeat step 3, a.
4. Normalize result, if specified (instruction word S bit = 0):

Check accumulator bit 9:

(1) If AC 9 = 1, proceed to step 5.

(2) If AC 9 = 0:

(a) Transfer accumulator bits Q-8 to adder.

(b) Add 1's to bits Q-8 (effectively subtracting 1 from accumulator characteristics).

(c) Place result in accumulator bits Q-8.

(d) Shift combined accumulator-MQ register fraction left one position.

(e) Proceed to step 5.

5. End operation:

(a) Transfer accumulator bits Q-8 to adder.

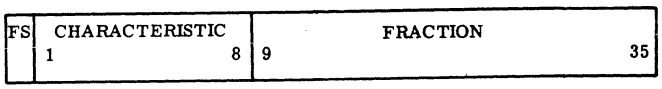
(b) Add 1 to bits Q, P, 1, 2, 3, 6, and 8 (effectively subtracting 33_8 from accumulator characteristic).

(c) Transfer result to MQ register bits 1-8.

(d) Transfer accumulator sign to MQ sign.

Single-Precision Division

During single-precision division in the 7040-7044, the accumulator serves as the dividend, and the storage register, which contains the addressed operand, as the divisor. Their formats are identical:



The quotient appears in the MQ register, and the remainder appears in the accumulator. Both formats are identical with that shown above. In division, the result characteristic is obtained by subtracting the division characteristic from the dividend characteristic. This result characteristic is placed in the MQ register. A remainder characteristic is also computed; it is the dividend characteristic minus 27_{10}.

Sign determination is governed by the rules of algebra. Therefore, like signs yield positive results; unlike signs, negative results. The remainder keeps the sign of the dividend.

If the dividend is equal to or greater than twice the divisor, division is not allowed to take place, and the instruction is terminated. Also, if the dividend fraction is 0, accumulator and MQ register bits 1-35 are reset to 0, and the accumulator sign is made positive. Further, the quotient characteristic is partially computed. In a single-precision division, the quotient or result characteristic equals $\overline{SR\ (1\text{-}8)} + \overline{AC\ (1\text{-}8)} + 200_8$.

When the dividend is equal to or greater than the divisor, a quotient greater than 1 is implied. As the actual division takes place, this quotient greater than unity would be shifted out of the MQ register and into accumulator bit 35. To make sure that this shifted quotient bit appears in the highest-order quotient bit position (MQ 9), the shift counter is decremented, and the dividend characteristic is increased by 1 before the division begins. When the dividend is less than the divisor, the quotient will not spill into the accumulator; therefore, the dividend characteristic does not have to be altered. In this case, the dividend fraction is shifted left one position and the shift counter is decremented.

In the single-precision division process, the complement of the dividend is subtracted from the divisor. During the execution of the subtraction, a check must be made for a "simulated adder 8 carry". This type of carry is the existence of a 9 carry due to a left shift and a second 9 carry due to the subtraction. When no simulated adder 8 carry exists, the result of the subtraction, which is in the adder, is placed in accumulator bits 9-35. This action constitutes a successful reduction and is accompanied by the placement of a 1 in MQ register bit 35. When a "simulated adder 8 carry" exists, a reduction is not possible and, therefore, nothing is done to MQ register bit 35. After each attempted subtraction, successful or unsuccessful, and until the shift counter is decremented to 0, the combined accumulator-MQ register fraction is shifted left one position.

After division of the fractions is completed, computation of the result characteristic is completed. The result characteristic is placed in MQ register bit positions 1-8. With this action accomplished, the original characteristic of the accumulator is determined, and the sign of the MQ register is set to the algebraic sign of the quotient (like signs = positive; unlike signs = negative). The final action taken is to reduce the original accumulator characteristic by 27_{10} if a remainder exists.

Machine Action

The action taken in the 7040-7044 during the execution of a single-precision division is as follows:

1. Prepare to divide:
   a. Set MQ register to 0.
   b. Set shift counter to 33_8.
2. Determine whether a divide can be performed:
   a. Shift combined accumulator and MQ register fraction right one position, thereby dividing the dividend by 2.
   b. Transfer complement of accumulator bits 9-35 to adder.
   c. Transfer storage register bits 9-35 to adder, and add.
   d. Check for a 9 carry:
      (1) If no 9 carry is present, the uncomplemented accumulator fraction is either equal to or greater than twice the storage register fraction value. Therefore, a divide will result in a quotient that equals machine capacity; the quotient will be greater than twice unity. End the operation by turning on the divide check indica-

tor and shifting the combined accumulator-MQ register fraction left one position.

(2) If a 9 carry is present, a divide can be performed, so proceed.

e. Shift combined accumulator-MQ register fraction left one position.

f. Check accumulator fraction for a value of 0:

(1) If the accumulator fraction is 0:

(a) Reset accumulator bits Q-8.

(b) Set accumulator sign positive, regardless of divisor sign.

(c) End operation.

(2) If accumulator fraction is not 0, continue.

3. Initiate computation of quotient characteristic:

a. Transfer storage register bits 1-8 to adder.

b. Transfer complement of accumulator bits Q-8 to adder, and add.

c. Place result in accumulator bits Q-8. These bits now contain the sum of the storage register characteristic and the complement of the accumulator characteristic ($SR + \overline{AC}$).

d. Transfer complement of accumulator bits Q-35 to adder.

e. Transfer adder bits Q-35 to accumulator. The accumulator characteristic bits now contain the complement of the result of adding the storage register characteristic and the complement of the original accumulator characteristic ($\overline{SR + AC}$), which equals the result characteristic less 200.

4. Check for the possibility of a quotient greater than 1:

a. Subtract accumulator fraction from storage register fraction. Since the accumulator fraction is already in complement form, just transfer accumulator fraction and storage register fraction to adder, and add them.

b. Check for a 9 carry:

(1) If a 9 carry is present, the storage register fraction > the accumulator fraction, thereby indicating that a quotient less than 1 will result. Therefore, shift accumulator-MQ fraction left one position, and step shift counter. Remember this 9 carry for step 5,b (no 9 into AC).

(2) If no 9 carry is present, the accumulator fraction > the storage register fraction, thereby indicating that a division will result in a quotient of unity, but less than 2; quotient will be 1 plus a fraction. In this case:

(a) Transfer accumulator Q-8 to adder.

(b) Add 1 to position 8.

(c) Place result in accumulator bits Q-8.

(d) Step shift counter.

5. Divide fractions:

a. Simultaneously transfer accumulator bits 9-35 and storage register bits 9-35 to adder, and add.

b. Check for a simulated adder 8 carry (a carryout of bit 9 due to addition after a carryout of bit 9 due to left shifting):

(1) If no 8 carry is present, proceed to step 5,c (AC > SR).

(2) If an 8 carry is present, shift combined accumulator-MQ fraction left one position (SR > AC); proceed to step 5,d.

c. Transfer result of addition to accumulator.

d. Put a 1 in MQ 35.

e. Shift combined accumulator-MQ fraction left one position. If a 1 is shifted out of position 9 at this time, it must be remembered for step 5,b.

f. Step shift counter.

g. Check shift counter for a value of 0:

(1) If SC = 0, complete characteristic computation.

(2) If SC $\neq$ 0, repeat step 5,a.

6. Complete characteristic computation:

a. Transfer accumulator bits Q-8 to adder.

b. Add 1 to position 1.

c. Place result in MQ register bits 1-8.

7. Compute accumulator characteristic:

a. Transfer accumulator bits Q-8 to adder.

b. Transfer storage register bits 1-8 to adder, then add.

c. Place result (original accumulator characteristic) in accumulator.

d. Transfer accumulator bits Q-8 to adder.

e. Add 1 to positions Q, P, 1, 2, 3, 6, and 8.

f. Place result in accumulator.

8. Determine result sign:

a. Compare accumulator and storage register signs.

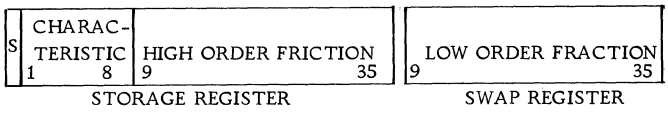
b. If they are alike, make MQ register sign positive (set it to 0).

c. If they are unlike, make MQ register sign negative (set it to 1).

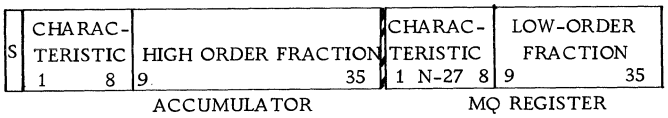
Double-Precision Addition and Subtraction

The addressed operand in double-precision addition is obtained from two sequential memory locations. The first memory location referenced must be an even-numbered location and contains the addressed operand characteristic and high-order fraction. The second memory location is automatically referenced and must be an odd-numbered location and one address higher than the first location referenced. In the

second location, only the fraction bits are used, and these bits form the low-order fraction of the addressed operand. The addressed operand is placed in the storage and swap registers:



The combined accumulator-MQ register is the implied operand in double-precision addition; its format is as follows:



Bits 1-8 in the MQ register are used as a characteristic for the low-order fraction. This characteristic is 27_{10} less than the accumulator characteristics.

In the addition process, the addressed operand is algebraically added to the implied operand. The characteristic of the larger operand is placed in the storage register (SR), and the difference in characteristics is placed in the shift counter. When the characteristic difference exceeds 100_8, the accumulator and MQ register fractions are cleared if the larger operand is the addressed operand. With a characteristic difference greater than 100_8 and the implied operand the larger operand, the accumulator-MQ register fraction is cleared. A characteristic difference of less than 100_8 results in equalization of the smaller operand. This action is accomplished by placing the fractions of the smaller operand in the accumulator and MQ registers and the larger operand in the storage and swap registers. The combined accumulator-MQ register fraction is then shifted left the number of places specified by the shift counter.

With the equalization completed, fraction true addition begins if the signs are alike. During fraction addition, the low-order fractions are initially swapped with their associated high-order fractions. Therefore, one low-order fraction is in the storage register, and the other is in the accumulator. The accumulator is then added to the storage register, with the result being the minor or low-order fraction sum, which is placed in the accumulator. Generation of a 9 carry during this phase of addition must be remembered.

After the low-order fraction addition, the register contents are again swapped: the accumulator with the MQ register, and the storage register with the swap register. Swapping involves only the fractions. Now the high-order fractions are in the storage register and accumulator. These registers are added. A carry 0 is used if the low-order fraction addition yielded no carry; a carry 1 is used if the low-order fraction addition yielded a carry. The result goes to the accumulator. The result characteristic also goes to the accumulator.

After equalization, if the signs are unlike, the least significant or low-order fractions are subtracted using the 2's complement. Fraction swapping is performed identically with that in true addition, so that the difference in low-order fractions is obtained first. The result goes to the accumulator, and a 9 carry, if generated, is remembered.

The storage and swap and accumulator and MQ registers are interchanged, and the 1's complement of the accumulator fraction is added to the storage register fraction. The result is increased by 1 if a 9 carry is remembered from the low-order fraction subtraction. If the storage register fraction is larger, the subtraction is complete; the accumulator sign is set to the storage register sign. If the result is zero, the accumulator or result characteristic is cleared. If the accumulator fraction is larger, the combined accumulator-MQ register fraction is complemented: the MQ register contents are swapped with the accumulator contents; then, the 2's complement of the accumulator contents is obtained. A 9 carry is remembered. Accumulator and MQ register contents are again interchanged. The accumulator is complemented. A 1 is added to the result if a 9 carry is remembered from low-order fraction recomplementing.

After the result is in the accumulator and MQ register in true form, normalization, if specified, is performed. After normalization is completed, the MQ register characteristic is computed. This action marks the end of the operation.

Double-precision subtraction is identical with double-precision addition, except the sign of the addressed operand is inserted before it enters the storage register.

Machine Action

The action that takes place in the 7040-7044 during double-precision addition and subtraction is as follows:

1. Arrange addressed operand:
   a. Transfer storage register bits 9-35 to swap register.
   b. Transfer storage register bits 1-8 to adder positions 1-8.
   c. Transfer adder positions 1-8 to MQ register positions 1-8.
   d. Receive second operand bits 1-35 in storage register.
   e. Interchange storage register and swap register fractions.
   f. Transfer MQ register bits 1-8 to storage register bits 1-8.
2. Determine which operand is larger:
   a. Transfer storage register bits 1-8 and com-

ment of accumulator bits Q-8 to adder.
b. Add by generating a 1 to adder bit 8.
c. Check for a Q carry:
(1) If a Q carry is present, the storage register is transferred to the accumulator:
(a) Check adder bits 1 and 2:
1. If they are both 0, transfer adder bits 3-8 to shift counter.
2. If either or both are 1 (characteristic difference $\geq 100_8$), reset MQ register, and transfer adder bits Q-35 to accumulator(reset accumulator).
(2) If no Q carry is present, the accumulator >the storage register:
(a) Transfer accumulator bits 1-8 to storage register positions 1-8.
(b) Transfer adder bits Q-8 to accumulator positions Q-8; this is the characteristic difference in complement form.
(c) Transfer complement of accumulator bits Q-8 to adder.
(d) Add by generating a 1 to adder bit 8 to get true difference.
(e) Check adder bits 1 and 2:
1. If they are both 0, transfer adder bits 3-8 to shift counter.
2. If either bit is a 1, remember and proceed to next step.

3. Register Swap or Register Swap and Interchange:
a. If a Q carry was generated in step 2:
(1) Simultaneously transfer storage register bits 9-35 to swap register and swap register bits 9-35 to storage register.
(2) Simultaneously transfer storage register bits 1-35 to MQ register and MQ register bits 9-35 to storage register.
(3) Swap storage register fraction and accumulator fraction:
(a) Transfer storage register bits 9-35 to adder.
(b) Transfer accumulator bits 9-35 to storage register.
(c) Transfer adder bits 9-35 to accumulator.
(4) Swap storage register fraction and MQ register fraction:
(a) Transfer MQ register bits 9-35 to storage register.
(b) Transfer storage register bits 9-35 to MQ register.
b. If no Q carry was generated in step 2:
(1) Swap storage register and accumulator fractions:
(a) Transfer storage register bits 9-35 to adder.
(b) Transfer accumulator bits 9-35 to storage register.
(c) Transfer adder bits 9-35 to accumulator.
(2) Swap storage register and swap register fractions:
(a) Transfer storage register bits 9-35 to swap register.
(b) Transfer swap register to storage register.
(3) Swap accumulator and storage register signs.
(4) Swap storage register and accumulator fractions:
(a) Transfer storage register bits 9-35 to adder.
(b) Transfer accumulator bits 9-35 to storage register.
(c) Transfer adder bits 9-35 to accumulator.
(5) If characteristic difference is less than 100_8, swap storage register and MQ register fractions.
(6) If characteristic difference is greater than 100_8:
(a) Reset MQ register.
(b) Transfer adder bits Q-35 to accumulator (reset accumulator).
(7) Check shift counter:
(a) If SC $\neq 0$, proceed to step 4.
(b) If SC = 0, compare storage register and accumulator signs:
1. If the signs are alike, proceed to step 5.
2. If the signs are unlike, proceed to step 7.

4. Equalize fractions:
a. Shift combined accumulator-MQ fraction right one position. Because the low-order bits are in the accumulator, MQ register bit 35 goes into accumulator bit 9, and accumulator bit 35 is lost.
b. Step shift counter.
c. Check shift counter for a value of 0:
If SC = 0, compare storage register and accumulator signs:
(1) If signs are alike, proceed to step 5.
(2) If signs are unlike, proceed to step 7.

5. Add low-order fractions; then swap for high-order add.
a. Perform low-order add:
(1) Transfer storage register bits 9-35 to adder.
(2) Transfer accumulator 9-35 to adder, and add.
(3) Place result in accumulator bits 9-35.
(4) If a 9 carry is generated, remember it.
b. Swap
(1) Transfer swap register to storage register.

(2) Transfer storage register bits 9-35 to adder.
(3) Transfer accumulator bits 9-35 to storage register.
(4) Transfer adder bits 9-35 to accumulator.
(5) Swap storage register and MQ register values:
(a) Transfer storage register bits 1-35 to MQ register positions 1-35.
(b) Transfer MQ register bits 9-35 to storage register positions 9-35.
(c) Proceed to step 6.

6. Add high-order fractions:
a. Transfer storage register bits 1-35 to adder.
b. Transfer accumulator bits 9-35 to adder.
c. If a 9 carry was generated in step 5, add by generating a 1 to adder bit 35.
d. If no 9 carry was generated, add by generating a 0 to adder bit 35.
e. Place addition result in accumulator bits Q-35.
f. Check for a 9 carry:
(1) If a 9 carry is present:
(a) Shift combined accumulator and MQ register fraction right one position.
(b) Make accumulator bit 9 a 1.
(2) If no 9 carry is present, proceed to step g.
g. Check combined accumulator-MQ register fraction for a value of 0:
(1) If equal to 0, reset accumulator (Q-35) and end operation.
(2) If not equal to 0, check accumulator bit 9:
(a) If accumulator bit 9 = 1, end operation.
(b) If accumulator bit 9 = 0, normalize result (step 9).

7. Subtract low-order fractions; then swap for high-order subtract:
a. Transfer complement of accumulator bits 9-35 to adder.
b. Transfer storage register bits 9-35 to adder.
c. Add by generating a 1 to adder bit 35.
d. Transfer adder bits Q-35 to accumulator, and remember a 9 carry, if any.
e. Swap storage register and swap register fractions.
f. Swap storage register and MQ register:
(1) Transfer storage register bits 1-35 to MQ register.
(2) Transfer MQ register bits 9-35 to storage register.
g. Swap storage register and accumulator fractions:
(1) Transfer storage register bits 9-35 to adder.
(2) Transfer accumulator bits 9-35 to adder.
(3) Transfer adder bits 9-35 to accumulator.
h. Swap storage register value and MQ register fraction:
(1) Transfer storage register bits 1-35 to MQ register.
(2) Transfer MQ register bits 9-35 to storage register.
i. Check for a 9 carry from step d.
(1) If no 9 carry is present, proceed to step 8.
(2) If a 9 carry is present, proceed to step 8.

8. Subtract high-order fractions:
a. Transfer complement of accumulator bits 9-35 to adder.
b. Transfer storage register bits 1-35 to adder.
c. If a 9 carry is present, generate a 1 to adder bit 35.
d. If no 9 carry is present, generate a 0 to adder bit 35.
e. Place adder bits 9-35 in accumulator bits 9-35.
f. Check for and remember a 9 carry, if generated.
g. Transfer MQ register bits 9-35 to storage register.
h. If a 9 carry is present:
(1) Transfer storage register sign to accumulator sign.
(2) Check combined accumulator-MQ fraction for a value of 0.
(a) If value is 0, reset accumulator and end operation.
(b) If value is not 0, check accumulator bit 9:
1. If accumulator bit 9 = 1, end operation.
2. If accumulator bit 9 = 0, proceed to step 9.
i. If no 9 carry is present, the combined accumulator-MQ register fraction>the combined storage-swap register fraction, and the result must be complemented to obtain the true result.
(1) Transfer storage register bits 9-35 to adder.
(2) Transfer accumulator bits 9-35 to storage register.
(3) Transfer adder bits 9-35 to accumulator.
(4) Make a 2's complement correction to accumulator:
(a) Transfer complement of accumulator to adder.
(b) Add 1 to bit 35.
(c) Place result in accumulator.
(d) Remember a 9 carry, if generated.
(5) Place high-order difference in accumulator:

(a) Transfer storage register 9-35 to adder.
(b) Transfer accumulator bits 9-35 to storage register.
(c) Transfer adder bits 9-35 to accumulator.

(6) Complete 2's complement correction:
(a) Transfer storage register bits 1-35 to MQ register.
(b) Transfer complement of accumulator bits 9-35 to adder.
(c) If a 9 carry is remembered, add 1 to adder bit 35.
(d) If no 9 carry is remembered, add 0 to adder bit 35.
(e) Transfer adder bits 9-35 to accumulator.

9. Normalize result:
a. Check accumulator bit 9:
(1) If it is a 1, end operation and assign MQ characteristic.
(2) If it is a 0:
(a) Shift accumulator bits 9-35 left one position, and decrement shift counter.
(b) Shift MQ register bit 9 left one position into accumulator bit 35.
(c) Shift MQ register bits 10-35 left one position.
(d) Place a 0 in MQ register bit 35.
(e) Transfer accumulator bits Q-8 to adder.
(f) Add 1's to adder bits Q, P, 1, and 2.
(g) Transfer shift counter contents to adder positions 3-8, and add.
(h) Transfer adder bits Q-8 to accumulator bits Q-8.
(i) Repeat step 9, a.

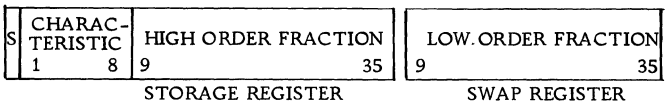
10. Assign MQ characteristic:

Check combined accumulator-MQ register fraction for a value of 0:

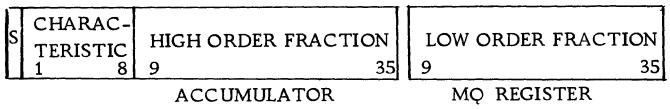
a. If not equal to zero:
(1) Transfer accumulator bits Q-8 to adder.
(2) Add 1 to adder bits Q, P, 1, 2, 3, 6, and 8.
(3) Transfer result to MQ register bits 1-8.
b. If equal to zero:
(1) Reset accumulator and MQ register.
(2) Transfer accumulator sign to MQ register sign position.

Double-Precision Multiplication

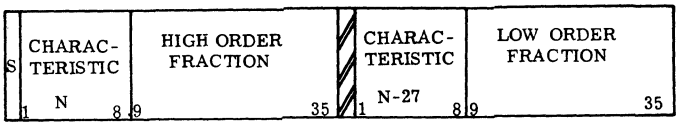
In double-precision multiplication, the multiplicand is placed in the storage and swap registers. The multiplicand characteristic and high-order fraction are obtained from an even-address memory location and appear in the storage register; the low-order fraction is obtained from the next-higher memory location and placed in the swap register:



The multiplier characteristic and high-order fraction bits are contained in the accumulator; the low-order fraction bits, in the MQ register:



The result appears in double-precision form in the accumulator and MQ registers, with the MQ register containing a characteristic 27_{10} less than the accumulator characteristic:



If the result fraction is zero, the accumulator and MQ register characteristics are set to zero. The sign of the result is the algebraic sign of the multiplication.

Double-precision multiplication is based on the algorithm

$$(A^{n}+B^{n-27})(C^{m}+D^{m-27})=(AC)^{n+m}+(BC)^{n-27+m}+(AD)^{n+m-27}+(BD)^{n+m-54}$$

However, the last term in the expression is not realized. In the following discussion, $(AC)^{n+n}$ is called P3, $(BC)^{n+n-27}$ is called P2, and $(AD)^{n+m-27}$ is called P1.

Initially, the multiplier fraction is checked for a 0 value. If it is 0, the accumulator and MQ register characteristics are cleared. If it is not 0, the action continues.

When both the multiplier and multiplicand significant fractions (high-order fractions) are 0, the result is 0. These major fractions are tested; if both are 0, the operation ends. Further, the accumulator and MQ register characteristics are cleared. If neither is 0, the result sign is determined and set in the accumulator sign bit.

With the result sign determined, the high-order multiplicand fraction (A) and the Low-order multiplier fraction (D) are multiplied. To accomplish this, the accumulator fraction is placed in a 27-bit register called the latch register. With this arrangement of operands effected, the accumulator fraction is cleared. If the MQ register is not 0 at this time, multiplication takes place, identical with single-precision multiplication. The result is 54 bits and appears in the combined accumulator-MQ register fraction. Only the accumulator portion (P1) is saved and is equivalent to

A x D in the algorithm. If the MQ register, however, is 0, the multiplication of A x D does not take place.

Next, the low-order multiplicand fraction (B) and the high-order multiplier fraction (C) are multiplied. The appropriate registers are arranged for the multiplication: the low-order fraction multiplicand to the storage register and the high-order multiplier fraction to the MQ register. The accumulator fraction is cleared, but P1 is saved. Multiplication is accomplished, and the result characteristic is determined. The result of the multiplication is placed in the combined accumulator-MQ register fraction, and the result characteristic is placed in the accumulator. The low-order result bits, which are in the MQ register, are dropped, and only the accumulator or high-order fraction bits are saved. These bits are equivalent to BC in the algorithm or P2.

The results of the two multiplications performed thus far have identical characteristics. At this time, these two partial products are added (P1 + P2), with their sum going to accumulator bits 9-35. If, as a result of this addition, a 9 carry is generated, it must be added to the major product, AC. Generation of a 9 carry at this time is remembered.

The two major fractions are now multiplied. However, the sum of the two partial products is left in the accumulator. In one operation, the result of AC plus P1 + P2 is realized. When the operation is complete, accumulator bits 9-35 contain the high-order result fraction bits and MQ register bits 9-35 contain the low-order result fraction bits. If a 9 carry is remembered from the addition of P1 and P2, 1 is added to the high-order result fraction. At this time, the multiplication is finished.

Normalization, if specified and if necessary, is now performed. In double-precision multiplication, the result can only be normalized one position. After this action, the MQ register characteristic is computed by subtracting 27_{10} from the accumulator characteristic. With the MQ register characteristic computed, double-precision multiplication is completed.

Machine Action

The action in the 7040-7044 during the performance of a double-precision multiplication is as follows:

1. Zero-test multiplier:
   a. If combined accumulator-MQ register fraction = 0, clear accumulator and MQ register fractions and end operation.
   b. If combined accumulator-MQ register fraction ≠ 0:
      (1) Simultaneously transfer storage register 9-35 to MQ register and MQ register 9-35 to storage register.
      (2) Transfer storage register sign to MQ register sign position.
   c. Set shift counter to 33_8.
   d. Transfer storage register bits 1-8 to adder.
   e. Transfer adder bits 1-8 to MQ register bit positions 1-8. At this point, the multiplicand characteristic and high-order fraction bits are in the MQ register; the multiplicand high-order fraction is in the swap register; the low-order multiplier fraction is in the storage register; the multiplier characteristic and high-order fraction bits are in the accumulator.
2. Zero-test high-order multiplicand fraction bits:
   a. Check combined accumulator-MQ register fraction for a value of 0.
   b. If combined accumulator-MQ register fraction = 0:
      (1) Reset accumulator characteristic.
      (2) Reset MQ register characteristic.
      (3) End operation.
   c. If combined accumulator-MQ register fraction ≠0:
      (1) Transfer storage register bits 9-35 to MQ register.
      (2) Transfer accumulator sign to storage register sign position.
      (3) Receive second operand from storage in storage register; block receipt of sign bit.
      (4) Swap storage register fraction and swap register fraction.
      (5) Transfer MQ register bits 1-8 to storage register bits 1-8. The MQ register still contains the sign of the multiplicand.
      (6) Compare storage register and MQ register signs:
         (a) If alike, make accumulator sign positive.
         (b) If unlike, make accumulator sign negative. At this point, the multiplicand characteristic and high-order fraction are in the storage register; the multiplicand low-order fraction is in the swap register; the result sign and the multiplier characteristic and high-order fraction are in the accumulator; the multiplicand characteristic and low-order multiplier fraction are in the MQ register.
3. Arrange operands so high-order multiplicand fraction can be multiplied by low-order multiplier fraction:

a. Transfer storage register bits 9-35 to adder.
b. Transfer accumulator bits 9-35 to storage register.
c. Transfer adder bits 9-35 to accumulator.
d. Transfer storage register bits 9-35 to latch register.
e. Transfer accumulator bits 9-35 to storage register.
f. Reset accumulator.
g. Check MQ register fraction:
   (1) If = 0, proceed to step 5.
   (2) If $\neq 0$, proceed to step 4. At this point, the multiplicand characteristic and high-order fraction bits are in the storage register; the multiplicand low-order fraction bits are in the swap register; the high-order multiplier fraction in the latch register; the accumulator fraction is zero; the low-order multiplier fraction is in the MQ register.

4. Multiply high-order multiplicand fraction by low-order multiplier fraction : P1
   a. Check MQ register bit 35:
      (1) If MQ 35 = 0, shift combined accumulator-MQ register fraction right one position.
      (2) If MQ 35 = 1, add storage register fraction to accumulator fraction, and then shift combined accumulator-MQ fraction right one position.
   b. Step shift counter.
   c. Check shift counter for a value of 0:
      (1) If SC = 0, proceed to step 5.
      (2) If SC $\neq 0$, repeat step 4, a.
5. Arrange operands so low-order multiplicand fraction can be multiplied by high-order multiplier fraction:
   a. Transfer swap register to storage register fraction and storage register fraction to swap register.
   b. Transfer storage register bits 1-35 to MQ register.
   c. Transfer latch register to storage register fraction positions.
   d. Transfer storage register bits 9-35 to adder.
   e. Transfer accumulator bits 1-35 to storage register.
   f. Transfer adder bits 9-35 to accumulator.
   g. Transfer storage register bits 9-35 to latch register.
   h. Set shift counter to 33_8.
   i. Transfer accumulator bits 1-35 to storage register.
   j. Reset accumulator. At this point, the storage register contains the multiplier characteristic and high-order fraction, the swap register contains the high-order multiplicand fraction, the latch register contains the first partial product, the accumulator contains the multiplier characteristic and a zero fraction, and the MQ register contains the multiplicand characteristic and the low-order multiplicand fraction.
6. Multiply high-order multiplier fraction by low-order multiplicand fraction, and determine result characteristic:
   a. Check MQ register bit 35:
      (1) If MQ 35 = 0, shift combined accumulator-MQ register fraction right one position.
      (2) If MQ 35 = 1, add storage register and accumulator fractions, and then shift combined accumulator-MQ register fraction right one position.
   b. Step shift counter.
   c. Check shift counter for a value of 0:
      (1) If SC = 0, proceed to step 7.
      (2) If SC $\neq 0$, repeat step a. At this point, the storage register, swap register, and latch register contents are the same as before the multiply. However, the accumulator now contains the multiplier characteristic and the high-order bits of the second partial product, and the MQ register contains the multiplicand characteristic and the low-order bits of the second partial product. These low-order bits are truncated.
   d. Position characteristics for adding:
      (1) Transfer storage register bits 1-8 to adder.
      (2) Transfer adder bits Q-8 to Accumulator.
      (3) Transfer MQ register bits 1-8 to storage register.
   e. Add characteristics:
      (1) Transfer storage register bits 1-8 and accumulator bits Q-8 to adder, and add them.
      (2) Transfer result to accumulator bits Q-8.
      (3) Subtract 200_8 from result by transferring accumulator bits Q-8 to adder and adding 1's to positions Q, P, and 1.
      (4) Place final result in accumulator bits Q-8. The accumulator now contains the final or result characteristic and the second partial product; the MQ register is considered to be empty; the storage register contains the multiplicand characteristic and high-order multiplier fraction; the swap register

contains the high-order multiplicand fraction; the latch register contains the first partial product.

7. Add partial products, and arrange operands so high-order multiplier can be multiplied by high-order multiplicand:
   a. Transfer storage register bits 1-35 to MQ register.
   b. Transfer latch register to storage register fraction positions.
   c. Add storage register and accumulator fractions, and place result in accumulator bits 9-35.
   d. Remember a 9 carry, if generated.
   e. Transfer swap register to storage register.
   f. Set shift counter to 33_8. At this point, the storage register contains the multiplicand characteristic and high-order fraction, the swap register contains the multiplicand high-order fraction, the latch register contains the first partial product, the accumulator contains the result characteristic and the sum of the first and second partial products, and the MQ register contains the high-order multiplier fraction.
8. Multiply high-order multiplicand fraction by high-order multiplier fraction:
   a. Check MQ register bit 35:
      (1) If MQ 35 = 0, shift combined accumulator-MQ register fraction right one position.
      (2) If MQ 35 = 1, add storage register fraction to accumulator fraction, and then shift combined accumulator-MQ register fraction right one position.
   b. Step shift counter.
   c. Check shift counter for a value of 0:
      (1) If SC = 0, proceed to step d.
      (2) If SC $\neq$ 0, repeat step a.
   d. Check for a 9 carry:
      (1) If a carry is remembered from step 7, c:
         (a) Transfer accumulator bits Q-35 to adder.
         (b) Generate a 1 carry to adder bit 35.
         (c) Transfer adder bits Q-35 to accumulator.
         (d) If a 9 carry was generated:
            1. Shift combined accumulator-MQ fraction right one position.
            2. Make accumulator bit 9 a 1.
      (2) If no 9 carry is remembered from step 7, c, proceed to step 8, e. At this point, the final characteristic is in the accumulator, and the final fraction is in the combined accumulator-MQ register.
   e. Normalize one position:
      (1) Check accumulator bit 9.
      (2) If accumulator bit 9 = 1, assign MQ characteristic.
      (3) If accumulator bit 9 $\neq$ 0:
         (a) Shift combined accumulator-MQ fraction left one position.
         (b) Transfer accumulator Q-8 to adder.
         (c) Add 1's to adder bits Q-8.
         (d) Transfer adder bits Q-8 to accumulator.
         (e) Assign MQ characteristic.
9. Compute MQ characteristic:
   a. Check combined accumulator-MQ fraction.
   b. If it $\neq$ 0:
      (1) Transfer accumulator Q-8 to adder.
      (2) Add 1 to adder bits Q, P, 1, 2, 3, 6 and 8.
      (3) Transfer adder bits 1-8 to MQ register.
      (4) Transfer accumulator sign to MQ register sign position.
   c. If it = 0:
      (1) Reset accumulator fraction.
      (2) Reset MQ register fraction.

Double-Precision Division

The addressed operand becomes the divisor in double-precision division and is placed in the storage and swap registers. Storage registers contents are obtained from some even-numbered memory location (Y); swap register contents are obtained from the next-higher location (Y + 1). Divisor format is identical with the double-precision multiplicand. The dividend is formed by the combined accumulator-MQ register and has a format identical with the double-precision multiplier. Double-precision division, therefore, is the division of the contents of the accumulator-MQ register by the contents of the combined storage-swap register. The answer appears as a double-precision quotient in the combined accumulator-MQ register and has a format identical with the result in double-precision multiplication. When the dividend fraction is zero, the characteristic portions of the accumulator and MQ registers are cleared and the accumulator sign is made positive.

The process of double-precision division is initiated by a check of the dividend for a value of 0. If a value of 0 is found, it is remembered. Next, a check is made to determine whether the dividend is greater than or equal to twice the divisor. In fixed-point division, the divisor must be greater than the accumulator portion of the dividend in order to con-

fine the quotient to the capacity of the machine registers. In floating-point division, the characteristics of the operands can be adjusted to compensate for machine register size limitations. Therefore, the divisor may be smaller than the dividend in floating-point division. In the 7040-7044, an arbitrary maximum ratio of 2:1 between the dividend and the divisor was chosen for ease of characteristic adjustment. If the dividend is greater than or equal to twice the divisor, the divide check indicator is turned on and the instruction is ended. If the dividend is less than twice the divisor, but a value of 0 is remembered, the characteristics of the accumulator and MQ registers are reset, the accumulator sign is made positive, and the instruction ends. If the dividend is less than twice the divisor and no value of 0 is remembered, operand equalization is performed. Before equalization takes place, however, the difference in characteristics is obtained.

A dividend equal to or greater than the divisor implies a quotient greater than 1. When the division takes place, this 1 is shifted into accumulator bit position 35. However, the machine needs the 1 to appear in MQ register bit position 9. To make sure that it does appear there, prior to division, the shift counter is decremented by 1 and the characteristic is increased by 1. When the dividend is less than the divisor, the quotient will not spill and therefore no adjustment is necessary.

Double Precision Division Algorithm

It will prove helpful to consider some of the limitations that the computer hardware will impose (such as a 35 bit adder instead of a 70 bit adder) on an attempt to divide double precision numbers.

First let us consider a long division problem in the decimal numbering system, such as we might encounter in day-to-day arithmetic.

$43\overline{)1500}$

The quotient and method of solution should be familiar.

```
      34 + 38/43      (Quotient)
43 )1500
    129
     210
     172
      38             (Remainder)
```

However, if we consider 43 to be a double "precision" number (the sum of single "precision" numbers 4 x 10' and 3 x 10°) and we consider 1500 to be a double "precision" number (the sum of 1 x 10^3, and 5 x 10^2) we can see that the division process could not be carried out in the normal way if we were restricted to single precision addition or subtraction. In the above example, we would not have been able to perform the double precision subtractions (circled).

Let's then explore the possibilities of arriving at a correct quotient without having to perform repetitive double precision trial subtractions.

If we perform the same division problem as above, only using 40 as a divisor instead of 43, we get:

```
      37
40 )1500
(+3) 120
     300
     280
      20
```

We could now say that we have reduced the dividend by 40 x 37, leaving us with a remainder of 20 to still be divided by 40.

However, if the original dividend were reduced by 43 x 37 instead of 40 x 37, our remainder would be more accurate. It is presently in error by 3 x 37. If we adjust the remainder of 20 above by subtracting 3 x 37 from it, we get:

```
      37     Quotient
40 )1500
(+3) 120
     300
     280
      20
    -111     (3 x 37)
     -91     Corrected Remainder
```

As a result of correcting our remainder, we see that further division is possible. Again using 40 as a divisor:

```
      -2
40 )-91
(+3) 80
      11
```

Once again our remainder is not as accurate as it could be. Instead of reducing -91 by 43 x 2 it was reduced by 40 x 2. We could correct our remainder of 11 by reducing it by 3 x 2.

This gives us:

```
      -2
40 )-91
(+3) 80
      11
     -6
      5
```

The quotient is -2 5/43.

Since this second division was really a continuation of the first division, we will add our second quotient to the first.

This gives us:

37 + (-2 5/43) = 34 38/43

This is identical to the quotient we calculated in the original example solved by conventional methods.

What advantage did we gain by the second method?

We solved the problem of having to perform double precision trial subtractions. Instead of having to subtract the double precision number 43 (or multiples thereof) from the dividend, we only had to subtract the single precision 4 x 10' (or multiples). The single precision 3 x 10° was only used to correct the remainder.

The advantage of this method becomes greatly magnified when it is remembered that there are 27_{10} "trial" subtractions in a single precision divide operation. Only one remainder correction will be necessary.

Now let's apply "the method" to a double precision divide as it would occur in the computer.

The double precision dividend will originally be contained in the AC and MQ. Let the dividend be:

$$(A \cdot 2^n) + (B \cdot 2^{n-27})$$

A and B are the high and low order fractions respectively, n and n-27 are the high and low order characteristics respectively.

We will say, for simplicity, that the dividend is A + B.

The double precision divisor will originally be contained in the SR and SWR.

Let the divisor be:

$$(C \cdot 2^m) + (D \cdot 2^{m-27})$$

C and D are the high and low order fractions respectively, m and m-27 are the high and low order characteristics respectively.

We will say, for simplicity, that the divisor is C + D.

Our computer problem then becomes:

$C + D\overline{)A+B}$

Let's go back and again solve our decimal division problem, this time using the above letters along with the numbers they correspond to.


```
          37   1st Quotient (Q1)
40(C) )1500    (A+B)
       120
       300
       280
        20   1st Remainder (R1)
```

This step could correspond to a single precision divide in the computer, namely:

1 - $\frac{A+B}{C}$ (Q_1)

The next step involved correcting the remainder computed during the first steps:

```
  20     1st Remainder (R1)
-111     3 x 37 Correction (Q1 . D)
 -91     (R1 - Q1D)
```

This step could correspond to a single precision multiplication followed by a single precision subtraction:

2 - $Q_1 \cdot D$
3 - $R_1 - Q_1D$

Next we performed a second single precision division:

```
          -2    2nd Quotient (Q2)
40(C) )-91     (R1 - Q1D)
        80
        11    2nd Remainder (R2)
```

This step could correspond to another single precision divide, using the values of obtained previously for the dividend:

4 - $\frac{R_1 - Q_1 D}{C}$ (Q_2)

In our decimal problem we then continued by correcting our second remainder. However, in the computer this step will not be performed because the value computed would be too small in magnitude to affect the final answer.

Out last step involved adding the second quotient to the first.

37 + (-2) = 35 Final Quotient

This last step would correspond to the single precision addition of the two computed quotients:

5 - $Q_1 + Q_2$ = Final Quotient

To summarize then, double precision division will be executed in the computer in five general steps:

1. $\frac{A+B}{C}$ (Q_1) , R_1 is saved
2. $Q_1 \cdot D$
3. $R_1 - Q_1D$
4. $\frac{R_1 - Q_1D}{C}$ (Q_2)
5. $Q_1 + Q_2$

These five steps are sometimes referred to as the double precision division ALGORITHM.

INDEX OPERATIONS

Index operations can be divided into two distinct areas: index arithmetic and address modification. The former involves activities which deal directly with the index facility; the latter pertains only to the modification of an instruction word.

Index Arithmetic

Index arithmetic is the establishment and maintenance of the index facility available in the 7040-7044 equipment. Basically, any operation which deals exclusively with placing information in or storing information from an index register falls within the realm of index arithmetic. Three index registers are provided in the CPU: index register A (XRA), index register B (XRB), and index register C (XRC). Instruction word bits 18 through 20 serve to specify the desired index register. A configuration in these bits of 001 specifies XRA; a configuration in these bits of 010, XRB; a configuration in these bits of 100, XRC: bit 18 specifies XRC, bit 19 specifies XRB, and bit 20 specifies XRA. Therefore, the index registers are referred to as XRA, XRB, and XRC, or 1, 2, and 4, respectively.

Figure 42 shows the data flow paths used in loading index registers in the 7040-7044 CPU. Before discussing the loading operations, it is necessary to define the portions of a machine word that can be used to load an index register. These portions are the address field and the decrement field. The address field is formed by bits 21 through 35 of the specified operand, and the decrement field is formed by bits 3 through 17 of the specified operand. A specified index register can be loaded with either the address field or the decrement field in either true form or 2's complement form. The specified operand may be obtained from an explicitly addressed core storage location, from the accumulator, or from an AXT instruction word. In the last case, only the address field in true form can be used. An index register may also be set by the contents of the instruction counter. This case, however, is somewhat special because a transfer operation is associated with it.

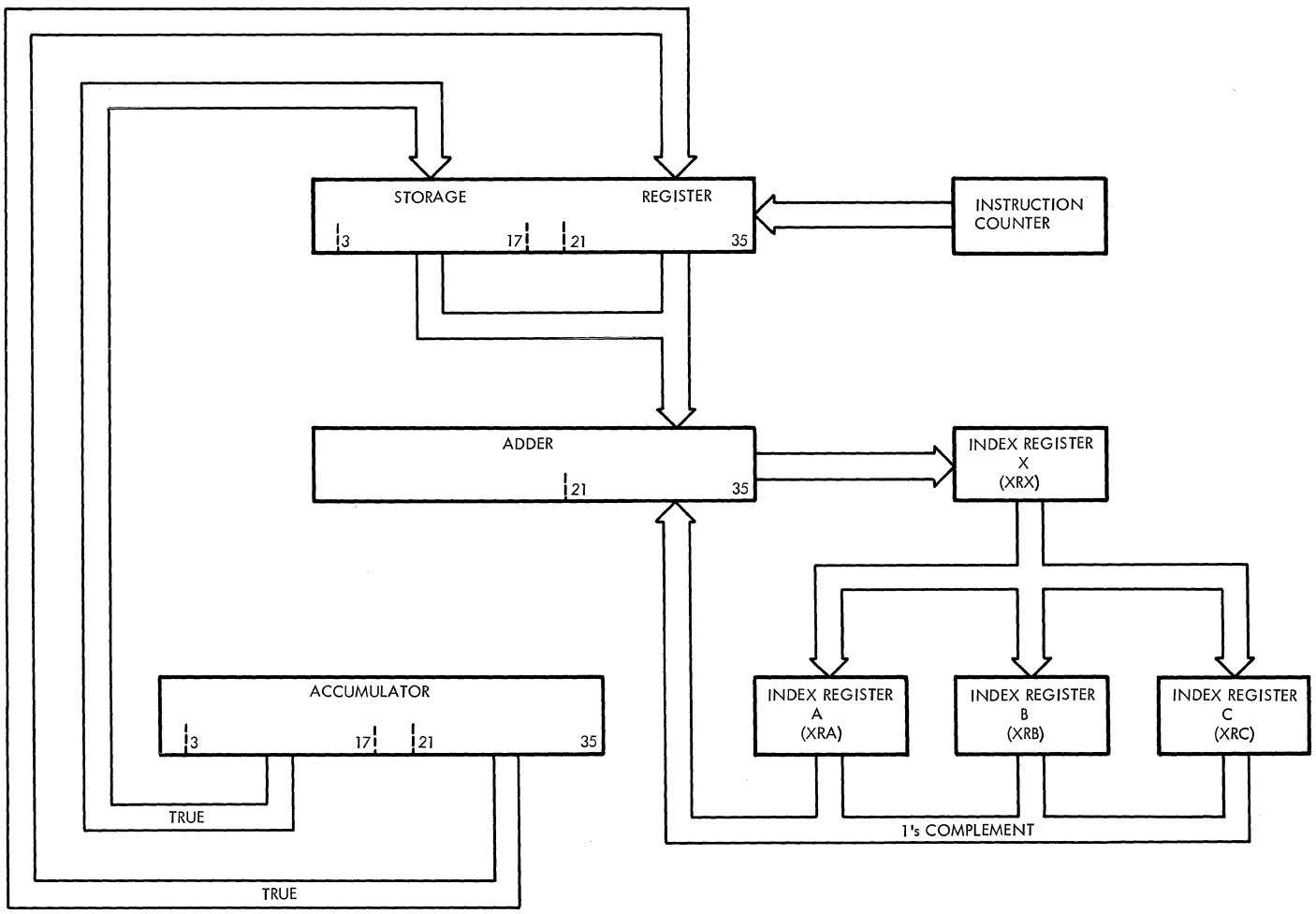


FIGURE 42. INDEX REGISTER LOAD PATHS

Consider the load paths used when the operand is obtained from an addressed memory location. The addressed operand is received from memory in the storage register. From the storage register, the field to be used is transferred to adder positions 21 through 35. If the address field is to be used, the transfer from the storage register to the adder is direct. If the decrement field is to be used, the bits are automatically right-shifted 18 places as they are transferred and enter the adder in positions 21 through 35.

From the adder, the field is transferred into index register X (XRX) which serves as an input buffer to the index registers. The contents of XRX are then placed in the specified index register. If the true form of the specified field is used, the operation is finished at this point. However, if the complement form of the specified field is desired, the 1's complement of the specified index register is now transferred back to the adder, where a hot 1 is added to adder bit position 35. The result of this action is the 2's complement of the original field. This 2's complement is now transferred to XRX and, from here, to the specified index register.

When using the accumulator to load an index register, the field to be used (address or decrement) is transferred to corresponding storage register positions in true form. From this point, the action is identical with that when using the storage register positions in true form. From this point, the action is identical with that when using the storage register as a load source.

One instruction is provided, Address to Index True (AXT), which allows the address field of that instruction to be used for index register loading. In this case, the instruction word is received from memory in the storage register. The address field is then transferred directly to adder positions 21 through 35. From these positions, the field is routed to XRX and, from here, to the specified index register. If the index register contents are to replace an address field, the contents of the specified index register enter the storage register in positions 21 through 35. Then, storage register bits 21 through 35 are transferred directly to corresponding memory data register positions. If the contents of the specified index register are to replace a decrement field, the transfer from the index register to the storage register is directly to positions 3 through 17. Storage register bits 3 through 17 are then transferred directly to memory data register positions 3 through 17.

When transferring the contents of an index register to the accumulator to replace an address field, the contents of the specified index register are initially transferred to storage register positions 21 through 35 in true form. From here they go to adder positions 21 through 35. Adder bits 21 through 35 are then transferred directly to accumulator bits 21 through 35. However, if the decrement field is to receive the information from the index register, the contents of the specified index register are transferred to storage register positions 3 through 17. From here, the transfer is directly to adder positions 3 through 17. From the adder, the information goes to the accumulator, entering in positions 3 through 17.

Address Modification

Address modification is the subtraction of the contents of a specified index register from the instruction word base address. The index register used is specified by bits 18 through 20 of the instruction word: bit 18 when a 1 specifies XRC, bit 19 when a 1 specifies XRB, and bit 20 when a 1 specifies XRA. Subtraction is performed with the 2's complement method. The instruction word base address is transferred from storage register positions 21 through 35 to corresponding adder positions. The 1's complement of the contents of the specified index register is transferred to adder positions 21 through 35. A hot 1 is then generated to adder position 35, and the quantities are added. The result is the difference between the quantities, which is the effective address. This effective address is transferred to the CPU address register and, from there, eventually goes to the memory address register.

Note: It is possible to select one, two, or three index registers for any operation that provides for the use of an index register. Multiple selection occurs when the instruction word tag field contains configurations of 011, 101, 110, or 111. During a load operation, multiple selection causes all selected registers to be loaded with the specified value. During an index store operation or an address modification operation, multiple selection results in a logical addition of the contents of the selected register: corresponding bit positions must be zero to result in a zero; otherwise, the result bit is a 1.

Assume index registers A, B, and C contain 14, 1, and 3, respectively. With a tag field of III, the value of 17_8, during address modification, is subtracted from the address field to form the effective address. During an index store, the value of 17_8 is stored in the specified field of the store location.

Addressing

The methods of addressing available in a particular machine are closely related to index operations,

although not a part of them. In the 7040-7044, two types of addressing are employed: direct and indirect. No matter which method is used, the important factor is obtaining the effective address. The effective address is the usable address, the address that identifies the core storage location containing the desired operand. The following description defines the effective address under all possible addressing combinations.

Each instruction that references core storage for an operand contains a base address in bits 21 through 35. If no indexing is specified, or if the index register specified contains all 0's, the instruction word base address becomes the effective address. However, if indexing is specified, and the specified index register contains some value other than 0, the difference between the instruction word base address and the contents of the specified index register forms the effective address. For example, assume the base address is 2000_8 and the contents of the specified index register are 100_8. The effective address is $2000_8 - 100_8$ or 1700_8.

Determining the effective address appears to become difficult when using indirect addressing. It is actually simple, however, when applied step by step. Indirect addressing is specified when instruction word bits 12 and 13 are both 1's. Assume and Add instruction is given, the instruction word specifies only indirect addressing, and the base address is 3000_8. In this case, the base address specifies a location in core storage whose contents are to be used to obtain the effective address; it is the indirect address. Assume location 3000_8 contains CLA 1000_8 with its tag field (18-20) all 0's: no indexing is specified. Address 1000_8 becomes the effective address: the contents of location 1000_8 form the addressed operand of the Add instruction. Note that only the address and tag field bits of the indirect address are used; these are bits 21 through 35 and 18 through 20. All other bits in the indirect address are not used.

Indirect addressing is also used when the instruction word specifies indexing as well as indirect addressing. Assume the base address is 3000_8 and the specified index register contains the value 200_8. The indirect address is then $3000_8 - 200_8$ or 2600_8. Assuming the indirect address contains ACL 1500_8 with no indexing specified, the effective address is 1500_8.

Assume the instruction word does not specify indexing, but the contents of the indirect address do. Let the instruction word base address be 2600_8. This location is the indirect address. Assuming the address field in the indirect address to be 1300_8 and the index register specified in the indirect address to contain 400_8, the effective address becomes $1300_8 - 400_8$ or 700_8.

Consider the case in which both the base address and the contents of the indirect address are indexed. Let the instruction word base address equal 4500_8 and the contents of the index register specified by the instruction word equal 700_8. The indirect address becomes $4500_8 - 700_8$ or 3600_8. Assuming the contents of the address field in the indirect address equal 7700_8 and the contents of the index register specified by the contents of the indirect address equal 500_8, the effective address becomes $7700_8 - 500_8$ or 7200_8.

The effective address in all cases is the final address computed. When computing an effective address, take the instruction word base address (bits 21-35) and apply indexing if specified to get the indirect address. If indexing is not specified, use the instruction word base address as the indirect address. When the indirect address is determined, examine its contents to get the effective address. Take the address field in the indirect address: (1) if no indexing is specified, use the contents of this field as the effective address; (2) if indexing is specified, subtract the contents of the specified index register from the address field in the indirect address to get the effective address.

A very important point to remember is that indirect addressing is effected only by bits 12 and 13 of the instruction word. Although the contents of the indirect address can have a 1 in positions 12 and 13, these positions are not decoded.

When the instruction counter is used to load an index register, instruction counter contents are transferred to storage register positions 21 through 35. From here they are sent directly to adder positions 21 through 35. Adder positions 21 through 35 then load XRX, and the contents of XRX are transferred to the specified index register. After the index register is loaded, the 1's complement of its contents is transferred to adder positions 21 through 35. Here, a 1 is added to position 35 to produce the 2's complement. This value is then transferred via XRX to the specified index register. The transfer action associated with this operation is described under transfer operations.

Figure 43 shows the data paths used when transferring information from an index register to some other register. In these operations, the contents of a specified index register may go to either core storage or the accumulator. The information in these operations is always in true form and can be placed in either the address field or the decrement field of the specified location.

When transferring the contents of an index register to storage, the action is as follows. Instruction word bits 18 through 20 specify the index register to be used. The contents of the specified index register are then transferred to the storage register.

TRANSFER OPERATIONS

A data processing system is used by executing programs which basically consist of various routines. Each routine is formed by a sequence of instructions which are placed in core storage in ascending sequential locations. By extension, associated routines (all routines forming a single program) are found in sequential locations. Fetching of instructions is controlled by the CPU instruction counter. Initially, a value is placed in the instruction counter which specifies the first location to reference in the first routine to be performed. The contents of the location specified by the instruction counter are transferred into the CPU and decoded as an instruction, and the desired operation is performed. The instruction counter is incremented by 1, and the new value, which is the next sequential address, is the address of the next instruction to be executed. The sequence of fetching and stepping in a sequential pattern continues until it is altered by placing a new value in the instruction counter which is out of sequence with the preceding values. Since the instruction counter controls the instruction fetched for execution, it also controls the routine to be executed. Inserting in the instruction counter a new value which is out of sequence with the preceding values is called a transfer of program control. The inserted value generally is the first location to reference in a new routine. Program control is therefore transferred from one routine to another.

It is possible for normal instruction counter stepping to transfer program control to a new routine. During the execution of a given program, however, it may be desirable to skip the next sequential routine or even several routines; this action of skipping one or more routines is termed a transfer operation. In addition, it may even be desirable to provide multiple paths through a single routine. Since provision is made to skip routines, the same instructions can be used to choose one of the multiple paths that may exist in a single routine.

Transfer operations may be divided into two areas: unconditional and conditional. Unconditional transfers effect the change of instruction counter contents regardless of existing conditions in the machine. Conditional transfers, however, change instruction counter contents only if a specific condition exists in the machine: the transfer is conditional on the presence of the specified condition. Besides changing instruction counter contents, a transfer operation may involve loading or modifying an index register, establishing trap and parity controls, or storing the present instruction counter contents.

Unconditional

Unconditional transfers automatically place the instruction word effective address in the instruction counter. If some other action is also specified, it is accomplished before the transfer is effected. For example, assume an index register is to be incremented along with the transfer. Before considering the possible data path to use, consider the instruction word. Bits 21 through 35 form the address field and specify the transfer address. Tag field bits 18 through 20 serve to identify the index register to be incremented. Since the tag field is the only instruction word field that can specify the use of an index register, and since the index register specified is to be incremented, no provision is made for address modification. The instruction word base address is therefore the effective address. With bits 18 through 35 used to identify the index register involved and the transfer address, the remaining bits (S through 17) must be associated with specifying the increment and the operation code. Bits S through 2 serve as the operation code field, and bits 3 through 17 form the decrement field. This field is used as the increment value.

The action unfolds as follows. Initially, the instruction word address (Figure 44), which is in storage register positions 21 through 35, is transferred to adder positions 21 through 35. From here, the address is transferred to the address register. With the transfer address in the address register, the contents of the specified index register are transferred in 1's complement form to the adder when a hot 1 is added to adder position 35. The result is the 2's complement of the contents of the specified index register. This value is returned to the specified index register via XRX. Following this action, the contents of the specified index register are again transferred to the adder. Instruction word bits 3 through 17 are also transferred to the adder; they are shifted right 18 positions during the transfer and enter the adder in positions 21 through 35. A hot 1 is then generated to adder position 35. The result of this action is the addition of the instruction word decrement field and the contents of the specified index register. This result is transferred from the adder to the specified index register via XRX. With the associated action completed, the transfer is effected by transferring address register contents to the instruction counter.

