7300 Processing Unit

Design Description Manual

Volume II: Shared Resources

2501.002

MEMOREX

Computer System Products

February 1973 Edition

Memorex Corporation
Santa Clara, California 95052

Requests for copies of Memorex publications should be made to your Memorex representative or to the Memorex branch office serving your locality.

This publication is provided for informational purposes. Contact Memorex for the latest periodic enhancement.

A readers' comments form is provided at the back of this publication. If the form has been removed, comments may be addressed to the Memorex Corporation, Publications Dept., Santa Clara, California 95052.

7300 Processing Unit Design Description
Volume 2, Shared Resources

This volume provides Memorex Field Engineers with detailed operating principles of the 7300 Processing Unit's shared resources. The information is presented in two sections. Section 2, Principles of Logic Operation, describes the hardware making up the shared resources - main storage, control storage, arithmetic-logic unit, timing, and control - plus the basic and extended register files and System Control Panel. Section 3, Micro-Instruction Repertoire, contains a detailed description of each micro-instruction by means of a narrative explanation and hardware execution flow diagram. This section also describes how micro-instructions are used to implement machine-language instructions and gives directions for reading the micro-instruction assembler listing. Hardware descriptions are keyed to corresponding drawings in the 7200/7300 Logic Diagrams Manuals, and thus provides the Field Engineer with a comprehensive maintenance package.

This volume is part of a four volume set comprising the 7300 Processing Unit Design Description Manual. The set of four volumes is assembled as a continuum of section numbers containing the following information:

Volume 1, Overview (2501.001)

Section 1. A general description of the 7300 Processing Unit.

Volume 2, Shared Resources (2501.002)

Section 2. A detailed description of main storage, control, timing and arithmetic parts of the 7300 Processing Unit.

Section 3. A detailed description of the formats, characteristics and implementation of the micro instructions associated with the 7300 Processing Unit.

Volume 3, Dedicated Resources (2501.003)

Section 4. A detailed description of the two basic data (selector) channels for the 7300 Processing Unit.

Section 5. A detailed description of the Integrated Communications Adapter (ICA) for the 7300 Processing Unit.

Section 6. A detailed description of the Integrated File Adapter (IFA) for the 7300 Processing Unit.

Volume 4, Power System (2501.004)

Section 7. A detailed description of the 7300 Processing Unit power system.

NOTE

Because Volume 1 provides an overview of the 7300 Processing Unit, it should always be used as an introduction to the other volumes in the set.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF TABLES

2. PRINCIPLES OF LOGIC OPERATION

INTRODUCTION

This section contains a detailed logic description of the shared resources portion of the MEMOREX 7300 Processing Unit. The section begins with an over-all block diagram description of the shared resources, discussing the principal data paths and explaining some of the basic concepts of time slicing and machine language instruction implementation by micro instructions. Following the block diagram description is a comprehensive analysis of each functional part of the shared resources. Supplementing the narrative description are in-text logic drawings illustrating each functional part, and portion thereof. These drawings are based on the logic diagrams contained in the 7300 Processing Unit Support Diagrams manual. For ease in correlating the in-text drawings to the logic diagrams, each drawing references the physical module (PC board) on which the logic shown in that drawing is contained, both by a dashed rectangle to indicate the module boundary and the module number. The module number is of the form 1AXX and is (usually) located in the lower right corner of each module boundary.

NOTE

> Signal names prefixed with a + or — in the in-text drawings are identical to corresponding signal names shown in the logic diagrams, where the + or — represents the polarity of the signal in the active state. Signal names without such a prefix represent a combination of individual signals, where the polarity of the signal is not conveyed.

BLOCK DIAGRAM DESCRIPTION

The shared resources encompasses all logic shared by the eight processor states during their operation. It includes elements for assigning time slices, reading and executing Micro Instructions (μI's), reading and writing file registers, accessing Main Storage (MS), and communicating with the four I/O processors and the System Control Panel. (When enabled for operation, the Panel is granted time slices just as if it was a ninth processor.) As discussed in Chapter 1, the shared resources consists of four major parts: the Arithmetic-Logical Unit (ALU), MS, Control Storage (CS), and Timing and Control logic. For purposes of discussing the operation of the shared resources, however, it is useful to also discuss operation of the Basic Register File (BRF) and Extended Register File (ERF) portions of the dedicated resources because of their intimate relationship with the shared resources. The shared resources plus the combination of BRF and ERF are referred to as the Central Processing Unit (CPU).

A block diagram of the CPU is shown in Figure 2-1. This block diagram is similar to the CPU block diagram, drawing 503247, in the logic diagram manual but has been simplified by removing some of the auxiliary elements such as (most) register fan-in and MS and CS parity check circuits. Three of the four major elements (ALU, MS, and CS) are indicated by dashed line boxes. In addition, the BRF and ERF (both Groups I and II) are similarly designated. The block diagram will be discussed by describing each data path shown on the diagram and its relationship to other such paths during execution of μI's during one time slice. The numbers appearing in parentheses in text refer to a corresponding data path in Figure 2-1.

TIME SLICE ALLOCATION

Assignment of a time slice to a particular processor state actually begins at E560 of the previous time slice. (The notation *E560* means 60 nanoseconds into minor cycle E5 of the major cycle.) At this time, priority is granted to the processor state under consideration. This sequence of events begins when the processor's Busy flip-flop in the Busy/Active (B/A) register of the ERF Group II is set. Setting this flip-flop essentially informs the shared resources that the processor under consideration has been assigned a task to perform and, consequently, will need a

time slice to perform this task. Setting the Busy flip-flop can be done by software, under program control, via the ALU fan-in (1) or by manual control from the Panel (2). In addition, each of the four I/O processors can issue a request to set the Busy flip-flop (3) when they are ready to perform an I/O-related operation. The Busy flip-flop output is routed to the Resync register in the Resource Allocation Network (RAN) via path (4). The Resync register functions as a *job queue* register by holding all requests for time slices from the various processors until granted by the RAN. The requests are fed to the priority network, which assigns priority to each request. Normally, the priority network assigns time slices to each processor in a cyclic fashion: 0 through 7, 0 through 7, and so forth. The four I/O processors, however, operate in a real-time environment; consequently, their needs for time slices are often critical to avoid loss of data transmitted to or from an I/O device. Therefore, these processors can override the normal cyclic assignment of time slices by setting a corresponding bit in the Priority register (5). This register enables the affected I/O processor to secure an out-of-sequence time slice according to one of two schemes: Enable Priority (secure a time slice when needed) or Invoke Priority (secure alternate time slices whether needed or not). At E560, the priority network is sampled and the number of the processor granted the next time slice is fed to the Read register (6).

R-PORTION READ OPERATIONS

The Read register contents are used to select the Pμ and F_{RF} registers in the Group I ERF to obtain *housekeeping* information required to begin the present time slice. This housekeeping information consists of the present MLI being executed, contained in F_{RF}, and the address of the first μI of that MLI to be executed during this time slice, contained in Pμ. Normally, this housekeeping information will reflect where the last time slice assigned to this processor left off, that is, the MLI will remain the same and the starting μI address will be one greater than that of the last μI executed during the last time slice (unless a jump or skip occurred at the end of the last time slice). Under some circumstances, however, the starting μI address will have been modified between the time slices due to a boundary-crossing operation. This operation allows processor state 4 (Executive) to access registers associated with the present processor state, making it possible for processor state 4, during its assigned time slice, to load a different starting μI address into the present processor's Pμ.

The housekeeping *Read* operations take place during the R portion of a time slice, which overlaps minor cycles E6 and E7 of the previous time slice. These timing relationships are shown in Figure 2-2. For convenience, the R portion is considered to consist of two 100-nanosecond minor cycles: R0 and R1. During R0, the processor number in the Read register is routed to the Group I ERF via (7), and the starting μI address is read from Pμ and clocked into the Sμ register at E680 (8). The Sμ register holds the address of the next μI to be read from CS. During R1, the MLI contained in F_{RF} is read and transferred to the shared resources F register (9). Meanwhile, the starting μI address clocked into Sμ at E680 has already accessed CS to read the first μI to be executed during the new time slice (10). This operation also takes place during R0, and at E000 the starting μI is clocked in Fμ (11) for translation and subsequent execution during E0.

Because of the overlapping facility of shared resources, the next μI in a sequence is (usually) read from CS during the minor cycle that the present μI is being executed. This can be seen from the previous paragraph where the first μI of the next time slice is being read at R0 (E6) simultaneous with translation/execution of the next to last μI of the present time slice during E6 (R0). In general, this overlapped operation holds true for most μI's that take only one minor cycle to execute. Therefore, a single one-minor-cycle-execute μI actually takes two minor cycles to implement: one minor cycle to read the μI from CS and one minor cycle to execute. If the μI is a *Register File Write* (and ALU propagation requirements are met), the execute portion (store into file register) does indeed take place during the second minor cycle. However, if the μI is a *Register File Read,* the execute portion actually extends into a third minor cycle.

This timing relationship for successive *Register File Read* μI's executed during one time slice is shown in Figure 2-3. The top portion of this figure shows the logic elements through which the *Register File Read* μI must pass during its implementation. As can be seen, the route traversed by the μI consists of two register-to-storage-to-register paths. The first path begins with Sμ through a fan-in to CS through a fan-out to Fμ. This path is used to read the μI from CS at the location specified by the address in Sμ, and route it to Fμ. The time required to traverse this path is 120 nanoseconds, from the time that Sμ is clocked with the μI address (CLKSμ at t) to the time that Fμ is clocked with the μI read from CS (CLKFμ at t+120). The second path begins with Fμ through a fan-in to the register file through a fan-out to Aμ and Bμ. This path is used to translate the μI in Fμ, access the register in either the BRF or ERF defined by the μI, and process the register contents in the ALU via Aμ and/or Bμ. The time required to traverse this path is also 120 nanoseconds, from the time that Fμ is clocked with the μI (CLKFμ at time t) to the time that Aμ and/or Bμ are clocked with the register contents (CLKAμ/Bμ, at t+120). If the μI being executed is the first one in a time slice, the two aforementioned paths are preceded by a third path from the Read register in the

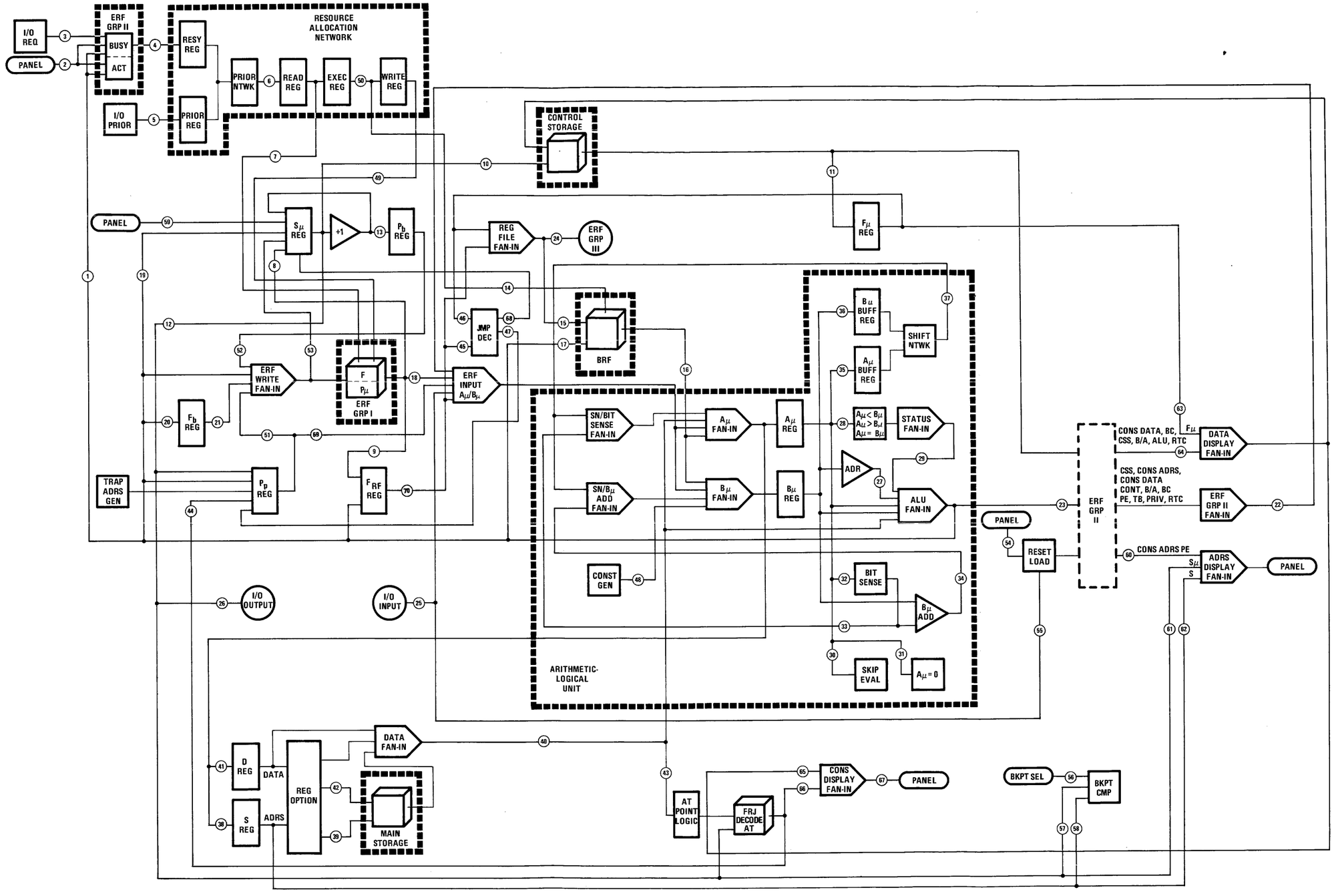


Figure 2-1. Central Processing Unit Block Diagram

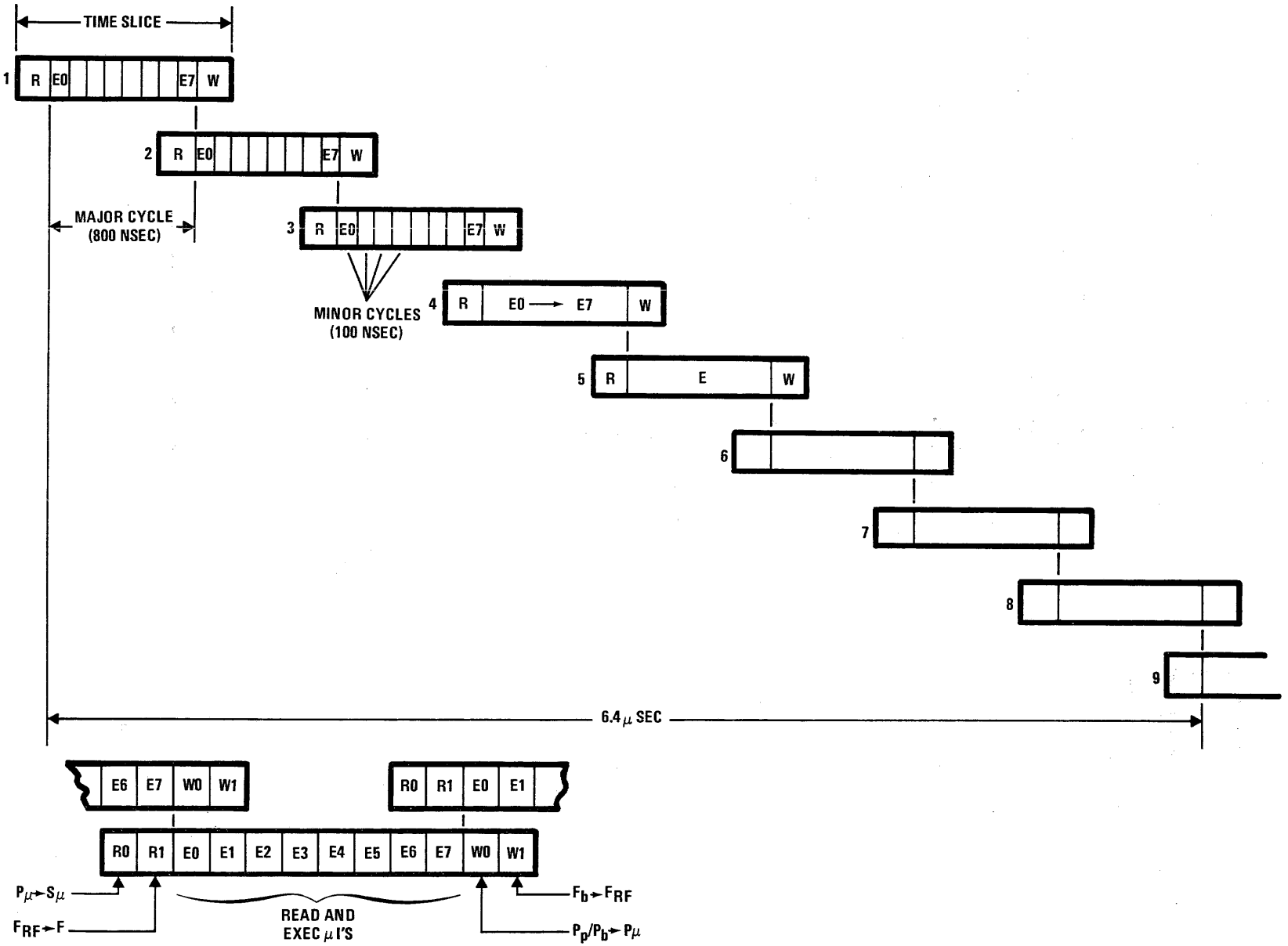


Figure 2-2. Time Slicing

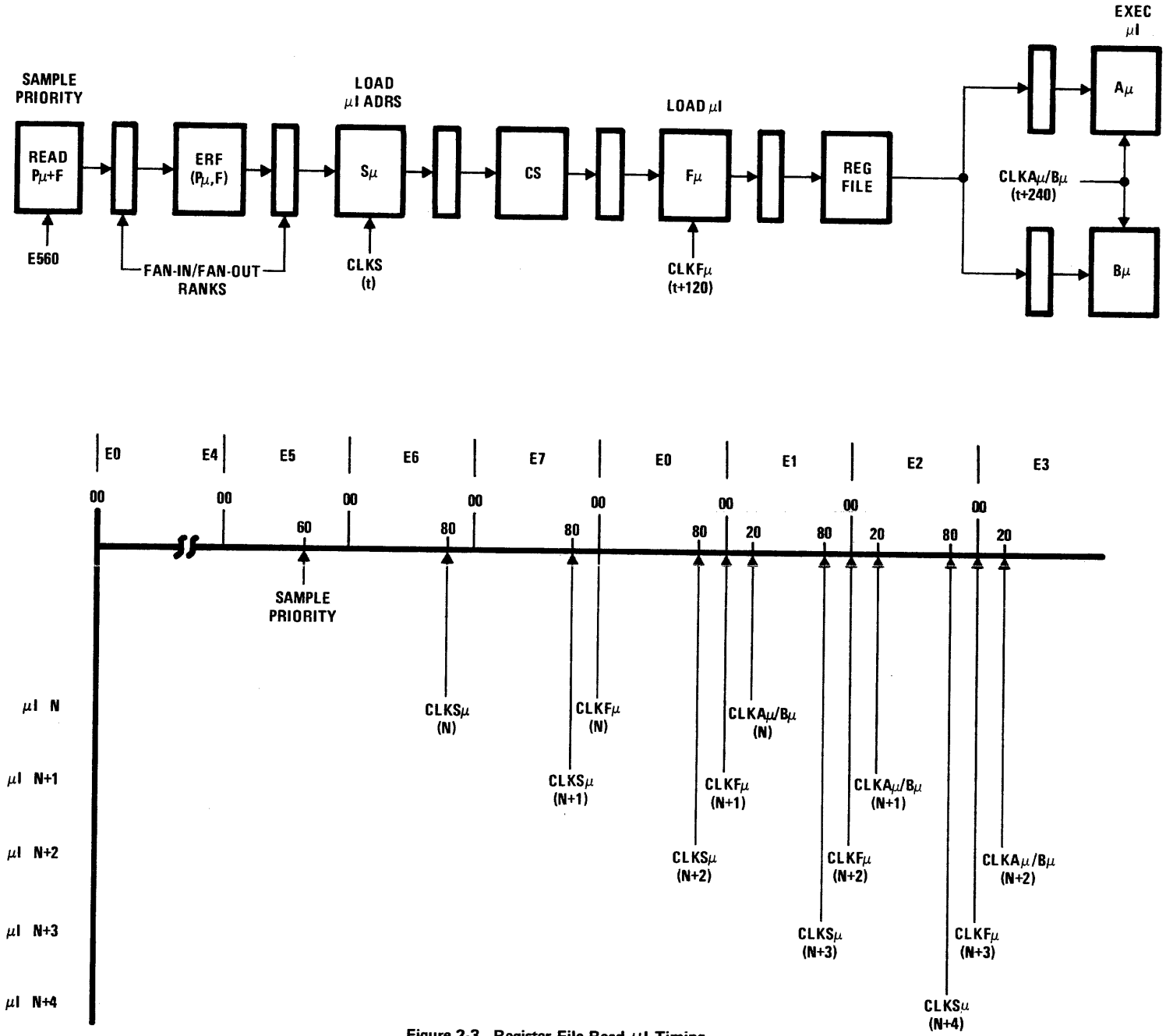


Figure 2-3. Register File Read μI Timing

RAN to Pμ in the Group I ERF to Sμ. This path, which starts at E560, when the processor number is clocked into the Read register, also takes 120 nanoseconds to traverse.

Since Aμ and/or Bμ are not loaded with data until 240 nanoseconds after the μI address has been loaded into Sμ, execution of a single *Register File Read* μI actually takes two complete minor cycles plus part of a third and a fourth minor cycle. This can be seen from the bottom part of Figure 2-3, which shows the times at which Sμ, Fμ, and Aμ/Bμ are clocked for each μI of a time slice. Using I N (the first μI executed in the time slice) as an example, its total execution time extends through minor cycles E6 and E7 of the previous time slice and through E0 and E1 of the present time slice, starting with CLSK at E680 and ending with CLKAμ/Bμ at E120. The relative times at which the same clocking operations occur for successive μI's, however, is only 100 nanoseconds. For example, CLKSμ for μI N occurs at E680 and CLKSμ for μI N+1 occurs at E780, 100 nanoseconds later. In this sense, then, execution of a μI is considered to take only 100 nanoseconds since an operation performed on μI N, for example, can be followed by the same operation performed on μI N+1 only 100 nanoseconds later. This so-called *pipeline* effect extends through the whole time slice, so that operations associated with μI's begun near the end of the time slice assigned to processor X can actually extend into the beginning of the next time slice assigned to processor Y.

Returning to Figure 2-1, the μI address in Sμ used to read the μI presently in Fμ is automatically updated by the Sμ+1 adder. This adder adds +1 to the contents of Sμ and routes it back to Sμ to form the address of the next μI to be read from CS. This is the *normal* manner in which the next μI address is formed since for any one sequence, μI's are arranged in consecutive order in CS. However, there are several other ways of loading Sμ with the address of the next μI, depending on several factors such as execution of a *Branch* μI, storing data into Pμ, and operating in the consecutive cycle (CC) mode. These alternate loads of Sμ are discussed in the following paragraphs. If the μI being executed is a *Blockpoint* (BP) μI, its address updated by +1 will be fed to Pp for use as the anticipated starting μI address for the next time slice (12), subject to the boundary-crossing operation discussed above. In addition, the updated address is sent to Pb (13) for certain conditions when the anticipated starting μI address cannot be sent to Pp.

The μI in Fμ is now ready for execution. From this point on, the paths traversed by the μI will depend on the type of μI, as discussed in the following paragraphs.

MAJOR CYCLE EXECUTION

BRF Read

A BRF *Read* μI selects a register of the BRF by a combination of the processor number obtained from the RAN Read register (14) and the register number obtained from the μI via the register file fan-in (15). The contents of the selected register are routed to the Aμ and/or Bμ registers of the ALU (16) for processing defined by the μI operation code. The contents may also be routed to the S register (38) or D register (41) via the Aμ register fan-in.

BRF Write

A BRF *Write* μI selects a register of the BRF in which to write data from some shared resources register (usually the Aμ, Bμ, or D register) or logical/arithmetic combination of the contents of such registers, as defined by the μI operation code. The register is selected in the same manner as for a BRF *Read* μI: processor number from Read register (14) and register number from the μI (15). Upon selecting the register, the contents to be written are gated through the ALU fan-in logic to the BRF (17).

ERF Group I Read

The ERF Group I read μI's do not read Pμ or F_{RF} directly; instead, they read the contents of the Buffer registers (Sμ/Pp and F) that reflect the most recent contents of Pμ and F_{RF}, assuming that Pp has not been updated by a blockpoint μI. The contents of Sμ/Pp (the notation Sμ/Pp indicates that status bits 0 and 1 are read from Sμ and CS address bits 2 through 15 are read from Pp) or F are routed to the ERF fan-in Aμ/Bμ via paths (69) and (70). They are gated through the fan-in by corresponding enables and sent to the Aμ and Bμ registers.

ERF Group I Write

Like the ERF Group I read μI's, the ERF Group I write μI's access Pμ and F_{RF} through their buffers, Sμ/Pp and F (and Fb). If Pμ is to be written, data is routed from the ALU fan-in to Pp via Sμ and/or Pp (19). Then at the end of the time slice, the data in Pp is written into Pμ as part of the W0 cycle housekeeping operation. If F_{RF} is to be written, data is routed from the ALU fan-in to F/Fb (20). Then at the end of the time slice, the data in Fb is written into F_{RF} as part of the W1 cycle housekeeping operation (21).

ERF Group II Read and Write

The ERF Group II register is selected by a combination of processor number and register number, similar to that for a BRF register. If a *Read* operation is specified, the contents of the selected register are fed to the ALU via the ERF Group II fan-in and the ERF input Aμ/Bμ fan-in (22). If a *Write* operation is specified, the data to be written is sent from the ALU via the ALU fan-in (23).

ERF Group III Read and Write

These μI's are programmed as part of an I/O data transfer operation, since the ERF Group III registers associated with the four I/O processors are located in the corresponding adapters. The Group III register is selected by the μI through the register file fan-in (24). Data received from a Group III register is put in to the ERF input Aμ/Bμ fan-in (25). Data transmitted to a Group III register is sent out via the ALU fan-in (26).

Arithmetic μI'S

Arithmetic μI's comprise those executed by the ALU. They include *Sum, Compare, Skip, Bit Sense,* and *Shift* μI's. The *Sum* μI's add the contents of Aμ and Bμ and route the sum to the ALU fan-in (27). The *Compare* μI's compare the contents of Aμ and Bμ for less than, equal, and greater than conditions by the compare network (28). The results are used to generate corresponding compare status bits which are sent to the ALU fan-in (29) for storage in a designated register. The *Skip* μI's determine conditions for skipping the next μI by the skip evaluate logic (30) and Aμ=0 logic (31). The *Bit Sense* μI's scan the contents of Aμ for a designated bit (32). When found, a number equal to the number of bit positions scanned without a find is added to the contents of Bμ (33). The result is then stored back in Bμ (33). In addition, the bit in Aμ providing the find may also be toggled. The *Shift* μI routes the contents of Aμ and Bμ to Aμ buffer and Bμ *buffer* (34), then shifts the contents of the two buffer registers by a specified amount. The shifted result is stored back in Aμ and Bμ (37).

MS Reference μI's

Both read and write references to MS require loading an address into the Sμ register from the Aμ register fan-in (38). This address is sent to MS via the Register Option (39). If an MS read is specified, the data is read from MS and routed to the FRJ decode address table (AT) pointer logic and to the ALU via path (40). If an MS write is specified, the word to be stored is loaded in the D register (41) and routed to MS via the Register Option (42).

FNJ and FRJ μI's

The FNJ (*Function Jump*) and FRJ (*Format Jump*) μI's are executed as part of the process of reducing the number of possible μI routines required to implement a MLI down to one particular routine applicable to that MLI only. The scheme for accomplishing this reduction is shown in Figure 2-4. Implementing a MLI requires three sequences: Read Next Instruction (RNI) sequence, Format Jump (FRJ) sequence, and Function Jump (FNJ) sequence. The RNI sequence is used to read the MLI from MS and isolate it to a group of several MLI's sharing common characteristics by performing an FRJ, or first-level, decode. This decode is performed by executing an FRJ μI, and determines the format of the MLI; that is, its type (register/register, memory/register, memory/memory, and so forth) and the addressing mode specified (direct or indirect). Figure 2-4 shows the sequences associated with a ADDR MLI (function code of 22). The ADDR MLI belongs to a class of MLI's identified as register/register MLI's, meaning that the operands processed by these MLI's are obtained either from a file register (direct accessing) or from MS at a location specified by the contents of a file register (indirect addressing). All MLI's with function codes of 20 through 29 belong to this class of MLI's. For the example shown in Figure 2-4, both operands required by the ADDR MLI are to be obtained by direct addressing (D/D).

Upon executing the FRJ μI, a branch is made to an area of CS determined by the FRJ decode to begin the FRJ sequence. Note from the figure that any of four different FRJ sequences for the register/register class of MLI's could have been entered, depending on the type of operand addressing specified. The FRJ sequence reads the first operand and prepares to enter the FNJ sequence by performing an FNJ, or second-level, decode. This decode is performed by executing an FNJ μI, and *picks out* the ADDR MLI from the rest of the register/register MLI's by identifying its function (add register to register) and causes a branch to another area of CS containing μI's required to implement the move register to register function. Note again from the figure that any one of 10 different FNJ sequences could have been entered, depending on the function of the MLI. The FNJ sequence reads the second operand, performs the required addition, stores the result, and branches back to the RNI sequence to read the next MLI.

Execution of both the FRJ and FNJ μI's form a branch address to *branch* to the start of the FRJ and FNJ sequences. Formation of the FRJ branch address, shown in Figure 2-1, is accomplished by developing an intermediate address tha points to the required FRJ branch address stored in the FRJ decode address table (AT). This pointer address is developed by feeding the MLI read from MS (43) to the pointer logic. The resultant FRJ branch address is read from the AT, combined with the contents of F, and routed to Pp (44). Formation of the FNJ branch address is performed directly by the jump decode logic. This logic, which is also used to form branch addresses for other jump μI's, is fed with bits from both the MLI in F (45) and the FNJ μI in Fμ (46). The FNJ branch address is formed by a combination of these bits and fed to Sμ (68) and/or Pp (47).

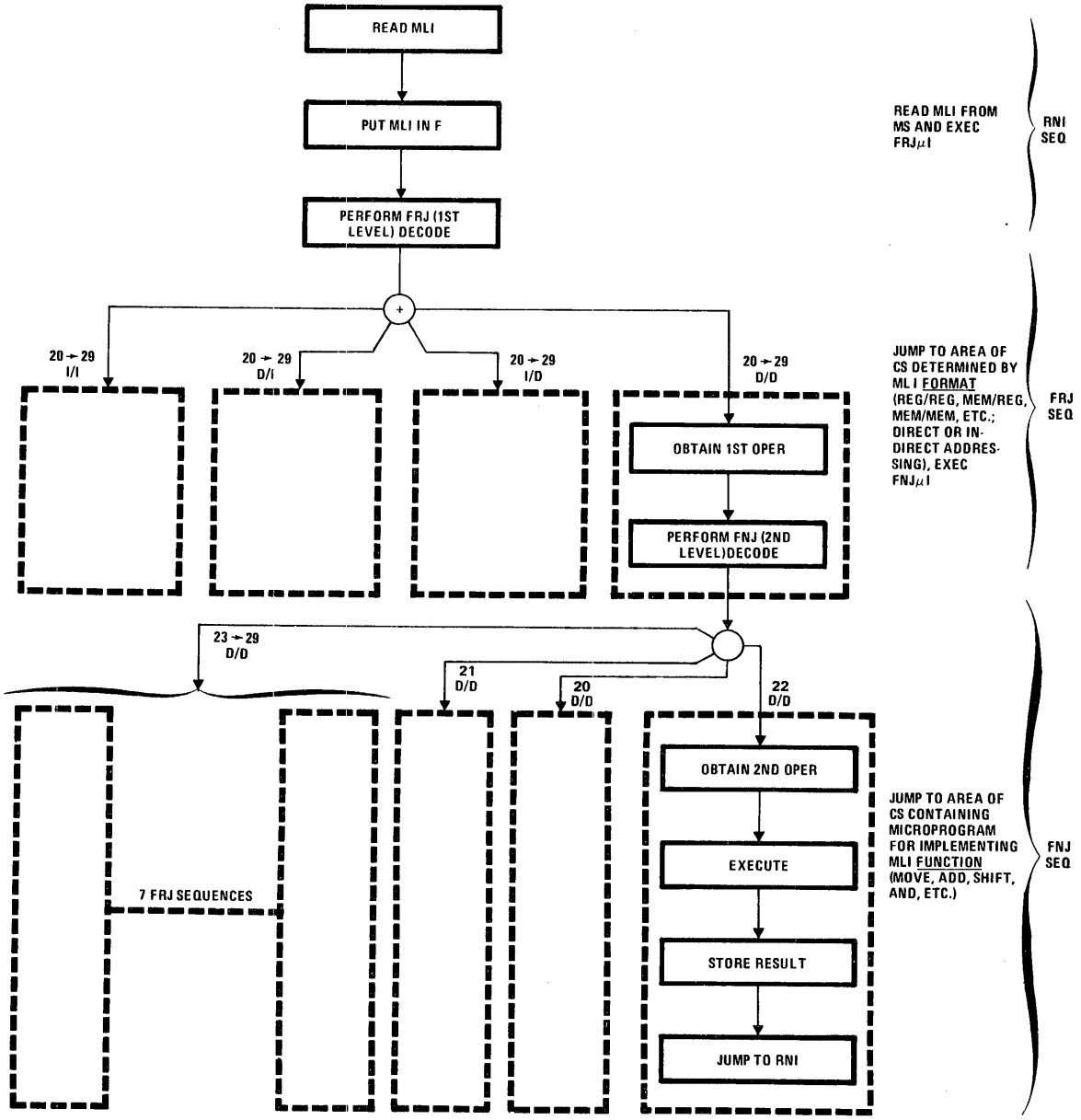


Figure 2-4. FRJ and FNJ Sequences for ADDR (22) MLI

Micro-Instructions Requiring Constants

A number of μI's require certain constants for their execution. These constants are generated by the constant generator, either by itself or with other logic. Among the constants generated are −1, 0, and +1 generated in conjunction with the Forced Carry Register for the load S μI's; 3 and D_{16} for the CORC μI; and certain word-, byte-, and nybl-length constants for the load Bμ, enter Bμ, and DIG μI's. The generated constants are fed to the Bμ register (48).

W-PORTION WRITE OPERATIONS

At the end of the time slice assigned to the processor state, the starting μI address in Pp and the MLI in F/Fb must be written back into Pμ and F_{RF} of the processor state's Group I ERF to enable resumption of the microprogram when the next time slice is assigned to the processor state. These operations take place at W0 (E0) and W1 (E1), respectively, as shown in Figure 2-3. The Pμ and F_{RF} registers are selected by the processor number, now contained in the Write register of the RAN (49). This register number was transferred from the Execute register during E5 (50). At W0, the starting μI address is written back into Pμ. This address comes from either Pp (51), if the BP μI was executed at E0 through E6, or from Pb (52), if the BP μI was executed at E7. The contents of Pb are used for the latter situation because Sμ has already been loaded with the starting μI address for the new time slice; consequently, the updated BP address cannot be routed to Pp via Sμ. Instead, it is fed directly to Pb after being updated. At W1, the MLI is written back into F_{RF} from Fb (21). (The contents of Fb must be used since F already contains the MLI for the next time slice.) If the processor is enabled for Consecutive Cycle (CC) operation, the starting μI address in Pp is routed back to Sμ (53) to enable the processor to continue running during the next time slice.

SYSTEM CONTROL PANEL OPERATIONS

Several operations initiated by the System Control Panel are shown in Figure 2-1. The reset/load operation is initiated by the Panel (54) to load CS with μI's from either a disc or card device (55). Breakpoint comparisons are made by comparing an address selected by the Panel breakpoint selectors (56) with a μI address in Sμ (57) or an MS address in S (58). A starting μI address can be manually set into Sμ from the Panel (59). Address-related information may be displayed on the Panel by means of the address display fan-in via paths (60), (61), and (62). Likewise, data-related information can be shown by means of the data display fan-in and Console display fan-in via paths (63) through (67).

DETAILED LOGIC ANALYSIS

TIMING

All timing needed by the various parts of the system, including all I/O processors, is derived from timing logic in the shared resources. A block diagram of this timing logic is shown in Figure 2-5. The master clock, from which all subsequent timing is derived, is a 10-megahertz crystal oscillator. This master clock feeds pulses to a 100-nanosecond delay line. This delay line is tapped at 10-nanosecond intervals and the resultant outputs fed to several long pulse and short pulse circuits. Each type of circuit is nearly identical and generate its respective output once every 100 nanoseconds (one minor cycle). The long pulse circuit generates write signals which are 45 to 60 nanoseconds wide. The short pulse circuit generates control timing pulses of 20 to 30 nanoseconds for a number of purposes: 1) register clock signals, 2) inputs to the real time clock (RTC) generator, 3) initiate E pulses via the E timing generator logic, and 4) furnish basic clock signals to the I/O processors. The E pulses are nominally 100 nanoseconds wide and are generated once during every major cycle. They are generated by means of a gray code counter whose binary outputs are ANDed together as required to generate each E pulse.

Basic Timing

Logic for the basic timing is shown in Figure 2-6. The 10-megahertz master clock output is adjusted by a potentiometer for a pulse width of 30 nanoseconds, as shown in Figure 2-7. This figure shows typical pulses generated by the basic timing over a period of 200 nanoseconds (two minor cycles). (Times for all pulses generated by the basic timing logic are found in Section 6 of the 7200/7300 Processing Unit Maintenance manual.) The adjusted master clock output is fed to a delay line, which contains 10 taps. Each tap provides a delay of 10 nanoseconds from the previous tap, therefore, a total delay of 100 nanoseconds from the previous tap, therefore, a total delay of 100 nanoseconds can be realized from the delay line. These taps are connected to the inputs of two types of pulse generate circuits, identified as long pulse circuits and short pulse circuits.

Each long pulse circuit consists essentially of two networks which feed the pre-set and pre-clear sides of a type D flip-flop producing pulses of 40 to 60 nanoseconds. Each network contains a potentiometer for independent adjustment of the leading and trailing edges of the flip-flop output. An emitter-follower is used to feed each network from the particular delay line tap for delay line isolation and impedance matching purposes.

There are three such long pulse circuits, used to generate NORMWR, LATEWR, AND BRFWRITE. Normally, the starting μI address and MLI are written into Pμ and F_{RF} of the active processor during the W portion of the time slice. For this purpose, NORMWR is used. During an **invoke** condition, however (after an IVK μI has been executed), the starting μI address and MLI are written into the Pμ and F_{RF} of another processor, specified by the contents of the Boundary Crossing (BC) register. For this purpose, LATEWR is used. Signal LATEWR is generated about 15 nanoseconds later than NORMWR to accommodate the extra time needed by the μI translation logic. Signal BRFWRITE is used to write into any register of the Basic Register File (BRF).

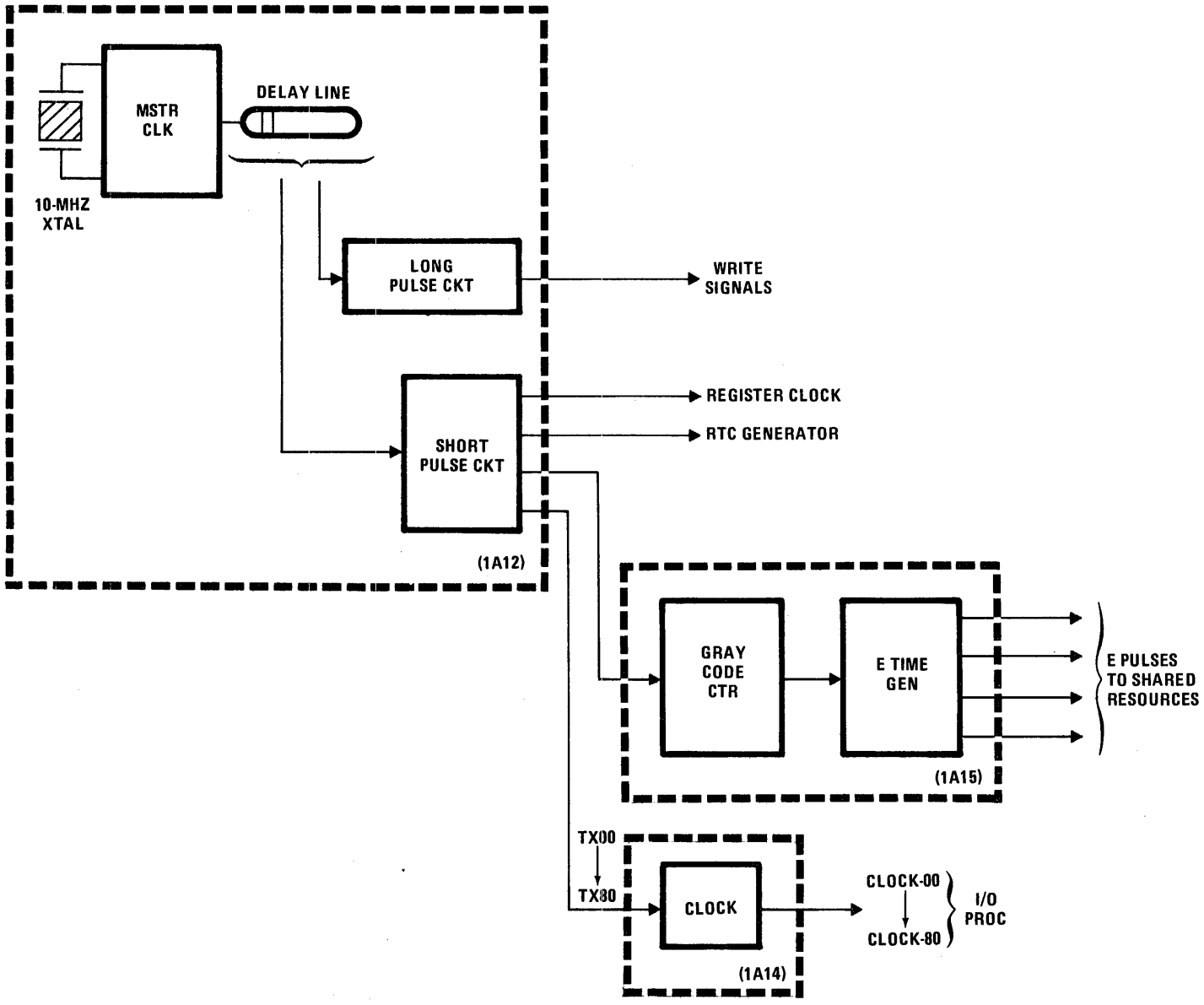


Figure 2-5. Timing Block Diagram

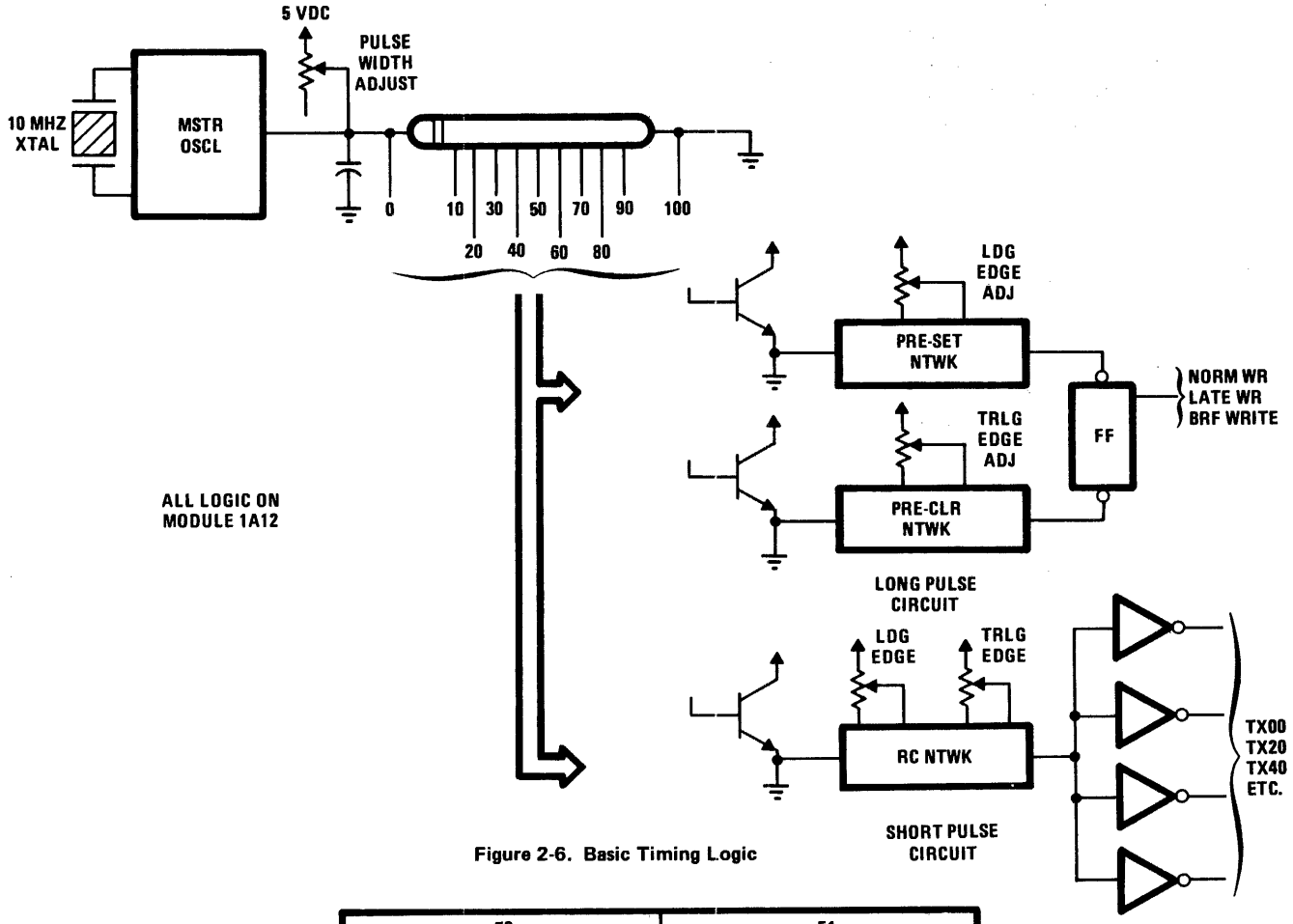


Figure 2-6. Basic Timing Logic

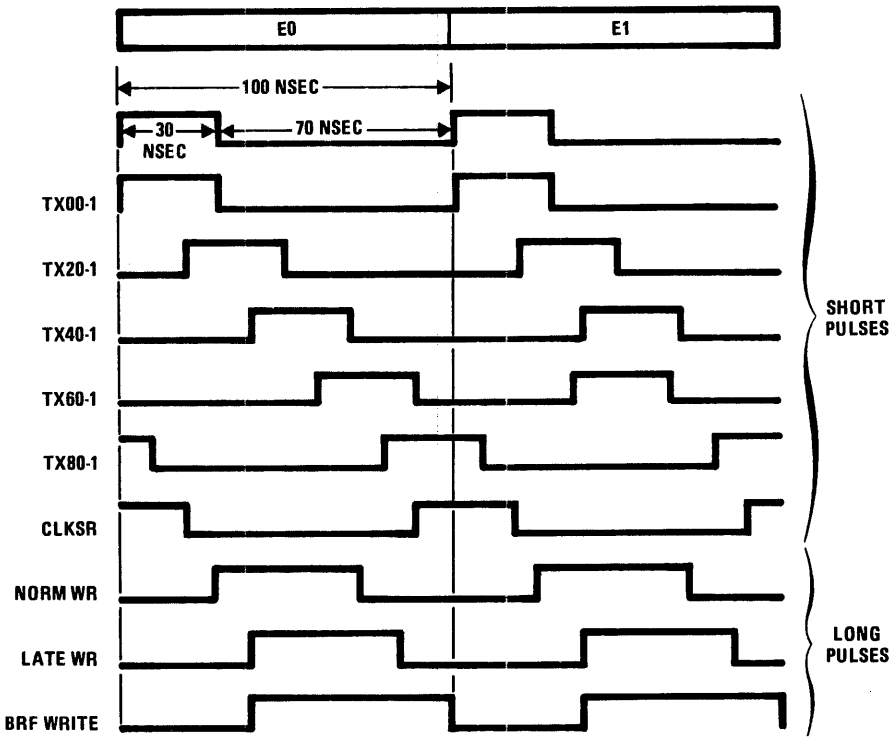


Figure 2-7. Basic Timing Waveforms

Each short pulse circuit consists essentially of two RC networks which generate pulses of 20 and 30 nanoseconds in width. Each network is adjustable providing independent adjustment of the output pulse leading and trailing edges. The output pulse is fed through several inverter drivers to provide the high fan-out requirements of these pulses. There are eight short pulse circuits, which generate TXXX signals and register clock signals. The TXXX signals are 20 and 30 nanoseconds wide, and are generated at intervals of 20 nanoseconds. The signal name identifies when it occurs during the 100 nanosecond period, i.e., TX20 indicates a signal generated 20 nanoseconds after TX00. These signals are used in their generated form for purposes of initiating operations at specific points within a minor cycle. For this purpose, they are usually combined with an E pulse, which defines the particular minor cycle. They are also used to generate E pulses via the E timing logic and clock pulses for use by the I/O processors. The register clock signals are either 20 or 30 nanoseconds wide and are used to preset, preclear, or enter data into a register at a specific time during a minor cycle. For this purpose, they are usually combined with a register clock enable signal which defines the condition under which data can be entered into the register (usually resulting from translating a particular μI).

E Pulse Timing

Logic for the E pulse timing is shown in Figure 2-8. The logic consists of two ranks of E pulse generators driven by corresponding ranks of a *gray code* counter. The two E pulse generator ranks produce a series of overlapping pulses nominally 100 nanoseconds in width called E pulses. Each pulse overlaps the preceding pulse by 50 nanoseconds (nominal)*. The *on-time* (OT) rank generates pulses that each start at the beginning of a minor cycle, i.e., pulse E1XX-O starts at the beginning of minor cycle E1. The *early time* (ET) rank generates pulses that each start 50 nanoseconds preceding the corresponding on time E pulse i.e., E1XX-E starts 50 nanoseconds before E1XX-O or in the middle of minor cycle E0. Waveform for typical *on time* and *early time* E pulses are shown in Figure 2-9. These E pulses are used in their generated form for combining with TXXX pulses of the basic timing to initiate operations as discussed in the previous paragraph. The E pulses are also used in combined forms with each other to generate pulses, two or more minor cycles wide. For example, E1/2XX-E is two minor cycles in

*The overlap actually ranges between 40 and 60 nanoseconds, due to flip-flop and gate delays.

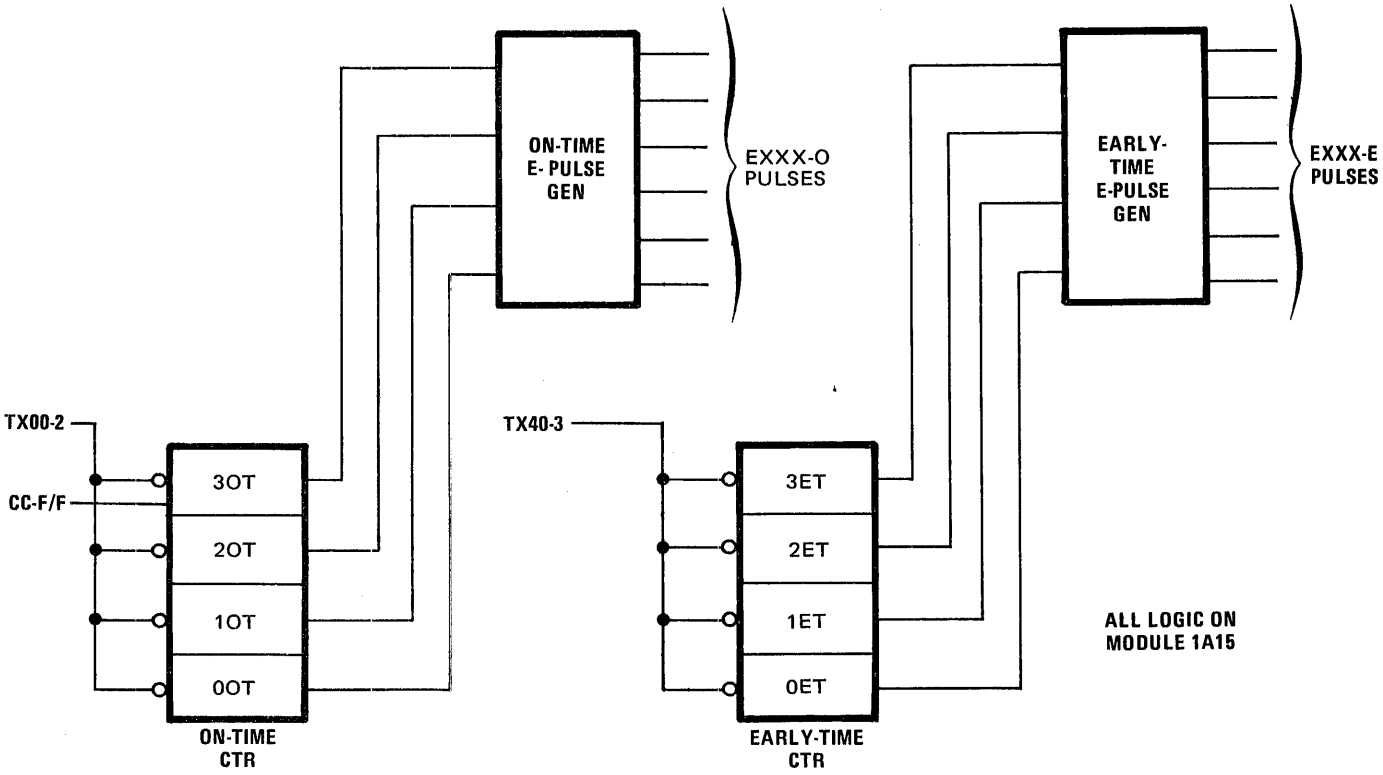


Figure 2-8. E-Pulse Timing Logic

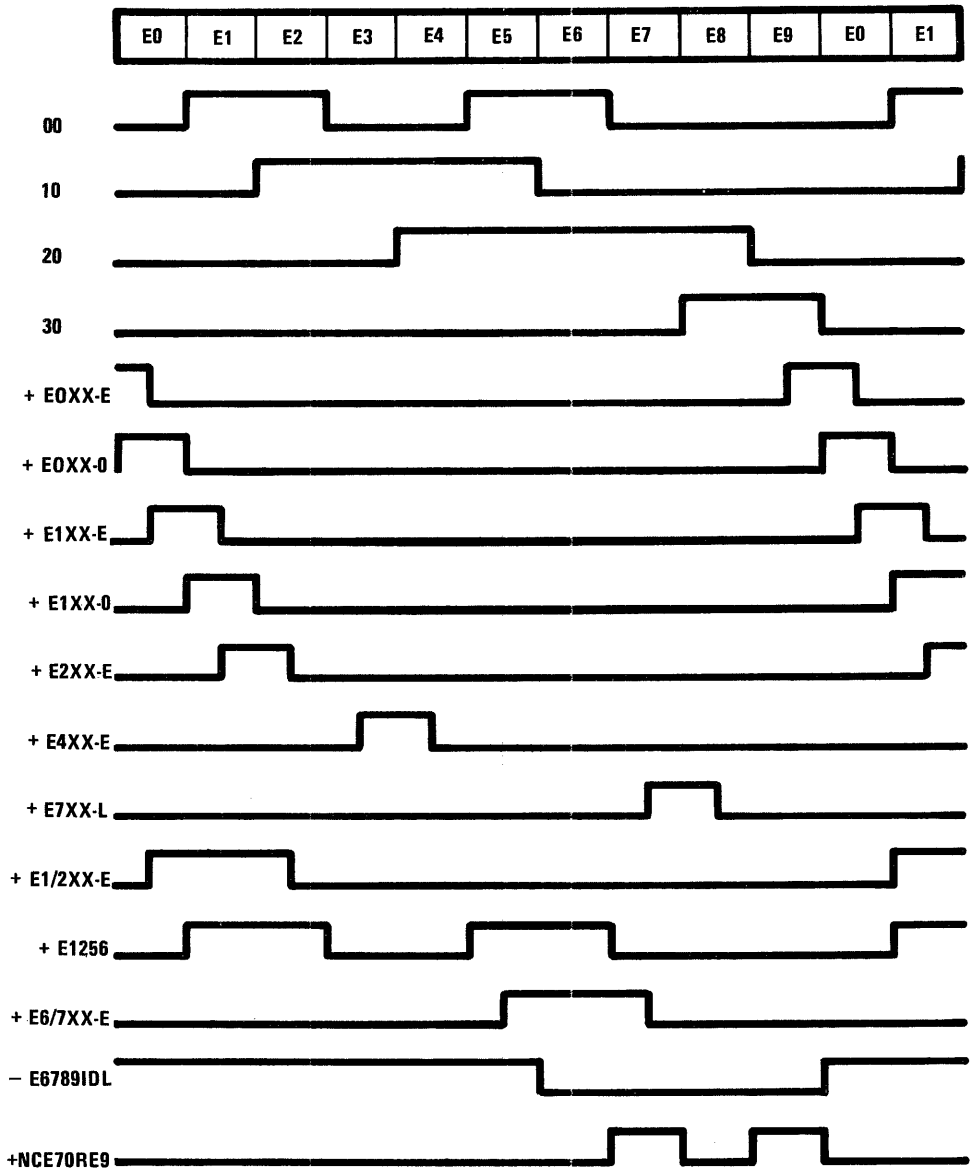


Figure 2-9. Gray Code Counter and E Timing Waveform

width, starting at E050 (E1XX-E start time) and ending at E250 (E2XX-E end time). Pulse E1256 is active for minor cycles E1, E2, E5, and E6, and inactive for the remaining minor cycles.

All E pulses are derived from two ranks of a *gray code* counter, an *on time* (OT) rank and an *early time* (ET) rank. Each rank consists of four flip-flops, numbered 0 through 3, that are interconnected so as to generate a *gray code* output*. Outputs from each of the flip-flops comprising the on time counter are shown in the upper part of Figure 2-9. The counter is initiated at E000 by TX00 from the basic timing. The early time counter generates the same counts as the on time counter, but starting 50 nanoseconds earlier. The BCD equivalent of each count produced by the on time counter and the corresponding E pulse generated is listed in Table 2-1.

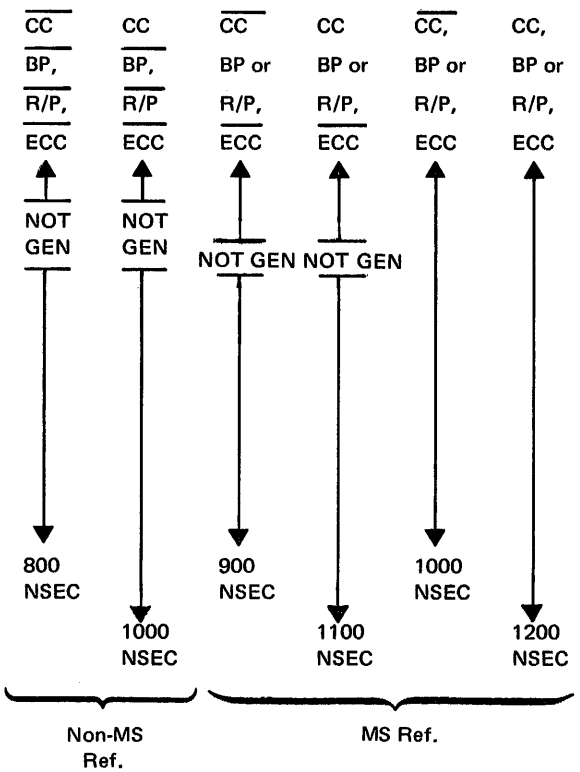
Major Cycle Duration

The number of E pulses generated per major cycle depends on whether the processor state is operating in the Consecutive Cycle (CC) mode and/or if it is making a reference to Main Storage (MS). If the processor state is making an MS reference, the major cycle timing is also influenced by which features of the Register Option (RO) that require additional propagation time are present. This information is tabulated to the right of the table in Table 2-1. If the processor state is not making an MS reference, the only variable is whether or not the processor is operating in the CC mode. If not, the major cycle time is 800 nanoseconds, formed by generation of E pulses E0XX-O through E7XX-O in sequence. If operating in the ECC mode, the major cycle time is increased to 1000 nanoseconds by the addition of E pulses E8XX-O and E9XX-O. These two pulses are generated by the output from *on time* counter flip-flop 3O. This flip-flop is enabled only if operating in the CC mode by signal CC-F/F, as shown in Figure 2-8. When a BCD count of either 4 if not in the CC mode, or 8 if in the CC mode is reached, the counter recycles itself back to 0 to start another series of E pulses.

*A *gray code* is a binary code in which only one bit position changes state ("0" or "1") each time the counter is advanced.

Table 2-1. On-Time Gray-Code Counter Timing and Major Cycle Durations

Counter Flip-Flops				BCD Result	E-Pulse Generated
3OT	2OT	1OT	0OT		
0	0	0	0	0	E0XX-O
0	0	0	0	0	E0XX-O'
0	0	0	0	0	E0XX-O"
0	0	0	1	1	E1XX-O
0	0	1	1	3	E2XX-O
0	0	1	0	2	E3XX-O
0	1	1	0	6	E4XX-O
0	1	1	1	7	E5XX-O
0	1	0	1	5	E6XX-O
0	1	0	0	4	E7XX-O
1	1	0	0	12	E8XX-O
1	0	0	0	8	E9XX-O



LEGEND
CC, $\overline{BP}$, $\overline{R/P}$, $\overline{ECC}$ – Major cycle duration if operating in consec-cycle mode; Basic Protect, Relocation and Protect, and ECC features not present.

If the processor state is making an MS reference, the variables include not only whether or not operating in the CC mode, but which features of the RO that require additional time for propagation are present also. These features include the Basic Protection (BP), Relocation and Protection (R/P), and the Error Correction Code (ECC) feature. If R/P, but not the ECC feature, is present, the major cycle timing is increased by 100 nanoseconds from a non-MS reference cycle to either 900 or 1100 nanoseconds, depending on whether or not the CC mode is enabled. This increase of 100 nanoseconds is provided by generating a second E0 pulse called E0XX-O'. Pulse E0XX-O' allows for the extra time required by the MS address to propagate through either the BP or R/P feature. If both the R/P and the ECC feature are present, the cycle time is increased by 200 nanoseconds from a non-MS reference cycle to either 1000 or 1200 nanoseconds. This increase of 200 nanoseconds is provided by generating not only pulse E0XX-O', but a third E0 pulse also, called E0XX-O". Pulse E0XX-O" allows for the extra time required by the ECC feature to check and correct, if necessary, data read from an MS location. (The operation of MS requires reading data from a MS location during both a read and a write operation; therefore, extra time must be allowed for ECC operation during both a read and a write operation.) It should be pointed out that the extra 100 nanoseconds added by pulse E0XX-O' is added regardless of whether the BP or R/P feature of the RO is present (since either one or the other must be present) even though the BP feature does not require the increased access time.

Pulses E0' and E0"* are generated during operations collectively referred to as *long access* operations, and are initiated by the long access logic shown in Figure 2-10. Generation of either just E0', or both E0' and E0", is determined by the adjustment of a delay network. This network is initially clocked at either E650 if not in CC mode, or E850, if in CC mode, by NCE70RE9. The output delay is adjusted on the basis of which RO features are present in the system, such that the output goes low at either E070, if either the BP or R/P, but not the ECC feature is present; or at E070', if either the BP or R/P, and ECC features are present. These delays are shown in Figure 2-11, along with subsequent timing, for both possibilities: BP or R/P but no ECC (solid lines) and BP or R/P, and ECC (dashed lines). The delay network output is clocked into a flip-flop at E720 to generate signal TIMER. This signal is combined with FXEQ-3, indicating that an MS reference is to be made (load SμI) and master enable ENLGACC. The result is $\overline{\text{LONGACC}}$, which goes low at E040 (worst case). As shown in Figure 2-10, this signal is used to block clocking of Fμ and Sμ with the next μI and following μI address. This is necessary to inhibit reading or executing a μI during the period that E0' and E0" are active. In addition, $\overline{\text{LONGACC}}$ is gated to two flip-flops that control the setting and clearing of the counter flip-flops. The output of the On Time flip-flop sends a high to the pre-clear input of the 0OT flip-flop in the on time counter. The result of this high pre-clear is to delay the flip-flop from setting for eithe; 100 or 200 nanoseconds. This action effectively generates pulse E0' and E0" by extending the E0 pulse width from 100 to either 200 or 300 nanoseconds. The high output from the Early Time flip-flop to the pre-clear input of the 1ET flip-flop in the early time counter produces a similar action to delay the *early time* E1 pulse by the required amount.

It is important to note that the train of E pulses generated, including the inserting of E0' and E0" pulses, is completely under hardware control (except for adding E8 and E9 if in the CC mode). In addition, every major cycle will contain eight distinct E pulses, even though the intervals of these pulses may vary as previously discussed. If the program being executed determines that it does not need the remaining minor cycles in a time slice, it cannot truncate the unneeded portion of the time slice. Instead, it must *cycle through* the rest of the time slice by performing NOP's until the end of the time slice. This is (usually) done by inserting non-blockpoint or resync μI's (either the SYNC μI itself or one that performs a resync as part of its execution) to account for the unused trailing portion of such major cycles. In this respect, the timing is completely synchronous in that every time slice will run to completion even if the program being executed during the time slice does not.

The I/O processor clock logic consists of five buffer drivers that are driven by basic timing pulses TX00, TX20, TX40, TX60, and TX80 respectively, as shown in Figure 2-5. These buffer drivers, in turn, generate CLOCK-00, CLOCK-20, CLOCK-40, CLOCK-60, and CLOCK-80, which are routed to the four I/O processors.

Real Time Clock Pulse Generator

The real-time clock (RTC) pulse generator generates two waveforms, one used to increment the RTC register and the other sent to both the Integrated Communications Adapter (ICA) and the Busy/Action (B/A) register. Each waveform is derived from clock pulse TX60 from the basic timing logic by means of appropriate countdown logic. A block diagram of the RTC pulse generator is shown in Figure 2-12. The basic timing initiate pulse, designated RTCINPUT, is fed to a divide-by-4 network

*To avoid possible confusion in the following discussion, it should be pointed out that pulses E0' and E0" are not generated as such. In reality, they represent pulse E0 extended in time by either 100 or 200 nanoseconds (both on-time E0 and early-time E1 pulses are extended). The result is a long E0 pulse of either 200 nanoseconds (E0 plus E0') or 300 nanoseconds (E0 plus E0' and E0") in length nominally. It is useful, however, to think in terms of adding the E0' and E0" pulses to retain the idea that all E pulses are nominally 100 nanoseconds wide.

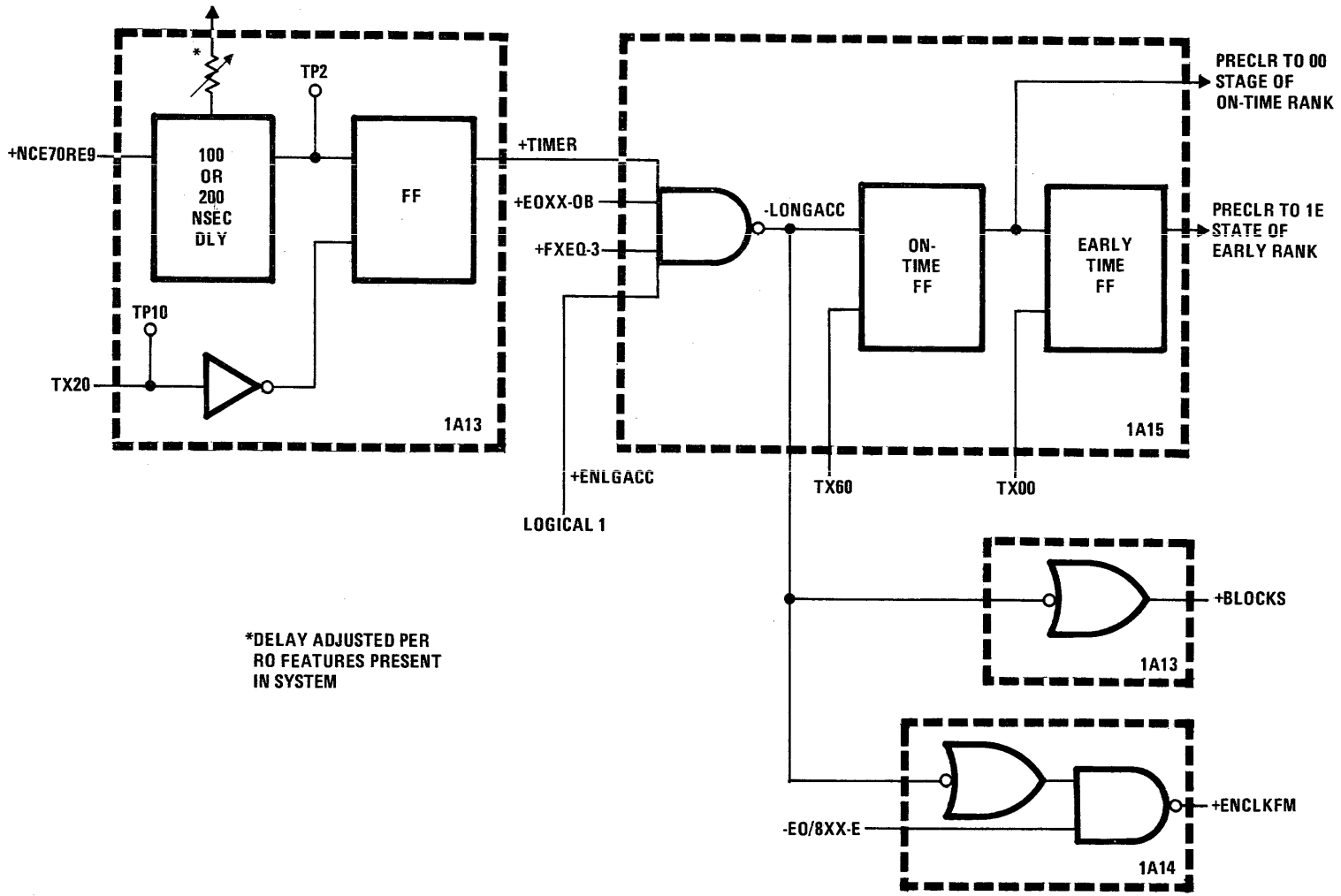


Figure 2-10. Long Access Logic to Generate EO′ and EO″ Pulses

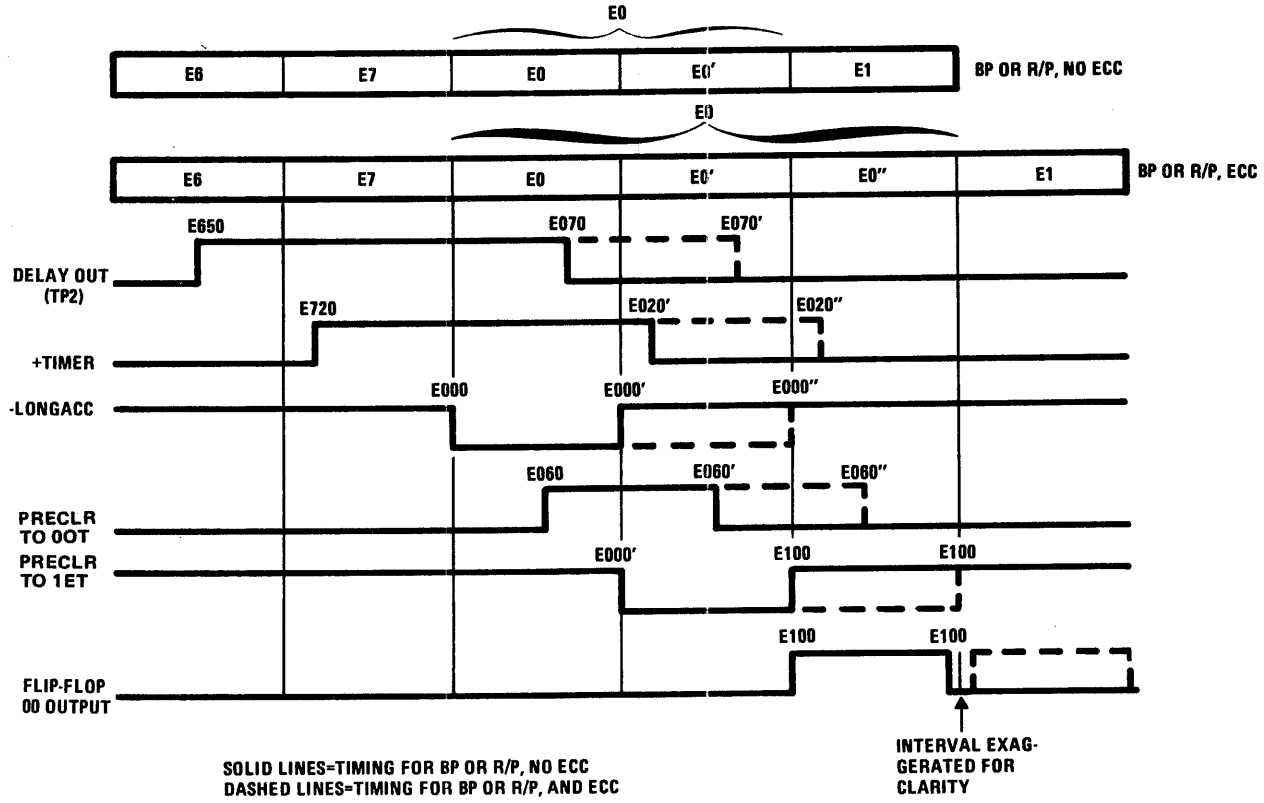


Figure 2-11. Long Access Logic Waveforms

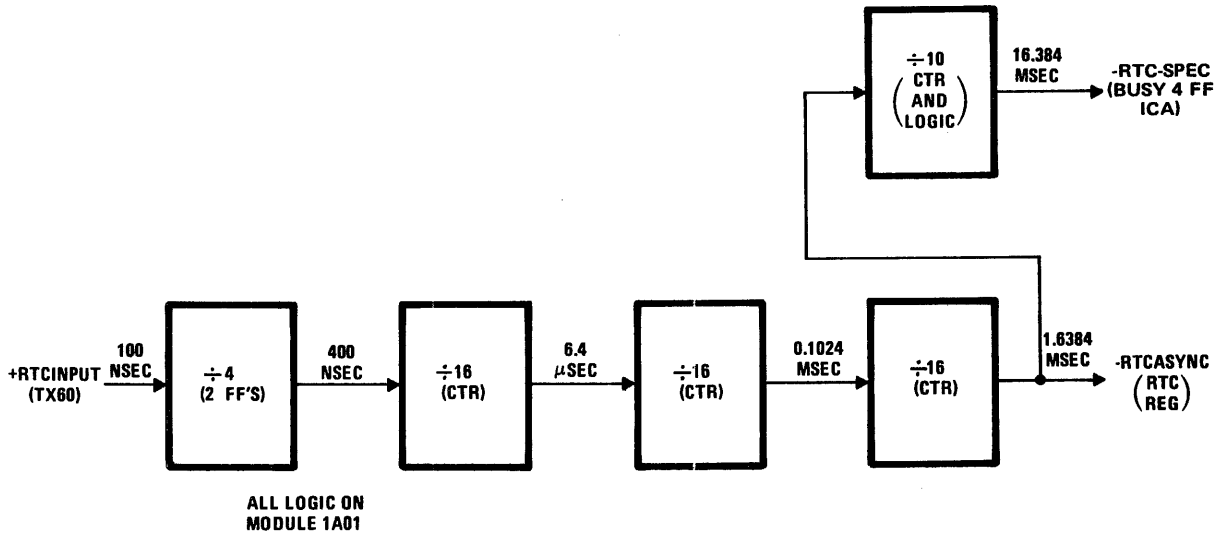


Figure 2-12. Real Time Clock Pulse Generator

consisting of two flip-flops connected in cascaded fashion. This results in a waveform repetition rate of 400 nanoseconds. This waveform, in turn, is fed through three divided-by-16 networks in serial fashion. Each network consists of a four-bit up/down counter that overflows when a count of 16 is reached. The resultant output is a waveform with a 1.6384 millisecond repetition rate (600 Hz) that is fed to the RTC register as RTCASYNC. This signal is also fed to a divide-by-10 network to generate RTC-SPEC at 16.384 millisecond intervals (60 Hz). This signal is fed to the processor 4 Busy flip-flop in the B/A register for purposes of *waking up* processor 4 at these intervals. The signal is used also in the ICA for character framing during synchronous transmission and for generating dial digits in the auto-call logic.

RESOURCE ALLOCATION

The resource allocation logic detects and stores requests for time slices from the eight processors that communicate with the shared resources and the System Control Panel. It then allocates time slices to each processor or the Panel on the basis of its needs. A block diagram of the resource allocation logic is shown in Figure 2-13. As shown, the logic consists of the Busy/Active (B/A) register, Console Busy flip-flop, Resource Allocation Network (RAN), and Consecutive Cycle (CC) logic. Requests from each processor are stored in corresponding flip-flops of the B/A register (register 02 of the ERF). The left-most eight flip-flops comprise the busy portion of the register; the remaining eight flip-flops make up the active portion. The Busy and Active flip-flops of each processor perform related functions during execution of a processor task. The Active flip-flop is set by software alone when the program determines that a particular processor should perform a particular task. The flip-flop is set at the beginning of the task and remains set until the task is completed. The Busy flip-flop can be set by either hardware or software, and informs shared resources that another time slice is needed by the processor to execute another portion of its assigned task. Requests from the Panel are handled in an analogous manner, by setting the Console Busy flip-flop. This flip-flop is set under hardware control only.

Upon being set, the Busy or Console Busy flip-flop output is entered in the task queue with other Busy flip-flop outputs for assignment of time slices in accordance with the priority level of the request. The task queue is defined as those processors which have requested time slices and are waiting for them to be granted. The task queue is entered by setting corresponding Resync flip-flops in the RAN, whose outputs are assigned priority in a cyclic fashion by the priority encoder. For processors 0 through 3, recognizing requests in this cyclic fashion (called the scanner mode), may be altered by setting the corresponding Priority flip-flop.

The Priority flip-flop may be set for either of two conditions: *enable* and *invoke*. The *enable* condition sets the flip-flop when the corresponding I/O processor determines that it is about to lose data if it cannot obtain a time slice expeditiously. The *enable* condition, therefore, enables a processor to obtain an out-of-sequence time slice. The *invoke* condition assures that a processor will unconditionally be granted every other time slice, whether it really needs them or not (provided conditions of higher priority are not present). Both priority conditions are initiated by software, which sets a corresponding *enable override* or *invoke priority* bit position in the individual processor's Control register.

The processor state number assigned the next time slice is routed to the Read register. This register initiates the R0 and R1 cycles of the time slice to read the starting μI addresses and present MLI from Pμ and F_{RF}, respectively, in the ERF. From the Read register, the processor number is routed to the Execute register, and then to the Write register near the end of the time slice. The Write register initiates the W0 and W1 cycles of a time slice, which stores away the anticipated starting μI address for the next time slice and the present MLI for starting the next time slice assigned to the processor. In addition, the Read and Execute register contents are routed to the Read/Execute compare circuits of the Consecutive Cycle (CC) logic. These contents are evaluated for equality, which they will be if no other processor is in the queue and the present processor requests a second time slice. The Read register contents are also used to determine if software has enabled the processor to operate in the CC mode. This is done by comparing the processor number in the Read register with the corresponding CC bit of the Control register. If the CC bit is set, the Clear CC flip-flop is not set. The flip-flop output is combined with that from the Read/Execute compare circuits to generate signals required by the processor to operate in the CC mode.

If neither a processor nor the Panel has requested a time slice, the resource allocation logic schedules a *null* condition during the following time slice(s). The null condition inhibits clocking Sμ with an updated μI address and disables the output from Control Storage (CS) which effectively loads NOP μI's into Fμ. A *refresh request* from MS will also generate a null condition if the refresh was scheduled for the previous major cycle but was preempted by a MS *access request.* During the next (present) cycle, the refresh request will lock out another MS access request (if generated) by setting up a null condition.

Busy/Active Register

The Busy/Active (B/A) register consists of 16 flip-flops, divided into two groups of eight flip-flops each. Flip-flops

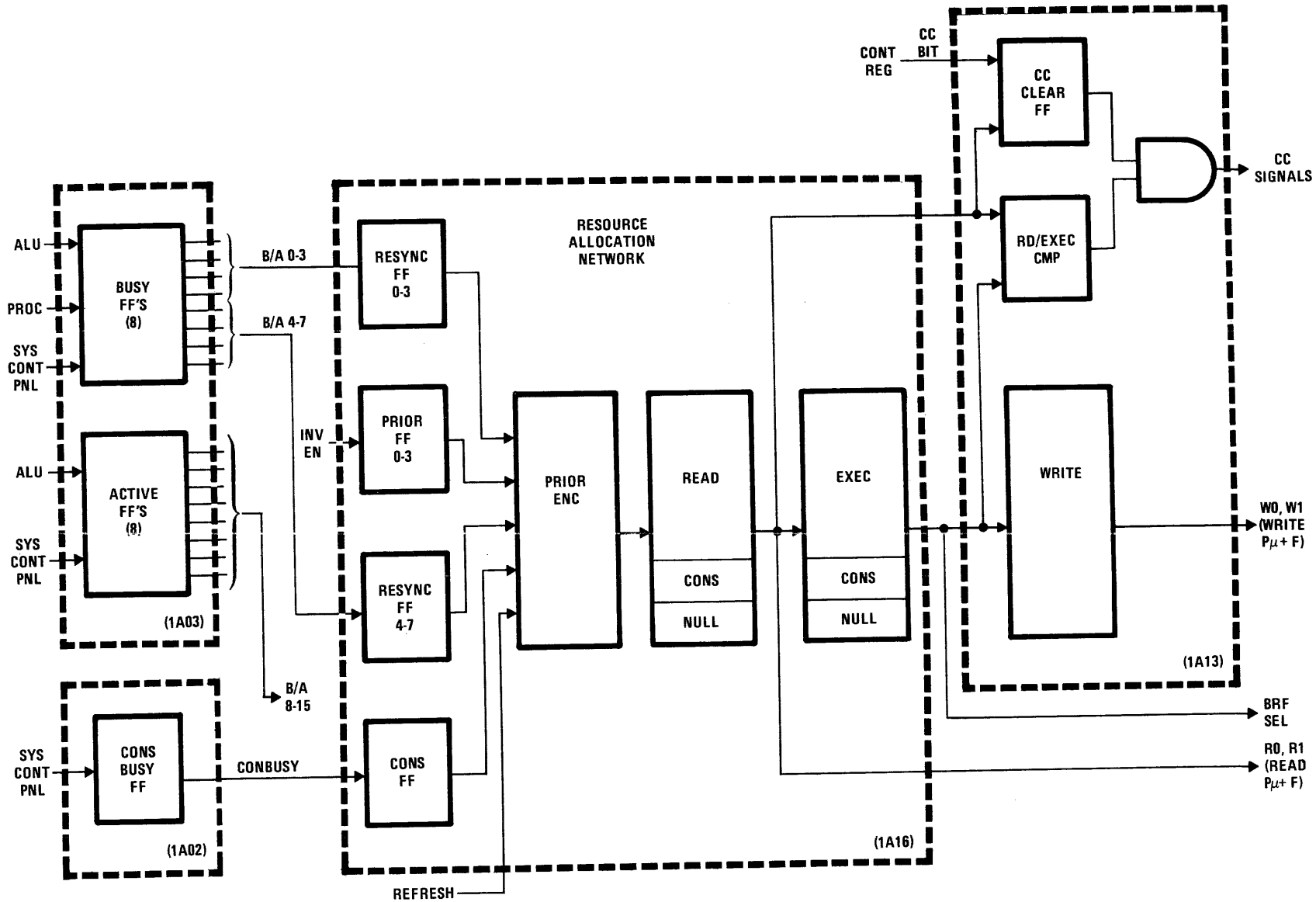


Figure 2-13. Resource Allocation Logic, Block Diagram

0 through 7 comprise the Busy flip-flops for the eight processors; flip-flops 8 through 15 make up the Active flip-flops for each processor. Because of the differences in processors, their Busy flip-flops are set and cleared under different conditions. All Busy flip-flops, however, are similar in that they are set or cleared by either an asynchronous (forced) or synchronous (clocked) input to the flip-flop. The forced inputs occur as a result of beginning or ending an input data transfer from a processor to the shared resources. Requests for such data transfers occur either under program control, from the requesting processor, or under manual control from the System Control Panel. The clocked inputs to the Busy flip-flops are generated under program control by the Executive processor.

The Busy flip-flops for both processor 0 (communications processor) and processor 4 (Executive processor) are shown in Figure 2-14. The "normal" method of setting these two flip-flops is by the REQ signal, indicating that an external event under hardware control wants a time slice for the processor. For processor 0, REQ is generated by the communications adapter signifying that it is ready to either send or receive data. For processor 4, REQ is generated after initialization of an Autoload operation, and at 16.384-millisecond intervals thereafter by the real-time clock (RTC) to *wake up* the Executive processor.

Setting the Busy flip-flop for processor 0 is inhibited if the corresponding PROCESSOR CONTROL SELECT switch on the Panel is set to STOP/STEP (SWSTOP is high). Signal SWSTOP enables a processor task to be executed in the stop/step mode from the System Control Panel. For processor 4, setting the Busy flip-flop via REQ is inhibited if the $\overline{\text{INHIBIT REQ}}$ signal is present. This inhibit signal is generated whenever the Busy flip-flop is under control of a *Breakpoint* operation from the Panel and provides a software debug facility for system programmer use.

The Panel may also *turn on* (initiate) a processor under manual control. This is accomplished by setting the PROCESSOR SELECT selector to either the 0 or 4 position and pressing the PROCESSOR RUN pushbutton. This action generates SWSELGO and GO FF, respectively. This manner of setting the Busy flip-flop is disabled if the switches are set when the processor is executing during minor cycles E6 through E9.

Occurrence of a CS parity, MS parity, or outbound error condition during the last time slice of a task sets the Busy flip-flop for one more time slice. This is necessary for two reasons (1) the error might occur at E7 so that no more time would be available in the time slice to form the trap address, or (2) unaware of the occurrence of an error condition, the microprogram might stop as a result of clearing the Busy flip-flop during the major cycle in which the error condition occurred. The problem is overcome by forcing the processor to run for one more time slice and forming the trap address at E7 of this forced time slice. Setting the Busy flip-flop for this condition is accomplished by START, which specifies the present processor executing a task, and TRAP-1 which specifies occurrence of the error condition.

Setting and clearing the Busy flip-flops for processors 0 and 4 (as well as for the other six processors) is accomplished under software control by means of the clocked input to the flip-flops. This is done at t80 of any minor cycle, providing ENCLKB/A is present. This clock enable is generated for any register file write μI when the destination is register 2 (the Busy/Active register) of the ERF. Unlike the other six processors, there are inhibiting conditions that may prevent software from setting the Busy flip-flop at ENCLKB/A time. Namely, the Busy flip-flops for processors 0 and 4 can be cleared at ENCLKB/A time only if the corresponding Active flip-flop has been previously cleared. In other words, these Busy flip-flops may not be cleared at ENCLKB/A time if the corresponding Active flip-flop is still set.

Clearing of the Busy flip-flops for processors 0 and 4 is accomplished when the PROCESSOR CONTROL SELECT switch of the Panel is set to STOP/STEP. In this mode, the selected processor runs for only one MLI or only one major cycle as determined by the setting of the CYCLE STEP switch. For whichever setting is selected, indication that one major cycle or one MLI has been executed is provided by RNI-TX. Signal STATE is included to insure that the Busy flip-flop is not cleared before the processor has been assigned a time slice during step mode operation from the Panel. The Busy flip-flop is also cleared under Panel control when the PROCESSOR CONTROL SELECT switch is set to BKPT (signal SWBKPT). This signal is ANDed with BKPT-TX, which is generated when any of the three BREAKPOINT MODE SELECT switches (READ INSTR, READ DATA, and WRITE DATA) is activated. These switches define the type of breakpoint action selected: *read* the MLI at the location specified by the breakpoint address (READ INSTR switch), *read* the operand at the location specified by the breakpoint address (READ DATA switch), or *store* the operand at the location specified by the breakpoint address (WRITE DATA switch). The breakpoint stop will occur at the end of the major cycle in which the breakpoint occurred.

Upon completion of a one-word transfer, the Busy flip-flop is cleared by the CIO signal. This signal is generated when a CIO μI compare condition is not met, indicating that additional words have yet to be transferred. The two CIO μI's compare the last byte

address with the current byte address. Signal CIO is generated for either an Aμ=Bμ or an A$\mu\neq$Bμ condition, depending on the predetermined μl program.

The Busy flip-flop for processors 1, 2, and 3 is shown in Figure 2-15. As shown, the flip-flops for these three processors are set via the forced set input by means of the same conditions as for processors 0 and 4. In addition, these flip-flops can also be forced to a set condition by an ATTN signal from the processor. This signal is related to the REQ signal in that it informs the shared resources that the corresponding processor wants a time slice. It differs from REQ, however, in that it is generated for a condition not associated with the operation currently being executed by the processor. Therefore, ATTN is inhibited from setting the Busy flip-flop until the corresponding Active flip-flop is cleared, meaning that the present operation has been completed and the operation that generated ATTN can now be executed. In addition, ATTN cannot set the Busy flip-flop during times E6 through E9 to eliminate timing problems associated with CC and stop/step operations at these times.

Setting the Busy flip-flops for processors 1, 2, and 3, via the clocked input, is done by means of software, the same as for the Busy flip-flops of processors 0 and 4. The ALU input must be enabled by both ENCLKB/A and the fact that either the Busy flip-flop is already set or the corresponding Active flip-flop is cleared.

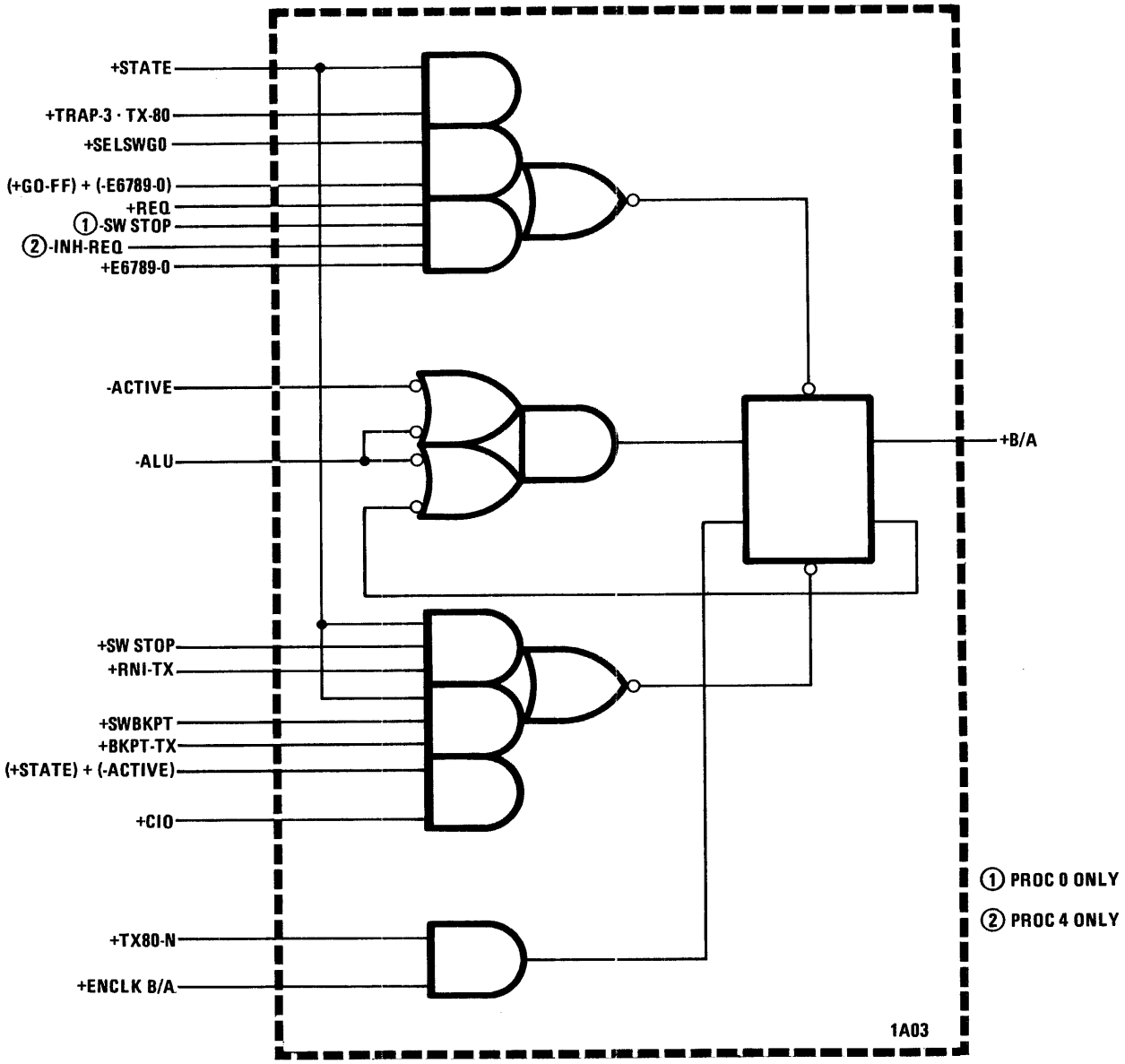


Figure 2-14. Busy Flip-Flops, Processors 0 and 4

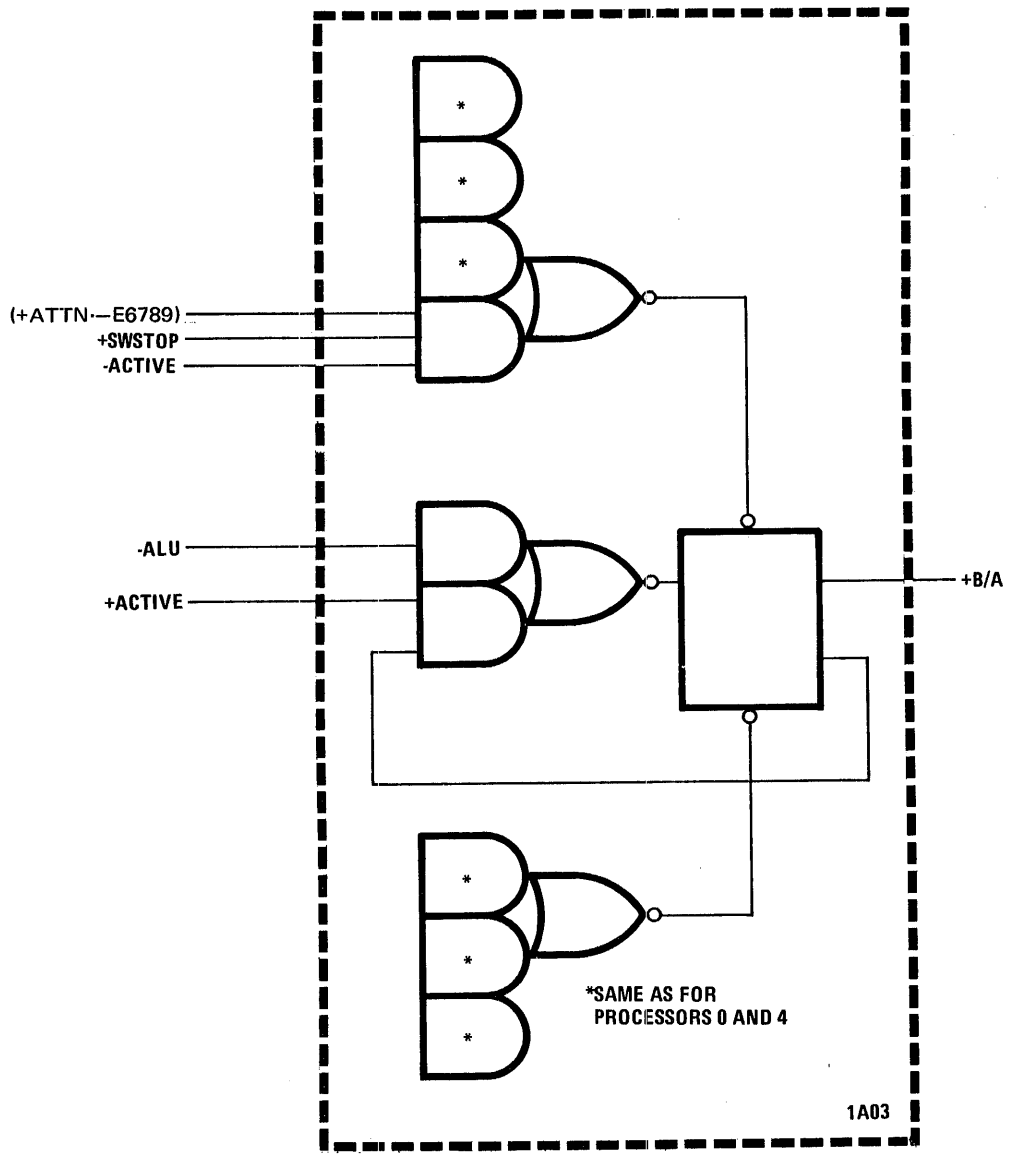


Figure 2-15. Busy Flip-Flops, Processors 1, 2, and 3

A diagram of the Busy flip-flops associated with processors 5, 6, and 7 is shown in Figure 2-16. Because these processors are general-purpose processors, the corresponding Busy flip-flops do not require either the REQ or ATTN forced set inputs. The only forced set inputs are those required for generating a trap address and selecting the processor from the Panel. The flip-flop is set or cleared under program control at ENCLKB/A time. Unlike the Communications and Executive processors, the general-purpose processors can be cleared under software control regardless of the state of the corresponding Active flip-flop. The Busy flip-flop is cleared via the forced clear inputs for an RNI and breakpoint condition initiated by the Panel. The RNI condition is implemented in a manner similar to that for processors 0 through 4, except that the action of the STOP/STEP switch is manifested by clearing the corresponding Active flip-flop to generate ACTIVE. This differs from that for processors 0 through 4 where the STOP/STEP switch input was supplied directly to the forced clear logic.

The Active flip-flops for all eight processors are very similar as shown in Figure 2-17. All eight flip-flops are set and cleared at ENCLKB/A time under program control. In addition, the Active flip-flops for general-purpose processors 5, 6, and 7 can be set and cleared via the Panel by means of the force set and force clear inputs. This is done by means of the corresponding PROCESSOR CONTROL SELECT switches, which set a flip-flop when set to the NORMAL (run) position or clear a flip-flop when set to the STOP/STEP position.

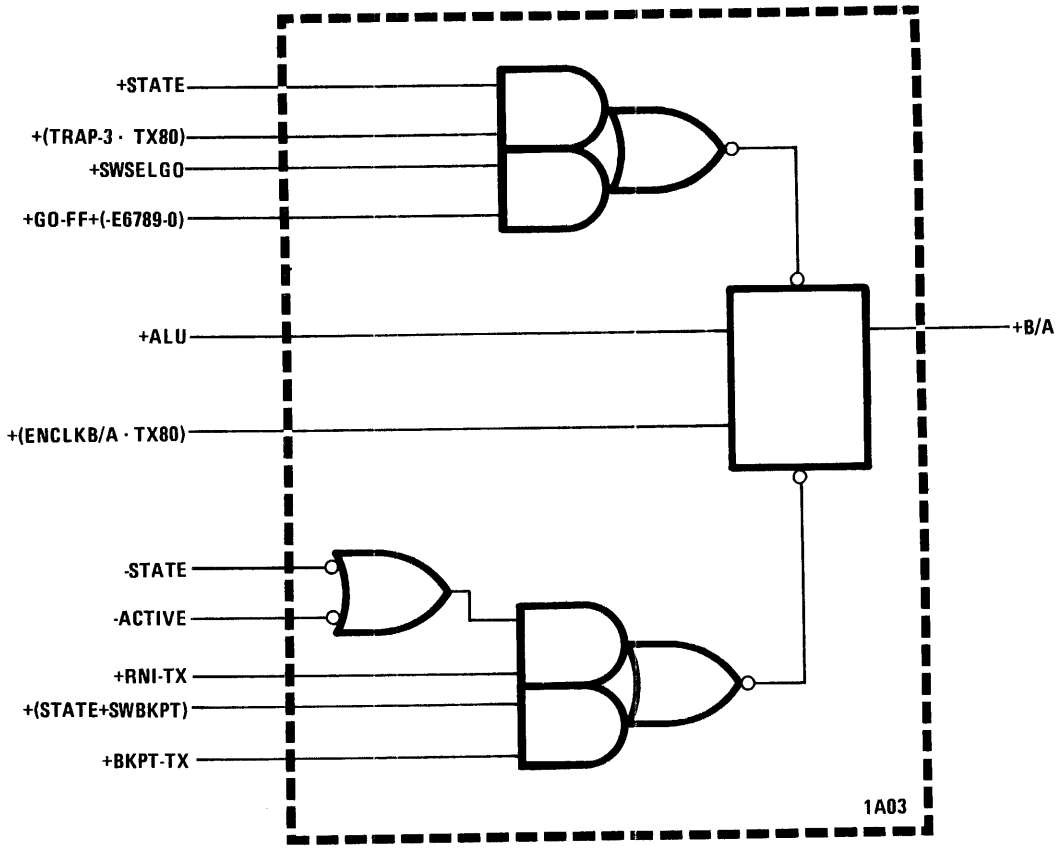


Figure 2-16. Busy Flip-Flops, Processors 5, 6, and 7

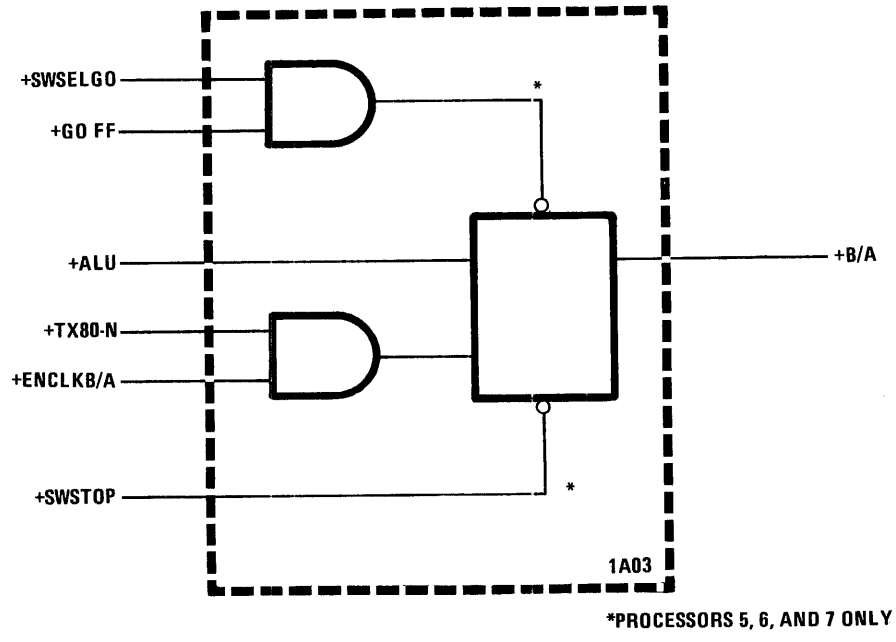


Figure 2-17. Active Flip-Flops, Processors 0 through 7

Resource Allocation Network

Scanner and Priority Logic

The Resource Allocation Network (RAN) assigns time slices to each requesting processor in accordance with its needs and its position in the queue relative to other requesting processors. The network consists of two sections: the scanner and priority logic, which grants time slices to a particular processor; and the time slice control logic, which sets up conditions to perform R (read P_{μ}, and F registers), E (execute μl's), and W (write Pμ and F registers) cycles during the processor's assigned time slice. A block diagram of the scanner and priority logic is shown in Figure 2-18. The logic consists of eight Resync flip-flops (one per processor), four Priority flip-flops (one for each of processors 0 through 3), and a priority encoder. The eight Resync flip-flops grant time slices to a requesting processor on a cyclic basis, wherein all requesting processors are granted one time slice in succession starting with the lowest numbered processor. Each Resync flip-flop is set at E160 time if its corresponding Busy flip-flop is set, meaning that the processor wants a time slice. The clock signal which sets each Resync flip-flop is ANDed with INH RSYN signals from each higher-numbered Resync flip-flop. These signals inhibit a Resync flip-flop from being set again until all higher numbered processors have been granted their time slices.

This method of granting time slices is referred to as the *scanner mode*, since the logic simply scans all processors requesting time slices and grants them in a cyclic sequence. Under certain conditions, however (such as imminent loss of data from an I/O device), the normal scanner mode can be overridden by a *priority mode* to grant a processor an out-of-sequence time slice if necessary. This is accomplished by the Priority flip-flops. Only processors 0 through 3 are provided with this override capability since they are used with I/O devices where rapid and timely data transfers are vital. These Priority flip-flops can be set by either one of two priority levels: *enable priority* and *invoke priority.* The *enable priority* level is implemented under software control by setting the EP (Enable Priority) bit position of the Control register in the ERF for that particular processor (CONTR-00 for processor 0, CONTR-01 for process 1, and so forth). This level allows an I/O processor to secure an out-of-sequence time slice if there is danger of losing data, provided that no lower numbered processor is also in an enable priority state. Indication of possible data loss is provided by the PRI signal from each I/O processor, which is ANDed with the corresponding CONTR bit. The *invoke priority* level is also implemented under software control by setting the IP (Invoke Priority) bit of the Control register in the ERF for that particular processor (CONTR-04 for processor 0, CONTR-05 for processor 1, and so forth). This level assures an I/O processor of at least one alternate time slice, (even though there is not necessarily danger of losing data) provided that no lower numbered processor is also in a priority mode. If neither EP or IP bit positions for a processor is set, the processor reverts to the scanner mode previously discussed.

The outputs of both the Resync and Priority flip-flops are fed to the *priority encoder*. This circuit makes the final determination of priority by generating the number of the processor assigned highest priority in BCD form. This BCD number, represented in the block diagram as outputs A, B, and C, are fed to the R, E, and W cycle logic. If no processor is requesting a time slice, output D goes high to generate either a null condition or allow the Panel to gain access to the system. The null condition schedules NOP operations during the interval that no processor is requesting time slices, unless the Panel desires entry to the system. For this condition, the Panel is treated by the priority logic as a ninth processor, with the lowest priority.

The REFRESH signal fed to the priority encoder is used to resolve a conflict between simultaneous requests for a *refresh cycle* and an *MS access cycle*. If a refresh request occurs in the absence of an MS access request, the refresh operation takes place in MS transparent to the rest of shared resources. However, if the refresh request occurs simultaneous with an MS access request, the refresh request is pre-empted by the MS access request. During the following major cycle, however, the refresh request pre-empts all other operations performed during that major cycle by setting up a null condition. This null condition effectively blocks the next processor in the queue from getting a time slice until the following major cycle, even if that processor was not going to access MS. Logic for generating this null condition is shown in Figure 2-19; associated timing is shown in Figure 2-115 located in the paragraph titled Main Storage. As the timing of Figure 2-115 shows, the Refresh Request flip-flop is set at E0 to generate REFRESH when the refresh counter reaches a count of 52. If an MS access request is not present (signal ACCESSEN is low), the refresh operation takes place on schedule and the Refresh Request flip-flop is cleared at E3. For this situation, the fact that REFRESH blocked other requests into the priority encoder while it was high had no effect since the signal dropped before E560, the *processor committed time*. If an MS access request is present (ACCESSEN is high), the Refresh Request flip-flop remains set past E3 to block all outputs from the priority encoder (outputs go high). At E560, the Null State flip-flop is set, which sets the Null flip-flop at E620 and finally causes ENCLKSM to go low.

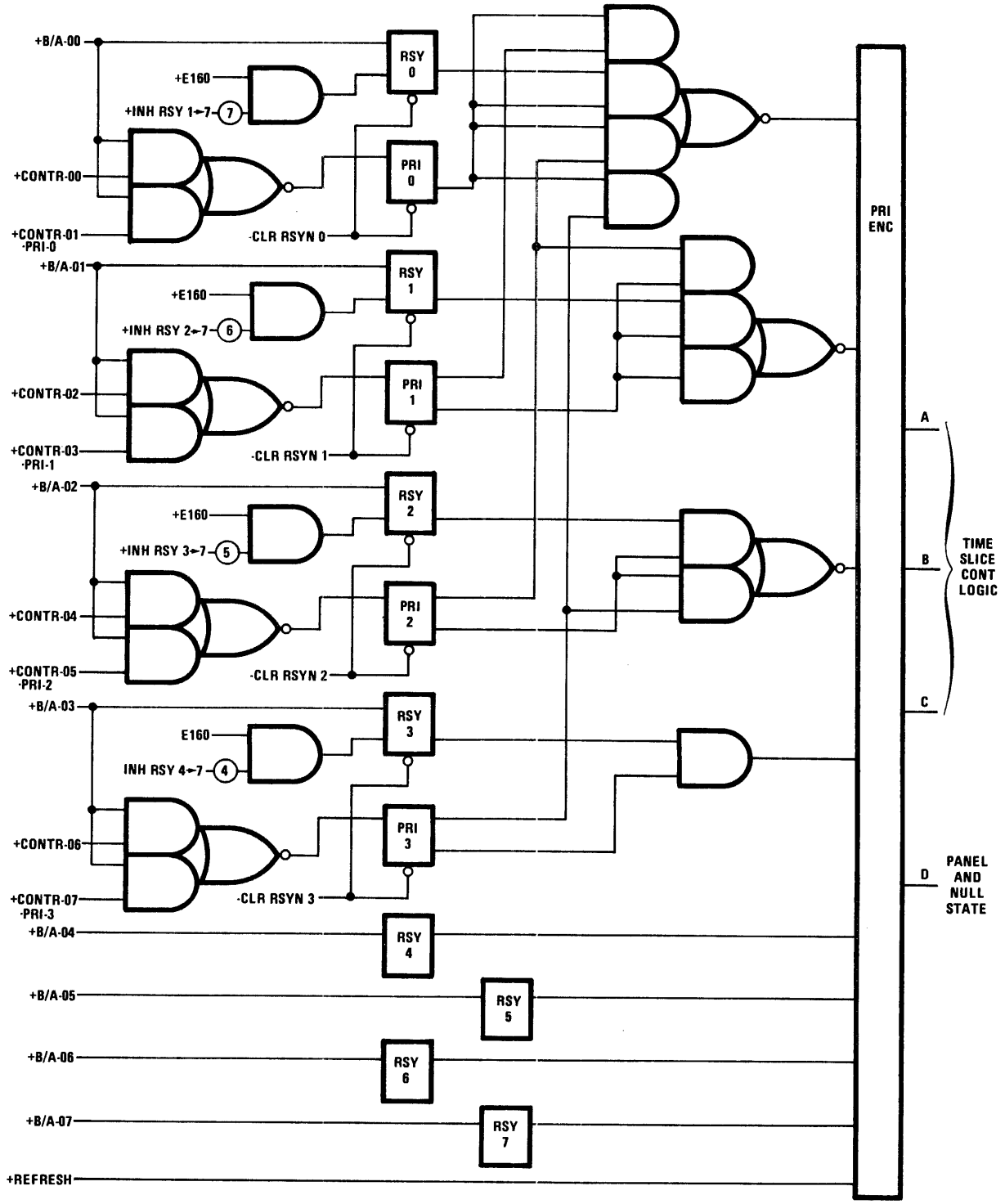


Figure 2-18. Scanner and Priority Logic

Table 2-2. Priority Logic Operations

Scanner (Revoke Priority) Mode

1) all processors requesting (all Busy FF's set)

   0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, . . .

2) processors 0, 1, and 7 requesting

   0, 1, 7, 0, 1, 7, . . .

   clear Busy 7 FF

   0, 1, 0, 1, 0, 1, . . .

Enable Priority Mode

1) Processors 1 through 7 requesting; processor 0 in **enable priority** mode

   1, 2, 3, 4, 5, 0, 6, 7, 1, 2, 3, 4, 5, 0, 6, 7

   PRI-0 Set PRI-0 Set

2) processors 2 through 7 requesting; processors 0 and 1 in **enable priority** mode

   2, 3, 4, 5, 1, 6, 7, 2, 3, 0, 4, 5, 6, 7 , . . .

   PRI-1 Set PRI-0 Set

Invoke Priority Mode

1) processors 0, 1, and 2 requesting; processor 1 in **invoke priority** mode

   1, 0, 1, 2, 1, 0, 1, 2, . . .

2) processors 0, 1, 2, and 3 requesting; processors 2 and 3 in **invoke priority** mode

   2, 3, 2, 3, 2, 3, . . .

Inhibiting this signal sets up the required null condition for the following major cycle by preventing updated μI addresses from being clocked into Sμ. The result is to prevent execution of a microprogram during the next major cycle (N+1) by issuing NOP's from CS. The refresh operation, therefore, can be performed during major cycle N+1. At E560 of this major cycle, the Null State flip-flop is cleared and ENCLKSM goes high. Execution of μI's for the processor that would normally have run during time slice N+1 will instead be performed during time slice N+2.

Because priority assigned to a processor depends on many different conditions (relative position of processor in queue, whether enable or invoke priority control bits are set, and so forth), it is useful to present several different examples showing how the priority encode logic operates under different conditions. These examples are listed in Table 2-2. The two examples shown when operating in the scanner mode grant time slices to each processor in a sequential manner. The sequence is interrupted only when a Busy flip-flop is cleared (or set); however, the interruption does not alter the cyclic nature of granting time slices. The two examples shown when granting in the enable priority mode indicate how a processor can obtain an out-of-sequence time slice when its PR1 signal is active, providing that priority is enabled for that processor via software. The two examples for the invoke priority mode show how processors 2 and 3 can lock out any other requesting processors in a lower priority mode, even processors 0 and 1, whose relative positions in the queue are higher.

Time Slice Control Logic

Logic to perform the R, E, and W portions of the active processor's time slice is shown in Figure 2-20. These portions of a time slice are initiated by corresponding registers, which are clocked with the encoded processor state number at the times necessary to start these portions. The Read register, clocked at E560 of the previous time slice, initiate signals to read the contents of Pμ and F for the active processor. (Time E560 is

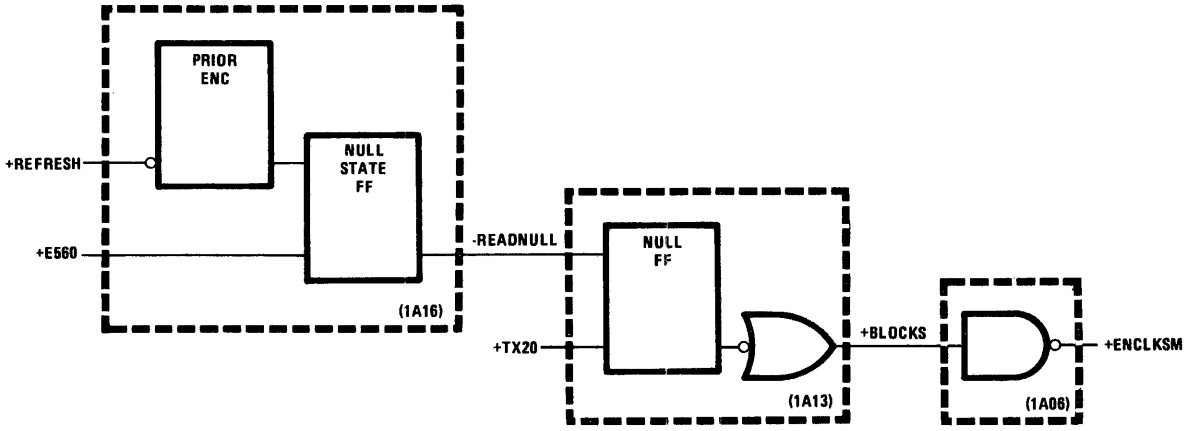


Figure 2-19. Null Condition During MS Refresh

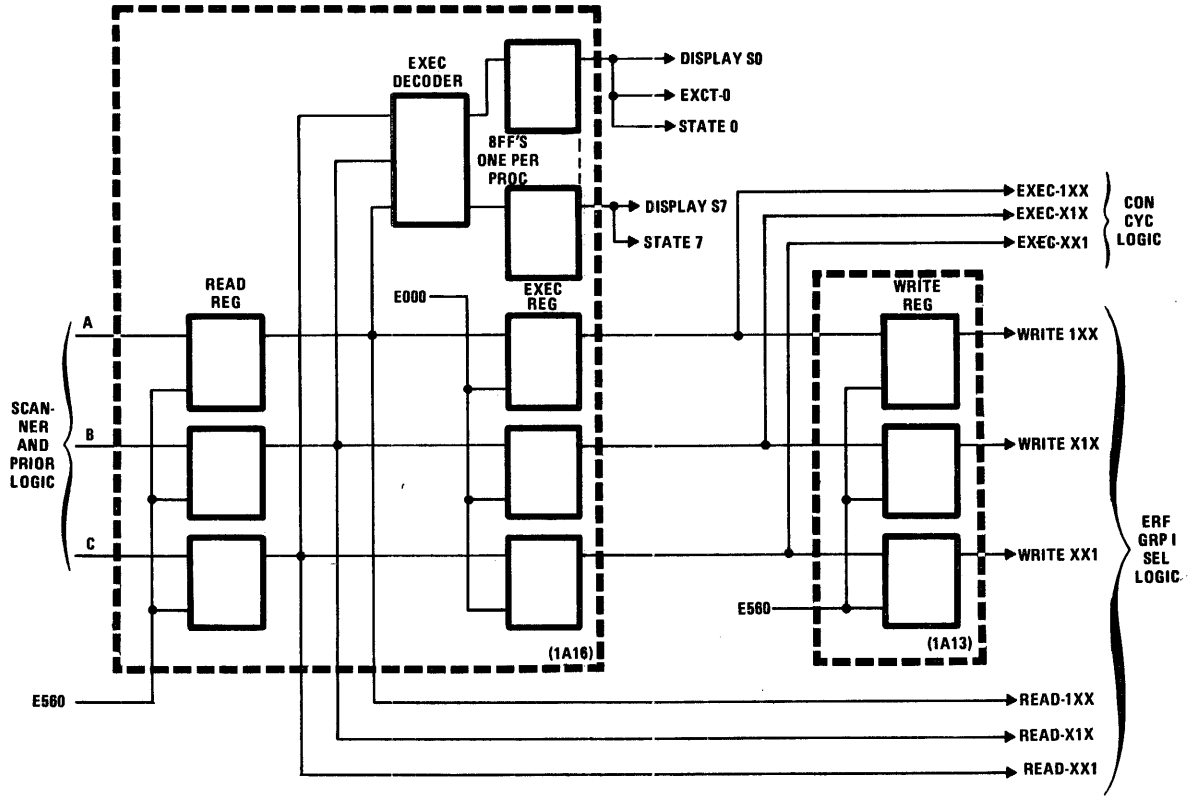


Figure 2-20. Time Slice Control Logic

considered to be the *processor committed time*, at which time the priority encoder logic is committed to grant a time slice to a particular processor. This committal time is in contrast to any time preceding this point, during which the priority logic may change or abort a processor request.) These two operations occur during two minor cycles, identified as R0 and R1. (Recall that these two minor cycles do not exist as such. They occur during E6 and E7 of the previous time slice.) This is done by routing the three READ signals to the ERF group I select logic.

The Execute register, clocked with the processor state number at E000, is used for two purposes: (1) Its contents are routed to the execute decoder logic, which decodes the processor number in BCD form to set one of eight Execute flip-flops. The flip-flop, in turn, generates two outputs designated DISPLYS and STATE. Signal DISPLYS is used to light the corresponding PROCESSOR ACTIVITY DISPLAY indicator on the Panel. Signal STATE is used by the Busy register to conditionally set or clear a particular Busy flip-flop. In addition, the Execute flip-flops associated with processors 0 through 3 generate an EXCT signal, which is returned to the corresponding I/O processor as an acknowledge that it is starting a time slice. (2) The Execute register contents are also fed to the Consecutive Cycle (CC) logic to make the Read register/ Execute register comparison required as a prerequisite to starting consecutive-cycling.

The Write register, clocked at E560, initiates the W portion of a time slice required to store the next μI in Pμ and the MLI presently being executed in F. This is accomplished in a manner similar to the Read register, by sending WRITE address signals to the ERF group I select logic. These signals select Pμ and F associated with the active processor for storing the above quantities.

Time Slicing

Normal Operation

Normal time slicing consists of granting time slices to processors in order of their priority, as discussed in the previous paragraphs. This normal condition is illustrated in Figure 2-21, which shows granting of time slices for processors 0, 1, and 6. The timing assumes an initial starting condition where no processors were executing prior to requests from processors 0, 1, and 6. The first time slice, therefore, is set up as a null since processor 0 (the first processor to be granted a time slice) will not begin executing until the following time slice (recall that priority for a particular processor is always determined one time slice before the processor begins executing). The requests from all three processors set their corresponding Busy flip-flop simultaneously at E080.

At E160, the Busy flip-flop contents are clocked into Resync flip-flops 0, 1, and 6 in the scanner and priority logic. The requests are scanned by the logic, which determines that processor 0 will be granted the first time slice. The processor number is sent to the Read register at E560 in preparation for initiating the R0 and R1 cycles of processor 0's time slice. These cycles read the address of the first μI to be executed during the time slice from Pμ and the MLI, of which the μI is a part, from F_{RF}. At E600, the Resync flip-flop for processor 0 is cleared to remove this processor from the job queue (since its request for a time slice has been honored). Clearing of the Resync flip-flop at this time is a reflexive action and occurs during E6 of every time slice, as shown in Figures 2-21 and 2-22. The State 0 flip-flop is set at E000. At the same time, the processor number in the Read register is transferred to the Execute register. At this point, the execute time slice for processor 0 begins.

Processor 0 reads the first μI from CS at the address specified by (Sμ) during E0, and executes this μI at E0 and the beginning of E1. At E160 of the processor 0 execute time slice, the Resync flip-flops of the scanner and priority logic are again scanned to note that processors 1 and 6 are still waiting for time slices. The priority encoder determines that processor 1 will get the next time slice and at E560, the encoded processor number is clocked into the Read register. At this time also, the encoded number for processor 0 is clocked into the Write register from the Execute register to initiate the W0 and W2 cycles for processor 0. During these cycles, the address of the first μI to be executed during the next time slice assigned to processor 0 is transferred from Pb or Pp to Sμ and the MLI currently being executed is stored into F_{RF}. From this point on, processors 1 and 6 time slices are handled in exactly the same way as for processor 0, as will time slices for all subsequent processors that may enter the queue. During normal operation, therefore, the only difference in granting time slices to a requesting processor is the processor number, which determines its position in the job queue.

Consecutive Cycle Operation

Consecutive cycle (CC) operation is a means of increasing processing efficiency when only one processor is requesting time slices. During normal operation, one processor requesting time slices can execute only during every other time slice. The reason is due to the overlap of R and W cycles of successive time slices, as shown in

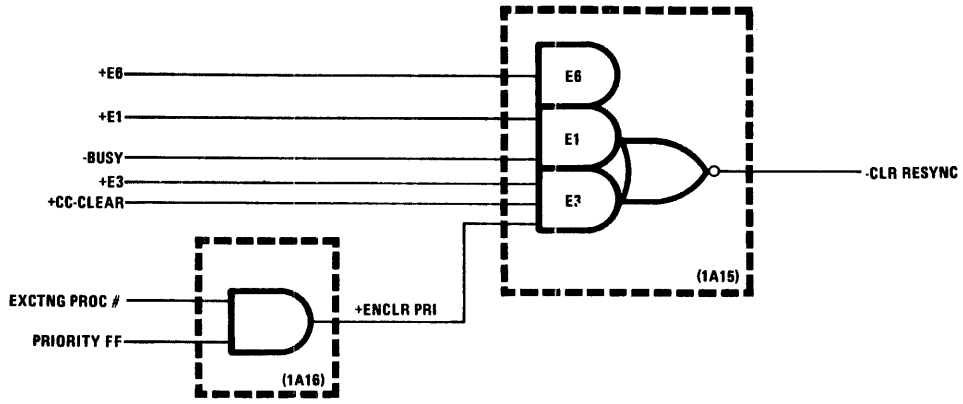


Figure 2-21. Normal Priority Timing, Three Processors in Queue

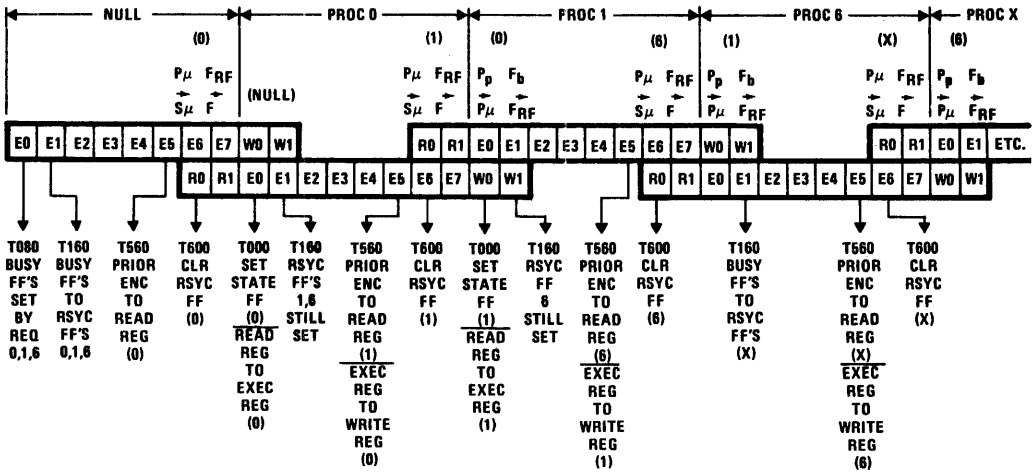


Figure 2-22. Clear Resync Flip-Flop Logic

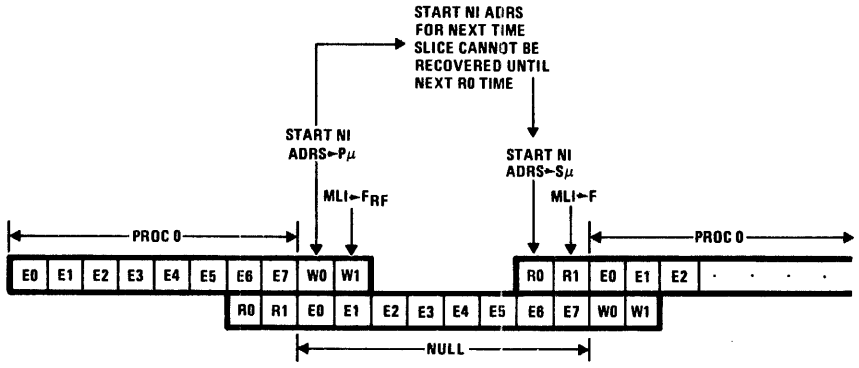


Figure 2-23. Alternate Time Slices for One Processor Requesting

Figure 2-23. Assume that processor 0 is the only processor requesting time slices. Upon completion of the last μI at E7, it stores the address of the first μI to be executed during its next assigned time slice in Pμ during W0 and the present MLI in F_{RF} during W1. If another processor was requesting time slices, it would have already read its starting μI address and MLI from Pμ and F_{RF} cycles before, during the R0 and R1 cycles of the next assigned time slice. Since only one processor is requesting, it cannot retrieve this information until the following R0 and R1 cycles. Therefore, the time slice following the one that processor 0 was assigned must be nulled out. This null is implemented by issuing NOP's from CS. The result is only a 50% utilization of shared resources (only every other time slice can be used) when only one processor is requesting.

Timing for one processor requesting a time slice and not enabled; for CC generation is shown in Figure 2-24. The first time slice is a null to allow priority to set up conditions so that processor 0 can execute during the next time slice. During the next time slice, processor 0 executes the μI's while priority determines if there are any other processors requesting time slices. Since there are no others, it prepares to grant a second time slice to processor 0 by clocking processor number 0 into the execute register a second time. Prior to beginning this second time slice, however, the processor is interrogated to see whether it has been enabled for CC operation. The CC mode of operation depends on two conditions being present: (1) no other processor is requesting time slices, and (2) the CC bit in the Control Register corresponding to the single processor is set. Interrogating for these two conditions is performed by the logic of Figure 2-25. This logic checks for the first condition by comparing the contents of the Read and Execute registers of the time slice control logic at E6 for equality. During normal operation (more than one processor requesting), the contents of these registers will not be the same at E6 since the Read register will already have been loaded with the number of the next processor in the queue. During CC operation, however, only one processor is in the queue so the Read register contents will not have been changed. The second condition is checked by the CC Clear flip-flop, which sets by ANDing the processor number in the Read register with the corresponding CC bit in the Control register. If both conditions are present, signal $\overline{\text{RD=EXEC}}$ is generated. A second signal, $\overline{\text{CC-ENABLE}}$ is also generated. This signal indicates only that the processor is enabled for CC and not necessarily that it is the only one in the queue. For this present example, $\overline{\text{CC-ENABLE}}$ is high since processor 0 is not enabled for CC. The signal generates $\overline{\text{ABANDCC}}$ at E640, which sets the Null flip-flop in the priority logic. Setting this flip-flop blocks accesses to CS for all of the next time slices, which sets up the required null time slice.

Enabling a single processor to operate in the CC mode is accomplished by three operations: (1) At the end of the time slice, Sμ is loaded with the contents of Pp instead of Pμ. Since Pp contains the starting μI address for the present processor, assuming that it could not get another time slice until some later time, the processor can resume execution during the following time slice at the same point that it left off during the present time slice. Reading this starting μI address from Pp and the resultant μI from CS is performed during two extra minor cycles inserted between E7 of the present time slice and E0 of the following time slice, designated E8 and E9. During E8, the contents of Pp are routed back to Sμ. During E9, the μI located in CS at the address contained in Sμ is read for execution during the following E0. (2) The read Pμ and F operation associated with the R portion of the following time slice is inhibited, to prevent reading the starting μI address and MLI read by the same processor at the beginning of the present time slice. This is done by inhibiting selection of the Group I ERF containing Pμ and F_{RF}. (3) Sμ is blocked during E6 and E7 and Fμ is blocked during E8 and E9. These blocks are necessary to prevent erroneous μI addresses from making CS references prior to the correct starting μI address being loaded into Sμ at E8. The sequence of events for implementing these operations is described in the following paragraphs.

Assume now that processor 0 is the only processor requesting time slices and is also enabled for CC. Timing for this example is shown in Figure 2-26. Like Figure 2-24, it is assumed that no processor was executing before a request was made by processor 0 for time slices. Therefore, the first time slice shown is a null. The sequence of events during this null time slice is the same as that for Figure 2-23. At E0 of the next time slice, the CC Clear flip-flop is set. Essentially, this flip-flop is used to get out of the CC mode when the CC bit in the Control register is cleared. Therefore, it is set when the processor enters the CC mode and remains set until the CC bit is cleared. When set, the $\overline{\text{CC-CLEAR}}$ output from the clear side goes low, as shown in Figure 2-25. Since this flip-flop is clocked at E060 of every time slice, it also serves the purpose of *snapshotting* the CC bit every time slice. This is the only time during a time slice that the CC bit is snapshotted and commits the time slice to react accordingly. In other words, even if the CC bit was to be cleared before the end of the time slice, the time slice would be committed to the CC mode even if not really necessary.

At E6, the Null flip-flop is set and the CC flip-flop, which is normally set if not in the CC mode, is cleared. Logic for these flip-flops and associated logic generating other

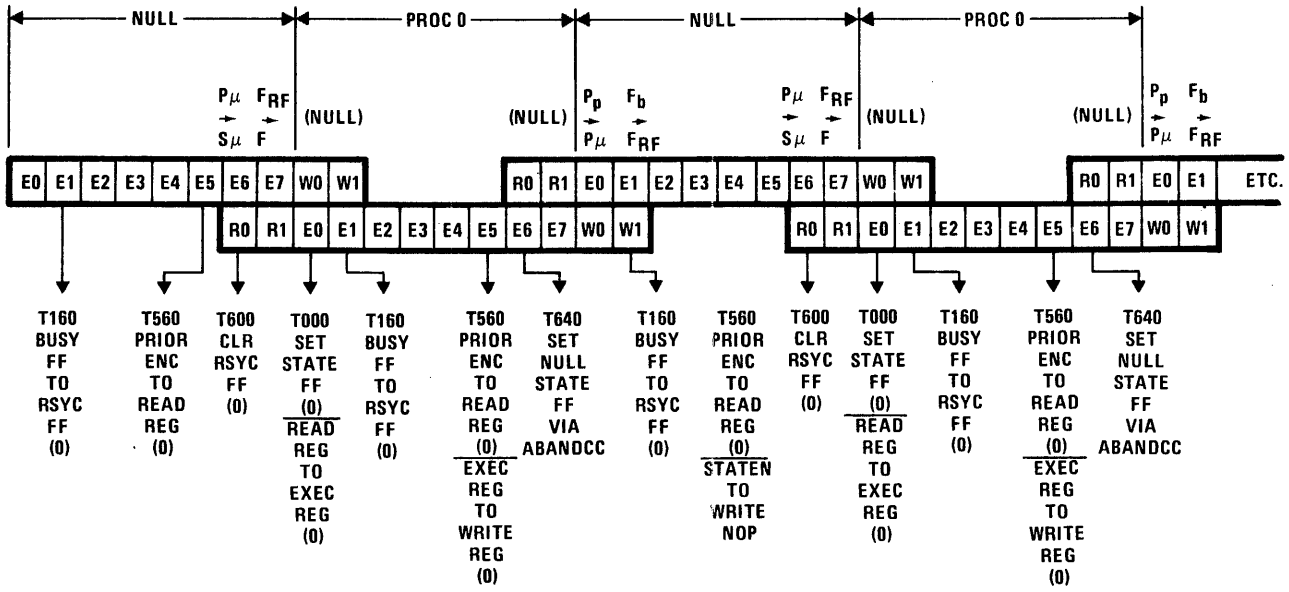


Figure 2-24. One Processor in Queue, Not Enabled for CC

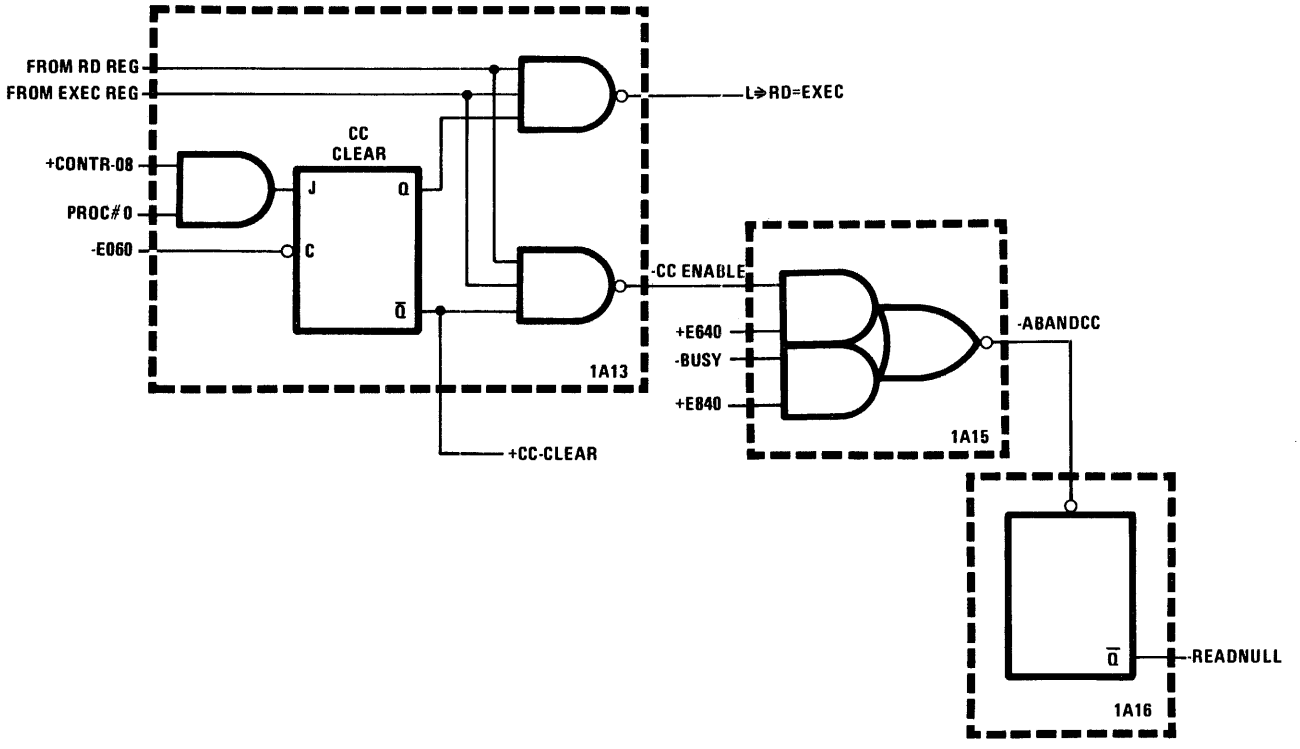


Figure 2-25. Interrogation Logic

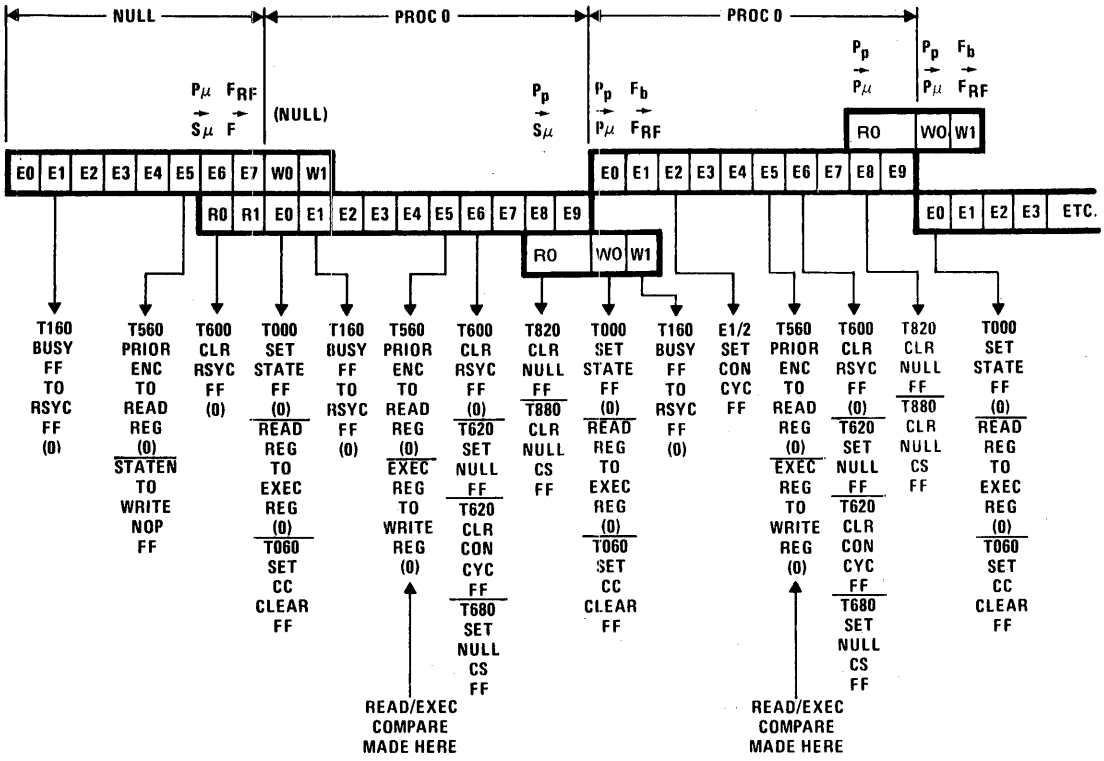


Figure 2-26. One Processor in Queue, Enabled for CC

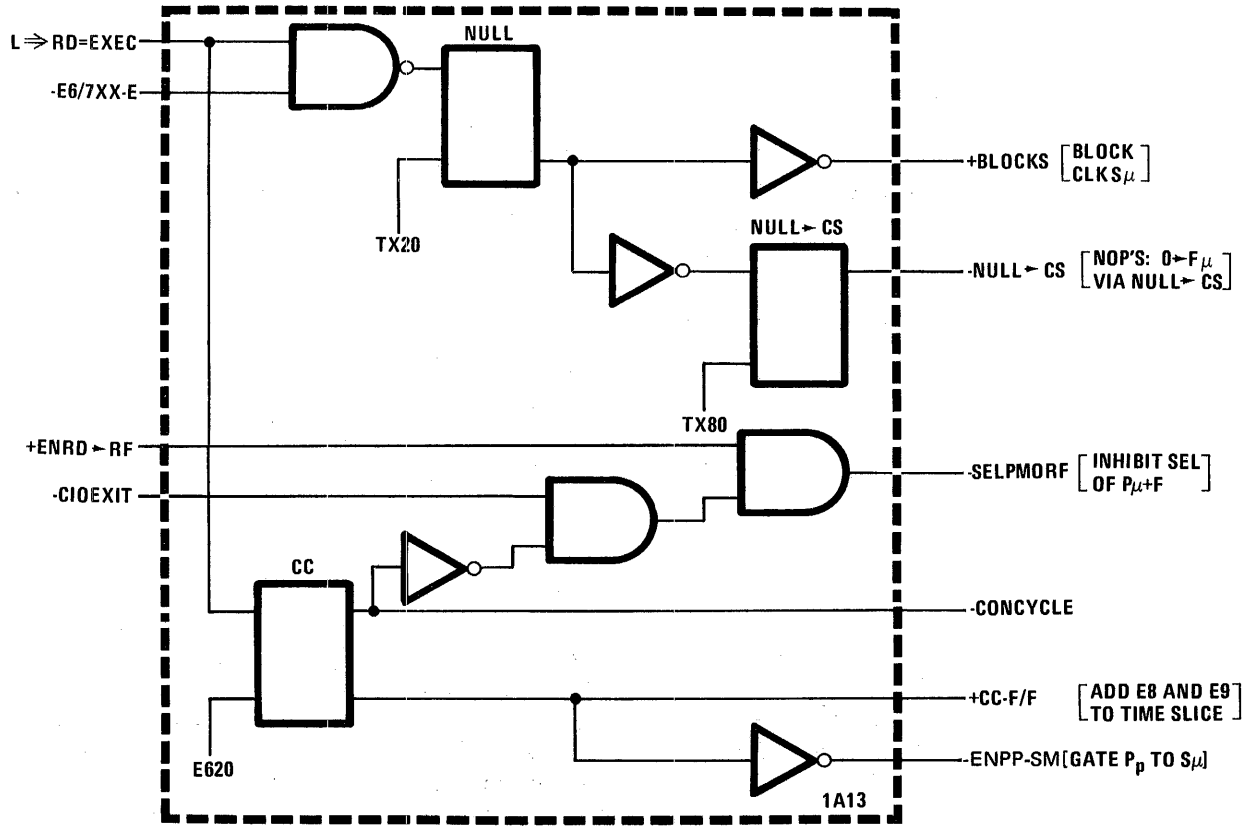


Figure 2-27. Generation of CC Mode Signals

signals necessary for CC generation is shown in Figure 2-27. Corresponding timing for these signals is shown in Figure 2-28. The two flip-flops are set and cleared, respectively, at E620 upon occurrence of a low $\overline{\text{RD=EXEC}}$ signal. (Recall from Figure 2-24 that this signal will be low for a single processor enabled for CC.)

Setting the Null flip-flop generates BLOCKS, which blocks clocking of Sμ at E680 and E780. The clear side of this flip-flop is also used to set the Null→CS flip-flop, which generates NULL→CS. This signal is sent to CS to effectively *shut off* CS during E8 and E9. The result is to clock "0's" into Fμ at E800 and E900. Clearing the CC flip-flop generates the following four signals which perform the indicated operations: (1) Signal $\overline{\text{SELPMORF}}$ is driven high to inhibit selection of the IC element containing Pμ and F_{RF} in the ERF. Since the same processor is executing, nothing is needed for another processor from the ERF. (2) Signal $\overline{\text{CONCYCLE}}$ is generated to hold $\overline{\text{SELFH/PL}}$ low. This is done for the special case when a CIO μI is executed in CC as part of an I/O data transfer loop, and the condition for exiting from the loop is met. For this situation, the address of the next μI is put into Pμ as part of the μI execution. Therefore, it must be loaded into Sμ from Pμ. This is accomplished by signal $\overline{\text{CIOEXIT}}$ in Figure 2-27. If the exit condition is met, $\overline{\text{CIOEXIT}}$ drives $\overline{\text{SELPMORF}}$ low to enable selection of Pμ by $\overline{\text{SELFH/PL}}$ to load Sμ. (3) Signal CC-F/F is generated to prevent the excursions counter from recycling back to E0 after generating E7. The result is to add minor cycles E8 and E9 to the timing chain. (4) Signal $\overline{\text{ENPP→SM}}$ is generated to enable gating the contents of Pp, containing the starting μI address of the next time slice back to Sμ. This address gating takes place during E8. At this point, processor 0 can begin executing another time slice. Successive time slices will be executed in exactly the same manner until a condition arises to remove the processor from the CC mode. These conditions are (1) the processor's CC bit is cleared, (2) another processor requests time slices, or (3) the processor's Busy flip-flop is cleared.

Removing a single processor from the CC mode by clearing the CC bit position is accomplished by the CC Clear flip-flop, the same as for a single processor which began executing with its CC bit initially cleared. At E640, the evaluation is made to determine if the CC bit of a single processor is still set. If not, the Read Null flip-flop of the RAN is set via $\overline{\text{ABANDCC}}$ to null out the following time slice. A special case is stopping the CC-enabled processor completely because its Busy flip-flop is cleared. Clearing the Busy flip-flop indicates that the task performed by the processor is completed. Stopping the processor is done by means of $\overline{\text{ABANDCC}}$, as shown in Figure 2-25. Clearing the Busy flip-flop, however, is done at E7 which occurs after $\overline{\text{ABANDCC}}$ will have been generated by the $\overline{\text{CC-ENABLE}}$ signal. For this case, therefore, $\overline{\text{ABANDCC}}$ is generated at E840 by $\overline{\text{BUSY}}$, which indicates that the present processor has completed its task. Generating $\overline{\text{ABANDCC}}$ in this manner prevents the processor from being trapped in a condition in which it could not turn itself off. The result is to generate a null during the following time slice to allow the next processor in the queue (if any) to read up its starting μI address. The processor whose Busy flip-flop has cleared is removed from the queue by clearing its Resync flip-flop in the priority logic at the next E1, as shown in Figure 2-22.