Another operation that can accompany an unconditional transfer is storing of the instruction counter. In this case, the instruction word must specify the operation to be performed and provide a store address as well as a transfer address. Since only one address field is available, some other means

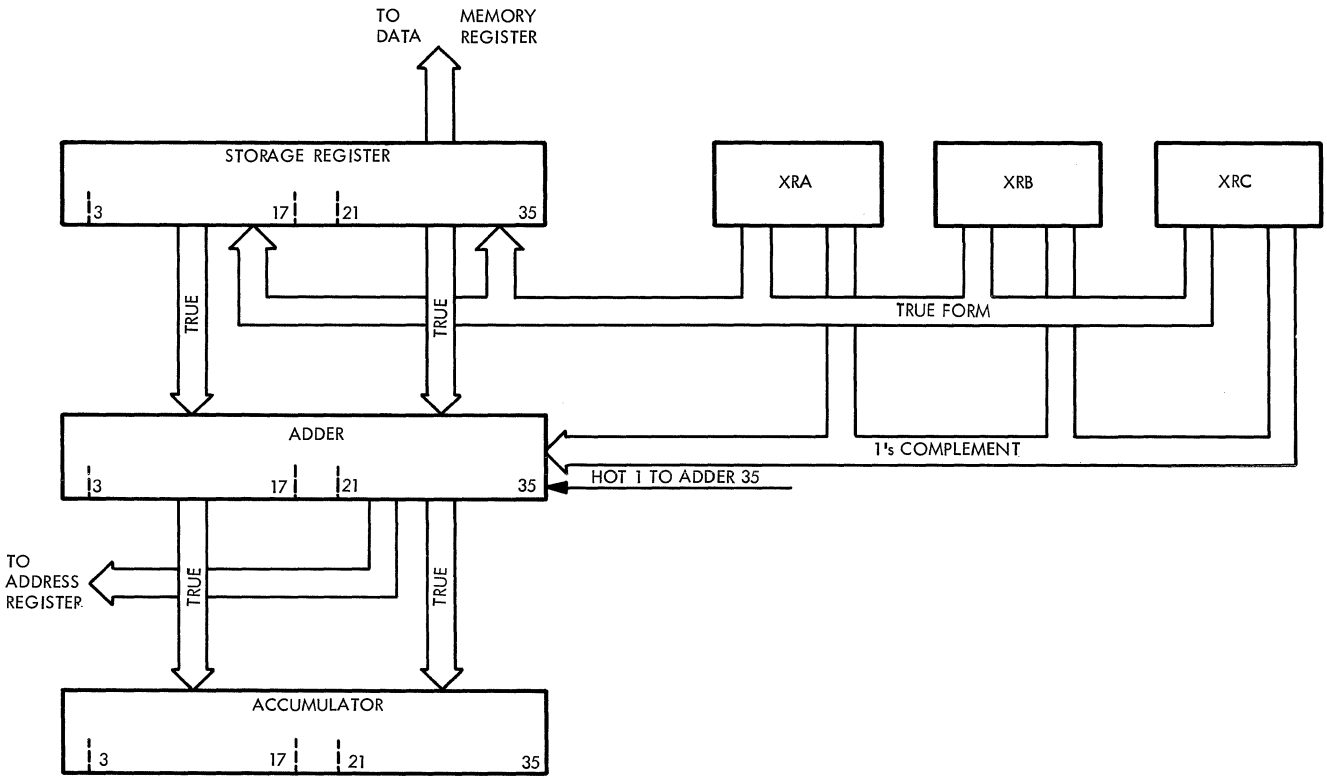


FIGURE 43. ADDRESS MODIFICATION AND INDEX REGISTER STORE OPERATIONS DATA PATHS

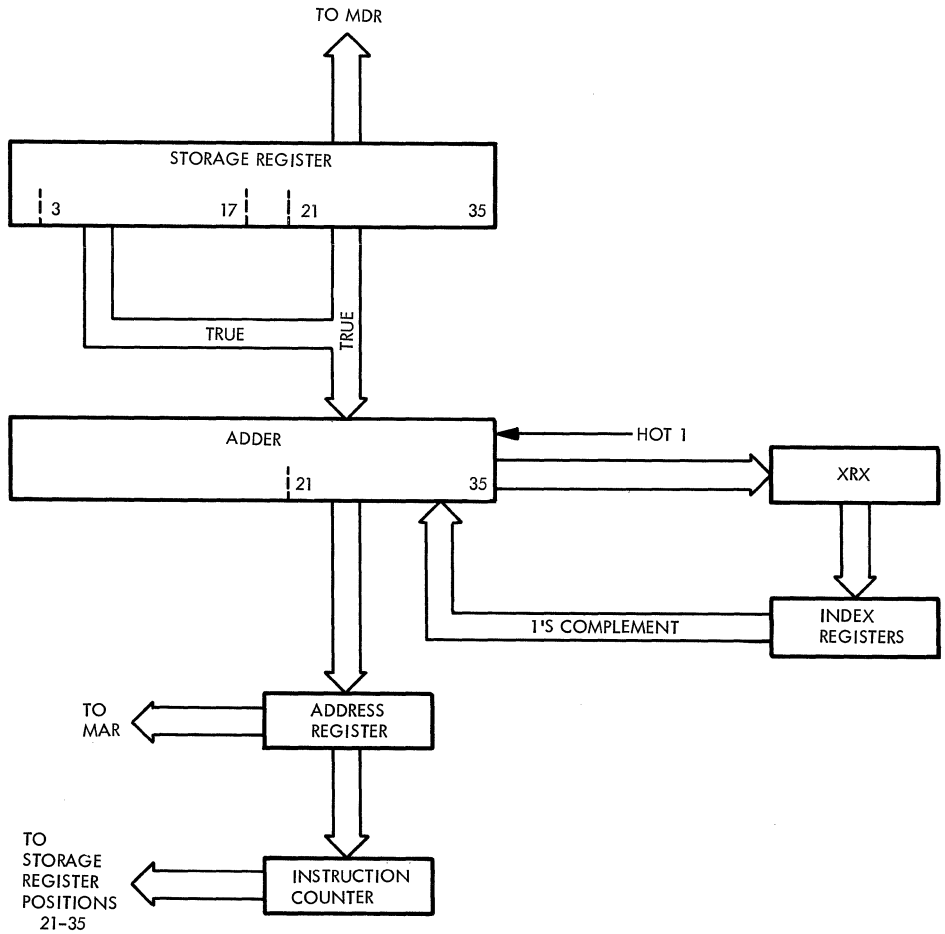


FIGURE 44. TRANSFER OPERATIONS DATA PATHS

must be provided to accommodate the two addresses. This becomes apparent as the data flow is described. Since the action associated with the transfer involves the instruction counter, the instruction word tag field is available to specify an index register for address modification. In addition, indirect addressing is possible with this operation, although the following description does not include it.

Initial action involves the transfer of the instruction word address field (base address) to the adder. If indexing is specified, the operation is performed at this time, and the result in the adder is the effective address. However, if no indexing is specified, the base address becomes the effective address. In either case, adder bits 21-35 are transferred to the address register; this transfer always involves the effective address. Since a storage address is needed, the contents of the address register, the instruction word effective address, are transferred to the memory address register (MAR). With the effective address formed and in the address register and MAR, the instruction word base address is no longer needed. At this time, the instruction counter contents are transferred to storage register positions 21-35. With this transfer accomplished, the address register contents are gated into the instruction counter. At this point, the effective address is in both the instruction counter and MAR, and the original instruction counter contents are in the storage register. The address in the storage register is the address of the transfer instruction plus 1. First, the storage register is transferred to the memory data register, is transferred to the memory data register, satisfying the store action. Then the instruction counter is stepped to specify the location from which the next instruction to be executed will be obtained; thus, the transfer action is completed. Returning to the instruction word, its effective address specifies the store address, and its effective address plus one specifies the transfer address.

When trap and parity controls are to be established along with effecting a transfer of program control, the instruction word need only specify the action desired and the transfer address. The tag and flag fields are therefore available for addressing: both address modification and indirect addressing are possible.

Initially, the base address is transferred from the storage register to the adder, where indexing is performed if specified, following which the effective address is transferred from the adder to the address register. The appropriate controls are then turned on. With this action accomplished, the transfer is completed: address register contents (the effective address) are placed in the instruction counter.

The simplest unconditional transfer is the transfer that has no associated action. In this case, the instruction word can specify both indexing and indirect addressing. In this case, the instruction word can specify both indexing and indirect addressing. Whether neither or both are used, a transfer from the adder to the address register is eventually effected. This transfer involves the effective address. From the address register, the effective address is inserted in the instruction counter, completing the operation.

The action of setting an index register with the 2's complement of the address of the transfer instruction can also accompany an unconditional transfer. In this case, no address modification is possible. The instruction word base address becomes the effective transfer address and is transferred via the adder to the address register. The instruction counter contents are then transferred to the storage register. From the storage register, the address of the transfer instruction goes to the adder, and from the adder to the specified index register via XRX. The contents of the specified index register are then transferred in 1's complement form to the adder, where a hot 1 is added to position 35. The result of this action is the 2's complement, which is then transferred via XRX to the specified index register. Following this action, the address register contents are transferred to the instruction counter, terminating the operation.

Conditional

Conditional transfers are contingent on the presence of a specific condition. Such transfers can be divided into three groups: (1) those that just check an index register to determine whether a transfer condition exists; (2) those that check an index register to determine whether a transfer condition exists and modify the index register checked; (3) those that check something other than an index register to determine whether a transfer condition exists. A description of each follows.

When just checking an index register to determine whether a transfer condition exists, the instruction word decrement field (bits 3-17) is compared with the contents of the specified index register (Figure 44). No address modification is possible in this case, and the transfer condition is specified by the instruction word operation code. A transfer condition can be contingent on the index values being greater than the decrement or equal to or less than the decrement.

Initial action involves transferring the instruction word base address, which becomes the effective address, to the address register via the adder. With the potential transfer address in the address register, the instruction word decrement field is transferred in true form to adder positions 21 through 35. The 1's complement of the specified index register is then transferred to adder positions 21 through 35. A hot 1

is added to adder position 35, thus effecting subtraction. If the index register value is the greater value, no carry is generated out of adder position 21, whereas a carryout of adder position 21 indicates the index register value is either equal to or less than the decrement. Therefore, the existence of the transfer condition is ascertained by checking for a carryout of bit 21 (X carry). When the transfer is contingent on the index register value's being larger, the absence of an X carry causes the address register contents to be transferred into the instruction counter, thereby effecting the transfer. However, when the transfer is contingent on the index register value's being equal to or less than the decrement, the presence of an X carry causes the address register contents to be transferred to the instruction counter, thereby effective the transfer.

For transfers which check an index register to determine whether a transfer condition exists and modify the index register checked, the action is similar. In this case, the instruction word tag field specifies the index register to be checked and modified; no address modification is possible. Consequently, the instruction word base address becomes the effective address. Initial action involves transferring this address through address positions 21 through 35 to the address register. With the potential transfer address in the address register, the presence or absence of the transfer condition is determined by comparing the instruction word decrement field with the contents of the specified index register. If the transfer is contingent on the index register value's being the larger value, the following takes place. The instruction word decrement field is transferred in true form to adder positions 21 through 35. The contents of the specified index register are transferred in 1's complement form to adder position 21 through 35. A hot 1 is added to adder position 35, effecting a subtraction. Following the subtraction, a check is made for an X carry. If no X carry is present, the transfer condition exists. The absence of an X carry, in this case, causes the contents of adder positions 21 through 35 to be transferred into the specified index register. Following this action, the contents of the specified index register are transferred back to the adder in 1's complement form, where a hot 1 is added to adder position 35. The result of this action is the true difference between the decrement field and the index register value. Said another way, the index register value has been reduced by the decrement value. This result is placed in the specified index register. In addition, the absense of an X carry causes address register contents to be transferred into the instruction counter after the specified index register is modified. Should the transfer condition not exist, an X carry is generated, the operation is terminated, and the next sequential instruction is fetched for execution.

When the transfer is contingent on the index register value's being less than or equal to the decrement, the potential transfer address (instruction word base address, which becomes the effective address) is routed from the storage register through the adder to the address register. The instruction word decrement field is then transferred to the adder in true form. The contents of the specified index register are also transferred to the adder, but in 1's complement form. A hot 1 is added to adder position 35, and a subtraction takes place. If an X carry is generated, address register contents are transferred into the instruction counter, effecting the transfer. However, if no X carry is generated, the specified index register value is reduced by the decrement value (loop the result of the subtraction from the adder to the specified index register, back to the adder, add a hot 1, and place this result in the specified index register) and the next sequential instruction is fetched for execution.

The final group of conditional transfer operations checks for a specific condition in the CPU, and if the condition exists a transfer is effected. In one case, a turn-off action is associated with the transfer. A transfer can be contingent on the following conditions: (1) if the accumulator sign is negative, (2) if the accumulator sign is positive, (3) if accumulator Q-35 is not equal to 0, (4) if accumulator Q-35 equals 0, (5) if the accumulator overflow indicator is on. With the last case, the indicator is turned off before the transfer is effected.

In any of the cases, the instruction can specify an index register to modify the base address. The action takes place as follows. The instruction word base address is transferred to adder positions 21 through 35, where it is modified if indexing is specified. Adder positions 21 through 35 are then transferred to the address register; this transfer always involves the effective address, which is the potential transfer address. After the transfer from the adder to the address register, the specified condition is checked for its presence. If present, address register contents are transferred into the instruction counter. If the transfer condition is not present, the next sequential address is referenced for the next instruction to be executed.

STORE OPERATIONS

Store operations in the CPU involve placing the contents of a register or portions thereof in a specified core storage location. Figure 45 shows the data flow paths for CPU store operations. Included are

the paths used in the overlapped data channel store operations.

When the contents of the instruction counter are stored, they contain the address of the store instruction plus 1. A transfer is effected from the instruction counter to storage register bits 21 through 35. The contents of the store address, the instruction word effective address, are transferred from core storage to the storage bus. For simplicity, this bus is shown as a register and its 37th check bit is not shown. The storage bus routes the contents of the store address to the storage register. However, only bits S through 20 are allowed to enter the storage register. Thus, the original contents of the store address are modified. Storage register contents S through 35 are then placed on the storage bus and eventually returned to the store address.

The above type of operation is called a partial store: only part of the contents of the store address is altered. Another example of a partial store is the storing of index register contents, previously described. Further, the accumulator contents can be manipulated with partial store operations. For instance, the operation store address causes accumulator bits 21 through 35 to be transferred to storage register positions 21 through 35. The contents of the store address, except bits 21 through 35, are transferred via the storage bus into the storage register. Following this action, storage register contents are transferred via the storage bus to the store address.

Similar action takes place when the accumulator decrement field is stored. In this case, accumulator bits 3 through 17 are transferred to storage register positions 3 through 17. The contents of the store address, except bits 3 through 17, are transferred via the storage bus to the storage register. Following this action, storage register contents are placed on the storage bus and are eventually transferred into the store address.

A full-word store can be accomplished with the contents of the accumulator or the MQ register. When using accumulator contents, either a machine word or a logical word may be stored. A machine word is 36 bits long with one sign bit and 35 data bits: S through 35. If a store instruction is given, accumulator bits S and 1 through 35 are transferred to the storage register. From the storage register the word goes to the storage bus and eventually into the store address, the instruction word effective address. Identical action takes place when a store logical word instruction is given, except accumulator bit P is transferred to the storage register sign position in place of the accumulator sign.

Execution of a store MQ register instruction causes the contents of the MQ register (bits S through 35) to be transferred to the storage register, and from the storage register to the storage bus. The information eventually enters core storage, where it is placed in the store address.

Another example of a full store is the storing of 0's in the specified store address. This action has the same effect as resetting a register. The storing of 0's is accomplished by blocking the transfer of Storage Register contents to the Storage Bus.

The operation store location and trap causes the location of the store instruction plus 1 to be placed in location 00000. Actually, this operation is a special case of the store instruction counter operation. Only bits S through 11 of the instruction word are used; the remaining instruction word bits are not used. Addressing is therefore implied by the operation code. Initially, the address register is cleared, thereby placing the desired address in this register. From the address register, the all-zero configuration goes to MAR, which effects the readout of the desired location. The instruction counter is incremented, yielding the address of the store instruction plus 1. This value is transferred to storage register positions 21 through 35. Although location 00000 is read out, its contents are destroyed by preventing them from being transferred into the MDR. Instead, storage register contents are transferred to the storage bus and eventually to location 00000. The final configuration of location 00000 is all 0's in bits S through 20 and an address in bits 21 through 35. When the store action is complete, the value 00002_8 is loaded into the instruction counter. This value is the address of the next instruction to be executed.

The CPU can be used as a transfer path for I/O operations. During such operations, information may be configured in 6-bit bytes. Each byte, then, would represent a character. A store accumulator character operation is available. In this operation, however, only accumulator bits 30 through 35 are involved. These bits are transferred to positions in the storage register which are specified by a count in the position register. Here they are joined with the remaining bits of the store address. The new word is then placed on the storage register for transfer into the specified address.

Note that in all CPU store operations the action centers around the storage register. It is in this register that either a full store or a partial store is effected. In contrast to this method is the method used with overlapped data channel stores. During such operations, both the word counter and the address register of the specified data channel are transferred to storage bus positions 3 through 17 and 21 through 35 respectively. Formation of the new contents of the store address is therefore effected on the storage bus in this case as opposed to the storage register during CPU store operations. In

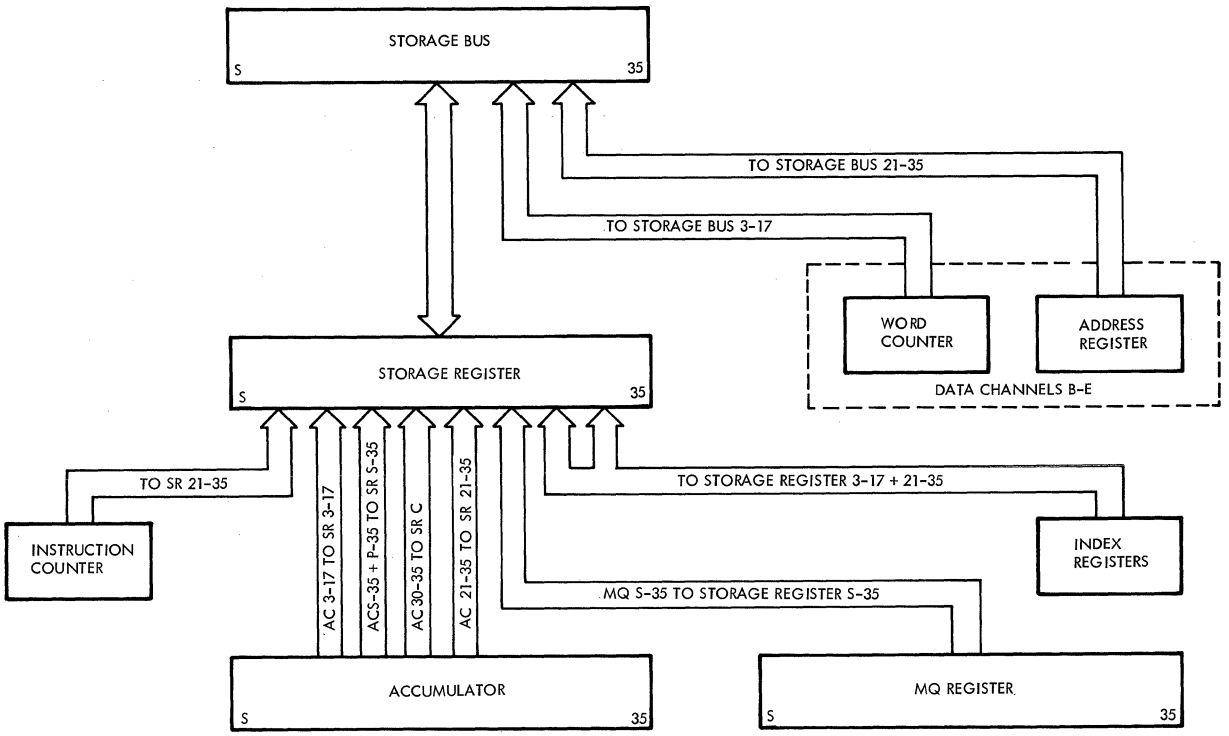


FIGURE 45. STORE OPERATIONS DATA PATHS

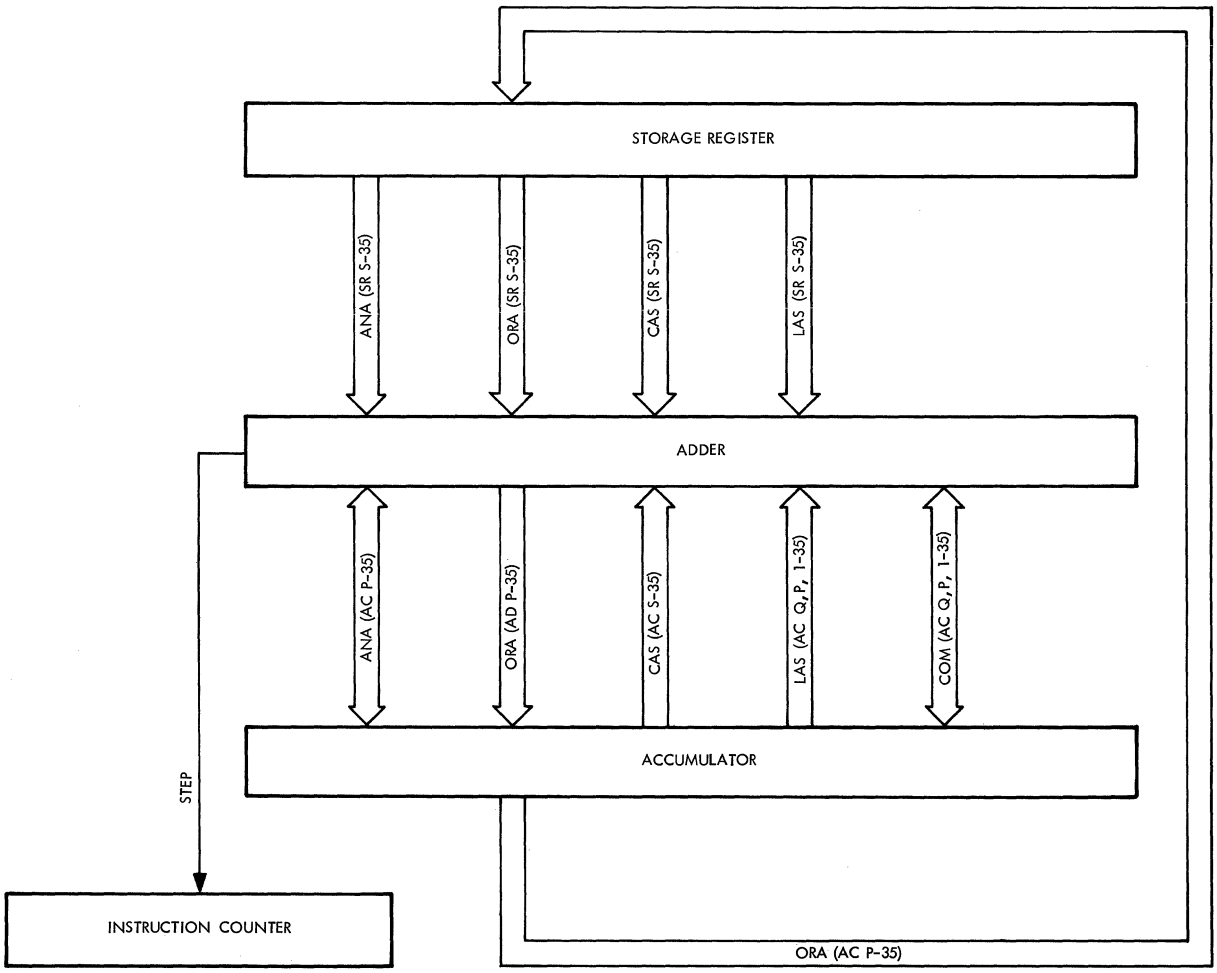


FIGURE 46. LOGICAL OPERATIONS DATA PATHS

either case, the information on the storage bus goes to the store address.

LOGICAL OPERATIONS

Logical operations provide the means for dealing with a 36-bit unsigned word. These operations are especially useful in working with BCD information. Figure 46 shows the data flow associated with each logical operation. Each of these operations is individually described in the following paragraphs.

Consider first the operation AND-to-accumulator. In this operation, accumulator bits P through 35 are matched with storage register bits S through 35, which contain the contents of the instruction word effective address. Matching is accomplished by transferring each operand to the adder, where they are added. The adder contains lookahead circuits which determine whether a carry is going to be generated out of an individual position. If corresponding operand bits are 1's, or if corresponding operand bits are 1 and 0 with a carry from the preceding low-order position, a carry is produced. The adder lookahead circuits can check for either of these conditions. In the AND-to-accumulator operation, only that part of the lookahead circuits which checks for corresponding 1 bits is used. When corresponding bits are 1, the lookahead circuits generate a carry which, in this case, is returned to the corresponding accumulator position. An AND-to-accumulator operation may be defined as the matching of the accumulator operand P through 35 with the storage register operand S through 35 to produce a result operand which contains 1's only in positions which originally contained a 1 in both operands. A 00 match or a 10 match yields a 0 result bit. The result of the accumulator position P and storage register S match is placed in accumulator position P.

Similar to the AND-to-accumulator operation is the OR-to-accumulator operation. In this operation, the accumulator operand P through 35 is matched with the storage register operand S through 35 to produce a result operand which contains a 1 in any position that contained a 1 in either of the original operands: the accumulator and storage register operands are logically added. The contents of the instruction word effective address are transferred into the storage register. Simultaneously, accumulator bits P through 35 are transferred into the storage register on top of the other operand. The result is a logical add, where a 1 results in any position that had a 1 in either of the original operands. Storage register bits S through 35 are then transferred to adder bits P through 35. Adder bits P through 35, in turn, are transferred to accumulator bits P through 35.

A compare-accumulator-with-storage operation is available which enables determination of the algebraic relationship between the operand contained in the effective address and accumulator bits S through 35. Initially, the 1's complement of the accumulator operand is transferred to the adder along with the true form of the storage register operand. A subtraction results. After the subtraction, a check is made for a Q carry. The presence or absence of a Q carry is combined with the sign of each operand to determine whether the instruction counter is to be stepped:

a. If the accumulator is positive, the presence of a Q carry has no meaning with respect to stepping the instruction counter.
b. If the accumulator is negative, the instruction counter is stepped (incremented by 1) if no Q carry is present.
c. If the storage register is positive, the presence of a Q carry causes the instruction counter to be stepped.
d. If the storage register is negative, no stepping takes place, regardless of the presence or absence of a Q carry.

Stepping of the instruction counter at this time indicates that the accumulator operand is smaller than the storage register operand. Following this action, the original storage register operand is again transferred to the adder with the 1's complement of the accumulator operand. A hot 1 is then added to adder bit 35, effecting subtraction. If a Q carry is generated, the accumulator operand is either equal to or smaller than the storage register operand and the instruction counter is stepped. This step determines the equal condition if the first step did not result in instruction counter stepping, and if the first step did result in instruction counter stepping, this step distinguishes from the equal result. Summarizing, (1) if the accumulator operand is greater than the storage register operand, no instruction counter stepping takes place, (2) if the accumulator operand is equal to the storage register operand, the instruction counter is stepped once, and (3) if the accumulator operand is smaller than the storage register operand, the instruction counter is stepped twice.

A logical compare accumulator with storage operating enables the logical comparison of a 37-bit accumulator operand (Q, P, and 1-35) with a 36-bit storage register operand (S-35). The storage operand is obtained from the instruction word effective address. Initially, the storage register operand is transferred in true form to the adder with the 1's complement of the accumulator operand; a subtraction results. A check is then made to determine whether a Q carry is present. If a Q carry is present, the accumulator operand is the smaller operand and the instruction counter is stepped. If no Q carry is present, the accumulator is either equal to or

greater than the storage register operand and no stepping takes place. Next, the action is repeated using the 2's complement of the accumulator operand. If a Q carry is generated this time, the instruction counter is stepped. The meaning of the Q carry depends on whether a Q carry was generated on the first pass: if a Q carry was not generated on the first pass, operation of a Q carry on the second pass indicates equal operands, whereas generation of a Q carry on each pass indicates the accumulator is the smaller operand. In summary, (1) if the accumulator operand is the larger operand, no instruction counter stepping takes place, (2) if both operands are equal, the instruction counter is stepped once, and (3) if the accumulator is the smaller operand, the instruction counter is stepped twice.

A complement magnitude operation is available which allows accumulator bits Q, P, and 1 through 35 to be complemented: 0's are made 1's, and 1's are made 0's. In this operation, the 1's complements of accumulator bits Q, P, and 1 through 35 are transferred to corresponding positions in the adder. Adder positions Q, P, and 1 through 35 are then transferred to corresponding accumulator positions. The accumulator sign position remains unaltered.

CHARACTER HANDLING OPERATIONS

Three character-handling operations are available which expedite handling of 6-bit character operations. In each case, instruction word bits 15, 16, and 17 form a character position (c) field which specifies the character in the instruction word effective address involved in the operation. Valid bit configurations for the C field range from 000 to 101:

000 - C0 (bits S-5)
001 - C1 (bits 6-11)
010 - C2 (bits 12-17)
011 - C3 (bits 18-23)
100 - C4 (bits 24-29)
101 - C5 (bits 30-35)

In the place-character-from-storage operation, the instruction word effective address character specified by the instruction word C field is placed in accumulator positions 30 through 35. Initially, instruction word bits 13 through 17 are placed in the position register (Figure 47). An effective address is formed, and the specified location in core storage is accessed. Its contents are routed to the CPU storage register. Position register bits 15 through 17 are decoded to produce a character selection signal. This signal serves to gate only the desired character from its storage bus positions to storage register positions 30-35. The remaining storage register bits are cleared. Consequently, the word in the storage register always has the selected character in positions 30-35. The complete storage register word is then transferred to corresponding adder positions. Accumulator bits Q, P 1-29 are also fed to the adder. The adder sum (accumulator Q, P, 1-29 and SR 30-35) is returned to the AC.

The store-accumulator-character operation is similar. In this case, the character selection signal resulting from position register decoding gates accumulator bits 30 through 35 into the storage register positions corresponding to the selected character. The contents of the effective address are transferred into corresponding storage register positions, except for the selected character positions. Storage register contents are then transferred to the effective address.

The compare-character-with-storage operation compares the character formed by accumulator bits 30 through 35 with a specified character in the effective address. Position register bits 15 through 17 are decoded to select the data character from the storage bus for placement in storage register positions 30 through 35. The entire storage register, including sign, is then gated to the adder along with the 1's complement of the accumulator. A check is made for an adder 30 carry, and, if one is present, the accumulator character is the smaller character and the instruction counter is stepped. If a 30 carry is not present, the accumulator character is the larger character and no stepping takes place. The action is repeated, using the true form of the storage register character and the 2's complement of the accumulator character. Again a check is made for a 30 carry: if it is present, the instruction counter is stepped. Generation of a 30 carry on the second pass means (1) the accumulator character is the smaller character if a 30 carry was generated on the first pass or (2) both characters are equal if no carry was generated on the first pass. In summary, if the accumulator character is larger, no instruction counter stepping takes place, whereas the instruction counter is stepped once if both characters are equal and twice if the accumulator character is the smaller character.

SHIFTING OPERATIONS

Four shifting operations are available in the 7040-7044. In each operation, instruction word bits 28 through 35 form a shift count which specifies the number of places to shift. No shifting operation references core storage, but the instruction word

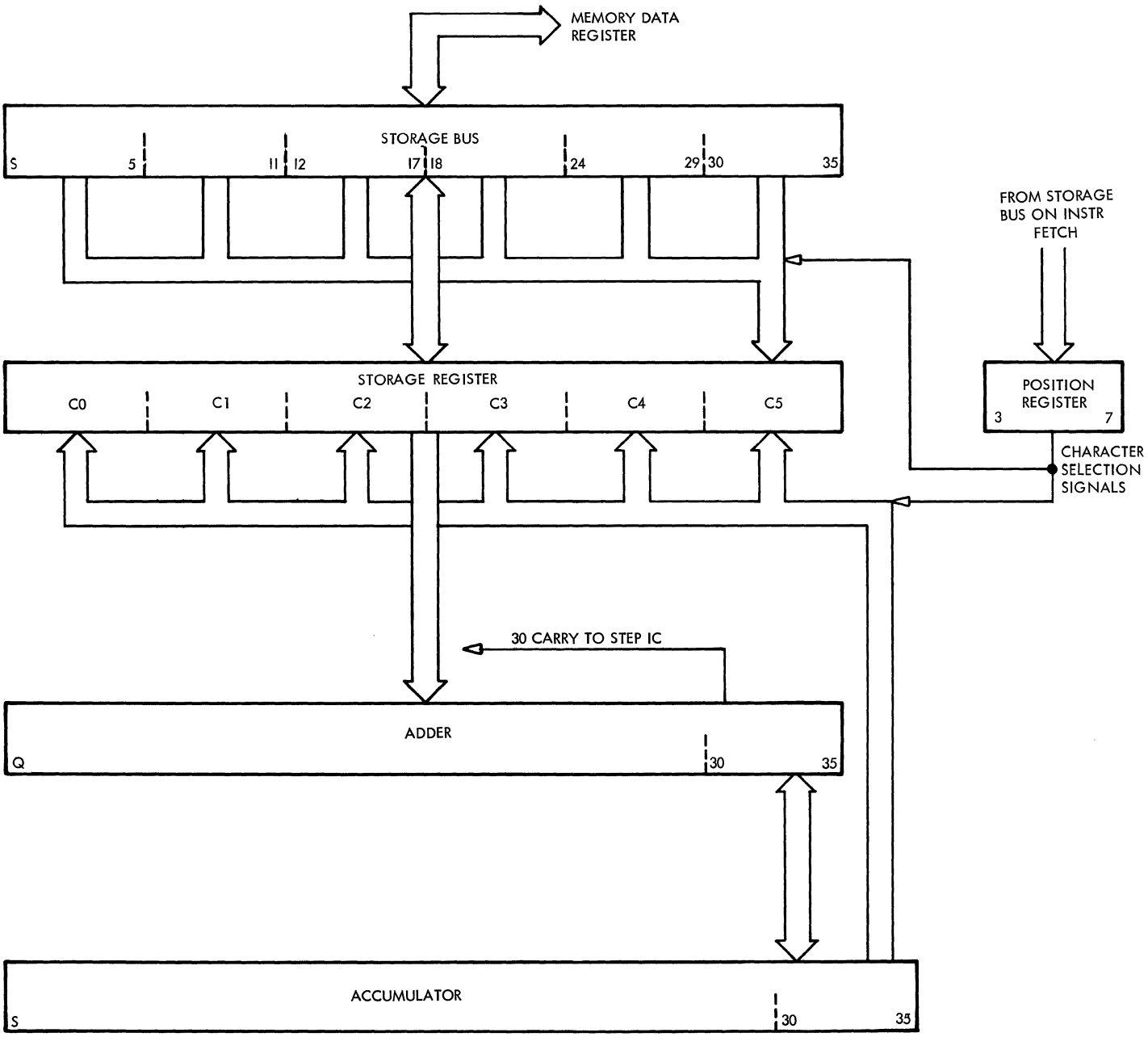


FIGURE 47. CHARACTER-HANDLING OPERATIONS DATA PATHS

address field is indexable. The direction of the shift is specified by the operation code.

In the logical left shift operation, instruction word bits 21 through 35 are transferred from the storage register to the adder (Figure 48). Here indexing is performed if a configuration other than 0 is in the instruction word tag field. The result is the effective address. If no indexing is specified, the instruction word base address is the effective address. In either case, effective address bits 28 through 35 are loaded into the shift counter. The term effective address in this case is misleading. Perhaps the term effective count would be better. However, the operation is identical with address modification, except the result is used differently. After the shift counter is loaded, a one position left shift is performed if the shift counter does not contain 0 using accumulator bits Q through 35 and MQ register bits S through 35. The bit shifted left out of MQ register bit S enters accumulator bit 35. Bits shifted out of accumulator position Q are lost. In performing the shift, MQ register bit position 35 is vacated, but this vacated position is made a 0. After a one position shift is completed, the shift counter is decreased by one. It is then inspected and if it is other than 0, another one position left shift is performed. This action of shifting one position and then checking the shift counter continues until the shift counter is reduced to 0.

Note that the accumulator sign is not affected by the shifting: it remains unaltered throughout the operation. It is possible to cause overflow on a logical left shift. An overflow condition results when a 1 bit is shifted out of accumulator bit position 1. In this case, the overflow indicator is turned on.

A logical right shift is identical, except the shift direction is reversed and overflow cannot occur. In a logical right shift operation, accumulator bit 35 enters MQ register bit S. Bits shifted out of MQ register bit position 35 are lost, and the vacated accumulator bit position Q is made 0.

Similar to the logical left and right shifts are the long left and right shifts. In these cases, the instruction word is handled identically, as is the one-position-at-a-time shift. The only differences are the exclusion of MQ register bit position S in the shifting and handling of the accumulator sign position. During a long left shift, MQ register bit 1 enters accumulator bit position 35. Bits shifted out of accumulator position Q are lost, and vacated MQ register position 35 is made 0. After the shifting operation is completed, the MQ register sign is transferred to the accumulator sign position. Again, overflow is possible.

During a long right shift, accumulator bit 35 enters MQ register bit 1. Bits shifted out of MQ register bit position 35 are lost, and vacated accumulator position Q is made 0. At conclusion of shift operations AC (S) is transferred to MQ (S).

Shifting operations are useful in dealing with BCD information: arranging characters, sorting, etc. These operations also provide a means of getting the result of a division operation into the accumulator or of getting the accumulator into the MQ register for use as a multiplier.

ROTATE OPERATION

A rotate operation (Figure 49) is similar to a shift operation. The differences are (1) no bits are lost and (2) only the MQ register is involved in rotation operations. Instruction word handling is identical, and the MQ bits are rotated left one position at a time.

First an effective address is formed in the adder. Bits 28 through 35 of the effective address are loaded directly into the shift counter. If the shift counter does not contain 0, the MQ register is rotated one position left with position S entering position 35. The shift counter is decremented by 1 and then checked for a 0 value. If it is 0, the operation is ended. If it is not 0, the MQ register contents are again rotated left one position. This pattern continues until the shift counter is reduced to 0.

Rotate operations are useful in positioning information for transfer into the accumulator or storage.

SIGN ALTERATION AND TEST OPERATIONS

Sign alteration operations in the 7040-7044 center around the accumulator sign bit. The two operations, change sign and set sign plus, do not involve any data flow. With each address modification can be employed; however, such modification could change the operation to be performed because the address field of each associated instruction is used to hold part of the operation code. The change sign operation makes the accumulator sign negative if it is positive and positive if it is negative. The set sign plus operation makes the accumulator sign positive regardless of its present condition.

Test operations involve checking a specific bit of a particular register, the condition of a specific indicator, and the position of a specific switch on the operator's console. Five test operations are available: low-order bit test; P bit test; divide check test; sense switch test; input-output (I/O) check test. The low-order bit test effects a check of the current status of accumulator bit 35. If this bit is a 1, the shift counter is stepped. This stepping is in addition to the normal stepping. Consequently, with a low-order bit test and a 1 in accumulator bit 35, the next sequential instruction is skipped. If bit 35 is a 0, no alteration of the programmed instruction sequence takes place.

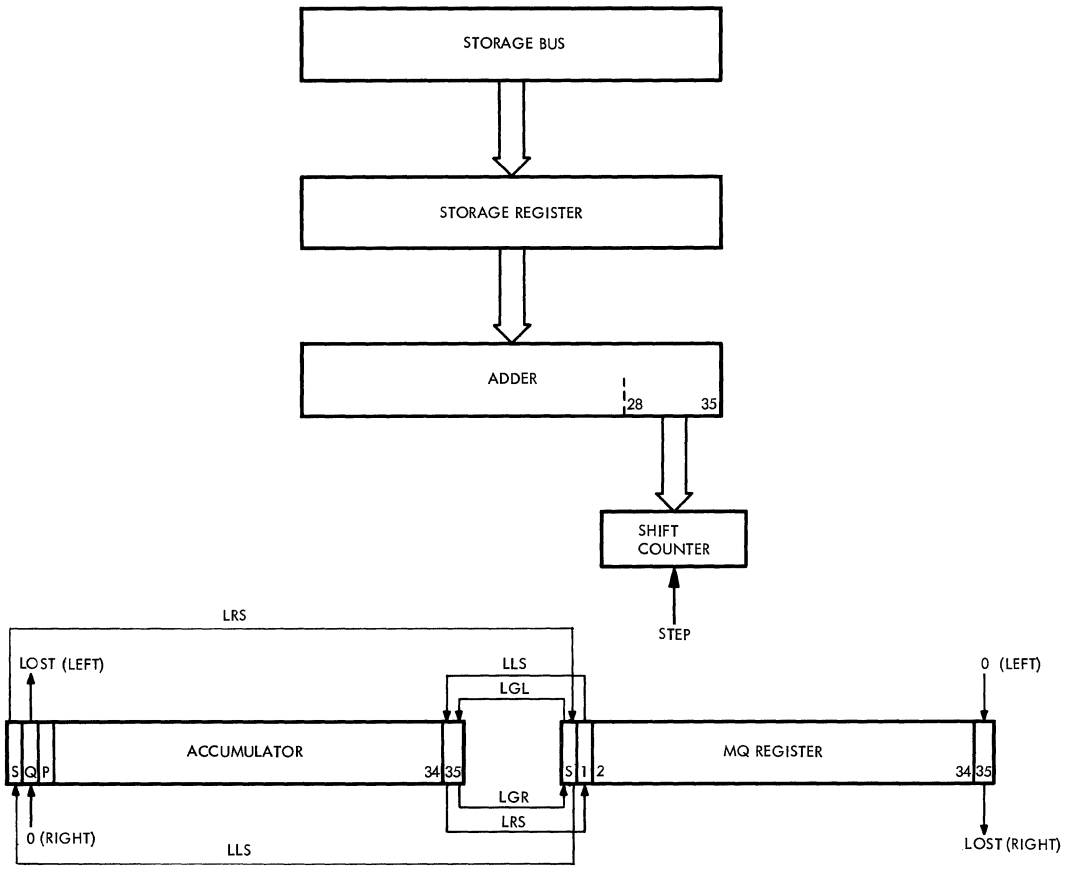


FIGURE 48. SHIFTING OPERATIONS

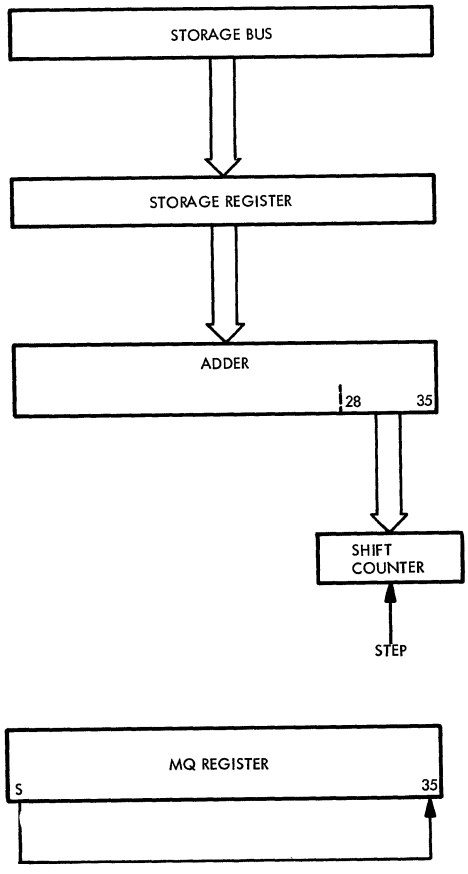


FIGURE 49. ROTATE OPERATION

The P bit test is identical with the low-order bit test, except the accumulator P bit is checked for the stepping condition.

During a divide check test, the divide check indicator is inspected. If this indicator is on, it is turned off and the next sequential instruction is fetched: no additional stepping of the instruction counter is effected. However, if the indicator is off, an additional stepping of the instruction counter is effected. Consequently, the next sequential instruction is skipped. Identical action occurs with the I/O check test operation, except the I/O check indicator is the reference.

Sense switch test operations provide a means of testing the status of each of the sense switches on the operator's console. Only one switch can be tested at a time, and the operation code specifies the desired switch. If the specified switch is in the depressed (on) position, the instruction counter is stepped in addition to its normal stepping. This action causes the next sequential instruction to be skipped. If the specified switch is in the released (off) position, no additional instruction counter stepping takes place. The next sequential instruction is therefore fetched for execution.

Each instruction used for test operations employs part of the address field to complete the operation code, and each instruction also has a provision for specifying address modification. Such modifications can result in changing the operation code.

SPECIAL STORAGE SIGN HANDLING OPERATIONS

Four special storage sign handling operations are available to set and test the sign position of a core storage word: make storage sign minus; make storage sign plus; storage minus test; storage plus test. In a make-storage-sign-minus operation, the contents of the instruction word effective address are transferred via the storage bus to the storage register. The storage register sign is made a 1. Storage register contents are then transferred to core storage via the storage bus, and eventually the original word with the sign bit made minus is returned to its original address. Identical action takes place in the make-storage-sign-plus operation except that, when the contents of the specified core storage address are in the storage register, the sign bit is made 0.

In a storage-minus-test operation, the contents of the effective address are transferred via the storage bus to the storage register. Here the sign position is tested. If it is minus (a 1), the instruction counter is stepped in addition to its normal stepping. This action causes the next sequential instruction to be skipped. If the sign position is positive, no additional instruction counter stepping takes place; therefore, the next sequential instruction is fetched for execution.

The storage plus test is identical, except the additional instruction counter stepping occurs if the sign position of the accessed storage location is a 0.

The most outstanding quality of these operations is that all the work is accomplished in the storage register. Accumulator contents are not bothered by special storage sign handling operations and therefore do not have to be protected in another register or address.

EXECUTE OPERATION

The execute operation causes the contents of the instruction word effective address to be interpreted as an instruction which is then executed. Initially, the execute instruction is transferred from core storage to the CPU (Figure 50). The entire instruction enters the storage register, and instruction word bits S and 3 through 11 are also transferred to the program register. Storage register bits 21 through 35 are transferred to the adder, where they are indexed if indexing is specified by the instruction word tag field. In any case, a transfer is effected from adder bits 21 through 35 to the address register. This transfer always involves the effective address. Address register contents are then transferred to MAR as an instruction fetch.

Meanwhile, program register contents are decoded. This action reveals the presence of the execute instruction and results in blocking the instruction counter stepping signal.

When the contents of the reference core storage location are transferred to the CPU, they go to the storage register, and bits S and 3 through 11 go to the program register. The instruction is performed in a normal fashion. However, decoding the new program register contents no longer indicates the presence of an execute instruction. Consequently, the instruction counter is stepped in the normal fashion.

An execute operation allows execution of an additional instruction between two sequential instructions. Although the additional instruction may be any of those available and may be used as desired by the programmer, the additional instruction normally will not be a transfer instruction. If the additional instruction is a transfer instruction, the original instruction sequence may be altered.

TRANSMIT OPERATION

A transmit operation causes the contents of a specified core storage location or the contents of a block of specified core storage locations to be transferred to another core storage location or to another block of core storage locations. Before a transmit operation is performed, the accumulator must be loaded because accumulator bits 3 through 17 serve to specify the "from" address and accumulator bits 21 through 35

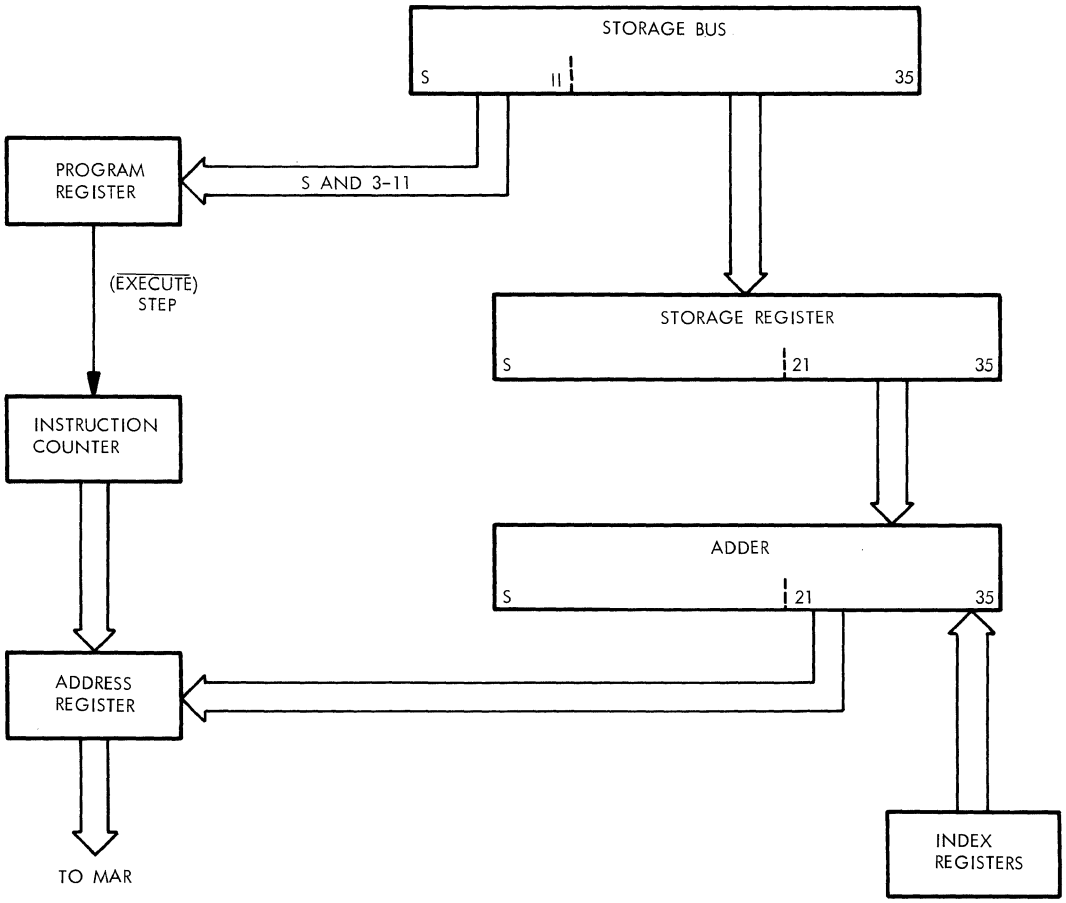


FIGURE 50. EXECUTE OPERATION DATA PATHS

serve to specify the "to" address. Since the accumulator is used in this fashion, it must be updated after each word relocation. To avoid undesired results when updating the accumulator, its bits 18 through 20 should originally contain 0's.

Figure 51 shows the paths used during the execution of a transmit operation. The following is a step-by-step account of the action:

1. Receive the instruction from core storage in the storage register.
2. Transfer storage register bits 21 through 35 to adder positions 21 through 35.
3. Determine the effective address.
4. Load effective address bits 28 through 35 into the shift counter. (Because of the physical size of the shift counter, the maximum count possible is 377_8).
5. Check the shift counter for a value of 0:
   a. If it is 0, end the operation.
   b. If it is not 0, continue.
6. Transfer accumulator bits 3 through 17 to storage register positions 3 through 17.
7. Transfer storage register positions 3 through 17 to adder positions 21 through 35.
8. Transfer adder positions 21-35 to the address register.
9. Transfer address register contents (from address) to MAR to get the desired word.
10. Receive the word from memory in the storage register.
11. Transfer accumulator bits 21 through 35 to adder positions 21 through 35.
12. Transfer adder positions 21 through 35 to the address register.
13. Transfer address register contents to MAR as a store (to) address.
14. Transfer storage register contents to the storage bus and eventually into the "to" address.
15. Decrement (step) the shift counter.
16. Transfer accumulator bits Q through 35 to the adder.
17. Add 1 to adder positions 35 and 17.
18. Transfer the result to the accumulator.
19. Return to step 5.

At the completion of a transmit instruction, accumulator bits 3 through 17 specify the address of the last word transferred plus 1, and accumulator bits 21 through 35 contain the address of the last store location plus 1. Successive transmit operations can be performed if it is desired to transfer or relocate more than 377_8 words.

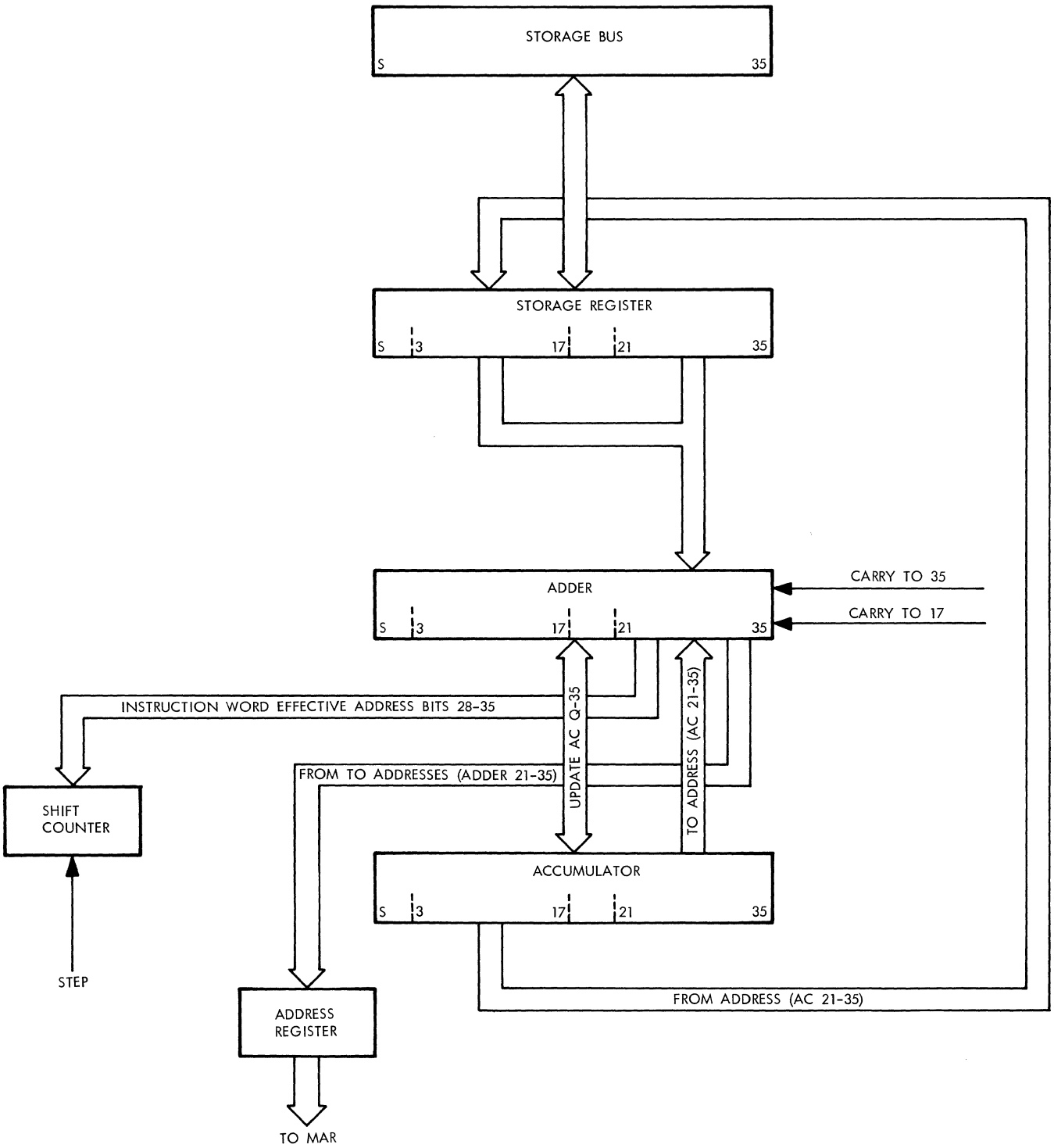


FIGURE 51. TRANSMIT OPERATION DATA PATHS

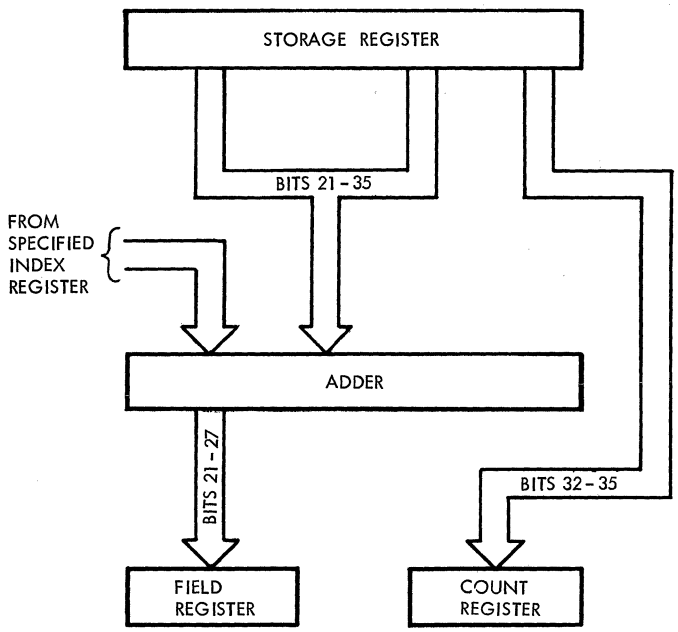


FIGURE 52. MEMORY PROTECT SETUP

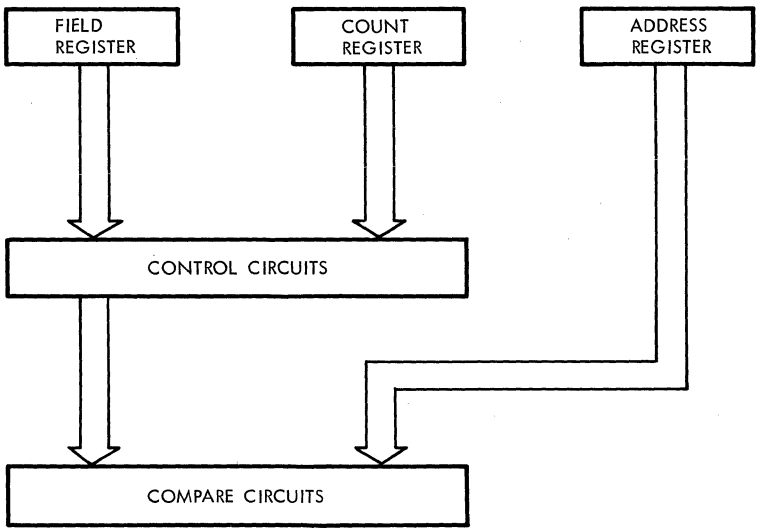


FIGURE 53. MEMORY PROTECT APPLICATION

Memory protection is a feature whereby a specified block of core storage locations is protected from alteration through storing. Assume that a master program is stored in memory locations 20000 through 37777, and that the programmer does not want an accidental intrusion (store) to destroy the program; by utilizing memory protection, locations 20000 through 37777 cannot be disturbed during store operations, thus protecting the contents of that block of addresses. Locations 20000 through 37777 are used as an example; other blocks (as desired) can be protected, but only one block can be protected at a time.

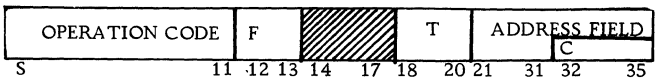
To protect a block of addresses, a Set Protect Mode (SPM) instruction must be executed. The address field of the SPM instruction designates which block of memory locations is to be protected; it does this by setting up two registers, the count register and field register, which are used to monitor the contents of the CPU address register (Figures 52 and 53). The contents of these two registers are compared with the high-order bits of an effective address (in the AR). The count register determines the number of high-order bits to be examined and the field register determines the pattern of bits to be compared against. Violations, attempts by the CPU to store data in a protected area, cause trapping by either an equal or an unequal compare result, according to the selected mode.

The compare-equal mode operates in the following manner. The high-order bits in the field register indicate the block of addresses that are protected. Any time that the corresponding high-order bits in the address register are the same as those in the field register, CPU circuits interpret the identical comparison as an attempt to enter a protected area. The compare-unequal mode operates altogether differently. The high-order bits in the field register indicate the block of addresses that are not protected. Any time that the corresponding high-order bits in the address register are not the same as those in the field register, CPU circuits interpret the nonidentical compare as an attempt to enter a protected area.

The field register is a 7-position register whose bits are labeled 21 through 27; the count register is a 4-position register whose bits are labeled 32 through 35. Figure 54 is a table that shows which field register and address register bits are compared for any given count register setting (C field count).

Set Protect Mode (SPM) Instruction

The SPM instruction word format is as follows:



The operation code is formed by bit positions S-11 and is -1160. Indirect addressing can be employed with the SPM instruction as indicated by the presence of a flag field in bit positions 12 and 13. Bit positions 14-17 are not used by the SPM instruction. The tag field is formed by bit positions 18-20, and the address field is formed by bit positions 21-35.

FIGURE 54. BIT COMPARISON TABLE

C Field (Octal)	Count Register and Address Register Bits Compared in Each Storage Size				
	32K	16K	8K	4K	
00	None	None	None	None	Trap if unequal compare result
01	21	None	None	None	
02	21-22	22	None	None	
03	21-23	22-23	23	None	
04	21-24	22-24	23-24	24	
05	21-25	22-25	23-25	24-25	
06	21-26	22-26	23-26	24-26	
07	21-27	22-27	23-27	24-27	
10	None	None	None	None	Trap if equal compare result
11	21	None	None	None	
12	21-22	22	None	None	
13	21-23	22-23	23	None	
14	21-24	22-24	23-24	24	
15	21-25	22-25	23-25	24-25	
16	21-26	22-26	23-26	24-26	
17	21-27	22-27	23-27	24-27	

Note: Bit 23 indicates addresses above 4K; bit 22 indicates addresses above 8K; bit 21 indicates addresses above 16K.

Decoding an SPM instruction causes the field register to be loaded with bits 21-27 of the effective address and the count register to be loaded with instruction word bits 32-35. The field register is therefore loaded from the adder, and the count register, from the storage register. Instruction word bit 32 controls the mode of protection, whereas instruction word bits 33-35 specify the number of high-order address bits to be compared against corresponding field register bits on subsequent memory references in store operations.

Note in Figure 52 that the count register receives instruction word bits (32-35) and that the field register receives effective address bits. In determining the effective address, the entire

instruction word address field or base address is used. Further, an SPM instruction does not reference memory unless indirect addressing is specified by the instruction word flag field.

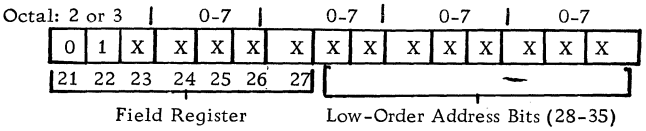
When an SPM instruction is decoded with the machine already in the protect mode, a trap occurs and the machine is removed from the memory protect mode.

Memory protection is an optional feature and may not be available on all machines. If an attempt is made to execute an SPM instruction in a machine that does not have the memory-protect feature, a no-operation results and the next sequential instruction is fetched.

The memory protect mode is also removed by program control. A Release Protect Mode (RPM) is provided for this purpose.

Memory-Protect Examples

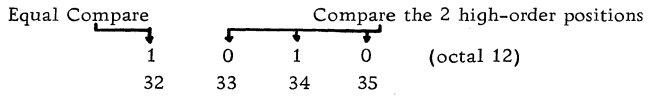
Assume it is desirable to protect memory locations 20000 through 37777 from being referenced by store class instructions. Initially, then, the field register must be set as shown:



The X in positions 23-27 indicates any value: these bits can contain either 1 or 0. Why?

In this case, only field register bits 21 and 22 are important because they are the only two bits that remain exactly the same for all addresses within the block to be protected; other blocks could require up to all seven positions of the register in order to be properly identified.

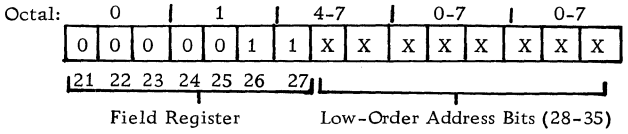
Next set the count register as shown:



Why? This count designates that only field register bits 21 and 22 are to be compared with address register bits 21 and 22, and that if the AR contains any address beginning with 01 there is an equal compare. (See Figure 54).

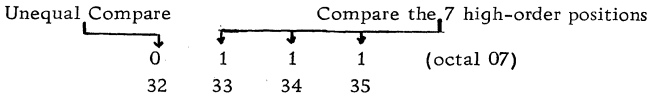
If the AR does have an address beginning with 01 (an equal compare), the store-cycle latch is prevented from being set, thus preventing a store in the core location, and the CPU program goes into a memory-protect trap routine.

To illustrate the unequal-compare mode operation, protect all memory locations except 01400 through 01777 from being referenced by store class instructions. Initially, the field register must be set as shown:



Why? In this case, all seven high-order bits of the specified addresses are exactly the same, so the field register must be set to reflect this.

Next, set the count register as shown:



Why? This count designates that all field register bits must be compared with address register bits 21 through 27, and that if the AR contains any address beginning with any digits other than 0000011 there is an unequal compare. (See Figure 54).

If the AR has an address that does not begin with 0000011 (unequal compare), the store-cycle latch cannot be set, and the program transfers to a memory-protect trap routine.

In the first example, memory locations 20000 through 37777 are protected. In the second example, locations 00000 through 01377 and 02000 through 77777 are protected.

Memory-Protection Control Setup

Figure 55 shows the memory-protection controls. The count register is loaded with bits 32 through 35 of the SPM instruction directly from the storage register, which means that address modification cannot be performed on data going into the count register. Count register position 32 determines whether the CPU will trap on an equal or on an unequal compare; the other count register positions determine which address register and field register bits will be compared (Figure 54).

The field register is loaded with bits 21 through 27 from the adder, which means address modification can be performed on data going to the field register. The outputs of the field register, along with outputs from the count register, are AND'ed with address register bits 21 through 27 in a comparison network to determine whether the address register and field register have equal or unequal addresses.

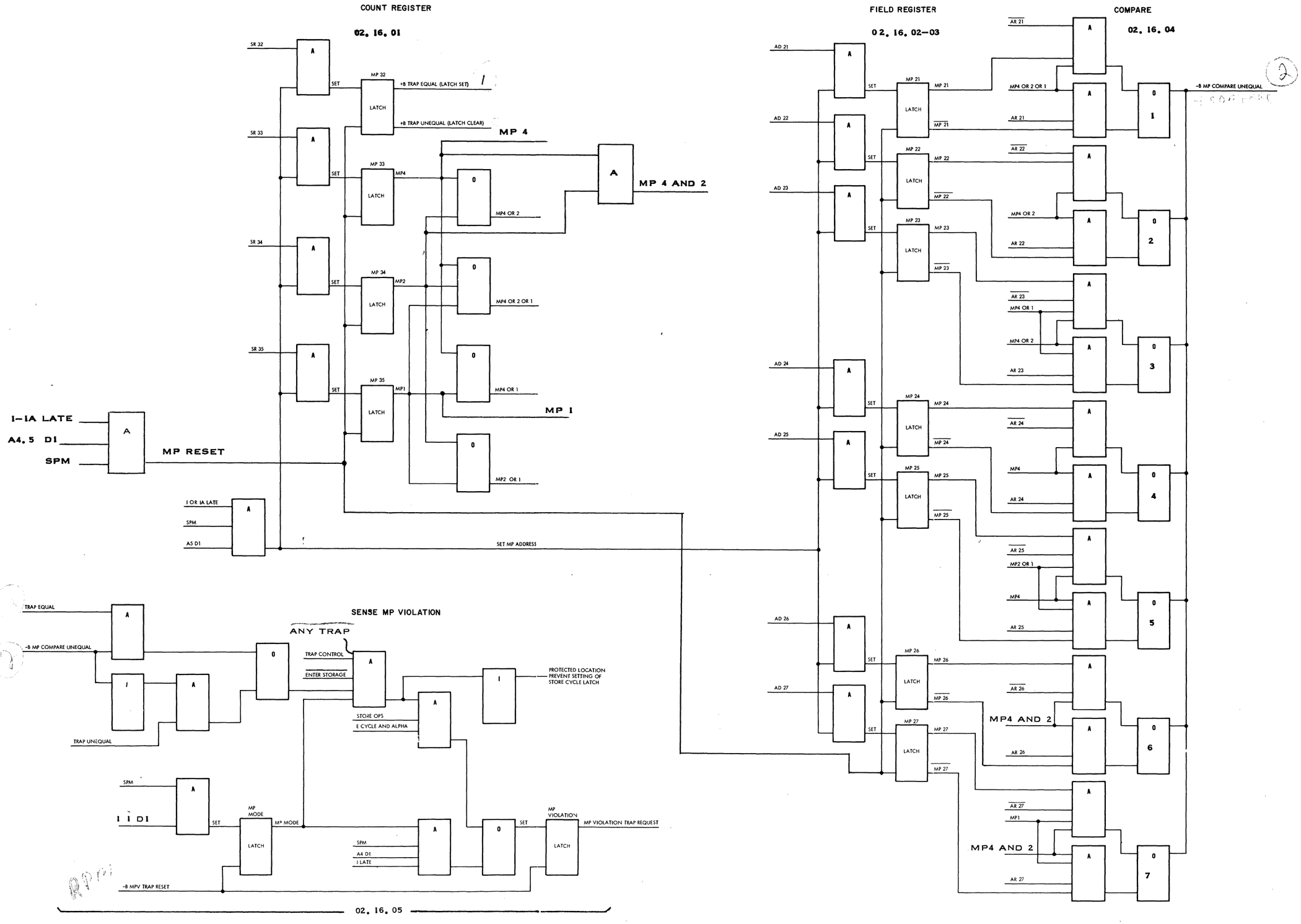


FIGURE 55. MEMORY PROTECT CIRCUIT

To illustrate the compare operation in terms of CPU signals:

1. Given:
   field register 21 = 0 ($\overline{\text{MP 21}}$)
   address register 21 = 1 (AR 21)
   Count registers 33, 34, and 35 are all 1's (MP 4 or 2 or 1); this is necessary because bit 21 is always compared (Figure 54).
2. What happens?
   The AND that is conditioned by $\overline{\text{MP 21}}$, AR 21, and MP 4 or 2 or 1 has a +B output; the +B, through the OR, becomes the -B MP compare-unequal. If AR 21 and MP 21 were the same (either 0's or 1's), neither AND would have a +B output; thus the OR would have a +B output, which is a compare-equal.

The -B MP compare-unequal level is then AND'ed with bit 32 of the count register to sense whether there has been a memory-protect violation. If the address register and field register do not compare, CR bit 32 must be a 0 (trap unequal) before an MP violation is sensed.

If a violation is sensed, the resulting level is AND'ed with several other levels to determine whether to request an MP trap. The two most significant levels involved in determining whether to request a trap are: MP mode and store ops. The MP-mode latch can only be set by an SPM instruction. The store-ops level is necessary because protection is needed only on a store operation; the CPU can read instruction and operand fetches from protected locations.

When all conditions for a valid MP violation are met, the MP violation latch is set; the latch output effects an MP trap. Note that the store cycle latch is prevented from being set as soon as a violation is sensed.

Memory protection is valid only when the CPU tries to store into protected locations; any I-O channel can store into any memory location. Memory protection does not work for channels B through E because they enter storage directly through the MAR without going through the CPU address register. Channel A could have memory protection because it utilizes the CPU address register, but, to make channel A compatible with the other I-O channels, memory protection is suppressed for channel A.

SECTION 6 - TRAPPING

GENERAL

Trapping is a method of signaling unusual or special system conditions to a program without requiring special test instructions. With trapping, system status is constantly monitored, and, when a special condition is detected, the normal program sequence is interrupted and a transfer is made to a trap routine.

An unusual or a special system condition can be either a normal condition or an error. For example, reading the end-of-file mark on a tape is normal, yet it is classified as a special condition. Such classification simplifies the program handling of the condition. On the other hand, the presence of floating-point spill is also considered an unusual condition. This classification appears more reasonable because it constitutes an error. However, each poses a question to the program in progress, and incorporating the trapping scheme simplifies solution of the problem.

Some types of traps are automatic; that is, under any machine conditions, their appearance causes a branch from the normal program sequence to a trap routine. Other types of traps, however, can occur only if the machine is in the trap mode. Still other types of traps can occur only if the machine is in the trap mode and the particular trap is validated. For example, floating-point spill automatically causes a trap, whereas the machine must be in the trap mode to act on an interval timer overflow. The machine must both be in the trap mode and have the parity-error circuits activated before action can be taken on a parity error. Similarly, a channel trap must be validated before the machine recognizes it. A channel trap is validated through the use of a mask register. The mask register is composed of latches, each allowing a different type channel trap when on. If a mask bit is a 1 (latch on) the corresponding trap condition is validated and will be recognized by the machine. If a mask bit is a 0 (latch off) the corresponding trap condition is considered invalid for trapping and will never be recognized by the machine. This method of validation provides great flexibility in programming the 7040-7044.

In the trap scheme, each trap category is assigned two core storage locations. One location serves as a trap record location in which the conditions causing the trap and the instruction counter value at the time of trap recognition are stored. In this manner, a particular trap can be identified and, after execution of a trap routine, a return to the point of program departure can be effected. In most instances the second reserved location sequentially follows the trap record location and is known as the instruction location. Following the recording of trap particulars in the record location, the instruction location value is forced into the instruction counter; therefore, the first instruction executed in a trap operation comes from the instruction location.

The trapping scheme also provides for privileged instructions. Basically, a trap is recognized after the execution of the instruction causing the trap. With a privileged instruction, however, a trap cannot be recognized until after the execution of the instruction following the privileged instruction. The privileged instructions are: RDS, PRD, SEN, WRS, WBT, PWR, CTR, ENB, RCT, ICT, and SPM. The XEC is also a privileged instruction in that certain traps cannot occur between the XEC and the execution of its specified instruction.

TRAP CONTROL

For some types of traps, trap control is automatic; that is, regardless of the condition or state of the trapping-control triggers, the appearance of a certain condition causes a trap. For other than these special cases, trap control must be established: the machine must be placed in the trap mode. The trap mode is entered by turning on the trap-control trigger. Signals from this trigger serve as conditioning levels for many of the trap indication circuits. Three other triggers are also associated with trap control: parity-mode, memory-protect-mode, and channel-trap-control. Although these triggers work in conjunction with the trap-control trigger, they are turned on independently of the trap-control trigger. In addition, because the trap mode is entered by turning on a trigger and trap circuits are established by turning on associated triggers, it is common to use the expression turn on trapping.

The major portion of trap control is assigned to the trap-control trigger. This trigger can be turned on by executing a Transfer and Restore Traps (TRT) instruction. When the trap-control trigger is on, its output signals condition the memory protect, parity, interval timer overflow, and channel trapping circuits. Consequently, these types of traps cannot occur when the trap-control trigger is off. Another means of turning on the trap-control trigger is to execute a Transfer and Restore Parity (TRP) instruction. Executing a TRP causes all that a TRT causes and, in addition, turns on the parity-mode trigger. This trigger must be on to trap on any parity error.

The memory-protect-mode trigger is turned on only by executing a Set Protect Mode (SPM) instruction. Note that execution of an SPM instruction with the memory-protect-mode trigger already on results in a memory protect violation trap. If a parity error occurs on the SPM instruction, a parity trap results and is given priority.

The channel-trap-control trigger can be turned on by executing either an Enable (ENB) instruction or a Restore Channel Trap (RCT) instruction. An ENB instruction also serves to set up the data channel trap mask. Each data channel contains a 4-bit mask register which is loaded with selected bits from the contents of the instruction word effective address. The bits loaded into a mask register may be either 0 or 1. A 0 mask register bit negates the associated condition for trapping, whereas a 1 mask register bit allows trapping when the associated condition occurs. The following listing defines the bits used to load each mask register:

Mask Bit	Conditions Enabled	Channel	Effective if a 1 in Bit Position
Operation	Operation Complete or EOF or Word Parity or Unusual End or End	A	35
		B	34
		C	33
		D	32
		E	31
Direct Data	Direct Data Interrupt	B	25
		C	24
		D	23
		E	22
Parity	Word Parity or Redundancy Check	A	17
		B	16
		C	15
		D	14
		E	13
Attention	1401 Interrupt or Teleprocessing Interrupt	A	8
	CIF Attention	B	7
		C	6
		D	5
		E	4
Unit Record	Unit Record Interrupt	A	S

The RCT instruction is not associated with the mask register; that is, execution of an RCT instruction only turns on the channel-trap-control trigger. Consequently, an ENB instruction must be executed to define the valid trap conditions in addition to turning on channel trap control.

Why then is the RCT instruction available? After any trap occurs, the control trigger associated with that trap is turned off. To allow trapping during subsequent operations, trap controls must be restored after the present trap is dealt with. For channel trap operations, the RCT instruction is provided: after a channel trap has been handled, executing an RCT instruction restores the ability to trap (turns on the channel-trap-control trigger) without affecting the mask. To turn off channel trap control without a trap, an ENB instruction referencing a cleared location is executed. In this case, the control trigger is on, but no trap conditions are validated because of the zero masks. Channel trapping can be turned off manually by depressing the RESET pushbutton on the operator's console. However, depressing this pushbutton also resets all registers and indicators in both the CPU logic section and the data channels; the pushbutton must therefore be used discreetly.

Memory protection is turned off each time a memory protect violation trap or an SPM trap occurs. It is restored by executing an SPM instruction. Only one instruction is provided, therefore, to establish memory protection. Further, memory protection may be turned off without waiting for or causing a trap by executing a Release Protect Mode (RPM) instruction. However, execution of an RPM automatically causes a trap regardless of the condition of trapping-control triggers. The RESET pushbutton may also be used to turn off memory protection.

The trap-control and parity-mode triggers can only be turned off by a trap. No instruction or manual means is available to reset these triggers. In fact, depressing the RESET pushbutton turns these triggers on. Normal operation therefore includes the trap mode. If a trap occurs that turns off either the trap-control or parity-mode trigger or both, executing either a TRT or TRP, depending on the situation, will restore the desired trap control.

TYPES AND PRIORITY

There are approximately 15 types of traps in the 7040-7044 system. These types follow in order of priority:

1. Interval Timer Blast
2. Memory Protect Violation
3. Parity
4. Instruction - SPM
   RPM
   Floating Point
   STR
5. Pre-interrupt Memory Protect
6. Interval Timer Overflow
7. Direct Data

8. Channel E
9. Channel D
10. Channel C
11. Channel B
12. Channel A

The instructions in 4 have equal priority. This situation does not pose any problem, because only one instruction can be executed at a time.

In listing the channel traps, channel E was given highest priority for simplicity. Actually, the channel electrically farthest from core storage gets highest priority. This channel can be any of the overlapped channels. In fact, the arrangement of channels B through E with regard to priority has no limitations.

Under any circumstances, channel A has the lowest priority, because it is always the closest to core storage.

Interval Timer (IT) Blast

Every 16 2/3 milliseconds, the interval timer requests two memory cycles to read out core storage location 00005, increment it by 1, and place the incremented value back in location 00005. The two cycles required for this operation can occur only (1) between instructions, (2) during an RDS, WBT, PRD, SEN, WRS, PWR, CTR, BSR, REW, RUN, or WEF instruction if a wait is necessary for the channel, and (3) between unoverlapped cycles of an RCHA instruction. There are instructions in the instruction set and possible error conditions which prevent honoring of the interval timer request. If a second request for these cycles is made by the interval timer before the first request is honored, an interval timer blast trap occurs.

The interval timer blast trap does not allow completion of the instruction in process. It resets all data channels, including channel A. It does not reset the AC or MQ register. It stores the contents of the instruction counter, normally the present instruction location plus 1, in positions 21-35 of location 00036, and the computer takes its next instruction from location 00037. Trap control is turned off, thus inhibiting all other traps, and the two waiting interval timer cycle requests are reset. This action means that the contents of location 00005 are 2 less than they should be when an interval timer blast trap occurs. An interval timer blast trap also resets the interval timer overflow trap request if it is on.

Memory Protect Violation

A memory protect violation trap occurs when:

1. An RPM instruction is executed (RPM trap).
2. Memory-protect-mode trigger is on when an SPM instruction is executed (SPM trap).
3. The program attempts to store in a protected area while the memory-protect-mode trigger is on and the trap-control trigger is on (violation trap).
4. Memory-protect-mode trigger is on and the trap-control trigger is on and a channel, direct data, or interval timer overflow trap is requested (pre-interrupt memory protect trap).

Input operations on any channel are allowed to store anywhere without causing a memory protect violation trap.

Occurrence of any of the four traps listed above turns off the memory-protect-mode trigger and causes the location of the next sequential instruction to be stored in bits 21 through 35 of core storage location 00032. The next instruction to be executed is then obtained from location 00033. In the case of a pre-interrupt memory protect trap 00033 is stored in the address field of the location appropriate to the trap which caused the pre-interrupt memory protect trap.

The following bits are set in the location 00032 decrement field to identify the cause of the memory protect violation trap:

a. 14 - RPM executed with memory portect mode off.
b. 15 - RPM executed with memory protect mode on.
c. 16 - Violation trap - or an SPM trap.
d. 17 - Pre-interrupt memory protect violation trap.

Parity

Parity trapping is closely associated with machine cycles; therefore, the following cycle definitions are given:

1. I - A cycle used to fetch an instruction.
2. IA - A cycle used to access an indirect address.
3. E - A cycle taken to read or store in the execution of an instruction.
4. B - A cycle used to accommodate transfers to and from an I-O device on an overlap channel (the store cycle of an SCH and the readout of an IORD in an RCH are E cycles, not B cycles).
5. U - A cycle used to accommodate transfers to and from an I-O device on channel A (the store cycle of an SCHA and the readout of the IORD in an RCHA are E cycles, not U cycles).

6. C - An interval timer cycle to either readout or store into location 00005.

Since no parity is kept with CPU registers, a word that is stored has a check bit generated for it as it is stored. Therefore, CPU information is checked only during read cycles, which include I, IA, E, and C read cycles. If a parity error occurs during a read cycle, the word is returned to its original location in error. Parity is also checked during B and U cycle read and store activities. If a parity error is encountered during an I-O store cycle, the word is stored with a corrected parity bit.

The following partial-word store instructions require one I cycle and two E cycles: STA, STL, SAC, SXA, SXD, STD, and TSL. The first E cycle serves to read out and check the store location. The memory word is regenerated as it was during the first E cycle. If a parity error is detected, a parity trap is initiated and the instruction is not completed. However, if no error is detected during the first E cycle, the storage word is placed in the CPU storage register, and the applicable portion of the storage register is replaced with the new information. During the second E cycle, the complete storage register word is stored, and no parity error can occur.

If a parity error occurs during an I or IA cycle with the parity mode and trap-control triggers both on, the instruction is not executed. The location of the instruction in error, plus 1, is stored in the location 00040 address field. The address of the instruction in error is stored in the location 00040 decrement field. Location 00040 bit 18 is set to indicate that the error occurred during an I or IA cycle. The next instruction to be executed is obtained from location 00041.

If a parity error occurs during an E cycle with the parity-mode trigger on, the instruction is not executed and the location of the instruction in error, plus 1, is placed in the location 00040 address field. The address of the instruction in error is placed in the location 00040 decrement field. Location 00040 bit 19 is set to indicate that the error occurred during an E cycle. The next instruction to be executed is obtained from location 00041.

If a parity error occurs during a C cycle with the parity-mode and trap-control triggers both on, the computer waits until the instruction being executed is completed, and then the location of the next sequential instruction is placed in the location 00040 address field. Location 00040 bit 1 is set to indicate the error occurred during a C cycle. The next instruction to be executed is fetched from location 00041.

If a parity error occurs during an I, IA, E, or C cycle when either the parity-mode or trap-control trigger is off, the execution of instructions is not interrupted until both are turned on. At this time, the location of the next instruction to be executed is placed in the location 00040 address field. Location 00040 bit S is set to 1 to indicate a stacked parity error. Bits are set in location 00040 positions 1, 18, and 19 to indicate the type of cycle in which a stacked error occurred. One or all of these bits can be set in a stacked error parity trap.

Instruction

There are four types of instruction traps: SPM, RPM, floating point, and STR. They have equal priority, and each can occur when the machine is not in the trap mode.

SPM

If the memory-protect-mode trigger is on when an SPM instruction is given, an SPM trap results. The location of the SPM instruction, plus 1, is placed in the location 00032 address field. Location 00032 bit 16 is set to indicate the type of trap. When the SPM trap occurs, it causes a memory protect violation trap to occur. The next instruction to be executed is obtained from location 00033.

RPM

Execution of an RPM instruction causes the location of the RPM instruction, plus 1, to be placed in the location 00032 address field. Location 00032 bit positions S through 20 are made 0's. The next instruction to be executed is obtained from location 00033. If the memory-protect-mode trigger is on when an RPM instruction is executed, it is turned off and location 00032 bit position 15 is set to 1. If the memory-protect-mode trigger is off when the RPM instruction is executed, location 00032 bit position 14 is set to 1.

Floating Point

During the execution of floating-point instructions, the result characteristic in the accumulator and MQ register may exceed eight bit positions, thus meaning the result is too large for storage. The capacity of the machine is exceeded when the exponent goes above $+177_8$ or below -200_8. Above $+177_8$ is called overflow, and below -200_8 is called underflow. Overflow and underflow may occur in either the accumulator or the MQ register. Upon sensing either condition, the CPU places the address plus 1 of the instruction that caused the condition into the location 00000 address field. The following location 00000 bits are also set to indicate the type of error:

1. 12 - Double-precision instruction word effective address specifies an odd location.
2. 14 - Single-precision divide instruction.
3. 15 - Overflow in either the accumulator or the MQ register or both.
4. 16 - Accumulator overflow or underflow.
5. 17 - MQ register overflow or underflow.

Location 00010 is then accessed for the next instruction to be executed.

STR

The location of the STR instruction, plus 1, is placed in the location 00000 address field. Positions S through 20 are made 0's. The next instruction to be executed is fetched from location 00002.

Pre-Interrupt Memory Protect

If the memory-protect-mode trigger and the trap-control trigger are both on and a channel, direct data, or interval time overflow trap is requested, a pre-interrupt memory protect trap results. The memory-protect-mode trigger is turned off, and the location of the next sequential instruction is placed in the location 00032 address field. Location 00032 bit 17 is set to indicate the type of trap. The requested trap is then performed.

Interval Timer Overflow

When the interval timer increments location 00005 and an overflow occurs, a trap is requested. This trap cannot occur unless the trap-control trigger is on. Further, it cannot occur between the execution of a privileged instruction and the execution of the instruction following the privileged instruction. If the memory-protect-mode trigger is on when an interval timer overflow occurs, it must be determined that no pre-interrupt memory protect trap is present before the interval timer overflow trap can be handled (pre-interrupt memory protect trap has higher priority).

Upon honoring an interval timer overflow trap, the contents of the instruction counter (normally the location of the next sequential instruction to be performed in the main program) replace positions 21-35 of location 00006 and the computer takes its next instruction from location 00007.

When an interval timer overflow trap request is waiting, the interval timer is blocked from incrementing location 00005. If the interval timer trap request waits more than 33 milliseconds, an interval timer blast trap will occur which resets the interval timer overflow trap request.

Direct Data

A direct data trap is a means of enabling overlapped channels to signal or interrupt processing by trapping. When a direct data trap occurs, the contents of the instruction counter (the location of the next sequential instruction) are placed in the location 00003 address field. The location 00003 decrement field is used to identify the channel making the direct data trap request. Location 00004 is then accessed for the next instruction. Note that the instruction in location 00004 must be an unconditional transfer instruction to ensure compatibility with the 7090.

A direct data trap can occur only when the trap control trigger is on and the channel-trap-control trigger is on. Further, a direct data trap cannot occur between the execution of a privileged instruction and the execution of the instruction following the privileged instruction. A direct data trap turns off the channel-trap-control trigger, thereby preventing other direct data traps and channel traps until the channel-trap-control trigger is again turned on with either an ENB or RCT instruction.

Each channel has an associated four bit mask register, of which one bit controls direct data interrupt requests from that channel. This mask bit can be made a 0 or a 1 by executing an ENB instruction. In addition, each overlapped channel contains a latch which is turned on only by the direct data device, and then only when a trap is requested. Recognition of the trap request, however, is possible only if the associated mask bit is a 1 coincident with the latch being on. When the latch is on but the associated mask bit is a 0, no recognition is possible. Further, when a direct data trap is honored, the associated latch effects storage of a 1 into the corresponding decrement field bit position in location 00003:

13 - Channel E
14 - Channel D
15 - Channel C
16 - Channel B

If the trap is honored and the decrement bit is set, the latch is turned off. If the mask bit prevents honoring of the trap, the latch remains on. However, a direct data latch for a particular channel may also be turned off by executing an RCH instruction addressing that channel or by depressing the RESET pushbutton.

Channel Traps

A channel trap allows a particular channel to signal or interrupt processing by trapping the CPU program. Channel traps may be initiated by the following:

1. Completion of any channel operation.
2. A redundancy check.
3. An end of file.

4. A word parity check (U or B cycles only).
5. Tape word incomplete or corporate interface unusual end.
6. Corporate interface attention.
7. 1401 attention (channel A only).
8. Teleprocessing interrupt (channel A only).
9. Unit record interrupt (channel A only)

When a channel trap occurs, the instruction counter value (the location of the next sequential instruction) is stored in the trap record location address field. Bits indicating the conditions which caused the trap are set in the trap record location decrement field. All other bit positions in this location are 0. The next instruction to be executed is obtained from the next sequential address after the trap record location: the instruction location. Trap record and instruction locations for each channel are as follows:

Channel	Trap Record Location	Instruction Location
A	00012	00013
B	00014	00015
C	00016	00017
D	00020	00021
E	00022	00023

Note that instructions in the instruction locations must be unconditional transfer instructions to be compatible with the 7090.

A channel trap can occur only when both the trap-control and channel-trap-control triggers are on. A channel trap cannot occur between the execution of a privileged instruction and the instruction following the privileged instruction. Further, a channel trap turns off the channel-trap-control trigger, thus preventing other channel traps and direct data traps until the channel-trap-control trigger is again turned on; this action can be accomplished by executing either an ENB or an RCT instruction.

Each channel employs a 4-bit mask register to specify the currently valid trapping conditions. The mask bits are arranged by an ENB instruction. If an all -0 mask is desired (no valid traps), depress the CLEAR, RESET, or LOAD pushbutton on the operator's console, execute an RDC instruction, or execute an ENB instruction referencing a cleared storage location.

For each condition that can cause a channel trap there is a latch, which can be turned on and off by certain conditions. A latch can only request a trap if the associated mask bit is a 1. When a trap request is made, each latch which is on places a 1 in the corresponding decrement field bit position of the trap record address. If a latch is on but the associated mask bit is a 0, the latch can be turned on or off, but a trap request cannot be recognized and a trap record location decrement field bit cannot be set. All latches in a particular channel are turned off by an RDC addressing that channel, or by depressing the appropriate operator's console pushbuttons.

When a trap request is recognized, the associated latch is automatically turned off. If recognition is prevented by a 0 mask bit, the associated latch remains unaltered.

The following table (Figure 56) gives the various decrement field bits used in the trap record locations, the associated latches, the name of the associated mask bit, and the meaning of each.

FIGURE 56. CHANNEL TRAPS TABLE (THEIR IDENTIFICATION AND MEANING)

Decrement Field Bit Position	Latch	Mask Bit	Remarks
8	Unit Record Interrupt	Unit Record	The latch is turned on whenever the following devices attached to the 1414-III or IV have completed their cycle: card read buffer full; paper tape reader full; card punch buffer empty; print buffer empty. The latch cannot request a trap unless the channel is not in use. This type of trap applies only to channel A.
9	Tele-processing Interrupt	Attention	The latch is turned on whenever an inquiry buffer in the 1414-IV or V has a message waiting, when an output buffer has emptied. Included in this area are local inquiry, teletype, and 1009. The latch is masked by the attention bit and can request a trap even when the channel is in use. This type of trap applies only to channel A.
10	1401 Attention	Attention	The latch is turned on by the 1401 and is masked by the attention mask bit. The latch can request a trap even when the channel is in use. This type of trap applies only to channel A.
11	Corporate Interface Attention	Attention	The latch is turned on by the corporate interface attention line and is masked by the attention mask bit. The latch can request a trap and store into the trap record location even when the channel is in use. This type of trap is not applicable to channel A.

Decrement Field Bit Position	Latch	Mask Bit	Remarks
12	Unusual End	Operation	The latch is turned on at the end of a tape operation if the total number of characters handled was not a multiple of 6. The latch is not used when an end of file is read. The latch is also turned on by the corporate interface unusual end line to indicate some unusual condition. A sense operation is generally required to determine the condition. The latch is masked by the operation mask bit and cannot request a trap unless the channel is not in use. This type of trap is not applicable to channel A.
14	Word Parity	Parity or Operation	The latch is turned on by a word parity error during read or write U or B cycles to memory. It may also be turned on during channel write operations by checking the 37th bit of a word with the sum of the six parity bits of a disassembled word. When the parity mask bit is a 1 and the word parity latch is on, the channel stops the transfer of information to or from memory. The channel address register contains the address, plus 1, of the last word transferred. Therefore, if the parity enable bit is 1 when an invalid word is fetched from memory during a write operation, executing an SCH will locate the invalid word if 1 is subtracted from the address. This latch can be recognized only when the channel is not in use. Note that the latch is enabled by two different mask bits. The parity mask bit stops channel transmission when an error occurs; the operation mask bit does not.
15	End of File	Operation	The latch is turned on by the end-of-file signal from the I-O device. When the associated channel operation mask bit is 0, the latch may be tested and turned off with a TEF instruction. When the associated mask bit is a 1, the TEF does not transfer or turn off the latch. Latch recognition is possible only when the channel is not in use.
16	Redundancy Check	Parity	The latch is turned on by a parity check received from the I-O device or by the byte parity check in the channel. When the associated mask bit is 0, the latch can be tested and turned off by a TRC instruction. When the associated mask bit is 1, a TRC neither transfers nor turns off the latch. Coincidence of a 1 in the mask bit and the latch being on causes information transfers to stop. The channel address register contains the address, plus 1, of the last word transferred. Latch recognition occurs only when the channel is not in use. For read operations, the channel remains busy for the entire record, although nothing is transferred to memory.
17	Operation Complete	Operation	The latch is turned on whenever the channel in use indicator goes from on to off, which is at the completion of every read, write, sense, and control operation, when the tape completes a BSR or WEF, or after relays are picked for a RUN or REW. If a BSR or REW is given at load point, the latch is turned on, although no mechanical motion occurs. When the channel in use indicator goes from off to on, the latch is turned on.

TRAPPING SCHEME

The trapping scheme is shown in Figure 57. Although not every possible situation is covered in the following paragraphs, enough examples are given for a thorough understanding of trapping.

Channel Trap

Assume that a tape read operation is programmed using channel B, and that the instructions used precede entry of the program into an arithmetic loop. This condition could be programmed as follows:

```
100    RTBB    1
101    RCHB    WRD
102    CLA     B
103    ADD     C
104    SUB     D
105    MPY     E
```

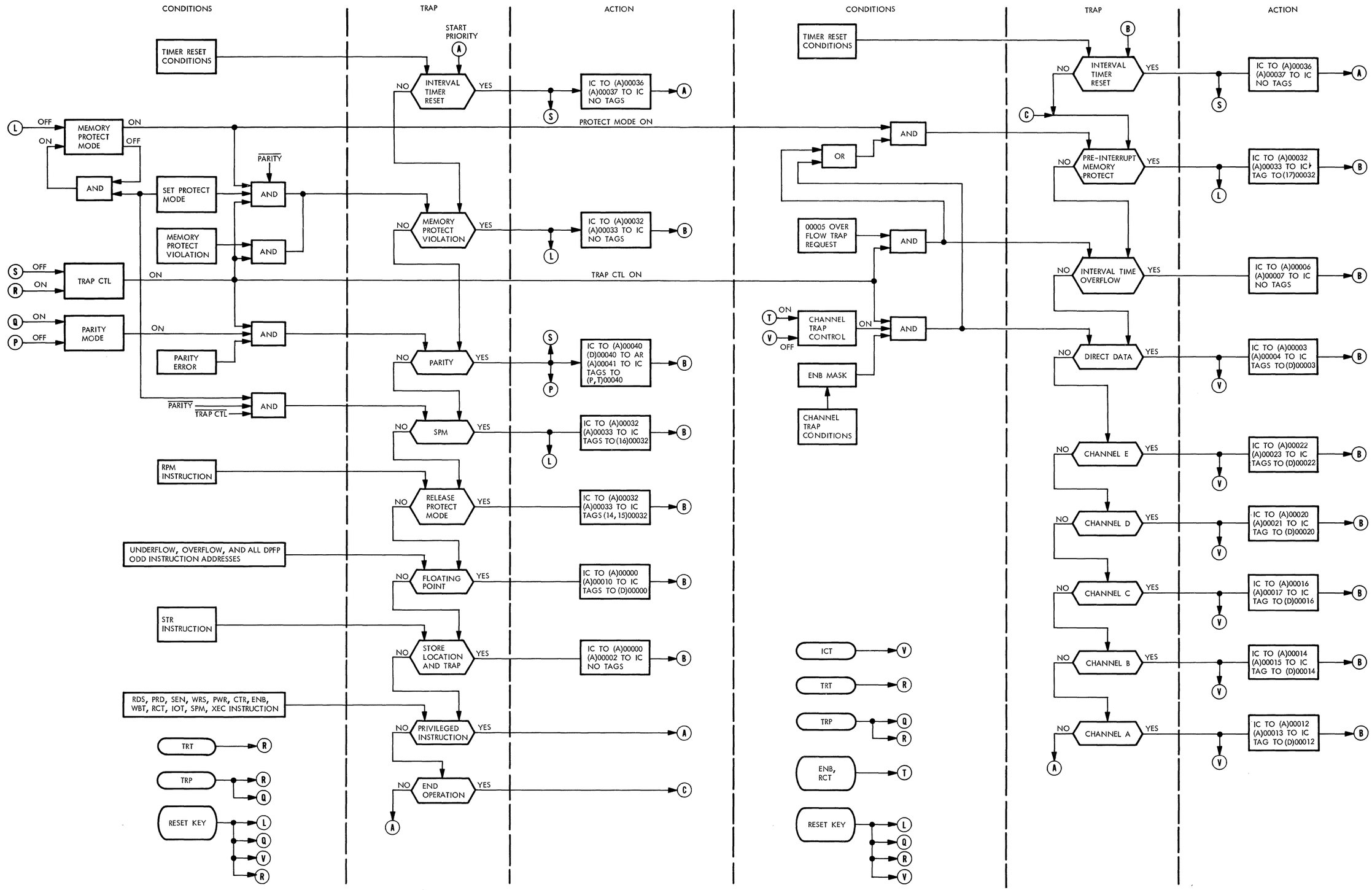


FIGURE 57. TRAPPING SCHEME

```
106     STO     F
107     TPL     103
110     TRA     **
```

By the time channel B is ready to transfer data, this simple routine is looping through the arithmetic activities. This looping continues throughout the data channel B transfers. Data channel B manifests it is ready to stop transferring by generating a trap request. Therefore, the machine must be in the trap mode, the data channel trap control circuits must be on, and the associated mask must be of a configuration that validates this particular type of trap. For simplicity, assume the following mask was previously established by an ENB instruction:

Name	Att	Parity	Direct Data	Oper
Bit	0	0	0	1
Ref Loc Bit	7	16	25	34

It is impossible to predict just where in the arithmetic loop the trap request will occur, so assume during the execution of the subtract instruction the channel RCHB (read) operation is finished. As soon as the channel goes "not in use," the trap request is generated. The trap request is honored at the completion of the instruction in progress (in this case, the subtract instruction).

To trace the action for this data channel B trap operation, enter Figure 57 at point A. Since this is not an interval timer request trap, a memory protect violation trap, a parity trap, an SPM trap, an RPM trap, a floating-point trap, an STR trap, or a privileged instruction, the answer to each decision as to the type of trap being requested is no and the scan (which is what these decision blocks represent) falls through to the end operation decision block. It has already been stated that the trap request cannot be generated until the end of the operation (instruction) in progress. Since the priority is determined as a result of the trap request, the answer to this question not only is yes, but was yes upon entering the trap priority circuits. The yes condition directs priority determination to point C. Point C marks the beginning of the next group of decisions to be made concerning the type of trap, or, said another way, point C marks the beginning of the next level of priorities in the trapping scheme.

Starting at point C, the priority circuits, in effect, ask whether this trap request is for a pre-interrupt memory protect trap, an interval timer overflow trap, a direct data trap, a channel E trap, a channel D trap, or a channel C trap. The answer to each is no, and the priority determination falls through to the decision channel B trap block. At this block, a yes is realized. This yes condition results in the routing of a signal to V, which is the turn-off input to the channel-trap-control trigger. Thus, upon recognition of the particular data channel trap, action is taken to prevent any other channel trap from occurring until the current data channel trap is satisfied. Simultaneously with the turn-off action, the present contents of the instruction counter are stored in location 14_8. These contents specify the location of the subtract instruction plus 1, or the location of the multiply instruction (105_8). Along with storing the instruction counter contents, location 14_8 tag bit 17 (operation complete) is set to identify the type of trap. Following this action, the next instruction to be executed is fetched from location 15_8. This instruction must be, in this case, a TRA to a trap routine for 7090 compatibility.

The channel trap routine may be formed in many ways; however, it probably will determine the cause of the trap. Location 14_8 is therefore referenced by the routine and brought into the CPU. The trap cause is determined by checking location 14_8 bit 17. This bit can be checked in various ways. One way is to put the contents of location 14 in the accumulator and then shift the accumulator left 17 positions. This shift places location 14_8 bit 17 in accumulator bit position P. A P bit test can then be made. In any event, the check of tag bit 17 of location 14_8 must, in this case, effect the insertion of an address into the address field of location 110_8. This inserted address, in turn, references another routine which acts on the data just read from the tape. After the address is inserted, an RCT instruction is executed to turn on the channel-trap-control trigger. Immediately following execution of the RCT, a TRA 14_8 is executed which returns program control to the original program at location 105_8.

The functions of the trap routine may be summarized as follows:

1. Check location 14_8 tag bit 17.
2. Effect the insertion of an address in the location 110_8 address field.
3. Restore channel trap control as the next-to-last step in the routine.
4. Return control to the original routine as the last step in the trap routine.

Pre-Interrupt Trap

Using identical conditions as those assumed in the channel trap discussion, further assume that the memory-protect-mode trigger is turned on. Again, enter Figure 57 at point A, and examine each of the decision blocks. Each block yields a no until the end operation block is entered. Here, a yes results and the scan is directed to C.

The first decision to be made at this point is whether this trap is a pre-interrupt memory protect. The answer is yes. Before proceeding, notice the

conditions necessary for this type of trap. First, the memory-protect-mode trigger must be on. Second, coincidence of the trap-control trigger being on and a timer overflow trap request must occur with the first condition, or the trap-control trigger must be on along with the channel-trap-control trigger and the associated mask must be other than all 0's coincident with the first condition. Thus, conditions which honor channel traps or an overflow trap along with the memory-protect-mode trigger being on constitute the conditions for a pre-interrupt memory protect trap.

During the last I cycle of the subtract instruction execution (the time during which end operation occurs and when normally the next instruction is fetched), the trap controls are set, thereby recognizing the trap request. During the following E cycle, the signal is generated to point L on the diagram which turns off the memory-protect-mode trigger. Further, during this E cycle, the contents of the instruction counter are stored in location 32_8, and bit position 17 of this location is set to identify the trap. Once the memory-protect trigger is turned off, the pre-interrupt memory protect trap is completed.

Following this E cycle, an I cycle occurs during which location 33_8 is normally referenced for the next instruction to be executed. Instead, however, referencing of location 33_8 is blocked, and the channel trap is honored by setting up the channel trap controls. An E cycle is then entered during which the contents of the instruction counter, which contains 33_8, are stored in location 14_8, and tag bit 17 is set. The channel-trap-control trigger is turned off, and then location 15_8 is referenced for the next instruction.

The location 15_8 instruction, again, must be a TRA to a trap routine, which references location 14_8, and checks bit 17 of that location to determine the cause of the trap. Checking this bit must effect insertion of an address into the location 110_8 address field which will reference a routine designed to take advantage of the data just read from the tape. The final actions in the trap routine must be as follows:

1. An RCT to restore the channel trap circuits.
2. A TRA to location 14_8, which must contain a TRA.
3. Since the location 14_8 address field contains 33_8, the transfer to location 14_8 causes, in turn, a transfer to location 33_8.
4. Location 33_8 must contain a TRA 32 which transfers program control to location 32_8. This location contains the original point of departure and effects transfer to that point.

In addition, somewhere in the trap routine prior to restoring the channel trap circuits, provision must be made for restoring the memory-protect-mode trigger.

Privileged Instruction Trap

If a trap request is generated during the execution of a privileged instruction, the trap is not recognized until the instruction following the privileged instruction is executed. This limitation is necessary because instructions classified as privileged are instructions that involve (1) data transfers, (2) the condition of trap control circuits, or (3) execution of an instruction out of sequence.

With instructions that involve data transfers, selection of tape operations is the basic consideration. When a tape is selected, an RCH must be given within 3 to 15 milliseconds after the select instruction. Since this timing restriction is mandatory for tapes, it simplifies the trapping scheme to make it standard for all select instructions. All select instructions are therefore considered privileged because of this time limitation. Consequently, no trap can be honored between execution of a select instruction and execution of an RCH instruction, except on an interval timer blast trap.

With instructions that involve the condition of trap control circuits (ENB, RCT, ICT, SPM), the major point of consideration is their execution at the end of a trap routine. In this case, it must be insured that program control is returned to the original routine before a new trap is recognized. Thus, the programmer can maintain a clear record of the point of departure with each trap, and under almost any circumstances return to that point.

The execution of an instruction out of sequence occurs when an XEC instruction is given. In this case, it is again important that program control be returned to the original sequence before a trap is recognized.

When a trap is requested during a privileged instruction, the priority determination starts at point A (Figure 57). Each decision block is examined and yields a no condition until the privileged instruction block is entered. Here, the yes condition is realized, and the action is looped back to point A. Between leaving the priority loop at the privileged instruction block and re-entering it at point A, the instruction following the privileged instruction is executed.

Floating-Point Trap

Assume that during the execution of a single-precision floating-point divide instruction, accumulator and MQ register underflow occur. Let the original accumulator characteristic, for example, be so small that underflow results from characteristic subtraction. Although a floating-point-trap-request signal is immediately generated, the request is not recognized until the instruction is

completely executed. During the last I cycle time, the end-operation phase of the instruction takes place. At this time, the request is recognized by setting up the trap controls: set the trap trigger, block instruction counter stepping, etc.

Following this I cycle, an E cycle is taken, during which the present instruction counter contents are stored in location 00000, and bits 14 and 17 of that location are set to identify the type of trap. In addition, during this E cycle, the value 00010_8 is placed in the instruction counter. The next instruction to be executed is obtained from location 10_8, and this instruction must be a TRA to a trap routine.

The trap routine referenced must inspect location 00000 to determine the cause of the trap. With the type of trap assumed, the trap routine would probably provide for the adjustment of the operands used in the divide so that a legal operation can be performed. The last step in the routine must be a TRA to location 00000, thereby returning control to the original program sequence.

TRAPPING EXECUTION

This portion discusses the timing of each individual trap, how each trap is enabled and requested, and how priority is established and executed.

To effect a trap routine requires one I and one E cycle. During I time, the trap is established according to priority, and the storage location in which to store data pertaining to the trap is determined. During E time, the contents of the instruction counter (and address register if a parity error) and flag bits are stored in the predetermined location in storage. Also during E time, the transfer-to location is determined, and the next instruction is taken from this transferred-to location. The program may take action on the trap, return to the point at which the trap occurred, or start a new part of the program.

The trapping scheme as discussed in this section is divided into five paragraphs:

1. General timing.
2. Trap enabling.
3. Trap request generation.
4. Trap priority.
5. Individual traps.

Figure 58 is a timing chart of the I time and E time of a trap, showing data transfer and control during a trap routine. The traps are listed from left to right according to trap priority. From top to bottom are listed the important controls, register setting, and data transfer. In addition, this chart reflects the addresses used during each individual trap routine.

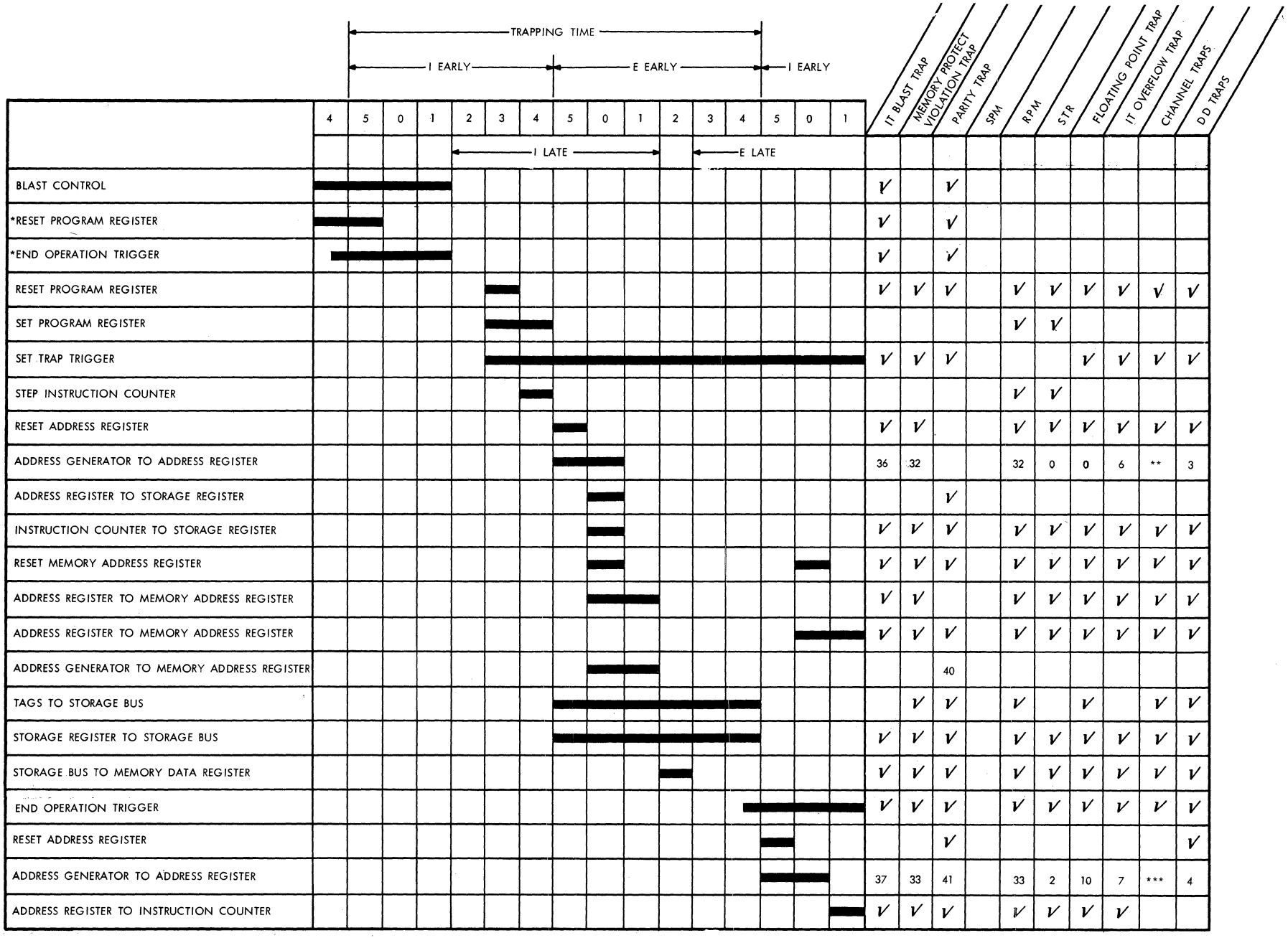
Unlike the RPM and STR instructions, the SPM trap (not included in Figure 58) does not cause an automatic trap. The SPM instruction is executed when not in memory-protect mode. If an SPM instruction in memory-protect mode is given, a memory-protect-violation trap request is generated. This trap has second highest priority.

Assume for this general discussion that all traps are enabled. If a trap is enabled, the trap is executed when a trap request occurs. The conditions that necessitate setting the blast-control trigger are a parity error and an interval timer blast. Occurrence of a parity error during an I or IA, E, or C cycle generates the parity request if parity traps are enabled. No other trap requests are honored until completion of the parity trap. A blast will not occur if a stacked parity trap occurs. If the interval timer (IT) has not been incremented for 33ms, an IT blast-trap request is generated. As shown in the timing chart (Figure 58), the blast-control trigger is set at A4 time. Setting this trigger causes an immediate end-operation (A4.5 time), and the program register is reset by an A4 D2 pulse. The object is to get into trapping I time immediately, regardless of the instruction cycle in which the IT blast-trap request occurred.

For an example of a typical trap, assume that an IT overflow trap request was generated and that all traps are enabled. At the completion of the instruction being executed, trapping I time is entered. At I3 time, the program register is reset (Figure 58), and the IT trap trigger is set. The address register is reset, and 00006 is loaded into the address register by late I5 D1 and late I5 D2 pulses. During E time, the instruction counter is sent to the storage register by an E0 D1 pulse. During E0 time, the MAR is reset and loaded to specify the storage location. The storage register is sent to the storage bus during all of E early time. The contents of the instruction counter are stored in location 00006. During E late time, bit 35 of the address register is set to a 1. Note that the address register is not reset at this time (E5 D1). The address is therefore equal to 00007. Location 00007, then, is the address to which the program is transferred before entering the trap-correction routine of the program. The address register is sent to the instruction counter at I1 time of the next instruction (location 00007).

Channel traps and DD traps are not discussed in detail in this section (see the sections dealing with the theory of operation of the channel), but general data is given to show the similarity between various traps.

The timing of any of the traps can be traced by referring to figure 58 and following the trap timing similarly to the method explained above for the IT overflow trap.



	IT BLAST TRAP	MEMORY PROTECT VIOLATION TRAP	PARITY TRAP	SPM	RPM	STR	FLOATING POINT TRAP	IT OVERFLOW TRAP	CHANNEL TRAPS	D D TRAPS
BLAST CONTROL	✓		✓							
*RESET PROGRAM REGISTER	✓		✓							
*END OPERATION TRIGGER	✓		✓							
RESET PROGRAM REGISTER	✓	✓	✓		✓	✓	✓	✓	✓	✓
SET PROGRAM REGISTER					✓	✓				
SET TRAP TRIGGER	✓	✓	✓				✓	✓	✓	✓
STEP INSTRUCTION COUNTER					✓	✓				
RESET ADDRESS REGISTER	✓	✓			✓	✓	✓	✓	✓	✓
ADDRESS GENERATOR TO ADDRESS REGISTER	36	32			32	0	0	6	**	3
ADDRESS REGISTER TO STORAGE REGISTER			✓							
INSTRUCTION COUNTER TO STORAGE REGISTER	✓	✓	✓		✓	✓	✓	✓	✓	✓
RESET MEMORY ADDRESS REGISTER	✓	✓	✓		✓	✓	✓	✓	✓	✓
ADDRESS REGISTER TO MEMORY ADDRESS REGISTER	✓	✓			✓	✓	✓	✓	✓	✓
ADDRESS REGISTER TO MEMORY ADDRESS REGISTER	✓	✓	✓		✓	✓	✓	✓	✓	✓
ADDRESS GENERATOR TO MEMORY ADDRESS REGISTER			40							
TAGS TO STORAGE BUS		✓	✓		✓		✓		✓	✓
STORAGE REGISTER TO STORAGE BUS	✓	✓	✓		✓	✓	✓	✓	✓	✓
STORAGE BUS TO MEMORY DATA REGISTER	✓	✓	✓		✓	✓	✓	✓	✓	✓
END OPERATION TRIGGER	✓	✓	✓		✓	✓	✓	✓	✓	✓
RESET ADDRESS REGISTER			✓							✓
ADDRESS GENERATOR TO ADDRESS REGISTER	37	33	41		33	2	10	7	***	4
ADDRESS REGISTER TO INSTRUCTION COUNTER	✓	✓	✓		✓	✓	✓	✓		

* OCCUR AS A RESULT OF SETTING THE BLAST CONTROL TRIGGER

CHANNEL	** STORE LOCATION	*** TRANSFER LOCATION
A	00012	00013
B	00014	00015
C	00016	00017
D	00020	00021
E	00022	00023

FIGURE TRAP TIMING

Trap Mode Setup

Four triggers control the honoring of the trap and trap requests:

1. Trap-control trigger.
2. Parity-mode trigger.
3. MP-mode trigger
4. Channel-trap-control trigger

Figure 59, a table, lists the traps, the conditions for generating a trap-request level, the conditions for setting the trap trigger, and the trap-control triggers that are reset as a result of a particular trap. In Figure 60, which supplements the table in Figure 59, the triggers that enable trapping are shown in heavy-weight lines; dashed lines indicate the trap, and normal-weight lines indicate the important logical actions that occur before entering the trap routine. As shown in Figure 59, not all traps are controlled by the triggers listed above. For example, the only means of preventing an interval-timer-blast trap is to turn off the STORAGE CLOCK switch, and, since a floating-point error results in erroneous computations, a floating-point trap is always honored. RPM and STR are instruction traps. All other traps are under control of the trap-control trigger and one other (parity-mode, MP-mode, or channel-trap-control) trigger. A parity trap cannot occur without the trap-control and parity-mode triggers set; a memory-protect-violation trap cannot be honored if the trap-control and MP-mode triggers are reset. If an SPM instruction is executed in MP mode, the memory-protect-violation trap request is honored without additional trap restrictions.

If a memory-parity error occurs during readout from memory of any instruction, including an RPM, STR, or SPM instruction, the parity trap is always executed (if traps are enabled).

Trap Requests

Three conditions determine the honoring of a trap:

1. No enabling is needed to generate the trap request or to execute the trap.
2. A trap request is generated and honored when the trap is enabled.
3. A trap condition may exist, but a trap request is never generated.

Since it is possible for the CPU to hang up on instructions that inhibit C cycles, the IT blast-trap request is generated and honored to eliminate the hang condition. Figure 61 shows how the interval-timer-blast request is generated. No enabling is necessary to execute this trap.

The second trap that is executed without enabling is the floating-point trap. The conditions that cause a floating-point trap are an overflow, an underflow, and a double-precision operand address odd. Figure 62 shows how the floating-point trap request is generated.

The trap conditions that generate a trap request and that are honored when the trap is enabled are shown in Figures 63, 64, and 65. These traps are the parity trap, the MP violation trap (which is the pre-interrupt memory protect), interval timer overflow, channel traps, and direct-data traps. If a parity error occurs when traps are not enabled, the stacked PT trigger is set. When the trap-control and parity-mode triggers are set, a parity trap may occur. The stacked-PT trigger generates a flag bit (S position) and a flag bit indicating the cycle in which the parity occurred (I or IA, E, or C). Note that a blast does not occur if the parity trap is delayed. A flag in the S position of the stored location indicates that the contents of the instruction counter and address register as stored in memory do not indicate the point of error. An IT overflow trap request and a DD or channel-trap demand are also honored when the traps are enabled (Figure 64).

A pre-interrupt memory protect causes an MP violation trap routine to be executed. The MP violation request is not generated, but the MP-trap trigger is set and a trap routine is executed. The conditions that cause a pre-interrupt memory protect are shown in Figure 64. The pre-interrupt memory protect trap (MP-violation trap routine) resets the MP-mode trigger. The trap condition still exists, trap priority is re-established, and the original trap request is honored.

Another trap condition that may exist and never be honored if traps are not enabled is memory-protect-violation (Figure 65). Note that the SPM memory-protect-violation trap is not controlled by the trap-control trigger.