A request from a second processor for time slices removes the single processor enabled for CC from the CC mode by preventing $\overline{\text{RD=EXEC}}$ from being generated. This prevents the Null and CC flip-flop from being set, which inhibits the associated CC mode signals. A special situation arises, however, when the second processor is of lower priority than the one enabled for CC. Under this condition, the processor enabled for CC will run for one additional time slice after the lower priority processor enters the queue. The reason is that the CC snapshot logic will have determined that the processor enabled for CC should execute in the CC mode before the scanner and priority logic recognizes that another processor has entered the queue. This situation is shown in the timing of Figure 2-29. Assume that initially, only processor 0 is requesting. The first time slice is a null to set up processor 0, enabled for CC operation, to run during the next time slice. During this next time slice, processor 6, not enabled for CC, enters the queue at E160. At E050, however, the CC snapshot logic has already determined that processor 0 will run in the CC mode for the present time slice. One minor cycle later, at E160, the scanner and priority logic determines that both processor 0 and 6 are in the queue. Since this determination is not made until after the CC snapshot decided that processor 0 could run in the CC mode (only processor in queue and CC bit set), the logic assumes that both processors 0 and 6 have just entered the queue and are to be granted priority. In accordance with these processor numbers, the result is that processor 0 is granted a second time slice solely on the basis of its position in the queue. After completion of this time slice, the CC snapshot logic is disabled and priority is granted in the normal manner so that processors 0 and 6 get alternate time slices. (The above sequence of events is modified somewhat if the processor running in CC is also in the invoke priority mode. Under these circumstances, the RAN will prevent the second processor from entering the queue even for alternate time slice. This level of priority — CC mode and invoke priority mode — is the highest level of priority available to a processor and assures that any of processors 0 through 3 in this mode will absolutely lock out the other three processors for as long as the present processor is in this mode.)

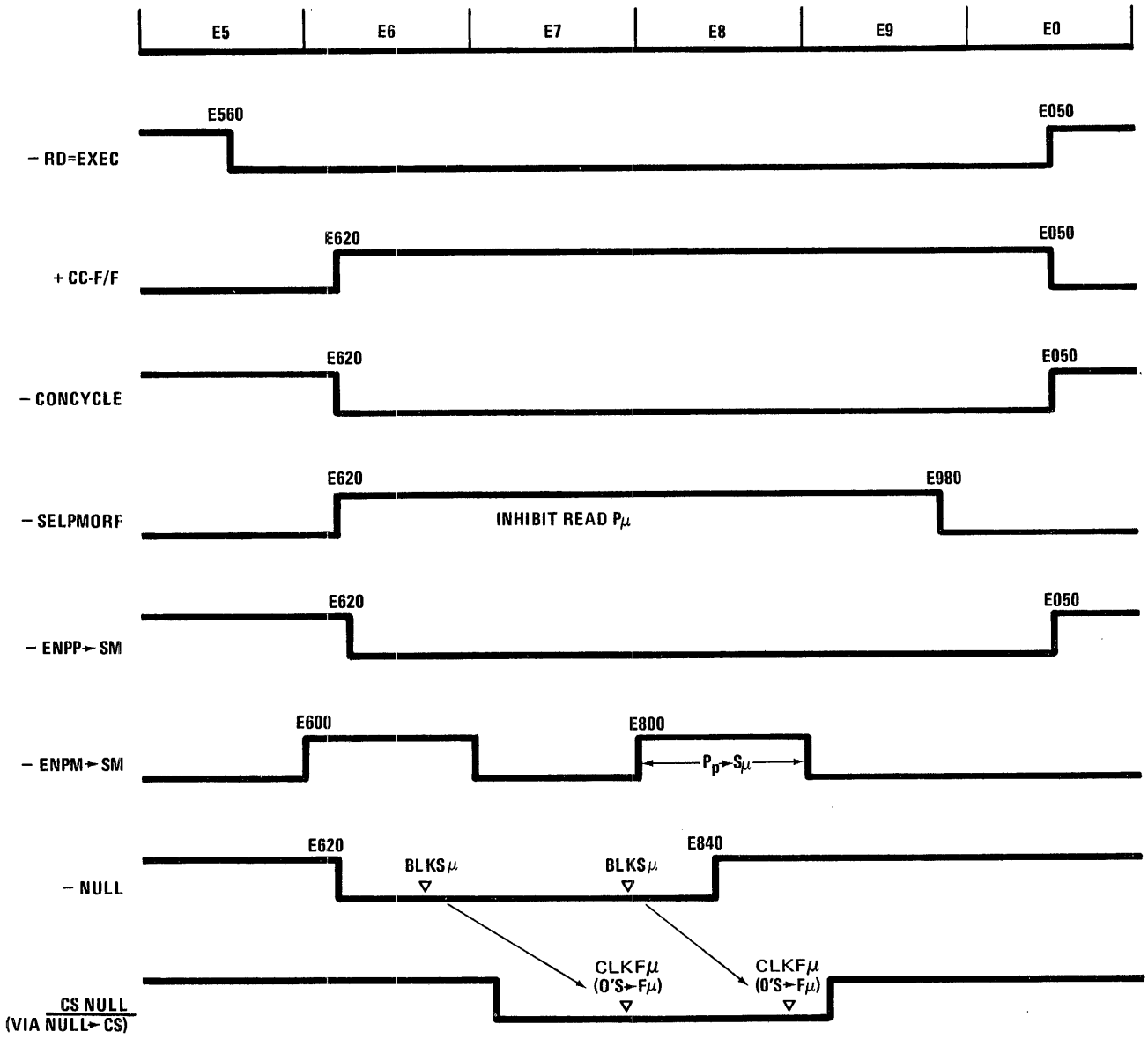


Figure 2-28. CC Mode Signal Timing

As discussed previously, the Resync flip-flop for a processor granted a time slice is normally cleared at E6 to remove the processor from the queue. The flip-flop stays cleared until all lower-numbered processors in the queue are serviced. One exception to clearing the Resync flip-flop at E6 is the case of a processor whose Busy flip-flop was cleared at E7. In this case, the Resync flip-flop is cleared at the next E1. A second exception is the case of two or more processors in the queue, one of which is in the priority mode. For this situation, the Resync flip-flop of the processor enabled for priority must be cleared prior to E560 to avoid locking out all processors not enabled for priority. Timing for this condition is shown in Figure 2-30. Assume that processors 0 and 6 are in the queue and processor 0 is in the priority mode (Priority flip-flop set). At E160 of the null time slice, the Resync flip-flop for processor 0 is set to initiate the execute time slice. Because processor 0 is in a priority condition, its Resync flip-flop is set again at the next E160. This second resync request must be cleared before E560 to avoid generating $\overline{\text{ABANDCC}}$ because both Read and Execute registers contain the same processor number (0). If $\overline{\text{ABANDCC}}$ were to be generated in this manner, it would simulate the same condition as a single processor requesting, not enabled for CC, by generating a null during the next time slice. The effect would be to lock out all requests for these processors in the queue not in a priority condition. This effect is inhibited by clearing the Resync flip-flop for processor 0 at E3 via $\overline{\text{CLR RESYNC}}$, as shown in Figure 2-22. The signal is generated by CC-CLEAR, indicating the processor is not enabled for CC, and ENCLR PRI. Signal ENCLR PRI, in turn, is generated if the present processor executing (EXCTING PROC #) is in a priority condition (PRIORITY FF). Signal EXCTING PROC # obtained from the execute decoder of the scanner and priority logic and PRIORITY FF is obtained from the corresponding processor Priority flip-flop.

CONTROL

The following paragraphs discuss various control circuits that are directly related to, but do not logically fit into other sections of the shared resources information. These are: *skip control, branch control, cycle delay* logic, and *system reset* logic. The *skip control* logic evaluates skip conditions and implements the skip operation for the eight *skip* μI's, identified as the 5,X,X μI's. Four of the eight skips (5,X,0 μI's) enable a skip depending on the results of an operand in Aμ. The other four skips (5,X,1 μI's) effect a skip depending on the results of a compare between Aμ and Bμ. The *branch control* logic specifically deals with generating a final branch address for the FNJ, FRJ, FZJ, RNI, and JMP μI's. These branch μI's form their address from the contents of Sμ and a partial branch address formed by branch address translation peculiar to each branch μI. In addition, this logic implements the branch-to-next-CS module anomaly associated with all branch μI's.

The *cycle delay* logic is used to delay execution of a SUM, DSUM, CMP, or CMU (2,X) μI for one minor cycle if programmed immediately following any μI that feeds data into either Aμ or Bμ. Data loaded into Aμ or Bμ by such a μI takes almost one cycle to propagate through the ALU.

Therefore, the results of such a μI are not available for the 2,X μI to process until one minor cycle after the μI that loaded Aμ or Bμ. The *system reset* logic performs a System Reset on the system initiated by a power-on condition, performing a Reset/Load or Autoload operation, or pressing the SYSTEM RESET pushbutton on the System Control Panel.

Skip Control

A simplified diagram of the *skip control* logic is shown in Figure 2-31. The results of the Aμ/Bμ compares made in the ALU are fed to the skip evaluation logic. This logic combines the Aμ/Bμ compare results with the skip μI sub-operation codes (bit positions S_0 and S_1) to determine if the skip condition defined for the skip μI was met. The skip evaluation logic is enabled by the Not Skip flip-flop. This flip-flop is normally in a set condition to block the μI following the skip μI from being clocked into Fμ if the skip condition is met. This action essentially skips the next μI by setting up a null condition for the next minor cycle. The flip-flop is cleared for 100 nsec during the skipped μI minor cycle, however, to enable the μI following the skipped μI to be clocked into Fμ. Timing for this sequence of events is shown in Figure 2-32. This figure shows the skip μI (NI) being skipped during E3, and the μI following NI (NI+1) being executed at E4. The flip-flop is cleared by SKIP from the skip generate logic as a result of a low output from the skip evaluate logic and the rest of the skip μI operation code bits (5, X, 0 or 5, X, 1). These outputs are also sent to the clock Fμ logic to inhibit ENCLKFM at the start of E3. At the end of E3, the Not Skip flip-flop is set again which disables the skip evaluate logic. This allows the NI+1 μI to be clocked into Fμ for translation and subsequent execution.

Execution of a skip μI at E7 differs from execution at E0 through E6 because the following μI to be skipped will not appear until the next assigned time slice. If the processor is not operating in the Consecutive Cycle (CC) mode, this next time slice will not be granted until several other processors have been allocated their requested time slices. It is necessary, therefore, to store away skip status information in a register until the next assigned time slice. This is done by routing $\overline{\text{SKIP}}$ from the skip generate logic to the Skip Status flip-flop, which is set at E000. The output of this flip-flop, in turn, is stored in the Skip Status

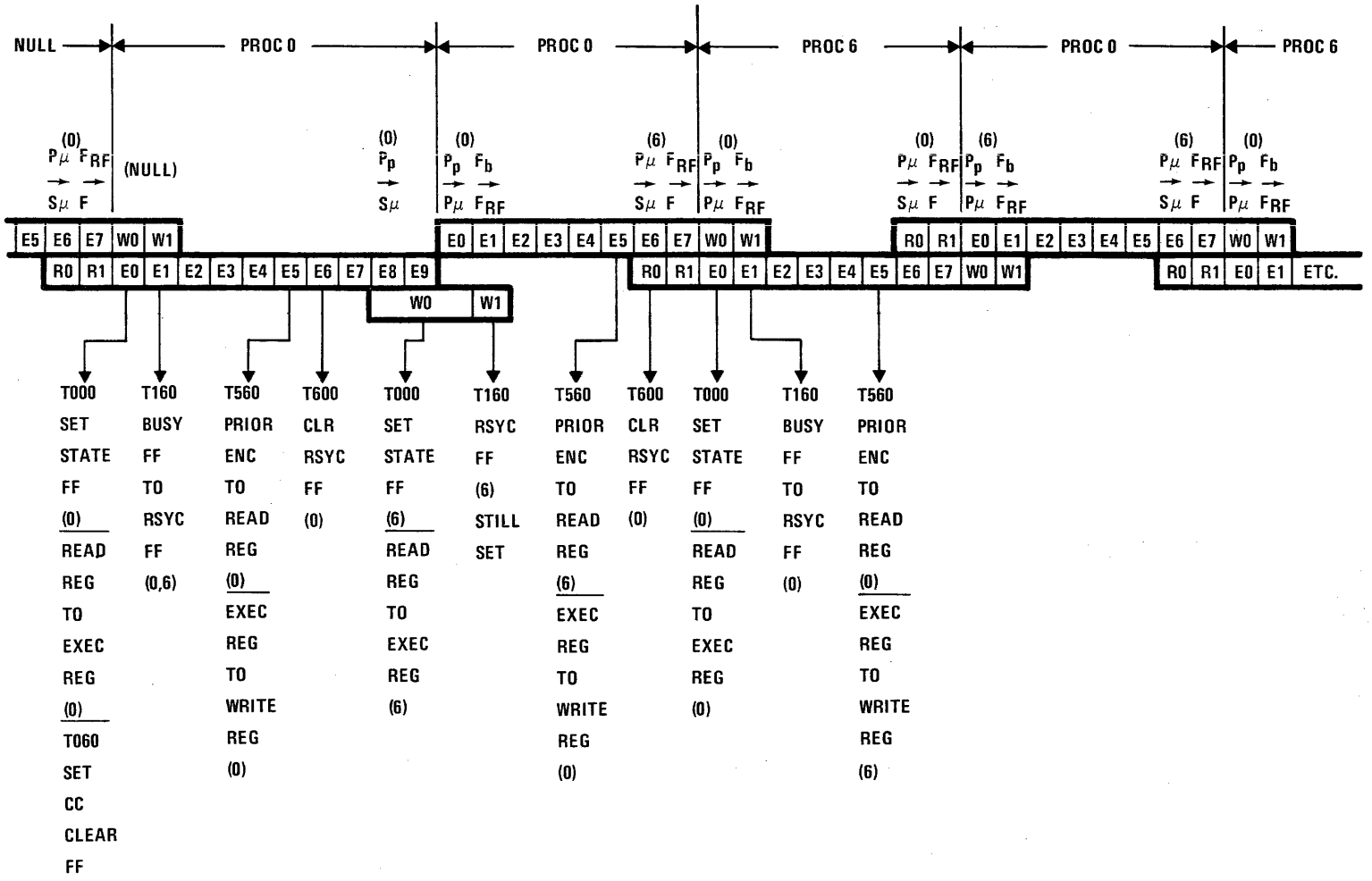


Figure 2-29. One Processor in Queue, Enabled for CC, Another Processor Enters Queue

register during W0. This register is addressed and written into by the same signals that store the starting μI address in Pμ, via the ERFG1, SELFH/PL, and EFIRH/WL signals. Timing for this skip status write operation is shown in part a of Figure 2-33. During R0 of the next time slice, the skip status information is read from the register and routed to the Null CS flip-flop which, in turn, is set by skip status at E680. This flip-flop is used to skip the NI μI at E0 by preventing it from being read from CS. (This μI skip differs from the case discussed in the previous paragraphs in that the *abort* can be effected before the μI is even read from CS. For the previous case, the μI to be skipped had already been read from CS before the skip evaluation logic detected that a skip should be made.) If the skip μI is executed at E7 by a processor running in the CC mode, it is not necessary to store skip status information since the next time slice assigned to the processor will follow the present time slice. Timing for this situation is shown in part b of Figure 2-33. Signal $\overline{\text{SKIP}}$ is generated as before to set the Skip Status flip-flop. Since E0 of the next time slice must be delayed two minor cycles when operating in the CC mode to accommodate E8 and E9, the Skip flip-flop is set for this case at E800. The Null → CS flip-flop is set, in turn, at E880 to block reading of NI from CS during E9.

Branch Control

The *branch control* logic operates on two classes of partial branch* μI's: the FNJ, FRJ, FZJ, RNI μI's and the JMP μI. The main difference between the two classes of μI's is how much of Sμ is used to form the branch address. The first class of partial branches uses only bits 2 and 3 of Sμ with bits 4 through 15 derived from a corresponding jump address generated by the particular μI. The JMP μI uses bits 2 through 7 of Sμ with bits 8 through 15 generated by the JMP μI. In both cases, the Sμ bits are under hardware control as opposed to μI control. Furthermore, the branch control logic determines how the branch address will be used to form a starting μI address (since all branch μI's are blockpoint μI's). Specifically, this means using either P_p as a holding register if the branch μI is executed during E0 through E6, or P_p if the μI is executed at E7. Depending on whether P_p or Pb is used as a holding register, the branch-to-next-CS storing unit* anomaly may result. This anomaly causes the branch to

*Partial branch μI's are so identified because, at the most, they replace only 14 of the 16 address bits in Sμ (bits 0 and 1 are not changed). This is in contrast to the full branch μI's (CLR, STA, STB, and AND when X designates Pμ). This group of μI's replaces all 16 bits of Sμ when the X designator of these μI's specifies Pμ since Sμ is in the same path.

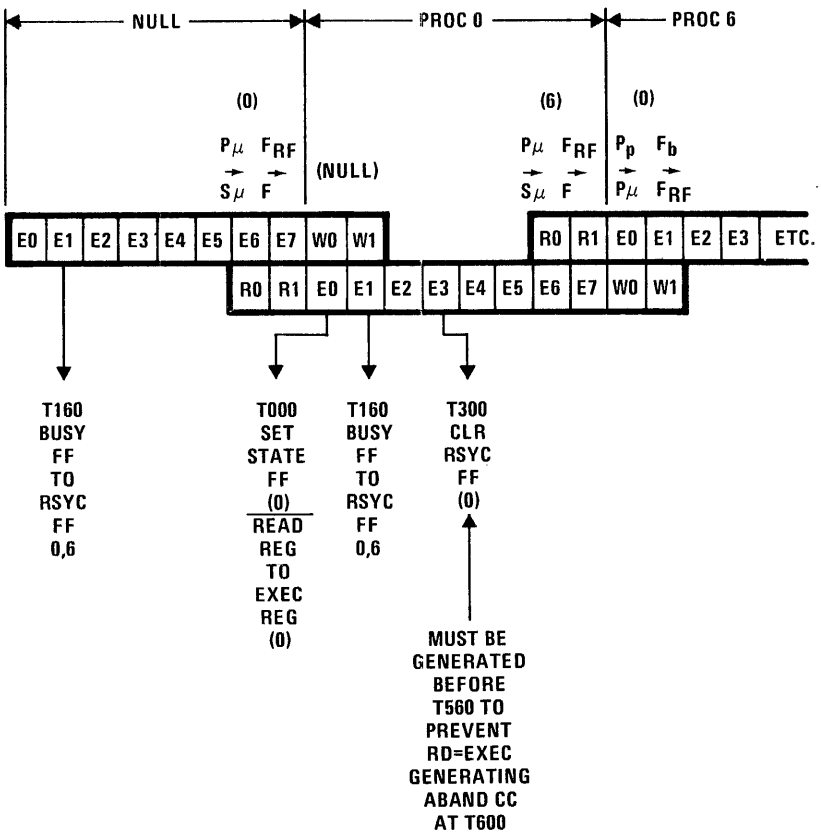


Figure 2-30. Two Processors in Queue, One Enabled for Priority

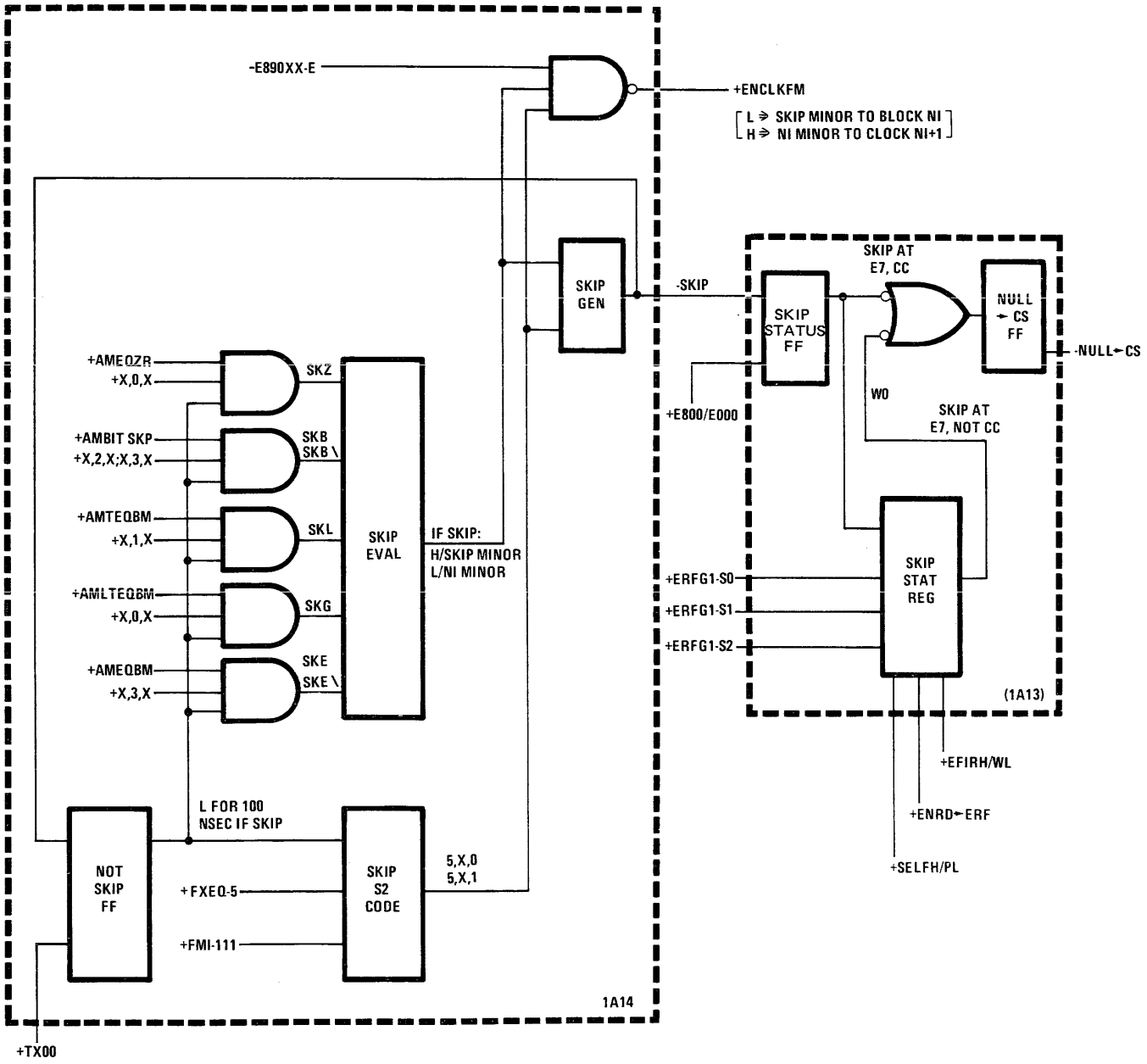


Figure 2-31. Skip Control Logic

take place within the same 4096-word unit in which the branch μI is located, except for the following two cases in which case the branch is made to the following unit: (1) If the branch μI occupies the last location of a unit, or (2) If the branch μI occupies the next-to-last location of a unit and is executed at any time other than E6 or E7.

In addition to the above anomaly, the JMP μI has a second anomaly associated with it that allows a branch to the next 256-word page within a unit: (1) If the JMP μI occupies the last location of a page, or (2) If the JMP μI occupies the last location of a page and is executed at any time other than E6 or E7.

It should be noted that normal implemenation of the partial branch μI's do not make use of these anomalous characteristics. The anomalies "fall out" of the hardware design by default rather than by intent.

Generation of the branch address for the first class of branch μI's is illustrated in Figures 2-34, 2-35, and 2-36 using the FNJ μI as an example. These three figures show execution of the FNJ μI at E4, E6, and E7, respectively. As will be seen, the branch address and starting μI address formed will differ according to the time that the μI is executed. For all three examples, the FNJ μI is assumed to be located in the next to last address (FFE) of 4096-word CS unit 0. Referring to Figure 2-34, the address of the FNJ μI to be executed at E4 is clocked in Sμ at E280 and the μI read from CS during E3. At E380, Sμ is clocked with address 0FFF (0FFE+1). Bits 2 and 3 of this updated address, in turn, are clocked into Pp at E480 to form bits 2 and 3 of the starting μI address. These two bits are enabled to Pp by $\overline{E7}$, indicating the enable is active during every minor cycle except E7. Bits 2 and 3 also clocked into Sμ at E480, but after being updated a second time to form address 1NNN. This indicates that the resultant jump will be made to some address in module 1. What has been clocked into Pp, however, is address 0NNN meaning that for blockpoint purposes the resultant jump will be made to the same address as in Sμ but in unit 0. Bits 4 through 15 of the jump address are formed by a translation of bits from both the FNJ μI and the MLI in the F register as discussed in the paragraph titled Jump Decode. These twelve bits are clocked into both Pp and Sμ by the enable shown in Figure 2-34. Note that these enables clock the jump address in two parts: bits 4 through 7 and bits 8 through 15. The μI normally executed at E5 is inhibited by blocking it from going into Fμ. Instead, the μI at jump address 1NNN is read and then executed at E6.

*CS storage units are 4096-word portions of CS located on boundaries of 0_{10} (0000_{16}), 4096_{10} (1000_{16}), 8192_{10} (2000_{16}), and so forth.

Execution of the FNJ μI at E6 and E7, shown in Figures 2-35 and 2-36, differs from that executed at E4 in that the μI to which the jump is made is not executed until the next time slice. In Figure 2-35, the jump will be to address NNN in the same unit since the address clocked into Pp does not get updated a second time by the Sμ+1 logic. At E680, instead, the starting μI address for the following time slice (XXXX) is gated into Sμ. This μI is then executed at E0 of the next time slice. Figure 2-36, is similar to Figure 2-35 except that bits 2 and 3 cannot be routed into Sμ even after the first Sμ+1 update. Therefore, they are routed to Pb along with bits 4 through 15. At E780, they are clocked into Pp along with the rest of the jump address from the FNJ translation logic. Note that bits 2 and 3 are enabled from Pb specifically at E7, in contrast to the two preceding examples where the bits were enabled from Sμ by $\overline{E7}$.

In contrast to the preceding class of branch μI's, the JMP μI not only can jump to the next unit but also to the next 256-word page within a unit if located at address FFE and executed at any time other than E6 or E7. This condition arises from the fact that a JMP μI jump address is formed using bits 2 through 7 of Sμ instead of just bits 2 and 3. Therefore, the page address (bits 4 through 7) can be updated by the Sμ + 1 logic as well as the unit address. This page updating facility is shown in Figures 2-37 and 2-38, which illustrate execution of the JMP μ I at E4 and E7, respectively. Both examples assume the JMP μI is located in address 00FE, the second to the last address of CS page 0. Execution of the JMP μI at E4 is similar to that of the FNJ μI at E4, except that only bits 8 through 15 of Pp and Sμ are loaded with the translated jump address from the JMP μI. Bits 2 through 7 of Pp are loaded with the bits 2 through 7 of the JMP μI address updated once by the Sμ+1 logic (00FE+1=00FF). Since only one update of this address is not sufficient to advance to the next page, the address in Pp causes a jump to address NN in the same

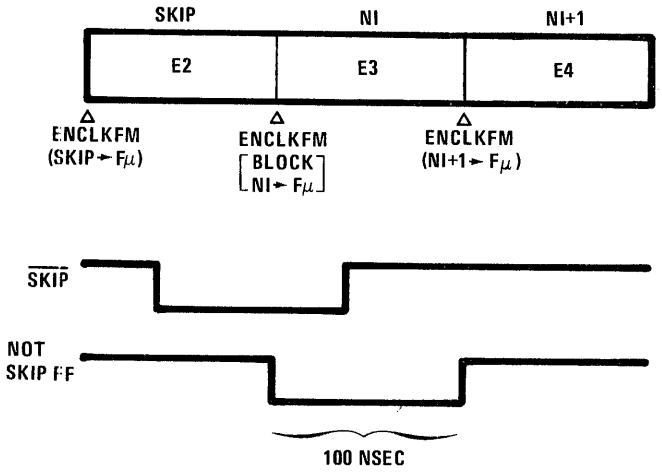
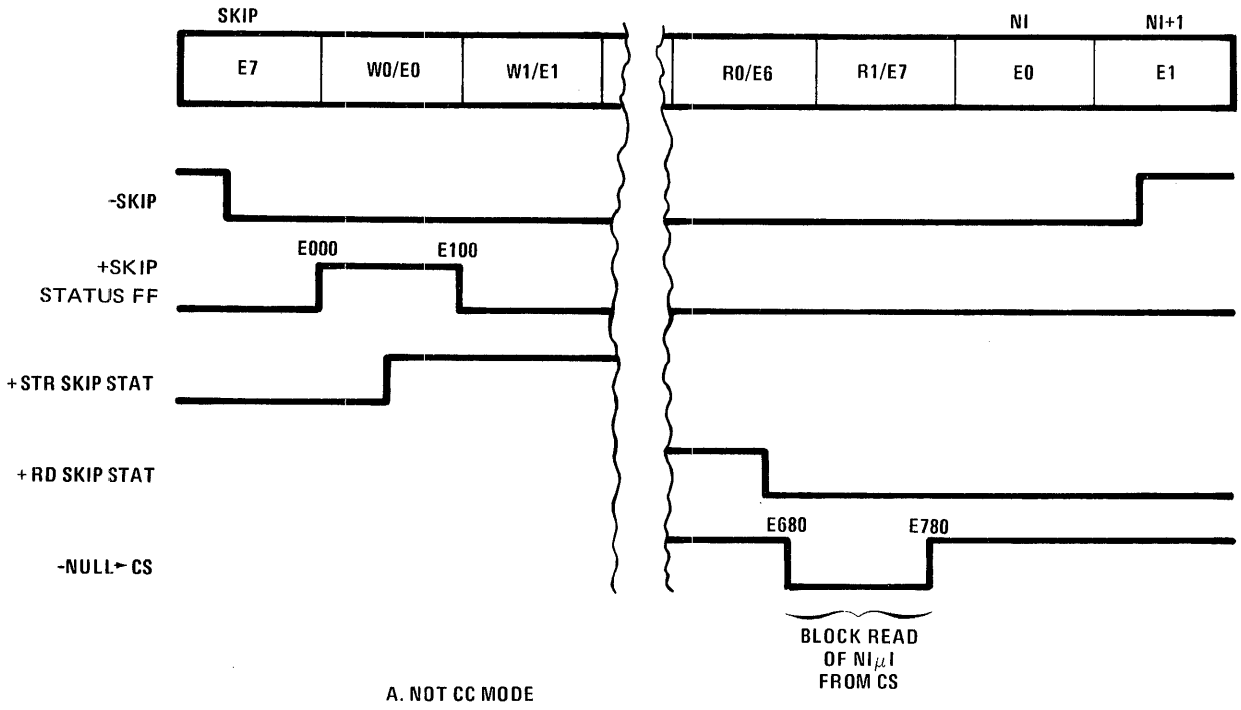


Figure 2-32. Timing for Skip Executed at E0 through E6



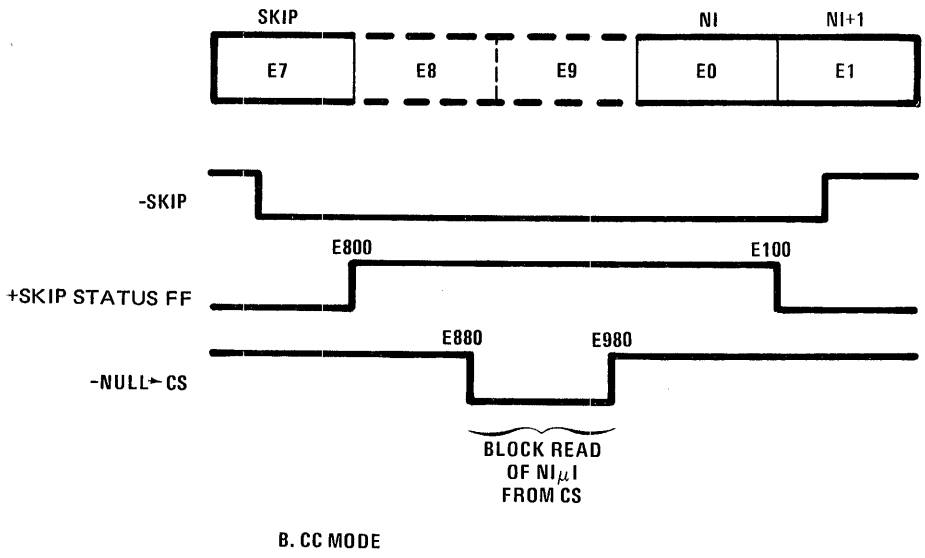


Figure 2-33. Timing for Skip Executed at E7

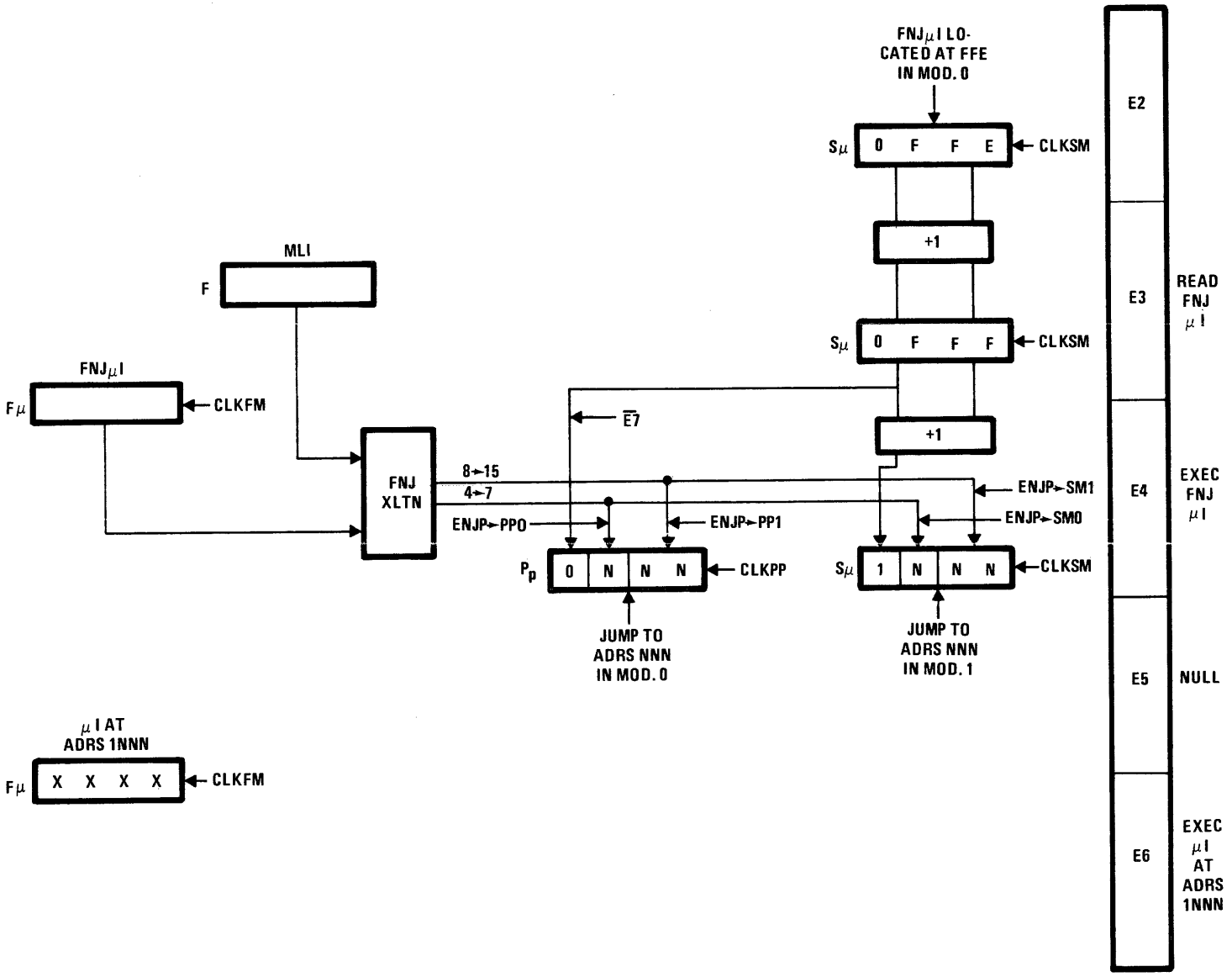


Figure 2-34. FNJ μI at Location FFE, Executed at E4

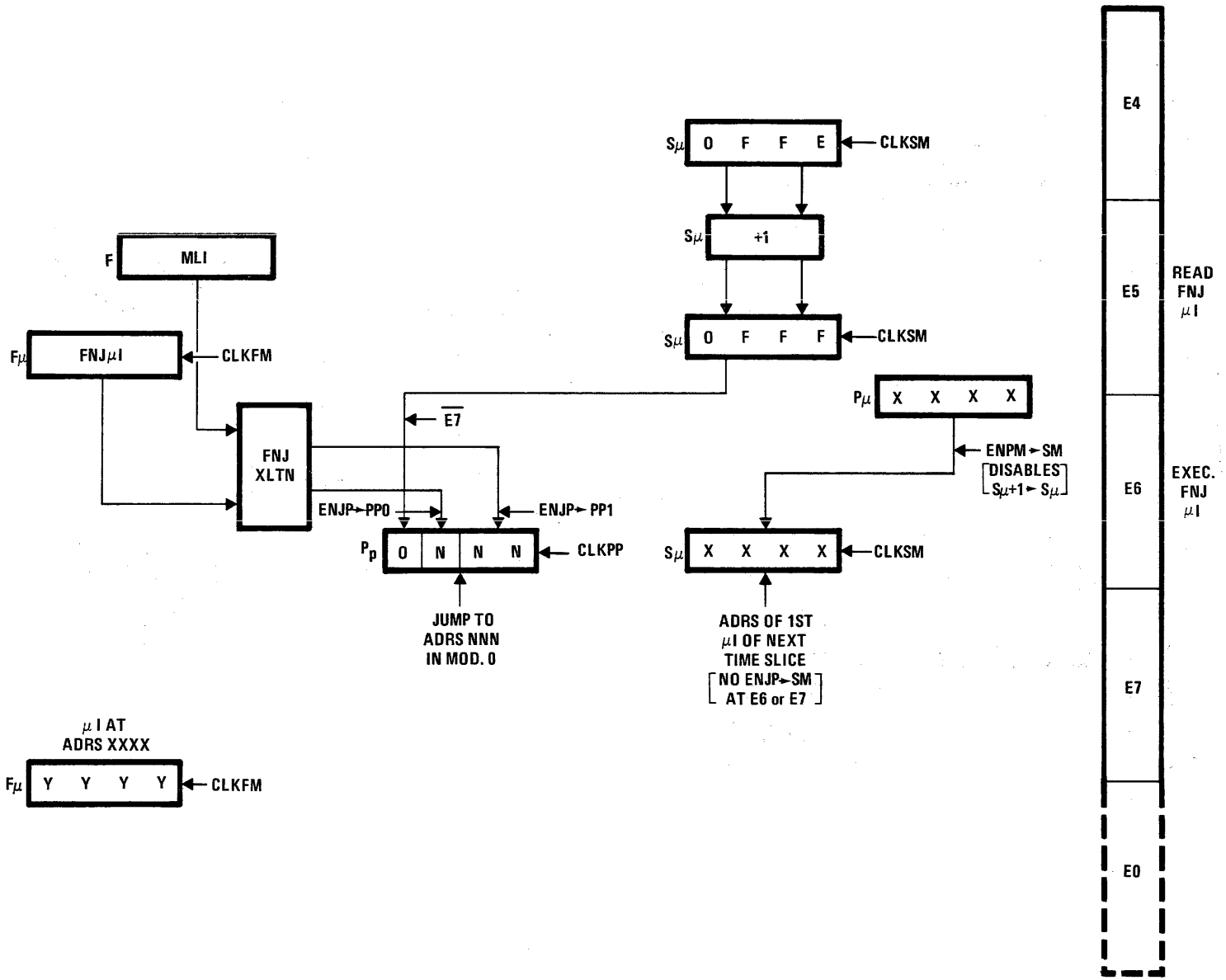


Figure 2-35. FNJ μI at Location FFE, Executed at E6

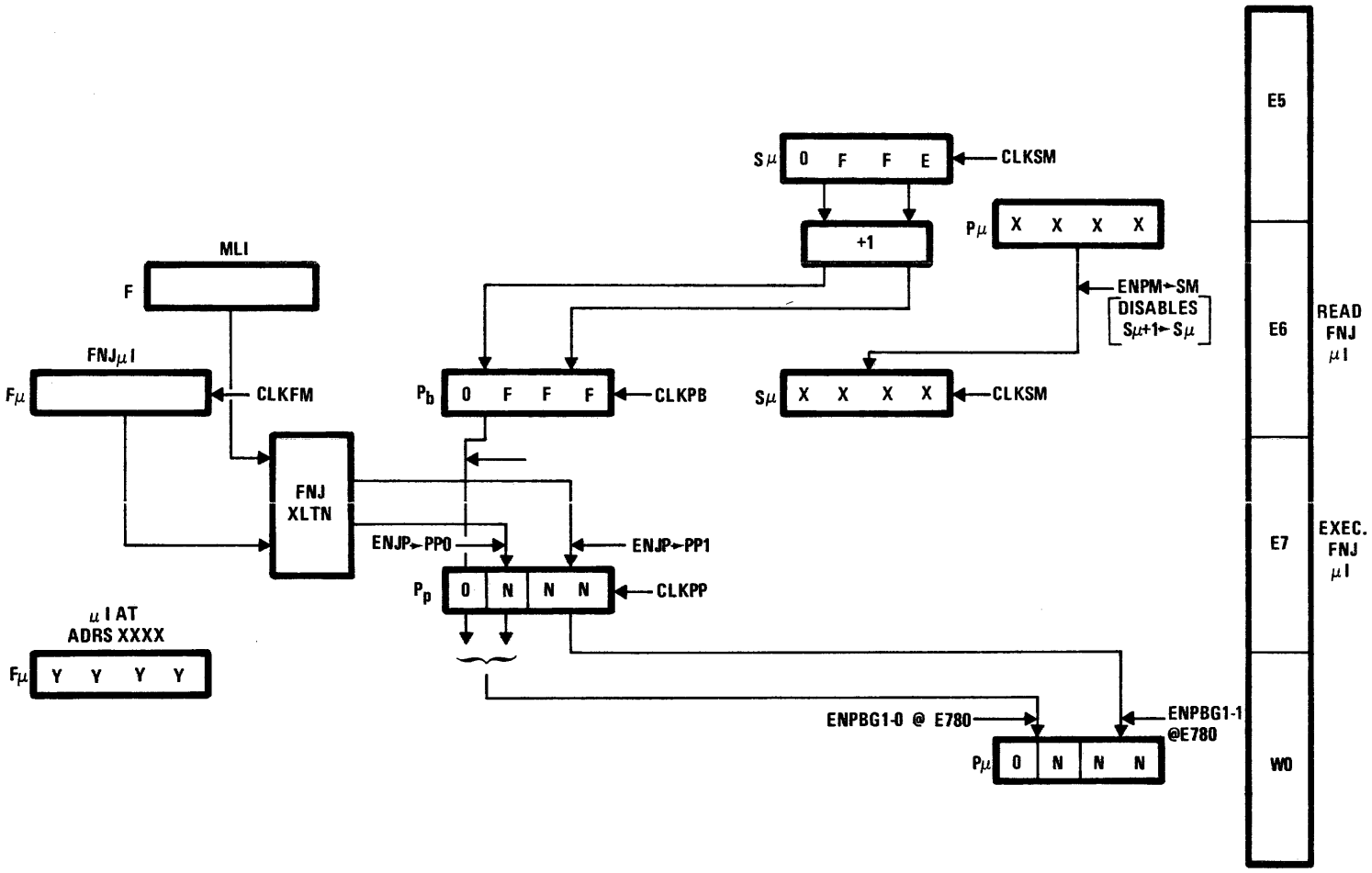


Figure 2-36. FNJ μI at Location FFE, Executed at E7

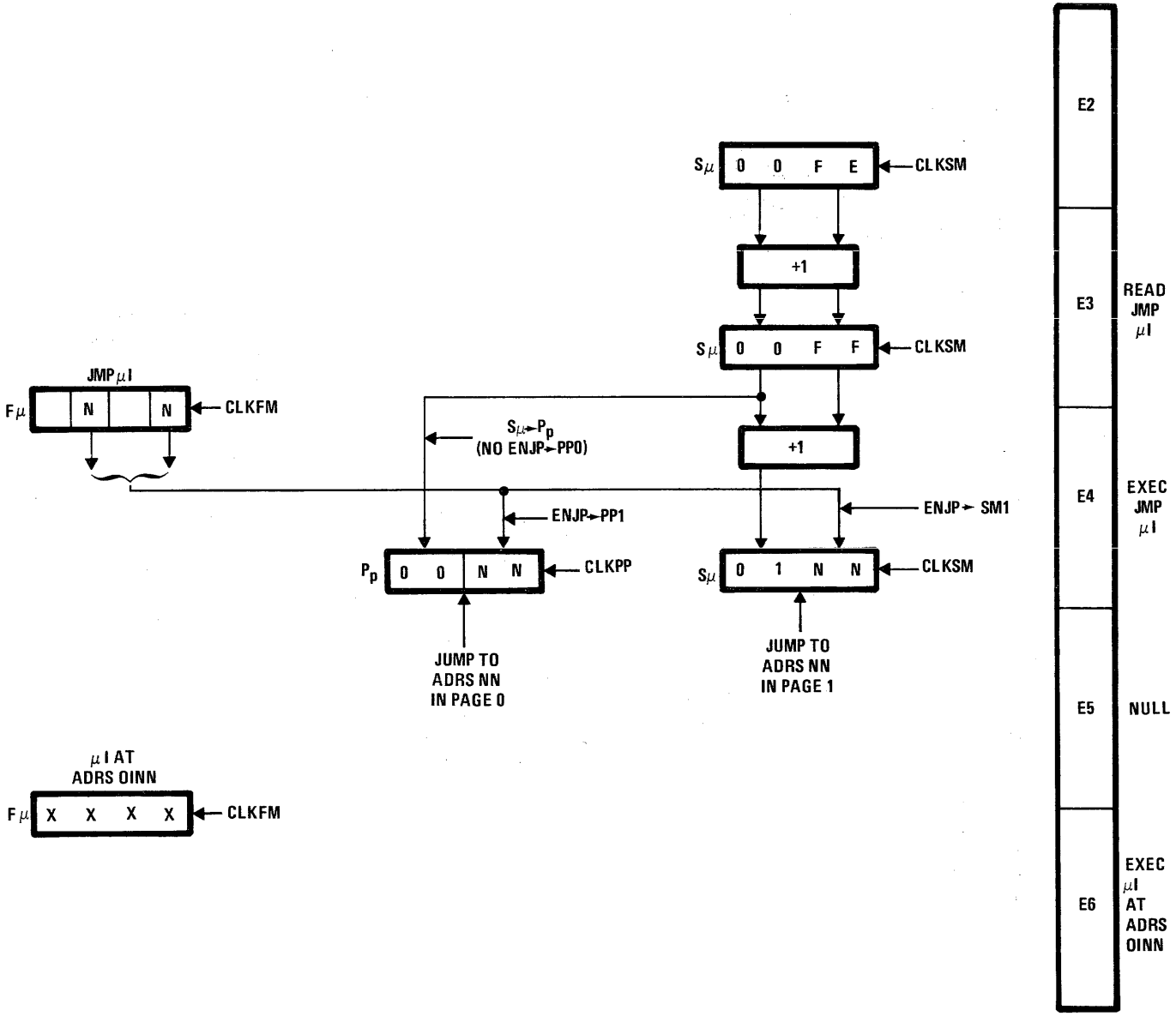


Figure 2-37. JMP μI at Location FE, Executed at E4

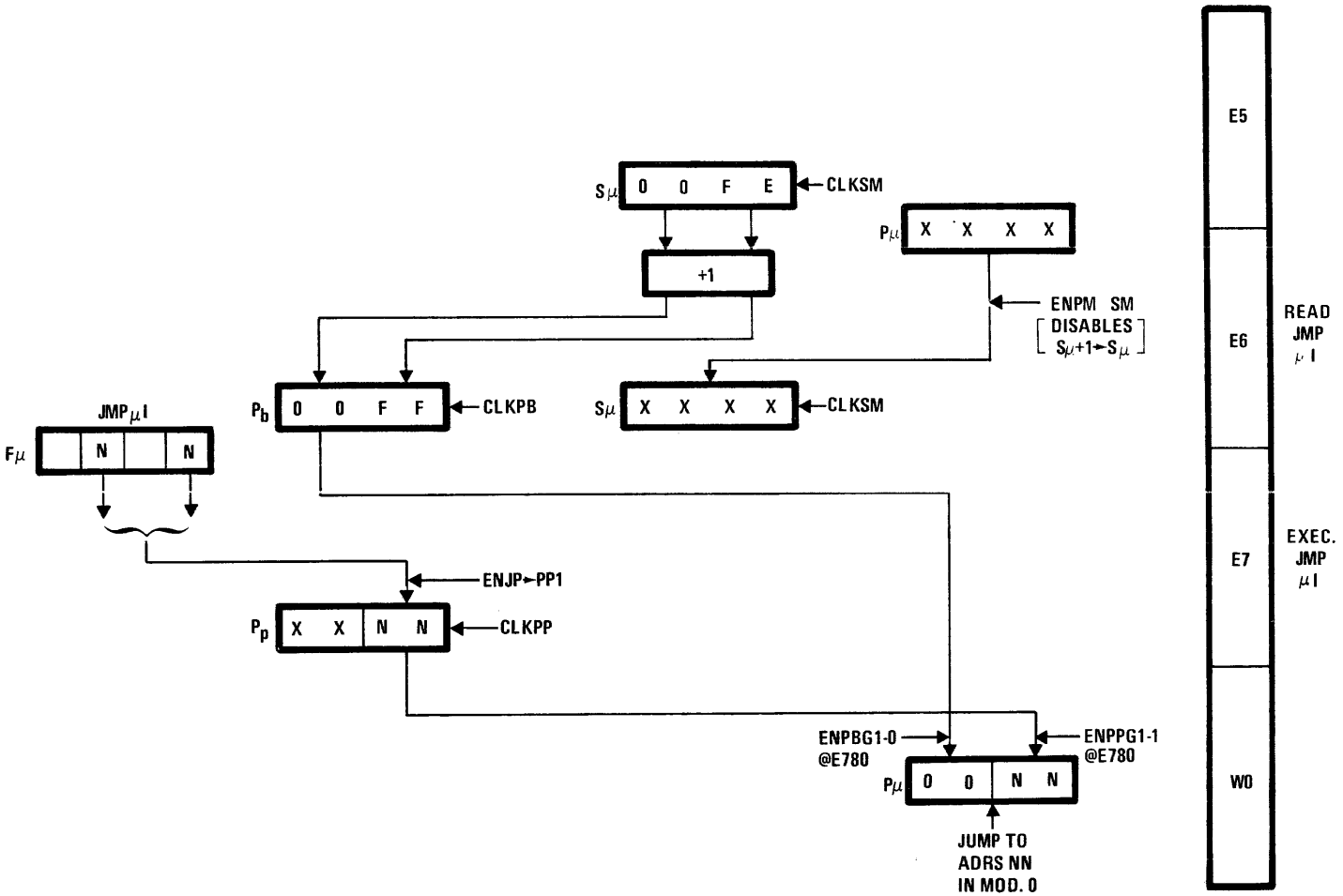


Figure 2-38. JMP μI at Location FE, Executed at E7

page (page 0). The resultant jump address in Sμ, however, represents bits 2 through 7 after having been updated twice by the Sμ+1 (00FE+2=0100). This causes a jump to address NN in the next page (page 1). The JMP μI executed at E7 is similar to the FNJ μI executed at E7 in that Pb must be used to hold the updated bits 2 through 7 from Sμ. Since these bits can be updated only once, the jump is confined to the same CS page.

Cycle Delay Logic

The *cycle delay* logic is shown in Figure 2-39. Essentially, the logic consists of the Cycle Delay flip-flop, which is set at t00 for 100 nanoseconds by any μI that feeds Aμ and/or Bμ. These μI's are the Load S (3,X), Load Bμ (6,X), EBU and EBL (A and B), D→A (C,X), Load Aμ (D,X), and the Sense (E,X,1) μI's. Setting this flip-flop generates 1ST CYCLE from the set side and 2ND CYCLE from the clear side. As shown in the timing of Figure 2-40, signal 1ST CYCLE remains high for 100 nanoseconds followed by 2ND CYCLE, which goes high after 100 nanoseconds. This figure shows two examples of a 2,X μI (a SUM μI) following a load Aμ μI (a LAW μI). Part *a* of the figure shows the SUM μI immediately following the LAW μI; part *b* shows the SUM μI separated from the LAW by a non-load Aμ/Bμ μI (a LDW μI). As shown in both examples, the Cycle Delay flip-flop is set at E100 and remains set while the operand loaded in Aμ propagates through the ALU. In part *a* enable $\overline{\text{RF-WR}}$ used to write the sum of Aμ and Bμ into register X (that is, the register selected by the μI X-field) is inhibited by the high 1ST CYCLE signal during E1. This signal goes low at E200 to allow the sum to be written into register X during E2. Since the SUM μI has overlapped into E2, it is necessary to delay all following μI's on the time slice for one minor cycle. This is done by routing 1ST CYCLE to a NOR gate, which blocks ENCLKSM for one minor cycle, and to an AND gate, which generates BLKFM for one minor cycle. These inhibit conditions prevent the NI μI from being clocked into Fμ and the address for the NI+1 μI from being clocked into Sμ for one minor cycle. Signal $\overline{\text{E67IDL}}$ inhibits the Cycle Delay flip-flop from blocking Sμ at either E6 or E7 to allow the starting μI address for the following time slice to be clocked into Sμ. Part *b* of the figure shows the Cycle Delay flip-flop being set again at E100 to block $\overline{\text{RF-WR}}$. This time, however, a LDW μI is being executed during E1. Since the LDW μI does not feed data into Aμ or Bμ, the operand in Aμ is able to propagate freely through the ALU. At E2, the SUM μI is executed to store the resultant sum of Aμ and Bμ at E250. For this case, then, the SUM takes only one minor cycle to execute.

System Reset Logic

The system can be *reset* to an initial condition (master cleared) in one of four ways: from a power-on condition, pressing the SYSTEM RESET pushbutton on the System Control Panel, initiating a Reset/Load operation by pressing the RESET LOAD pushbutton on the Panel, or initiating an Autoload operation by pressing the AUTOLOAD pushbutton on the Panel. When initiated in one of the ways described above, the system reset sequence performs the following operations:

1. The output of the ALU is cleared.
2. The eight Pμ registers within the Extended Register File, Group I, are cleared.
3. The Busy/Active, Tie-Breaker, Control, Privileged Mode, Boundary-Crossing, CS Scan, Panel Address, and Panel Data registers within the Extended Register File, Group II are cleared.
4. The Aμ, Bμ, D and Forced Carry registers within the ALU are cleared.
5. A *clear* signal is transmitted to the Extended Register File, Group III. (For the effects of this signal within the integrated adapters, see the appropriate I/O processor document.)
6. The Resource Allocation Network (RAN) is forced to issue Null cycles only.
7. The gray code counter is forced to issue ten minor cycles per major cycle.
8. The Sμ, Fμ-1, and Fμ-2 registers are cleared.
9. The RTC increment pulses are disabled at the set input of the Busy flip-flop for processor state 4, (Bit position 04 of the Busy/Active register).
10. The logical inter-lock which is set by a breakpoint stop operation with processor state 4 (and when set, disables the output of the Busy flip-flop for processor state 4 from appearing at the input of the RAN) is cleared.
11. In the presence of the Register Option (RO) Relocation and Protection feature, the Addressing Mode register is cleared.

The system reset sequence lasts 0.4 to 0.6 milliseconds if initiated from a Reset/Load or Autoload operation, as long as the pushbutton is held pressed if initiated from the SYSTEM RESET pushbutton, or until the POWER ON indicator lights if initiated by a power-on condition.

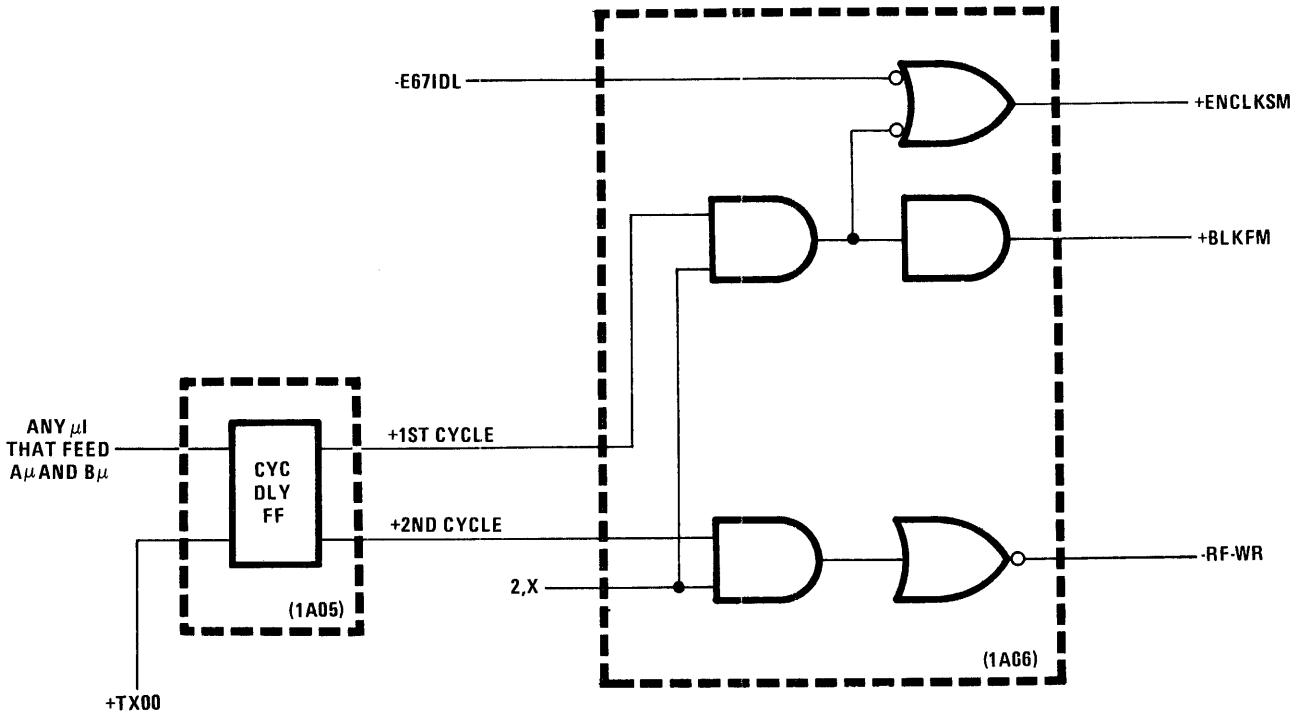


Figure 2-39. Cycle Delay Logic

A block diagram of the system reset logic is shown in Figure 2-41. Depending on how the system reset sequence is initiated, one of three signals will be generated: $\overline{\text{SW-AUTO}}$ if initiated by an Autoload operation, $\overline{\text{MC-LD}}$ if initiated by either a power-on condition or a Reset/Load operation, or $\overline{\text{SW-MC}}$ if initiated from the SYSTEM RESET pushbutton. The $\overline{\text{SW-AUTO}}$ and $\overline{\text{SW-MC}}$ signals are fed through flip-flops to eliminate switch bounce. The resultant three signals are then fed to a NOR gate, which feeds two one-shots to generate system reset signals MC-ALU, MC-IO, MC-1, MC-2, and MC-3.

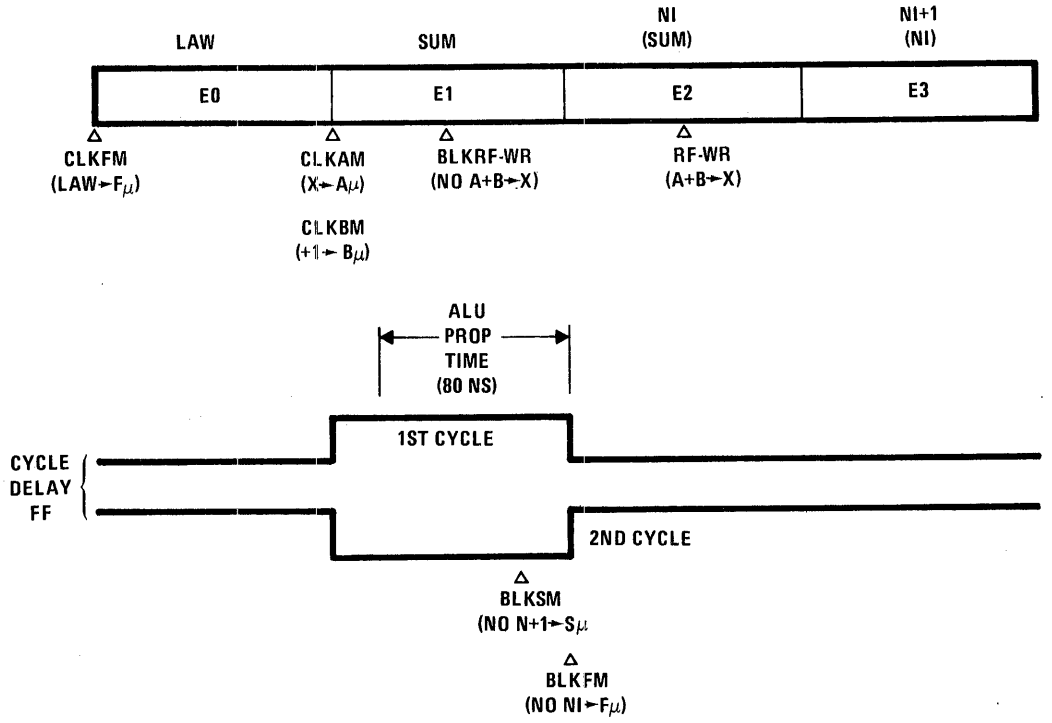
The MC-ALU signal is fed to the ALU enable logic to generate $\overline{\text{SEL-ZR-0}}$ and $\overline{\text{SEL-ZR-1}}$. These select signals, in turn, are fed to the ALU fan-in to effectively gate an output of all "0's" on the 16 lines from the ALU. These "0's" are then routed to the Group II registers of the ERF and to the Sμ register, where they are clocked into the registers to clear them. The clock and clock enable signals for the ERF Group II registers are generated by MC-1, and the clock enable signal for the S register by MC-2. The Aμ, Bμ, Fμ, D, and Fμ-1 and Fμ-2 registers are also cleared by means of MC-1. These registers, however, differ from the ERF Group II and Sμ register in that they can be cleared directly by a forced clear input to each register stage flip-flop. Clearing these registers is accomplished by clear signals CLRFM, ENRDR, ENRAM, and ENRBM. The Force Carry register is cleared in a similar manner.

The gray code counter is forced to issue a count of ten minor cycles (E0 to E9) by simulating a Consecutive Cycle (CC) condition. This is done by combining MC-2 with STATEN from the Null flip-flop (which will be set during a system reset condition) to clear the Consecutive Cycle flip-flop. The high output from the clear side (CC-F/F) is sent to the counter, which interprets the signal as a request for CC operation. The result is to enable the E8 and E9 stages of the counter to generate the ten minor cycles.

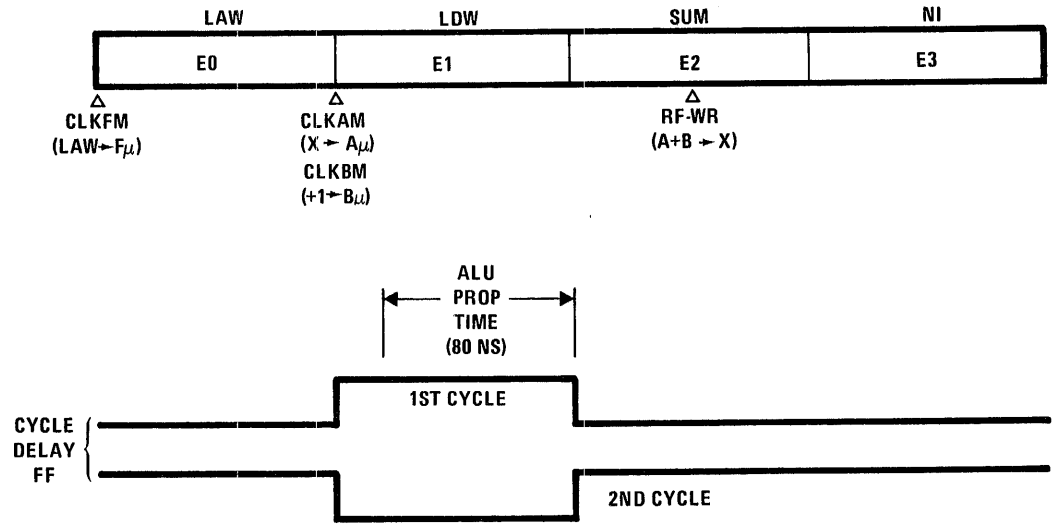
The Pμ registers associated with all eight processor states are cleared by MC-3, which travels through three stages of inversion to generate SELFH/PL and EF1RF/WL. Both of these signals are low; therefore, Pμ is selected to be written. Since there is no data on the lines which fed Pμ, the registers are filled with "0's". Each of the eight registers is selected in sequence by MC-2, which is combined with E timing pulses from the gray code counter to generate the three Pμ select signals (ERFG1) in a cyclic manner.

The RAN is forced to issue null cyclic by means of $\overline{\text{MC-4}}$, which sets the Null State flip-flop in the RAN. Setting this flip-flop, in turn, sets the Null flip-flop which sets up the null conditions (block clocking of Sμ and inhibit accesses to CS).

If the Relocation and Protection feature of the RO is present, the Addressing Mode register is cleared by MC-3. This signal generates register write enables $\overline{\text{ADDWR-0}}$ and $\overline{\text{ADDWR-1}}$ in combination with timing pulse E5 to write "0's" into the register.



A. SUMμI FOLLOWED BY LAWμI



B. SUMμI FOLLOWED BY LDWμI

Figure 2-40. Cycle Delay Timing

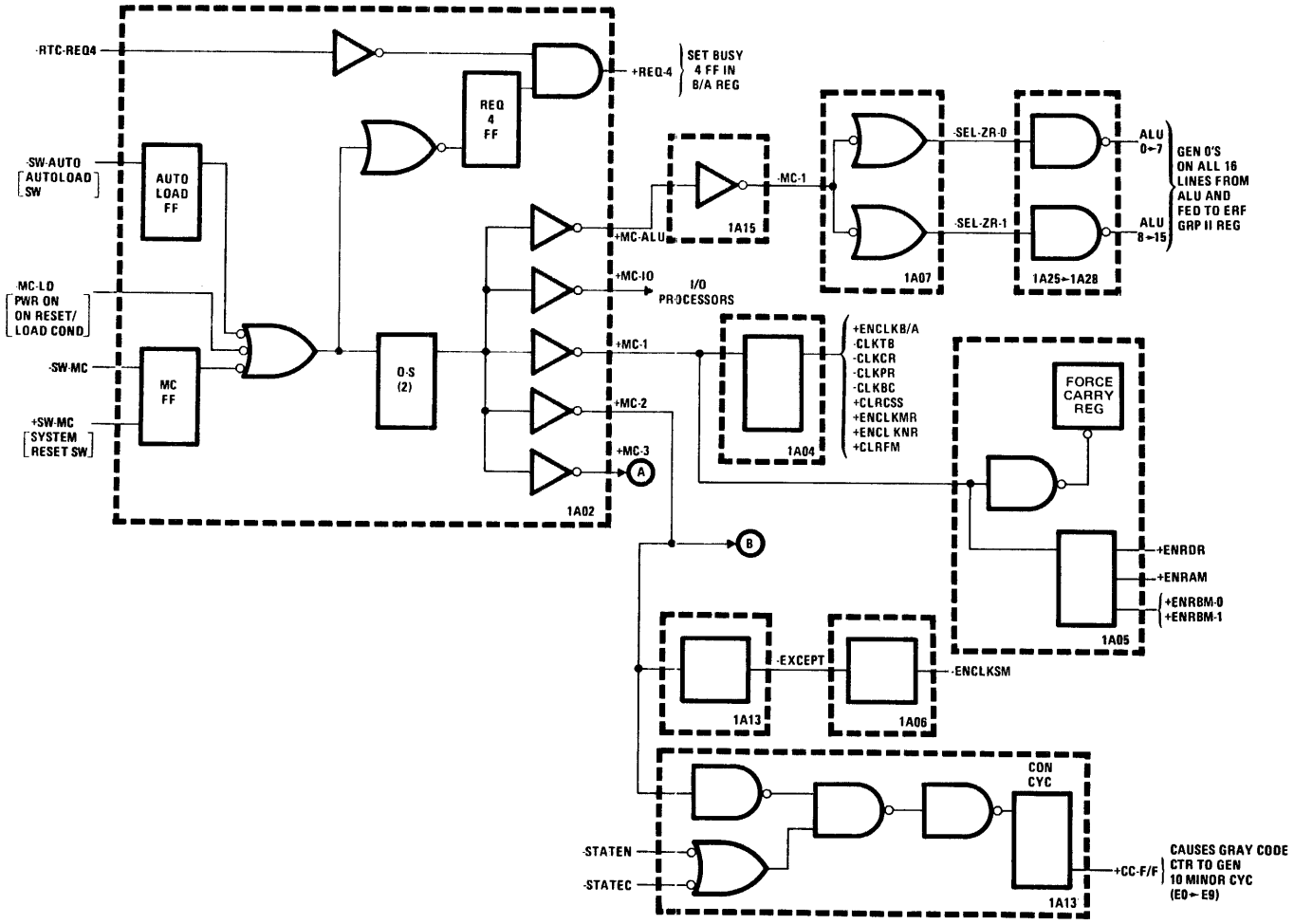


Figure 2-41. Systems Reset Logic

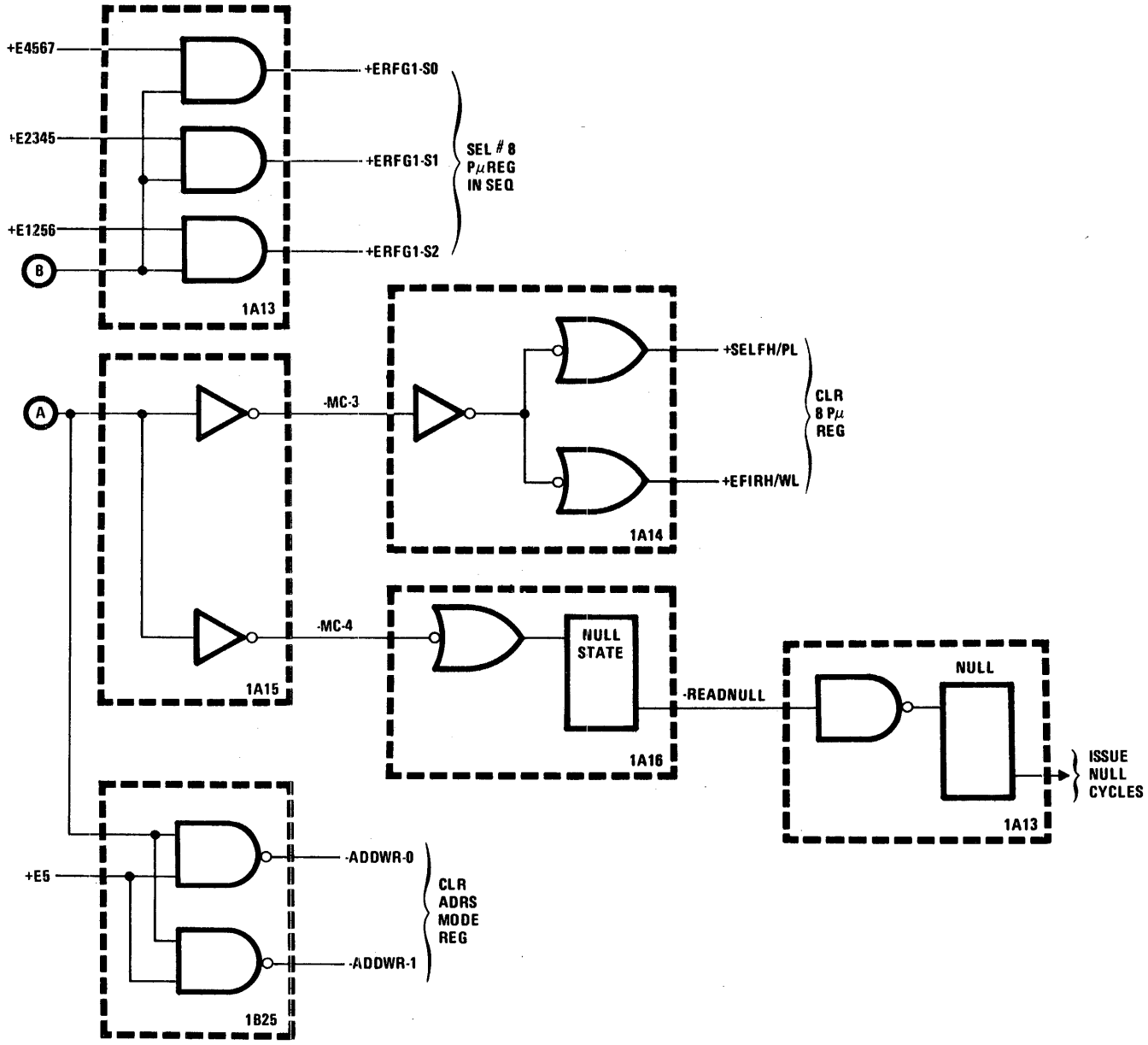


Figure 2-41. Systems Reset Logic (Cont)

The RTC increment pulses used to set the Busy flip-flop associated with processor state 4 in the B/A register are prevented from doing so during a system reset condition by clearing the Request 4 flip-flop. This flip-flop is set during an autoload sequence to give control of the autoload routine to processor state 4 by setting the Busy 4 flip-flop on the B/A register via REQ-4. During a system reset, however, the flip-flop is cleared to prevent the Busy 4 flip-flop in the B/A register from being set by REQ-4 until completion of the system reset sequence.

Idle and Resync Conditions

Several of the μI's require either *idling* through the minor cycle following their execution, or through the rest of the time slice so that the next μI is executed at E0. Micro instructions which fall into the former category are the FNJ; JMP; and CLR, STA, STB and AND if X specifies Pμ. These μI's cause either a partial branch (FNJ and JMP I's) or a full branch (CLR, STA, STB, and AND if X specifies Pμ) to a new μI address. The μI located in CS at this new address is read during the minor cycle following the one in which the branch μI was executed. Since this following minor cycle would normally have been used to execute the μI following the branch μI if the branch had not taken place, clocking this following μI into Fμ must be inhibited since the branch did take place. This inhibit operation is provided by generating BLOCKFM for one minor cycle.

Logic for generating BLOCKFM is shown in Figure 2-42. Operation code translation signals for the partial and full jump μI's are fed to gates 1, 2, and 3. During the execute minor cycle of these μI's, enable signal IDLE-F/F is high. The result is to generate BLOCKFM for one minor cycle, which inhibits ENCLKFM. Timing for signal BLOCKFM, as well as other signals associated with the idle operation, is shown in part *a* of Figure 2-43. (This figure assumes execution of the branch μI at E0; however, the relative times shown are the same if the μI is executed at any minor cycle E0 through E6.) Simultaneous with generating BLOCKFM, signal IDLE is also generated by the same translation signals via gates 4, 5, and 6. Signal IDLE sets the Idle flip-flop by means of gate 7 at E100 (first TX00 after IDLE if not E7), causing IDLE-F/F to go low. As a consequence, gates 1, 2, and 3 are disabled which drops BLOCKFM and, in turn, causes ENCLKFM to go high again. Dropping BLOCKFM after one minor cycle is necessary so that the μI read from the branch address, and all subsequent μI's, can be clocked in Fμ. Signal IDLE remains high through E1, however, since no new μI was loaded into Fμ at E100 due to Fμ being blocked. If an FNJ or JMP μI is being executed, Pp must be inhibited from being updated by the branch address +1 since the starting μI address formed by an FNJ or JMP μI is the branch address itself. This is accomplished by inhibiting ENCLKPP at E1 via gate 8 for an FNJ μI and via gate 9 for a JMP μI. This action retains the branch address clocked into Pp at E0 as the starting μI address. At E200, the Idle flip-flop is cleared due to the low on the flip-flop clear output fed back to gate 10. The result is to cause both IDLE-F/F and ENCLKPP to go high once again.

As can be seen from Figure 2-43, branch μI's executed at E0 through E6 take 200 nanoseconds to execute: 100 nanoseconds to form the branch address and 100 nanoseconds to read the μI from CS at the location specified by the branch address. If the branch μI is executed at E7, however, the total execution time is only 100 nanoseconds, since the μI specified by the branch address will not be read out until R0 of the next time slice assigned to the processor. In this respect, then, the branch μI acts like an ordinary blockpoint μI and blocking of Fμ is not required. In fact, Fμ must be clocked at E0 to enable the first μI of the next time slice to be executed. This is accomplished by nullifying the effect of BLOCKFM by E0/8XX-E, which forces ENCLKFM high at E750. During E0 of the next time slice, BLOCKFM goes low when the first μI of the next time slice is loaded into Fμ.

In contrast to the branch μI's, which require idling through just one minor cycle, the FRJ, RNI1, RNI2, CIO1, CIO2, ROM, SYNC, and FZJ (if Aμ is 0) μI's result in an idle through the remainder of the time slice so that the next μI is not executed until the next E0. These μI's are called resync μI's, because they resynchronize μI execution back to E0. These μI's achieve resynchronization by blocking Fμ for the remainder of the time slice via gates 10, 11, and 12 of Figure 2-42. These gates do not have to be enabled by IDLE-F/F as do those for the partial branch μI's since BLOCKFM will remain high through E0 of the next time slice. As for the branch μI's, however, the effect of BLOCKFM is negated at E0 by the action of E0/8XX-E to force ENCLKFM at E650. Timing of BLOCKFM for a resync μI is shown in part **b** of Figure 2-43. One minor cycle later, the Idle flip-flop is set via gates 13, 14, and 15 for the purpose of inhibiting ENCLKPP. For resync, clocking of Pp must be inhibited after the minor cycle in which the resync μI is executed to avoid continuously updating Pp by every update of Sμ throughout the remainder of the time slice. At the end of the time slice the Idle flip-flop is cleared and ENCLKPP is allowed to go high updating Pp as required during the next time slice.

CONTROL STORAGE

The Control Storage (CS) section is an alterable 14-bit, word-oriented solid-state memory capable of storing 5120

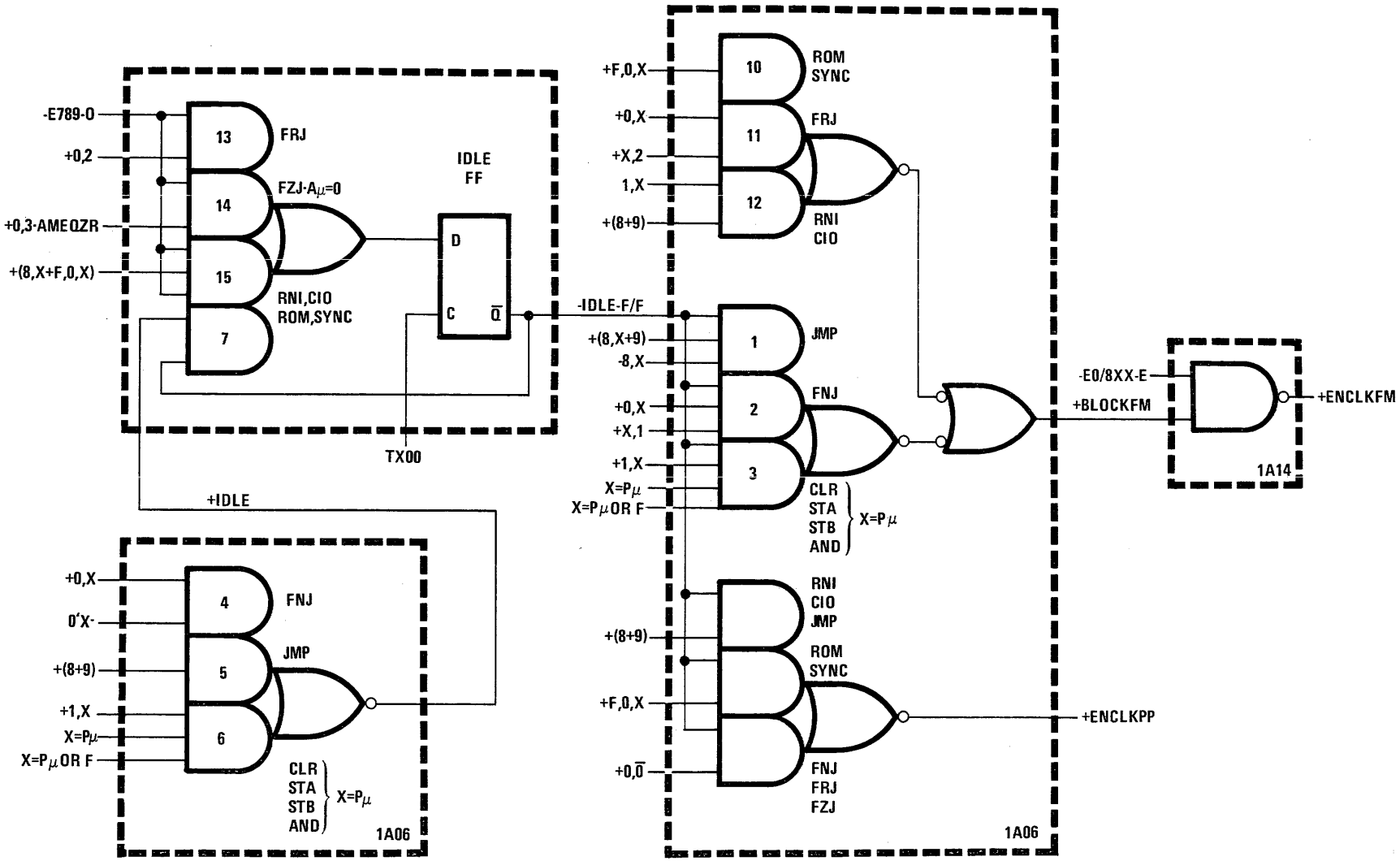
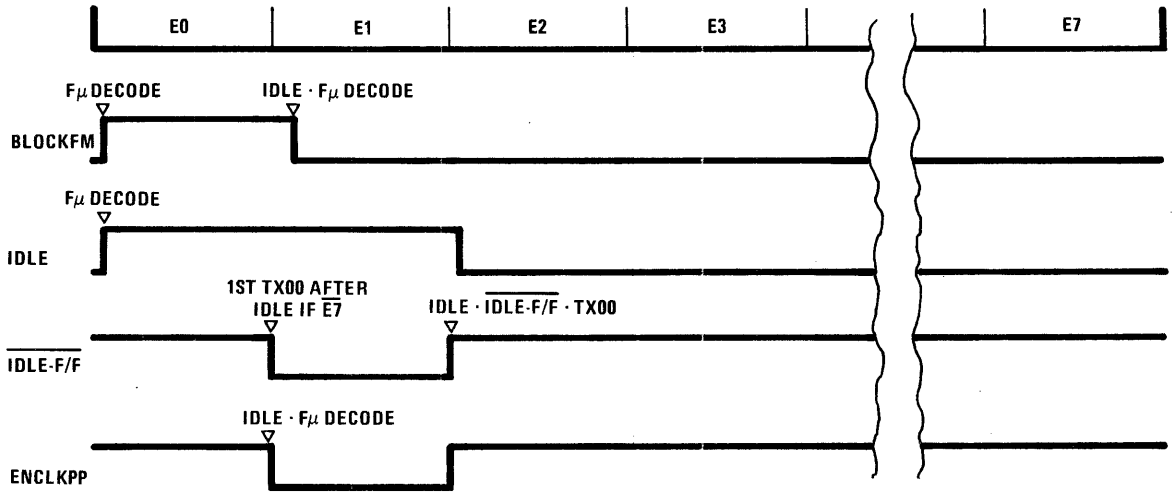


Figure 2-42. Idle and Resync Logic



A. BRANCH μI'S

B. RESYNC μI'S

Figure 2-43. Idle and Resync Timing

words (basic size). The CS stores all μI's used by the processing unit. These μI's execute MLI's under program control and perform functions initiated by the System Control Panel and peripheral devices such as Reset/Load and Auto Load. The CS also stores verification and diagnostic routines used during checkout and maintenance operations. A block diagram of the CS is shown on Figure 2-44.

CS Operation

The CS section is a rapid-access semi-conductor memory that stores 14-bit words in bipolar random-access memory (RAM) integrated circuits (IC's). It is organized on the basis of 4096-word storage units and is expandable in increments of 1024 words to a maximum of 16,384 words (addressing limit). At present, the system is provided with a basic CS of 5120 (5K) words (one 4096-word storage unit plus a 1024-word portion of a second unit) with an 8192-word (8K) CS offered as an optional feature. Each storage unit consists of 224 RAM IC's and corresponding address select logic. Each IC stores 256 bits, and is interconnected with other IC's so that each stores one bit of 256 words. Each word, therefore, is partially stored on 14 IC's, one bit per IC. This partial storage of words, 14 bits *wide* is referred to as *page storage.* A page is a block of 256 words. Since each storage unit consists of 4096 words, each unit consists of 16 pages (4096/256).

A physical representation of the paging concept is illustrated in Figure 2-45, which illustrates the pages making up two storage units. Note that the 14 bits of a word are numbered 0 through 8 and 11 through 15, with bit positions 9 and 10 not used. The 16 pages can be thought of as 16 loaves of bread, each loaf consisting of 14 slices. Each slice represents one RAM chip. Address ranges per page run in ascending order by page number, as shown in the figure. (For example, page 0 stores 256 words at addresses 0000_{16} through $00FF_{16}$ page 1 stores 256 words at addresses 0100_{16} through $01FF_{16}$, and so forth.)

Each CS module stores two bits of each word. The modules for the 8K CS contain 64 IC's each for storing two bits of 8192 words. The 64 IC's are arranged in two groups of 32 IC's each, wherein each group stores one bit of two storage units (32 x 256 = 8192). The modules for the 5K CS contain 40 IC's each for storing two bits of 5120 word. Each group of 20 IC's stores one bit of one storage unit (16 x 256 = 4096) plus one bit of a 1024-word portion of a second storage unit (4 x 256 = 1024). The total bits stored by each group then is 4096 + 1024 or 5120 bits. In essence, each 5K CS module for a 5K CS is an 8K CS module "depopulated" by the number of IC's required to reduce the number of words stored.

Words are addressed in CS by the upper 14 bits of the Sμ register, as shown in Figure 2-46. (The register is actually 16 bits in length; however, bits 0 and 1 are μI status bits and do not pass through the CS address logic.) As the figure shows, bits 2 and 3 select one of the four storage units, bits 4 through 7 select one of the 16 pages comprising each unit, and bits 8 through 15 select one of the 256 words in each page. Selection of a storage unit and addressing one of the 256 words in a page is accomplished via corresponding Sμ register bits directly. However, selection of a particular page is performed by an intermediate coding of bits 4 through 7 to generate page select (SELP) signals. These SELP signals are divided into three groups: SELPX-0, SELPX-1, and SELPX-2. The SELPX-0 signals select bits 0 through 3, the SELPX-1 signals select bits 4 through 8 and bit 11 of the data word, and the SELPX-2 signals select bits 12 through 15. The X value designates one of 16 page numbers (0_{16} through F_{16}). Each SELP signal is generated by a combination of Sμ register bits 5, 6, and 7, and either an ENRD-CS or ENWR-CS enable signal derived from Sμ register bit 4 in conjunction with other signals that define whether a read (ENRD-CS) or write (ENWR-CS) operation is to be performed. Logic for generating these enable signals is shown in Figure 2-47. Signal ENRD-CS0 will be generated whenever SM-CS04 is low, except when any of the following inhibiting conditions is present:

1. A parity error has been detected in the μI read from CS. This causes $\overline{\text{SWCS-OFF}}$ to go low.

2. The next μI is to be skipped, the processor state is operating in the Consecutive Cycle mode, or the CS has been disabled by the CS DISABLE switch on the System Control Panel.

For the above three conditions, NULL-CS is forced low. In the case of a skipped μI, it is still necessary for the processor state to idle through one minor cycle. This is accomplished by a NOP condition, wherein the CS is inhibited from transferring a μI to the Fμ register. The effect is to write all "0's" into Fμ. The Consecutive Cycle mode also requires a NOP condition during E8 and E9 time. (These times would normally be E0 and E1 for the next time slice, when Fμ would be loaded with the first and second μI's of the next assigned processor state. Since the same processor state will be granted the following time slices, these loads must be aborted.) Signal ENRD-CS1 is generated in a similar manner to ENRD-CS0 except that ENRD-CS1 is enabled when SM-CS04 is high. This enables ENRD-CS0 to select pages 0_{16} through 7_{16}

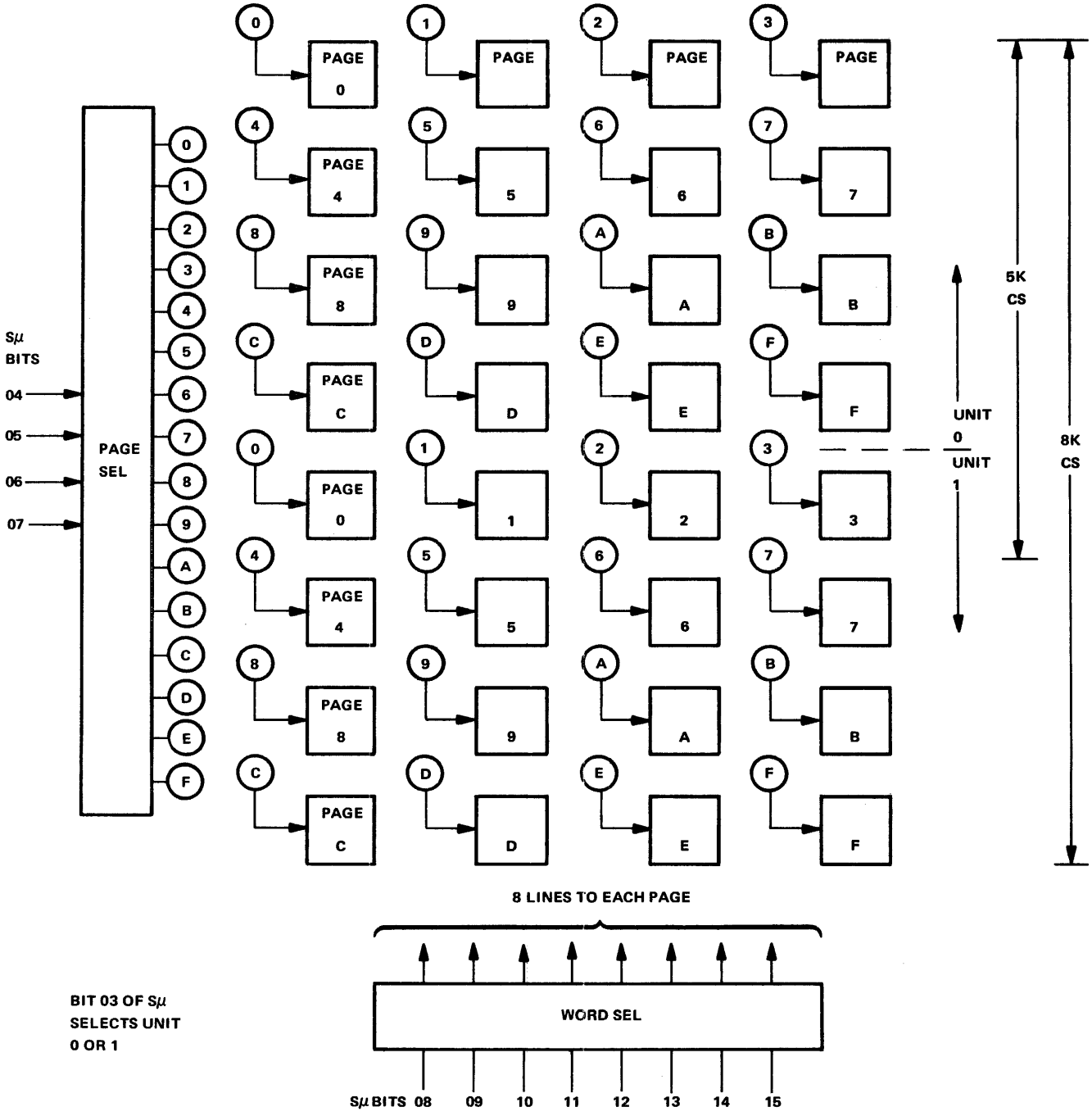


Figure 2-44. Control Storage Block Diagram

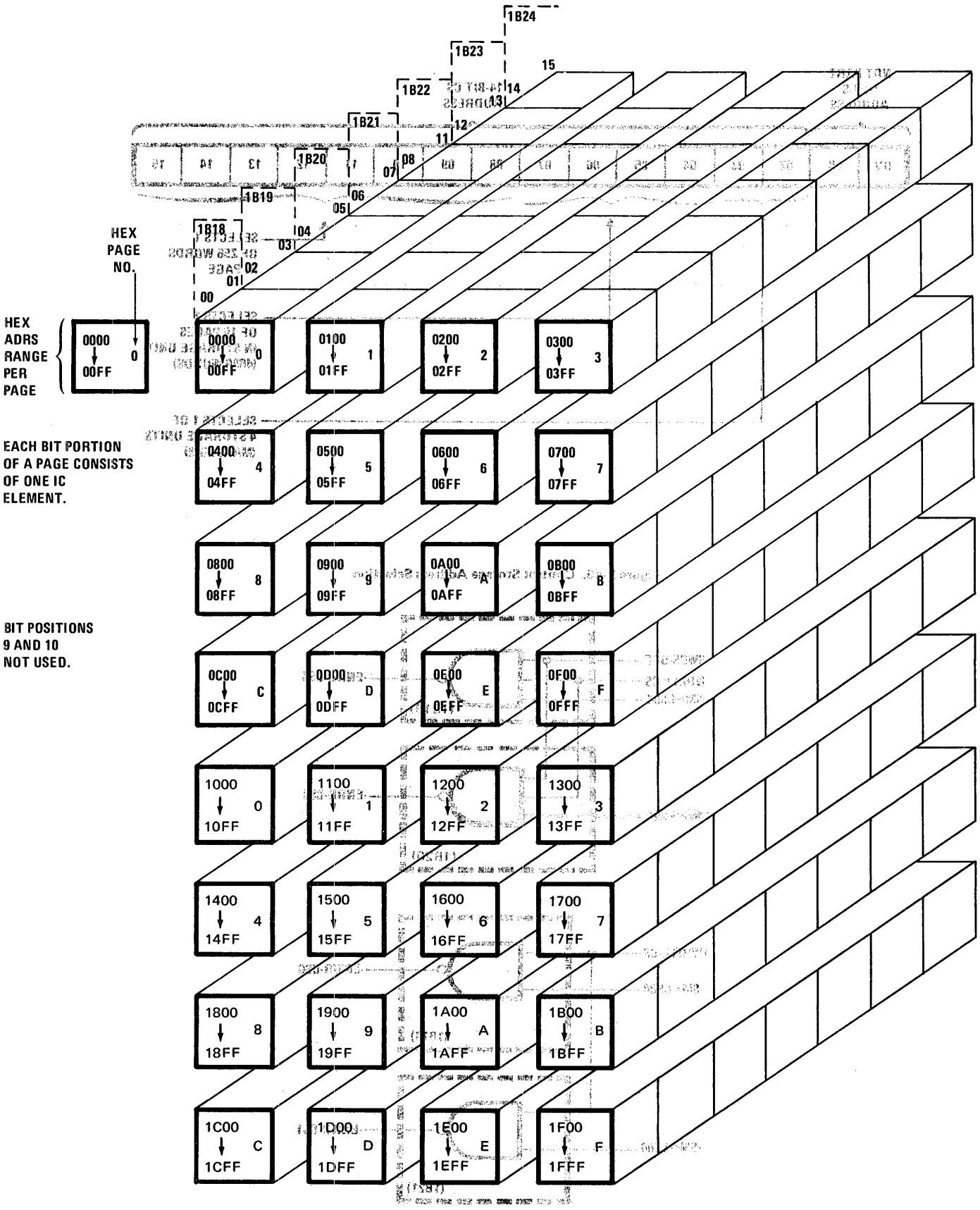


Figure 2-45. 8K Control Storage Page Organization

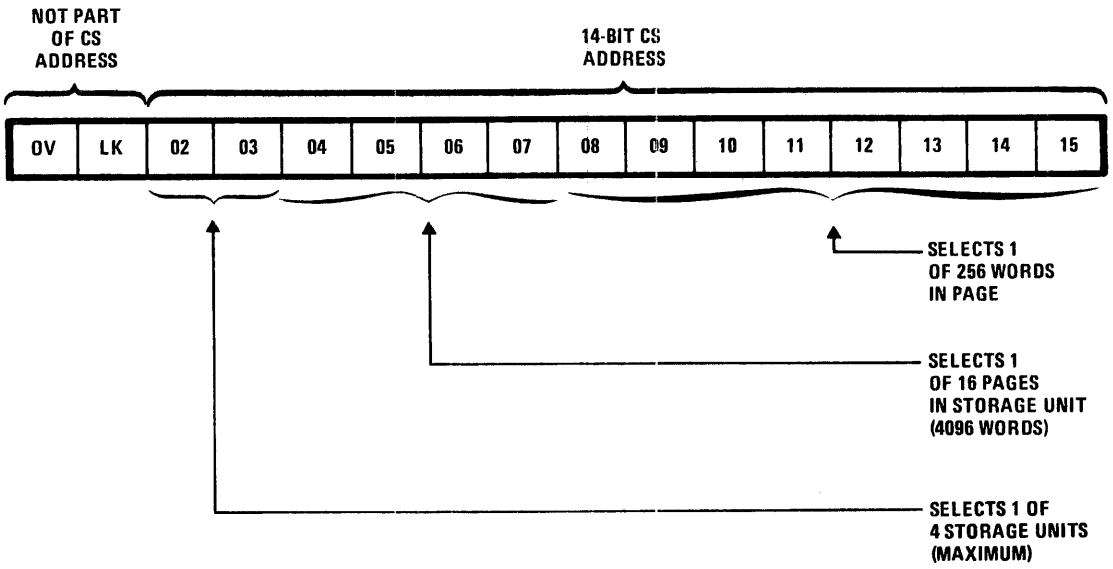


Figure 2-46. Control Storage Address Selection

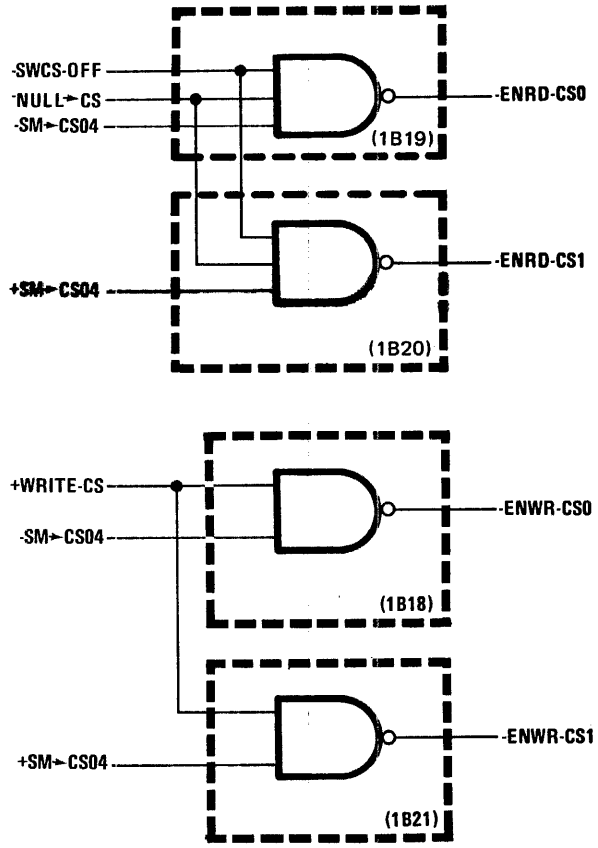


Figure 2-47. Generation of ENRD-CS and ENWR-CS Signals

and ENRD-CS1 to select pages 8_{16} through F_{15}. When either of these enables is present, data is read from the location addressed by bits 8 through 15 of the S register.

Signal ENWR-CS0 is generated when SM←CS04 is low and WRITE-CS is present. Signal WRITE-CS is generated by the System Control Panel during a CS load operation. Signal $\overline{\text{ENWR-CS1}}$ is generated in like manner, except that SM←CS04 is high. As with the two $\overline{\text{ENRD-CS}}$ signals, $\overline{\text{ENWR-CS0}}$ selects pages 0_{16} through 7_{16}, and $\overline{\text{ENWR-CS1}}$ selects pages 8_{16} through F_{16}. Either of these two signals is used to generate a second write enable signal, WRITECS, through a NOR gate. When either $\overline{\text{ENWR-CS0}}$ or $\overline{\text{ENWR-CS1}}$ is present, along with WRITECS, the data present on the N←CS input lines is stored in the location specified by bits 08 through 15 of the Sμ register.

Correlation of the SELP signals with the data bit groups they select, and the Sμ register bits and $\overline{\text{ENRD-CS}}$ and $\overline{\text{ENWR-CS}}$ signals which generate each SELP signal, is shown in Table 2-3.

MICRO-INSTRUCTION TRANSLATION AND ADDRESS UPDATE

The μI translation and address update logic reads a μI from CS at the location specified by the contents of Sμ and decodes it to generate the enables required to execute the μI. Upon making the CS read access, the contents of Sμ are updated to form the address of the next μI in the program. A block diagram of the μI translation and address update logic is shown in Figure 2-48. Depending on the μI routine, the next μI address will be:

1. Incremented by one by the Sμ+1 network
2. A jump address generated by the jump decode logic
3. A beginning address set from the System Control Panel, or
4. A jump address derived from the Aμ and Bμ registers through the ALU fan-out logic,

Within a particular time slice, the CS reads μI's as addressed by the contents of the Sμ register. If ending a time slice for a particular processor, the address of the beginning μI to be executed during the next time slice available for the active processor must be stored in a designated location in the Extended Register File (ERF). At these times, the updated μI address is routed to either the Pp register or Pb register for storage in the ERF via the ERF write logic.

Micro-Instruction Decoding

Fμ Register

The Fμ register holds the μI read from CS in preparation for translation by the first-level and second-level μI translation networks, and other decoding circuits. The register consists of two ranks, each rank 16 bits in length. Use of the double rank is necessary due to the high fan-out requirements of most of the μI bits. A typical stage of the Fμ register is shown in Figure 2-49. This figure shows the two ranks associated with the bit 00 stage, together with their interconnections. Each rank is double-gated to assume definite set and clear conditions.

Depending on the state of data bit CSDATA00, the flip-flop is set or cleared upon activating clock pulse CLKFM when enabled by ENCLKFM. Enable ENCLKFM is generated constantly, except for certain conditions when clocking Fμ must be inhibited. During certain idle conditions, the enable is inhibited by BLOCKFM. This signal is generated when executing either a ROM, SYNC, FRJ, RNI, μI; or a Load S μI executed at any time other than E0. These μI's result in idling through the rest of the major cycle so that the next μI in the sequence starts at the beginning of the next time slice. Clocking of Fμ, therefore, is inhibited for the remainder of the present time slice. Indication of a parity error in Fμ (PE-FM) or a long MS access (LONGACC) also inhibits ENCLKFM. Signal LONGACC indicates the addition of timing pulses E0', or E0' and E0'', required for the extra propagation time needed by the Register Option (RO) during MS references. Adding these pulses essentially sets up a *hold* condition during which Fμ must be inhibited from *clocking-in* the next μI. Execution of a FZJ (0,3) μI when the jump condition (Aμ is 0) is met returns control to the RNI2 subroutine. This situation causes an idle condition through the rest of the major cycle. Meeting the conditions for a skip when executing a skip (5,X,X) μI inhibits ENCLKFM for one minor cycle. This causes the following μI would normally be executed is not aborted, however .

The enable signal is specifically generated during E8 and E9 during consecutive cycle operation to perform NOP's during these minor cycles (transfer "0's" to Fμ). This prevents Fμ from being loaded with spurious μI's during these minor cycles.

Routing of bits from Fμ to the various translation networks is shown in Figure 2-50. Bits contained in rank 1 of Fμ are identified as FM1 bits; those in rank 2 of Fμ as FM2 bits. In addition, some bits of each rank pass through another stage of buffering before being used. In such cases, the bits carry another identifier. For example, bits

Table 2-3. CS Address Select Signals

Bit Groups Selected	SELP Signals	Sμ Reg Bits 4	5	6	7	Enables
00 → 03	SELP0-0 ↓ SELP7-0	0 ↓ 0	0 ↓ 1	0 ↓ 1	0 ↓ 1	$\overline{\text{ENRD-CS0}}$/$\overline{\text{ENWR-CS0}}$
	SELP8-0 ↓ SELPF-0	1 ↓ 1	0 ↓ 1	0 ↓ 1	0 ↓ 1	$\overline{\text{ENRD-CS1}}$/$\overline{\text{ENWR-CS1}}$
04 → 07	SELP0-1 ↓ SELP7-1	0 ↓ 0	0 ↓ 1	0 ↓ 1	0 ↓ 1	$\overline{\text{ENRDCS-0}}$/$\overline{\text{ENWR-CS0}}$
	SELP8-1 ↓ SELPF-1	1 ↓ 1	0 ↓ 1	0 ↓ 1	0 ↓ 1	$\overline{\text{ENRD-CS1}}$/$\overline{\text{ENWR-CS1}}$
08 and 11	SELP0-1 ↓ SELPF-1	0 ↓ 0	0 ↓ 1	0 ↓ 1	0 ↓ 1	$\overline{\text{ENRD-CS0}}$/$\overline{\text{ENWR-CS0}}$
	SELP8-1 ↓ SELPF-1	1 ↓ 1	0 ↓ 1	0 ↓ 1	0 ↓ 1	$\overline{\text{ENRD-CS1}}$/$\overline{\text{ENWR-CS1}}$
12 → 15	SELP0-2 ↓ SELP7-2	0 ↓ 0	0 ↓ 1	0 ↓ 1	0 ↓ 1	$\overline{\text{ENRD-CS0}}$/$\overline{\text{ENWR-CS0}}$
	SELP8-2 ↓ SELPF-2	1 ↓ 1	0 ↓ 1	0 ↓ 1	0 ↓ 1	$\overline{\text{ENRD-CS1}}$/$\overline{\text{ENWR-CS1}}$

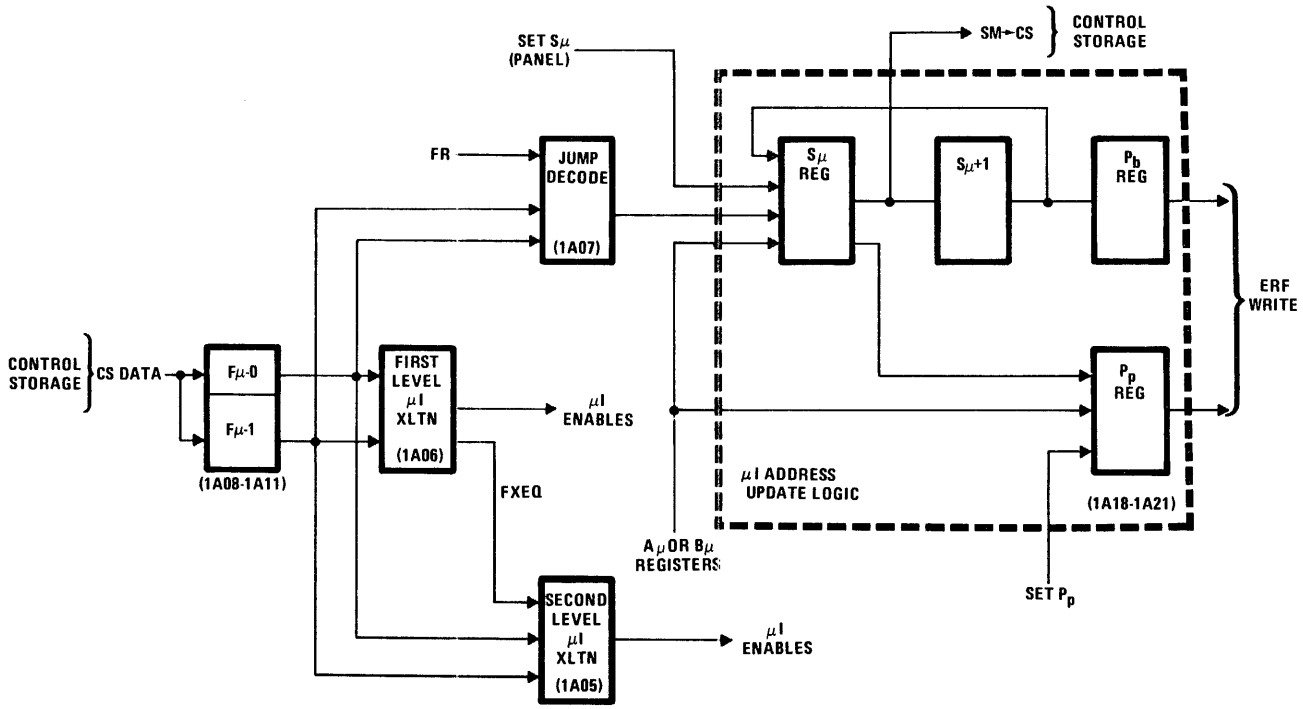


Figure 2-48. CS Control, Block Diagram

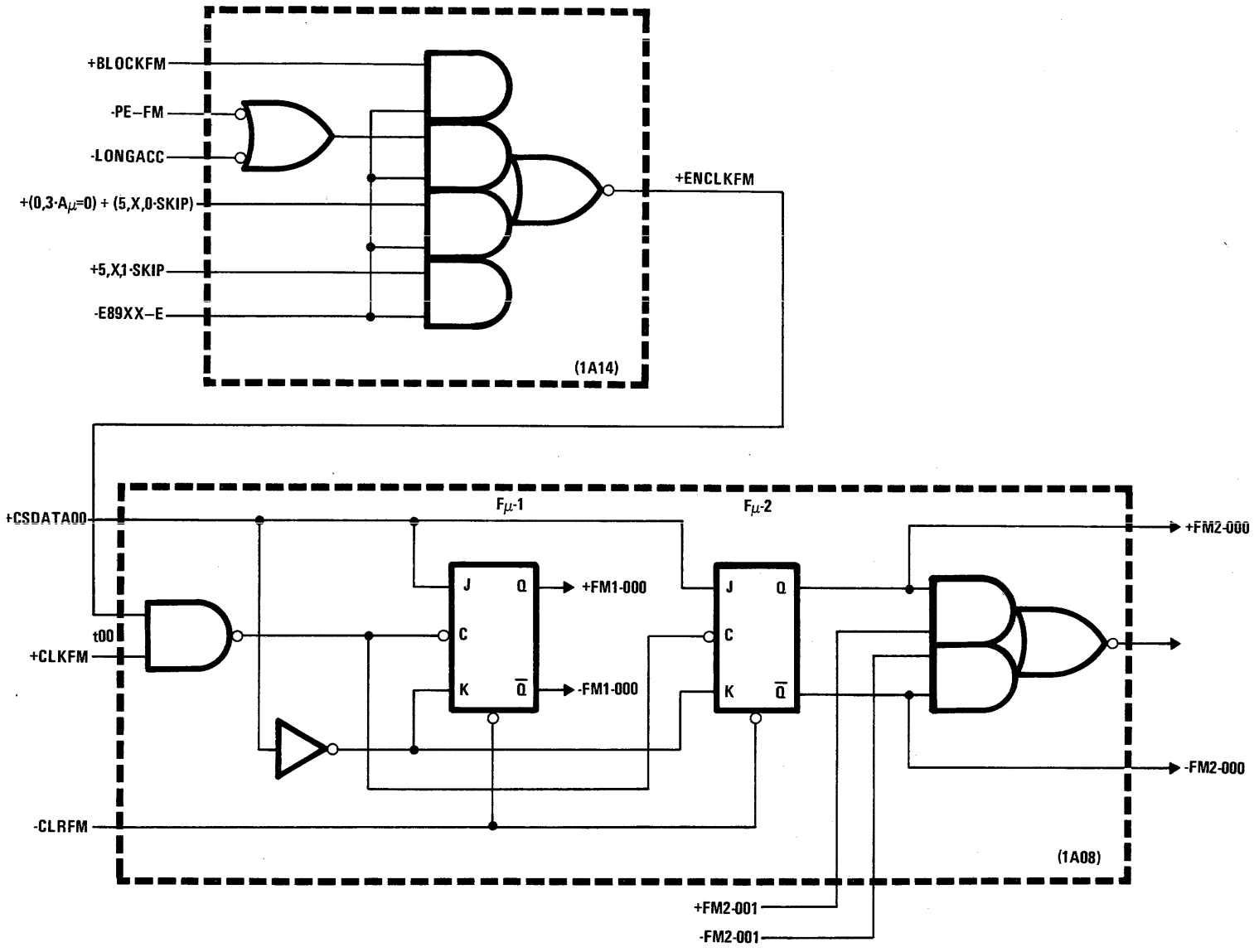


Figure 2-49. Fμ Register

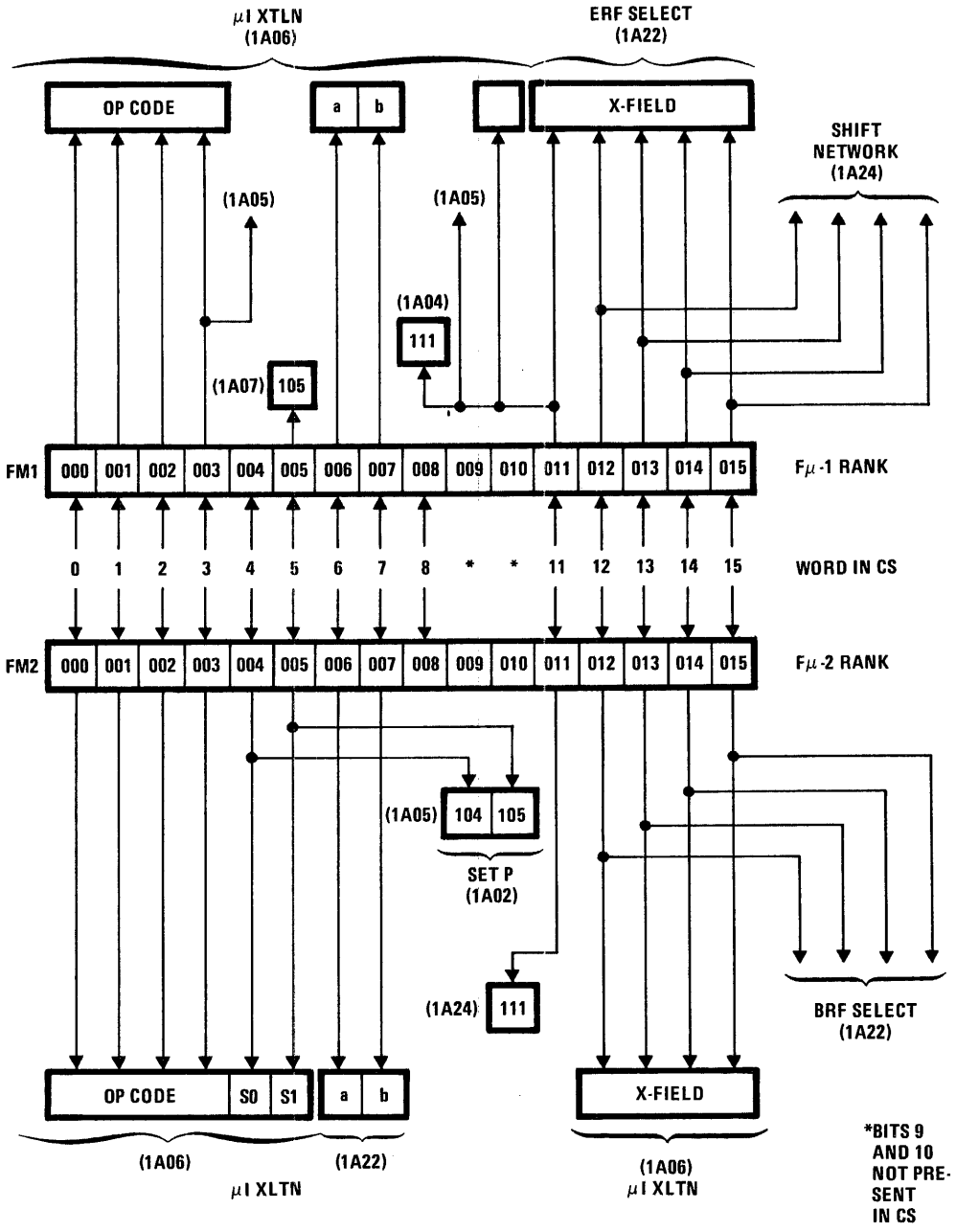


Figure 2-50. Micro Instruction Decoding

4, 5, and 11 of the Fμ-2 rank are directly interpreted by the μI translation logic as sub-operation codes S_0, S_1, and S_2. However, they are also routed to the set P logic through another stage of buffering. These bits are designated as bits 104 and 105 to differentiate from bits 004 and 005 coming directly from the Fμ-2 ranks. Note that bits 9 and 10 of the Fμ register are not used since the μI is only 14 bits in length. From the Fμ-1 and Fμ-2 ranks of the Fμ register, the μI bits are routed to the μI translation network for decoding the various bit fields of the μI, and to various other translation circuits for specialized decoding of certain μI bits for particular applications. Each of these translation networks shown in Figure 2-50 is discussed in more detail in the following paragraphs.

Micro-Instruction Translation

Translation of the μI function code and designators is accomplished by the μI translation logic. This logic, shown in block diagram form in Figure 2-51, consists of two parts: a first-level translation network and a second-level translation network. The first-level network translates bits of the Fμ register used to form the μI to generate the function (F) code, sub-operation (S) code, and a and b designators. The F code values are collectively referred to as FXEQ-X/X signals and FXEQ-X signals. The FXEQ-X/X signals represent an ORed combination of two adjacent Fμ code values. (For example, FXEQ-2/3 represents an F code of either 2 or 3.)

The FXEQ-X represents a single F code value (i.e., FXEQ-3 means an F code value of 3). The FXEQ-X/X signals are routed to the second-level translation network, and the FXEQ-X signals are combined with S code signals and the a and b designators to generate enable signals for specific μI's.

The second-level translation network decodes the FXEQ-X/X signals into particular F code values via the microcode bits 00-03 logic. These F codes are combined with additional S code values decoded by the micro instruction bits 04-05 logic to form signals which are used to set control flip-flops and execute other enable signals used by a particular μI. Generally speaking, the output signals generated by the second-level translation logic are of a more specialized nature, such as register enables for executing operations associated with individual μI's. This is in contrast to output signals from the first-level translation, which generates basic F and S codes, and signals applicable to a large number of μI's requiring similar operations (such as all μI's which generate a jump address). Each of the particular μI enable signals will be discussed in greater detail in the description of that section of logic used to implement the particular μI.

Jump Decode

The jump decode logic performs a second-level jump address decode of the FNJ (0,1) μI and a jump address decode of the JMP (9) μI. Formation of the FNJ jump address is called a second-level decode because of its relationship to the first-level jump address decode of the FRJ (0,2) μI when implementing MLI's via μI's (see the paragraph titled Implementing MLI's by μI's). Jump addresses formed by the jump decode logic for both the FNJ and FRJ μI's are routed to the Sμ register in place of the normal updated μI address to cause a jump to a new sequence of μI's in CS.

Formation of the FNJ jump address is performed in one of two ways, depending on the value of bit 06 of the FNJ μI (Fμ 06). If Fμ06 is 0, the jump address is formed as shown in part *a* of Figure 2-52: bits 4 through 9 of the jump address are made up of bits 7 and 11 through 15 of the FNJ μI in the Fμ register, bits 10 through 13 of the jump address are made up of bits 4 through 7 of the MLI of which the FNJ μI is a part and bits 14 and 15 are forced to zero. If Fμ 06 is 1, the jump address is formed as shown in part **b** of Figure 2-52: jump address bits 4 through 9 are made up of bits 7 and 11 through 15 of the FNJ μI (the same as for the FNJ/Fμ06=0 I). However, jump address bits 10 through 15 are forced to zero, except for bit 12 which is made up of bit 8 of the MLI. Formation of the JMP jump address is accomplished by transferring the jump address contained in bits 4 through 7 and 12 through 15 of the JMP μI to bits 8 through 15 of the new jump address as shown in part **c** of Figure 2-52.

For all three jump addresses, bits 0 through 3 are not altered from what they were before the jump address was formed. Bit positions 0 and 1 contain μI status information and are not used as part of the μI address. Bits 2 and 3, which define which 4096-word portion in CS is to be selected, remain unchanged also. In addition, bits 4 through 7 of the JMP μI address remain unchanged by the JMP μI (although they are incremented as necessary by the normal Sμ + 1 operation). Since all 12 bits of the jump address are retained, each of the branch μI's allow jumping through a 4096-word portion of CS. However, the μI's are usually implemented to jump only within a 256-word page. The JMP μI can jump to any location within a page (or 4096-word portion); however, the two FNJ μI's can jump only in certain increments because some of their jump address bits are preset by hardware. The FNJ/Fμ06=0 μI can jump only in 4-address incremented, starting at 0000* (0000, 0004, 0008, and so forth). The FJN/Fμ06=1 μI can jump only in 12-address increments, starting on 64-word boundaries

*All addresses represented in hexadecimal form.

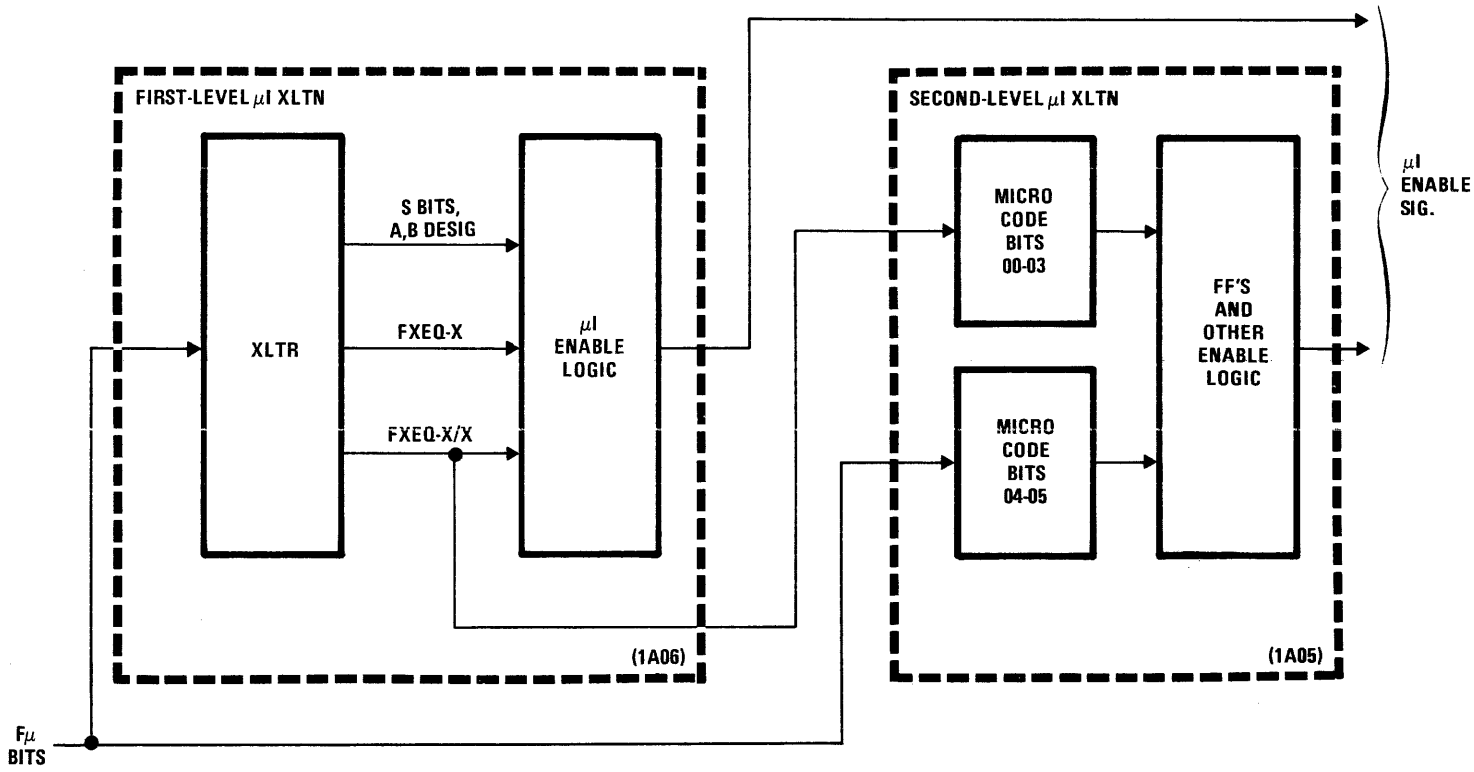
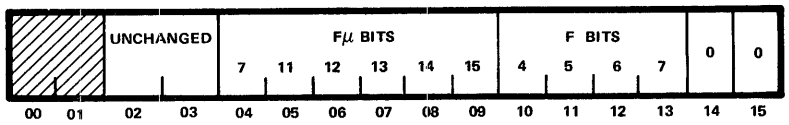


Figure 2-51. Micro Instruction Translation Block Diagram

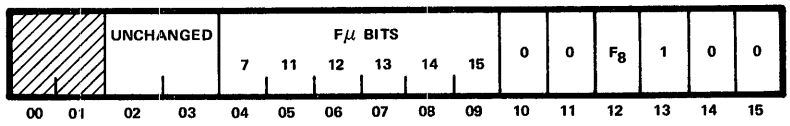
(0004, 000C, 0044, 004C, and so forth). All three jump addresses are loaded both into the Sμ register (next CS address) and Pp register (blockpoint address). However, if either μI occurs during E6 or E7 time, the jump address goes only to the Pp register (and then to Pμ) for use as the starting μI address for the processor's next time slice. (For details of the timing involved for this situation, see the paragraph, Branch Control.)

Referring to Figure 2-52 it can be seen that jump address bits 4 through 9 ($\overline{\text{JMP-04}}$ through $\overline{\text{JMP-09}}$) are formed in the same manner for the FNJ/Fμ06=0 and FNJ/Fμ06=1 μI's, i.e., they both are formed by bits 7, 11, 12, 13, 14, and 15 of the jump μI. Bits $\overline{\text{JMP-10}}$ through JMP-15, however are formed in a manner peculiar to that particular jump μI. Simplified logic showing the deviations of these jump address bits is shown in Figure 2-53. Bits $\overline{\text{JMP-08}}$, $\overline{\text{JMP-09}}$, $\overline{\text{JMP-14}}$ and $\overline{\text{JMP-15}}$ are formed in two different ways, depending on whether the μI is a FNJ or JMP. Bits $\overline{\text{JMP-10}}$ through $\overline{\text{JMP-13}}$ are formed in three different ways, depending on whether the μI is a FNJ/Fμ06=0, FNJ/Fμ06=1, or JMP. For a JMP μI, bits $\overline{\text{JMP-08}}$ through $\overline{\text{JMP-15}}$ are generated by appropriate Fμ register bits when enabled by FM1-000. This bit is a 1 for the JMP μI, since the JMP F code is 9_{16} (1001_2).

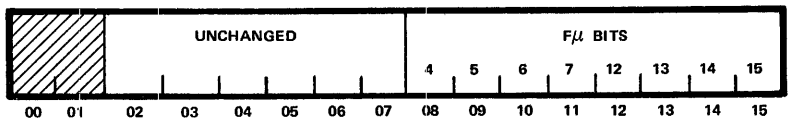
For both FNJ/Fμ06=0 and Fμ06=1, μI's bits $\overline{\text{JMP-08}}$ $\overline{\text{JMP-09}}$, $\overline{\text{JMP-14}}$, and $\overline{\text{JMP-15}}$ are derived in the same manner. Bits $\overline{\text{JMP-08}}$ and $\overline{\text{JMP-09}}$ are generated by Fμ register bits 14 and 15 when enabled by FM1-000. This bit is a 0 for the FNJ μI, since the FNJ F code is 0_{16} (0000_2). Bits $\overline{\text{JMP-14}}$ and $\overline{\text{JMP-15}}$ are forced to 1 (which forces address bits 14 and 15 in the Sμ register to 0) by the absence of an enable signal to make them 0. For a FNJ/Fμ06=0 μI (identified as FNJ 0 in the corresponding enable gates in Figure 2-53), bits $\overline{\text{JMP-10}}$ through $\overline{\text{JMP-13}}$ are generated from Fμ register bits 4 through 7. These bits are enabled by $\overline{\text{FM1-000}} \cdot \overline{\text{FM1-006}}$, where $\overline{\text{FM1-000}}$ defines the FNJ μI and $\overline{\text{FM1-006}}$ defines bit Fμ 06 as 0. For a FNJ/Fμ06=1 μI (identified as FNJ 1 in the corresponding enable gates in Figure 2-53), bits JMP-12 is generated from bit 8 of the MLI (contained in the F register) and bit JMP-13 is set to 1 (via bit 6 itself of the FNJ/Fμ06=1 μI). These two jump address bits are enabled by $\overline{\text{FM1-000}} \cdot \overline{\text{FM1-006}}$. Bits $\overline{\text{JMP-10}}$ and $\overline{\text{JMP-11}}$ of the FNJ/Fμ06-1 μI are forced to 1 by the absence of an enable signal to make them 0. This forces address bits 10 and 11 in the Sμ register to 0.



A. FNJ JUMP ADDRESS, Fμ 06 = 0



B. FNJ JUMP ADDRESS, Fμ 06 = 1



C. JMP JUMP ADDRESS

Figure 2-52. FNJ and JMP Jump Address Formats

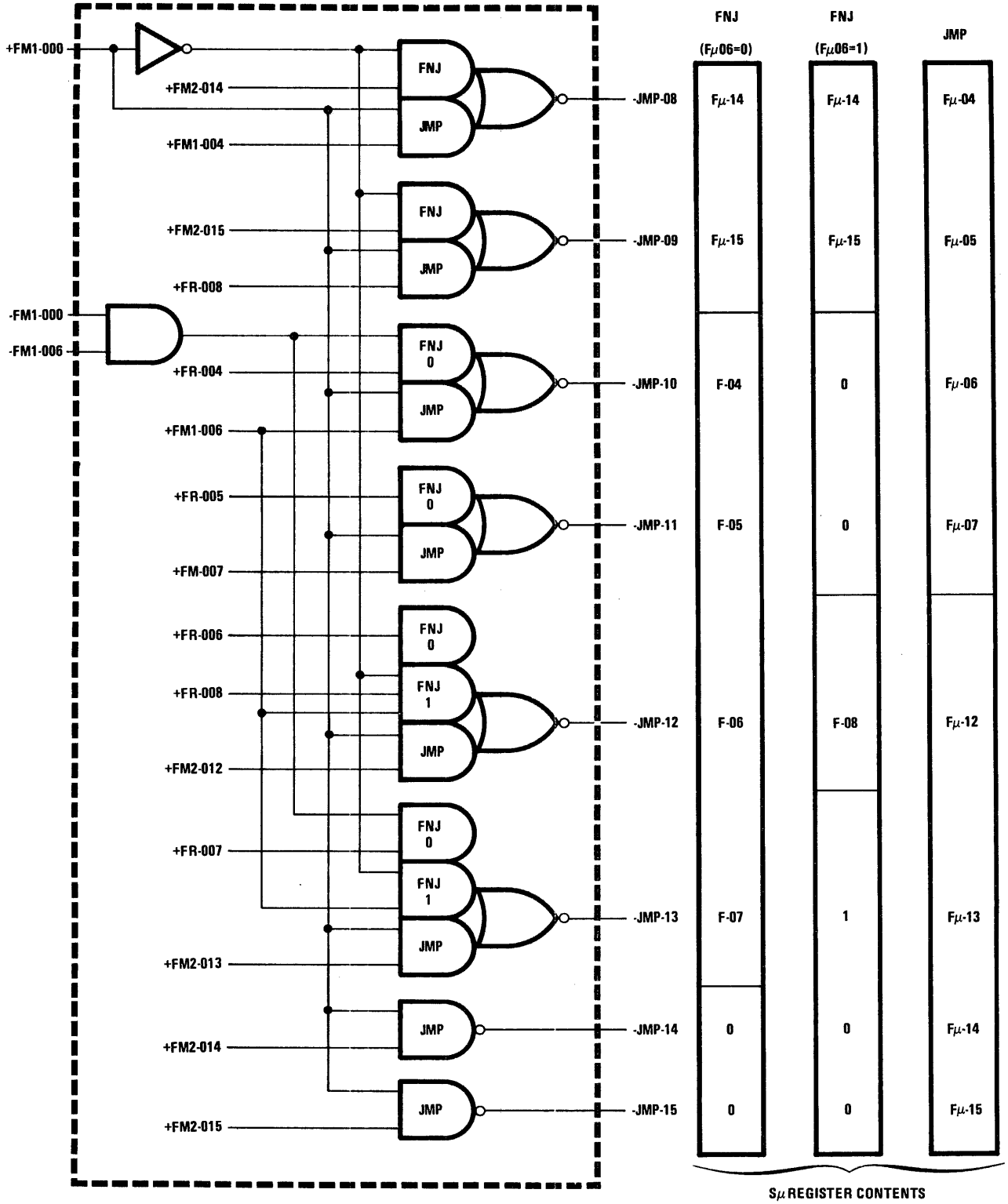


Figure 2-53. Generation of FNJ and JMP Addresses

Micro-Instruction Address Update

Sμ Fan-In

The Sμ fan-in logic selects the 14-bit CS address to be loaded in the Sμ register from a number of sources, as controlled by corresponding enable signals. A simplified diagram of the logic is shown in Figure 2-54.

Note that the signal fan-in for bits 2 and 3 of the μI address is different from that for bits 4 through 15. During the time slice, the Sμ register is normally fed with the updated μI address from the Sμ+1 logic via the $\overline{S\mu+1}$ bits. This logic adds one to the present μI address to form the next μI address. This updated address is gated directly through the Sμ fan-in logic in the absence of an enable signal for some other input to the Sμ fan-in. During a FNJ or JMP μI, however, a new jump address is loaded into the Sμ register via the JMP bits. This jump address affects only bits 4 through 15 of the μI address; therefore, the JMP bits do not appear as inputs to the bit 2 and 3 stages of the Sμ fan-in logic. These JMP bits are enabled by ENJP-SM. The ALU inputs represent data from either the Aμ or Bμ register to be stored in the active processor's assigned Pμ register in the Extended Register File (ERF), as the starting address for the active processor's next time slice. This data will be transferred to the Sμ register for either a STA or STB μI when the μI X-field specifies the Pμ register, and the μI is being executed at some time other than E6 or E7.

Since the STA and STB μI's are both blockpoint instructions, a Sμ→Pp transfer will take place to transfer the Sμ register contents to Pp for storage in the assigned Pμ register. The $\overline{ALU}$ inputs also represent the contents of the Bμ register used to access the CS during a ROM μI, as follows: B→Sμ→CS→CS Scan register. If beginning a new time slice, the starting address for the new time slice will normally come from the processor's Pμ register, via the $\overline{PU}$ bits. However, if no other processor has requested a time slice, the present processor may run in consecutive cycles (CC) if its CC bit is set. For this case, the next μI address is read from the Pp register, which holds the μI address updated by the last blockpoint μI. (This address would normally be stored in the processor's Pμ register for use as the starting μI address for the processor's next time slice. However, since the processor will run through the next time slice in CC, there is no need to go through this extra step of storing the contents of Pp in Pμ). Inputs from either the Pμ register or Pp register are enabled by ENPM-SM. The $\overline{SETS}$ bits represent a CS address set by the System Control Panel which defines the starting address of a Panel function. These functions allow data to be read from or written into CS in individual locations or in blocks during the maintenance mode or to initially load the CS via the Reset/Load routine. The address generated by the System Control Panel is only 12 bits in length (bits 4 through 15); therefore, all panel function sequences must be located in the first 4096-word portion of CS. (At present, only seven of the 12 SETS lines have been assigned address functions: 7, 10, 11, 12, 13, 14, and 15. The remaining five lines are tied to a logic "1" to simulate 0 inputs to the corresponding bit positions of Sμ.)

Sμ Register

The Sμ register is a 16-bit register that contains the address of the next μI to be read from CS. This address is contained in the lower 14 bits (bits 2 through 15) of the register. The upper two bits (bits 0 and 1) normally contain the two μI status bits: Overflow (OV) and Link (LK). This format of Sμ is shown in part *a* of Figure 2-55. Under certain conditions, however, the upper two bits contain other information as shown in parts *b* and *c* of Figure 2-55. During a CS breakpoint operation initiated from the System Control Panel (provided that Sμ is not selected for display by the Console Address register indicators), bits 0 and 1 are forced to 0 so that only the 14-bit CS address is used for breakpoint comparison purposes. This format is shown in part *b* of Figure 2-55. (If Sμ is selected for display, then bits 0 and 1 do participate in the breakpoint comparison.) During a Reset/Load operation, bit 0 indicates that a burst check error occurred and bit 1 indicates that either the CS load is complete (bit 1 clear) or that the FRJ decode address table (AT) load is being loaded (bit 1 set). This format of Sμ is shown in part *c* of Figure 2-55. In addition, bits 8 through 15 of Sμ are specifically interpreted as an AT address during the AT load portion of a Reset/Load operation and the AT read/write portion of a CS read/write operation.

Simplified logic showing details of the 14-bit portion of Sμ is shown in Figure 2-56. (Details of the 2-bit portion are discussed in the paragraph titled Status Logic and the paragraph titled Disc CS Load.) Each bit stage consists of a flip-flop clocked at t80 of every minor cycle when ENCLKSM is present. This enable is generated constantly, except upon occurrence of a specific condition to inhibit the enable. Signal BLKSMS inhibits the enable for all but one minor cycle (E0) of a time slice during a CS read or CS write operation initiated from the System Control Panel. These operations access CS only once during each time slice assigned to the Panel; therefore, only one μI address update and consequent clocking of the updated address back into Sμ is allowed per time slice. Signal BLOCKS inhibits ENCLKSM during (1) an MS access, (2)

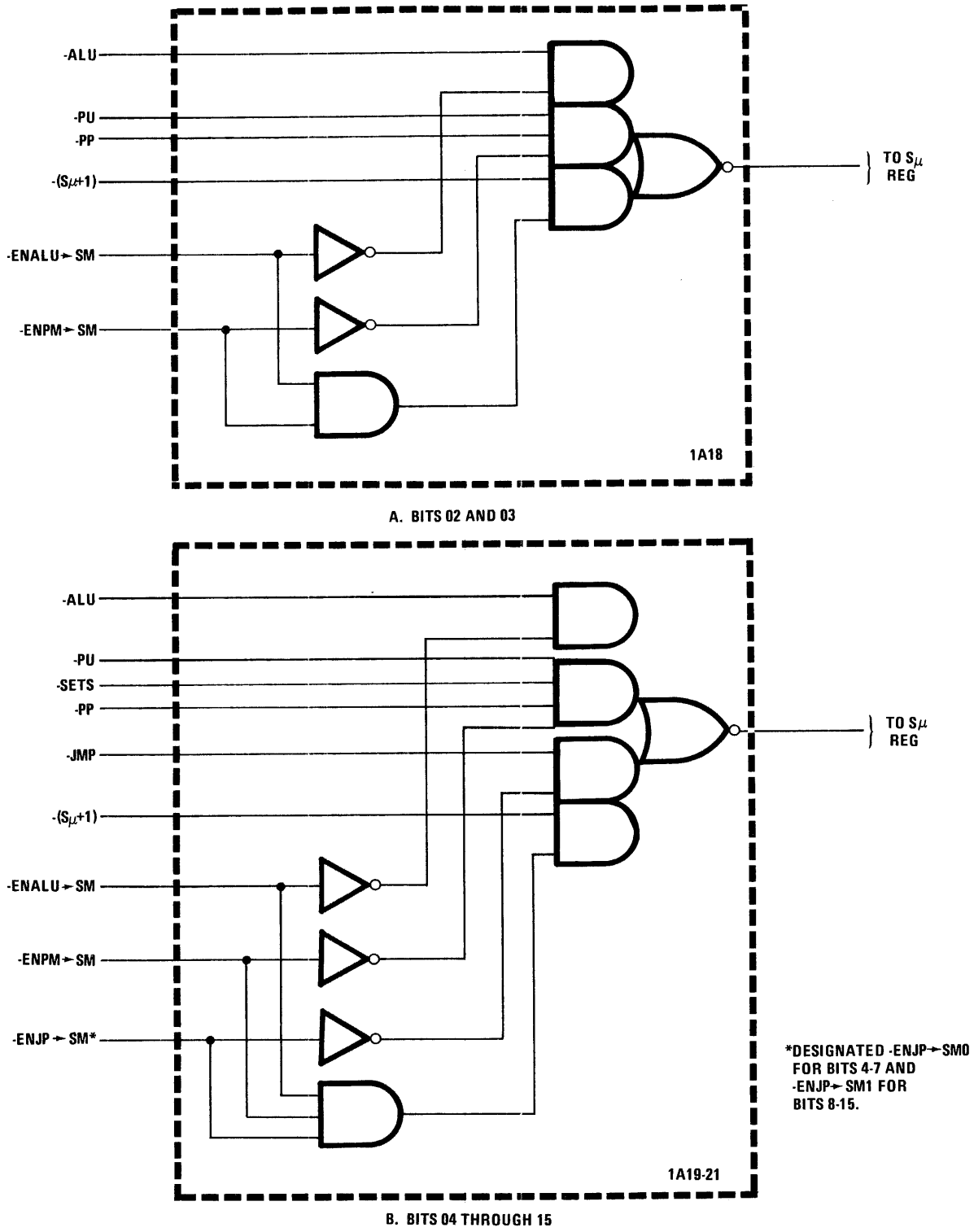


Figure 2-54. Sμ Fan-In

a null condition (none of the eight processors has requested a time slice), or (3) E6 and E7 if a processor is operating in the Consecutive Cycle (CC) mode. An access to MS increases the cycle time from 800 to 900 or 1000 nanoseconds to allow for address propagation through the Register Option. This extra time essentially sets up a *hold* condition during which Sμ must be blocked. During a null condition, clocking Sμ with a new μI address would be meaningless if no processor was running to execute the μI. During CC operation, Sμ must be blocked during E6 and E7 to prevent clocking in addresses that would normally be those of the first and second μI's to be executed by the next processor assigned a time slice. Since the present processor will continue executing, the next μI address must come from Pp. Blocking Sμ during E6 and E7 allows this address to be obtained from Pp. Signal $\overline{\text{EXCEPT}}$ is ANDed with BLOCKS to override BLOCKS during a CS load operation to clear Sμ to address 0000_{16}.

Signals 2,X+4,X+C,X, and SHIFT blocks Sμ for one of the following conditions: (1) Execution of a SUM, DSUM, CMP, or CMU (2,X) μI, when preceded by a Feeder Load μI. For this condition, the 2,X μI must be delayed one minor cycle to allow the Feeder Load μI operand to propagate through the ALU. (2) Execution of a SDW or SDB (4,X) or D→A (C,X) μI if part of an MS read operation. For this condition, the 4,X or C,XμI cannot be executed until E5, at which time the data read from MS is available. (3) Execution of a shift μI. A shift μI takes two minor cycles to execute; therefore, clocking the address of the next μI address into Sμ must be delayed for one minor cycle. For the last two conditions, blocking of Sμ is overriden if the μI is executed at E6 or E7 by $\overline{\text{E67IDL}}$. This allows the addresses of the first and second μI's of the next time slice to be clocked into Sμ to begin this time slice in the normal manner.

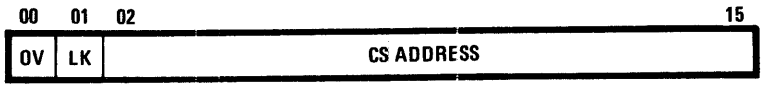
The address bits from Sμ are fanned out to several destinations. All 14 bits are routed to the Sμ+1 logic for address updating, and to CS via the SM-CS signals to read the next μI. In addition, all 14 bits are sent to FRJ decode logic to form jump addresses as discussed in the paragraph titled Jump Decode. Bits 2 and 3 of the address are routed to the CS loader logic via SM-LD. These bits are used during the Reset/Load routine to inform the loader that all n X 1024 words of a CS unit have been loaded. Bits 4 through 15 of the address are routed to the Console Data register in the System Control Panel for purposes of displaying the address during maintenance operations (bits 2 and 3 are also sent to the Console Data register after they pass through the CS loader logic.)

Sμ+1 Logic

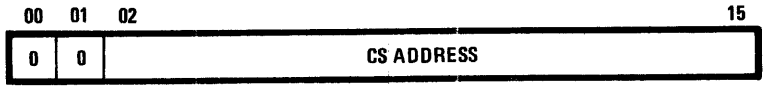
The Sμ+1 logic updates (increments by one) the present μI address, and routes the updated address back to the Sμ register as the next μI address. Since individual μI's for a given sequence are stored in consecutive locations in the CS, this update process enables reading all μI's of a particular sequence. A portion of the Sμ+1 logic, that used to update bits 2 and 3 of the μI address, is shown in Figure 2-57. Essentially, the Sμ+1 consists of an exclusive-OR gate for each bit stage, which functions as a simple counter whenever a group carry-in signal (GX-CIN) is present. There are four such group carry-in signals: GO-CIN for bits 2 and 3 of the Sμ register (the two MSB's of the μI address), G1-CIN for bits 4 through 7, G2-CIN for bits 8 through 11, and G3-CIN for bits 12 through 15. The G3-CIN signal is always enabled, since this lowest-order bit group will always be counting. Each higher-order bit group, however, will be counting depending on the ability of a next lower-order bit group to satisfy an address update within its group without having to propagate a carry into the next higher-order group. For that reason, G0-CIN, G1-CIN and G2-CIN are generated by group propagate signals ($\overline{\text{GX-PROP}}$) from a lower-order group: G2-CIN by $\overline{\text{G3-PROP}}$, G1-CIN by $\overline{\text{G3-PROP}}$, and $\overline{\text{G2-PROP}}$, and G0-CIN by $\overline{\text{G3-PROP}}$, $\overline{\text{G2-PROP}}$, and $\overline{\text{G1-PROP}}$. Looking at the μI address update example in Figure 2-57, the μI address at time t is to be incremented by one to form the next μI address at time t'. At time t, bits 2 and 3 of the Sμ register equal "0" and "1", respectively, to generate inputs to the exclusive-OR gates of each stage which are low (L) and high (H), respectively. Since all lower-order bits of the Sμ register equal one, the three $\overline{\text{GX-PROP}}$ signals are low as shown. This condition produces a high G0-CIN signal as the other input to the two exclusive-OR gates. The resulting outputs from each exclusive-OR gate are routed back to the set side of each Sμ register flip-flop to set the bit 2 stage and clear the bit 3 stage. The resultant change on the outputs of these two flip-flops after updating is shown in the dashed portion of the μI address. The three low $\overline{\text{CX-PROP}}$ signals cause G1-CIN and G2-CIN to go high along with G0-CIN. These high carry-in signals, along with G3-CIN (which always remains high) cause bit groups 4-7, 8-11, and 12-15 to be incremented by one also, to form the complete new updated μI address at time t* as shown in Figure 2-57. The updated result is fed both to the Sμ register and the Pb register.

Pb Register

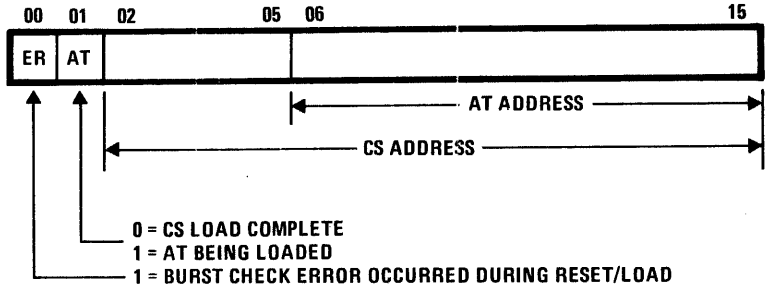
The Pb register is a buffer register that stores the starting address for the active processor's next time slice in Pμ for certain conditions when the starting address normally obtained from Pp is no longer available. These abnormal conditions are discussed in the paragraph titled Storing of Starting μI Address.