Request Recognition

Since it is possible for more than one trap request to be generated at any given time, a priority scheme must be established. Figure 66 is a flow diagram of the trapping priority scheme. The order of priority is as follows:

1. Interval timer blast.
2. Memory protect violation.
3. Parity.
4. Instruction traps.
   a. SPM
   b. RPM
   c. STR
   d. Floating point
5. Pre-interrupt memory protect.
6. Interval timer overflow.
7. Direct data.
8. Channel traps (channels A-E).

FIGURE 59. TRAP-ENABLE CONDITIONS TABLE

Trap Name	Conditions For:		Control Triggers Reset As a Result of a Trap
	Trap Request Generation	Setting the Trap Trigger	
Interval timer blast	Storage clock on and no C cycle within two clock cycles	IT blast request	Trap control
Memory protect violation	1. Storing in a protected area in memory and trap-control and MP-mode triggers set. 2. SPM instruction when the MP-mode trigger is set.	MP violation trap request	MP mode
Parity	1. Trap-control and parity-mode triggers set and memory parity error. 2. If the trap-control or parity-mode trigger is reset, the stacked-PT trigger is set. The parity-trap-request level will cause a trap after both these triggers are set.	Parity trap request	1. Trap Control 2. Parity Mode
RPM	Instruction trap	RPM instruction	MP mode (if set)
STR	Instruction trap	STR instruction	None
Floating point	Overflow, underflow, and/or DPFP odd address	Floating-point trap request	None
Pre-interrupt memory protect	IT overflow trap request or DD trap request, or channel trap request and MP mode trigger set. *	1. Interval timer overflow trap request and not-privileged instruction and MP-mode trigger set. 2. MP-mode trigger set and (DD trap request or channel trap request) and not-privileged instruction.	MP mode
Interval timer overflow	Overflow	Not MP-mode trigger and trap-control trigger set.	None

* Cannot get DD or channel trap requests unless the trap-control and channel-trap-control triggers are set.

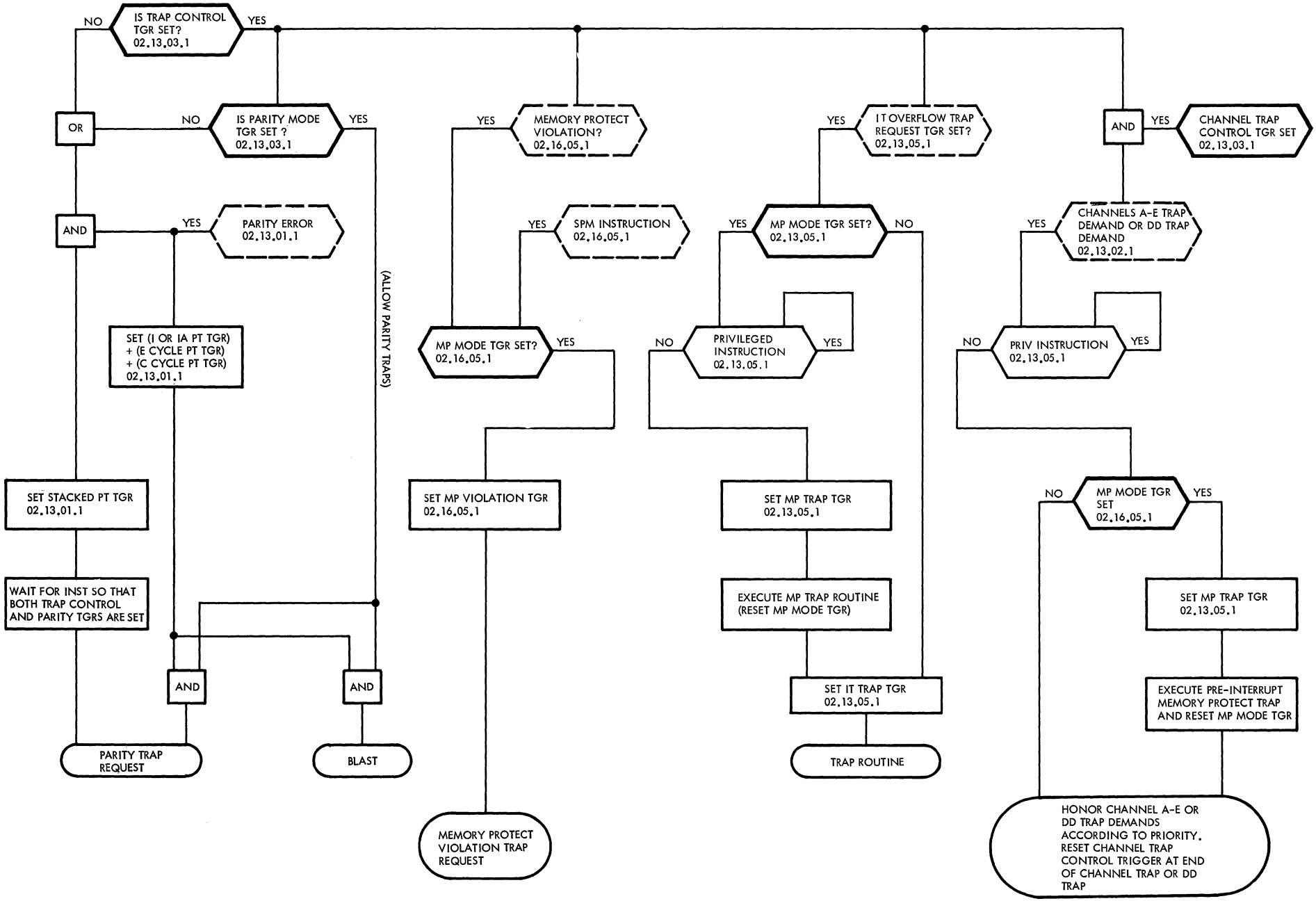


FIGURE 60. TRAP ENABLE SCHEME

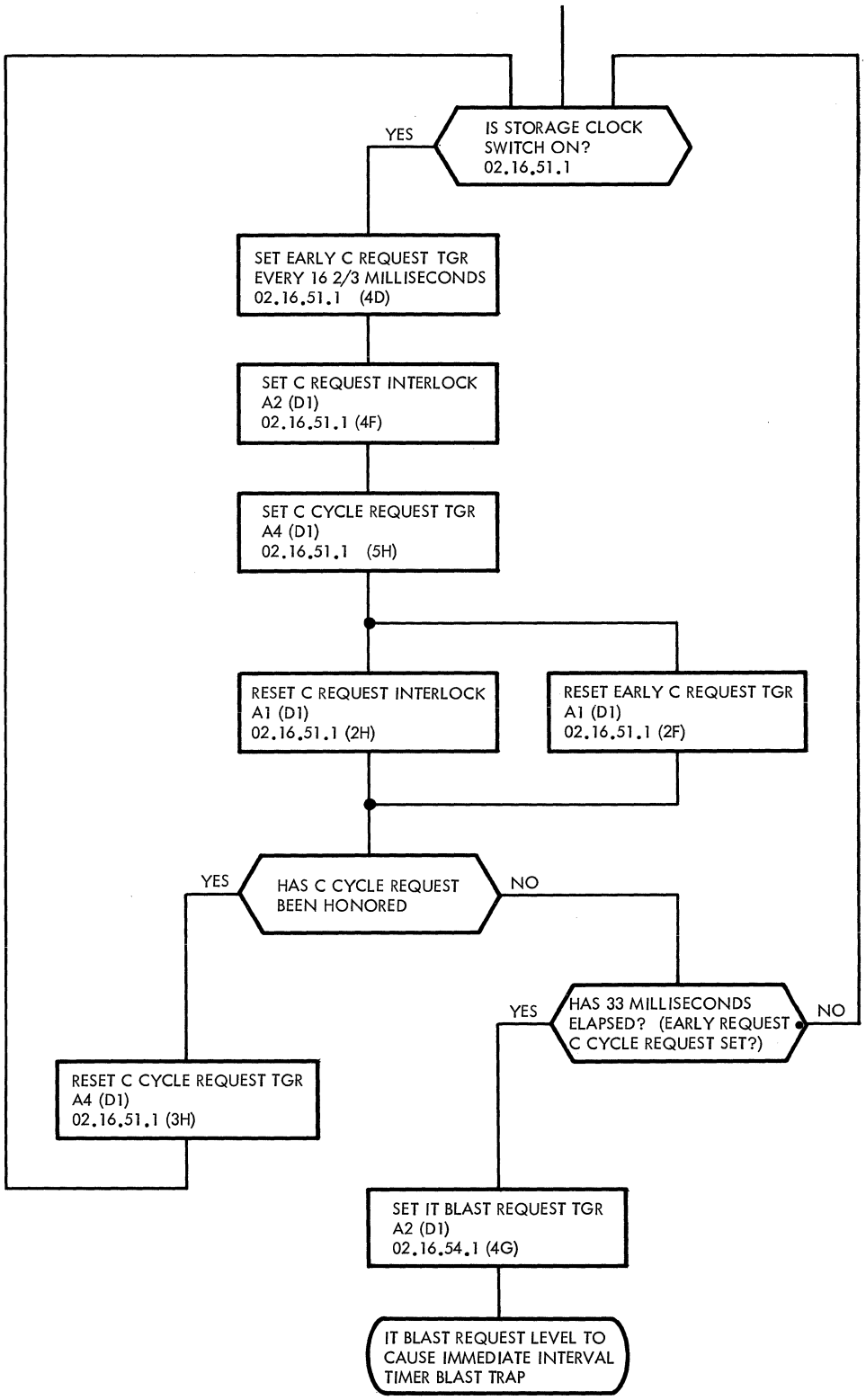


FIGURE 61. INTERVAL TIMER BLAST TRAP REQUEST

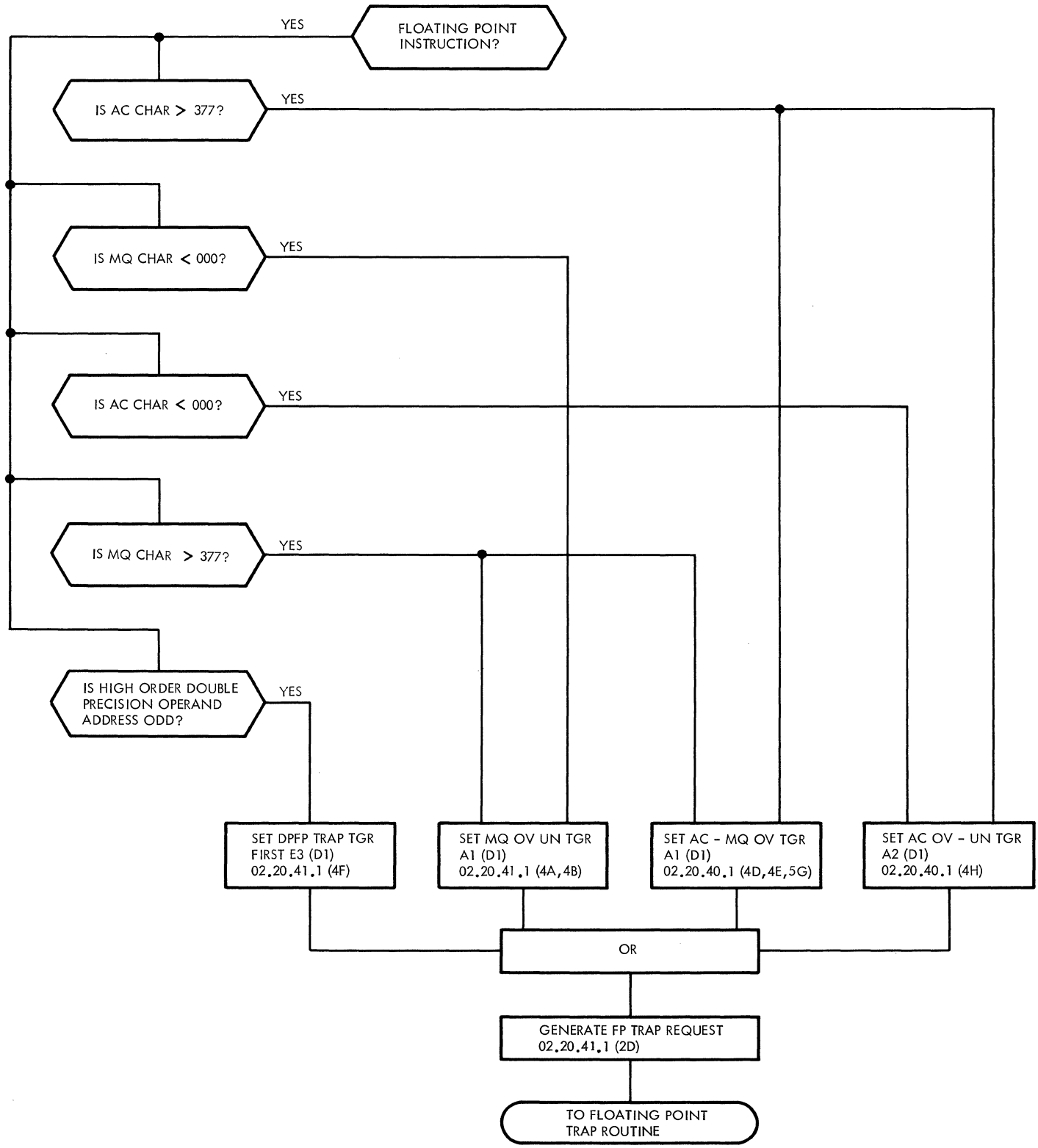


FIGURE 62. FLOATING-POINT TRAP REQUEST

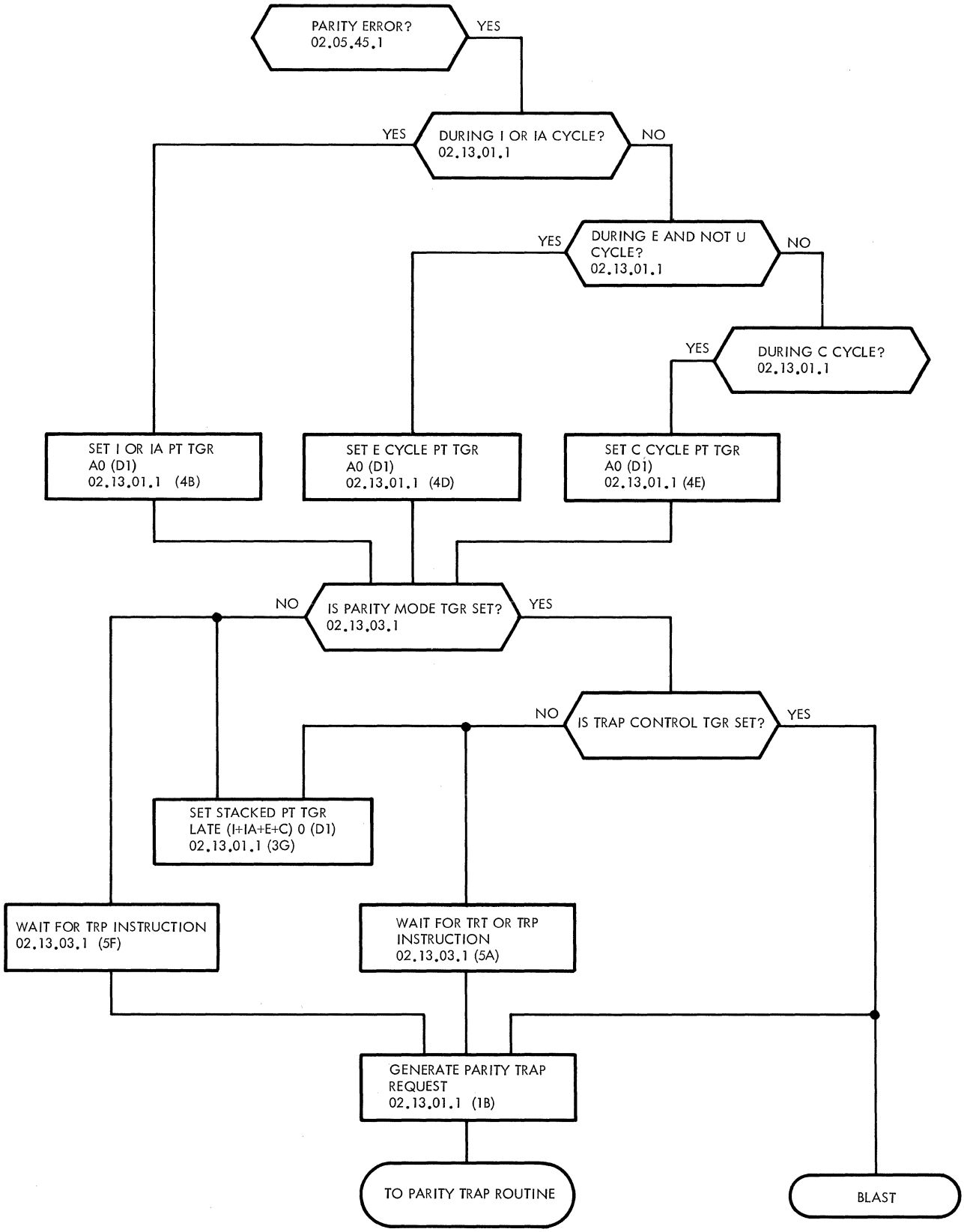


FIGURE 63. PARITY TRAP REQUEST

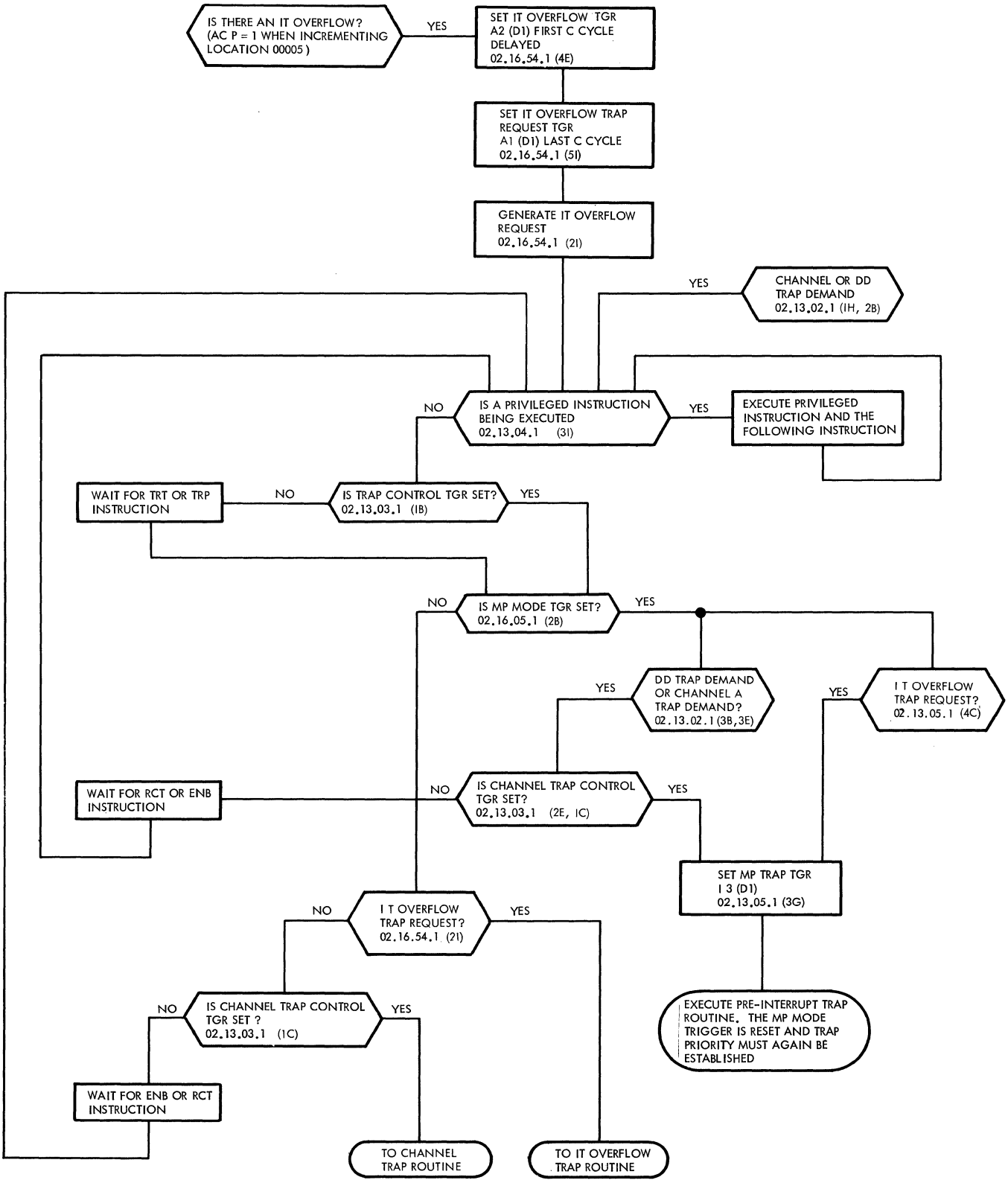


FIGURE 64. INTERVAL TIMER OVERFLOW AND CHANNEL TRAP REQUESTS

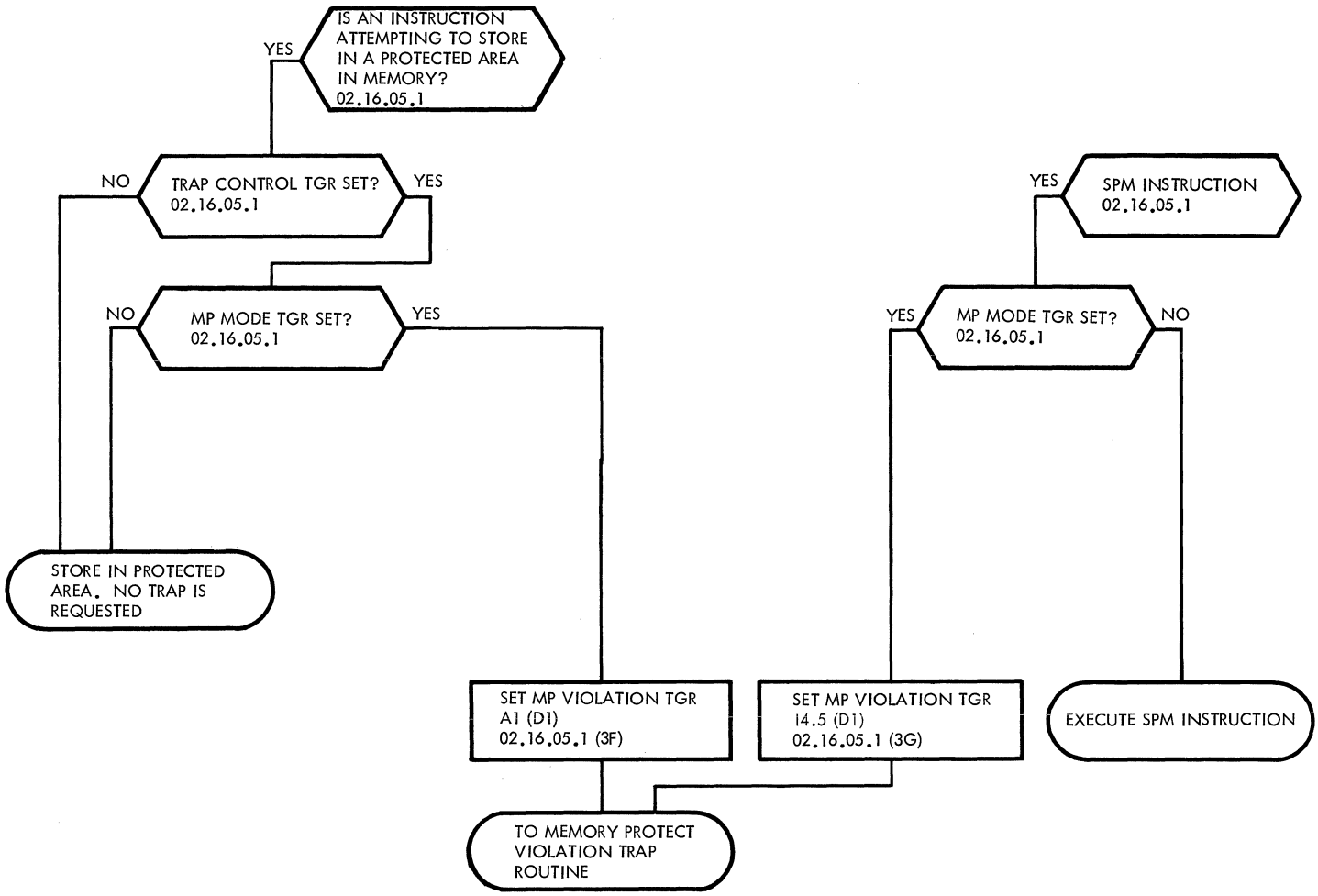


FIGURE 65. MEMORY PROTECT TRAP REQUEST

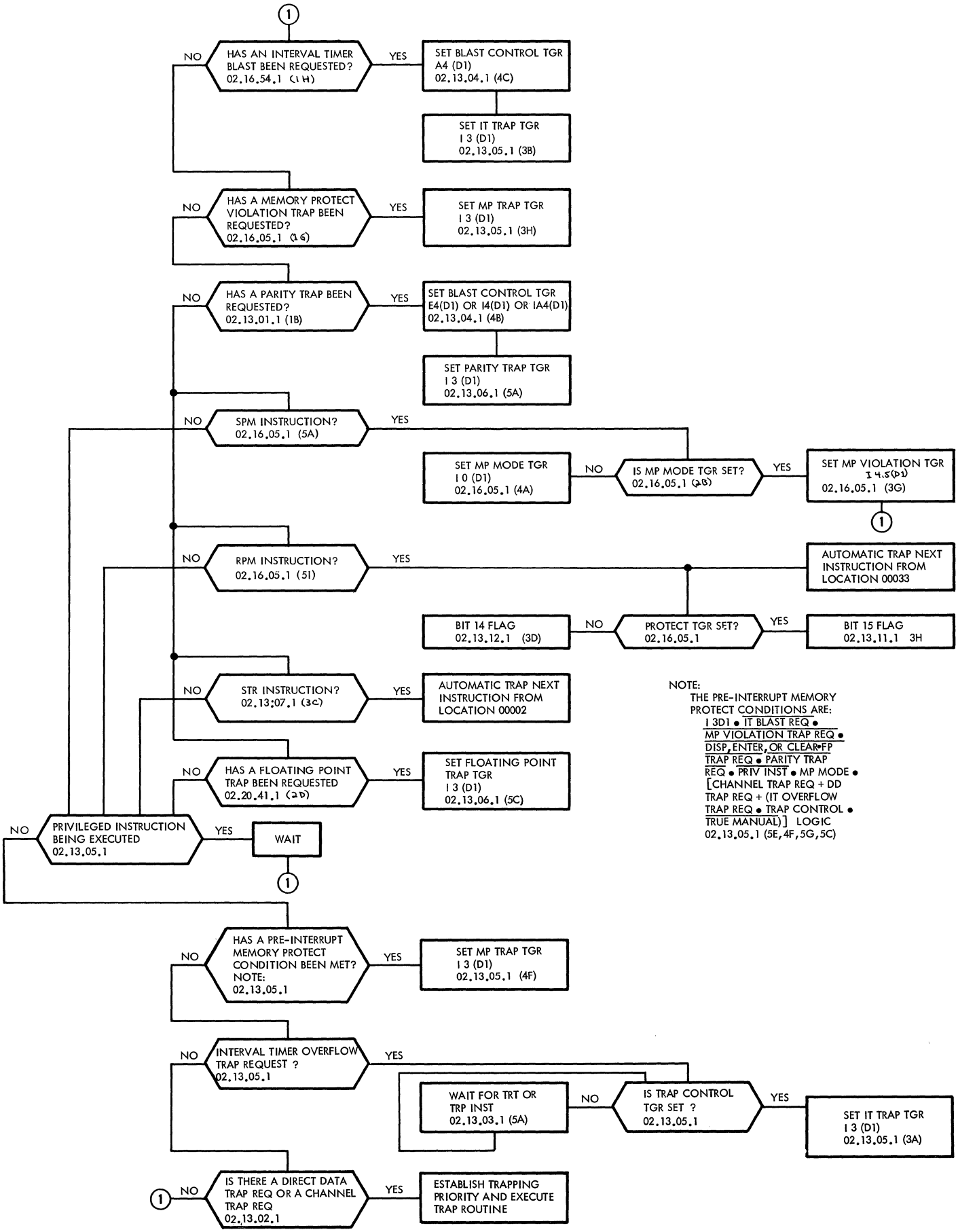


FIGURE 66. TRAPPING PRIORITY SCHEME

Instruction traps have equal priority. This presents no problem since only one instruction can be executed at a time. Note that the privileged instruction is important in allowing certain traps to be executed. The privileged instruction trigger inhibits three types of traps for one additional instruction (if not a privileged instruction). The traps that must wait are:

1. Pre-interrupt memory protect.
2. Interval timer overflow.
3. All channel traps and DD traps.

When the privileged instruction trigger is reset, the trap executed is determined by priority (Figure 66).

Individual Traps

Once the trap is enabled, the trap request generated, and priority established, the trap routine is executed. Each trap is flow-diagrammed in this program to show the trap routine sequence. After entering the trap routine, all other trap requests are inhibited from setting their own trap triggers (regardless of priority) until the trap routine is completed.

The flow diagrams show the logic action performed and the logical decisions. The timing, logic, and location of the logic block are included to aid in locating the action in logic.

The flow diagrams are in sequential order according to priority (Figures 67 through 74):

1. Interval-timer-blast trap (Figure 67).
2. Memory-protect-violation trap (Figure 68).
3. Parity trap (Figure 69).
4. SPM instruction trap (Figure 70).
5. RPM instruction trap (Figure 71).
6. STR instruction trap (Figure 72).
7. Floating-point trap (Figure 73).
8. Pre-interrupt-memory-protect trap and interval-timer-overflow trap (Figure 74).

The channel and DD trap requests show how the trap routine is entered. For the channel trapping scheme, refer to the theory of operation of the channels.

The interval timer blast request, interval timer overflow request, channel trap request, or DD trap request sets the start trigger (if the machine is in automatic status) at A5 time. This is shown as a machine restart in Figures 67 and 74.

Channel A Trapping

There are eight possible channel A traps:

1. Unit record interrupt.
2. Teleprocessing interrupt.
3. 1401 attention.
4. Unusual end.
5. Word parity.
6. End of file.
7. Redundancy check.
8. Operation complete.

A trap can be effected only if it has been enabled either by a TRT or TRP instruction which sets the trap control latch or by an Enable (ENB) instruction which sets up a pseudo mask register to designate which traps are allowed to take place.

Figure 75 shows how the Enable instruction sets up trap enables. Note that this instruction always sets the channel trap control latch. The mask register consists of four triggers, each of which will be set if the appropriate bit of the ENB instruction's effective address is a 1. A mask register trigger must be set in order for any of the traps associated with the trigger to be effected.

Also note, in Figure 75, that the ENB instruction resets the mask triggers before it sets any of them. This resetting action eradicates the effects of any previous ENB instructions.

Unit Record Interrupt

Figure 76 is a flow chart of the unit record interrupt trap. Whenever the 1414-III or IV card reader, paper tape reader, card punch, or printer goes from a busy condition to a not-busy condition, the unit-record-interrupt trigger is set. If the mask register trigger, in this case the enable-buffer-interrupt trigger, is also set, a unit record interrupt trap is effected. The UR INT TRAP stores a 1 in bit position 8 of memory location 00012 and stores the contents of the instruction counter in bit positions 21-35. Memory location 00013 is then brought out and will contain an unconditional branch to the trap routine.

Teleprocessing Interrupt

Figure 77 is a flow chart of the teleprocessing interrupt trap. Whenever the 1414-IV or V has a buffer inquiry or outquiry, the buffer-attention trigger is set. If the mask register trigger, in this case the enable-attention trigger, is also set, a teleprocessing interrupt trap is effected. The teleprocessing trap stores a 1 in bit position 9 of memory location 00012 and stores the contents of the instruction counter in bit positions 21-35. Memory location 00013 is then brought out and will contain an unconditional branch to the trap routine.

1401 Attention

Figure 78 is a flow chart of the 1401 attention trap. An I-O 6 select level from the 1401 sets the 1401 attention trigger. If the mask register trigger, in this case the enable-attention trigger, is also set, a 1401 attention trap is effected. The 1401 attention

trap stores a 1 in bit position 10 of memory location 00012 and stores the contents of the instruction counter in bit positions 21-35. Memory location 00013 is then brought out and will contain an unconditional branch to the trap routine.

Unusual End

Figure 79 is a flow chart of the unusual-end trap. Whenever the total number of characters read from or written onto tape is not a multiple of 6, the unusual-end-trap trigger is set. If the mask register trigger, in this case the enable-end trigger, is also set, an unusual-end trap is effected. The unusual-end trap stores a 1 in bit position 12 of memory location 00012 and stores the contents of the instruction counter in bit positions 21-35. Memory location 00013 is then brought out and will contain an unconditional branch to the trap routine.

Word Parity

Figure 80 is a flow chart of the word parity trap. Whenever there is a word parity error during a U cycle, the word-parity trigger is set. If the mask register trigger is also set, a word parity trap is effected. In this case, there are two mask register triggers, either of which will cause the trap: the enable-parity or the enable-end trigger. If the enable-parity trigger is set, a 1 is stored in bit position 14 of memory location 00012. If the enable-end trigger is set, a 1 is stored in bit position 17 of memory location 00012. If both triggers were set, bits 14 and 17 would each set to a 1. Also, the instruction counter contents are stored in bit positions 21-35 of location 00012. Memory location 00013 is then brought out and will contain an unconditional branch to the trap routine.

End of File

Figure 81 is a flow chart of the end-of-file trap. Whenever there is an end-of-file signal from I-O, the end-of-file trigger is set. If the mask register trigger, in this case the enable-end trigger, is also set, an EOF trap is effected. The trap stores a 1 in bit position 15 of memory location 00012 and stores the instruction counter contents in bit positions 21-35. Memory location 00013 is then brought out and will contain an unconditional branch to the trap routine.

Redundancy Check

Figure 82 is a flow chart of the redundancy check trap. Whenever there is a byte check bit error, the redundancy-check-indicator trigger is set. If the mask register trigger, in this case the enable-parity trigger, is also set, a redundancy check trap is effected. The redundancy-check trap stores a 1 in bit position 16 of memory location 00012 and stores the contents of the instruction counter in bit positions 21-35. Memory location 00013 is then brought out and will contain an unconditional branch to the trap routine.

Operation Complete

Figure 83 is a flow chart of the operation complete trap. Whenever an operation ends (signified by an EOR), the end-trap trigger is set. If the mask register trigger, in this case the enable-end trigger, is also set, an operation complete trap is effected. The trap stores a 1 in bit position 17 of memory location 00012 and stores the instruction counter contents in bit positions 21-35. Memory location 00013 is then brought out and will contain an unconditional branch to the trap routine.

SUMMARY

The table in Figure 84 summarizes each of the possible traps incorporated in the 7040-7044 equipment.

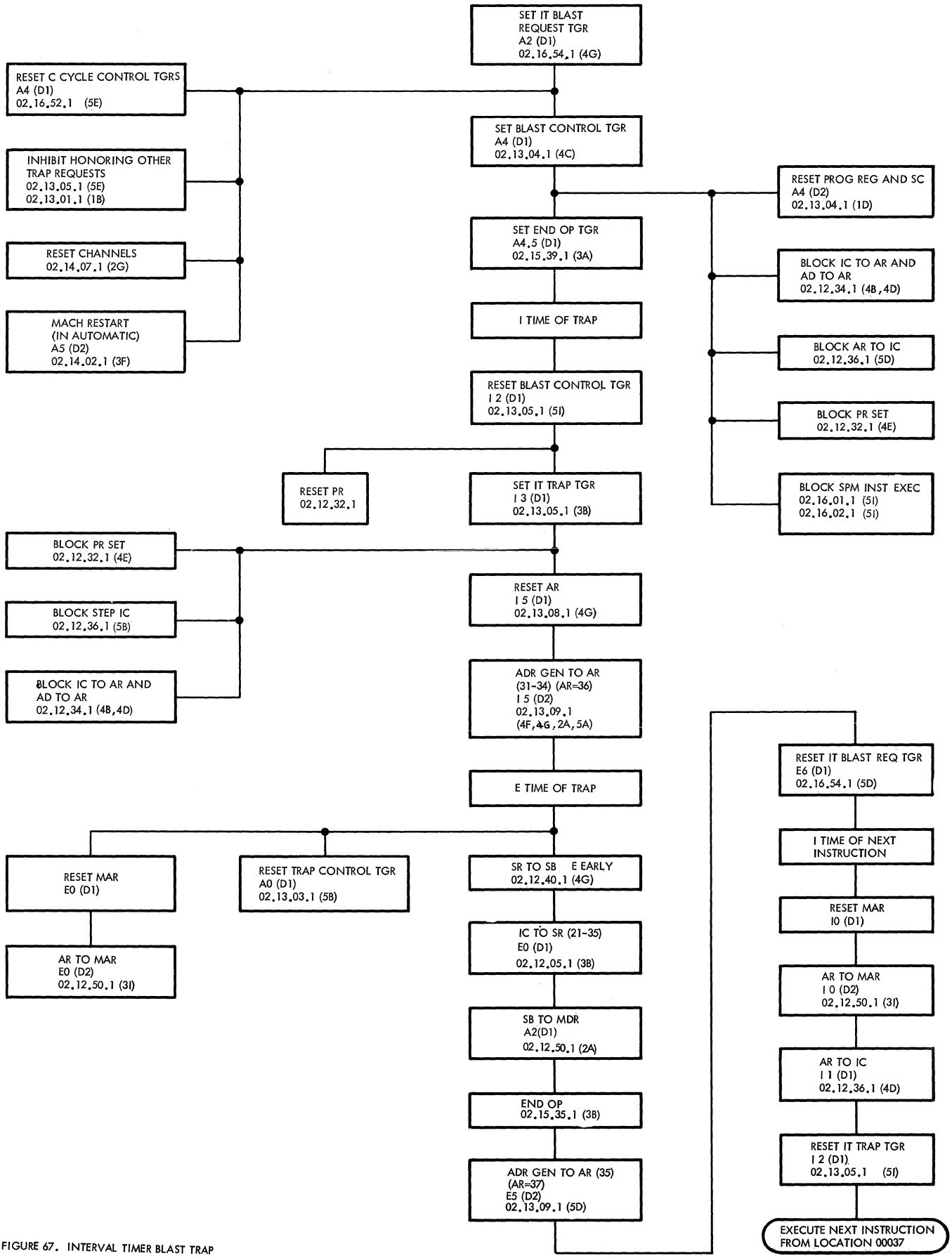


FIGURE 67. INTERVAL TIMER BLAST TRAP

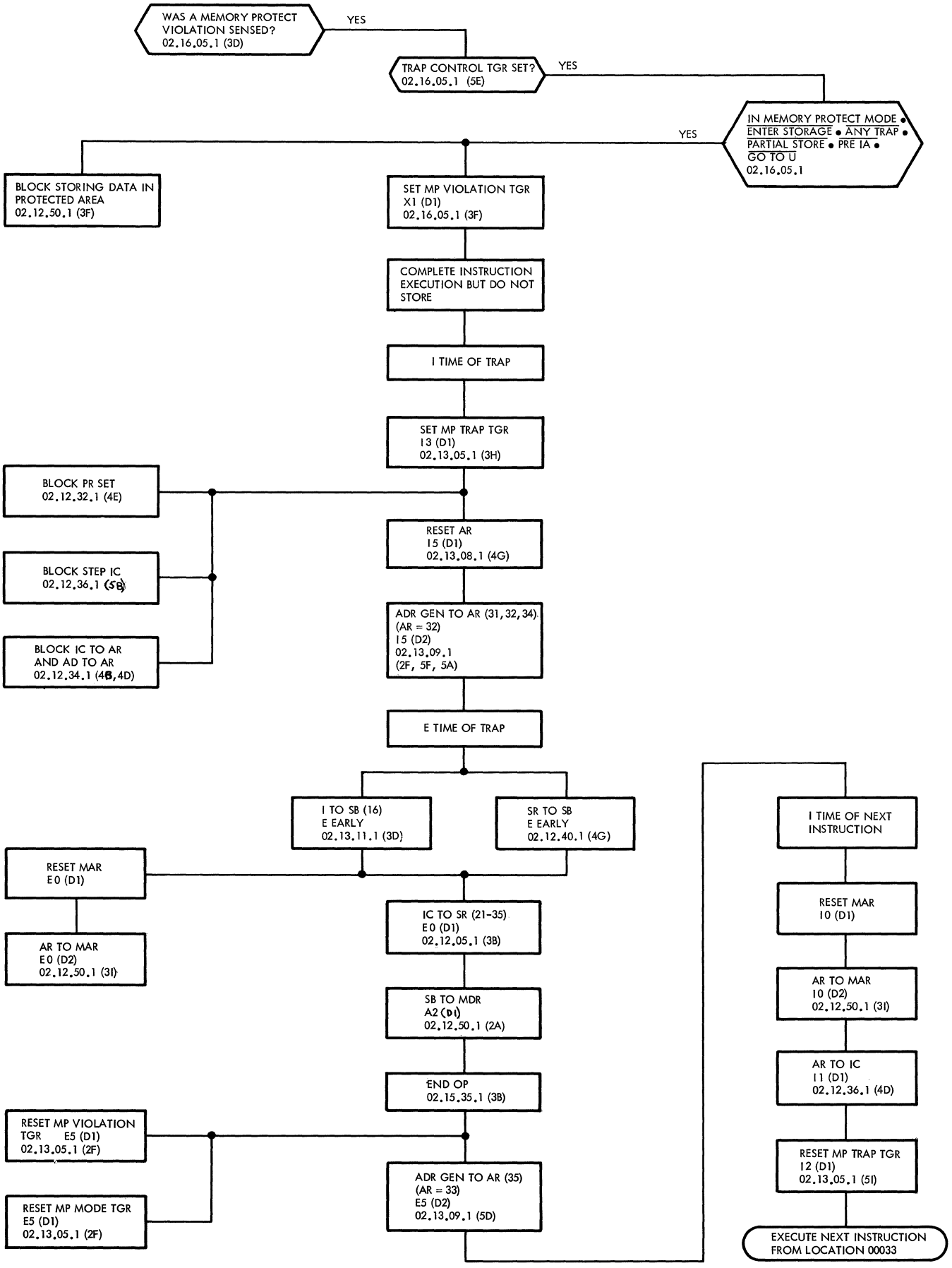


FIGURE 68. MEMORY PROTECT VIOLATION TRAP

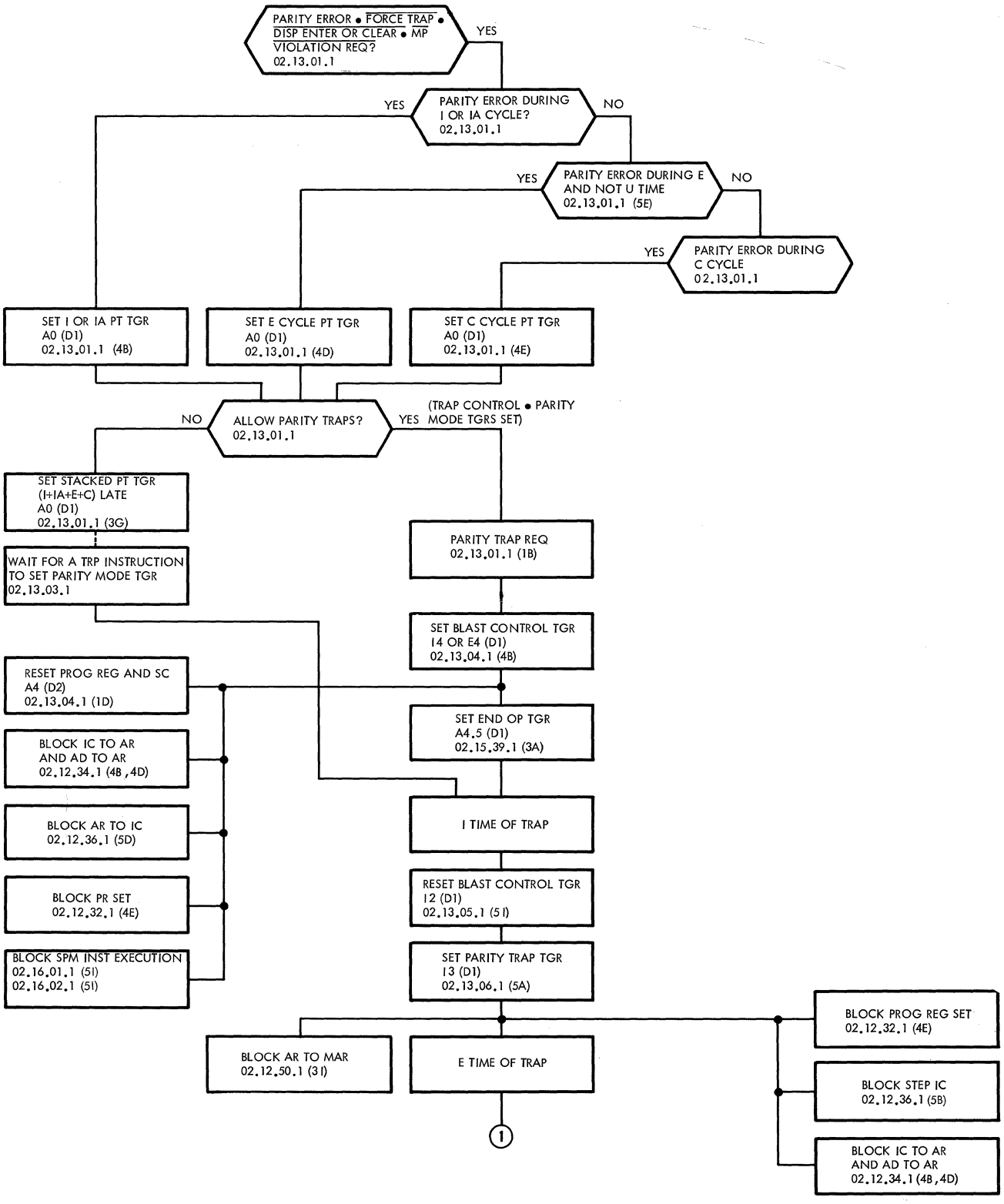


FIGURE 69. PARITY TRAP (SHEET 1 OF 2)

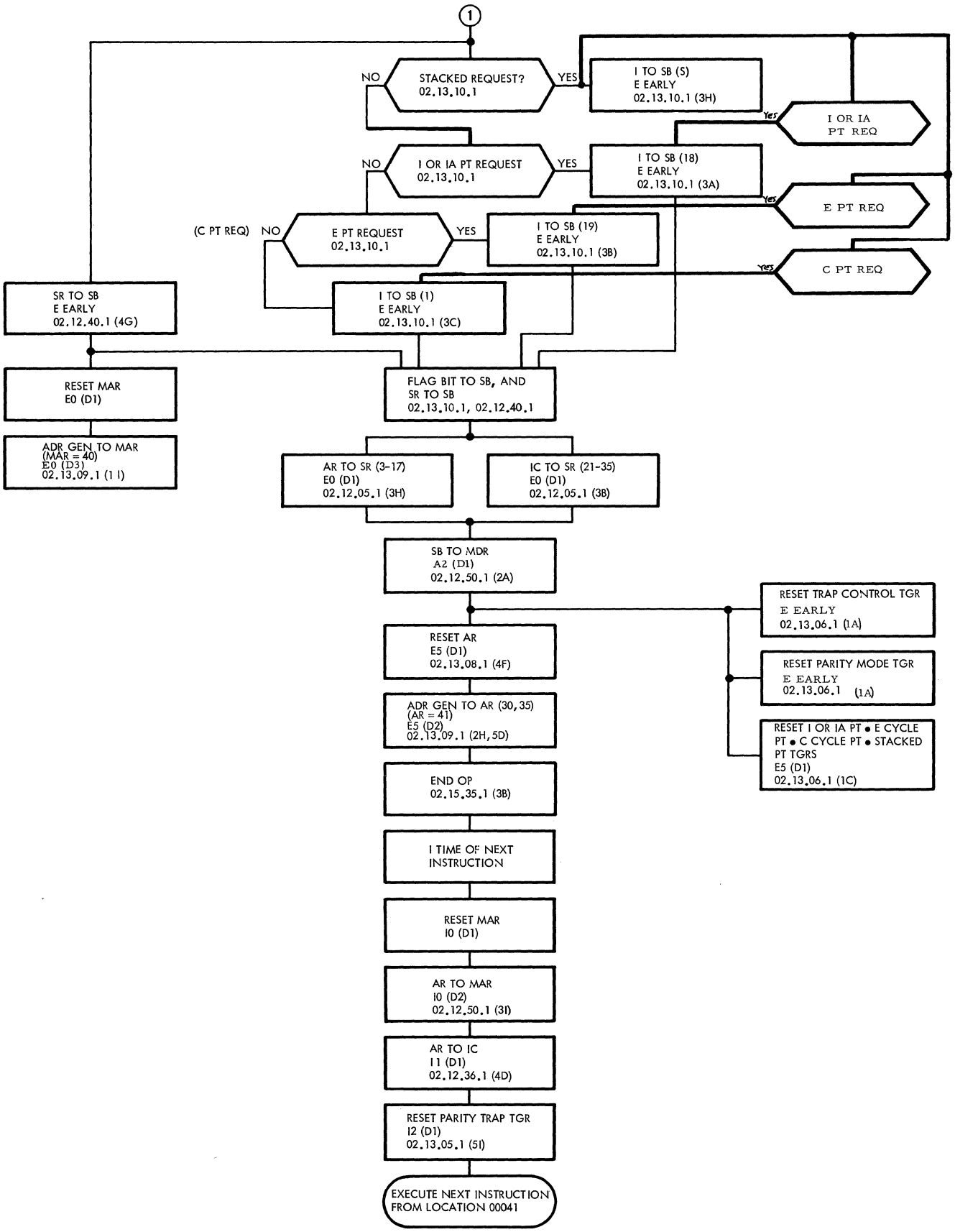


FIGURE 69. PARITY TRAP(Sheet 2 of 2)