A. NORMAL μI EXECUTION FORMAT



B. CS BREAKPOINT FORMAT



C. CS READ OR CS WRITE FORMAT

Figure 2-55. Sμ Register Formats

Pp Register

The Pp register holds a μI address formed during the active processor's present time slice for use as the starting address for the processor's next time slice. The register is loaded from a number of sources through the Pp fan-in logic, as shown in Figure 2-58. The fan-in logic consists of 14 stages, one for each of the 14 μI address bits. Note that the stages for bits 2 and 3 (part a) are different from those for bits 4 through 15 (part b). Normally, Pp receives the starting address from the Sμ register after having been updated by the Sμ+1 logic. This transfer is enabled by the absence of any other enable signal and occurs during execution of a blockpoint (BP) μI. Certain error conditions, such as a CS Parity Error or Bounds Error will alter normal program operation by jumping to an error recovery routine. (Generation of starting addresses for these routines is discussed in the paragraph titled Set Pp Logic.) These error conditions will force a particular 12-bit starting address for the next time slice into the Pp register via the SETP bits. These bits are enabled by ENSPECPP which is generated for these conditions by a TRAP signal. Enable ENSPECPP is also generated during execution of a FRJ, FZJ, RNI1, or RNI2 μI. These four μI's cause a programmed jump (as opposed to the unconditional error recovery jumps) to another part of the MLI routine at the start of the next time slice. Depending on the μI, the enable gates in the 14 FRJ bits (bits 2 through 15) to form the corresponding jump address.

The JMP bits form a 12-bit jump address when executing a JMP μI. This address can be formed in different ways, depending on when the μI is executed in the time slice. The JMP bits, therefore, are gated by two different

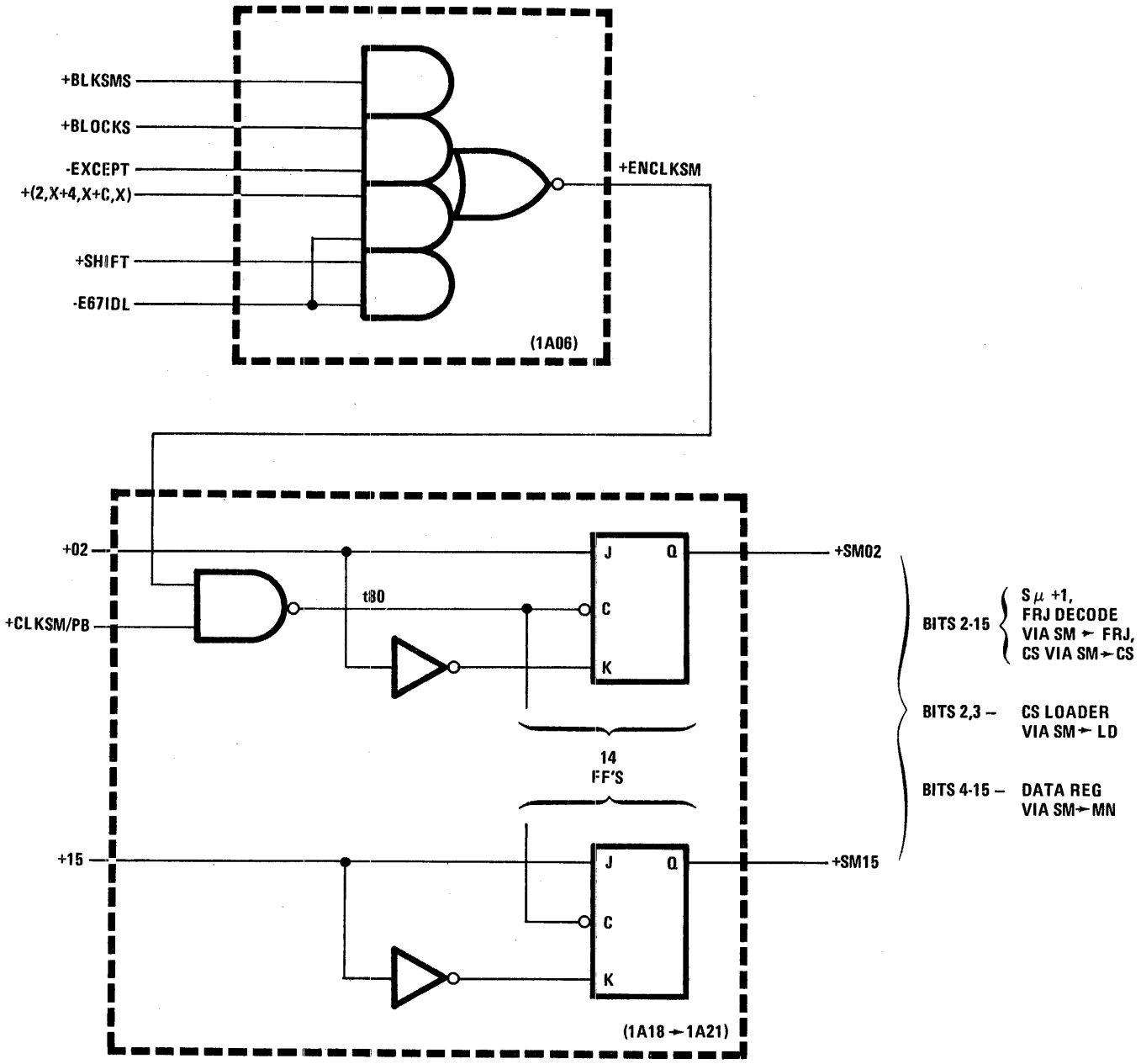
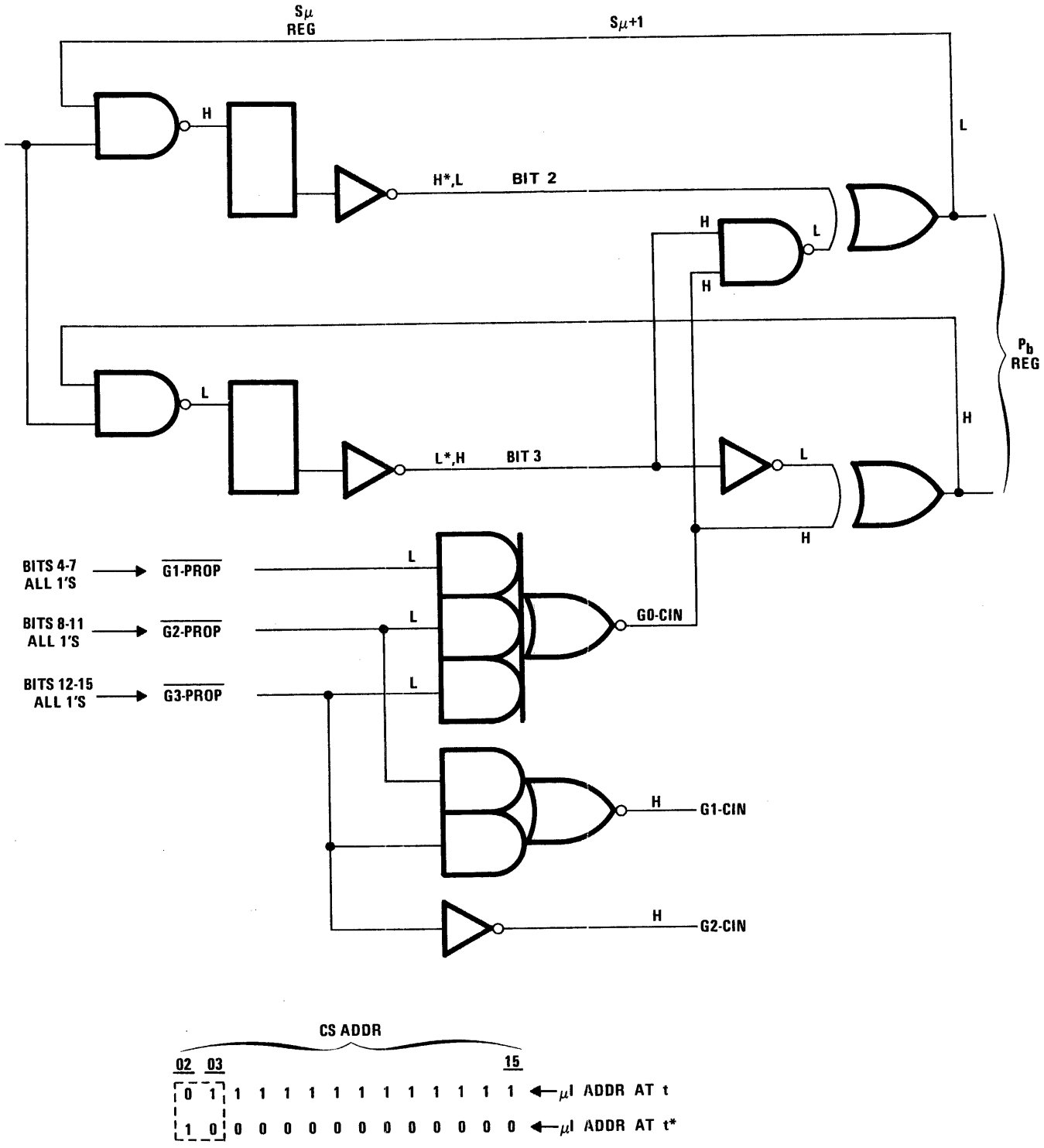


Figure 2-56. Sμ Register



*REFER TO TEXT DESCRIBING THIS ILLUSTRATION.

Figure 2-57. Sμ+1 Logic

enables: ENJP-PP0 for bits 4 through 7 and ENJP-PP1 for enables 8 through 15. (See the paragraph titled Storing of Starting μI Address for a more detailed discussion.) The 14 ALU inputs represent data from within the A μ or B μ register to be stored in the active processor's assigned Pμ register in the ERF. This data will be transferred directly to the Pp register for either a STA or STB μI when the μI X-field specifies the Pμ register, and the μI is being executed at E6 or E7 time. (If executed this late in a time slice, the STA and STB blockpoint instruction will not be able to execute a Sμ Pp transfer; therefore the address must be loaded into Pp directly.) The FRJ inputs to the bit 2 and 3 stages of the Pp fan-in logic and their corresponding enable, ENFRJ-PP, are not used at present. These bits would normally be used to select one of the four possible CS modules to which a jump would be made when executing a FRJ μI. Since only one CS module is used at present, these three lines are tied to +5 vdc which effectively removes them from the fan-in logic.

The selected output, from the Pp fan-in logic is fed to the set side of the Pp register in true form and to the clear side in complement form. The data is stored in the register upon occurrence of clock enable ENCLKPP and clock signal CLKPP. The stored data is fed to the ERF write logic.

ERF Write Fan-In

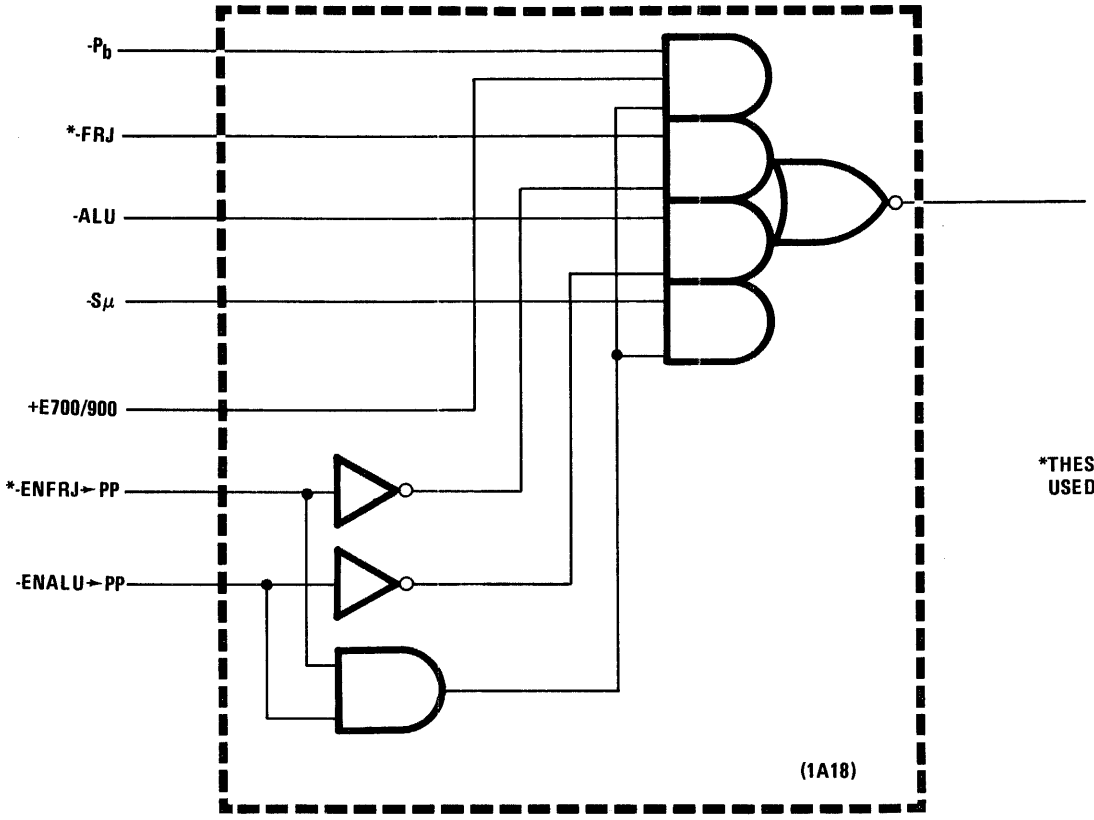
The ERF write fan-in logic provides data inputs to F_{RF} and Pμ registers, which comprise the Group I ERF, for storage therein. Generally, data is stored in Pμ during WO unless the last μI performed in the time slice was a CIO μI and the condition for exiting from the I/O routine was not met. This condition suppresses the Pp Pμ transfer, causing the routine to be repeated. In a similar manner, data stored into F_{RF} during WI is conditioned by occurrence of the F_b register clock signal, as discussed in the paragraph entitled F_b and F Registers. The data to be stored will be the result of either or both of the following operations: (1) store data in F or Pμ, or (2) store MLI and next μI address in F_{RF} and Pμ, respectively. Although similar in execution, these two operations result from different conditions. The second operation always occurs during a major cycle, to enable continuing with the present processor's program during the next assigned time slice. The first operation is a function only of a particular μI routine, and may or may not occur during every major cycle.

A diagram of the ERF write fan-in logic is shown in Figure 2-59. This logic is fed with all 16 bits that will be stored in both F_{RF} and Pμ at the end of a time slice. Data to be stored in F_{RF} comes from the Fb register and ALU fan-out logic. The Fb register contains the MLI presently being executed. Inputs from the ALU fan-out represents data to be stored in F_{RF} as a result of a Register File Write μI when F_{RF} is specified as the storage register. Data to be stored in Pμ consists of the two status bits, Overflow and Link, and the 14-bit next μI address updated by a BP μI. This address is derived from either Pp, Pb, or a combination thereof, as determined by the type of BP operation performed and at what time in the time slice. Since data from both Pp and Pb is combined, the corresponding enables are divided so that the upper 8 bits going to Pμ (two status bits and bits 2 through 7 of the next μI address) and the lower 8 bits (bits 8 through 15 of the next μI address) can be gated separately.

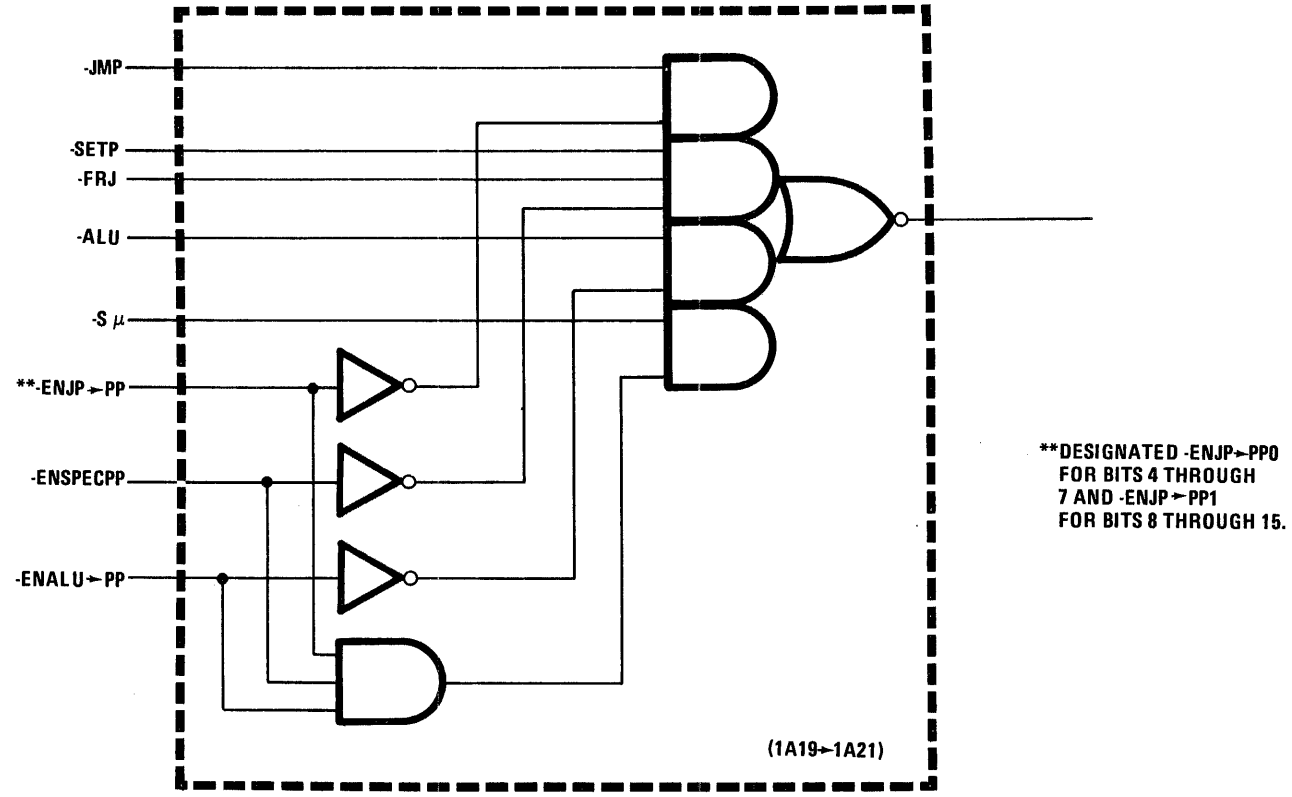
Storing of Starting μI Address

Storing the starting μI address (that is, the address of the first μI to be executed during the active processor's next assigned time slice) in the processor's assigned Pμ register enables the processor to continue executing its task during the next assigned time slice. This starting μI address is formed by the last blockpoint (BP) μI executed during the present time slice. Usually, this address is formed by adding +1 to the BP μI address so as to form the address of the next sequential μI in the MLI being executed. In some cases, however, the starting μI address will be a branch address to cause a jump to a different μI routine. Branch μI's are also BP μI's, so they enable the jump address to be stored in Pμ. Generally speaking, there are four different conditions that govern how the starting μI address is formed and how it is stored in Pμ. These are: (1) execution of a non-branch BP μI at E0 through E6, (2) execution of a non-branch BP μI at E7, (3) execution of a branch BP μI at E0 through E6, and (4) execution of a branch BP μI at E7.

Forming the starting μI address by executing a non-branch BP μI at either E0 through E6 or E7 differs mainly in the register used to hold the address until it is sent to Pμ. Execution of a non-branch BP μI during E0 through E6 loads the starting μI address (BP μI address +1) into the Pp register during the minor cycle that the μI is executed. Timing for such a situation, that of BP μI executed at E4, is shown in part *a* of Figure 2-60. The address for the BP μI (104) is clocked into Sμ at E280. During E3, the μI is read from CS and the contents of Sμ are updated by 1 and clocked back into Sμ at E380. At E480, The BP address +1 (105) is clocked into Pp for storing into Pμ at W0. Execution of a non-branch μI at E7 is similar to that executed at E0 through E6 in that the starting μI address is formed in the same way. For this



A. BITS 2 AND 3



B. BITS 4 THROUGH 15

Figure 2-58. Pp Fan-In

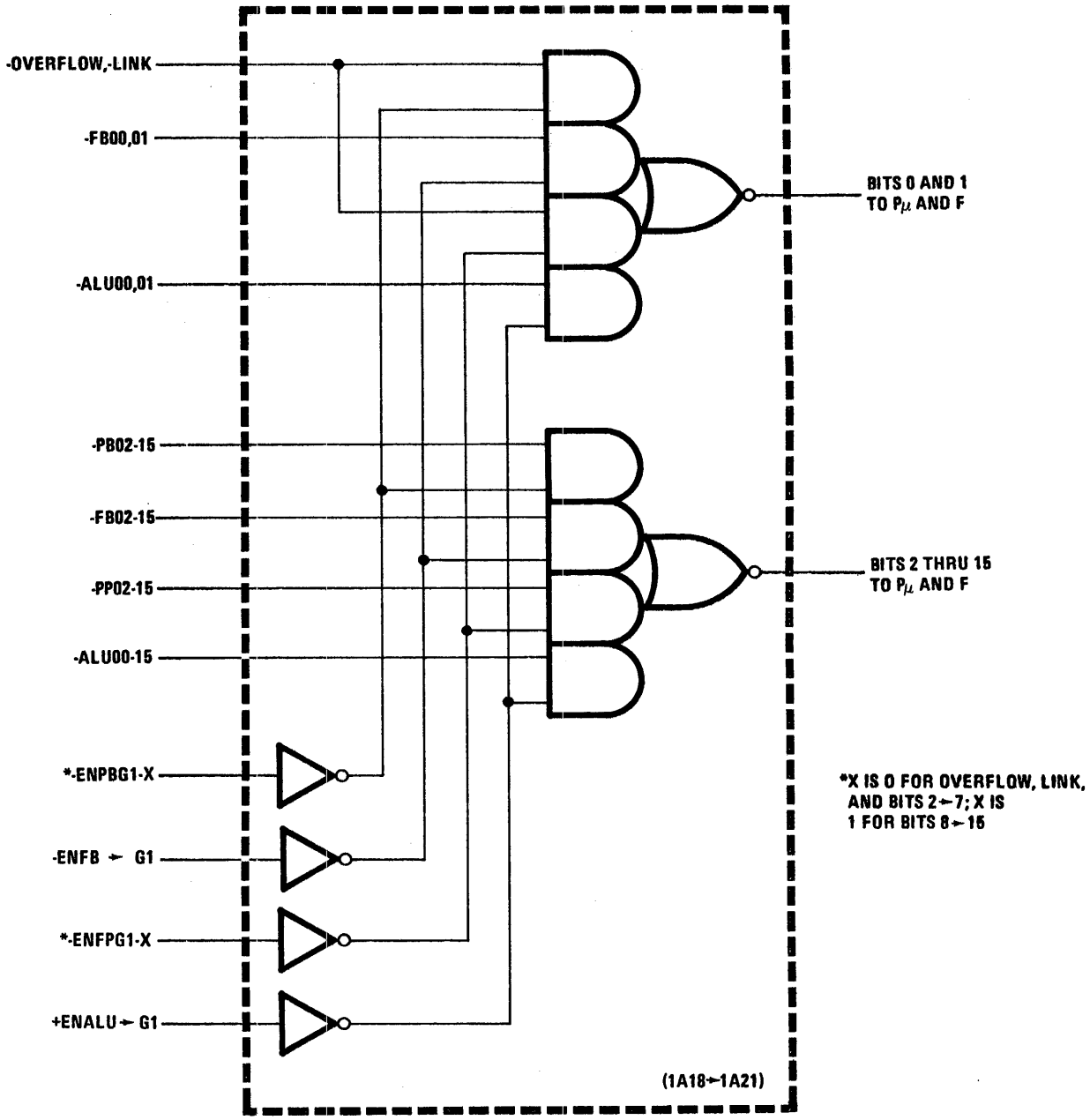


Figure 2-59. ERF Write Fan-In

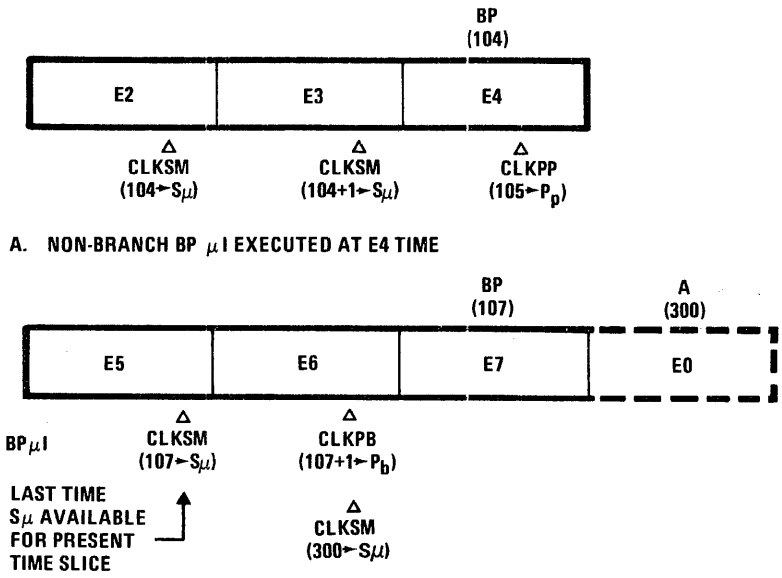


Figure 2-60. Use of Pp and Pb to Hold Starting μI Address

situation, however, the starting μI address must be clocked into Pb immediately after being updated by the Sμ+1 logic. This is because Sμ is no longer available to route the updated address from the Sμ +1 logic to Pp. By the time that the BP address has been updated, Sμ has been loaded with the first μI address for the following time slice. This situation is shown in part *b* of Figure 2-60. The BP μI address (107) is clocked into Sμ at E580 in preparation for executing the μI at E7. At E680, however, Sμ, is loaded with the address of the first μI to be executed during the following time slice (300). Therefore, the starting μI address (108) must be held in Pb until it can be stored in Pμ at W0.

Use of the Pp and Pb registers for form the starting μI address for non-branch BP μI's is shown in Figure 2-61. This figure shows the contents of Pp and Pb being fed through the ERF write fan-in logic by means of enables generated for the non-branch BP μI's, as shown in the top half of Table 2-4. For all non-branch BP μI's except the SUM, DSUM, CMP, and CMU μI's, the starting μI address is formed exactly as discussed in Figure 2-61: BP+1→Pp→Pμ if executed at E0 through E6 or BP+1→Pb→Pμ if executed at E7. (The CIO1 and CIO2 μI's are the only exceptions that they cannot be executed at E7.) The SUM, DSUM, CMP, and CMU (2,X) μI's differ from the other non-branch BP μI's in that they sometimes use Pb as a holding register if executed at E6 as well as at E7. The criterion which determines if Pb instead of Pp is to be used is whether the μI preceding the 2,X μI was one which altered the contents of Aμ and Bμ. As discussed in the paragraph titled Cycle Delay Logic, the Cycle Delay flip-flop is set if such is the case and the 2,X μI executed at E6 also extends into E7 as well. Timing for a SUM μI executed at E6 for both the above conditions, shown in Figure 2-62, illustrates the difference in using either Pp or Pb as the holding register.

Part *a* of this figure shows a SUM μI executed at E6 preceded by a LDW μI at E5. Since a LDW μI does not alter the contents of Aμ and Bμ, the SUM μI can be executed in one minor cycle. Since the μI following the SUM is not a blockpoint (NBP) μI, the Pp register is fed with the updated SUM μI address which forms the starting μI address. The contents of Pp are routed to the ERF fan-in logic via the ENPPGI enables at E780. Part *b* of Figure 2-62 shows a SUM μI executed at E6 preceded by a LAW μI at E5. Since a LAW does alter the contents of Aμ and Bμ, the Cycle Delay flip-flop is set to inhibit clocking the sum of Aμ and Bμ to register X until E7. The logic that generates enables ENPPGI and ENPBGI to gate Pp and Pb data through the ERF write fan-in logic is fed with inputs from the Fμ translation logic. Since the SUM μI still resides in Fμ at E7, the ENPPGI and ENPBGI logic assumes that a new one-cycle BP μI was executed at E7, and that its updated address (105+1) is now in Pb and should be routed to Pμ. What is in Fμ, however, is not a new BP μI executed at E7 but the SUM μI begun at E6. Therefore, the contents of Pb must reflect the updated SUM address (104+1), which is made available at E6. As a consequence, Pb must be clocked with this address at E580 to produce a meaningful starting μI address to be sent to the ERF fan-in at E780.

The branch BP μI's differ from non-branch BP μI's in that they form the starting μI address as a result of a branch to a new address. These μI's generate ERF fan-in enables as shown in the bottom half of Table 2-4. The CLR, STA, STB, and AND (1,X) μI's when X specifies Pμ form the starting μI address by routing a 14-bit branch address (bits 2 through 15) from the ALU to both Sμ and Pp if executed at E0 through E5 or to Pb only if executed at E6 or E7. (If executed at E0 through E5, the branch address formed becomes both the jump address to branch

to a new subroutine during the current time slice and the starting μI address for the next assigned time slice.) The 1,X μI's are normally considered non-branch (register file write) μI's. They act as branch μI's, however, when the μI X-field specifies Pμ as a file register in which to store data. For this special case, the path to store data in Pμ is through Sμ. Since Sμ is effectively loaded with a new address, the μI essentially becomes a branch μI. the contents of the Pp are then routed to the ERF fan-in logic via the ENPPGI enables at E780. Since the starting μI address formed by these BP μI's does not involve an address update through Sμ, the address can always be held in Pp until the end of the time slice. The FJN, FRJ, FZJ, RNI and JMP μI's differ from the 1,X μI's in that they do use a portion of Sμ to form the jump address and, therefore, the starting μI address. For the FRJ, FZJ, and FNI μI's, bits 2 and 3 of Sμ are used along with the jump address formed as bits 4 through 15. These two bits from Sμ, which enable a jump to another 4096-word CS module, are actually the result of an Sμ+1 update. Since the two bits always go to Pp regardless of when in the time slice the μI's are executed, (either from Sμ or from Pb) the resultant 14-bit address can always be obtained from Pp via ENPPGI enables. For the JMP μI, bits 2 through 7 of Sμ are used in connection with bits 8 through 15 from Fμ to form the jump address. Bits 2 through 7 are derived from the Sμ+1 logic to enable a jump not only to a different 4096-word module in CS but also to a different 256-word page within the module. These upper six bits of the jump address are treated in the same was as the upper six bits of the updated address of a non-branch BP μI; that is, the bits are routed to Pp if the JMP μI is executed at E0 through E6 or to Pb if executed at E7.

The difference between using Pb for a JMP μI as compared with using it for the non-branch BP μI's is that only the upper half of the register is used. Therefore, a JMP μI executed at E7 requires only that the upper half of Pb be enabled through the ERF fan-in logic. This is implemented by generating enable ENPBG1-0 only. The lower half of the jump address than is enabled through the ERF fan-in logic from Pp via ENPPG1-1.

Set Pp Logic

The Set P_p logic generates starting addresses of the RNI and storage error routines stored in CS upon detection that such a routine must be executed. These addresses are

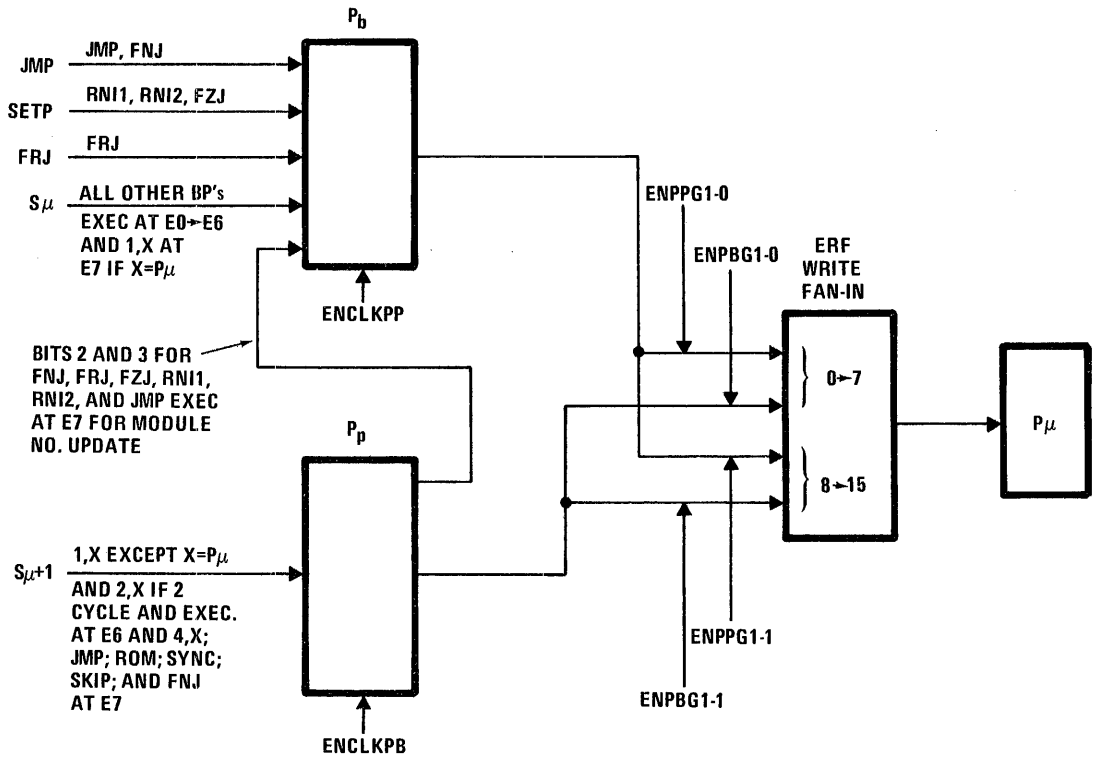


Figure 2-61. Derivation of Starting μI Address

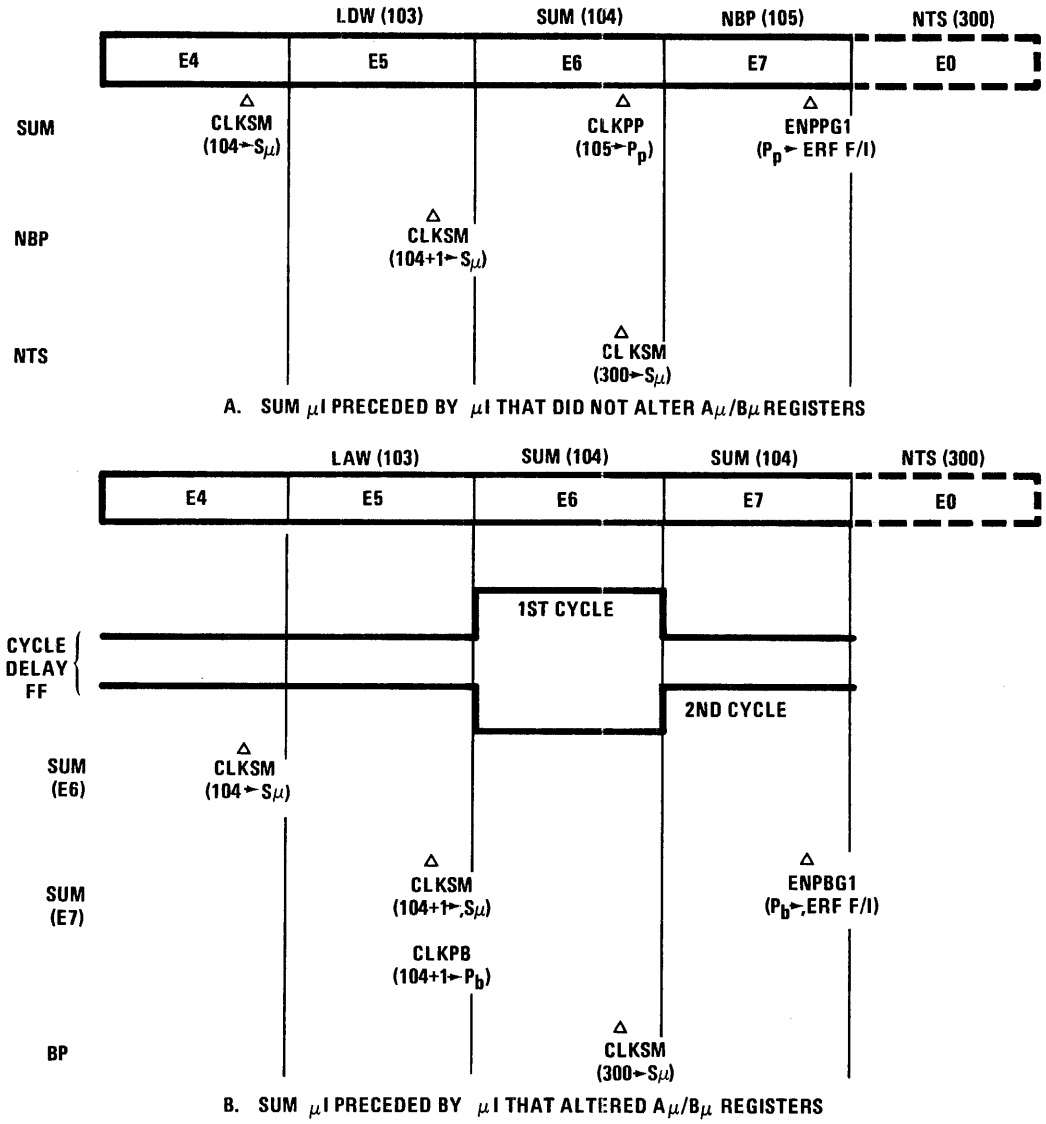


Figure 2-62. Execution of Sum μI at E6 if Cycle Delay Flip-Flop is Set

fed to the Pp register in place of the normal updated μI address from the Sμ +1 logic. A simplified diagram of the set Pp logic is shown in Figure 2-63. Although the Pp register is loaded with a 12-bit jump address (bits 4 through 15), only 5 of these 12 bits are set to a particular value corresponding to the RNI or error recovery jump address. These 5 bits are 10, 11, 12, 14, and 15, as shown in Figure 2-63. Figure 2-63 also shows the state of these five bits when generating a corresponding jump address. These five bits form the following addresses in hexadecimal form:

RNI-0000_{16}	Bounds – 0010_{16}
RNI1-0002_{16}	MS PE – 0018_{16}
RNI2-0009_{16}	CS PE – 0028_{16}

(Note than Figure 2-63 defines each address in complement form, as indicated by the $\overline{\text{SETP}}$ address designation.)

Table 2-4. Generation of Enables for Starting μI Address

BP μI	Executed at		P_p/P_b Enables		ERF Write Enables			
	E0→E6	E7	ENCLKPP	ENCLKPB	ENPPG1-0	ENPPG1-1	ENPBG1-0	ENPBG1-1
NON-BRANCH μI								
(CLR + STA + STB + AND) · X ≠ Pμ + SDW + SDB + IOR + EOR + Skips + ROM + SYNC	X	X	X	X	X	X	X	X
CI01 + CI02	X	①	X		X	X		
SUM + DSUM + CMP + CMU	E0→E5 X / E6,E7 X		X	X	X	X	X	X
BRANCH μI								
(CLR + STA + STB + AND) · X = Pμ + FNJ + FRJ + FZJ + RNI1 + RNI2	X	X	X		X	X		
JMP	X	X	X X		X	X X	X	

①Cannot execute at E7.

The three RNI routines enable a processor to obtain the address of the next MLI in the task program it is executing. Initially, the RNI0 routine provides a starting point for a processor beginning a new task. When starting the task, the processor jumps to address 0000_{16} under control of the operating system which has written address 0000_{16} into the processor's Pμ register.

A μI is located at this address which instructs the processor where it will find the address of the first MLI in the task program. During execution of this MLI, the address of the next MLI in the program will be developed as a normal part of the MLI and stored in some transient register. Upon completion of the MLI, the last μI will usually specify a jump back to 0002_{16} (RNI1 routine) to read out this next MLI address and begin its execution. For certain MLI's, however, there is not enough time during their execution to develop the next MLI address. Under these conditions, the last μI will specify a jump back to 0009_{16} (RNI2 sequence). This routine has already found the next MLI address in anticipation of such a problem, and can furnish this address immediately. This prevents any loss of time that could result if the address update had to be developed as a separate step, apart from normal MLI execution. The jump back to either 0002_{16} or 0009_{16} is implemented by the RNI1 (8,0) and RNI2 (8,1) μI's, respectively.

The remaining three addresses generated by the set P logic result from an MS parity error (PE), CS PE, or out-of-bounds condition. For each type of condition, the error routine sets an applicable bit in the condition register of the processor experiencing the error. This bit informs the operating system that an error occurred and to take appropriate corrective action. In the case of an MS parity and bounds error, the condition is recoverable in that only the processor in which the error occurred is shut down (except if the Executive processor experienced an MS parity error). For these cases, the error routines are referred to as *traps* since the error condition can be isolated to a particular processor without interferring with the rest of system operation. A CS parity error, however, is critical to overall system performance since it indicates a μI parity error. Since all processors share in the use of μI's, they can all be adversely affected. The only recourse, therefore, is to shut down the entire system.

Referring to the logic of Figure 2-63, starting addresses for the RNI1 and RNI2 routines are formed by a translation of the corresponding μI function codes. The RNI0 starting address is generated as a result of no other address being generated, which will be the case when the processor initially jumps to this address to begin a task program. The bounds error, MS PE, and CS PE trap addresses are generated upon detection of the corresponding error condition. Signal CS PE is generated when a parity error is detected in the present μI being executed. The bounds error-trap address is generated upon detection of the OUTBOUND signal from the bounds control logic in the Register Option. The bounds control logic limits the address range in MS into which each processor may read or write, thus providing data protection. If this range is exceeded by a processor accessing MS, OUTBOUND is generated which sets the Bounds Error flip-flop. The MS PE flip-flop is set for three conditions: OUTRANGE, ECC ERR, and BTYE PE. Signal OUTRANGE indicates a reference to MS has been made to a storage module that is not present in the system. Signal ECC ERROR signifies an irrecoverable error (error in two or more data bits) occurred in reading a word from MS. Since irrecoverable errors are not correctable by the ECC logic, an error routine must be performed. The BYTE PE indicates a parity error that occurred in a word read from MS when the ECC logic is not present on the system. Under these circumstances, the parity check logic of the MS interface logic is used to check for correct parity. (When present in the system, the ECC logic disables outputs from the MS interface logic.) Any of these three errors will set the MS PE flip-flops, providing OUTBOUND is not present.

PROCESSOR REGISTER FILES

The shared resources contains two sets, or files, of 16-bit addressable registers. One set, called the Basic Register File (BRF) is intimately associated with executing machine language instructions (MLI's) by the eight processors. The other set, called the Extended Register File (ERF) is used in conjunction with housekeeping, I/O, and other special-purpose applications. The BRF consists of eight subsets, one for each of the eight processors. Each subset contains 32 registers. The ERF subsets contain only those registers (up to a total of 32) that are needed by the associated processor to perform its particular functions. In addition, the ERF subsets are further subdivided into groups, depending on which registers of the ERF are made available to each processor. Group I contains two registers each for all eight processors (Pμ and F), Group II contains *common block* registers which can be accessed by all eight processors, and Group III contains registers in processors 0 through 4 but which are restricted for use by only the associated processor. Addressing of a register is accomplished by specifying three elements: (1) the register number, 0-15 (0-F, hexadecimal), (2) the processor number, 0-7, and (3) the register file set, whether basic or extended. Usually, the hardware determines the processor number, the μI determines the register set, and the MLI specifies the register number (or

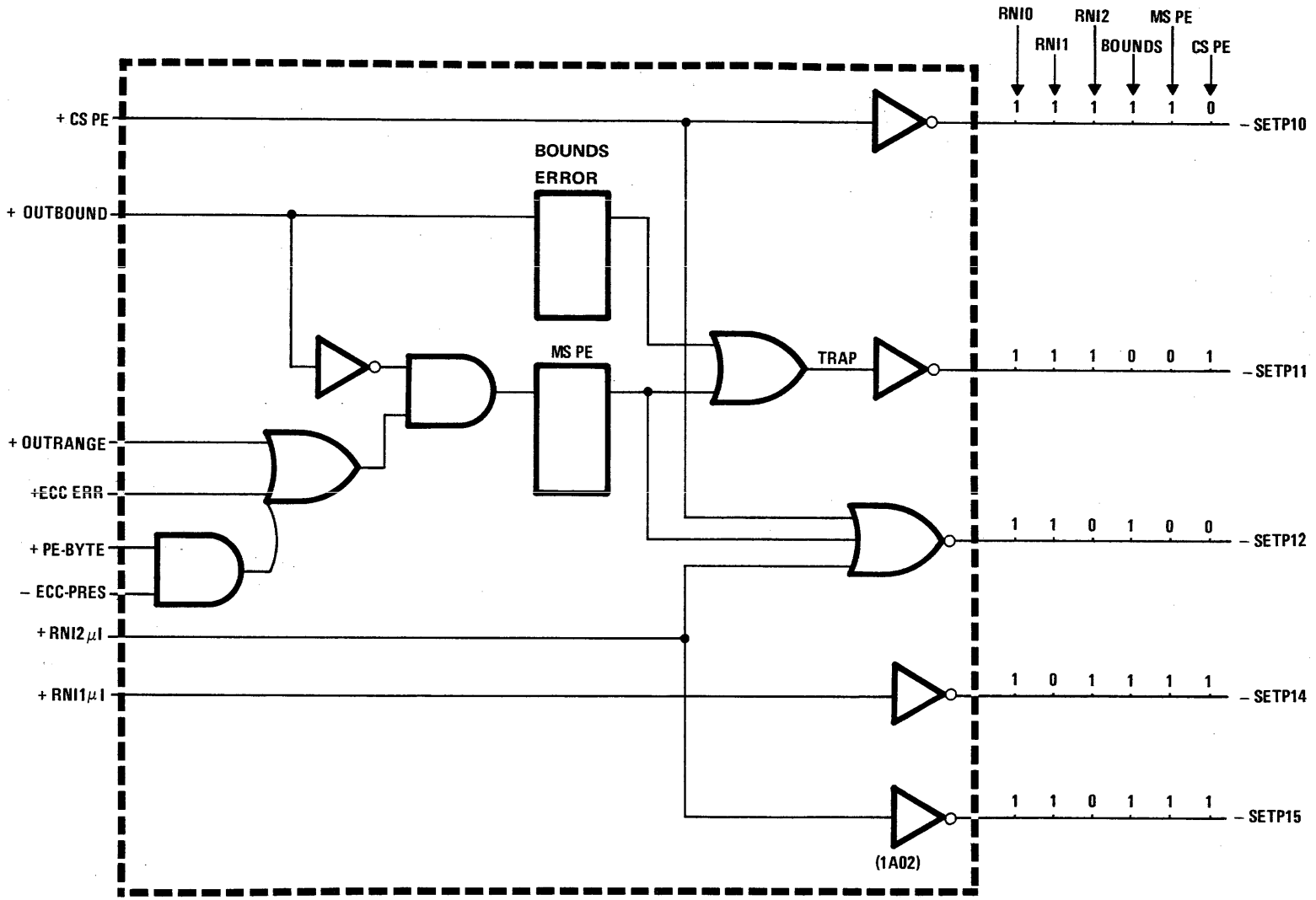


Figure 2-63. Set P Logic

numbers, since basically the computer is a two-address machine). Deviations from this general rule will be described when appropriate.

A block diagram showing how each register file is addressed is shown in Figure 2-64. Registers of the BRF are addressed by a processor number and register number. The processor number is derived from the resource allocation network, depending on acknowledging requests from processors. The register number is derived from either the MLI or μI, or a combination of both, depending on the a and b designators of the μI. Addressing of registers in the ERF depends on the register group being accessed. If Group I is accessed, the register is selected by a processor number from the priority network and a Pμ or F select signal. If Group II is accessed, the register is selected by a register number alone, since these registers can be accessed by any processor. Registers of Group III are selected by a processor number from the priority network and register number from the register select logic.

BASIC REGISTER FILE

Assignment and Functions

The Basic Register File (BRF) array is shown in Figure 2-65. As shown, the BRF is a matrix of 256 registers, 32 registers associated with each of 8 processors. The registers are made up of sixteen 256-bit LSI memory elements. Each element stores one bit of the 256 words comprising the BRF; therefore, each register word is stored 16 bits *wide*, one bit per element.

Each register or register group of the BRF is assigned a use. These assignments are made by microcode convention only and are not constrained in any way by hardware requirements. The first eight registers for each processor are general-purpose registers, addressed as 00 through 07. These registers are used for temporary storage of data involved in and resulting from

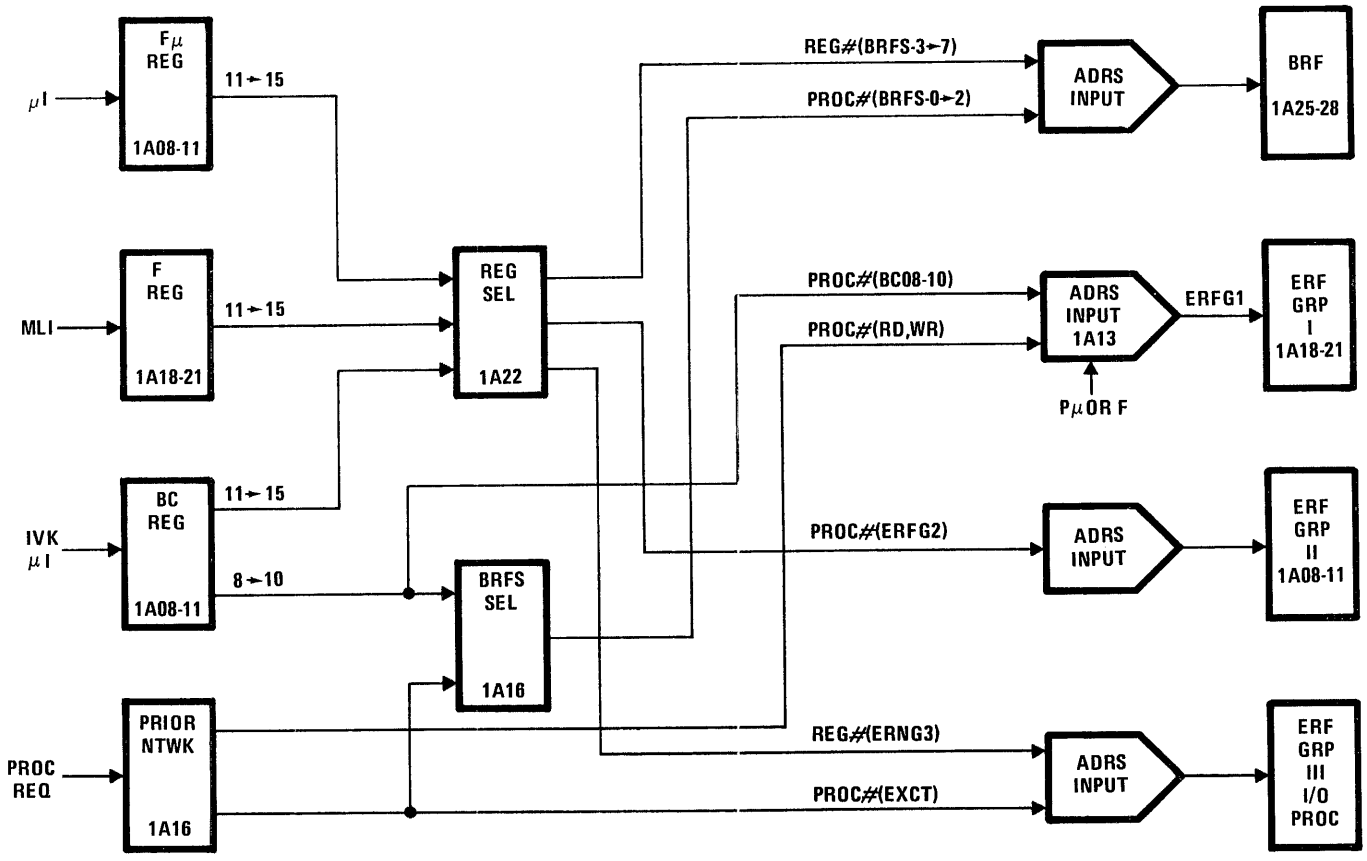


Figure 2-64. BRF and ERF Addressing

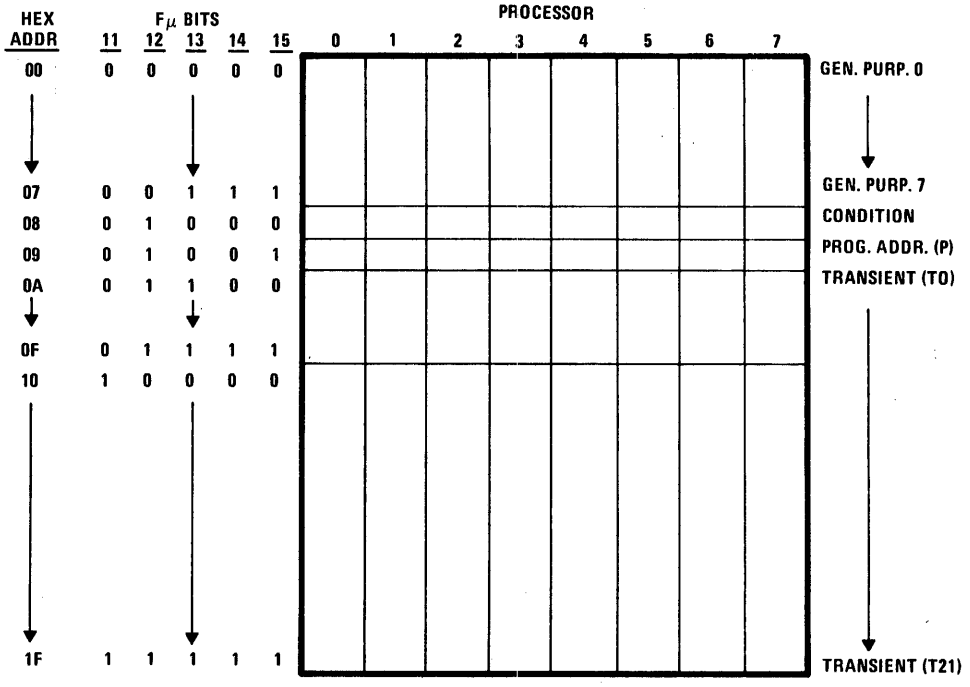
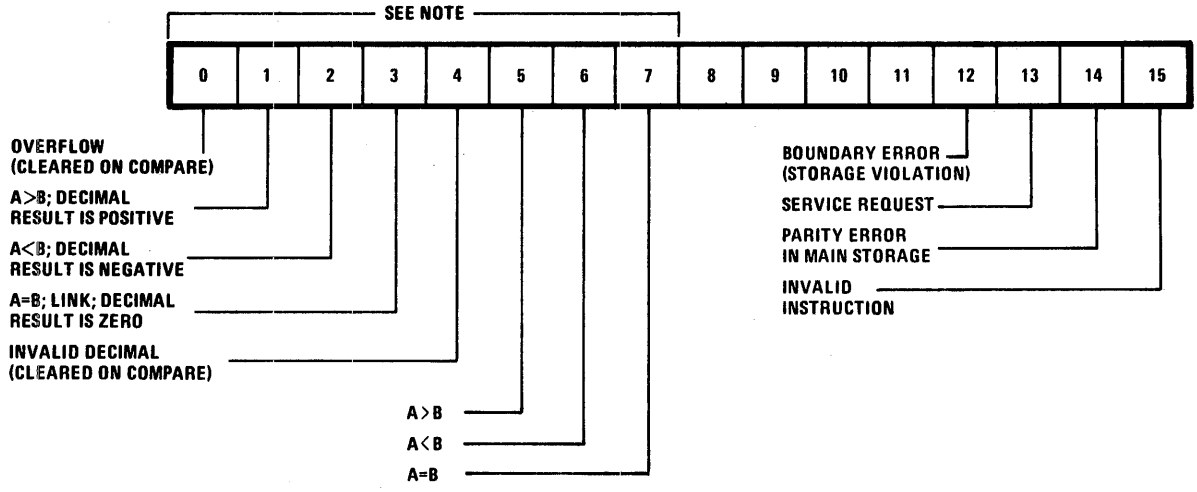


Figure 2-65. Basic Register File Array



NOTE

BIT GROUPS 0-3 AND 4-7 ARE BOTH SET AFTER ANY OF THE COMPARE INSTRUCTIONS. INTERPRETATION, WHETHER LOGICAL (MAGNITUDE ONLY) OR ARITHMETIC (SIGNED VALUES), DEPENDS UPON THE INSTRUCTION, AS FOLLOWS:

INSTRUCTION	PURPOSE	0-3	4-7
CMPX, CBY, CBYM	MAGNITUDE ONLY, BYTE-ORIENTED	LOGICAL	LOGICAL
CMPK	ARITHMETIC, PACKED DECIMAL	ARITHMETIC	ARITHMETIC
CMPF	ARITHMETIC, FLOATING POINT		
CMP, CMPD, CMPI, CMPM, CMPR, CMPT	ARITHMETIC, WORD-ORIENTED	ARITHMETIC	LOGICAL

Figure 2-66. Condition Register Bit Designations

executing MLI's (operands for an ADD instruction, for example). Registers 08 and 09 are identified as the Condition register and Program Address (P) register, respectively. The Condition register records certain conditions resulting from executing MLI's (results equal, for example). These conditions and their corresponding bit assignments in the Condition register are shown in Figure 2-66.

The P register contains the address of the MLI currently being executed. The remaining 22 registers are transient registers (0A through 0F and 10 through 1F). These registers are used for temporary storage of data involved in and resulting from executing μI's (for example, partial results accumulated while executing a machine language multiply instruction). Of these 22 registers, the last six (1A through 1F) are reserved for special use. Registers 1A through 1D are reserved for floating-point μI's. Registers 1E and 1F can be used as any of the other transient registers except when executing a Load S (LS1, LSF, LS2, or LSE) μI. If loading S from either of these two registers, the Load S μI is interpreted as a reference to the Register Option instead of to Main Storage. To the left of the register array is listed the corresponding address of each register, both in hexadecimal form and in binary form, as designated by Fμ bits 11 through 15.

Basic Register Selection

Selection of a register in the BRF is accomplished by forming an 8-bit BRFS (BRF select) address, as shown in Figure 2-67. Bits 0 through 2 of this address specify one of the eight processors, and bits 3 through 7 specify one of the 32 registers of a processor subset. The processor select bits are usually obtained from the priority network, which determines which processor will be granted the next time slice. The register select bits are obtained from several sources, depending on the condition initiating selection of a register.

Normal selection of a register is determined by the μI X-field alone, or an inclusive-OR of the μI X-field with either the MLI R_1 or R_2 fields depending on the values of the μI a and b designators (bits FM2-006 and FM2-007, respectively). The combinations of the various μI and MLI fields for selecting registers is shown in Figure 2-67 and summarized below:

a·b = 0·0 — register selected by μI X-field only (bits FM2-011 through FM2-015)

a·b = 1·0 — register selected by inclusive-OR of μI X-field and MLI R_1-field (bits FR-009 through FR-011)

a·b = 0·1 — register selected by inclusive-OR of μI X-field and MLI R_2-field (bits FR-013 through FR-015)

a·b = 1·1 — BRFS address inhibited and a register of the ERF is selected by the ERF select logic as discussed in the paragraph titled Extended Register Selection.

A BRF register can also be selected by an IVK μI. The IVK μI selects any of the 32 registers by the contents of the Boundary Crossing (BC) register instead of the Fμ and F register contents. Execution of the μI selects a register by means of bits BC-007 through BC-015.

A simplified diagram of the BRF select logic is shown in Figure 2-68. Each of the 8 BRFS lines connect to all 16 elements of the BRF in parallel to enable all 16 bits of a particular register. As shown, BRFS-0 through BRFS-2 come from the resource allocation logic to select a particular processor. These three processor select bits, of which BRFS-0 is shown in detail, are selected by the EXEC bits from the Execute register during normal operation or bits 8, 9, and 10 of the BC register during an IVK μI. In either case, the three input bits represent the processor number in BCD form. Bits BRFS-3 through BRFS-7 are used to select a register of a processor BRF. For illustrative purposes, logic for generating bits BRFS-3 and BRFS-5 is shown in detail. During normal operation, bit BRFS-3 is generated unconditionally by FM2-011 and bit BRFS-5 by FM1-013, in combination with FR-009 or FR-013 depending on the presence of enable FM2-006 (a-designator) or FM2-007 (b-designator). During an IVK μI, BRFS-3 and BRFS-5 are generated by BC-011 and BC-013, respectively, when enabled by INVOKE.

Writing into a selected register of the BRF is enabled by the ENBRFW-0 and ENBRFW-1 signals. As shown in Figure 2-68, ENBRFW-0 is used to enable writing into bits 0 through 7 of a register, and ENBRFW-1 into bits 8 through 15. Both enables are generated for basically the same conditions, those being translation of a Register File Write (1,X; 2,X; 4,X; or 8,X) μI in F. The difference between the two enables is that only ENBRFW-0 is generated for the CMP and CMU (2,2 and 2,3) μI's, since only bits 0 through 7 of a register can be written into by these two μI's. The write strobe pulse is furnished via BRFWRITE, which occurs at t50 of every minor cycle.

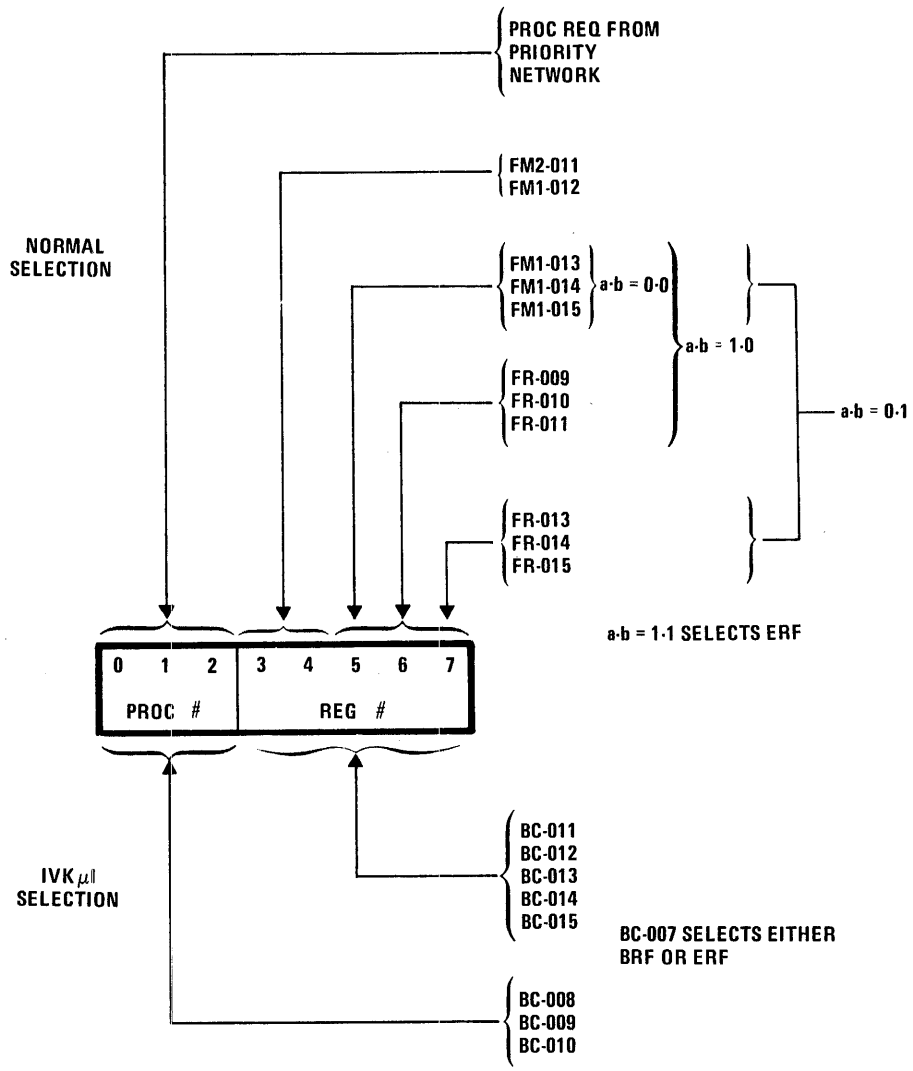


Figure 2-67. BRF Addressing Methods

When the write enables and write strobe are present, the data present on the DATA IN lines from the ALU fan-out logic is written into the selected register.

EXTENDED REGISTER FILE

Assignment and Functions

The Extended Register File (ERF) array is shown in Figure 2-69. As shown, the ERF consists of three groups, according to the number of registers associated with each processor and their use.

Group I Registers

Group I consists of two registers for each processor: the Function (F) register and the μI Address (Pμ) register. The F register is 16 bits in length and contains the MLI currently being executed by the associated processor. The Pμ register is 18 bits long and contains the address of the first μI to be executed during the next time slice assigned to the active processor, plus four status bits. The format of this register is shown in Figure 2-70. As shown, the lower 14 bit positions (bits 02 through 15) contain the μI address, and the upper four bit positions contain the four status bits. Functionally, the Pμ register is considered to be an 18-bit register; physically, however, it consists of a

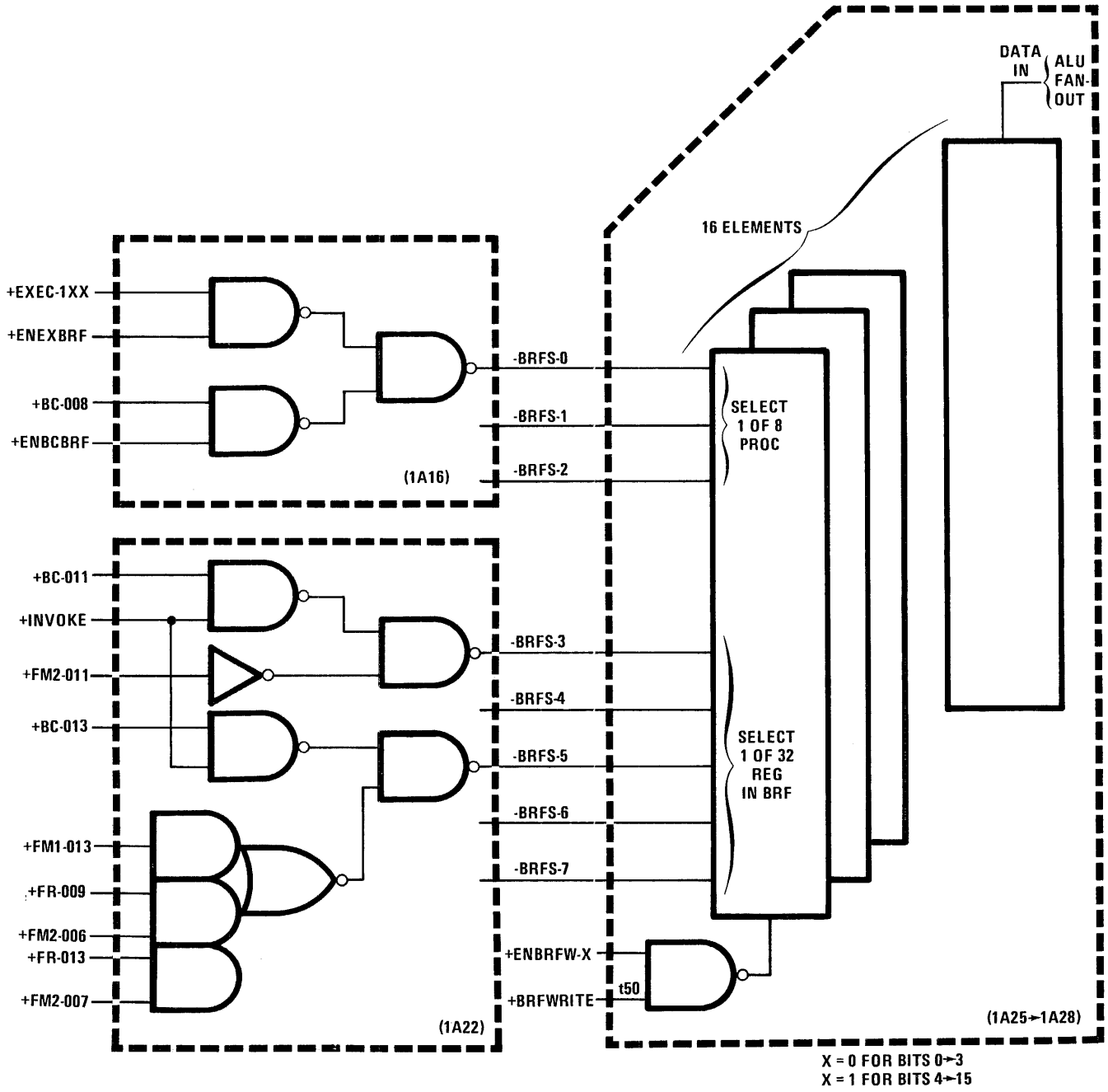


Figure 2-68. BRF Select Logic

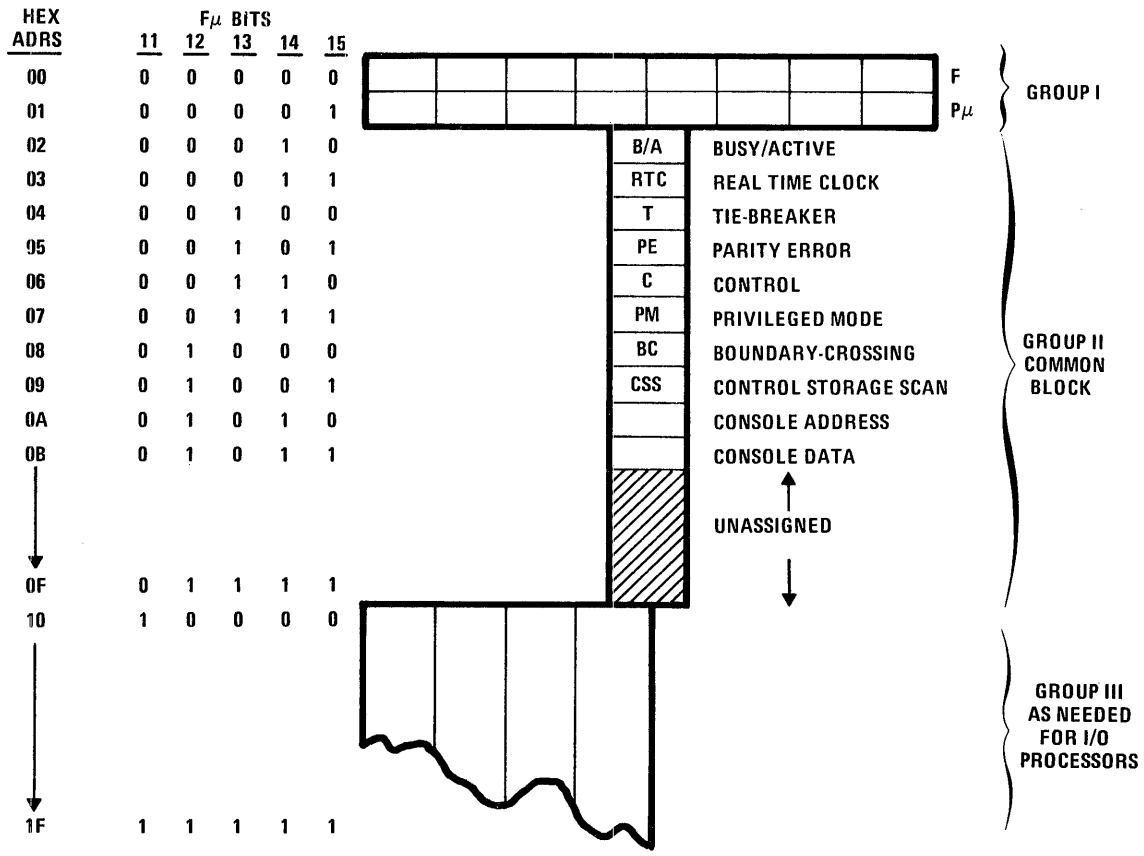


Figure 2-69. Extended Register File Structure

16-bit basic register supplemented by a 2-bit extension register. Since two extra bit positions are available in the 16-bit register, the Overflow (O) and Link (L) status bits are stored in these positions. The CS parity error (E) and Skip (S) status bits, however, are located in the 2-bit register. The E and S bits are hardware-controlled only and are inaccessible for μI control, except by μI's which write into Pμ when operating in the boundary-crossing mode. Although physically separate, the 2-bit register is addressed, read, and written by the same lines that control the 16-bit register.

Conceptually, these two register sets belong to the BRF since these two registers are associated with each of the eight processors. However, limitations in the ability to address the registers as part of the BRF plus the *look-ahead* capability desired by these registers, described in the paragraph titled Access Capabilities and Limitations, make it necessary to include the F and Pμ registers as part of the ERF. The two registers are numbered 00 and 01.

Group II Registers

Group II consists of 14 registers, numbered 02 through 0F. At present, only registers 02 through 0B are assigned specific uses. These 14 registers form only one subset that are shared by all eight processors. Ordinarily, a processor cannot gain access to a register in another processor's register set because the processor number is part of the register addressing scheme. This processor number is not translated for the Group II registers, however. Therefore, any processor may gain access to these registers. For this reason, the registers of Group II are referred to as the *common block* registers. Each of the Group II registers is discussed below:

Busy/Active (B/A) Register — The B/A register indicates which processors are presently engaged in processing a task (active condition) and which processors that are active require a time slice to execute the next portion of their tasks (busy condition). The difference between the two conditions is that the busy condition will go through many on-off cycles during execution of a task, whereas the active condition will generally remain set until that task has been completed. The B/A register is made up of a Busy flip-flop and an Active flip-flop associated with each processor. These 16 flip-flops make up the B/A register as shown in Figure 2-71, where the busy conditions are represented as bits 0 through 7 and the active conditions represented as bits 8 through 15. (More details of the B/A register are contained in the paragraph titled Busy/Active Register.)

Real Time Clock (RTC) Register — The RTC register is a 16-bit register/counter that is incremented every 1.6384 milliseconds. The contents of this register are used by processor 4 (Exec) for software timing purposes, such as updating the time-of-day clock and initiating time-out operations. A simplified diagram of the RTC register is shown in Figure 2-72. The register is made up of four 4-bit binary counters, where each higher-order counter is incremented by a carry from the preceding lower-order counter. Incrementing of the lowest-order counter (bits 12 through 15) is initiated by $\overline{\text{RTCASYNC}}$ from the RTC pulse generator. This signal is generated every 1.6384 milliseconds.

Because the 1.6384-millisecond $\overline{\text{RTCASYNC}}$ signal is developed asynchronously from the rest of the CPU timing, it must be synchronized to the E pulse timing to correctly update the RTC register. This is done by sending $\overline{\text{RTCASYNC}}$ through a pulse catcher network composed of flip-flops No. 1 and No. 2. The output of No. 1 goes high upon receipt of the low $\overline{\text{RTCASYNC}}$ signal and causes flip-flop No. 2 to set at E150. Since $\overline{\text{RTCASYNC}}$ is asynchronous with the E-timing pulses, however, only a sliver of $\overline{\text{RTCASYNC}}$ may be present at E050 time with the result that the output of No. 2 may be indeterminate for a couple of E-times. By E7, however, the output has stabilized sufficiently that it can be gated as $\overline{\text{RTC-G3CI}}$ to the lowest-order stage of the RTC register. Incrementing the register at E7 avoids problems in reading this register at E0 through E6.

A carry from the bit 12 through 15 counter generates $\overline{\text{RTC-G3C0}}$, which initiates incrementation of the bits 8 through 11 counter. The remaining two counters are incremented in a similar fashion, until a final count of 1.8 minutes is reached (1.6384 msec x 2^{16}). Because the counter overflows at this point (RTC is generated), the processor must read the register at least this often if the timing interval information is to be meaningful. Upon reaching overflow of the bits 0 through 3 counter, the register clears itself and begins counting again from zero.

Tie Breaker (T) Register — The Tie Breaker register is a 16-bit register used for recording the status information pertaining to system tables in Main Storage (MS). System programs use the tables to communicate with one another. Before using them, however, each program must check the Tie Breaker register to determine if the desired table of MS is being used by another program. This is done by assigning each bit of the T register to a particular System table (as differentiated from a User table) stored in MS. Before calling up a System table, a processor must examine the bit of the Tie Breaker register representing that table to see if it is already being used by another processor. This precaution prevents one processor from reading tabular material that another processor is updating, or vice versa. A processor sets the associated T-bit before starting to use a table via the Test and Set Tie-Breaker Register MLI (11), and clears the bit when finished via the Clear Tie Breaker Register MLI (12).

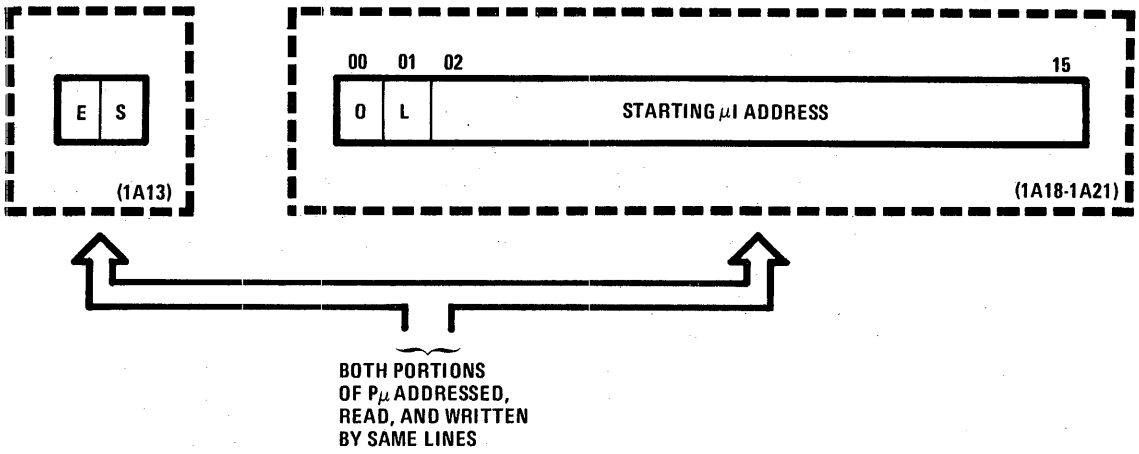


Figure 2-70. P_{μ} Register

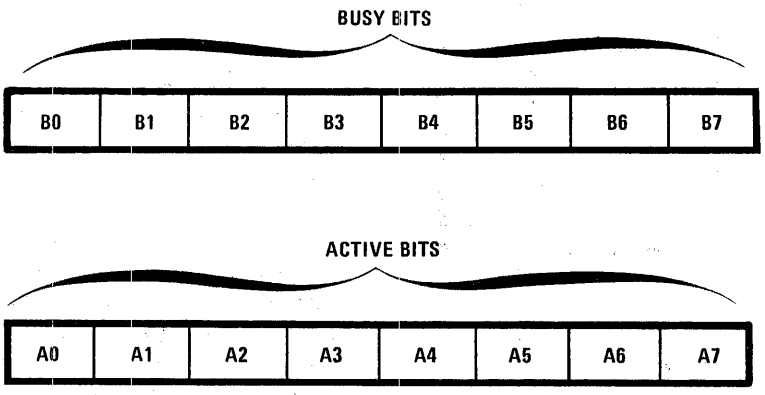


Figure 2-71. Busy/Active Register

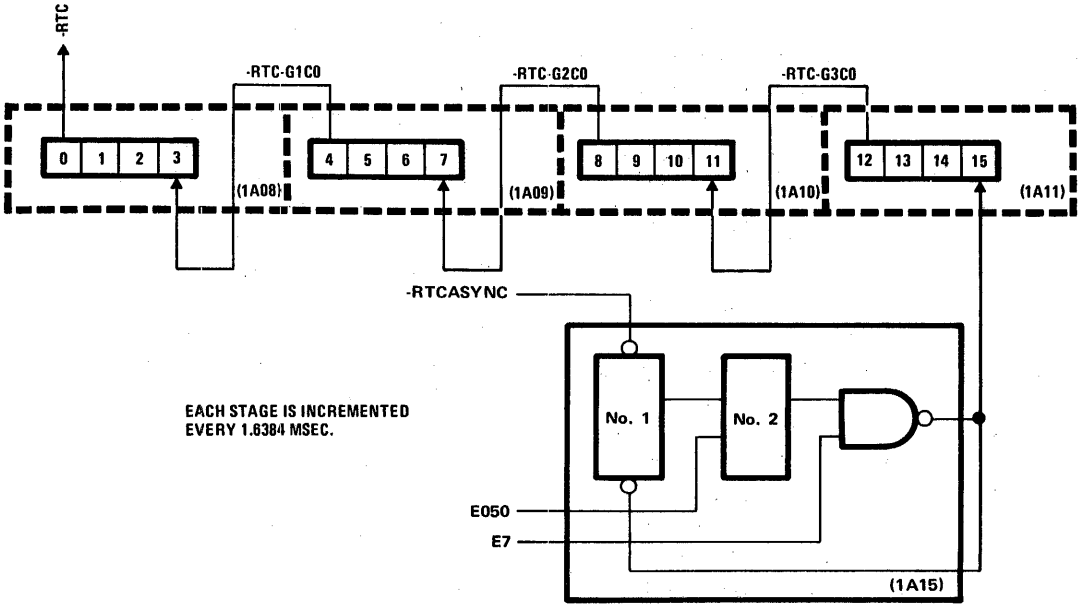


Figure 2-72. Real Time Clock Register

Inputs to the T register are from the ALU fan-out logic under control of these two MLI's. Bit assignments for the various tables are generated as part of the *Operating System* (control program).

Parity Error Address (PE) Register – The PE register contains the MS address at which the last parity error occurred. This register is constantly fed with MS addresses from the S register, as shown in Figure 2-73. However, no address is gated into the register until $\overline{\text{CLKPE}}$ is activated. This signal is activated upon detection of a parity error, and is timing so that the address it gates into the PE register is where the parity error occurred.

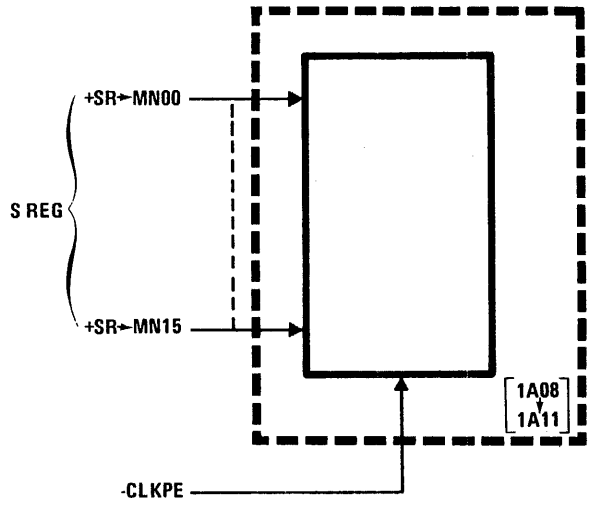


Figure 2-73. Parity Error Register

Control Register – The Control register stores control bits for each processor that define the type of priority assigned to each processor and whether it is to run in the consecutive-cycle mode. A simplified diagram of the register is shown in Figure 2-74. The left-most eight stages store Enable Priority (EP) and Invoke Priority (IP) status bits associated with the priority logic of processors 0 through 3. The right-most eight stages store Consecutive Cycle Enable (CCE) status bits associated with all eight processors. The register receives its input from the ALU fan-out logic, and is set and cleared by the Set/Reset Control Register MLI.

Privileged Mode (PM) Register – The PM register is a 16-bit register, of which only the right-most eight bits (bit positions 8 through 15) are used at present. These eight bits allow the Executive processor to set any of the eight processors to a privileged state. This is done by setting a privilege mode bit in the register corresponding to each processor, as shown in Figure 2-75. When set to the privileged state, the processor is able to execute privileged MLI's.

Boundary Crossing (BC) Register – The BC register holds the processor and register number used by a processor or the System Control Panel to select a register in another processor's register file. This is done during execution of an IVK μI. The format of the Boundary Crossing register is shown in Figure 2-76. Although 16 bits in length, only the lower 9 bits of the Boundary Crossing register are used for boundary crossing monitor purposes. Typical reasons for crossing boundaries by processor 4 are to:

1. Set P
2. Set, clear, or examine Condition register
3. Set, clear, or examine general-purpose registers
4. Set, clear, or examine transient registers
5. Set, clear, or examine Pμ and F registers
6. Test registers (diagnostic routines)

Control Storage (CS) Scan Register – The CS Scan register is used to check longitudinal parity on 256-word pages in CS. This is done by performing an exclusive-OR on all words of a page. If the contents of the register yields all 1's after the last word of a page has been checked, this indicates that the page was loaded correctly. The CS Scan register receives its input from 14 bits of each word stored in CS (bit positions 9 and 10 are not used), as shown in Figure 2-77. The register output feeds the FRJ decode logic checking the contents for all "1's". (More details of the CS scan operation are discussed in the paragraph titled CS Scan/Read.)

Console Address Register – The Console Address register is used in conjunction with the row of 20 address pushbutton/indicators on the System Control Panel to provide entry and display of address-related information. See the MEMOREX 7300 Processing Unit Maintenance manual for a discussion of the Panel and how the Console Address register is used during maintenance operations.

Console Data Register – The Console Data register is used in conjunction with the row of 20 data pushbutton/indicators on the System Control Panel to provide entry and display of data-related information. See the MEMOREX 7300 Processing Unit Maintenance manual for a discussion of the Panel, and how the data register is used during maintenance operations.

EP=ENABLE PRIORITY
IP=INVOKE PRIORITY
CCE=CONSECUTIVE CYCLE ENABLE

Figure 2-74. Control Register

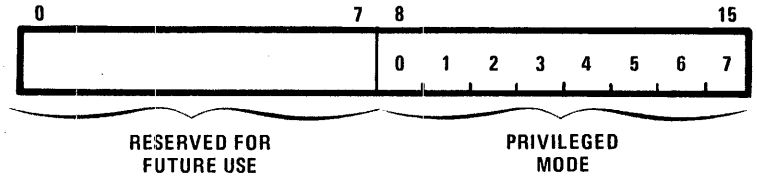


Figure 2-75. Privileged Mode Register

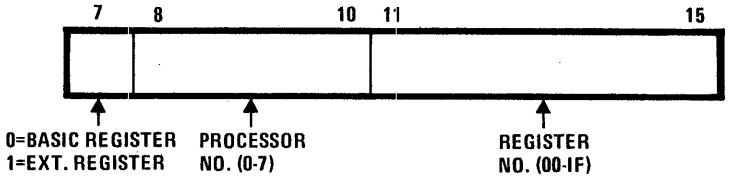


Figure 2-76. Boundary Crossing Register

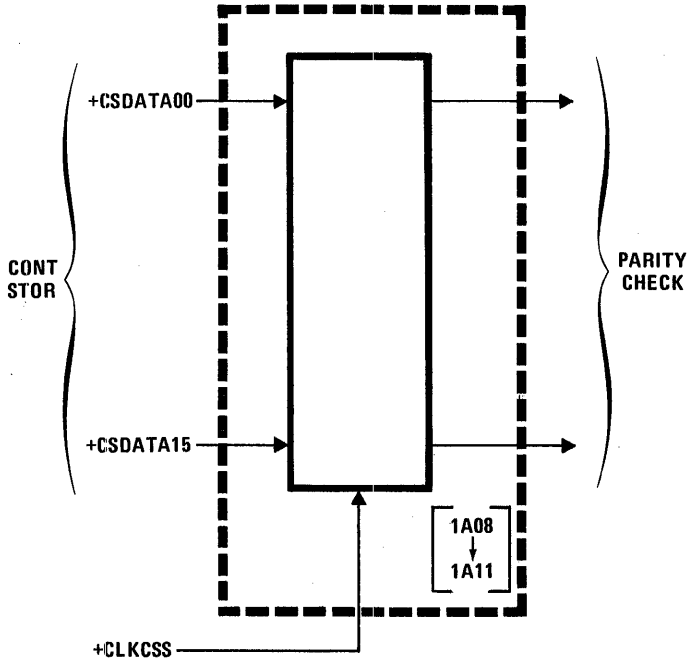


Figure 2-77. CS Scan Register

Group III Registers

There are four subsets of Group III registers, one for each of processors 0 through 3. Each subset has provision for addressing up to 16 registers. The Group III registers are I/O-oriented registers, addressable only by the associated I/O processor, and not part of the shared resources. Because of the restrictions on addressing these Group III registers, they are designated as *dedicated extended registers*. A description of these registers, including their use, is contained in the applicable section of Volume 3 of this manual.

Extended Register Selection

Selection of a particular extended register of the ERF depends in which group the register is located. For each group, specific register select signals are generated as described below.

Group I – The Pμ and F registers of Group I are selected by four bits: three bits which define the processor number and one bit which defines either the Pμ or F registers of that processor's ERF. This selection is shown in Figure 2-78. The processor number is defined by bits $\overline{\text{ERFG1S0}}$, $\overline{\text{ERFG1S1}}$, and $\overline{\text{ERFG1S2}}$, which comprise a three-bit BCD. These bits are normally generated by the priority logic via corresponding read and write bits after deciding which processor gets the next time slice. Separate read bits (READ-XXX) and write bits (WRITEXXX) must be generated by the priority logic since reading the Pμ register for the next processor to be granted a time slice occurs before writing into the F register of the current processor. Therefore, the current processor number defined by the WRITEXXX bits must be present along with the next processor number defined by READ-XXX until the end of the present processor's major cycle. Enables ENRD-ERF and ENWR-ERF define the times that the ERFG1 address bits are generated. Enable ENRD-ERF occurs at R0 and R1 times of the next major cycle (E6 and E7 times of the present major cycle) to read out the starting μI address from Pμ and the associated MLI from F for the next processor to be honored. Enable ENWR-ERF occurs at W0 and W1 times of the present major cycle to store the starting address of the first μI to be executed during the next assigned time slice, and to store the associated MLI into the Fμ register. Execution of an IVK (F,1,1,) μI substitutes a processor number contained in bits 8, 9, and 10 of the Boundary Crossing (BC) register for that originally supplied by the priority network. This is shown as inputs BC-008, BC-009, and BC-010 to the ERF Group I address logic, which generate the ERFG1 address bits in place of the READ-XXX bits from the priority logic. Signal ENBC-ERF enables this processor number from the Boundary Crossing register at E2 through E5 times when the Invoke flip-flop is set. During a master clear condition, the MC-2 signal is ANDed in succession with counts E1256, E2345, and E4567 from the timing chain gray-code counter. These counters are generated in sequence to generate processor numbers 0 through 7 in a cyclic fashion. The effect is to clear out the Pμ register associated with all eight processors. This operation causes an address of 0000_{16} to be written into all Pμ registers so that each processor routine will begin with an RNI0 sequence.

Besides selecting the processor via the ERFG1 bits, it is also necessary to select either the Pμ or F register of the selected processor, and to enable either a read or write of the selected register. Selection of either Pμ or F is provided by SELFH/PL, according to its state:

SELFH/PL=high – selects F register

SELFH/PL=low – selects Pμ register

Reading or writing the selected register is provided by EF1RH/WL, according to its state:

EF1RH/WL=high – read operation

EF1RH/WL=low – write operation

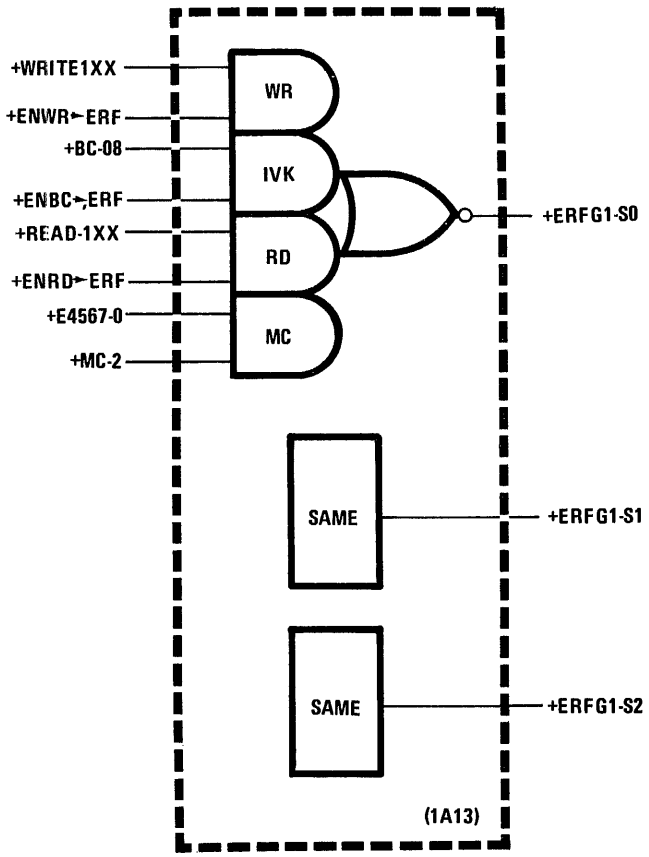


Figure 2-78. ERF Group I Select Logic

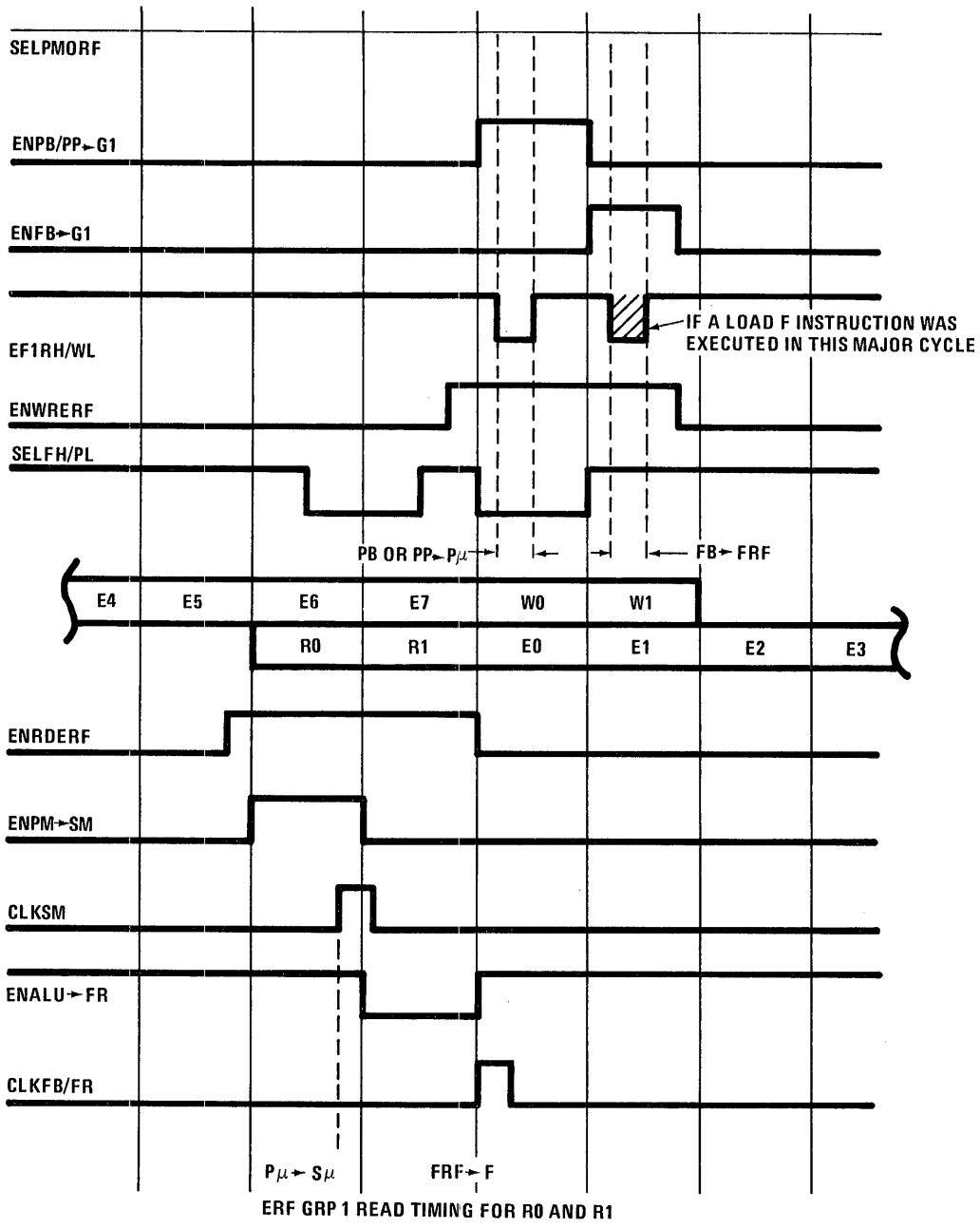


Figure 2-79. ERF Group I Read and Write Timing

These signals are used together to read or write the selected register, as shown in Figure 2-79. Reading Pμ and F occurs during E6 and E7 of the present time slice, in preparation for the processor to run during the next time slice. During these two minor cycles, EF1RH/WL is high to enable the read operation and SELFH/PL is either low or high to select either Pμ or F as the register to read from. Writing into Pμ and F occurs during W0 and W1 times of the present time slice. During these two minor cycles, EF1RH/WL is low to enable the write operation and SELFH/PL is again either low or high to select either the Pμ or F register.

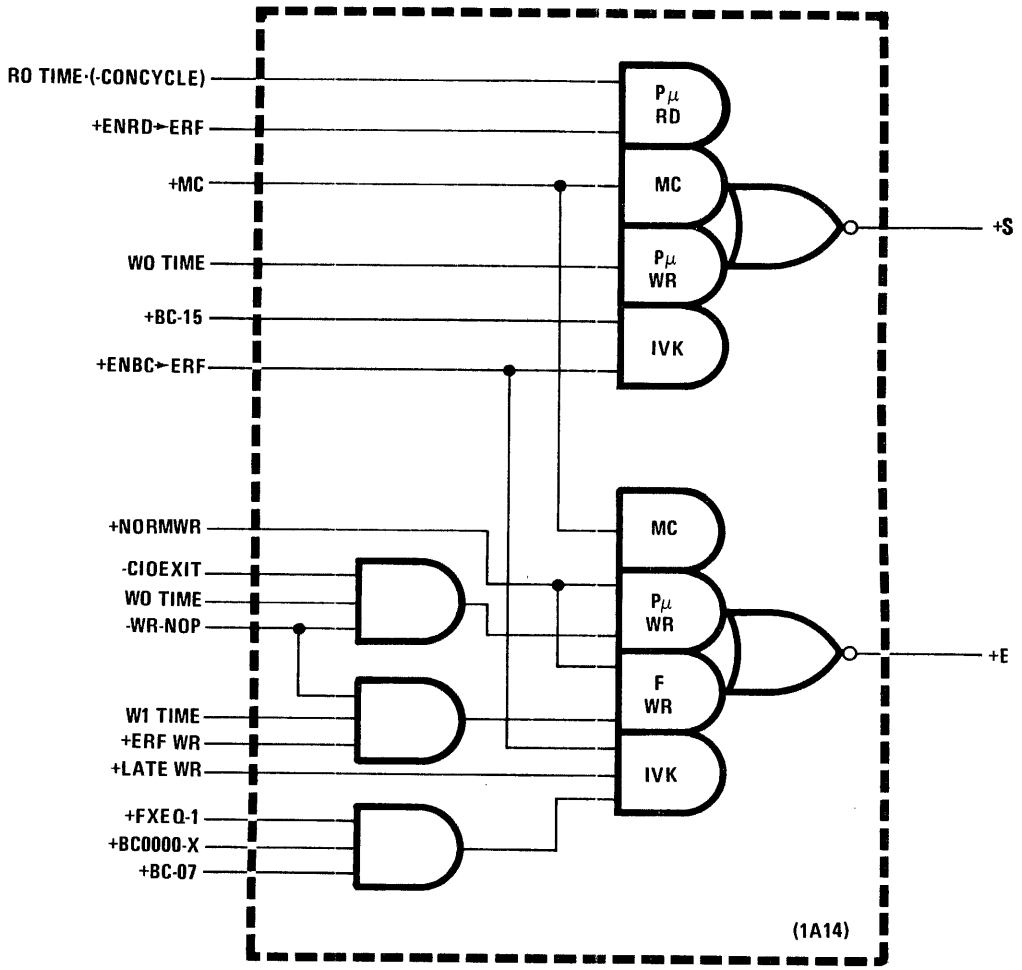


Figure 2-80. Generation of SELFH/PL and EF1RH/WL

Generation of the SELFH/PL and EF1RH/WL signals is shown in Figure 2-80. Both SELFH/PL and EF1RH/WL are generated for a master clear condition to clear out the Pμ registers (write an address of 0000_8) of all eight processors. Signal SELFH/PL is also generated at the times shown in Figure 2-79 to read data from and write data into the Pμ register as part of the normal housekeeping operations associated with each time slice. The IVK μI generates SELFH/PL to select a Pμ register specified by the contents of the BC register. This signal is generated by BC-15, which specifies register 0001 (Pμ register) of the ERF, and ENBC-ERF, which enables data to be read from the BC register during an IVK μ I.

Besides being generated during a master clear condition, signal EF1RH/WL is driven low for three other operations: write Pμ, write F, and IVK μI. During normal write Pμ and write F operations, EF1RH/WL is generated by NORMWR and an ANDed combination of enabling conditions. Signal NORMWR is generated during the middle half of every minor cycle and is used as the basic write signal. For write Pμ operations, NORMWR is enabled at W0 time providing neither $\overline{\text{CIOEXIT}}$ or $\overline{\text{WR-NOP}}$ is low. If either $\overline{\text{CIOEXIT}}$ is low (indicating the compare condition of a CIO μI to exit from the data transfer loop has been met) or $\overline{\text{WR-NOP}}$ is low (indicating that a NOP μI is being executed), the Pμ Write operation is inhibited (Pμ is not updated by Pp) by driving) EF1RH/WL high. If either condition is not met, however, EF1RH/WL remains low. For write F operations, NORMWR is enabled at W1 time except if NOP μI is executed. During write Pμ and write F operations initiated by the IVK μI, E1FRH/WL is generated by LATWR and enabling conditions. Signal LATEWR is generated for the same period as NORMWR (about 50 nanoseconds) but about 15 nanoseconds later in the minor cycle. This additional delay gives the IVK μI sufficient time to perform the required translation necessary to implement the μI. Since this μI selects only one register at a time, the EF1RH/WL signal is generated only once. Enabling conditions for the IVK μI are FXEQ-1, which specifies the IVK K μI itself; BC-0000X, which specifies either register 0000 (F) or 0001 (Pμ); and BC-07, which specifies the ERF.

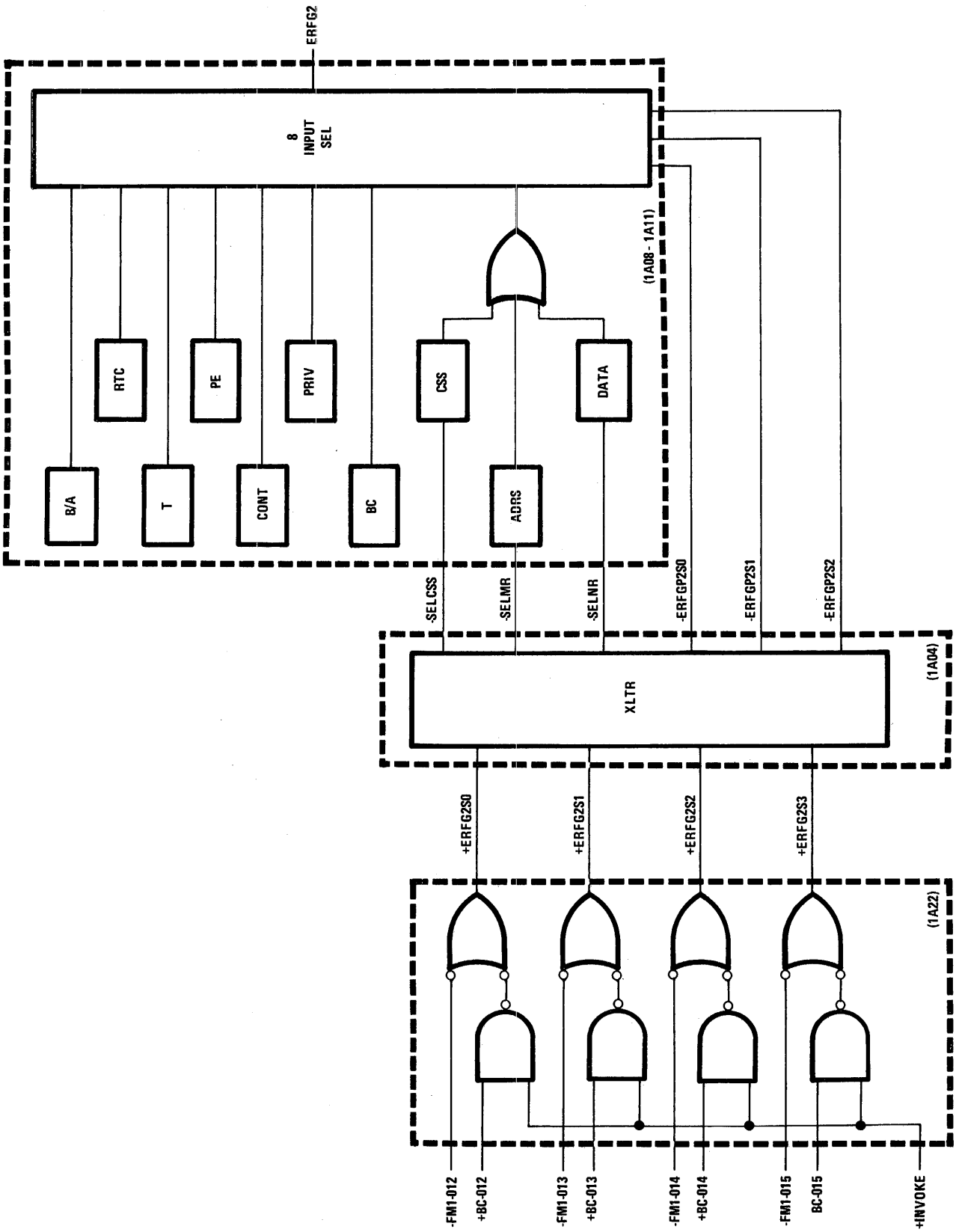


Figure 2-81. ERF Group II Address Format, Read Operation

Group II – Selection of a Group II register for reading or writing depends on whether the register is selected via μI control or hardware control. All Group II registers may be read via μI control, and all but the PE, RTC, and CS scan registers may be written via μI control. Logic for selecting a Group II register via μI control is shown in Figure 2-81. During a normal read operation, with the a and b designators set, each Group II register is selected by means of bits 12 through 15 of the μI X-field (Fμ bits 12 through 15). If the read operation is under control of an IVK μI, the register address is obtained from bits 12 through 15 of the Boundary Crossing register. In either case, the 4-bit address generates four select signals, ERFG2S0 through ERFG2S3 These select signals are fed to a translator where they undergo further decoding to generate select bits $\overline{\text{ERFGP2S0}}$ through $\overline{\text{ERFGP2S2}}$. These three select bits are routed to a one-of-eight selector. The selector is fed with eight register inputs and enables reading the selected register input. Note that while there are only 8 inputs to this selector, there are 10 registers of Group II that need to be selected. Selection of the two extra registers is accomplished by ORing the outputs of the CSS, Console Address, and Console Data registers and feeding the result into the selector as one register input. Each of these registers, then, is selected by a corresponding select bit: $\overline{\text{SELCSS}}$, $\overline{\text{SELMR}}$, or $\overline{\text{SELNR}}$, which are generated by select signals ERFG2S2 and ERFG2S3. The correspondence of $\overline{\text{ERFGP2}}$ and $\overline{\text{SEL}}$ select bits to the Group II register addresses is shown in Table 2-5. (Note that the select bits are defined in complement form while the address bits are defined in true form.)

Selection of a Group II register for a write operation is basically done by selecting the appropriate clock or clock enable signal for a particular register. For the Busy/Active, Tie Breaker Control, Privileged Mode, Boundary Crossing, CS Scan, Console Address, and Console Data registers, the clock or clock enable signal is generated by the four select signals, ERFG2S0 through ERFG2S3. This method of register selection is shown in Figure 2-82 for selecting the Tie-Breaker (T) and Boundary Crossing (BC) registers as examples of Group II register selection. Of the four select signals, ERFG2S1 through ERFG2S3 are routed to a BCD/one-of-eight decoder. The two decoder outputs are sent to one side of corresponding AND gates used to generate the clock enable signal for each register. The other side of the AND gates are fed with ERFG2S0. This signal is ANDed with ENERG2WR, indicating a Group II write operation, and SELERFG3, indicating that the ERFG2S0 through ERFG2S3 select bits are selecting one of the lower 16 registers of the ERF. (As shown in Figure 2-69, the upper 16 register addresses are reserved for Group III registers.) The result is fed in true form to the AND gate used to clock the BC register, since its address is greater than 7 (08_{16}), and in complement form to the AND gate used to clock the T register since its address is 7 or less (04_{16}). The AND gate outputs are enabled with CLKERFG2 (generated at TX80 time) to generate register clock signals CLKTB and CLKBC. During a master clear operation, "0's" are written into all Group III registers (except the PE and RTC registers). This is accomplished by satisfying both sides of each AND gate with MC, which essentially clocks each register with no data ("0's") present on the register input lines. The result is to clear each register.

Table 2-5. ERF Group II Read Select Bits

Reg	Hex ADRS	Fμ Reg Bits 12	13	14	15	ERFGP2 Bits $\overline{S2}$	$\overline{S1}$	$\overline{S0}$	SEL Bits $\overline{CSS}$	$\overline{MR}$	$\overline{NR}$
B/A	02	0	0	1	0	1	1	1	1	1	1
RTC	03	0	0	1	1	1	1	0	1	1	1
T	04	0	1	0	0	1	0	1	1	1	1
PE	05	0	1	0	1	1	0	0	1	1	1
CONT	06	0	1	1	0	0	1	1	1	1	1
PRIV	07	0	1	1	1	0	1	0	1	1	1
BC	08	1	0	0	0	0	0	1	1	1	1
CSS	09	1	0	0	1	0	0	0	0	1	1
ADRS	0A	1	0	1	0	0	0	0	1	0	1
DATA	0B	1	0	1	1	0	0	0	1	1	0

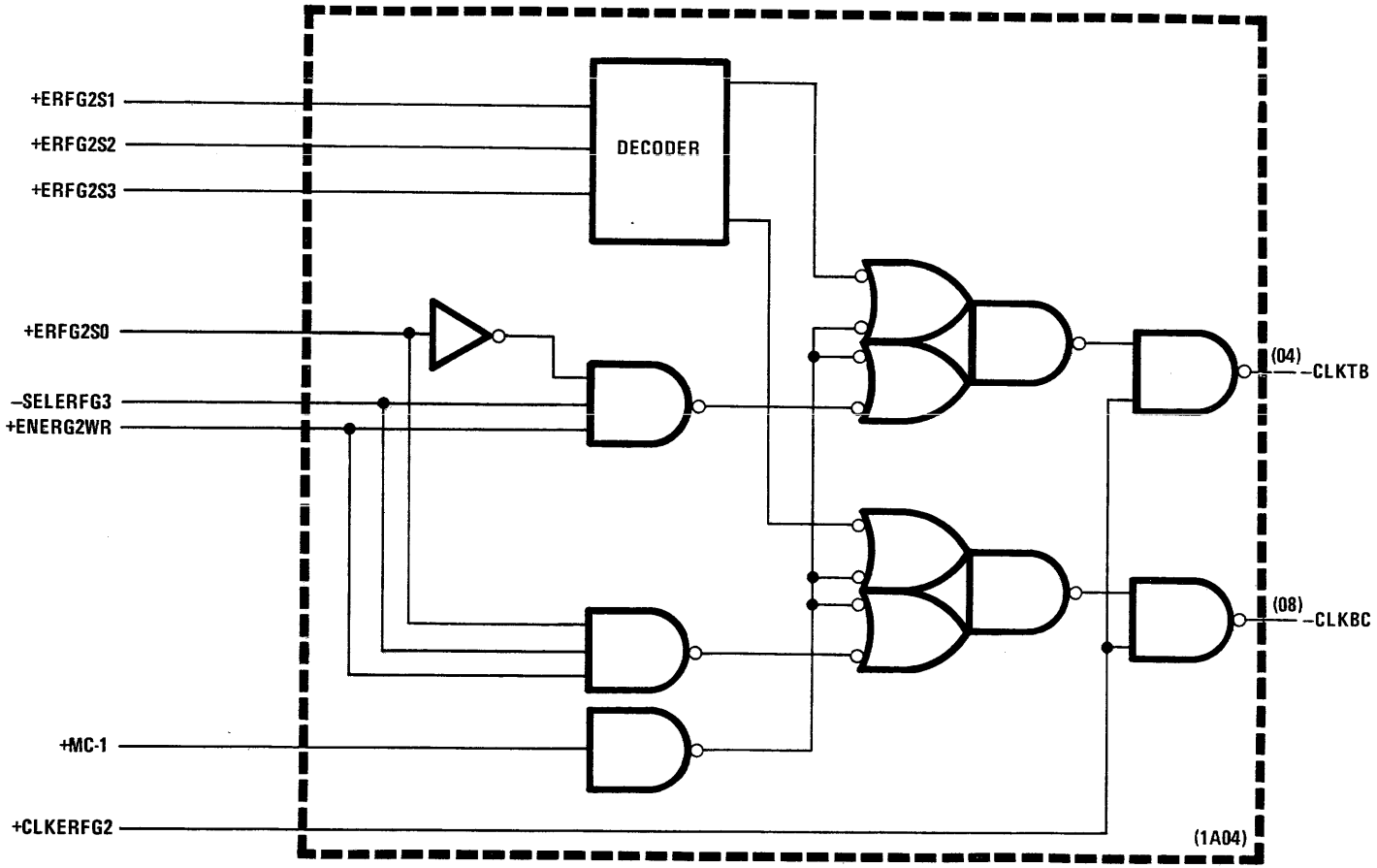


Figure 2-82. ERF Group II Address Format, Write Operation

Selection of clock signals for the Busy/Active, Console Address, and Console Data registers differs slightly from those for the T and BC registers in that the select logic generates a clock enable signal only. The reason is because the flip-flops comprising these registers require a positive-going signal to clock them instead of the negative-going signal required by the other registers. The clock signals for these flip-flops, therefore, are generated by ANDing them with TX80 in a non-inverting fashion to furnish the required polarity. The AND gates are located on the same module with the registers.

As mentioned previously, writing the PE, RTC, and CS scan registers is accomplished under hardware control only. The PE register is written with the MS address at which a PE occurred during an MS read operation, as discussed in the paragraph titled MS Read. The RTC register is continuously written with a clock pulse derived from the RTC pulse generator, as described in the paragraph titled Real Time Clock Pulse Generator. The CS scan register is written with cumulative longitudinal check data during a CS scan operation, as described in the paragraph titled Scan/Read.

Group III — Selection of Group III registers is performed in a manner similar to that for the BRF registers: generation of a processor select number and a register address. Like the BRF registers, each I/O processor can only access registers of its assigned file. A simplified diagram showing selection of Group III registers is shown in Figure 2-83. The processor number is specified by the EXCT signal from the resource allocation network. Each I/O processor is selected by a unique EXCT signal. The Group III register is selected by the SELERFG3 signal and the four ERNG3 signals. The SELERFG3 is over-all select for all Group III registers. It defines the digit 1 for all Group III register address (10_{16} through $1F_{16}$) and enables selection of the Group III registers by disabling selection of the Group II registers (02_{16} through $0F_{16}$). The four ERNG3 select signals comprise the register address within Group III. These five select signals are normally generated by corresponding bits of the X field of a Register File Read or Register File Write μI (Fμ bits 11 through 15). During an Invoke condition, the select signals are derived from corresponding bits of the BC register. The decision to perform a read or write into these registers is determined by the ERFG3RD or ERFG3WR signals. Each of these signals is generated from a corresponding Register File Read or Register File Write μI.

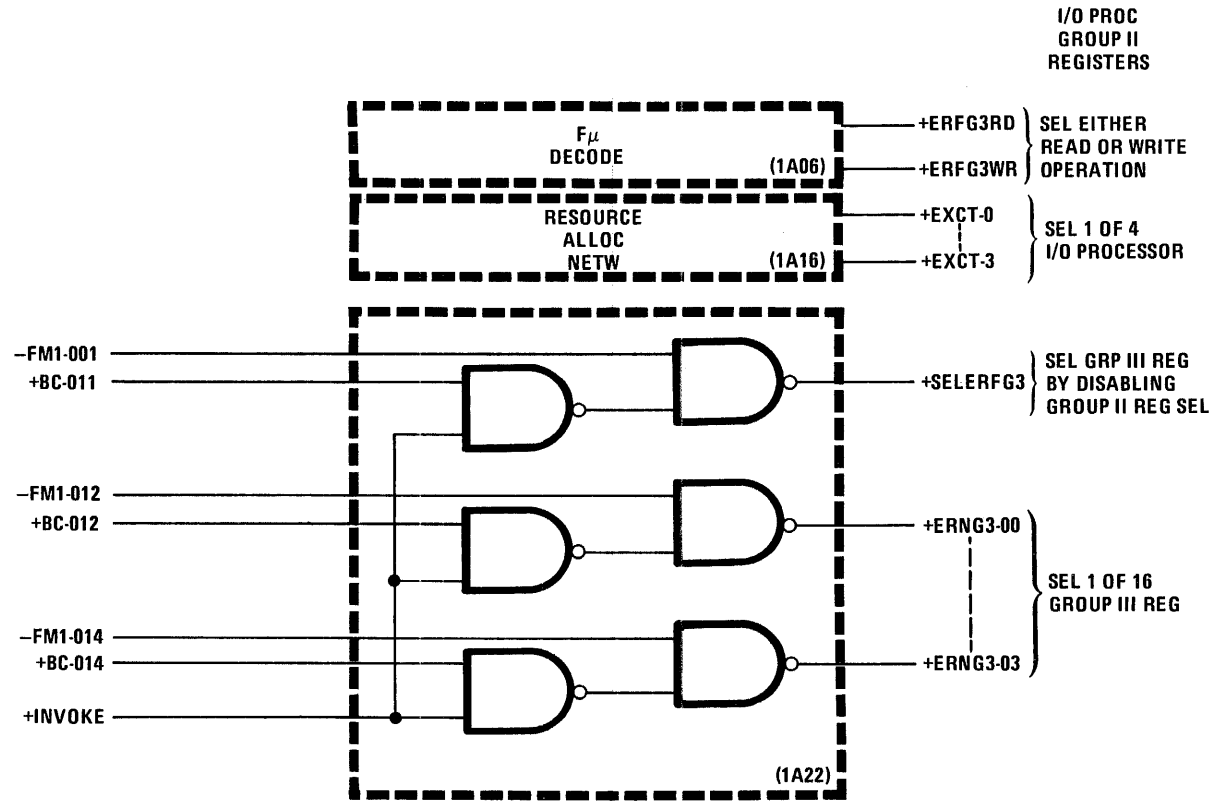


Figure 2-83. ERF Group III Selection

Table 2-6. Reading and Writing File Registers

Register	Processor Oriented	Read Capability	Write Capability	Master Clear
Basic Register File	yes	μI control	μI control	no
Extended Register File				
Group I				
F	yes	hardware or μI control	hardware or μI control	no
Pμ	yes	hardware or μI control	hardware or μI control*	yes
Group II				
Busy/Active	no	hardware or μI control	hardware or μI control	yes
RTC	no	hardware or μI control	hardware control	no
Tie-Breaker	no	μI control	μI control	yes
PE	no	μI control	hardware control	no
Control	no	hardware or μI control	μI control	yes
Privileged Mode	no	μI control	μI control	yes
Boundary Crossing	no	hardware or μI control	μI control	yes
CS Scan	no	μI control	hardware or μI control**	yes
Panel Address	no	μI control	hardware or μI control	yes
Panel Data	no	μI control	hardware or μI control	yes
Group III				
Processor 0 (4 registers)	yes	μI control	μI control	no
Processor 1 (5 registers)	yes	μI control	μI control	no
Processor 2 (5 registers)	yes	μI control	μI control	no
Processor 3 (1 register)	yes	μI control	μI control	no

* μI control write into E and S bit positions will clear them.
**All μI's that attempt a write, except ROM, will clear CSS.

Access Capabilities and Limitations

Because the registers of the BRF and ERF are read and written by a variety of conditions, it is useful to tabulate their conditions in one central location. Table 2-6 lists these conditions in summary form for each file register. Specifically, this table lists each register of the BRF and ERF, whether or not it is processor-oriented, conditions for reading and writing the register, and whether or not it can be master-cleared.

MAIN STORAGE INTERFACE

The Main Storage (MS) interface logic controls data transfers between the MS sections of shared resources and the rest of shared resources (from here on referred to as the Central Processing Unit (CPU) section). All circuits of the MS interface logic function during one of two basic operations: write and read.

Each type of operation performs on data in either word

mode (16 bits) or byte mode (8 bits). The two bytes of a word are referred to as the left-most byte (bits 0 through 7) and the right-most byte (bits 8 through 15). The left-most and right-most bytes are also referred to as bytes 0 and 1, respectively.

Parity is calculated in the CPU for each byte of a word using odd parity (odd number of "1's" in each byte, including the parity bit). The upper byte parity bit is referred to as P0, the lower byte parity bit at P1. The format of a word transferred to MS is shown in Figure 2-84.

An MS write or read operation is always initiated by a Load S (LS1, LSF, LS2, or LSE) μI which transfers the contents of a register in the Basic Register File (BRF) to the S register. These contents specify the address of a location in MS. If the Load S μI is followed by a Load D (LDW, LDW-, or LDB) μI, the contents of a register in the BRF is transferred to the D register and the MS write is initiated. The MS write then transfers the contents of D to the location in MS defined by the address in S. If the Load S μI is not followed by a Load D μI, an MS read operation will unconditionally take place by reading the operand stored at the location in MS and sending it to the data fan-out logic in the MS interface. Normally a Load S μI programmed for a read operation will be followed by some μI that takes the data read from MS and uses it according to the particular μI. These μI's include the Store D (SDW or SDB) μI's and the D → A (DTA, DTA-, IDX, and DFA) μI's. A special use of the SDW μI is during the RNI sequence to read the next MLI to be executed in the program. When used for this purpose, the SDW μI gates the first-level decoded results of the MLI from the FRJ decode address table (AT) and saves the MLI in the F and F_{RF} registers until its execution is completed.

Operating details of the MS interface logic will be presented by discussing the basic elements of the logic (S register, D register, and data fan-in) followed by a discussion of the MS write and MS read operations plus associated timing and control signals involved. Because they are intimately associated with MS control signals, Register Option control signals are also discussed in this section. (A description of the Register Option itself is contained in the paragraph titled Register Option.)

S REGISTER

The S register holds the address of the word or byte to be either read from or written into MS. The register consists of 16 flip-flops, fashioned from cross-coupled NAND gates. The advantage of these flip-flops is that they do not require any data set-up time (about 10 nanoseconds) and provide immediate propagation. This enables the address data to be entered into the register immediately, an important feature since the MS access operation is time critical. A diagram showing one stage of the S register is shown in Figure 2-85. The address obtained from the BRF through the Aμ register fan-in is gated through the EN SET gate when CLKSR and ENCLKSR are present. Signal ENCLKSR is generated by a Load SμI (3,X) at E040 and cleared normally at E140 or upon occurrence of a CS read error (SWPERR). Assuming the address bit to the flip-flop is a 1, the feedback path from the SR (set) output to the CLK LATCH gate keeps the flip-flop set while the address bit is present during the clock period. After both the clock pulse and address bit have been removed, the flip-flop remains set due to the feedback path from the SR output to the NORM LATCH gate. The flip-flop remains set until cleared by a subsequent address bit of 0, together with clock and clock enable signals.

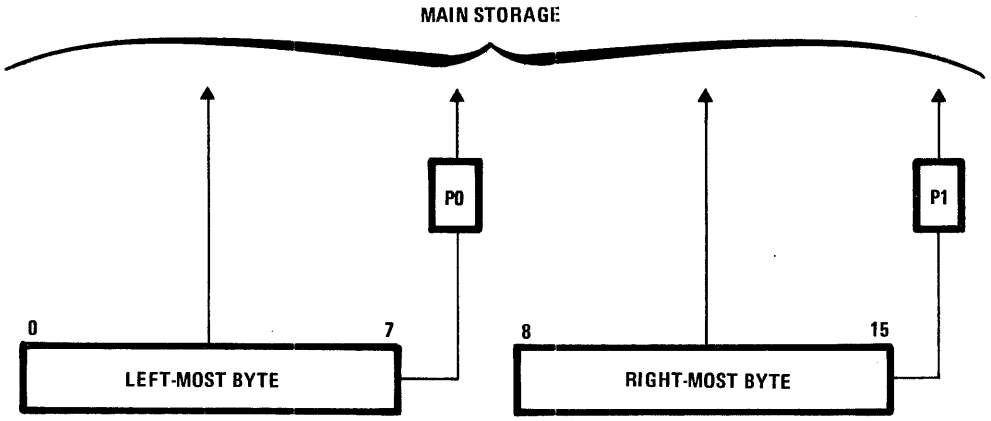


Figure 2-84. Format of Word Transferred to MS

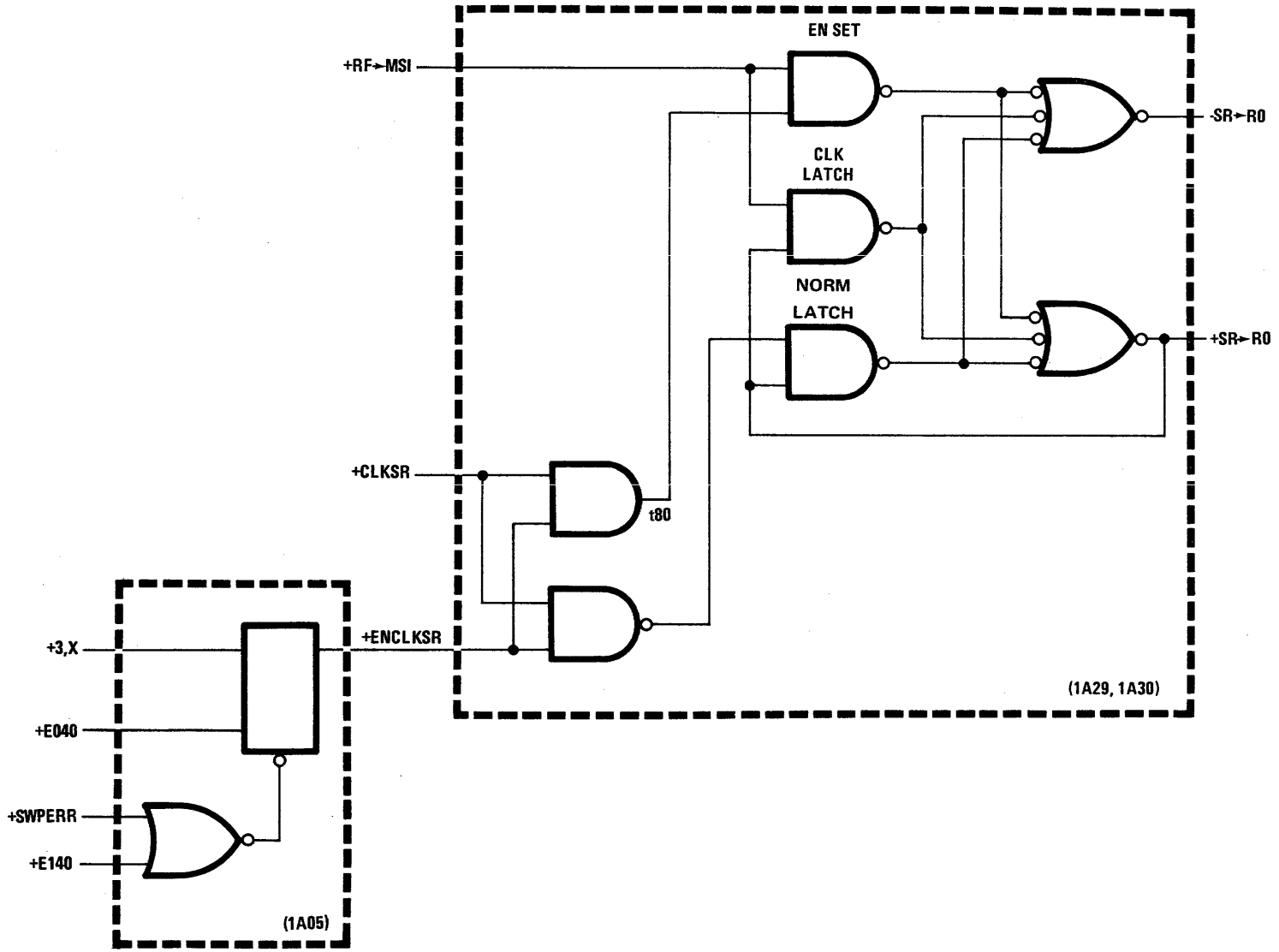


Figure 2-85. S Register

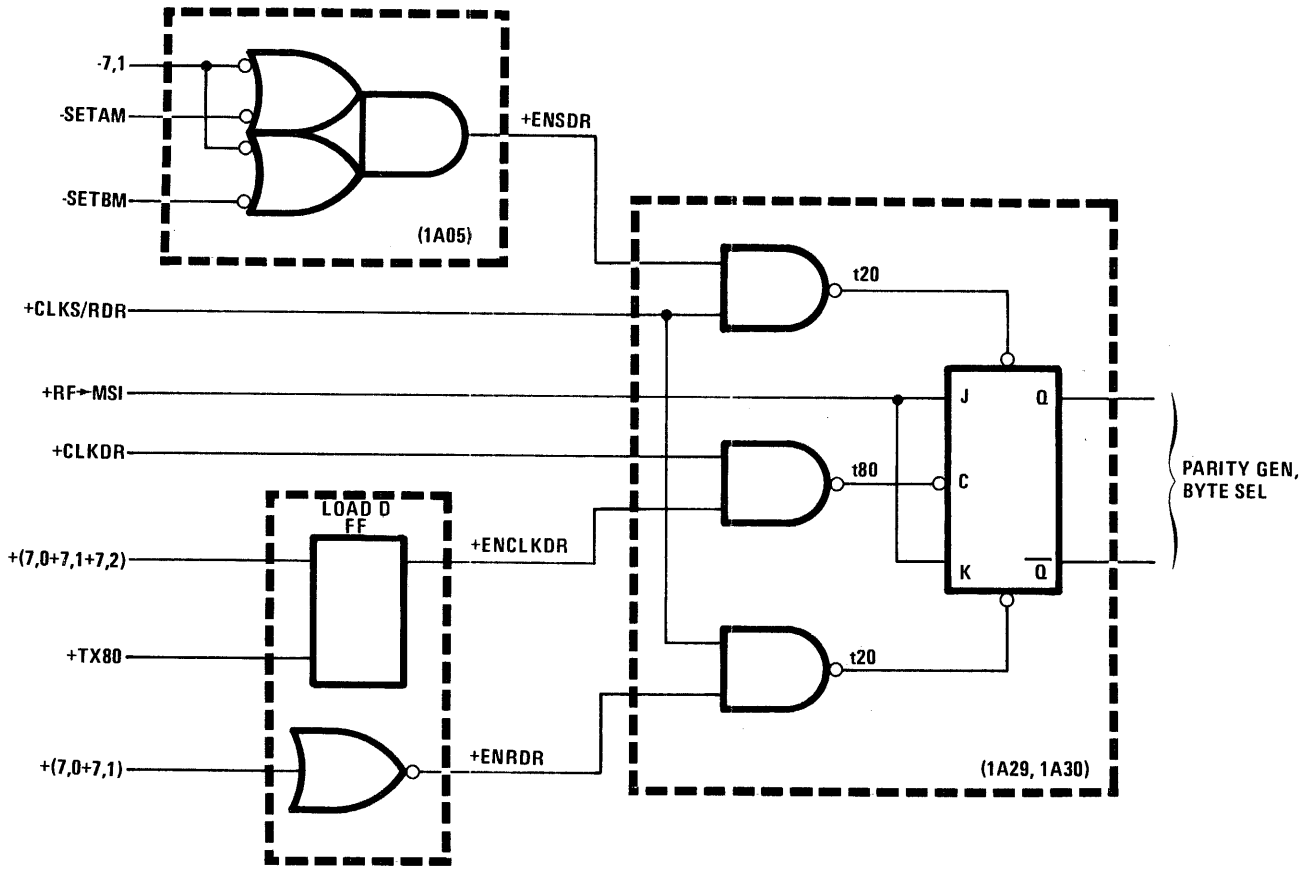


Figure 2-86. D Register

D REGISTER

The D register holds the data to be written into MS at the location specified by the contents of the S register. This data also comes from a BRF register, but under control of a Load D (7,X) μI. The D register is composed of 16 edge-triggered J-K flip-flops. The D register flip-flops can be preset or precleared, as required, to store data in either true or complement form. (See the paragraph titled Aμ and Bμ Registers for a description of storing data in true or one's complement form in the Aμ and Bμ registers.) Logic for a stage of the D register is shown in Figure 2-86. The register is pre-cleared via ENRDR and CLKS/RDR to enable storing data in true form during execution of a LDW (7,0) or LDB (7,2) μI, or during a master clear condition. The register is pre-set via ENSDR and CLKS/RDR to enable storing data in one's complement form during execution of a LDW- (7,1) μI. For all these μI's, data on the RF-MSI lines is clocked in via CLKDR and ENCLKDR. The enable signal is generated by the Load D flip-flop for any Load D μI. If the system is in the maintenance mode, the register can be pre-set to store all "1's" by the SET AM and SET BM signals. These signals are generated by simultaneously pressing the SET Aμ and SET Bμ pushbuttons on the System Control Panel.

DATA FAN-IN

The data fan-in logic, shown in Figure 2-87, provides for selecting one of three data paths to the ALU fan-out logic:

1. Register Option (RO)
2. D register
3. Main Storage (MS)

Data read from either the RO or MS is done so in a similar manner, by means of a Load S μI. For this μI, MSREADY is generated at E440 time of a time slice. In the case of a RO read, ROREAD must also be present. In the case of an MS read, ROREAD must be low to specifically inhibit an attempted read from the RO. Data from the D register can be enabled when neither the RO or MS is reading out data (whenever MSREADY is low). For this condition, the data contained in D will usually represent an operand obtained from a file register.

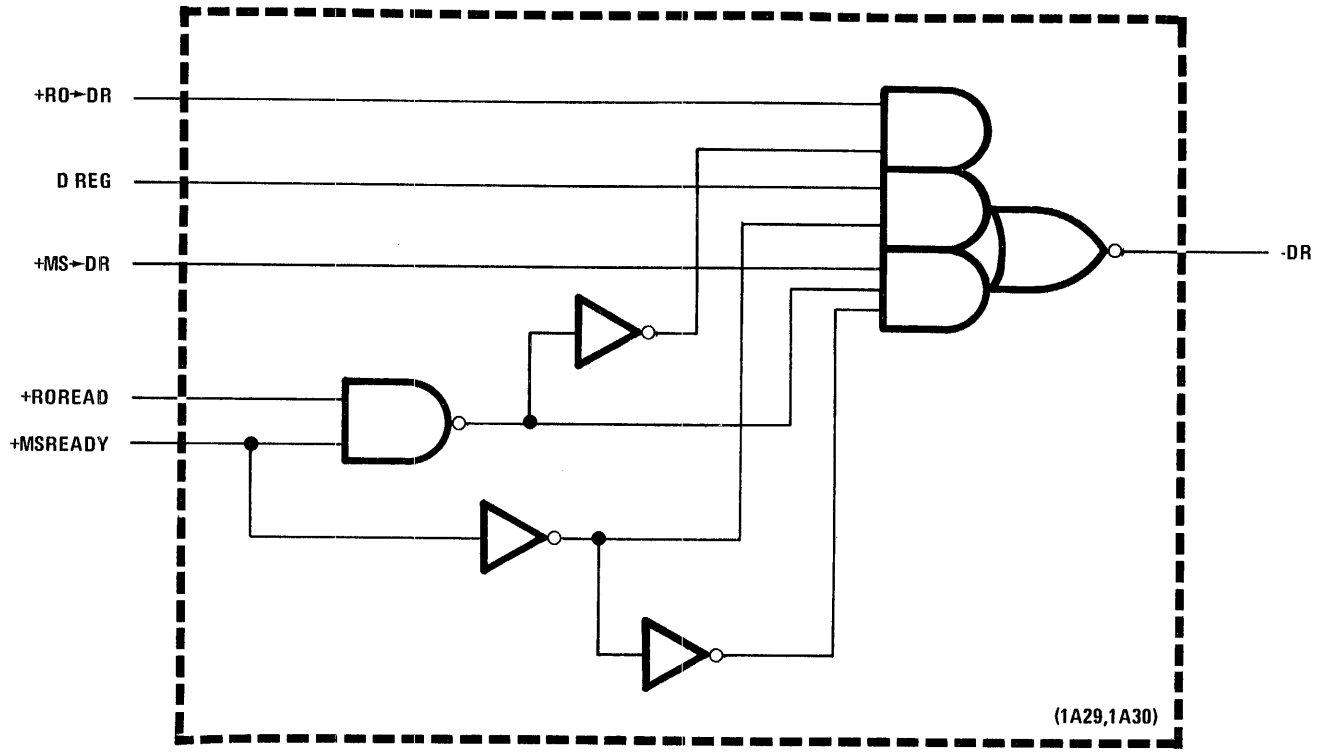


Figure 2-87. Data Fan-in

MS WRITE OPERATION

A block diagram for the MS write operation is shown in Figure 2-88. Data can be stored transferred to MS in either word mode (16 bits) or byte mode (8 bits). Selection of a byte to be transferred is under control of the STDBYTE enable, which enables the byte select logic. When STDBYTE is low, all 16 bits of the D register are stored in 16 bits of a location in MS specified by the address in S. When STDBYTE is high, the byte contained in bits 8 through 15 of D is transferred in parallel to both the left-most byte and right-most byte of the location in MS. The choice of storing bits 8 through 15 of D in either the left-most byte or right-most byte of MS is determined by the state of bit position 15 of the address in S, as discussed in greater detail in the following paragraph (Word and Byte Write Operations). Odd parity for the word or byte to be stored is generated on a byte basis by the parity generator. The parity generated for each byte is stored along with the byte as shown in Figure 2-84. Prior to being routed to MS, the contents of both S and D are routed to the Register Option (RO) logic. Among other things, the RO may contain segment tags whose contents are appended to the address contained in S to form an effective address at which data in D will be stored in MS. The data from D and the byte select logic also pass through the RO for the singular purpose of initially loading certain registers and tables in the RO. Thereafter, the RO is transparent to all data passing through to MS.

Word and Byte Write Operations

Word and byte writes into MS are performed by the logic of Figure 2-89. Selection of either a word or a byte to be transferred to MS is controlled by the byte selector. The byte selector is fed with outputs from both the left-most half and right-most half of D, and routes the byte contained in either half to the left-most half of a location in MS as specified by the state of STDBYTE. This select signal works in conjunction with MS byte write enables STOREUPP and STORELOW to store data in MS. As Figure 2-89 shows, there are threee ways in which data can be stored in MS depending on the μI executed. Part *a* shows a whole word store operation as performed by either a LDW or LDW-μI. For this case, STDBYTE is low so that data from D is sent directly to MS without the right-most byte being multiplexed onto the lines feeding the left-most half of the MS location. (This path is indicated by dashed lines signifying that the path is not enabled for a whole word store.) In addition, both STOREUPP and STORELOW are high to store both halves of the 16-bit word. Parts *b* and *c* of Figure 2-89 show a byte store operation: part *b* showing a byte write into the left-most half (bits 0 through 7) of the MS location and part *c* showing a byte write into the right-most half (bits 8 through 15) of the MS location. In both cases, the byte to be stored in MS must be located in the right-most half of D. In both cases, STDBYTE is high so that the bytes in bits 8 through 15 are transferred in

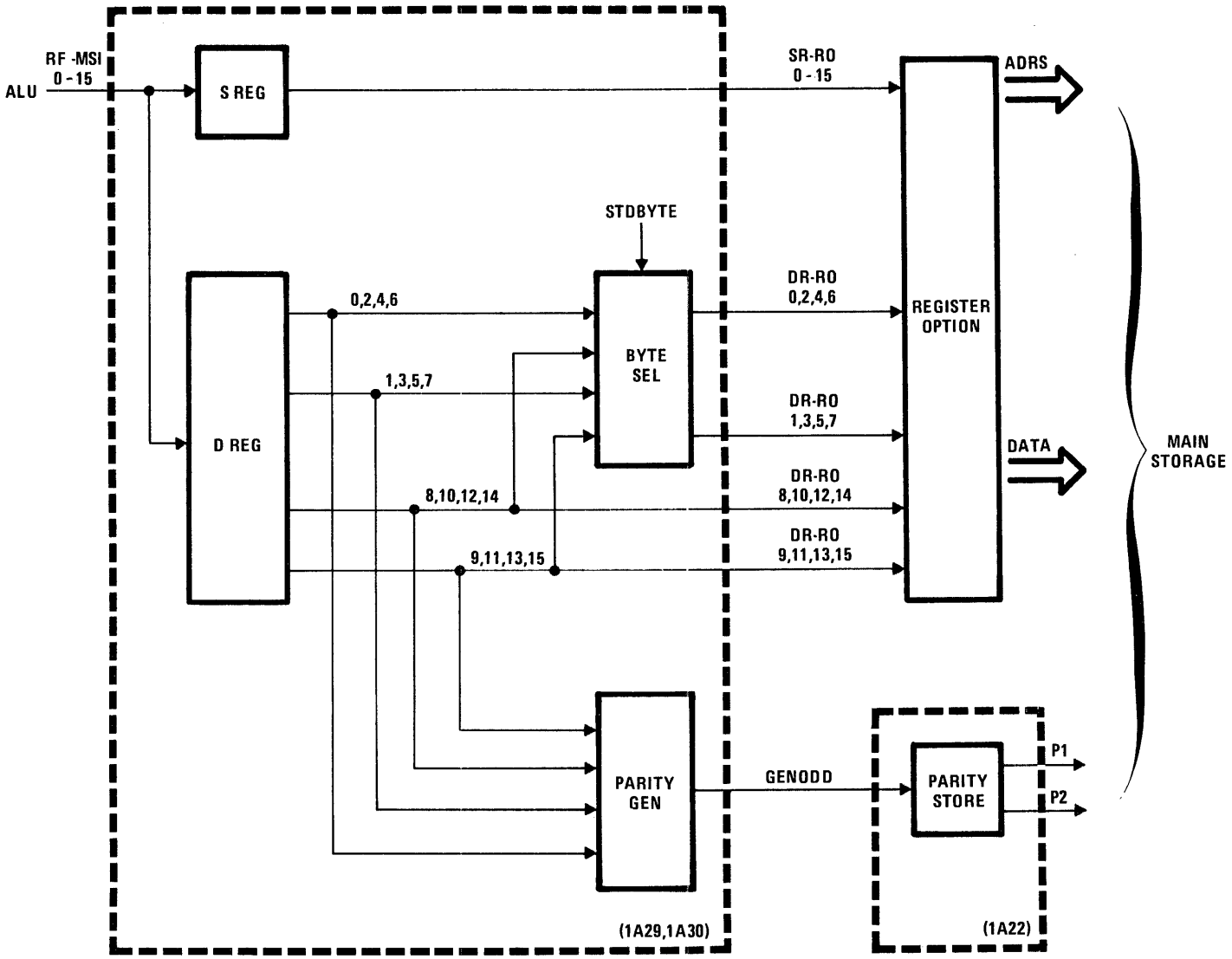


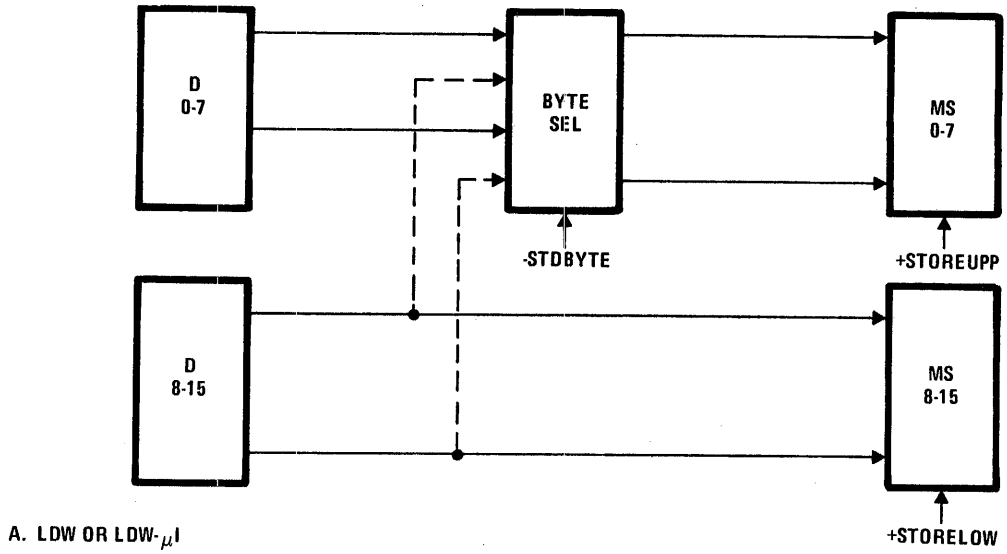
Figure 2-88. MS Write Operation

parallel to both the left-most and the right-most half of the MS location. However, the byte will be stored into only one half depending on which MS write enable is generated (STOREUPP if writing into the left-most half or STORELOW if writing into the right-most half).

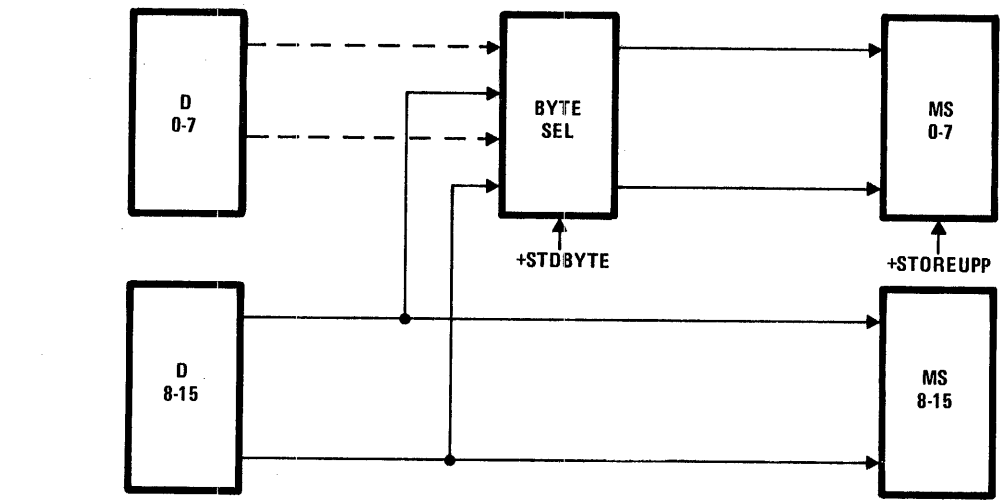
Logic for generating the byte write enables is shown in Figure 2-90. Enables STOREUPP and STORELOW are generated simultaneously during a word store by means of STOREMS and STDBYTE. Signal STOREMS is generated during any MS access requiring data to be stored and STDBYTE is low for a word store operation. During a byte store, either STOREUPP or STORELOW is generated, depending on the state of SELBYTE0. If a left-most byte store is indicated, SR 15 is low and SELBYTE0 is high. If a right-most byte store is to be performed, SR 15 is high and SELBYTE0 is low. Before being sent to MS, the two store enables are first passed to the RO for combining with the bounds check signals. These check signals determine whether or not the addressed location can be written into. If not, the store enables are disabled.

Parity Generate and Store

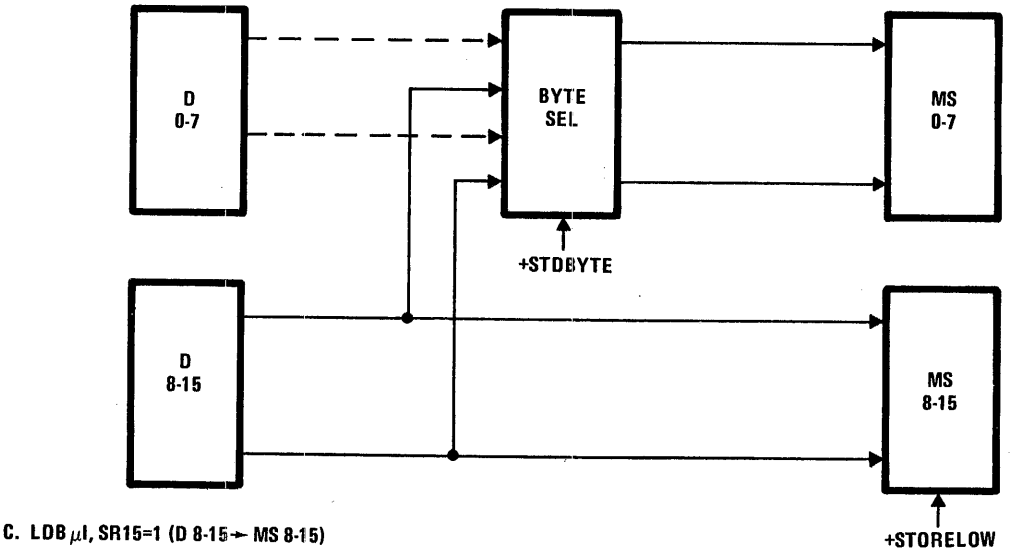
During a word write, parity is calculated for each byte of a word. The parity bit calculated for the left-most byte of a word is transferred to MS along with the word as bit 16, and that for the right-most byte is transferred as bit 17. During a byte write, the parity bit calculated for the right-most byte in D is transferred to MS as both bits 16 and 17. Either one of the two bits will be written with the byte depending on whether the byte is to be stored in the left-most half or the right-most half of the location on MS. A block diagram showing generation and storage of



A. LDW OR LDW-μI



B. LDB μI, SR15=0 (D 8-15 → MS 0-7)



C. LDB μI, SR15=1 (D 8-15 → MS 8-15)

Figure 2-89. Word and Byte Store in MS

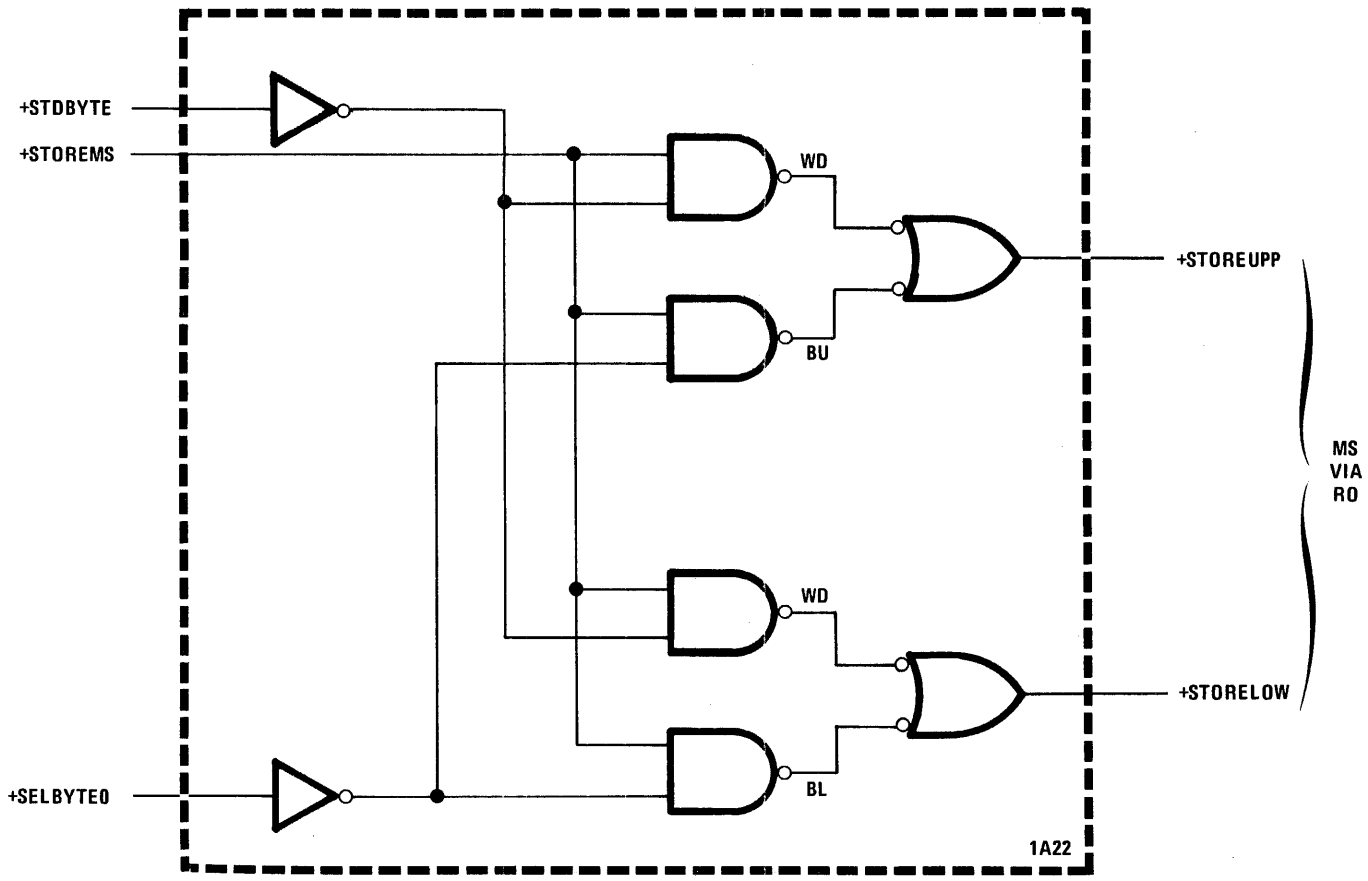


Figure 2-90. Generation of Byte Write Enables

parity bits is shown in Figure 2-91. Parity is calculated by a parity tree which generates parity for each byte in three stages. Stage 1 combines the true and complement states of the even-numbered bits of a byte together to generate corresponding GENODD signals. Each GENODD signal is high if the number of "1's" combined is odd. Stage 2 combines the true and complement states of the GENODD signals for each byte to generate two other signals. These signals are fed to the third stage, which is also fed with other inputs, to generate the final byte parity bit.

During a byte store, only the parity bit generated for the right-most byte in D has meaning. However, this bit must be fed in parallel to MS as both bits 16 and 17 since the byte in D may be stored in either the left-most or the right-most half of the MS location. For this operation, STDBYTE is high to (1) enable the left-most byte store gate so that the right-most byte parity bit may be sent out over the bit 16 line, and (2) disable the word store gate to prevent the parity bit generated for the (meaningless) left-most byte in D from being sent out over the bit 16 line. During a word store, both bytes contained in D have meaning; therefore, separate parity bits must be generated for each. For this case, STDBYTE is low to enable the word store gate and disable the left-most byte store gate for bit 16.

Signals SWEVEN16 and SWEVEN17 are generated by the BYTE PARITY 0 and BYTE PARITY 1 pushbuttons, respectively, on the System Control Console. These pushbuttons are used to change the parity generated for each byte from odd to even as a means of manually checking the parity detect logic.

MS READ OPERATION

A block diagram for the MS read operation is shown in Figure 2-92. Like the MS write operation, the S register is loaded with an MS address via a Load S μI. This μI performs the MS read and loads the data read into the data fan-in logic. This data is then routed from the data fan-in logic via a Store D or D→A μI and used according to the particular μI. If one of the above μI's is executed in a time slice that did not reference MS by a preceding Load S μI, the data obtained will come from the D register instead of from MS. All data read from MS is checked for correct parity. This is done by routing the data from MS in parallel to both the data fan-in logic and the parity

Table 2-7. Word and Byte Read Operations

μI or Condition	Operation	Read Enables Generated
SDW μI	DR 00-07→ALU 00-07 DR 08-15→ALU 08-15	SEL-DR-0 SEL-DR-1
SDB μI (SR 15 = 0)	0's→ALU 00-07 DR 00-07→ALU 08-15	SEL-ZR-0 SEL-DRB-0
SDB μI (SR 15 = 1)	0's→ALU 00-07 DR 08-15→AL U 08-15	SEL-ZR-0 SEL-DR-1

check logic, as shown in Figure 2-85. The parity check logic checks the data for even parity by combining the data bits (bits 0 through 15) with the parity bits (P1 and P2) on a byte basis. A parity error in either the upper byte or lower byte of a word generates an error signal which causes a jump to a parity error trap routine.

Word and Byte Read Functions

Although data from MS is always read in word form, 16 bits at a time, the data may be masked to byte form, depending on the particular μI that initiated the read operation. This byte masking is performed by the ALU fan-out logic under control of read byte enable signals. The types of MS read functions that can be performed are listed in Table 2-7. The SDW μI reads whole words and routes them to register X via the ALU fan-out. The SDB μI also reads whole words but masks out (sets to 0) either the upper 8 bit positions or the lower 8 bit positions of the word read to form a byte to be operated on. The choice of forming either a left-most byte or a right-most byte is determined by the state of bit 15 of the S register.

Note that regardless of whether the SDB μI specifies either the left-most byte or the right-most byte, the byte is always handled as a 16-bit word, right-justified if necessary, with the upper 8 bit positions set to 0.

The portion of the ALU fan-out logic used to perform the aforementioned read operation is shown in Figure 2-93. The figure has been simplified to show bit transfers in groups of four bits each. The enables required for each type of read operation listed in Table 2-7 are also shown in this figure. Each operation requires two enables: one to handle bits 0 through 7 and one to handle bits 8 through 15. Generation of these read enables is shown in Figure 2-94. Signal SELBYTE0 is genrated from the complemented output of the S register bit 15 flip-flop. It is used to select either the left-most byte (SELBYTE0 high) or the right-most byte (SELBYTE0 low) during execution of a SDB μI.

MS PARITY CHECK

Parity is checked on all data read from MS by the parity check circuits. Checks are made on a byte basis using odd parity. A portion of the parity check logic is shown in Figure 2-95. This logic checks parity of the left-most byte (bits 0 through 7) of the MS word. As shown, the logic consists of two parts: the parity checker circuit, which checks parity of bits 0 through 3 and bits 4 through 7, and the MS parity check/display circuit, which generates a parity error signal (PE-BYTE0) if parity is not correct. As an example, assume the left-most byte stored in MS was all "1's". Since this constitutes an even number of "1's", the parity bit generated is also "1" so that the total number of "1's" in the byte stored, including parity, is odd (8 + 1 = 9 "1's"). Now assume that when the byte was read, bit 0 was erroneously read as a "0" instead of a "1". Since the error occurred in one of the four odd bits of the byte, CHKEVEN0 will go high. Bits 1, 3, 5, and 7 were read without error, however, and CHKEVEN1 goes low. These two outputs are combined with the P1 parity bit from MS to generate a low PE-BYTE0 signal, signifying an error in the data read. The low PE-BYTE0 signal is routed to the MS parity error display logic and to a parity error trap routine to recover from the error condition.*

*If the ECC option is present, parity bit P1 is meaningless since the ECC will provide automatic correction of single-bit errors. For this condition, the trap routine will be performed only if an error occurs that the ECC cannot correct.

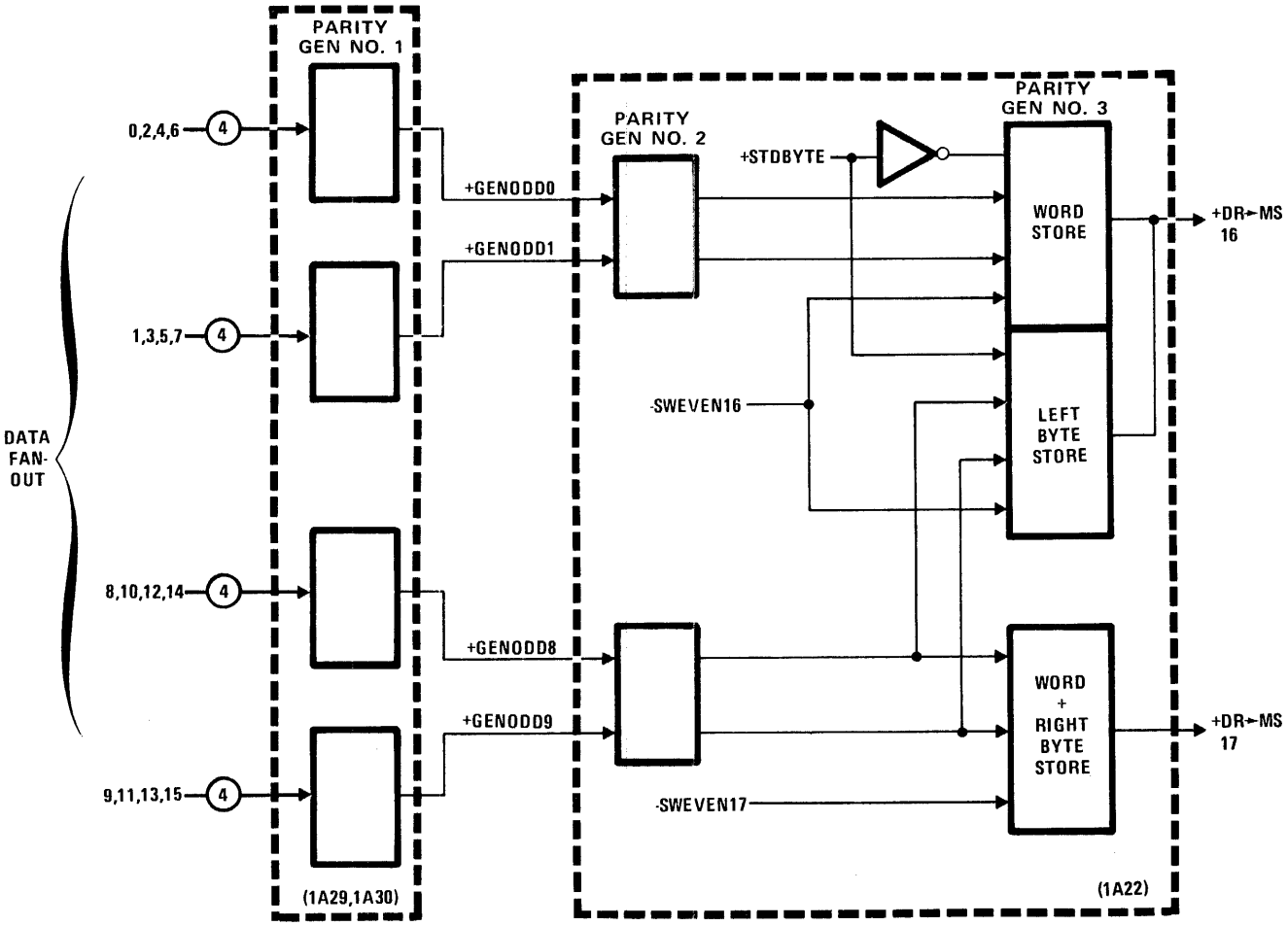


Figure 2-91. Parity Generate and Store

MS Parity Error Display

The results of the MS parity check logic are sent to the MS parity error (PE) display logic to light the MS PARITY ERROR indicator on the System Control Panel upon detection of an MS PE. This logic also lights the indicator upon detection of an irrecoverable ECC error if the ECC option is present in the system. The MS PE display logic is shown in Figure 2-96. Information from the parity check logic and the ECC logic are fed to gate 1. This gate generates a high output if (1) the ECC option is present (ECCPRES) and a non-recoverable ECC error ($\overline{\text{ECCERROR}}$) is generated, or (2) the ECC operation is not present and a PE in either byte 0 or byte 1 read from MS ($\overline{\text{PE-1}}$ + $\overline{\text{PE-2}}$) is present. For either condition, signal STOREMS must be low to indicate that the operation being examined is not an MS store. The output of gate 1 is ANDed with $\overline{\text{OUTBOUND}}$ · $\overline{\text{OUTRANGE}}$ by gate 2. These two signals indicate that the PE occurred within the assigned bounds protect limits ($\overline{\text{OUTBOUND}}$) and within an existent portion of MS ($\overline{\text{OUTRANGE}}$). The result, designated MS PE, is sent to the MS PE flip-flop to light the MS PE indicator on the Panel and to the Console Busy flip-flop clear logic to turn off the Panel upon detection of an MS PE (see the paragraph titled Console Modes).

MS INTERFACE SIGNALS

References to MS require a number of interface signals used both for initiating a reference and to restrict certain time-constrained operations associated with MS references. These signals are divided into three categories: MS reference, MS write, and MS read signals. Because they are intimately associated with those required for MS references, control signals used during references to the RO are also discussed in this section. As an aid is discerning the differences between all MS and RO control signals, many of which perform identical functions but when different conditions, the purpose of each control signal is listed in Table 2-8. Timing for all MS control signals is shown in Figure 2-97.

Table 2-8. MS Interface Signals

Signal	Function
	MS Reference (Write or Read)
ASYNC	Prevent Fμ from being clocked for remainder of time slice if Load S μI executed at $\overline{E0}$.
ACCESSEN	Initiates MS reference (either read or write)
	MS Write
$\overline{DREADY}$, $\overline{DREADE}$	D register ready to be loaded with data to be stored on MS via Load D μI.
STOREMS	Indicates MS write operation, used to enable word and byte write signals.
	MS Read
DREADY	Contents of D from MS ready to be transferred to register X via Store D μI.
DREADE	Contents of D from MS ready to be transferred to Aμ via D→Aμ I.
MSREADY	Enable to gate data read from MS through data fan-out.
	RO Reference (Write or Read)
MS-SPEC	Reference to be made to register in ECC feature of RO.
RO-SPEC	Reference to be made to register in basic protection, relocation and protection or job accounting feature of RO.
ROREADY	Enable to gate data read from RO through data fan-out.

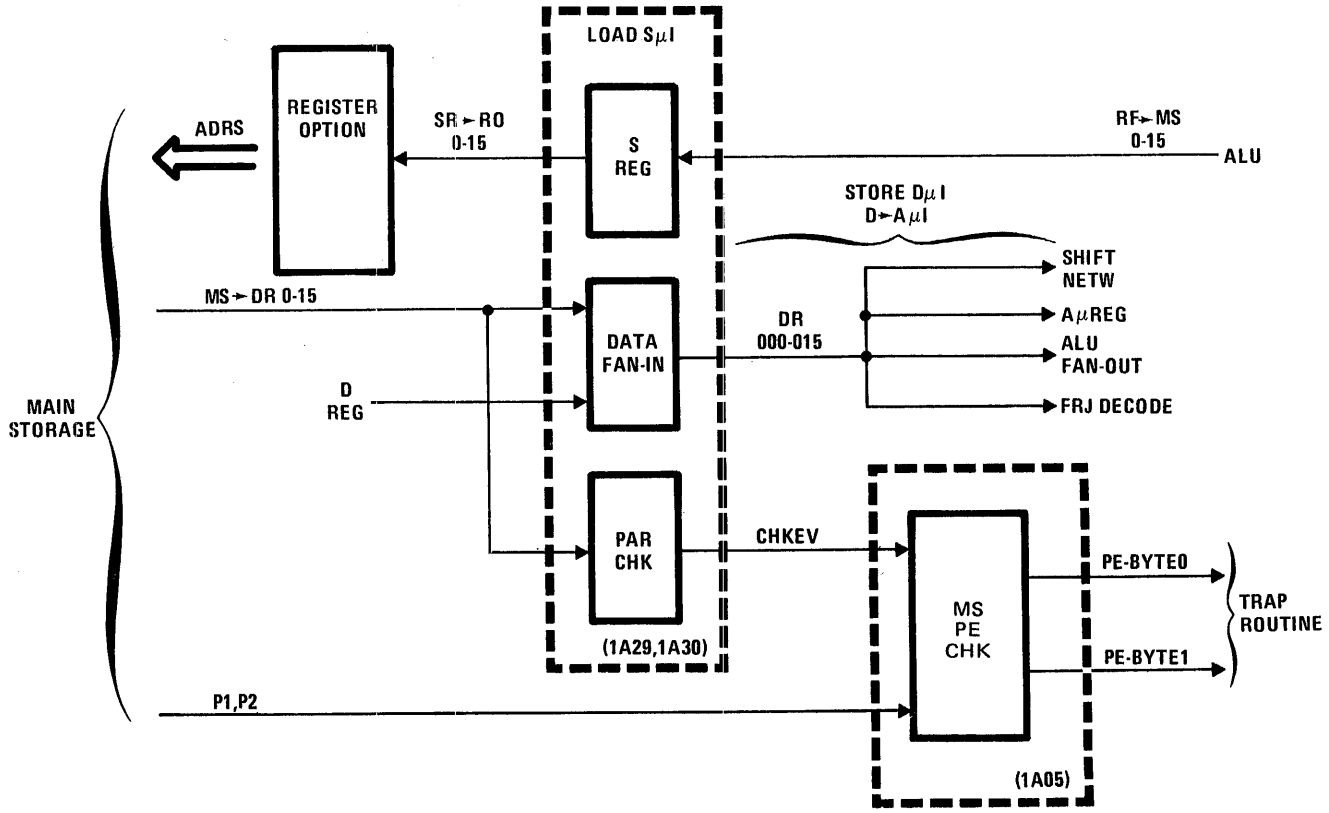


Figure 2-92. MS Read Operation

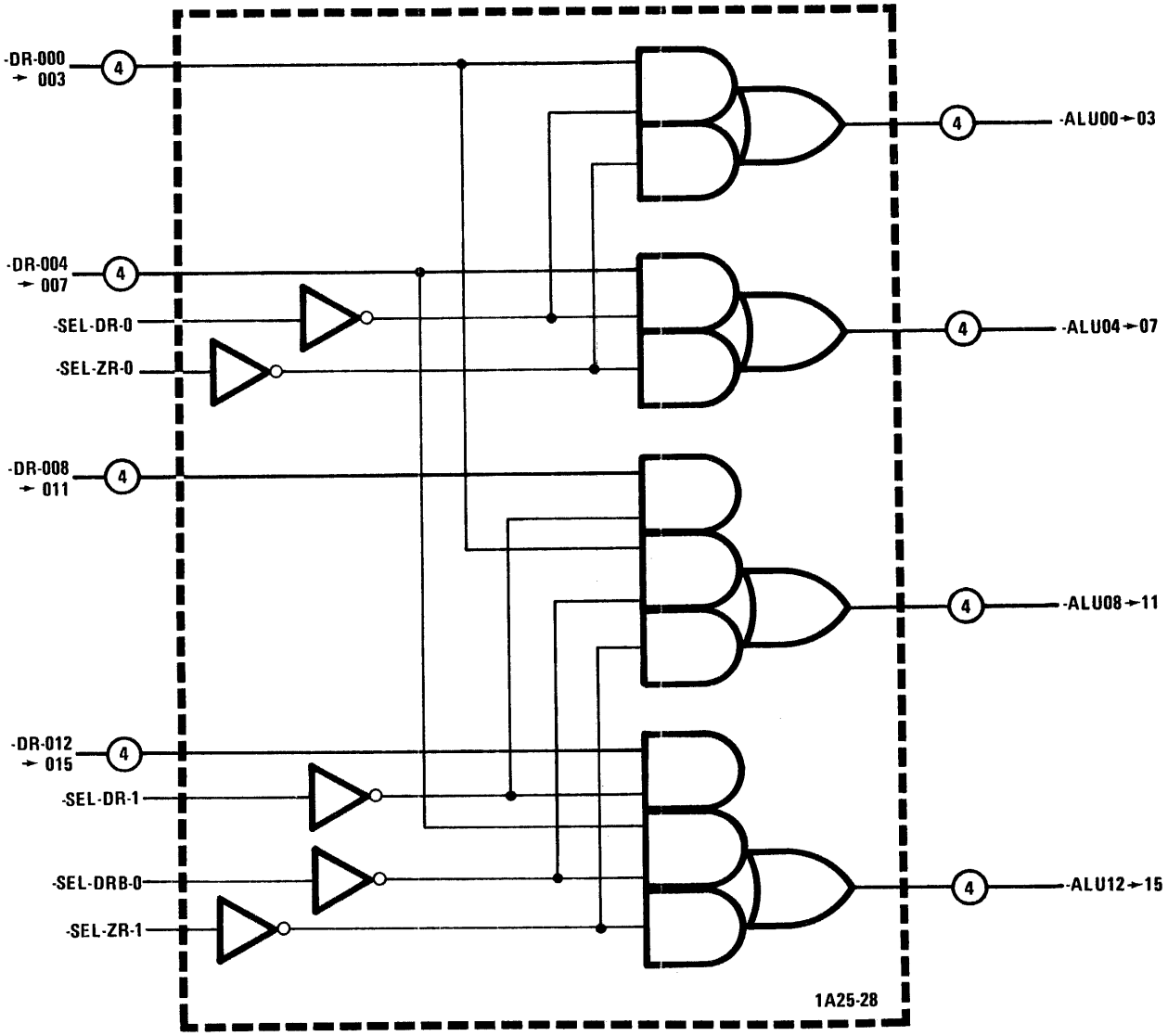


Figure 2-93. Word and Byte Read Data Transfers

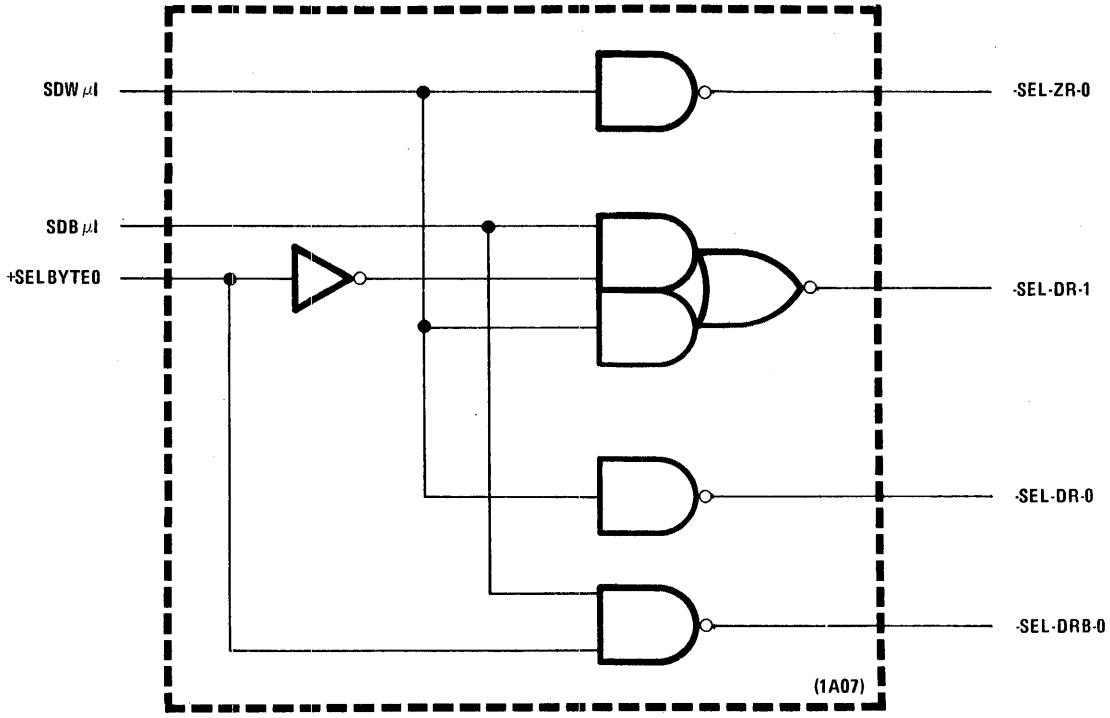


Figure 2-94. Generation of Byte Read Enables

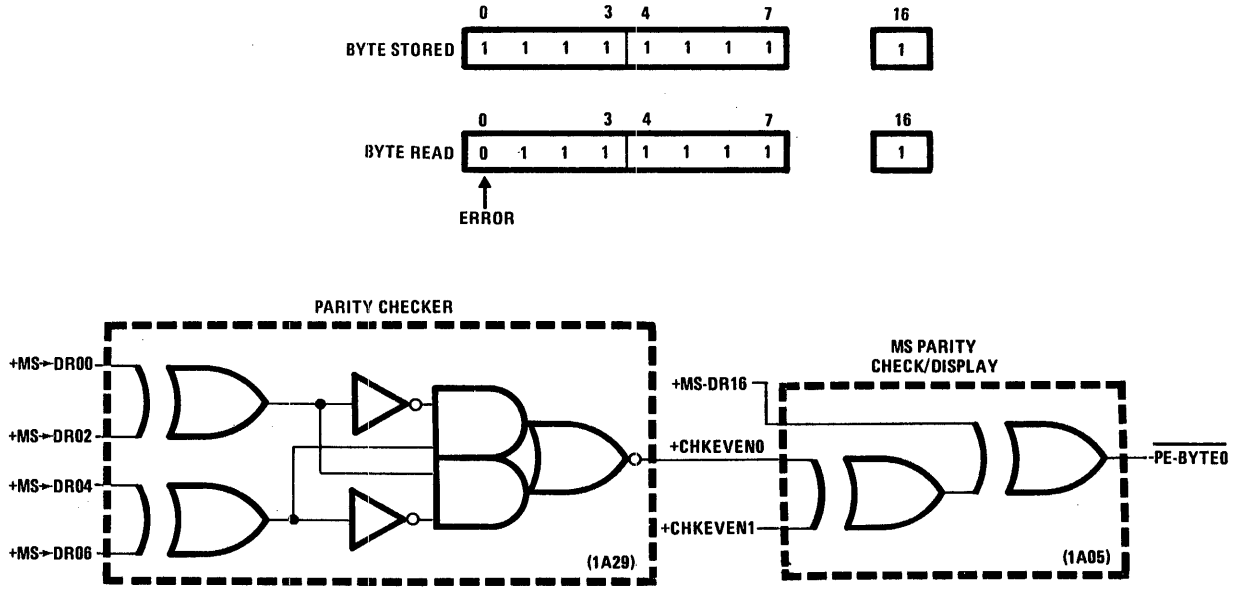


Figure 2-95. Parity Check Logic

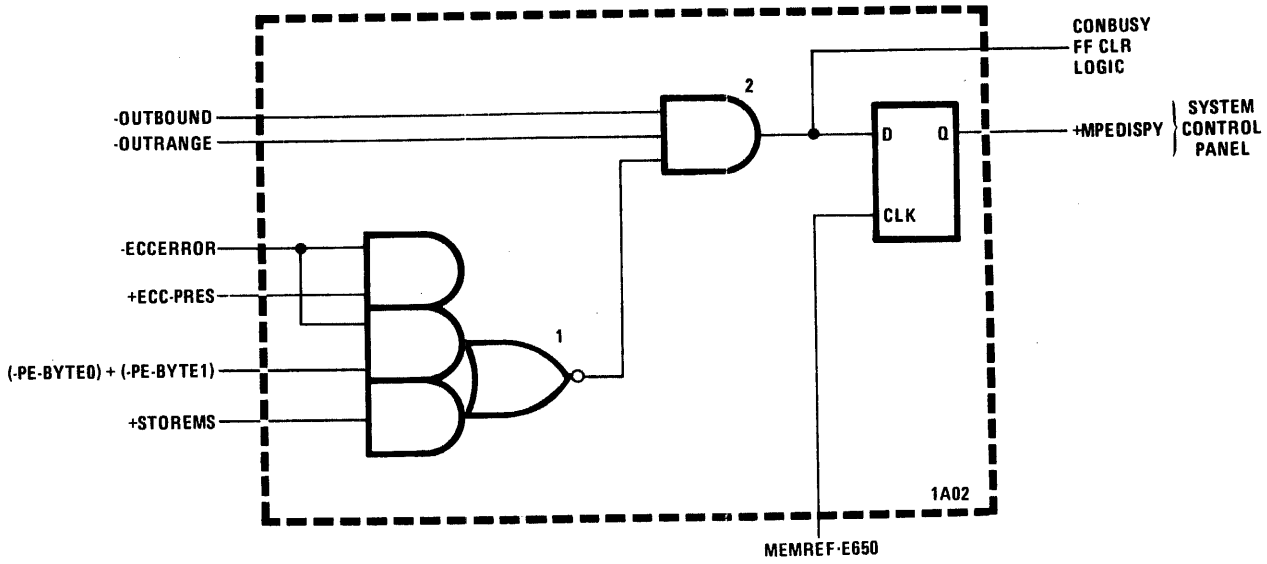


Figure 2-96. MS Parity Error Display Logic

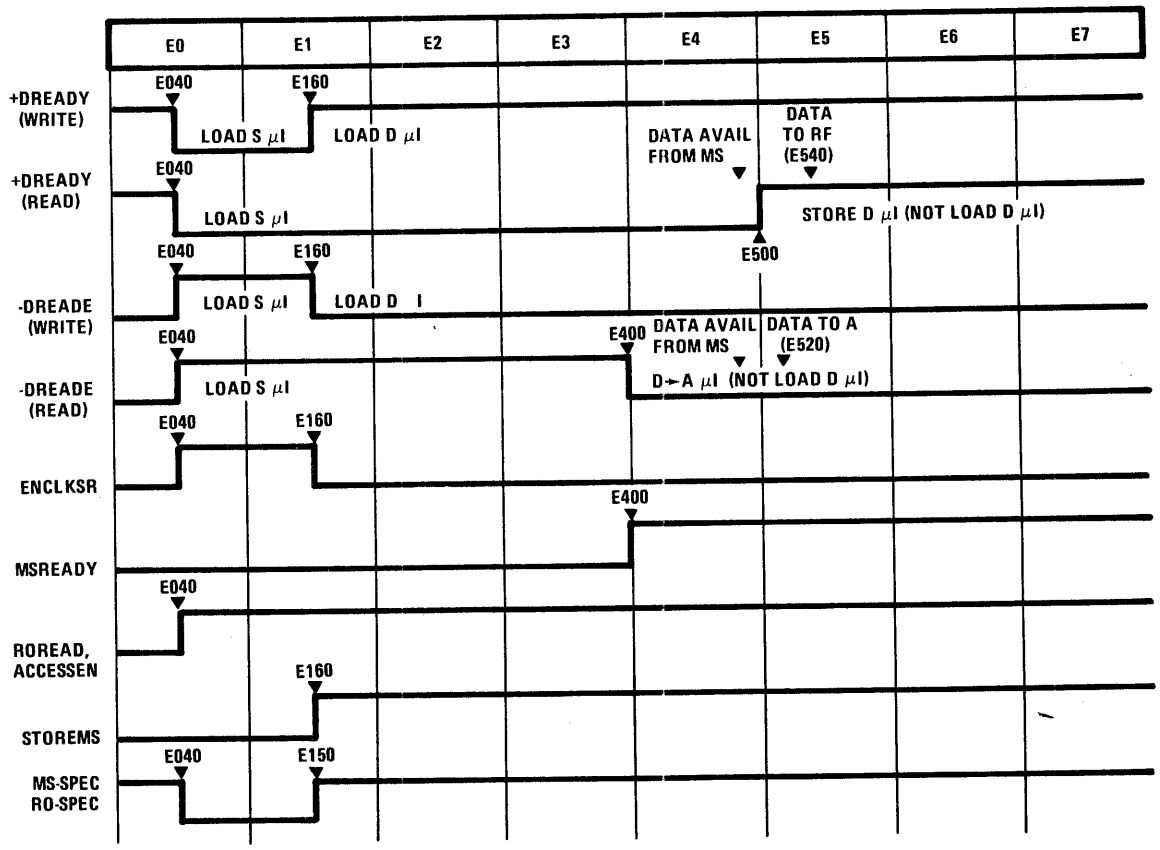


Figure 2-97. MS Control Signal Waveforms

MS Reference Signals

As discussed previously, an MS reference operation (whether read or write) always begins with a Load S μI. As far as the CPU is concerned, an MS write can be performed in two minor cycles (200 nanoseconds), the time required to execute a Load S μI followed by a Load D μI. An MS read, however, takes five minor cycles to perform from the start of the Load S μI (E000) until data read from MS is available at the data fan-in (E480 to E505). To assure that enough time will always be available to read MS during the same time slice, the MS interface logic unconditionally forces execution of the Load S μI during E0 regardless of whether a read or write is to be performed. If occurring in the program at any time other than E0, the non-MS reference portions of the μI will be performed (that is, (X)→Aμ, constant→Bμ, and 0/+1→FCR). However the (X)→S transfer will not be performed since S can be clocked only during E0 which effectively keeps the MS read from taking place. Instead a resync condition is set up and the Load S μI is re-executed at E0 of the next time slice. The resync condition is implemented by the ASYNC signal, shown in Figure 2-98. The signal is generated by a Load S μI executed at any time other than E0. This signal, in turn, generates BLOCKFM which keeps Fμ from being loaded with the following μI for the rest of the time slice. To assure that the Load S μI be executed during the next E0 time, it is necessary to program a blockpoint μI immediately preceding the Load S μI. (If the blockpoint μI were not used, the μI routine would execute through the Load SμI, idle through the rest of the time slice; then in the next time slice, start at the same μI as the present time slice (since no new block point address was provided) and repeat the same μI's up to the Load S μI.)

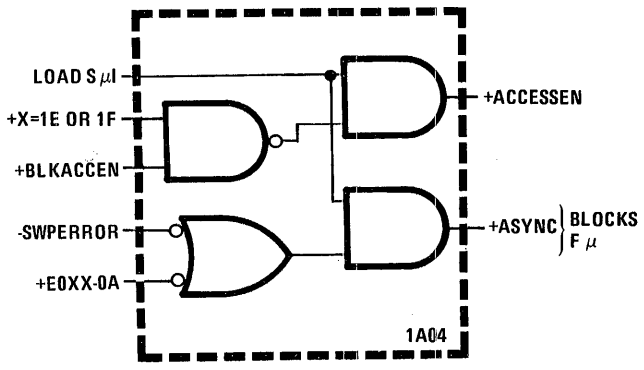


Figure 2-98. MS Reference Signals

Signal ACCESSEN is sent directly to MS to initiate the MS reference. The signal is generated upon translation of a Load S μI if an RO reference has not been requested; that is, neither X=1E or 1F or BLKACCEN is high. Signal X=1E or 1F is generated if the X-field of the Load S μI specifies the contents of transient register 1E or 1F of the BRF. These two registers, reserved for exclusive use by the RO, contain the address of a register in the RO to be referenced. Signal BLKACCEN is generated if the system is in the maintenance mode when a RO read or RO write is initiated by the CONSOLE MODE SELECT selector on the System Control Panel.

MS Write Signals

Indication that an MS write operation is to be performed is furnished by a Load D μI executed at E1. This sets the Store MS flip-flop at E160 to generate STOREMS, as shown in Figure 2-99. This signal is used to enable generation of word and byte store signals STOREUPP and STORELOW, as discussed previously.

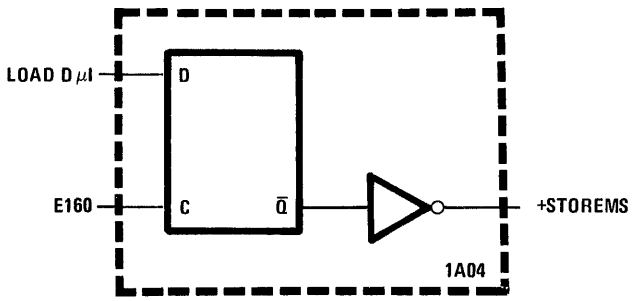


Figure 2-99. MS Write Signal

MS Read Signals

As explained earlier, data read from MS is not available at the interface until about E480. Therefore, the execution times of the μI's that use this data (the Store D and D→A μI's) are restricted accordingly. For a Store D μI, data from the MS interface is transferred to a file register at t40 of the minor cycle in which execution of the μI began. This means that a Store D μI cannot be executed prior to E5 to allow storage of data at E540. For a D→A μI, however, the transfer of MS data to Aμ does not take place until t20 of the minor cycle following that in which the μI began its execution. This means that a D → A μI can begin execution at E4 because the data from MS is not transferred to Aμ until E520.

Signals DREADY for the Store D μI's and DREADE for the D→A μI's are generated for the purpose of providing these timing restrictions. Logic for generating the signals is shown in Figure 2-100; associated timing is shown in Figure 2-97. Each signal is generated for two different conditions: a Load SμI followed by a Load DμI (an MS write operation), and a Load S μI followed by any μI other than a Load D μI (an MS read and store operation). For either type of operation, both DREADY and

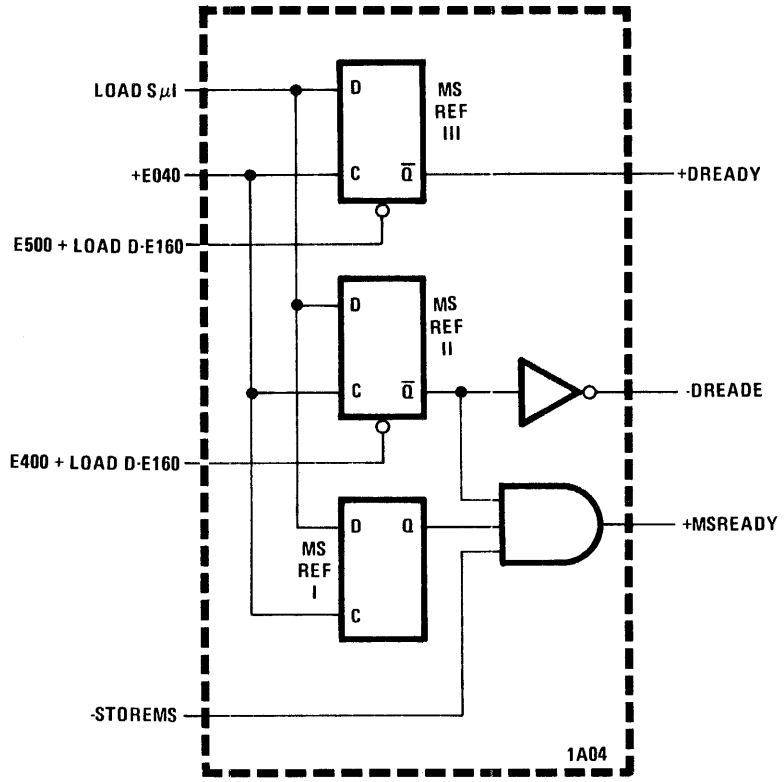


Figure 2-100. MS Read Signals

DREADE go low at E040 when their respective flip-flops are set. If a write operation is being performed, both flip-flops are set. If a write operation is being performed, both flip-flops are cleared at E160 time. This allows the Load D μl to be executed during the next minor cycle to store the data at the address contained in S. If a read operation is being performed (not Load D μl), the flip-flops stay set to keep DREADY low until E510 and DREADE low until E400. During this time, DREADY and DREADE are inverted and ANDed with STORE D and D→A respectively. The result is to schedule NOP's by preventing the following μl address and from being loaded into Sμ and Fμ, respectively. The flip-flops stay set until E400 and E510, at which time the Store D and D→A, μl's can be executed as discussed previously.

For either of the two above situations, data from MS is gated through the data fan-in logic by MSREADY. This signal is generated as a result of a Load S μl at E400 unless the next μl is a Load D (signal DREADE is low). The Load D μl signifies that an MS write is to be performed; therefore, no data is to be read from MS.

RO Reference Signals

Logic for generating the three RO reference signals is shown in Figure 2-101 with associated timing shown in Figure 2-97. Signals RO-SPEC and MS-SPEC are generated as a result of addressing a RO register either in the RO iteslf (Basic Protect feature, Relocation and Protect feature, or Job Accounting feature) or in MS (ECC feature). Both signals are generated upon setting the MS Reference/RO Select flip-flop, indicating that a RO access is to be made. In addition, both signals are low during E040 to E150 to allow the Load S μl to gate the RO register address into either the RO or MS. Signal RO-SPEC is generated specifically if the register addressed is not in the ECC feature (SR-MN04 is low). Signal SR-MN04 indicates that bit 4 of the RO register address in BRF register 1E or 1F is cleared ("0"), which eliminates selection of the ECC. Signal MS-SPEC is generated specifically if the register addressed is in the ECC feature (SR-MN04 is high). Note that MS-SPEC does not begin an MS reference to read or write from MS proper. It only allows accessing the ECC registers of the RO.

Data read from a register in the RO *proper*, (meaning not the ECC feature in MS itself), is gated through the data fan-in logic in the interface by ROREAD. This signal is generated at E040 if either X = 1E OR 1F is present, meaning that an RO reference has been specified. For both of these enabling conditions, SR-MN04 must be high. (Data from an ECC register is gated through the data fan-in as MS data via enable MSREADY.)

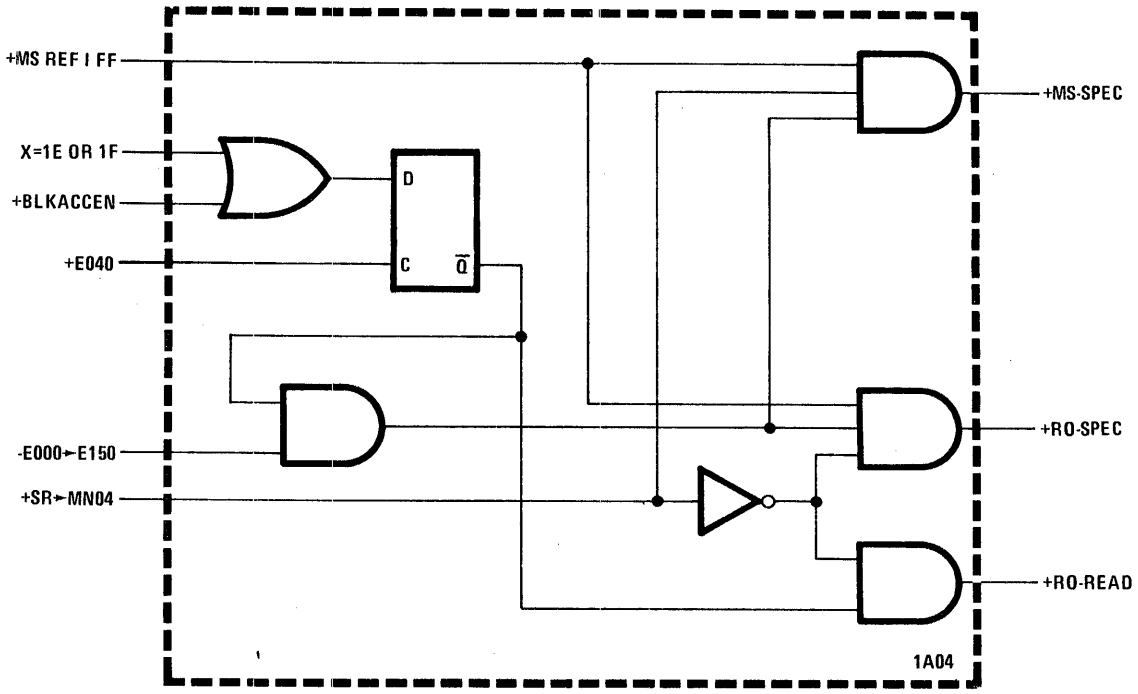


Figure 2-101. RO Reference Signals

MAIN STORAGE

The Main Storage (MS) section of the shared resources stores machine language instructions (MLI's) and data processed by the system. The primary characteristics of MS are as follows:

- Storage Element –
  MOS integrated circuits
- Storage Capacity –
  16 bit words
  8K*to 32K
  64K**
  8 bit bytes
  16K to 64K
  128K**

  (without Register Option)
  (with Register Option)
- Storage Access –
  Random access with an access time of 385 nanoseconds
- Storage Cycle Time –
  900 nanoseconds without ECC
  1000 nanoseconds with ECC

* K=1024
**Capacity of one chassis; the Register Option provides an addressing capability up to 512K words

- Error Detection –
  Parity error detection is standard. Error Correction Code (ECC) is an option.

MS ORGANIZATION

MS Chassis

The MS logic modules are in the three card rows (A, B and C) of chassis 2. Figure 2-102 shows that the card rows are organized as follows:

- Card row A contains the leftmost byte of up to 64K words
- Card row C contains the rightmost byte of up to 64K words
- Card row B is used for the ECC feature when present

Each card row is divided into two zones; each zone has a printed circuit backpanel with common address data and control connections for up to 32K addresses. As an example, the second page of Figure 2-102 lists the pins, signal names and logic mnemonics for Zone A1. The storage elements are on the HH module which is the basic storage module for MS. A 64K word MS with the ECC option uses 48 HH storage modules.
The timing, addressing and control logic is located on three modules in card locations 2B11, 2B12 and 2B13.

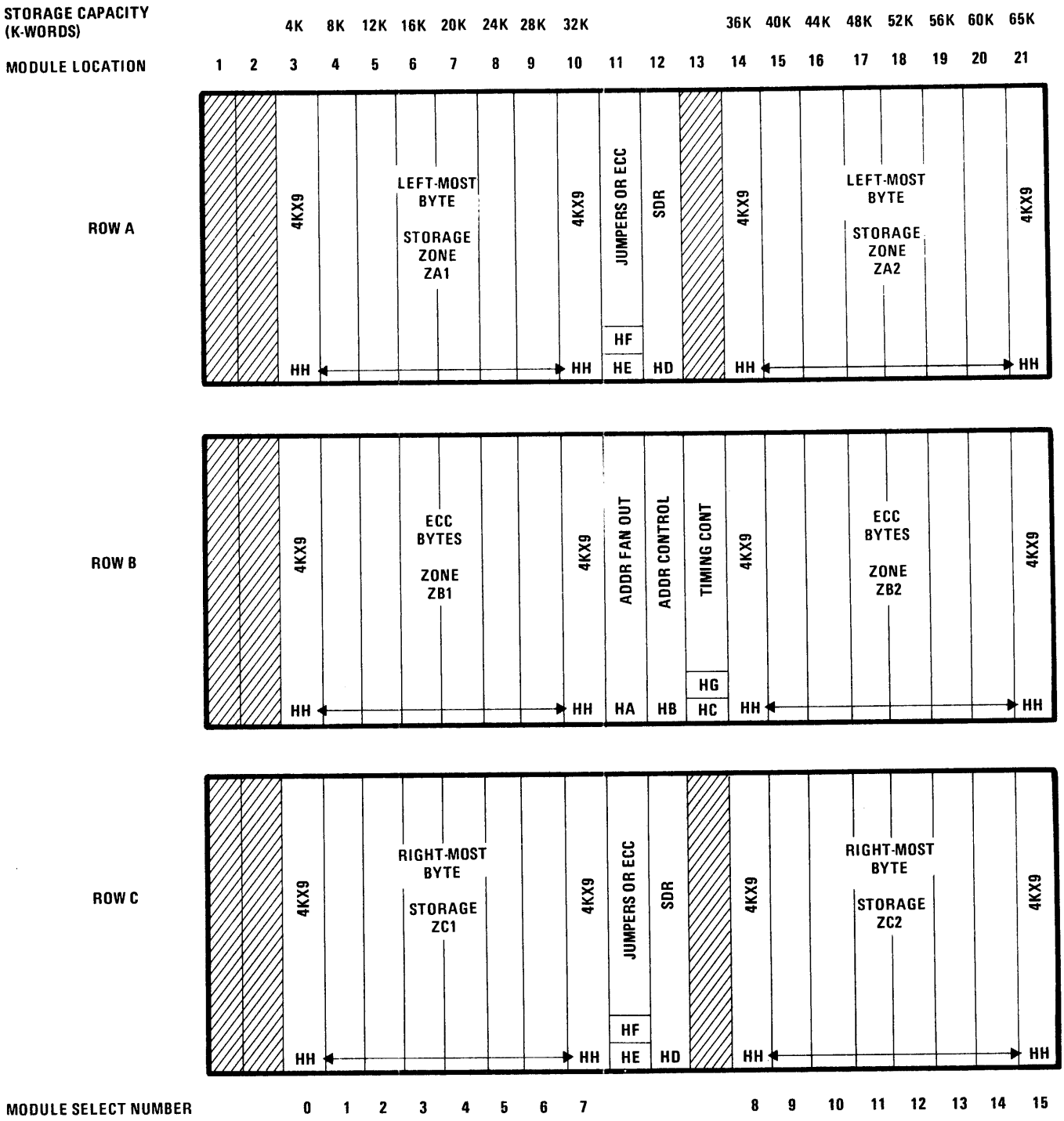


Figure 2-102. Main Storage Chassis

Common Bus Pin	Name	Logic Mnemonic (for Zone AI)
92	Address 14	AD14 →ZA1
96	Address 13	AD13 →ZA1
89	Address 12	AD12 →ZA1
90	Address 11	AD11 →ZA1
94	Address 10	AD10 →ZA1
91	Address 9	AD9 →ZA1
85	Address 8	AD8 →ZA1
82	Address 7	AD7 →ZA1
88	Address 6	AD6 →ZA1
93	Address 5	AD5 →ZA1
86	Column Select 0	CLS 0→ZA1
84	Column Select 1	CLS 1→ZA1
83	Column Select 2	CLS 2→ZA1
80	Column Select 3	CLS 3→ZA1
38	Strobe Time	STRB →ZA1
37	Digit Time	DIFT →ZA1
87	Write Time	WRT →ZA1
76	Column Select Time	CLST →ZA1
81	Address Timing	ADT →ZA1
95	Precharge Time	PRCH→ZA1
24	Data In, Bit 0	D00 →ZA1
25	Data In, Bit 1	D01 →ZA1
26	Data In, Bit 2	D02 →ZA1
27	Data In, Bit 3	D03 →ZA1
31	Data In, Bit 4	D04 →ZA1
32	Data In, Bit 5	D05 →ZA1
33	Data In, Bit 6	D06 →ZA1
34	Data In, Bit 7	D07 →ZA1
78	Data In, Bit 8	D08 →ZA1
22	Data Out, Bit 0	00ZA1→SR
23	Data Out, Bit 1	01ZA1→SR
28	Data Out, Bit 2	02ZA1→SR
29	Data Out, Bit 3	03ZA1→SR
30	Data Out, Bit 4	04ZA1→SR
35	Data Out, Bit 5	05ZA1→SR
36	Data Out, Bit 6	06ZA1→SR
39	Data Out, Bit 7	07ZA1→SR
40	Data Out, Parity 1	P1C1ZA1

Figure 2-102. Main Storage Chassis (Cont)

Except for the data lines and the ECC control lines, all interface signal wiring comes to these three modules.

The data control modules (Storage Data Register and Fan-in) and ECC control for the leftmost byte (bits 0-7, card row A) and the rightmost byte (bits 8-15, card row C) are in locations A12 and C12, respectively. The ECC control modules are in locations A11 and C11. When the ECC option is not installed, these locations have jumper modules in them to route the storage data to the CPU.

Zone Organization

Figure 2-103 is a simplified zone organization diagram showing address and data lines for the eight HH storage modules in a zone. Each HH storage module is a 4096 address by 9 data bit building block; a zone with a full complement of eight HH storage modules stores a data byte (8 data bits plus 1 parity bit) for 32K addresses.

In Figure 2-103, a typical data bit (bit 2) is shown going to all 8 storage modules. Selection of an address occurs basically as follows:

- An HH storage module is selected by a module select (1 of 8).
- A column of storage elements on the module is selected by a column select (1 of 4).
- Using 10 address bits, the 9 storage elements in the selected column decode corresponding storage cells (1 of 1024).

The data bit is then stored in the selected storage element. Using the identical address, the same data bit can be read out of the element.

HH Storage Module

Figure 2-104 is a block diagram of the HH storage module. Thirty-six, 1024 X 1, MOS integrated circuits are arranged on the module in a 4 column by 9 row array. Each of the 9 rows comprises a data bit (0 through 8) of 4096 addresses with 1024, 9 bit addresses in each column.

Address selection is achieved as described in the preceding paragraph, Zone Organization.

Data is controlled by nine digit drivers and nine sense amplifiers — one of each per bit. Because data flow to and from an HH module is a 9 bit block, the control signal for the digit drivers (Digit Timing) is common to the nine digit drivers, and the sense amplifier strobe is common to the nine sense amplifiers. Data to be written into storage is gated with Digit Timing to enable a digit driver. The digit driver provides MOS voltage levels for "1" or "0" to the selected IC. Data read from storage is sensed by a sense amplifier which is then strobed. Data from the sense amplifiers goes into the Storage Data Register (SDR).

Write and precharge drivers provide the read/write and precharge timing pulses required for IC operation. Timing pulses from the storage control logic are gated with module select to activate the drivers.

Basic Storage Element

The basic storage element consits of a 1024 word by 1-bit integrated circuit (IC). This element contains 1024 storage cells, address decoding to select one of them, and read or write controls to read or store the data. The relationship between addressing and data is shown in Figure 2-105, the block diagram of the basic storage element. One feature of this storage element, not shown in Figure 2-105, is that it is a dynamic storage element. This means that the power inside the storage element charges the internal cell capacitors. Using charged cell capacitors is known as dynamic because power is required only during selection. When the cells are not selected, the cells are insulated from other circuitry to prevent the charge from leaking away. Power consumption is minimized because the charge is stored about 2 milliseconds without refreshing (charging).

Each storage element has a matrix of 32 by 32 cells on it as shown in Figure 2-105. The matrix allows each cell to be independently accessed by using the ten decoded address bits with the column select. The initial activation of address, precharge and column select is the same for a read or write operation. Typically the data is read from the cells and then a Read/Write signal is activated, if necessary, to gate an external data bit into the element. The description of the storage element operation can be divided into a read, write, and refresh operation.

For a read operation, the precharge gate is activated simultaneously with the address as shown in Figure 2-106. With pre-charge active, the address stabilizes into the selected row and column as shown in Figure 2-105. When column select is activated, its leading edge gates the contents of 32 storage cells (selected by the X address bits A0 through A4) into 32 refresh amplifiers. The Y address bits A5 through A9 then select 1 of 32 refresh amplifiers for gating one data bit out. At the trailing edge of pre-charge, 32 data bits are gated from the refresh amplifiers back into the cells and one (1 of 32) data bit is gated out. Data out is then valid from end of pre-charge to end of column select as shown in Figure 2-106.

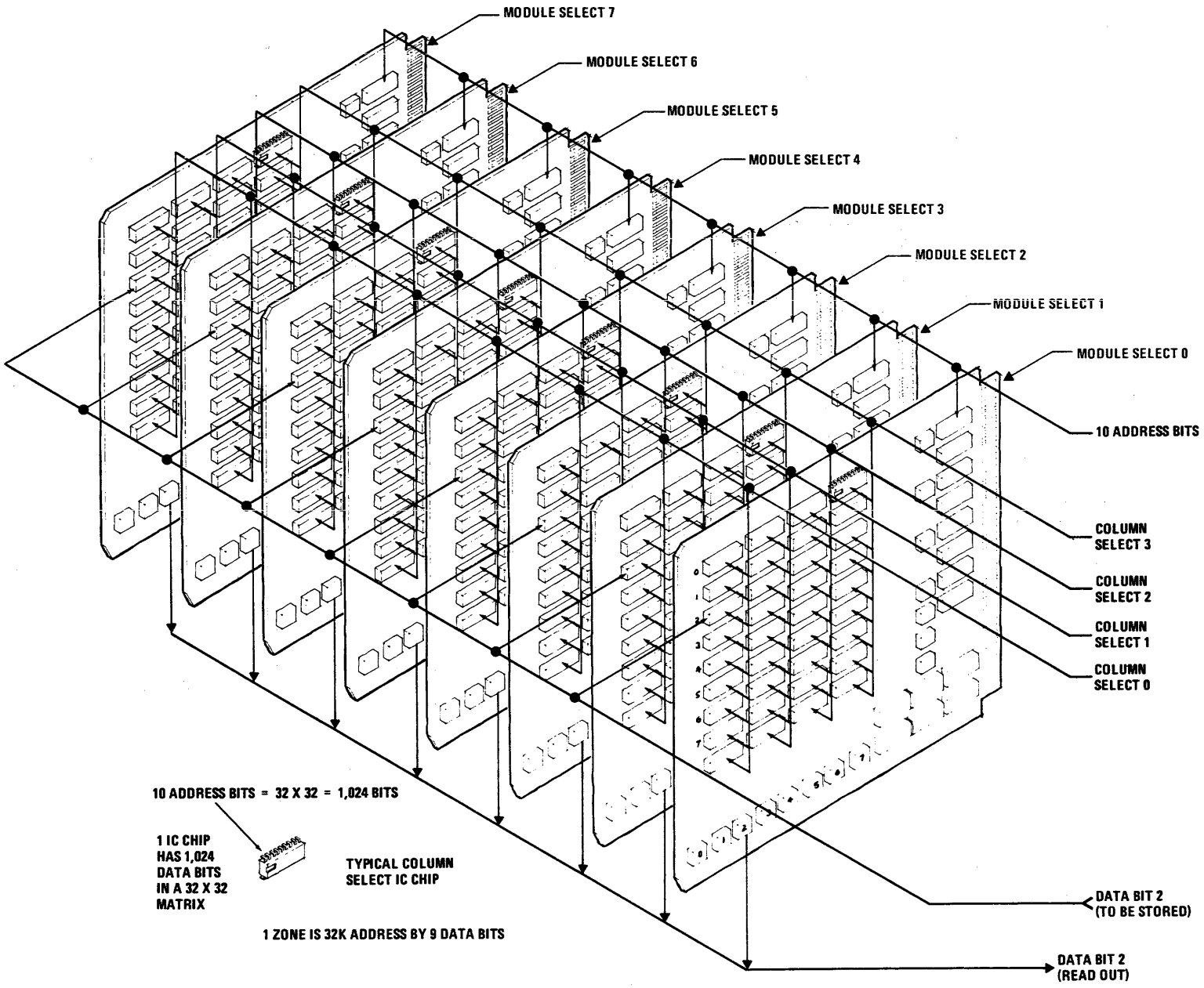


Figure 2-103. Selection of a Bit from MS

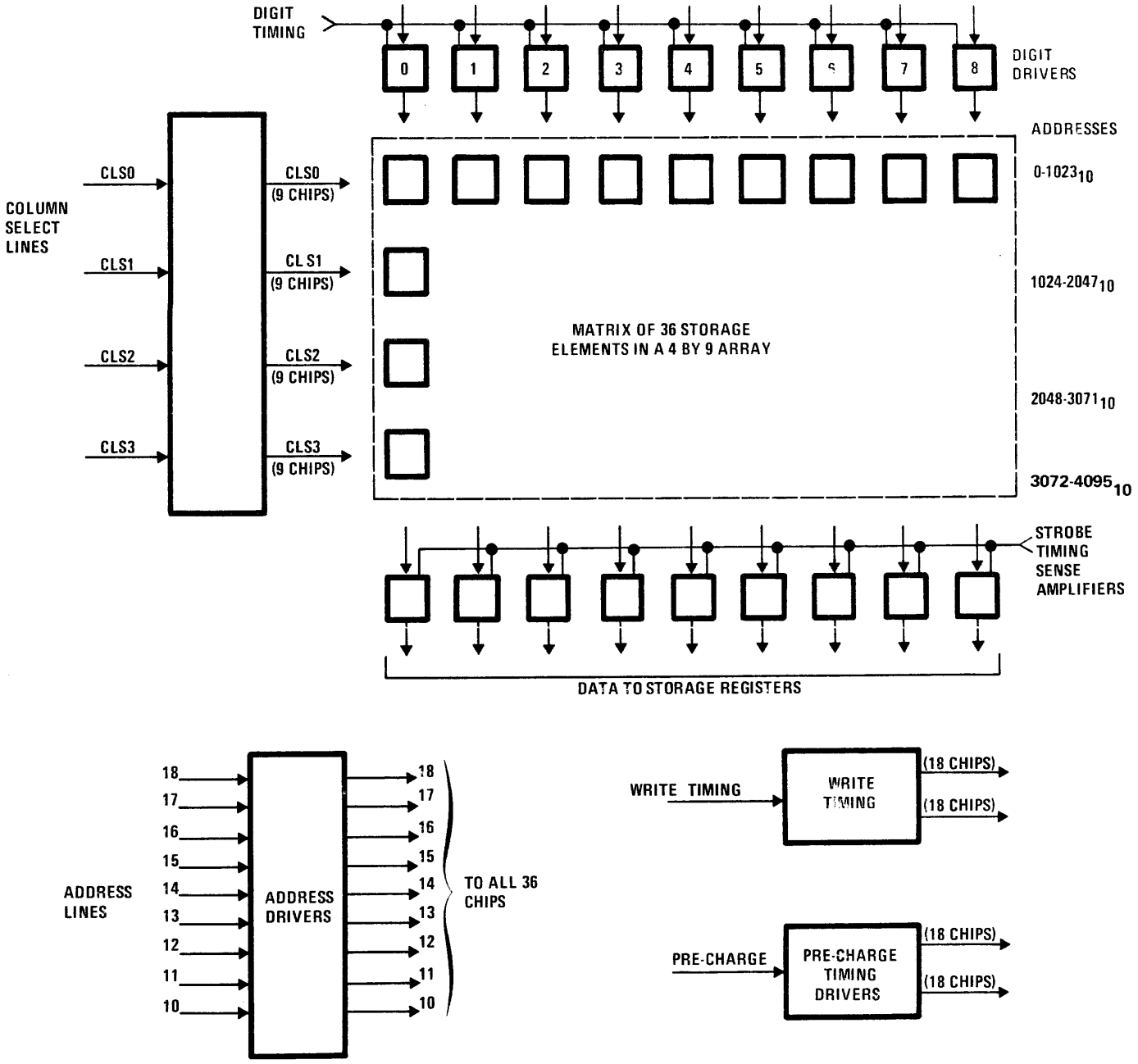


Figure 2-104. MOS Storage Module Block Diagram

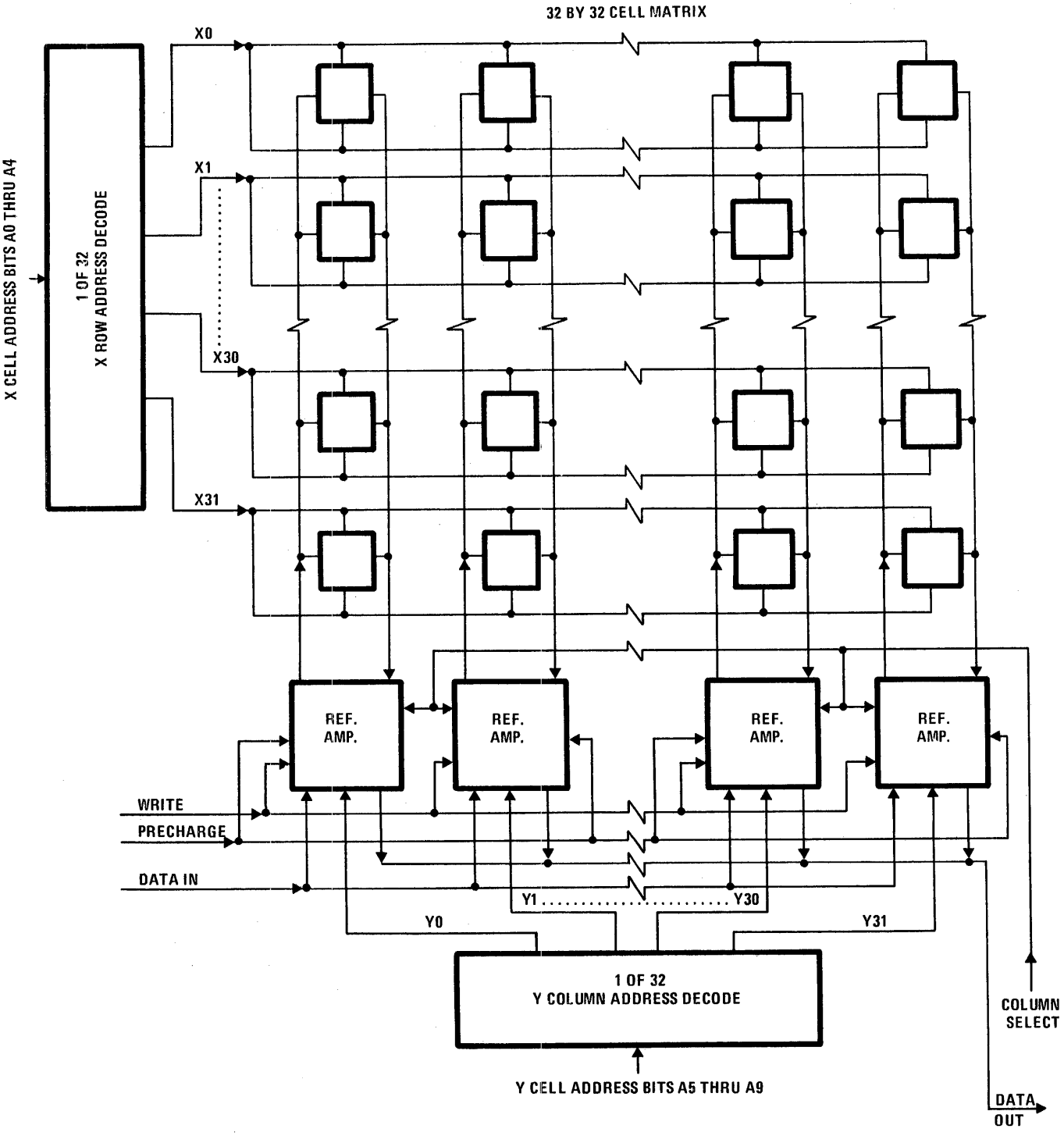
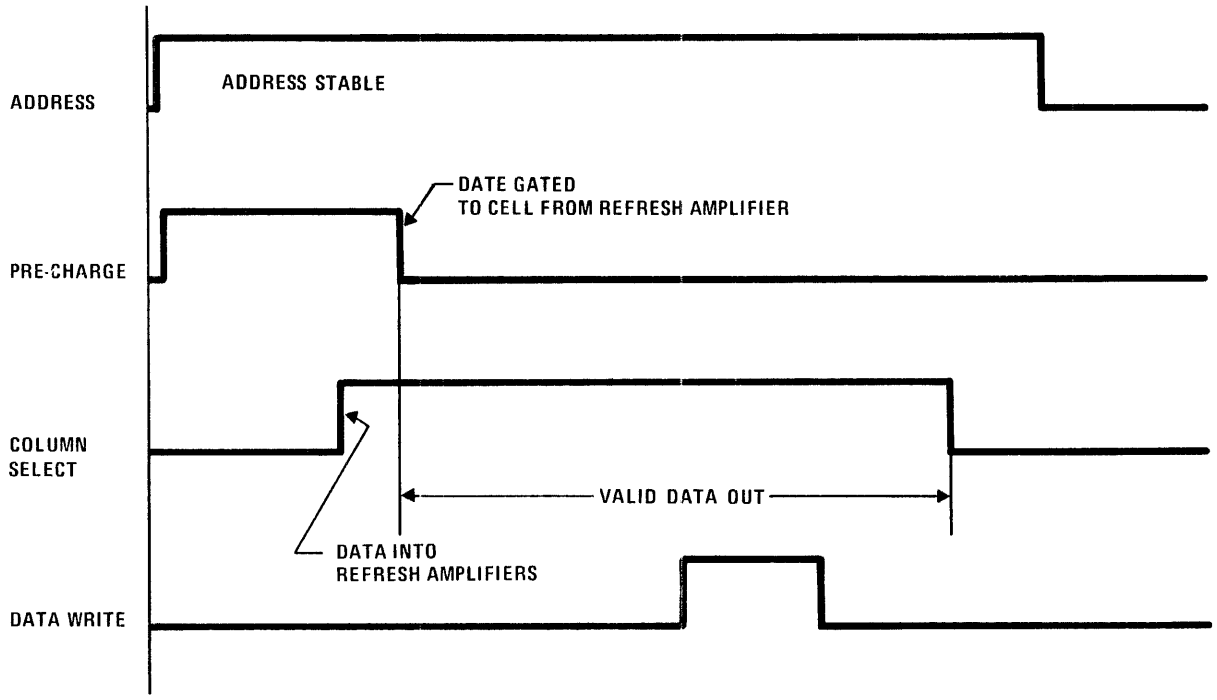


Figure 2-105. 1024 Bit Storage IC Block Diagram



NOTE: FOR INSTRUCTIONAL CLARITY, SIGNAL POLARITIES MAY BE INVERTED.

Figure 2-106. Timing for Storage Element

For a write operation, the initial activation of address, pre-charge and column select is the same. After these signals are stable the data write is activated as shown in Figure 2-106, to gate an external data bit into the storage element. The timing is the same as the read operation except for activating the data write.

For a refresh operation, a read operation is performed. The purpose of the refresh is to periodically recharge the capacitor of the storage cell. A small amount of charge leaks off of these capacitors so that refreshing each memory cell is necessary every 2 milliseconds to recharge the cell's capacitor when they are not accessed. The refresh amplifier, shown in Figure 2-105, recharges 32 memory cells at one time when address inputs bits A14 through A10 are sequenced through the 32 row addresses in the storage element. This refresh addressing and timing is controlled internal to storage. The refresh cycle is identical to the normal read cycle timing and address selection except that addressing from refresh control is sequential.

Block Diagram Description

A block diagram of MS is shown in Figure 2-107. Only the main interface signals such as addressing, data, and pertinent control are shown. Because of symmetry, data is flow-charted for one byte only with either a parity bit (treated as a data bit) or with five of the ECC bits. The other byte of data with either parity or ECC has identical control signals, addressing and control signals. Data flow on the diagram is from left to right while addressing and control flows from the bottom of the page.

The CPU sends sixteen address bits (0-15) to MS. Fifteen bits (0-14) are used by MS to decode 32K addresses; bit 15 is used for byte control (select leftmost byte, select rightmost byte, or select a complete word). For expansion of MS beyond 32K addresses, the CPU sends four additional address extension bits (X0, X1, X2, X3) which provide addressing capability to 512K words.

The refresh operation is a function of MS and occurs when an MS Refresh Request is acknowledged by the CPU (NULL CYCLE). The refresh logic determines a block of 32 addresses to be refreshed and records time so that each address will be refreshed at least every 2 milliseconds.

Control signals from the CPU provide the timing synchronization and initiation of a storage cycle. When a write cycle is required by the CPU, the write controls for the desired byte are activated. Other interface signals provide for refresh control and error recovery interrogation. Within MS, the control signals are

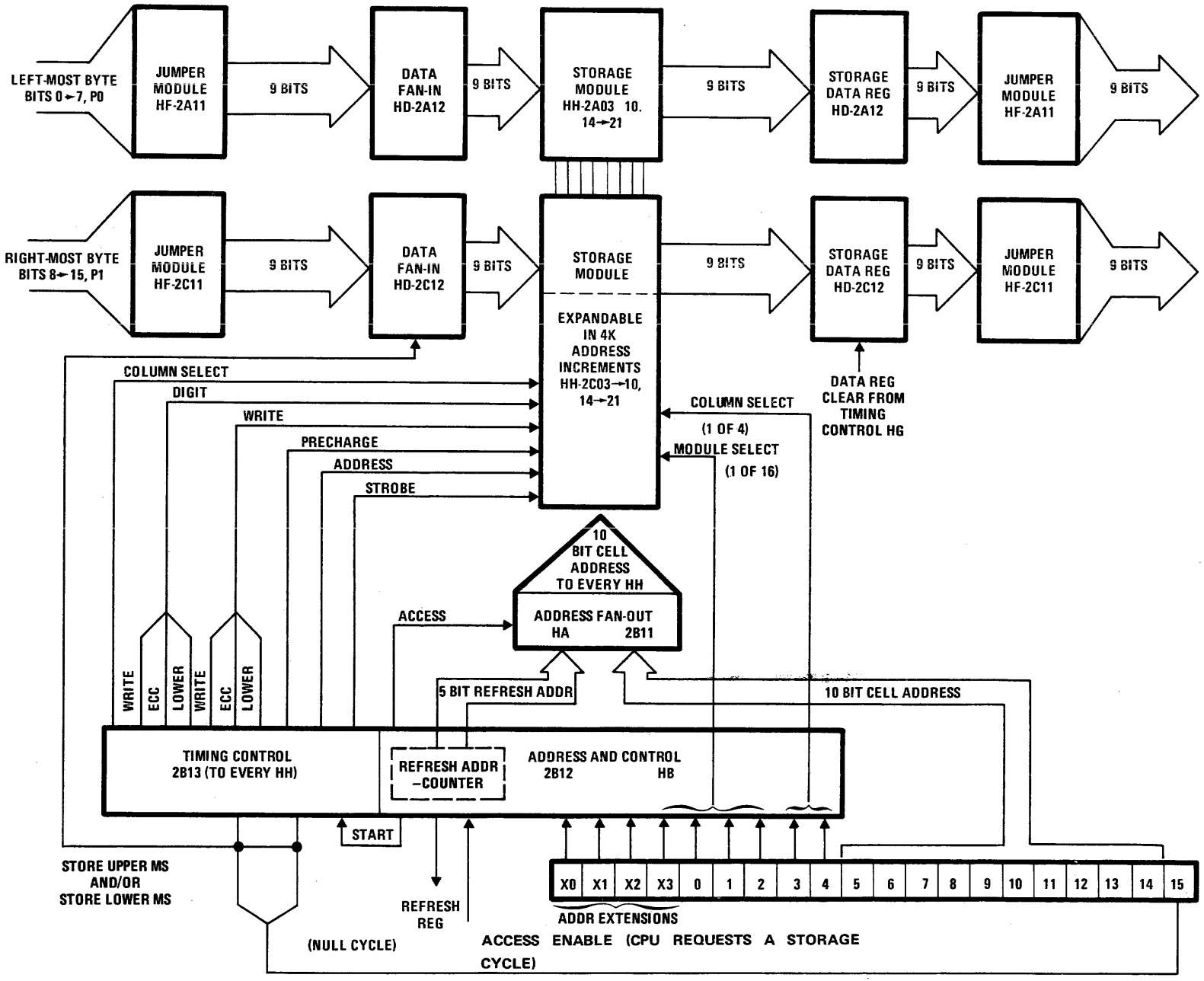


Figure 2-107. Main Storage Block Diagram

primarily timing commands because storage always does a READ/WRITE cycle. One timing/control sequence automatically provides for reading stored data, correcting that data if ECC is present, and writing new data when necessary.

Addressing

The address from the CPU is sent to MS formatted as shown in Figure 2-108. This format allows for features such as Relocation and Protection to be used without altering the address bit numbering scheme. Least significant address bits are to the right end with the extension bits on the left end to provide for expanding storage. Features may alter the expanded storage addressing capacity but will have no effect on the addressing capacity of a storage unit with 32K or less words. Using Figure 2-108, the lowest order addresses start at the left and ascend in order to the right. The 12 least significant address bits are common to all storage modules. Fan-out of these bits is on a zone basis, where each zone is 8 storage modules having identical interconnections on the backpanel, as shown in Figure 2-105 and Figure 2-103. With the 12 common address lines providing decoding for each 4096 word storage module, the 1 of 16 module select from bits 0 through 2 expands the addressing decode to 32K words of storage.

When the address from the CPU is stable, an access enable signal from the CPU provides the initiation of a storage cycle. Since MS does not have an address register, the CPU address must be stable during the entire cycle. When an access enable signal is active, the address from CPU enters storage and then is fanned out. The only exception to gating the address to MS is when a refresh cycle is required. In this case, the access enable signal is blocked by the CPU resulting in the refresh address being gated into address bits 10 through 14. The refresh control holds address bits 9 through 5 at logical zero and blocks address bits 4 through X3 from changing during the refresh cycle.

B-1 Row and Column Chip Addresses; A14 through A5

As shown in Figure 2-109, the address fan-out module distributes the 10 least significant address bits to each of six zones. The 10 CPU address bits are used in conjunction with refresh control.

The 10 least significant address bits are gated into MS from the CPU, without decoding, by an access enable signal. On the address fan-out module, these 10 address bits fan out to all storage modules via the 6 back panels as previously shown in Figure 2-103. On the storage module, the ten bits are gated onto the board by ANDing address timing with board select. Each address line has a discrete driver to drive all 36 storage elements. The address lines are equally divided between the storage element column and row decode to make the 32 by 32 matrix selection in the element. The address is then decoded within each element.

When MS requests a refresh cycle, the access enable is blocked by the CPU and the refresh address is used instead. The 1 of 32 refresh addresses controls address bits 10 through 14. Because of the intrinsic storage element address decoding, the 32 row refresh addresses will refresh 32 column addresses per element. Since all elements are selected during refresh, all addresses for all data bits in storage will therefore be refreshed using only 32 refresh addresses that are controlled by address bits 10 through 14.

B-2 Column Select Address Bits A4, A3

As shown in Figure 2-110, the most significant address bits are decoded or controlled on this board. The three most significant address bits (X0 through X3) determine the out-of-range. The four column selects are decoded from two address bits and distributed to each zone. Four address bits are decoded into 1 of 16 module selects then respectively distributed to each row. Each module select controls the left-most byte, right-most byte, and ECC modules. Refresh control determines how often and what address the refresh cycle needs.

The column select address bits 3 and 4 are also common to all storage elements. On the address control board, address bits 3 and 4 are gated from the computer by inactive Refresh. The decoding of column select (CLS) fans out 4 column selects that are common to all storage modules. As column select enters the storage module, it is gated with timing and board select signals. On the storage module, CLS provides the addressing expansion from 1024 to 4096 addresses or physically from 1 to 4 storage elements (per data bit). The resultant column select(s) activate the *chip enable* pins for 9 parallel storage elements.

Inactive Refresh is normally low throughout the entire timing cycle. When Refresh is requested, the computer address is blocked and, by using another inverter, all column select outputs are activated.

B-3 Module Select Addresses; A3 through A0

The most significant address bits, 0 through 3, decode which storage module board is selected. Selection of a module really is a selection of three modules: one module corresponding to the left-most byte; the second module corresponding to the right-most byte; and, when ECC is used, the third module corresponding to the check bits.

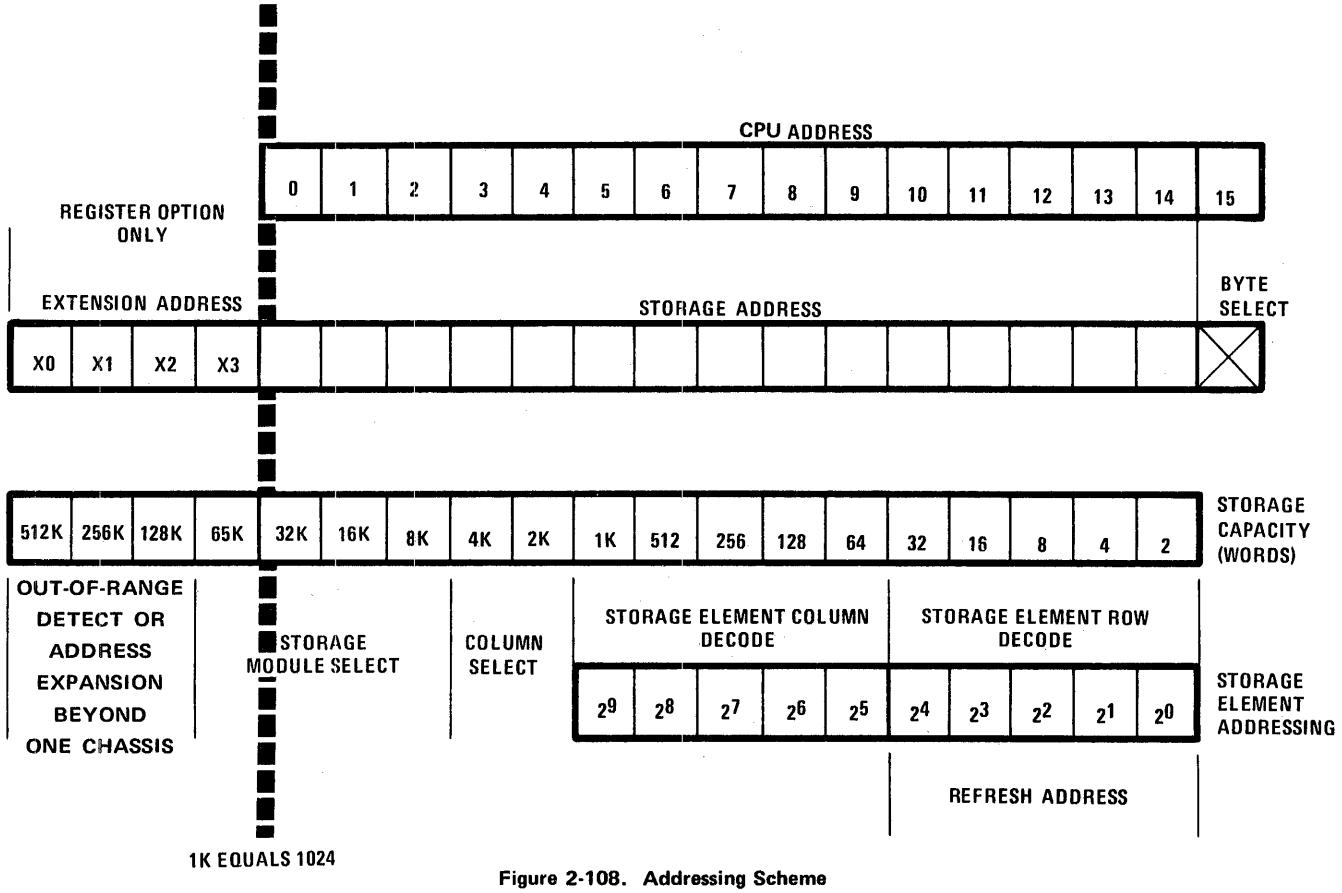


Figure 2-108. Addressing Scheme

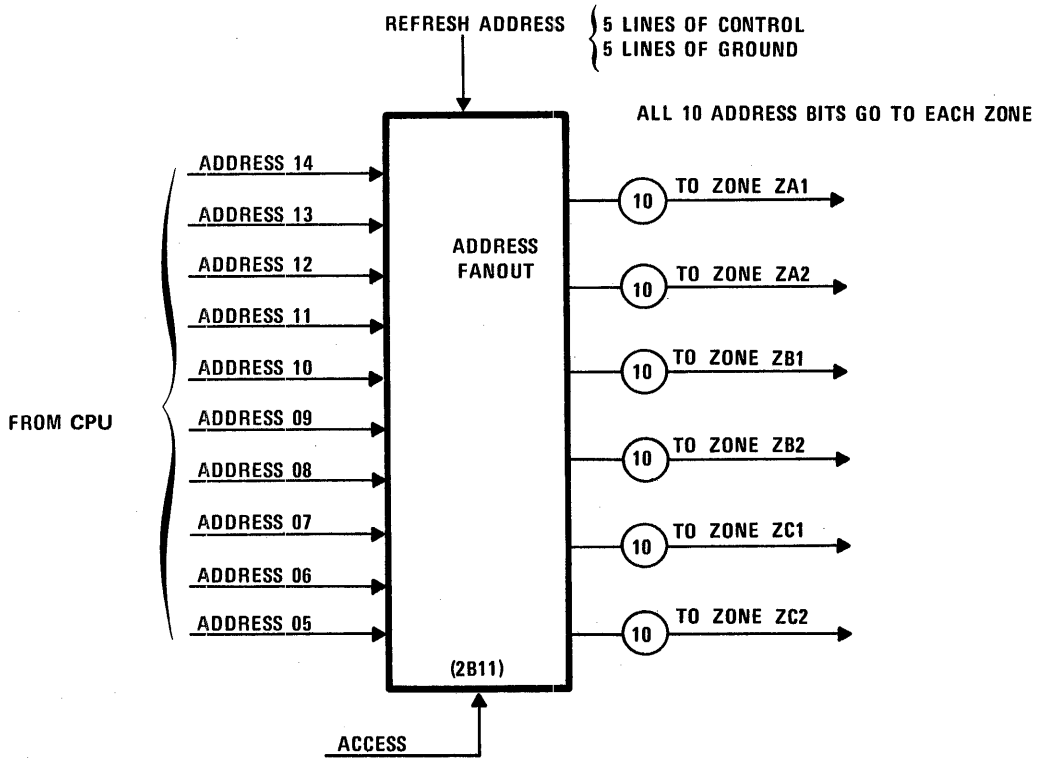


Figure 2-109. Address Fanout

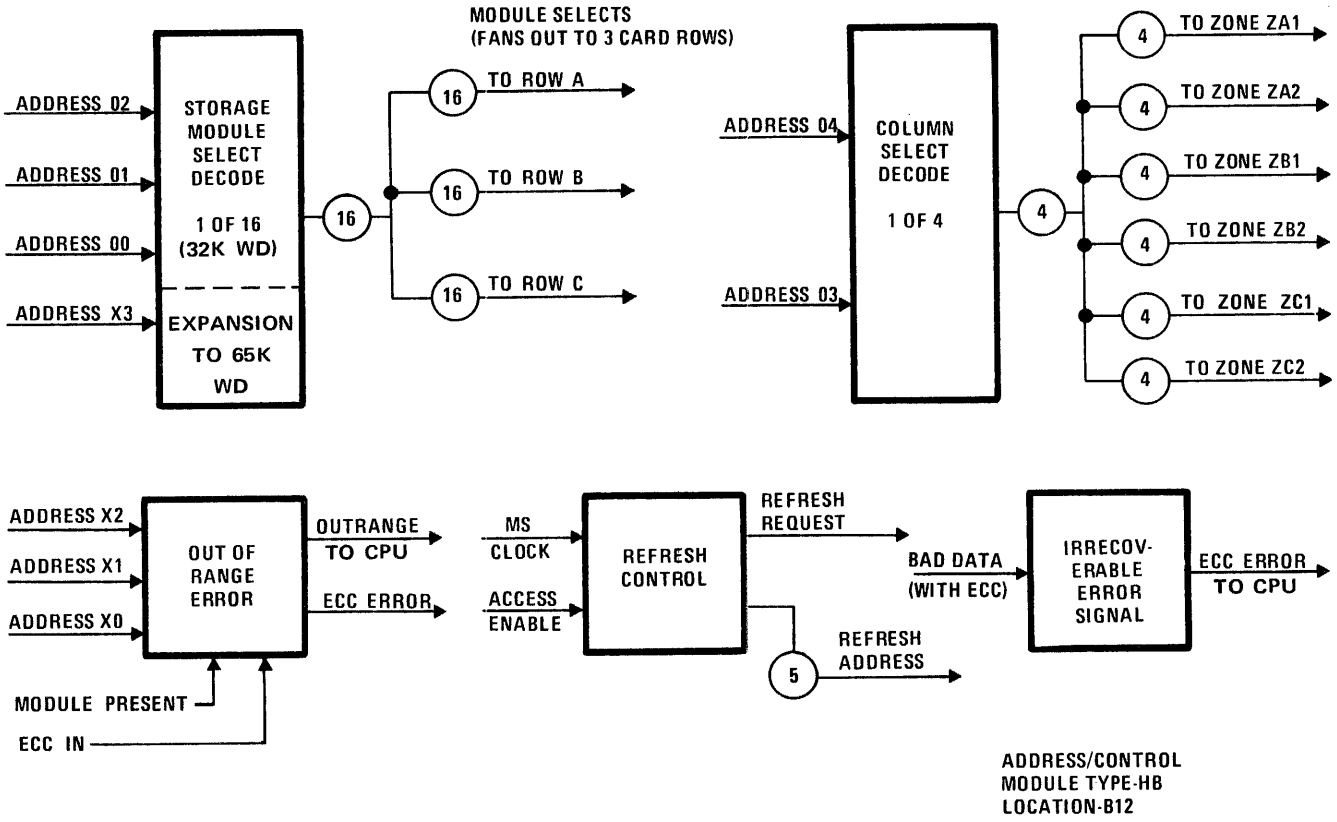


Figure 2-110. Address Control

See Figure 2-108 for the address organization. On the address control module, address bits 0 through 3 are decoded into a 1 of 16 storage module selects so that a storage capacity of 64K addresses can be randomly accessed. The most significant bit, bit 0, determines physically which 32K of addresses are selected — when active the left half of MS is used; inactive, the right side. Again, when Refresh is active, the address from CPU is over-ridden so that all storage modules are selected.

DATA CONTROL

Storage Data Register

The data control logic uses digit drivers, sense amplifiers and the Storage Data register (SDR). Figure 2-111 illustrates a simplified data loop, for controlling the data flow from the CPU to the storage element for writing; and from the storage element to the CPU for reading.

New data to be written into storage uses a digit driver as shown in Figure 2-111. Data read from the storage element is sensed by the sense amplifier and temporarily stored in the Storage Data register (SDR). From the SDR, the data is sent back to the CPU.

As shown in Figure 2-112, the SDR is 18 data bits long, contained on two modules of 9 bits each. One module is located in row A and stores the left-most byte plus the corresponding parity bit. The other module is located in a corresponding location in row C and stores the right-most byte and parity bit. The data fan-in logic, which gates one byte of data from the CPU to the storage elements, is organized in a similar fashion.

When error correction is added, generated check bits have an equivalent set of drivers, storage, sense amplifiers and SDR's. Instead of sending the check bits back to the CPU when they are read, error correction logic corrects erring data or check bits. The corrected data without check bits is then sent back to the CPU. There are two advantages in having an equivalent set of hardware for check bits. First, the error correction feature can be added primarily using already existing card types. Second, the SDR for check bits can be used for special storage cycles that can interrogate check bits for maintenance purposes. The intricacies of generating check bits and performing error correction is discussed later.

Sense Bias

Sense bias is distributed to each storage module from the logic +5 volts connected to 6 connector pins on the HB board. Using the HB board to disperse sense bias to the 6 zones allows future changes to sense bias such as a possible maintenance switch, a regulated special voltage or some other convenient logic voltage. Data read from 1 of

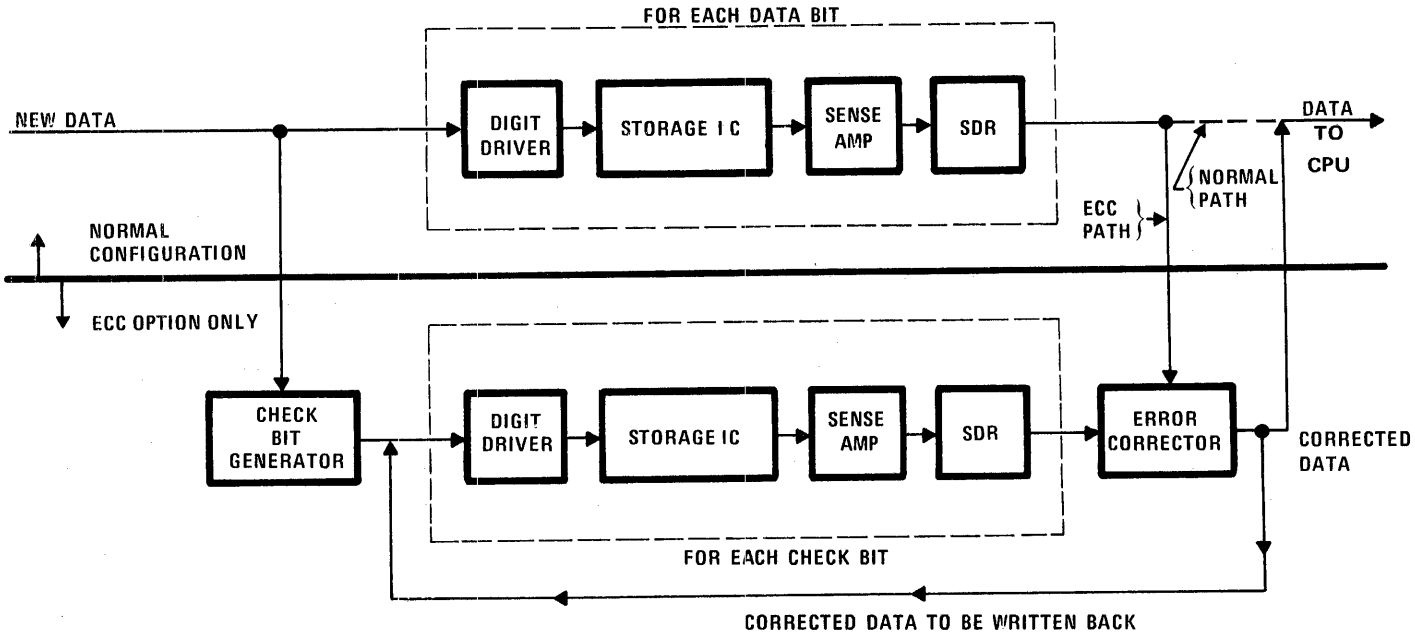


Figure 2-111. Simplified Data Flow for One Data Bit

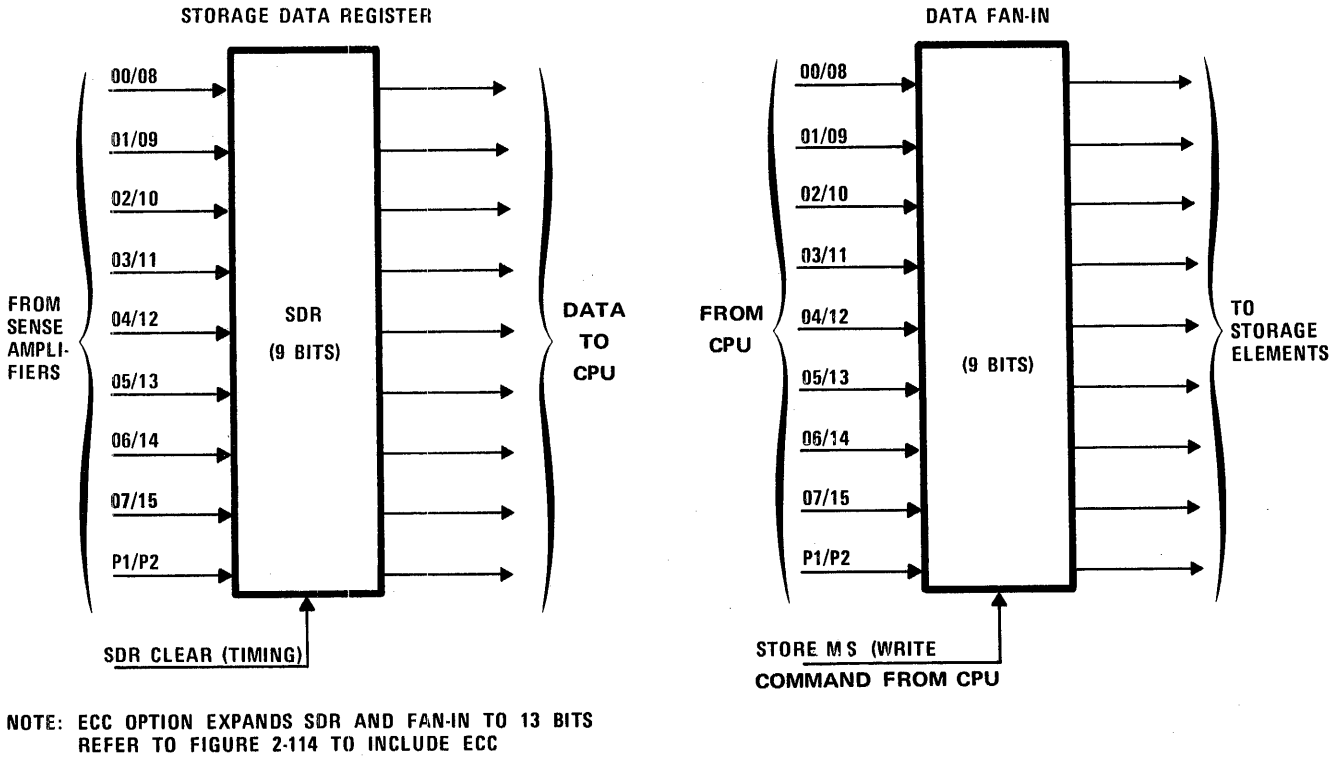


Figure 2-112. Storage Data Register (No ECC)

4 selected elements is compared against a threshold voltage in the sense amplifier. The threshold voltage is the sense amplifier bias voltage which provides the DC reference for determining a one or zero.

REFRESH CONTROL

An inherent requirement of MOS-type storage is recharging or refreshing the capacitive cells within the element. The refresh cycle consumes an entire major timing cycle so that the CPU cannot address storage during refresh. Due to priorities when transferring high speed data, some control between the CPU and MS must be established. Logic internal to MS controls what address is to be refreshed next and when this address will be required. This control uses an incrementing counter to determine the refresh address and cascade counter to count the number of clock cycles. The cascaded counter converts clock pulses into a time-out calculated to refresh 32 addresses every two milliseconds (2 ms). From the previous description of the storage element, the 32 by 32 matrix of storage cells have 32 refresh amplifiers along one axis so that by sequencing the 32 addresses along the other axis, all 1024 storage cells of the element can be refreshed 32 cells at a time.

To accomplish the refresh with a minimum of interference, only one consecutive major cycle is taken from the CPU. The CPU can then continue using MS until another refresh request is activated. If 32 addresses must be refreshed in 2 milliseconds, the refresh request should be activated every 62.5 microseconds (200μs ÷ 32 = 62.5 μs). Therefore, if the processor accessed storage every 900 ns, which is the MS reference cycle time without ECC, a counter incrementing up to about 71 accesses would be enough to refresh storage in time. Practically, however, the processor will run at a worst-case rate of 1.2 microseconds. Using 1.2 microseconds as worst case, the counter would then count up to 52 before a refresh request should be made.

The clock pulse from the CPU is present 200 nanoseconds before the storage can be accessed as shown in Figure 2-114. The early clock allows the timing control logic to initialize in anticipation of an Access Enable signal. The clock pulse's leading edge is used to form a 50 nanosecond wide Start pulse by using a delay line as shown in the logic. This start pulse initiates the timing control and increments the cycle counting up to 52. Since the starting of timing is discussed under the paragraph titled Timing, only the refresh control and its timing is covered here.

As shown in Figure 2-113, when the cycle counter has reached a count of 52, its output updates the refresh address and sets the Refresh Request flip-flop. First, the refresh address counter updates only once each refresh cycle keeping in mind that, due to storage element geometry, one row address will refresh 32 column addresses. The address counter is always incremented and never cleared so that all 32 addresses are cyclically sequenced. Second, the need for refresh is sent to the CPU via the Refresh Request flip-flop. In the CPU, the request for the next major cycle to be a null state is determined in the priority sequences. If the refresh request is honored, the Access Enable to MS is blocked. About 20 nanoseconds before a storage cycle begins, the access timing interrogates the Access Enable state (on Figures 2-117 and 2-118, this is 180 nanoseconds after the clock). If Access Enable is active, an MS reference is imminent and preempts MS's desire to refresh. If Access Enable is blocked at access time, the Refresh Granted flip-flop sets to gate the refresh address instead of a CPU address, and it also resets the cycle counter. The Refresh Granted flip-flop implies that storage can now initiate a refresh cycle and and therefore drops (clear) its Refresh Request flip-flops. As soon as internal storage timing programs, the Refresh Request Clear will be activated to clear the Refresh Request flip-flop. Timing is shown in Figure 2-114.

The Refresh Request Clear provides the timing for another special feature of refresh control; the Refresh Time-out. There are three conditions that must be satisfied before time-out occurs:

1. There must be an active Refresh Request to the CPU.

2. The request for refresh has been ignored by the CPU for 3 major cycles. The cycle counter is therefore at count of 3.

3. Timing from the Refresh Request Clear is active once during each major cycle.

INTERFACE CONTROL SIGNALS

All of the interface signals between the ALU and MS are shown in Table 2-9. The storage initiation, addressing and data functions are not discussed here. Interface signals for the ECC feature (module type HE) differ when the jumper module (type HF) replaces the ECC module. Without ECC, the jumper module routes data bits directly to and from MS because the data does not have to be coded for error correction. The detailed description of error/recovery and write control functions are as follows:

1. Out-of-Range

   OUTRANGE on the HB module is used by the CPU to detect a missing HH module whether or not the module is missing by intent (the upper

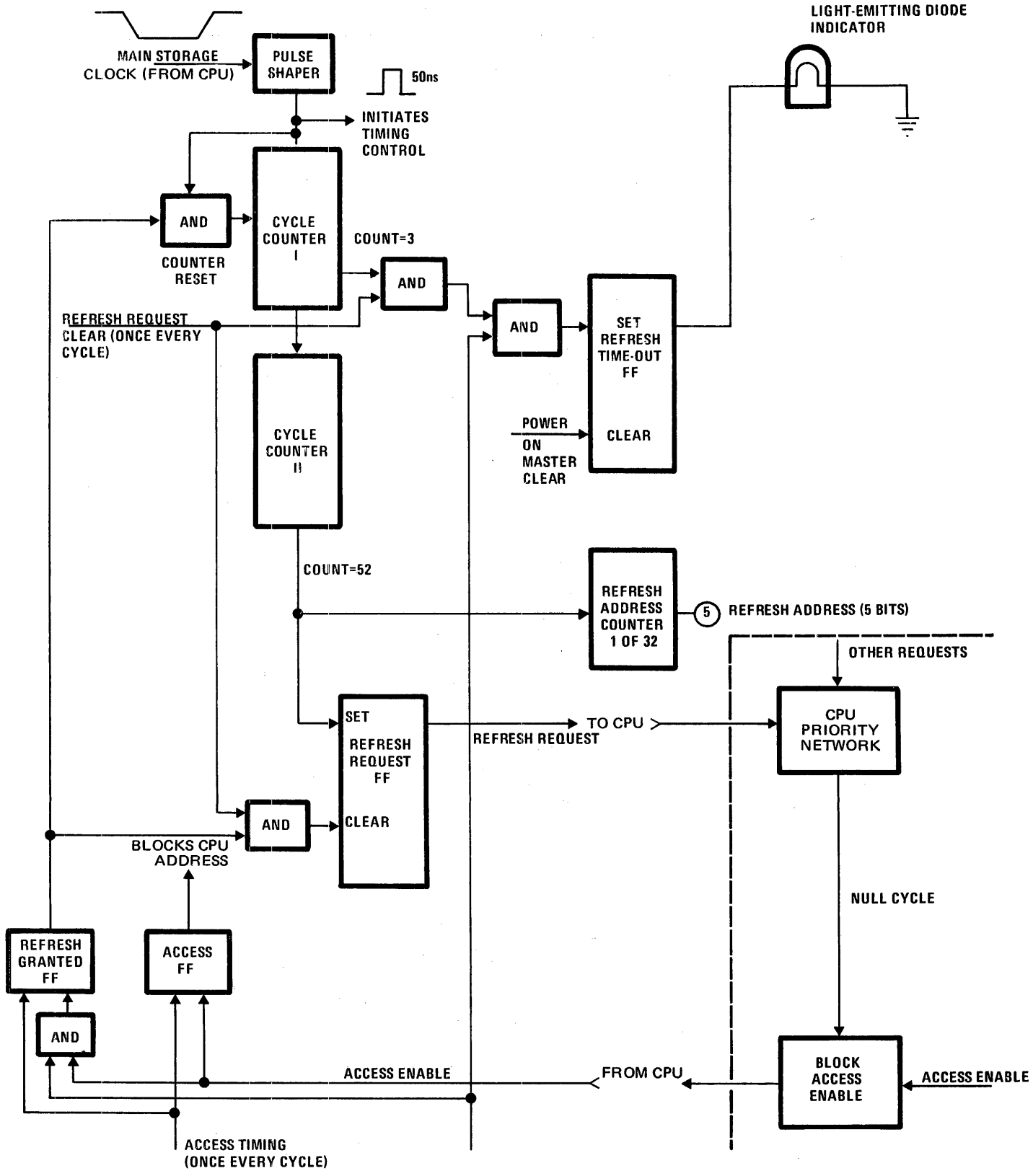


Figure 2-113. Refresh Control Block Diagram

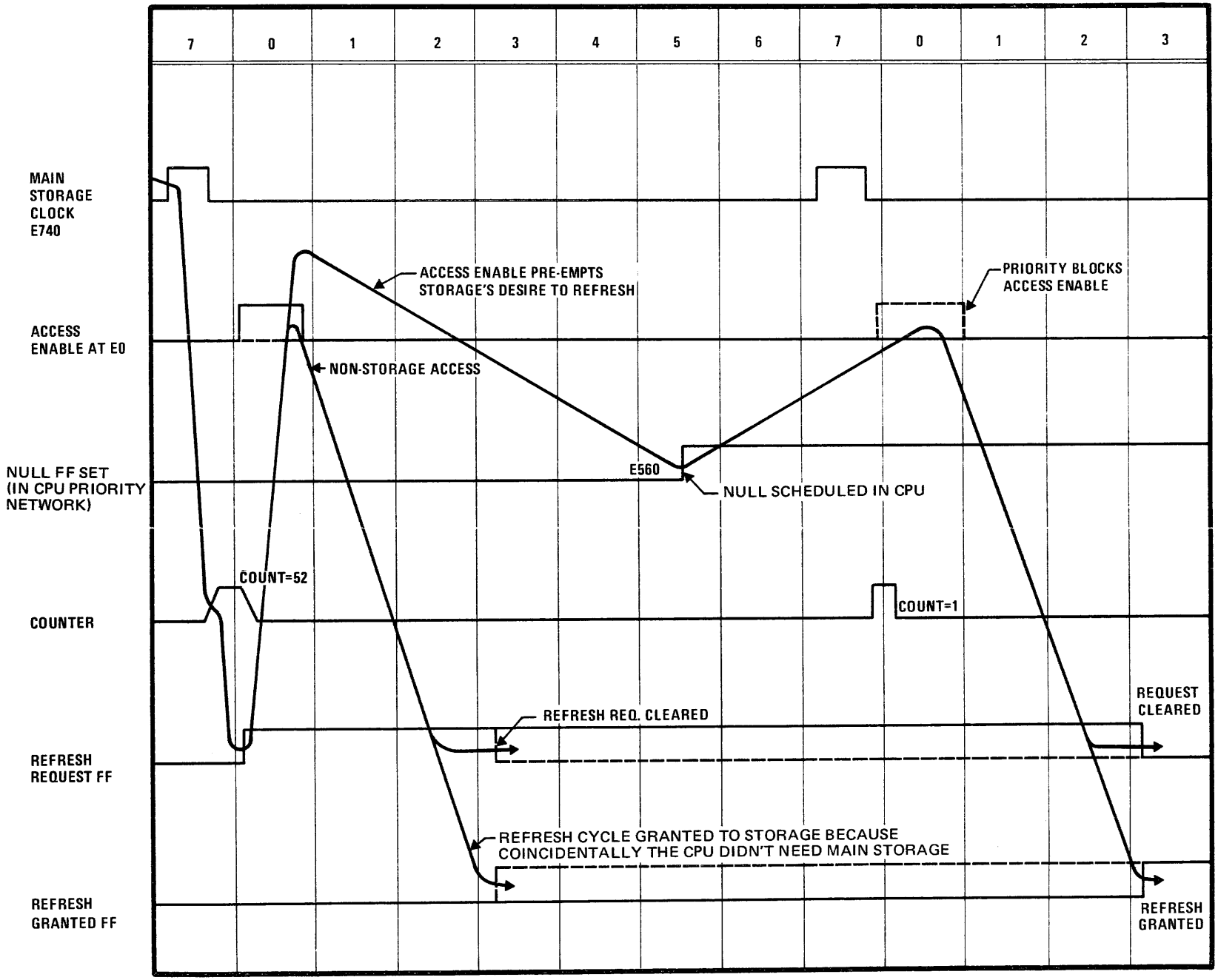


Figure 2-114. Refresh Control Interface Timing

Table 2-9. Interface Signals References

Function	Signal	Can be Scoped on Pin:	
Storage Initiation	Main Storage Clock	B12-19	
	Access Enable	B12-25	
	Refresh Req. (To CTU)	B12-31	
Addressing	Address bits 00-18		
	Address 14	B11-16	
	13	B11-13	
	12	B11-33	
	11	B11-19	
	10	B11-50	
	09	B11-37	
	8	B11-70	
	7	B11-57	
	6	B11-86	
	5	B11-73	
	4	B12-34	
	3	B12-39	
	2	B12-26	
	1	B12-94	
	0	B12-95	
	X3	B12-75	
	X2	B12-44	
	X1	B12-45	
	X0	B12-58	
Data	Data Bits 00-15	To	From
	00	A11-72	A11-27
	01	A11-73	A11-26
	02	A11-74	A11-22
	03	A11-71	A11-24
	04	A11-70	A11-4
	05	A11-69	A11-6
	06	A11-56	A11-10
	07	A11-61	A11-11
	P1	A11-47	A11-28
	08	C11-72	C11-27
	09	C11-73	C11-26
	10	C11-74	C11-22
	11	C11-71	C11-24
	12	C11-70	C11-4
	13	C11-69	C11-6
	14	C11-56	C11-10
	15	C11-61	C11-11
	P2	C11-47	C11-28
Error/Recovery	ECC Error	B12-37	
	ECC In, Lwr,	A11-95	
	Upper	C11-95	
	ECC Present Upper,	C111-9C	
	Lwr	A11-96	
	Out-of-Range	B12-32	
	Special (MS-SPEC)	B13-53	
	SLX1-ECC	B13-69	
	SL1X-ECC	B13-68	
Write Controls	Store Upper Byte	B13-66	
	Store Lower Byte	B13-67	

addressing limit of storage) or by mistake (maintenance man has removed a board within contiguous storage). When an address is decoded in MS, the Board Select is active for only the set of modules used for that word; the left-most byte, right-most byte and if ECC is used, the ECC module. Board Select in turn generates a Board Present signal to detect the missing HH module. Board Present will be active from only 1 of 16 modules in either card rows A, B, or C. This singly active signal (1 of 16) allows the outputs of all 8 modules, in each of the two zones, to be connected in common. Each zone is wired to the common point on the input of the out-of-range circuit. Additionally, when ECC is not present, the jumper module that replaces ECC also provides the disabling of the ECC gates on the out-of-range circuit. An added feature of out-of-range is the detection of the most significant CPU relocation bits. If any of these bits are active, storage reports this as out-of-range.

2. ECC Error

ECC Error on the HB module detects the Bad Data signal from either the left-most or right-most byte. From Figure 2-121, when the bad data line is active an irrecoverable error has occurred. The Bad Data signal is sent back to the CPU as ECC Error to force the CPU into a trap routine that will be software controlled. Basically, the trap routine will read the error log in storage in an attempt to decipher what happened and possibly recover from the multiple errors.

3. Parity (No ECC)

The parity bit is generated and detected in the CPU. It is shown in the storage logic diagrams as either P0 or P1. Since the CPU generates the parity bit on a byte basis, MS stores the parity bits as if they were data bits — MS cannot differentiate between data and parity bits.

4. Jumper Module (No ECC)

The jumper module disables ECC functions on other modules as follows:

a. ECC In — disables the out-of-range logic from detecting storage modules that are not plugged into row B — the ECC row. The out-of-range logic on the HB module will still detect out-of-range on the most significant address bits.

b. ECC PRES — Grounds line to CPU indicating that ECC is not present. When ECC is used, the line is permanently held active.

c. FIXBIT — Grounds and disables write control from ECC. Only the CPU can then initiate a write operation. With ECC, the corrected data will be written back into MS.

TIMING

As shown in Figure 2-115, the timing control is primarily single-shots for timing. For ECC, the error log control logic is on this module. To allow ample set-up time in the single-shots used for timing, the clock pulse, precedes the Access Enable. The clock pulse used to initiate storage timing is generated by the CPU as part of its standard timing, so that it appears precisely related to all CPU events. For reference, the timing of MS will be relative to the leading edge of the clock pulse.

There are two timing diagrams used for storage depending on whether or not the ECC feature is used. Timing is shown in Figure 2-118 (with ECC) and Figure 2-117 (no ECC). For reference on these figures, the abscissa has two timing scales. For convenience, the timing is referenced to the clock pulse so that scoping waveforms is easier. When referring to internal memory operations, the address stable time (200 nanoseconds after the clock) starts the true initiation of a storage access.

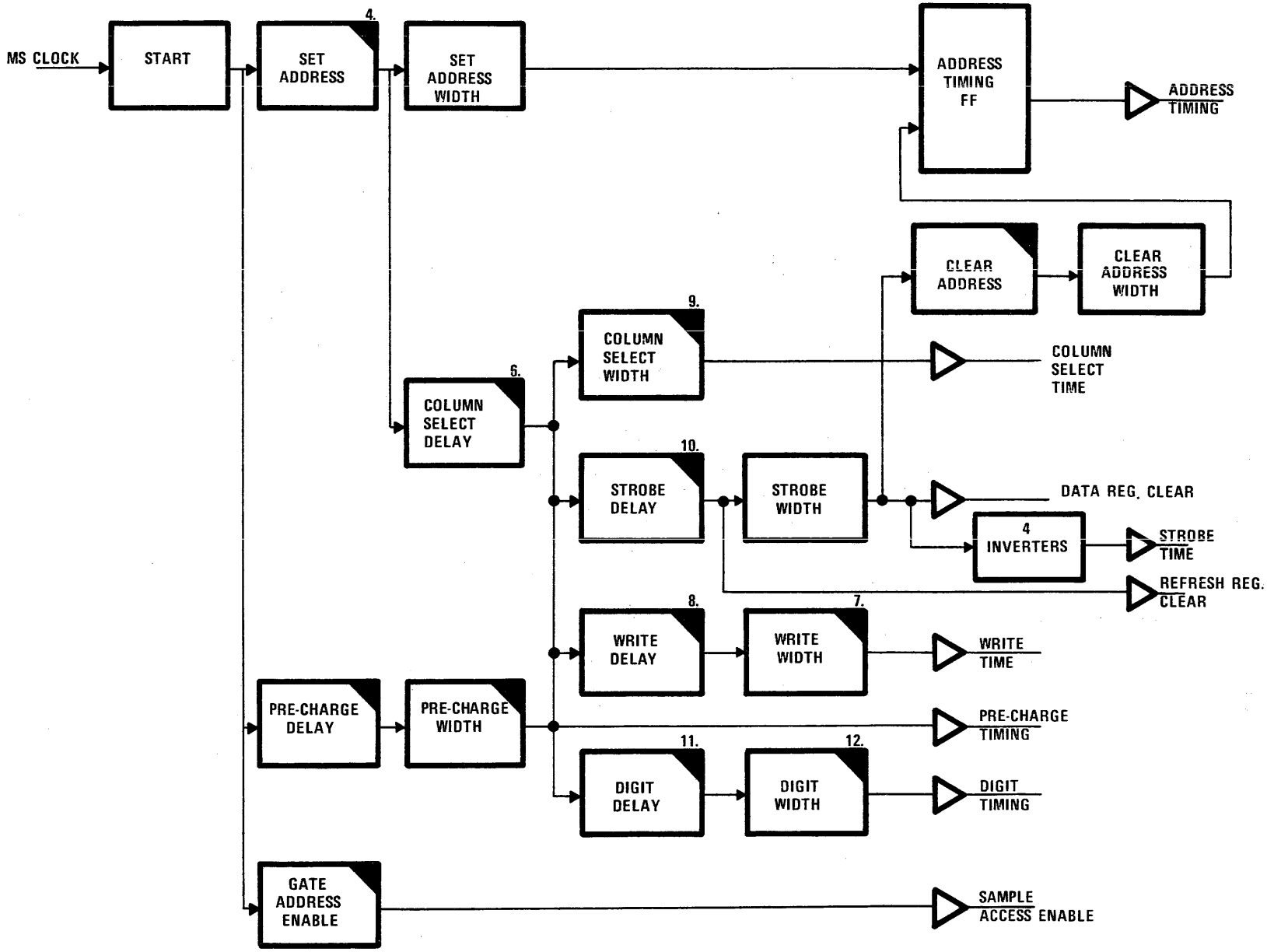
Figure 2-115 shows the relationship between the timing adjustment of each single-shot. The flagged corners of most of the blocks mean that the timing is adjustable. Next to the flag is a number corresponding to the physical location of the potentiometer as shown in Figure 2-116. Because of the intricate dependence of one adjustment to others, as shown in Figure 2-115, timing is adjusted at the factory. For example, adjusting the Column Select Delay also affects Strobe, Write and Digit timing. The following is a description of each significant timing signal shown in Figure 2-117 and Figure 2-118.

Main Storage Clock

This pulse occurs 180 nanoseconds before storage activates Access Enable. The clock conditions timing in anticipation of a storage cycle.

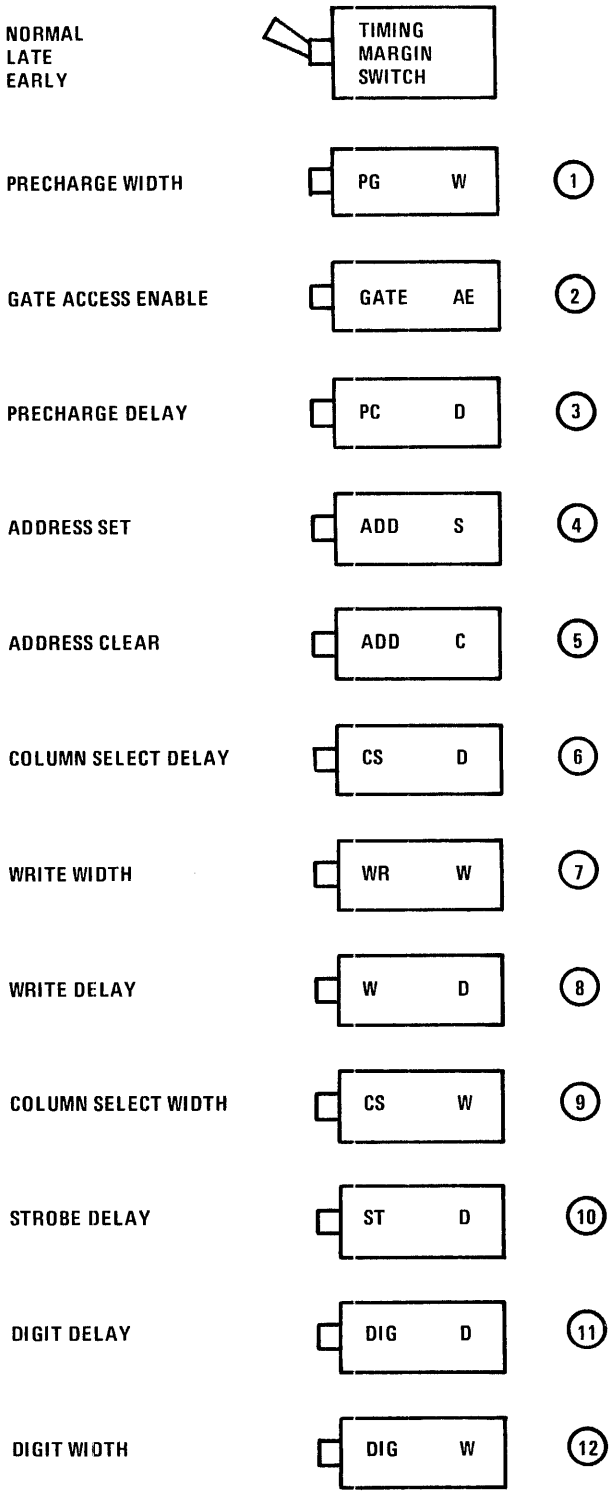
Address Time FF

This flip-flop sets when the address is stable. Holds CPU address active during entire storage cycle. It is cleared by the time-out of strobe timing.



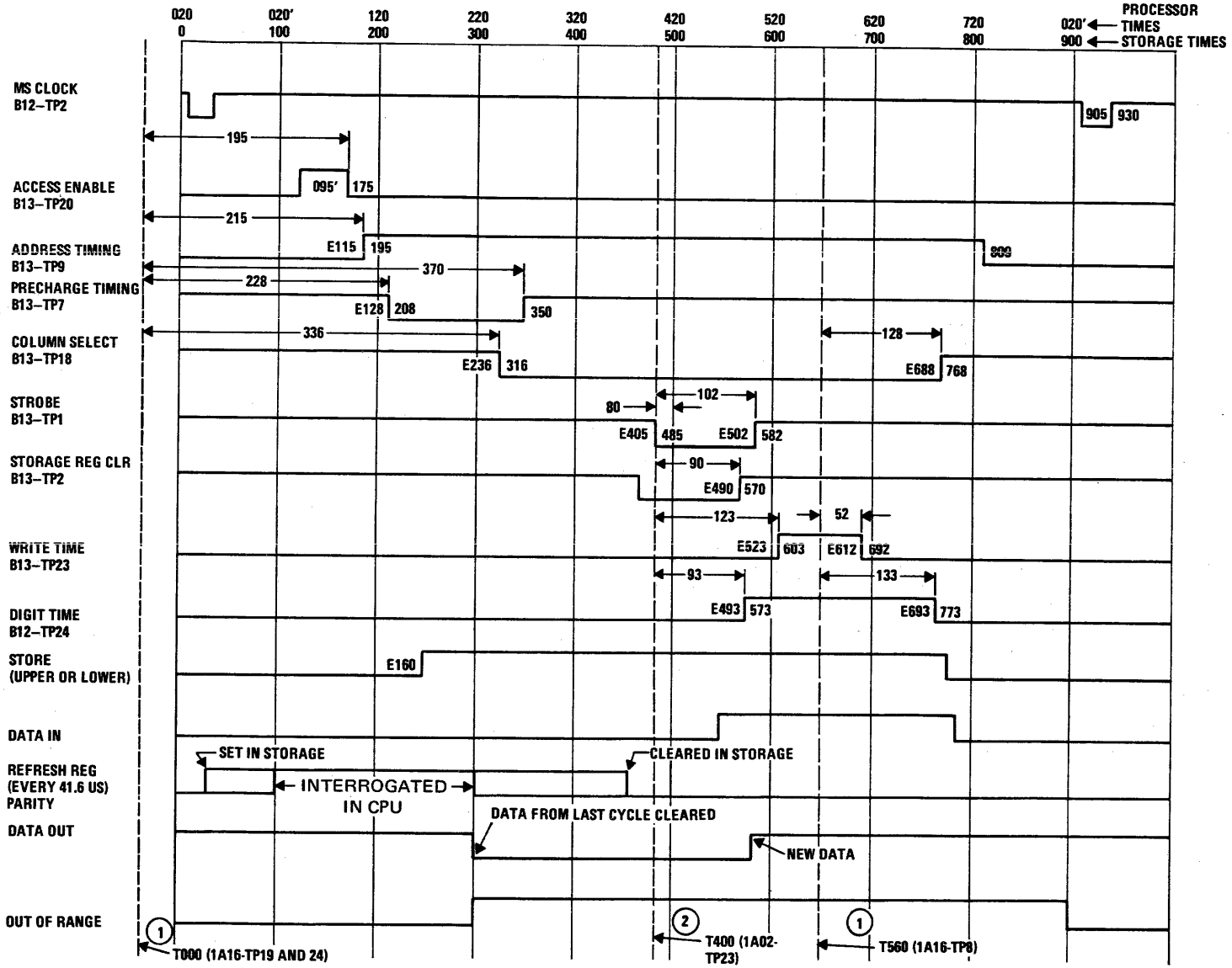
NOTE: NUMBERED FLAG MEANS ADJUSTABLE – REFER TO FIGURE 2-116 FOR PHYSICAL LOCATION OF POT.

Figure 2-115. Timing Flow Diagram



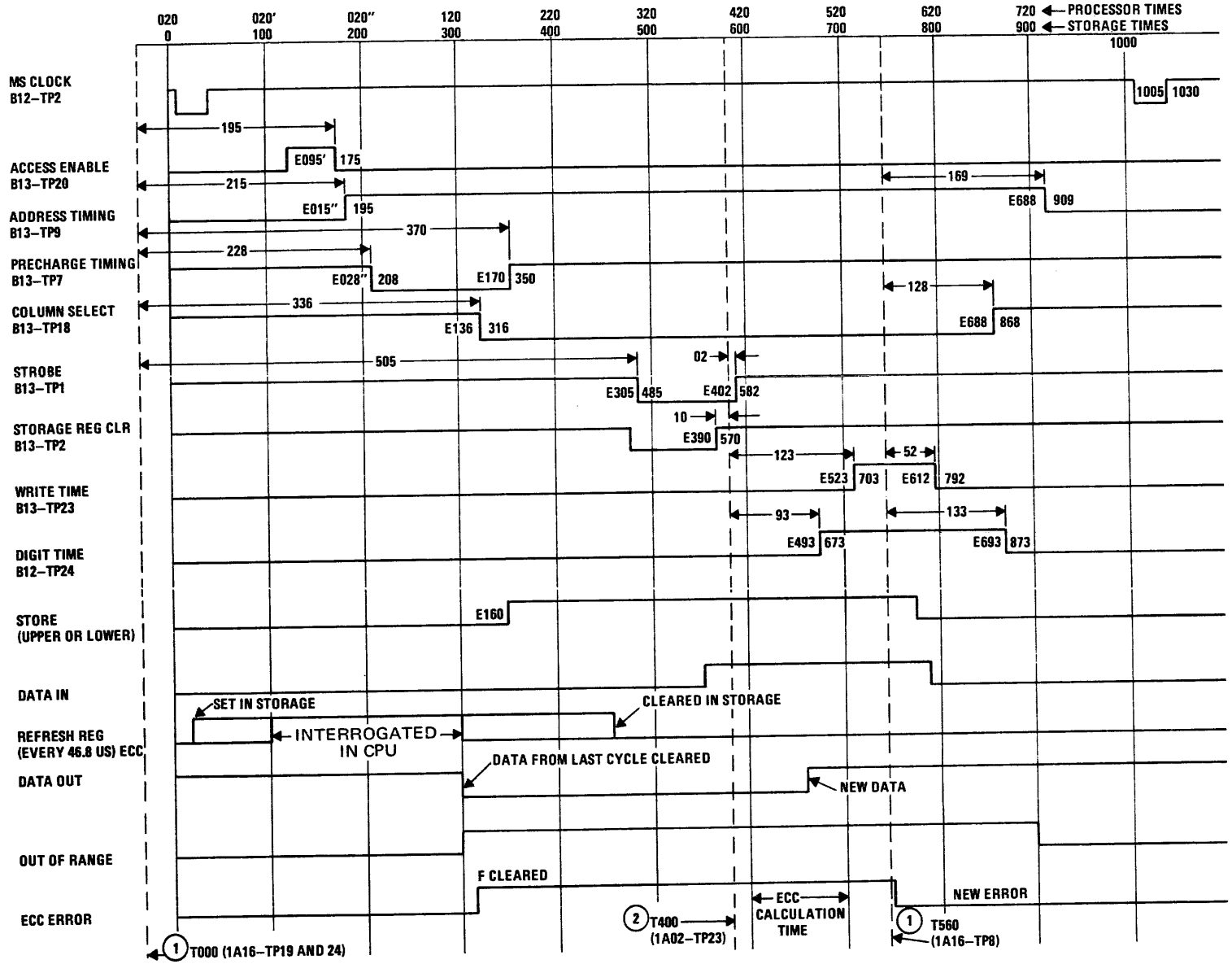
NOTE: TIMING HAS BEEN FACTORY ADJUSTED

Figure 2-116. Potentiometer Adjustment Locations



NOTES:
1. SYNC OCCURS EVERY CYCLE.
2. OCCURS ONLY DURING AN MS-RD (SWEEP) OR MS-WR (ENTER).
3. TIMES LISTED TO RIGHT OF PULSE ARE STORAGE TIMES; TO LEFT ARE PROCESSOR TIMES; BETWEEN ARROWS ARE SYNC TIMES.

Figure 2-117. Storage Timing (900 Nanosecond Cycle Time)



NOTES:
1. SYNC OCCURS EVERY CYCLE.
2. OCCURS ONLY DURING AN MS–RD (SWEEP) OR MS–WR (ENTER).
3. TIMES LISTED TO RIGHT OF PULSE ARE STORAGE TIMES; TO LEFT ARE PROCESSOR TIMES; BETWEEN ARROWS ARE SYNC TIMES.

Figure 2-118. Storage Timing (1000 Nanosecond Cycle Time)

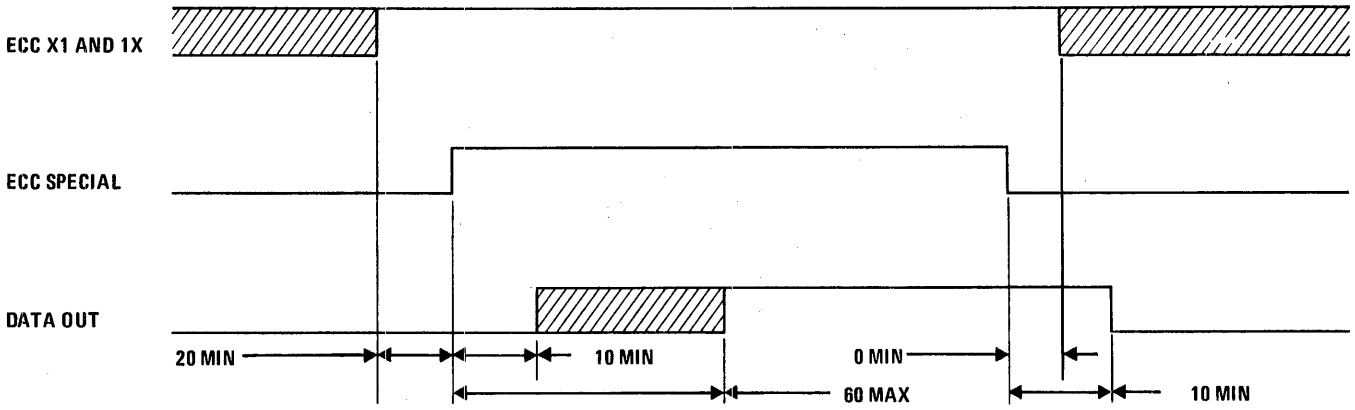


Figure 2-119. ECC Special Selection Lines (Timing)

Addressing Timing

The timing is derived from the Address Time flip-flop. The activation of Address Timing requires either Access or Refresh be present. If storage is in an idle state, the address will not be enabled and, therefore, minimizes power consumption.

Pre-Charge Time

As shown in Figure 2-107, the negative transition of precharge occurs shortly after address stabilization. In the element pre-charge acts as a clock pulse to allow the address to become stable in the row and column decoders, in preparation for a column select. When pre-charge starts positive, the chip refresh amplifiers are clocked to write data back into the respective rows and columns, and also clocks data out of the chip for sensing.

Column Select Time

As shown in Figure 2-106, column select time gates the decoded column selects so that 9 chips (or bits) are enabled using *chip enable* on the element. Column select occurs at least 30 nanoseconds before the end of pre-charge so that the selected cells within the chip present data to the chip refresh amplifiers. Column select is active until completion of any writing of new information into the chip.

Strobe Time

The strobe gates data from the sense amplifiers to the Storage Data register. Strobe timing adjustments are the same as Storage Register Clear except it is delayed by using 4 series inverters.

Storage Register Clear

Storage Register Clear is activated with the timing adjustments for strobe. Holding the register clear during strobe time improves the access time by allowing the sensed data bit to set or leave clear its corresponding SDR bit position. Because strobe is delayed from storage register clear, the trailing edge of strobe outlasts storage register clear by that delay. This delay allows the data bit to determine the state of its respective SDR bit.

Write Time

Write Time enables the read/write input of the chip allowing new data to be written in the cell selected by (active) column select and the address. Write Time is active for the last 90 nanoseconds of column select time.

Digit Time

Digit Time gates the data bits from the data fan-out to the chips as shown in Figure 2-107. When digit time is active, data can be gated to storage by two paths: the usual path is data to be stored from the CPU. The other path is (when ECC is installed) the corrected data from the ECC checking logic being written back into storage. In either case, the timing must be active long enough to gate correct data in and late enough so that ECC has enough time to correct the data. Digit time terminates after column select is terminated.

ERROR CORRECTION CODING

An introduction to Error Correction Coding (ECC) is found in Appendix 2A. The ECC used for storage is implemented on a per byte basis to save time spent on

reading a word from storage, check it, calculate new check bits for the new byte in conjunction with the entire word and then store it. The steps for storing one byte are shown in Figure 2-120 and are as follows:

1. Initiate storage to read the whole word.

2. Send a new byte from CPU so its check bits can be generated (here is where the time-savings is done – new check bits are being generated while the whole word is still being read from storage).

3. The whole word is read from storage into SDR and its ECC checked.

4. New byte with its generated check bits and old byte with its check bits are written back into storage.

When the whole word is read, the check bits are compared logically against the same data bits that generated them. Using the decoding of check bits and data bits provides the capability to correct a single error when it occurs. Should multiple errors occur, the decoding process can recognize that error correction is not possible and therefore interrupts the CPU because of a storage malfunction. When the single error is corrected, a log entry is made. (Recurring correctable errors are noted by software to determine a threshold, beyond which, a maintenance call should be made.)

Coding Check Bits

The coding of the check bits has both theoretical and practical constraints. Theory says that only five check bits will suffice to correct a single error and detect double errors. Practice takes advantage of *off the shelf* hardware parity generators having up to eight inputs. To code the check bits, the "1" or active state of the data is used because an active logical state implies correctly working circuitry. In other words, like odd parity, the lack of a signal should not be used because a disconnected cable may be interpreted as if the logic was actually present. Coding the check bits therefore uses the "1" state of the data.

Generating Check Bits

Choosing the data bits for each check bit must be done systematically so that the generation and checking are identical. The methods used to choose the check bits theoretically guarantee single error correction and double error detection (SEC-DED). Using Table 2-10, the data bit code is generated. From the decimal numbers is the conversion to its binary equivalent. Note that every column is therefore distinct. By simply counting the number of ones in a column determines the weight of that column. Using only columns of odd weight (i.e., weight 1, 3, or 5) mathematically simplifies double error detection logic. From the table, the check bits C1 through C5 are of weight one. The number of required check bits is determined from formulas proving Hamming error codes and not necessarily the number of weight-one binary numbers. The ten available data bit codes are of weight three, to code the 8-bit byte. To determine which two codes are not needed, the practical considerations of logic implementation take over. First, however, refer to Table 2-18 for constructing the error coding matrix. Eight of the ten data bit codes (weight 3) and five check bit codes (weight 1) are reproduced from Table 2-10, in the identical format. Across each row, the number of one bits is counted and tabled under Number of Inputs. In all rows, except one, there are 6 input gates. For example, generating check bit one, the five one bits from data bits 0, 1, 2, 4, and 6 are gated together.

To reduce any input or output loading problems, it is desirable to load each gate equally. The two bits eliminated from the ten originals would have caused uneven gate loading. The generation of the five check bits result from using Table 2-11 to implement the logic. Each parity generator makes a check bit produce even parity from the input data. Upon completing the check bit generation, the 8 data bits and 5 check bits are stored.

Correcting Data

When the stored data is read from storage and loaded into SDR, (Figure 2-120) the data and check bits are cross-checked for errors. The data to the corrector logic is the same polarity as the generated data even though the storage element and the data register perform one inversion each. The input to the syndrome bit generator is the same eight data bits that produced the five generated check bits and those five check bits. The output is called the syndrome bit to differentiate between the check bit alone and the data combined with the check bit. For example, syndrome bit 2 contains data bits 0, 1, 3, 4, 5 and check bit C2 as shown in Table 2-11. The output of the syndrome generator is sensed for a "1" indicating a failure in one of the input bits. To recover the bad data bit, three-input AND gates are used in a coding scheme shown in Table 2-11. For example, to recover data bit 0, syndrome bits 1, 2, and 3 must be in an active state. The

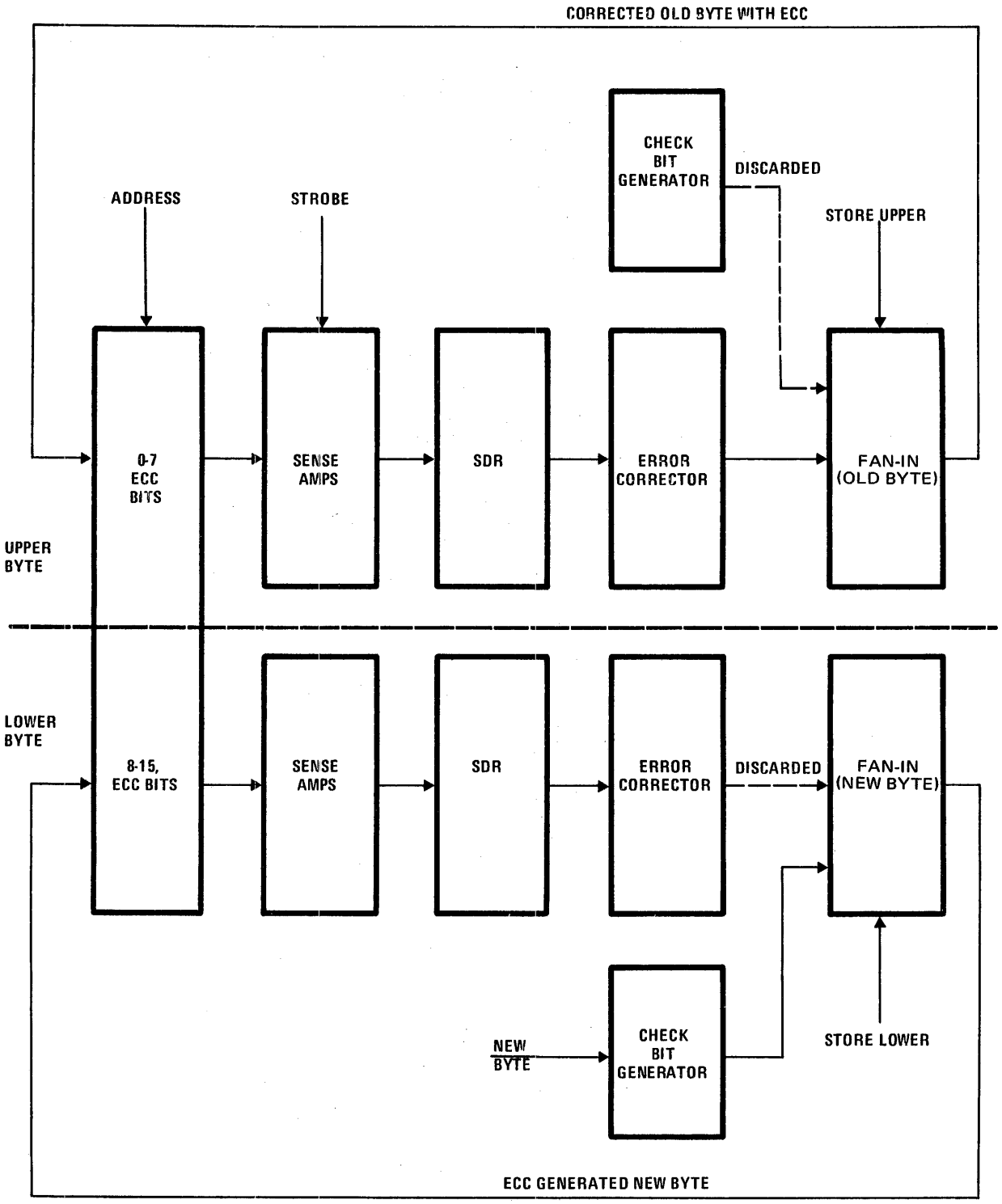


Figure 2-120. Storing Lower Byte with ECC

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	DECIMAL NUMBER
0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	BINARY EQUIVALENT
0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	
0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	
0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
0	1	1	2	1	2	2	3	1	2	2	3	2	3	3	4	1	2	2	3	2	3	3	4	2	3	3	4	3	4	4	5	WEIGHT (NO. OF "1's" IN COLUMN)
	C_1	C_2		C_3				C_4								C_5																CHECK BITS (WT 1)
							0				1		2	3					4		5	6			7	8		9				DATA BIT CODE (WT 3)

Table 2-10. Error Coding Table

Table 2-11. Syndrome Bit Generating Matrix

	DATA BITS								CHECK BITS					NO. OF INPUTS
	0	1	2	3	4	5	6	7	C_1	C_2	C_3	C_4	C_5	
S1	1	1	1		1		1		1					6
S2	1	1		1	1	1				1				6
S3	1		1	1		1		1			1			6
S4		1	1	1			1	1				1		6
S5					1	1	1	1					1	5
NO. OF ACTIVE INPUTS	3	3	3	3	3	3	3	3	1	1	1	1	1	

syndrome bits used for recovering each data bit are shown in each column of Table 2-11 with the logic gate-input requirements at the bottom of the column. Note that if a check bit failed, only one syndrome bit will be active. For a single data bit error, the recovery generation is defined according to Table 2-11. Any other error is non-recoverable.

Error Interpretation Control

The syndrome bits are decoded into three error interpretation classes as shown in Figure 2-121. These three classes cover all the combinations of syndrome bits needed for error interpretation. Once the syndrome bits generate the control signals, corrective actions are taken.

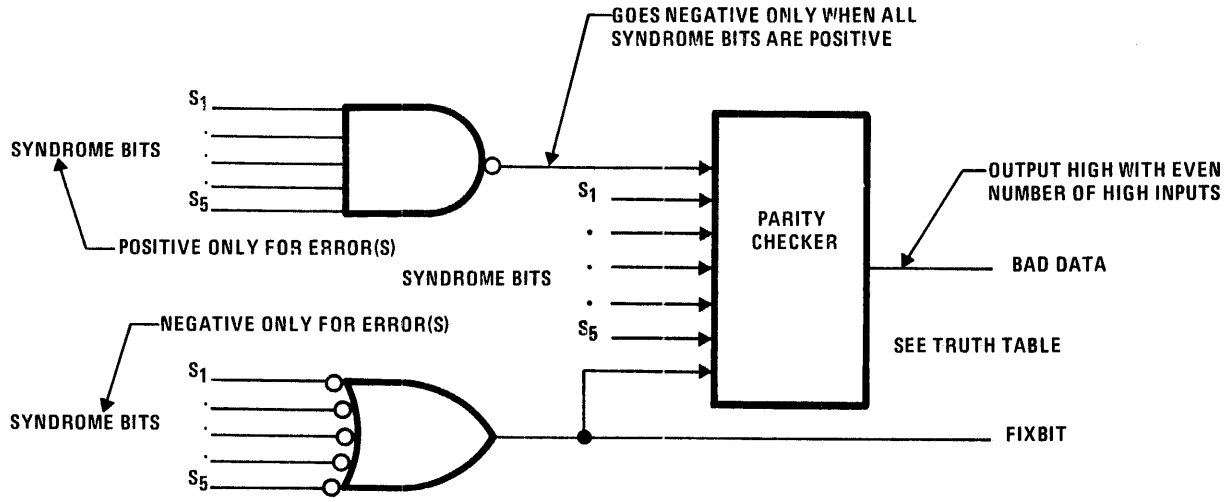
When all syndrome bits are "0" the data from storage is good. Using the logic in Figure 2-121, the syndrome bits are checked for unrecoverable errors called Bad Data, and recoverable errors called FIXBIT. When unrecoverable errors occur, the Bad Data signal is sent to the processor. The FIXBIT signal controls the write timing discussed in the paragraph titled Write Time and indicates that correcting the data will be attempted. In the case of all syndrome bits being "0", there are no active outputs for FIXBIT, Bad Data or the eight toggle control gates which indicates good data.