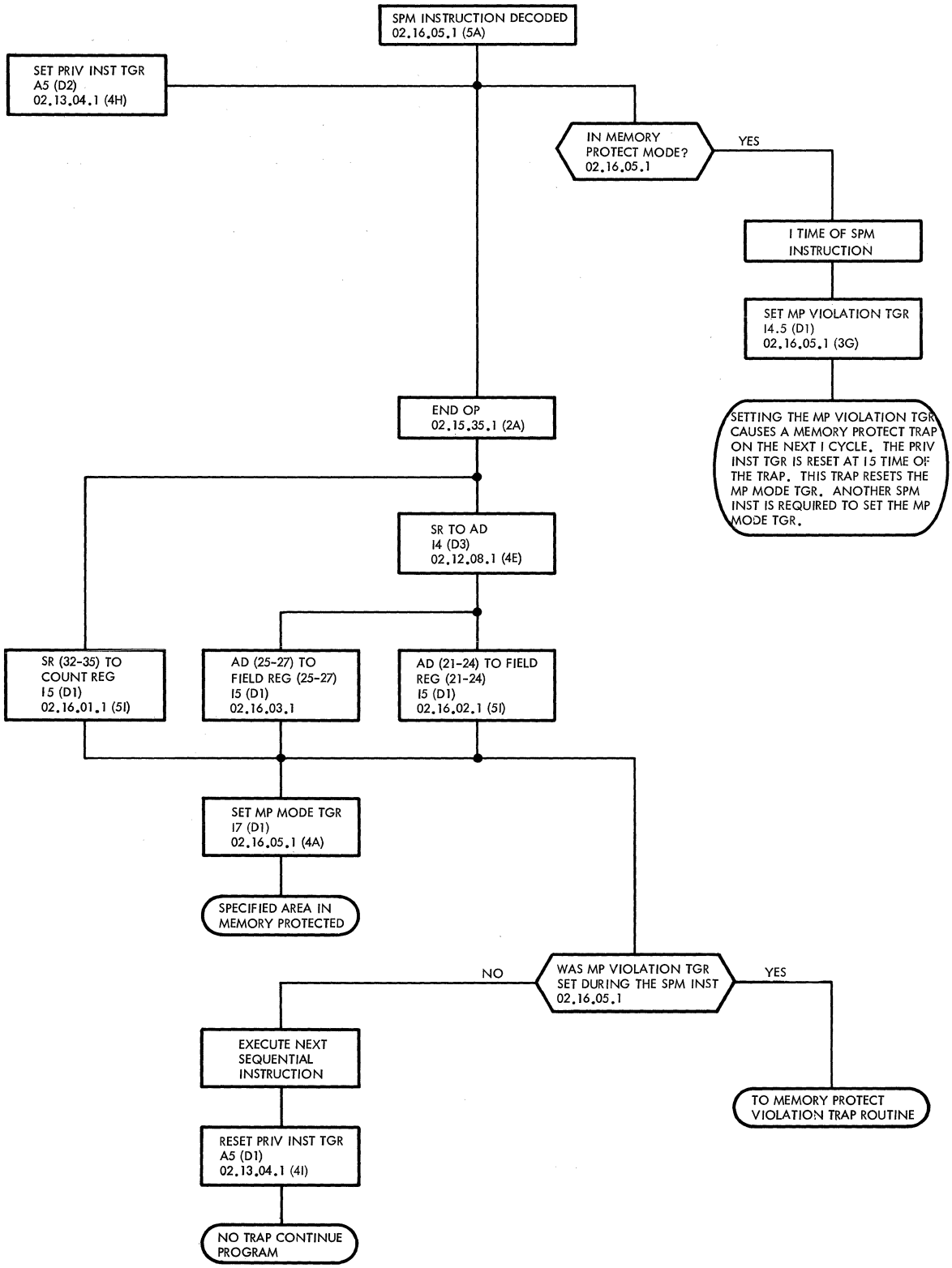


FIGURE 70. SPM INSTRUCTION TRAP

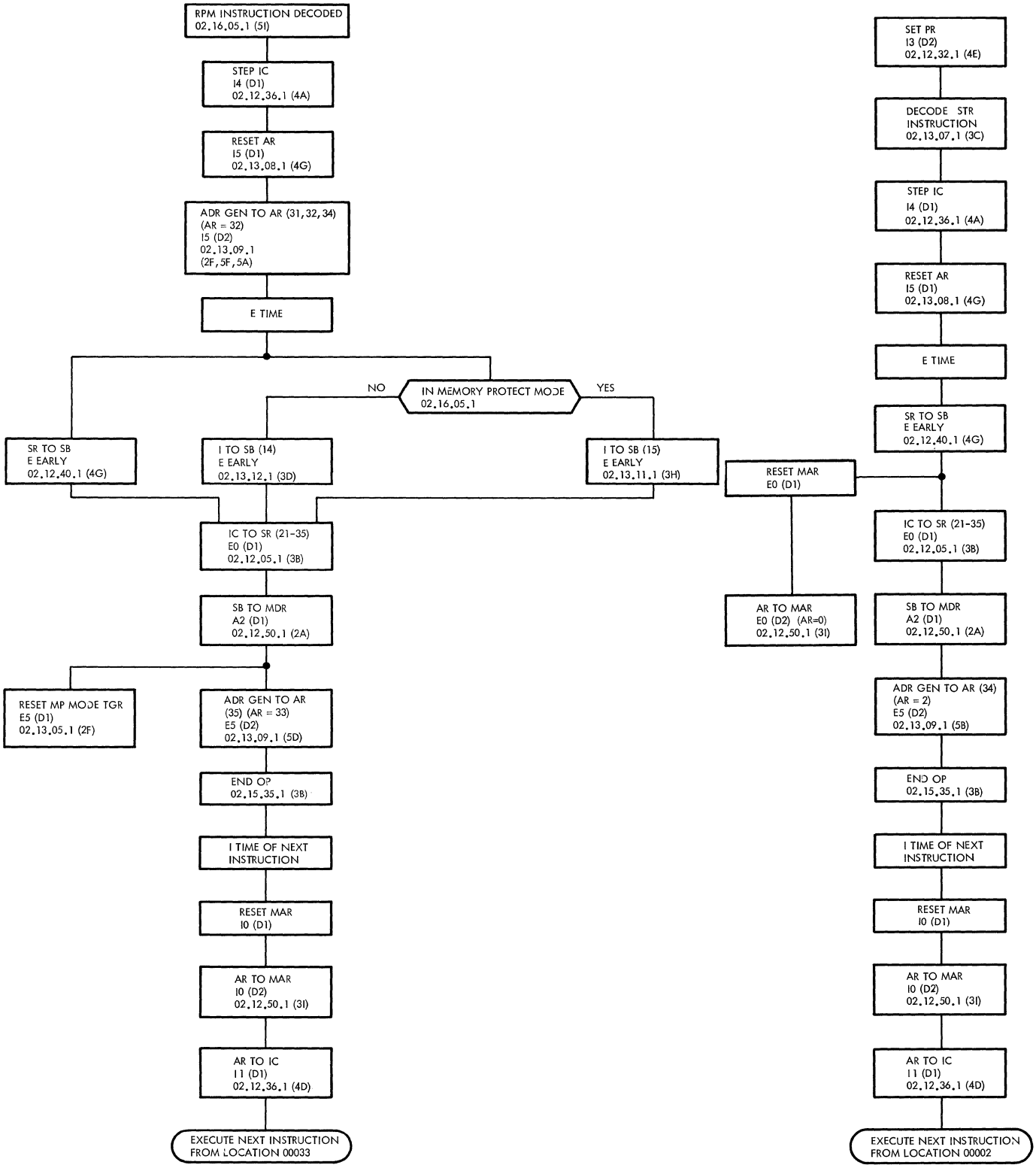


FIGURE 71. RPM INSTRUCTION TRAP

FIGURE 72. STR INSTRUCTION TRAP

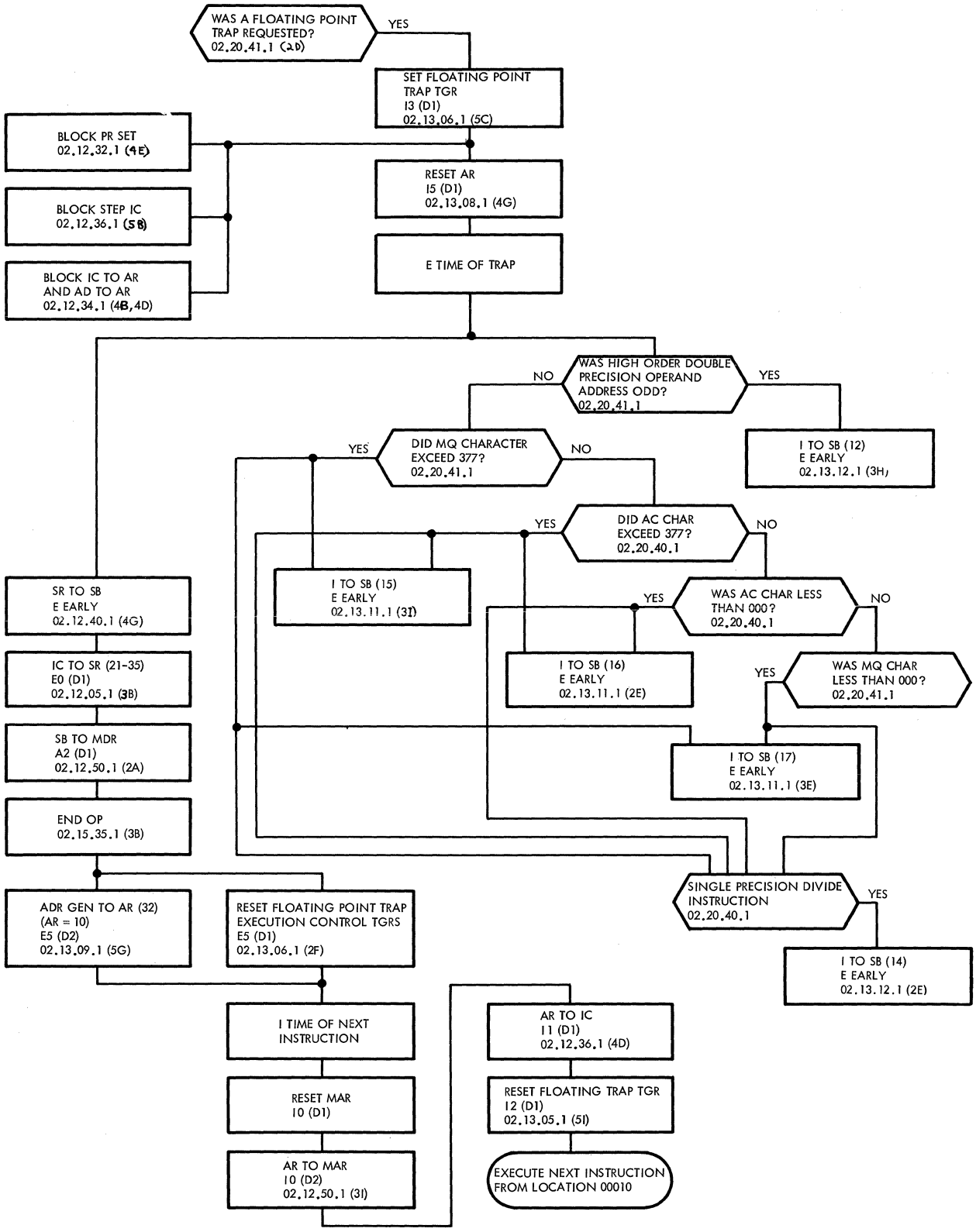


FIGURE 73. FLOATING POINT TRAP

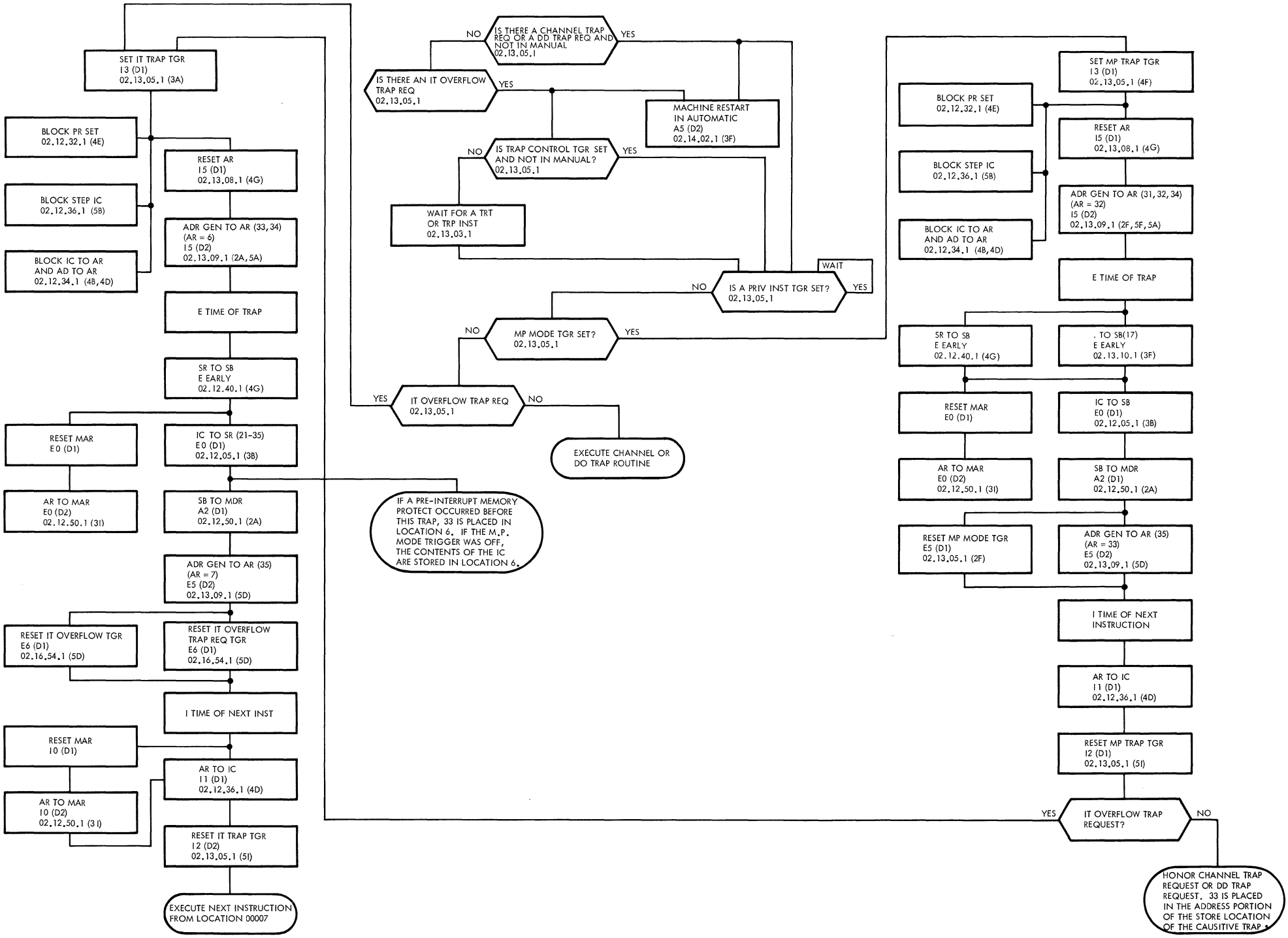
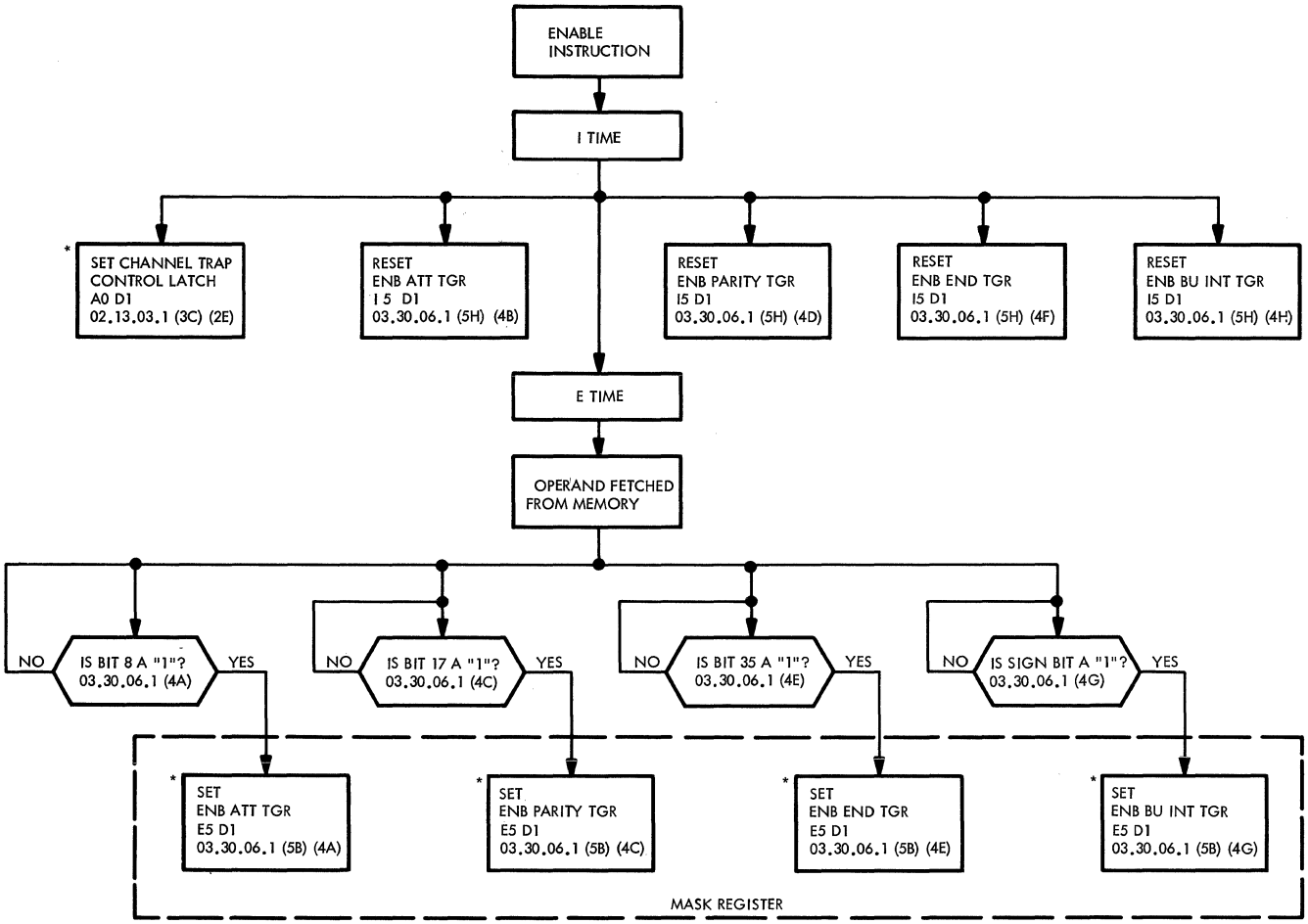


FIGURE 74. PRE-INTERRUPT AND INTERVAL TIMER OVERFLOW TRAPS



*THESE 5 LATCHES REMAIN SET AFTER TERMINATION OF THE ENABLE INSTRUCTION:

CHANNEL TRAP CONTROL LATCH (ALWAYS SET BY ENABLE INSTR)
ATTENTION TRIGGER (SET IF ENB OPERAND 8-BIT WAS A ONE)
PARITY TRIGGER (SET IF ENB OPERAND 17-BIT WAS A ONE)
END TRIGGER (SET IF ENB OPERAND 35-BIT WAS A ONE)
BUFFER INTERRUPT TRIGGER (SET IF ENB OPERAND SIGN BIT WAS A ONE)

FIGURE 75. TRAP ENABLES

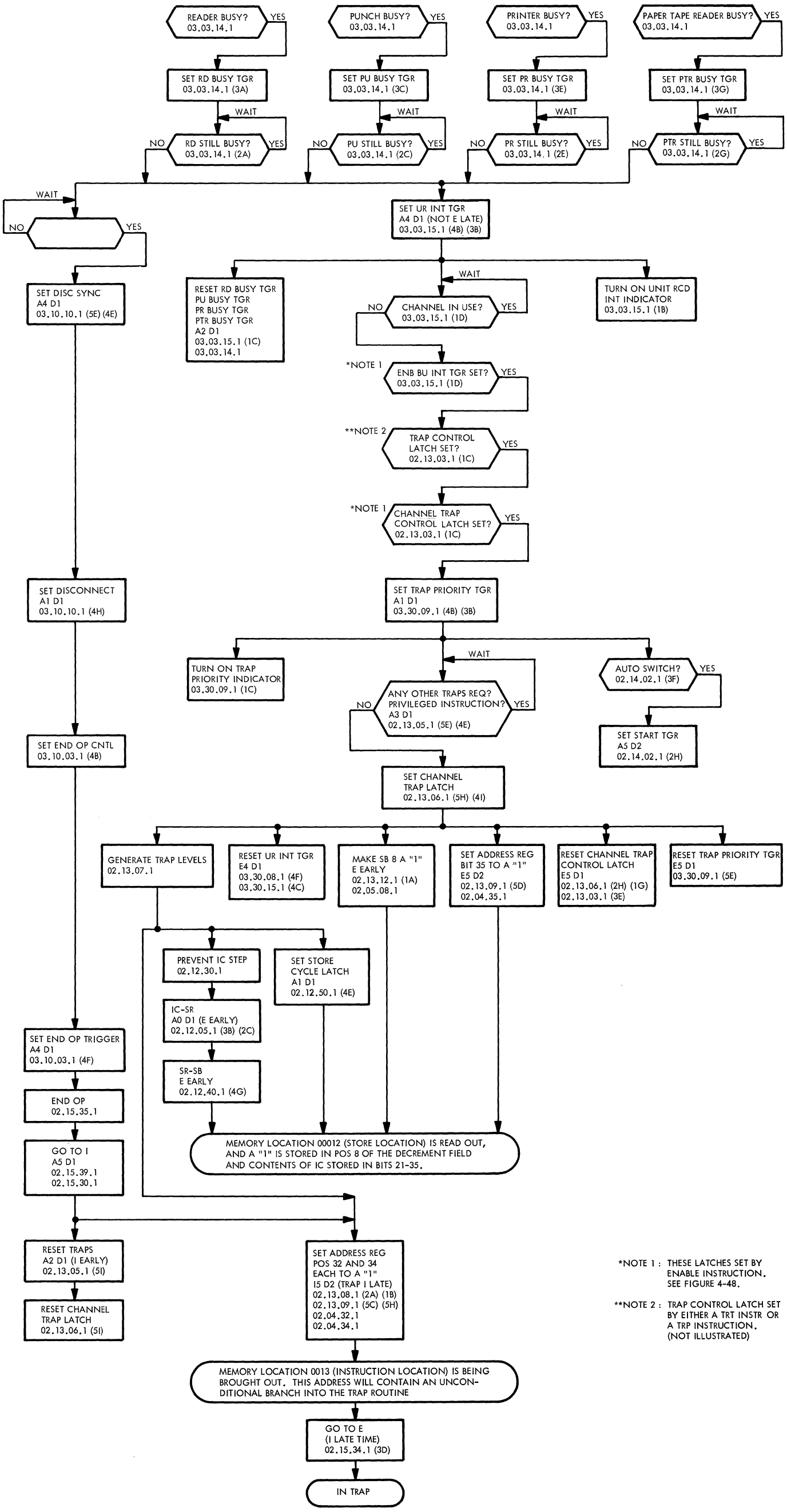


FIGURE 76. UNIT RECORD INTERRUPT (TRAP)

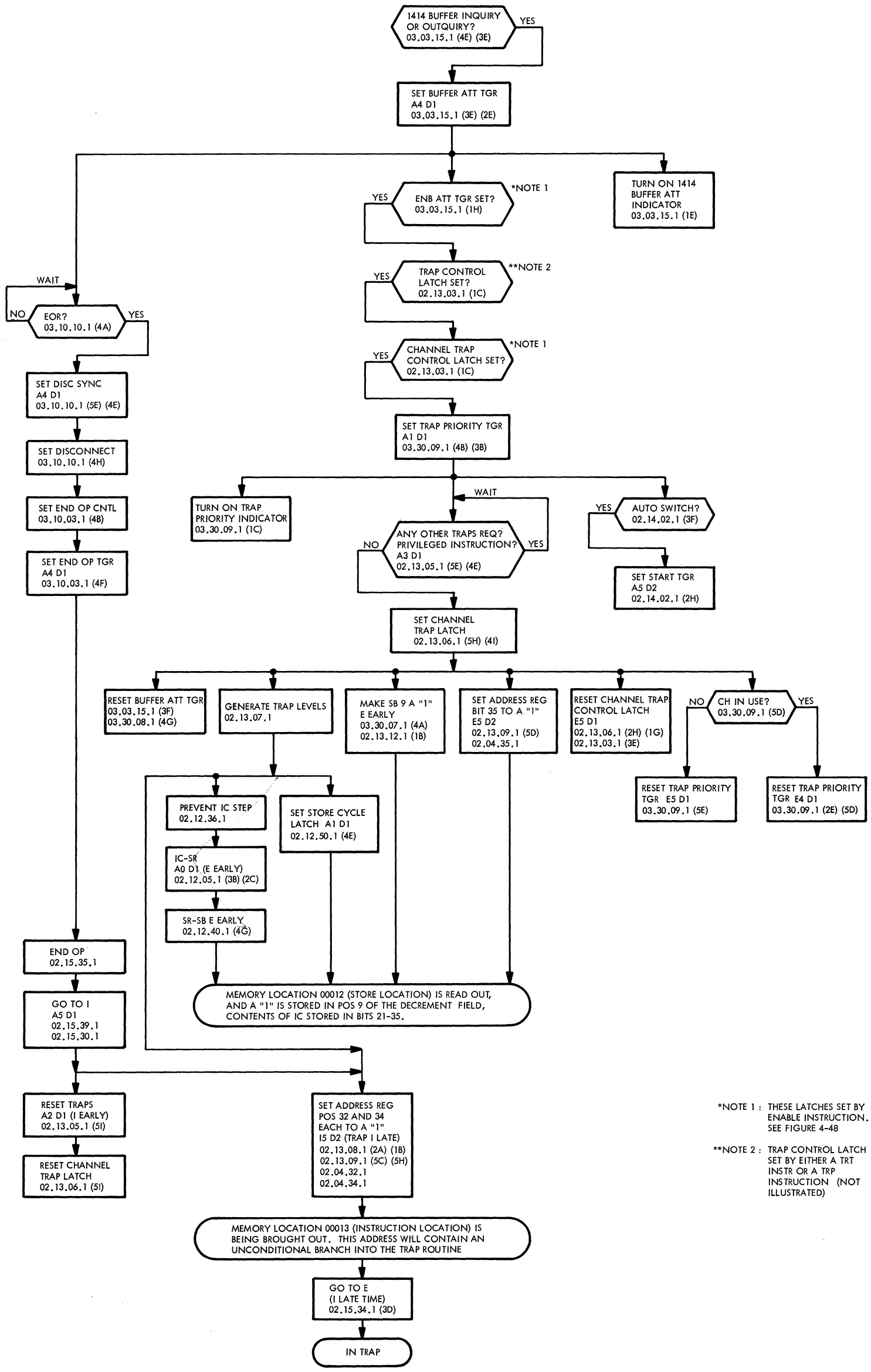


FIGURE 77. TELEPROCESSING INTERRUPT (TRAP)

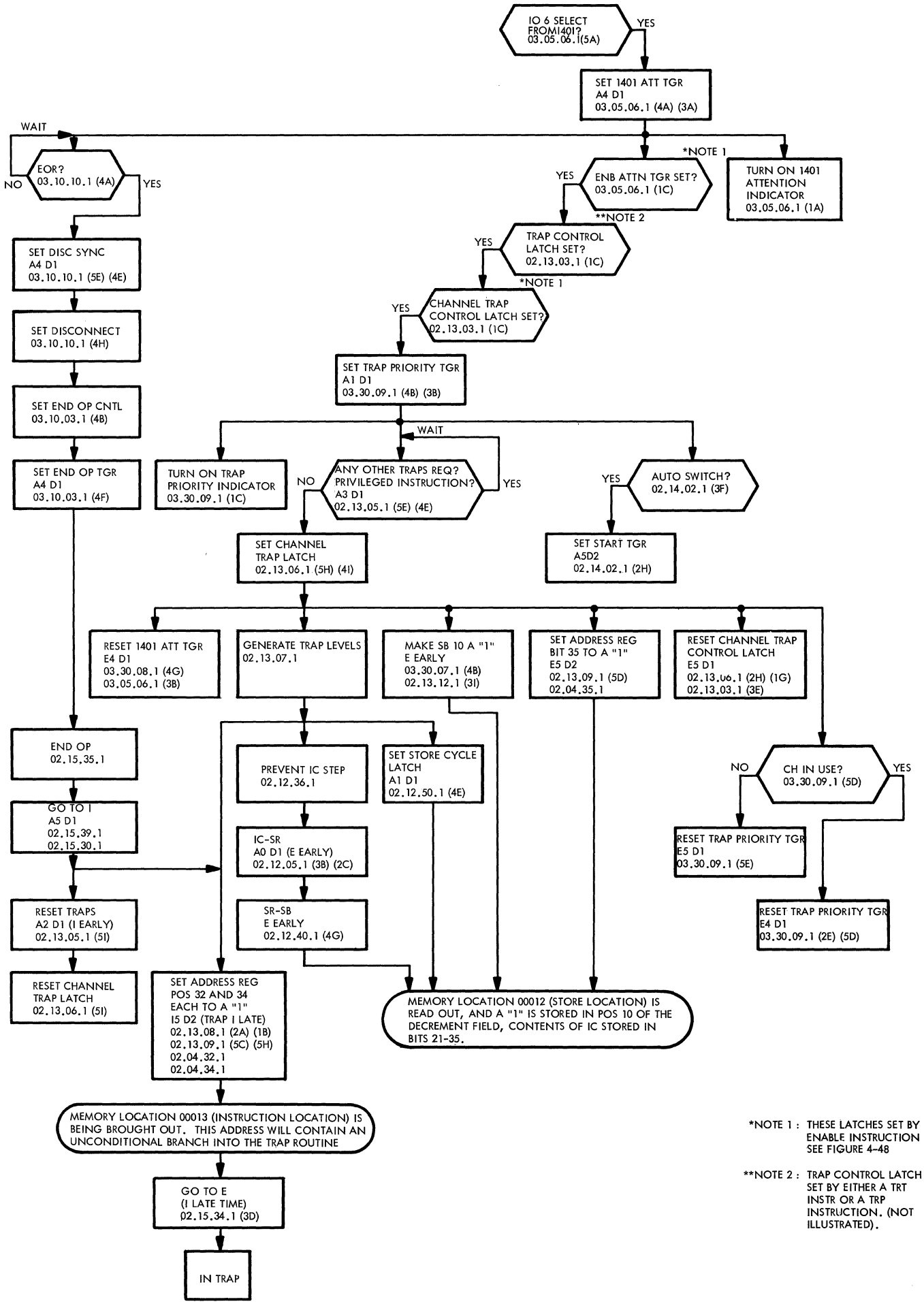


FIGURE 78. 1401 ATTENTION (TRAP)

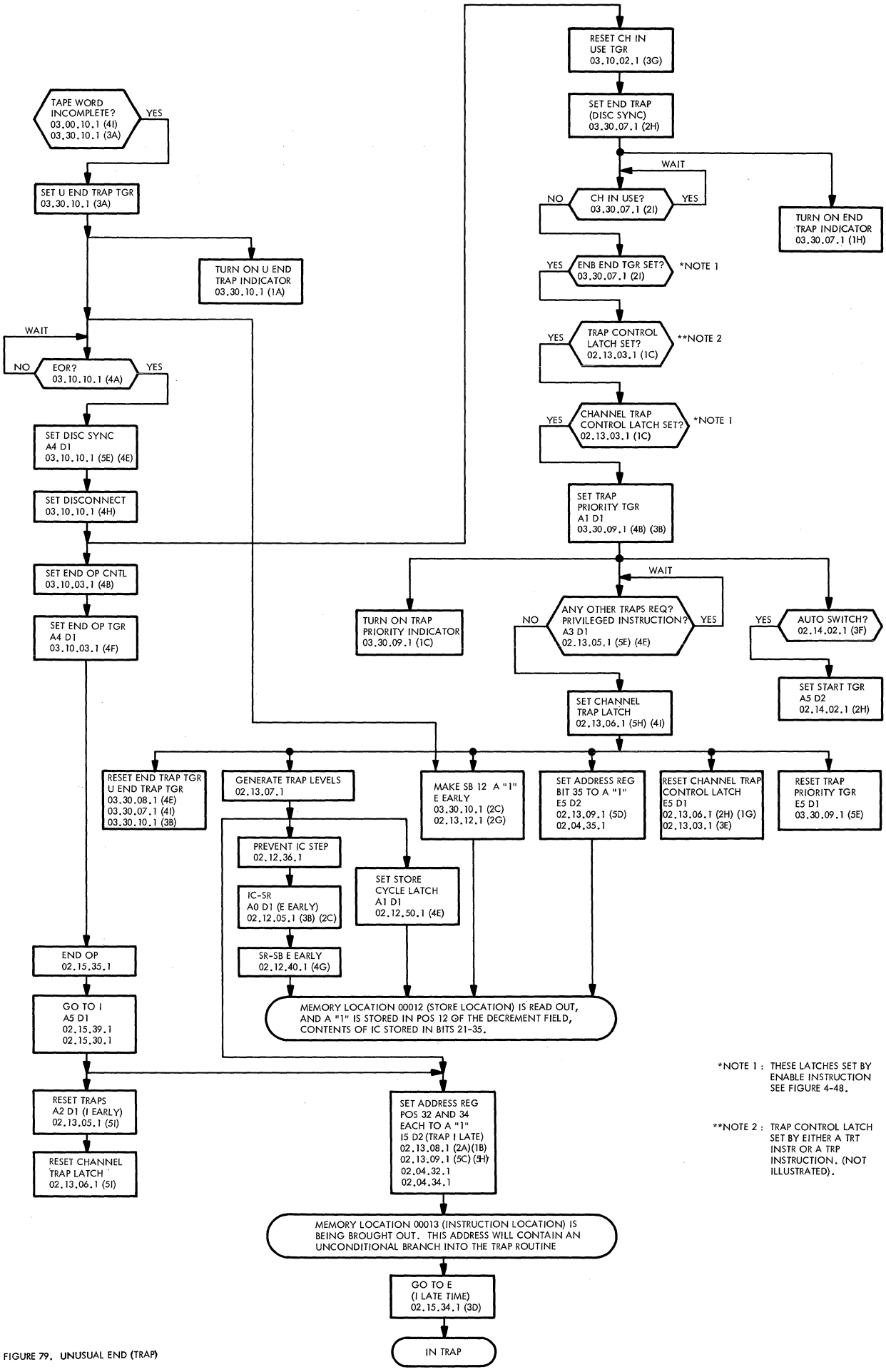


FIGURE 79. UNUSUAL END (TRAP)

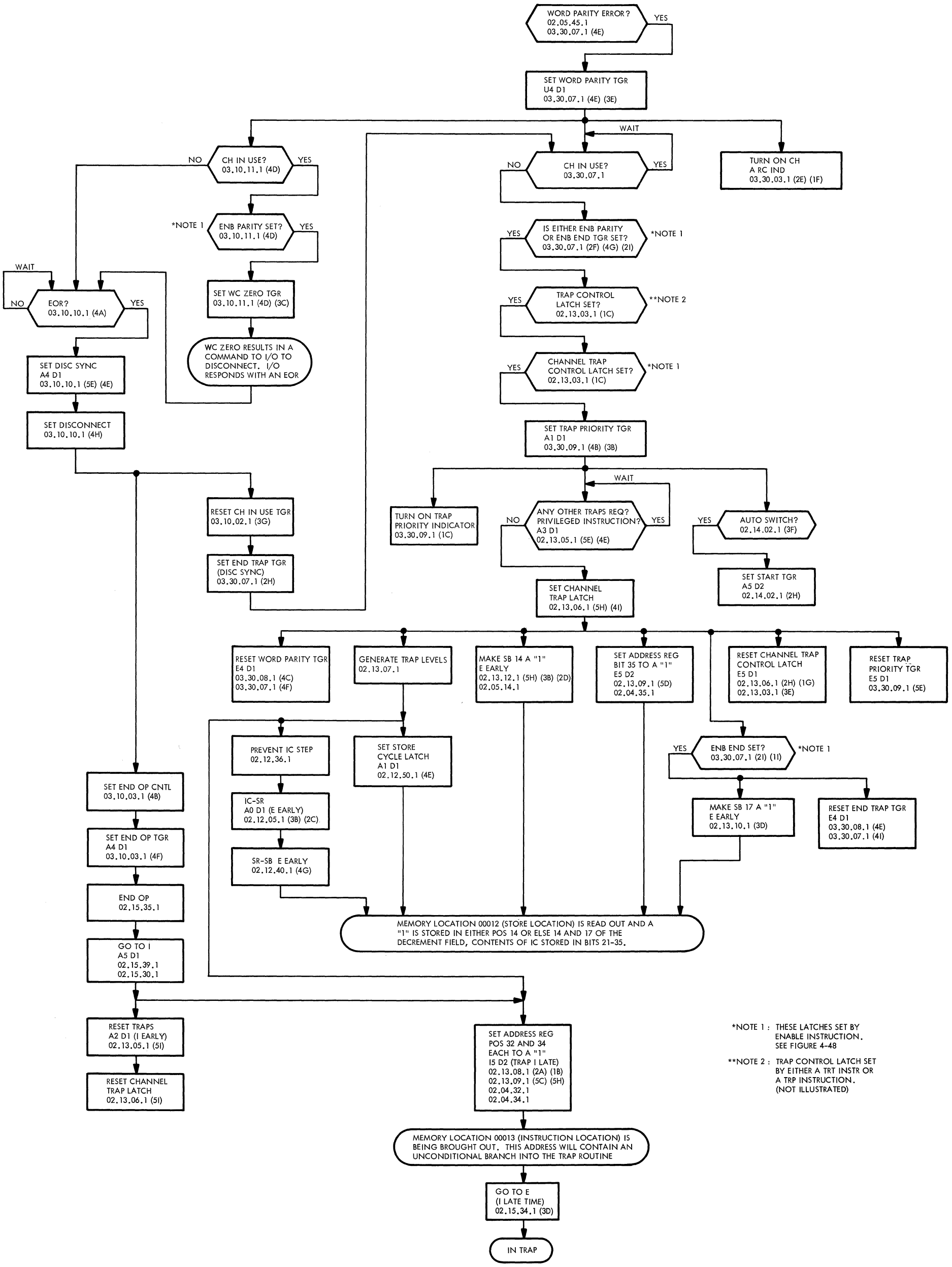


FIGURE 80. WORD PARITY (TRAP)

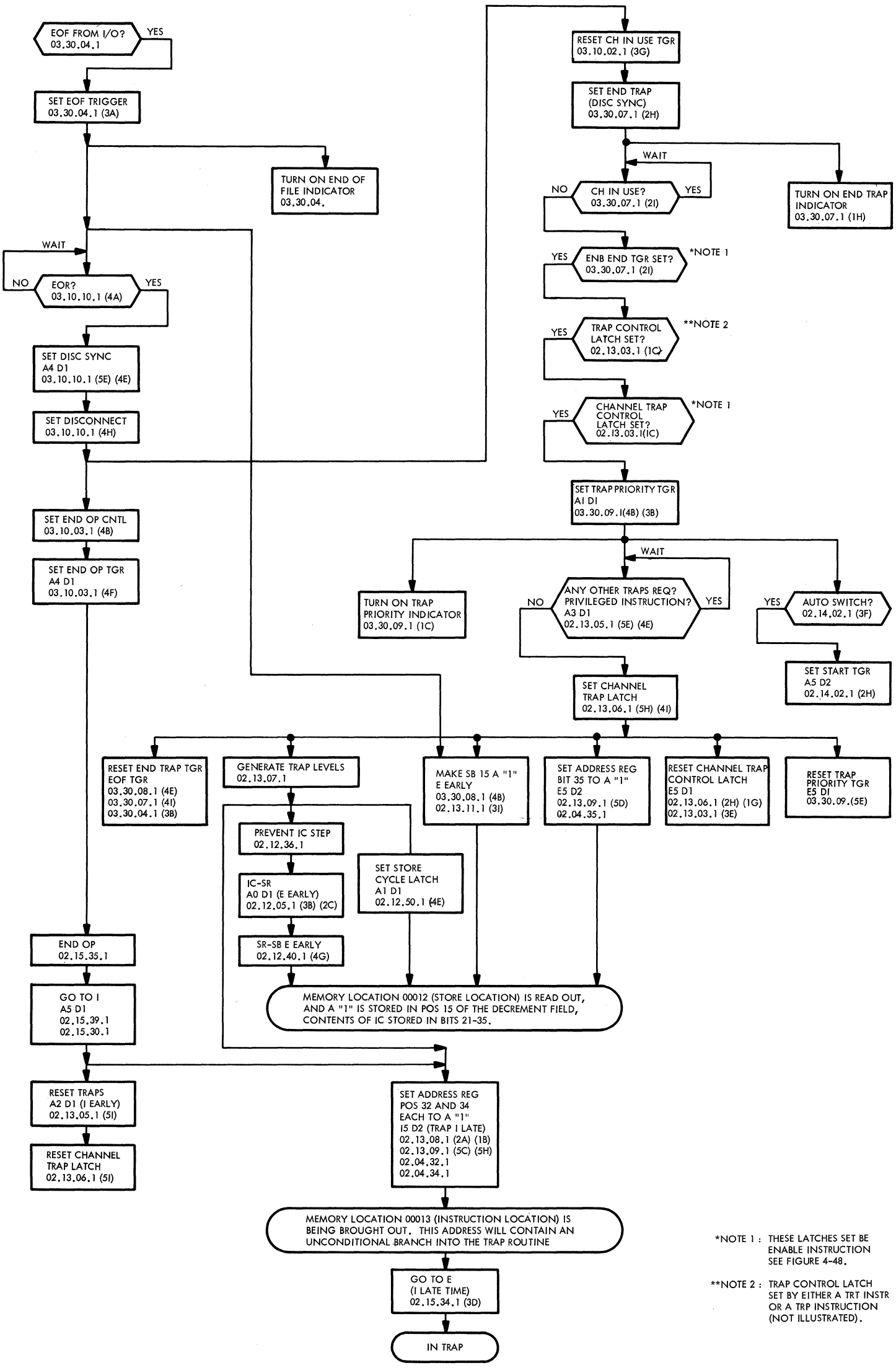


FIGURE 81. END OF FILE (TRAP)

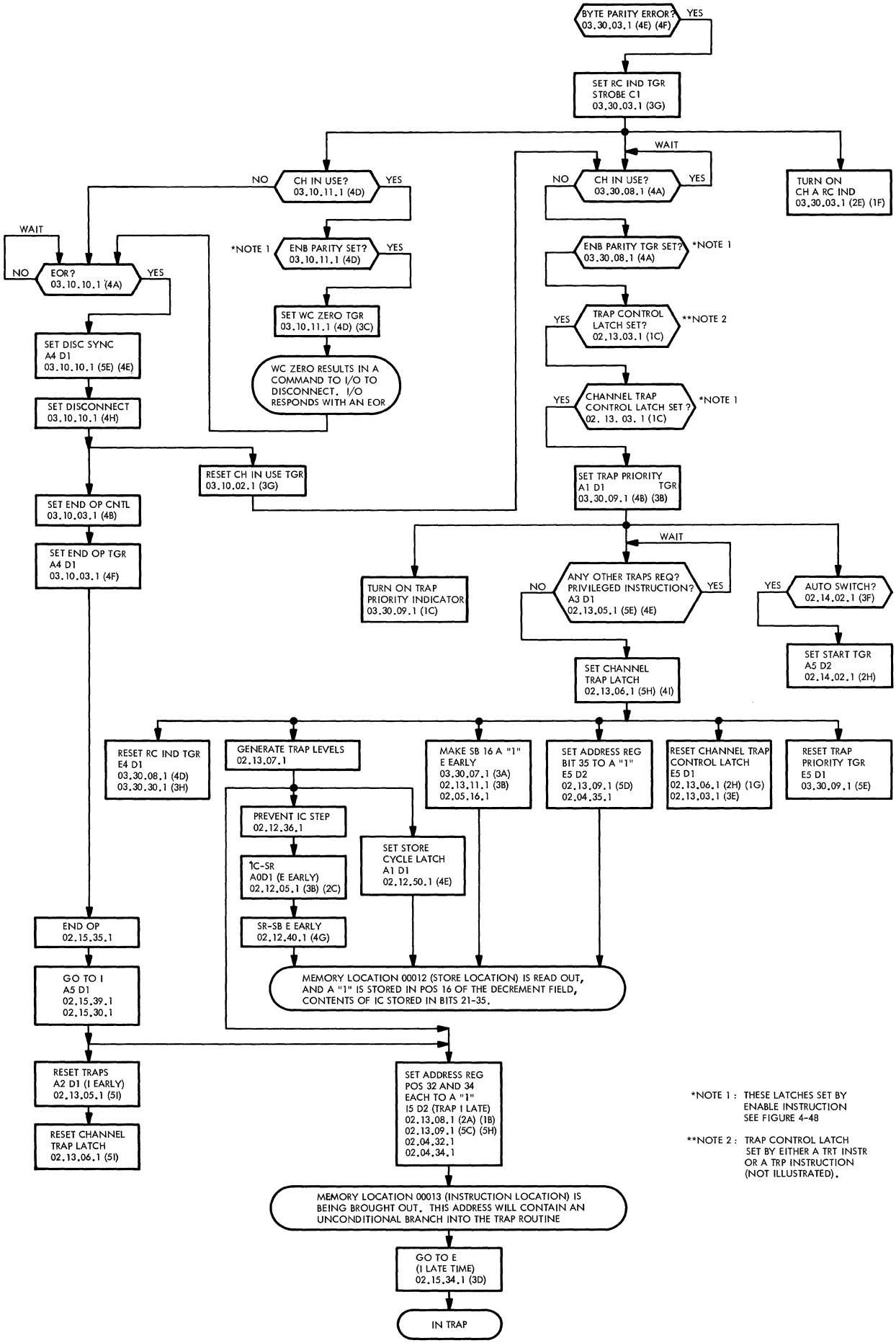


FIGURE 82. REDUNDANCY CHECK (TRAP)

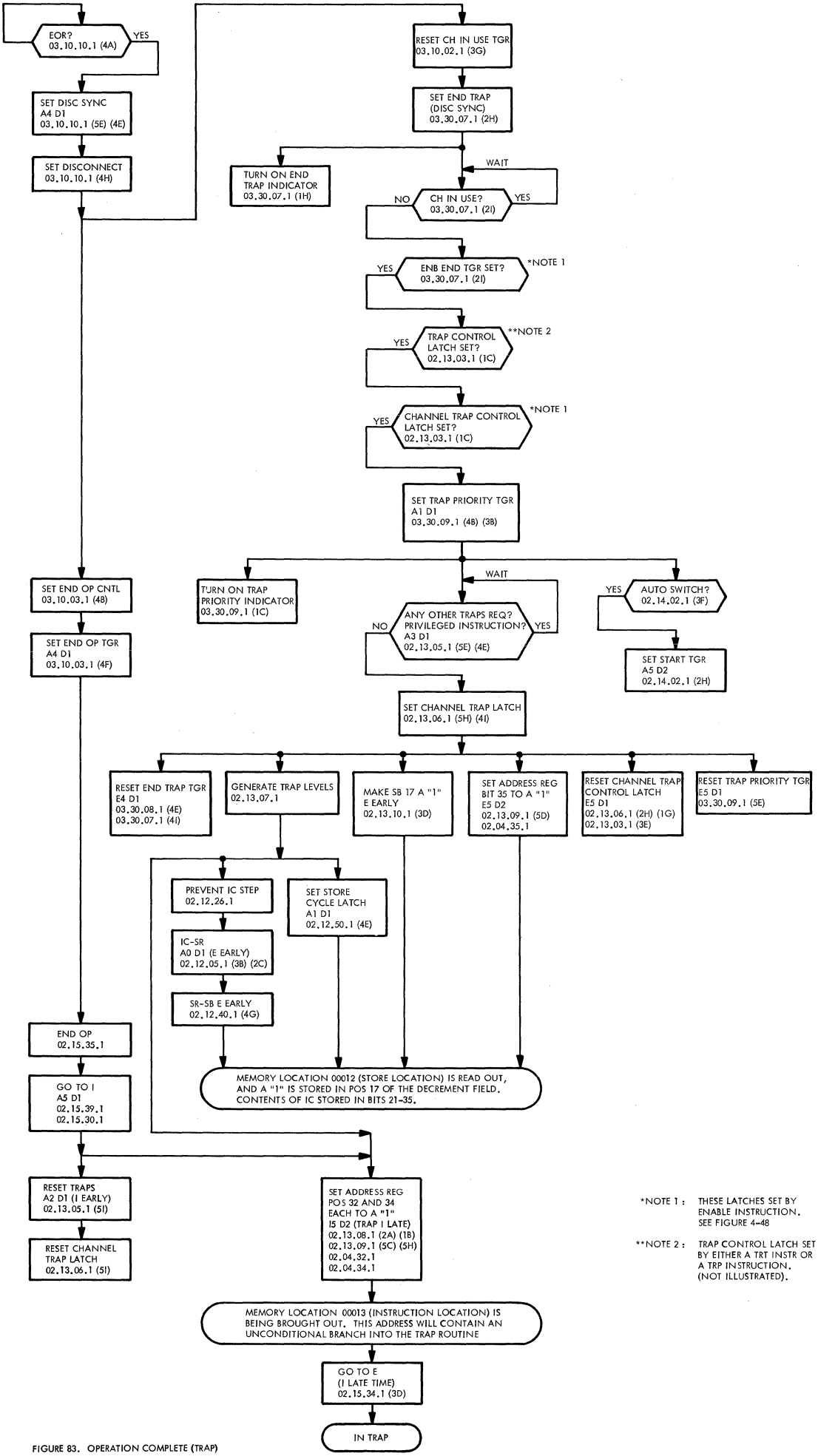


FIGURE 83. OPERATION COMPLETE (TRAP)

TRAP PRIORITY	TRAP NAME	REASON FOR TRAP	WHEN TRAP CAN OCCUR	TRAP RESTRICTIONS	STORE *8 LOCATION	TRANSFER *9 LOCATION	DISABLING EFFECT OF TRAP	HOW DISABLING EFFECT MAY BE NULLIFIED
1	Interval Timer Blast	A "C" cycle request is received before the preceding "C" cycle request is honored. 33 milliseconds have expired since the interval timer was last stepped.	A "C" cycle request may occur (and therefore a Trap request) 1) Between instructions 2) During RDS, PRD, SEN, WRS, PWR, CTR, BSR, REW, WBT, RUN, WEF instructions. 3) Between "U" cycles during execution of a RCHA instruction. A trap request initiates an immediate trap.	None	00036	00037	1) Trap control turned off 2) Resets all channels 3) Resets program reg. in CPU and shift ctr. 4) Resets waiting "C" cycle requests and interval timer overflow trap requests. 5) Interval timer will contain 2_8 less than it should. 6) An interrupted instruction is not completed.	Trap control may be turned on by executing a TRT or TRP instruction. Interval timer may be corrected by: CLA 00005 ADD =02 STO 00005
2 **	Memory Protect Violation	A store is attempted in a protected area of core. NOTE: Input information from an I/O device is never prevented from storing in a protected area.	Immediately after an "E" cycle attempting (it will not succeed) to store in a protected area of core.	1) Trap control on *' 2) Protect mode on *' If trap control is off, protect mode on, and a violation occurs, the store will be successful and a trap will not occur.	00032 Bit 16 Flag	00033	1) Protect mode turned off. 2) Violating instruction not allowed to store or complete *'.	Protect mode may be turned on by executing a SPM instruction.
3 ** Memory Protest Violation and Parity Traps are Mutually Exclusive, and therefore their relative priority is arbitrary.	Parity	Storage parity error during: 1) "I" or "IA" cycles 2) "E" or "C" cycles which do not store. NOTE: Parity is not checked during the "E" cycle of a cap instruction. "B" or "U" Cycle parity errors will not cause a parity trap. A channel trap may be requested if the operation or parity mask bit is enabled for that channel. See channel traps. If bad parity is detected when reading out of core during any type cycle, the word is regenerated back in core with bad parity.	1) Immediately after the "I", "IA", or "E" cycle causing the parity error. An instruction requiring additional cycles will not complete. 2) For a parity error during a "C" cycle taken during the execution of an instruction, the instruction will be allowed to complete before the trap is initiated. 3) For a parity error during a "C" cycle taken between instructions, the trap will be immediate.	1) Trap control on *2 2) Parity control on *2 *2 parity errors detected with either trap control or parity off do not cause an immediate trap and do not interrupt an instruction. However, the error is remembered and will cause a trap as soon as trap and parity control are on.	00040 Bit S Flag indicates error occurred when parity or trap control was off. *2 Bit 1 Flag "C" cycle error. Bit 18 Flag "I" or "IA" cycle error. Bit 19 Flag "E" cycle error. Bits 3-17 will contain the location in error if not a "C" cycle or delayed error.	00041	1) Trap control turned off. 2) Parity control turned off. 3) Parity errors detected during multi-cycle instructions will prevent any remaining cycles from being taken to complete the instruction.	Trap control and parity control may be turned on by executing a TRP instruction. Trap control may be turned on and parity control left off by executing a TRT instruction.
4 All instruction traps are mutually exclusive and therefore have no priority with regard to each other. *7 If a parity error occurs during an I or IA cycle for this instruction, a parity trap occurs and the instruction trap does not occur.	SPM*7	SPM instruction with protect mode on.	Execution of SPM instruction causes the trap.	Protect mode on	00032 Bit 16 Flag	00033	Protect mode turned off.	Protect mode may be turned on by executing another SPM instruction
	RPM*7	RPM instruction	Execution of RPM instruction causes the trap.	None	00032 Protect mode On: Bit 15 Flag Off: Bit 14 Flag	00033	Protect mode turned off.	Protect mode may be turned on by executing a SPM instruction.
	STR*7	STR instruction.	Execution of STR instruction causes the trap.	None	00000	00002	None	Not applicable.
	Floating Point	Floating-point instruction and any one or combination of the following: 1) AC char. computed to exceed 377. 2) MQ char. computed lower than 000 3) AC char. computed lower than 000*3 4) MQ char. computed to exceed 377*4 5) High order double precision operand address ODD. *3 can occur without (2) during divide only. *4 can occur without (1) during divide only.	After completion of instruction for (1) + (2) + (3) + (4). After first "E" cycle for (5) instruction will not complete.	None	00000 (5) yields Bit 12 Flag (1) + (4) yields Bit 15 Flag (1) + (3) yields Bit 16 Flag (2) + (4) yields Bit 17 Flag (1) + (2) + (3) + (4) during single prec. Divide yields Bit 14 Flag	00010	Instruction not completed for (5).	Not applicable.

FIGURE 84-1.

TRAP PRIORITY	TRAP NAME	REASON FOR TRAP	WHEN TRAP CAN OCCUR	TRAP RESTRICTIONS	STORE *8 LOCATION	TRANSFER *9 LOCATION	DISABLING EFFECT OF TRAP	HOW DISABLING EFFECT MAY BE NULLIFIED
5	Pre-Interrupt Memory Protect	Interval timer overflow, channel or direct data trap requested with protect mode on.	Trap is initiated at the time the causitive trap would have been initiated had memory protect been off. Refer to individual causitive trap descriptions.	1) Trap control on. 2) Protect mode on. 3) Restrictions for causitive trap met.	00032 Bit 17 Flag	00033 33_8 will be placed in the store location of the causitive trap. Causitive trap will begin before instruction in 33 is executed.	1) Protect mode turned off. 2) Delays execution of causitive trap until protect mode is turned off.	Protect mode may be turned on by executing a SPM instruction.
6	Interval Timer Overflow	Adder 1 carry while interval timer is being incremented.	After "C" cycle storing - incremented interval timer if "C" cycle occurred between instructions. If "C" cycle occurred during a BSR, REW, RUN, WEF or RCHA instruction, the instruction will complete before trap is initiated. If the trap request occurs during or immediately after a privileged *5 instruction, the trap will be delayed until after the completion of the instruction following the privileged instruction.	1) Trap control on. 2) Protect mode off. *6 *6 If on, pre-interrupt memory protect trap will be initiated to turn if off.	00006	00007	Blocks "C" cycle requests while waiting for trap to be executed.	If trap execution is delayed long enough to block two "C" cycle requests, an interval timer blast trap will be initiated which will reset overflow trap request. At this time, the interval timer should contain 2_8 (it will contain 0_8). This may be corrected by: CLA =02 STO 00005
7	Direct Data	Interrupt signal from direct data external device with direct data mask bit for subject channel a one. Channel / Mask Bit *11 (From ENB Inst.) B 25 C 24 D 23 E 22	After completion of the instruction in progress when the interrupt signal is received. If the trap request occurs during or immediately after a privileged *5 instruction, the trap will be delayed until after the completion of the instruction following the privileged instruction.	1) Trap control on. 2) Channel trap control on. 3) Protect mode off. *10 4) Direct data mask bit for interrupting channel must be a one *11. *10 If on, pre-interrupt memory protect trap will be initiated to turn if off.	00003 Channel B interrupt yields Bit 16 Flag Channel C interrupt yields Bit 15 Flag Channel D interrupt yields Bit 14 Flag Channel E interrupt yields Bit 13 Flag *11 Flag Bits will be stored only if the mask bit is on. It is possible to have more than one Flag Bit stored if the execution of the trap is delayed long enough to allow more than one channel to send an interrupt signal. Only one trap will be executed in this case.	00004 The instruction contained in this location must be an unconditional transfer to maintain 7090/94 compatibility.	1) Channel trap control turned off. Channel trap control may also be turned off by executing an ICT instruction. Direct data traps may also be blocked by an ENB O instruction.	1) Channel trap control may be turned on by executing a RCT or ENB instruction. The RCT instruction will allow the mask bits specified in the last ENB instruction to retain control. Any interrupt signal received while channel trap control was off will be honored after execution of RCT. The ENB instruction will permit an interrupt signal received while channel trap control was off to cause a trap if the mask bit for the channel is turned on by the new ENB instruction, even though the prior ENB instruction did not specify that mask bit. Waiting interrupt requests are reset by a direct data trap (only for those channels covered by a mask bit) an RDCX instruction (all channels) or by reading or writing from the DD channel requesting the interrupt.
8 Trap priority among channels is in order of	Channel E Channel D Channel C	1) An I/O operation completes with operation "mask bit for subject channel a one. This will occur whenever any channel	After completion of the instruction in progress when the trap request is generated. A trap request may be generated by a channel only when it goes not in	1) Trap control on 2) Channel trap control on 3) Protect mode off *12	00022*15 00020*15 00016*15 00014*15	00023*14 00021*14 00017*14 00015*15	1) Channel trap control turned off. Channel trap control may also be turned off by executing an ICT instruction.	1) Channel trap control may be turned on by executing a RCT or ENB instruction.

FIGURE 84-2.

TRAP PRIORITY	TRAP NAME	REASON FOR TRAP	WHEN TRAP CAN OCCUR	TRAP RESTRICTIONS	STORE *8 LOCATION	TRANSFER *9 LOCATION	DISABLING EFFECT OF TRAP	HOW DISABLING EFFECT MAY BE NULLIFIED
physical remoteness from CPU. Channel "A" has the lowest priority. For this discussion, it is assumed that Channel "E" is the furthest from CPU. No channel may request a trap (except an attention trap) while it is still in use. However, the condition that will eventually cause a trap request may be present for much of the channel operation. If a higher priority channel goes "not in use" and requests a trap first, even though the condition that caused its trap occurred after a trapping condition in the channel that is still in use.	Channel B Channel A	command completes, tapes complete a back space or write end of file or blank tape, or when the relays pick for a rewind. 2) Redundancy check from I/O device or channel parity error with "parity" mask bit for subject channel a one *18. This will stop the transmission of data although the channel will remain in use. 3) End of file *13,18 from tapes or from 1401 (KB instruction from 1401), or when 1622 or 1402 reader runs out of cards with "operation" mask bit a one. 4) Word parity error while reading or writing from core during "U" or "B" cycles or channel parity error during a write operation. Either "parity" mask bit (data transmission stops) or "operation" mask bit (transmission continues) must be a one. 5) Unusual end *16 signal from simplex interface I/O device with "operation" mask bit a one. The significance of the signal depends upon the simplex interface device and it will usually require a sense operation to determine the condition. 6) Simplex interface *16 attention signal with "attention" mask bit a one. The significance of the signal depends upon the simplex interface I/O device. 7) 1401 attention signal *17 with "attention" mask bit a one. The attention signal is the result of a KF instruction on the 1401. 8) Teleprocessing attention signal *17 from 1414 IV or V I/O sync with "attention" mask bit a one. It indicates a message is waiting or an output buffer is empty. 9) Unit record interrupt signal *17 with "unit record" mask bit a one. It indicates the card reader buffer is full, the punch or printer buffer is empty or the paper tape reader is full.	use. The exception to this is an attention request which does not have to wait until the channel goes not in use. If the trap request occurs during or immediately after a privileged*5 instruction, the trap will be delayed until after the completion of the instruction following the privileged instruction.	4) Appropriate mask bit must be on for both the channel and the condition for which the trap is requested. *12 if on, pre-interrupt memory protect trap will be initiated to turn it off.	00012*15 The store and transfer locations associated with a specific channel are fixed and do not change if physical remoteness from CPU is altered. *15 (1) yields bit 17 Flag (2) yields bit 16 flag (3) yields *13 bit 15 flag (4) yields bit 14 flag (5) yields bit 12 flag (6) yields bit 11 flag (7) yields bit 10 flag (8) yields bit 9 flag (9) yields bit 8 flag More than one bit may be stored if the trap execution is delayed long enough to allow multiple trapping conditions to occur on a single channel. Only the first condition will request a trap.	00013*14 *14 The instruction contained in this location must be an unconditional transfer to maintain 7090/7094 compatibility.	Channel traps may also be blocked by an ENB O instruction. This is not advisable however because data transmission will not be stopped in the event of a parity error or redundancy check with the "parity" mask bit off. This will also make it difficult to locate the word in error because the channel address counter will continue to step after the error occurs.	The RCT instruction will allow the mask bits specified in the last ENB instruction to retain control. Any trap request received while channel trap control was off will be honored after execution of RCT. The ENB instruction will permit a trap request occurring while channel trap control was off to be honored if the mask bit for both the channel and the condition is turned on by the new ENB instruction, even though the prior ENB instruction did not specify that mask bit. The conditions that will request a trap can be reset by executing the trap for that channel or by a RDCX instruction (this will reset all trapping conditions on the specified channel).

*5 Privileged instructions are: RDS, PRD, SEN, WRS, PWR, WBT, CTR, ENB, RCT, ICT, XEC, SPM

*8 The core address of the instruction following the instruction being executed when a trap request occurs will be stored in positions 21-35 of the store location, in addition to any indicated flag bits.

*9 The final operation during a trap consists of starting memory with this address selected during I time.

*13 Channel "A" end of file cannot directly cause a trap, but will induce an "operation" complete "Trap." In this case bits will be stored in positions 15 and 17 of Location 00012.

*16 Overlapped Data channels only

*17 Channel "A" only

*18 If the mast bit for redundancy check or end of file is a one, the respective TRCX or TEFX instructions will always be executed as no-operations regardless of whether the tested conditions exist.

SUMMARY OF MASK BITS FOR CHAN TRAPS

Channel	Bit Position of ENB Instruction			
	Operation	Parity	Attention	Unit Record
E	31	13	4	N/A
D	32	14	5	N/A
C	33	15	6	N/A
B	34	16	7	N/A
A	35	17	8	S

FIGURE 84-3.

SECTION 7 - OPERATOR'S CONSOLE

GENERAL

The operator's console is mounted on the right end (facing the wiring side) of the central processing unit (CPU). It is divided into five panels, labeled A through E from top to bottom.

In general, the operator's console contains the pushbuttons (also called keys and switches) and indicators that are provided as operator aids. An understanding of the console and development of its use will prove a valuable troubleshooting tool. This chapter provides the data necessary to understand the function of the various operator console switches and the meaning of each operator console indicator. Switches and indicators pertinent only to power are not included in this section but are discussed in the 7040-7044 Power Supply manual. Only the pushbuttons and indicators associated with CPU and channel functions are described in this chapter.

Figure 85 shows the physical arrangement of all the operator console pushbuttons and indicators. Although not shown, a console printer (commonly known as the output typewriter) is mounted on a table below the operator's console. The output typewriter is, as the name implies, an output device. Operation of this device falls within the realm of I-O operations and is therefore not included in this chapter. Refer to the Channel A instruction manual for typewriter operation.

SWITCHES AND FUNCTIONS

Channel Bit Density Switches

Five switches allow the operator to select the tape bit density for each of the five I-O channels (channels A-E).

Each switch is a 3-position switch with positions labeled (Figure 85) as follows:

1. $\frac{556}{200}$
2. $\frac{800}{200}$
3. $\frac{800}{556}$

The two numbers on each position indicate the high density and the low density possible for each channel. The operator sets the applicable switch to one of the three available character density positions at the operator's console. This action is a broad selection which reduces the overall choice to one of two possible character densities. The final selection is made at the tape unit. The operator selects high or low density on the tape unit. Assume that the channel A CHANNEL BIT DENSITY switch is in the 556/220 position. The tape units connected to channel A record characters per inch as either 556 or 220. The switch setting on the tape unit determines the final selection; that is, the actual character density used in a given operation is determined by the setting of the high-low density switch on the tape unit. It is therefore possible to have several tape units on 556 and the remaining tape units on 200 characters per inch on channel A.

The character density per inch also depends on the tape unit model used with the 7040-7044 system. The tape unit models and the character density capabilities are as follows:

Tape Unit Model	Character Density		
729 II	200	556	
729 IV	200	556	
729 V	200	556	800
729 VI	200	556	800
7330 I	200	556	
7330 II	200	556	800

Storage Clock Switch

The STORAGE CLOCK switch (also commonly known as the Interval Timer switch) applies only to the interval timer; it has no relationship to the timing generation circuits sometimes referred to as the "clock". Since a timer satisfies the same functions as a clock, and since the timer, in this case, is contained in core storage location 00005, the name of the STORAGE CLOCK switch is appropriate.

When the STORAGE CLOCK switch is in the ON position, storage location 5 is incremented automatically 60 times each second. The clock is stopped when the STORAGE CLOCK switch is in the OFF position. The STORAGE CLOCK switch must be in the OFF position when the SINGLE STEP or MULTIPLE STEP key is being used in either the cycle or pulse mode. The STORAGE CLOCK switch must also be in the OFF position when the DISPLAY STORAGE key is in use, or the data displayed in the storage register will be destroyed.

Entry Switches

There are two banks of entry switches. The first is an 8 x 5 matrix of switches allowing the operator to select a location in core storage in octal. The outputs from these switches are gated to the address register.