There are two kinds of correctable errors; either data bits or check bits. The difference between failing data bits or check bits will be the number of active syndrome bits. For example, if data bit 7 fails, by using Table 2-11 there will be three syndrome bits S3, S4, and S5 active. If check bits C1 fails only syndrome bit S1 will be active. In other words, by decoding the syndrome bits using Table 2-18, the erring data bit can be corrected by using three-input AND gates and check bits can be corrected with one active and four inactive syndrome bits. Across the bottom of Table 2-18 is the number of required active inputs which is either one or three. When one or three syndrome bits are active the combination of bits will decode into a data or check bit and therefore the error is correctable. The error interpretation logic will activate FIXBIT and attempt to correct the error. From Figure 2-121 either one or three active syndrome bits causes Bad Data to go low and be interpreted as the data has been corrected.

Unrecoverable errors occur when the active syndrome bits cannot be decoded into erring data or check bits. For example, if two syndrome bits S2 and S3 were active, there are no data bits or any single check bits of weight 2. Similarly there are no data or check bits of weight 4 or 5. When errors that activate syndrome bits with weights equal to 2, 4 or 5 are sensed, they are interpreted as Bad Data (active) with FIXBIT also active. The computer is signaled that an unrecoverable error has occurred with the Bad Data line which is the ECC error interface line, while error recovery is attempted internal to storage with the FIXBIT line. However, unless the proper syndrome bits shown in Table 2-11 are active, data or check bits may be incorrectly altered. With all five syndrome bits active for example, all eight data bits would be selected and their data altered. By providing some inspection logic, the data and check bits can be examined for future reference.

Error Logging

When one or more syndrome bits are non-zero, an error condition exists that may or may not be recoverable. Recovery internal to storage occurs when the syndrome bits match the matrix in Table 2-11. Any other combinations of syndrome bits result in unrecoverable errors which activate the ECC error line. In either case a



TRUTH TABLE FOR ABOVE LOGIC

NUMBER OF SYNDROME BITS THAT ARE ONES	CONDITION OF		ERROR INTERPRETATION		
	BAD DATA	FIXBIT	GOOD DATA	CORRECTABLE ERROR	NOT RECOVERABLE
0	LOW	LOW	X		
1	LOW	HIGH		X	
2	HIGH	HIGH			X
3	LOW	HIGH		X	
4	HIGH	HIGH			X
5	HIGH	HIGH			X

Figure 2-121. Error Interpretation Logic

log entry is made. To isolate the error, an error logging register is used with the format shown in Table 2-12. The 10 syndrome bits for both bytes and the 4 board select address bits are gated into a temporary storage register called the *log*. As each new entry is made into the log, the 16th log bit is set signifying that new information has been entered into the log. When the CPU inspects the log, the 16th bit (or enter bit) is cleared by the CPU so that if a subsequent inspection occurs before another error is loaded into the log, the same error will not be inspected twice. The log is a clocked register that is loaded only when the clock input is high and then retains that information after the clock goes low. As each error is recognized by FIXBIT a new entry is made independent of whether or not the last entry was inspected. At any time if there is an entry in the Error Log registers it is the last error made.

ECC CONTROL

After the syndrome bits have been decoded into an erring bit, that bit must be corrected before it is sent to the CPU and then written back into storage. If an error did occur, the Error Log register is loaded. The data or check bit corrections occur independent of timing; however, timing does control the writing back into memory. Since error correction is done without timing for gating, the raw or uncorrected data cannot be examined by hardware or software for proper action. Using Figure 2-122, the data paths for normal error correction and diagnostic interrogation can be shown.

Table 2-12. Error Logging Format

DATA BIT LINE	12	13	14	15	11	10	9	8	7	6	5	4	3	2	1	0
FUNCTION	BOARD SELECT ADDRESS					SYNDROME BITS										
SIGNAL NAME	X3	A0	A1	A2	SPARE	10	9	8	7	6	5	4	3	2	1	ENTER NEW BIT INFOR-MATION
INPUT PIN (B13)	45	46	35	36	52	51	42	41	76	63	77	70	71	58	57	–
OUTPUT PIN (B13)	48	47	38	37	49	50	39	40	74	75	61	62	73	72	60	59

Normal Error Recovery

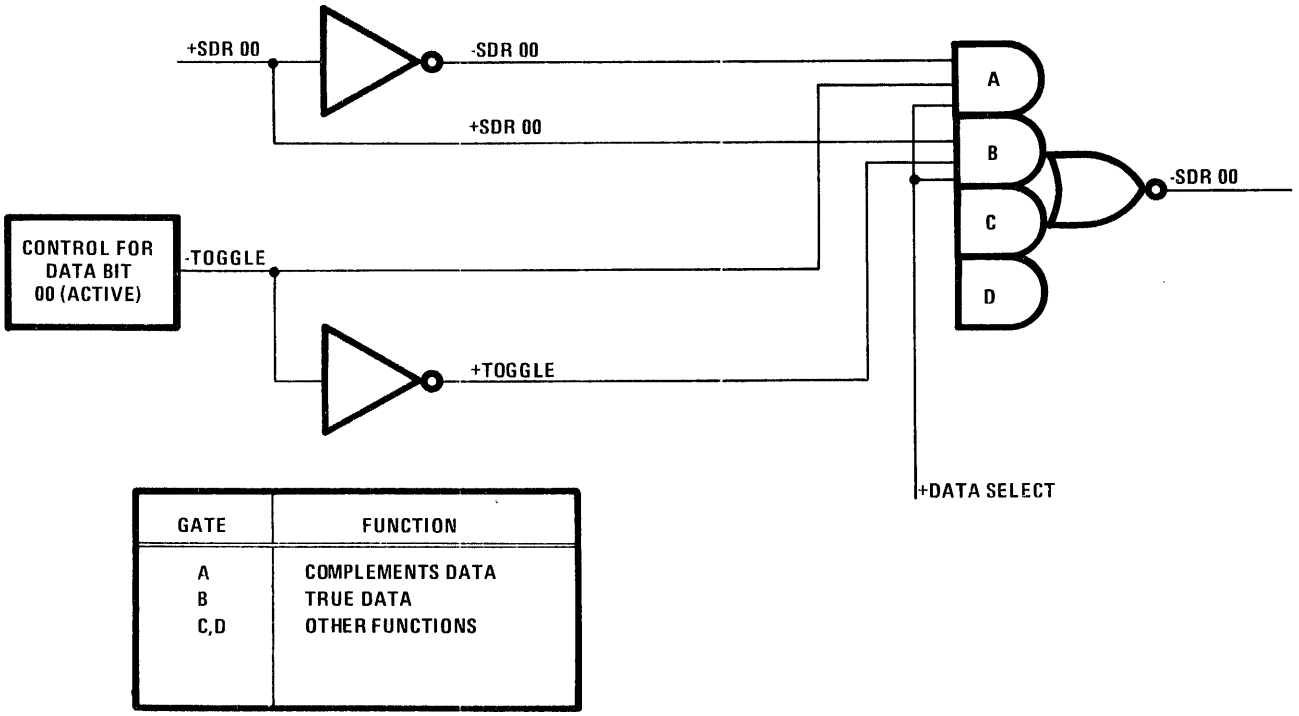
The clearest way to understand a block diagram such as Figure 2-122 is to take one data bit and follow it through. Since the check bit is slightly more complex than data, it will be more instructive to follow check bit, C1. Check bit C1 is corrected (when necessary) by generating a pair of corrector bit lines coming from the syndrome bit decoding logic and logically comparing these lines to a pair of uncorrected lines (for C1) from storage. The pair of lines from storage is just bit C1 and its inversion (complement). Logically combining C1 and its complement with the C1 corrector bits controls the correcting of C1 regardless of C1 originally being a "0" or a "1". All data and check bits are corrected with this scheme. Figure 2-123 illustrates how the corrected check bits are corrected with this scheme. From Figure 2-123, the corrected check bits are gated back to storage for writing and the raw check bits are made available to the CPU via the Error Log register for software to check. Similarly the data bits are corrected and made available to the CPU and then written into storage.

Diagnostic Control

Storage provides logic to interrogate the Error Log register and the Storage Data register (which includes check bits). A special signal line (MS-SPEC) from the CTA initiates the diagnostic inquiry along with two other special lines that send the diagnostic code. When the MS-SPEC is active, the other select lines are decoded as shown in Table 2-13. Normally, the DATA SEL gate is active in absence of MS-SPEC so that data and check bits are automatically corrected as shown in Figure 2-123. When another diagnostic selection code is used, DATA SEL blocks all error correcting. The signal code RAW CHK enables 10 raw check bits generated from input data to be read and interpreted by CPU hardware/software. In contrast to RAW CHK, signal code RD CHK (code 11) enables 10 uncorrected check bits read from storage to be gated to the CPU in the format shown in Table 2-14 (right column). For diagnostic maintenance, code 11 can be used to help isolate failing memory bits.

ECC Write Controls

When a recoverable error occurs, the active FIXBIT enables the write controls. These write controls are controlled by the ECC FIXBIT. As shown in Figure 2-124, the left-most (upper) and right-most (lower) bytes overlap each other on the ECC card row B. For example, either store upper from the CTA or FIXBIT upper from ECC activate the write controls for the left-most byte and the ECC check bits. The resultant write controls arythen combined with the write and digit timing.



GATE	FUNCTION
A	COMPLEMENTS DATA
B	TRUE DATA
C,D	OTHER FUNCTIONS

Figure 2-123. Data Correction Logic

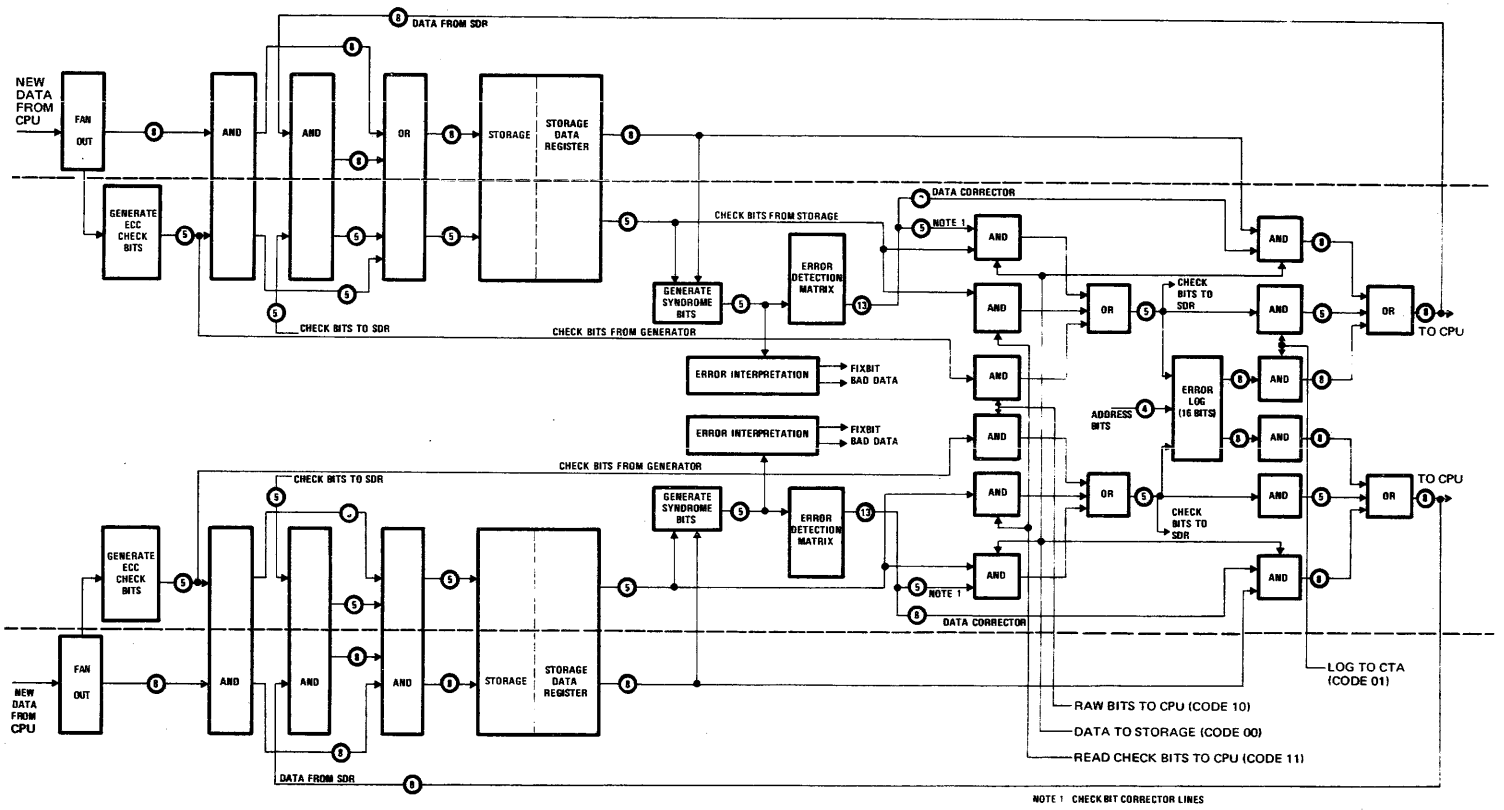


Figure 2-122. ECC Detailed Block Diagram

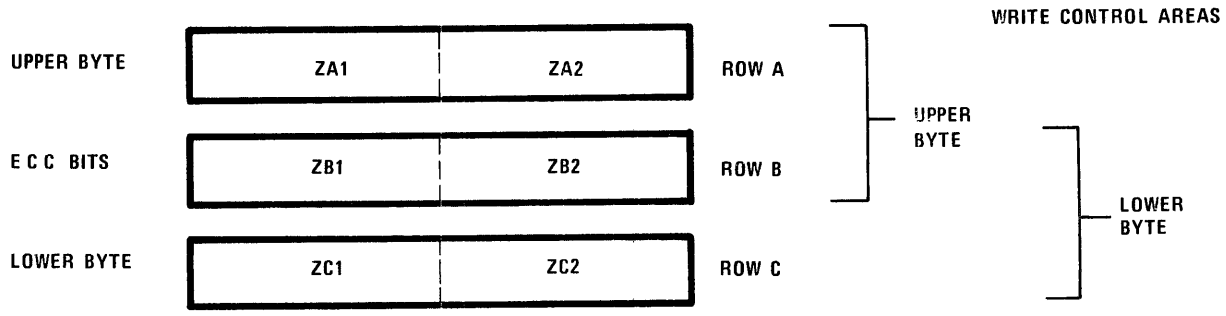


Figure 2-124. Write Control for ECC

Table 2-13. Diagnostic Selection Codes

Select Lines		Code	Signal Name	Function
SLX1	SL1X			
0	0	00	DATA SEL	Enables 16 corrected data bits from SDR to CPU. Normally active in absence of special signal. Normally gates corrected check bits to SDR for writing.
0	1	01	LOG SEL	Enables 16 log bits to CPU in format of Table 2-12. Entry bit clears when code is removed.
1	0	10	RAW CHK	Enables 10 check bits from input data check bit generates directly to CPU.
1	1	11	RD CHK	Enables 10 corrected check bits to CPU.

NOTE: Detailed references for these signals are found in Table 2-14.

Table 2-14. Detailed Diagnostic Selection Code References

Function Select Name	Can be Scoped On	Signal Name	Normal Data	Data	Error Log	Generated Check Bits	Read Check Bits
Select Lines Format	2B13-53	+ MS-SPEC	0	1	1	1	1
	2B13-69	- SLX1-ECC	X (X=don't care)	1	0	1	0
	2B13-68	- SL1X-ECC	X	1	1	0	0
	2A11-27	MS-DR00	Data Bit 0	Data Bit 0	Entry Bit	ECC CHK 1	ECC CHK 1
	2A11-26	MS-DR01	1	1	Syndrome 1 Bits	2	2
Data Lines (used for the bits selected)	2A11-22	MS-DF02	2	2	2	3	3
	2A11-24	MS-DF03	3	3	3	4	4
	2A11- 4	MS-DF04	4	4	4	5	5
	2A11- 6	MS-DF05	5	5	5	Spare	Spare
	2A11-10	MS-DF06	6	6	6	Spare	Spare
	2A11-11	MS-DF07	7	7	7	Spare	Spare
	2C11-27	MS-DF08	8	8	8	ECC CHK 6	ECC CHK 6
	2C11-26	MS-DF09	9	9	9	7	7
	2C11-22	MS-DF10	10	10	10	8	8
	2C11-24	MS-DF11	11	11	Spare	9	9
	2C11- 4	MS-DF12	12	12	ADD X3	10	10
	2C11- 6	MS-DF13	13	13	0	Spare	Spare
	2C11-10	MS-DF14	14	14	1	Spare	Spare
	2C11-11	MS-DF15	15	15	2	Spare	Spare

NOTE: When the select lines are in the format of a given column, that column of bits are gated on the data lines.

REGISTER OPTION

The Register Option (RO) comprises registers and associated control logic that implement the following features:

1. Basic Storage Protection
2. Relocation and Protection
3. Job Accounting
4. Error Correction Code (ECC)

The first three features are physically part of the Central Processing Unit (CPU) portion of shared resources. The ECC feature is located in the Main Storage (MS) portion of shared resources and, for purposes of convenience, is discussed in the paragraph titled Main Storage. This section, therefore discusses operations of only the Basic Storage Protection feature, Relocation and Protection feature, and the Job Accounting feature.

The Basic Storage Protection, and Relocation and Protection features are mutually exclusive; that is, when the Basic Storage Protection feature is present, the Relocation and Protection feature is absent, and visa versa. The Basic Storage Protection feature is used for MS sizes of 65,536 bytes or less, while the Relocation and Protection feature is mandatory for MS sizes greater than 65,536 bytes. The Job Accounting and ECC feature may be present in the machine if either the Basic Storage Protection or Relocation and Protection feature is installed. Figure 2-125 shows the placement of RO modules in chassis 1 of the module deck. The modules comprising the Basic Storage Protection and Relocation and Protection features are both installed in locations 1B27 and 1B28 as shown. Since all logic necessary for the Basic Storage Protection feature is contained on module BK at location 1B28, module BL at location 1B27 is simply a jumper board to interface signals that would be processed by module BJ of the Relocation and Protection feature if it was installed.

The Basic Storage Protection feature checks storage bounds on write operations only for processor states 5, 6, and 7. The check is made by defining both an upper page limit and lower page limit beyond which a write reference may not be made without error. If a bounds error occurs, the write is inhibited and a trap routine is entered.

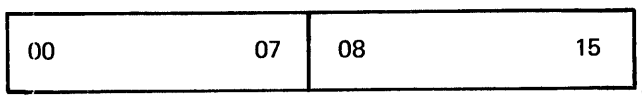
The Relocation and Protection feature expands the MS addressing structure from 16 to 20 bits allowing addressing of up to 1 million bytes. This is accomplished under program control by furnishing a 4-bit segment tag value that can be either appended directly to the 16-bit address in S, or used to select a 12-bit relocation constant that can be added to the contents of S to relocate all subsequent MS references by the amount of the constant. The relocation constant is obtained from a segment relocation table that contains 16 such entries. These 16 entries essentially divide MS into 16 separate segments for purposes of providing areas of common usage, certain combinations of read and write protection, and other factors under control of the operating system. In addition, the Relocation and Protection feature also furnishes both read and write protection for all eight processor states. This protection may be implemented in either of two forms (1) reading or writing a particular portion of MS, or (2) attempting access to a particular portion of MS without regard to the type of reference.

The Job Accounting feature consists of eight 32-bit registers, one per processor state. These registers log the number of time slices (major cycles) assigned to each processor state.

A block diagram showing the address and data paths to and from the RO is shown in Figure 2-126. Since the RO is located between the S and D registers and MS, all address and data bits put into these registers must first pass through the RO before going to MS. An address put into S may be used to either address a location in MS or a register in the RO. If addressing an MS location, the address is checked for bounds protection and, if the Relocation and Protection feature is present, added to the relocation constant to generate the expanded 20-bit physical memory address. If addressing a register in the RO for purposes of reading or writing the register, the address is routed through processor and register select logic in the RO to select the appropriate register. An address used to access an RO register must be loaded into S from only transient registers 1E or 1F of the BRF. Data loaded into the D register is stored in either MS, if addressing MS, or in an RO register, if addressing the RO.

BASIC STORAGE PROTECTION FEATURE

The Basic Storage Protection feature is implemented by three 16-bit registers, a compare network, and MS write inhibit logic. One bounds register is assigned to each of the processor states protected: 5, 6, and 7. The format for each of the bounds registers is as follows:



UPPER BOUNDS LOWER BOUNDS

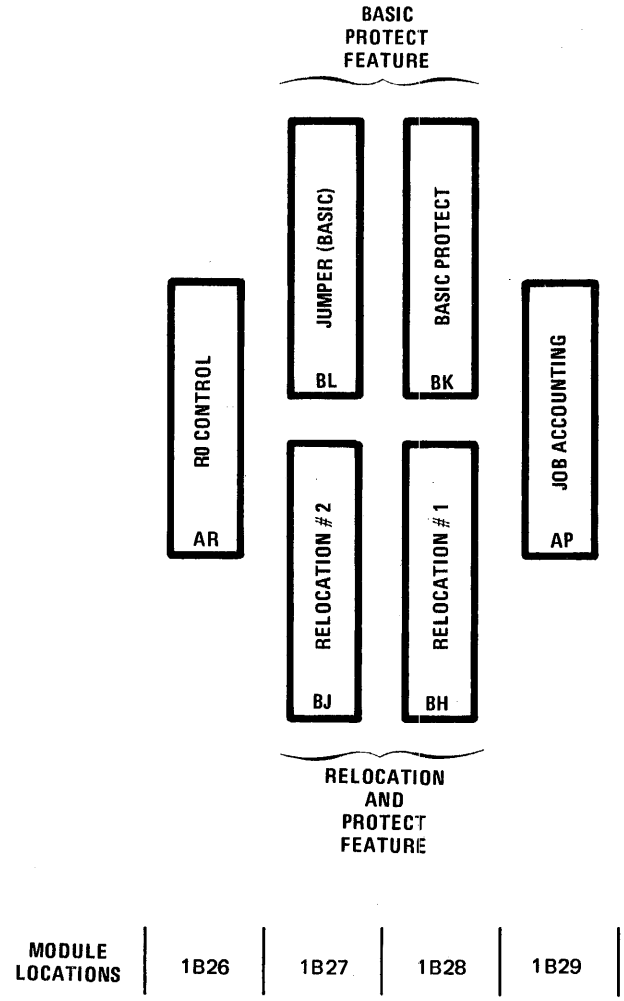


Figure 2-125. Installation of Either Basic Storage Protect or Relocation and Protection Feature

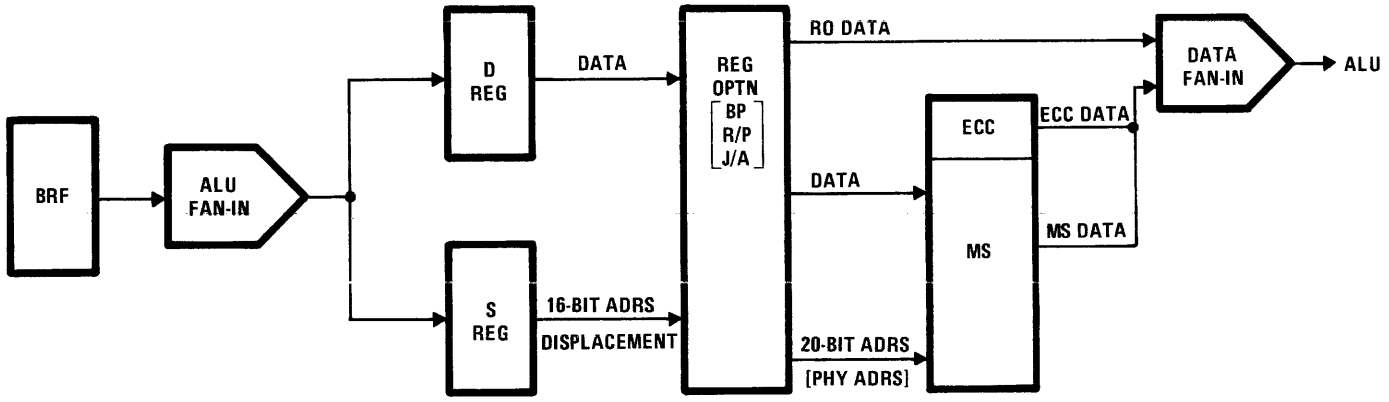


Figure 2-126. Register Option, Block Diagram

The bounds protect concept is based on dividing MS into pages of 256 bytes each. The upper bounds half (bits 00 through 07) designates a maximum MS page number, whereas the lower bounds half (bits 08 through 15) designates a minimum MS page number. When the upper and lower bounds are equal, the MS write references are restricted to that one page. When the upper bounds equals FF_{16} and the lower bounds equals 00_{16}, no main storage protection takes place.

Logic of the Basic Storage Protection feature used during normal bounds compare operation is shown in Figure 2-127. Each half of the three bounds regisers feeds a corresponding selector enabled by processor select signals ROST-1XX and ROST-X1X. These select signals are derived from the logic shown in Figure 2-128 and are generated for either of two conditions: normal operation (MS reference) or register read/write. For either case, three ROST signals are generated which represent the processor number in binary form. During normal operation, the ROST signals are generated from corresponding EXCT signals from the resource allocation network, which defines which processor has been granted the present time slice. During a bounds register access (read or write operation), the ROST signals are derived from bits 8, 9, and 10 of the S-register. These three bits define the processor number in either a Read Register Option (RRO) or Write Register Option (WRO) MLI, for specifically accessing a particular RO register. These bits are enabled by RO-SPEC, which is generated during an RO access (see the paragraph entitled RO Reference Signals). When selected by a particular value of ROST signals, both halves of the bounds register selected are routed through the corresponding selector to individual bounds compare networks. Each network is also fed with the MS page address contained in bits 0 through 7 of the S register. These 7 bits of S are also routed to MS as ROS-MS bits 0 through 7. The two bounds compare networks make the following comparison of the page address agains the upper and lower bounds limit:

Page Number Less Than or Equal to Upper Bounds Limit

Page Number Greater Than or Equal to Lower Bounds Limit

If both these compare conditions are met, each compare network generates a low output which is combined with enable (5+6+7) · MS-WR. This enable indicates that processor state 5, 6, or 7 is executing an MS write operation, the necessary prerequisite for performing a basic storage protect bounds check. The result is to make ROACCESS go high to permit the write operation to take place.

If either compare condition is not met, i.e., page address greater than upper bounds limit or less than lower bounds limit, ROACCESS goes low to abort the write operation. Logic for setting up the abort condition is shown in Figure 2-129. The low ROACCESS signal clears the Outbound flip-flop to generate a low from the Q output, provided the System Control Panel has not requested an RO access (CONST-RO is low). This low is sent to the STOREUPP and STORELOW gates to disable them, thus forcing MS write signals STOUPPMS and STOLOWMS low. In addition, and low flip-flop output is combined with the STOREUPP and STORELOW signals to generate OUTBOUND. This signal is sent to the trap routine starting address logic (see the paragraph entitled Set Pp Logic) to cause a jump to the MS parity error trap routine.

The four extended MS address bits RO-MSX0 through RO-MSX3 shown connected to ground on Figure 2-127 are so connected to eliminate a floating condition that might be interpreted by MS as extended address bits set to "1's". As discussed in the paragraph titled Register Option, this basic storage protection module, type BK, is interchangeable with relocate and protection module BH if the Relocate and Protection feature is installed. Since the Relocation and Protection feature uses these four bits as the upper four-bit extension to the 16-bit address in S, these bits must be purposely grounded out if the basic protection feature is installed.

RELOCATION AND PROTECTION FEATURE

For discussion purposes, the relocation and protection portions of the Relocation and Protection feature will be treated as separate functions. During an actual MS reference, however, the two operations are performed at the same time.

Relocation

The general procedure for relocation is shown in Figure 2-130. The 16-bit MS address in S, obtained from the BRF register as defined by the Load SμI X-field, is called a displacement address. The register number is also used to select a segment tag, a four-bit value that points to one of sixteen 24-bit entries in the segment relocation table. This segment tag resides in a register of the segment tag file corresponding to a register in the BRF, and is addressed concurrent with the BRF register. In effect, the Segment Tag register constitutes a four-bit extension of the BRF register to permit the expanded addressing capability provided by the Relocation and Protection feature. The combination of the BRF register contents

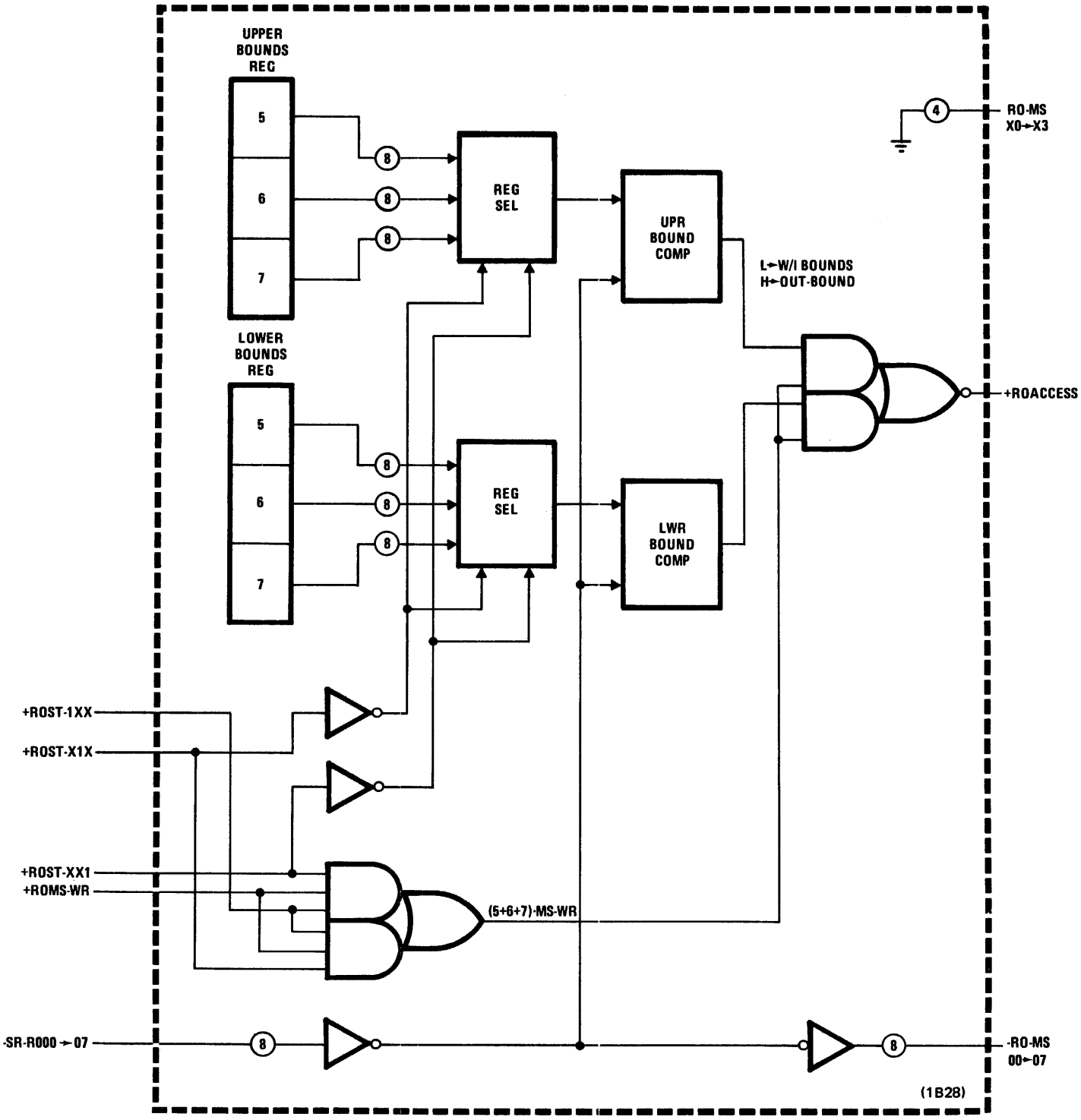


Figure 2-127. Basic Protect, Bounds Compare

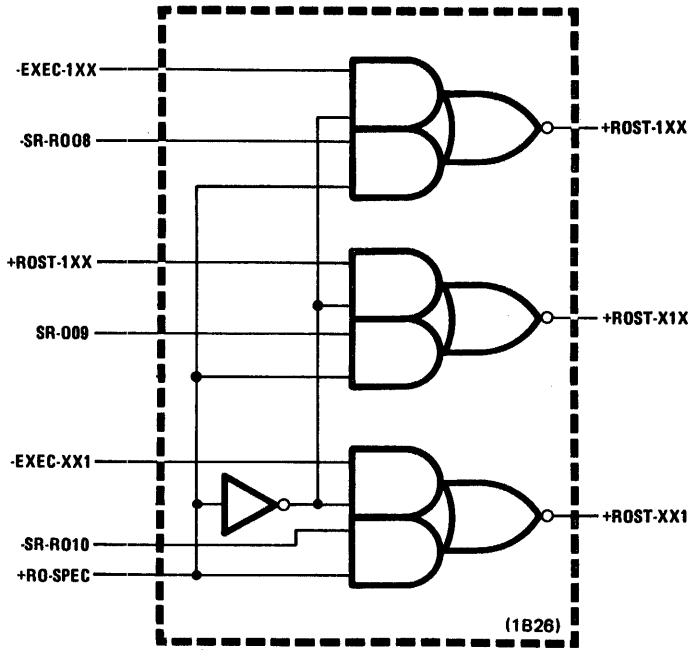


Figure 2-128. Generation of Processor Select Signals

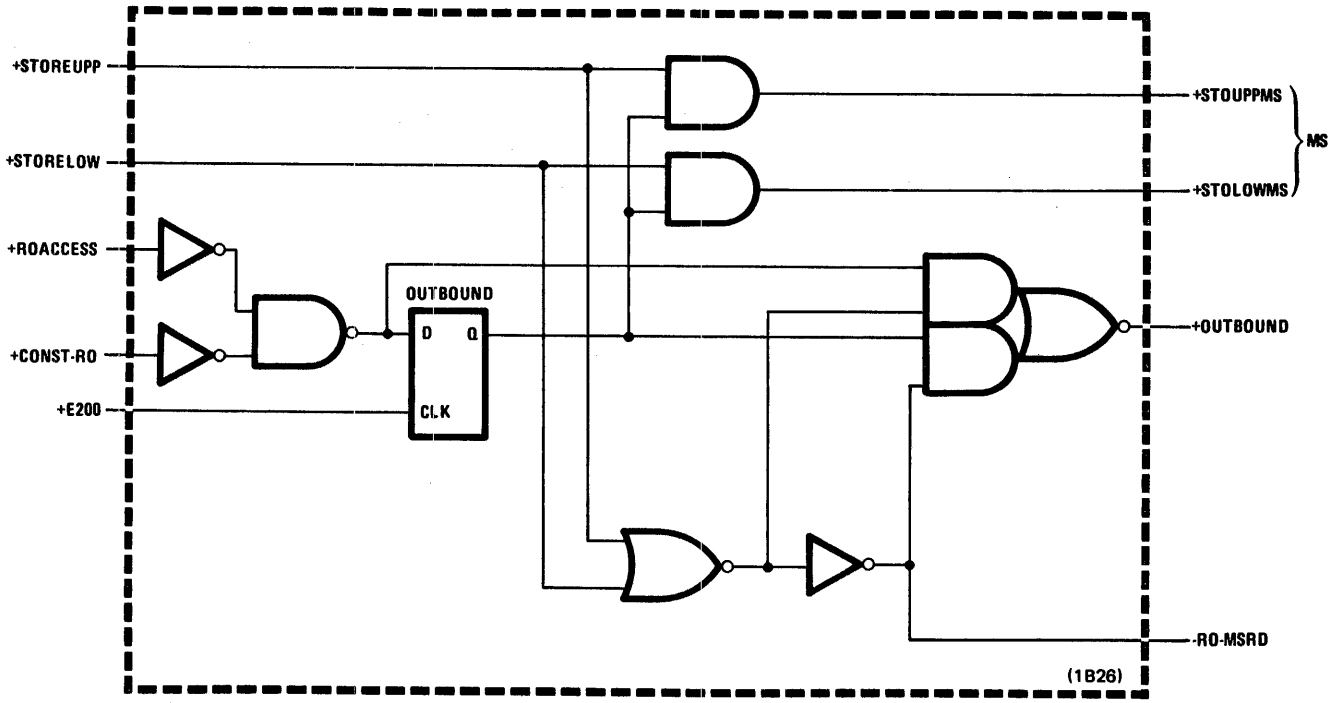


Figure 2-129. Write Operation Abort Logic

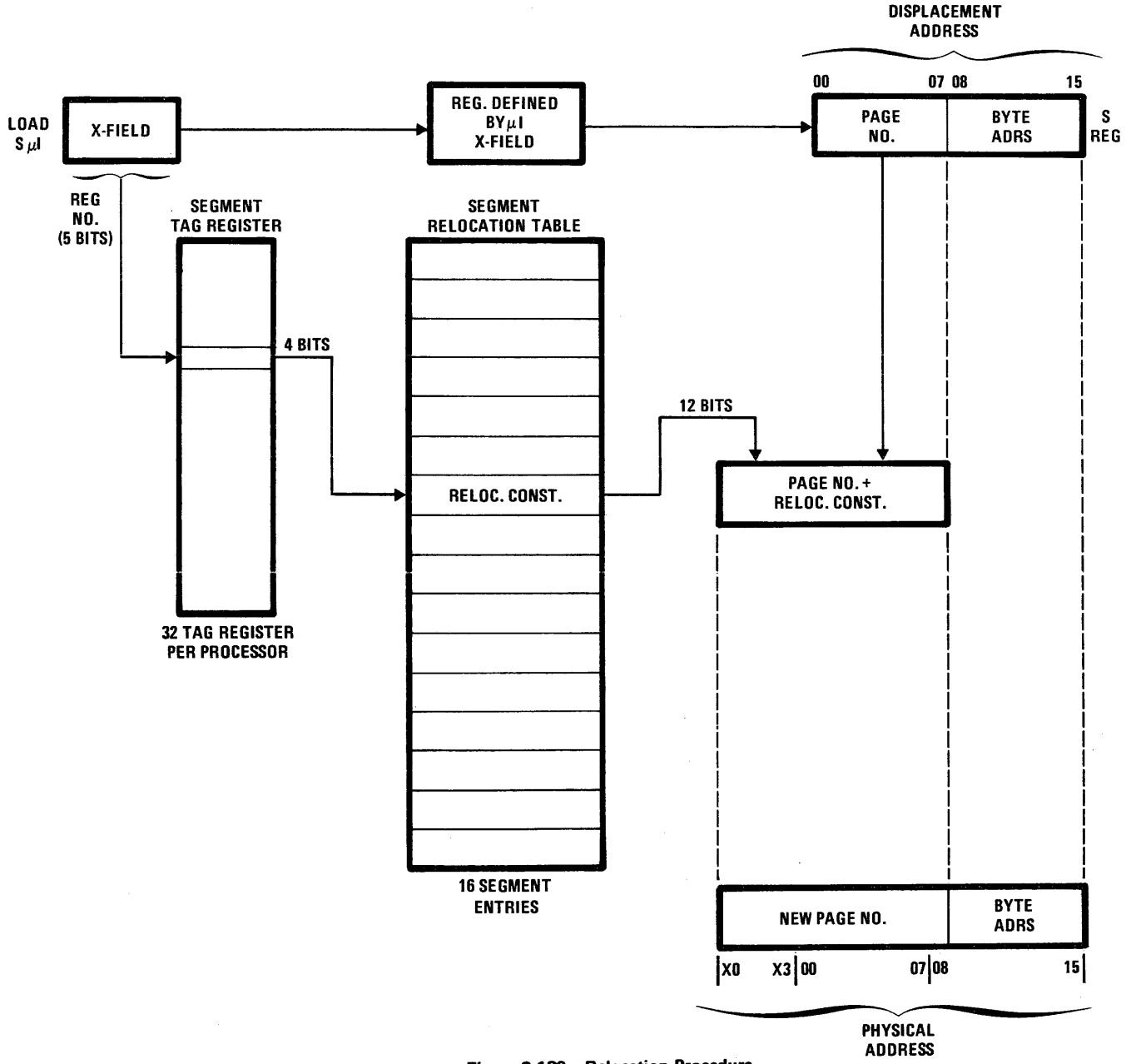


Figure 2-130. Relocation Procedure

and that of the associated Segment Tag register is called the system address. The right-most 12 bits of the segment relocation table entry, called the relocation constant, are added, right-justified to the page number portion of the displacement address, to obtain a new 12-bit page number. This number, combined with the unchanged bits from the byte address portion of the displacement, forms the 20-bit physical address to which the MS reference is made. As far as bit numbering is concerned, the physical address is considered to consist of two parts: a 16-bit right-most part, made up of bit positions 0 through 15, and a 4-bit left-most part, made up of bit positions X0 through X3.

Logic which perform the relocation function is shown in Figure 2-131. The segment tag register file is addressed by a combination of ESXXX-RO bits, which define the processor state executing, and $\overline{\text{BRFXS0}}$ bits, which define one of the 32 segment tags associated with the executing processor state. The four-bit segment tag value is gated through a selector to the S register extension and to the Sb register. This selector is fed with segment tag values from three sources: the Segment Tag register, pushbuttons X0 through X3 of the CONSOLE REGISTER ADDRESS DISPLAY pushbuttons, and S register bits 11 through 14. During normal operation, the segment tag value is obtained from the Segment Tag register by the absence of both enables $\overline{\text{ROSTSEL}}$ and $\overline{\text{MSSTSEL}}$. The segment tag value is clocked into the S register extension at the same time that the 16-bit displacement address is clocked into S by CLKSTR and ENCLKSTR. Signal ENCLKSTR is generated for two different conditions, as shown in Figure 2-132. During normal operation, it is generated by $\overline{\text{ENCLKSR}}$ which enables clocking the S register. During a read or write into the RO, it is generated at E150 by RO-SPEC. Simultaneous with clocking into the S register extension, the tag value is also clocked into the Sb register. This register holds the tag during indexing operations, as explained later.

The segment tag in the S register extension is sent to the segment tag table to select one of the 16 relocation entries in this table. Each entry is 24 bits long, consisting of two 12-bit words. The right-most word consists of the relocation constant to be added to the displacement address in S. The left-most word contains the maximum page number and validity bit used for bounds protect evaluation, as discussed in the paragraph titled Protection. Each entry is stored in six storage elements, 4 bits per element. When addressed by the segment tag value, the corresponding entry is read from the storage elements with the relocation constant being routed to three adder elements, as shown in Figure 2-131. Although stored in the relocation table as a 24-bit entry, the software which reads or writes the relocation table considers each entry to be 32 bits long, consisting of two 16-bit words. The right-most word of this entry consists of the relocation constant in bit positions 4 through 15 with bit positions 0 through 3 set to "0's". The left-most word consists of the validity bit in bit position 0 and the maximum page number in bit positions 8 through 12 with bit positions 1, 2, and 3 set to "0's". This correlation between the two forms of a relocation table entry as interpreted by hardware and software is shown in Figure 2-133.

The relocation constant portion of a segment table entry addressed by the segment tag value is fed to three relocation adder elements, along with bits 0 through 7 of S (page number). In addition, the segment tag value itself is fed to the bit positions 0 through 3 relocation adder. The result is to form a 20-bit physical address from which the location in MS will be addressed. (In reality, the physical address presented to MS is really only 19 bits long, since the right-most bit (bit 15) is used in the MS interface logic to develop separate byte write signals.) This physical address is determined in one of two ways, depending on the position of the CONSOLE MAIN STORAGE switch on the System Control Panel. This switch generates enable RELOCATE, as shown in Figure 2-134. Logic for generating this enable assumes that either an MS read or write operation has been selected, and the Panel has been granted a time slice (CONST → RO high). If the switch is in the RELOCATE position, signal RELOCATE goes high to enable the relocation adder. The result is to form the physical address by relocating the system address, via addition of the relocation constant, as shown in part *a* of Figure 2-134. If the switch is in the OFF position, RELOCATE goes low and the relocation adder is inhibited. The result is to form the physical address directly from the system address, bypassing the relocate mechanism, as shown in part *b* of Figure 2-134. For the case of relocation, signal ADRS MODE RELOC is ANDed with RELOCATE. This signal is developed from the Address Mode register, which indicates that relocation for the selected processor state is specified.

The segment tag value routed to the Sb register is used during load S operations to insure that once a reference is made to a relocated segment of MS, as determined by the segment tag corresponding to a particular BRF register, that all subsequent references to MS in the same program will be made to the same segment even through a reference might be made from a different BRF register. This sequence is altered, however, when an indexing operation is performed which changes the relocation from that of the original Load S μI to that furnished by the segment tag of the index register. An example of using segment tags for relocating MS references during both non-index and index operations is shown in Figure 2-135. This figure shows execution of a MOVM (60) MLI, using both indirect addressing and indexing, in both pictorial form and by a partial listing of the corresponding μI program. (This partial listing has been simplified to show

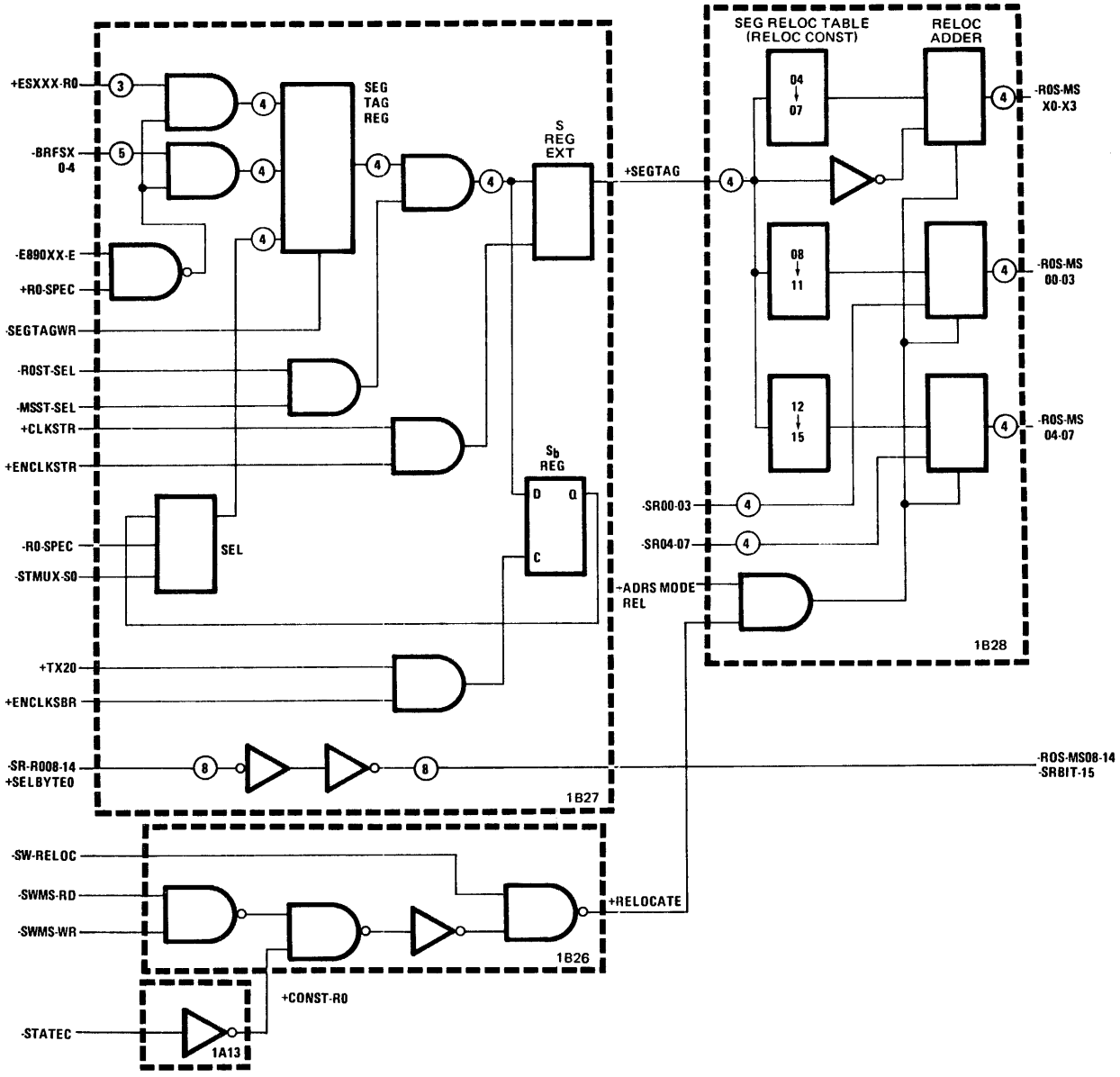


Figure 2-131. Relocation Function Logic

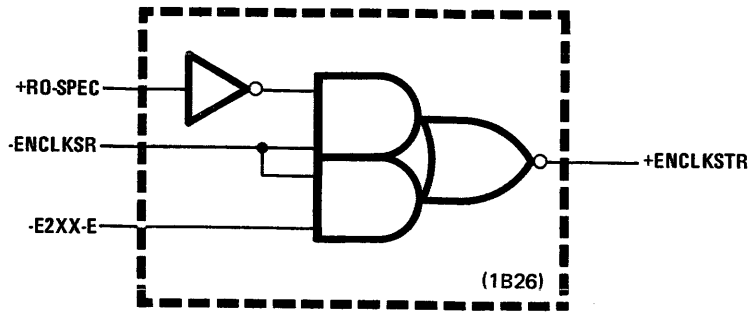


Figure 2-132. Generation of ENCLKSTR

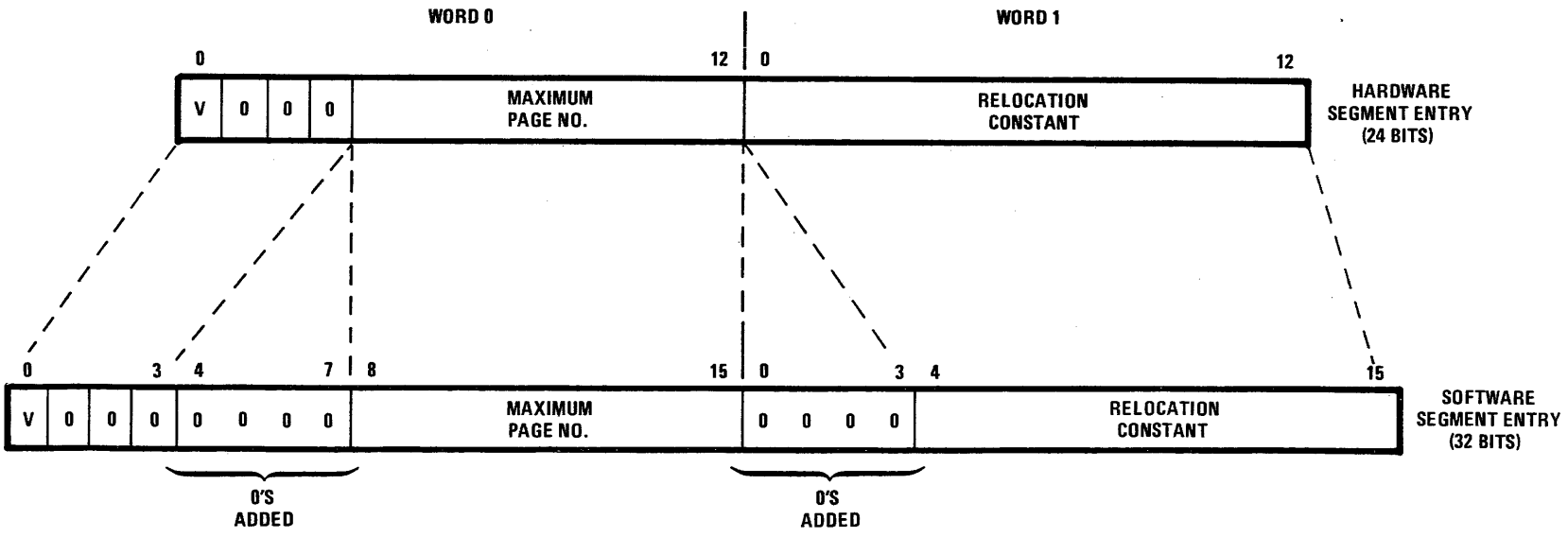


Figure 2-133. Segment Relocation Table Entry Interpretations

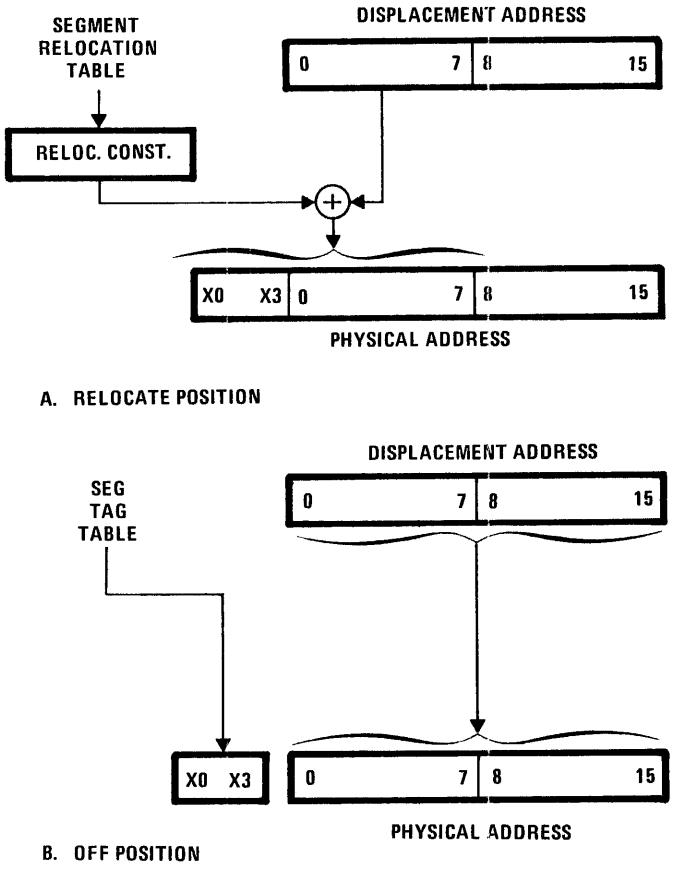
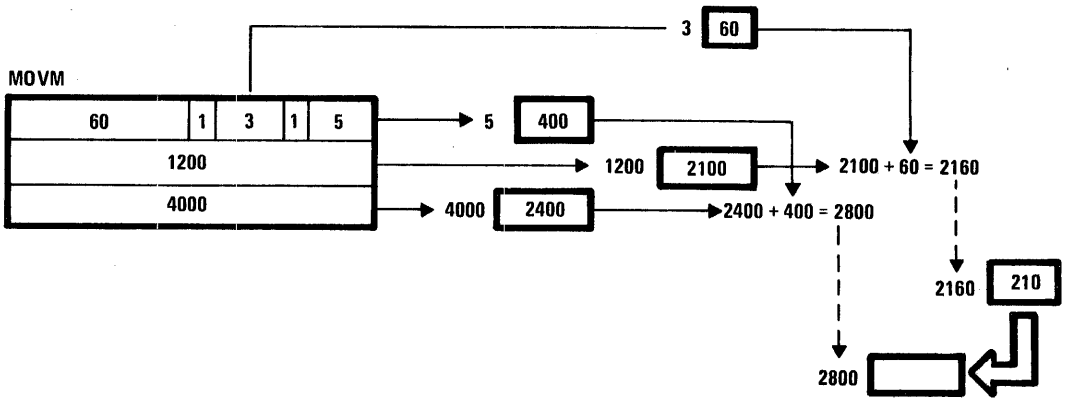


Figure 2-134. Derivation of Physical Memory Address from Console Main Storage Switch



MLI 1ST WORD READ (RNI SEQ)	LS2 Q1	READ 1ST WORD OF MLI	Q1 TAG (4) → S_b
	STA P	SAVE ADRS ON P	(1) 4 → Q1 TAG REG
	SUM Q1	UPDATE ADRS	(1) 4 → Q1 TAG REG
MLI 2ND WORD READ	LS2 Q1	READ 2ND WORD OF MLI	Q1 TAG (4) → S_b
	SUM Q1	UPDATE ADRS	(1) 4 → Q1 TAG REG
	SDW T3	1200 → T3	(2) 4 → T3 TAG REG
MLI 3RD WORD READ	LS2 Q1	READ 3RD WORD OF MLI	Q1 TAG (4) → S_b
	SUM Q1	UPDATE ADRS	4 → A1 TAG REG
	SDW T4	4000 → T4	(3) 4 → T4 TAG REG
1ST OPERAND READ + INDEX	LS2 T3	1200 → T3	4 → S_b
	IDX M	2100 → Aμ, 60 → Bμ	IDX TAG (8) → S_b
	SUM T3	2100 + 60 → T3	(4) 8 → T3 TAG REG
2ND OPERAND READ + INDEX	LS2 T4	4000 → T4	4 → S_b
	IDX R	2400 → Aμ, 400 → Bμ	IDX TAG (A) → S_b
	SUM T4	2400 + 400 → T4	(5) A → T4 TAG REG

Figure 2-135. Use of Segment Tag in Relocation and Index Operations

only the concept of indexing using segment tags. As will be shown in Figure 2-136, every write back into the Segment Tag register from Sb must be initiated by an IDX μI (except during an RNI sequence) even if an indexing operation is not performed. As the picture shows, the MLI transfers the contents of MS location 2160 (210), specified by the contents of MS locations 1200 (2100) which is addressed by the second MLI word and modified by the contents of the index register (60) specified by the MLI R_1 field (3); to MS location 2800, specified by the contents of MS location 4000 (2400) which is addressed by the third MLI word and modified by the contents of the index register (400) specified by the MLI R_2 field (5). The μI program listing shows how the segment tags are initially chosen and then used by the rest of the program to *key off* this original tag until altered by an indexing operation. Initial selection of a segment tag is performed by the LS2 μI of the RNI sequence to read the first word of the MLI. For this example, the segment tag corresponding to BRF register Q1 (containing the first MLI word address) is 4. This indicates that all MS references made by this MOVM MLI are to be made to a segment of MS addressed by the relocation constant contained in entry 4 of the segment relocation table (assuming the CONSOLE MAIN STORAGE switch on the Panel is set to the RELOCATE position).

The LS1 μI routes segment tag 4 to both the relocation table and to Sb. The following register file write μI (and all subsequent register file μI's until an indexing operation is performed) will write segment tag 4 back into the Segment Tag register corresponding to the BRF register selected by the μI so that all future references to that BRF register will key off of segment tag 4. This is shown at points (1) and (2) of the μI listing. The segment tag write at (1) is of no consequence since segment tag 4 originally corresponded to BRF register Q1 anyway. At point (2), however, segement tag 4 is written into the Segment Tag register corresponding to BRF register T3. This means that a subsequent read of T3 will not key off a tag associated with T3, but instead the tag associated with Q1. In like manner, the Q1 segment tag is written into the Segment Tag register corresponding to BRF register T4 at point (3). At point (4), however, the segment tag is changed by the preceding IDX μI, which routed a new tag (8) to Sb corresponding to BRF register 3 being used as an index register. This means that all subsequent references to register T3 will key off of segment tag 8. In a similar manner, segment tag A corresponding to BRF register 5 being used as an index register is written into the Segment Tag register corresponding to T4 at point (5).

Logic showing the flow of data into Sb and back to the Segment Tag register is shown in Figure 2-131. The segment tag is clocked into Sb in the presence of ENCLKSBR. This enable is generated during execution of either a Load S μI or an IDS μI when X=0 (the condition for indexing). The output from Sb is fed back to the Segment Tag register through a selector. For writing into the register from Sb both selector enables RO-SPEC and ST-MUX are high. Register file write enable $\overline{\text{SEGTAGWR}}$ is generated for a segment tag rewrite by the logic shown in Figure 2-136. As discussed in the footnote to Figure 2-135, all writes from Sb back into the Segment Tag regis-

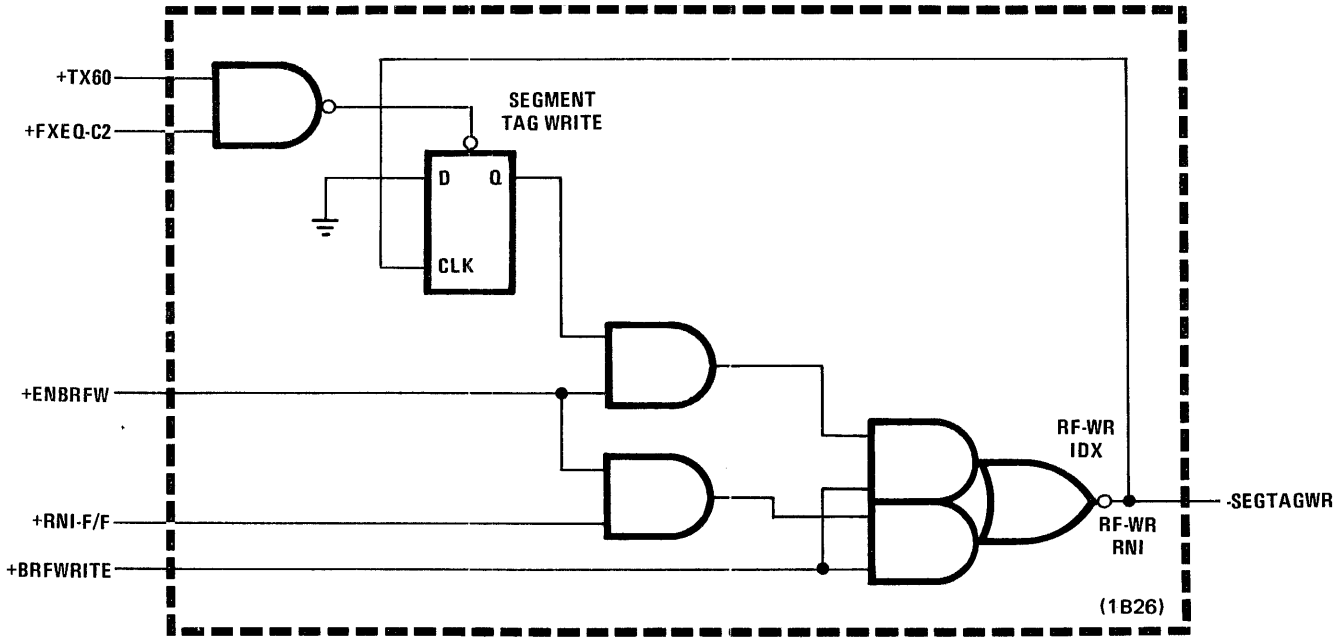


Figure 2-136. Generation of $\overline{\text{SEGTAGWR}}$ for Segment Tag Re-Write

ter must be initiated by an IDX (0,2) μI whether or not an indexing operation was actually performed, except during an RNI sequence Therefore, the write enable is generated by two different conditions. During an RNI sequence, $\overline{\text{SEGTAGWR}}$ is generated by RNI-F/F, indicating that the RNI sequence is being performed, and ENBRFWR and BRFWRITE, indicating that a register file write μI is being performed and the time during execution of the μI that the register is to be written into. During the sequences following the RNI sequence, the index mechanism that generates $\overline{\text{SEGTAGWR}}$ must be used. Execution of an IDX μI sets the Segment Tag Write flip-flop. Then, when the register file write μI is executed to perform the actual write back into the file, ENBRFW and BRFWRITE are generated which, in combination with the flip-flop output, generate $\overline{\text{SEGTAGWR}}$. As soon as this enable is generated, the flip-flop is cleared to de-activate the write enable until the next segment tag rewrite is initiated.

Protection

Protection is accomplished during the course of performing relocation. This protection is implemented in three different ways: validity bit protect, bounds protect, and write/read protect. The validity bit and bounds protect evaluations are made on the contents of the left-most word read from a particular entry in the segment relocation table. The write/read protect evaluation is made on the contents of the protection matrix. All three types of protect depend on whether or not protection is defined for a particular processor, as determined by the contents of the Address Mode register. Each of the three protection schemes is discussed in the following paragraphs, referencing Figure 2-137.

Validity Bit Protect

Validity bit protect is performed by examining the validity (V) bit (bit 0) of the left-most word of the segment relocation entry read by the four SEGTAG bits. If this bit is set ("1"), an access (either read or write) may be made to the MS segment defined by the relocation constant by generating ROACCESS. This bit offers the operating system a more convenient means of preventing access to a MS segment than using the protection matrix described below. If the V bit is not set ("1"), ROACCESS goes low to inhibit the access and a jump is made to a bounds error trap routine.

The validity protect scheme is effective only if relocation is enabled for the particular processor state. This relocation enable is furnished by the Address Mode register, which specifies whether a particular processor is enabled for relocation and/or protection. This 16-bit register is composed of an 8-bit relocate (R) field and an 8-bit protect (D) field, as shown in Figure 2-138. When a particular bit is set in either field, the corresponding processor is enabled for the specified relocate or protect condition. As the figure shows, four different relocate/protect conditions are possible depending on the combination of R and D bits. These conditions are discussed below:

1. R·D = 0·0 — If neither relocation nor protection is enabled for a particular processor, the physical memory address is made up of the displacement address and the segment tag value without any reference to a segment relocation table entry. This is the condition defined when the CONSOLE MAIN STORAGE switch is set to OFF position. In addition, none of the protection checks will be made.

2. R·D = 0·1 — This condition is unique in that even though protection is enabled, none of the protection checks is made. The reason is that without relocation, a validity bit or bounds check cannot be made. Without these checks, a write/read protect check is not needed so it is not made either. Essentially, then, this condition becomes the same as an R·D = 0·0 condition.

3. R·D = 1·0 — If relocation only is to take place, the segment relocation table entry will be used to develop a physical memory address as discussed under the above Relocation paragraph. Because the relocation table is accessed, the validity bit check will be performed automatically even though protection is not enabled. Any other protection check, however, will not be performed.

4. R·D = 1·1 — Relocation and protection will take place using the segment relocation table, and the protection matrix as described in the Write/Read Protect paragraph.

For the validity bit check, the R bit corresponding to the present processor state is ANDed with the V bit from the segment relocation table to set up the protect condition previously described.

Bounds Protect

The bounds protect check is made by comparing the page number portion of the displacement address (bits 0

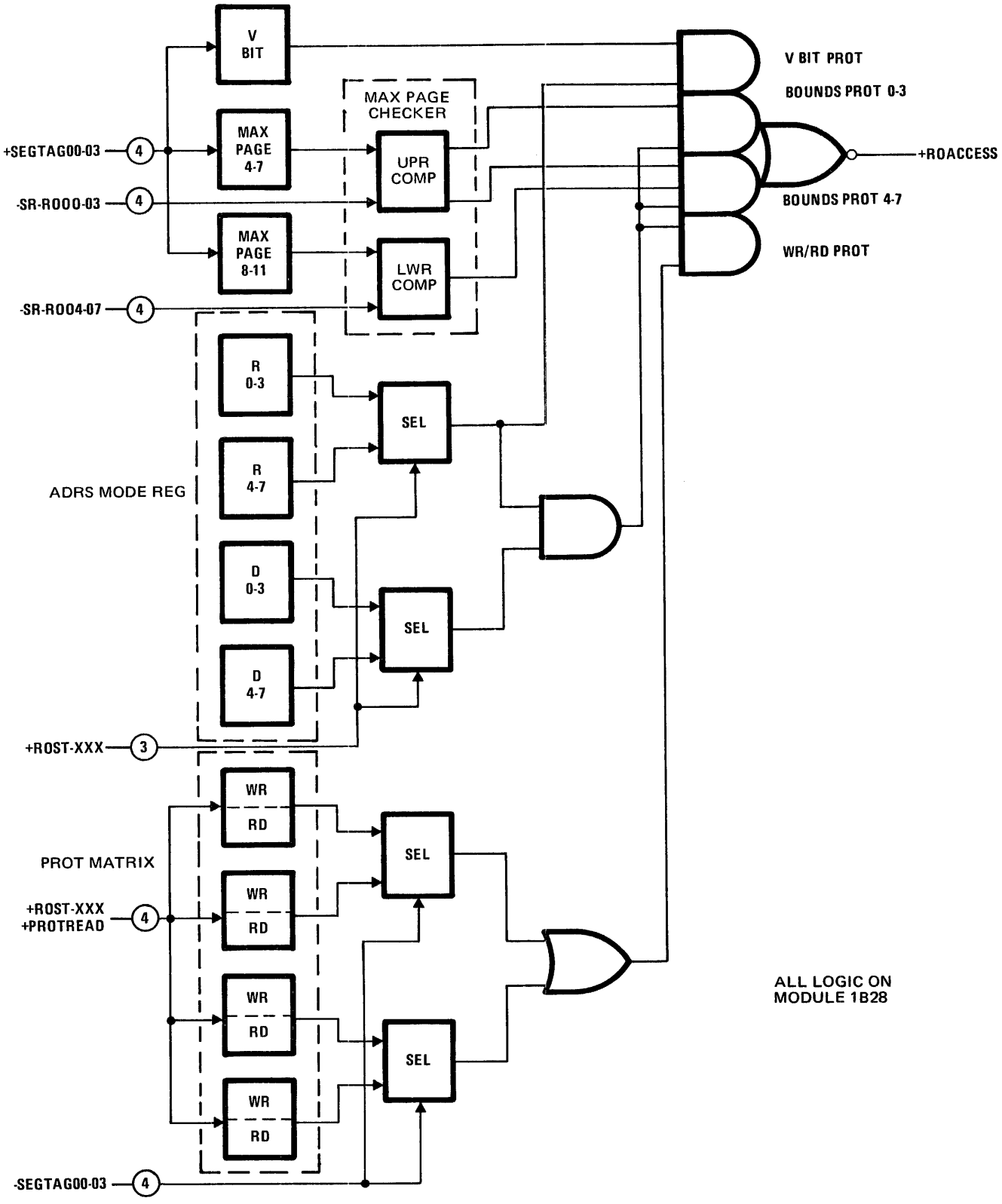
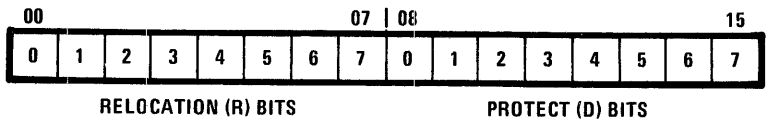


Figure 2-137. Protect Function Logic



R	D	
0	0	NO RELOCATION, NO PROTECTION
0	1	NO RELOCATION, PROTECTION
1	0	PROTECTION, NO RELOCATION
1	1	RELOCATION, PROTECTION

Figure 2-138. Address Mode Register

through 7 of S) with the maximum page number portion of the segment relocation table entry. The maximum page number, in effect, constitutes the upper boundary page address of the MS segment to be accessed. If the page number in S is greater than the maximum page number, an MS parity error condition is generated by forcing ROACCESS low. The check is made by two comparators, each comparing 4 of the 8 bits comprising the page number in S and the maximum page number. The check is made only if both the relocate and protect conditions are enabled from the Address Mode register.

Write/Read Protect

The write/read protect check is made by examining the state of a read and write protect bit assigned to each entry of the segment relocation table. These read and write restrictions are accomplished by means of the protection matrix. The protection matrix consists of a 16-bit Write Protect register and a 16-bit Read Protect register assigned to each of the either processor states, as shown in Figure 2-139. Bits 0 through 15 of each register represent the 16 segment entries 0 through F of the segment relocation table. A processor may access a segment in MS only if that segment number in the appropriate Write Protect or Read Protect register of the protection matrix is a 0.

The protection matrix consists of four storage elements, each element storing four bits (segment numbers) of each of the 16 registers. A particular register is selected by the three processor select (ROST) signals and the PROTREAD signal. The PROTREAD signal serves a dual function of selecting the Read Protect register during E0 through E3 of the processor's time slice (PROTREAD high), and the Write Protect register during E4 through E7 (PROTREAD low). In this way, both write and read protect checks are made on the selected MS segment. Finally, the particular segment number of the selected processor's Write Protect and Read Protect registers is selected by the four SEGTAG signals through two selector elements. The resultant protect bit selected is ANDed with the R·D = 1·1 signal from the address mode register to generate ROACCESS if the protect condition is met.

The result of these three protection schemes is to drive ROACCESS high if the protect condition is met. If the protect condition is not met, ROACCESS goes low to inhibit an MS write operation, if requested, and generate an MS parity error trap condition in exactly the same manner as the basic protect feature discussed in the paragraph titled Basic Storage Protection Feature.

Parity Error Register Extension

The Parity Error (PE) register extension functions as an upper four-bit extension of the 16-bit PE register in the Group II ERF. In this regard, it displays the upper four bits of the physical address at which the last PE occurred. Logic for the PE register extension is shown in Figure 2-140. The upper four address bits come from either one of two sources, depending on the setting of the SYSTEM/PHYSICAL switch on the System Control Panel. If in the PHYSICAL position, selector enable SYSTEM goes low and the PE register extension is loaded with the upper four bits of the physical address via the four ROS-MS signals. This physical address may be either the relocated or un-relocated system address, depending on the setting of the CONSOLE MAIN STORAGE switch. If the SYSTEM/PHYSICAL switch is in the SYSTEM position, enable SYSTEM goes high and the PE register extension is loaded with the upper four bits of the system address regardless of the setting of the CONSOLE MAIN STORAGE switch. The address bits are clocked into the register via CLKPE, at the same time that the PE register in the ERF is clocked with the lower 16 bits of the address.

JOB ACCOUNTING FEATURE

A block diagram of the Job Accounting feature is shown in Figure 2-141. The eight 32-bit job accounting registers are contained in four storage elements as shown. Each

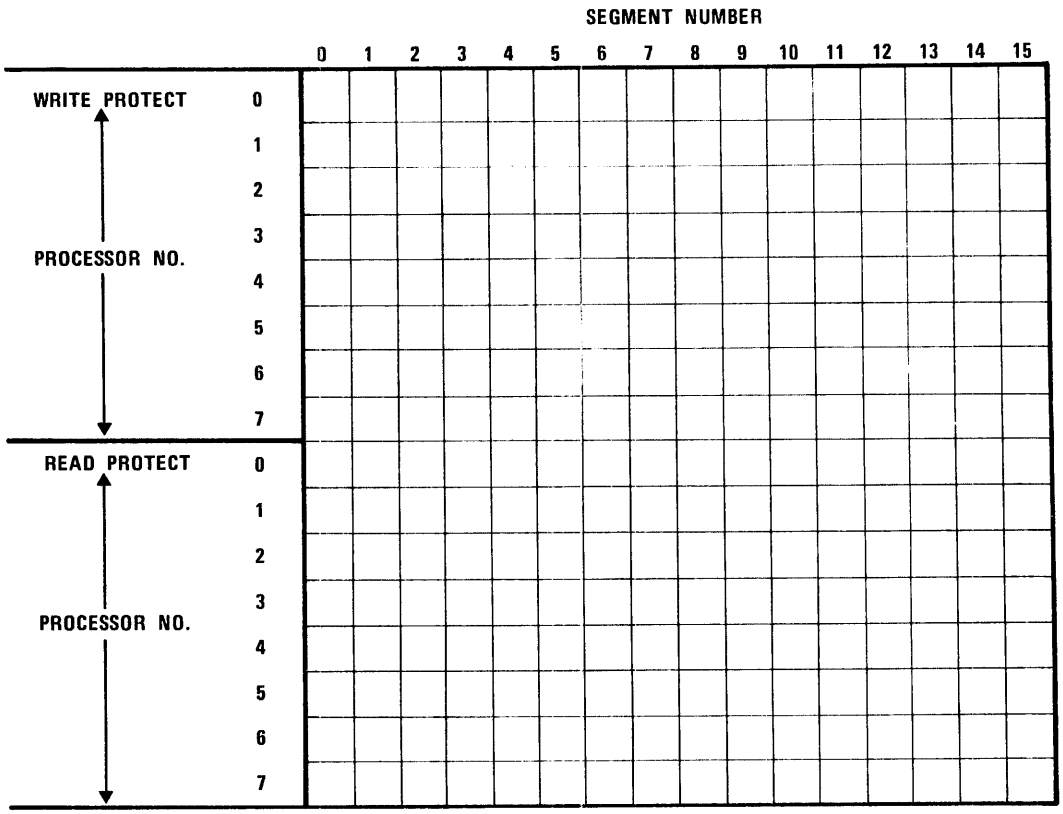


Figure 2-139. Protection Matrix

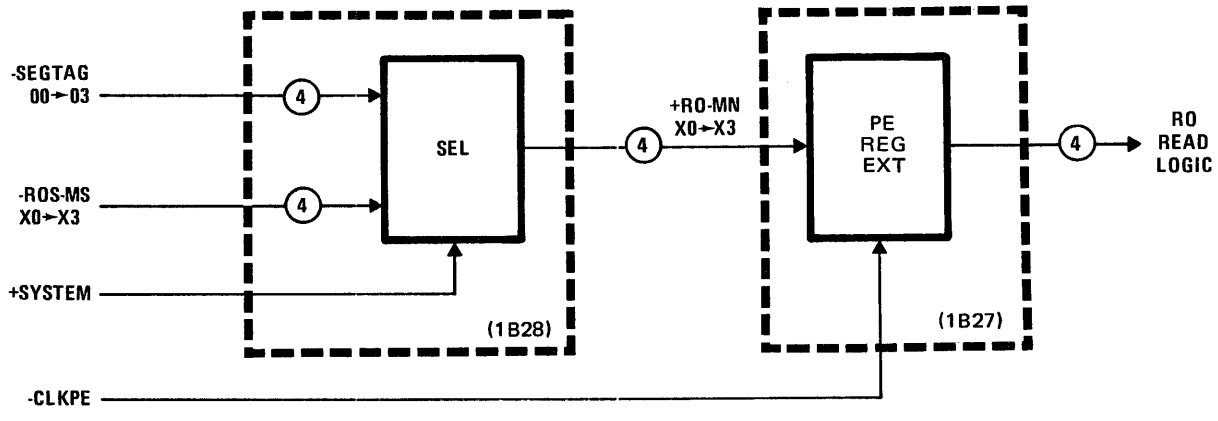


Figure 2-140. Parity Error Tag Register

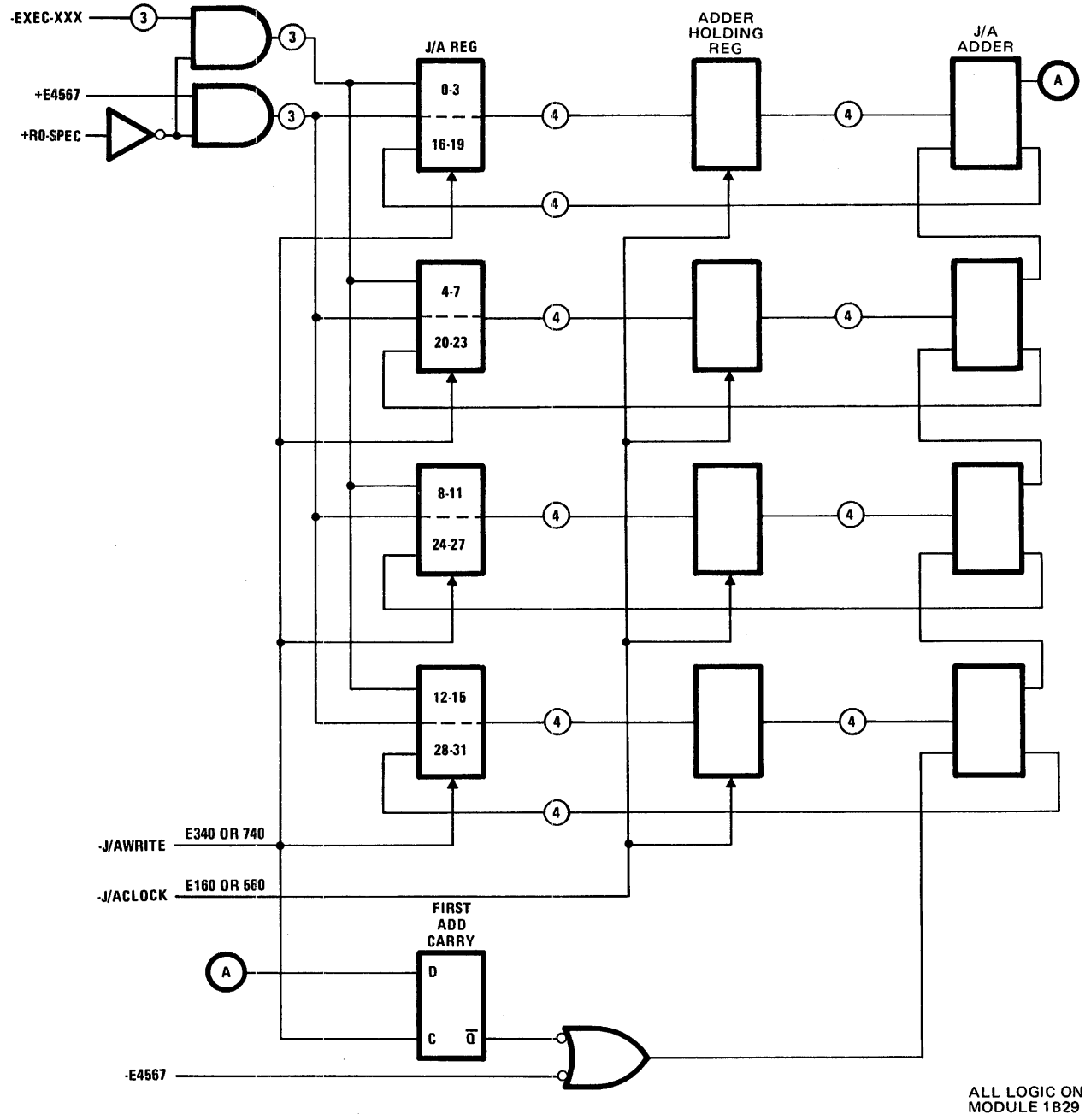


Figure 2-141. Job Accounting Feature Block Diagram

register consists of two 16-bit words, each individually addressable, as shown in Figure 2-142. The storage elements are interconnected so that each element stores the corresponding four bits of each 16-bit word. For example, the top-most element in Figure 2-141 stores the left-most four bits of both word 0 (bits 0 through 3) and word 1 (bits 16 through 19). During normal operation each element is addressed in two halves, wherein word 1 if a register is read, incremented by 1 and written back during the first half of a time slice (E0 through E3), followed by a read, increment, and rewrite of word 0 during the second half of a time slice. Occurrence of each operation for incrementing each word of a register is shown in Table 2-15.

During normal operation, a particular register is addressed by the processor number specified by the three EXEC signals, and the particular word of the addressed register by signal E4567. During the first half of a time slice, E4567 is low to read word 1 from the register. This word is clocked into the adder holding register by J/ACLOCK at E160. The holding register is used to hold the word while the word is incremented by 1. This incrementation occurs as soon as the holding register is loaded by unconditionally routing the word to the adder. The +1 added to the word in the adder is generated through a NOR gate from two different sources, depending on whether word 1 or word 0 is being incremented. During an increment of word 1, the +1 is obtained from E4567, which is inverted to the 1 state through the NOR gate. At E340, the incremented word is stored back into the register by J/AWRITE.

At E400, E4567 goes high to read word 0 of the selected register. Incrementing and subsequent rewriting of this word is performed in the same manner as for word 0, except for the clock times and the source of +1. During a word 0 increment, the +1 results from a carry-out, if generated, from the most significant bit (MSB) stage of the adder, indicating that the +1 added to word 1 produced an overflow. The carry-out sets the First Add Carry flip-flop by $\overline{\text{J/AWRITE}}$ (E340 time). The resultant low from the Q output is fed through the NOR gate and added to word 1 now in the adder.

Generation of $\overline{\text{J/AWRITE}}$ is accomplished by the logic shown in Figure 2-143. The signal is generated for two different conditions: during normal (MS reference) operation to update the job accounting register contents by 1, and during Panel-initiated operations to clear the job accounting register. For either condition, $\overline{\text{J/AWRITE}}$ is generated at both E340 and E740 by the combination of timing signals E3 or E7 and BRF WRITE. Signal BRFWRITE furnishes a pulse width of 60 nanoseconds, starting at t40 of both E3 and E7. The two conditions during which $\overline{\text{J/AWRITE}}$ is generated are defined by enables NOT NULL and NOT CONS EXC J/A SPECIFIED. Specifically, these enables eliminate all other conditions during J/AWRITE could be generated: a null state and a Panel state where an operation other than a job accounting register reference (either read or write) has been initiated. A further resolution of the job accounting reference specified by the Panel is provided by signal L← J/A READ. This signal inhibits J/AWRITE during a job account register read operation, or discussed in the paragraph titled Register Read. Therefore, the signal is generated specifically for a write operation.

REGISTER READ/WRITE

Reading and writing registers of the RO during other than normal (MS reference) operations is performed under program control by the Read Register Option (RRO) and Write Register Option (WRO) MLI's, and under manual control from the System Control Panel. The RRO and WRO MLI's are each two-word MLI's, of which the second word of the MLI addresses a particular register by register group number; processor number, if applicable; and, in the case of two-word registers, a designator that selects either word 0 or word 1 of the register. The System Control Panel permits selection of a RO register by setting the above register select information into the CONSOLE ADDRESS REGISTER DISPLAY pushbuttons. This section describes reading and writing RO registers by MLI's only. Reading and writing RO registers from the Panel is described in the paragraph titled MS/RO and RF Read and Write.

The 16-bit format for addressing registers of the various register groups, including the ECC feature, is shown in Figure 2-144. Note that for selection of any RO register, bits 0 through 3 are always "0's". The register group numbers, defined by bits 4 through 7 of the address, are listed in Table 2-16. Note that each register of the ECC feature may be selected by two adjacent group numbers. The complete 16-bit address for each RO register is shown in hexadecimal form in Figure 2-145.

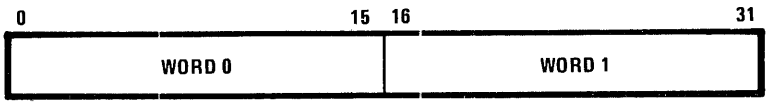


Figure 2-142. Job Accounting Register Format

Table 2-15. Occurrence of Job Accounting Register Increment Operations

Time	Signal	Operation
WORD 1 UPDATE		
ER000-E300	$\overline{\text{E4567}}$	READ BITS 16-31
E160	$\overline{\text{J/A CLOCK}}$	J/A REG→HOLDING REG→ADDER
E000-E300	$\overline{\text{E4567}}$	+1 FROM $\overline{\text{E4567}}$→ADDER
E340	$\overline{\text{J/A WRITE}}$	ADDER→ J/A REG
WORD 0 UPDATE		
E400-E700	E4567	READ BITS 0-15
E560	$\overline{\text{J/A CL OCK}}$	J/A REG→HOLDING REG→ADDER
E340	$\overline{\text{J/A WRITE}}$	+1 FROM 1st ADD CARRY FF
E740	$\overline{\text{J/A WRITE}}$	ADDER→ J/A REG

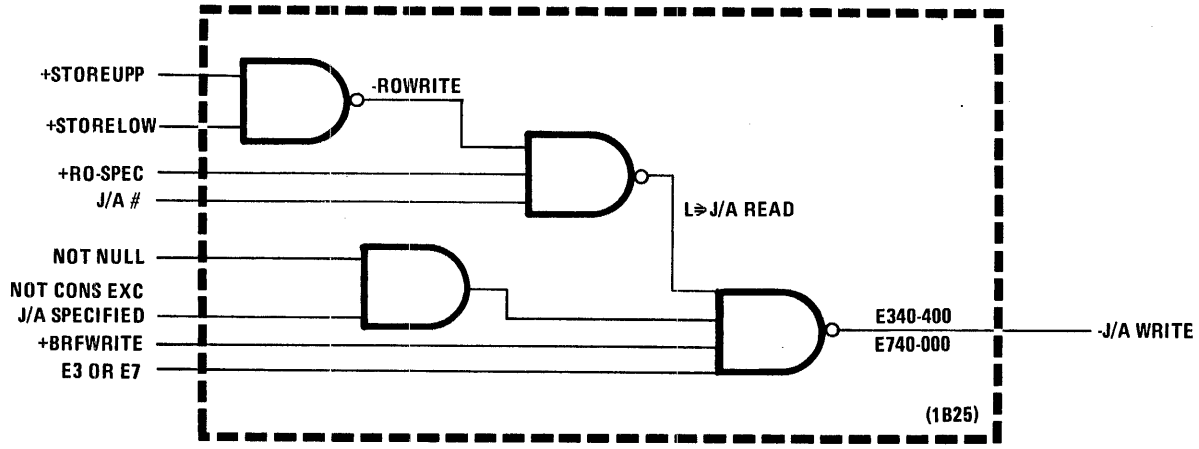


Figure 2-143. Generation of $\overline{\text{J/A Write}}$

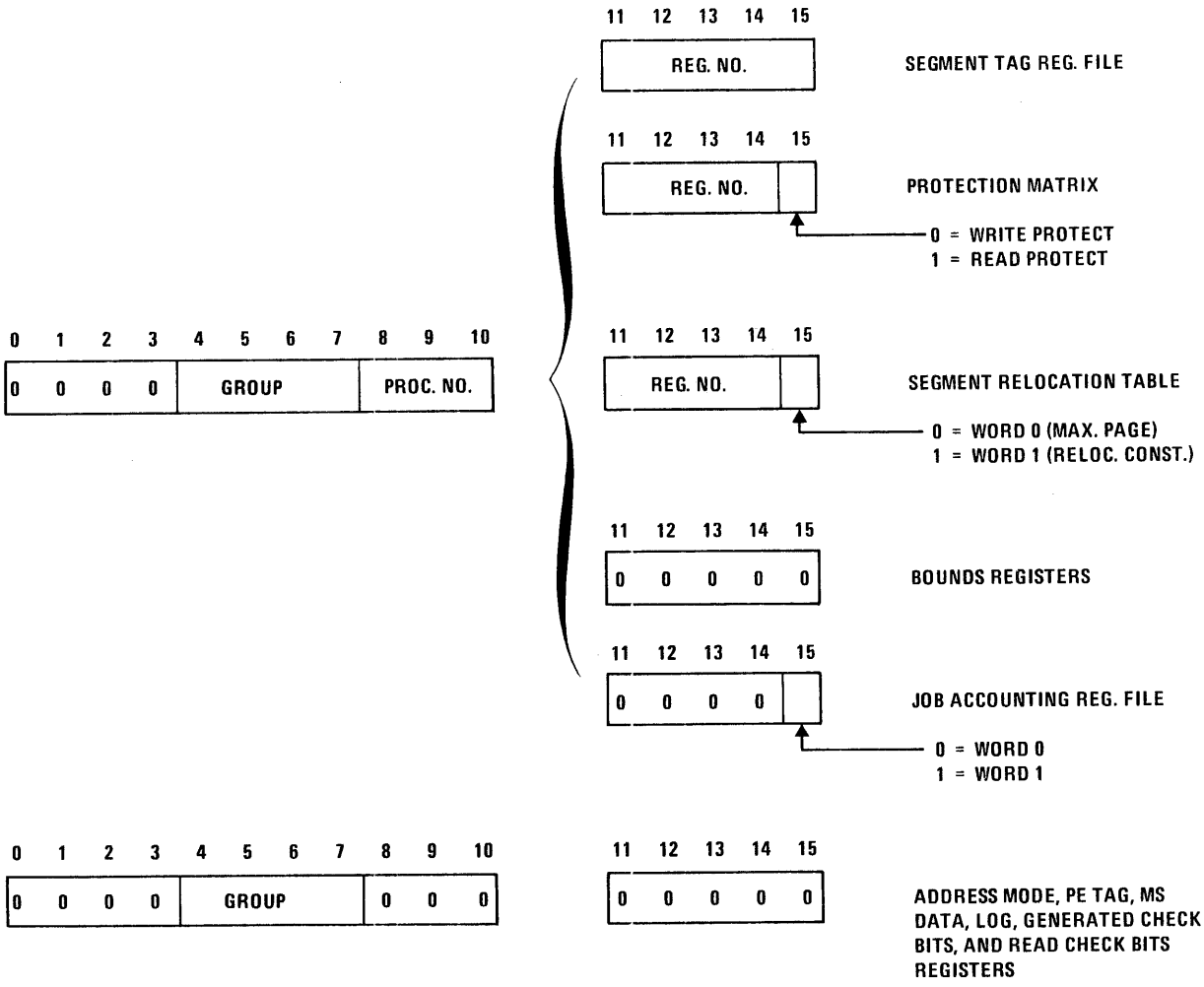


Figure 2-144. RO Register Address Format

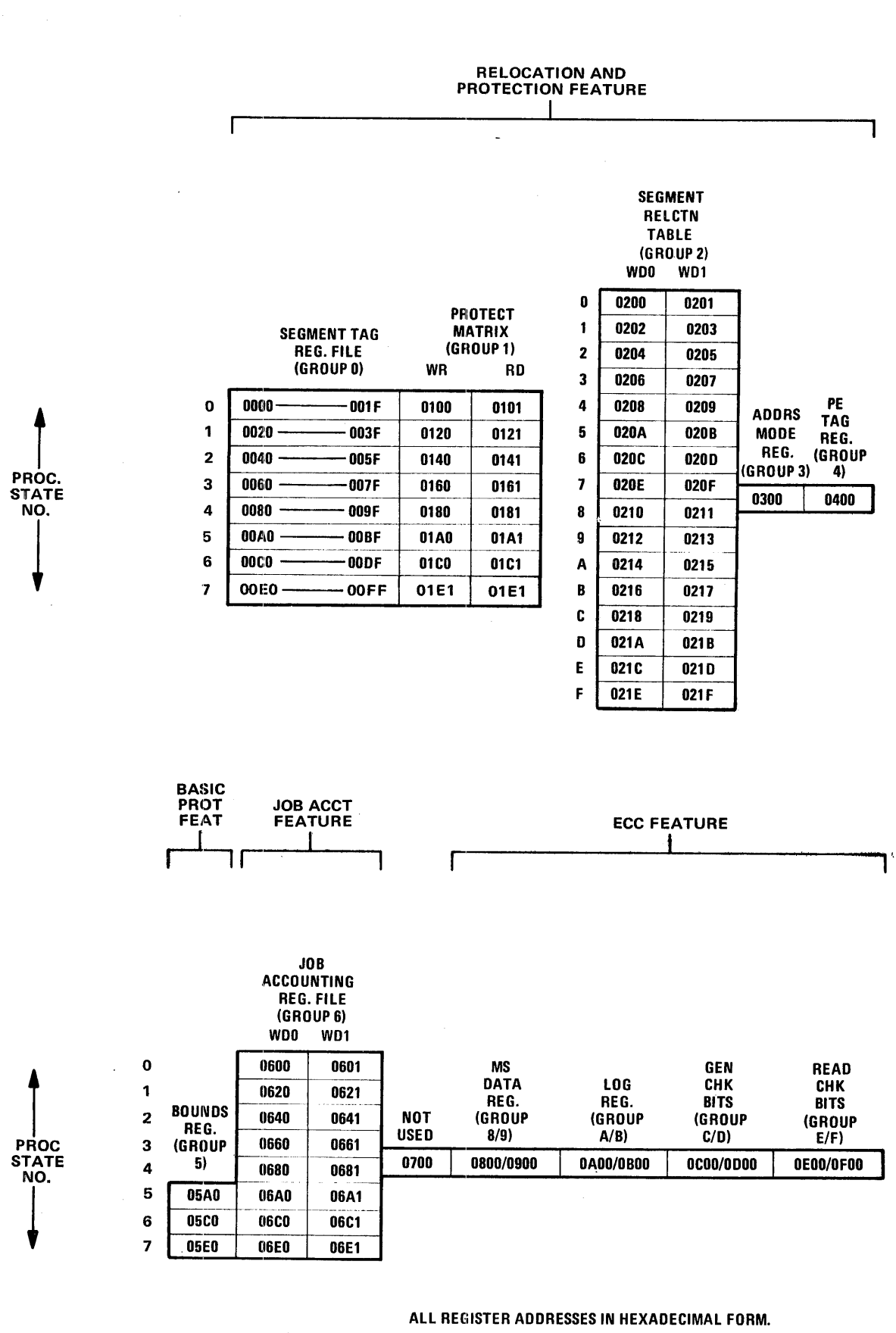


Figure 2-145. Register Option Registers and Associated Addresses

Register Read

Reading a RO register by means of an MLI is accomplished by the logic shown in Figure 2-146. The basic approach is to select a register from the processor number (bits 8, 9, and 10), register number (bits 11 through 15), and register word designator (bit 15) of the address contained in S. The data read travels over one of six data paths from each register group to a selector, which selects a particular data path by the register group number (bits 4 through 7) contained in S. This intermediate selection, which is also fed with the register group number. The final path selected routes the information read to the data fan-in in the MS interface logic. Selection of any RO register requires that enabled RO-SPEC be in the high state, indicating that a reference is being made to the RO for the express purpose of reading or writing an RO register.

Segment tag registers are selected by both processor number (SR-RO bits 8, 9, and 10) and register number (SR-RO bits 11 through 14 and SELBYTE0). The segment tag selected is passed to the PE/segment tag selector, which is also fed with output from the PE register extension. (There is only one PE register extension; therefore, selection of this register is made by the register group select bits alone.) This selector selects either the Segment Tag or the PE register extension contents, depending on the state of SELSTAG, as shown in Table 2-17. The result is routed to the RO multiplexer as ST/PE bits.

Entries in the segment relocation table are also selected by processor number and register number, via a corresponding segment tag as for normal MS reference operation. The resulting 24 bits of the selected entry are sent to the relocation/protection register fan-in. The 24 inputs to the fan-in are applied as two input groups of 12 bits each, corresponding to the two words that make up each entry in this table. The fan-in is also fed with outputs from a selected register in the protection matrix and from the Address Mode register. One of the 16 two-register entries in the protection matrix is selected by a processor via the ROST select bits. Selection of either the read or write register of the selected entry is made by bit 15 of S, which generates PROTREAD. If PROTREAD is high, the read register is selected; if PROTREAD is low, the write register is selected. Selecting one of the four inputs groups to the ROBIT selector is performed by the two SELRO select bits, as shown in Table 2-17. The output of this fan-in is fed to the RO multiplexer as ROBIT bits.

If the Basic Storage Protection feature is installed in place of the Relocation and Protection feature, the ST/PE bits are not generated and the ROBIT are developed from a selection of one of the three bounds registers. This selection is made from the processor number via the ROST select bits.

Table 2-16. Register Option Register Group Numbers

Feature	Register Group	Group No. (Hexadecimal)
Relocation and Protection Feature	Segment Tag Register File	0
	Protection Matrix	1
	Segment Relocation Table	2
	Address Mode Register	3
	PE Register Extension	4
Basic Protection Feature	Bounds Registers	5
Job Accounting Feature	Job Accounting Register File	6
ECC Feature	MS Data Register	8/9
	Log Register	A/B
	Generated Check Bits	C/D
	Read Check Bits	E/F

Table 2-17. First-Level Selection of Register Groups

Selector Name	Selector Signals and States		Register Group Selected
ST/PE	SEL TAG		
	0		PE Register Extension
	1		Segment Tag Register
RO BIT	$\overline{\text{SELRO-S0}}$	$\overline{\text{SELRO-S1}}$	
	0	0	Address Mode Register
	0	1	Protection Matrix
	1	0	Segment Relocation Entry, bits 12-33 (Relocation Constant)
	1	1	Segment Relocation Entry, bits 0-11 (V Bit and Max. Page No.)

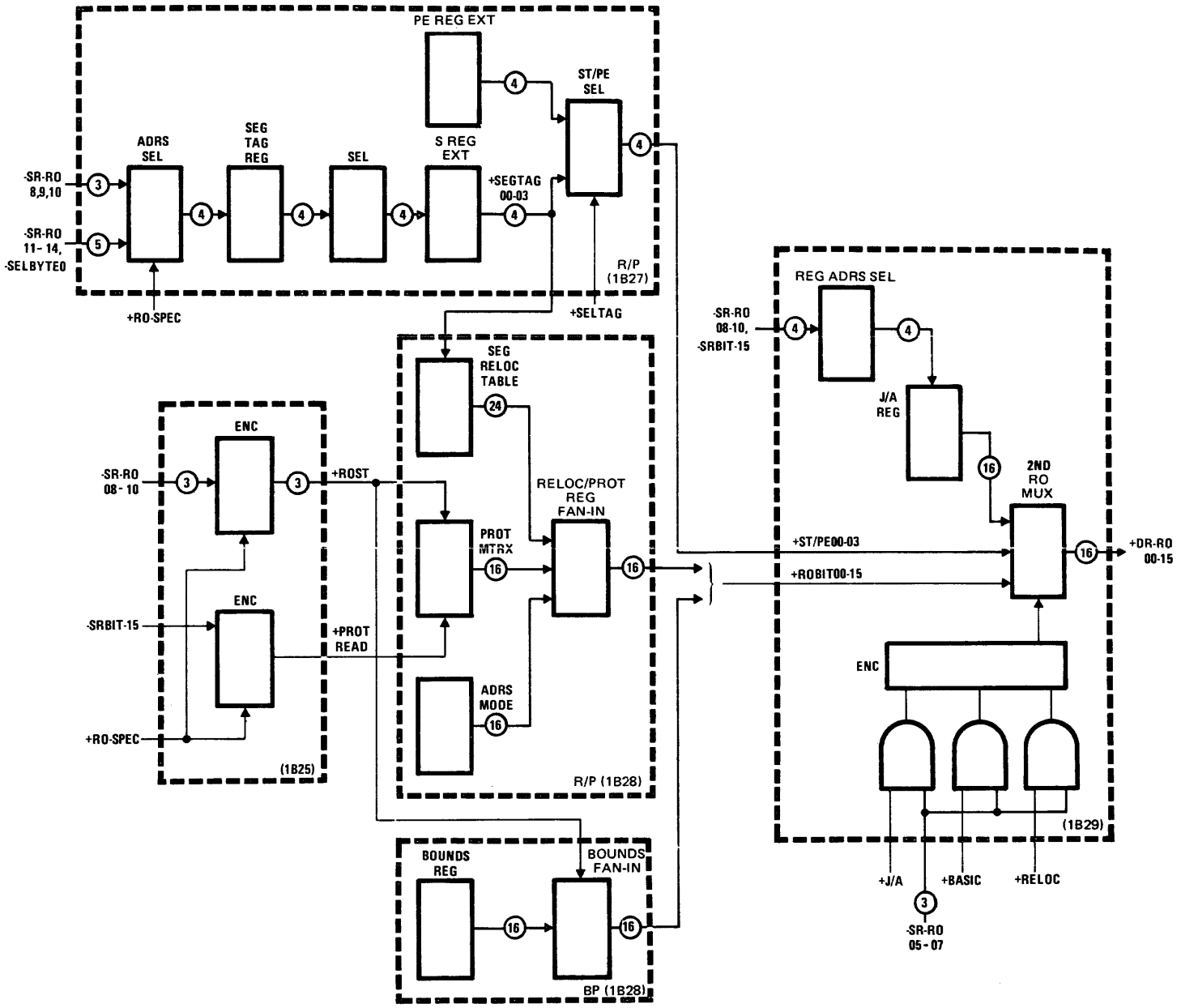


Figure 2-146. Register Option Register Read

The ST/PE and ROBIT bits are sent to the RO multiplexer for final selection and routing to the data fan-in logic. This multiplexer is also fed with outputs from a selected register in the Job Accounting feature. The Job Accounting register is selected by S register bits 8, 9, and 10, and either word 0 or word 1 of the register is selected by bit 15 of S. One of the three inputs to the RO multiplexer is gated by an encoded combination of register group select bits 5, 6, and 7 from S and master enables J/A, BASIC, and RELOC. (Bit 4 of the register group select field is not needed since it is always 0 for the non-ECC features of the RO.) These three master enables provide an over-all select enable for the three non-ECC features by defining which of the three features are present or enabled for selection in the system. If either the Basic Storage Protection or the Relocation and Protection feature is present, the corresponding master enable is connected to the high state (+5 vdc) to enable selecting the feature by bits 5, 6, and 7. The Job Accounting feature will always be present in the system, since the module containing this feature is also used for register read operations; however, the job accounting registers can be selected for read operations only if the master enable J/A is connected to the high state. Any feature not present or not available for selection is disabled by connecting its master enable to the low state (ground).

During a read of the job accounting registers, the update operation must be inhibited so that a steady-state value from the register may be reea. This is accomplished by inhibiting the $\overline{\text{J/AWRITE}}$ signal. When this signal is inhibited, the contents of the selected register are updated in the normal manner by adding +1; however, they are prevented from being written back into the register to keep the register contents from being altered during the read operation. The signal is inhibited as shown in Figure 2-143 by applying a low to the two AND gates which generate $\overline{\text{J/AWRITE}}$. This low is generated by RO-SPEC if a read of the job accounting registers is specified ($\overline{\text{ROWRITE}}$ is high). This low overrides the E times that would normally generate $\overline{\text{J/AWRITE}}$ for the duration of the time slice during which the read is being performed.

Register Write

Writing a RO register by means of an MLI is done so by the logic of Figure 2-147. The register is selected by register group number, processor number, and register number Upon being selected, data from the D register is entered in the register in the presence of a corresponding write enable generated from the feature number. These write enables are generated by the logic shown in Figure 2-148. Each write enable is generated by a corresponding register gorup select signal decoded from bits 5, 6, and 7 of S, and a RO write enable generated at E5 time from RO-SPEC, STOREUPP, and STORELOW. Two write enables are generated for the segment relocation table to allow separate writes of word 0 and word 1 in each entry. The word designator is supplied by bit 15 of S, as follows:

Bit 15 = "0" – write word 0

Bit 15 = "1" – write word 1

The Address Mode register requires two write enables because of the logic used to implement this register. Both enables are generated simultaneously for identical conditions. The Address Mode register can also be written by a master clear operation, for the purpose of clearing the register to 0's.

Referring back to Figure 2-147, the Segment Tag register to be written into is selected by bits 8, 9, and 10 of S (processor number) and bits 11 through 14 of S and SELBYTE0 (register number). Data to be written into the selected register is derived from bits 12 through 15 of the D register in the MS interface logic through a selector. For routing this data to the selected segment tag register, selector enables RO-SPEC and $\overline{\text{STMUX-S0}}$ are "1" and "1", respectively. Writing into other RO registers are selected in a similar manner, much the same as for reading the registers, except for the additional write enable required.

Writing into a selected job accounting register is done so for the specific purpose of clearing the register. This is done by generating RO-WRITE and combining it with RO-SPEC and the job accounting feature number, as shown in Figure 2-149, to generate both a low and a high output. The low output is applied to the function select input of the four adder elements, and the high output to the mode control input of the adder elements. The state of these two inputs determines how the four "adder" elements, which are really multi-purpose function generators, are to operate. During normal job accounting update operation, these elements function as adders. During a register write operation, however, the state of these two inputs is altered as discussed to make the elements generate all "0's" on their outputs, regardless of the inputs. These 16 "0's" are routed back to the selected register to clear the register upon occurrence of the $\overline{\text{J/AWRITE}}$ enable.

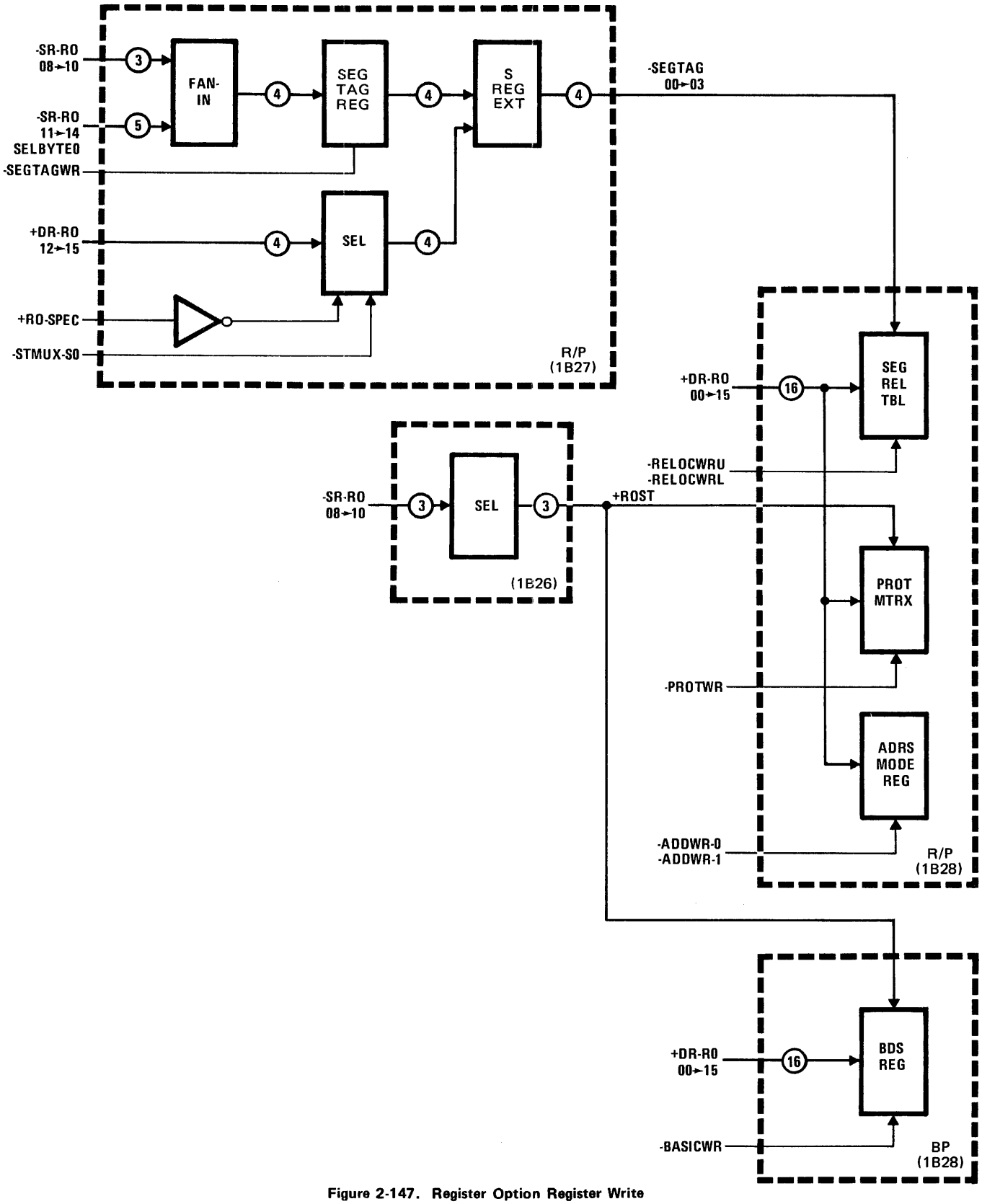


Figure 2-147. Register Option Register Write

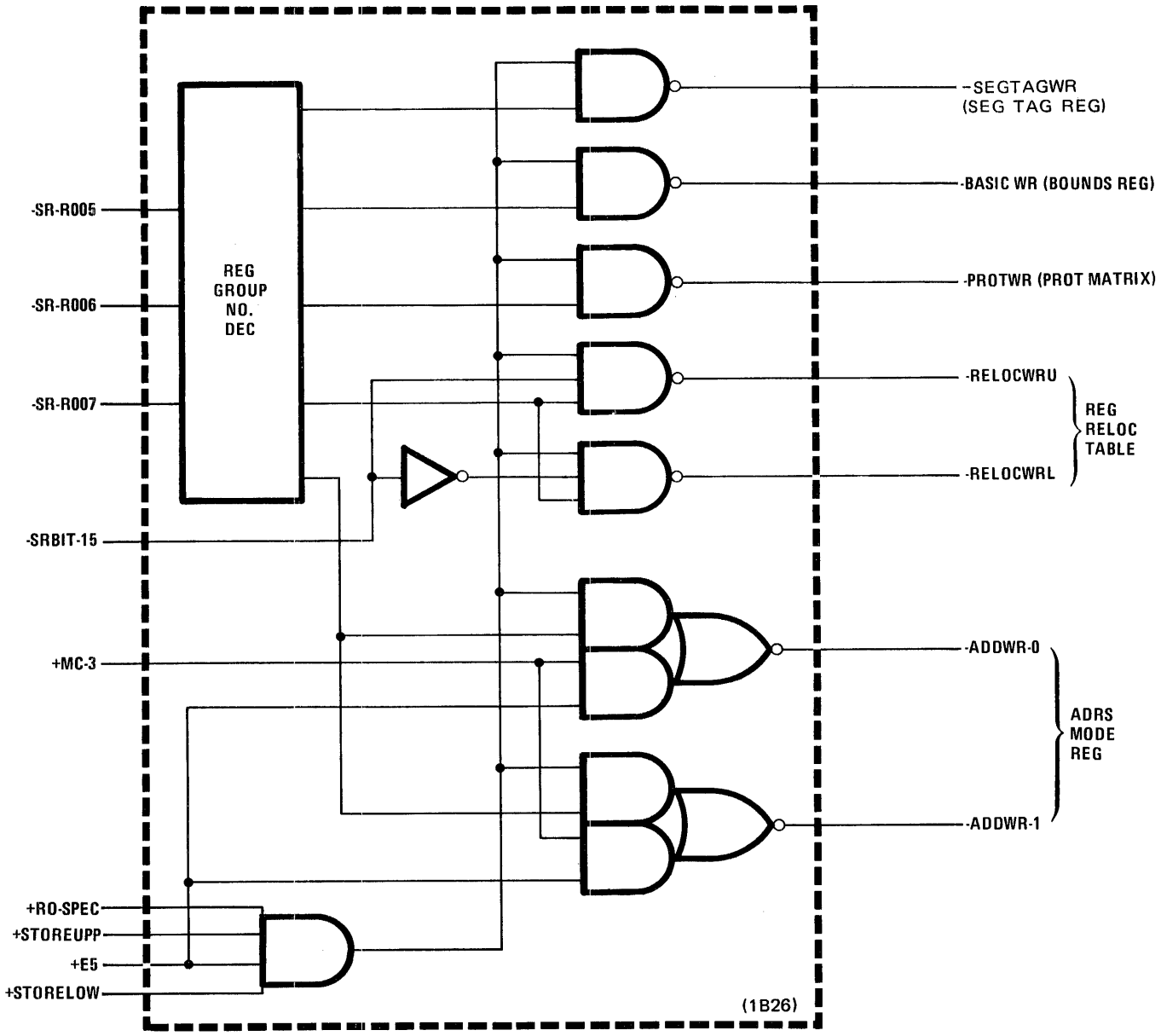


Figure 2-148. Register Group Write Enables

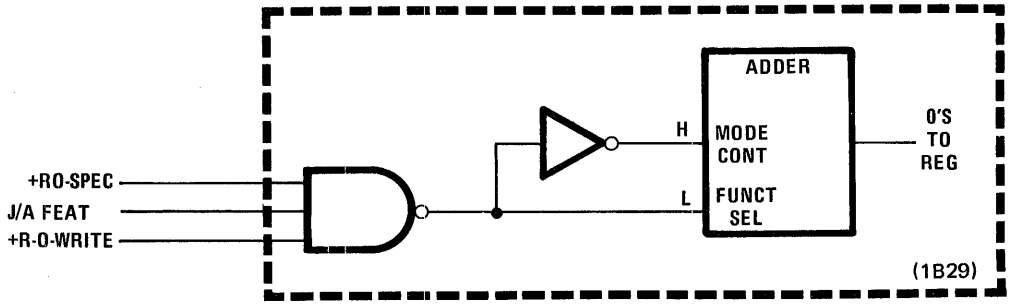


Figure 2-149. Writing 0's into Job Accounting Registers

MLI DECODE AND STORE/SAVE

The machine language instruction (MLI) decode and store/save logic performs a first-level (format) decode on the new MLI read from main storage (MS) for purposes of branching to a routine required to read the first MLI operand. The logic also saves the MLI from one time slice to the next until execution of the MLI is completed. A block diagram of the logic involved is shown in Figure 2-150. The MLI read from MS during the RNI sequence is routed to the MLI decode and store/save logic by a SDW μI with F_{RF} in Group I of the Extended Register File (ERF) as the destination. The μI reads the MLI from MS and passes it to the Format Jump (FRJ) decoding logic to determine the format of the MLI and obtain the first operand to be processed by the MLI. The μI also routes it to the F_b register to be saved in the assigned F_{RF} of the ERF at the end of the time slice, and to the F register via the F register fan-in logic. This auxiliary operation of routing the MLI to the F register provides for immediate modification of the MLI, if necessary, during the present time slice.

FRJ DECODE

The FRJ decode logic performs a first-level decode of an MLI to determine its format. The format of an MLI consists of two parts: instructions type and addressing mode. The type of MLI (register/register, memory/register, and so forth) is defined by the class of its function code (2X, 3X, 4X, and so forth). The addressing mode is indicated by the state of bits 8 and 12 of the MLI, which determine whether the operands are to be obtained from MS or a file register. Upon determining the format, the decode logic generates a two-digit hexadecimal address. This address points to one of 256 locations in an address table. This address table consists of ten bipolar storage elements, each element storing one bit of 256 words. Each word, therefore, consists of ten bits: a parity bit plus the right-most nine bits of a branch address to a routine required to read the first MLI operand. The nine bits from the address table are appended to the left-most five bits of the 14-bit CS address. These left-most five bits are also generated by the FRJ decode logic, as shown in Figure 2-151. Of these left-most five bits, bit positions 4 and 5 are set to "0" and "1", respectively. Bit 6 is obtained from bit 3 of the MLI contained in the F register. Bits 2 and 3 are used to specify the 4096-word storage unit in CS to which the FRJ jump is made. These bits are not changed from what is presently in the Sμ register. The FRJ address table is loaded during the initial CS load operation, immediately after CS is loaded.

An example of how the FRJ decode logic operates to generate jump addresses is shown in the block diagram of Figure 2-152. The FRJ decode logic consists of a translator, two address bit selectors, address table, output gating, and a parity check circuit. The MLI to be decoded is 2A 0·0, which means the MLI whose operation code is 2A, and whose bit positions 8 and 12 are both "0" (indicating that the MLI operands are to be obtained from or stored in a file register). The MLI is obtained from the D register via an FRJ μI and translated to generate two hexadecimal digits that point to the address table location containing the right-most nine bits of the FRJ branch address. For the 2A 0·0 MLI, the address table pointer is FB_{16} (1111 1011).

The contents of location FB are routed to the output gating logic in preparation for transmitting the final FRJ branch address to the set P logic. This logic also appends bits 2 through 6 of the right-most 9 bits as shown. Note that data fed to the output gating logic is in complement form, so that the NAND gates can invert it to true form. The inputs to the gates of bit positions 4 and 5 are tied to a logic high (+5 vdc) and a logic low (ground), respectively, so that their outputs will be "0" and "1", respectively. The inputs to the bit positions 2 and 3 gates are also tied high to generate outputs of "0" and "0". This causes the FRJ branch to be made within the same 4096-word CS storage unit. In this respect, the bits are said to be *unchanged* from what they were in the Sμ register before the jump address was produced. However, more storage units could be added at a later date (up to a maximum of 4). This might require these bits to be set to some value other than 00 if some FRJ routine should be located in another storage unit. It is for this reason that these bits are also generated in the FRJ decode logic besides bits 4 and 15, even though not actually necessary at present.

Since the address table is part of CS, it is alterable. The branch addresses are loaded in the table with the rest of CS as part of the Reset/Load sequence. This is accomplished by specifying each location in the address table by means of an address contained in bit positions 8 through 15 of Sμ. These address bits from Sμ are fed to the address bit selectors along with the outputs from the translator. During an initial CS load, however, the DR or Sμ enable will be such as to select the address bits from Sμ instead of from the translator. As each address table location is selected, the corresponding jump address to be stored is fed in on the N→CS bit 7 through 15 lines and the parity bit fed in on the N→CS bit 0 line.

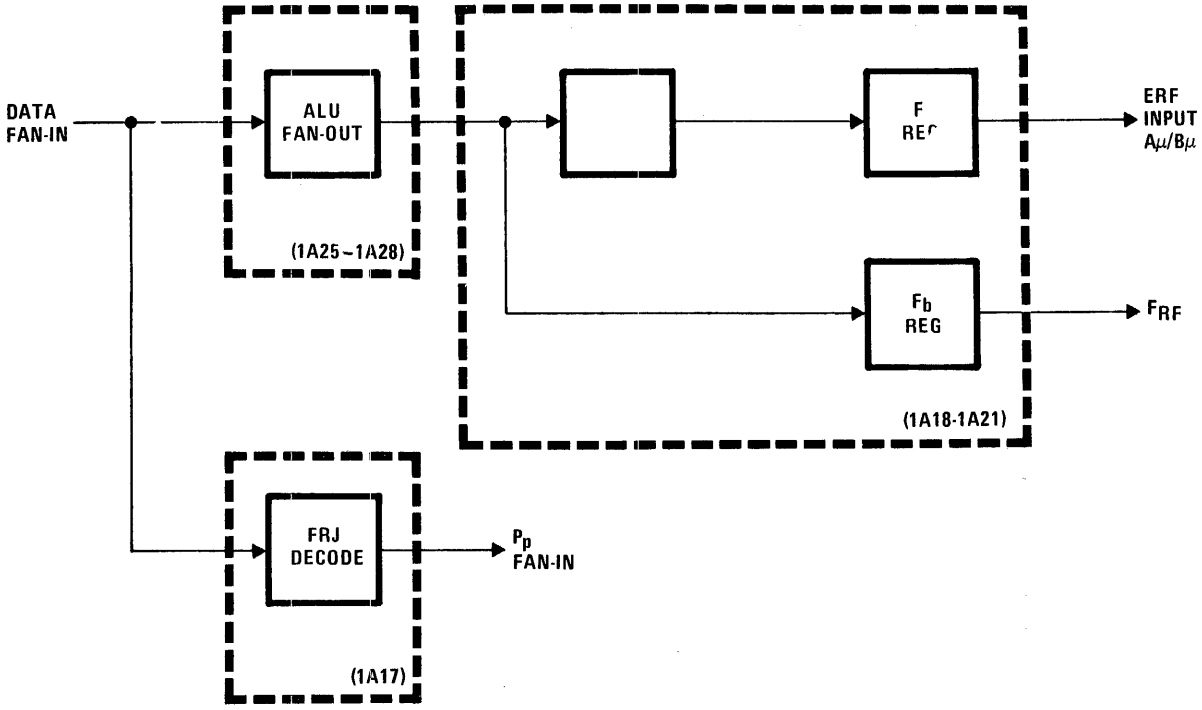


Figure 2-150. MLI Translation and Save/Store, Block Diagram

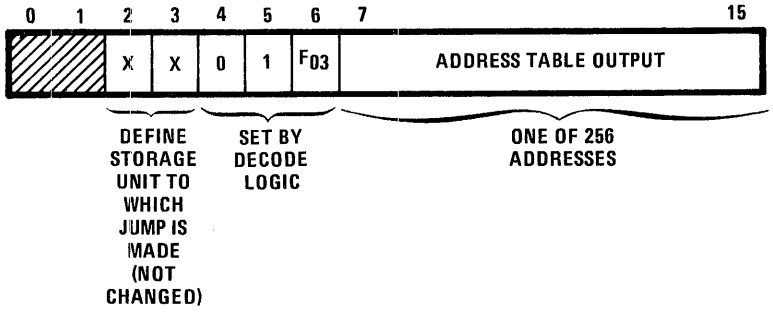


Figure 2-151. FRJ Branch Address

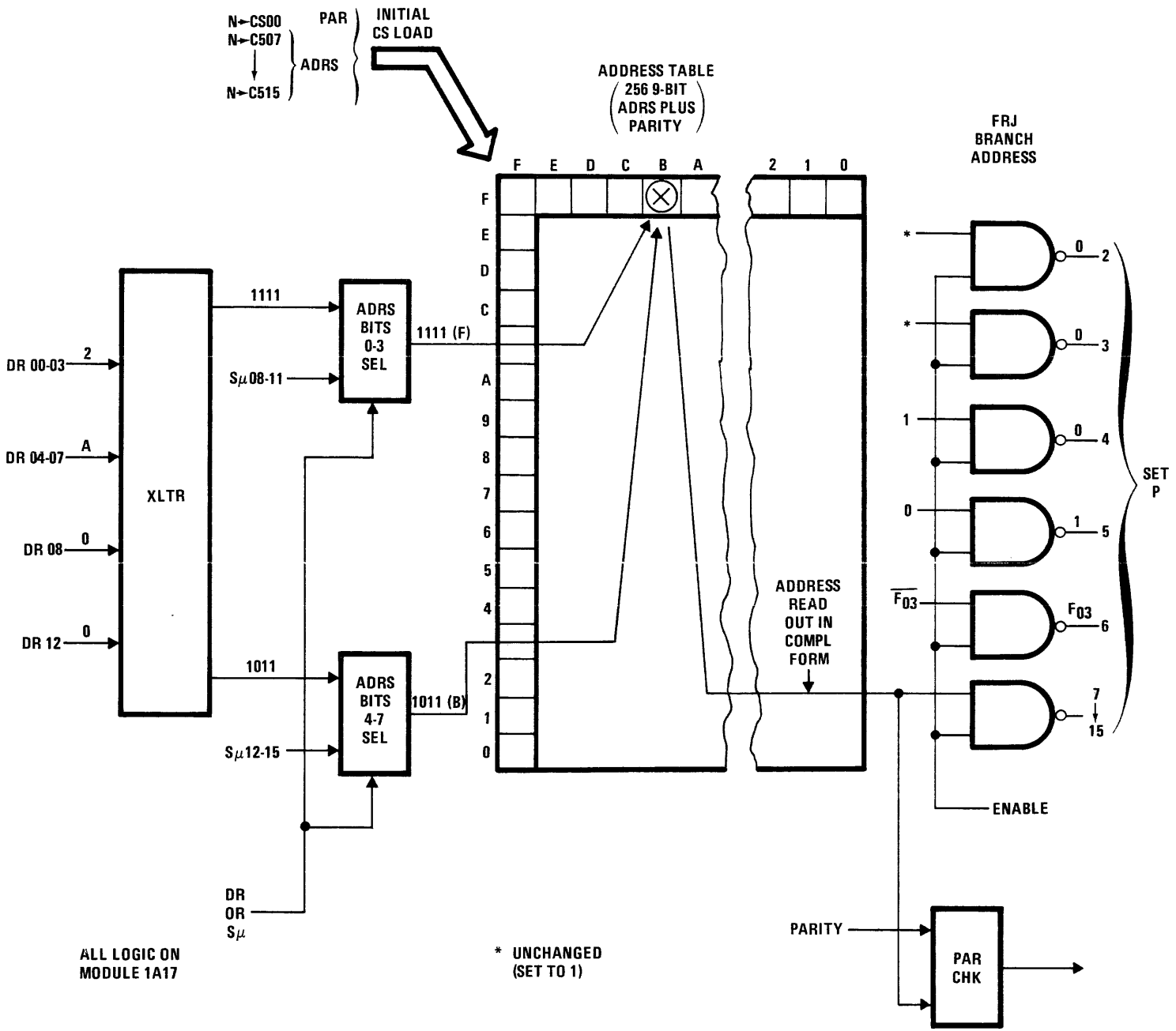


Figure 2-152. FRJ Decode of MLI 2A 0-0

A map of the address table, showing the location of the FRJ jump address for each type of MLI is shown in Table 2-17. The vertical boxhead contains the hexadecimal address generated by D register bits 0 through 3; the horizontal boxhead contains the hexadecimal address generated by bits 4 through 7.

F_b AND F REGISTERS

The F_b (F buffer) and F registers are used to hold the MLI being executed during the present time slice. Normally, the MLI is loaded in F from F_{RF} assigned to that processor at the beginning of the processor's present time slice (at E000 time). It stays in F where it is fanned out to various sections of control logic required to execute the MLI until the next E000 time. At this time, the MLI for the next processor is loaded into F. The above procedure is altered somewhat during an RNI sequence when a new MLI is read from MS for a processor. For this situation, the MLI is loaded into both F and F_b by a SDW μI at E5 time. Loading the new MLI into F provides the fan-out necessary to begin executing the MLI as before. The F_b register provides a place to hold the MLI until it can be stored in F_{RF}. It is necessary to provide this buffer register for this singular purpose because the MLI is not stored in F_{RF} until W1 time. However, at E0 time (one minor cycle time before W1), the MLI for the next processor is routed to F which displaces the MLI for the present processor. The F_b register, therefore, eliminates this overlap problem by providing a holding register for the MLI.

A simplified diagram of the F and F_b registers is shown in Figure 2-153. Both the F and F_b registers are fed with the MLI from the ALU fan-out logic via the ALU input. This MLI will be either a new MLI read from MS during an RNI sequence, or an existing MLI that has been modified during the course of its execution. (For example, double precision MLI's require $R_1 \pm 1$ and/or $R_2 \pm 1$ modifications to their R-fields.) At the beginning of the time slice, F is fed with the present MLI from F_{RF} during the R_1 cycle.

Selection of one of the two inputs to F is made via ENALU → FR through a two-input selector element. The two registers are clocked by a common signal but under different conditions. The F register is clocked unconditionally at E000 time for transferring the MLI presently being executed from F_{RF}. The F and F_b registers are both clocked during execution of a SDW μI for purposes of reading a new MLI from MS and transferring it to F and F_b as discussed previously. Clocking F with the contents of F_{RF} is inhibited at E000 if the present processor is operating in the consecutive-cycle (CC) mode. Under this condition, there is no need to clock the next processor's MLI into F since the present processor will keep executing. Indeed, it may happen that the ALU might have modified the present MLI at E8; consequently, it must be routed back to both F_b and F at E0 of the next time slice.

ARITHMETIC-LOGIC UNIT

The arithmetic-logic unit (ALU) performs all arithmetic and logical operations required by the μI's. These operations include the following: addition, comparison, shifts, logical sum and product, bit sense, and sense/toggle. A block diagram of the ALU is shown in Figure 2-154.

The ALU adder performs both addition and subtraction by an additive process. Addition is performed by adding both numbers in true form; subtraction is accomplished by adding the minuend in true form and the subtrahend in two's complement form. Two's complementing is performed by the LAW– and LBW– μI's in conjunction with the Forced Carry Register (FCR).

Data to be operated on by the ALU is fed to the Aμ and Bμ registers through corresponding fan-in logic. Generally, this data will be from either a file register or from main storage via the data fan-in path. Data in these registers is unconditionally added and compared by the adder and compare networks. The results, however, are used only if required by the μI being executed. Data to be manipulated by any of the other operations is done so only if the μI so specifies.

Shift, bit sense, and sense/toggle operations are implemented by corresponding logic. Both these operations feed the shifted or sense/toggle data back to the Aμ and Bμ registers to complete the operation. In the case of a shift, the (up to) 32-bit result is held in Aμ and Bμ after being shifted for use by another μI. A bit sense or sense/toggle operation required sending the location to the sensed or toggled bit in Aμ back to Bμ for addition to the contents of Bμ. Because these two operations require a longer than normal propogation path through the ALU, they may delay execution of the next μI as explained in greater detail in the paragraphs that discuss these operations.

Logical sum and product operations on the contents of Aμ and Bμ are performed by means of the ALU fan-out, wherein the logical operations is effected by certain combinations of enables according to the μI being executed.

Some μI's require certain constants to be generated as part of their execution. These constants enter the Bμ register as 16-bit words, 8-bit bytes, or 4-bit nybls, depending on the μI. They are generated by the constant generator which feeds the Bμ fan-in.

Table 2-18. FRJ Address Decode Matrix

4-7 / 0-3	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
F	20-29 $R_1 R_2$	20-29 $R_1 (R_2)$	20-29 $(R_1) R_2$	20-29 $(R_1)(R_2)$	2A $R_1 R_2$	2A $R_1 (R_2)$	2A $(R_1) R_2$	2A $(R_1)(R_2)$	2B $R_1 R_2$	2B $R_1 (R_2)$	2B $(R_1) R_2$	2B $(R_1)(R_2)$	2C-2F $R_1 R_2$	2C-2F $R_1 (R_2)$	2C-2F $(R_1) R_2$	2C-2F $(R_1)(R_2)$
E	30-39	30-39	30-39	30-39	3A	3A	3A	3A	3B	3B	3B	3B	3C-3F	3C-3F	3C-3F	3C-3F
D	A0-A9	A0-A9	A0-A9	A0-A9	AA	AA	AA	AA	AB	AB	AB	AB	AC-AF	AC-AF	AC-AF	AC-AF
C	B0-B9	B0-B9	B0-B9	B0-B9	BA	BA	BA	BA	BB	BB	BB	BB	BC-BF	BC-BF	BC-BF	BC-BF
B	60-63	60-63	60-63	60-63	64-67	64-67	64-67	64-67	68-6B	68-6B	68-6B	68-6B	6C-6F	6C-6F	6C-6F	6C-6F
A	70-73	70-73	70-73	70-73	74-77	74-77	74-77	74-77	78-7B	78-7B	78-7B	78-7B	7C-7F	7C-7F	7C-7F	7C-7F
9	00-03	04-07	08-0B	0C-0F	10-13	14-17	18-1B	1C-1F	40-43	44-47	48-4B	4C-4F	50-53	54-57	58-5B	5C-5F
8	80-83	84-87	88-8B	8C-8F	90-93	94-97	98-9B	9C-9F	C0-C3	C4-C7	C8-CB	CC-CF	D0-D3	D4-D7	D8-DB	DC-DF
7	E0	E0	E0	E0	E1	E1	E1	E1	E2	E2	E2	E2	E3	E3	E3	E3
6	E4	E4	E4	E4	E5	E5	E5	E5	E6	E6	E6	E6	E7	E7	E7	E7
5	E8	E8	E8	E8	E9	E9	E9	E9	EA	EA	EA	EA	EB	EB	EB	EB
4	EC	EC	EC	EC	ED	ED	ED	ED	EE	EE	EE	EE	EF	EF	EF	EF
3	F0	F0	F0	F0	F1	F1	F1	F1	F2	F2	F2	F2	F3	F3	F3	F3
2	F4	F4	F4	F4	F5	F5	F5	F5	F6	F6	F6	F6	F7	F7	F7	F7
1	F8	F8	F8	F8	F9	F9	F9	F9	FA	FA	FA	FA	FB	FB	FB	FB
0	FC	FC	FC	FC	FD	FD	FD	FD	FE	FE	FE	FE	FF	FF	FF	FF

*Value of A and B same as Row F
**Values of A and B do not apply

NOTE: (Boxheads represent hexadecimal address in a 256-word address table that points to the starting address, as modified by bits 2-6 of $S\mu$, for implementing the machine-language instruction indicated in the matrix.) Actual starting addresses in CS are listed in the CS printout.

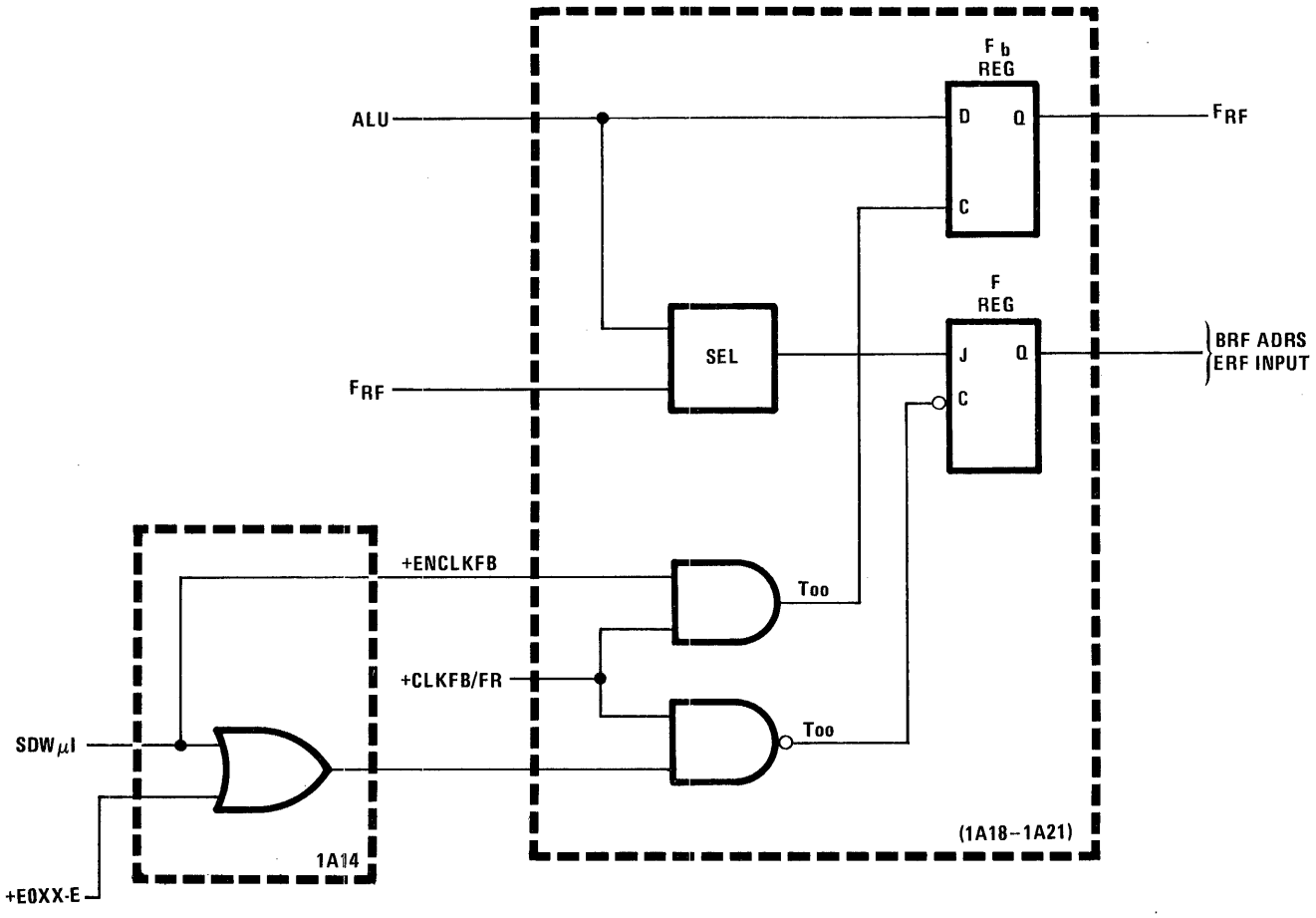


Figure 2-153. F and Fb Registers

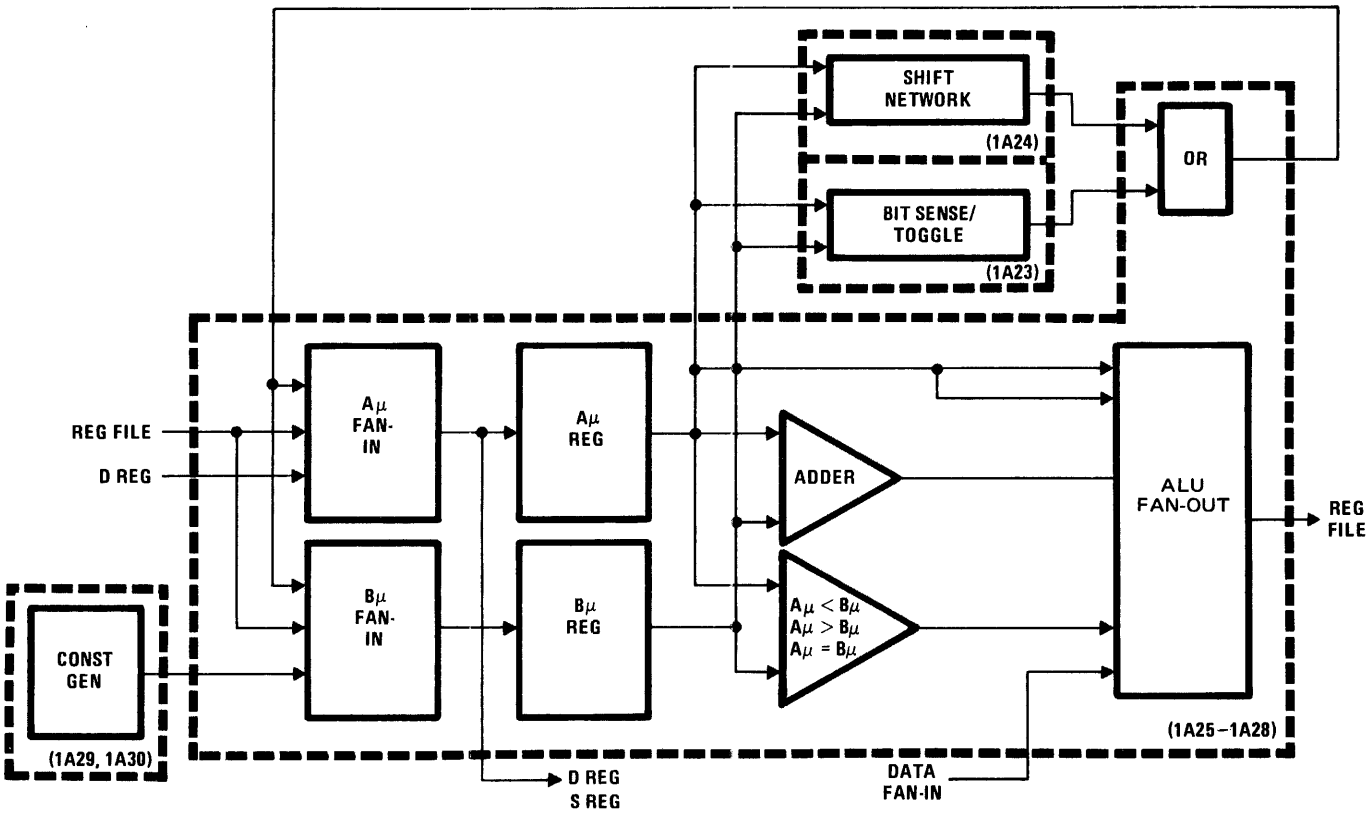


Figure 2-154. ALU Block Diagram

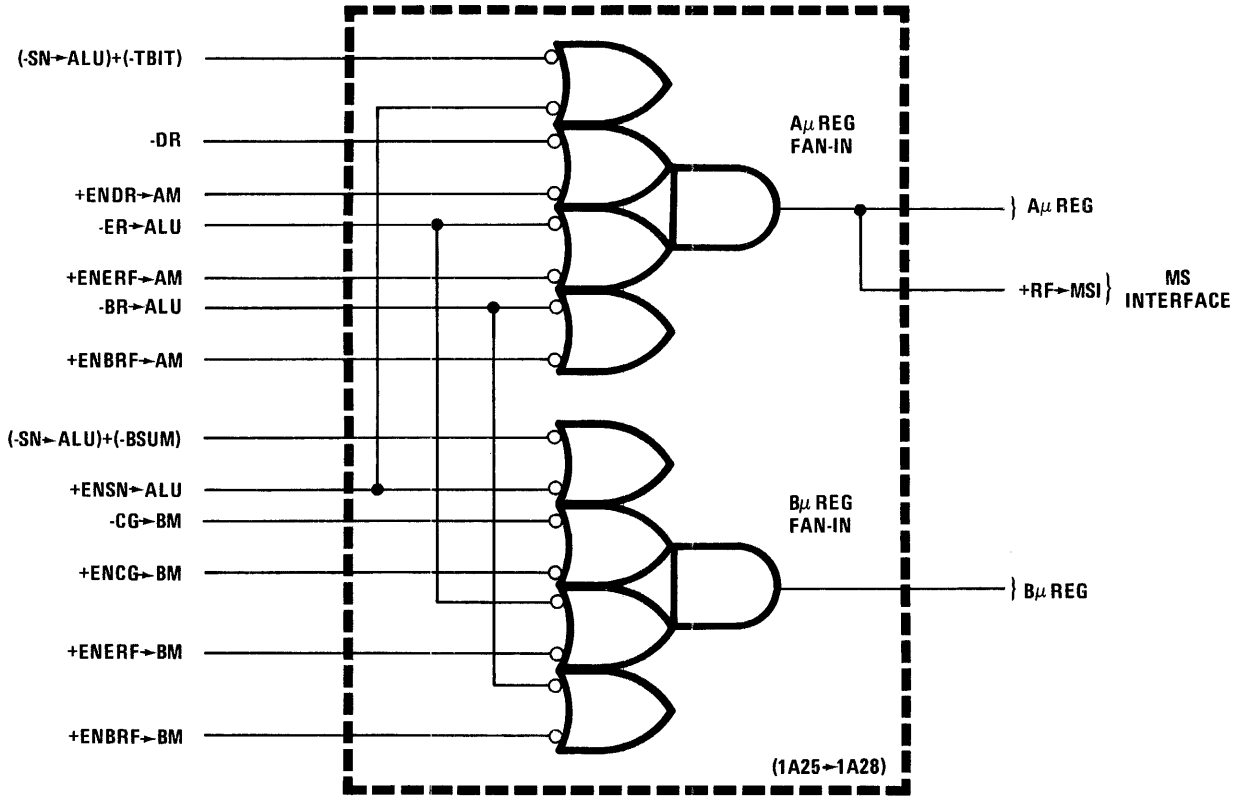


Figure 2-155. Aμ and Bμ Register Fan-In

Aμ AND Bμ REGISTER FAN-IN

The Aμ and Bμ register fan-in logic provides inputs to the Aμ and Bμ registers from a variety of sources, as shown in Figure 2-155. As shown, the logic consists of two gating networks for each bit, one for the Aμ register and one for the Bμ register. Data comes from the sources listed below:

1. Shift network (SN bits) — to both Aμ and Bμ registers.
2. Bit toggle generator (TBIT bits) — to Aμ register only.
3. Main Storage, Register Option, or D register (DR bits) — to Aμ register only.
4. Extended register files (ER bits) — to both Aμ and Bμ registers.
5. Basic register files (BR bits) — to both Aμ and Bμ registers.
6. Bμ register adder (BMSUM bits) — to Bμ register only.
7. Constant generator (CG bits) — to Bμ register only.

All data is routed to the fan-in logic in complement form and gated through the logic by an enable in true form. The result is an output signal in true form that is routed to the clocked set and clear inputs of the Aμ and Bμ register flip-flops. A second output from the Aμ register fan-in logic, labeled RF-MSI, routes data from a selected register of the BRF to the S and D registers in the control storage interface logic during execution of load S and load Dμl's.

The SN bits are enabled by ENSN-ALU during execution of a shift μl (SHF, SHR, DLS, or DRS). Data from the D register is gated by ENDR → AM during execution of a D → A μl (DTA, DTA/, IDX, or DFA). The ER bits from a selected extended register are gated to the Aμ and Bμ registers by ENERF-AM and ENERF-BM, respectively. These two enables are generated for different μl's when an overall *permit* condition, AANDB + INVERF, is present. Signal AANDB indicates that the X-field and the μl is addressing a register in the ERF (*a* and *b* designators are both 1). Signal INVERF is generated when executing an IVK μl, wherein the processor and ERF register numbers are obtained from the Boundary Crossing register. Enable ENERF-AM is generated when AANDB + INVERF is present, and executing any μl except a Load D μl or a Shift μl. These two classes of μl's specifically inhibit ENERF-AM since they require their own enables to transfer data to the Aμ register, as described above. Enable ENERF-BM is generated when RANDM + INVRFE is present, and executing an LBW, LBW/, LAB, CLA, or LBL μl; or a Load D μl. The Load D μl's require, in addition to loading the D register, that the contents of an ERF be transferred to the Bμ register, in either true or complement form.

The BRF bits from a register of the BRF are enabled by ENBRF →AM and ENBRF-BM. These two enables are generated in a manner similar to those for extended register data: the presence of an overall permit condition ANDed with specific μl function codes. The permit condition for enabling BRF data is $\overline{\text{AANDB}} \cdot \overline{\text{INVERF}}$. The specific μl's for generating each BRF enable are identical to those for generating the ERF enables, as summarized below:

$$\text{ENBRF-AM} = (\overline{\text{AANDB}} \cdot \overline{\text{INVERF}}) \cdot (\overline{\text{LOAD D}} + \overline{\text{SHIFT}})$$

$$\text{ENBRF-BM} = (\text{AANDB} \cdot \text{INVERF}) \cdot (\text{LBW} + \text{LBW/} + \text{LAB} + \text{CLA} + \text{LBL} + \text{LOAD D})$$

Data from the constant generator is gated through the fan-in logic to the Bμ register via ENCG-BM. This enable is generated for any μl except for the above μl's referenced, which require their own enables for the peculiarities of the particular μl.

Aμ AND Bμ REGISTERS

The Aμ and Bμ registers receive data from the Aμ and Bμ fan-out logic that is to be processed by the adder. These two registers can be considered as the *addend* and *augend* registers since they hold these two quantities during add operations. Each register consists of 16 J-K flip-flops with data, clock, preset, and preclear inputs. Since some μl's require that data to be processed by the adder be in complement form, the Aμ and Bμ registers provide the capability for one's complementing data if the μl so requires (all data routed to the Aμ and Bμ registers is in true form; therefore, complementing must be done in the registers themselves). This complementation is provided by using the ability of a J-K flip-flop output to toggle its output state when both inputs are "T", and by conditioning the register prior to storing data via the preset and preclear inputs. (Two's complementing of the data, required by the SUM and DSUM μl's for performing subtraction, is effected by adding +1 from the FCR to the one's complement data in Aμ and Bμ.)

An example of how data can be stored in one stage of the Aμ register in either true or complement store is shown in Figure 2-156. Part *a* shows how data is stored in true form. Prior to reception of data on the input line connected to both the J and K inputs, the flip-flop is pre-

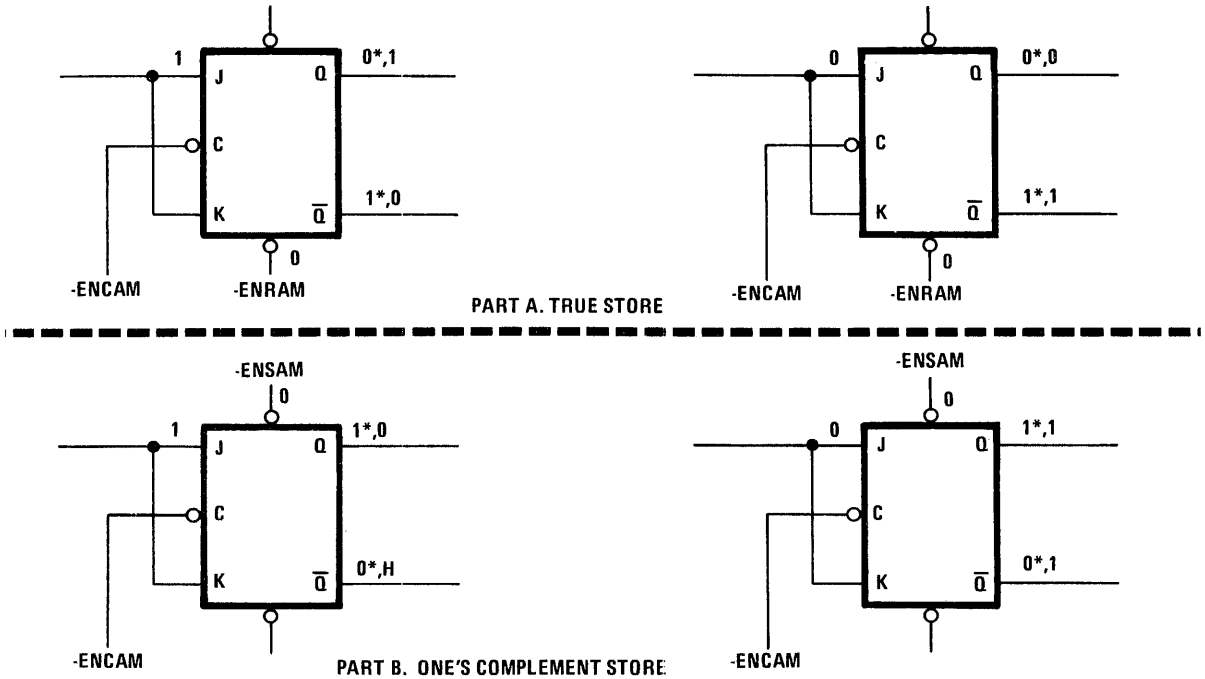


Figure 2-156. Aμ and Bμ Register Data Store

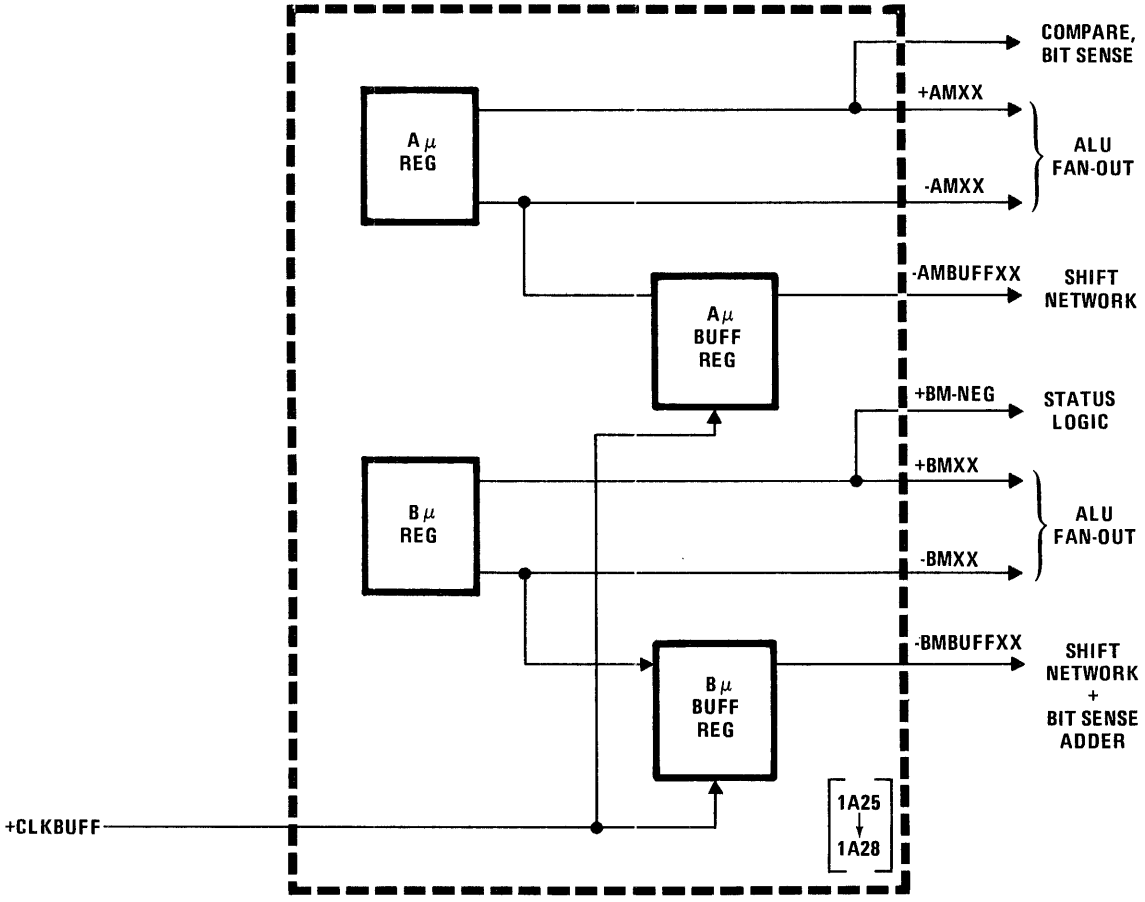


Figure 2-157. Aμ and Bμ Register Destinations

cleared by ENRAM to generate "1"* on the Q output line and "0"* on the Q output line (the * indicates the state of the output lines due to either the preset or preclear conditioning). Assuming the input data is a "1", the Q and Q outputs are toggled, when clock pulse ENCAM goes low, so that now the Q output is a "0" and the Q output is a "1". If the input data is a "0", no toggling takes place and the Q and Q output states remain unchanged. Since data is stored in the flip-flop stage in the same form as that on the input lines, the stage is said to store data in true form. Part *b* of Figure 2-156 shows storing data in complement (one's complement) form. The flip-flop is conditioned by presetting it via the ENSAM signal. The result is to change the Q output to a "1"* and the Q output to a "0"*. Assuming again that the input data is a "1", the state of the output lines will toggle so that the Q output goes to "0" and the Q output goes to "1". The resultant output then becomes the complement of the input. If the input data is a "0", no toggling takes place and the Q output remains at "1" and the Q output remains at "0". The Bμ register stores data in either true form or complement in exactly the same way, conditioned by means of enables ENRBM and ENSBM. A list of μI's involving transfer of data between the Aμ and Bμ registers and the adder, and their corresponding register enables is shown in Table 2-19.

Outputs from the Aμ and Bμ registers are routed to several destinations, as shown in Figure 2-157. Both set and clear sides of both registers are fed to the ALU fan-out logic for distribution to other sections of the shared resources. In addition, the set side of the Aμ register is routed to the compare and bit sense logic for evaluation during execution of compare and bit sense μI's. Bit 00 from the set side of the Bμ register is routed to the status logic as BM-NEG for status bit compare operations. Also, the clear side of the Bμ register is fed to a *buffer* register. Outputs from this *buffer* register are routed to the bit sense adder, during execution of bit sense (E,X,1) μI's; and to the shift network, during execution of shift (E,X,0) μI's.

ADDITION

Adder Element Operation

The adder consists of four MSI adder elements and a full-carry look-ahead circuit, as shown in Figure 2-158. The adder performs addition of two 16-bit operands from the Aμ and Bμ registers. This addition is performed unconditionally whenever operands are loaded in the Aμ and Bμ registers. The result is used, however, only when so directed by a SUM or DSUM μI. Each ALU element performs upon four bits of the operand and is designated by one of four group numbers: 0 through 3. Each adder element generates four SUM bits, a group carry generate bit ($\overline{\text{GCGEN}}$) and a group carry propagate bit ($\overline{\text{GCPROP}}$). The $\overline{\text{SUM}}$ bits are fed to the ALU fan-out logic; the $\overline{\text{GCGEN}}$ and $\overline{\text{GCPROP}}$ bits are fed to the look-ahead carry generator logic. This logic provides simultaneous carries for each ALU element by combining the $\overline{\text{GCGEN}}$ and $\overline{\text{GCPROP}}$ bits from each element to generate a corresponding group carry input signal (GPCRIN). Each carry input is fed to the next higher group ALU element in an attempt to satisfy the lower-order carry. The carry generate logic for the highest-order group (group 0) generates a sum word carry, $\overline{\text{ADDERGEN}}$, in place of a GCRIN signal. This sum word carry is fed to the Forced Carry register during multiple-precision operations to generate a lowest-order carry (GPCRIN-3) to be satisfied during addition of subsequent words during the multiple-precision operation.

Group Carry Generate

The $\overline{\text{GCGEN}}$ signal from an adder element is the Group Carry Generate signal, indicating that the inputs to the four stages have generated a sum that is in excess of what can be respresented by the $\overline{\text{SUM}}$ bits of that element. A group carry generate will be developed whenever any of the input conditions to an element listed in Table 2-20 is present.

Group Carry Propagate

The $\overline{\text{GCPROP}}$ signal from an ALU element is the Group Carry Propagate signal, indicating that the particular element cannot absorb a group carry generate from a lower-order element. Therefore, if a carry generate from a lower-order element occurs it must be propagated to a higher-order element. A group carry propagate is generated whenever the sum of a stage is either "1" (addend and augend bits are "0" and "1" or "1" and "0"), or a "0" with a carry of "1" (both addend and augend bits are "1"). Table 2-21 shows input states of the $\overline{\text{AM}}$ and $\overline{\text{BM}}$ bits which will generate a group carry propagate.

Look-Ahead Carry Generator

The look-ahead carry generator evaluates the group carry generate and group carry propagate bits from each adder element to develop group carry input bits for each higher-order adder element. Since each adder element generates carry generate and carry propagate bits

Table 2-19. Aμ and Bμ Register Enables

μI		Enables			
Op Code	NMEM0	ENSAM (COMP→Aμ)	ENRAM (TRUE→Aμ)	ENSBM (COMP→Bμ)	ENRBM (TRUE→Bμ)
3, 0	LS1		X		X
3, 1	LSF		X	X	
3, 2	LS2		X		X
3, 3	LSE		X	X	X
6, 0	LBW				X
6, 1	LBW–			X	
6, 2	LBB				X
6, 3	LBB–			X	
7, 3	LBL				X
A	EBU				X
B	EBL				X
C, 0	DTA		X		X
C, 1	DTA–		X		
C, 2	IDX		X		X
C, 3	DFA	X			X
D, 0	LAW		X		
D, 1	LAW–	X			
D, 2	LAB		X		X
D, 3	CLA		X		X
E, 0, 0	SHF		X		X
E, 1, 0	SHR		X		X
E, 2, 0	DLS		X		X
E, 3, 0	DRS		X		X
F, 2	DIG				X
F, 3	CORC				X

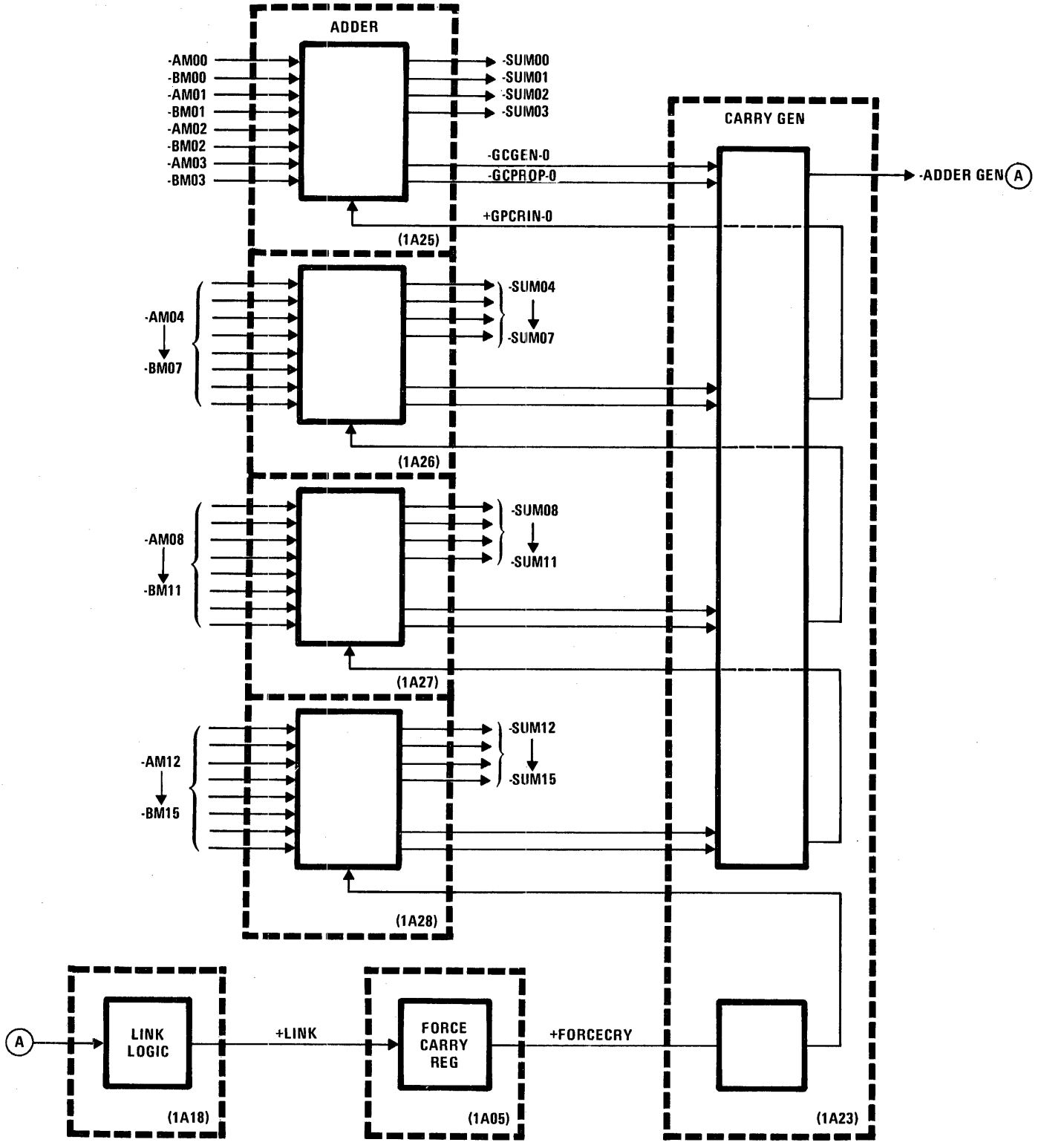


Figure 2-158. Adder

Table 2-20. Group Carry Generate Truth Table

$\overline{AM00}$	$\overline{BM00}$	$\overline{AM01}$	$\overline{BM01}$	$\overline{AM02}$	$\overline{BM02}$	$\overline{AM03}$	$\overline{BM03}$
0	0	X	X	X	X	X	X
0 1	1 0	0	0	X	X	X	X
0 1	1 0	0 1	1 0	0	0	X	X
0 1	1 0	0 1	1 0	0 1	1 0	0	0

Table 2-21. Group Carry Propagate Truth Table

$\overline{AM00}$	$\overline{BM00}$	$\overline{AM01}$	$\overline{BM01}$	$\overline{AM02}$	$\overline{BM02}$	$\overline{AM03}$	$\overline{BM03}$
0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0
0	0	0	0	0	0	0	0

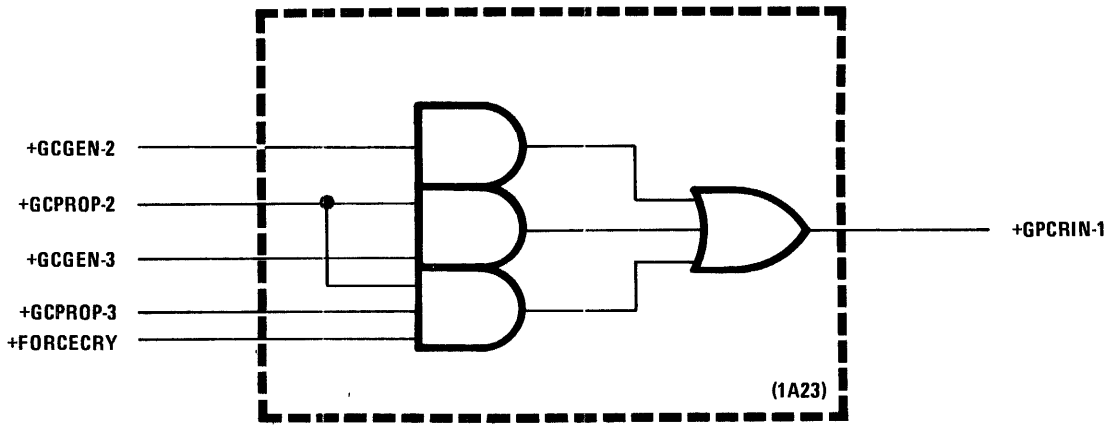


Figure 2-159. Generation of GPCRIN-1

simultaneously, the carry generator produces the required carry input bits to each adder element simultaneously.* It is this *look-ahead* capability of the carry generator in generating carry input bits simultaneously that speed up the addition operation as compared with an add performed using serial carry inputs. A portion of the carry generator, that used for generating GPCRIN-1 (the group carry input bit to adder element group 1), is shown in Figure 2-159. The GPCRIN-1 signal is generated if any of the following conditions occurred:

1. group 2 produced a carry out (GCGEN-2), or
2. group 2 produced a carry propagate (GCPROP-2) and group 3 produced a carry out (GCGEN-3), or
3. both groups 2 and 3 produced a carry propagate, and a carry was forced from the FCR (FORCECRY).

As mentioned previously, ADDERGEN is produced as a sum-word carry during multiple precision operations. Normally this sum-word carry will be generated when a carry out results from bit 15 of the sum. During execution of a CMPK, ADDK, SUBK, or ZADK MLI (F codes of 50 through 53), however, the sum-word carry is generated when bit 7 of the sum generates a carry out. This is because these four MLI's execute decimal numbers in byte form (two 4-bit hexadecimal digits) instead of in whole-word form (four 4-bit hexadecimal digits). For these four MLI's, ADDERGEN is generated by MLI-5053. Signal ADDERGEN is fed to the *link* logic to generate a link status bit in the Pμ register as shown in the adder block diagram of Figure 2-158. From the link logic, it is fed to the Forced Carry register which generates FORCECRY. This signal then is fed back to the carry generator to generate GPCRIN-3.

Forced Carry Register

The Forced Carry register (FCR) is a single flip-flop used to store either a "0" or "1" for use as a constant during the following three types of operations:

1. MLI address updates,
2. two's complement arithmetic operation, and
3. storing a sum-word carry generated for subsequent additions during multiple precision arithmetic adds.

The type-D edge-triggered flip-flop can be set or cleared in three different ways, as shown in Figure 2-160: +1 → FCR, 0 → FCR, and LINK → FCR.

The +1 → FCR operation is effected by a low into the forced set input which causes the Q output to go low. For LS1 and LS2 μI's, the operation is performed for purposes of forming the address of the next MLI. These two μI's load register S (the MLI address register) with the contents of a register designated by the μI X-field. The μI's are programmed as part of the MLI RNI sequence to form the address of the MLI. The +1 → FCR operation is then used by the SUM μI in the RNI sequence to add either 1 (LS1 μI) or 2 (LS2 μI) to the contents of the S register to form the address of the next MLI to be executed. For LBW-, LBB-, DTA, DFA, LAW-, and CLA μI's, the +1 → FCR operation is used to add one to the one's complement of the word loaded by these μI's to express them in two's complement form.

The 0 → FCR operation is produced by a low into the forced clear input which causes the Q output to go high. For LSF and LSE μI's, the operation is performed for purposes of forming the address of the next MLI by subtracting either 1 (LSF μI) or 2 (LSE μI) from the contents of the S register. (The constant 1 or 2 to be subtracted comes from the Bμ register; therefore, the FCR must be loaded with a 0.) For the LBW, LBB, DTA, IDX, and LAB μI's, the 0 → FCR operation is performed to inhibit adding 1 to the word loaded by these μI's. These μI's are the *true form* equivalent of the one's complement load μI's discussed above. Since addition of the FCR contents to the word loaded by these μI's occurs unconditionally as part of the μI, the FCR must be loaded with a 0 to avoid correcting the word loaded in true form. The DIG and CORC μI's provide for encoding and post-addition correcting of decimal numbers expressed in excess-3 form. Both these μI's involve adding (or subtracting) 3 from the decimal number represented in hexadecimal form. However, the carries (if produced) by these operations must be inhibited. Since the FCR would ordinarily furnish this carry, due to its function as a sum-word carry generator, the register must be set to 0 to inhibit the carry.

During execution of an LBL μI, the Link bit from the Pμ register is sent to the FCR. This operation is effected by storing LINK in the flip-flop when clocked by LBL. This causes the Q output to go low to generate the forced carry. The flip-flop output is sent in complement form through a selector/inverter to generate FORCECRY in true form. This selector/inverter also provides for generating FORCECRY upon simultaneous depression of the

*Actually, the carry input to group 0 is generated two gate delay times later than the group 3 carry input and those to groups 1 and 2 are generated one gate delay time later. These delays, however, are negligible compared to gate delays incurred in the adder elements.

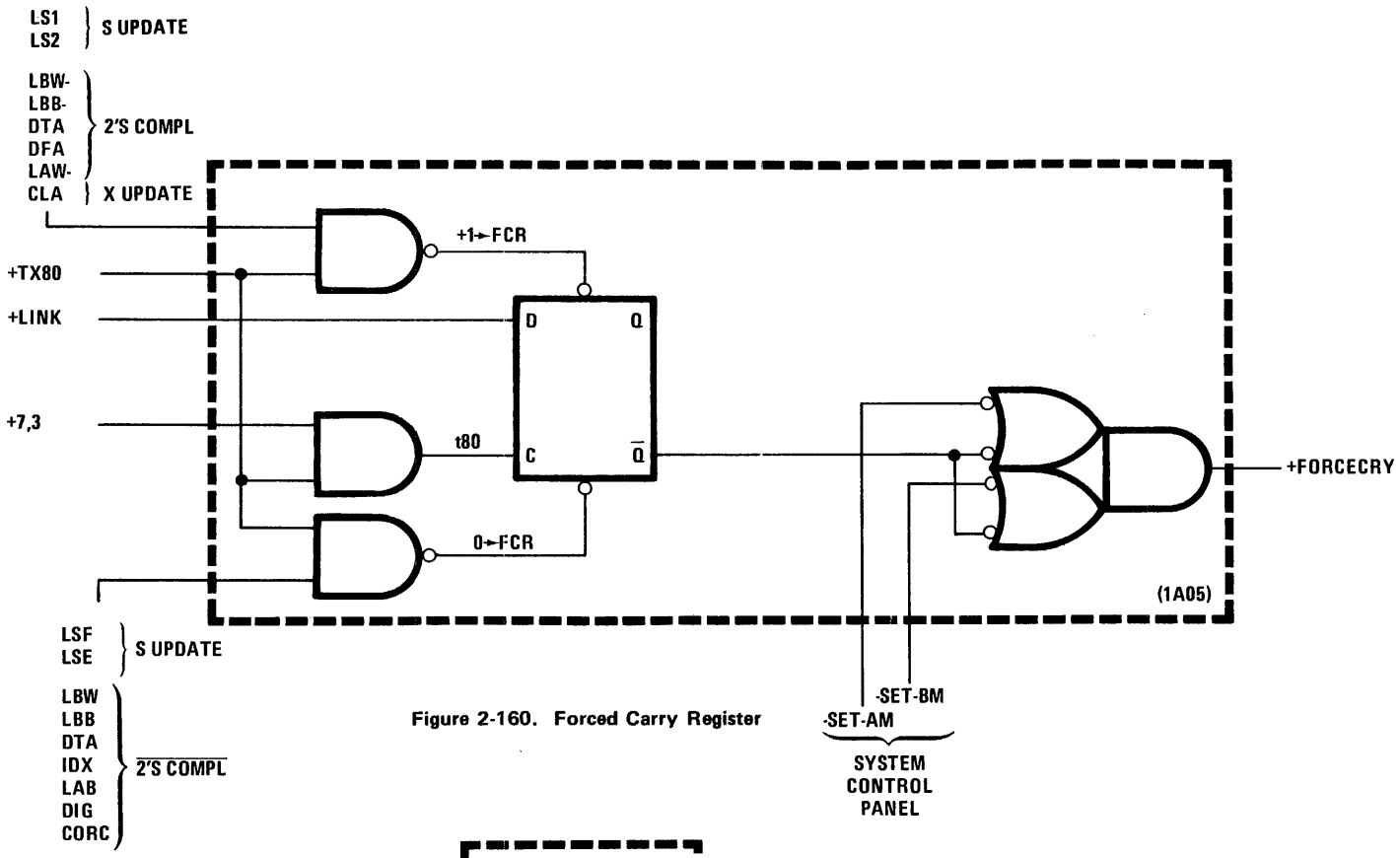


Figure 2-160. Forced Carry Register

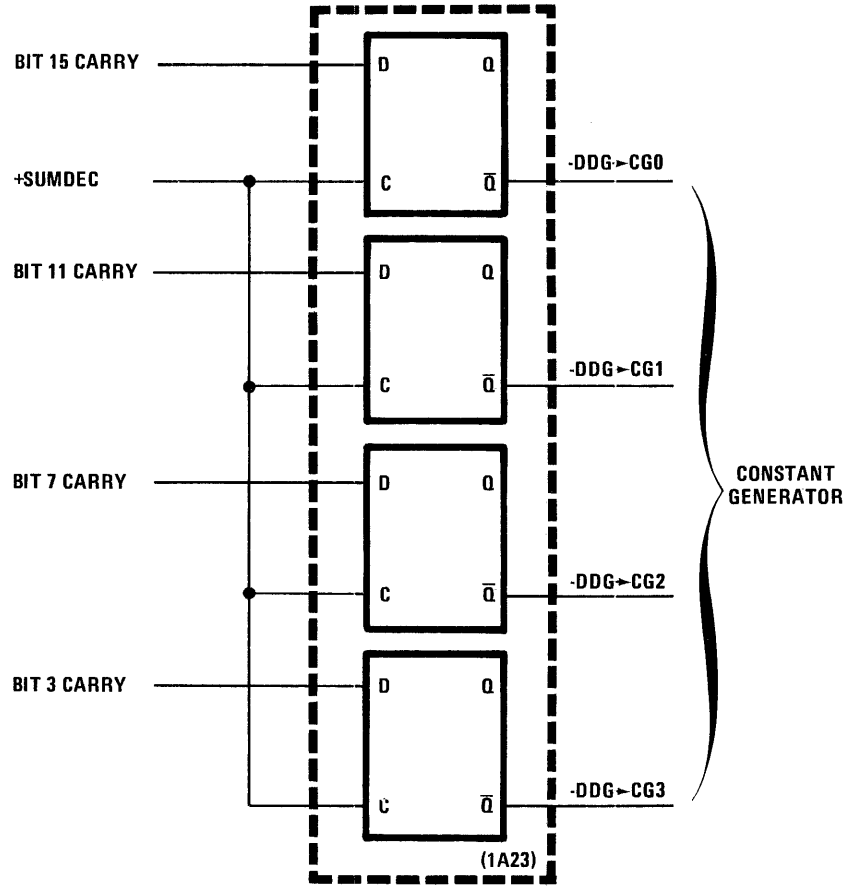


Figure 2-161. Inner Carry Register

SET A and SET B pushbuttons on the System Control Panel. Simultaneously pressing these pushbuttons enables a carry to be forced into the ALU adder during Panel operations.

Inner Carry Register

The Inner Carry register (ICR) evaluates the carry outputs from the look-ahead carry generator during execution of decimal sum operations via the DSUM μI. The outputs from the ICR determine whether a +3 or a -3 is generated by the constant generator to correct each 4-bit hexadecimal group of the decimal sum. The register consists of four type-D flip-flops, as shown in Figure 2-161.

Each flip-flop is fed with the carry output from a 4-bit group corresponding to the hexadecimal equivalent of each digit of the decimal sum. The carries are clocked into the flip-flops when SUMDEC is generated via translation of the DSUM μI. The $\overline{\text{DDG-CG}}$ outputs from the ICR are then routed to the constant generator for generation of either +3 or -3 for decimal sum correction.

COMPARE

Compare operations performed by the ALU consist of making five types of comparisons between operands in the Aμ and Bμ register: Aμ < Bμ, Aμ > Bμ, Aμ = Bμ, A$\mu \neq$ Bμ, and Aμ = 0. Both algebraic (sign and magnitude) and logical (magnitude only) compares are made, in an unconditional manner whenever operands are loaded into the Aμ and Bμ registers. Their results, however, are used only when so directed by aμ I or other command enable. The results are used for the following three purposes:

1. storing compare status information in the Condition register during execution of a compare (CMP, CMU) μI,

2. evaluating conditions under which the present processor is turned off and the next processor in the queue is granted priority, during execution of a CIO1 or CIO2μ I, and

3. evaluating skip conditions during a skip μI.

Aμ < Bμ and Aμ > Bμ Compares

The Aμ < Bμ and Aμ > Bμ compares are made as shown in Figure 2-162. The 16 outputs from both the Aμ and Bμ registers are routed to four LSI compare elements. Each element makes an Aμ < Bμ and Aμ > Bμ comparison of four pairs of bits. The two outputs from each element, AMGTBM (Aμ greater than Bμ) and AMLTBM (Aμ less than Bμ), are routed to final combinational logic which combines all four AMGTBM outputs and all four AMLTBM outputs with AMEQBM (Aμ = Bμ) signals to generate final AMLTEQBM and AMGTEQBM signals, respectively. Consider first the logic that generates AMLTEQBM. The signal is generated by a NANDing operation which combines each AMGTBM signal of a particular 4-bit group with the AMEQBM signal of the next higher-order bit group.

Essentially, this logic generates AMLTEQBM if no group of Aμ bits is greater than a corresponding group of Bμ bits. The reason for considering the AMEQBM signal from a previous group is to account for all possible combinations of operand magnitudes in both Aμ and Bμ registers. The example shown in Figure 2-163 illustrates how the AMLTEQBM logic operates. Each digit of the decimal number contained in the Aμ (1399_{10}) and Bμ (1400_{10}) registers is represented by four bits of one group. Below each pair of digits of a group is shown the corresponding compare evaluation. Note that each evaluation contains one signal that is low when the compare evaluation is satisfied for that group. This is necessary to ensure that at least one input to each AND gate of the AMLTEQBM logic is low to generate a high AMLTEQBM signal. The AMGTEQBM logic works in basically the same way, except that the output signal is generated if no group of Bμ bits is greater than or equal to a corresponding group of Aμ bits.

Aμ = Bμ and A$\mu \neq$ Bμ Compares

The Aμ = Bμ and A$\mu \neq$ Bμ compares are made as shown in Figure 2-164. The comparison of Aμ = Bμ is made again in groups of four bits by pairing the true outputs of the Aμ register with the complement outputs of the Bμ register, and vice versa.

The result of such pairing yields a high output for each comparison, as shown in the example at the top of Figure 2-164. The output from each group comparison is fed to an AND gate to generate AMEQBM, and through an inverter to generate $\overline{\text{AMEQBM}}$.

Aμ = 0 Compare

The Aμ = 0 compare is made as shown in Figure 2-165. The 16 complement outputs from the Aμ register are fed to a NAND gate, which generates a low output when all inputs are high (all 16 register flip-flops contain "0"s). The low output is designated AMEQZR.

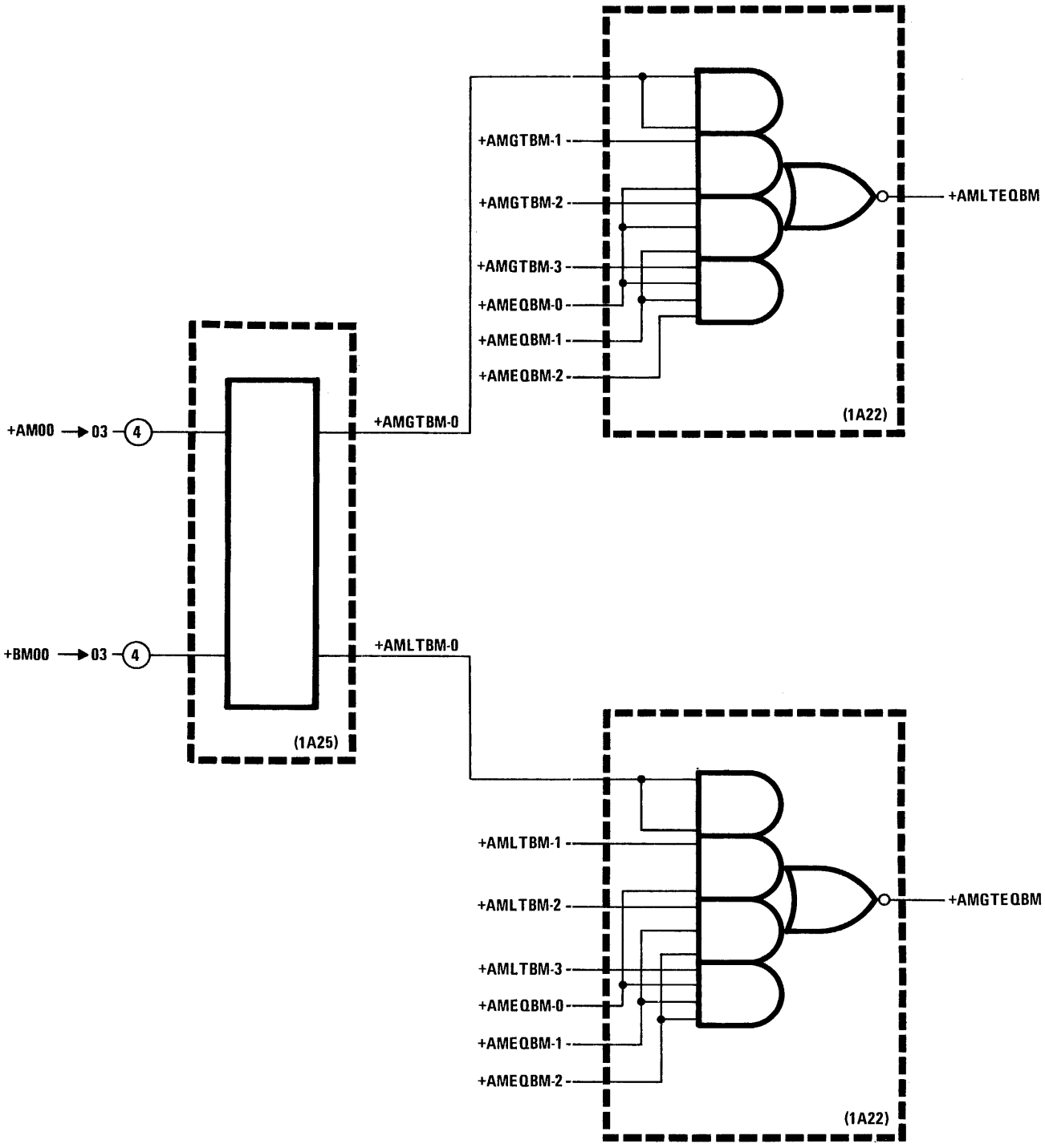


Figure 2-162. Aμ < Bμ and Aμ > Bμ Compare Logic

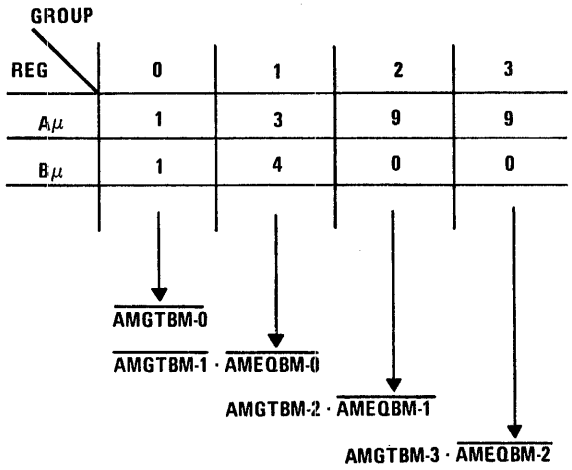
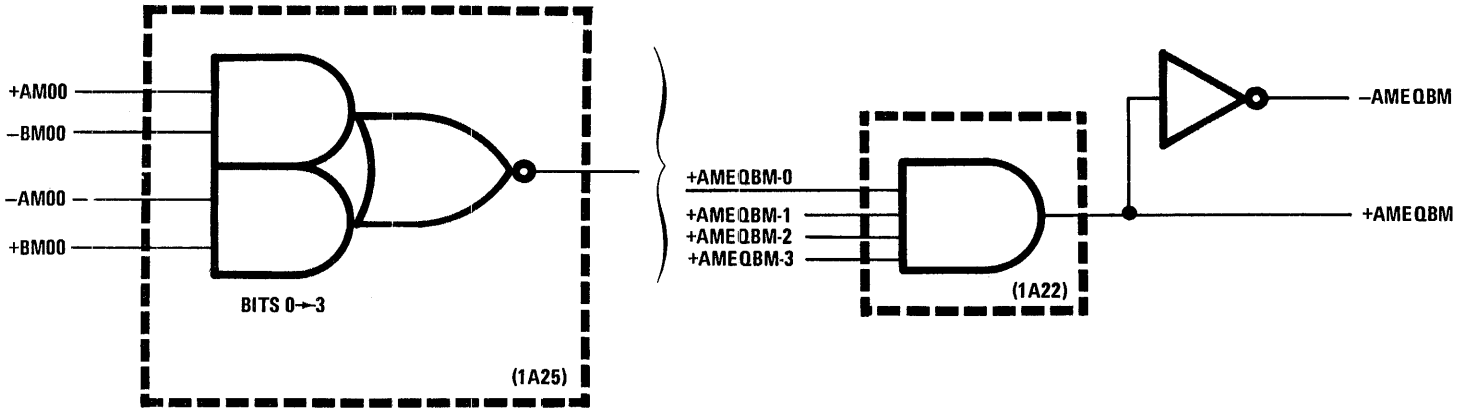


Figure 2-163. Input Signals to $A\mu \leq B\mu$ Compare Logic



TRUTH TABLE FOR BIT 0 COMPARE (TABLE WOULD HOLD TRUE FOR ALL BIT POSITIONS)

BIT POSITION	Aμ	Bμ	$\overline{A\mu}$	$\overline{B\mu}$	COMPARE RESULT
0	1	X	X	0	1
0	X	1	0	X	1

Figure 2-164. $A\mu = B\mu$ and $A\mu \neq B\mu$ Compare Logic

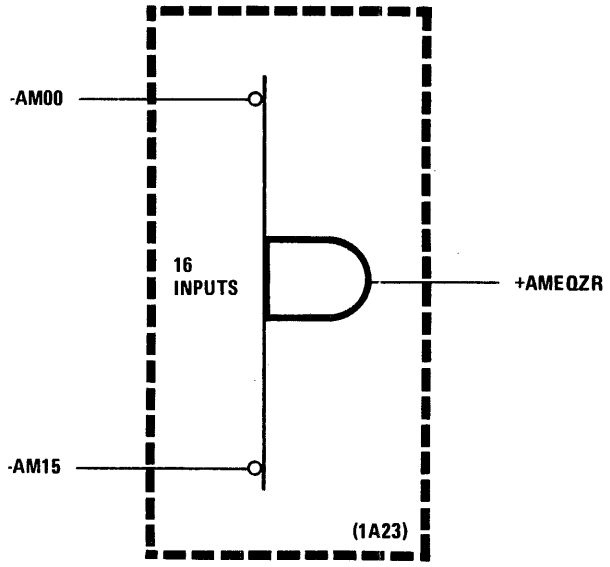


Figure 2-165. $A\mu = 0$ Compare Logic

Table 2-22. ALU Fanout Exclusive-OR and Inclusive-OR Function

Input				Output $\overline{\text{ALU-00}}$		
				Excl. OR	Incl. OR	
AM00	BM00	$\overline{\text{AM00}}$	$\overline{\text{BM00}}$	1 1	0 1	SEL-EOR SEL-OR
1	1	0	0	0	1	
1	0	0	1	1	1	
0	1	1	0	1	1	
0	0	1	1	0	0	

Algebraic and Logical Compare

The algebraic and logical compare logic, shown in Figure 2-166, evaluates the results of the $A\mu < B\mu$ and $A\mu > B\mu$ logic, and the state of the MSB's of $A\mu$ and $B\mu$ to make the compares required by the CMP (2,2), CMU (2,3), and RNI (8,0+1) μI's. For the CMP μI, both an algebraic compare (sign and magnitude) and a logical compare (magnitude only) are made on the contents of $A\mu$ and $B\mu$. The $A\mu > B\mu$ algebraic compare is made by comparing the complement state of the MSB of $A\mu$ ($\overline{\text{AM-00}}$) with the true state of the MSB of $B\mu$ (BM-NEG). If both signals are high, the contents of $A\mu$ are greater (more positive) then the contents of $B\mu$ to generate $\overline{\text{STATUS-1}}$. The $A\mu < B\mu$ algebraic compare is made by inverting the state of the above MSB's, so that the true state of the MSB of $A\mu$ is compared with the complement state of the MSB of $B\mu$. If both signals are high, the contents of $B\mu$ are greater than those of $A\mu$ to generate $\overline{\text{STATUS-2}}$. For both compares, bit FM1-005 is used to specify the CMP μI. The logical compare is made simultaneous with the algebraic compare via signals AMEQBM and AMGTEQBM to evaluate the 16-bit quantities in $A\mu$ and $B\mu$ on a magnitude basis only. The results of this compare generates $\overline{\text{STATUS-5}}$, $\overline{\text{STATUS-6}}$, and $\overline{\text{STATUS-7}}$.

For a CMU μI, the two above compares are also made; however, the algebraic compare essentially reduces in implementation to a logical compare. In effect, then, the CMU μI performs two simultaneous logical compare operations and generates two identical status bits for each compare noted: bit positions 1 and 5 if $A\mu < B\mu$, bit positions 2 and 6 if $A\mu > B\mu$, and bit positions 3 and 7 if $A\mu = B\mu$.

The $\overline{\text{STATUS-3}}$ output has two different meanings, depending on during which μI it is generated. If generated during a CMP or CMU μI, it indicates an $A\mu = B\mu$ compare condition, as previously noted. If generated during an RNI μI, however, it indicates a Link bit has been generated. For this condition, $\overline{\text{STATUS-3}}$ is generated by LINK and FMI-000, which is high if an RNI μI is being executed. All seven status bits are fed to the status logic via the ALU fan-out.

ALU FAN-OUT

The ALU fan-out logic provides output gating from the ALU for selecting one of the following eight logic functions:

1. Exclusive-OR between the outputs of the $A\mu$ and $B\mu$ registers.
2. Inclusive-OR between the outputs of the $A\mu$ and $B\mu$ registers.
3. Outputs from main storage (MS) data fan-out.
4. Outputs from status logic.
5. Outputs from adder.
6. Outputs from $A\mu$ register.
7. Outputs from $B\mu$ register.

   Together equal logic product $A\mu$ and $B\mu$.
8. Clear conditions.

There are 16 fan-out stages, one stage per bit. As shown in Figure 2-167, the upper eight stages differ from the lower eight stages because of the byte read capability from the MS data fan-out logic. Except for the $A\mu$ and $B\mu$ register outputs used to perform the exclusive-OR function, all data is gated to the fan-out logic in complemented form. The other 15 bits are gated by similar fan-out logic stages. (Note that the STAT inputs appear in only the upper eight stages of the fan-out logic since there are only eight status bits generated in the ALU.) The exclusive-OR and inclusive-OR functions of the $A\mu$ and $B\mu$ registers are performed by the two top AND gates of the fan-out logic. The top gate is fed with $A\mu$ and $B\mu$ register bits in true form, and enabled by SEL-EOR. The next gate is fed with $A\mu$ and $B\mu$ register bits in complement form, and enabled by SEL-OR. When both enable signals are high, the exclusive-OR function (either one, but not both) is performed on the two register outputs. When enable SEL-EOR is low and enable SEL-OR is high, the inclusive-OR function (either one or both) is performed. These two logic functions are summarized in Table 2-22.

Each of the remaining six functions is presented with its respective enable to one of the remaining AND gates. When the enable is high, the data is gated and inverted to true form. The logical product of $A\mu$ and $B\mu$ is realized by gating the contents of both registers simultaneously. Data from the MS data fan-out logic can be gated in either word form (16 bits) or byte form (8 bits).

These data transfers occur in connection with operations involving the MS interface logic; details of these data transfers, therefore, are discussed in the paragraph titled Main Storage Interface Word and Byte Read Functions. Use of the ALU fan-out logic to generate clear conditions is discussed in the paragraph titled System Reset Conditions.

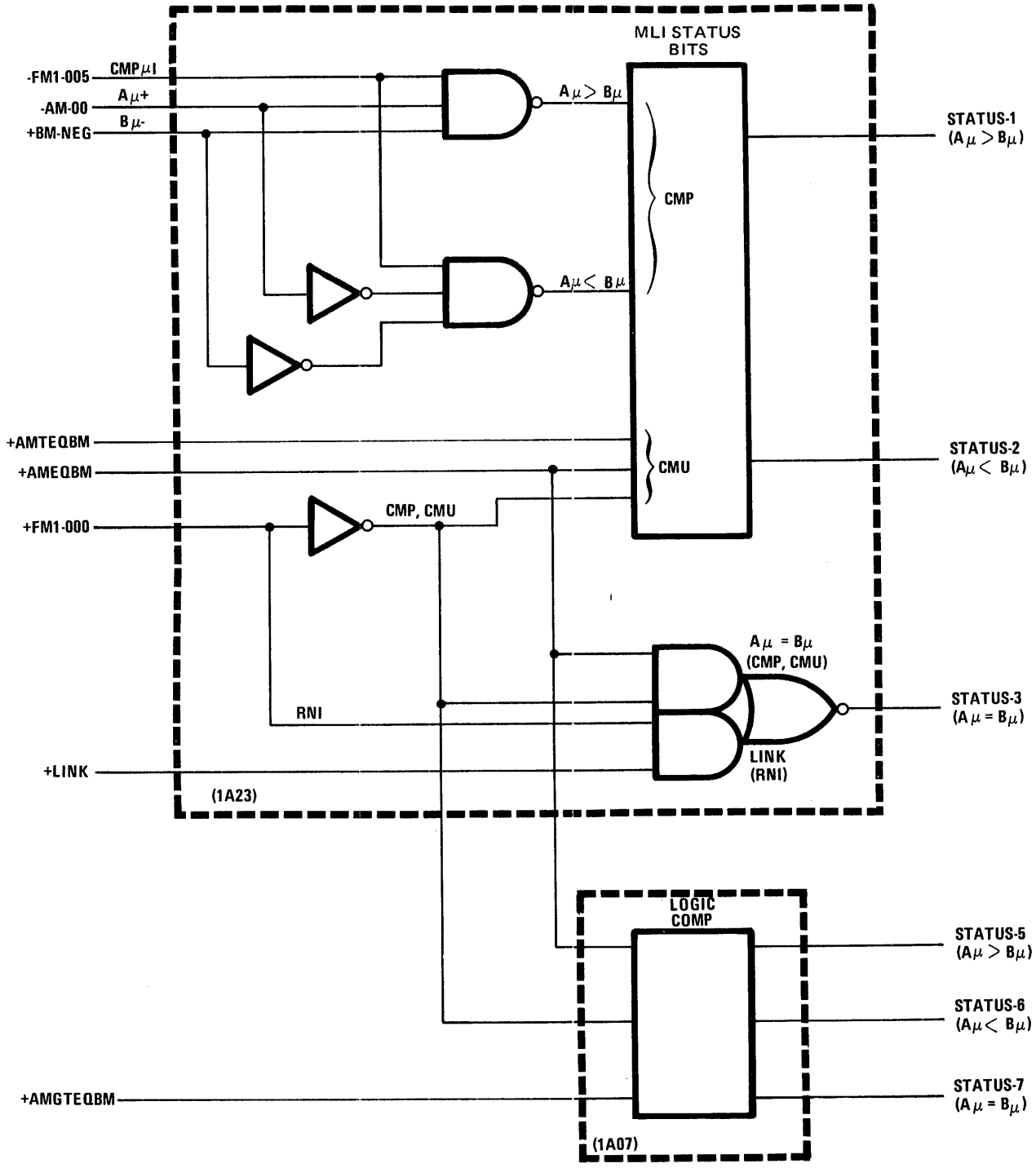


Figure 2-166. Sign and Magnitude Compare Logic

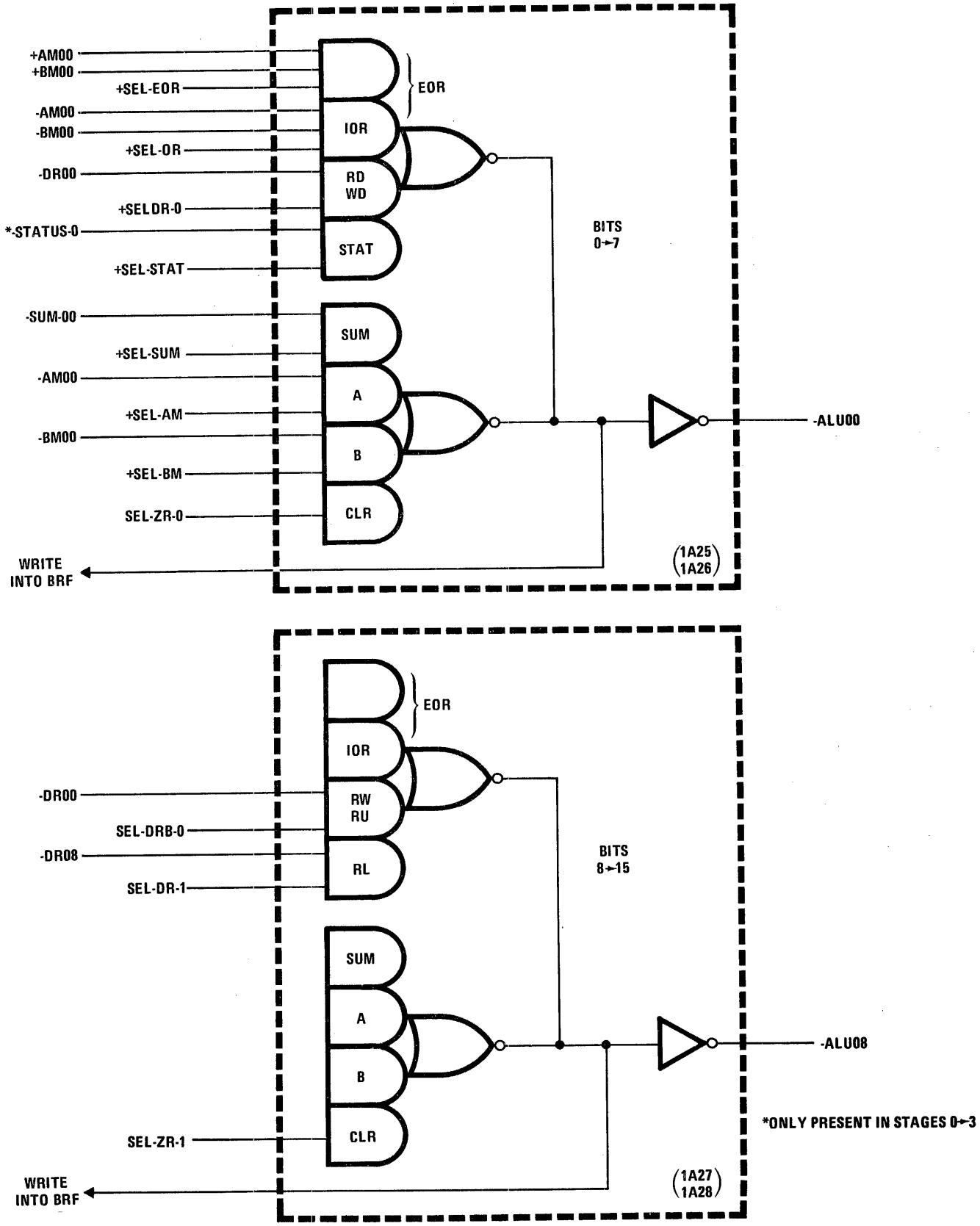


Figure 2-167. ALU Fan-Out

STATUS LOGIC

The status logic takes the results of the compare evaluations discussed in the previous paragraph and arranges for their storage in a specified register during execution of a compare μI. The logic also detects and processes the arithmetic status bits, Overflow and Link generates during arithmetic operations. These two bits are conditionally stored in a specified register via an RNI1 or RNI2 μI, and unconditionally carried along with the address of the present μI contained in the Sμ register. The description of the status logic is divided into the following two sections:

1. detection, reading and writing of the arithmetic status bits into the Pμ register, and

2. storing of the compare and arithmetic status bits in a specified register.

Arithmetic Status Bit Detect

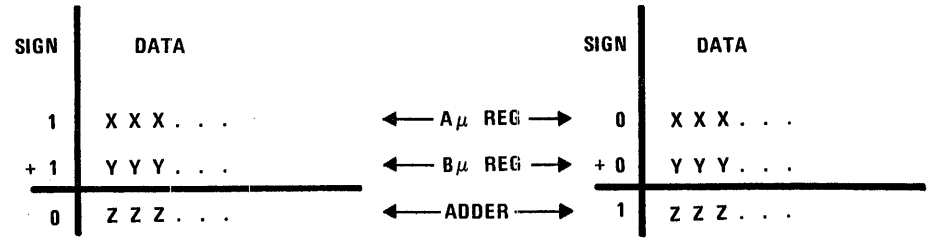
During arithmetic operations, two status bits are generated along with the operand result: Overflow and Link. The Overflow bit indicates that during a 16-bit operand add operation, the MSB of the sum (bit 01) has overflowed into the sign bit position (bit 00). The Link bit indicates that during one iteration of a multiple-precision add operation, a carry-out has been generated from bit position 00 of the adder which must be added to the partial sum to be processed during the following iteration. The Link bit essentially indicates that the partial sum processed during the present iteration is *linked* with that to be processed during the active processor's next time slice, since they are part of the same over-all operation. The bit is normally generated during execution of an MLI containing several SUM or DSUM μI's that are executed during different time slices.

Depending on the sign of the addend and augend, two types of overflow are possible: positive overflow or negative overflow (underflow). The two types of overflow are depicted in Figure 2-168. Positive overflow, shown in part *a* of Figure 2-168, is produced when the MSB of the sum of two negative numbers overflows into the sign bit of the sum. The result is to make the sum look like a positive number (sign bit of "0"). Negative overflow, shown in part *b* of Figure 2-168, is produced when the MSB of two positive numbers overflows into the sign bit of the sum. The result is to make the sum look like a negative number (sign bit of "1"). The two overflow types are generated by the logic shown in Figure 2-169. This logic evaluates the state of the MSB of the Aμ and Bμ registers, and the adder to detect an overflow condition.

The Overflow and Link bits generated during a sum operation (SUM or DSUMμ I) are detected and routed to both the μ status current register and the μ status write register, as shown in Figure 2-170. The μ status write register holds the detected status bits for the remainder of the time slice, and then transfers these bits to the active processor's Pμ register in the ERF at W0. The register is fed with the detected bits through selector 4 when enabled by EN-SUM. This enable is generated whenever a SUM or DSUM μI is executed (FM2-005 high or low) and the associated MLI is not a 50 through 53, or if a DSUM μI is generated ($\overline{\text{FM2-005}}$ low) and the associated MLI is a 50 through 53. The μ status current register performs a dual function. If new status bits are detected during the present time slice, it holds these bits for possible transfer to another register upon execution of a store status μI (such as an RNIμ I). For this purpose, the current register is loaded from the status detectors through selector 3 at any time other than E750 through E000 (E789XX-L low). These previously detected bits are obtained from either the processor's assigned Pμ register if not running in the Consecutive Cycle (CC) mode, or from the μ status write register if running in the CC mode. If not in the CC mode, the status bits are read from Pμ during the RO cycle of the present time slice in the manner as the starting μI address. However, the status bits will not be useful to the present time slice until E0; therefore they must be held in the status buffer register during R1. This is accomplished by the data path listed in part *a* of Figure 2-171. Along with this path are shown the corresponding times at which data is enabled through a selector or clocked into a register. Note that the status bits are held in the buffer register during R1 of the present time slice (actually from E680 of the previous time slice to E000). If in the CC mode, the status bits are read from the write register at E880, at the same time that the starting I address is read from Pμ. The data path and associated timing for a CC condition is shown in part *b* of Figure 2-171. For this case, however, the enable used to gate status from the write to selector 1 occurs at E880, which is the same time that the buffer register would be clocked if in the CC mode. Since there is not enough time to put status into the buffer register from selector 1, the status bits instead bypass the buffer register and instead are gated directly to selector 2. Note from the timing that the same one-minor cycle buffer delay is still achieved (from E880 until E000) when bypassing the buffer register in CC mode as for the case of not operating in the CC mode (from E680 until E000).

Status Bit Storage

As discussed previously, the compare and arithmetic status bits generated during a time slice can be stored in a selected register. Generally, the register selected is the Condition register (register 8 of the BRF); however, under



A. POSITIVE OVERFLOW

B. NEGATIVE OVERFLOW

Figure 2-168. Overflow Definition

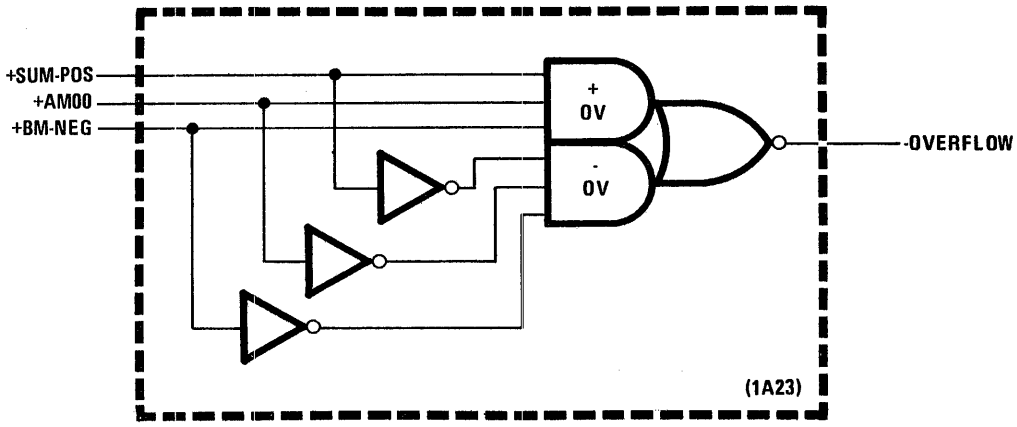


Figure 2-169. Overflow Detection

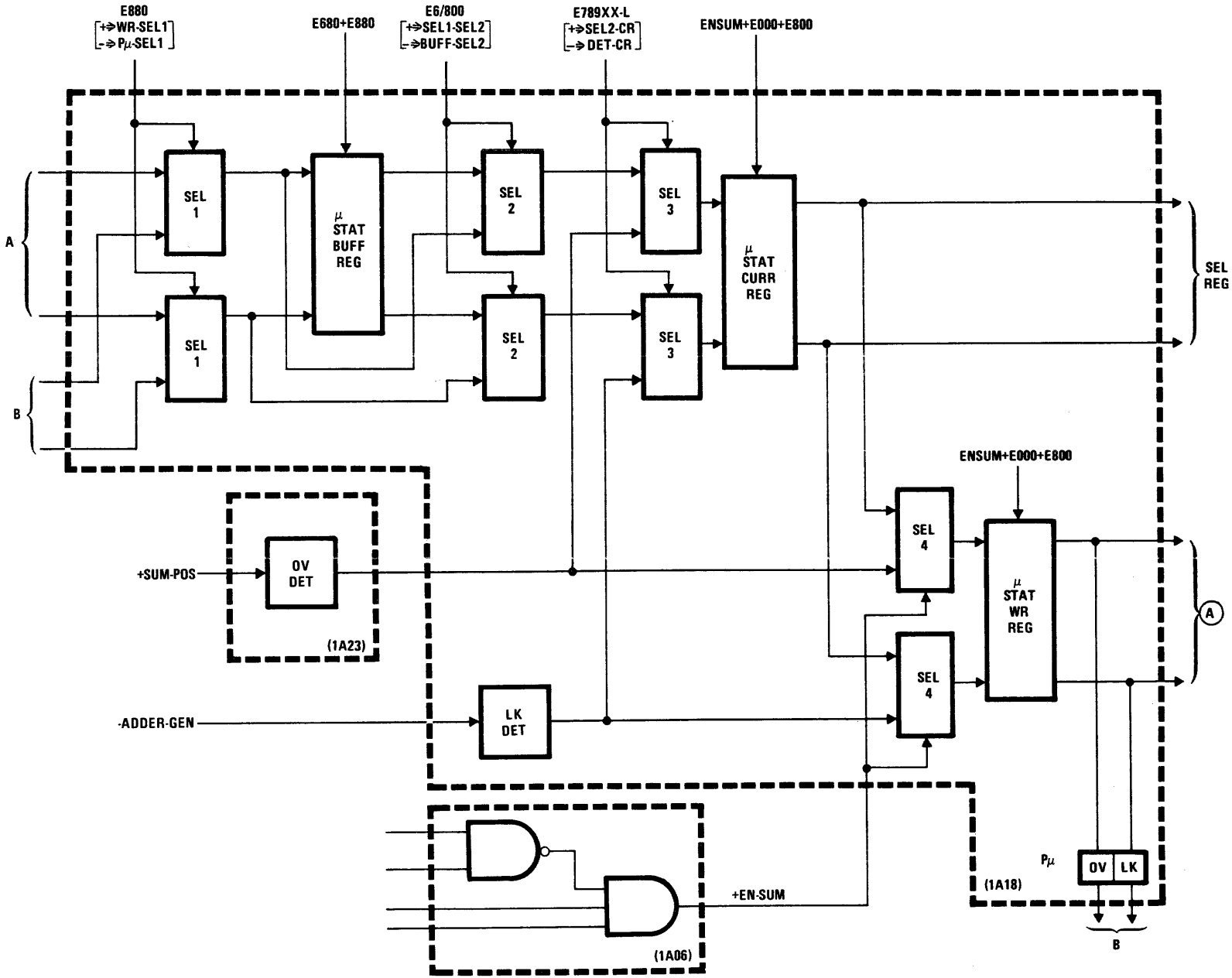


Figure 2-170. Arithmetic Status Bit Detect Logic

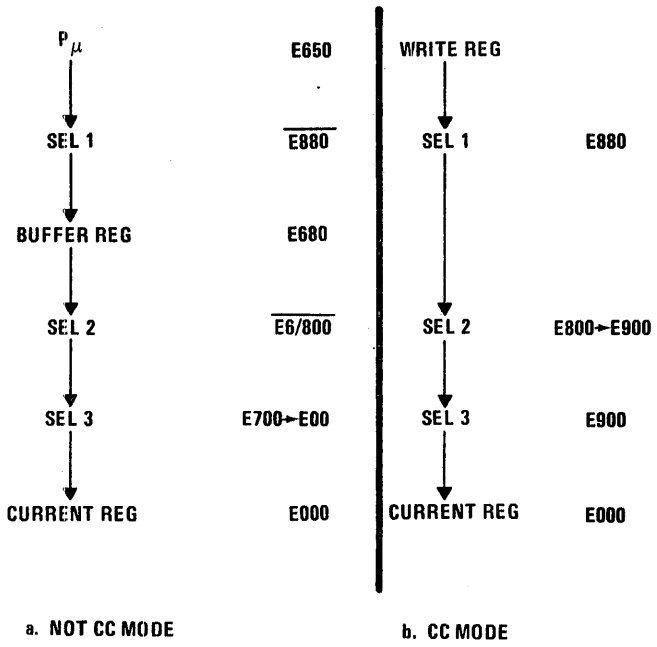


Figure 2-171. Status Bit Read Data Flow

certain conditions the register specified will be a *transient* register. Storing of the status bits is accomplished by the CMP and CMU μI's, which store the six compare status bits ($A\mu > B\mu$, $A\mu < B\mu$ and $A\mu = B\mu$) and the RNI1 and RNI2 μI's, which store the arithmetic status bits, OVERFLOW and LINK. These μI's implement the storage operation via the status bit storage logic of Figure 2-172. This logic feeds eight status lines to the ALU fan-out logic. When gated by $\overline{\text{SEL-STAT}}$, the fan-out network routes the eight status lines to bit locations 0 through 8 of the selected register in the BRF for storage.

The three algebraic compare bits are routed to the ALU fan-out via the $\overline{\text{STATUS-1}}$, $\overline{\text{STATUS-2}}$, and $\overline{\text{STATUS-3}}$ lines. The $\overline{\text{STATUS-3}}$ line is also used to carry the LINK status bit during execution of an RNI μI. This μI also gates the OVERFLOW bit over the $\overline{\text{STATUS-0}}$ line when high. The three logical compare bits are routed to the ALU fan-out via the $\overline{\text{STATUS-5}}$, $\overline{\text{STATUS-6}}$, and $\overline{\text{STATUS-7}}$ lines. To enable storing a complete byte of status information, a $\overline{\text{STATUS-4}}$ line is added to the seven above lines. This line is tied to a logical 1 via the +5 vdc supply to enable storing a "0" in the corresponding bit position 4 of the register.

The data present on the four status lines is gated through the ALU fan-out via $\overline{\text{SEL-STAT}}$, which is generated for the CMP, CMU, RNI1 and RNI2 μI's. The status bits are then stored in a register of the BRF designated by the X-field of the particular μI. The format of the register, showing locations of the arithmetic and compare status bits, appears in Figure 2-172. For the RNI1 and RNI2 μI's, the X-field will designate either the Condition register or an irrelevant register, depending on what type

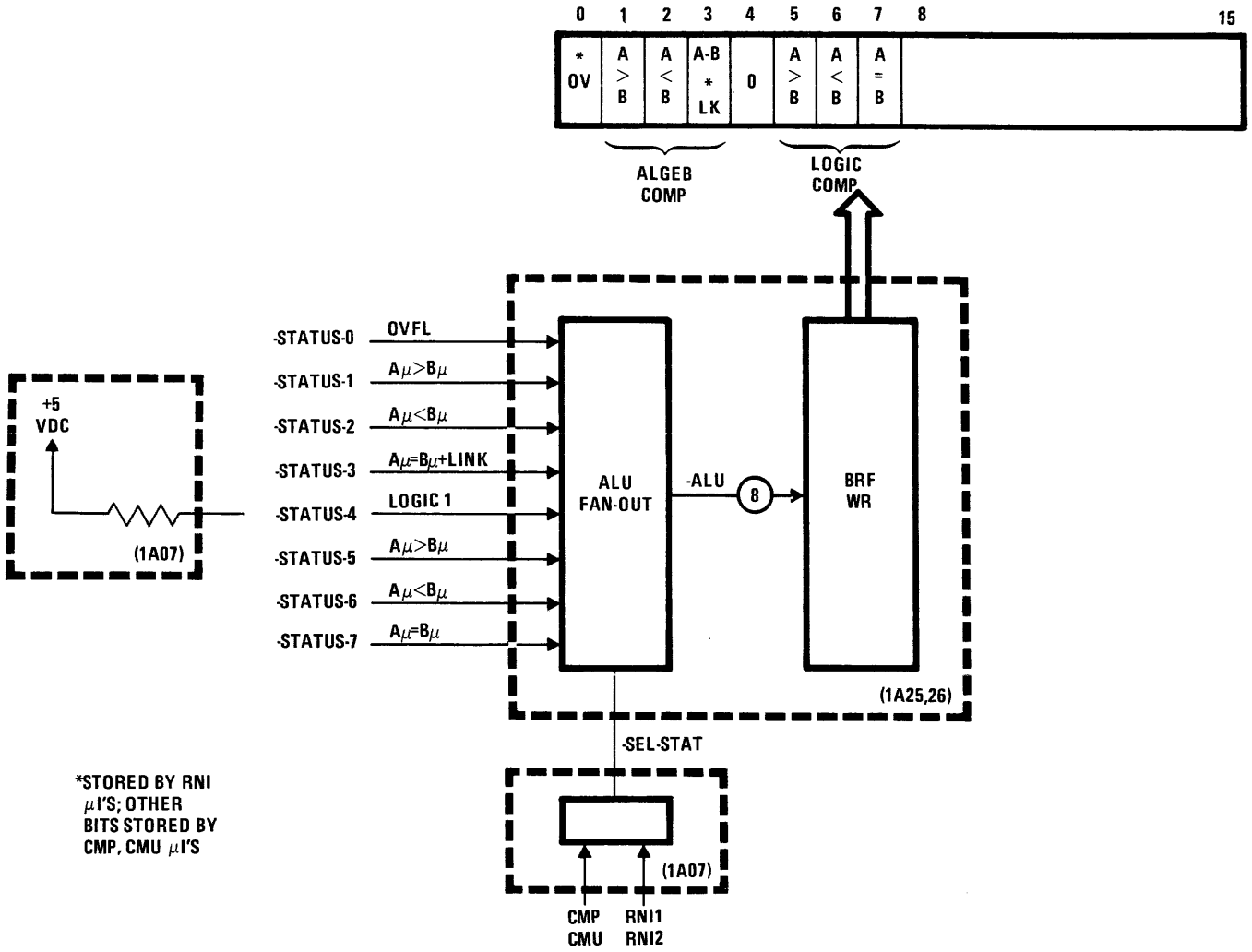


Figure 2-172. Status Bit Register Storage

of operation the RNI μI is a part of. If the RNI μI is associated with an arithmetic operation, the register specified is the Condition register so that LINK and OVERFLOW may be saved for future use. If the RNI μI is not associated with an arithmetic operation, the X-field is coded to specify some irrelevant register. This is to satisfy the requirement of specifying some register by the X-field.

CONSTANT GENERATOR

Inputs and Outputs

The constant generator provides values of constants to the Bμ register as required by the following μI's: LS1, LSE, LS2, LSF, LBB, LBB-, EBU, EBL, DIG, and CORC. The constant generator consists of four pairs of MSI selector elements, each element containing two four-input/one-output selectors. Inputs to each selector are selected by two select signals, $\overline{\text{ENCG-0}}$ and $\overline{\text{ENCG-1}}$, which feed both selectors of an element. A simplified block diagram of the constant generator, showing one pair of selector elements, is shown in Figure 2-173. Each element of the pair is fed with two bits of a particular input.

Depending on the μI, the outputs are used to develop one of the following:

1. bits 0, 1, 8, and 9 of a 16-bit word constant,
2. the two most significant bits of the two 8-bit byte constants, or
3. the two most significant bits of alternate 4-bit nybl constants.

The other six elements of the constant generator are paired together in the same manner to generate bit pairs separated by eight bits. The select signals are enabled by μI's to select corresponding inputs as shown in Table 2-23.

Table 2-23. Constant Generator Input Selection

Micro Instructions	Select Signal State		Inputs Enabled
	—ENCG-0	—ENCG-1	
LS1, LSF, LS2, LSE	0	0	+5 VDC, FORCE 1
DIG	1	0	ENT-CG
LBB, LBB- (a·b = 0·0, a·b = 1·0, a·b = 0·1)	0	1	BIT-CG
LBB, LBB- (a·b = 1·1)	1	0	ENT-CG
EBU, EBL	1	0	ENT-CG
CORC	1	1	DDG-CG

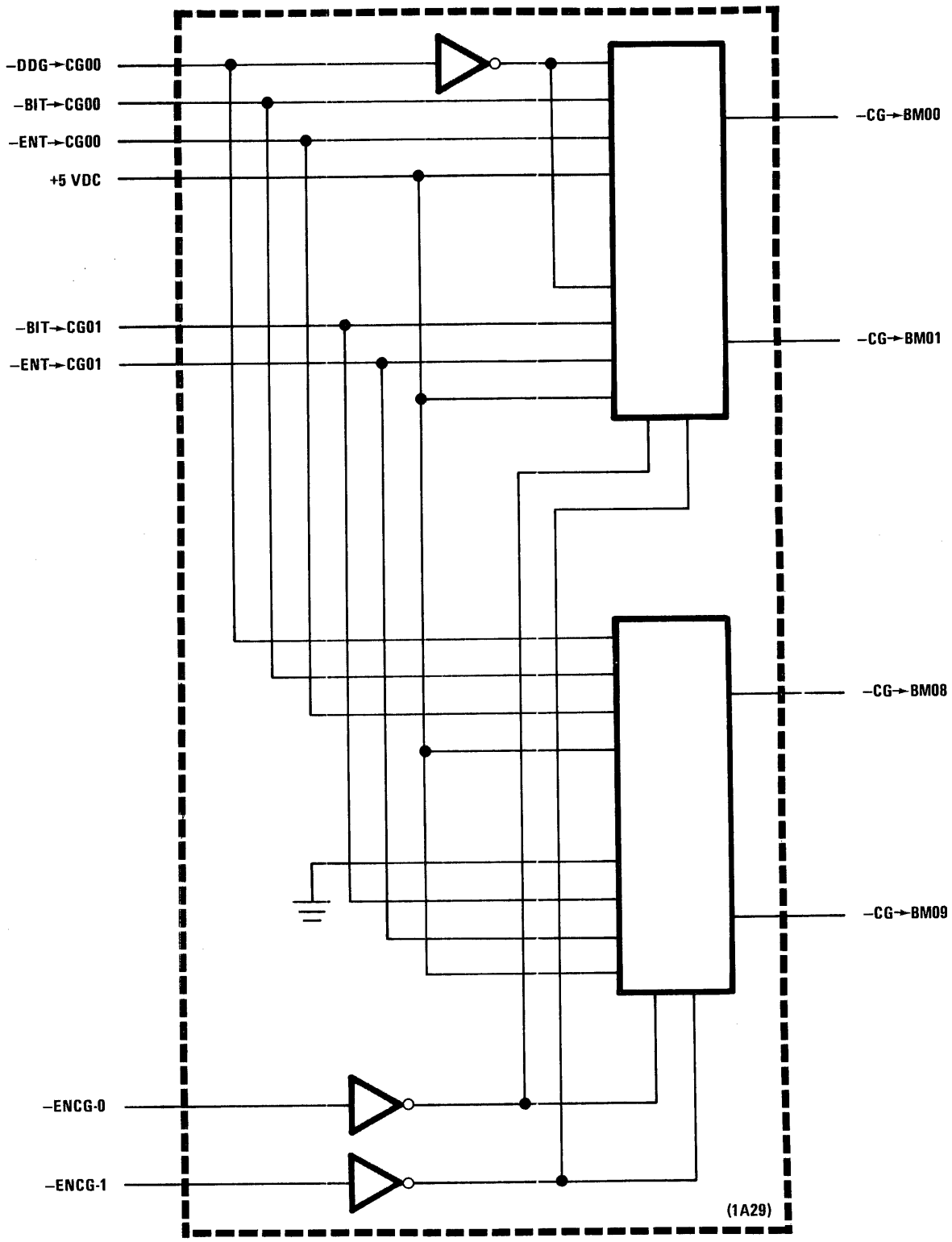


Figure 2-173. Constant Generator

The resulting constants generated by inputs selected for particular μI's are shown in Table 2-24. (These constants represent the final values stored in the Bμ register, not the outputs from the constant generator. Depending on the μI, the constant generator outputs may require complementing prior to Bμ register storage.)

Generation of each constant via the corresponding μI is discussed in the following paragraphs.

Load S Micro Instructions

Constant values required for the load S μI's are as follows:

LS1: 0's to Bμ register

LSF: -1 to Bμ register

LS2: +1 to Bμ register

LSE: -2 to Bμ register

To generate these constants, select signals $\overline{\text{ENCG-0}}$ and $\overline{\text{ENCG-1}}$ are "0" and "0" as shown in Table 2-23. The state of these select signals causes the +5 vdc logic level to be gated as outputs CG-BM00 through CG-BM14 and the FORCE1 signal to be gated on CG-BM15 (the MSB) of the constant. The FORCE1 signal is low for the LS1 and LSF μI's and high for the LS2 and LSE μI's.

Table 2-24. Constant Generator Output Constants

μI \ Bits	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
LS1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LSF	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
LS2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LSE	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
LBB (a·b = 0·0)	2^{15-X}															
LBB (a·b = 1·0)	$2^{15-(XVR_1)}$															
LBB (a·b = 0·1)	$2^{15-(XVR_2)}$															
LBB (a·b = 0·1)	$2^{15-PROC}$								2^{7-PROC}							
LBB- (a·b = 0·0)	$\overline{2^{15-X}}$															
LBB- (a·b = 1·0)	$\overline{2^{15-(XVR_1)}}$															
LBB- (a·b = 0·1)	$\overline{2^{15-(XVR_2)}}$															
LBB- (a·b = 0·1)	$2^{15-PROC}$								2^{7-PROC}							
DIG	X				X				X				X			
EBL (N)	0	0	0	0	0	0	0	0	N							
EBU (N)	N															
CORC CORC	1101 + 0011				1101 + 0011				1101 + 0011				1101 + 0011			

The resultant outputs are then either $FFFF_{16}$ for the LS1 and LSF μI's and $FFFF_{16}$ for the LS2 and LSE μIs. These outputs are presented to the Aμ and Bμ register fan-in logic and are gated through the logic by ENCG BM, as discussed in the paragraph titled Aμ and Bμ Fan-In and shown on the simplified logic of Figure 2-174. This logic inverts the inputs fed to it so that now the constant corresponding to LS1 and LSF is 0000_{16} and that for LS2 and LSE is 0001_{16}. These constants are then clocked into the Bμ register in either true form or complement form by means of the forced set and forced clear conditioning of the Bμ register flip-flops discussed in the paragraph titled Aμ and Bμ Registers. The constants are already in the desired form for the LS1 and LS2 μI's, so they are stored in the Bμ register in true form. However, the constants must be complemented for the LSF and LSE μI's so that they represent -1 and -2 in two's complement form. Therefore, the constants are stored in the Bμ register in complement form for these two μI's so that the final form of these constants is $FFFF_{16}$ (-1_{10}) and $FFFE_{16}$ (-2_{10}).

Load B Bit Micro Instructions

The LBB μI's set a bit into the Bμ register derived from values of the μI X-field and associated MLI R_1 and R_2 fields. The manner in which these various fields are used to set the bit is determined by the μI *a* and *b* designators, as summarized in Table 2-25. When set to the value determined by the *a* and *b* designators, the bit in Bμ forms a binary number whose value can be indicated as a *power of 2* as shown in Table 2-24. (For example, if the LBB μI specifies that bit 6 is to be set in Bμ and a · b = 0 · 0, the resultant binary number indicated is 2^{15-6} or 2^9 (512_{10}).)

Setting the bit in Bμ for *a* and *b* values of 0 · 0, 0 · 1, and 1 · 0 is performed by the logic of Figure 2-175. Depending on the particular value of the *a* and *b* designators, the bit to be set is accomplished by one of the following:

1. a decoding of the μI X-field (FM1-112→115),
2. an ORed combination of the μI X-field and MLI R_1-field (FR-008→112), or
3. an ORed combination of the μI X-field and MLI R_2-field (FR-012→015).

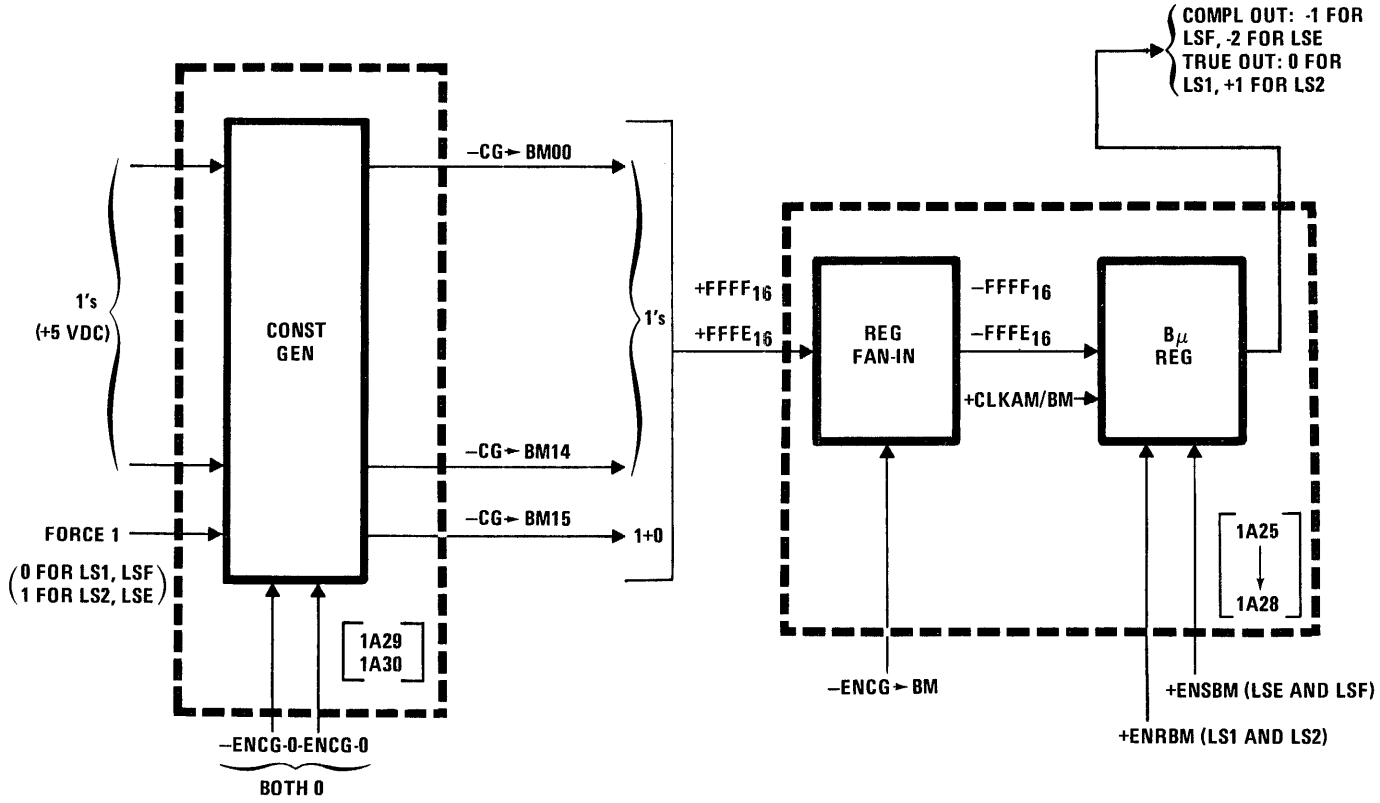


Figure 2-174. Generation of Constants for Load S μI's

Table 2-25. Field Selection for Setting Bits in Bμ Via Load B μI's

Designator Value		Determination of Bit in Bμ to be Set
a	b	
0	0	μI X-field (bits 12-15)
0	1	Inclusive OR of μI X-field and MLI R_2 field (bits 12-15)
1	0	Inclusive OR of μI X-field and MLI R_1 field (bits 8-11)
1	1	Active processor determined by ENT bits to constant generator

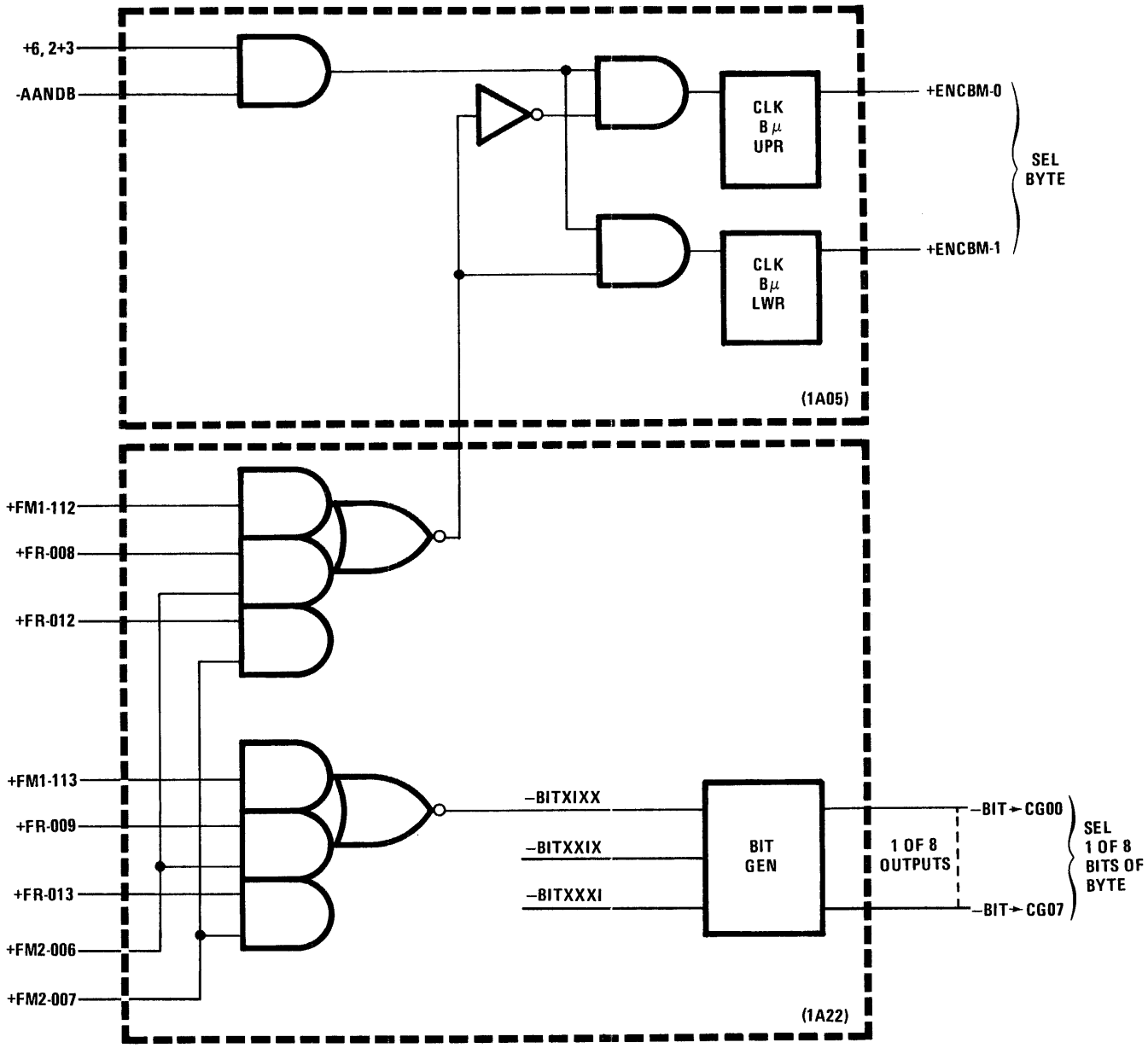


Figure 2-175. Generation of Constants for Load B μI's (a·b = 0·0, 1·0, and 0·1)

In each instance, the four bits comprise a bit select code that defines, in binary form, the bit of Bμ to be set. The MSB of this bit select code, BIT1XXX, selects either the left-most or the right-most byte of Bμ in which the bit is to be set. The *a* and *b* designators (FM2-006 and FM2-007) determine how BIT1XXX is formed: either directly or by the exclusive OR operation described above. This signal is used in either its generated complement form to clock the lower half of Bμ (bit positions 8 through 15) via ENCBM-1, or inverted to true form to clock the upper half (bit positions 0 through 7) via ENCBM-0. In both cases, the enable for generating either of these two clock signals is provided by 6,2+3 · AANDB which defines the LBB and LBB- μl's for *a* and *b* values of 0 · 0, 0 · 1, and 1 · 0. Selecting a particular bit of the selected byte is performed by the remaining three bits of the select code: BITX1XX, BITXX1X, and BITXXX1. These bits are generated in exactly the same way as the MSB. Generation of BITXIXX is shown in Figure 2-175. These three bits are fed to a three-input/one-of-eight line decoder to generate a bit select signal, BIT-CG. The result is fed to the constant generator, to set the specified bit of Bμ.

Setting of a bit in Bμ for *a* and *b* values of 1·1 is performed by the logic of Figure 2-176. This logic sets a bit in each byte of the Bμ register corresponding to the processor that is presently active. (For example, if processor 0 is presently active, bit 0 of the upper byte and bit 8 of the lower byte of Bμ will be set.) Using processor 0 as an example, bits 0 and 8 of Bμ are set by one input to the constant generator. $\overline{\text{ENT-CG00}}$, when enabled by $\overline{\text{ENCG-0}}$ and $\overline{\text{ENCG-1}}$ equal to "1" and "0", respectively. Signal $\overline{\text{ENT-CG00}}$, in turn, is produced by STATE0 from the priority logic which indicates that processor 0 has been assigned the present time slice. This processor signal is enabled by $\overline{\text{FM1-000}}$ which is high for μl's 0,0 through 7,3 (which includes the LBB and LBB- μl's). Since a bit is to be written in both bytes of Bμ, both enable clock upper byte and lower byte signals must be activated. This is accomplished by generating both ENCBM-0 and ENCBM-1 simultaneously via 6,2+3. The value function code appears alone with no instructions or values of *a and b* designators; therefore, the enable clock signals are produced for all combinations of *a* and *b* values including *a·b* = 1 · 1. (Generation of these enable clock signals for unrestricted values of *a* and *b* does not interfere with clock enables generated for restricted values of *a* and *b* as discussed in the last paragraph since different inputs to the constant generator are used for the two variations of Load B μl's: $\overline{\text{BIT-CG}}$ inputs for Load B μl's with *a* and *b* values of 0·0, 1·0, and 0·1; and ENT CG inputs for Load B μl's with a and b values of 1·1.) The LBB μl sets the bit number in Bμ in true form; the LBB-μl sets the bit number in Bμ in complement form. (See the paragraph titled Aμ and Bμ Registers for a discussion of setting bits in Bμ in either true or complement form.)

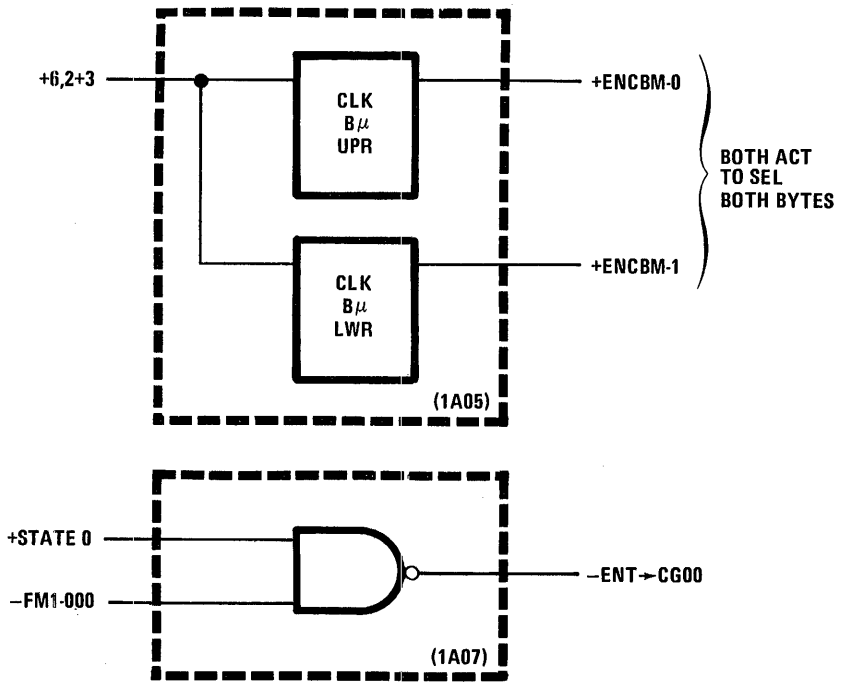


Figure 2-176. Generation of Constants for Load Bμ I (a·b = 1·1)

Enter B Micro Instruction

The Enter Bμ I's provide for entering an eight-bit value specified by the μI N field (bit positions 4 through 7 and 12 through 15) into either the upper byte (EBU μI) or lower byte (EBL μI) of the Bμ register. The μI's are implemented as shown in the logic of Figure 2-177. The eight bits of the μI N field are applied to the ENT–CG fan-out logic. Bits 4 through 7 of the N field are enabled through the fan-out logic by 10XX and bits 12 through 15 are enabled by 1XXX. Both enables 10XX and 1XXX are high for the EBU (1010) and EBL (1011) μI's. (Enable 10XX is used for enabling bits 4 through 7 of the μI to specifically eliminate the DIG and CORC μI's, which also load Bμ with bits from this field during their execution.)

The EBU μI enters the value specified by the μI N field into bits 0 through 7 of Bμ in true form and leaves the contents of bit positions 8 through 15 unchanged. It accomplishes this by generating ENRBM-0 to first set bits 0 through 7 to "0"s, and then enters the value of N via ENCBM-0.

The EBL μI enters the value of N in true form in bit positions 8 through 15 of Bμ and sets bits 0 through 7 to "0"s. It does this by generating ENRBM-0 and ENRBM-1 to set both bytes of Bμ to "0"s, and then enters the value of N into bit positions 8 through 15 via ENCBM-1.

DIG Micro Instruction

The DIG μI loads each four-bit nybl position of the Bμ register with a value specified by the μI X-field. The μI is implemented as shown in Figure 2-178. The μI X-field is presented to the $\overline{\text{ENT-CG}}$ fan-in logic as bits 12 through 15 of the Fμ register. These bits are fed in parallel to two gates each, so that the X-field value is enabled through the gates as $\overline{\text{ENT-CG00}}$ through $\overline{\text{ENT-CG03}}$ and as $\overline{\text{ENT-CG04}}$ through $\overline{\text{ENT-CG07}}$. These two 4-bit nybls are routed in parallel to the four 4-bit nybl positions of Bμ so that the original value defined by the X-field is copied into Bμ as four values. This one-to-four transformation is shown in Figure 2-179. Enabling of the four-bit X-field through the $\overline{\text{ENT-CG}}$ fan-in logic is accomplished by enable 11XX, which gates the X-field through the $\overline{\text{ENT-CG00}}$ through $\overline{\text{ENT-CG03}}$ gates; and enable 1XXX, which gates the X-field through the $\overline{\text{ENT-CG04}}$ through $\overline{\text{ENT-CG07}}$ gates. Both enable clock signals are activated for this μI so that the X-field value can be written into both upper and lower halves of Bμ.

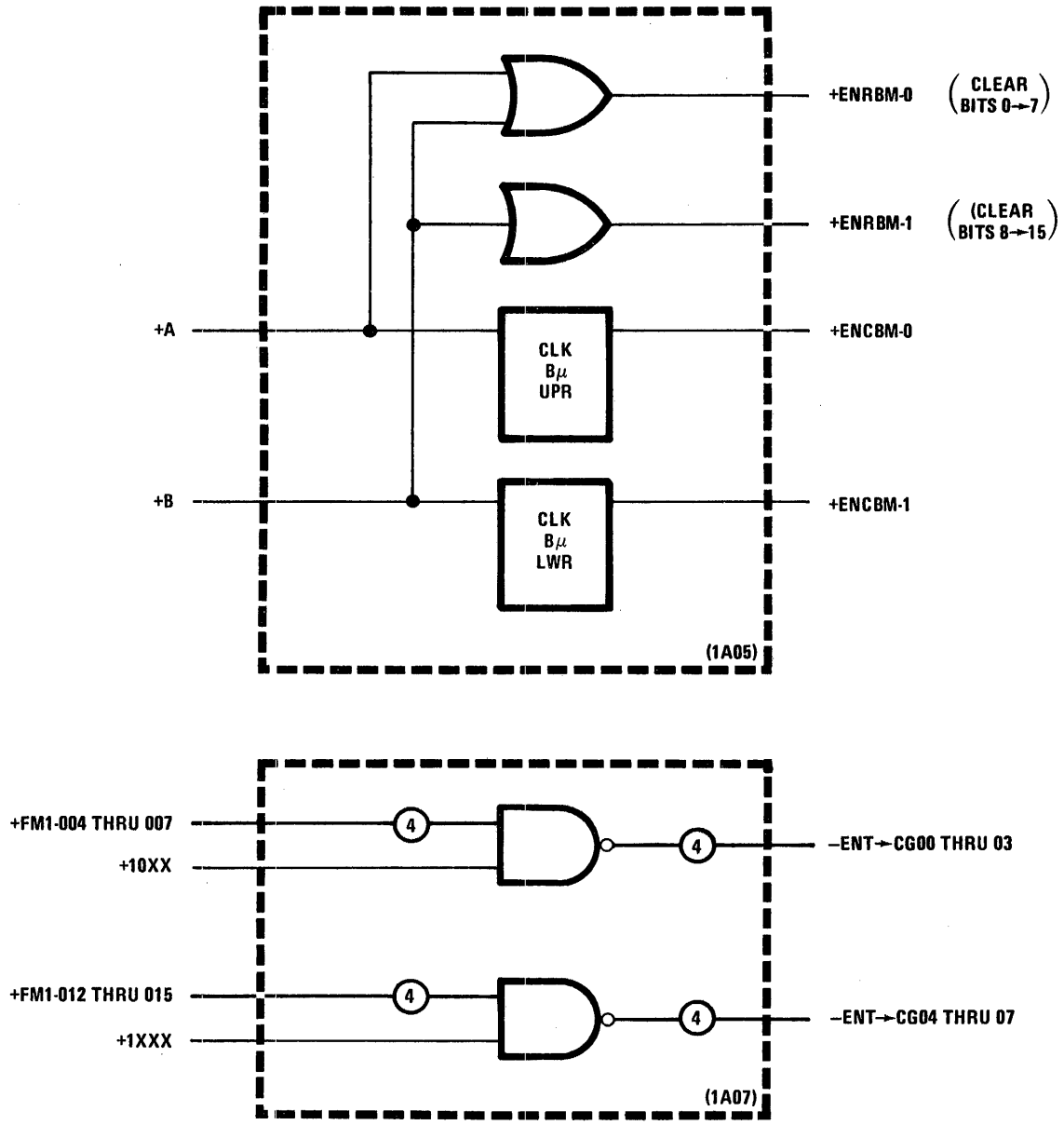


Figure 2-177. Generation of Constants for Enter B μI

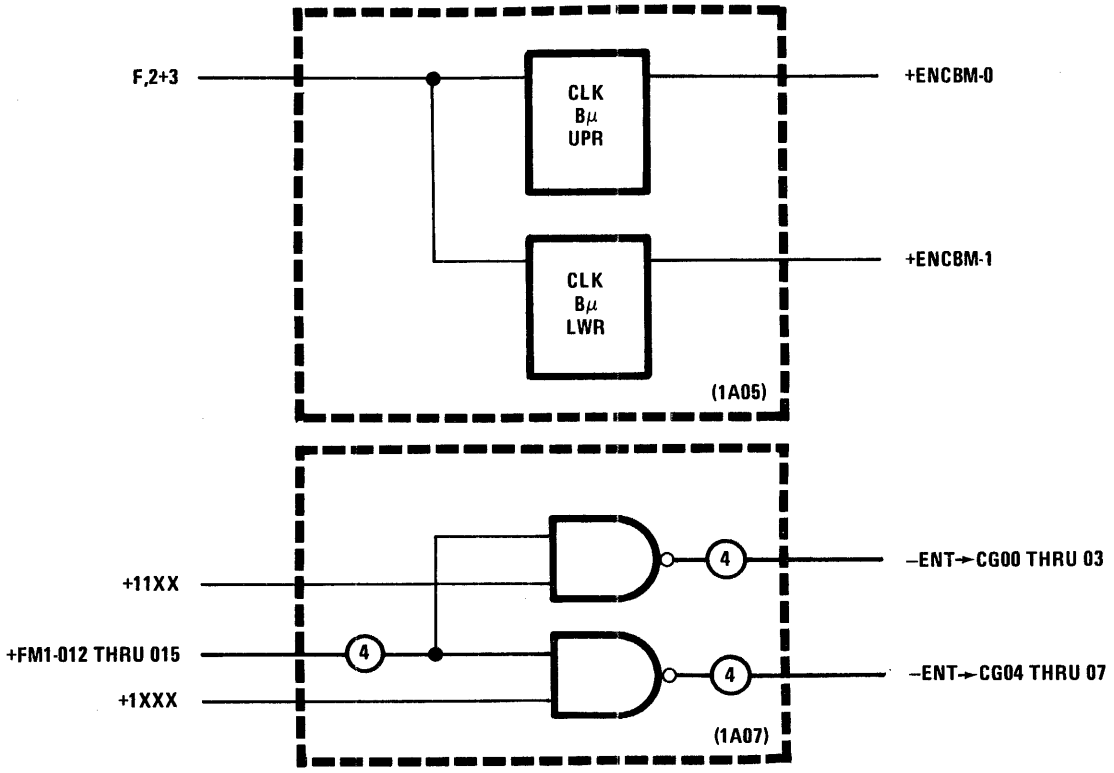


Figure 2-178. Generation of Constants for DIG μI

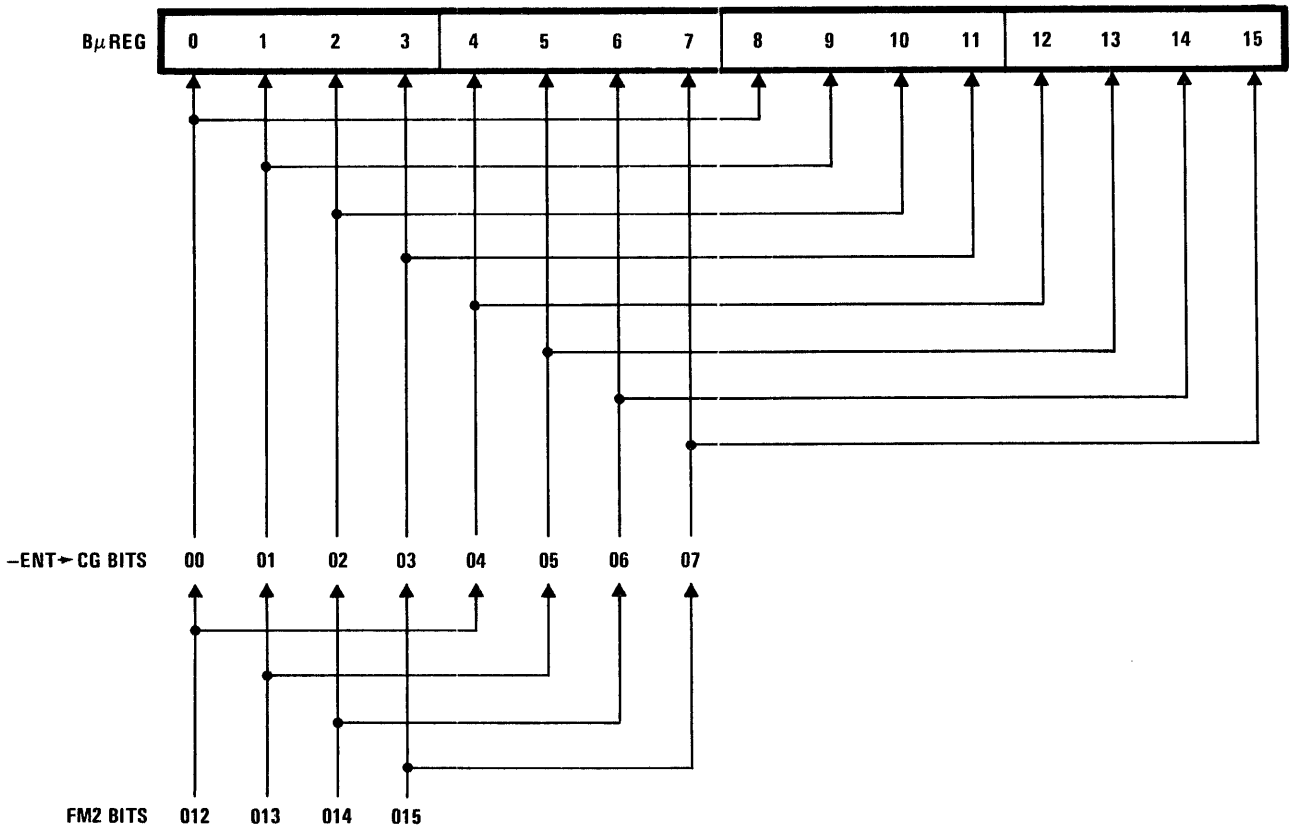


Figure 2-179. X-Field Fan-Out for DIG μI

CORC Micro Instruction

The CORC μI generates values of 3_{16} or D_{16} which are used to correct each decimal digit after executing a DSUM μI. The decision to generate either 3_{16} or D_{16} is made on the basis of whether or not a carry was generated during manipulation of the particular decimal digit. The μI is implemented by the logic of Figure 2-180, which shows generation of a decimal correction value associated with bits 0 through 3 of a decimal arithmetic operation. The input to the constant generator consists of $\overline{\text{DDG-CG00}}$ from the bit 0 through 3 stage of the Inner Carry register, and a ground connection. The $\overline{\text{DDG-CG00}}$ input is applied in generated complement form to bit stage 2, and inverted to true form and applied to bit stages 0 and 1 of the constant generator. Bit 3 is tied permanently to ground since its value is "0" regardless of which correction value is generated. Assuming that a carry is generated from stage 0 of the decimal operation, $\overline{\text{DDG-CG00}}$ is low and the $\overline{\text{CG-BM}}$ outputs are as shown in Table 2-26. Since the constant generator outputs are in complement form, they are inverted by the ALU fan-in logic prior to storage in Bμ. The result is 0011_2 (3_{16}) which is the required correction code. If no carry was generated from stage 0 of the decimal operation, the inputs to bit positions 0, 1, and 2 are reversed. The result stored in Bμ for this situation is 1101_2 (D_{16}), or the two's complement of 3_{16}.

BIT SENSE AND SENSE/TOGGLE

The bit sense and sense/toggle logic is used to execute the bit sense and bit sense/toggle (E,X,1) μI's. The two bit sense (E,0,1 and E,1,1) μI's scan the Aμ register contents sequentially from bit position 00 through bit position 15 for the presence of the first "0" or "1". In addition, the Bμ register contents are incremented by 1 for each bit position scanned without a find. The two bit sense/toggle (E,2,1 and E,3,1) μI's are executed similarly except that they additionally toggle the sensed bit to the alternate state. Logic for executing these four μI's is shown in Figure 2-181. The Aμ register contents are fed to priority encoder logic, which produces a BCD output corresponding to the first "0" or "1" detected in Aμ. Since the priority logic, however, cannot itself determine that it is to sense either the first "0" or the first "1" (all it can sense is the first "0"), the data presented to it must first be preconditioned to a form that will allow the priority encoder to, in effect, sense for the right bit state. This preconditioning is performed by routing the data

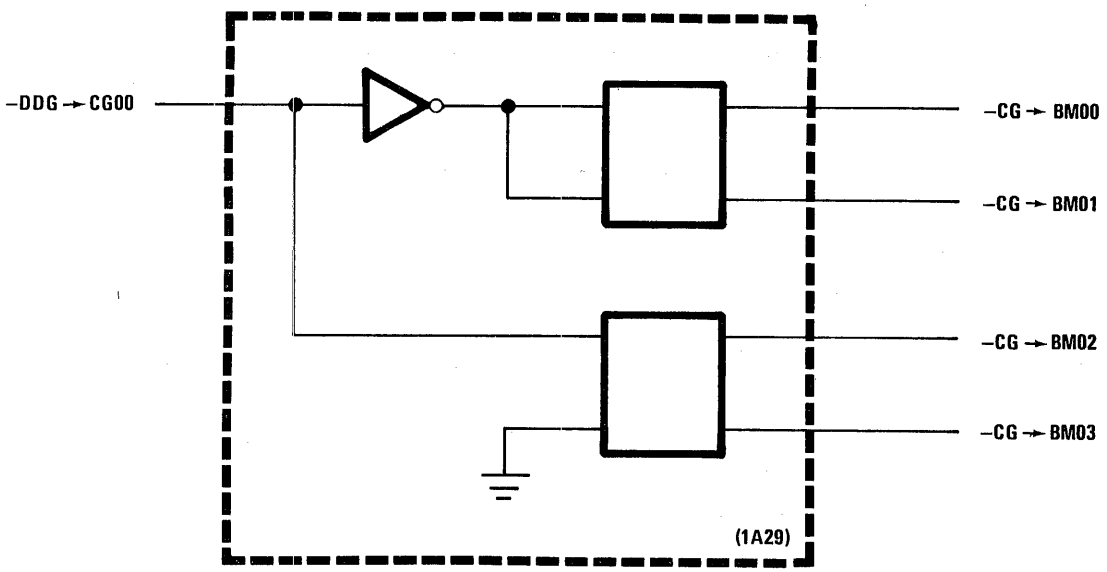


Figure 2-180. Generation of Constants for CORC μI

Table 2-26. Constant Generator Outputs to CORC μI

Carry Status	$\overline{\text{DDG-CG00}}$ Value	CG Outputs 001	002	003	004	Input to Bμ
Carry	0	1	1	0	0	0011 (3_{16})
No Carry	1	0	0	1	0	1101 (D_{16})

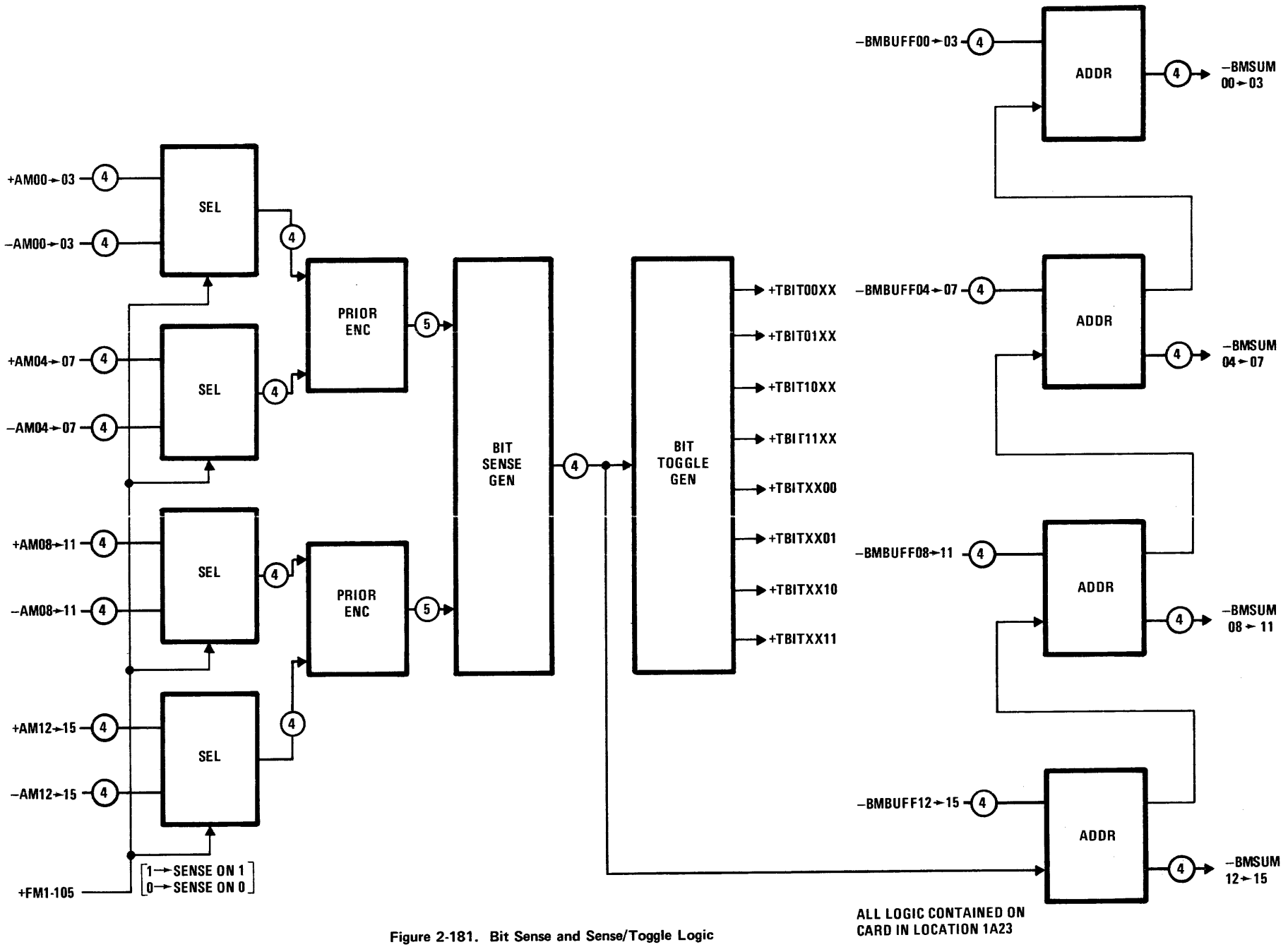


Figure 2-181. Bit Sense and Sense/Toggle Logic

from Aμ through a selector prior to being fed to the priority encoder. This selector gates the contents of Aμ in either true or complement form depending on the state of FM1-105 of the μI sub-operation code. If the bit is "0", the contents of Aμ are gated in true form so that the first "0" detected by the priority encoder is the first "0" of the Aμ. If the bit is "1", the contents of Aμ are gated in complement form so that the first "0" detected by the priority encoder is really the first "1" of Aμ. The priority encoder logic consists of two elements, each element determining priority of eight inputs (bit positions 00 through 07 and 08 through 15). The partial results are fed to the bit sense generator, which combines the outputs from the two priority encoder elements to generate a four-bit code representing the bit position where the first "0" or "1" was detected. This bit position number is fed to an adder which adds the number to the contents of Bμ. In reality, this addition simulates incrementing Bμ for each position of Aμ scanned without a find. Assume that a "1" is to be detected in bit position 4 of Aμ. This scan/increment sequence would require incrementing Bμ four times, since four bit positions would be scanned without a find (bit positions 0 through 3). By generating the bit position at which the "1" is found, only one addition need be performed and the result is the same (4 = bit position no. = no. of increments without find: 0→3).

If executing a bit sense/toggle μI, the bit sensed must be toggled as well as its position in Aμ being added to Bμ. This is accomplished by the bit toggle logic, which generates a two-bit toggle bit-position code from the four-bit sense bit-position code. For example, if bit 4 is to be toggled, the two-bit code consists of TBIT01XX and TBITXX00. This two-bit code is routed to an AND gate, as shown in Figure 2-182, to generate one signal that specifically designates bit 4 as the bit to be toggled. The result is routed to selector logic. This logic selects either the toggle bit designator or a corresponding bit from the shift network, depending on the state of FM2-211. Since a bit sense/toggle μI is being executed, FM2-211 is high to gate the toggle bit designator to the Aμ and Bμ fan-in logic. This logic is enabled, in turn, by ENSN-ALU which is high for both shift and bit sense/toggle μI's. The result is fed to both the J and K inputs of the bit 4 flip-flop of the Aμ register. Since both inputs are fed with the same signal, the flip-flop will toggle from its present state to the alternate state upon occurrence of ENCAM, which is also generated during execution of a bit sense/toggle μI.

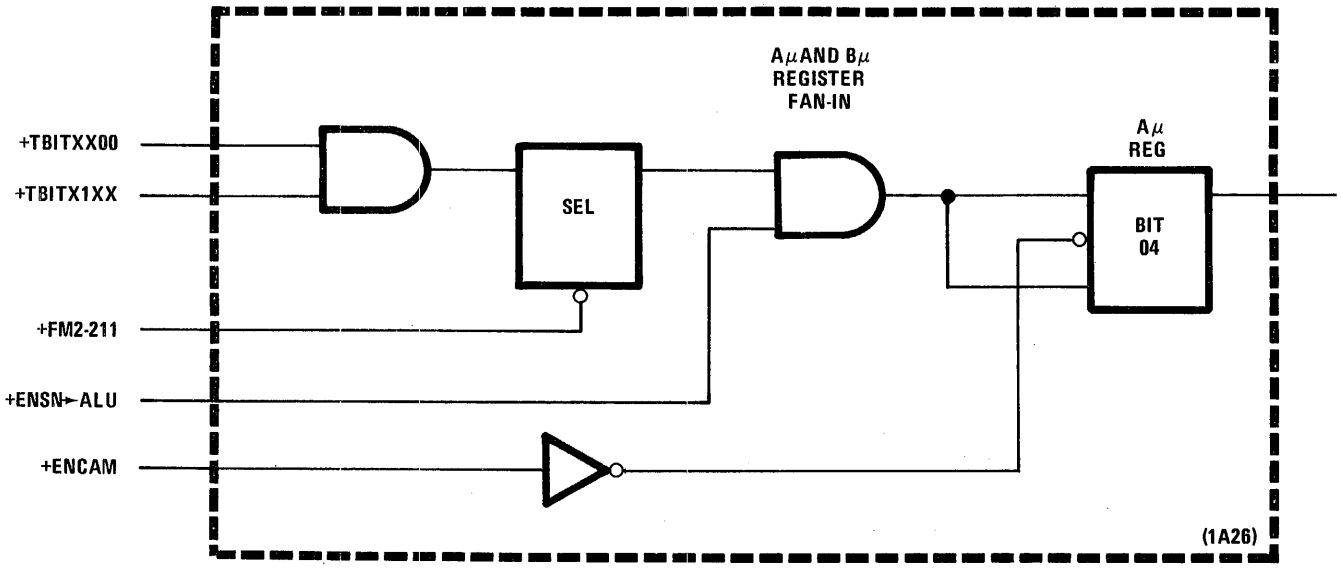


Figure 2-182. Toggling of Bit 4 of Aμ

SHIFT NETWORK

The shift network performs a left shift on each bit of a 32-bit operand as determined by a specified shift count. A block diagram showing the logic required to implement the shift operation is shown in Figure 2-183. The 32-bit operand in the Aμ and Bμ *buffer* registers is entered into the shift network and shifted by an amount defined by the shift count control output. This shift count is derived from one of three sources:

1. the X-field of a μI (SHF and SHR μI's),
2. bit positions 12 through 15 of the D register contents (DLS and DRS μI's if bit 1 of F is a "0"), or
3. bit positions 8 through 11 of the MLI in F (DLS and DRS μI's if bit 1 of F is a "1").

If either a SHF or DLS μI is executed, the shift count will be derived in true form and the shift will be to the left. If either a SHR or DRS μI is executed, the shift count will be derived in two's complement form. The result is to effect a right shift by an amount equal to the shift count in true form by left shifting the number of bits specified by the shift count in two's complement form. Indication that a right shift is to be performed is furnished by signal SHIFTR. This signal enables the two's complement logic to convert the shift count from true form to two's complement form. After being shifted, the 32-bit operand is routed back to the Aμ and Bμ registers through the shift network and bit sense fan-in logic, and the Aμ and Bμ fan-in logic.

The shift network itself consists of two ranks of 32 selectors each. The first rank is fed with the combined 32-bit output from the Aμ buffer and Bμ buffer registers. It is organized such that each bit of Aμ buffer and Bμ buffer is fed to four selectors to enable each bit to be left-shifted 0 to 3 places, depending on which of the four selectors is enabled. These enables are derived from the upper four bits of the eight-bit shift count from the shift count control. The output from each first-rank selector circuit is fed to four selector circuits of the second rank. These selector circuits perform a similar shifting function as the first-rank selectors, but on a nybl basis. Therefore, the bit shifted 0 to 3 places by the first-rank selectors is shifted 0, 4, 8, or 12 places by the second-rank selectors. These second-rank selectors are enabled by the lower four bits of the eight-bit shift count.

Shifting of a typical bit of Bμ buffer (bit 11) is shown in Figure 2-184. The four first-rank selectors fed by bit 11 are physically packaged in two selector elements: selectors 0A and 1A in one element and selectors 2A and 3A in the other element. Each selector is fed with other bits of Bμ buffer to effect a shift on them also; however, only bit 11 is fed to all four of the selectors shown. These four selectors shift bit 11 to the left 0 to 3 places as indicated by the SHF 0 through SHF 3 designations. These four outputs are each fed to four second-rank selectors to effect the shift on a nybl basis. Therefore, the SHF 0 output can be shifted 0, 4, 8, or 12 places to the left by selectors 0B, 4B, 8B, or 12B. Similarly, the SHF 1 output, which already represents bit 11 shifted left one place, can be shifted 1, 5, 9, or 13 places to the left of its original position in Bμ buffer by selectors 1B, 5B, 9B, and 13B. Therefore, bit 11 of Bμ (and all other bits of Aμ buffer and Bμ buffer) can be left-shifted 0 to 16 places by the combined action of the first-rank and second-rank shift selectors.

Examples of three different shifts of bit 11 of the B buffer are shown in Figure 2-185. Part a shows the selectors involved for a shift of 0 places to the left. The bit is simply gated straight through selectors 0A and 0B and appears at the output of the shift network as bit 27 of the 32-bit result (11 + 16). Part b shows the selectors involved in shifting bit 11 left 5 places. Selector 1A shifts the bit 1 place and selector 5B shifts the bit the remaining 4 places. Part c shows the bit shifted left 14 places: selector 2A shifts it 2 places and selector 14B shifts it 12 places.

As discussed previously, right shifts are implemented by converting the shift count to two's complement form and performing a left shift. The result, however, is developed in the lower half of the shift network (bit positions 16 through 31) instead of in the upper half as during a left shift operation. This difference in where the result is developed is interpreted as either a left shift or a right shift, as shown in Figure 2-186. This figure shows the principal μI's used to implement both a left shift and a right shift machine language instruction (MLI). Both examples assume that an operand in some register R will be shifted either left or right four places and stored back into R. Part *a* shows the μI's necessary to execute a left-shift MLI. The operand is transferred from R to B by a L.BW μI. From Bμ, the operand is left-shifted four places through the shift network by a DLS μI. During the shift, the four most significant bits are shifted end-off and the empty space resulting from shifting the four least

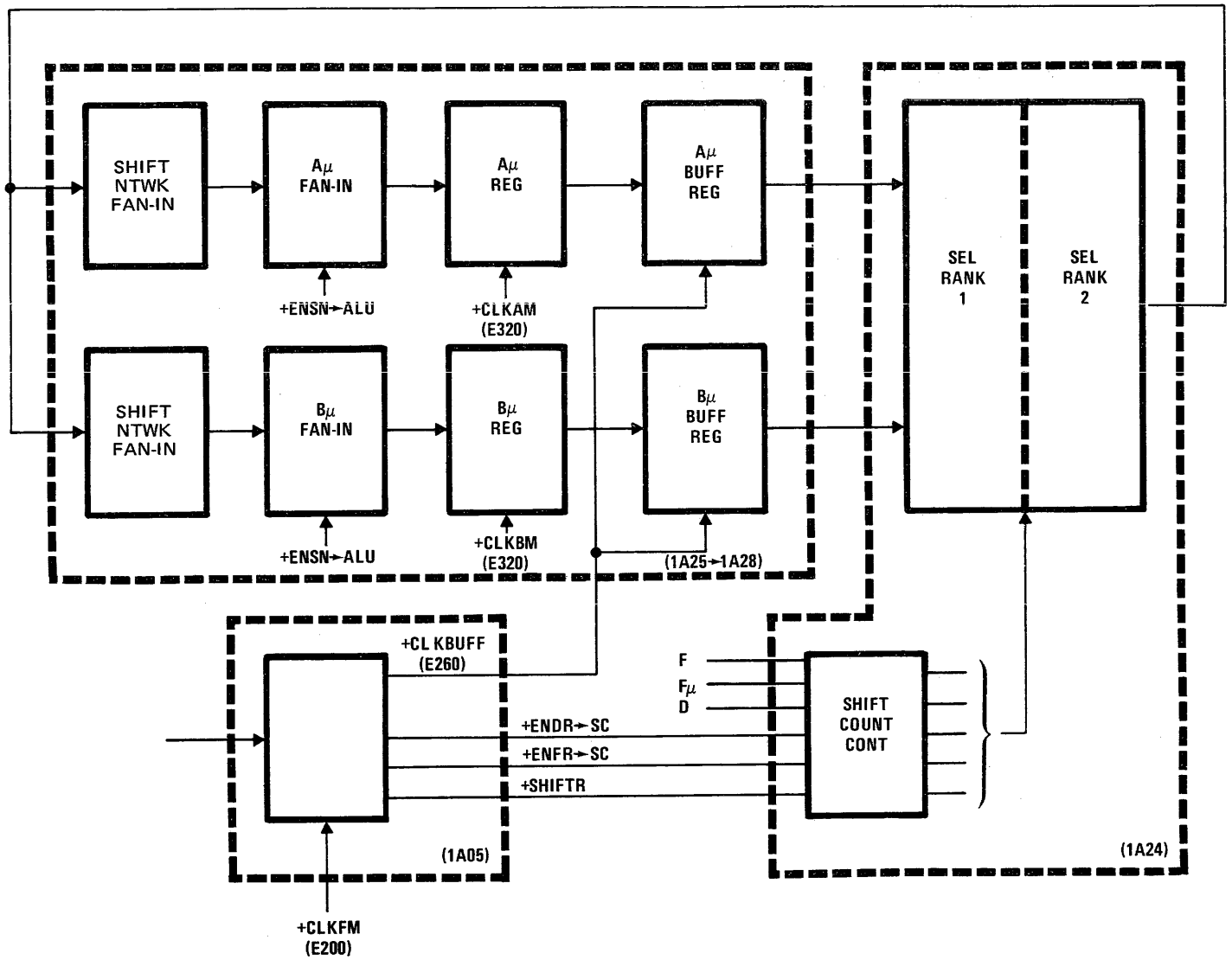


Figure 2-183. Shift Operation Block Diagram

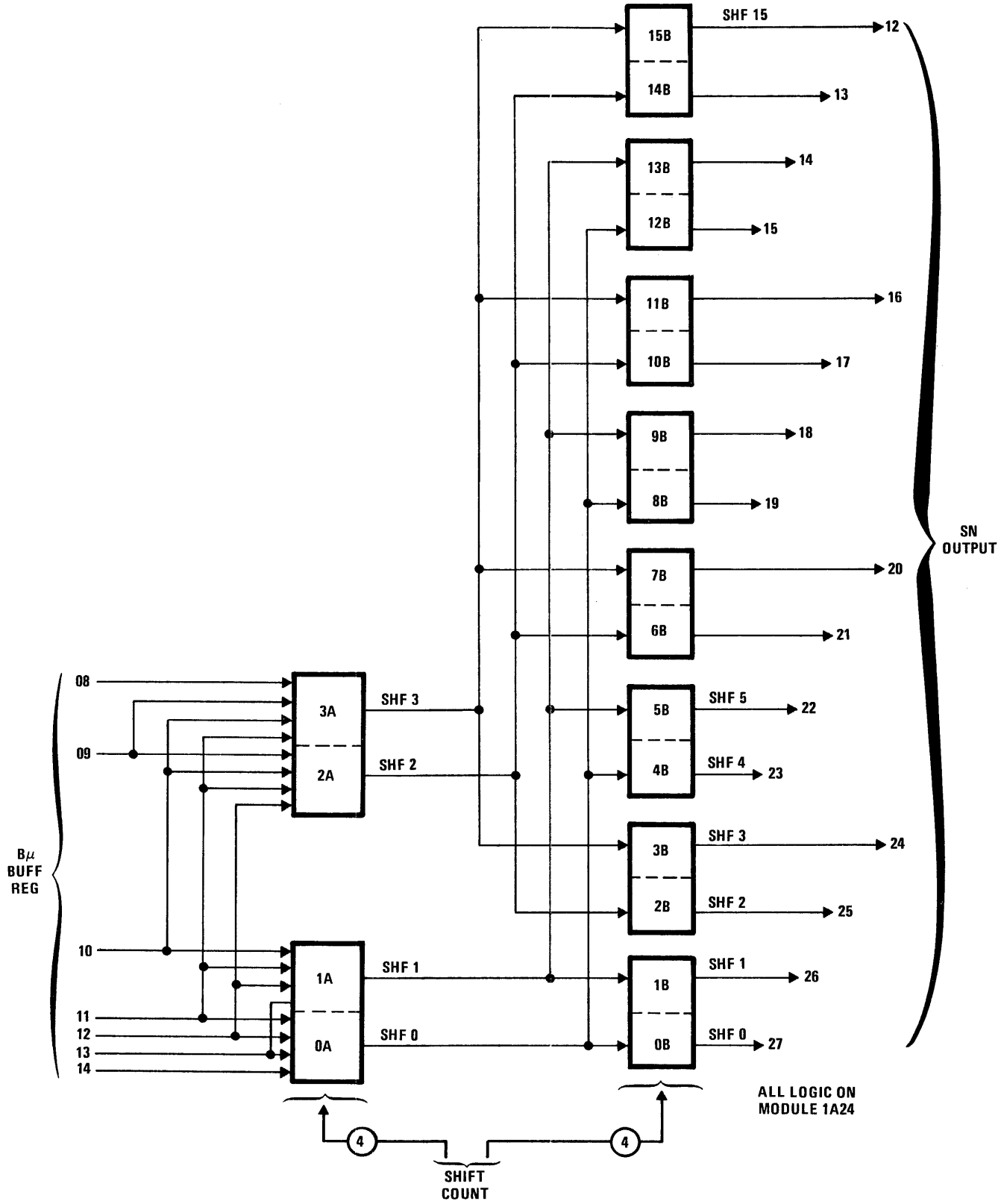
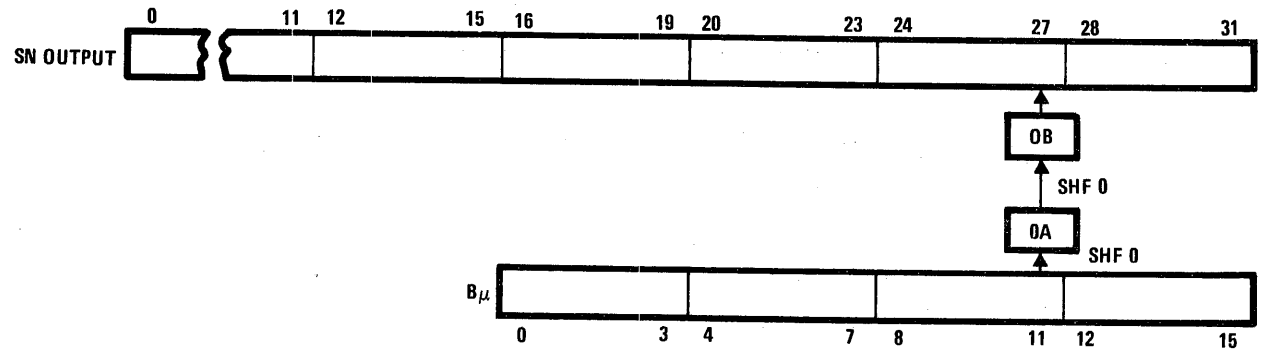
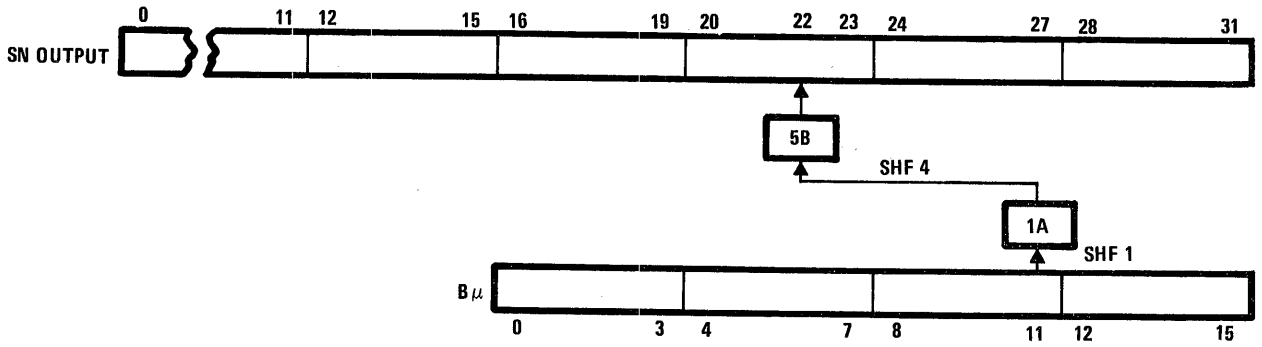


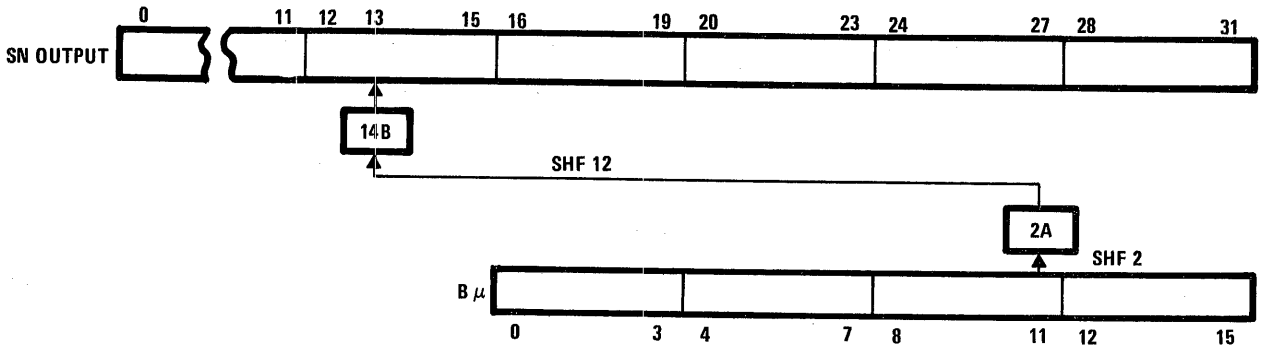
Figure 2-184. Shift Network



a. SHIFT LEFT 0 PLACES

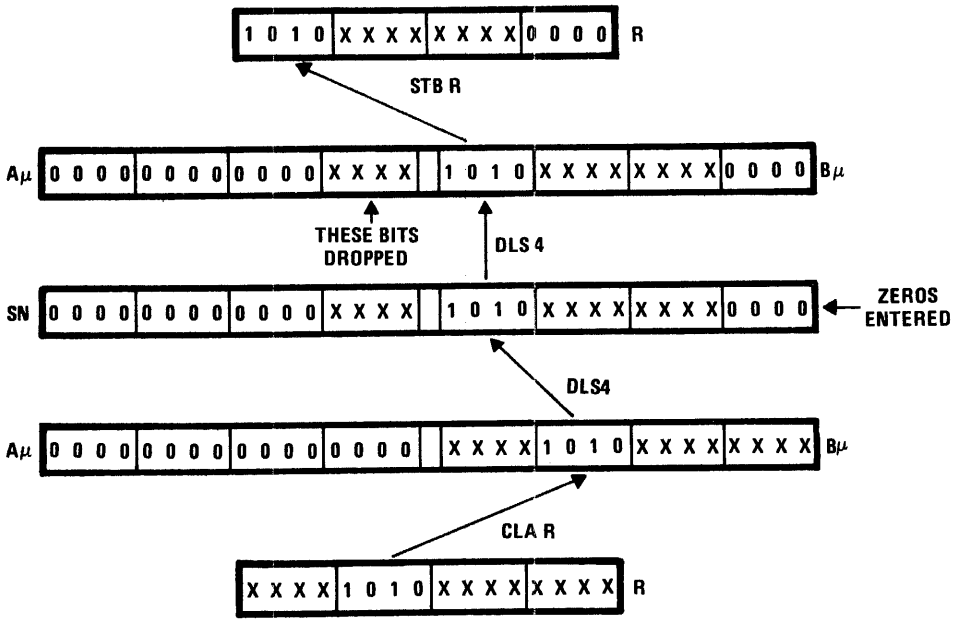


b. SHIFT LEFT 5 PLACES

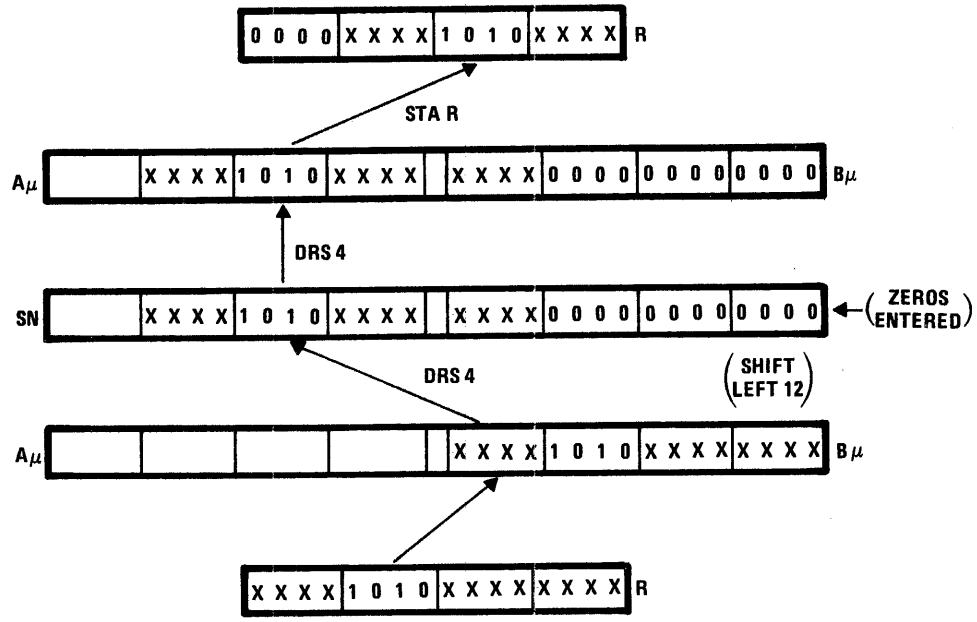


c. SHIFT LEFT 14 PLACES

Figure 2-185. Left Shift of Bμ Bit 5 Four Places



A. LEFT SHIFT MLI (4 PLACES)



B. RIGHT SHIFT MLI (4 PLACES)

Figure 2-186. Implementing Left Shift and Right Shift Operations

significant bits is filled with zeros. The result is sent to Bμ, and stored back in R by a STB μI. Part *b* shows execution of a right-shift MLI. The shift count (4) is converted to two's complement form to form C_{16} ($F_{16} - 4_{16} = C_{16}$) or 12_{10}. The operand again is transferred to Bμ and shifted left 12 places by a DRS μI. This time the shifted operand is developed in the upper half of the shift network and transferred to Aμ. The result is stored back into R by a STA μI.

Because of the propagation delay through the shift network (almost 100 nanoseconds total) and the length of the data path back to Aμ and Bμ, the shift μI's take two minor cycles to execute. It is necessary, therefore, to delay the following functions for one minor cycle:

1. clocking the shifted operand back into Aμ and Bμ,

2. execution of the first μI following the shift by blocking its transfer to Fμ (this μI has already been read from CS), and

3. delay reading the second μI following the shift μI from CS by blocking the transfer of its address into Sμ.

These inhibit conditions are generated by translation of the shift μI operation code and setting the Shift Delay flip-flop, as shown in Figure 2-187. Signal BLOCKFM is generated and $\overline{\text{ENCLKSM}}$ is disabled to block clocking of Fμ and Sμ. These conditions are generated during the first minor cycle of the shift μI by translation of the shift μI (E, X, X) and the fact that the Shift Delay flip-flop is not set. At the beginning of the second minor cycle of the shift μI, the flip-flop sets to enable clocking the shifted data back into Aμ and Bμ by generating ENCAM and ENCBM.

Timing for the above conditions is shown in Figure 2-188. This chart assumes that the shift μI is read from CS during E1 and executed during E2 and E3. The two μI's following the shift (SHIFT + 1 and SHIFT + 2) are delayed in their execution because of the constraints just discussed. Note that the shifted contents of the shift network cannot be clocked into Aμ and Bμ at E320 (which they would normally be if the μI was a one-minor-cycle μI) because enables ENCAM and ENCBM are inhibited at E280 due to the fact that the Shift Delay flip-flop has not yet set. When this flip-flop sets at E300, then ENCAM and ENCBM can be generated at E380 to clock Aμ and Bμ at E420.

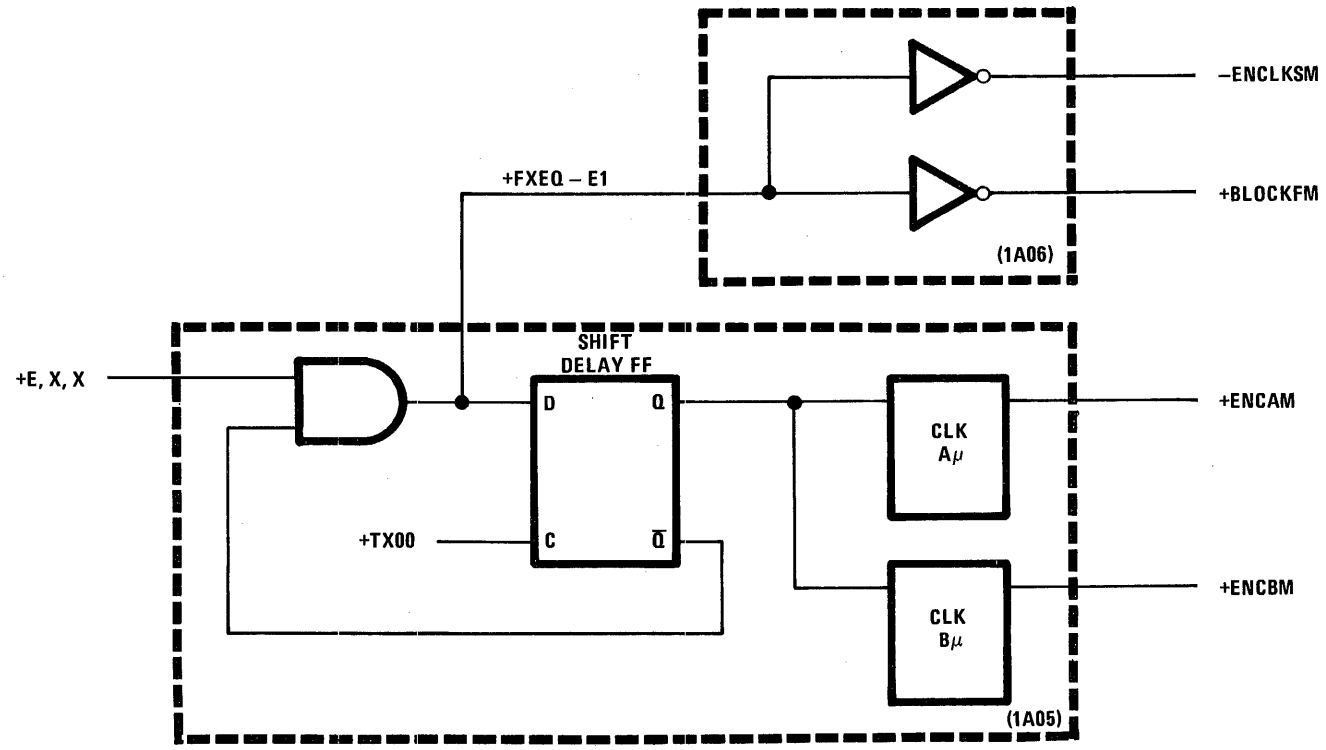


Figure 2-187. Shift Delay Logic

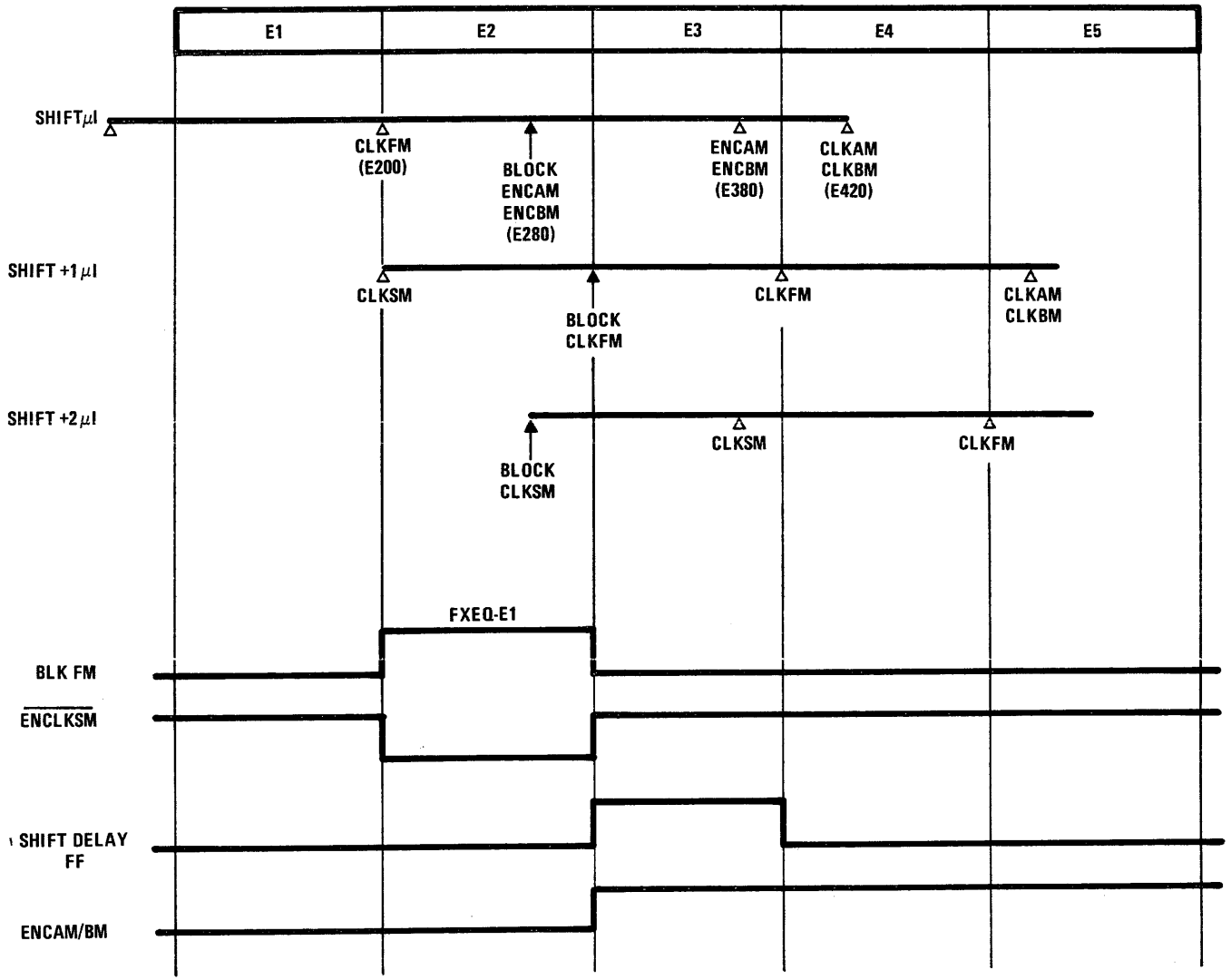


Figure 2-188. Shift μI Timing

I/O INTERFACE

The I/O interface logic controls the transfer of all data and associated control signals between the shared resources and I/O processors 0 through 3 during normal I/O transfer operations.* These operations involve transferring data between MS or registers in shared resources and designated register in each I/O processor. Each I/O processor contains registers dedicated to its exclusive use; for addressing purposes, however, they are collectively referred to as Group III registers of the Extended Register File (ERF). All I/O operations are initially started by the Executive processor under program control. However, each word transfer comprising the I/O operation is initiated by a request signal from the I/O processor. This request is sent to shared resources after determining that it either has assembled a complete word in its register for transfer to shared resources (if an input operation), or that the last word it received from shared resources has been written into its storage medium (disc, magnetic tape, and so forth) and it is now ready for another word (if an output operation). One word can be transferred during each time slice assigned to an I/O processor.