The second bank is an 8 x 12 matrix of switches enabling the operator to insert a word in the machine, using its octal configuration. This bank is subdivided into sign, instruction, tag, and address. When entering a word in core storage, proceed as follows:

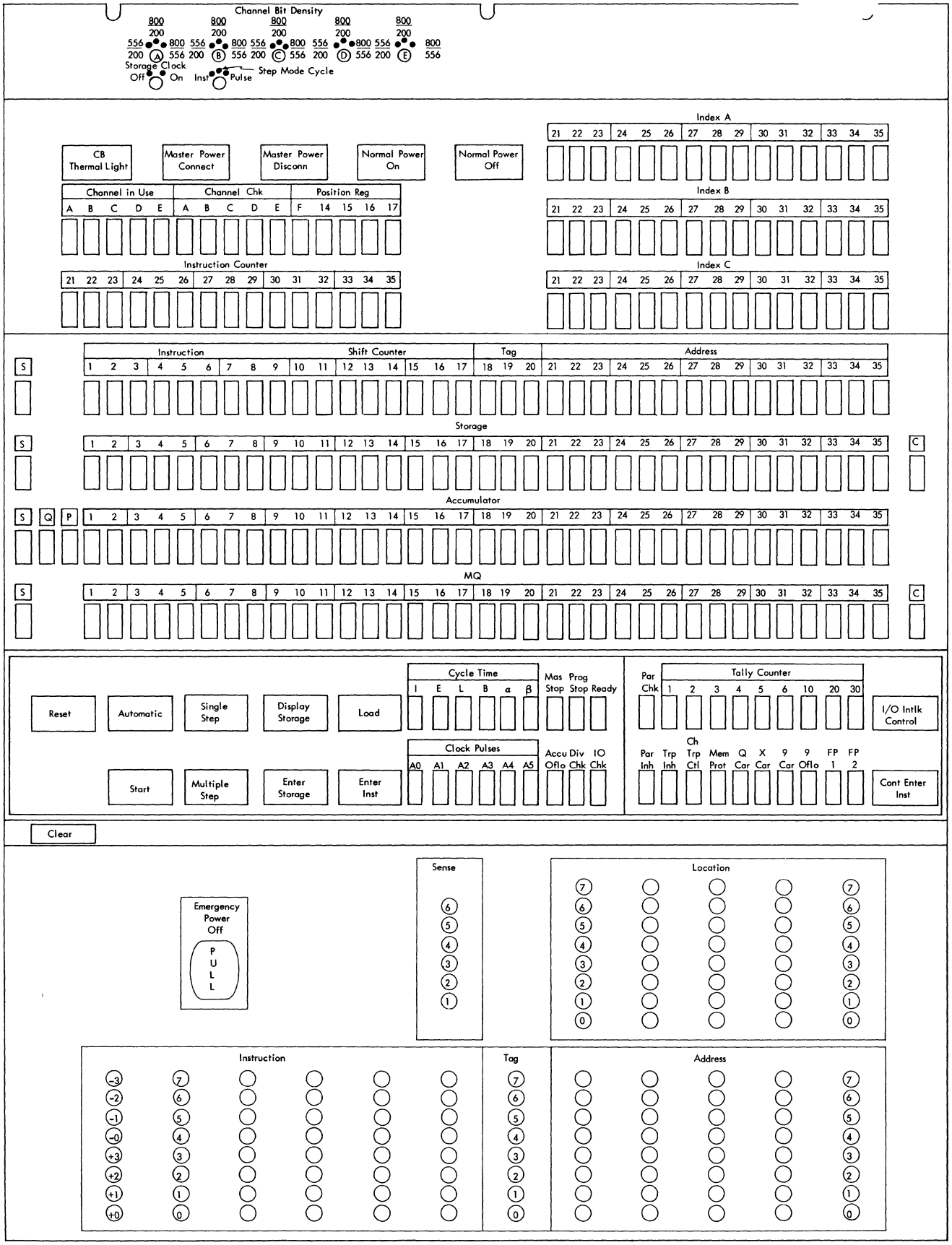


FIGURE 85. OPERATOR'S CONSOLE PANEL

1. Place octal representation of location in core storage to be referenced in location bank.
2. Place octal representation of actual word to be entered in word bank.
3. Depress ENTER STORAGE key (with the computer in MANUAL status).

When the ENTER STORAGE key is pressed, the word in the word bank is automatically stored in the core storage location as specified in the location bank.

The switches that use the location bank and the word bank are as follows:

1. LOCATION BANK switches
   a. ENTER STORAGE
   b. DISPLAY STORAGE
2. WORD BANK switches
   a. CONT ENTER INST
   b. ENTER STORAGE
   c. ENTER INST
   d. LOAD

The operation of these switches is discussed in this section. Refer to the portion that discusses the individual key to show how the LOCATION BANK and WORD BANK switches are used.

Any-Key Pulse Generation

Depressing any of the keys that perform a logical operation generates an any-key pulse. This pulse is necessary to control the execution of the desired operation. Figure 86 is a simplified diagram of the generation of the 3-usec any-key pulse.

Activating a key conditions -AND 4A, thereby triggering a single shot (SS), 3A. The 30-ms negative pulse is inverted by -OR 2A. The positive-going pulse from -OR 2A triggers single-shot 2B. The output of single-shot 2B is then sent out as an any-key pulse. In addition to the any-key pulse, an auto-any-key and auto-A2-D1 or manual-control-A1-D1 pulses are generated. These pulses depend on the setting of the AUTOMATIC switch.

Depressing the SINGLE STEP key generates the any-key pulse as explained above. A step pulse is generated if the program-stop trigger is off. Since the program-stop trigger is set by an HPR instruction, it is impossible to use the SINGLE STEP and MULTIPLE STEP keys after an HPR instruction without resetting the program-stop trigger.

The MULTIPLE STEP key allows instruction execution, single cycles, or pulses to be generated at a slow rate. The any-key pulse is generated every 50 ms, starting another instruction cycle, single cycle, or single-pulse cycle. If the MULTIPLE STEP key is depressed and held, the initial any-key pulse is generated as explained. Depressing the MULTIPLE STEP key triggers single-shot 3E. After 25 ms, single-shot 1F is triggered. The output of single-shot 1F goes positive 25 ms later, conditioning AND 5F. The output of the AND is a positive-going pulse, which again triggers SS 3E. This cycle (50 ms in duration) is continued until the MULTIPLE STEP key is released. The output of SS 3E triggers SS 2B via-OR 2A. The any-key pulse is therefore generated every 50 ms.

The MULTIPLE STEP and SINGLE STEP keys as operations are discussed later in this section.

Automatic-Manual Status

The AUTOMATIC switch is a dual-acting pushbutton used to put the computer in automatic or manual status. Moving this switch to the OFF position stops the CPU after it has completed the instruction being processed. If a channel is in use, the computer continues to execute instructions and remains in the automatic status until all channels have been disconnected. When the CPU stops, the machine is in true manual status. If the interval timer switch is on, the interval timer continues to function.

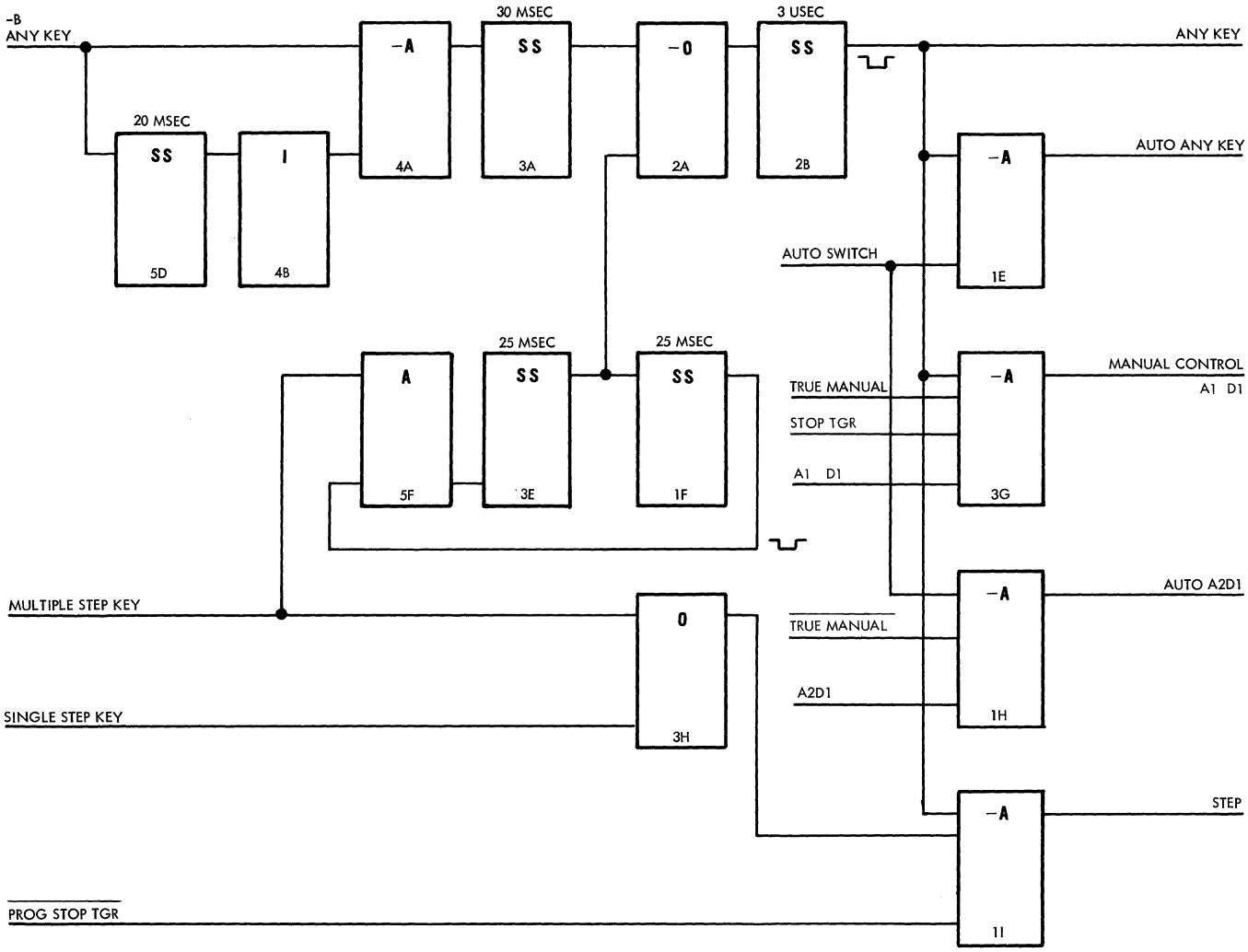
Figure 87 shows how the computer is switched to manual status. When the AUTOMATIC switch is placed in the manual position, one leg of AND 2C is conditioned. If the I-O interlock switch is on or if no channels are in use, the AND condition is met. The next I1D2 pulse sets the master-stop trigger. The setting of this trigger stops all logical computer operations. The machine is now in true manual status.

When the AUTOMATIC switch is in the automatic position, it is necessary to prevent the use of certain switches. The switches that are operative in the automatic status are as follows:

1. CLEAR
2. RESET
3. CONT ENTER INST (single step is operational with this switch on)
4. START
5. LOAD

The switches that are operative in manual status are as follows:

1. RESET
2. CONT ENTER INST
3. START (clears the program-stop trigger only)
4. SINGLE STEP
5. MULTIPLE STEP
6. DISPLAY STORAGE
7. ENTER STORAGE
8. ENTER INST
9. LOAD



NOTE:
LOGIC 02.14.01.1

FIGURE 86. ANY-KEY PULSE GENERATION

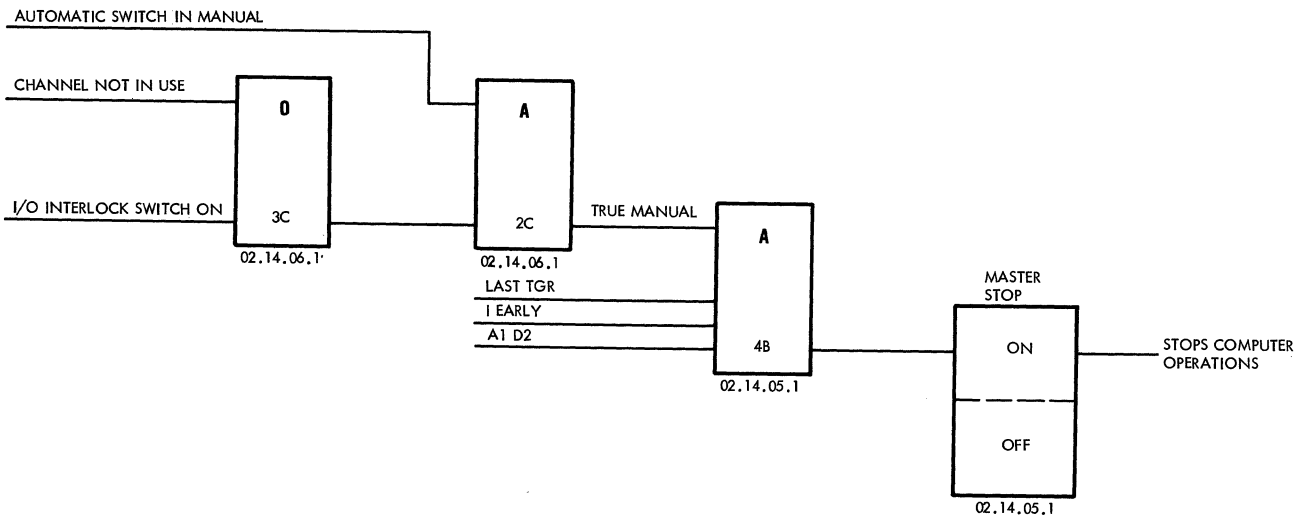


FIGURE 87. AUTOMATIC-MANUAL OPERATION

I-O Interlock Control Switch

The I-O interlock control switch is used in conjunction with the AUTOMATIC switch to help locate I-O troubles. The I-O INTLK control switch pushbutton light is normally off. The light is on when the I-O interlock switch is on.

If an I-O unit is selected with the I-O interlock switch off, system operation reverts to automatic status until the channel is no longer in use, even though the AUTOMATIC switch is in the manual position. When a unit is selected, the channel-in-use level will decondition AND 2C (Figure 87), resulting in an automatic status until the channel is no longer being used. The machine then returns to manual status.

If the I-O interlock control switch is on and the machine is in manual mode when an I-O unit is selected, the machine executes the select instruction and remains in manual status (Figure 87). The I-O interlock control switch overrides the channel-in-use level. Therefore, AND 2C remains conditioned, maintaining the true-manual level.

START Key Operation

Depressing the START key continues calculation at high speed. The START key continues the program only when the computer is in automatic status. Figure 88 is a simplified diagram of the function of the START key. Note that the program-stop trigger is set only on an HPR instruction. Setting the start trigger resets the program-stop trigger. When the START key is activated, the start trigger is set and the program-stop trigger is reset in either automatic or manual status. At A5 D1 time, the start trigger is reset. The START key must be depressed before the SINGLE STEP or MULTIPLE STEP key can be operated if the HPR instruction is executed and the computer is then placed in manual status.

When the computer is in automatic status, setting the start trigger allows the start-machine level to reset the master-stop trigger at A4 D2 time (Figure 88). The master-stop trigger allows the computer to perform the logical operations necessary in executing instructions.

Continuous Enter-Instruction Operation

The CONT ENTER-INST key is a dual-acting pushbutton. Operating this key forces the system to continuously execute the instruction set in the word bank. Figure 88 shows the function of this switch. An AND (2D) is conditioned during every I early cycle. When this AND is conditioned, the following occurs, which is not common during a normal I cycle:

1. Stepping of the instruction counter is inhibited.
2. Transfer of the memory data register to the storage bus is inhibited.
3. Operator's keys are transferred to the storage bus.

The data that is routed to the storage bus is sent to the program register. Execution of the instruction is the same as though it was taken from core storage. The CONT ENTER-INST key is valuable in trouble-shooting because it provides an easy way to scope the operation of a particular instruction. This button is lighted when in the continuous-enter instruction mode. The continuous-enter instruction function may also be used in manual status with the SINGLE STEP and MULTIPLE STEP keys.

RESET Key Operation

The RESET key is operative in both automatic and manual status. This key resets all registers and indicators in the logical section of the CPU and all channels but does not affect core storage. Figure 89 is a flow diagram of the reset and clear functions.

Depressing the RESET key sets the reset II trigger at A3 D1 time. Setting the reset II trigger sends a reset level throughout the CPU and to all I-O channels. The next A2 D1 pulse resets the reset II trigger. The computer is stopped, and further action must be taken by the operator.

Several degrees of resets are used in the 7040-7044 system. In general, the various levels of resets may be grouped. Following is a list of the resets, what can cause them, and what is reset (logic 04.14.07.1):

1. Interlock Reset (Power-On Reset; Reset, Load, or Clear Key)
   a. Reset or Cleared: All Channels (same action as RDCA-E)
      Program Register (both)
      Shift Counter
      Position Register
      Address Register
      Tag Register
      SR C Bit
      IA Trigger
      PRE-IA Trigger
      Channel Trap Control
      Pulse-Mode Trigger
      Program-Reset Trigger
      Program-Stop Trigger
   b. Set or Turned On:
      End OP
      Master-Stop Trigger
      Trap-Control Trigger
      Parity Mode
      Carriage-Return trigger (typewriter)

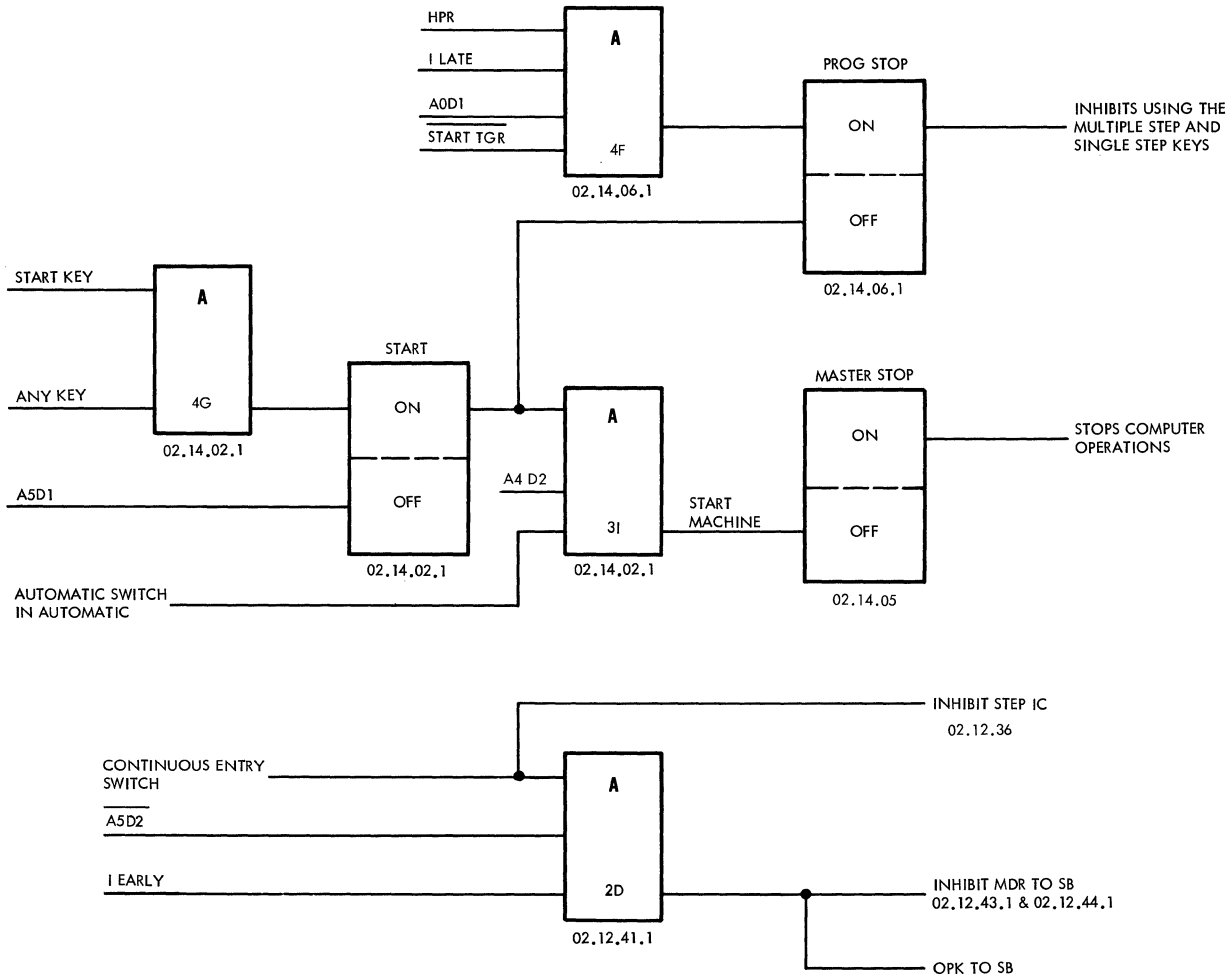


FIGURE 88. START AND CONTINUOUS ENTER INSTRUCTION OPERATION

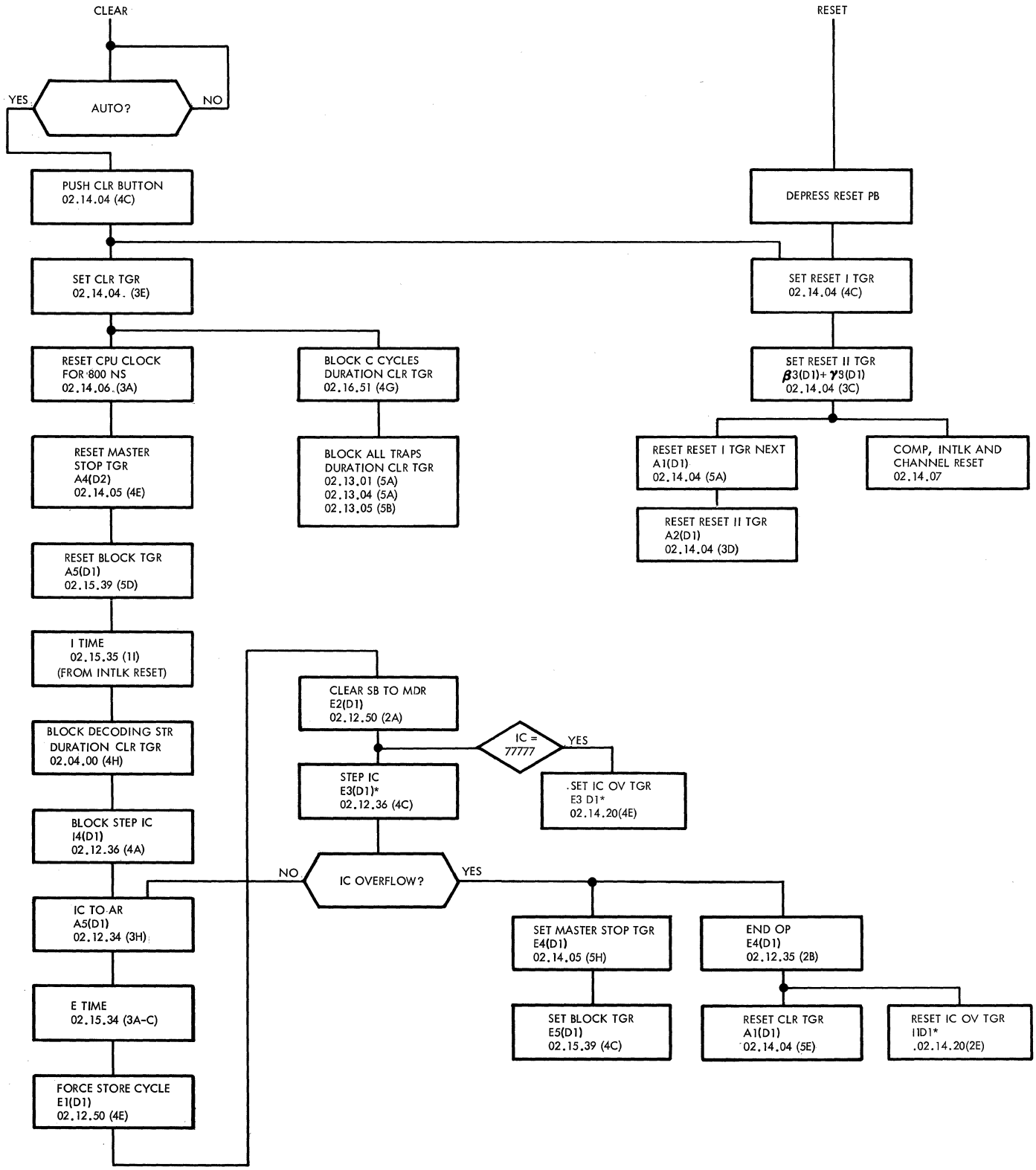


FIGURE 89. CLEAR AND RESET CONTROLS

2. Computer Reset (Power-On Reset; Reset or Clear Key)

Reset or Cleared:
- Storage Register
- Accumulator
- XRX
- XRA, XRB, XRC
- Instruction Counter
- MQ Register
- Div-Check Trigger
- IO-Check Trigger
- AC-OV Trigger
- Partial-Store Trigger
- Load Trigger
- Load Z Trigger
- Master C Trigger
- C-Cycle-Request Trigger
- Early C Request Trigger
- Memory Protect Mode
- All Trap Latches
- All Trap Request Latches
- Blast Control Latch
- Floating-Point Condition Latches

CLEAR Key Operation

With the computer in automatic status, activating the CLEAR key sets all areas of core storage to zero and resets all registers and controls in the CPU and channels. The CLEAR key is inoperative in manual status (Figure 89).

As shown in Figure 89, the reset trigger is set when the CLEAR key is depressed, generating the reset levels. In addition to the reset trigger being set, the clear trigger is set. The clear trigger controls the clearing of core storage.

When the clear trigger is set, an 800 NS single shot is fired, resetting the clock timing ring. The reset trigger is reset by clock ring pulse A2 D1. At A4 D2 time, the master-stop trigger is reset. At A5 D1 time, the block trigger is reset. The computer will now begin an I cycle. At the end of the I cycle, an E cycle is started with the AR cleared. During E time, 0's are stored in location 00000 of core storage (logic 02. 12. 50. 1). At E3 D1 time, the instruction counter is stepped. The next A5 D1 pulse transfers the contents of the instruction counter to the address register (00001). Zeros are then stored in location 00001. The machine will continue performing E cycles and storing zeros in each address of core storage.

When instruction counter overflow occurs, the end-operation trigger is set, which, in turn, allows setting of the master I trigger. The instruction counter overflow sets the master-stop trigger, and the next A1 D1 pulse resets the clear trigger. All core storage locations now contain 0's.

Storage Test and Parity Check Controls

Two switches are provided for use by Customer Engineers in diagnosing memory problems: storage-test and stop-on-storage-test-parity. These switches are located on a subpanel behind the main operator's station.

Storage-Test Switch

This on-off toggle switch controls the operation of memory test circuits. With this switch on, the ENTER STORAGE key causes the word set in the word bank of the entry switches to be consecutively stored in every position of core storage until the switch is turned off or a reset occurs. The DISPLAY STORAGE key causes consecutive locations to be read out of core storage and checked for proper parity until the switch is turned off or a reset occurs.

Stop-on-Storage-Test-Parity Switch

This on-off toggle switch is active only when the storage-test switch is on. If the stop-on-storage-test-parity switch is on and a parity error is detected during the "display storage" function, the word in error is displayed in the storage register and the "word in error" location plus 1 is displayed in the instruction counter. The "display storage" function terminates when a parity error is detected.

Enter Storage Operation

If the ENTER STORAGE key is depressed and the CPU is in manual status, the contents of the word bank entry keys are stored in core storage at the address specified in the location bank. If the computer is in automatic status, no operation is performed if the ENTER STORAGE key is depressed.

Figure 90 is a flow diagram of the sequence of events that occur when the ENTER STORAGE key is activated. Note that, if the storage-test switch is on, all locations in core storage will contain the data from the word bank.

Display Storage Operation

Figure 91 shows the sequence of events that occur after the DISPLAY STORAGE key is activated with the computer in MANUAL status and the various memory test functions.

The contents of one location may be displayed in the storage register, or all locations in core storage may be read out and checked for correct parity. If the storage-test switch is off, the location specified in the location bank is read out of core storage and displayed in the storage register. If the storage-

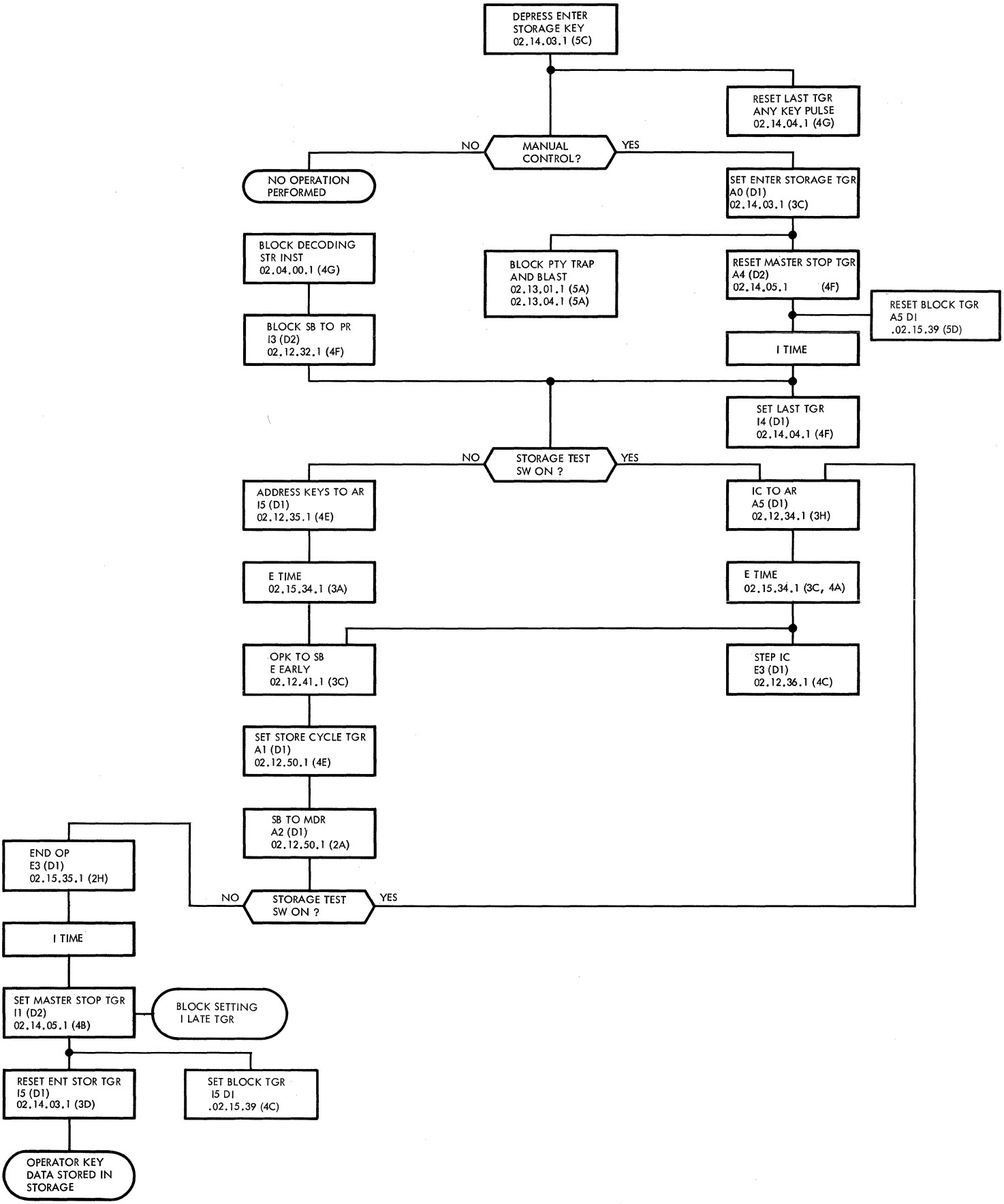


FIGURE 90. ENTER STORAGE

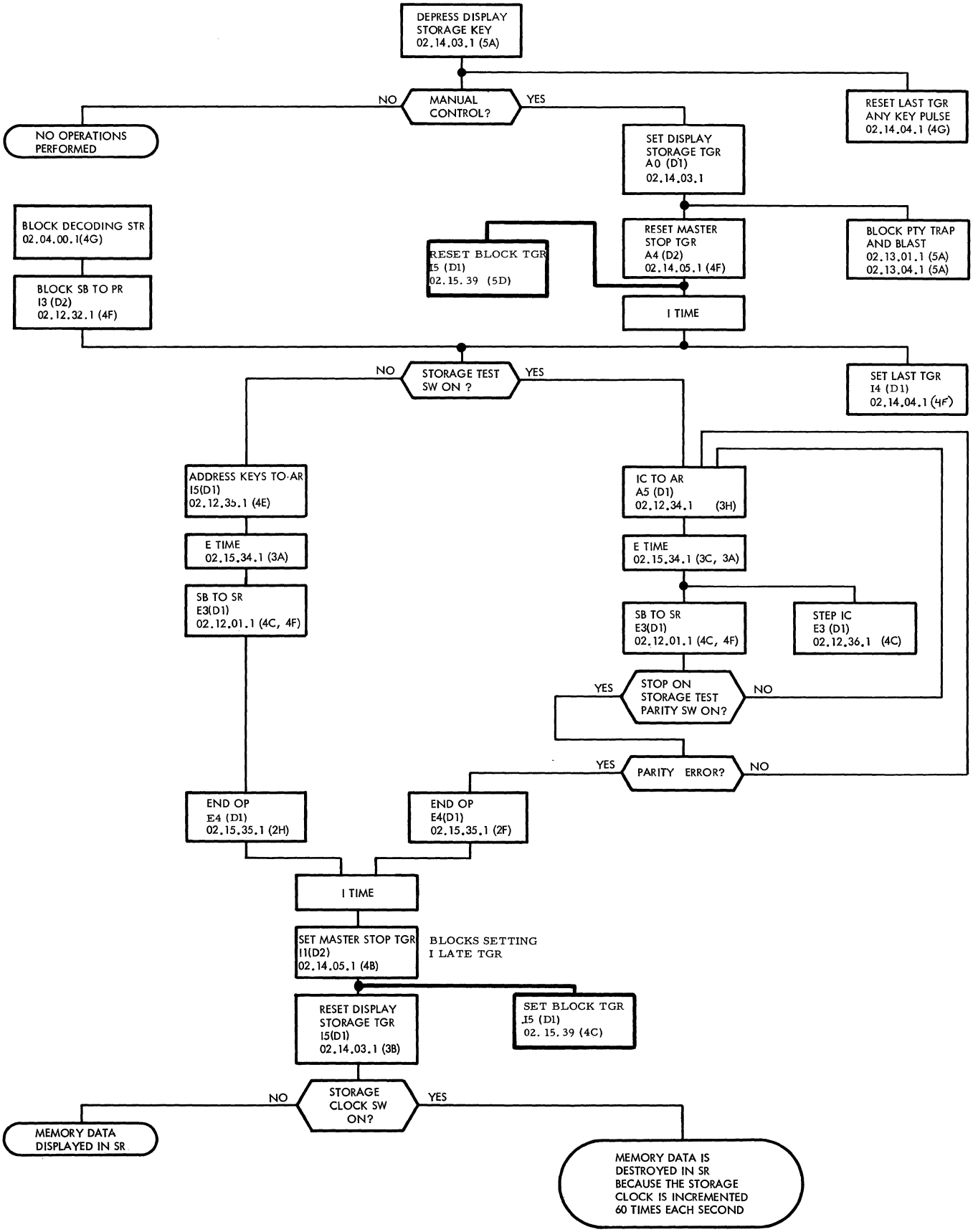


FIGURE 91. DISPLAY STORAGE

test switch is off, the location specified in the location bank is read out of core storage and displayed in the storage register. If the storage-test switch is on, all locations of storage are read and checked for correct parity. If a parity error occurs and if the check switch is on, the computer will stop. The storage register will contain the word in error, and the error location plus 1 will be displayed in the instruction counter. When activating the DISPLAY STORAGE key, turn off the INTERVAL TIMER switch to prevent destroying the contents of the storage register.

Enter Instruction Operation

To execute an instruction contained in the instruction word bank, the computer must be in manual status and the ENTER INSTRUCTION key depressed. Figure 92 is a flow diagram of the logical operations performed when this key is activated. Activating the ENTER INSTRUCTION key causes one instruction to be executed.

LOAD Key Operation

The LOAD key is normally active in automatic status when the CPU is stopped and no channels are in operation. If the LOAD key is depressed when in manual status and an instruction is being executed, the instruction is completed before an interlock reset then occurs.

Depressing the LOAD key in automatic status results in transferring the instruction in the word bank to the program register and decoding the instruction. If the instruction is a Read Select or a Write Select, a control word (IORD) with infinite word count and with an address of 00100 is loaded in the selected channel, and a read or write operation is performed. When an end of record is received from the selected channel and the channel-in-use trigger is reset, the computer transfers to location 00101 and proceeds from there.

If the instruction in the word bank is not a select instruction, the final results may be erroneous; therefore, any instruction other than a select instruction is considered illegal when the LOAD key is involved.

Figure 93 is a flow diagram of the operation of the LOAD key. Assume that all conditions are met for selecting a channel. Note that two instructions are executed when activating the LOAD key. The first instruction is a Read Select or a Write Select. This instruction is taken from the word bank as entered by the operator. The second instruction is a Reset and Load Channel (RCHX). The channel may be used for reading or writing. The ability to perform a Write Select is desirable when the contents of core storage must be saved for future reference. Since an infinite word count is sent to the channel, a write select to tapes requires manual intervention or the tape will run off the end of the reel. In addition, if a write select to interface 5 is given, data will be continuously sent to the 1401 Data Processing System until manually disconnected.

Step Mode Selector Switch Functions

This 3-position rotary switch controls the mode of operation when SINGLE STEP or MULTIPLE STEP is depressed. The three positions are INST, CYCLE, and PULSE. The first position (INST) is the normal operating position, which provides for execution of a single instruction at a time when SINGLE STEP is used. The second (CYCLE) and third (PULSE) positions are CE functions, which allow the operation to be slowed down still further to observe details of a single instruction. The SINGLE STEP key initiates a machine cycle (I or E or L) with the STEP MODE switch in the CYCLE mode and a single pulse with the switch in the PULSE mode. These two positions are inoperative when the STORAGE CLOCK switch is on.

Single-Step and Multiple-Step Operations

The single-step and multiple-step operations enable the operator, when the CPU is in manual status, to proceed with the program either step by step or at a slow automatic rate of speed. If an instruction is executed which causes an I-O unit to be selected, the computer operates in the automatic mode until the I-O unit is disconnected. When the disconnect occurs, the computer returns to manual status. The computer should be placed in manual status and the START key depressed before using the SINGLE STEP or MULTIPLE STEP keys.

The differences between the SINGLE STEP and MULTIPLE STEP keys are:

1. Single-step operation allows only one instruction cycle, one single cycle, or one single pulse.
2. Multiple-step operation allows an instruction cycle, a single cycle, or a single pulse to occur every 50 ms as long as the button is depressed.

The generation of the step pulse and the differences stated are illustrated in Figure 86. The logical operations performed by the SINGLE STEP and MULTIPLE STEP keys depend on the setting of the STEP MODE selector switch. Only the SINGLE STEP key is referred to in explaining the operation of the three positions of the STEP MODE switch.

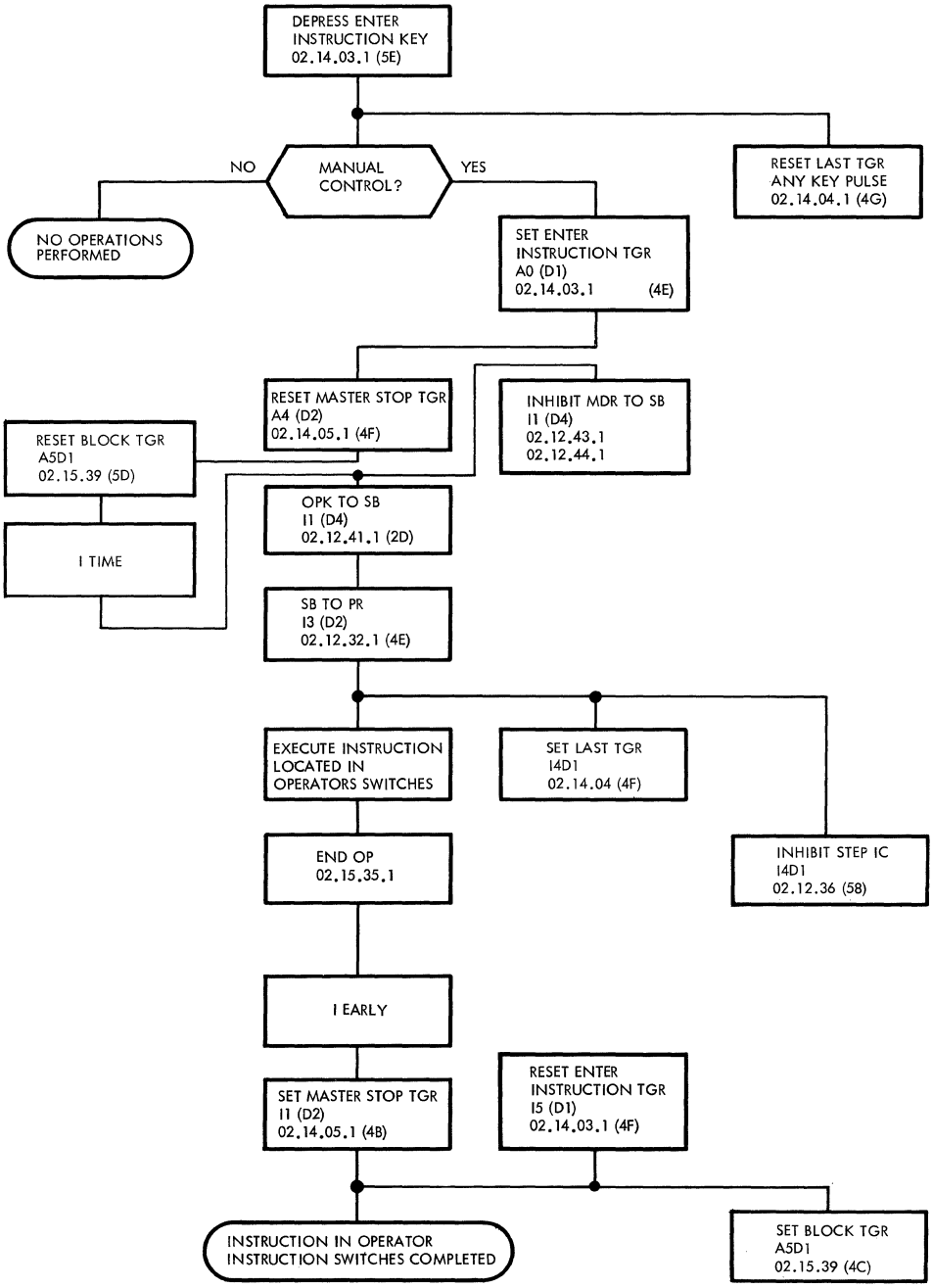


FIGURE 92. ENTER INSTRUCTION

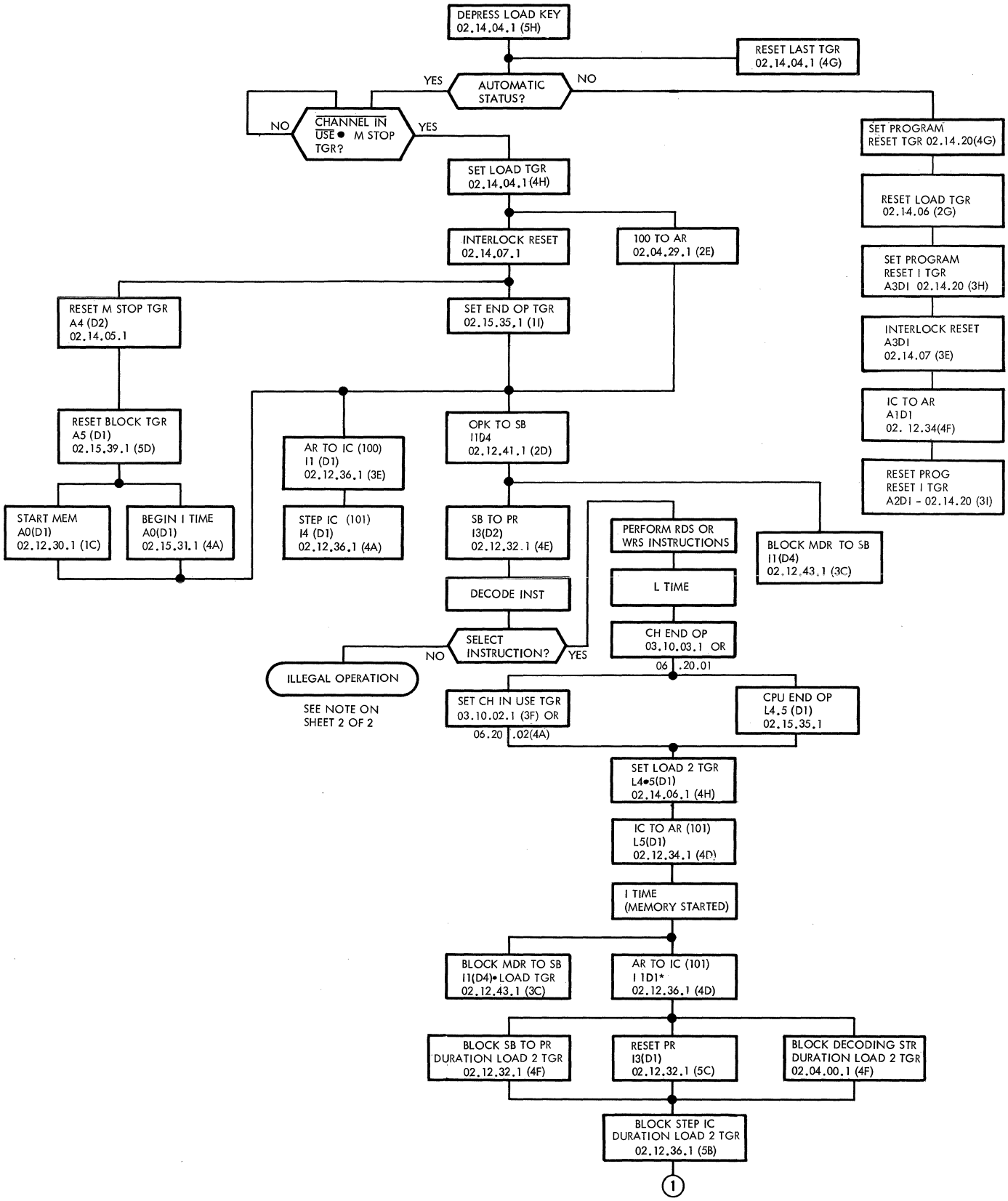
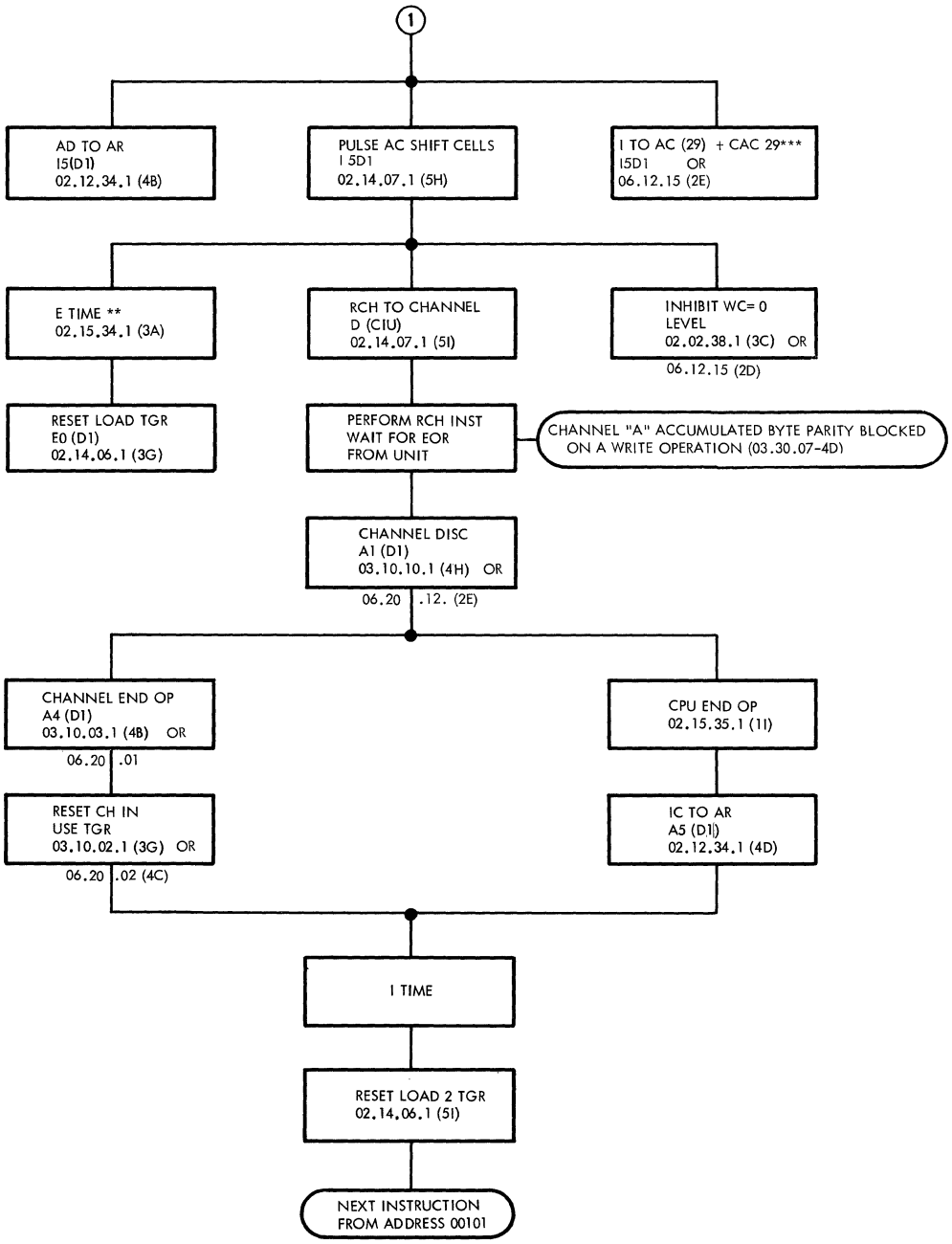


FIGURE 93. LOAD KEY OPERATION (SHEET 1 OF 2)



**SR NOT TRANSFERRED TO AD DURING E TIME (02.12.08.1) ***CAC 29 LOADED ONLY IF RD5 IS DECODED.

NOTE: IF AN INSTRUCTION OTHER THAN SELECT IS IN THE KEYS -----

1. THE INSTRUCTION WILL BE REPEATED CONTINOUSLY UNLESS IT HAS AN L CYCLE IN WHICH CASE A HANG CONDITION WILL OCCUR.
2. ADDRESS 100 IS LOGICALLY ADDED TO THE EFFECTIVE ADDRESS LOCATION DURING E CYCLES (100 IS FED TO AR AS LONG AS THE LOAD TRIGGER IS ON.)
3. TRANSFER OR SKIP INSTRUCTIONS WILL NOT TRANSFER OR SKIP.
4. PUSH RESET TO CLEAR LOAD TRIGGER.

FIGURE 93. LOAD KEY OPERATION (SHEET 2 OF 2)

Instruction Mode

Figure 94 is a flow diagram of the operation of the Single Step instruction when in true-manual and in instruction mode. Figure 95 is a simplified logic diagram of the action that occurs when the STEP key is depressed for an instruction cycle.

Assume that the STEP SELECTOR MODE switch is in the INST position, the program-stop trigger is reset, the computer is in manual status, and the SINGLE STEP key is depressed (Figure 95). The next A0 D1 pulse conditions AND 4A, setting the single-instruction trigger. The last trigger is reset by the any-key pulse and set by I late (logic 02. 14. 04. 1). The master-stop trigger is then reset by the A4 D2 pulse, and the instruction is executed. During I time of the next instruction, an I1 D1 pulse sets the master-stop trigger. No further instructions are executed until the SINGLE STEP key is depressed again. The single-instruction trigger is reset by the next A5 D1 pulse.

The information in the internal registers may be checked for accuracy. This provides a means of troubleshooting machine malfunctions or isolating a program error.

The clock ring is stepping and the CYCLE TIME I indicator is on at the completion of the instruction.

Single-Cycle Mode

Figure 96 is a flow diagram of single-cycle operation when the STEP key is depressed and the computer is in true-manual and in cycle mode.

In the single-cycle mode, one I, or E, or L cycle is executed each time the SINGLE STEP key is activated. An instruction requiring I and E cycles would require depressing the SINGLE STEP key twice to complete the instruction.

The single-cycle mode of operation is illustrated in Figure 95. Assume that the computer is in manual status, the STEP SELECTOR MODE switch is in cycle mode, the INTERVAL TIMER switch is off, the program-stop trigger is reset, and the SINGLE STEP key is activated. The next A2 D2 pulse conditions AND 4G, setting the single-cycle trigger. At A4 D2 time, AND 4E is conditioned to reset the master-stop trigger. The machine begins executing the instruction. Since the machine is in the cycle mode, AND 4A will be conditioned at I late, E late, or L late, and A4 time for a duration of three clock ring pulses. The output of AND 4A sets the master-stop trigger. After setting the master-stop trigger, the next A1 D1 pulse resets the single-cycle trigger (AND 5H). If the last cycle executed was an I cycle, the next cycle will be an E cycle for all instructions requiring an E cycle.

Single-Pulse Mode

In troubleshooting a machine failure, it is often necessary to single-pulse through an instruction to find the point at which the failure occurred. The single-pulse mode allows only one clock ring pulse to be sent to the CPU with each depression of the SINGLE STEP key. Figure 97 is a flow diagram of the single-pulse function when in pulse mode and in true-manual and the STEP key is activated. In normal operation, the shift cell (1C) is set, allowing the clock ring to run continuously (Figure 98). When in the single-pulse mode, the shift cell is reset (clock gate down) and the oscillator pulses are blocked. The only time that the clock gate is down and the clock ring stopped is when the computer is in true-manual and in pulse mode.

Assume that the computer is in manual status, the INTERVAL TIMER switch is off, and the STEP MODE switch is in the PULSE position. These conditions satisfy AND's 5A and 5B (Figure 98). When the next A3 D1 pulse occurs, the AND 3D conditions are met, setting the pulse-mode latch. Setting this latch resets the shift cell. Dropping the clock gate level blocks the oscillator pulses, and the clock ring stops stepping.

After the pulse-mode latch is set, and if the alpha-late trigger is reset, the master-stop trigger is reset. If a channel is put in use in pulse mode, the computer reverts to automatic status until the channel is no longer in use. The pulse mode latch is reset if in AUTOMATIC, allowing the clock ring to step. When the channel goes not in use, the true-manual level is restored and will satisfy the AND (5A, 5B) condition and revert to pulse mode, manual status.

Figure 99, A and B, shows the clock-gate control conditions before and after depressing the STEP key.

The initial starting conditions of -OR's 5C and 4C are (Figure 99, A) (1) -OR (5C), both inputs negative, and (2) -OR (4C), both inputs positive. Two inputs to AND 3C are conditioned at this time. As shown in Figure 86, the step pulse is generated when the SINGLE STEP key or the MULTIPLE STEP key is activated. Note that the program-stop trigger must be reset before these keys are activated (Figure 86). When the step pulse is generated, AND 3C (Figure 99, B) is conditioned. The output of AND 3C allows setting the shift cell on the next master-oscillator pulse. The out-of-phase output from the shift cell allows one output pulse from the clock ring to be distributed to the CPU. The in-phase output of the shift cell is now negative. This negative output causes the output of -OR 4C to go positive. The output of -OR 5C then goes negative, deconditioning AND 3C. The shift cell is reset by the next master-oscillator pulse. The circled polarities in Figure 99, B, indicate circuit conditions after resetting the shift

cell and before dropping the step level. The two -OR's (5C and 4C) and AND 3C return to the initial starting conditions, and the clock gate is down, inhibiting stepping the clock ring until the SINGLE STEP key is again activated.

In normal operation, the memory-select pulse is sent to memory at A0 D1 time and the address register is sent to the memory address register at A0 D2 time. The memory-select pulse initiates a memory cycle, which reads the data located in the specified memory location.

In single-pulse mode, the timing of memory selection occurs at the end of A1 time but after the SINGLE STEP key is activated for the A2 pulse. During A0 time, the clock ring is stepped to A1. Note that the clock ring levels are present at the AND's even though the shift cell is reset. As shown in Figure 98, an AND 1F condition is met at A1 D2 time, with the pulse-mode latch set and an any-memory-cycle early level present. At this time, the address register is transferred to the memory address register. An A1 D1 output from the clock ring conditions one input to AND 4H, and the shift cell is reset, conditioning a second input. Assume that the any-memory-cycle early level (the third input) to AND 4H is also present. When the AND condition is met, the pulse-mode-beta-1D1-delayed trigger is set. When the SINGLE STEP key is again depressed to generate the A2 pulse, setting the shift cell conditions AND 1G. This output generates the select-memory pulse, initiating the memory cycle. The clock ring is also stepped to A2 time, performing the logical functions that occur during A2 time. The next master oscillator pulse conditions AND 3H, resetting the pulse-mode-beta-1D1-delayed trigger. The memory cycle is completed, and the data from memory is available at the storage bus.

Sense Switches

Six sense switches are used as programmers' tools. Each sense switch may be individually checked by a Sense Switch Test (SWT) instruction. If the sense switch is on, the computer skips the next instruction and proceeds from there. If the sense switch is off, the next sequential instruction is executed. The sense switch feature allows certain program routines to be bypassed or selected. (See the CPU Logic Diagrams Manual for a flow diagram of the operation of the SWT instruction.)

INDICATORS

The indicators on the operator's console are provided as operator and CE aids. These indicators give valuable information, such as type of error, contents of a register, location in error, and status of computer. The purpose of this section is to give the function of the various indicators and the turn-on and turn-off conditions. The following lists the purpose of registers, counters, timers, and power indicators (the conditions that light the power indicators are discussed in the 7040-7044 Power Supply manual):

1. Internal Registers: The contents of the internal registers (accumulator, multiplier-quotient (MQ), storage register, instruction counter, address register program register, position register, shift counter, and index registers A, B, and C) are displayed directly on the panel.
2. Cycle Time: The cycle time indicators indicate the cycle in which the machine is currently operating, B, I, L, or E time; the status of alpha and beta triggers is also indicated for a 7106 CPU.
3. Tally Counter: The tally counter differentiates between the L cycles of a floating instruction and provides gating for their different operational steps. The tally counter is divided into two stages. The indicators on the test panel indicate which of the two steps the machine is currently operating. Positions 1 through 6 indicate the flow of a single-precision floating point. Tally counters 10, 20, and 30, with positions 1 through 6, indicate the flow of double-precision floating point.
4. Clock Pulses: (A0 through A5): These indicators indicate the state of the timing ring.
5. Storage Register C Bit: This indicator indicates the 37th bit of the memory word. This bit always makes the word parity odd.
6. M-Q Register C Bit: This indicator indicates the word parity bit of an I-O word being transferred between CPU and channel A.
7. Tag: These indicators indicate the index register to be used for the instruction.
8. CB Thermal: The CB thermal light will be on whenever a logic d-c supply circuit breaker trips, or a thermal or air flow switch opens within the basic machine.
9. Power-On: The NORMAL POWER-ON light will come on whenever all power-up sequencing is completed. If the power-on light does not come on after a suitable delay, the operator should check the power sequence indicators within the power distribution unit.
10. Master Power Connect: This pushbutton will be lit when the input service line power is connected to the power sequencing control of the system. It will be turned off when the MASTER POWER DISCONN is depressed.