NOTE

Input and output operations are discussed from point of view of the shared resources. Thus, input operations transfer data from an I/O processor to the shared resources and output operations transfer data from the shared resources to the I/O processor.

I/O REQUESTS

All data transfers between the shared resources and an I/O processor are initiated by requests from the I/O processor. These requests are implemented as part of an I/O transfer routine executed by the respective I/O processor (such as the DIO packet executed by disc processor 3) and are generated when the word to be transferred has been assembled in the appropriate register. In the case of an input data transfer (I/O processor to shared resources), the request is generated when the word has been read from the I/O device (card reader, disc, and so forth) and assembled in the transfer register of the I/O processor. In the case of an output data transfer (shared resources to I/O processor), the request is generated when the I/O processor has written the last word received onto the disc, magnetic tape, or whatever, and is now ready to receive the next word from shared resources. Prior to the actual transfer of data, the I/O processor has been initialized by software routines executed both by the processor needing the I/O data and the Executive processor. Among other duties, these routines perform the following activities:

1. set the I/O processor's Active flip-flop in the Busy/Active (B/A) register, and

2. set up a buffer in MS where the I/O data will be stored by defining a Current Byte Address (CBA) and a Final Byte Address (FBA).

Upon determining that it is ready for a word transfer, the I/O processor sends a request to shared resources for a time slice during which to perform the transfer. This is done by setting the I/O processor's Busy flip-flop in the B/A register, whose output is then routed to the priority logic for assignment of a time slice. After each word is transferred, a comparison is made of the updated CBA and the FBA by a CIO μI for a condition of equality. If the two addresses are not equal, the μI *shuts off* the I/O processor by clearing its Busy flip-flop until the processor is ready for the next word to be transferred. The I/O processor's Busy flip-flop is again set by another request and the above sequence is repeated until all words have been transferred. At this point, the CBA will equal the FBA. The result is to prevent the processor's Busy flip-flop from being cleared to execute the next sequential μI's in the I/O transfer routine until it is completed. During this I/O operation, the request from the I/O processor has been generated and deactivated many times (once for each I/O word), causing the processor to turn on and turn off in a corresponding manner. Throughout this operation, however, the I/O processor is *locked on* to the shared resources by means of the processor's Active flip-flop which remains set for the entire operation. This *locking up* of the processor is necessary to prevent either the Executive processor or another I/O device associated with the I/O processor from attempting to initiate another I/O operation while the processor's Busy flip-flop is cleared between word transfers, which could normally be done if the Active flip-flop was not set.

A simplified diagram of the B/A register showing the relationship between the Active flip-flops of each I/O processor and corresponding Busy flip-flops is shown in Figure 2-189. The Active flip-flop for a particular processor is set by software under control of the aforementioned I/O transfer initialization routines. The Active flip-flop outputs (B/A-08 through B/A-11) are fed back to the Executive processor informing it that the I/O processor is presently engaged in an I/O operation and may not be interrupted until completed. The request (REQ) signals from each I/O processor are fed to the Busy flip-flops when a word is ready for transfer. These flip-flop outputs (B/A-00 through B/A-03) are routed to

*The CS Load and MS Load operations from I/O processors 1 and 3 are not considered normal I/O transfers in that they are usually performed only once, during a power-on condition.

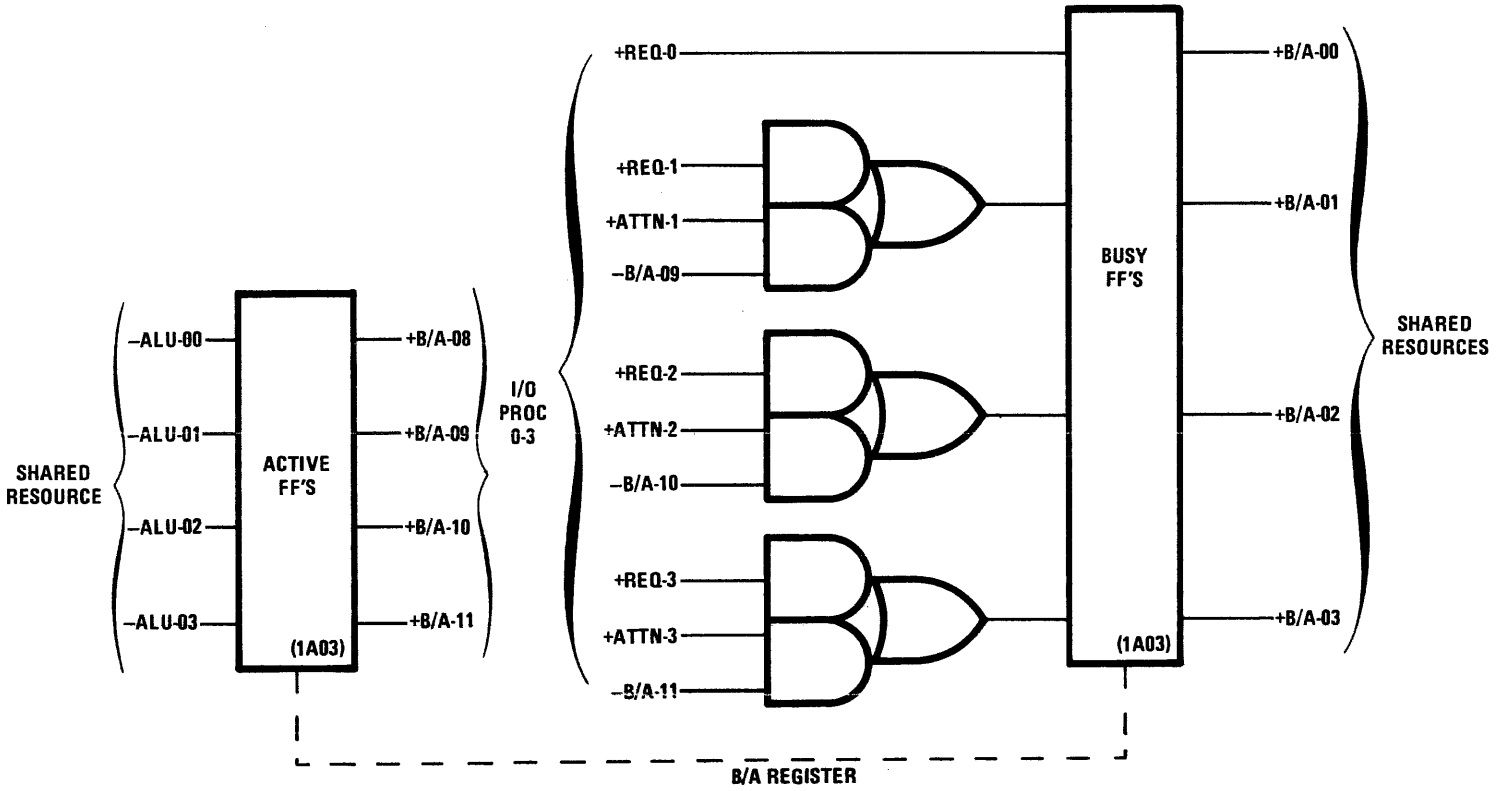


Figure 2-189. I/O Requests for B/A Register

the priority logic for assignment of time slices. Busy flip-flops for processors 1, 2, and 3 may also be set via corresponding ATTN signals in the event that an I/O device requests a time slice for a data transfer that is not initiated under software control. Setting the Busy flip-flop in this way will be permitted by the B/A register and then turn control over to appropriate software, but only if the processor associated with the I/O device is not presently engaged in another task as indicated by the corresponding Active flip-flop being set. This lock-out condition is implemented by ANDing the ATTN signal with the complemented output from the Active flip-flop.

Additional details about setting the Busy flip-flops by request signals may be found in the paragraph titled B/A Register.

REGISTER SELECTION

Selection of a register in one of the four I/O processors is accomplished by means of the Extended Register File (ERF) Group III selection logic discussed in the paragraph titled Extended Register Selection. As discussed in that paragraph, all registers of these I/O processors are selected by an encoded register address sent to the processor on the four ERNG3 lines. This address is developed both for read and write operations. This paragraph will discuss some of the peculiarities in register selection in each of the four processors.

Processor 0

Processor 0 contains four registers that may be addressed by the ERNG3 lines: write data 12, read data 13, write address 1D, and read address 1F registers. As the names imply, these registers are used to read or write associated data or address information when selected by the corresponding address. (Recall that the four ERNG3 lines encode only the least significant digit of the address (2, 3, D, or F). The most significant digit (1) is implied by the μI function code as a consequence of generating a register read or register write signal to enable information from or to the register.) Further details about addressing these registers may be found in Section 5 (Volume 3) of this manual.

Processors 1 and 2

Processors 1 and 2 contain five registers that may be addressed by the ERNG3 lines: Bus In register, Tag Out register, Channel Control register, Byte Count register, and Bus Out register. Three of these five registers (Tag Out, Channel Control, and Byte Count registers) may be addressed for either an input or output data transfer. The Bus In register may be addressed only for an input operation; conversely, the Bus Out register may be addressed only for an output operation. When addressing the eight-bit Tag Out register for an input data transfer, the data on the eight tag in lines is also selected for transfer to the shared resources. Further details about addressing these registers may be found in Section 4 (Volume 3) of this manual.

Processor 3

Processor 3 contains one register that is addressed by the ERNG3 lines. Depending on the address used to select this register and the time at which the address is generated on the ERNG3 lines, this register may be used to perform a variety of functions. For example, addressing the register as register 10 indicates to the IFA that the information contained in the register is to be used to select a particular disc drive. When addressed as register 12, the contents are interpreted as status select bits. Further details about addressing this register may be found in Section 6 (Volume 3) of this manual.

DATA INPUT

Data read from the four I/O processors is fed to the shared resources by means of the ERF Group III input logic shown in Figure 2-190. This logic consists of 16 data receivers which receive the 16 data lines from each of the four I/O processors. Data from a particular processor is selected by a corresponding ENERG3 enable, which gates data from the particular processor through the data receivers, and the ERFG3RD signal, which strobes data from the register in the selected I/O processor. Each ENERG3 enable is generated by a corresponding STATE signal from the priority logic which defines the processor from which data is to be read, and *a* and *b* designator values of both "1" which define an ERF register is being selected. Signal ERFG3RD is generated for any μI that can read an ERF Group III register (6, 1; 6, 2; 7, X, and D, X μI's) ANDed with a second line that indicates the particular μI is making a reference to an ERF Group III register (a·b = 1·1 and Fμ -011 = 1).

DATA OUTPUT

Data to be written into registers of the I/O processors is fed from the shared resources by means of the ERF Group III output logic shown in Figure 2-191. This logic consists of 16 data drives which receive the I/O data from the ALU and fan it out to the four I/O processors. (Although data is fed out to all four processors in parallel, only one of the four will be enabled to receive it as selected by the EXCT signal from the priority logic.) The register to be written into is enabled by the ERFG3WR signal. This signal is generated in a manner similar to ERFG3RD, except that it is generated during execution of a μI that can write into an ERF Group III register.

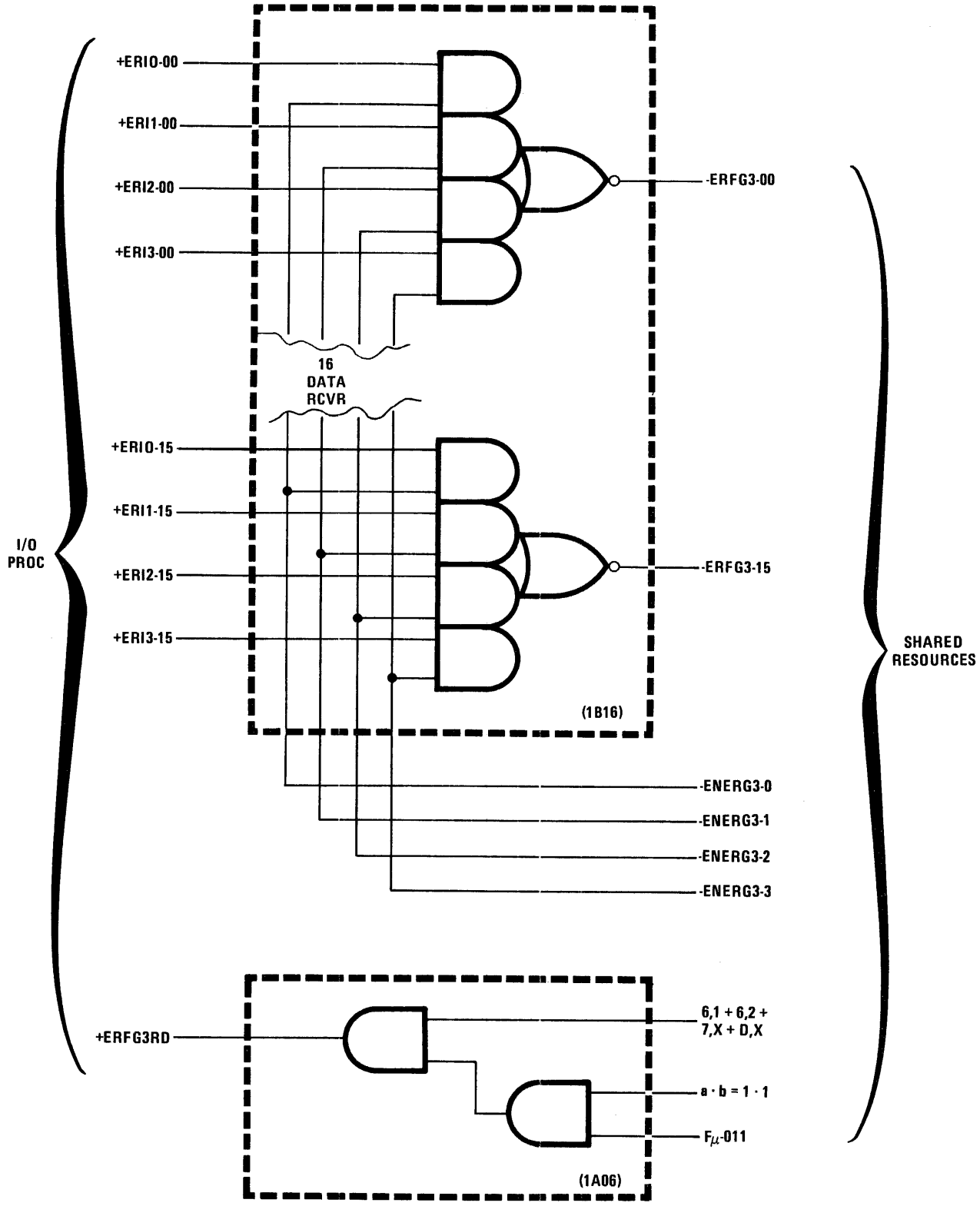


Figure 2-190. ERF Group III Input

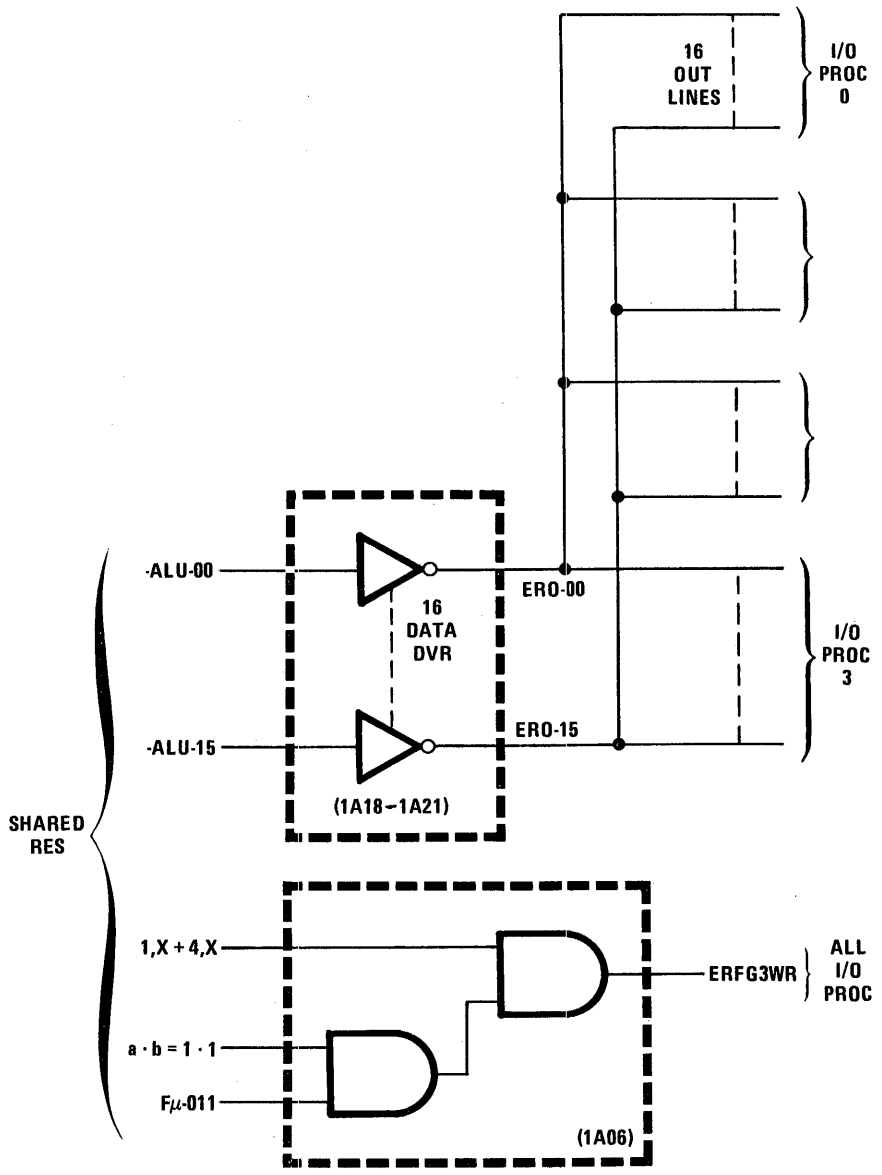


Figure 2-191. ERF Group III Output

TERMINATION OF I/O OPERATION

As discussed in the paragraph titled I/O Requests, an I/O operation is normally terminated when the CBA equals the FBA, indicating that the I/O buffer in MS has been completely filled. This condition is implemented by the CIO μI, which checks the contents of the Aμ and Bμ registers for a condition of either equality or inequality depending on which of the two CIO μI's is being executed. However, the I/O operation can also be terminated before normal completion because of an abnormal condition detected during the operation. This abnormal termination is indicated by an End of Transfer (EOT) signal sent from the processor to the shared resources. In the case of a normal I/O termination, the CIO condition keeps the Busy flip-flop set so that the I/O operation can terminate in an uninterrupted fashion by means of the I/O data transfer routine. In the case of an abnormal I/O termination, the EOT signal keeps the Busy flip-flop set to allow the Executive to detect the source of the abnormal condition. Each of the four I/O processors sends a respective $\overline{\text{EOT}}$ signal to fan-in logic in the shared resources, as shown in Figure 2-192. This logic AND's the $\overline{\text{EOT}}$ signal with a corresponding $\overline{\text{STATE}}$ signal from the resource allocation network to generate $\overline{\text{EOTEXIT}}$. This signal is routed to the I/O terminate/continue logic.

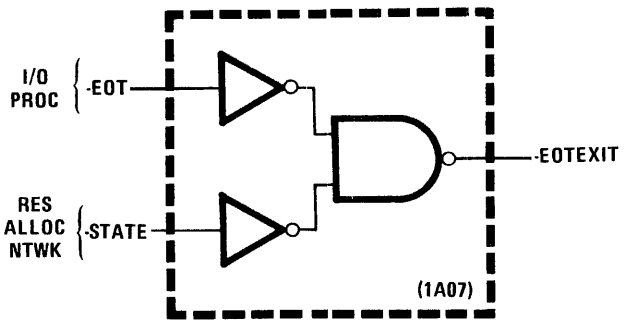


Figure 2-192. EOT Fan-In Logic

The I/O terminate/continue logic, shown in Figure 2-193, is used to either terminate or continue the I/O operation, based on receipt of the $\overline{\text{EOTEXIT}}$ signal and evaluation of the CBA/FBA compare during execution of a CIO μI. This logic generates three signals: IOEXIT, $\overline{\text{CIO-TX}}$, and $\overline{\text{CIOEXIT}}$. Signal IOEXIT is used to terminate the I/O operation in the I/O processors. The signal is generated for either a normal I/O (CIO μI) terminate condition, or an abnormal (EOT) terminate condition. For a CIO μI terminate, the signal is generated during execution of either a CI01 μI and an Aμ=Bμ condition, or a CI02 μI and an A$\mu\neq$Bμ condition. These are the conditions defined for these two μI's that indicate that the I/O operation has been completed. The result is to cause the I/O processor to idle (perform NOP's) through the rest of the time slice and update the Pμ register in the ERF with the contents of P_p, which defines the starting address of the first μI to be executed during the next time slice. This μI will be the first in a routine to obtain status information, which always follows after transfer of I/O words. For an EOT terminate, the signal is generated by means of $\overline{\text{EOTEXIT}}$ during execution of the following CIO μI in the I/O transfer routine regardless of whether or not the CIO compare condition is met. Combining $\overline{\text{EOTEXIT}}$ with signal CIO is necessary so that the I/O processor can begin the routine to obtain status information. (As discussed above, the starting μI address for this routine results from executing either the CI01 or CI02 μI.) For all three ways of generating IOEXIT, signal IDLE is included to prevent the signal from being generated if the I/O processor is in an idle condition.

Signals $\overline{\text{CIO-TX}}$ and $\overline{\text{CIOEXIT}}$ are generated for a condition opposite of that for generating IOEXIT, that is, if the condition for terminating an I/O operation is not met. Essentially, this means one of the following is true:

1. execution of a CI01 μI and an A$\mu\neq$Bμ condition,
2. execution of a CI02 μI and an Aμ=Bμ condition, or
3. no $\overline{\text{EOTEXIT}}$ signal.

Signal $\overline{\text{CIO-TX}}$ is fed to the B/A register to clear the I/O processor's Busy flip-flop. This action allows the I/O operation to continue by permitting the I/O processor to set the Busy flip-flop again when the next I/O word is ready for transfer. Signal $\overline{\text{CIOEXIT}}$ is routed to the Pμ select logic to inhibit writing the contents of P_p (starting μI address of routine for obtaining status) into the I/O processor's assigned Pμ register. Since additional I/O words are to be transferred, the $\overline{\text{CIOEXT}}$ signal effectively causes the I/O transfer routine to repeat by causing the un-updated contents of Pμ to be transferred back to Sμ at the beginning of the next assigned time slice. Inhibiting the Pp $\rightarrow$ Pμ operation is done by changing signal EFIRH/WL to the high state. When in the low state, this signal enables data to be stored in Pμ. When changed to the high state, however, this write operation is inhibited. If the I/O processor is running in the Consecutive Cycle (CC) mode, Sμ is inhibited from being written with the contents of P_p by inhibiting $\overline{\text{ENPP-SM}}$. This inhibiting condition is also generated by the $\overline{\text{CIOEXIT}}$ signal.

SYSTEM CONTROL PANEL INTERFACE

The System Control Panel (Panel) interface logic controls all Panel-initiated functions of the system. These functions include (1) reading and writing Main Storage (MS), Control Storage (CS), Register Option (RO) registers, and Register File (RF) registers; (2) selecting processor and panel operating modes; (3) initiating CS loads and MS loads; and (4) displaying file registers contents. The Panel interface also enables display of certain system status information such as MS and CS parity errors and processor states which are executing major cycles, the capability of transferring control of the system from the Panel to a remote location, a general system reset facility, and applying AC power to the system.

A block diagram showing the main functions controlled by the Panel interface logic is shown in Figure 2-194. Reading and writing MS, CS, RO, and RF registers are grouped under one category identified as *console control*. These operations are selected by means of the CONSOLE MODE SELECT selector. The MS, RO, and RF register *read* and *write* operations are similar in that each is performed by a μI subroutine. The *CS Read* and *CS Write* operations are also selected by the CONSOLE MODE SELECT selector. However, these operations are

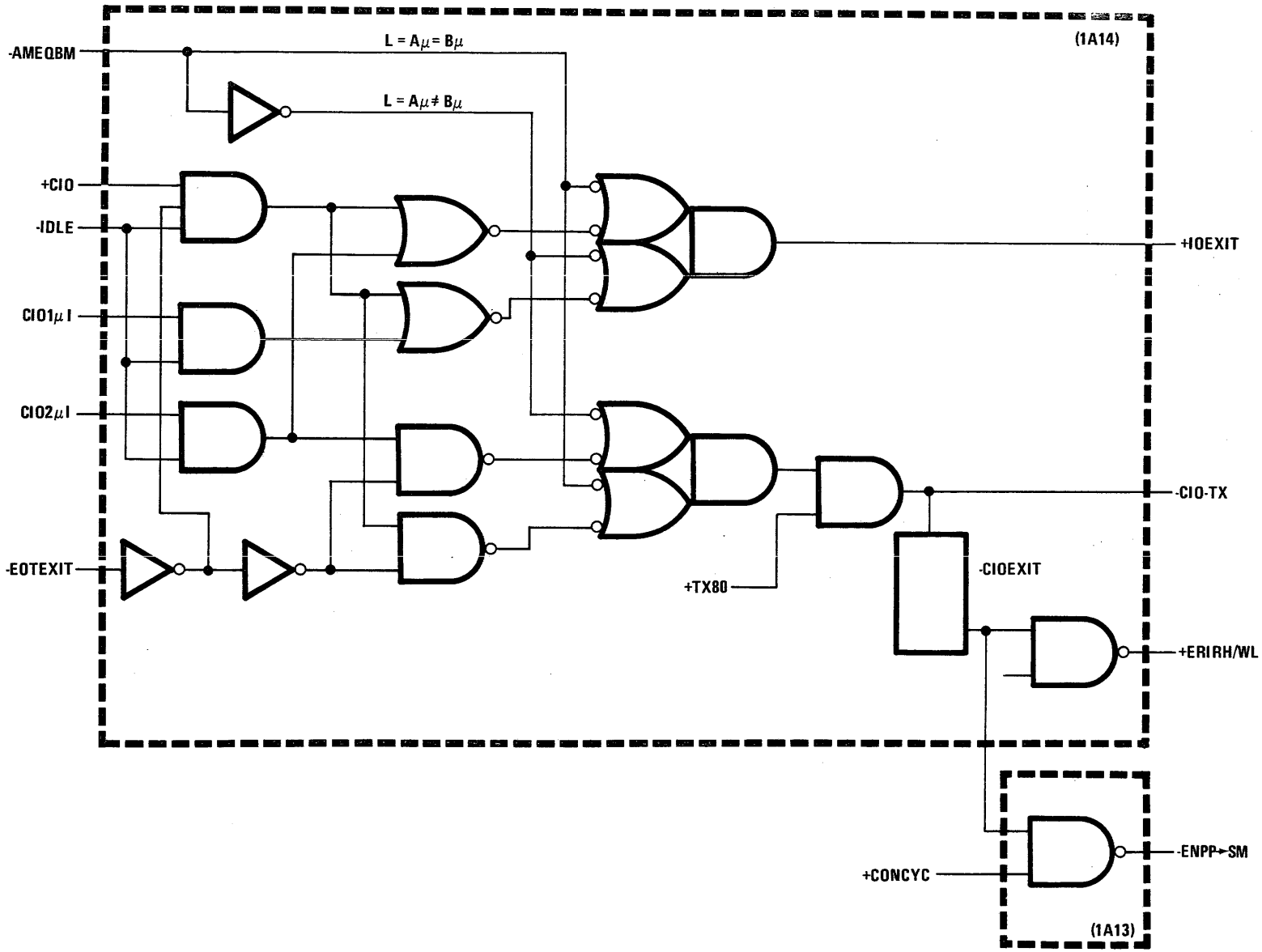


Figure 2-193. I/O Terminate/Continue Logic

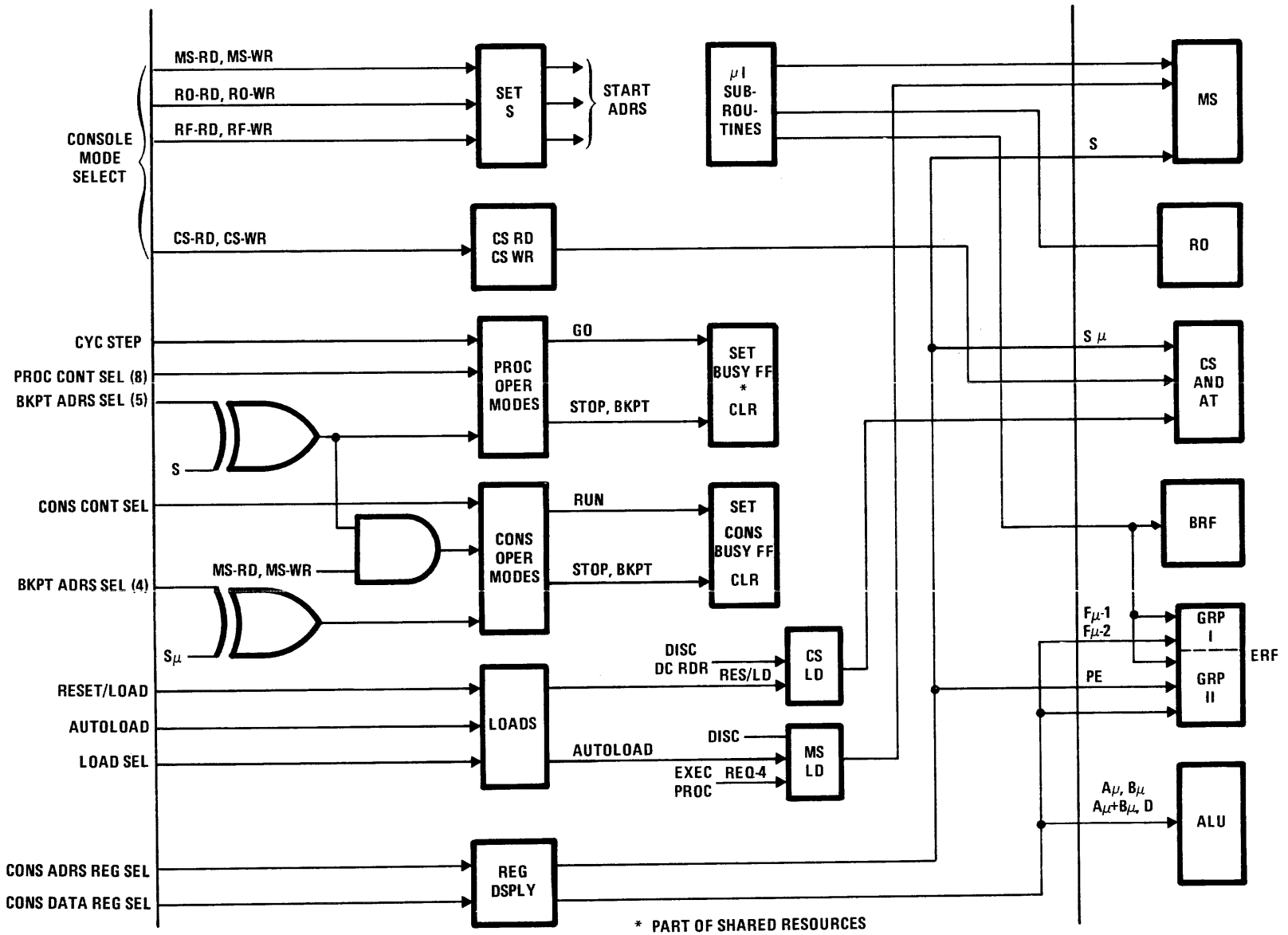


Figure 2-194. System Control Panel Interface Block Diagram

implemented by hardware only, due to the fact that a *CS Write* operation might alter the very subroutine in CS used to perform the *CS Write.* The *CS Read* operation is more accurately identified as the *CS Read/Scan* operation. These two operations are executed in a similar manner, the major difference being the operating mode in which they are performed. The *CS Scan* operation is performed in the normal mode (continuous operation) to verify that all data written in CS and the FRJ decode address table (AT) during a *Reset/Load* operation was stored without error. Data is read from CS in a continuous manner for purposes of making longitudinal parity checks on a page basis without regard to the contents of any particular location. The data is said to be *scanned* when performing this operation; hence, the term CS Scan. A *CS Read* operation, on the other hand, is performed in the stop/step mode so that particular locations in either CS or the AT may be read, one at a time. The *CS Scan* and *CS Read* operations are limited to off-line use only; that is, no other processor may be requesting slices when performing either of these operations. This means that the Panel will be granted alternate time slices by the resource allocation logic in which to perform the chosen operation. (The Panel does not have the facility to operate in the Consecutive Cycle mode.)

The Panel allows each of the eight processors or the Panel itself to function in one of three operating modes: *stop/step, normal* and *breakpoint*. Basically, the stop/step mode permits operation for only a short period of time (either one major cycle or one MLI), the normal mode permits operation for an indefinite period of time (usually until the processor or Panel is set to another mode), and the breakpoint mode permits operation only until some pre-determined address in either CS or MS is reached. The normal mode is the mode in which a processor would execute a program in an on-line situation. The stop/step and breakpoint modes are used primarily during maintenance operations. Selection of a processor mode is made by one of eight PROCESSOR CONTROL SELECT switches; Panel mode selection by the CONSOLE CONTROL SELECT switch. The processor stop/step mode is divided further into two sub-modes as determined by the position of the CYCLE STEP switch. This switch causes the selected processor to run for either one major cycle (when set to the up position) or for one MLI (when set to the down position). Similarly, the processor breakpoint mode can be run in one of three sub-modes, depending on selection of the three BREAKPOINT MODE SELECT switches: READ INSTR, READ DATA, and WRITE DATA. When the corresponding switch is set to the up position, a breakpoint stop will occur (1) immediately after the MLI at the breakpoint address is read (READ INSTR switch), (2) at the end of the storage reference cycle in which data was read at the breakpoint address (READ DATA switch) or (3) at the end of the storage reference cycle in which data was written at the breakpoint address (WRITE DATA switch).

Initial loading of CS and associated AT, and MS, is performed by the Panel RESET/LOAD and AUTOLOAD switches, respectively. The CS load can also be performed automatically during a power-up system reset condition. For whichever condition, the CS load is performed under hardware control using either disc or cards as the input medium. The choice of which medium will be used is determined by the LOAD SELECT switch. Either disc or cards may be used to load MS also. If the disc was used to load CS and will also be used to load MS, it will also load MS automatically upon completion of the CS load under control of an autoload routine. If loading from cards, the MS load operation must be started manually. The CS load operation is under control of the Panel. Conversely, the autoload operation is under control of Executive processor 4. This control enables the Executive processor to load data in pre-determined areas of MS as determined by processor number, bounds protect, and other related criteria.

Selected registers of the Extended Register File (ERF) Group II and the ALU may be selected for displaying their contents by means of hardware control as opposed to the software-controlled RF read and write operations. These registers are selected by the CONSOLE ADDRESS REGISTER SELECT and CONSOLE DATA REGISTER SELECT selectors, which select address-related and data-related registers respectively. These selectors permit display of register contents only; data may not be written into these registers by this method. In addition, only one processor may be running and then in the stop mode when addressing registers using these selectors. As such, selecting registers by this method is designed primarily for maintenance purposes when troubleshooting a single processor.

PANEL CONTROL

CS Scan/Read

The CS scan and CS read operation are performed by the same logic, but under different conditions. The CS scan operation is performed in the normal mode and verifies that all data written into CS during a CS load was entered without error. The CS read operation is performed in the stop/step mode to display the contents of individual locations in CS in a sequential manner. Logic for performing both CS scan and CS read operations is shown in Figure 2-195. As shown, both operations are executed in basically the same manner except for the duration of

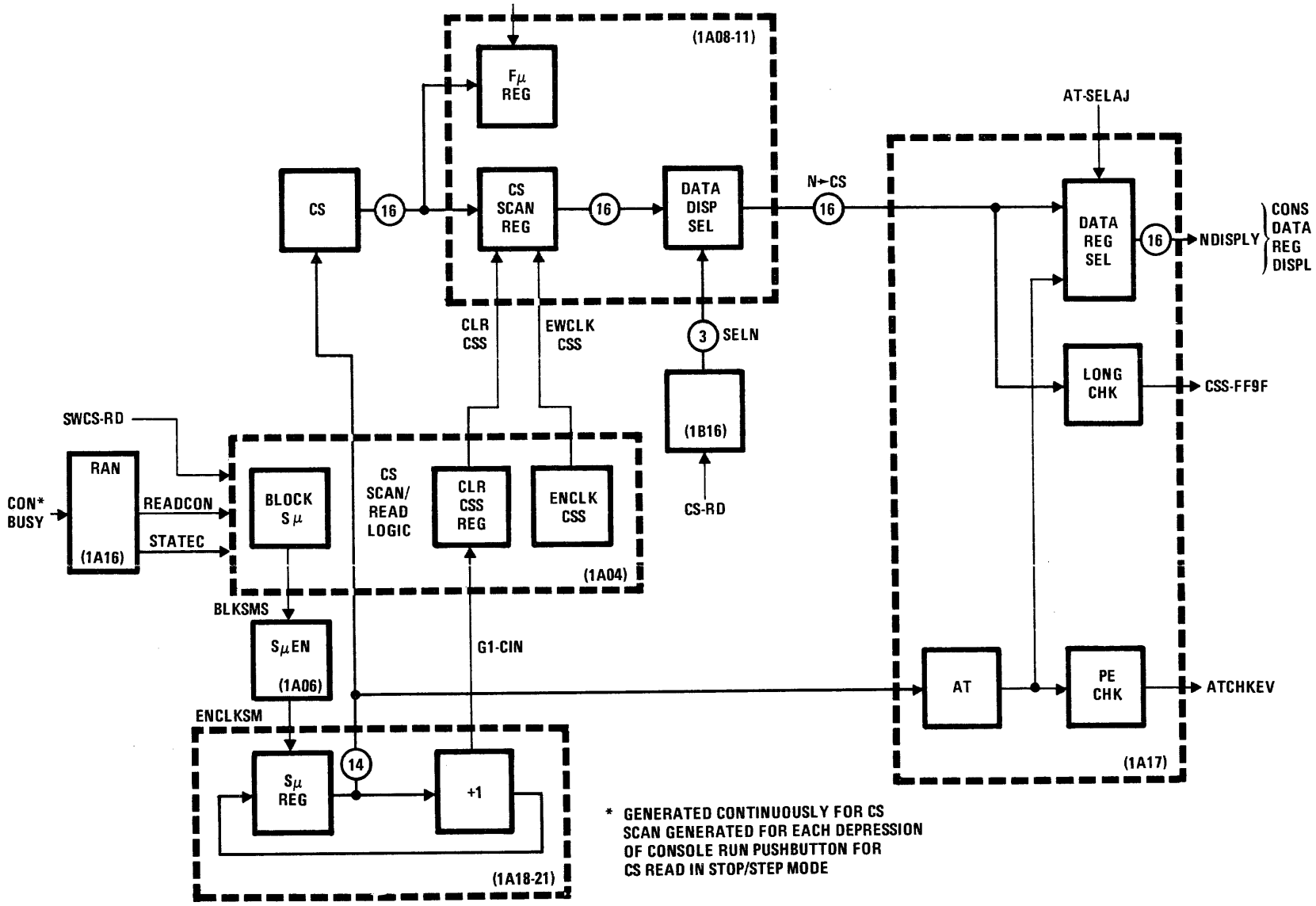


Figure 2-195. CS Scan and CS Read Operation

signal CONBUSY. This signal obtains time slices for the Panel during Panel-initiated operations by the shared resources. In effect, this signal is analogous to a Busy flip-flop output from the B/A register which indicates that a particular processor wants a time slice. During a CS scan operation, CONBUSY is generated continuously from the time that the CONSOLE RUN pushbutton is pressed until the last location in the AT is scanned. This enables the CS scan operation to proceed in a continuous manner by automatically granting time slices to the Panel. One CS location is scanned each time slice. During a CS read operation, however, CONBUSY is generated for only one time slice at a time upon pressing the CONSOLE RUN pushbutton. This happens as a result of the Panel being in the stop/step mode (CONSOLE CONTROL SELECT switch set to the STOP/STEP position). When set to this position, the stop/step signal clears out the Console Busy flip-flop that generates CONBUSY at the end of the Panel-assigned time slice. The result is that only one location in either CS or in the AT is read at a time. For whichever operation is being performed, CONBUSY is fed to the Resource Allocation Network (RAN) to generate READCON and STATEC. These two signals, along with SWCS-RD from the CS-RD position of the CONSOLE MODE SELECT selector, are routed to the CS scan/read logic.

The CS scan/read logic implements the CS scan operation by making a longitudinal parity check of all 256 words of each page in CS and a horizontal parity check of all 256 words in the AT. (A longitudinal parity check involves checking the same bit of all 256 words in a page in sequence as the words are scanned, as opposed to a horizontal parity check which checks all 16 bits of one word.) The logic scans each CS page in sequence in a continuous manner until a word is detected with erroneous bits. The remainder of the page containing the error is scanned, but the operation stops on the last address of the page. At this point, the CS Scan register will contain all 1's in every bit position except those in which the error occurred (and in bit positions 9 and 10, which are not used). These bits are displayed in the CONSOLE DATA REGISTER DISPLAY indicators on the Panel via the 16 NDISPLAY signals fed from the Data register display selector logic. The same sequence of events occurs during the scan of an unused page (essentially an unused page is interpreted by the CS Scan register as a page in which all words contain parity errors). Pressing the CONSOLE RUN pushbutton causes the CS scan operation to resume checking the rest of the pages in CS. If no errors were detected in any of the pages, the SC scan operation will continue until the last address of the last page on CS is reached. At this point, the operation will stop, and the Panel indicators will display the value FF9F, the hexadecimal equivalent of all 16 bits except 9 and 10 being 1-bits.

The ability of using a longitudinal parity check to check bit errors in the manner described above is accomplished by inserting a word called a checksum in each page of CS coded such that the CS Scan register output of the last word read from a page with no errors will equal all 1-bits. The word-by-word development of the longitudinal parity check is implemented by the toggling property of the 16 J-K flip-flops which make up the CS Scan register, wherein the output of each flip-flop will toggle (change state) whenever a J input of "1" is detected. This toggling property of the J-K flip-flops enables the word-by-word parity check to be developed for each page as shown in Figure 2-196. (Assume for purposes of discussing this figure that each word in CS is four bits in length and the page to be considered is page 0.) Initially the CS Scan register is cleared to zeros by means of CLRCSS. Then the first word in CS at address 0000_{16} (0110) is fed to the CS Scan register by the ENCLKCSS signal. This signal enables clocking the CS Scan register at E200 of every Panel time slice, as shown in Figure 2-197. Since the register was initially cleared, the output of the register is the same as the input. The contents of $S\mu$ are updated by the S +1 logic via ENCL KSM and the contents of address 0001_{16} (1011) are fed to the register. Since bits 0, 2, and 3 of this word are 1's, the register output corresponding to these three bit positions will toggle.

This same sequence of events repeats itself for all 256 words in the page (255 μl's plus the checksum). Note that the checksum (assumed to be stored at address $00FF_{16}$) is coded to generate a CS Scan register output of all 1's after it is scanned. In essence, each bit of the checksum enables checking parity of the corresponding bit in all 256 words of a page. At this point, the register output is compared with the contents of the $S\mu$ register. If the right-most eight bits of $S\mu$ (address of word within page) are all 1's (FF_{16}), the compare indicates that all words within the page were loaded without error.

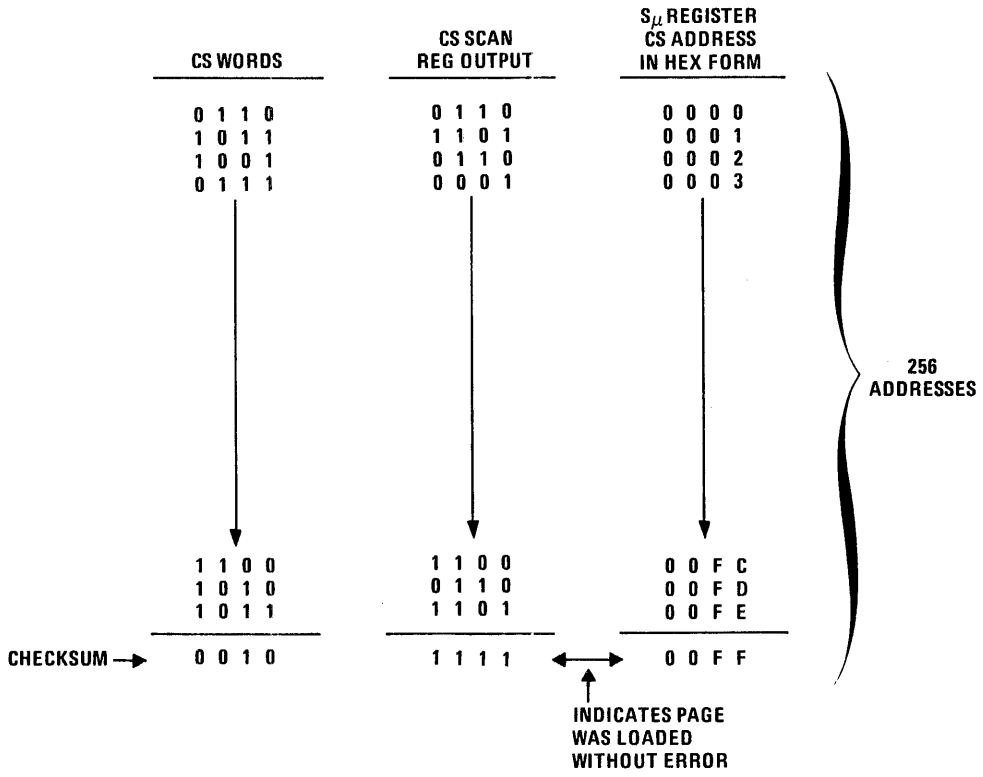


Figure 2-196. Parity Checking of CS Page

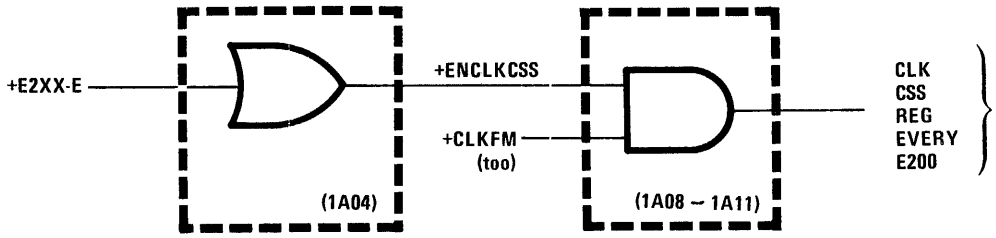


Figure 2-197. CLKCSS Logic

Upon completing a check of all CS pages, the CS scan logic begins to read the contents of each 256-word AT, in sequence, performing a horizontal parity check on each 9-bit address stored in the AT. (As discussed in the paragraph entitled FRJ Decode, the complete FRJ branch address is 14 bits in length. The upper 5 bits, however, are derived from other sources and therefore are not checked for parity.) Each 9-bit address is checked for odd parity. Upon detection of a parity error, signal ATCHKEV is generated (since an even number of 1's detected indicates an error) to generate a *scan error* signal.

As discussed previously, the CS Scan register is cleared initially prior to beginning the CS scan operation. However, it also cleared after scanning the last address of each page in CS in preparation for beginning a scan of the next page. Logic for generating the clear signal required at these two different times is shown in Figure 2-198. Initial generation of CLRCSS occurs between E560 and E000 of the time slice preceding that granted to the Panel, when READCON and $\overline{\text{STATEC}}$ are both high. Since the Panel is running in the continuous mode and no other processors may be running during a CS scan operation, these two signals remain at their final value (READCON high and $\overline{\text{STATEC}}$ low) until the end of the operation. This assures that the clear signal will not be generated during the scanning of a page to erroneously clear the longitudinal parity check being developed. Upon completing a page scan, the lower eight bits of Sμ contain all 1's to indicate that 256 (or a multiple of 256) addresses have been developed. This condition is indicated by signal G1-CIN from the Sμ+1 logic. Signal G1-CIN generates CLRCSS to begin the scan of the next sequential page in CS. During a CS read, the clear signal will be generated prior to every time slice granted the Panel. This enables the particular contents of a location in either CS or the AT to be entered in the CS Scan register for display on the Panel. These individual clear signals are generated as a result of the Panel running in the stop/step mode, where READCON and $\overline{\text{STATEC}}$ will be re-initiated every time the Console Busy flip-flop is set.

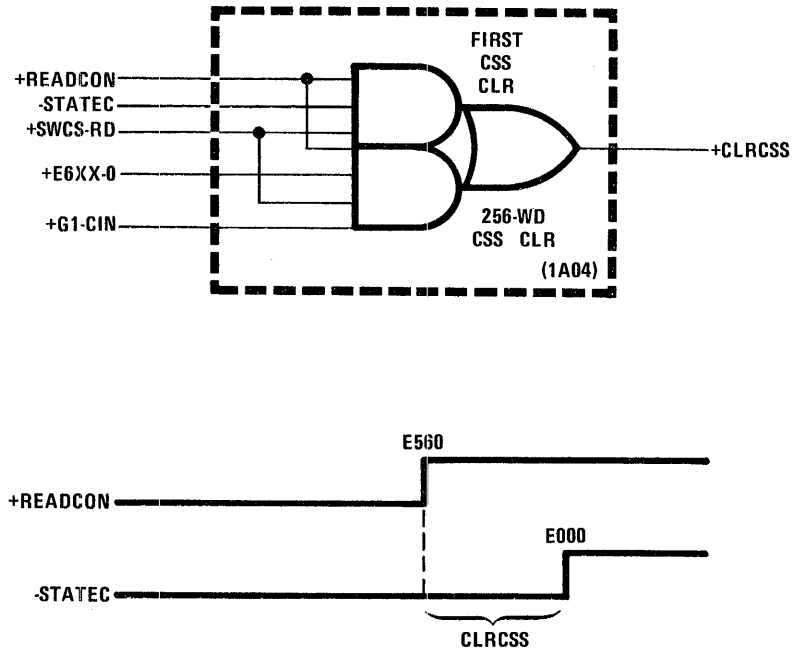


Figure 2-198. Clear CSS Register Logic

During normal μI execution, the μI's read from CS are unconditionally gated into the Fμ register for translation. During a CS scan/read operation, however, these μI's must be inhibited from entering Fμ. Blocking entry into Fμ for this purpose is provided by signal CLRFM, generated as shown in Figure 2-199. This signal is generated during the W portion of the previous time slice (at E560) via READCON to clear out the last μI executed before the Panel got its time slice. The signal stays high through the execute portion of the Panel time slice via STATEC to block all μI's read from CS during the Panel time slice. Another feature of normal μI execution is the updating of Sμ every time slice to address the next sequential μI in CS. Since only one μI is read per time slice when doing a CS scan or CS read operation, normal Sμ updating must be modified to occur only once during a time slice. This modification is accomplished by the BLKSMS signal, shown in Figure 2-199. The signal is held high during every Panel time slice to block clocking Sμ except at E000 time. At E000, signal E0XX-0 goes high for about 30 nanoseconds to allow the contents of Sμ to be gated to the Sμ+1 logic. During the rest of the Panel time slice, BLKSMS is held high by SWCS-RD and BLKSMFF. Signal BLKSMFF is generated from the set output of the Block Sμ flip-flop. This flip-flop is set by a master clear signal and remains set until completion of the CS scan/read operation.

Gating of the CS Scan register contents through the data display selectors is accomplished by three SELN signals, as shown in Figure 2-195. These signals are forced to a value required to gate data from the CS Scan register to the data register indicators on the Panel when the CONSOLE MODE SELECT selector is set to the CS-RD position. As a result, the data register selector does not have to be set to the CSS position, when doing a CS scan/read operation.

Parity errors detected during a CS scan operation are done so by the logic shown in Figure 2-200. This logic detects errors occurring both during the scan of CS and the AT. Errors that occurred during the scan of each CS page are detected by comparing the contents of the CS Scan register after the last address has been scanned in a page (CSS-FF9F) with the lower eight bits of Sμ (G1-CIN). If the CS Scan register contents are all 1's, except for bits 9 and 10 ($\overline{\text{CSS-FF9F}}$ low), at the same time that the lower eight bits of Sμ are all 1's (G1-CIN high) to indicate address 255_{10} (or a multiple of address 255_{10}), the page was loaded correctly. Any other combination signifies an error, which generates SCANERR at E350. This signal sets the CS PE flip-flop to light the CS PARITY ERROR indicator on the Panel. In addition, STOP-CS is also generated to stop the CS scan operation by clearing the Console Busy flip-flop. This signal, however, can be disabled by the DISABLE CS switch on the Panel to disable the CS scan stop condition. Parity errors detected during the AT scan are handled in a similar manner via generation of signal ATCHKEV.

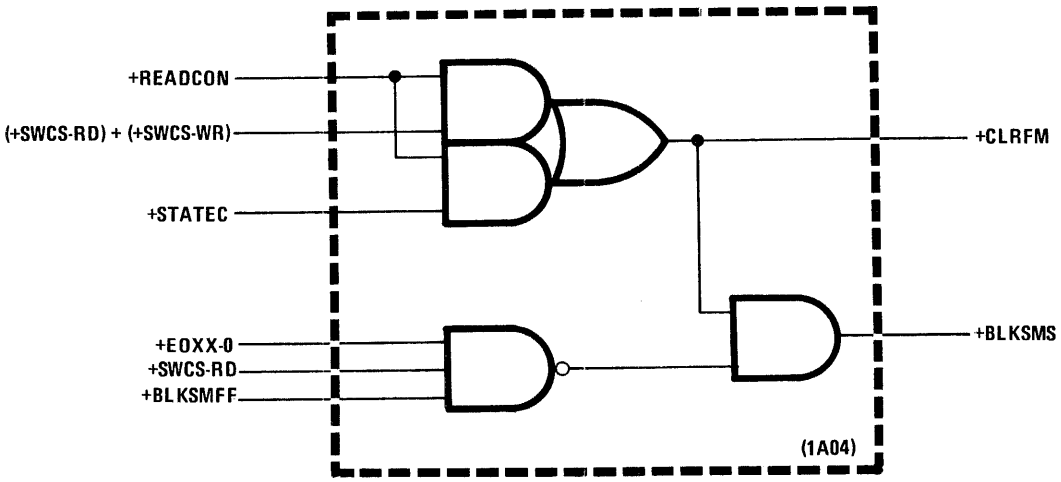


Figure 2-199. CLRFM and BLKSMS Logic

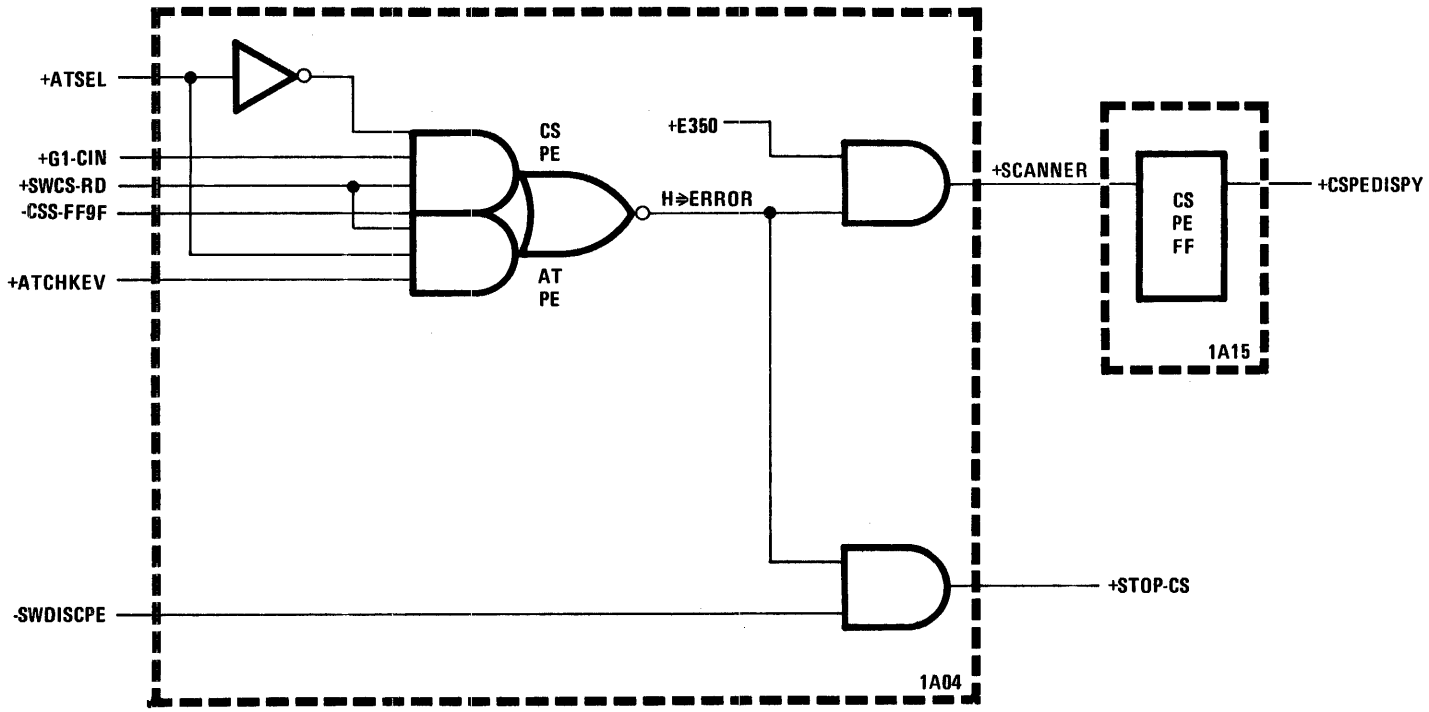


Figure 2-200. CS Scan Error Detect

Console Busy flip-flop. This signal, however, can be disabled by the DISABLE CS switch on the Panel to disable the CS scan stop condition. Parity errors detected during the AT scan are handled in a similar manner via generation of signal ATCHKEV.

CS Write

The CS write operation loads a particular word entered via the data register pushbuttons into the location in either CS or the AT specified by the contents of S . The operation may be performed in either the stop/step or the normal mode. If in the stop/step mode, different words may be entered into successive locations each time the CONSOLE RUN pushbutton is pressed. If in the normal mode, the same word may be dynamically written into successive locations of CS and through the last address of the AT automatically when the CONSOLE RUN pushbutton is pressed. In either case, the first location to be written into (if different from 0000_{16}) must be counted up to by first performing a breakpoint scan as described in the paragraph titled Console Modes. This scan is necessary since the Sμ register cannot be entered directly with an address. Like the CS scan and CS read operations, the CS write operation must be performed in the off-line mode (no other processor requesting time slices). (The CS write operation should not be confused with the initial CS load operation. The latter is used to load CS and the AT with new data at the beginning of a job. The former is used to change data already contained in CS and the AT as a result of the CS load operation.)

Logic for performing a CS write is shown in Figure 2-201. As in the CS scan/read, signal CONBUSY is generated for two conditions: continuously if the CS write operation is performed in the normal mode, or once per Panel time slice if performed in the stop/step mode. In addition, the contents of Sμ are updated once per time slice as controlled by BLKSMS. Data to be written is entered in the 16 console data register pushbuttons and stored in either CS or the AT via the data display selectors. These selectors are enabled by three SELN lines, which are forced by the CS-WR signal derived from the CS-WR position of the CONSOLE MODE SELECT selector switch to a value required to gate the data register output lines.

A write into either CS or the AT must be accompanied by a corresponding write enable: WRITE-CS or AT-WRITE. Both enables are generated at E700 of a Panel time slice, as shown in Figure 2-202. Selection of either WRITE-CS or AT-WRITE is controlled by the AT-SEL signal from the CS loader logic. During a CS write operation, this logic

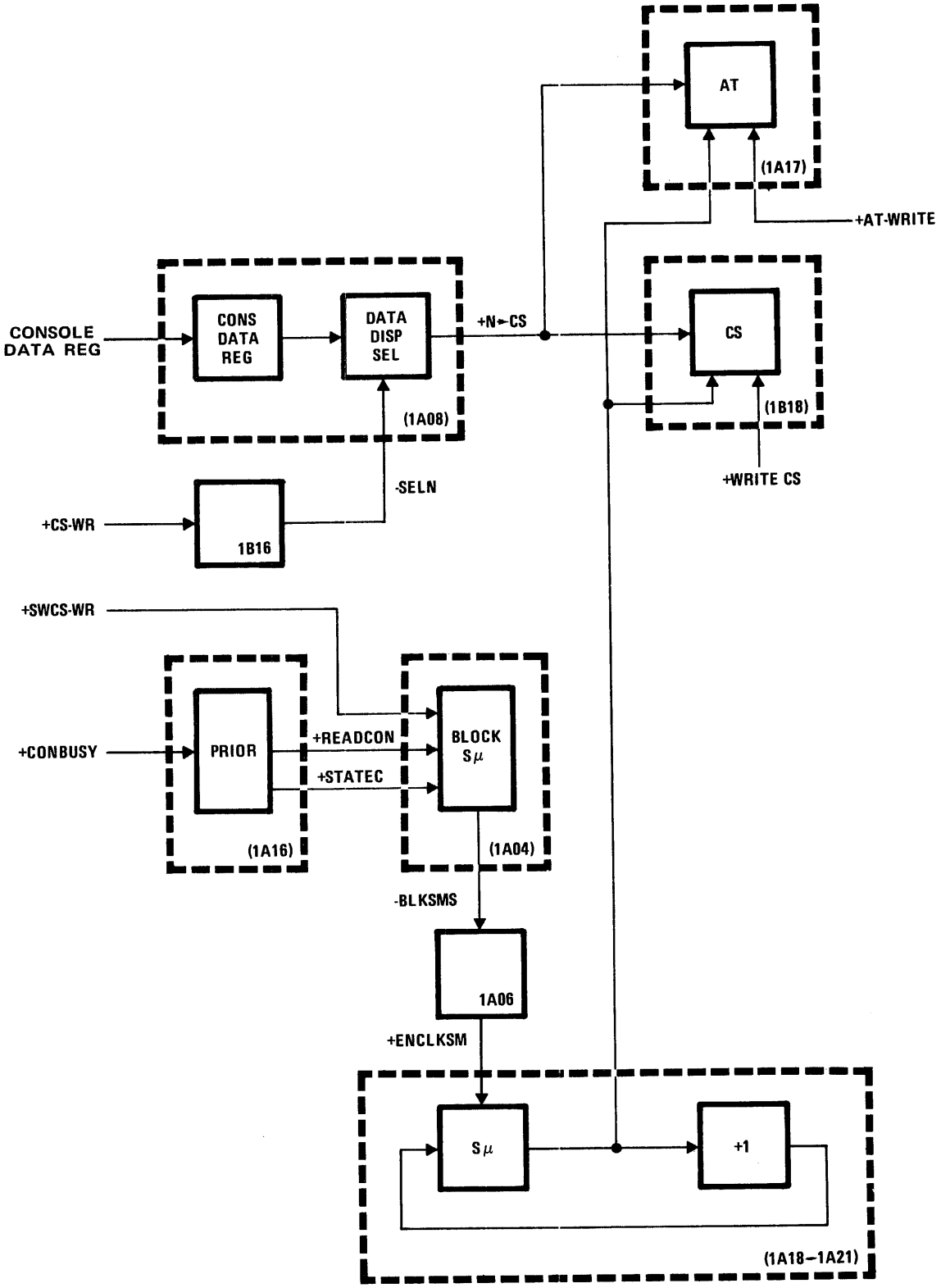
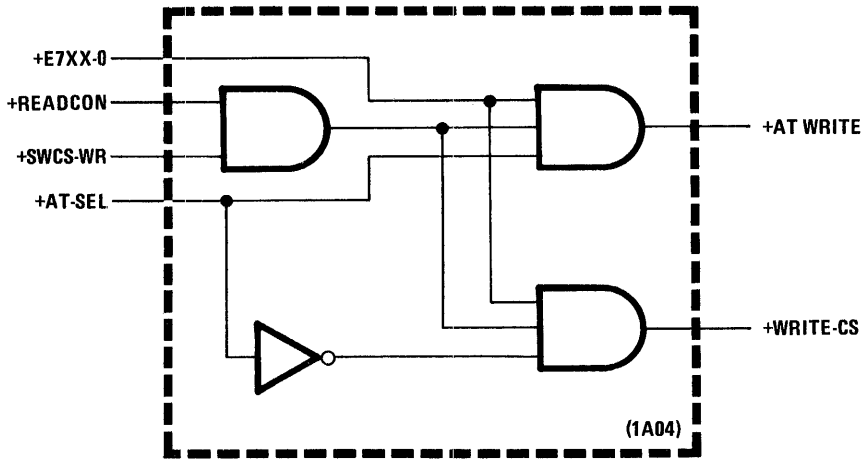


Figure 2-201. CS Write Operation



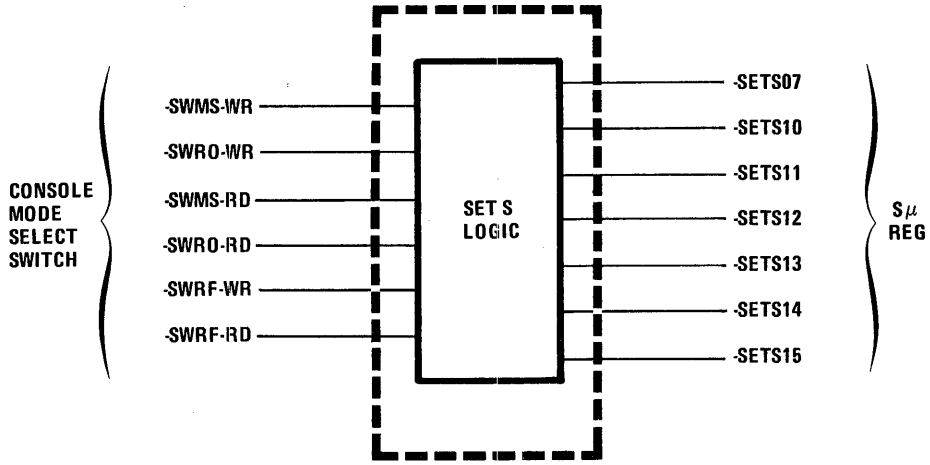


Figure 2-202. CS and AT Write Select

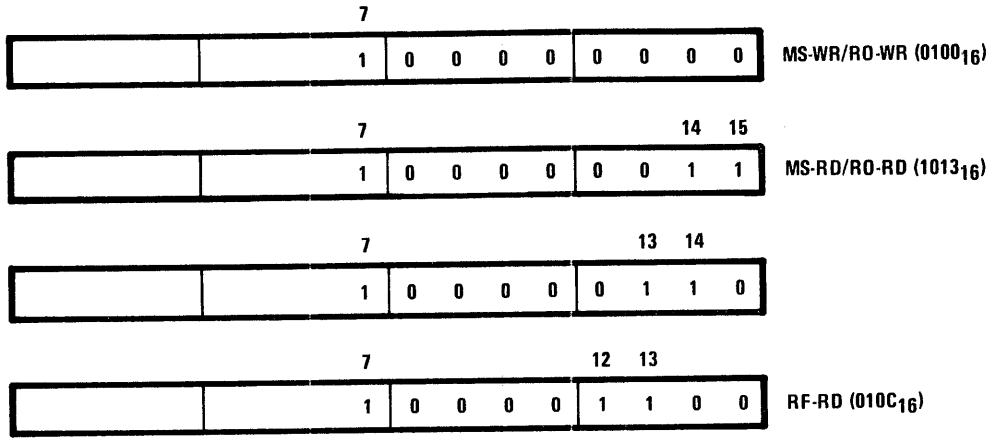


Figure 2-203. Set S Logic

also functions as during a CS load routine to monitor the contents of Sμ. Initially AT-SEL is low to generate WRITE-CS. When the CS loader logic determines that all of CS has been addressed, by comparing the contents of Sμ with the number of CS modules in the system, writing into the AT can begin. At this point, AT-SEL goes high to generate AT-WRITE.

In systems containing a 5120 (5K)-word CS, addressing errors will occur if attempts are made to address the upper 3072-word portion of the second storage unit. These errors will occur because "wrap-around" of the Sμ contents will not occur if addressing in this range; that is, the contents of Sμ will not have yet re-cycled back to a 4096-word boundary address. Addressing in this non-existent portion of CS will be indicated by the CS PARITY ERROR indicator on the Panel, which will light as a result of detecting what appears to be parity errors in the "bad data" located at these non-existent addresses.

MS/RO and RF Read and Write

Panel-initiated read and write operations performed on Main Storage (MS), or in registers of the Register File (RF) or Register Option (RO) are done so by means of corresponding μI subroutines located in CS. When the CONSOLE MODE SELECT selector on the Panel is set to the corresponding position (MS-RD, MS-WR, RF-RD, RF-WR, RO-RD, or RO-WR), a corresponding starting address is generated by the set S logic which causes a jump to the subroutine for performing the selected operation. The set S logic and the corresponding starting addresses generated are shown in Figure 2-203. The address is generated on seven lines that feed the Sμ register. (The complete jump address is 12 bits in length, where the remaining 5 bits are forced to zero in the Sμ register.) Each read or write subroutine causes one word to be read or written during each time slice assigned to the Panel. During the time slice, Sμ is updated in the normal manner until the subroutine is completed for one word. At the beginning of the next time slice, another word can be read or written since the position of the CONSOLE MODE SELECT selector automatically forces the starting address of the subroutine again.

Any of the subroutines may be performed in either the on-line mode (other processors running) or off-line mode (no processors running). If doing an RF operation, any of the registers in the BRF or Groups I and II of the ERF (except the Boundary Crossing register) may be accessed. (The Group III registers of the ERF, associated with I/O processors 0 through 3, may not be accessed by this mechanism.) The six read and write operations (MS read, MS write, RO read, RO write, RF read, and RF write) are implemented by four μI routines, with the MS and RO read and write operations sharing the same read and write routines.

The MS/RO read routine of Figure 2-204 reads the data in MS located at the address entered into the CONSOLE ADDRESS REGISTER DISPLAY pushbuttons and transfers it to the CONSOLE DATA REGISTER DISPLAY indicators. If the CONSOLE CONTROL SELECT switch is in the STOP/STEP position, one pass through the routines will be executed in one time slice each time the CONSOLE RUN pushbutton is pressed. If the CONSOLE CONTROL SELECT switch is in the NORMAL position and the CONSOLE RUN pushbutton is pressed, the routine will be repeated to read out the contents of all sequential locations above that entered into the address register pushbuttons until all of MS or the RO is read. The MS/RO write routine of Figure 2-205 operates in a similar manner: data to be stored at an address entered in the address register pushbuttons is entered in the data register pushbuttons. Again, data can

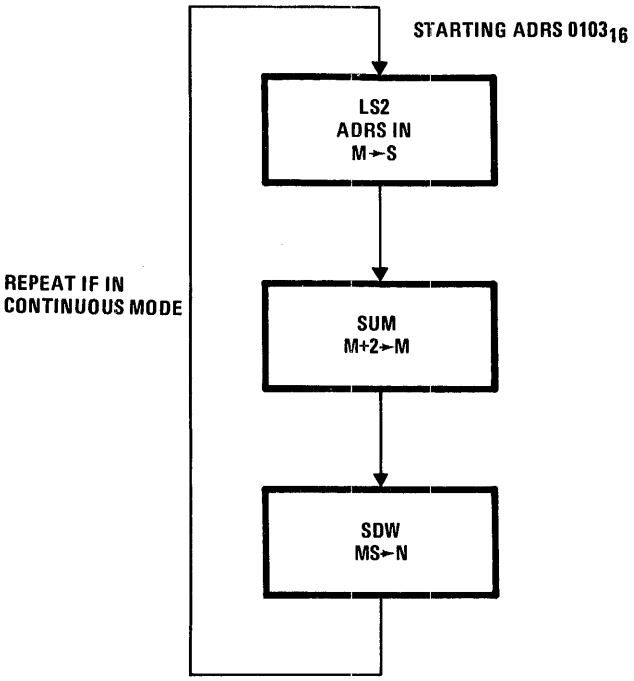


Figure 2-204. MS/RO Read Routine

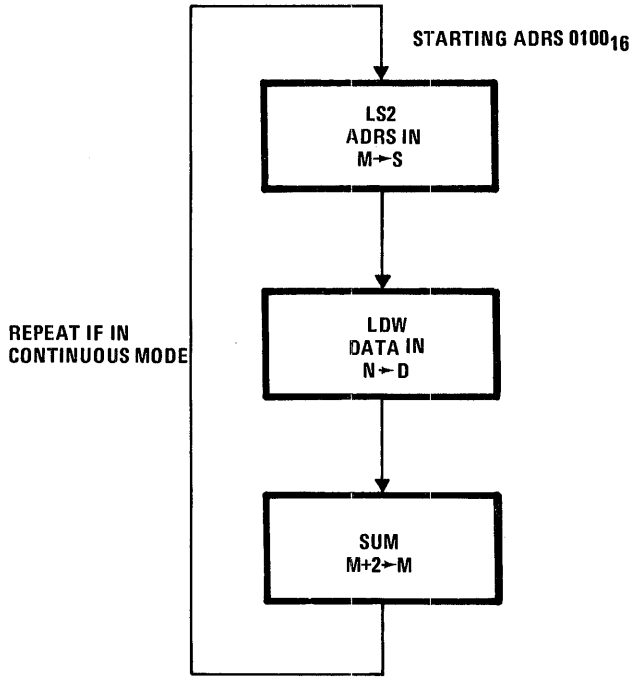


Figure 2-205. MS/RO Write Routine

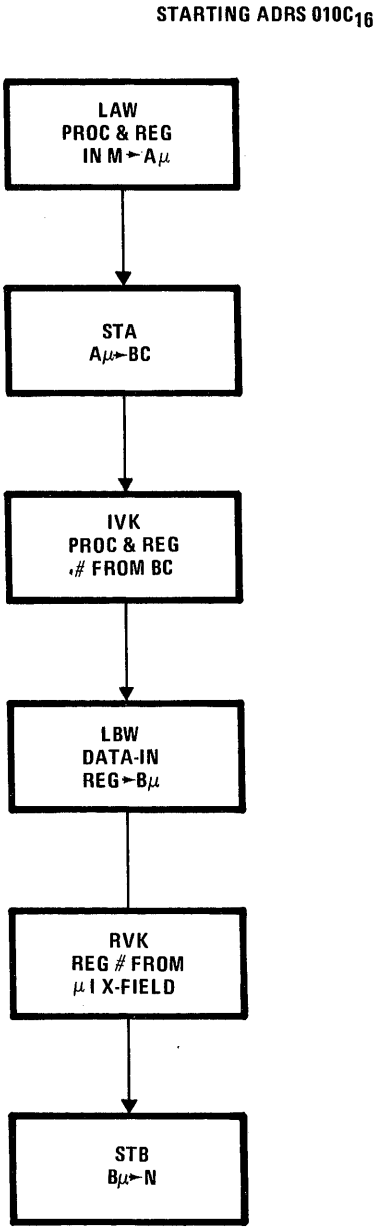


Figure 2-206. RF Read Routine

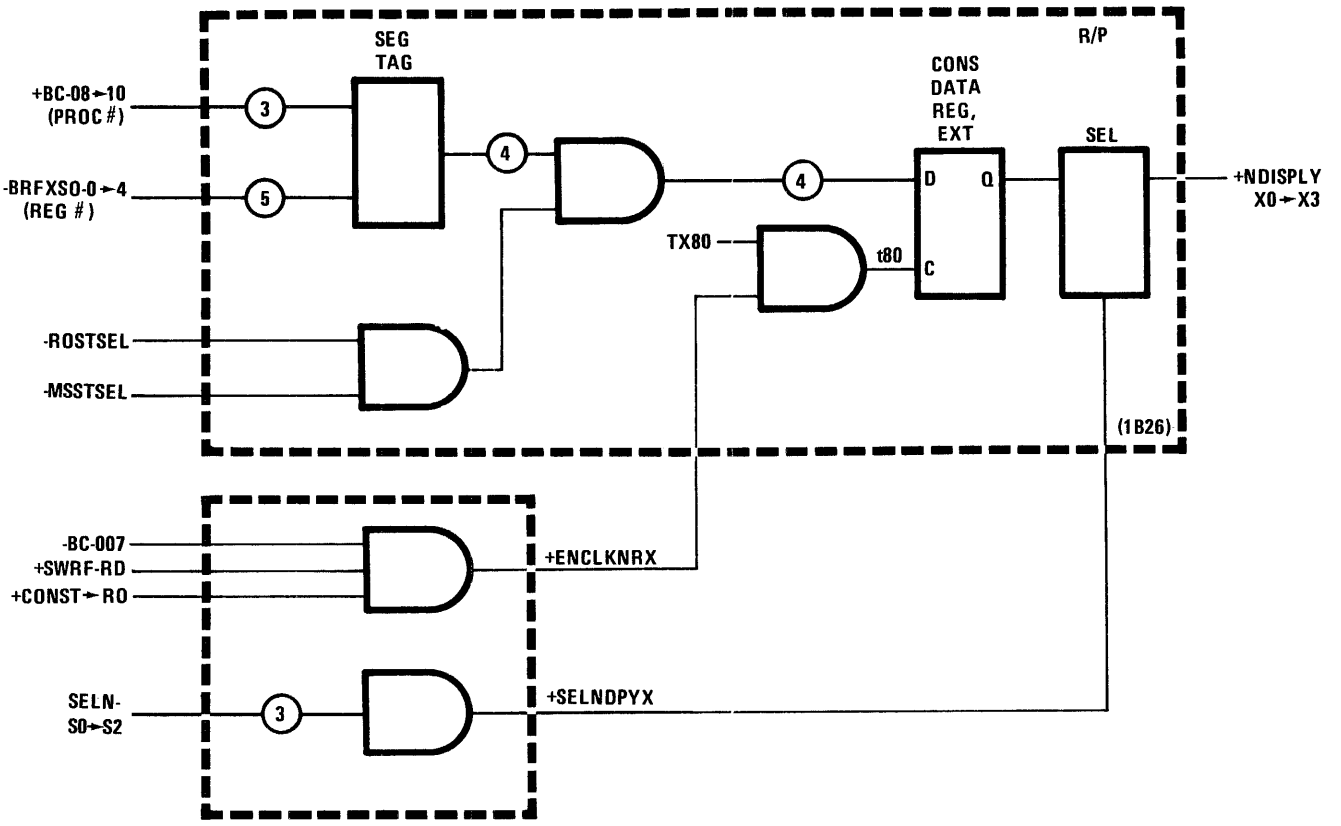


Figure 2-207. Reading Segment Tag Portion During Panel RF Read

be stored in either the stop/step or normal mode. When operating in the normal mode, the data entered into the data register is stored at the address entered into the address register and at all sequential locations above that address.

The RF read and write routines are somewhat more complex than those for MS and the RO because of the necessity to cross processor boundaries. Since the Panel is considered a processor, the only way it can gain access to the file registers of another processor is by means of the Boundary Crossing (BC) register. The RF read routine of Figure 2-206 takes the processor and register number entered into the address register pushbuttons and transfers it to the BC register via the Aμ register. Then an IVK μI is executed so that the processor file register to be read can be addressed by the contents of the BC register. The contents of the selected register are read and transferred to Bμ. The contents of Bμ are then routed to the data register by a STB μI preceded by a RVK μI. The RVK μI is necessary to cancel the IVK μI so that the desired register number (that of the console data register) can be derived from the STB μI X-field instead of from the BC register.

If the Relocation and Protection feature of the RO is present, the RF read routine will also read the Segment Tag register corresponding to a BRF register selected and display the four bits of this register in the X0 through X3 indicators of the CONSOLE DATA REGISTER DISPLAY. This is accomplished by the logic shown in Figure 2-207. The segment tag is selected by $\overline{\text{BC}}$ and $\overline{\text{BRFSXO}}$ bits derived from the BC register instead of from the resource allocation logic and μI X-field. The selected register is clocked into the extended portion of the Console Data register at t80 by ENCLKNRX. This enable is generated for a RF read operation assuming that an ERF register has not been selected ($\overline{\text{BC-007}}$ is high). The 4-bit tag value in the data register is selected for display in the CONSOLE DATA REGISTER DISPLAY X0 through X3 indicators by SELNDPYX. This select signal is generated when the CONSOLE DATA REGISTER SELECT selector is set to DATA.

The RF write routine of Figure 2-208 is executed similarly, the only significant difference being that data is to be written into the processor file register selected by the contents of the BC register. This data again is entered into the console data register. As for MS read and write operations, RF operations can be executed in either the stop/step or normal mode. If the Relocation and Protection feature of the RO is present, the RF write routine may also be used to *write* the Segment Tag register corresponding to the BRF register selected. This is

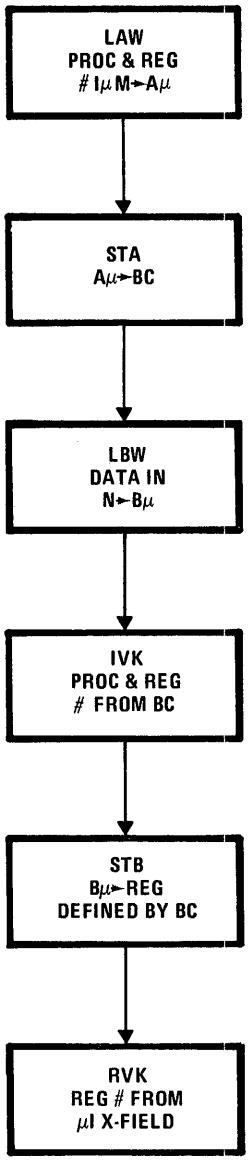


Figure 2-208. RF Write Routine

accomplished by the logic shown in Figure 2-209. The segment tag value set into the extended portion of the Console Data register via the SWSETN X0 through X3 signals is routed to the selected Segment Tag register. This register is selected by the BC and BRFSXO bits from the BC register. The value is written into the register by write enable SEGTAGWR. For this purpose, SEGTAGWR is generated for a register file write operation under control of an IVKμ I (signal INV-F/F high). The tag value is also selected for display in the extended portion of the data register indicators by select signal SELNDPYX.

The BC register itself may not be selected as a register to write into by the RF write method. For this unique case, a situation is created where the BC register attempts to address itself by its own contents. A circular condition results during which the write operation generates spurious address bits that attempt to select other registers at random.

OPERATING MODES

Processor Modes

The basic rationale governing processor execution in a selected mode is to start the processor by pressing the PROCESSOR RUN pushbutton, which sets the processor's Busy flip-flop in the B/A register; and stop the processor by means of signals generated according to the mode selected, which clear the processor's Busy flip-flop. Logic for starting the processor via the PROCESSOR RUN pushbutton is shown in Figure 2-210. Pressing this switch generates SW-GO, which sets the Processor Run flip-flop. The output of this flip-flop is fed to one side of a NAND gate to set the Go flip-flop. The other side of the NAND gate is fed with the Go Button flip-flop output routed through a one-shot circuit. The one-shot assures that the set pulse to the Go flip-flop is only 100 nanoseconds wide.

Setting the Go flip-flop generates GO-FF, which is routed to the B/A register to set the processor's Busy flip-flop upon receipt of a corresponding processor select signal. This signal is generated by setting the PROCESSOR SELECT selector to the desired processor number, which generates a SWSELGO signal. These two signals, GO-FF and SWSELGO, set the processor's Busy flip-flop, as described in the paragraph titled Busy/Active Register. Once the processor's Busy flip-flop is set, the Go flip-flop can be cleared to allow another processor to be turned on from the Panel. This is accomplished by the CLR-GOFF signal, which is generated from SWSELGO and the STATE signal from the RAN.

Normal Mode

The processor normal mode is initiated by the sequence of events just described plus setting the corresponding PROCESSOR CONTROL SELECT switch to the NORMAL position. This switch position does not produce a corresponding signal as do the processor select switch and GO button. Its selection, however, is implied by the absence of a signal from either the STOP/STEP or BREAKPOINT positions of the PROCESSOR CONTROL SELECT switch. As a result, the processor will continue to run indefinitely in this mode until either the stop/step or breakpoint mode is selected.

Stop/Step Mode

The processor stop/step mode is selected by setting the corresponding PROCESSOR CONTROL SELECT switch to the STOP/STEP position and proceeding as for the

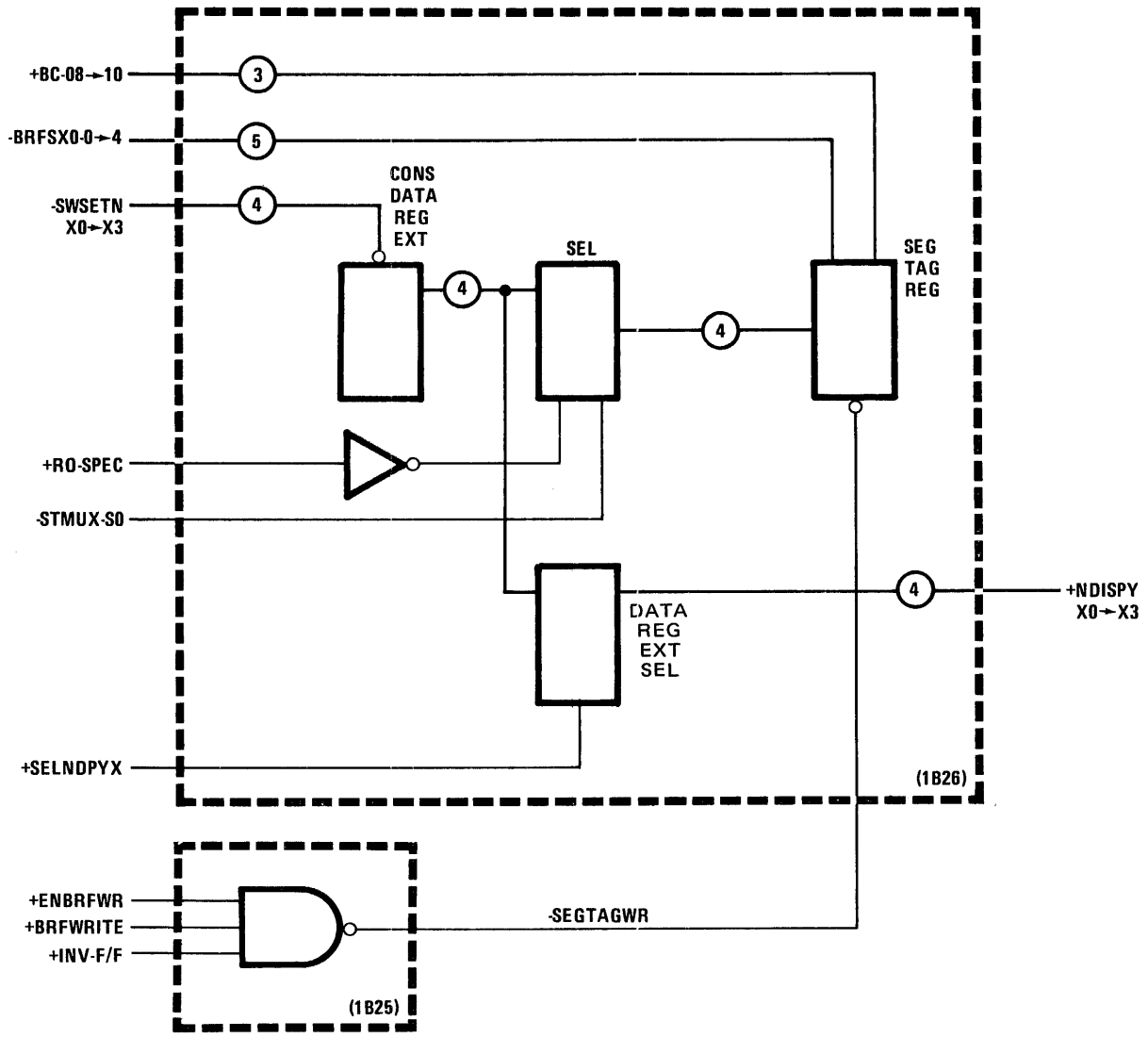


Figure 2-209. Writing Segment Tag During Panel RF Write

processor normal mode. As discussed previously, the stop/step can be executed by performing either one major cycle or one MLI per depression of the PROCESSOR RUN pushbutton, as determined by the setting of the CYCLE STEP switch. Logic for implementing these two sub-modes is shown in Figure 2-211. When the stop condition defined by either of these two sub-modes is met, signal $\overline{\text{RNI-TX}}$ is generated. This signal is fed to the clear side of the corresponding Busy flip-flop in the B/A register to clear the flip-flop. If the CYCLE STEP switch is set to the up position, the processor runs for one major cycle and is then turned off. This is implemented by signal $\overline{\text{SWCYCSTEP}}$ from the up position of the switch. This signal generates $\overline{\text{RNI-TX}}$ at E350 of at least one time slice preceding the one during which the selected processor will execute. The $\overline{\text{RNI-TX}}$ signal, therefore, is present during the execution time slice which means that the processor will be turned off at the end of this single time slice.

If the CYCLE STEP switch is set to the down position, the processor runs for one MLI as determined by signal $\overline{\text{RNI-F/F}}$ from the RNI flip-flop. This flip-flop sets upon detection that the present MLI has been completed and the RNI sequence of the next MLI has been executed. The RNI breakpoint sub-mode discussed in the paragraph titled Normal Mode makes use of the fact that the next MLI RNI sequence has been executed. The cycle step evaluation, however, is interested in knowing only that the present MLI has been completed. The logic that drives the flip-flop makes an evaluation of the address in Sμ to determine if the MLI RNI sequence has been executed. This is done by examining bits 4 through 11 of S to see if they are all 0's. If they are 0, $\overline{\text{SM→FRJ04}}$ through $\overline{\text{SM→FRJ11}}$ are all high which indicates that Sμ has been reset to either 0002_{16} (RNI1 sequence starting address) or 0009_{16} (RNI2 sequence starting address) to start the next MLI, but has not been updated past $000F_{16}$ (last

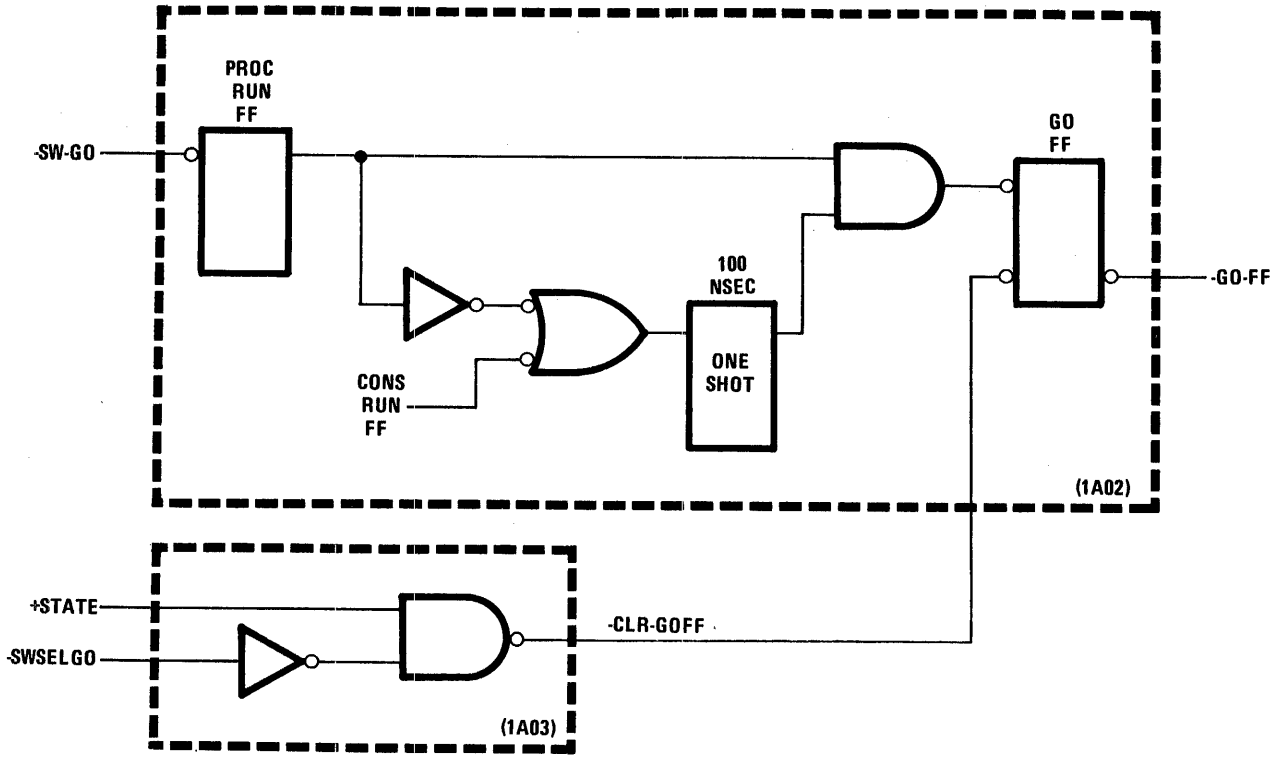


Figure 2-210. GO Flip-Flop

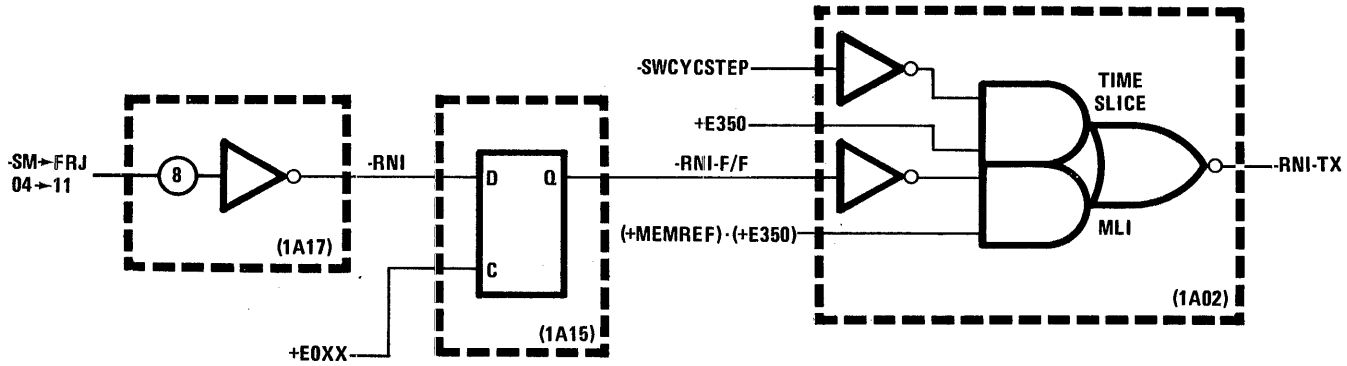


Figure 2-211. Cycle Step Logic

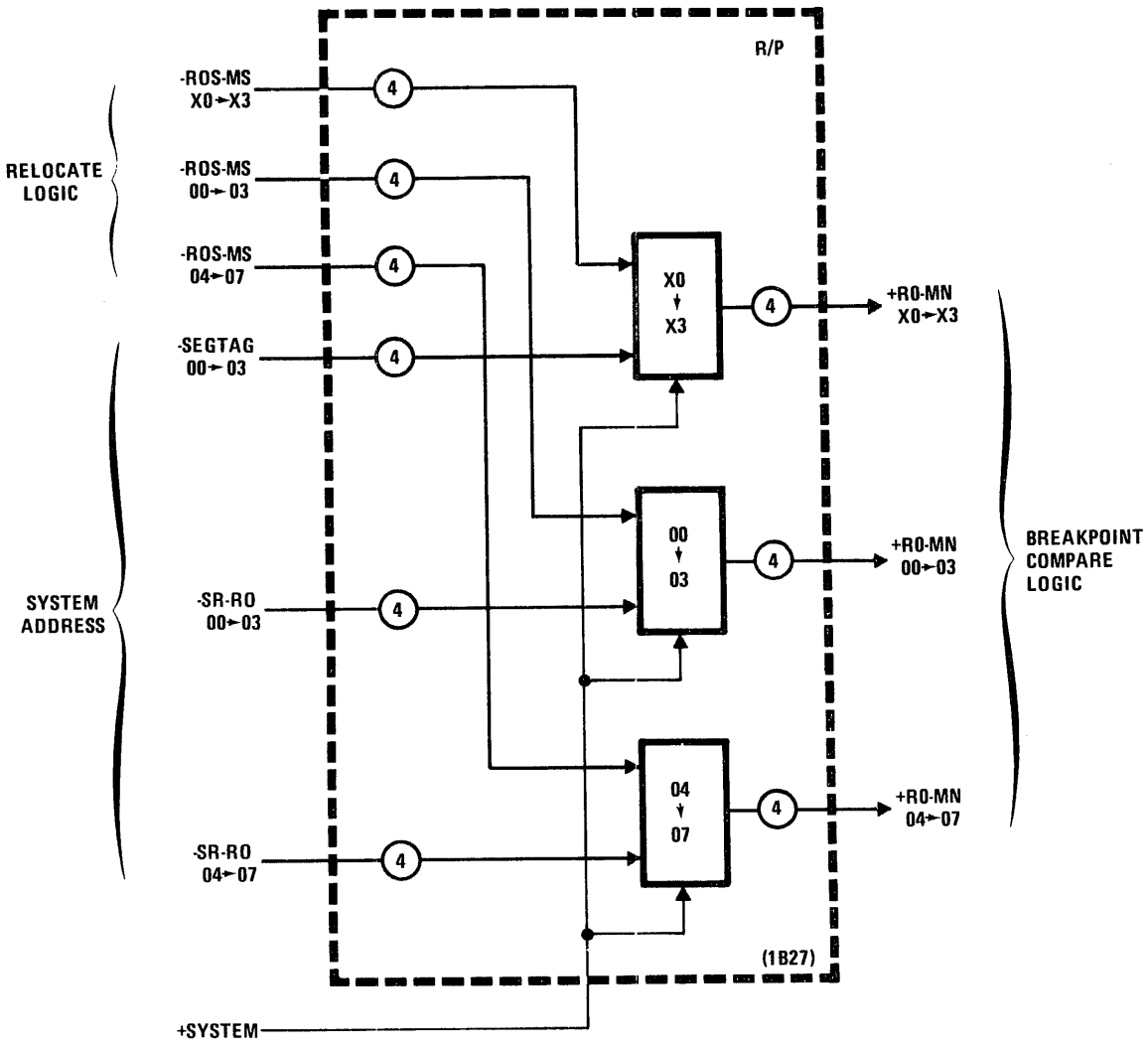


Figure 2-212. Selection of System or Relocation Address for Breakpoint Compare

address of RNI2 sequence). The result is to generate $\overline{\text{RNI-TX}}$ upon completion of either the RNI1 or RNI2 sequence.

Breakpoint Mode

The processor breakpoint mode is entered by setting the corresponding PROCESSOR CONTROL SELECT switch to the BREAKPOINT position. When set to this position, the 20-bit breakpoint address set in hexadecimal form by the five BREAKPOINT ADDRESS SELECT selectors on the Panel (assuming the Relocation and Protection feature of the RO is installed) is continuously compared with the last physical address that referenced a word (either MLI or data) in MS. The lower 8 bits of this physical address are derived directly from the S register; the upper 12 bits from the RO. These upper 12 bits may be derived from either the system address or from the relocation logic, depending on the position of the SYSTEM/PHYSICAL switch as shown in Figure 2-212. If set to SYSTEM, select signal SYSTEM is high and the upper 12 bits are derived from the system address (upper 8 bits of S and the 4-bit segment tag value). If set to PHYSICAL, signal SYSTEM is low and the upper 12 bits come from the relocation logic and will usually represent a relocated equivalent of the system address.

Upon reading a breakpoint condition (two addresses equal), the processor is stopped by clearing its Busy flip-flop. Logic for performing S register breakpoint comparisons is shown in Figure 2-213. For purposes of simplification, only the left-most of the five selectors is shown. This selector generates an encoded four-bit address corresponding to one of the 16 positions of the selector (0_{16} to F_{16}). This encoded address is compared in complement form with extension address bits X0 through X3 from the RO in true form for a match. The comparison is made on a bit-by-bit basis via exclusive-OR gates. If all four bits from the selector match the corresponding four bits from S, signal SRBKCP-X0 goes high. The other 16 bits of S are compared with encoded addresses from the other four selectors in a similar

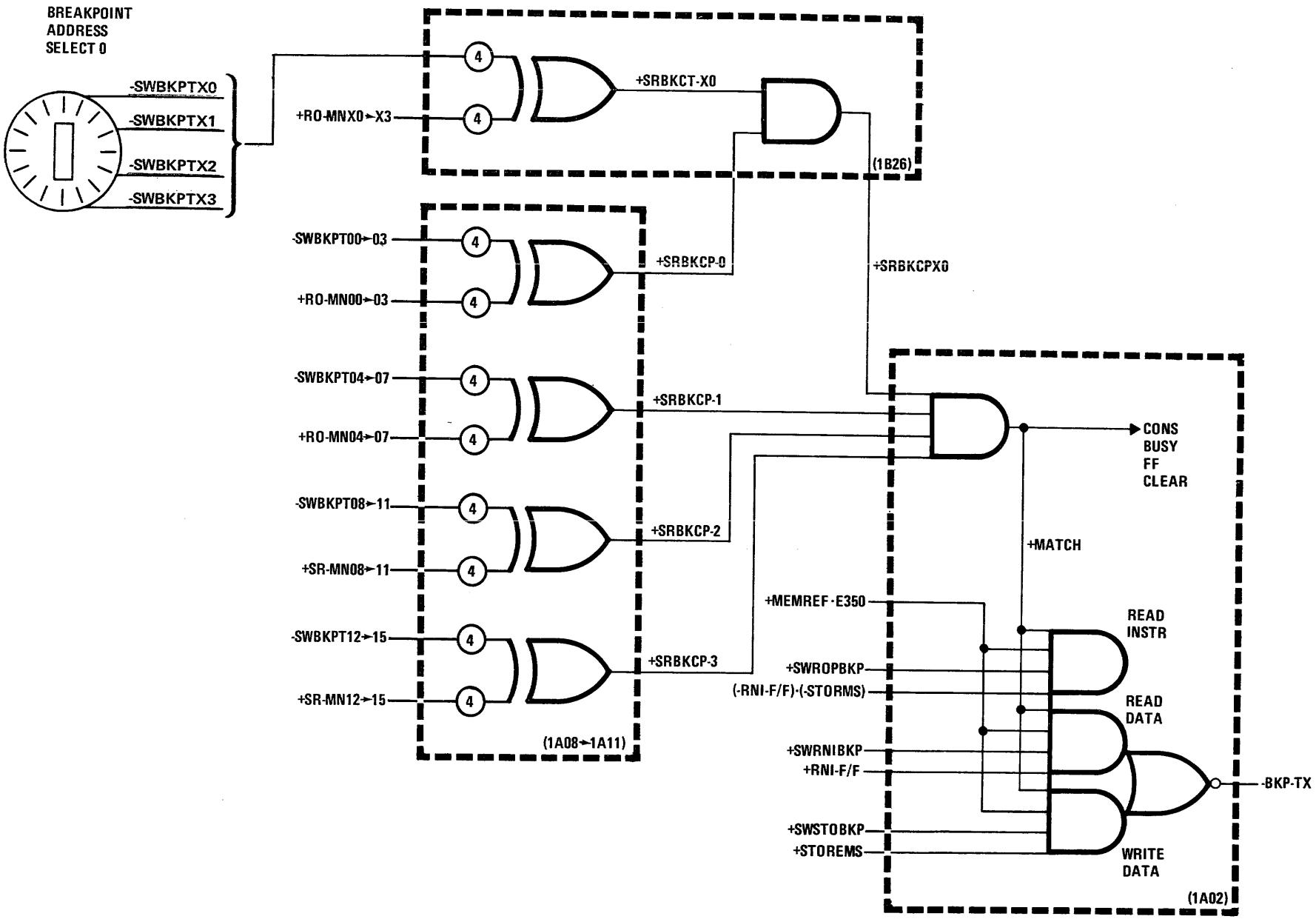


Figure 2-213. S Register Breakpoint

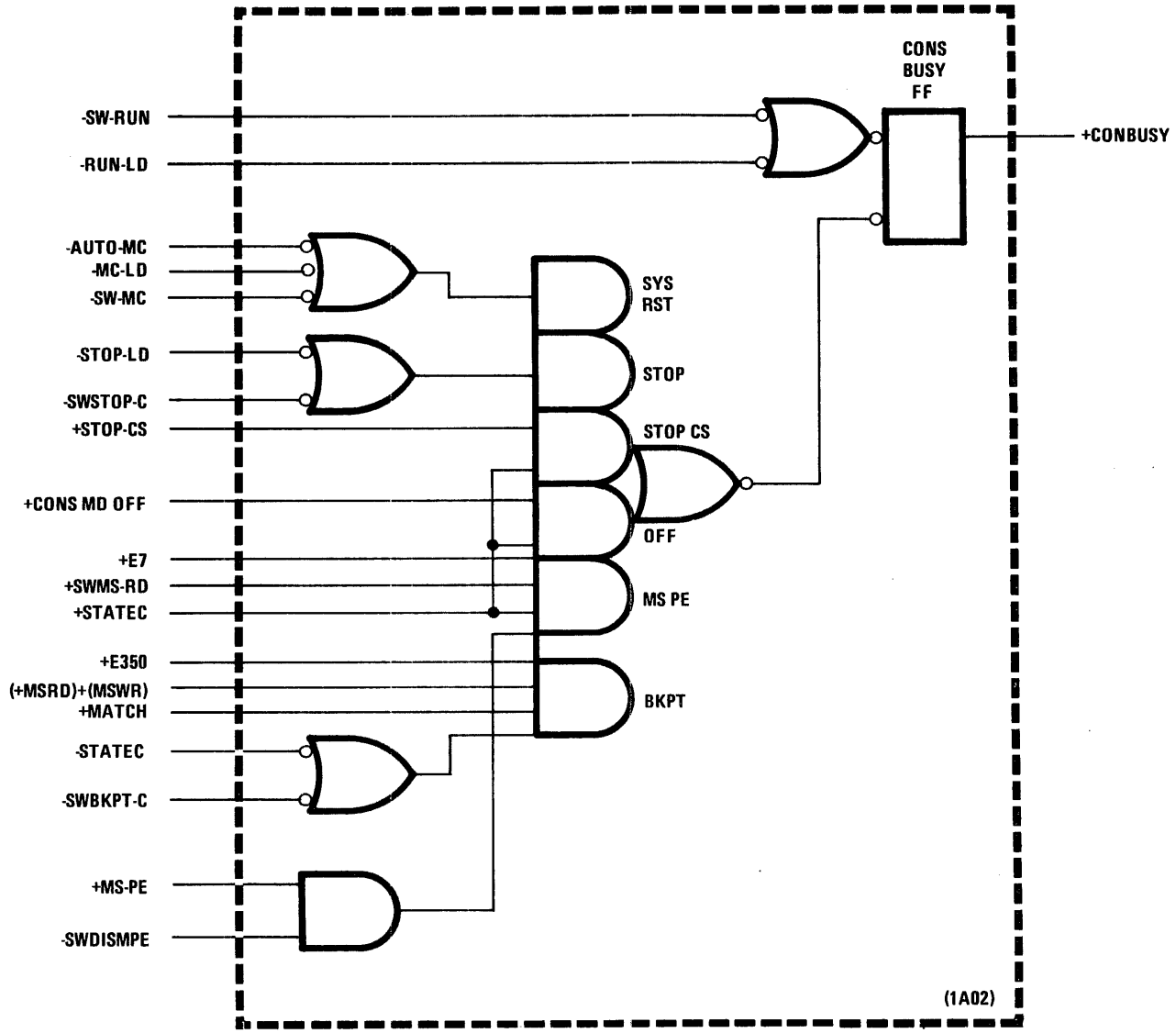


Figure 2-214. Console Busy Flip-Flop

fashion. If all 20 bits of the physical address match the hexadecimal address set in the selectors, signal MATCH is generated and routed to the breakpoint sub-mode logic. The signal is also routed to the Console Busy flip-flop clear logic of Figure 2-214 for use during Console-controlled MS read and MS write breakpoint scan operations.

The breakpoint sub-mode logic consists of three NAND gates corresponding to the three breakpoint sub-mode switches: READ DATA, READ INSTR, and WRITE DATA. If the READ DATA switch is set to the up position, SWROPBKP is generated to stop the processor after the operand located at the breakpoint address has been read. Indication of an MS read operation is furnished by $\overline{\text{RNI-F/F}} \cdot \overline{\text{STOREMS}}$, meaning neither an RNI or an MS store operation was performed. The other two sub-mode switches stop the processor upon indication that the breakpoint occurred for their particular conditions. (As discussed in the last paragraph, signal RNI-F/F is used here in the breakpoint compare logic to indicate that a new MLI has just been read.) When the particular sub-mode condition is met, signal $\overline{\text{BKP-TX}}$ is generated. This signal is routed to the B/A register to clear the processor's Busy flip-flop.

Panel Modes

Selection of a Panel mode is made by means of the CONSOLE CONTROL SELECT switch. Like the PROCESSOR CONTROL SELECT switches which select processor modes, this switch allows the Panel to operate in one of three modes: normal, stop/step and breakpoint. Each mode is initiated by pressing the CONSOLE RUN pushbutton, which sets the Console Busy flip-flop. This flip-flop is similar to the processor Busy flip-flops in the

B/A register in that it enables the Panel to obtain time slices through the RAN. If operating in the normal mode, the Console Busy flip-flop remains set until the CONSOLE CONTROL SELECT switch is set to either the STOP/STEP or BREAKPOINT position. (The NORMAL position of the CONSOLE CONTROL SELECT switch is implied by the absence of a signal from the STOP/STEP or BREAKPOINT positions of this switch.) When this is done, the flip-flop will be cleared when the corresponding stop/step or breakpoint condition is reached.

Logic which sets and clears the Console Busy flip-flop is shown in Figure 2-214. The flip-flop is set either manually by pressing the CONSOLE RUN pushbutton (SW-RUN signal) or under program control during a CS load operation (RUN-LD signal). Clearing the flip-flop is accomplished when any of six conditions is present: system reset, Panel stop mode, stop CS, off, MS parity error, or Panel breakpoint mode. Each of these conditions satisfies a corresponding NAND gate, which generates a low output to clear the flip-flop.

The system reset (SYSRST) gate is satisfied by either AUTO-MC, MC-LD, or SW-MC. Signal AUTO-MC is generated at the beginning of an autoload sequence. Since the autoload sequence is a processor-controlled operation (processor 4), the Console Busy flip-flop must be cleared. The MC-LD signal is generated at the beginning of a CS load routine to clear the flip-flop until a CS word is ready to be transferred from either the disc or card reader. Signal SW-MC is produced by the SYSTEM RESET pushbutton on the Panel for purposes of doing a general system reset.

The Stop gate clears the Console Busy flip-flop upon detection of a stop mode condition. Generally, this condition will be implemented by setting the CONSOLE CONTROL SELECT switch to the STOP/STEP position, generating SWSTOP-C. For this condition, the Panel will execute one major cycle per depression of the CONSOLE RUN pushbutton. During a CS load routine, however, the stop condition is implemented to clear the Console Busy flip-flop after each CS word has been transferred until the next word is ready for transfer. This condition generates STOP-LD which, in conjunction with RUN-LD, set and clear the flip-flop at one-major-cycle intervals.

The Stop CS gate clears the Console Busy flip-flop upon detection of either a CS scan error or an Sμ register breakpoint condition. For either case, signal STOP-CS is generated. Generation of STOP-CS due to a CS scan error condition is discussed in the paragraph titled CS Scan/Read; this paragraph discusses generation of the signal due to an Sμ register breakpoint condition. This condition is usually implemented for purposes of scanning up to a particular CS address to begin a CS read operation. The scan operation consists of comparing the present CS address in Sμ with the breakpoint address set in the right-most four BREAKPOINT ADDRESS SELECT selectors. Logic for accomplishing this is shown in Figure 2-215. The compare operation is identical to that for the S register breakpoint compare shown in Figure 2-213, except that the CS breakpoint scan compare is made on the contents of Sμ instead of S. Like 2-213, Figure 2-215 shows details for only one of the selectors and the corresponding four bits of Sμ. The breakpoint scan mode is entered by setting the CONSOLE CONTROL SELECT switch to the BREAKPOINT position, which generates signal SWBRKPT-C. Upon detection of a breakpoint compare, signal STOP-CS is generated which clears the Console Busy flip-flop.

The Off gate clears the Console Busy flip-flop when none of the Console functions has been selected by the CONSOLE MODE SELECT selector, that is, the switch is set to the OFF position. Detection of a parity error (PE) during an MS read operation clears the flip-flop via the MS PE gate. This gate is fed with PE information from the MS PE display logic via the MS PE signal. This signal is ANDed with SWDISMPE, which is generated by the STORAGE PARITY DISABLE switch on the Panel. If activated, this signal goes low to disable the MS PE signal. The resultant output is fed to the MS PE gate of the Console Busy flip-flop clear logic.

The BRKPT MODE gate is satisfied by a breakpoint stop during an MS read or MS write operation. This stop will occur as a result of reading or writing a block of data between some starting address and an ending address entered into the five BREAKPOINT ADDRESS SELECT selectors. The compare is made by the S register breakpoint logic of Figure 2-213. As shown in the figure, the MATCH signal generated upon reaching the breakpoint address is routed to the BKPT MODE gate of Figure 2-214.

LOADS

Disc CS Load

A disc CS load may be initiated in one of three ways:

1. Setting the POWER ON pushbutton to on (power on load)
2. Pressing the RESET/LOAD pushbutton (reset/load load)
3. Executing a CS Load disc command (CS disc command load)

The power on and reset/load loads are initiated under operator control via the System Control Panel. The power on load may be performed with the system in the

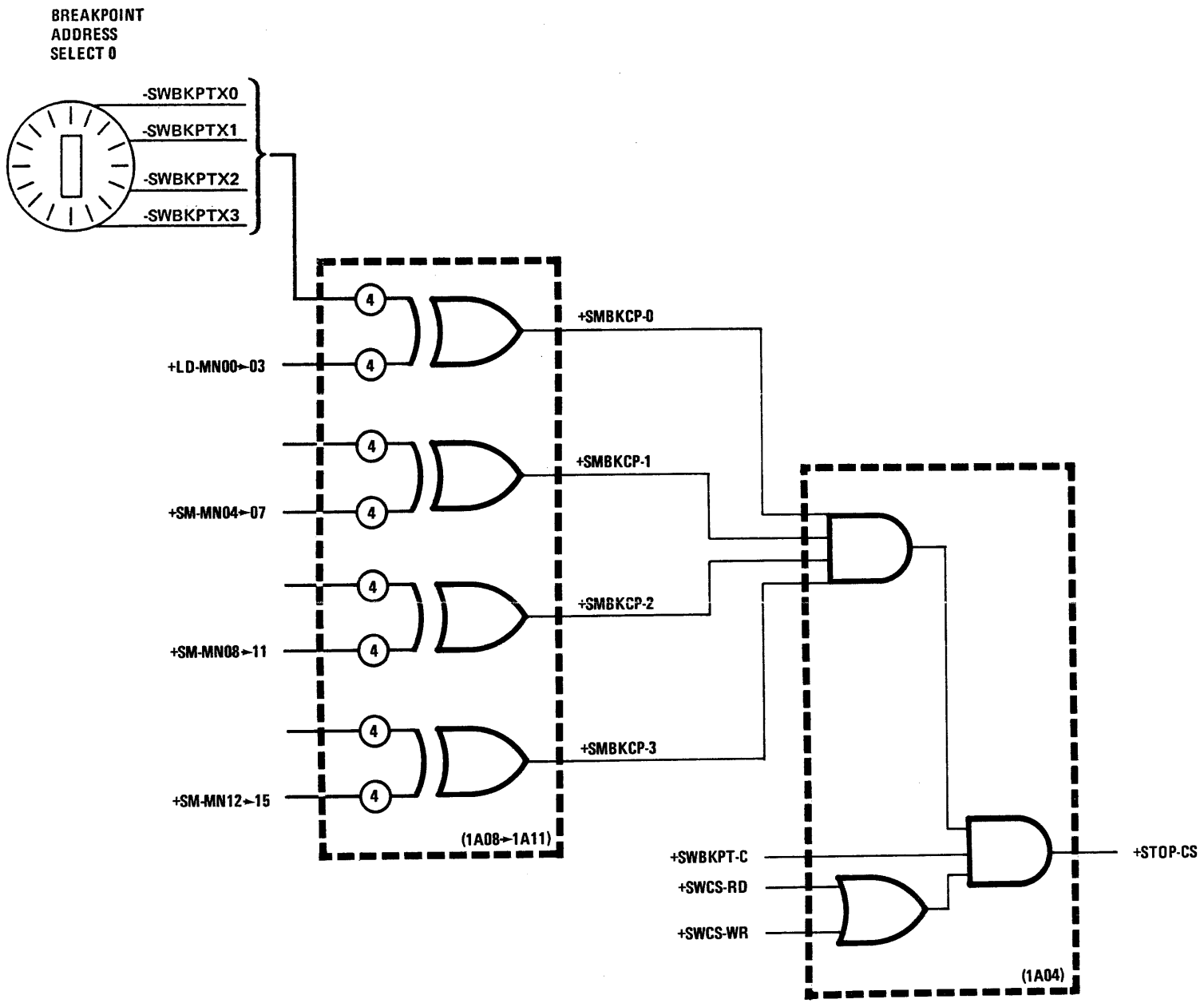


Figure 2-215. Sμ Register Breakpoint

operator or program mode, the reset/load load may be performed with the system in the operator mode, program mode, or maintenance mode. The third method is initiated under program control and may be performed with the system in the operator mode, program mode, or maintenance mode.

Each load may be divided into two parts: an initiate part and a data transfer part. The initiate part generates signals in the shared resources that set up conditions in both the disc IFA and shared resources in preparation for the subsequent transfer of words to be stored in CS. Logic used during the CS load initiate part is shown in Figure 2-216. The logic generates eight initiate signals. Four of these eight signals (DISCS, POMC-IO, DOA, and MC-IO) are sent to the IFA; the other five (LDCS-WR, STOP-LD, MC-1, MC-2, and MC-3) are used within the shared resources. Prior to beginning any of the three CS loads, the disc IFA must be selected as the device from which the load will be made. This is accomplished by setting the LOAD SELECT switch on the Panel to the DISC position. Setting the switch to this position generates SWDISC, which is sent to the shared resources to set the Load Select flip-flop. Setting this flip-flop generates DISCS, which enables the IFA for the CS load and subsequent MS load.

Power-On Load

The power-on load is initiated automatically when power is initially applied to the system via the POWER ON pushbutton on the Panel. When set to the ON position, the switch initiates a power-on system reset sequence. During this sequence, PWRON-MC is low which sets the Power On System Reset flip-flop. Through one level of inversion, the flip-flop (set output) generates POMC-IO. This signal is routed to the IFA to clear the First Seek Drive 0 flip-flop. The true form of the set output is fed to the Power On OR-gate to generate MC-LD which, in turn, generates MC-IO, MC-1, MC-2, and MC-3. Signal MC-IO is used to clear all registers and counters in the IFA except the Data Byte counter. This counter instead is set to a count of 3328_{10}, for use as a word transfer counter. Signals MC-1 and MC-2 are used in the shared resources to clear the Sμ register to address 0000_{16}, at which loading of CS will commence. Since Sμ cannot be cleared directly, it is done by clearing the Bμ register in the ALU and transferring its contents (zeros) to Sμ. Logic for accomplishing this is shown in Figure 2-217. Signal MC-1 generates ENRBM-0 and ENRBM-1 which resets (clear) both halves of Bμ. The output of Bμ is routed to S through the Sμ fan-in logic when enabled by ENALU-SM. Signal MC-2 generates EXCEPT which, in turn, is used to generate ENCLKSM. Signal EXCEPT is generated for this purpose of setting address 000_{16} into Sμ for beginning a CS load. Signal MC-3 is routed to the clear side of the DOA flip-flop to clear this flip-flop upon detection of a burst check error.

Upon completion of the power-on system reset sequence, PWRON-MC goes high to clear the Power On Master Clear flip-flop. The resultant low from the set side is inverted and fed to three one-shot circuits and an OR gate to set the DOA flip-flop. Signal DOA (Dead Start) initiates the CS load operation in the IFA, starting from cylinder 0, track 1. It is delayed about 600 nanoseconds from MC-LD (and therefore MC-IO and MC-1, MC-2, MC-3) via one-shot circuit 2 to allow the master clear operation initiated by these two signals to be completed. One-shot circuit 3 furnishes a negative pulse 60 nanoseconds wide to set the DOA flip-flop. Signal DOA is inverted to form STOP-LD. This signal is used in conjunction with RUN-LD (see Figure 2-218) to start and stop the RAN for enabling single-word transfers of CS data. Signal DOA is also ANDed with SWMAINT (MAINTENANCE MODE pushbutton not on) to generate LDCS-WR. This signal forces a CS load (CS write) condition and acts as if the CONSOLE MODE SELECT selector were set to the CS-WR position. The signal also forces selection of the Console Data register as the means for transferring data from the disc to CS. This forced selection simulates setting the CONSOLE DATA REGISTER SELECT selector to the DATA position. The action of clearing the Power On System Reset flip-flop also deactivates MC-IO, MC-1, MC-2, and MC-3.

Reset/Load Load

The reset/load load is initiated manually by means of the switch on the System Control Panel. This load is similar to the power-on load except that POMC-IO is not generated and that performing this load in the maintenance mode also depends on activating the SYSTEM RESET pushbutton and setting the CONSOLE MODE SELECT selector to CS-WR. Regardless of whether the system is in the operator mode, program mode, or maintenance mode, pressing the RESET/LOAD pushbutton activates SWDEADS. This signal sets the DOA flip-flop via one-shot circuits 2 and 3. In addition, the signal generates MC-LD via one-shot circuit 2 and the Reset/Load OR-gate if the system is in either the operator mode or program mode (signal SWMAINT is high). Signal MC-LD in turn, generates MC-IO, MC-1, MC-2, and MC-3 as in the power-on load sequence. If the system is in the maintenance mode, SWMAINT is low and generation of both MC-LD and LDCS-WR is inhibited. For this situation, MC-IO, MC-1, MC-2 and MC-3 are generated by MC-SW from the SYSTEM RESET pushbutton and the CS load condition is set up by SWCS-WR from the CS-WR position of the CONSOLE MODE SELECT pushbutton.

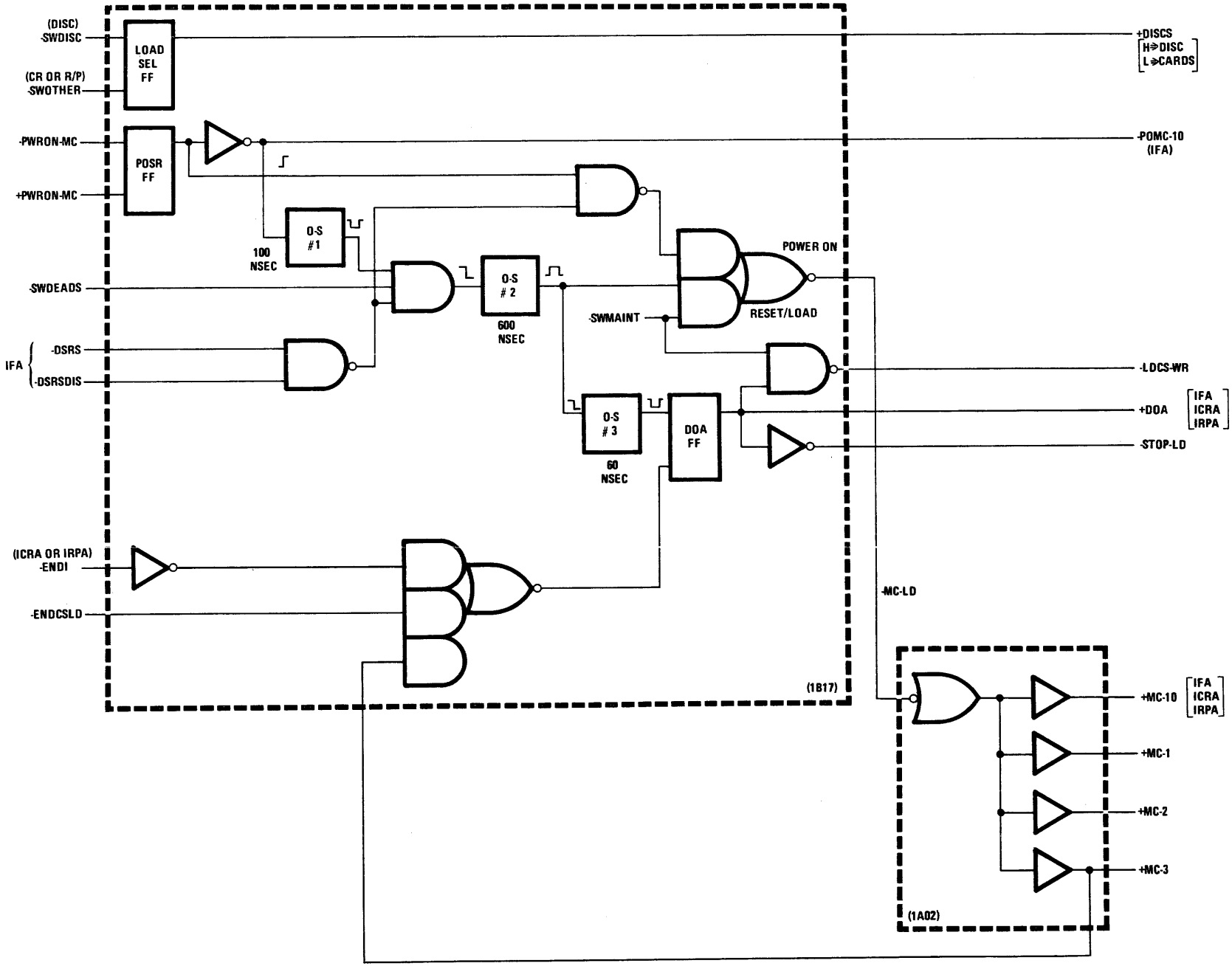


Figure 2-216. CS Load Initiate Logic

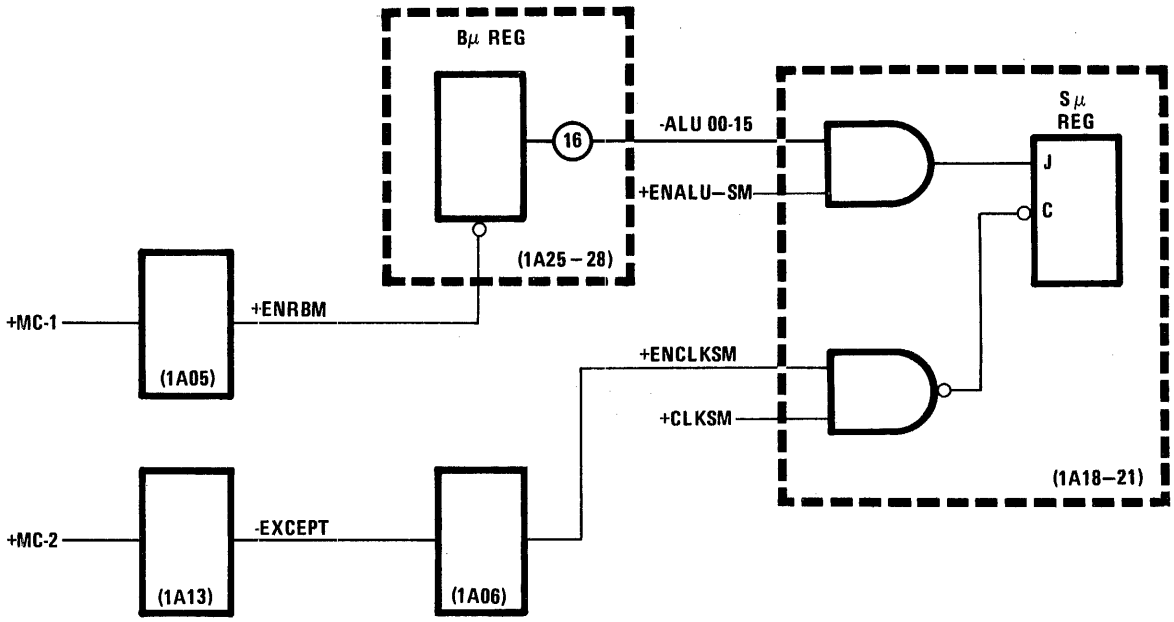


Figure 2-217. Formation of Address 0000_{16} in Sμ Register

CS Disc Command Load

The CS disc command load is initiated via execution of a CS Load disc command by the IFA in either the normal mode or maintenance mode. Executing this command generates DSRS (Dead Start Restart) in the IFA, which is routed back to the shared resources. Essentially, this signal provides a simulated setting of the RESET/LOAD pushbutton under software control. This signal is also generated upon detection of a burst check error in reading data (CS words) from the disc in the maintenance mode. Detection of such an error requires re-loading the CS data. For whichever reason, it generates DOA, MC-IO, MC-1, MC-2 and MC-3 in the same manner as pressing the RESET/LOAD switch. In addition, DSRS is routed to indicator 00 of the CONSOLE ADDRESS REGISTER DISPLAY on the Panel to indicate that the CS load is being performed as a result of either executing a disc command or detecting a burst error. Lighting this lamp, however, has real significance only upon detection of a burst error. If this condition occurs, DSRSDIS is generated to specifically inhibit generating DOA, MC-IO, MC-1, MC-2, and MC-3 by means of DSRS. This means that the system will stop upon detection of a burst check error. Re-loading CS must be re-initiated manually by means of the RESET/LOAD and SYSTEM RESET pushbuttons. Loading or reloading CS upon occurrence of the other three conditions that generate DSRS (disc command – normal mode, disc command – maintenance mode, and burst check – normal mode) will take place automatically.

Upon completing the initialization sequence in the disc IFA, the transfer of data from the IFA to the shared resources can begin. This is done by the logic shown in Figures 2-218 and 2-219. Figure 2-218 shows generation of signals which control the transfer of data and Figure 2-219 shows the logic involved in the data transfer itself. After receiving DOA from the shared resources, the IFA reads the first word to be stored in CS from the disc in serial fashion, assembles it in the IFA extended register, and sends DDS (Disc Data Strobe) to shared resources to inform it that the first CS word is available for transfer. The CS load control logic AND's DDS with SWDISC from the PRIMARY position of the AUTOLOAD SELECT switch to trigger one-shot circuit 1. This one-shot furnishes a pulse 60 nanoseconds in width to clear the console data register in preparation for receiving CS data from the IFA. The falling edge of this one-shot triggers one-shot circuit 2, which generates SEL EN. This signal, also 60 nanoseconds wide, is used with SWDISC on Figure 2-219 to generate an enable which is applied to four selector elements. These elements receive CS data from either the disc, via the sixteen ER13 bits, or the card reader, via the four ODI bits. When furnished with the corresponding enable, the elements gate data from the corresponding I/O device. In the case of the disc, the sixteen ERI3 bits are gated and passed to the Console Data register as SLSTEN bits. When clocked by CLKNR, the Console Data register passes the data to the data display fan-in logic. Selection of the Console Data register is forced by the CS write operation. Data from the fan-in logic is then routed to CS for storage at the address defined by the contents of the Sμ register. Initially, Sμ is loaded with 0000_{16} as discussed previously. For every subsequent word transfer, Sμ is updated by the Sμ+1 logic to form the address at which the next CS word will be stored.

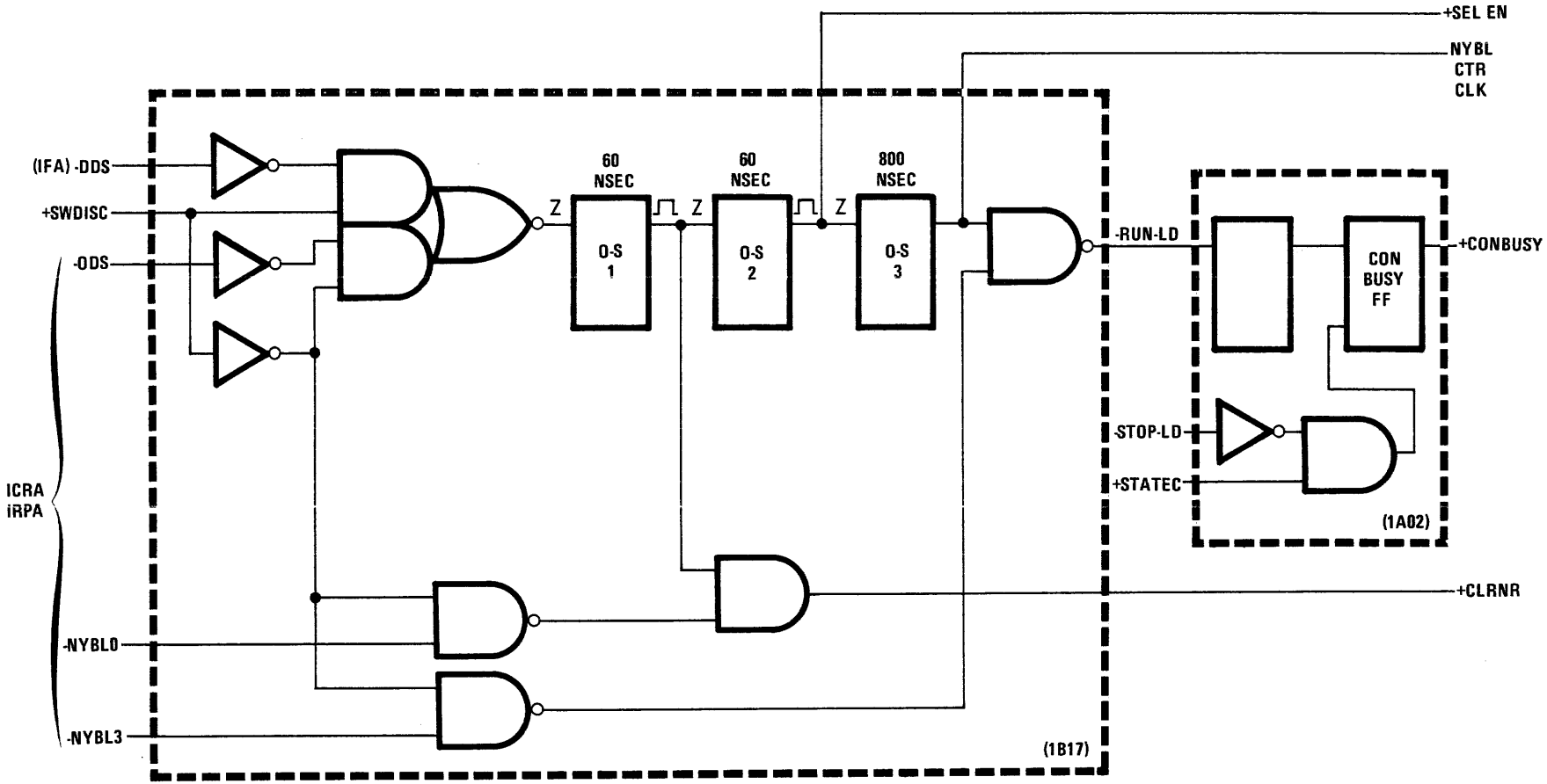


Figure 2-218. CS Load Control

During a CS load operation, data is transferred to CS from the disc in an asynchronous manner under control of a strobe generated for each word to be transferred. Essentially, the CS load can be characterized as a *start/stop* operation, where everything stops after a word is transferred until receipt of the strobe for the next word. The strobe required to transfer each word is the DDS signal. This signal is generated in the IFA for every word that the IFA assembles in its extended register. This signal, in turn, is used by the shared resources to generate RUN-LD which sets the Console Busy flip-flop, producing CONBUSY. Signal CONBUSY is routed to the RAN to set both the Console and Console State flip-flops. Setting these flip-flops generate the signals necessary to start the Sμ+1 update logic and store the data transferred to the Console Data register at the corresponding location in CS. Signal CONBUSY is analogous to the processor requests from the B/A register during normal operation in that it is used to obtain a time slice for the Panel. Upon completion of the single-word store, DDS is deactivated and the Console Busy flip-flop is cleared by the ANDed combination of STOP-LD and STATEC from the RAN. When the next word has been assembled in the IFA extended register, DDS is activated again and the above sequence of events is repeated. Each word transfer takes one major cycle to execute.

Transfer of data to CS will stop when one of the following four conditions occur:

1. a burst check error is detected,
2. the CS load is completed,
3. a system reset operation is performed, or
4. the system is shut down.

Detection of a burst check error is performed by the IFA and requires that the contents of CS be re-loaded. Indication of a burst check error is furnished by signal DSRS. This signal performs the same functions as if it was generated for starting a CS load via execution of a CS Load disc command, namely, generation of signals DOA, STOP-LD, MC-IO, MC-1, MC-2, and MC-3. When a burst check error occurs, however, the CS load operation is in progress and the DOA flip-flop is already set. To re-start the load operation, this flip-flop must be cleared and set again to initialize the disc heads. Clearing the DOA flip-flop is performed by MC-3, which is routed back to the clear side of the flip-flop as shown in Figure 2-195. After the 600-microsecond delay from one-shot circuit 1 is complete, the DOA flip-flop is set again to re-start the CS load operation. (Again recall that re-starting the CS load by means of a burst check error is inhibited if the system is in the maintenance mode, due to the presence of DSRSDIS.)

Determining that a CS load operation has been completed is performed by the CS load complete logic, shown in Figure 2-220. This logic determines that all data has been loaded in both the CS and FRJ decode address table (AT). Upon detecting this condition, the logic generates ENDCSLD. The logic essentially consists of five parts: the 1K-word detector, the CS present detector, the AT Selector flip-flop, the AT present decoder, and the AT present detector. During a CS load, the 1K-word detector monitors the state of bits 6 and 7 from Sμ to sense when Sμ reaches addresses of $X0FF_{16}$, $X2FF_{16}$ and $X3FF_{16}$, indicating that 256, 512, 768, and 1024 words, respectively, have been loaded. The states of bits 6 and 7 are ANDed with X-00FF, indicating that bits 8 through 15 are all 1-bits. This progress of address detects in 256-word increments as shown in Figure 2-221. When a load of 1024 words in CS is detected, signal 1K DET is generated and routed to one side of gate A.

The other side of gate A is fed with an output from the CS present detector. This detector performs a dual function of checking for CS loads in 1024-word increments up to 16,384 words (four CS storage units), and checking to see if a portion of CS addressed by Sμ is actually present in the system. The latter check is necessary since the Sμ+1 logic has no way of knowing whether Sμ has been updated past a CS location not present in the system. Indication that another 1024-word increment of CS is going to be loaded is provided by bits 2 through 5 of S . The states of these four bits are ANDed with four CSEQ bits, the encoded result of which represents hexadecimally the maximum number of 1024-word portions of CS present in the system. (For example, if the system contains two CS storage units (8192 words), the encoded CSEQ bit result will be CSEQ1000.) This progression of address detects in 1024-word increments ANDed with the corresponding CSEQ bit result is also shown in Figure 2-221. When a match is detected, signal CS PRES DET is generated. Note that generation of this signal is not dependent on the states of Sμ bits 6 through 15; therefore, the signal indicates only that the first location of the last 1024-word portion of CS present in the system has been addressed. Indication that this portion of CS has been completely loaded is furnished by signal 1K DET. When these two signals occur simultaneously, gate A is enabled and sets the AT Select flip-flop. The set side of this flip-flop generates AT-SEL and the clear side feeds one input to gate B used to generate END0 and ENDCSLD.

Signal AT-SEL enables the FRJ decode address table to be loaded with data from the disc. This is accomplished in basically the same manner as the CS load complete operation: determining the number of words loaded per address table and combining this information with the number of address tables present in the system.

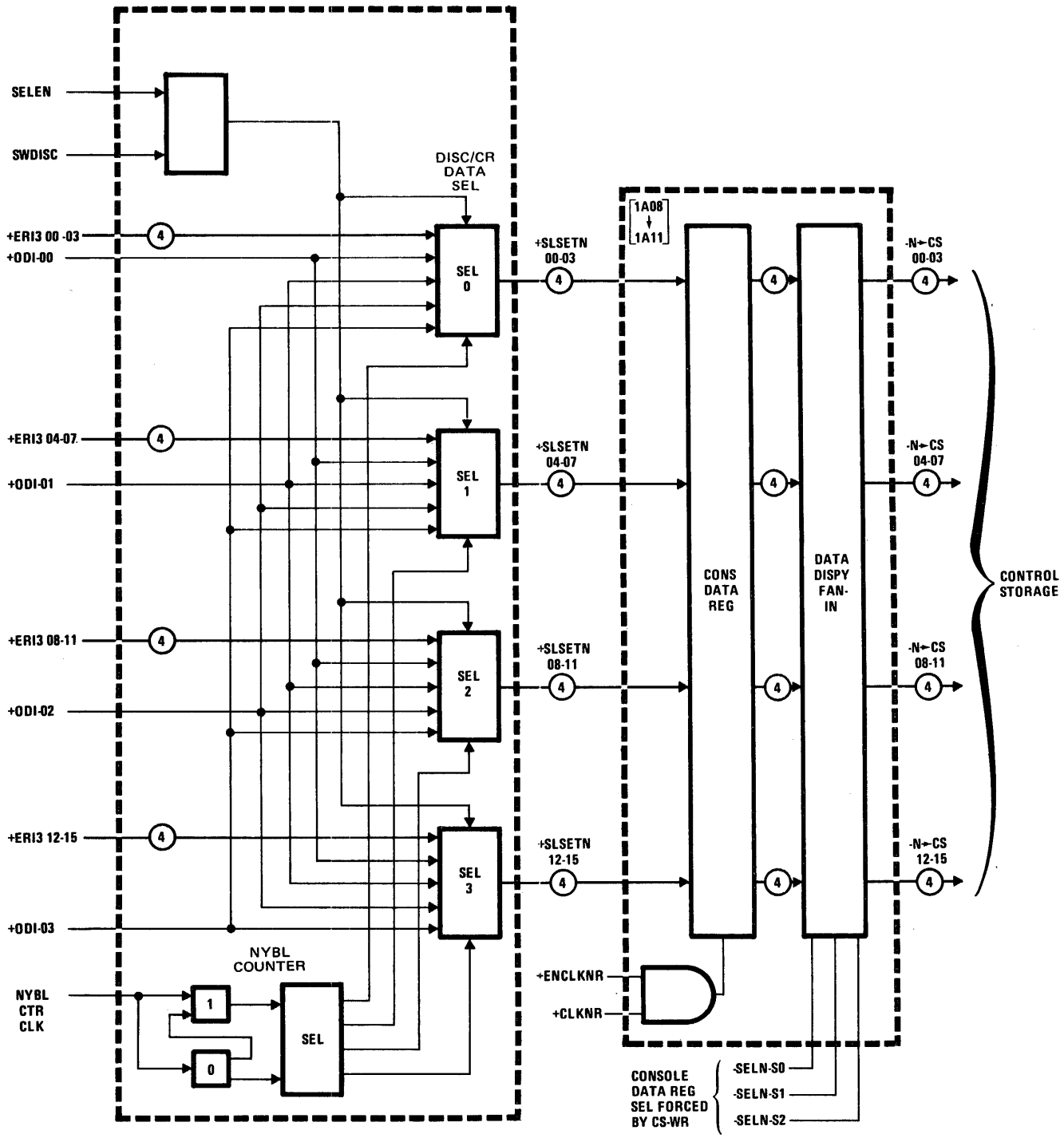


Figure 2-219. CS Load Data Transfer

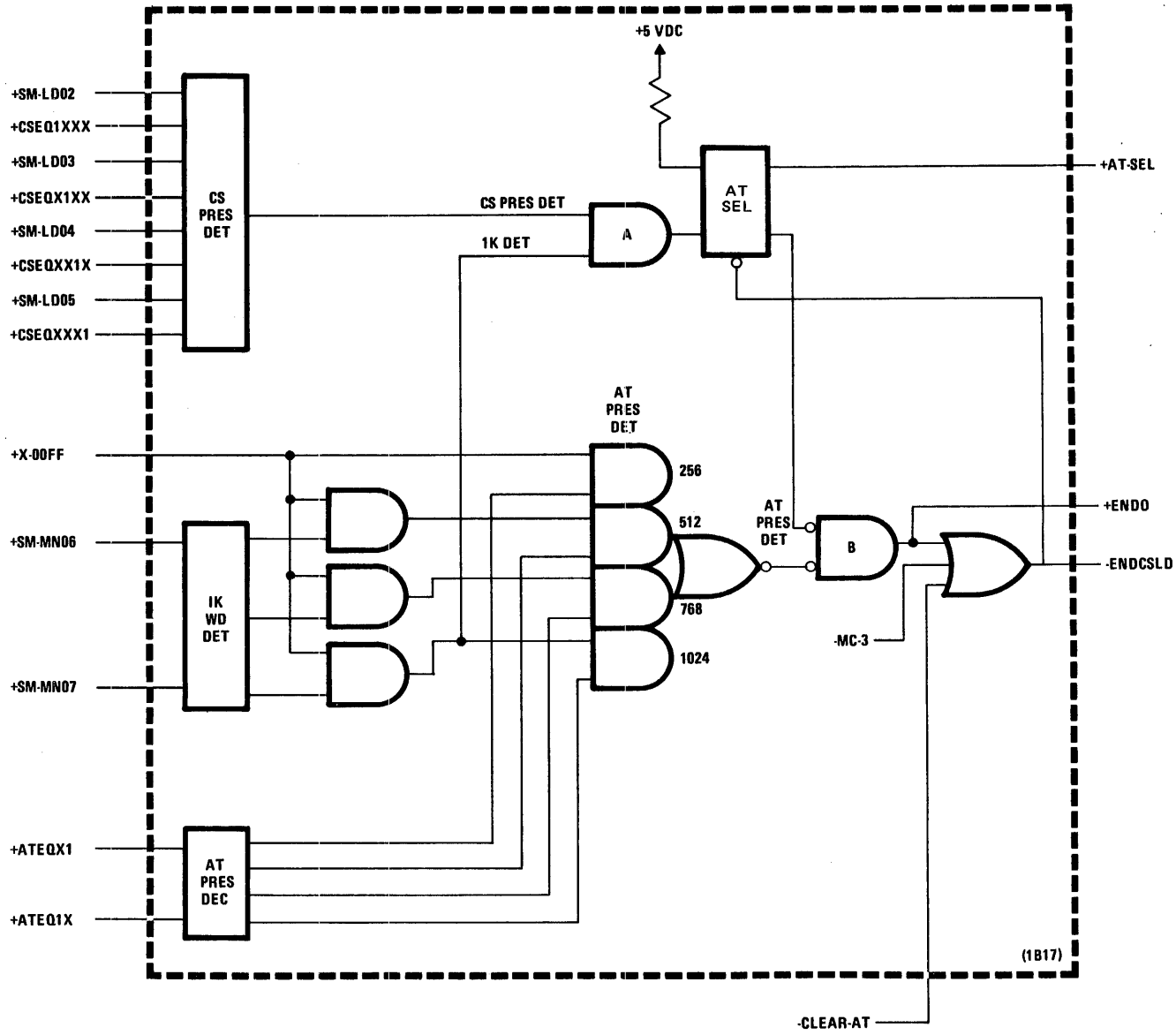


Figure 2-220. CS Load Complete Logic

Determination that an address table has been loaded with 256 words is accomplished by signal X-00FF. This signal is combined with those from the 1K word detector and the AT present decoder by the AT present detector. The 1K word detector determines which of the four (maximum) address tables is being addressed by bits 06 and 07 of Sμ. This information is combined with outputs from the AT present decoder, which determines from two ATEQ bits how many address tables are present in the system. The results of these two decoders are combined with signal X-00FF to enable one of four NOR gates, making up the AT present detector, as shown on Figure 2-221. This result, AT PRES DET, is combined with the output from the CS Load flip-flop to generate ENDO and EDNCSLD. Signal ENDO is fed to the IFA, informing it that the CS and AT load operations have been completed. Signal ENDCSLD is routed to the CS load initiate logic (Figure 2-216) to clear the DOA flip-flop, thus deactivating signal DOA to the IFA. This completes the disc CS load operation.