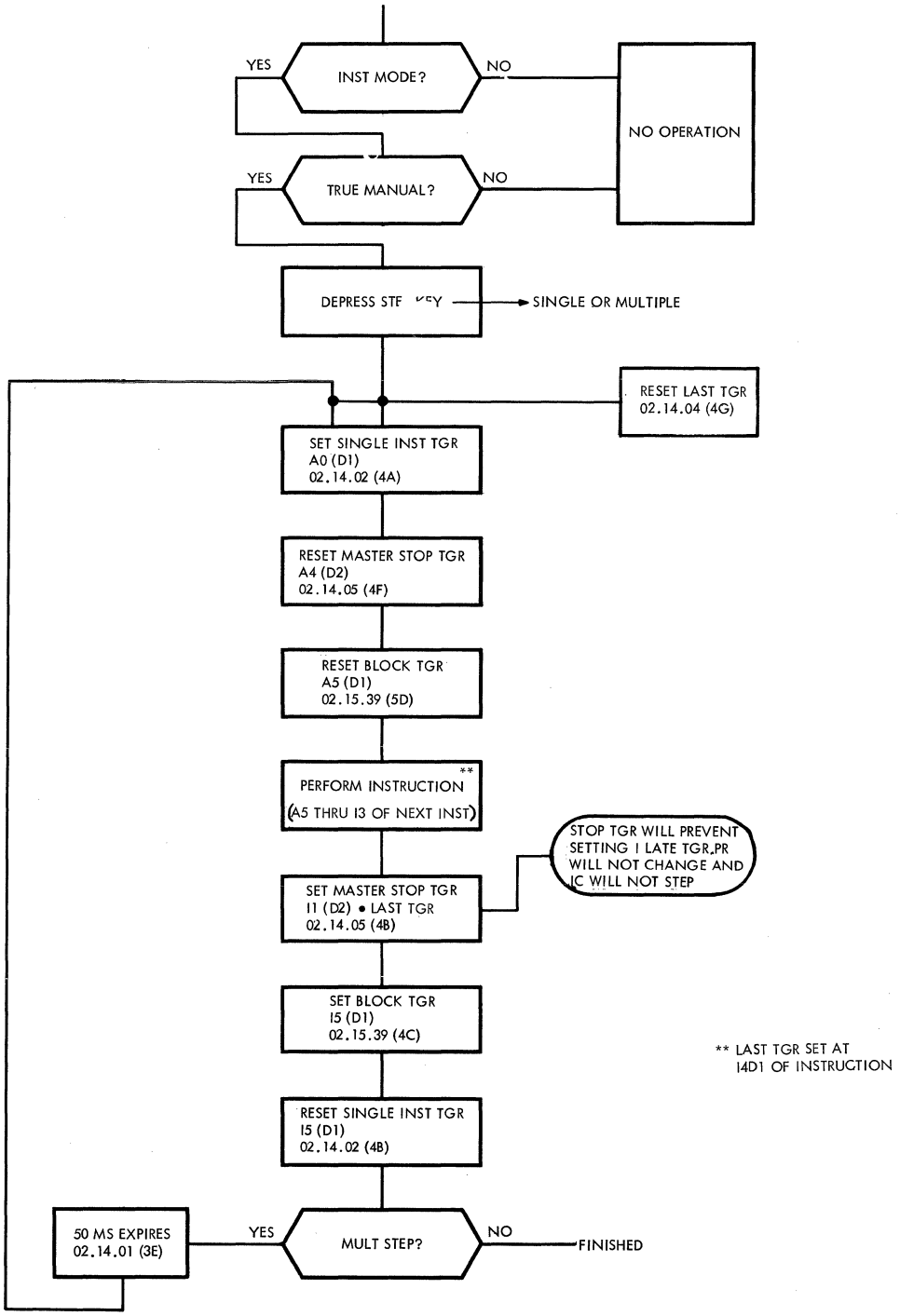


FIGURE 94. STEP SINGLE INSTRUCTION

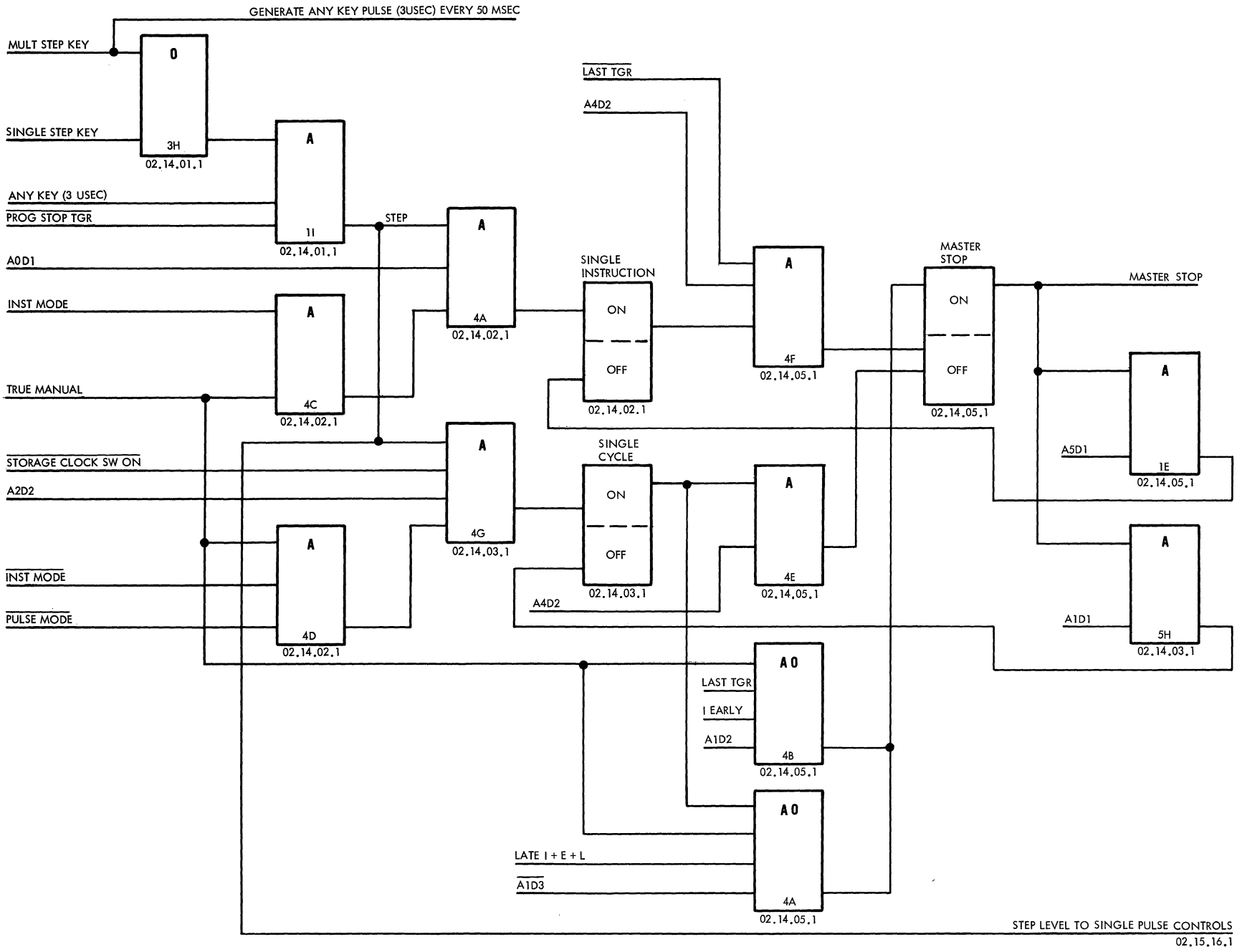


FIGURE 95. SINGLE INSTRUCTION AND SINGLE CYCLE OPERATION

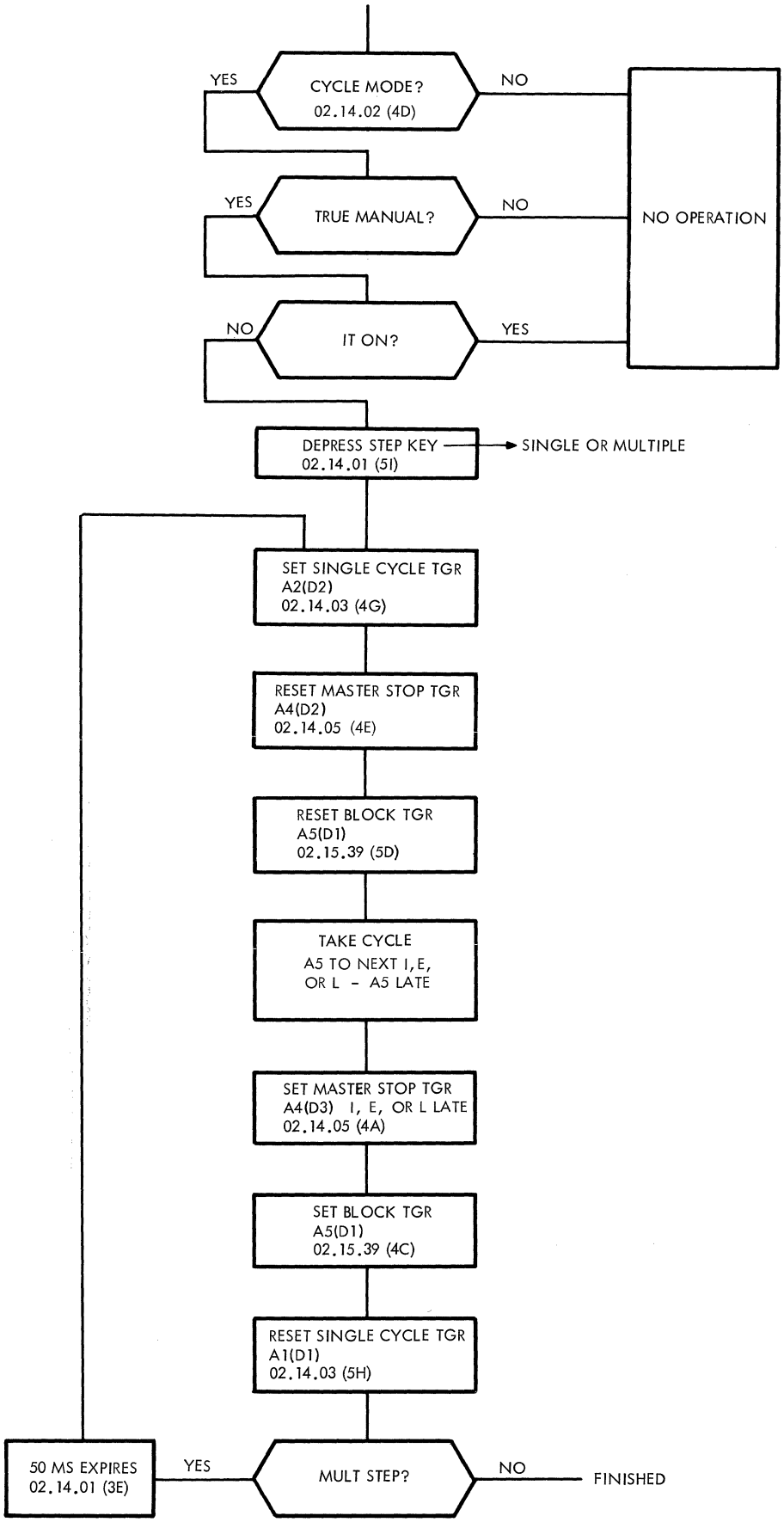
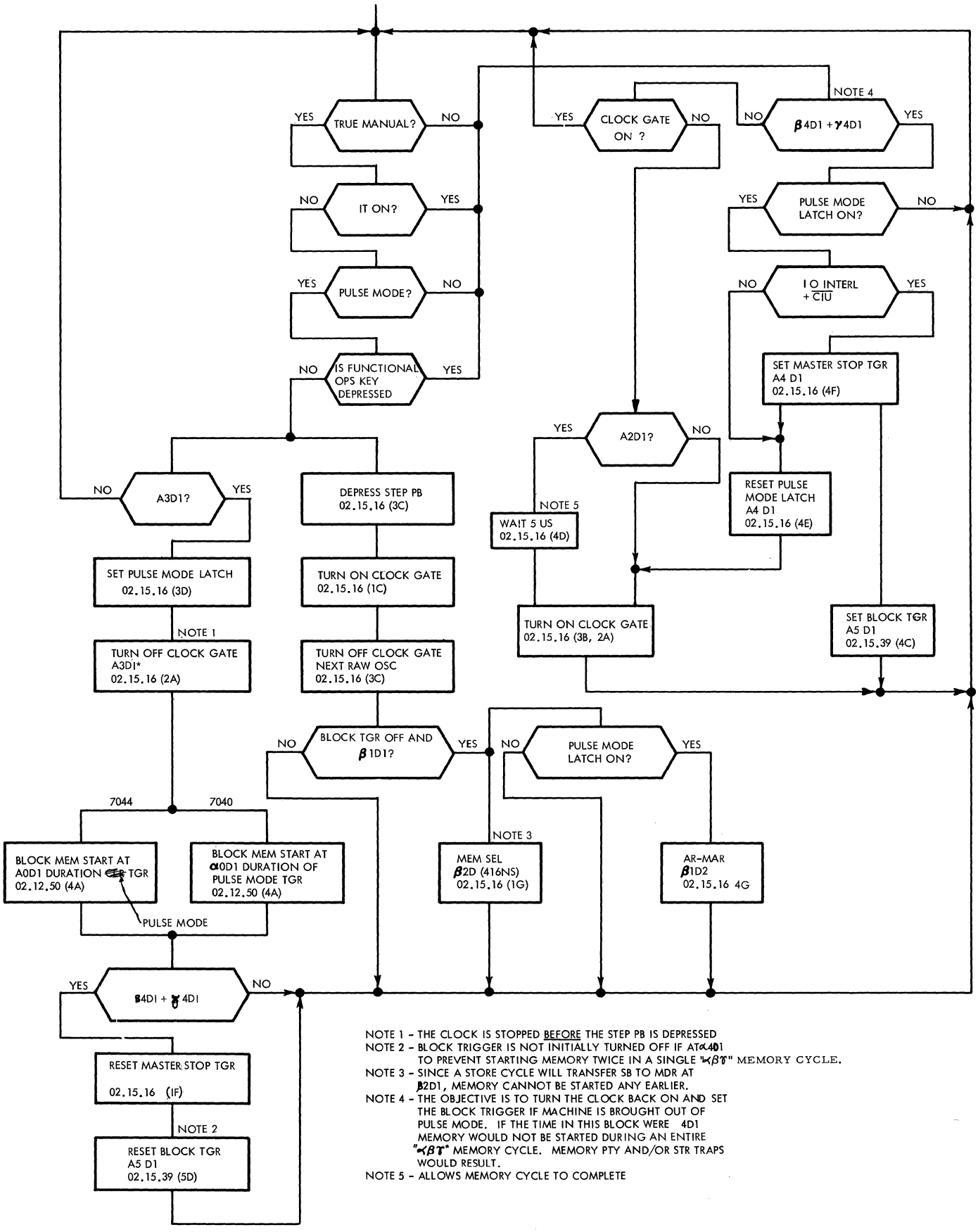
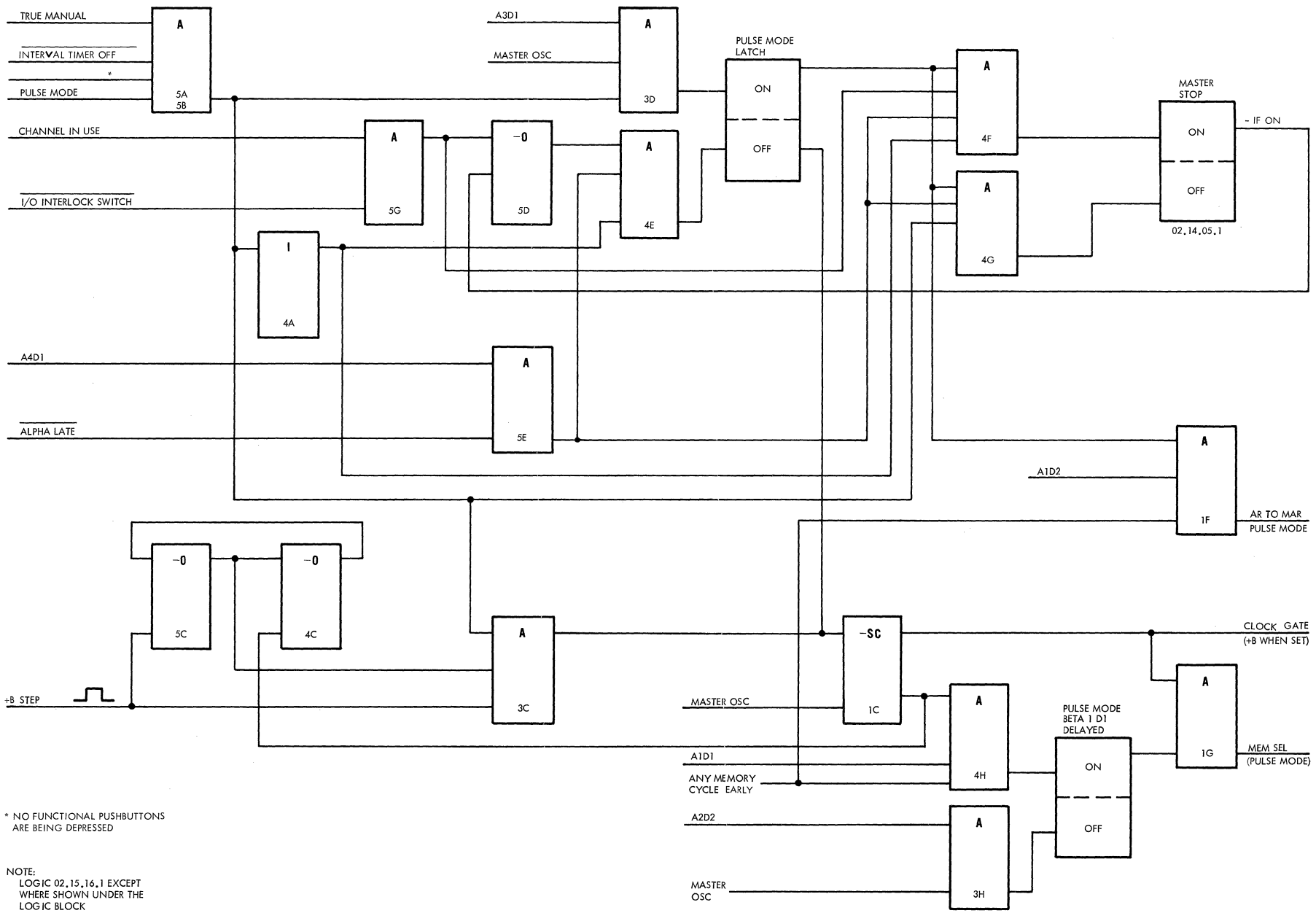


FIGURE 96. STEP-SINGLE CYCLE



NOTE 1 - THE CLOCK IS STOPPED BEFORE THE STEP PB IS DEPRESSED
NOTE 2 - BLOCK TRIGGER IS NOT INITIALLY TURNED OFF IF AT α4D1 TO PREVENT STARTING MEMORY TWICE IN A SINGLE "αβγ" MEMORY CYCLE.
NOTE 3 - SINCE A STORE CYCLE WILL TRANSFER SB TO MDR AT β2D1, MEMORY CANNOT BE STARTED ANY EARLIER.
NOTE 4 - THE OBJECTIVE IS TO TURN THE CLOCK BACK ON AND SET THE BLOCK TRIGGER IF MACHINE IS BROUGHT OUT OF PULSE MODE. IF THE TIME IN THIS BLOCK WERE 4D1 MEMORY WOULD NOT BE STARTED DURING AN ENTIRE "αβγ" MEMORY CYCLE. MEMORY PTY AND/OR STR TRAPS WOULD RESULT.
NOTE 5 - ALLOWS MEMORY CYCLE TO COMPLETE

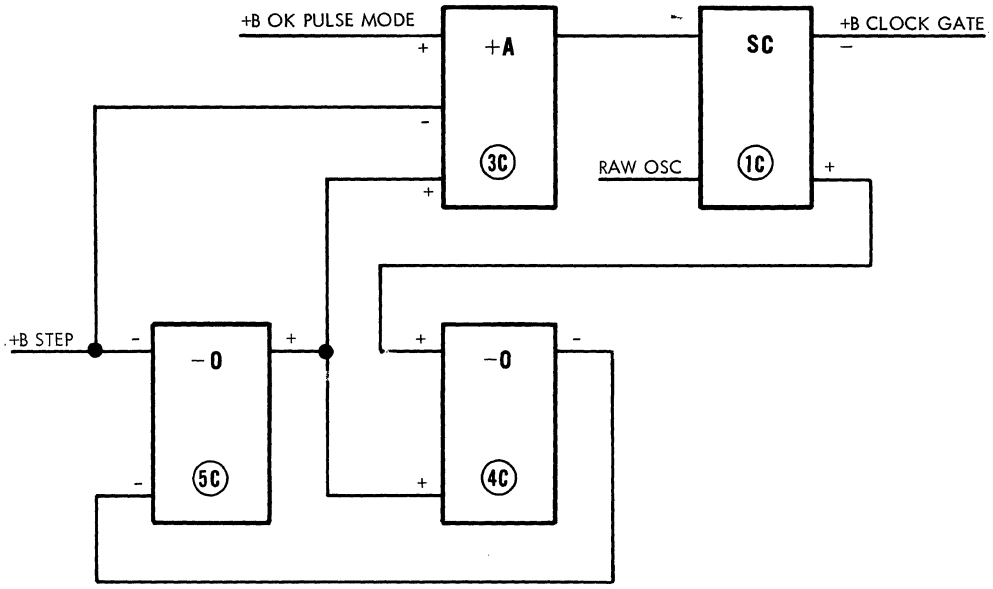
FIGURE 97. STEP SINGLE PULSE



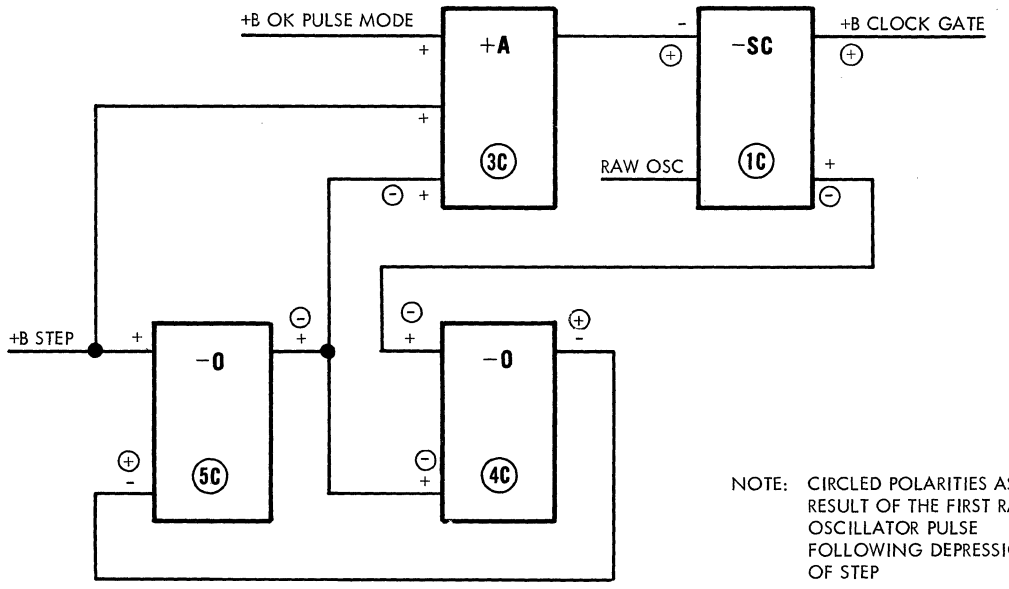
* NO FUNCTIONAL PUSHBUTTONS ARE BEING DEPRESSED

NOTE:
LOGIC 02.15.16.1 EXCEPT WHERE SHOWN UNDER THE LOGIC BLOCK

FIGURE 98. SINGLE PULSE OPERATION



A. CLOCK GATE BEFORE DEPRESSING STEP



NOTE: CIRCLED POLARITIES AS A RESULT OF THE FIRST RAW OSCILLATOR PULSE FOLLOWING DEPRESSION OF STEP

B. CLOCK GATE AFTER DEPRESSING STEP

FIGURE 99. CLOCK GATE CONTROL

APPENDIX A: TIMING CHARTS

Timing charts of the Basic Cycles, Trapping, and Manual (Console) Operations are included here. The figure number and names of these charts are as follows:

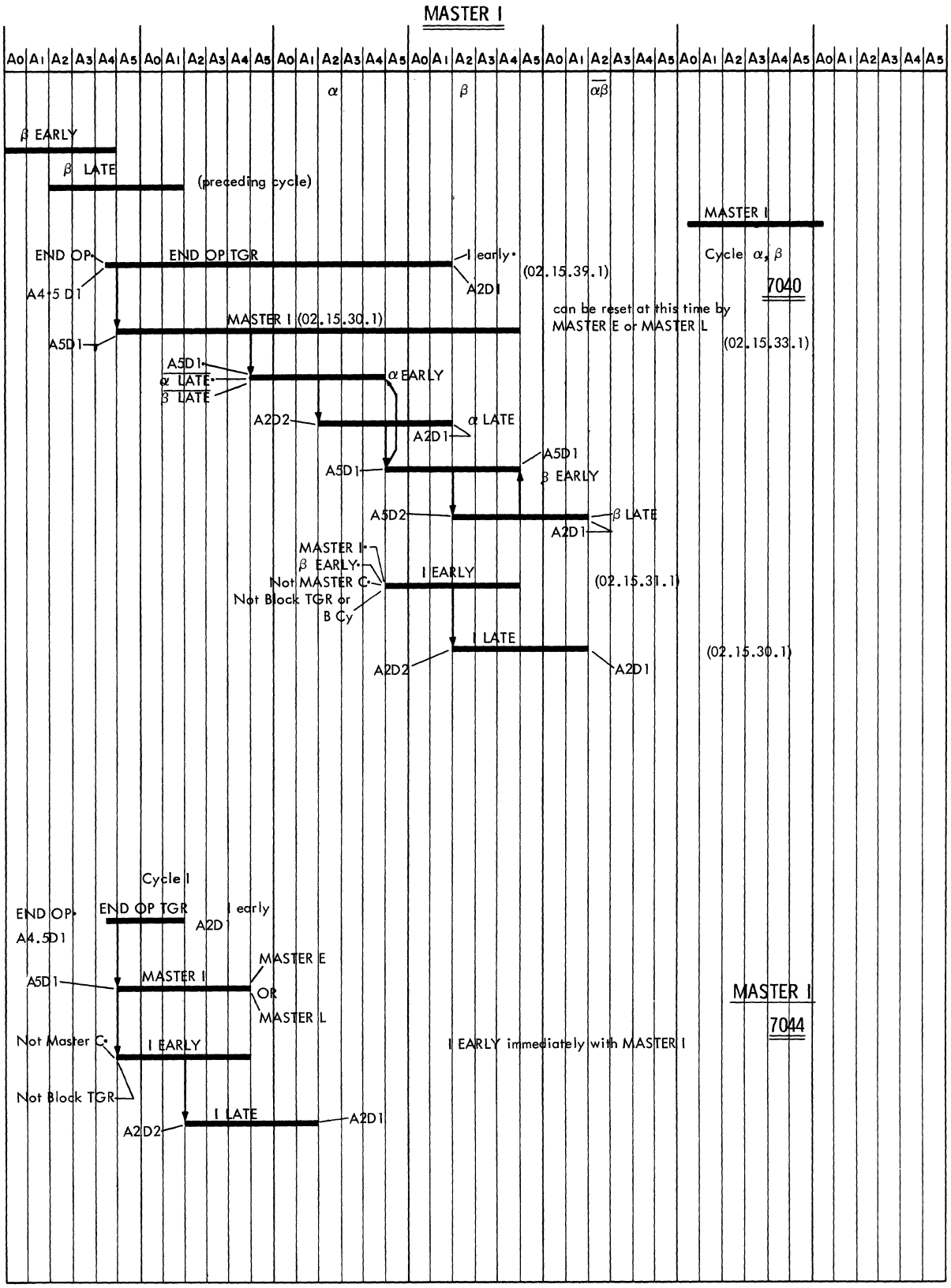


FIGURE A1. MASTER I (7044)

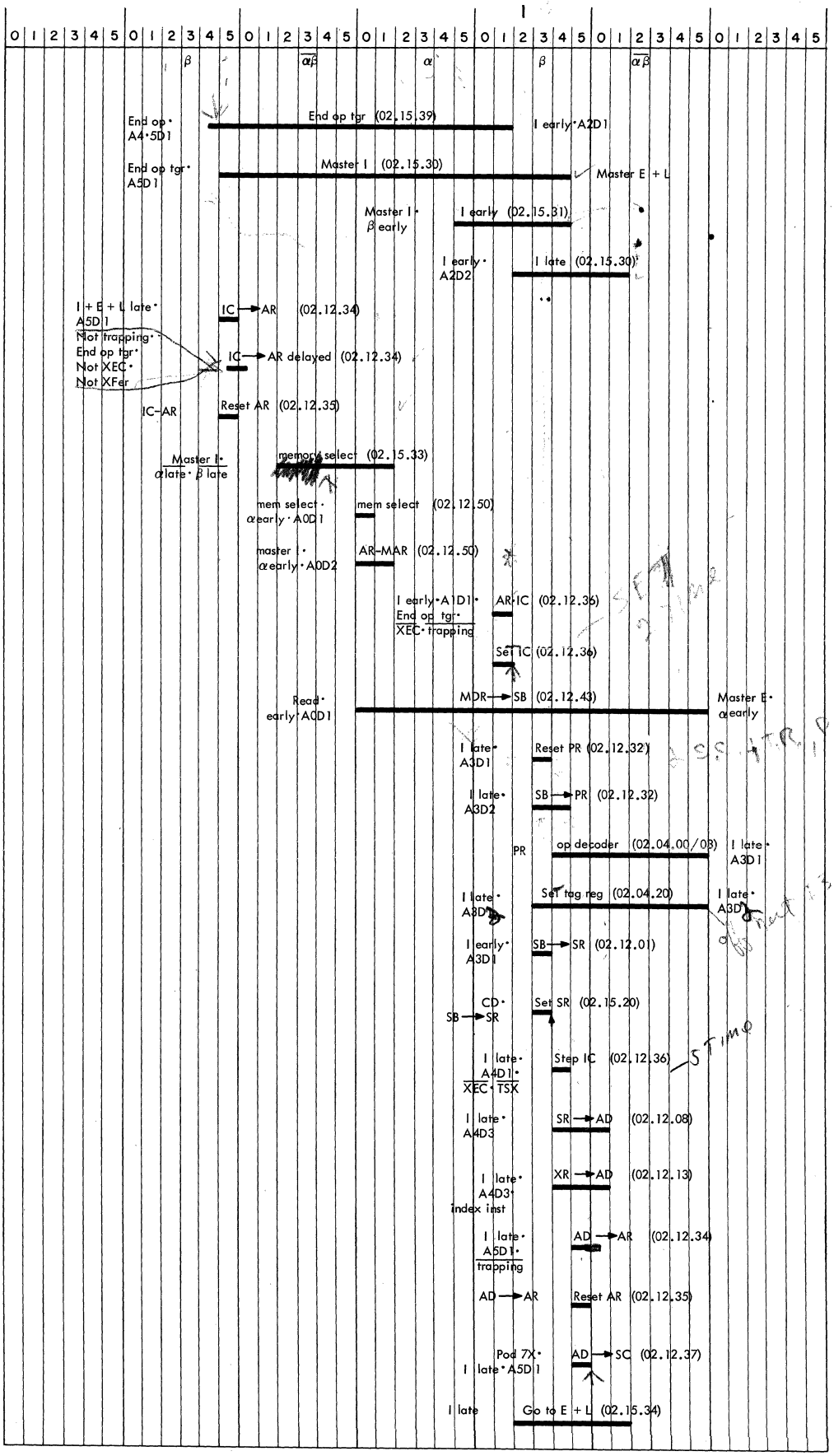


FIGURE A2. INSTRUCTION CYCLE

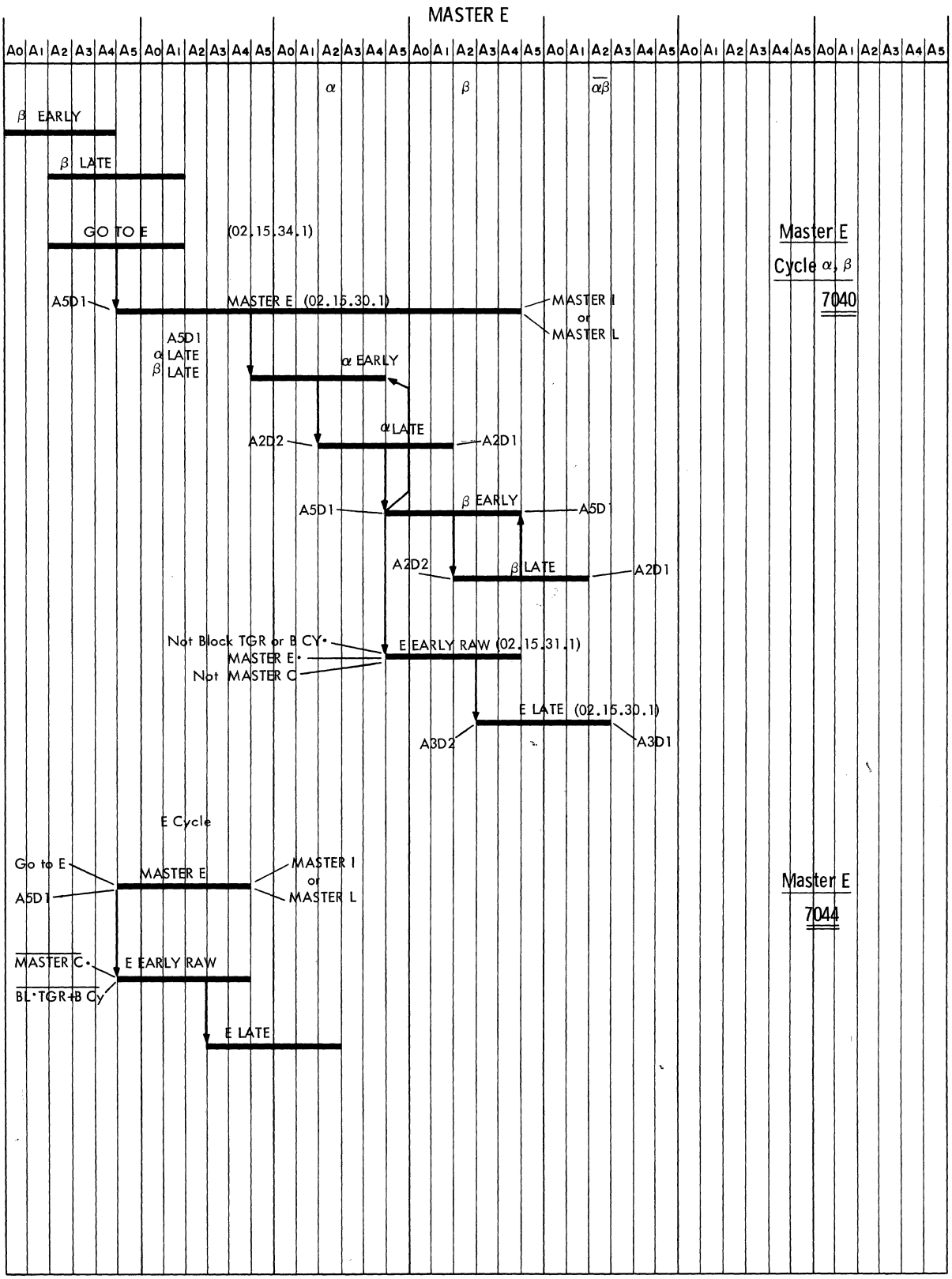


FIGURE A3. MASTER E (7044)

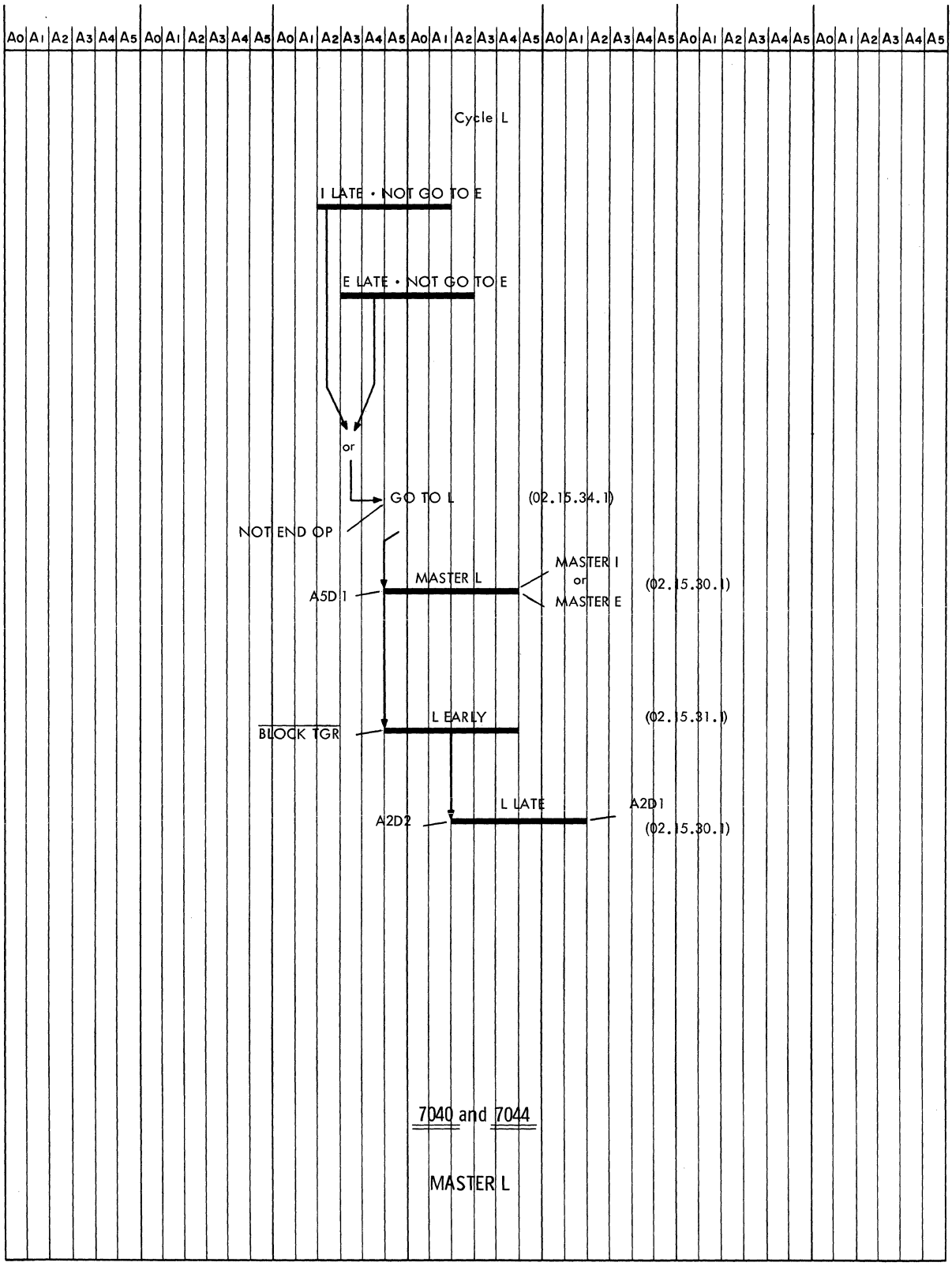


FIGURE A4. MASTER L (7040 And 7044)

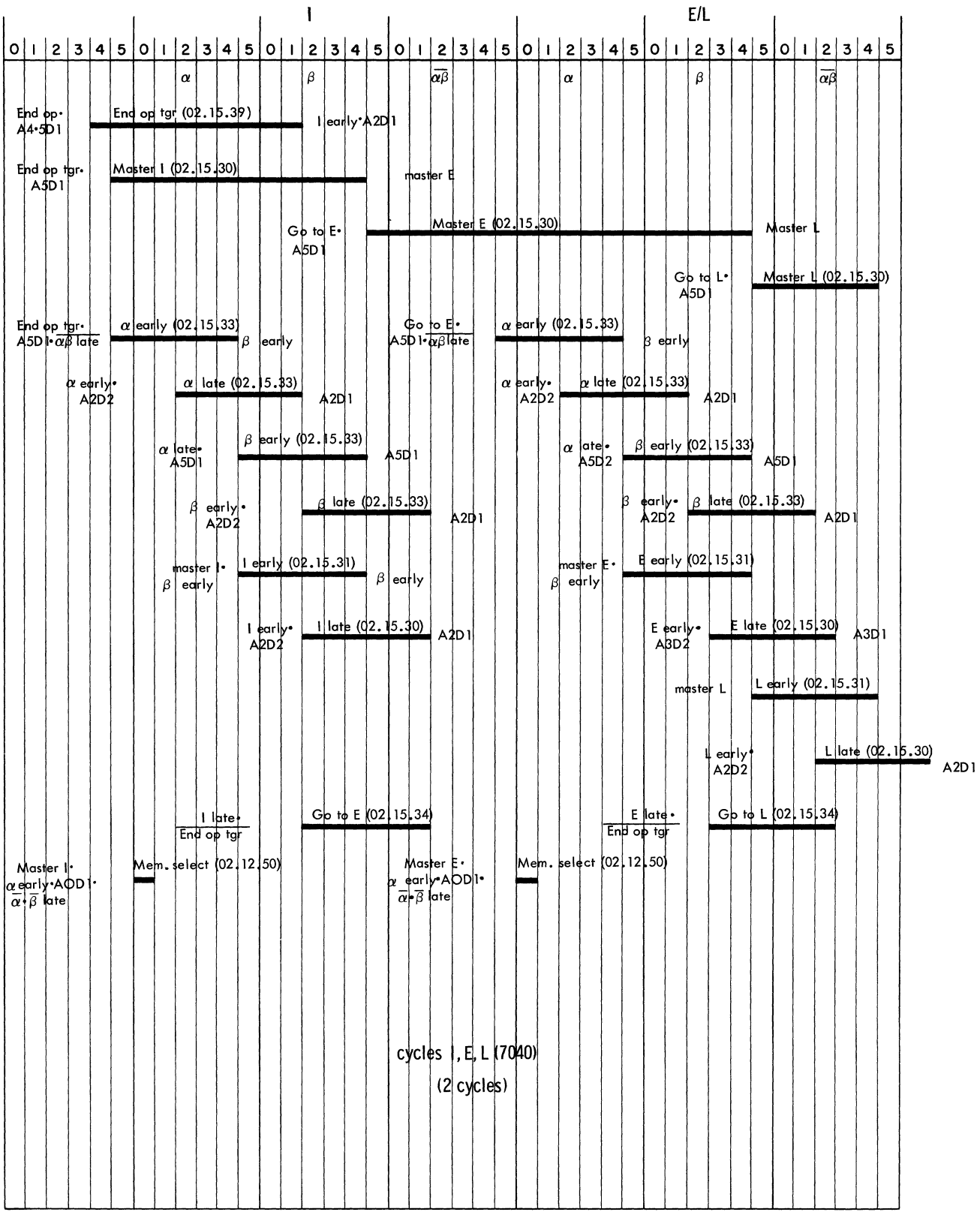


FIGURE A5. I,E,L CYCLES (7040-2 CYCLES)

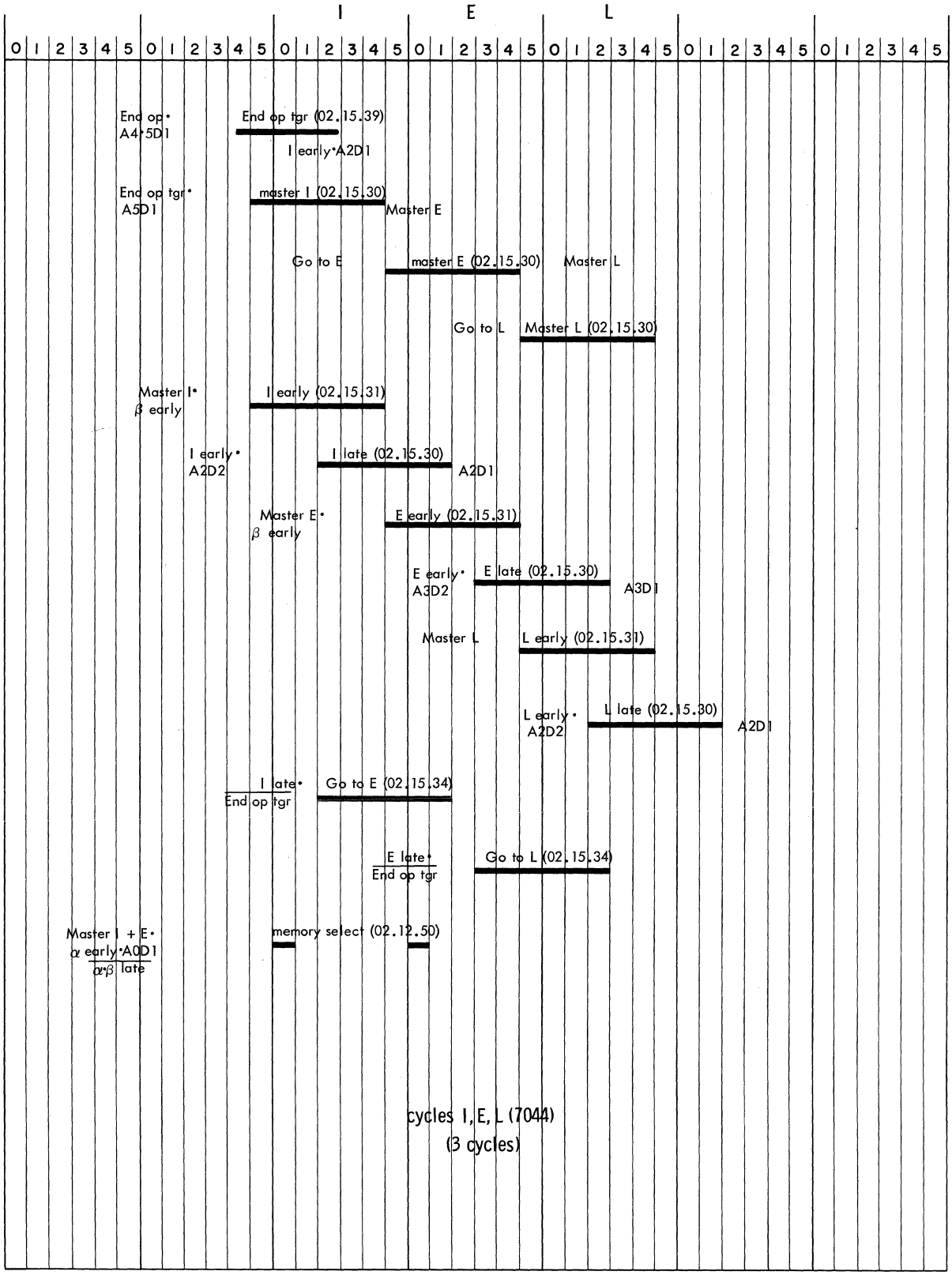


FIGURE A6. I, E, L CYCLES (7044-3 CYCLES)

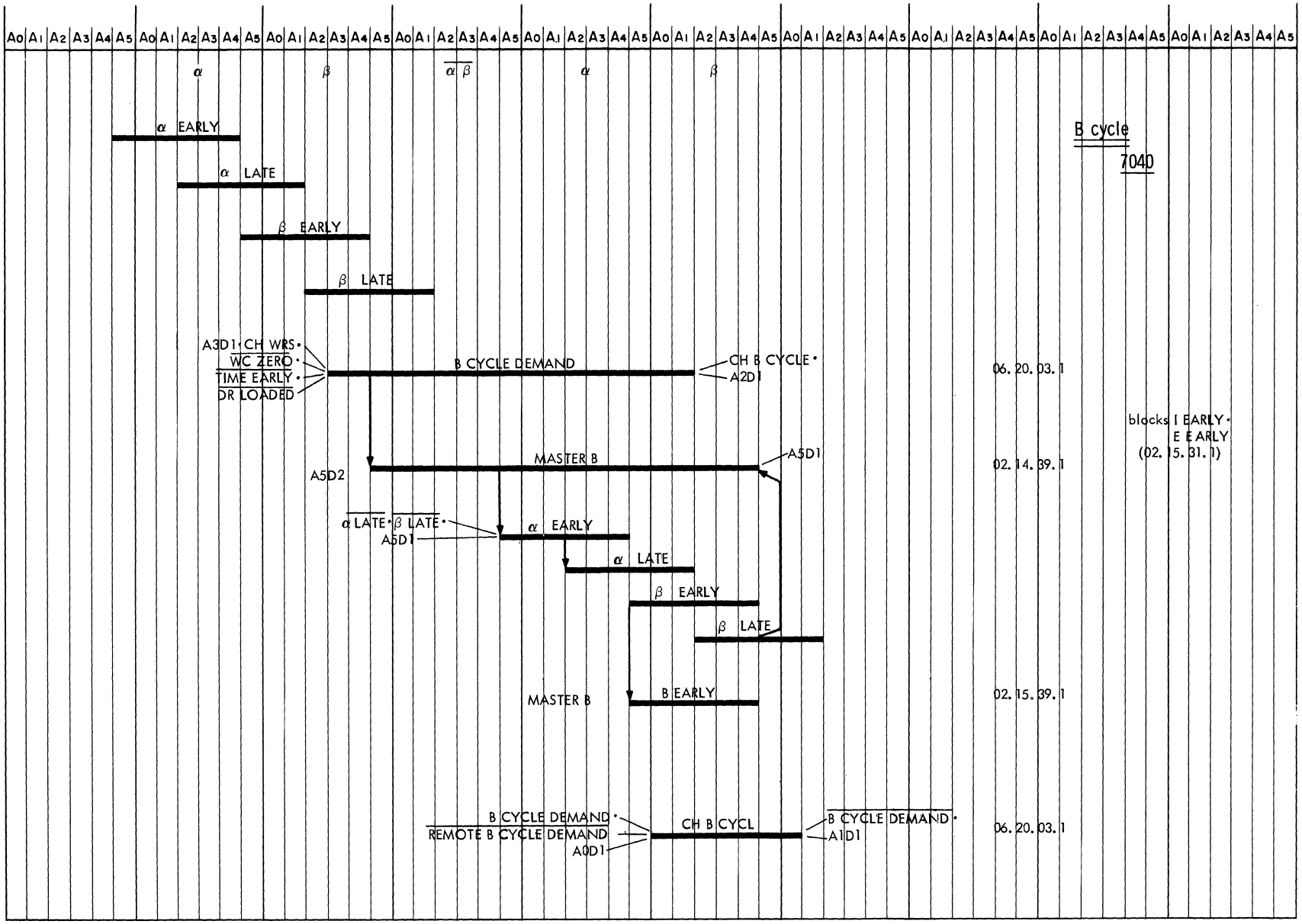


FIGURE A7. B CYCLE

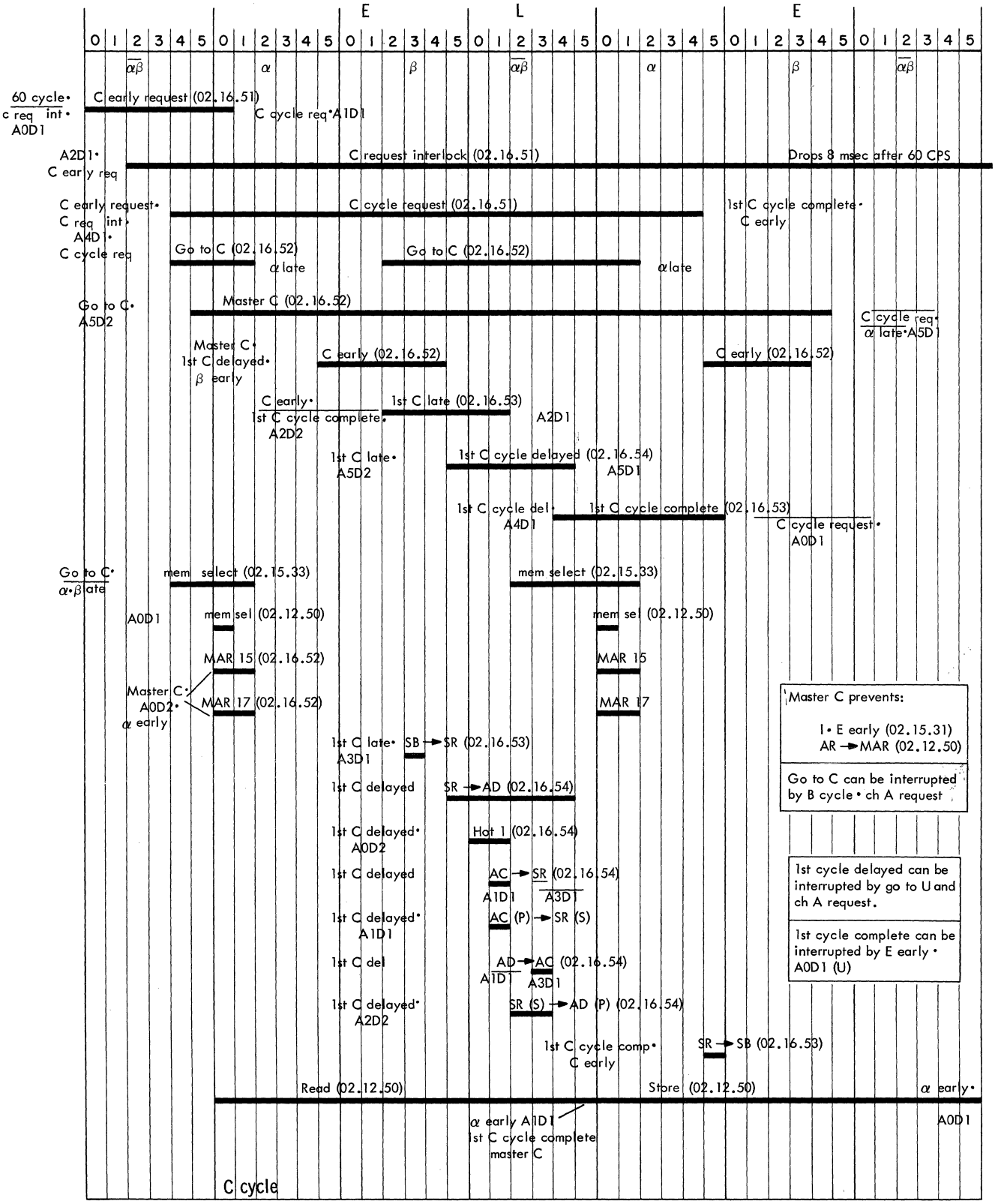


FIGURE A8. C CYCLE

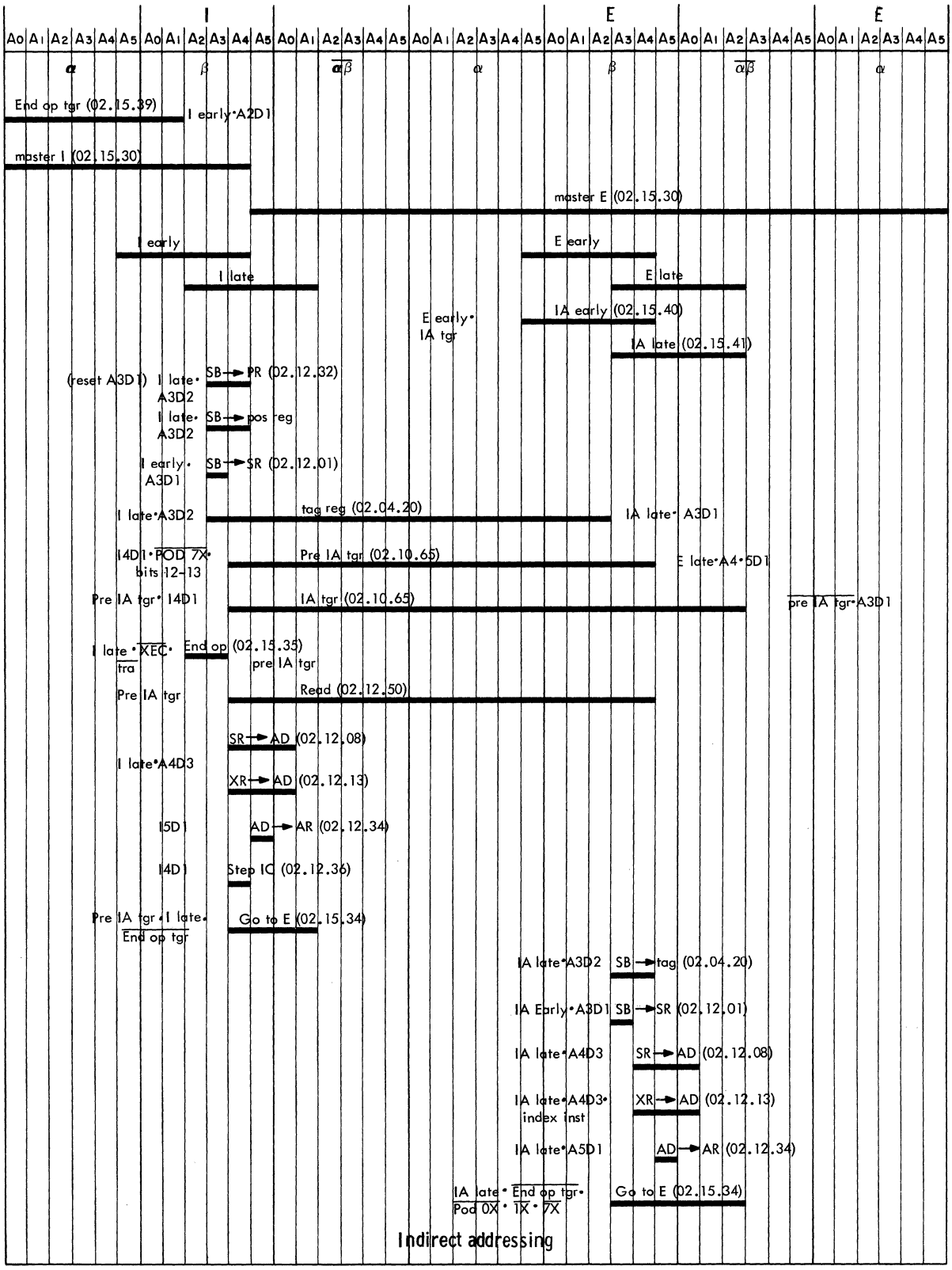


FIGURE A9. INDIRECT ADDRESSING

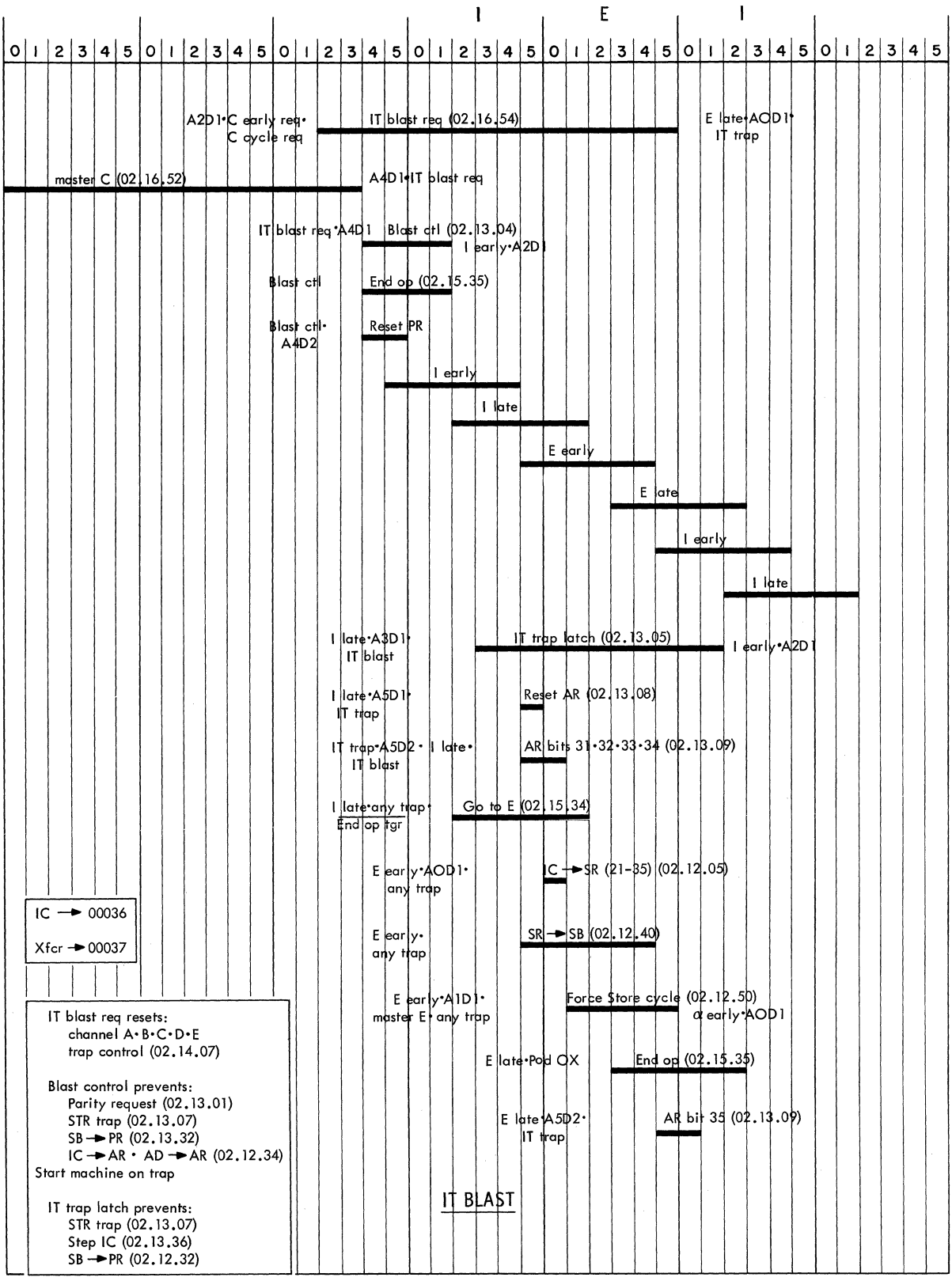


FIGURE A10. IT BLAST TRAP

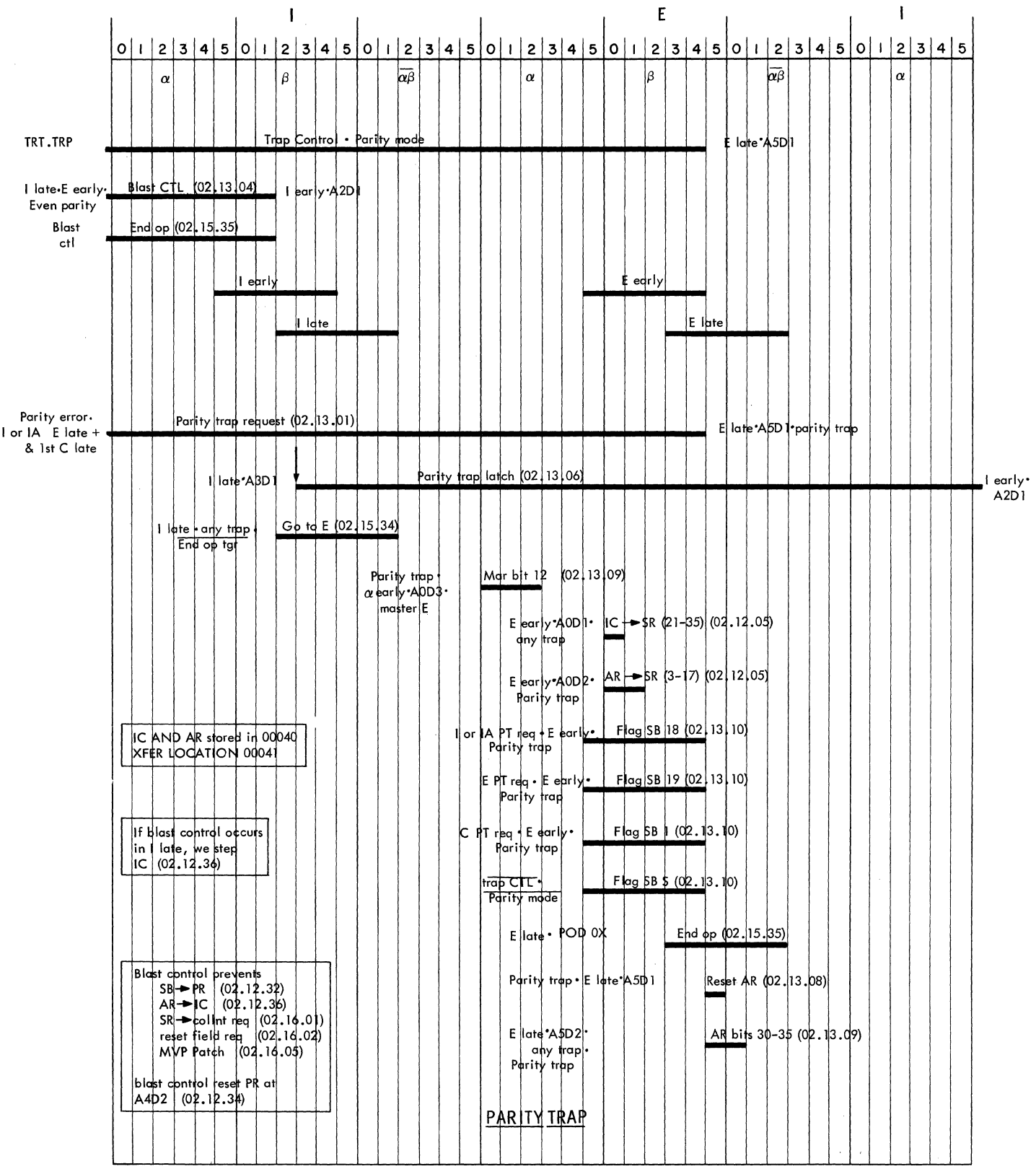


FIGURE A11. PARITY TRAP

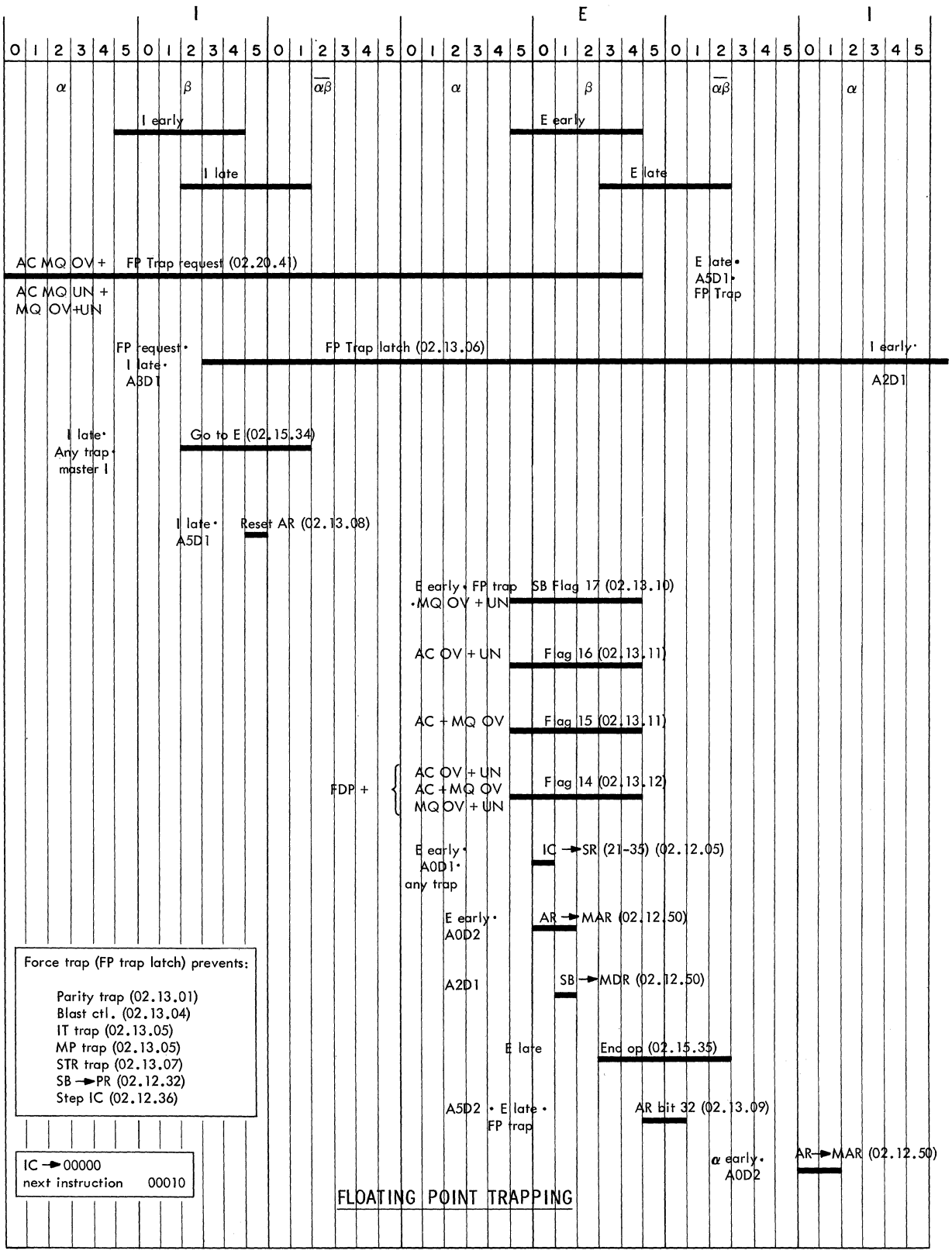


FIGURE A12. FLOATING POINT TRAPPING

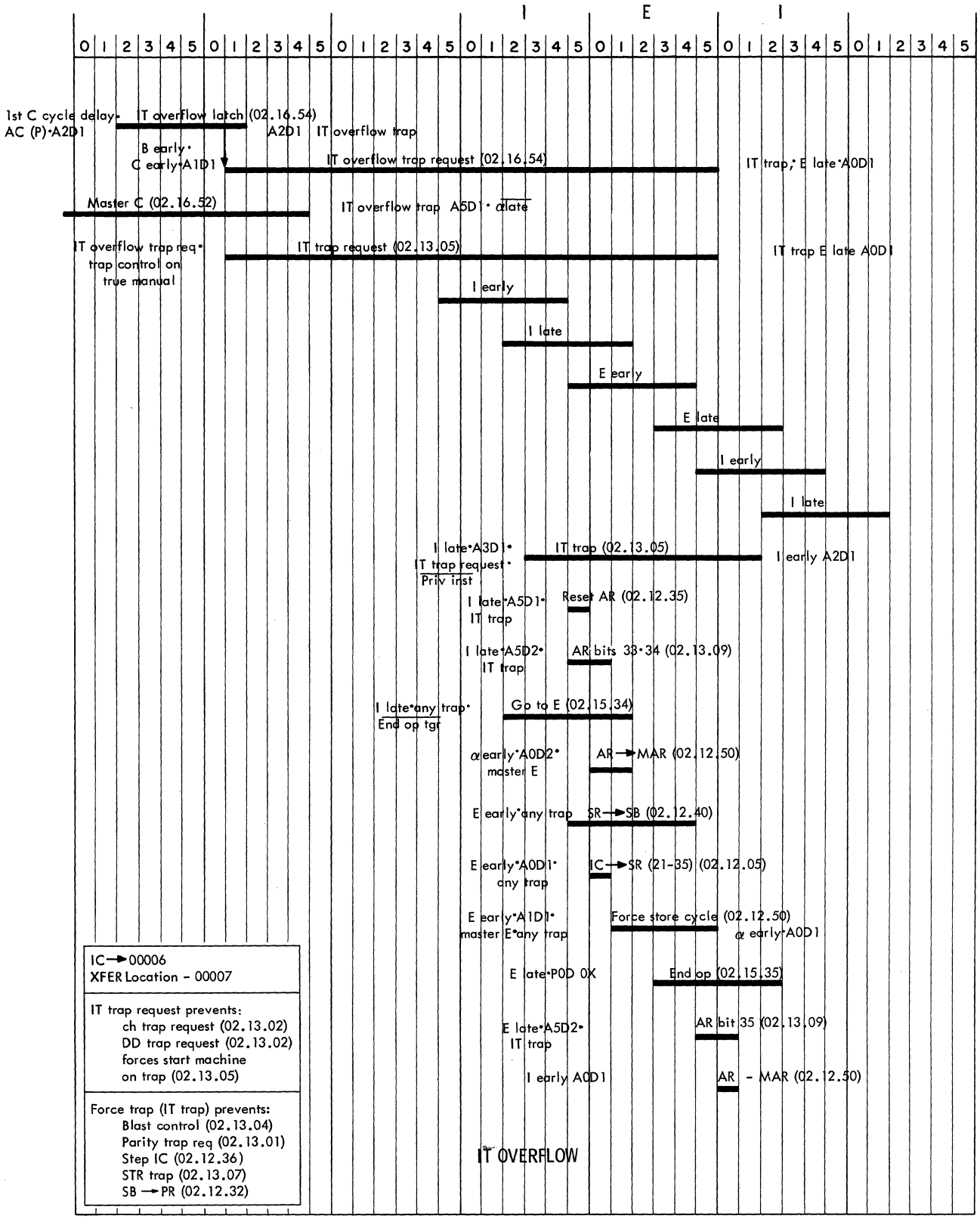


FIGURE A13. IT OVERFLOW TRAP

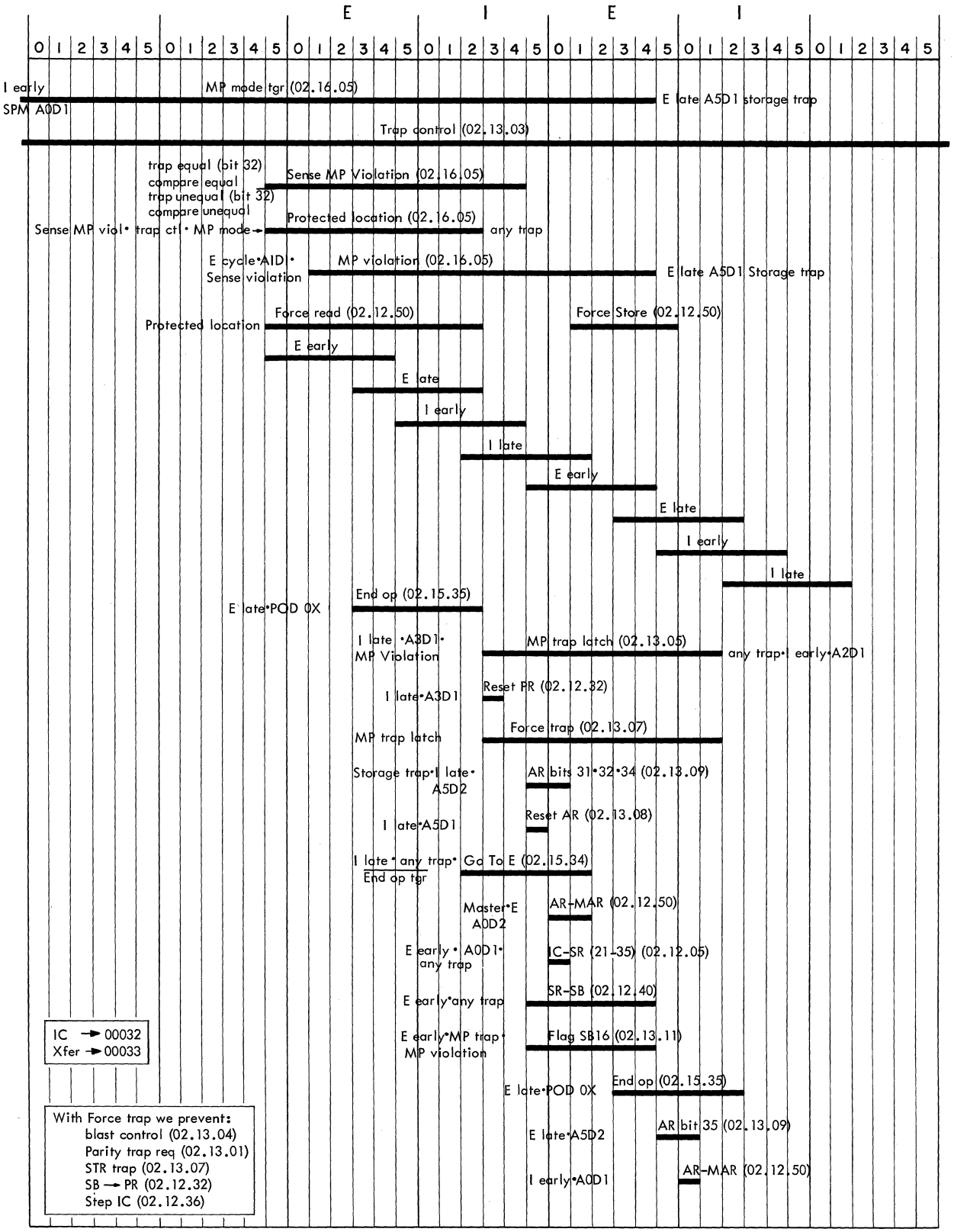


FIGURE A14. MEMORY PROTECT VIOLATION

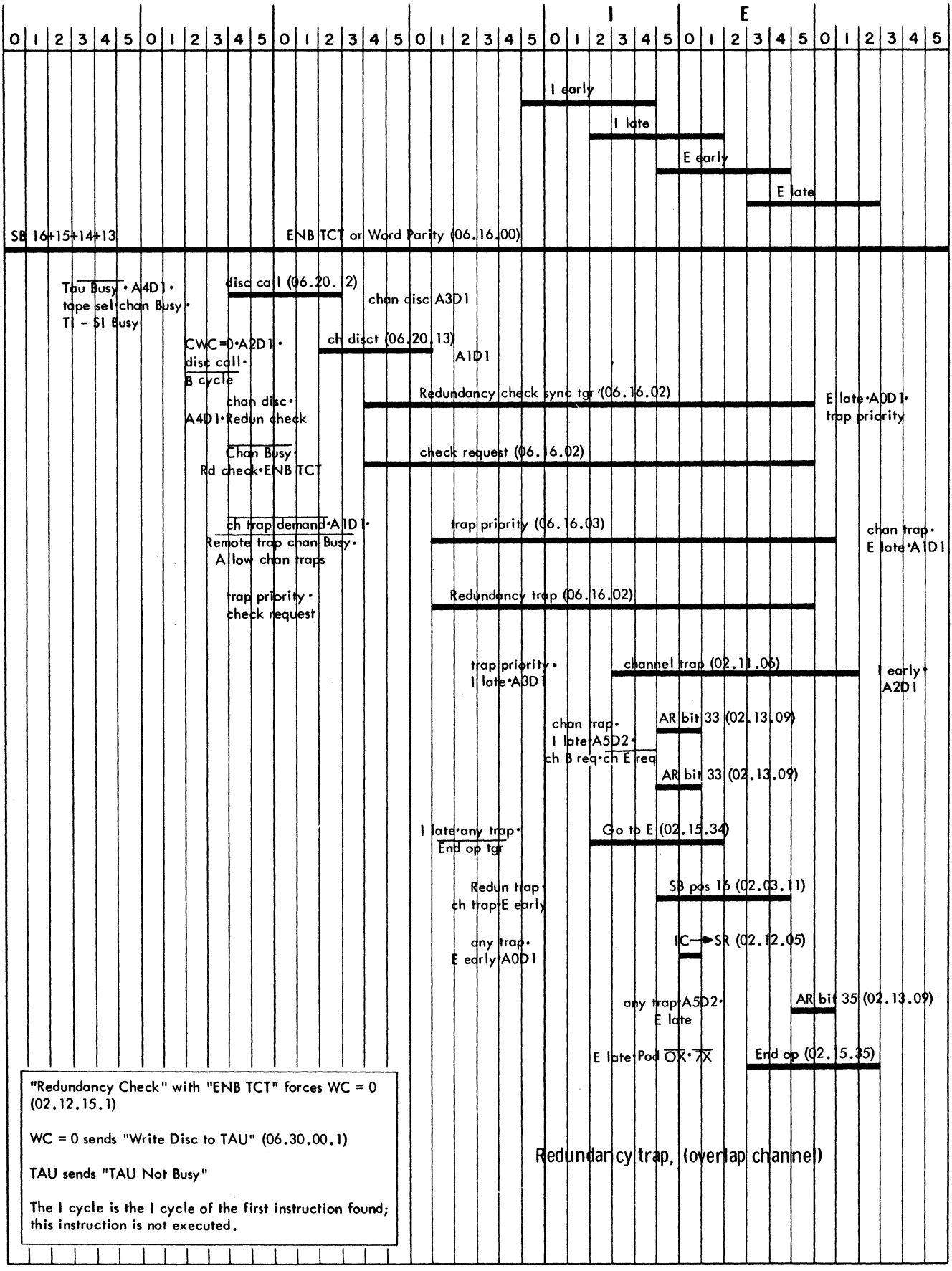


FIGURE A15. REDUNDANCY TRAP (OVERLAP CHANNEL)

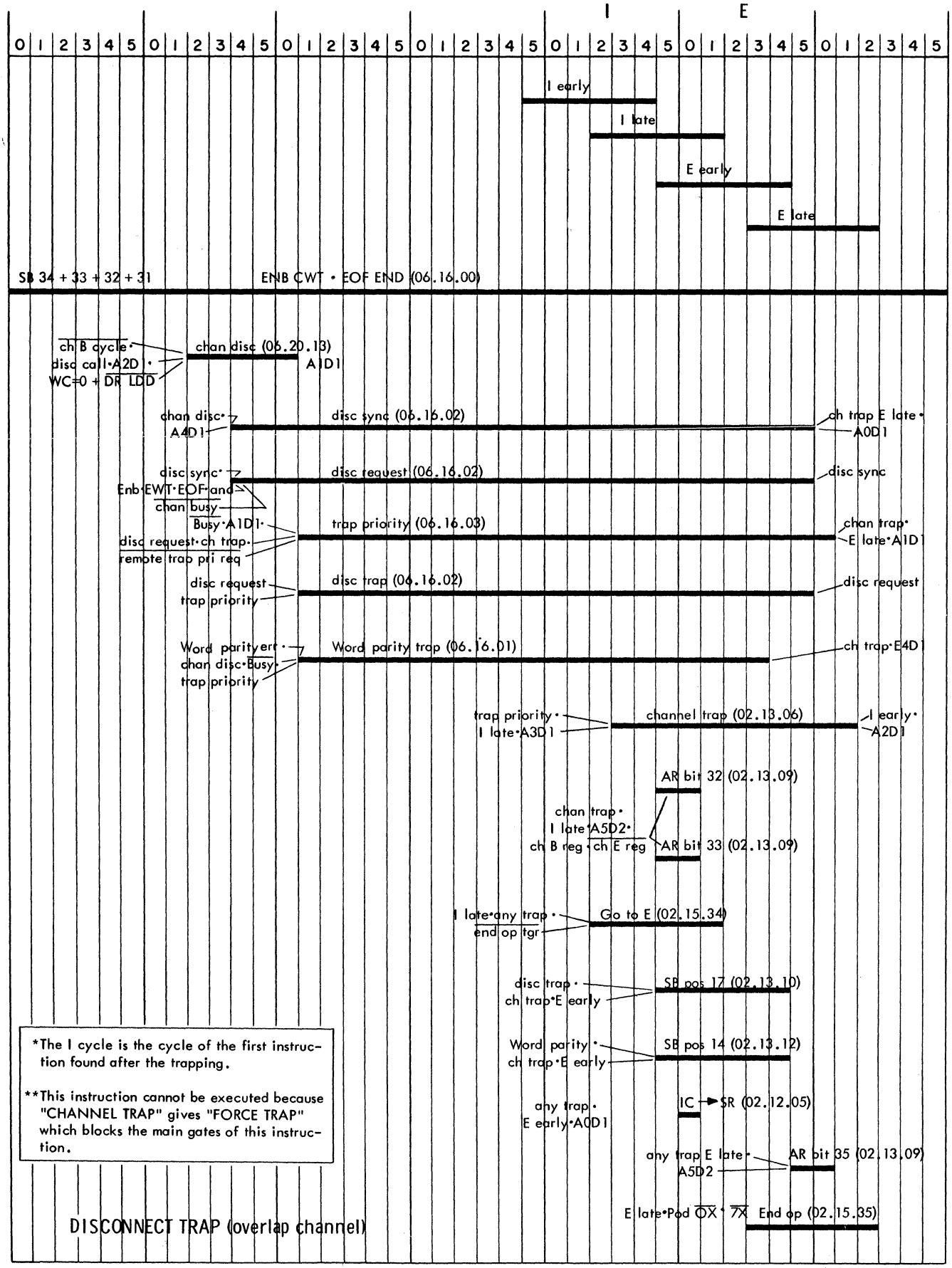


FIGURE A16. DISCONNECT TRAP (OVERLAP CHANNEL)

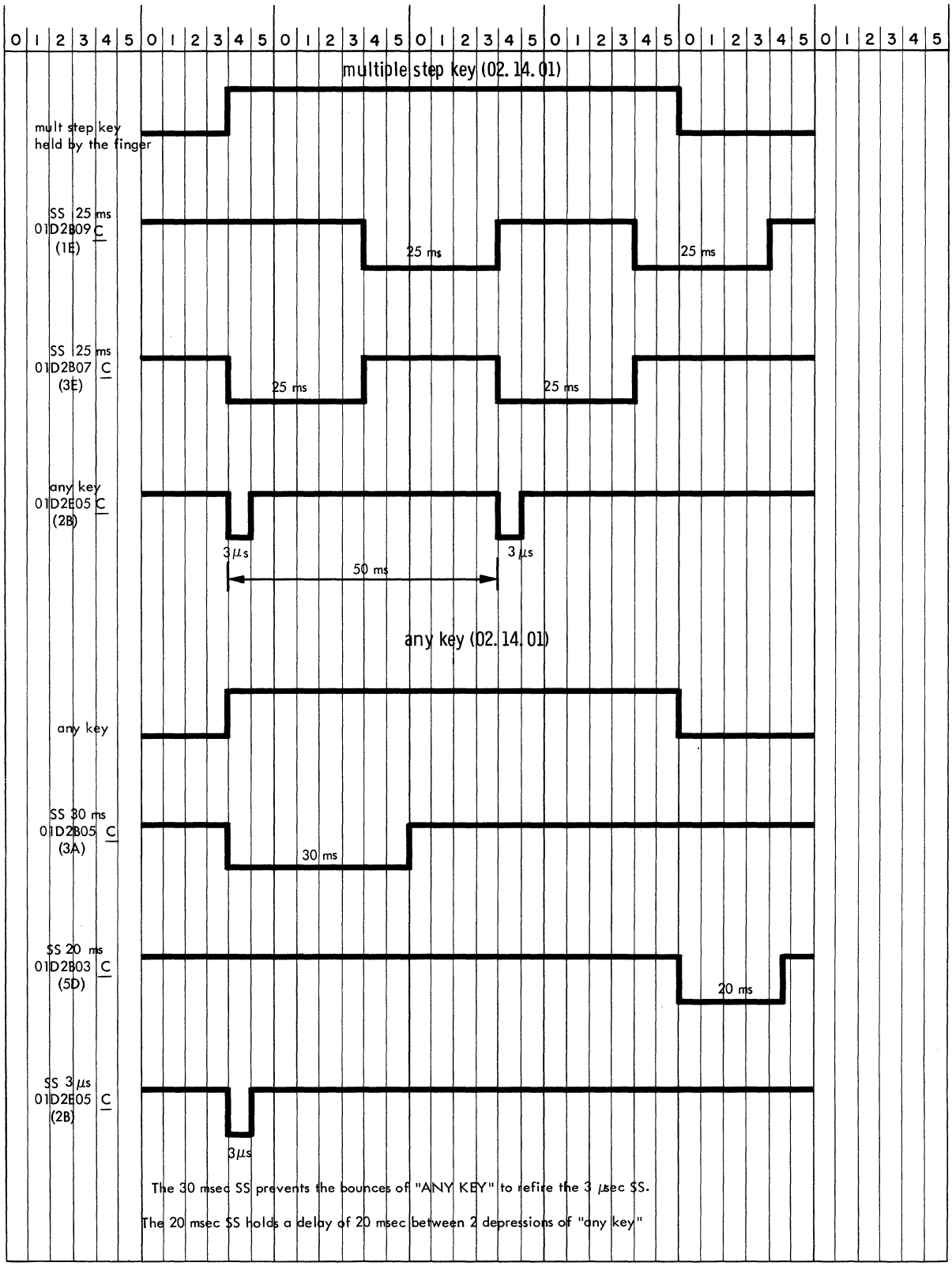


FIGURE A17. ANY KEY AND MULTIPLE STEP KEY PULSE GENERATION

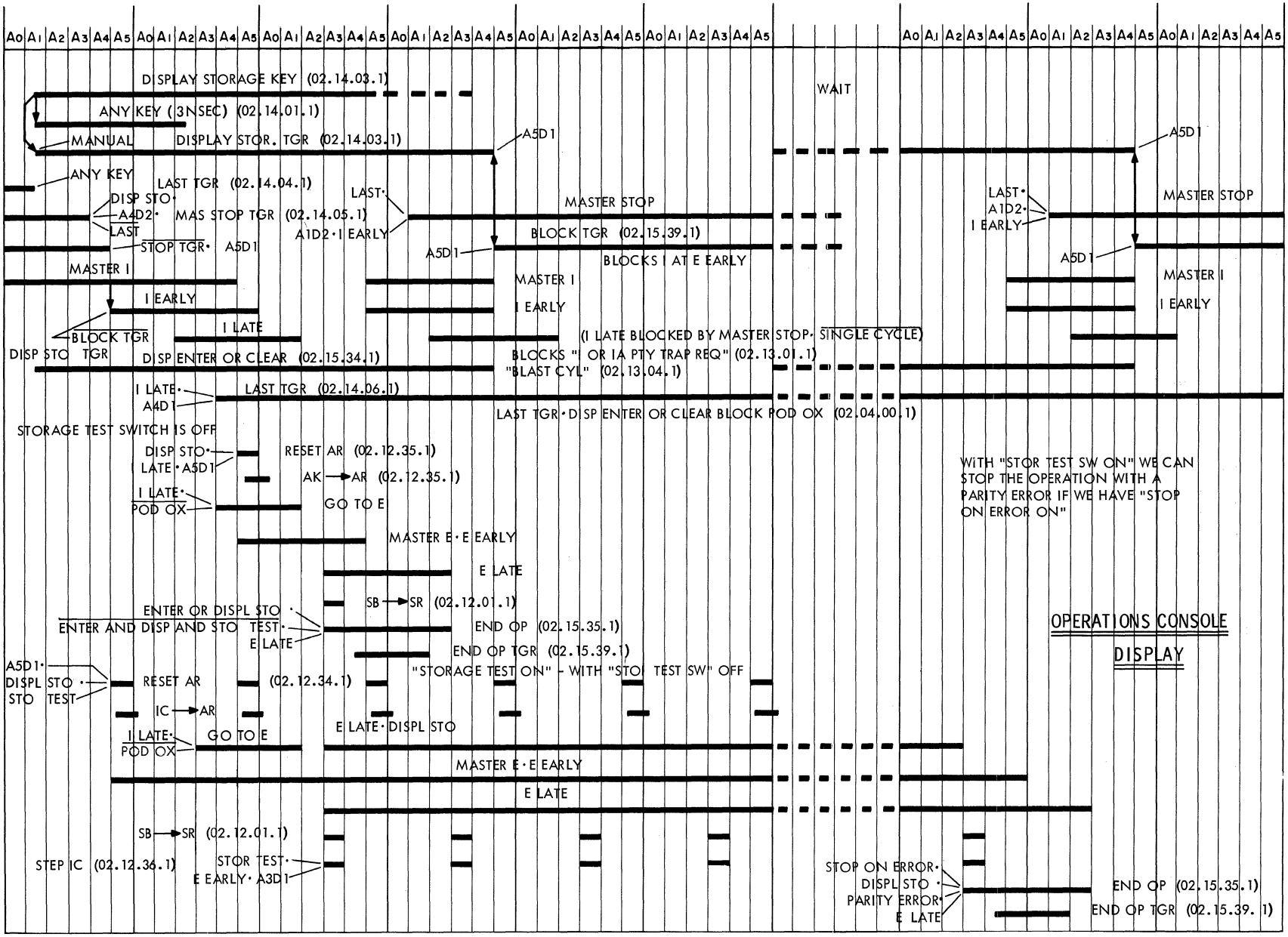


FIGURE A18. DISPLAY STORAGE

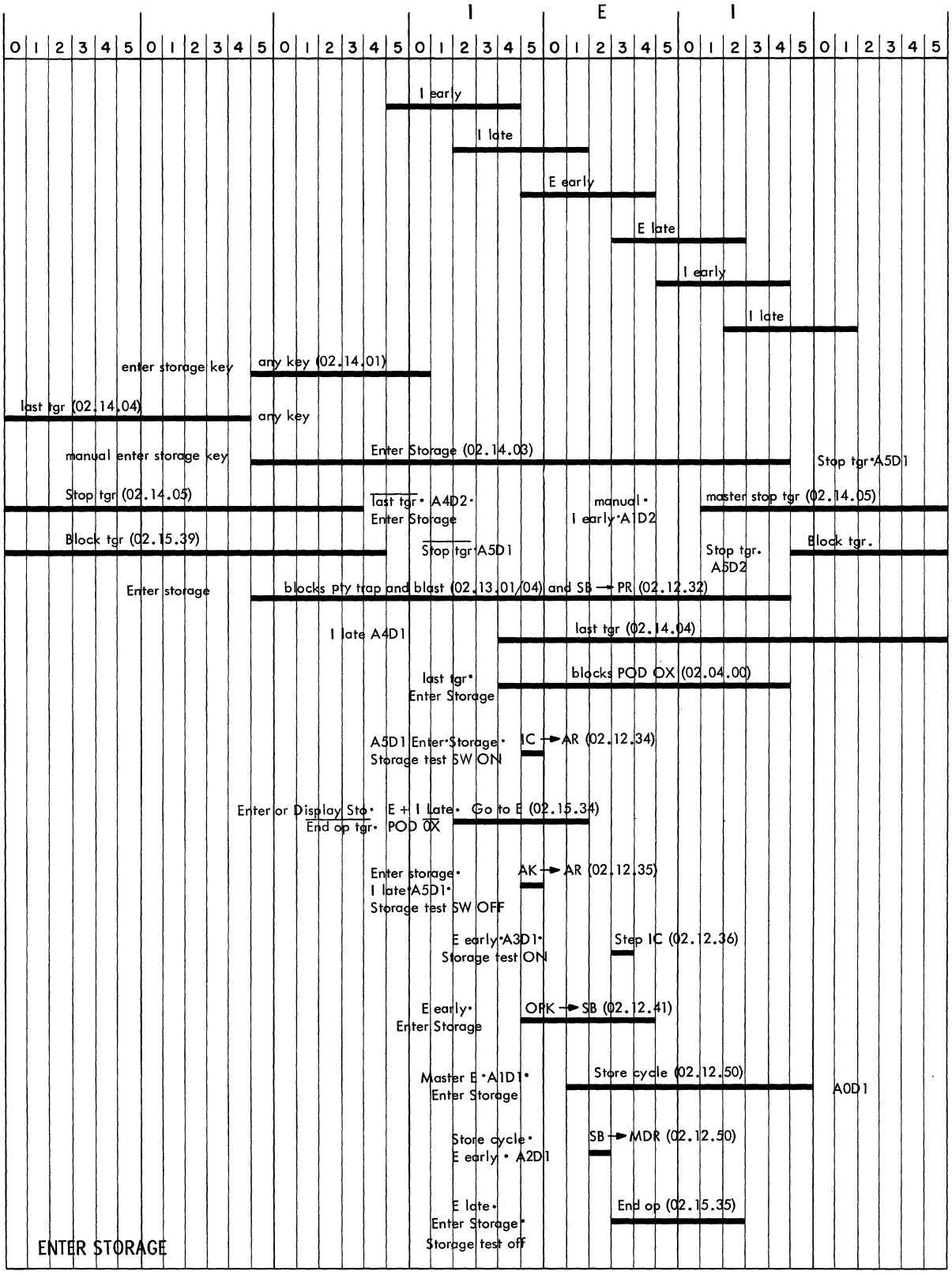


FIGURE A19. ENTER STORAGE

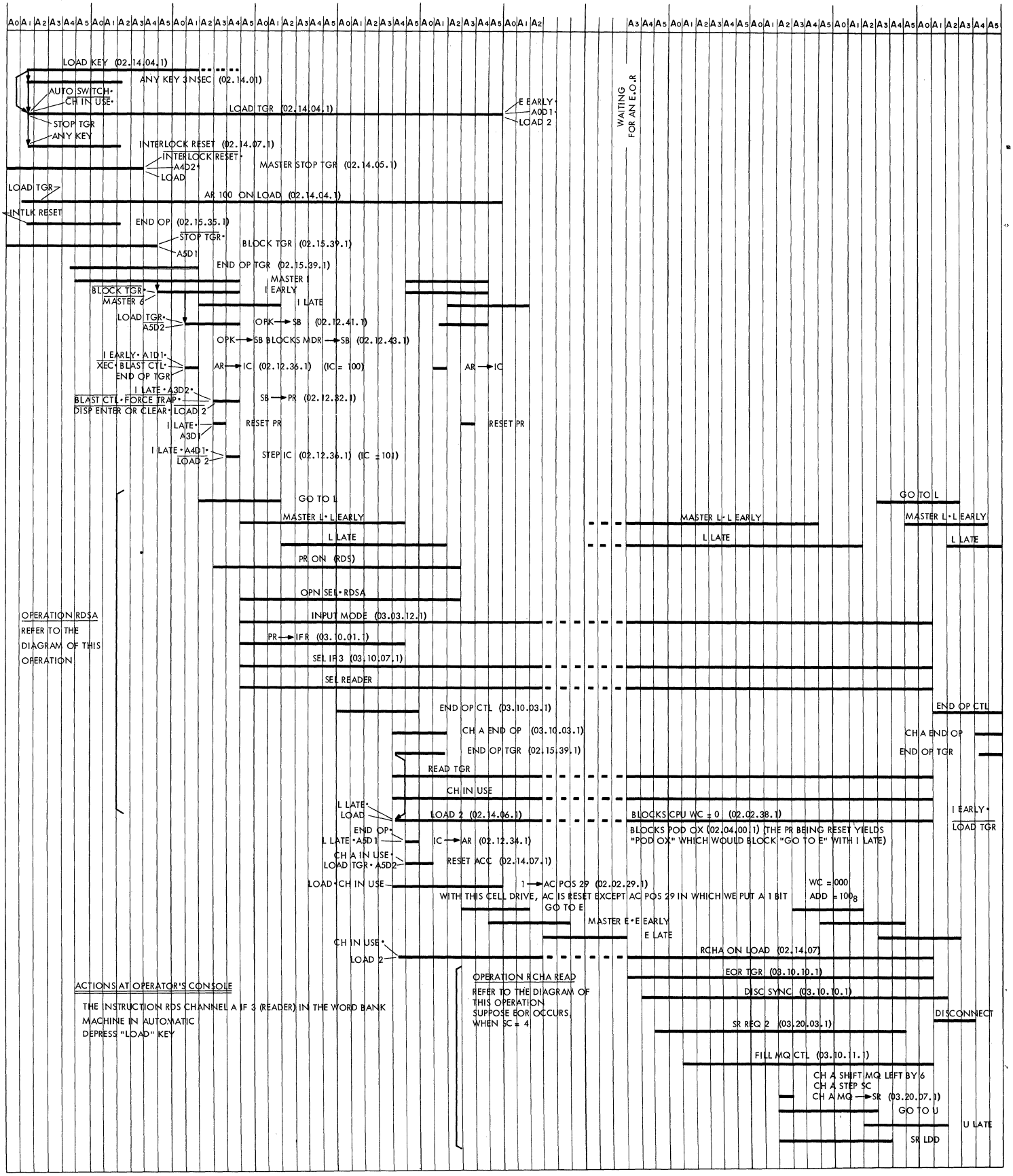


FIGURE A20. LOAD KEY OPERATION

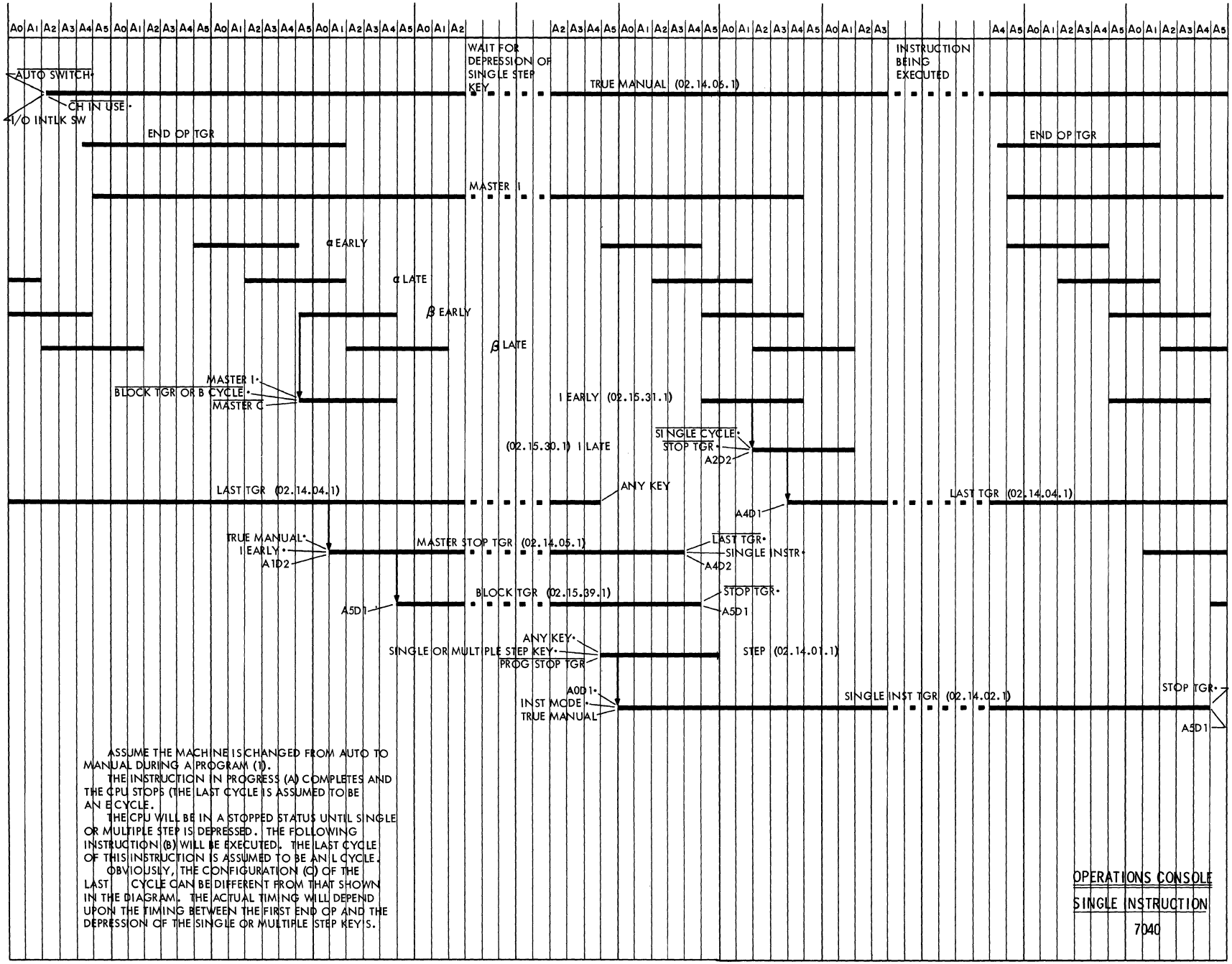


FIGURE A21. SINGLE INSTRUCTION

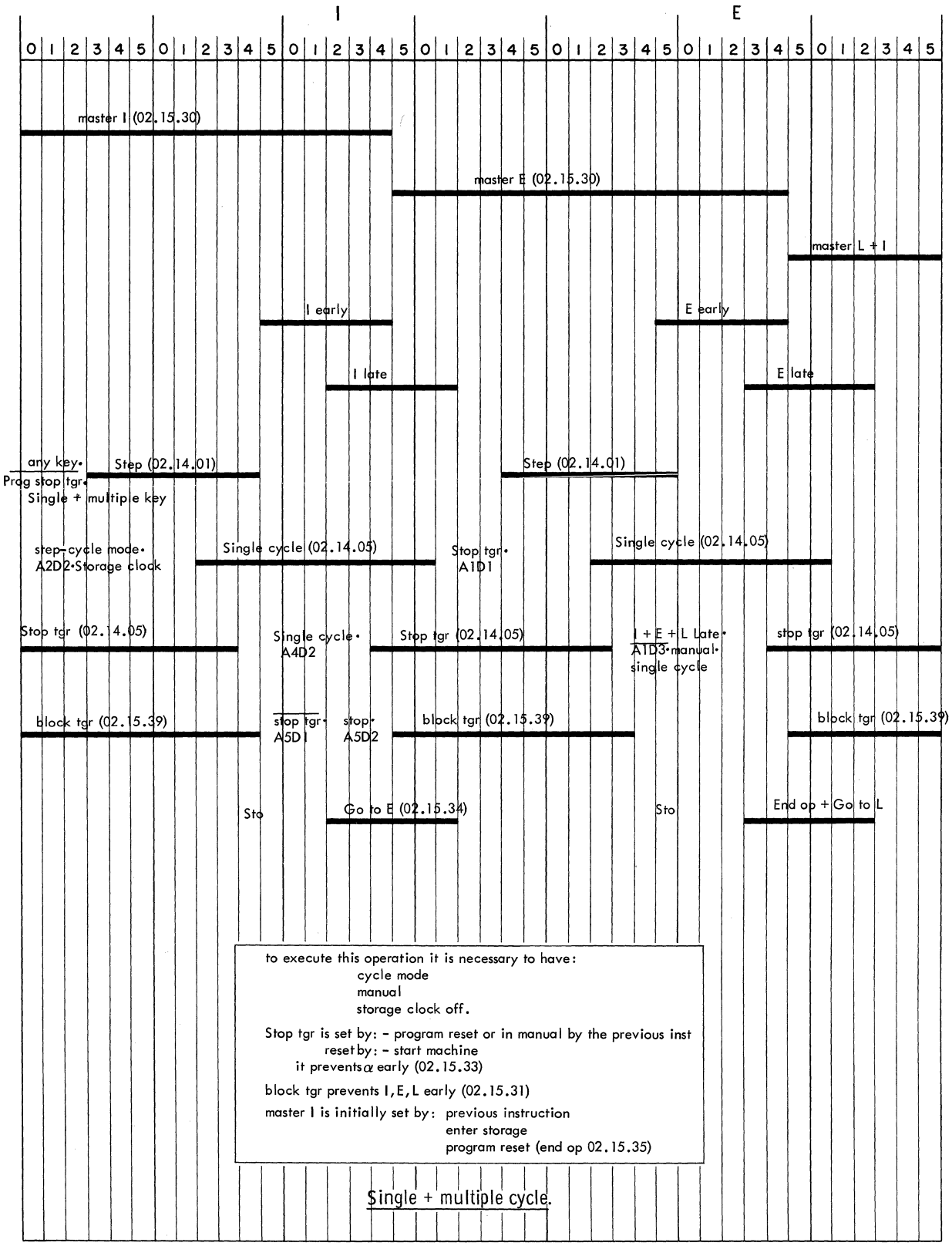
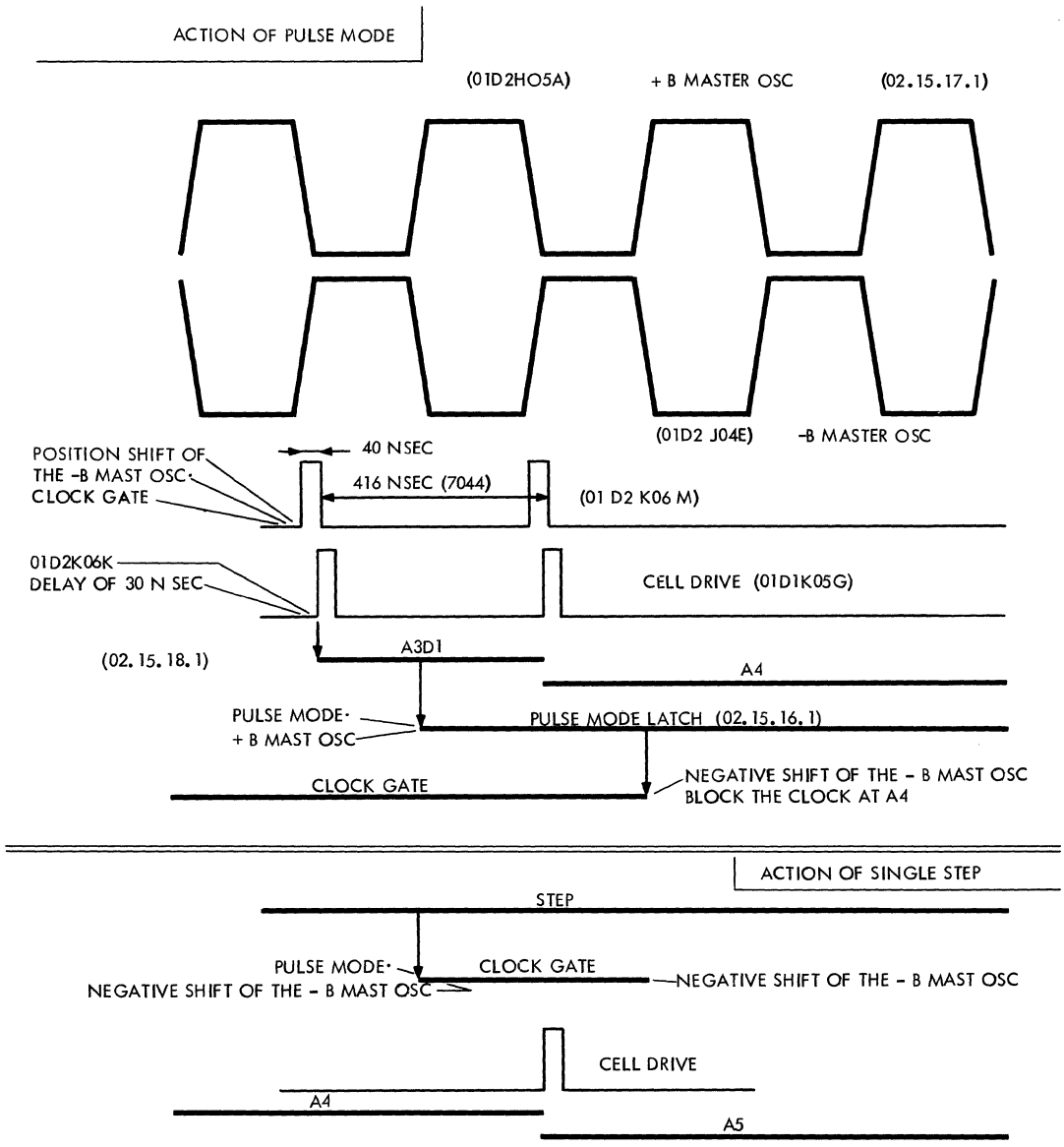
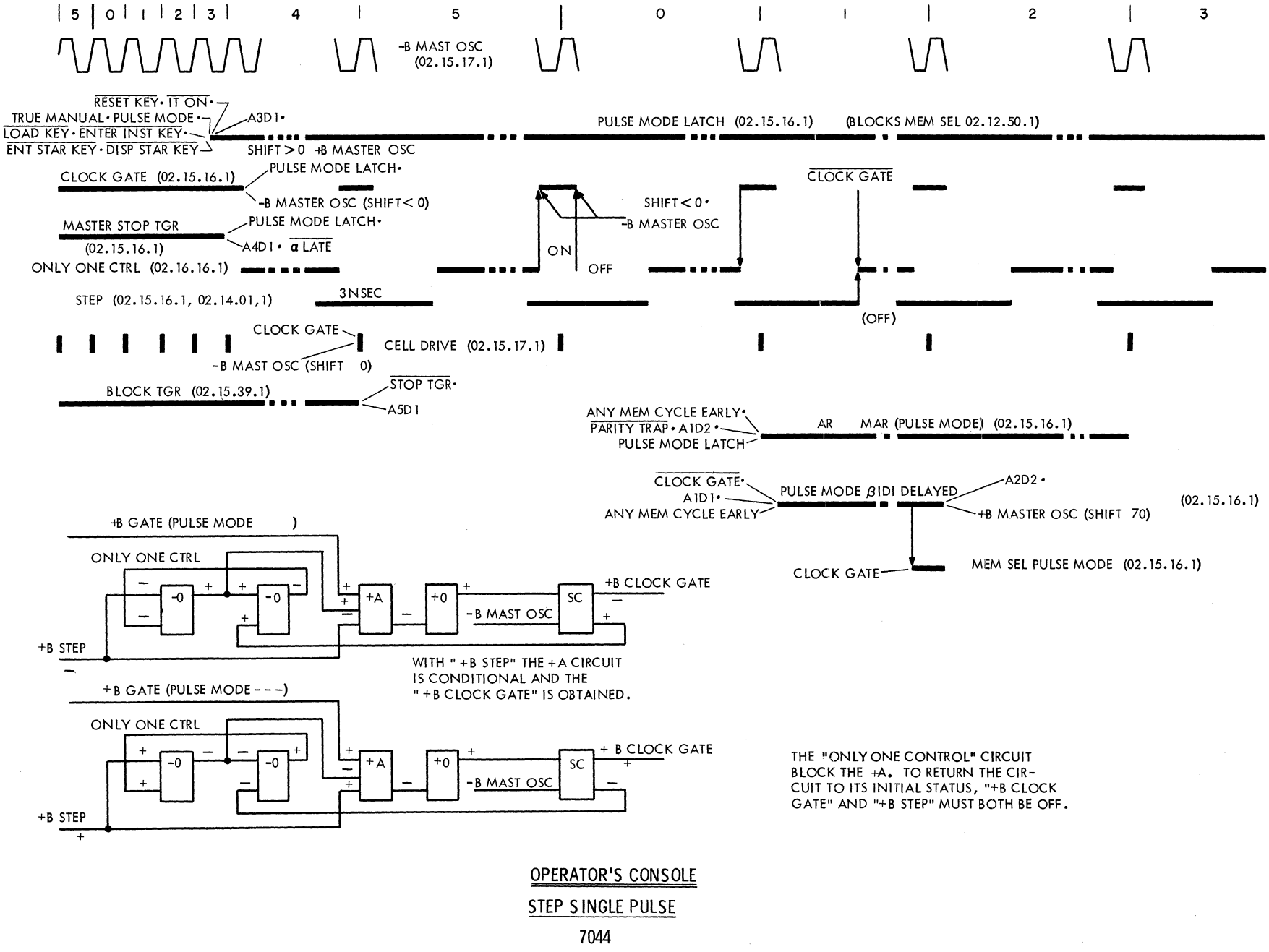


FIGURE A22. SINGLE + MULTIPLE CYCLE



SINGLE PULSE MODE CONTROL

FIGURE A23. SINGLE PULSE MODE CONTROL



OPERATOR'S CONSOLE

STEP SINGLE PULSE

7044

FIGURE A24. STEP SINGLE PULSE (7044)

CUT ALONG LINE

COMMENT SHEET

IBM 7040-7044 CENTRAL PROCESSING UNIT

CUSTOMER ENGINEERING MANUAL OF INSTRUCTION, R23-2651

FROM

NAME _____

OFFICE NO. _____

FOLD FOLD

CHECK ONE OF THE COMMENTS AND EXPLAIN IN THE SPACE PROVIDED

☐ SUGGESTED ADDITION (PAGE , TIMING CHART, DRAWING, PROCEDURE, ETC.)

☐ SUGGESTED DELETION (PAGE)

☐ ERROR (PAGE)

EXPLANATION

FOLD FOLD

NO POSTAGE NECESSARY IF MAILED IN U.S.A.
FOLD ON TWO LINES, STAPLE, AND MAIL

STAPLE

FOLD FOLD

FIRST CLASS
PERMIT NO. 81
POUGHKEEPSIE, N. Y.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

POSTAGE WILL BE PAID BY
IBM CORPORATION
P. O. BOX 390
POUGHKEEPSIE, N. Y.

ATTN: CE MANUALS, DEPARTMENT B95

CUT ALONG LINE

OLD FOLD

8/63:700-EP-216

STAPLE 102714774 STAPLE