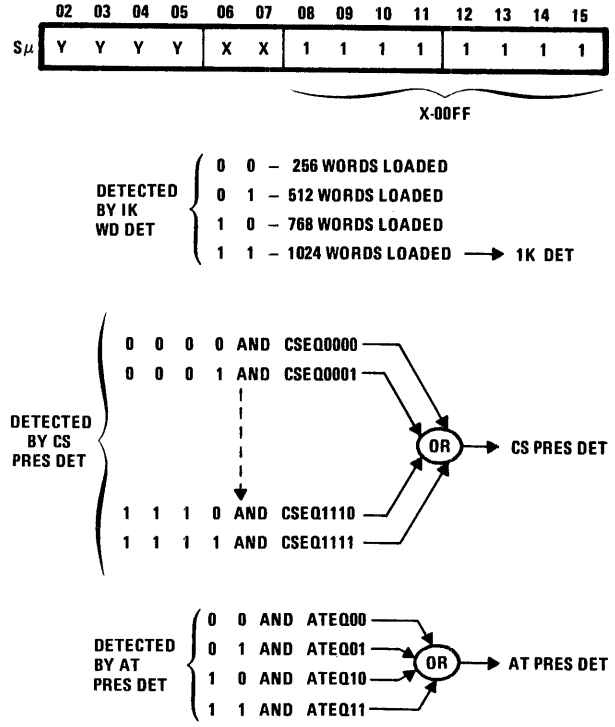


Figure 2-221. CS Add and CS Present Compares

Card CS Load

A CS load from cards may be performed by either a power-on system reset condition, or by pressing the RESET/LOAD pushbutton. Cards may be read from either the card reader or the reader/punch, as determined by the setting of the LOAD SELECT switch on the Panel (CR position for card reader and R/P position for reader/punch). The initiate part of a card CS load is very similar to that for a disc CS load. The only differences are that the LOAD SELECT switch must be set to one of the two settings described above and that signal MC-IO is not used. Referring to the CS load initiate logic of Figure 2-216, signal SWOTHER is generated by setting the AUTOLOAD SELECT switch to either the CR or R/P position. This signal clears the Autoload Select flip-flop which causes DISCS to go low. A low DISCS signal enables the selected card device for performing the CS load operation. Activating the RESET/LOAD pushbutton generates DOA and MC-LD which, in turn, generates MC-IO. Signal DOA is routed to either the Integrated Card Reader Adapter (ICRA) or the Integrated Reader Punch Adapter (IRPA) to set up logic for assembling the first word to be transferred to CS. Although generated, signal MC-IO is not used by the ICRA or IRPA since there are no pick-up heads to be positioned in the card reader device as in the disc.

Data transfers from the ICRA/IRPA to the shared resources differ from those from the IFA because data transferred from the ICRA/IRPA is done so in nybl form, four bits at a time, instead of in whole word form. This requires four separate data transfers, one per nybl, to assemble a complete word in the Console Data register prior to storing it in CS. These four data transfers are enabled by the output of a two-bit nybl counter, which generates four counts (00, 01, 10, and 11) in sequence. The nybl counter consists of two flip-flops, labeled 0 and 1, as shown in Figure 2-219. When clocked by signal NYBL CTR CLK, the nybl counter enables a nybl on the ODI lines to pass through a selector to the Console Data register. Signal NYBL CTR CLK is generated by ODS (Output Data Scan) from the ICRA. This signal serves a similar purpose as signal DDS from the IFA, namely, to initiate each nybl data transfer from the ICRA to the shared resources. The four counts enable four nybls through selector elements 0 through 3 in sequence, as shown in Figure 2-222. During each four-nybl transfer, signals NYBL0 and NYBL3 are sent to the CS load control logic. Signal NYBL0, sent concurrent with the transfer of nybl 0 to the shared resources, generates CLRNR to allow the next four-nybl word to be assembled in the Console Data register. Signal NYBL3, sent concurrent with nybl 3, generates RUN-LD which in turn generates CONBUSY. This signal sets up the RAN to write the assembled word into CS. Once assembled in the Console Data register, the resultant word from the ICRA is stored in CS in the same manner as a word from the IFA during a disc CS load.

Termination of a CS load from the ICRA is accomplished by means of the ENDI signal from the ICRA, which informs the shared resources that all the cards have been read. This signal clears the DOA flip-flop in the same manner as ENDCSLD clears the flip-flop at the end of a CS load from the disc.

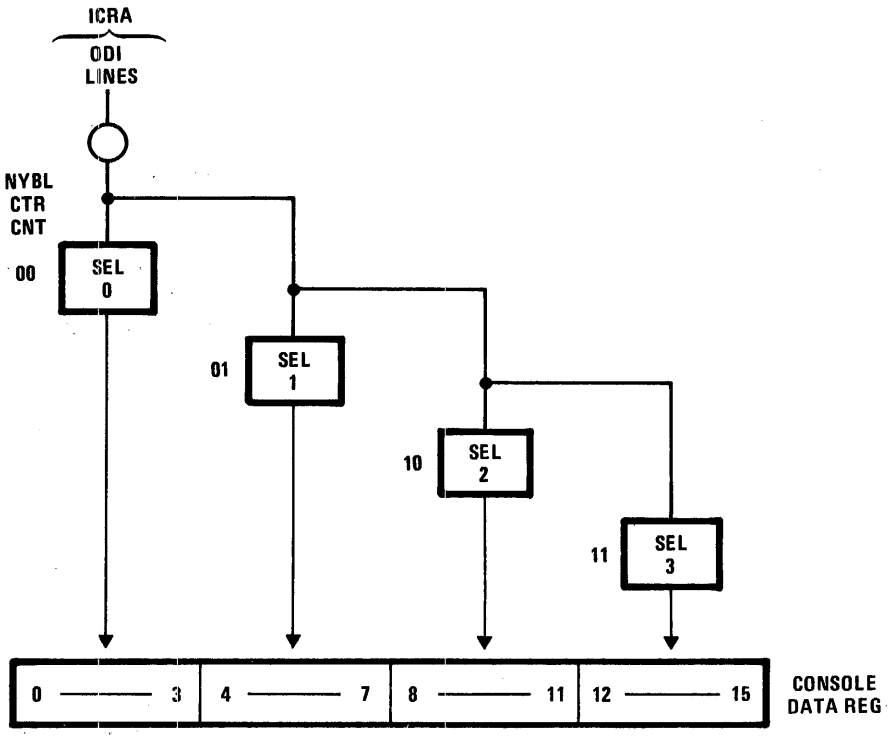


Figure 2-222. Transfer of ICRA Nybl Data to Console Data Register

Autoload

The autoload operation loads the operating system and user programs into Main Storage (MS) upon completion of the CS load. Either disc or card reader may be used to load MS.* Unlike the CS load, which is loaded in CS in sequential addresses under hardware control, the MS load is under control of a Disc Autoload or Card Reader Autoload routine stored as part of the CS load. These two routines turn control of the MS load over to the Executive processor, which controls the placement of various routines in MS in accordance with their use. If the system is in either the operator mode or program mode, and CS was loaded by a power-on or reset/load condition, the autoload operation begins automatically upon completion of the CS load operation. For any other condition, the autoload operation must be initiated manually by means of the AUTOLOAD pushbutton.

*At present, only the disc may be used to load MS via the autoload operation.

Control signals generated during an autoload operation are shown in Figure 2-223. These signals are initiated by either SWAUTO from the AUTOLOAD pushbutton on the Panel or AUTO-LD from the disc IFA. These two signals are ORed together to set the Autoload flip-flop, which removes the effects of switch bounce from the AUTOLOAD pushbutton. Setting this flip-flop triggers two one-shot circuits in sequential order. One-shot circuit 1 generates a negative pulse of 4 microseconds in width to set the System Reset flip-flop. This flip-flop is used to generate MC-1 and MC-2, which are used to clear the Sμ register in the manner shown in Figure 2-195. One-shot circuit 2 generates a 0.1-microsecond-wide negative pulse which sets the Request 4 and the Request Enable flip-flops. The Request 4 flip-flop generates REQ-4 through one side of an OR gate to set the Busy 4 flip-flop of the Busy/Active register. This flip-flop output, in turn, is used to obtain time slices via the RAN to effect the transfer of MS data. The set output from the Request 4 flip-flop is also used in combination with SWOTHER from the CR and R/P positions of the LOAD SELECT switch to generate the starting address of the autoload routine by

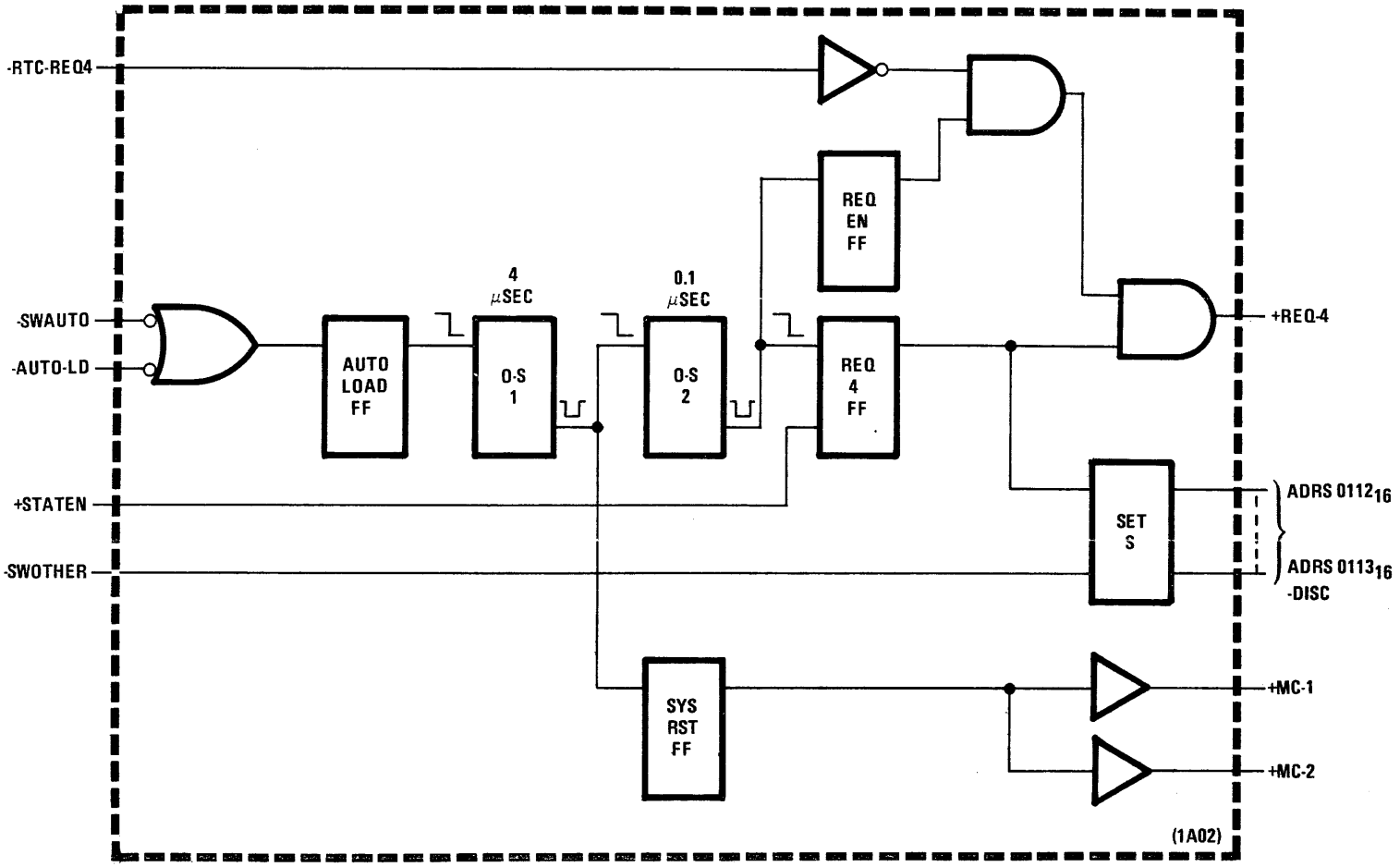


Figure 2-223. Autoload Control Logic

means of the set S logic. If loading from the disc, SWOTHER is not present and the starting address formed is 0113_{16}. If loading from a card device, SWOTHER is present and the starting address formed is 0112_{16}. The Request 4 flip-flop is cleared at E000 of the next minor cycle by STATEN from the Null State flip-flop in the RAN. Signal REQ-4 may also be generated by the normal executive request from the Real Time Clock (RTC) register. This request (RTC-REQ 4) is generated every 16.384 milliseconds when the RTC register overflows and is enabled by the set output from the Request Enable flip-flop.

Upon generating the starting address of the autoload routine, transfer of data from either disc or cards begins under control of I's in the autoload routine. The basic flow of data from the input device (disc or cards) to MS is shown in Figure 2-224. All enables are generated by the μI's of the autoload routine.

REGISTER SELECTION/DISPLAY

Certain registers of the shared resources and the ERF Group II may be selected for displaying their contents by means of hardware alone, in contrast to the software-controlled RF read routine described in the paragraph titled MS/RO and RF Read and Write. These registers are selected by the CONSOLE ADDRESS REGISTER SELECT and CONSOLE DATA REGISTER SELECT selectors on the Panel. The CONSOLE ADDRESS REGISTER SELECT selector permits display of address-related data contained in the S, Sμ, Console Address, and PE registers. The CONSOLE DATA REGISTER SELECT selector permits display of data-related information contained in the RTC, Fμ-2, Fμ-1, CS Scan, B/A, Console Data, D, Aμ, Bμ, and BC registers, plus the sum of Aμ and Bμ.

Logic for displaying data-related information in the CONSOLE DATA REGISTER DISPLAY indicators is shown in Figure 2-225. Each position of the CONSOLE DATA REGISTER SELECT selector is fed to one of two encoder circuits, depending on whether the register is associated with the ALU in the shared resources (D, Aμ, Bμ or sum of Aμ and Bμ) or with the ERF Group II (B/A, RTC, BC, CS scan, Fμ–1, or Fμ–2). The ERF Group II register positions are sent to an eight-input encoder, which generates a three-bit register address on the three SELN lines. This encoder is also fed with an overall ALU register select signal (SWSELALU) which is generated when any of the four ALU register quantities is selected. The three SELN lines are sent to the data display selector. This selector is fed with the 16 lines from each ERF Group II register selected by the CONSOLE DATA REGISTER SELECT selector plus 16 lines from the ALU fan-out logic, fed with inputs from the four ALU-associated registers. Each position of the Console Data register selector generates a three-bit register address as shown in Table 2-27 to select its corresponding rester for display. Note that the address generated for the four ALU-associated registers is the same for all. This is a result of signal SWSELALU, which is generated when any of the four ALU-associated registers is the same for all. This is a result of signal SWSELALU, which is generated when any of the four ALU-associated registers is selected. These registers are selected by their own encode logic because of the necessity of displaying their contents during a null condition only. Furthermore, the contents must be the results of only one of the eight processors running and then in the stop mode. These restrictions are necessary so that meaningful (non-changing) data may be displayed. The null restriction is implemented by signals READNULL and STATEN from the RAN. The result is to generate SELDISPY, which generates a corresponding enable to gate the contents of the selected ALU register to the selector. The selector output is sent to a second selector which is used during the CS scan and CS read operation to display the contents of either CS or the FRJ decode address table (AT). For console data display operations, signal AT-SELAJ will be high to gate data from the selector (N → CS) to the CONSOLE DATA REGISTER DISPLAY indicators.

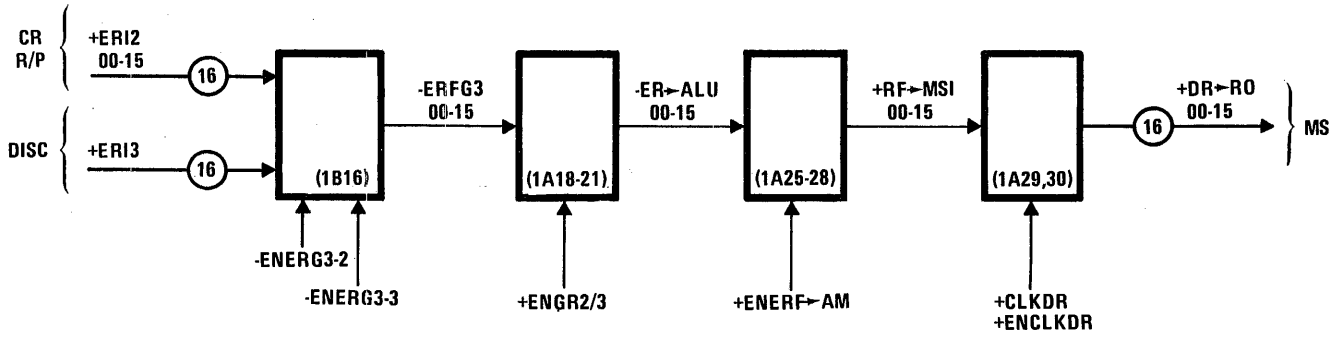


Figure 2-224. Autoload Data Storage

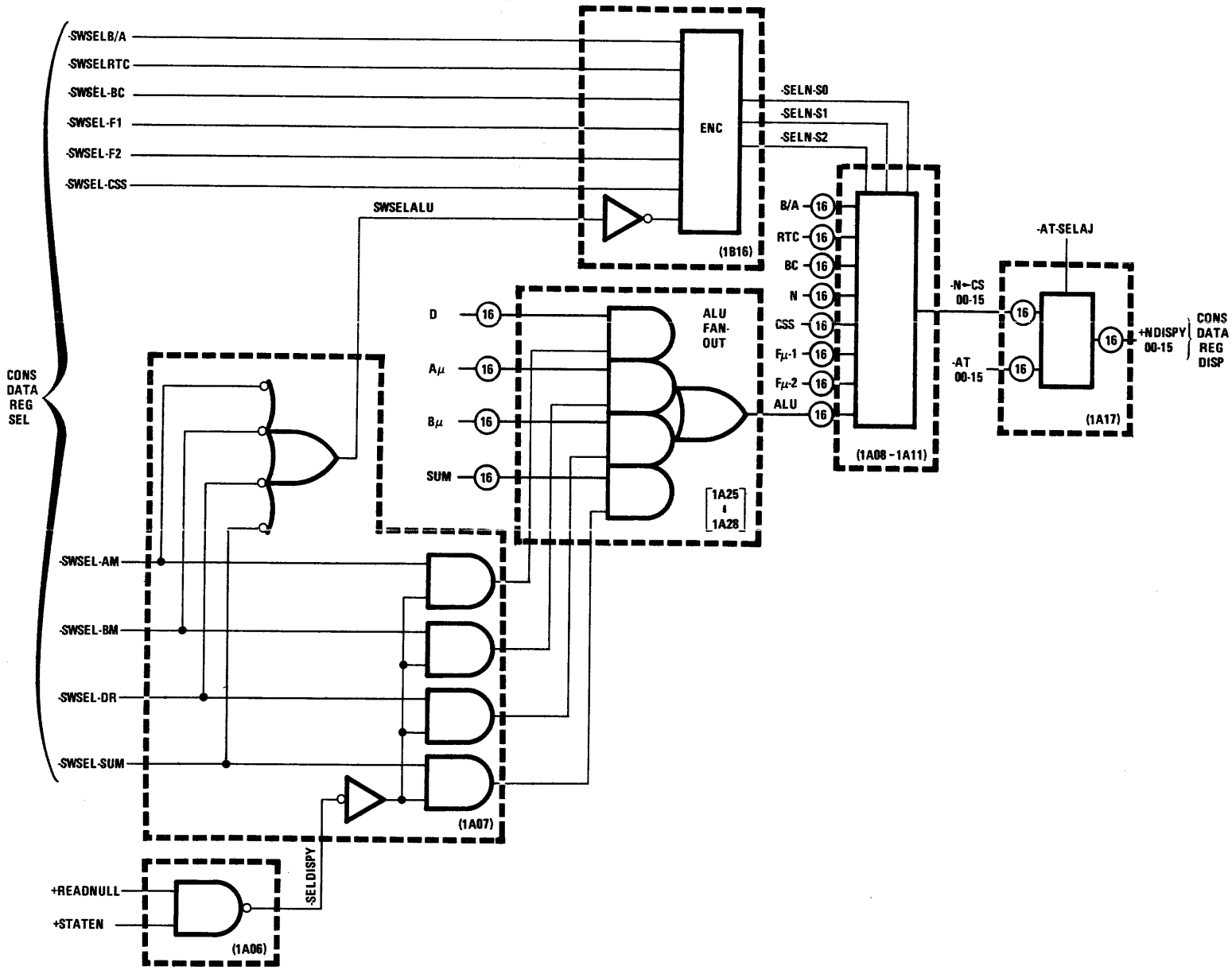


Figure 2-225. Console Data Display

Table 2-27. Console Data Register Selectors

Selector Setting	Select Signal States		
	$\overline{\text{SELN-S2}}$	$\overline{\text{SELN-S1}}$	$\overline{\text{SELN-S0}}$
B/A	1	1	1
CSS	1	0	0
Fμ1	0	1	1
Fμ2	1	0	1
RTC	1	1	0
BC	0	0	1
SUM	0	0	0
Bμ	0	0	0
Aμ	0	0	0
D	0	0	0
N	0	1	1

Logic for displaying address-related information in the CONSOLE ADDRESS REGISTER DISPLAY indicators is shown in Figure 2-226. The four positions of the CONSOLE ADDRESS REGISTER DISPLAY selector are fed to an encoder, which generates a two-bit register address on the two $\overline{\text{SELM}}$ lines for each register selected as shown in Table 2-28 These lines are fed to a driver and then to the address display selector that is fed with the 16 lines from each address-related register. The output from the selected register is routed to the CONSOLE ADDRESS REGISTER DISPLAY indicators on the Panel over the MDISPY lines.

Table 2-28. Console Address Register Selector

Selector Setting	Select Signal States	
	$\overline{\text{SELM-S1}}$	$\overline{\text{SELM-S0}}$
Sμ	1	1
S	0	1
M	1	0
PE	0	0

The Fμ2, Fμ1, RTC, CSS, D, Aμ, Bμ, SUM, and BC positions of the CONSOLE DATA REGISTER SELECT selector and the Sμ and PE positions of the CONSOLE DATA REGISTER SELECT selector are enabled only if the Panel is in the maintenance mode (MAINTENANCE MODE pushbutton set to on). These selector positions are disabled if the Panel is in either the operator mode or the program mode by appropriate grounding of the selector lug corresponding to the position on the Panel itself.

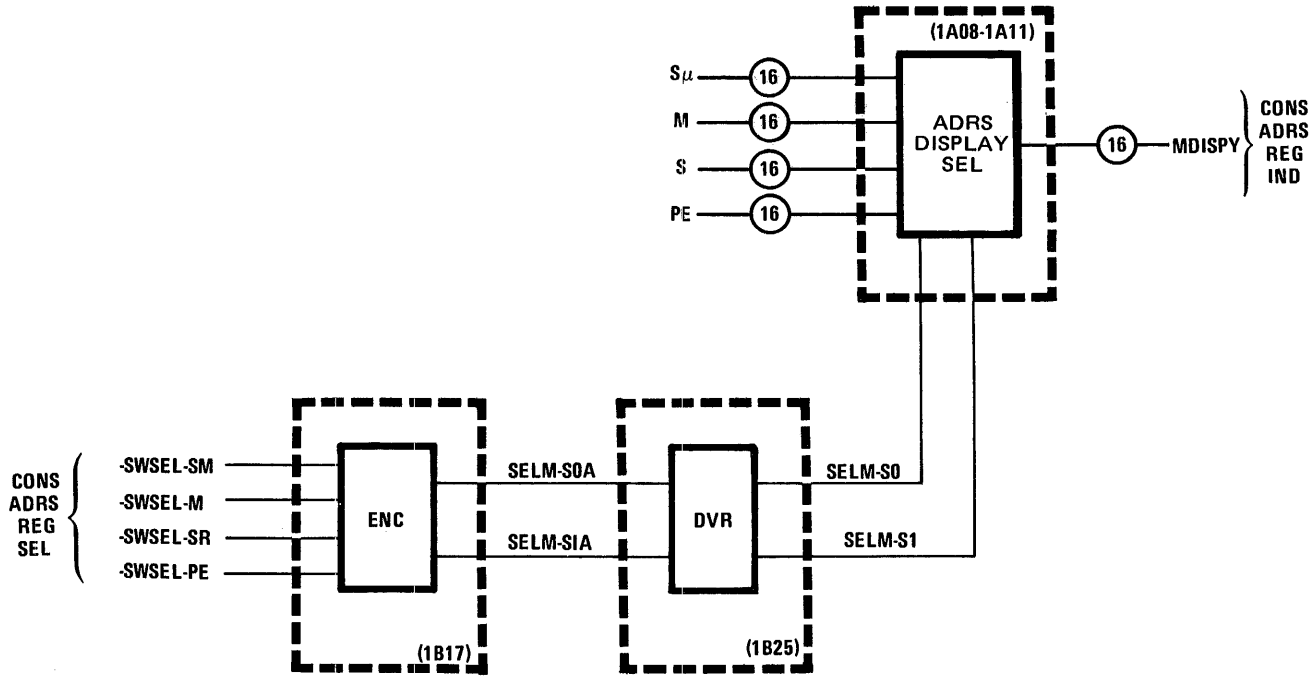


Figure 2-226. Console Address Register Display

APPENDIX 2A

INTRODUCTION TO ERROR CORRECTION CODES

In recent years, the demand is for increasingly sophisticated error detection and correction schemes to improve reliability as judged by performance, cost and size. The objective here is to give an insight to the idea (not the theory), some terminology, and a comprehensive but simplified example of what error correction means. The basic assumption is that all hardware has an intrinsic failure rate, however small, so that by minimizing the hardware in an entire system will tend to lower the system failure rate. The design objective is to provide, with minimum hardware, a redundancy coding that will combat statistically independent single errors. Statistically independent single errors means that about 99% of the time, random or intermittent errors will occur only one at a time. Over a long period of time the same identical error will not occur. A shorted diode for example, that fails every time it is used is not an independent error. Noise injected into the transmission medium tends to be random in nature and therefore redundant coding is used to combat it. To implement this objective, many random-error-correcting codes have been used. For computer applications, variations in the Hamming Single-Error-Correction (SEC) and Double-Error-Detection (DED) are the most useful.

Most single error correction coding was originally designed for bit serial transmission of data over a radio link where atmospheric noise is rather unpredictable. By sending redundant bits with the message, the errors caused by atmospheric noise can be overcome. Whether it was a data bit or a redundant bit doesn't matter because the coding allows a single error to occur and still recover from it. The basic assumption in a Hamming error code is that data transmission is done bit serially. For bit parallel memories, modifications to the coding hardware is necessary. The coding hardware simultaneously calculates parity on two bytes to minimize the calculating time required. Random access bit-parallel storage have used parity bits as their basic error detection scheme. Typically a parity generator calculates how many binary one bits there are in the word to be stored. Then a parity bit is generated to make the total summation of word data bits and the parity bit to be an odd number of one bits. The result is then stored. Upon reading this word from storage, the summation of one bits is checked to verify an odd number of ones. Any errors result in an interrupt to the computer warning it of an unidentifiable failure. To find the failing bit requires some redundance coding scheme not available with parity. The extreme redundancy code would be a bit for bit duplication of the original data word (if the duplicate word is known to be sent correctly). Using mathematical theorems, the number of duplicate bits, or check bits, can be minimized.

Table 2A1 shows an example of one method of producing redundancy for correcting errors. For the information shown in the three rows, the row and column parity is calculated and shown to make odd parity.

Once the row parity and column parity is calculated, the check on row parity is calculated and is shown in the lower right-hand corner of the matrix. The array of information can now be transmitted bit-serial over the radio link and then staticized by the receiver into the original format. If an error occurs, it will show up in both the row and column parity checks. The bad parity checks for the row and column will intersect at the erring information bit so that it can be corrected. In other words, a single error was detected by the check bits (row and column parity) and it was correctable by pinpointing the bad information bit.

Suppose that a double information error had occurred. If one error was in Byte 1 and the other in a different column of Byte 2, there would be two failing columns and two failing row parity bits. In this case a double error could be both detected and corrected if the parity bits are assumed to be correct. However, had both failing bits occurred in the same byte, there would be two erring column parity but no erring row parity. The double error is now detected but not necessarily corrected because one of the errors may have occurred in the check bits. Generally, error detection is logically easier to implement than error correction. For this reason, correcting single errors and only detecting multiple errors is the most common redundancy coding.

In the example, the code word consisted of 20 symbols of which 12 were information and 8 were check symbols. The check symbols provide the code word with error-correcting capability.

The example shows how redundant check bits can reconstruct a single error. To adopt this example to a useful code for bit parallel storage requires doing the coding simultaneously. Using three parity generators to calculate the three row-parity bits and five parity generators to calculate the five column parity bits, the entire matrix can be simultaneously generated. The one exception in the example is the column parity bit that is generated from the row parity bits. It must wait until the three row parity bits are generated. Finally, the entire code word of twelve information bits and 8 parity (check) bits can now be stored. When the code word is read from storage, similar simultaneous logic can detect and possibly correct the error.

Table 2A1. Odd Parity Example

	Information	Row Parity
Word 1	1 1 1 0	0
Word 2	1 1 0 0	1
Word 3	0 1 0 1	1
Column Parity	1 0 0 0	1

To minimize the logic and the number of check bits, many coding schemes have been developed. The easiest method of minimizing logic is to construct a table as shown in Table 2A2. The table shows the relationship for constructing the check bits from the information bits. Now however, the matrix of "X" is in the form that mathematical theorems can be used to minimize the redundant check bits. From this example, the technique of forming error correction codes is shown. After the mathematical theorems minimize the table, the new table can be used to satisfy the construction of the logic.

Variations and modifications to the Hamming SEC-DED codes have resulted in codes superior in cost, performance and reliability. The parallel generation of all check bits minimizes hardware and increases speed. For most codes, the capability or probability of correcting a single error and detecting all multiple errors can be empirically determined.

Table 2A2. Formatting the Example into a Table

	Byte 1				Byte 2				Byte 3				Row Parity Bit		
	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2
Row Check Bit 1	X	X	X	X											
2					X	X	X	X							
3									X	X	X	X			
Column Check Bit 1	X				X				X						
2		X				X				X					
3			X				X				X				
4				X				X				X			
5													X	X	X

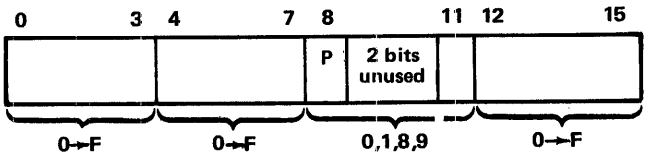
3. MICRO-INSTRUCTION REPERTOIRE

GENERAL

The micro instructions (μ's) are 14-bit codes stored in control storage (CS) and are used to implement the execution of machine-language instructions (MLI's) and to perform special manipulative routines initiated by the operator from the System Control Panel. There are 65 basic μI's, grouped into 10 classes, comprising the repertoire. Some of these basic μI's, such as the FNJ μI, can be executed in one of two ways depending on whether or not a certain bit of the μI is set. Each μI consists of 14 bits, arranged in a 16-bit format such that bit positions 9 and 10 are not used and are always in the clear state. For the most part, μI's enable inter-register transfers of data and address information. Some μI's, however, are used to access main storage (MS) or exercise control over a programmed operation.

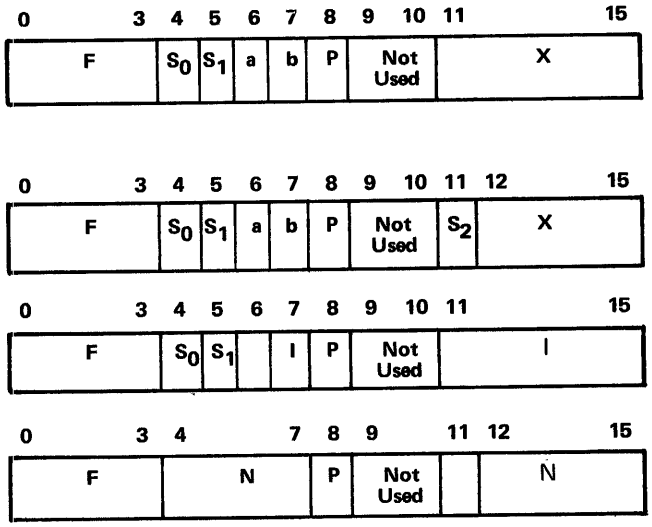
FORMATS

The 14-bit μI's (one bit of which is a parity bit) are read from CS and deposited in the 16-bit Fμ register, as they are needed. Since unused bit positions 9 and 10 are always 0, and the parity bit P (bit 8) is carried along with the μI instead of being generated separately, it is possible to express each μI as four hexadecimal characters as shown below:



Except for the hexadecimal character represented by bit positions 8 through 11 of the μI, each character can assume values of 0 through F depending on how the μI is coded. Bit positions 9 and 10 of the remaining character are always defined as 0's; therefore, this character can only assume values of 0, 1, 8, and 9.

The μI's are formatted in several different ways, depending basically on their particular function. The format used for each μI is shown with the description of that μI in the paragraphs that follow. All formats, however, use some or all of the field designators shown in the following four formats. Explanations of these designators follow the illustrations.



Field Designator	Meaning
F	The basic function, or operation, code
P	Parity bit (odd parity is used)
N	An 8-bit operand (two fields, bit 04 is the MSB)
I	A 6-bit jump index (two fields, bit 07 is the MSB)
X	A register designation, skip designator, a mathematical constant, or a hexadecimal value indicating a bit (one of 16) to be set, cleared, or toggled. Bit 11 is not used (is a "0") for the latter.

If bit 11 is a "zero" when X is used as a register designation, the register specified by bits 12-15 will be one of 16 in the lower half of the basic register file (i.e., registers 0-15). If bit 11 is a "1", then the register is one of 16 in the upper half of the basic file (registers 16-31).

S_0,S_1,S_2 Sub-operation codes

a,b These designators determine how the number of the register to be operated on is derived. If both a and b are "0", the X field designates a basic register. If both are "1", the X field designates an extended register. If either a or b is a "1", the register number is derived by performing an inclusive OR between the lower three bits of X and the 3-bit R_1 or R_2 field of the machine-language instruction, as the case may be, and that number specified a basic register. For the SKB, SKB-, LBB and LBB- μI's, the machine OR is performed between the lower four bits of X and the corresponding 4 bits of either the R_1 or R_2 field (including the indirect designator) of the machine-language instruction.

CHARACTERISTICS

In addition to grouping μI's into classes according to similarities in execution, μI's can also be grouped into more general categories according to certain basic characteristics that cut across sub-division by class. These categories are discussed below.

REGISTER ADDRESSABILITY

Since most μI's can address registers of either the Basic Register File (BRF) or the Extended Register File (ERF), it is often convenient to know which μI's are the exception. Furthermore, of those that can address the ERF, it is convenient to know which of the three groups making up the ERF can be addressed by a particular μI. This register addressability information is listed in Table 3-1. This table lists the capability of each μI to either read or write a register of the ERF. The letters "R" and "W" are used to indicate *read* and *write* operations, respectively. Any μI that can read or write a register of the ERF can also read or write a register in the BRF.

BLOCKPOINT μI'S

A micro program block is a series of μI's that must be executed in the same major cycle (800 nanoseconds) if the results of the data manipulations are to be valid. The last μI in the block, must then be one that stores data in a dedicated resource, and ensures that data is not lost in the shared resources. These μI's are called blockpoint (BP) instructions. Each time the hardware detects a BP μI, it remembers the BP address +1, so that the program can resume at the proper location on the next major cycle. This is accomplished by routing the output of Sμ+1 to the P_p register by means of the BP μI. At the end of the major cycle, the contents of P_p is transferred to Pμ as part of the W portion of the time slice.

All branch, skip, and register file write μI's are BP μI's. These μI's usually occur near the end of a time slice as a result of their intended use. Therefore, they are suitable for performing the BP function since any μI occurring after the BP μI in the present time slice will be repeated during the next time slice. Because time slices always begin by reading the μI following a BP μI, the microprogrammer must be sure that a BP μI occurs at least once during every time slice. Blockpoint μI's are also tabulated in Table 3-1.

FEEDER LOAD μI'S

A feeder load μI is one which loads data into either or both Aμ or Bμ (feeder) registers. As such, they inhibit execution of a μI that uses the results of this data (such as a SUM or CMP μI) for 100 nanoseconds following the feeder load μI to allow sufficient time for the data to propagate through the ALU (refer to the paragraph on Cycle Delay Logic). The feeder load μI's, listed in Table 3-1, generally have the following properties:

1. cause full execution time of 200 nanoseconds (100 nanoseconds null time plus 100 nanoseconds execute time) when immediately preceding a 2,X (SUM, DSUM, CMP or CMU) μI.

2. clear inhibit on inner carries. Referring to Table 3-1, the following anomalies to the above properties should be noted:

   a. The DIG and CORC μI's inhibit inner carries as part of their execution. Therefore, they do not clear the inhibit on inner carries as do the other feeder load μI's.

   b. The shift (SHF, SHR, DLS, and DRS) μI's cause full execution time on the 2,X μI's even if the shift count equals zero so that the Aμ and Bμ registers are not altered.

   c. The bit sense (SRO and SS1) μI's cause full execution time on the 2,X μI's even if the Bμ register is not incremented.

I/O INTERFACE μI'S

Micro instructions which either read or write a Group III register in the ERF are referred to as I/O interface μI's. Besides reading or writing the Group III register in the I/O processor, these μI's also furnish a read or write status signal to the control logic, which can also read or write these registers in addition to μI's. This is necessary because the read and write control logic is under hardware control and cannot otherwise determine that a particular register has been read or written by a μI.

It is the responsibility of the microprogram to insure that whenever a Group III register is read by executing an I/O interface μI, a subsequent blockpoint μI which stores the data in the shared resources file will be executed in the same major cycle.

PμWRITE μI'S

Aside from the BP μI's, which write a starting μI address into Pμ from P_p the following four μI's write and Pμ as a part of their execution: CLR, STA, STB, and AND. Since the only path to Pμ is from Sμ via P_p, these μI's cause a full 14-bit address branch to another μI routine. The two status bits, Overflow and Link, that are also carried along with the 14-bit branch address are under hardware control only. Therefore, they cannot be altered directly but only by means of an arithmetic operation.

RESYNC μI'S

Execution of some μI's require that the following μI start at the beginning of the next time slice. An example of such a μI is the RNI (Read Next Instruction) μI, which causes a branch to a routine that reads the next MLI from MS and decodes it to determine its format. These operations can always be executed in one time slice; therefore, the RNI μI idles to the end of the present time slice to assure that the first μI of the RNI routine will be executed at EO of the next time slice. These μI's that cause the following μI to start at the next EO are called resync μI's, and are listed in Table 3-1.

TIMING CONSTRAINTS

Several μI's are subject to particular timing constraints in their execution. Usually these constraints prevent the μI from being executed during certain minor cycles of a time slice (usually E0 or E7) or, conversely, force the μI to be executed at only a particular minor cycle. For other μI's the constraint increases the execution time from one to two minor cycle, depending either on the preceding μI executed or when the time-constrained μI was executed during the time slice. Applicable timing constraints for each μI are discussed in the paragraph which describes each μI. They are also summarized in this paragraph for convenience. The first category of timing constraints is summarized in Figure 3-1. These constraints must be implemented by the micro-programmer when preparing the μI program. The second category of constraints is also summarized in Figure 3-1 and in the items below:

1. The SUM, DSUM, CMP, and CMU μI's require two minor cycles to execute if the preceding μI altered the contents of Aμ and/or Bμ.

b. Execution of Load S μI's during E0 require one or two additional minor cycles if the system contains the basic protection feature or the relocation and protection feature, and/or the ECC feature.

c. Branch μI's which reference the address portion of both Sμ and P_p (FNJ, JMP, and AND, CLR, STA, and STB when X = Pμ) require one additional minor cycle if executed during any minor cycle other than E7.

d. Branch μI's which reference the address portion of P_p only (FRJ, FZJ when (Aμ) = 0, RNI1, and RNI2) cause a resync condition described in the paragraph on Resync μI's.

e. Control μI's C1O1, C1O2, ROM, and SYNC cause a resync condition described in the paragraph on Resync μI's.

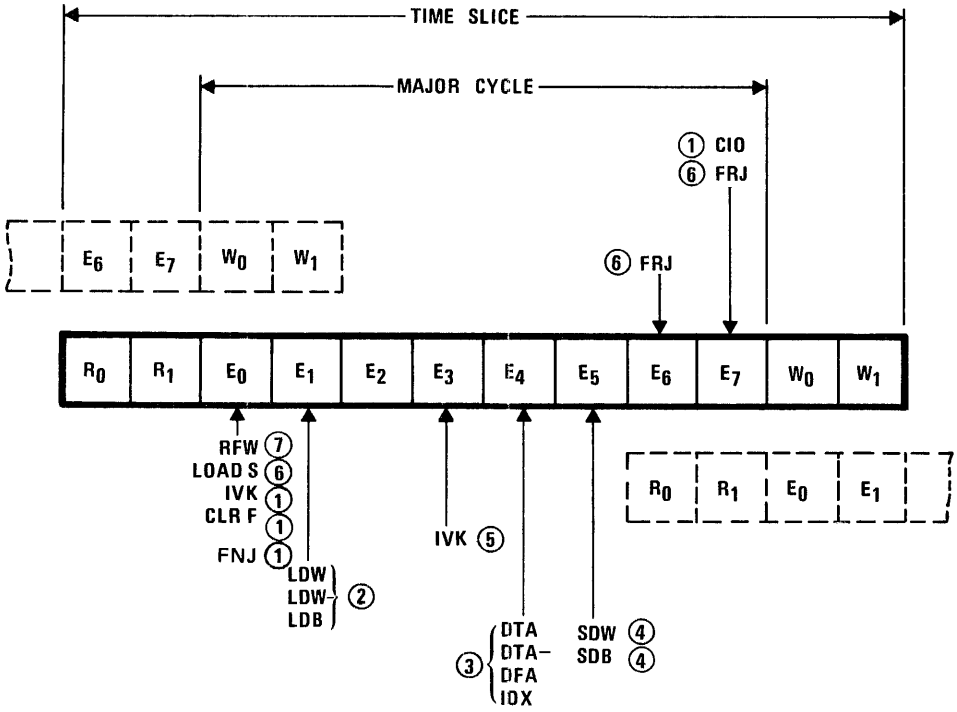
The above group of timing constraints are referred to as synchronous constraints because the timing restrictions occur as a result of predictable timing anomalies. In contrast, there are a number of asynchronous constraints which occur because of unpredictable signals generated as a result of operator intervention or I/O processor requests. When these signals are read, resolve time of 200 nanoseconds must be allowed before any actions dependent on the states of such signals are taken. This interval of time is necessary to allow the asynchronous signal to assume a final, steady-state condition. For example, if a LAW μI designating the Panel Address register (ERF register 0A) is followed by a Skip μI, which uses the result of Aμ to perform a skip, an interval of 200 nanoseconds must be inserted between the two μI's to prevent a possible machine malfunction. Such a malfunction could result from the fact that the operator could be altering the contents of the address register at the very instant its contents were being read by the LAW μI, resulting in an indeterminate skip evaluation. This resolve time requirement must be accommodated under

microprogram control. In the case of this example, the requirement should be satisfied by two NOP μI's immediately following the LAW μI, a STA and LAW combinations specifying an otherwise unused register within the BRF or ERF, a SHF μI designating a shift count of 0, or some other non-interferring combination of two μI's. Such precautions are particularly applicable to μI's which read or write the ERF Group III registers.

MICRO-INSTRUCTION DESCRIPTION

Descriptions of each of the μI's are presented in the following paragraphs. The μI's are arranged in order of operation code according to their class. Each description consists of an English title, mnemonic identifier, operation code in hexadecimal form, instruction format, and a narrative description. The description also lists the μI execution time(s), plus any timing constraints or anomalies peculiar to the μI. The ten classes of μI's are as follows:

1. register file read
2. register file write
3. register file read, MS related
4. register file write, MS related
5. immediate operand
6. shift
7. bit sense
8. skip
9. branch
10. control



MINOR CYCLE LEGEND

R – READ
W – WRITE
} HOUSEKEEPING (HARDWARE)
E – EXECUTE (MICROCODE)

NOTES

1. DO NOT EXECUTE AT TIME SHOWN.
2. MUST BE EXECUTED AT TIME SHOWN IF STORING (D) IN MS.
3. EARLIEST TIME VALID INFO IS AVAILABLE DURING READ MS.
4. EARLIEST TIME FOR STORING (D) IN REGISTER FILE DURING A MS READ
5. DO NOT EXECUTE AFTER TIME SHOWN IF ACCESSING F OR Pμ.
6. EXECUTE ONLY AT TIME SHOWN IF PERFORMED IN CONJUNCTION WITH A MS READ.
7. DO NOT EXECUTE A REGISTER FILE WRITE INSTRUCTION AT E0 IF DESTINATION X IS OF THE F REGISTER. TO DO SO MAY DESTROY THE CONTENTS OF THE F REGISTER FOR THE PROCESSOR HAVING THE PREVIOUS TIME SLICE.

Figure 3-1. Microcode Timing Restrictions

Table 3-1. Micro-Instruction Characteristics

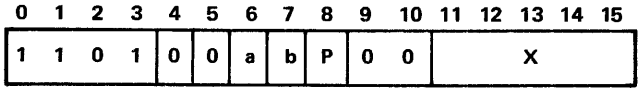
MNEMONIC	REG. ADDRESSABILITY Group I F	Pμ	GRP II	GRP III	BLOCK POINT	FEEDER LOAD	Pμ WRITE	RESYNC
AND	W*	W	W	W	X		X	
CIO1					X			X
CIO2					X			X
CLA	R*	R	R	R		X		
CLR	W	W	W	W	X		X	
CMP	W		W		X			
CMU	W		W		X			
CORC						X		
DFA	R	R	R			X		
DIG						X		
DLS						X		
DRS						X		
DSUM	W		W		X			
DTA	R	R	R			X		
DTA\	R	R	R			X		
EBL						X		
EBU						X		
EOR	W		W	W	X			
FNJ					X			
FRJ					X			X
FZJ					X			X
IDX	R	R				X		
IOR	W		W	R	X			
IVK								
JMP					X			
LAB	R	R	R	R		X		
LAW	R	R	R	R		X		
LAW\	R	R	R	R		X		
LBB						X		
LBB\						X		
LBL	R	R	R	R		X		
LBW	R	R	R	R		X		
LBW\	R	R	R	R		X		
LDB	R	R	R	R				
LDW	R	R	R	R				
LDW\	R	R	R	R				
LSE	R	R	R			X		
LSF	R	R	R			X		
LS1	R	R	R			X		
LS2	R	R	R			X		
NOP								
RNI1	W		W		X			X
RNI2	W		W		X			X
ROM					X			X
RVK								
SDB	W		W	W	X			
SDW	W		W	W	X			
SHF						X		
SHR						X		
SKB					X			
SKB\					X			
SKE					X			
SKE\					X			
SKG					X			
SKL					X			
SKN					X			
SKZ					X			
SR0						X		
SR1						X		
SS0						X		
SS1						X		
STA	W	W	W	W	X		X	
STB	W	W	W	W	X		X	
SUM	W		W		X			
SYNC					X			X

*W represents a *Write* operation, R represents a *Read* operation

REGISTER FILE READ MICRO-INSTRUCTIONS

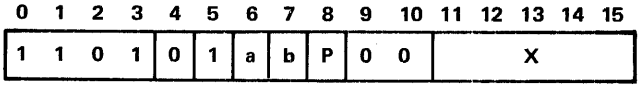
The μI's in this class perform register file read references which are unrelated to main storage operations.

Load Aμ Word (LAW)
D,0



Loads the Aμ register with the contents of the register-file-register designated by the X-field. Execution time: 100 nanoseconds.

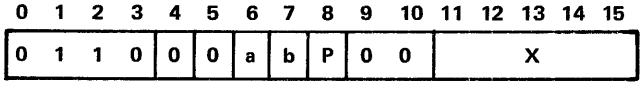
Load Aμ Complement (LAW-)
D,1



Loads the Aμ register with the one's complement of the contents of the register file register designated by the X-field. Stores 1 in the Forced Carry register (FCR).

Execution time: 100 nanoseconds

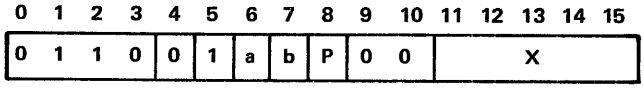
Load Bμ Word (LBW)
6,0



Loads the Bμ register with the contents of the register file register designated by the X-field. Stores 0 in the Forced Carry register (FCR).

Execution time: 100 nanoseconds

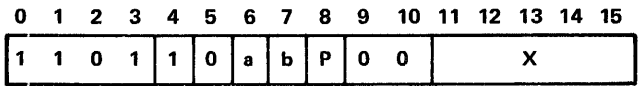
Load Bμ Complement (LBW-)
6,1



Loads the Bμ register with the one's complement of the contents of the register file register designated by the X-field. Stores 1 in the Forced Carry register (FCR).

Execution time: 100 nanoseconds

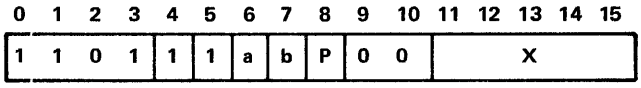
Load Aμ and Bμ (LAB)
D,2



Loads the Aμ register with the contents of the register file register designated by the X-field. Loads the Bμ register with the contents of the register file register designated by the X-field. Stores 0 in the Forced Carry register (FCR).

Execution time: 100 nanoseconds

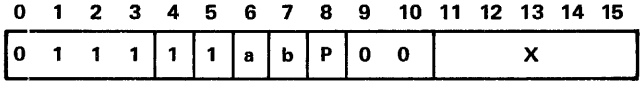
Clear Aμ (CLA)
D,3



Clears the Aμ register. Loads the Bμ register with the contents of the register file register designated by the X-field. Stores 1 in the Forced Carry register (FCR).

Execution time: 100 nanoseconds

Load Bμ Link (LBL)
7,3



Loads the Bμ register with the contents of the register file register designated by the X-field. Stores the Link bit (bit 1 of Pμ register) in the Forced Carry register (FCR).

Execution time: 100 nanoseconds

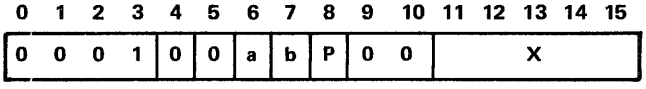
REGISTER FILE WRITE MICRO INSTRUCTIONS

The μI's in this class perform register file write references which are unrelated to main storage operations.

NOTE

The CLR, STA, STB and AND μI's cause a branch operation when the Pμ-Register is designated by the X-field.

Clear Contents of Register (CLR)
1,0

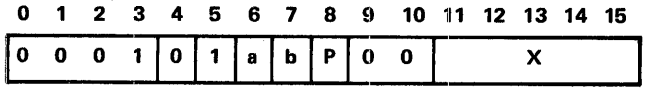


Clears the register file register designated by the X-field. Updates P_p.

Execution time: normally, 100 nanoseconds

When the register file register designated is Pμ, the execution time is 200 nanoseconds; however, the instruction can be executed at time E7. Do not use at time E0 when the X-field designates the F_{RF} register. This could result in clearing the previous processor's F_{RF} register (if changed during E7).

Store Aμ (STA)
1,1

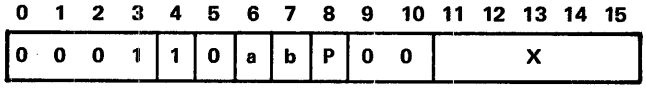


Stores the contents of the Aμ register into the register file register designated by the X-field. Update P_p.

Execution time: normally, 100 nanoseconds

When the register file register designated is Pμ, the execution time is 200 nanoseconds; however, the instruction can be executed at time E7.

Store Bμ (STB)
1,2

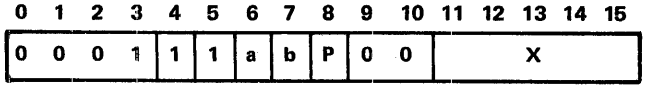


Stores the contents of the Bμ register into the register file register designated by the X-field. Updates P_p.

Execution time: normally, 100 nanoseconds

When the register file register designated is Pμ, the execution time is 200 nanoseconds; however, the instruction can be executed at time E7.

Logical Product, Aμ and Bμ (AND)
1,3



Stores the logical product of the Aμ register and the Bμ register into the register file register designated by the X-field.

Logical product is illustrated by the following truth table.

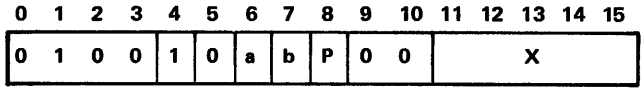
Aμ \ Bμ	0	1
0	0	0
1	0	1

Updates P_p.

Execution time: normally 100 nanoseconds

When the register file register designated is Pμ, the execution time is 200 nanoseconds; however, the instruction can be executed at time E7.

Inclusive OR, Aμ and Bμ (IOR)
4,2



Stores the inclusive OR of the Aμ register and the Bμ register into the register file register designated by the X-field.

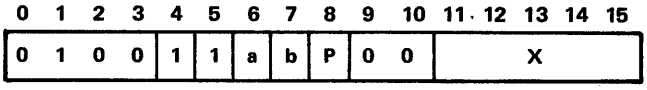
Inclusive OR is illustrated by the following truth table.

A \ B	0	1
0	0	1
1	1	1

Updates P_p.

Execution time: 100 nanoseconds

Exclusive OR, Aμ and Bμ (EOR)
4,3



Stores the exclusive OR of the Aμ register and the Bμ register into the register file register designated by the X-field.

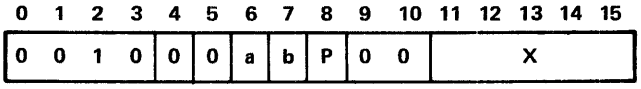
Exclusive OR is illustrated by the following truth table.

Aμ \ Bμ	0	1
0	0	1
1	1	0

Updates P_p.

Execution time: 100 nanoseconds

Sum, Aμ and Bμ (SUM)
2,0



Stores the sum of the Aμ register, the Bμ register and the Forced Carry register into the register file register designated by the X-field. Overflow occurs when both the Aμ register and the Bμ register have like signs, but the resultant sum has the opposite sign. Overflow is reflected in bit position 0 of the Pμ register; if overflow occurs, bit 0 is set, otherwise bit 0 is cleared. Link is the carry out of bit position 0 during the sum operation. Link is reflected in bit position 1 of the Pμ register.

NOTE

If bits 0-7 of the Function register (F) equal 50_{16}–53_{16}, or if inner carriers are inhibited as a result of a DIG or CORC μI, bits 0 and 1 of the Pμ register are not affected.

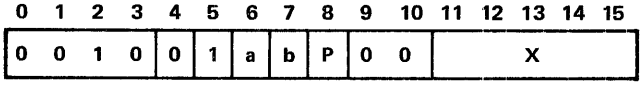
Updates P_p.

Execution time: normally 200 nanoseconds

NOTE

Whenever any Feeder Load μI (q.v.) is executed, the sum begins propagating and requires approximately 100 nanoseconds before it can be written in the register file. Consequently, if a SUM μI is preceded by a μI which is not a Feeder Load μI, the propagation time is overlapped with the execution of the non-Feeder Load μI and the actual SUM μI requires only 100 nanoseconds to execute.

Decimal Sum, Aμ and Bμ (DSUM)
2,1



Stores the sum of the Aμ register, the Bμ register and the Forced Carry register into the register file register designated by the X-field. Transfers the inner carries to the Inner-Carry register, unless inner carries are inhibited (q.v. DIG and CORC). Overflow occurs when both the Aμ register and the Bμ register have like signs, but the resultant sum has the opposite sign. Overflow is reflected in bit position 0 of the Pμ register; if overflow occurs, bit 0 is set, otherwise bit 0 is cleared. Link is the carry out of bit position 0 during the sum operation.

NOTE

If bits 0-7 of the Function register (F) equal 50_{16}-53_{16}, link is the carry out of bit position 8 during the sum operation. Link is reflected in bit position 1 of the Pμ register. If inner carries are inhibited as a result of a DIG or CORC μI, bits 0 and 1 of Pμ are not affected.

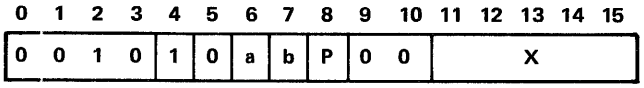
Updates P_p.

Execution time: normally 200 nanoseconds

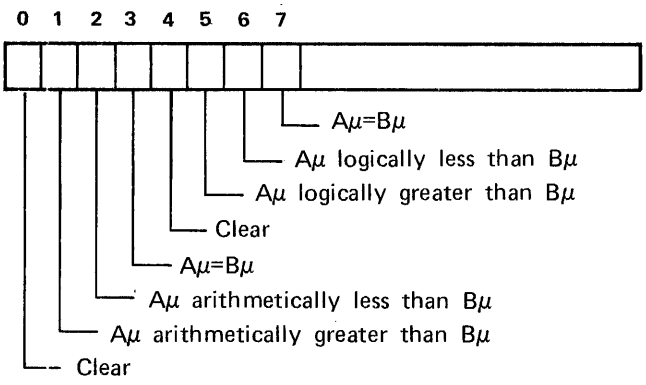
NOTE

Whenever any Feeder Load μI (q.v.) is executed, the sum begins propagating and requires approximately 100 nanoseconds before it can be written to the register file. Consequently, if a DSUM μI is preceded by a μI which is not a Feeder Load μI, the propagation time is overlapped with the execution of the non-Feeder Load μI and the actual DSUM μI requires only 100 nanoseconds to execute.

Sign and Magnitude Compare (CMP)
2,2



Performs a comparison of the Aμ register and Bμ register contents. A corresponding bit is set and all others cleared in the bit 0-7 field of the register file register designated by the CMP X-field to indicate the results of the compare as shown below:



Bits 8-15 of the register file register are unchanged, unless the register designated is an extended register, in which case bits 8-15 are set.

For logical results, $FFFF_{16}$ is the largest number that can be stored and 0000_{16} is the smallest number.

For arithmetic results, $7FFF_{16}$ is the largest number than can be stored and 8000_{16} is the smallest number.

Updates P_p.

Execution time: normally 200 nanoseconds

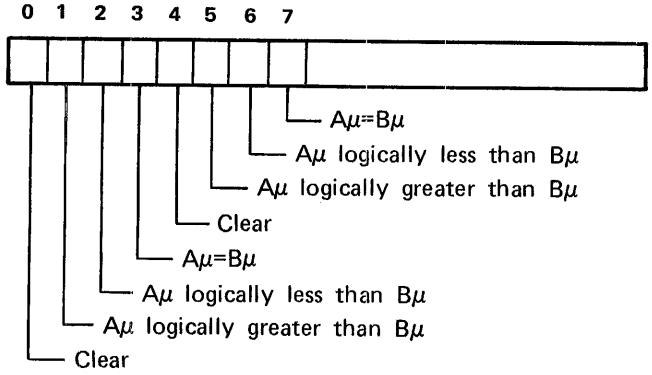
NOTE

Whenever any Feeder Load μI (q.v.) is executed, the compare begins propagating and requires approximately 100 nanoseconds before the result of the comparison can be written to the register file. Consequently, if a CMP μI is preceded by a μI which is not a Feeder Load μI, the propagation time is overlapped with the execution of the non-Feeder Load μI and the actual CMP μI requires only 100 nanoseconds to execute.

Magnitude Compare (CMU)
2,3

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	1	1	a	b	P	0	0	X				

Performs a comparison of the Aμ register and Bμ register contents. A corresponding bit is set and all others cleared on the bit 0-7 field of the register file register designated by the CMU X-field to indicate the results of the compare as shown below:



Bits 8-15 of the register file register are unchanged, unless the register designated is an extended register, in which case bits 8-15 are set.

$FFFF_{16}$ is the largest number that can be stored and 0000 is the smallest number.

Updates P_p.

Execution time: normally 200 nanoseconds

NOTE

Whenever any Feeder Load μI (q.v.) is executed, the compare begins propagating and requires approximately 100 nanoseconds before the result of the comparison can be written to the register file. Consequently, if a CMU μI is preceded by a μI which is not a Feeder Load μI, the propagation time is overlapped with the execution of the non-Feeder Load μI and the actual CMU μI requires only 100 nanoseconds to execute.

REGISTER FILE READ, MAIN STORAGE RELATED MICRO-INSTRUCTIONS

The μI's in this class perform register file read references which are, or may be related to, main storage (or register option) operations. Micro-instructions LS1, LSF, LS2 and LSE are unconditionally related to main storage (or register option) operations. The LDW, LDW-, and LDB μI's are main storage (or register option) related only when they are executed during E1 immediately following the execution of an LS1, LSF, LS2 or LSE μI at E0. All other μI in this class are main storage (or register option) related when they occur after, but within the same major cycle as LS1, LSF, LS2 or LSE μI executed for the purpose of performing main storage (or register option) read operations.

Selection of an input to the D Fan-In Network within the ALU is conditioned by the μI's within this class in the following manner:

a. During main storage read operations, the data from main storage is selected at the D Fan-In Network from E4 through E7.

b. During register option read operations, the data from the register option is selected at the D Fan-In Network from E4 through E7.

c. For the purpose of making D Fan-In Network selection only, register option read operations which specify the register set associated with the ECC feature are treated as main storage read operations.

d. During all minor cycles other than those described in items a, b, and c, the D register is selected at the D Fan-In Network.

Load S (LS1)
3,0

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	0	a	b	P	0	0	X				

Stores the contents of the register file register designated by the X-field into the Storage Address register (S) and into the Aμ register. Stores 0000_{16} (0_{10}) in the Bμ register. Stores 1 in the Forced Carry register (FCR).

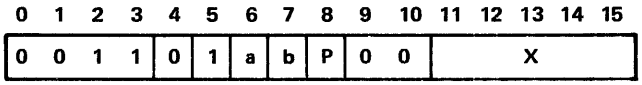
This μI initiates a main storage reference. If the next sequential μI is a load Storage Data register (LDW, LDW-,

or LDB), a write to main storage is performed; otherwise a read from main storage is performed.

This μI always begins execution at time E0. Consequently, the μI immediately preceding this μI must update P_p, since upon reading up this μI, if the time is other than E0, the hardware will cause an idle through the remainder of the current major cycle. Then the normal mechanism at time W0 of storing P_p into Pμ will cause the address of an already executed μI to be designated as the starting point for the major cycle, and a loop will result in microcode.

Execution time: 100 nanoseconds

Load S (LSF)
3,1



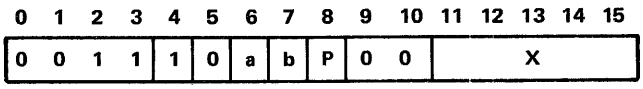
Stores the contents of the register file register designated by the X-field into the Storage Address register (S) and into the Aμ register. Stores $FFFF_{16}$ (-1_{10}) in the Bμ register. Stores 0 in the Forced Carry register (FCR).

This μI initiates a main storage reference. If the next sequential μI is a load Storage Data register (LDW, LDW-, or LDB), a write to main storage is performed; otherwise a read from main storage is performed.

This μI always begins execution at time E0. Consequently, the μI immediately preceding this μI must update P_p, since upon reading up this μI, if the time is other than E0, the hardware will cause an idle through the remainder of the current major cycle. Then the normal mechanism at time W0 of storing P_p and Pμ will cause the address of an already executed μI to be designated as the starting point for the major cycle, and a loop will result in microcode.

Execution time: 100 nanoseconds

Load S (LS2)
3,2



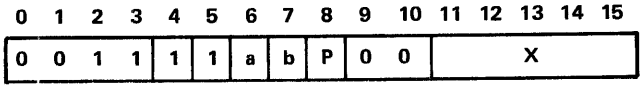
Stores the contents of the register file register designated by the X-field into the Storage Address register (S) and into the Aμ register. Stores 0001_{16} ($+1_{10}$) in the Bμ register. Stores 1 in the Forced Carry register (FCR).

This μI initiates a main storage reference. If the next sequential μI is a Load Storage Data Register (LDW, LDW-, LDB), a write to main storage is performed; otherwise a read from main storage is performed.

This μI always begins execution at time E0. Consequently, the μI immediately preceding this μI must update P_p, since upon reading up this μI, if the time is other than E0, the hardware will cause an idle through the remainder of the current major cycle. Then the normal mechanism at time W0 of storing P_p into Pμ will cause the address of an already executed μI to be designated as the starting point for the major cycle, and a loop will result in microcode.

Execution time: 100 nanoseconds

Load S (LSE)
3,3



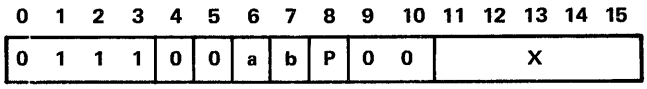
Stores the contents of the register file register designated by the X-field into the Storage Address register (S) and into the Aμ register. Stores $FFFE_{16}$ (-2_{10}) in the Bμ register. Stores 0 in the Forced Carry register (FCR).

This μI initiates a main storage reference. If the next sequential μI is a load Storage Data register (LDW, LDW-, or LDB), a write to main storage is performed; otherwise a read from main storage is performed.

This μI always begins execution at time E0. Consequently, the μI immediately preceding this μI must update P_p, since upon reading up this μI, if the time is other than E0, the hardware will cause an idle through the remainder of the current major cycle. Then the normal mechanism at time W0 of storing P_p into Pμ will cause the address of an already executed μI to be designated as the starting point for the major cycle, and a loop will result in the micro instruction routine.

Execution time: 100 nanoseconds

Load D Word (LDW)
7,0



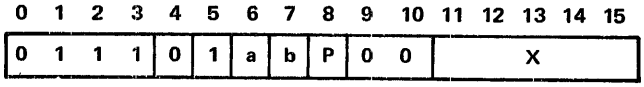
Loads the Storage Data register (D) with the contents of the register file register designated by the X-field.

Execution time: 100 nanoseconds

When executed during E1, immediately following an LS1, LSF, LS2 or LSE μI at E0, this μI will result in a main storage or register option write reference involving a full transfer of the D register output. In such cases, alteration of the contents of the D register by means of μI's during E2 through E7, may result in machine malfunction. The word locations of the write reference within main storage,

or within the register option, are designated by the contents of the S register (bit 15 irrelevant except for breakpoint) and are subject to appropriate hardware validity checks on the part of the Basic Storage Protection or Relocation and Protection features. Write references thus performed involve 16 data bits.

Load D Complement (LDW–)
7,1

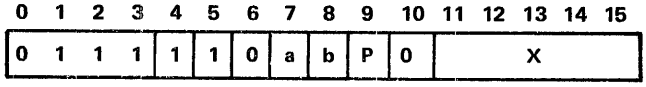


Loads the Storage Data register (D) with the one's complement of the contents of the register file register designated by the X-field.

Execution time: 100 nanoseconds

For a description of the relationship of this μI to main storage and register option write references see the comments for the LDW μI.

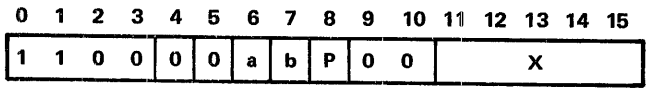
Load D Byte (LDB)
7,2



Loads the Storage Data register (D) with the contents of the register file register designated by the X-field.

When executed during E1, immediately following an LS1, LSF, LS2, or LSE μI at E0, this μI results in a main storage write reference involving a partial transfer of the D register output for which the right-most byte is duplicated in the left-most byte position (the D register output to the D Fan-In Network is not affected). In such cases, alteration of the contents of the D register by means of μI's during E2 through E7, may result in machine malfunction. The byte location of the write reference within main storage is designated by the contents of the S register and is subject to appropriate hardware validity checks on the part of the Basic Storage Protection or Relocation and Protection features. Write references thus performed involve the transfer of only the left-most data byte where bit position 15 of the S register is clear, or the transfer of only the right-most data byte where bit 15 of the S register is set.

D to Aμ, True (DTA)
C,0

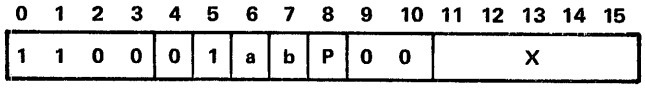


Transfers the output of the data fan-in to the Aμ register. Loads the Bμ register with the contents of the register file register file register designated by the X-field. Stores 0 in the Forced Carry register (FCR).

If a main storage reference was initiated at the beginning of this major cycle, this μI will not execute prior to time E4.

Execution time: 100 nanoseconds

D to Aμ, Complement (DTA–)
C,1

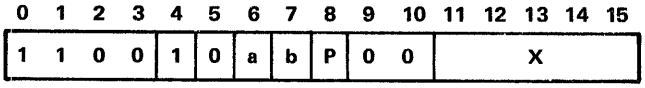


Transfers the output of the data fan-in to the Aμ register. Loads the Bμ register with the one's complement of the register file register designated. Stores 1 in the Forced Carry register (FCR).

If a main storage reference was initiated at the beginning of this major cycle, this μI will not execute prior to time E4.

Execution time: 100 nanoseconds

Index (IDX)
C,2



If the register file register designated by the X-field equals zero, the Bμ register is cleared; otherwise, loads the Bμ register with the contents of the register file register designated by the X-field. Transfers the output of the data fan-in to the Aμ register. Stores 0 in the Forced Carry register (FCR).

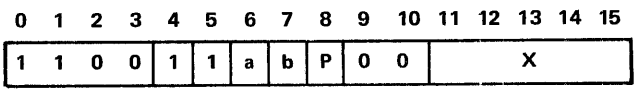
If a main storage reference was initiated at the beginning of this major cycle, this μI will not execute prior to time E4.

Execution time: 100 nanoseconds

In the presence of the Relocation and Protection feature, the IDX micro-command also serves as the implicit micro-command control mechanism for dynamic segment tag write references. Each IDX μI allows the next register file write reference, performed under μI control, to occur such that the associated segment tag is also written. The segment tag value so written will correspond to the segment tag value read during the last LS1, LSF, LS2, or LSE μI, whenever the associated IDX μI simply cleared the Bμ register. Alternatively, the segment tag value so

written will correspond to the segment tag value read during the associated IDX μI whenever this associated IDX μI performed a transfer of the register file output to the Bμ register.

D False to Aμ (DFA)
C,3



Transfers the one's complement of the data fan-in output to the Aμ register. Loads the Bμ register with the contents of the register file register designated by the X-field. Stores 1 in the Forced Carry register (FCR).

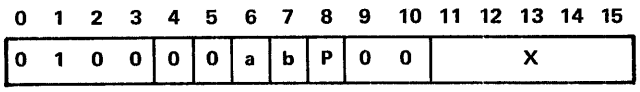
If a main storage reference was initiated at the beginning of this major cycle, this instruction will not execute prior to time E4.

Execution time: 100 nanoseconds

REGISTER FILE WRITE, MAIN STORAGE RELATED MICRO INSTRUCTIONS

The μI's within this class perform register file write references which may be related to main storage or register option read operations. These μI's will be main storage or register option related when they occur after, but in the same major cycle as LS1, LSF, LS2, or LSE μI's, which are executed for the purpose of performing main storage or register option read operations.

Store D Word (SDW)
4,0



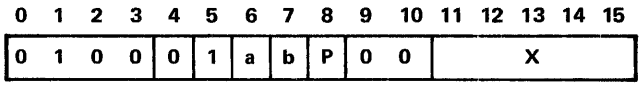
Stores the output of the data fan-in into the register file register designated by the X-field.

If a main storage reference was initiated at the beginning of this major cycle, this μI will not execute prior to time E5.

Updates P_p.

Execution time: 100 nanoseconds

Store D Byte (SDB)
4,1



Clears bits 0-7 of the register file register designated by the X-field. If the current contents of the Storage Address register is even, bits 0-7 of the data fan-in output are stored in bits 8-15 of the register file register designated, otherwise bits 8-15 of the data fan-in output are stored in bits 8-15 of the register file register designated.

If a main storage reference was initiated at the beginning of this major cycle, this μI will not execute prior to time E5.

Updates P_p.

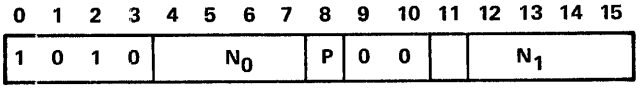
Execution time: 100 nanoseconds

IMMEDIATE OPERAND MICRO INSTRUCTIONS

The μI's within this class transfer immediate operands to the Bμ-register. These immediate operands are contained within the μI's themselves, with the exception of CORC and special cases of the LBB and LBB- μI's.

Undesignated bit positions within these μI's have no effect on μI execution except to the extent that they shall participate in the formation of valid parity.

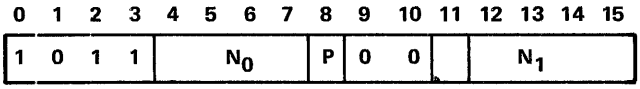
Enter Bμ Upper (EBU)
A



Transfers N_0 into bits 0-3 of the Bμ register and transfers N_1 into bit positions 4-7 of the Bμ register. Bits 8-15 of the Bμ register are unaffected.

Execution time: 100 nanoseconds

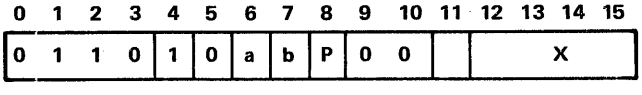
Enter Bμ Lower (EBL)
B



Clears bit positions 0-7 of the Bμ register. Transfers N_0 into bits 8-11 of the Bμ register. Transfers N_1 into bits 12-15 of the Bμ register.

Execution time: 100 nanoseconds

Load Bμ Bit (LBB)
6,2



If both a and b are set, sets the bit in both the upper and lower bytes of the Bμ register to correspond with the processor state number in which the μI is being executed and clears the remaining 14 bit positions, i.e.,

$$2^{(15-PROC\#)} + 2^{(7-PROC\#)} \rightarrow B\mu$$

If both a and b are clear, a bit in Bμ is set designated only by bit positions 12-15 of the μI.

$$2^{(15-X)} \rightarrow B\mu$$

If either a or b, but not both, is set, a bit in Bμ is set designated by four bits from the corresponding field of the F register inclusively ORed with bit positions 12-15 of the μI.

Stores 0 in the Forced Carry register (FCR).

Execution time: 100 nanoseconds

Load Bμ Bit Complement (LBB-)
6,3

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	0	1	1	a	b	P	0	0		X			

If both a and b are set, clears the bit in both the upper and lower bytes of the Bμ register which corresponds with the processor state number in which the μI is being executed and sets the remaining 14 bit positions, i.e.,

$$\overline{2^{(15-PROC\#)} + 2^{(7-PROC\#)}} \rightarrow B\mu$$

If both a and b are clear, a bit in Bμ is cleared, designated only by bit positions 12-15 of the μI.

$$\overline{2^{(15-X)}} \rightarrow B\mu$$

If either a or b, but not both, is set, a bit in Bμ is cleared designated by four bits from the corresponding field of the F register inclusively ORed with bit positions 12-15 of the μI.

Stores 1 in the Forced Carry register (FCR).

Execution time: 100 nanoseconds

Digit Duplication (DIG)
F,2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	1	1	1	0			P	0	0		X			

This instruction copies the absolute value of the 4-bit X-field (of the DIG instruction) into each 4-bit group of the Bμ register. Inhibits inner carries normally propagated for each digit position in the adder. Likewise, inhibits clocking Overflow and Link conditions in the Pμ register which occurs during sum operations. These disables remain in effect until a new value is inserted into either the Aμ or Bμ register by means of a Feeder Load μI other than DIG or CORC.

Stores 0 in the Forced Carry register (FCR).

Execution time: 100 nanoseconds

Correct Code (CORC)
F,3

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	1	1	1	1			P	0	0					

This instruction enters, into each of the 4-bit groups of the Bμ register, a hexadecimal value dictated by the state of the corresponding stage of the Inner Carry register (ICR). The ICR stages were set (or not set) by the previous DSUM instruction. If the bit (X_i) in the ICR is a 1, the value "3" is inserted into the appropriate 4-bit group of the Bμ register. If the ICR bit is a 0, the value "D" (2's complement of 3, expressed hexadecimally) is inserted in Bμ.

Inhibits inner carries normally propagated for each digit position in the adder. Likewise, inhibits clocking Overflow and Link conditions in the Pμ register which occurs during sum operations. These disables remain in effect until a new value is inserted into either the Aμ or Bμ register by means of a Feeder Load μI other than DIG or CORC.

Stores 0 in the Forced Carry register (FCR).

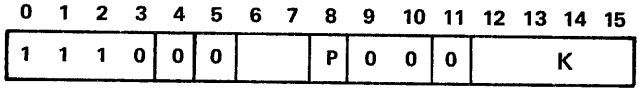
Execution time: 100 nanoseconds

SHIFT MICRO-INSTRUCTIONS

The μI's within this class left shift the contents of the Aμ/Bμ registers. Shift counts of 4 bits are μI-designated in true or 2's complement form, for shifts from 0 to 15_{10} binary places. Bits shifted from the Aμ register are end-off (lost). Bits from the Bμ register are shifted into the Aμ register with zeros inserted into the right-most bit positions of the Bμ register.

Undesignated bit positions within these μI's have no effect on μI execution except to the extent that they participate in the formation of valid parity.

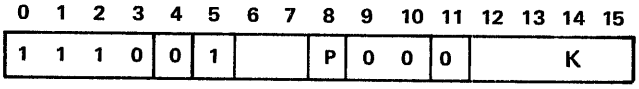
Shift Left (SHF)
E,0,0



Performs a left end-off shift of the combined Aμ register and Bμ register, with the Aμ register containing the most significant bits. The shift count is specified by K. Zeros are entered at the right end of the Bμ register.

Execution time: 200 nanoseconds

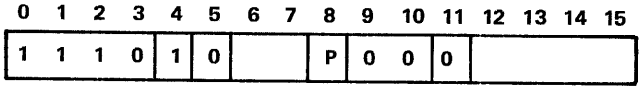
Shift Right (SHR)
E,1,0



Performs a left end-off shift of the combined Aμ register and Bμ register, with the Aμ register containing the most significant bits. The shift count is specified by the two's complement of K. Zeros are entered at the right end of the Bμ register.

Execution time: 200 nanoseconds

Left Shift, Dependent Count (DLS)
E,2,0



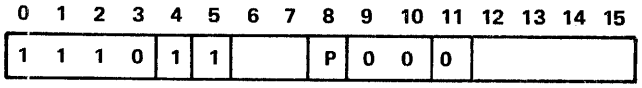
Performs a left end-off shift of the combined Aμ register and Bμ register, with the Aμ register containing the most significant bits. Zeros are entered at the right end of the Bμ register. Shift count is determined by the following scheme:

- If bit 1 of the Function register (F) is clear, the shift count is specified by bit positions 12 through 15 of the Storage Data register (D);
- If bit 1 of the Function register (F) is set, the shift count is specified by bit positions 8 through 11 of the Function register (F).

Execution time: 200 nanoseconds

If the shift count is to be obtained from the Storage Data register (D) and a main storage reference was initiated at the beginning of this major cycle, this μI cannot be executed prior to time E5. Otherwise, the shift count data will not be valid and the results are unpredictable.

Right Shift, Dependent Count (DRS)
E,3,0



Performs a left end-off shift of the combined Aμ register and Bμ register, with the Aμ register containing the most significant bits. Zeros are entered at the right end of the Bμ register. Shift count is determined by the following scheme:

- If bit 1 of the Function register (F) is clear, the shift count is specified by the two's complement of bit positions 12 through 15 of the Storage Data register (D);
- If bit 1 of the Function register (F) is set, the shift count is specified by the two's complement of bit positions 8 through 11 of the Function register (F).

Execution time: 200 nanoseconds

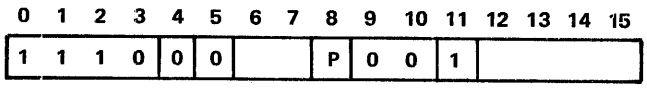
If the shift count is to be obtained from the Storage Data register (D) and a main memory reference was initiated at the beginning of this major cycle, this instruction cannot be executed prior to time E5, otherwise the shift count data will not be valid and the results are unpredictable.

BIT SENSE MICRO INSTRUCTIONS

The μI's within this class scan the contents of the Aμ register, from left to right, for the purpose of detecting the first bit position in the set or cleared state as specified by the associated μI. Bit positions thus detected are cleared or set within the Aμ register as specified by the SR1 and SS0 μI's, respectively. A value corresponding to the bit position detected, 00 through 15_{10}, is added to the contents of the Bμ register. When the entire Aμ register is scanned without detection of a bit in the specified state, 16_{10} shall be added to the contents of the Bμ register.

Undesignated bit positions within these μI's do not effect μI execution except to the extent that they participate in the formation of valid parity.

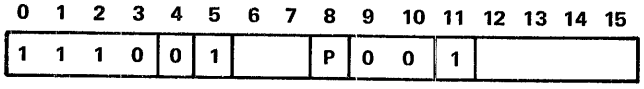
Sense for Zero (SRO)
E,0,1



Sequentially scans the Aμ register from bit position 00 toward bit 15, for the presence of the first "0". Increments the Bμ register by an amount equal to the number of bit positions scanned before finding the first "0". If no "0" is found, the Bμ register is incremented by 16_{10}.

Execution time: 200 nanoseconds

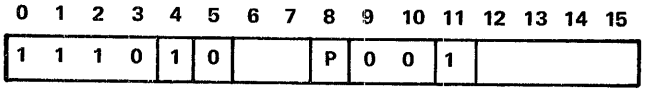
Sense for One (SS1)
E,1,1



Sequentially scans the Aμ register from bit position 00 toward bit 15, for the presence of the first "1". Increments the Bμ register by an amount equal to the number of bit positions scanned before finding the first "1". If no "1" is found, the Bμ register is incremented by 16_{10}.

Execution time: 200 nanoseconds

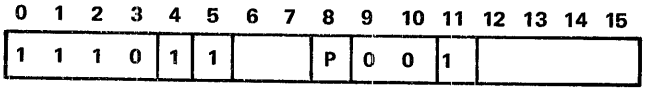
Sense and Set for Zero (SS0)
E,2,1



Sequentially scans the Aμ register from bit position 00 toward bit 15, for the presence of the first "0". Sets the first "0" and increments the Bμ register by an amount equal to the number of bit positions scanned before finding the first "0". If no "0" is found, the Bμ register is incremented by 16_{10}.

Execution time: 200 nanoseconds

Sense and Reset for One (SR1)
E,3,1



Sequentially scans the Aμ register from bit position 00 toward bit 15, for the presence of the first "1". Clears the first "1" and increments the Bμ register by an amount equal to the number of bit positions scanned before finding the first "1". If no "1" is found, the Bμ register is incremented by 16_{10}.

Execution time: 200 nanoseconds

SKIP MICRO INSTRUCTIONS

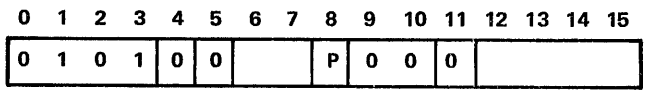
The μI's within this class provide for skipping the next successive μI when the specified conditions within the Aμ registers are met. These μI's require one minor cycle for translation and an additional minor cycle to skip the next successive μI when the specified conditions are met. Skip μI's for which the specified conditions are met as initially translated during E7 skip the next successive μI during E0 of the next appropriately-allocated major cycle.

NOTE

When the contents of the Aμ or Bμ registers are logically ambiguous as a result of transferring asynchronous signals into them, the execution of Skip μI's without an allowance for resolve time may result in machine malfunction in the form of undefined and unpredictable μI execution. See the paragraph on Timing Constraints.

Undesignated bit positions within these μI's have no effect on μI execution except to the extent that they participate in the formation of valid parity.

Skip if Aμ is Zero (SKZ)
5,0,0

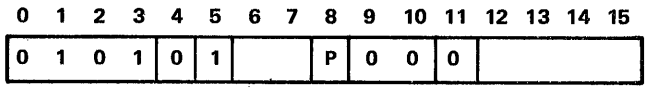


If the contents of the Aμ register are equal to zero, the next sequential μI is not executed; however, 100 nanoseconds are required to cycle through the skipped μI.

Updates P_p.

Execution time: 100 nanoseconds

Skip if Aμ is Non-Zero (SKN)
5,1,0

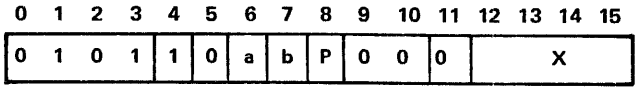


If the contents of the Aμ register are not equal to zero, the next sequential μI is not executed; however, 100 nanoseconds are required to cycle through the skipped μI.

Updates P_p.

Execution time: 100 nanoseconds

Skip if Aμ Bit is a One (SKB)
5,2,0



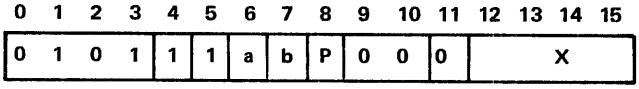
If the designated bit of the Aμ register is set, the next sequential μI is not executed; however, 100 nanoseconds are required to cycle through the skipped μI.

Designated bit – If either a or b is set, four bits from the corresponding field of the F register are inclusively ORed with bit positions 12-15 of the μI to determine the bit to be accessed.

Updates P_p.

Execution time: 100 nanoseconds

Skip if Aμ Bit is a Zero (SKB–)
5,3,0



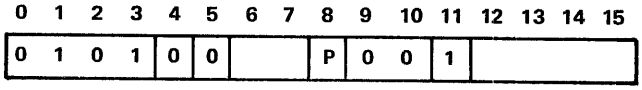
If the designated bit of the Aμ register is not set, the next sequential μI is not executed; however, 100 nanoseconds are required to cycle through the skipped μI.

Designated bit – If either a or b is set, four bits from the correspnding field of the F register are inclusively ORed with bit positions 12-15 of the μI to determine the bit to be accessed.

Updates P_p.

Execution time: 100 nanoseconds

Skip if Aμ > Bμ (SKG)
5,0,1

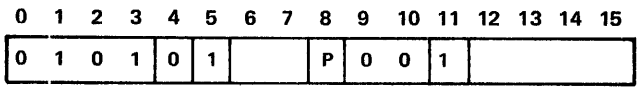


Performs a 16-bit logical compare of the Aμ register and the Bμ register. If Aμ > Bμ, the next sequential μI is not executed; however, 100 nanoseconds are required to cycle through the skipped μI.

Updates P_p.

Execution time: 100 nanoseconds

Skip if Aμ<Bμ (SKL)
5,1,1

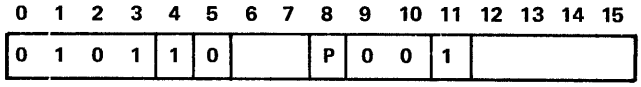


Performs a 16-bit logical compare of the Aμ register and the Bμ register. If Aμ< Bμ, the next sequential μI is not executed; however, 100 nanoseconds are required to cycle through the skipped μI.

Updates P_p.

Execution time: 100 nanoseconds

Skip if Aμ = Bμ (SKE)
5,2,1

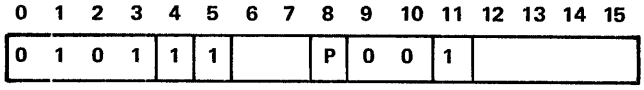


Performs a 16-bit logical compare of the Aμ register and the Bμ register. If Aμ = Bμ, the next sequential μI is not executed; however, 100 nanoseconds are required to cycle through the skipped μI.

Updates P_p.

Execution time: 100 nanoseconds

Skip if Aμ ≠ Bμ (SKE–)
5,3,1



Performs a 16-bit logical compare of the Aμ register and the Bμ register. If Aμ≠ Bμ, the next sequential μI is not executed; however, 100 nanoseconds are required to cycle through the skipped μI.

Updates P_p.

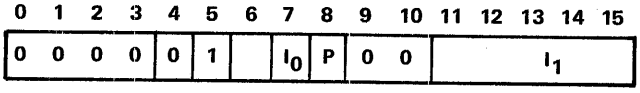
Execution time: 100 nanoseconds

BRANCH MICRO INSTRUCTIONS

In addition to the CLR, STA, STB and AND μI's which effect a branch operation when the Pμ register is designated as described in the paragraph titled Register File Writes, the six μI's in this class explicitly provide the means for performing branch operations.

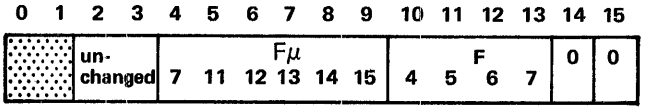
As opposed to the implicit μI's previously mentioned and described in the paragraph titled Register File Writes, the explicit μI's in this class are capable of only partial write references to the right-most address portions of the Sμ and P_p registers.

Function Decode Jump (FNJ)
0,1

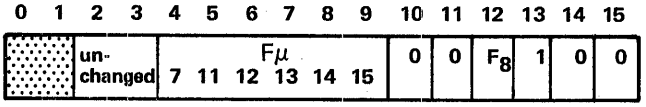


The function decode jump causes a branch by placing a value in Sμ according to the following algorithm:

if Fμ6 = 0



If Fμ6 = 1



FNJ cannot be executed at time E0

Updates P_p.

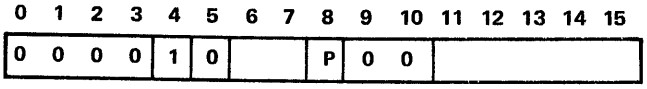
Execution time: 200 nanoseconds, however, can be executed at time E7

ANOMALY: Normally, the FNJ μI branches to a location within the same 4096-word CS module in which the jump is located. This is what is indicated by bits 02 and 03 of Sμ being "unchanged". However, there are two cases when the decode jump branches to a location within the next consecutive 4096 word module.

1. If the decode jump occupies the last location of a 4096-word module (address $XFFF_{16}$).

2. If the decode jump occupies the next-to-last location of a 4096-word module (address $XFFE_{16}$) and is executed any time other than E6 or E7.

Format Decode Jump (FRJ)
0,2



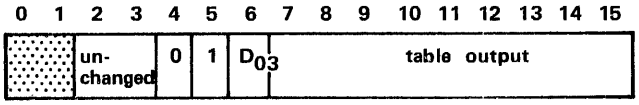
FRJ – The 1st level decode jump will access a 256 word address table whose contents are alterable and loaded at CS Load time. Input to this table is determined from the function code as shown in Table 3-2.

Table 3-2 Address Table Input Translation

Function Code	Address Table Input Translation							
2X,3X,AX,BX	0	0	F_{00}	F_{03}	XB or XD-XF	XA or XC-XF	F_{08}	F_{12}
6X,7X	0	1	0	F_{03}	F_{04}	F_{05}	F_{08}	F_{12}
0X,1X,4X,5X 8X,9X,CX,DX	0	1	1	F_{00}	F_{01}	F_{03}	F_{04}	F_{05}
EX	1	0	F_{04}	F_{05}	F_{06}	F_{07}	F_{08}	F_{12}
FX	1	1	F_{04}	F_{05}	F_{06}	F_{07}	F_{08}	F_{12}

Note that the function code must have been transferred to the Storage Data register (D), since the FRJ instruction actually keys off the D register.

The address table output consists of a parity bit plus 9 bits which are used as the right-most bits of the FRJ branch address. The left-most bits are as shown below:



Cannot be executed until a minimum of 200 nanoseconds has elapsed since the loading of the D register.

Resyncs so that the next μI will execute at time EO of the next major cycle assigned to this processor.

Updates P_p.

Execution time: 200 nanoseconds, however, can be executed at time E7

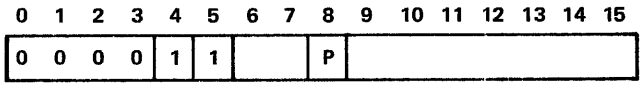
ANOMALY: Normally, the FRJ μI branches to a location within the same 4096-word CS module in which the jump is located. This is what is indicated by bits 02 and 03 of Sμ being "unchanged". However, there are two cases when the decode jump branches to a location within the next consecutive 4096-word module.

1. If the decode jump occupies the last location of a 4096-word module (address $XFFF_{16}$).

2. If the decode jump occupies the next-to-last location of a 4096-word module (address $XFFE_{16}$) and is executed at any time other than E6 ro E7.

Zero Jump

(FZJ 0,3)



Tests the contents of the Aμ register. If the contents equals zero, the next micro-instruction is taken from location $X009_{16}$ of control storage, otherwise the next micro-instruction is taken from the next sequential location.

Location $X009_{16}$ of control storage is the beginning of an RNI micro-instruction sequence.

If the contents of the Aμ register equals zero, a resync occurs such that the next μI will execute at time EO of the next major cycle assigned to this processor.

Updates P_p; however, the update address is always 0009_{16}. Therefore, to function properly, another blockpoint μI must occur later within the same major cycle.

Execution time: 200 nanoseconds, however, can be executed at E7

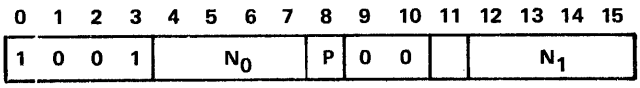
Normally, the FZJ instruction branches to location 0009_{16} within the same 4096-word CS module in which the jump is located. However, there are two cases when the jump branches to location 0009_{16} within the next consecutive 4096-word module.

1. If the FZJ occupies the last location of a 4096-word module (address $XFFF_{16}$).

2. If the FZJ occupies the next-to-last location of a 4096-word module (address $XFFE_{16}$) and is executed at any time other than E6 or E7.

Jump

(JMP 9)



Re-sequences the microcode by placing a value in Sμ according to the following scheme:

$N_0 \rightarrow S\mu_{8\text{-}11}$

$N_1 \rightarrow S\mu_{12\text{-}15}$

This will result in a branch in control storage to a location within the current 256-word page. Two conditions occur when the branch will be to the specified location in the next sequential page:

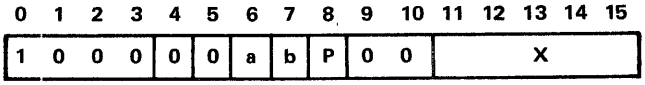
1. When the JMP μI occupies the last location of a page (address $XXFF_{16}$).

2. When the JMP μI occupies the next-to-last location of a page (address $XXFE_{16}$) and is executed at any time other than E6 or E7.

Updates P_p.

Execution time: 200 nanoseconds, however, can be executed at E7

Read Next Instruction 1 (RNI 1)
8,0



Re-sequences the microcode so that the next μI to be executed is at location $X002_{16}$ of control storage. Clears bits 1-2 and 4-7 of the register file register designated by the X-field. Transfers bit 0 of the Pμ

register (Overflow bit) to bit 0 of the register file register designated. Transfers bit 1 of the Pμ register (Link bit) to bit 3 of the register file register designated. Bits 8-15 of the register file register are unchanged, unless the register designated is an extended register, in which case bits 8-15 are set.

Updates P_p.

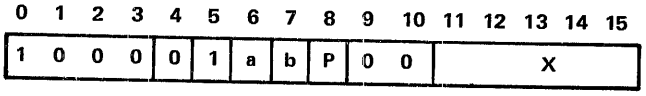
Execution time: 200 nanoseconds, however, can be executed at time E7

Normally, the RNI1 instruction branches to location 0002_{16} within the same 4096-word CS module in which the jump is located. However, there are two cases when the jump branches to location 0002_{16} within the next consecutive 4096-word module.

1. If the RNI1 occupies the last location of a 4096-word module (address $XFFF_{16}$).

2. If the RNI1 occupies the next-to-last location of a 4096-word module (address $XFFE_{16}$) and is executed at any time other than E6 or E7.

Read Next Instruction 2 (RNI 2)
8,1



Re-sequences the microcode so that the next μI to be executed is at location $X009_{16}$ of control storage. Clears bit positions 1, 2, and 4 through 7 of the register file register designated by the X-field. Transfers bit 0 of the Pμ register (Overflow) to bit 0 of the register file register designated. Transfers bit 1 of the Pμ register (Link) to bit 3 of the register file register designated. Bit positions 8-15 of the register file register are unchanged, unless the register designated is an extended register in which case bits 8-15 are set.

Resyncs so the next μI will execute at time EO of the next major cycle assigned to this processor.

Updates P_p.

Execution time: 200 nanoseconds, however, can be executed at time E7

Normally, the RNI2 instruction branches to location 0009_{16} within the same 4096-word module in which the jump is located. However, there are two cases when the jump branches to location 0009_{16} within the next consecutive 4096-word module.

1. If the RNI2 occupies the last location of a 4096-word module (address $XFFF_{16}$).

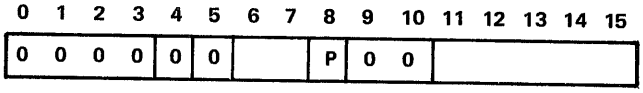
2. If the RNI2 occupies the next-to-last location of a 4096-word module (address $XFFE_{16}$) and is executed at any time other than E6 or E7.

CONTROL MICRO INSTRUCTIONS

The μI's within this class perform timing, input/output termination and boundary-crossing mode operations.

Undesignated bit positions within these μI's have no effect on μI execution except to the extent that they participate in the formation of valid parity.

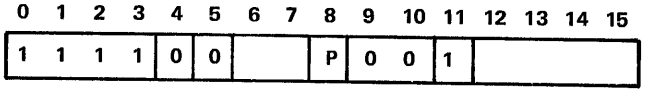
No Operation (NOP)
0,0



Does nothing

Execution time: 100 nanoseconds

Resynchronize (SYNC)
F,0,1

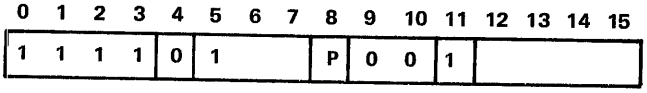


Resyncs the processing unit so that the next μI executes at time E0 of the next major cycle.

Updates P_p pointer

Execution time: 100 nanoseconds

Invoke Boundary Crossing Mode (IVK)
F,1,1



This instruction invokes the boundary-crossing (BC) mode, which allows a processor to access registers in another processor's register file. The condition continues until nullified by a RVK micro instruction, or until the end of the current major cycle. The method of specifying the register file address varies, depending upon which group of registers is being accessed.

a. Basic registers
- Bit 7 of the BC register and bits 6-7 of the μI must be cleared.

- Processor number is derived from bits 8-10 of the BC register.

- Register number is determined by inclusive ORing bits 11-15 of the BC register with bits 11-15 of the μI.

b. Group I extended registers
- Bit 7 of the BC register must be set and bits 11-14 must be cleared.

- Processor number is derived from bits 8-10 of the boundary crossing register.

- Register number is derived from bit 15 of the BC register.

These registers can only be accessed during time E_3 - E_4.

c. Group II extended registers
- Bit 7 of the boundary crossing register must be set and bit 11 must be cleared.

- Bit 11 of the μI must be cleared.

- Register number is determined by inclusive ORing bits 12-15 of the BC register with bits 12-15 of the μI. (Processor number is immaterial, since these are the common block registers.)

An attempt to read an unassigned register in this group (0C-0F) will yield zeros. An attempt to write an unassigned register in this group (0C-0F) will result in an effective NOP. Any write references addressing the BC register (08) while in the BC mode shall not be supported and may result in machine malfunction.

d. Group III extended registers
- Bits 6, 7 and 11 of the μI must be set, indicating group III.

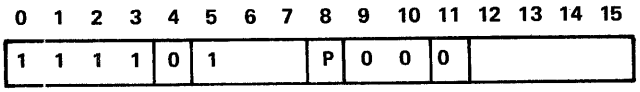
- Bits 7-11 of the BC register do not participate in address determination; hence only registers from the processor in execution can be accessed.

- Register number is determined by inclusive ORing bits 12-15 of the BC register with bits 12-15 of the μ.

Any operations not described above are undefined and, if attempted, cause unpredictable results.

Execution time - 100 nsec

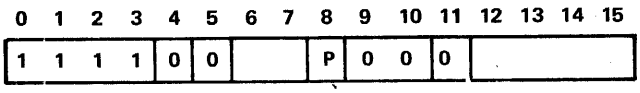
Revoke Boundary Crossing Mode (RVK)
F,1,0



Restores processor to "normal" mode after having been in "boundary crossing mode". See explanation under IVK.

Execution time: 100 nanoseconds

Read Control Memory (ROM)
F,0,0



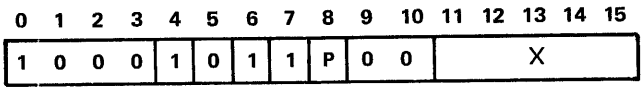
Performs an exclusive OR of the contents of the control storage location whose address is contained in the Bμ register and the contents of the Control Storage Scan Register (CSS), storing the results in the Control Storage Scan Register. Resyncs the processing unit so that the next μI will execute at time EO of the next major cycle.

Cannot execute later than time E5. If executed at time E6, the data transferred to the CS Scan register is unpredictable.

Updates P_p

Execution time: 200 nanoseconds

Compare I/O (CIOI)
8,2



Performs a 16-bit logical compare of the Aμ register and the Bμ register. If Aμ = Bμ, a resync occurs and the next sequenctial μI is executed at the beginning of the next major cycle. If A$\mu \neq$ Bμ, a resync occurs and the processor's busy bit is cleared. In addition, normal storing of P_p to Pμ at W0 time is suppressed so that the μI whose address is in Pμ is executed at the beginning of the next major cycle whenever the processor is reactivated.

Cannot be executed at time E7.

Updates P_p

Execution time: 100 nanoseconds

If bits 6 and 7 of the μI are not both set, a write will occur to the register file register designated by the

X-field, as follows: bits 1-2 and 4-7 will be cleared, bit 0 of the Pμ register will be transferred to bit 0, bit 1 of the Pμ register will be transferred to bit 3, bits 8-15 will not be affected.

Compare I/O (CI02)
8,3

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	0	0	1	1	1	1	P	0	0	X				

Performs a 16-bit logical compare of the Aμ register and the Bμ register. If A$\mu \neq$ Bμ, a resync occurs and the next sequential μI is executed at the beginning of the next major cycle. If Aμ = Bμ, a resync occurs and the next processor's busy bit is cleared. In addition, normal storing of P_p to Pμ at time W0 is suppressed so that the μI whose address is in Pμ is executed at the beginning of the next major cycle whenever the processor is reactivated.

Cannot be executed at time E7.

Updates P_p

Execution time: 100 nanoseconds

If bits 6 and 7 of the μI are not both set, a write will occur to the register file register designated by the X-field, as follows: bits 1-2 and 4-7 will be cleared, bit 0 of the Pμ register will be transferred to bit 0, bit 1 of the Pμ register will be transferred to bit 3, bits 8-15 will not be affected.

MICRO INSTRUCTION EXECUTION

Block diagrams which show the principal portions of logic required to execute each μI are shown in Figures 3-2 through 3-30. Figures 3-2 through 3-10 show details of the register file read and write operations that form a part of many μI's. Figures 3-11 through 3-30 show details of execution, particularly applicable to each μI, with references to the register file read and write operations of Figures 3-2 through 3-10 where required. Micro-instructions which are subject to timing constraints contain a reference on the block diagram to an applicable timing diagram. These timing diagrams are identified as Figures 3-31 through 3-36.

NOTE

All data and most control signals shown in the flow diagrams are represented in true form, regardless of whether the signal is actually defined in the true or complement form. Exceptions to this rule are certain control signals that are time-restricted, shown in their actual state (either true or complement form) during the time that they are restricted.

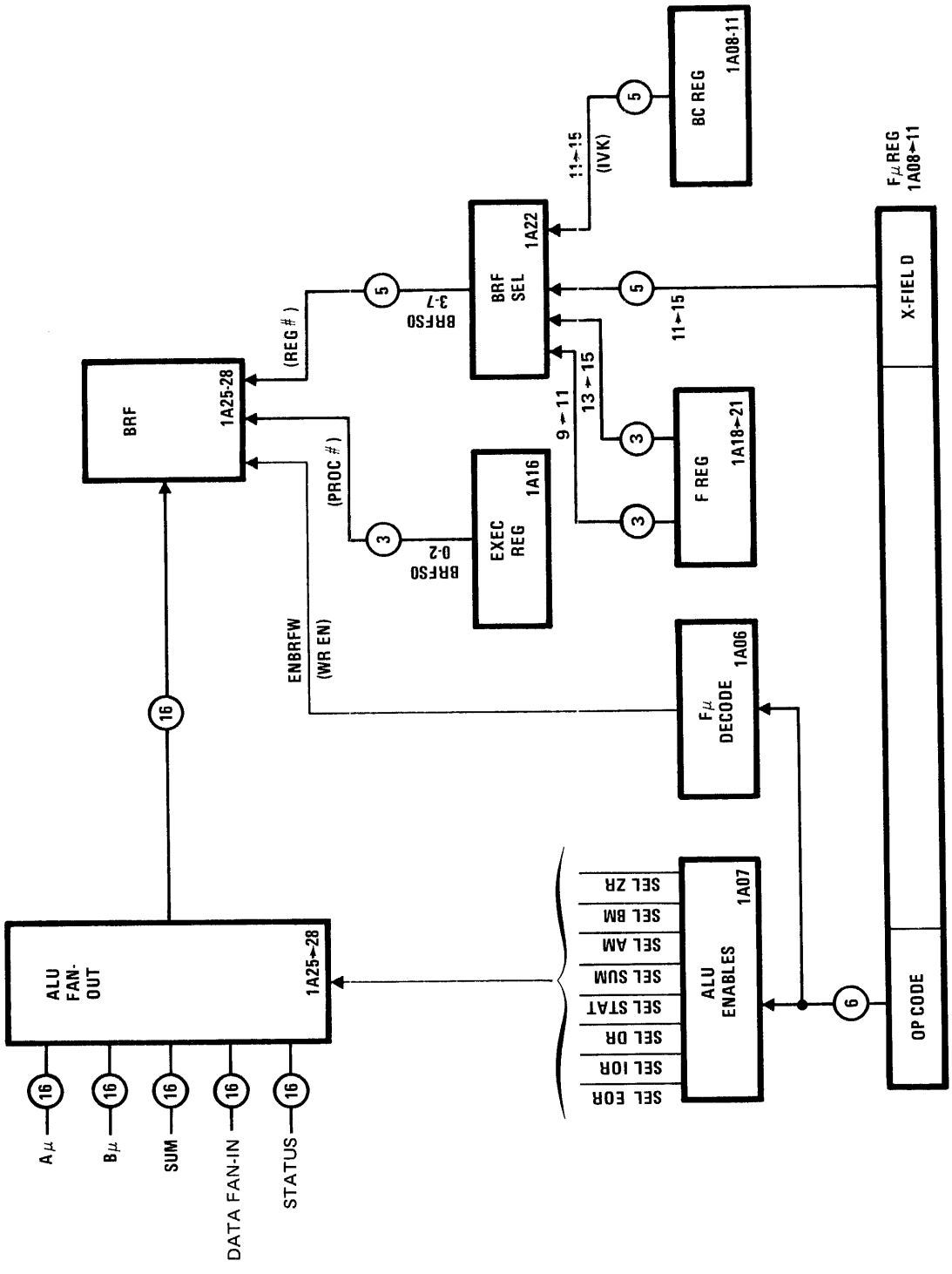


Figure 3-2. BRF Write

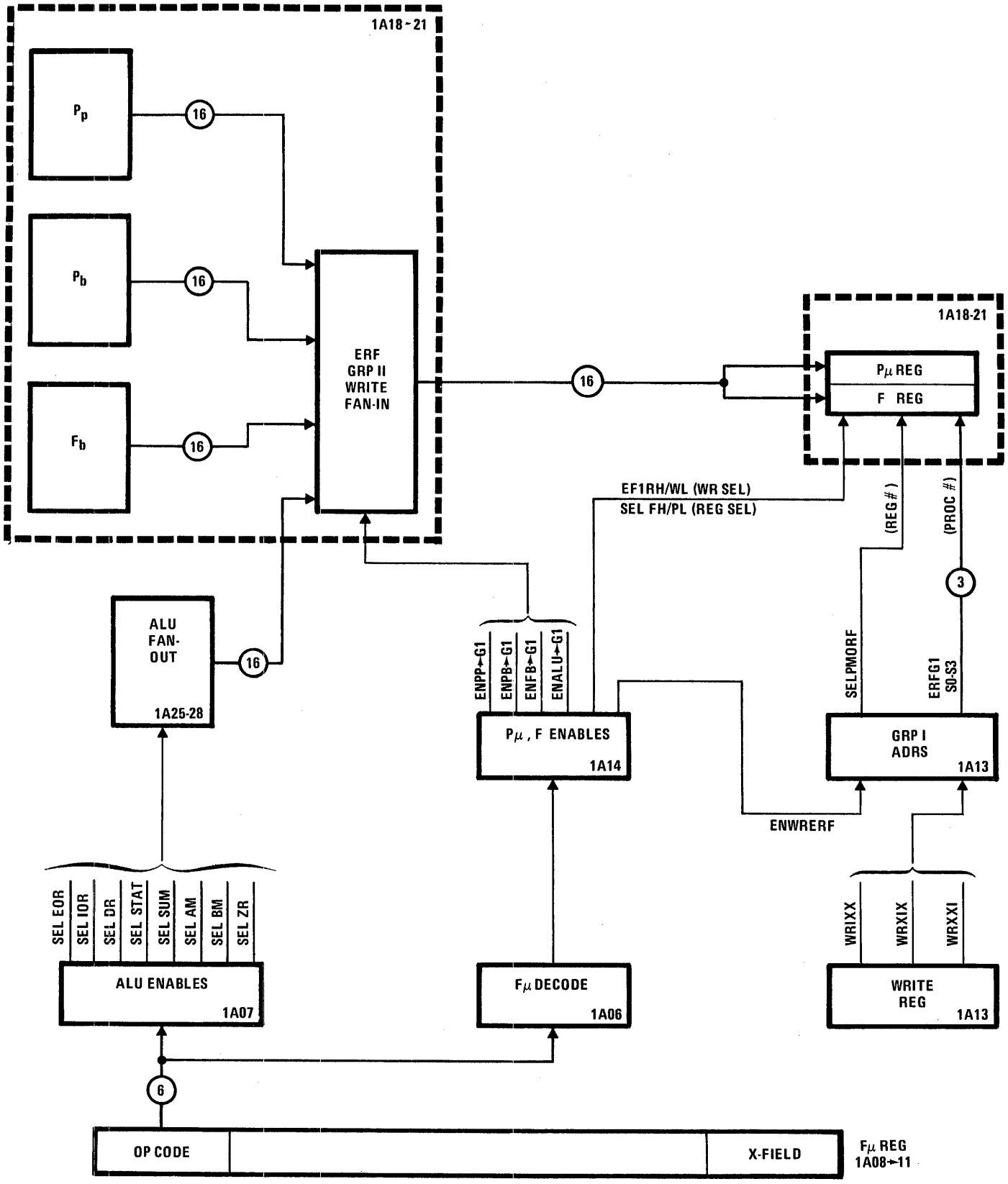


Figure 3-3. ERF Group I Write

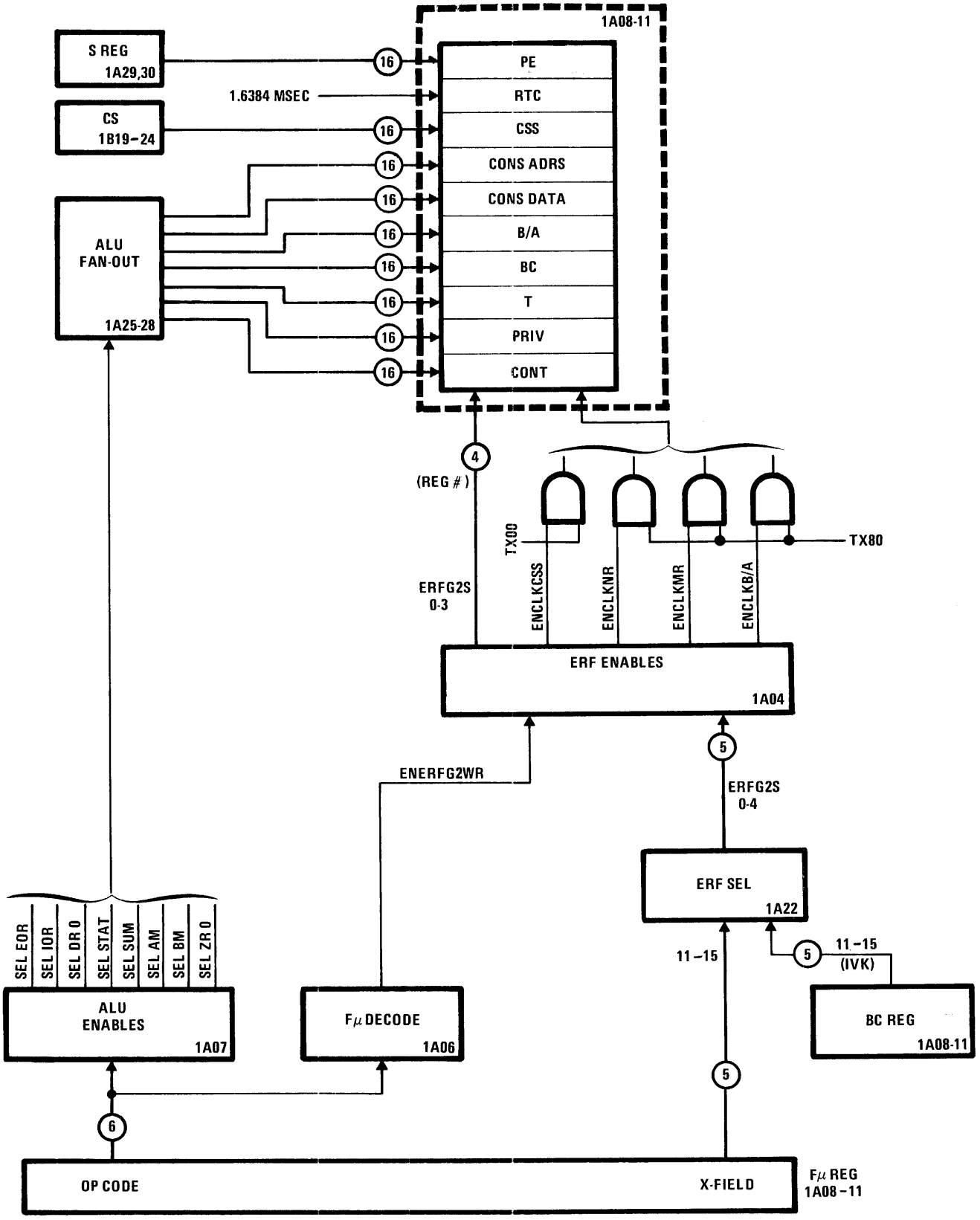


Figure 3-4. ERF Group II Write

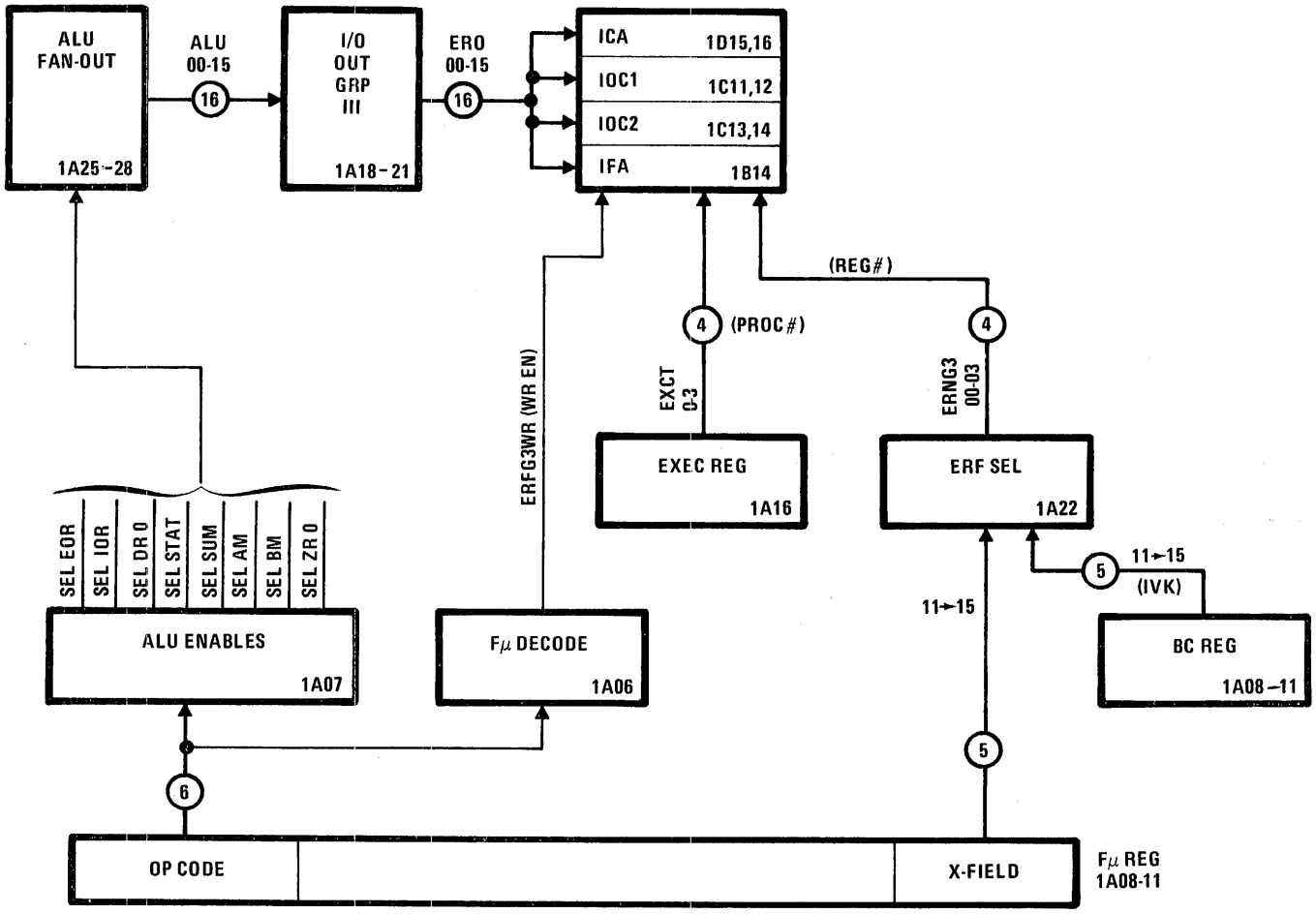


Figure 3-5. ERF Group III Write

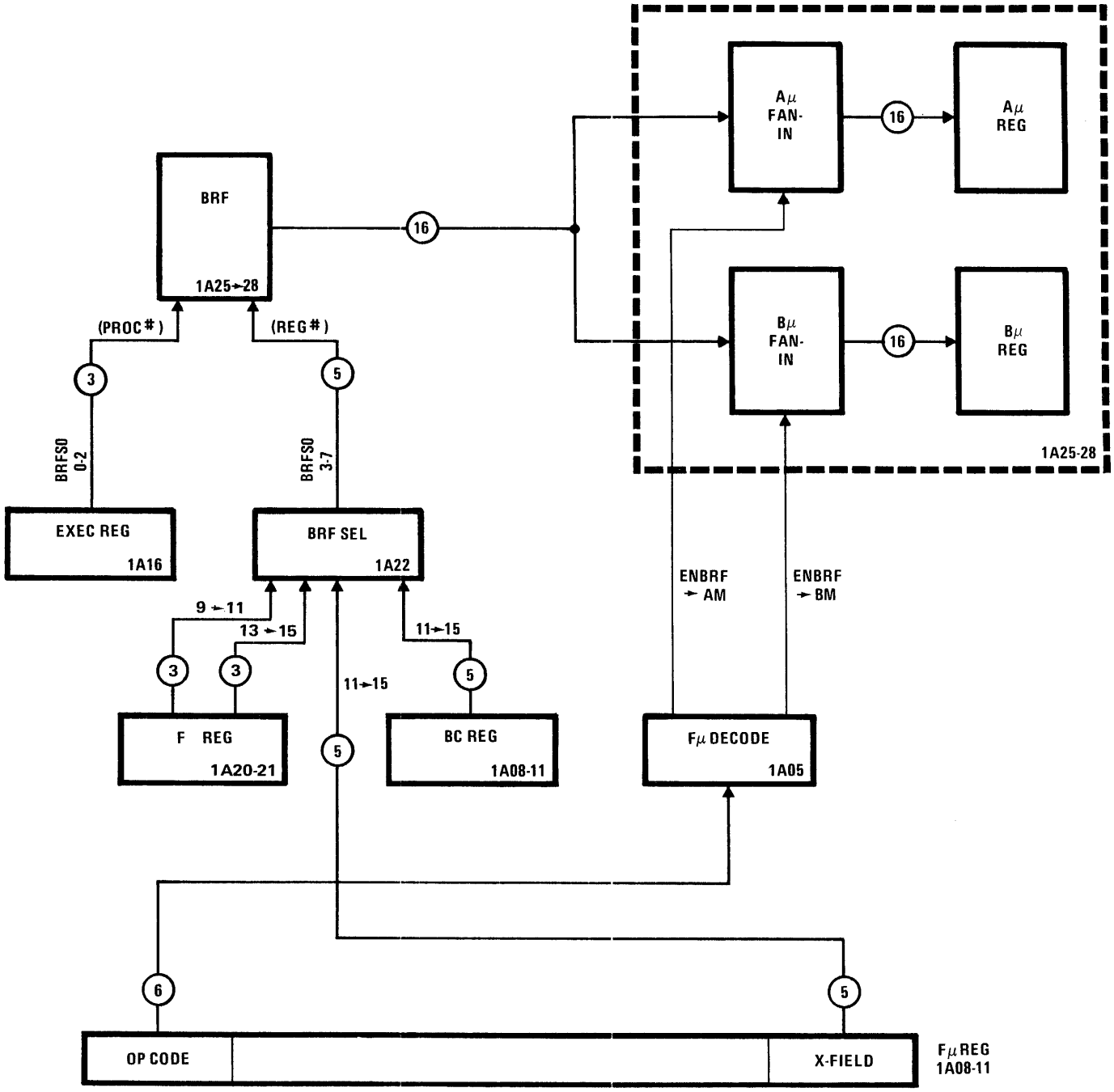


Figure 3-6. BRF Read (to Aμ and Bμ)

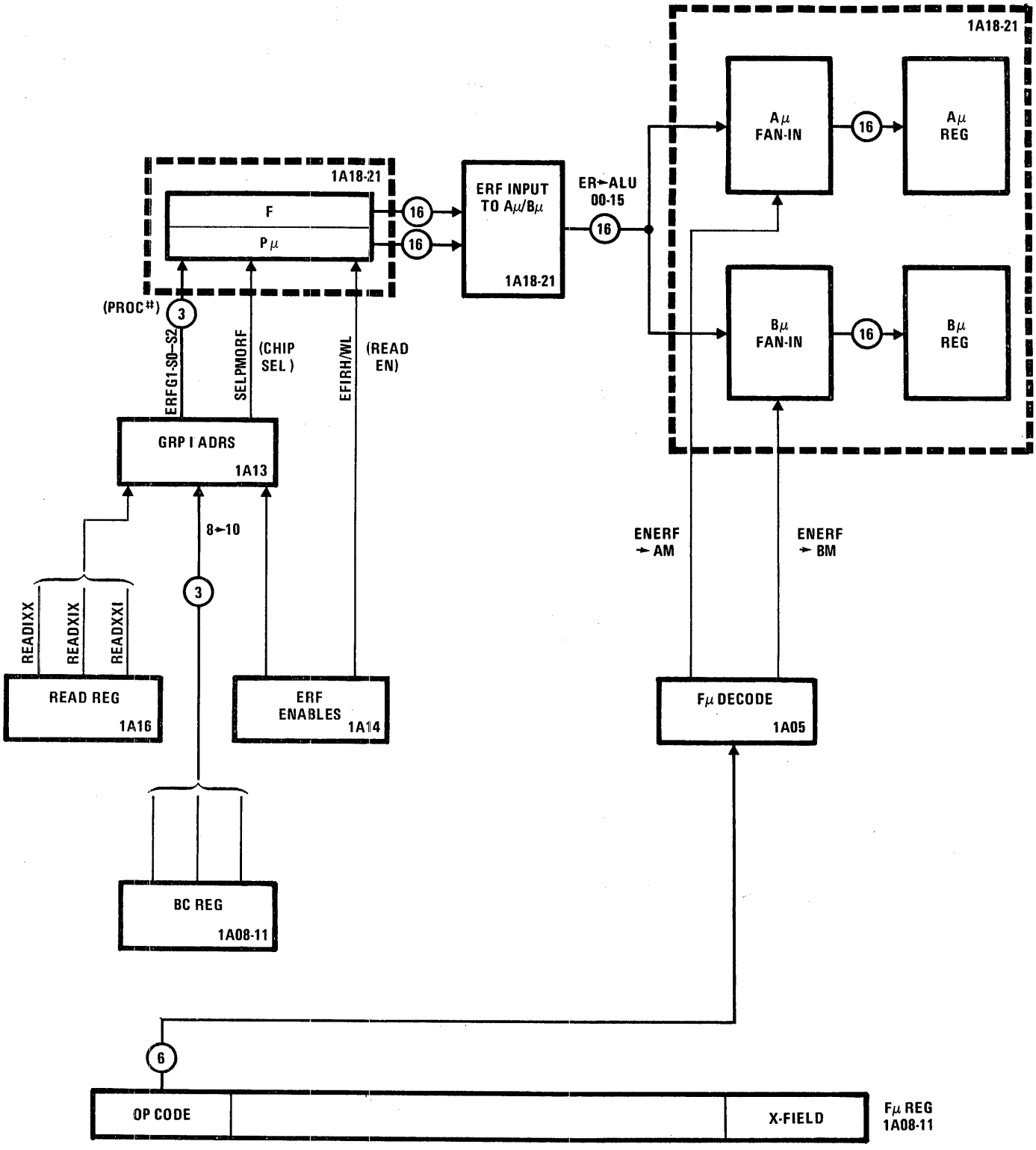


Figure 3-7. ERF Group I Read (to Aμ and Bμ)

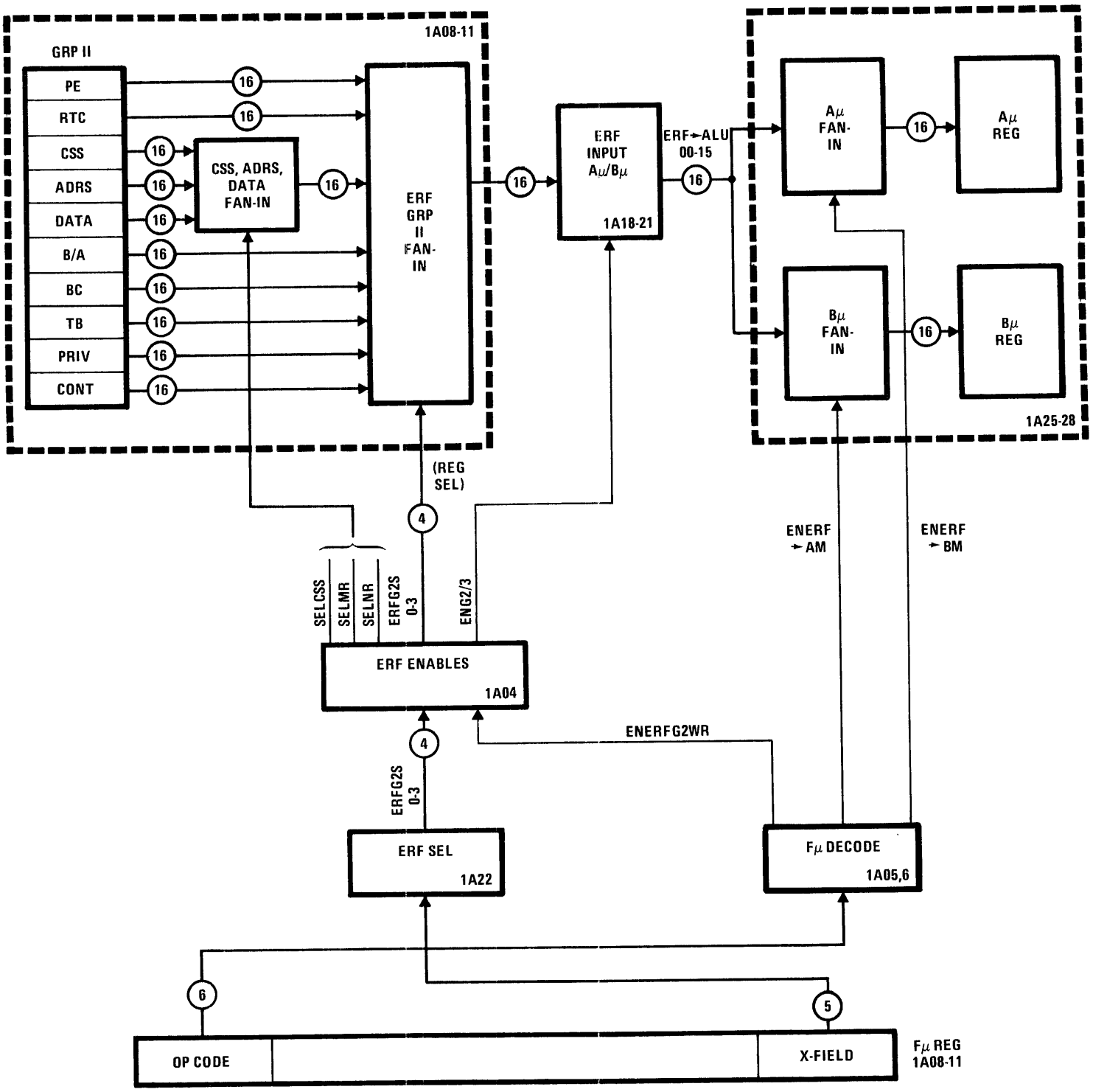


Figure 3-8. ERF Group II Read (to Aμ and Bμ)

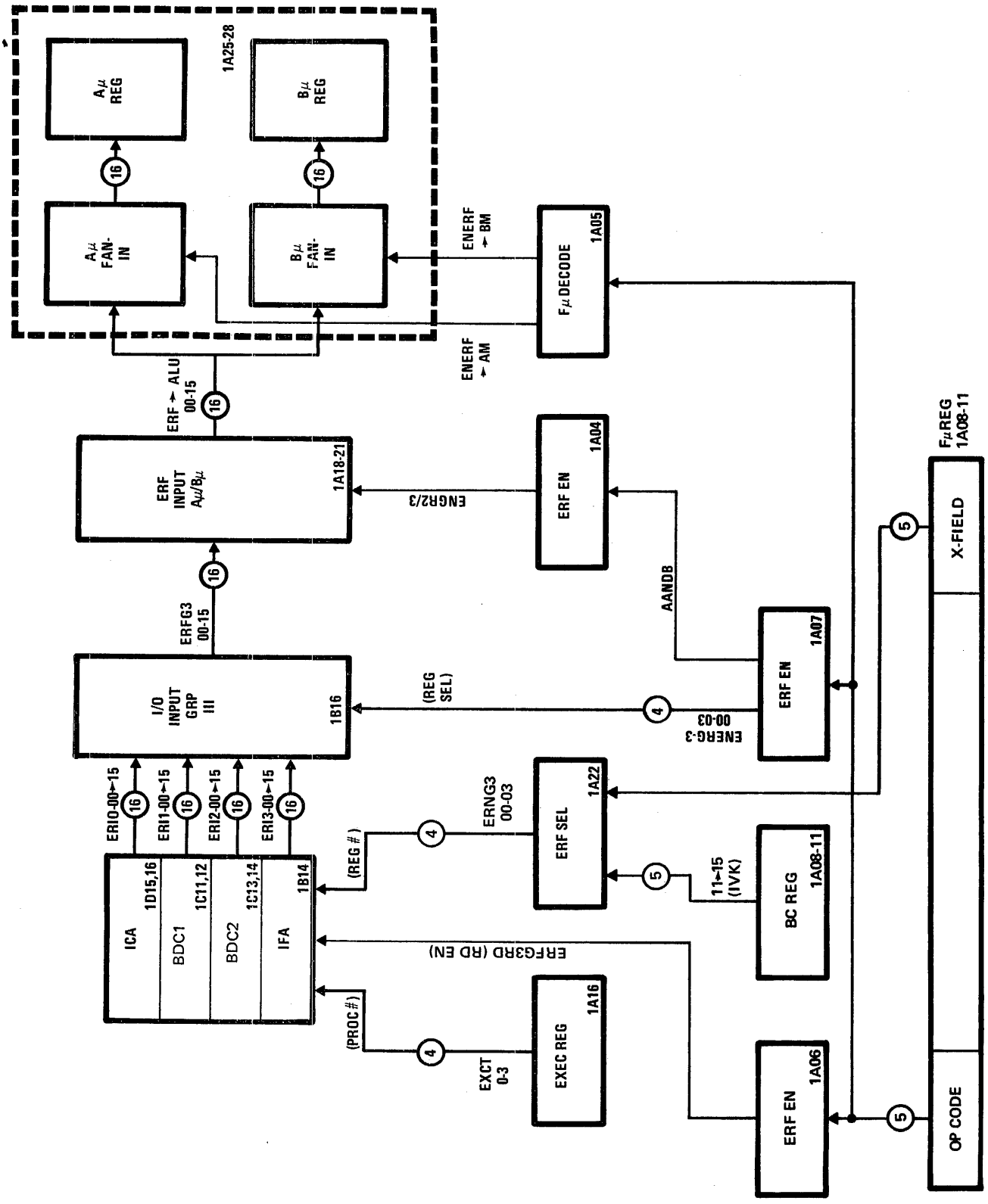


Figure 3-9. ERF Group III Read (to A μ and B μ)

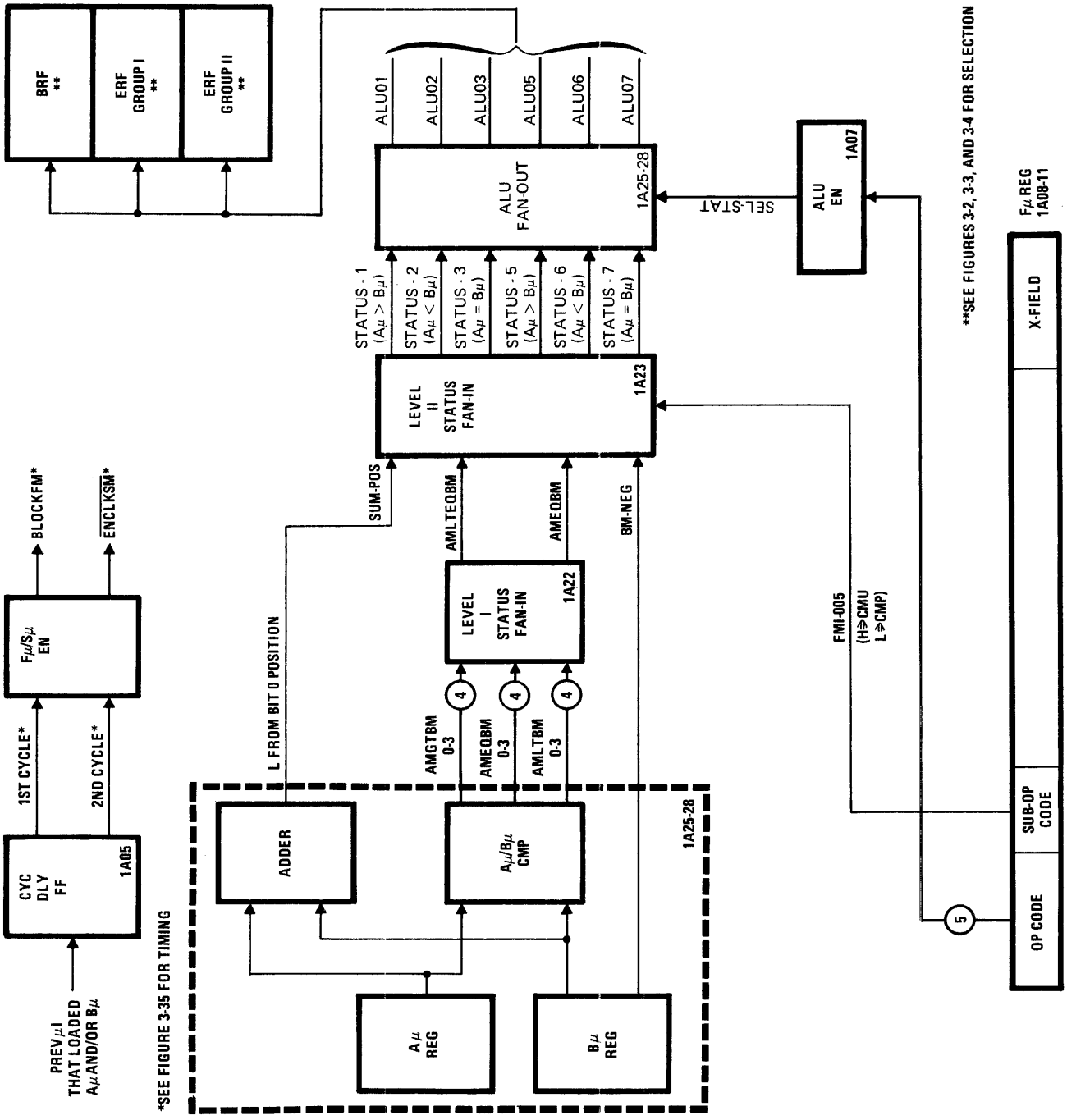


Figure 3-10. CMP and CMU μI's

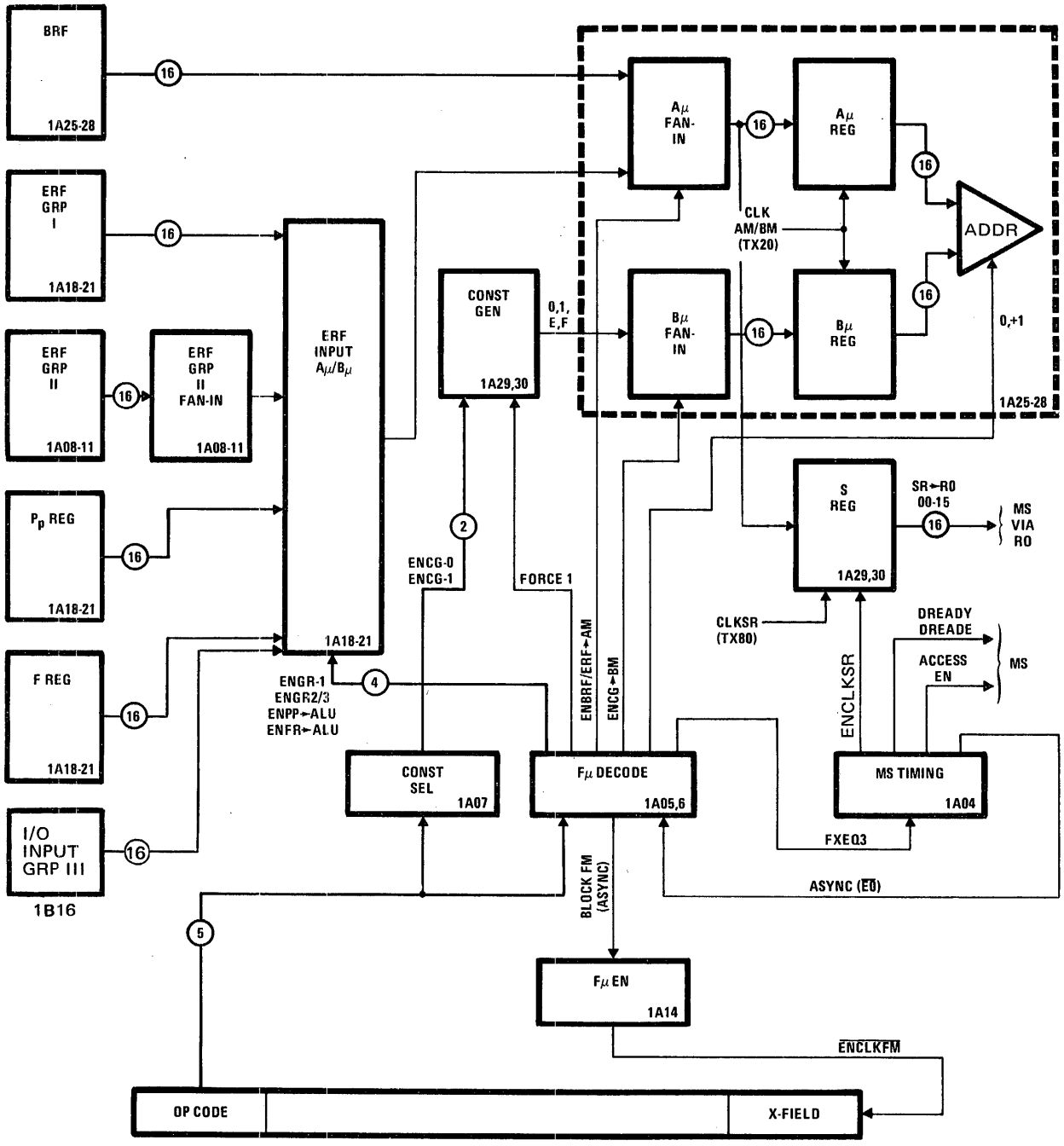


Figure 3-11. Load S μI

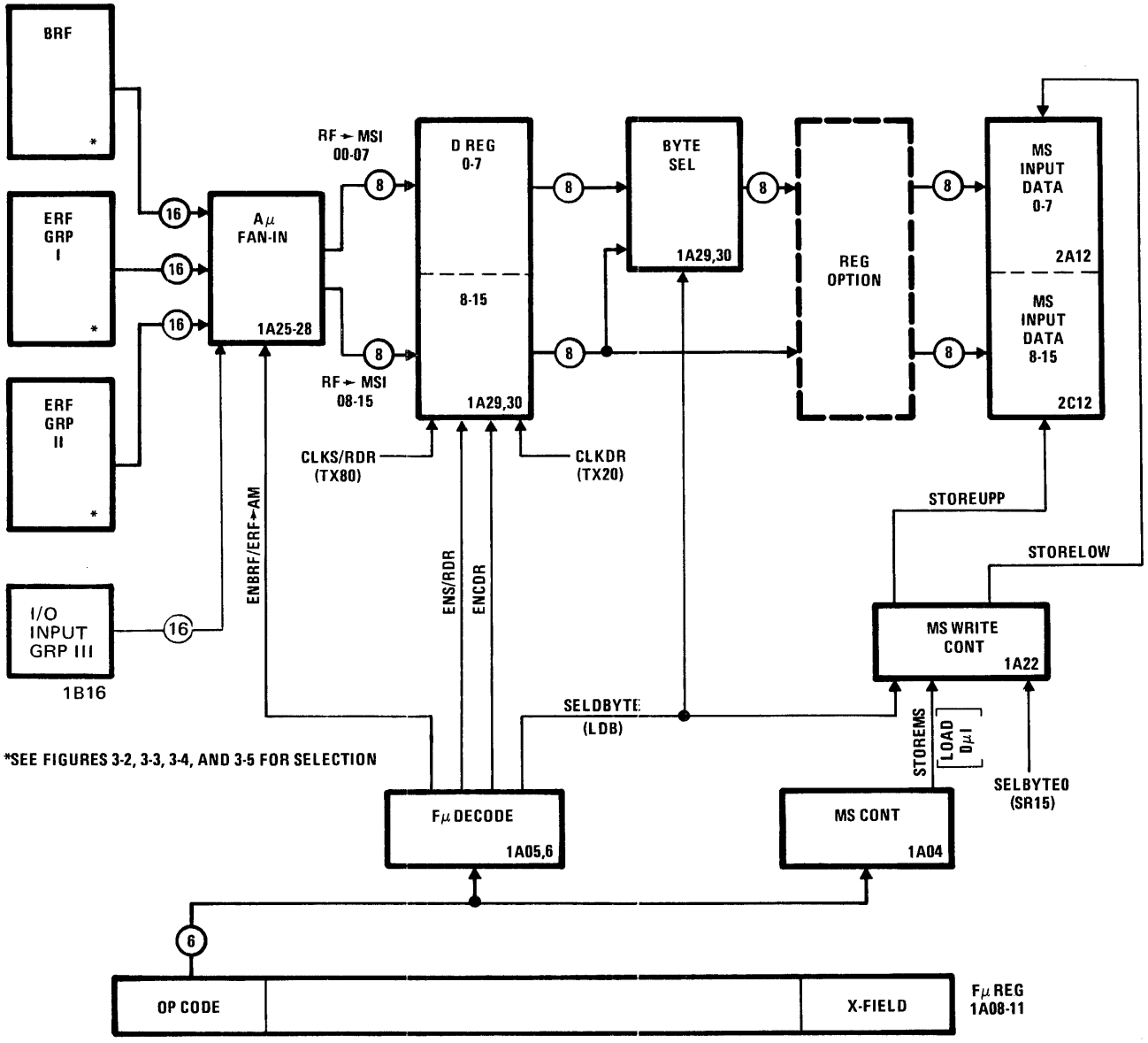


Figure 3-12. LDW and LDB μI's

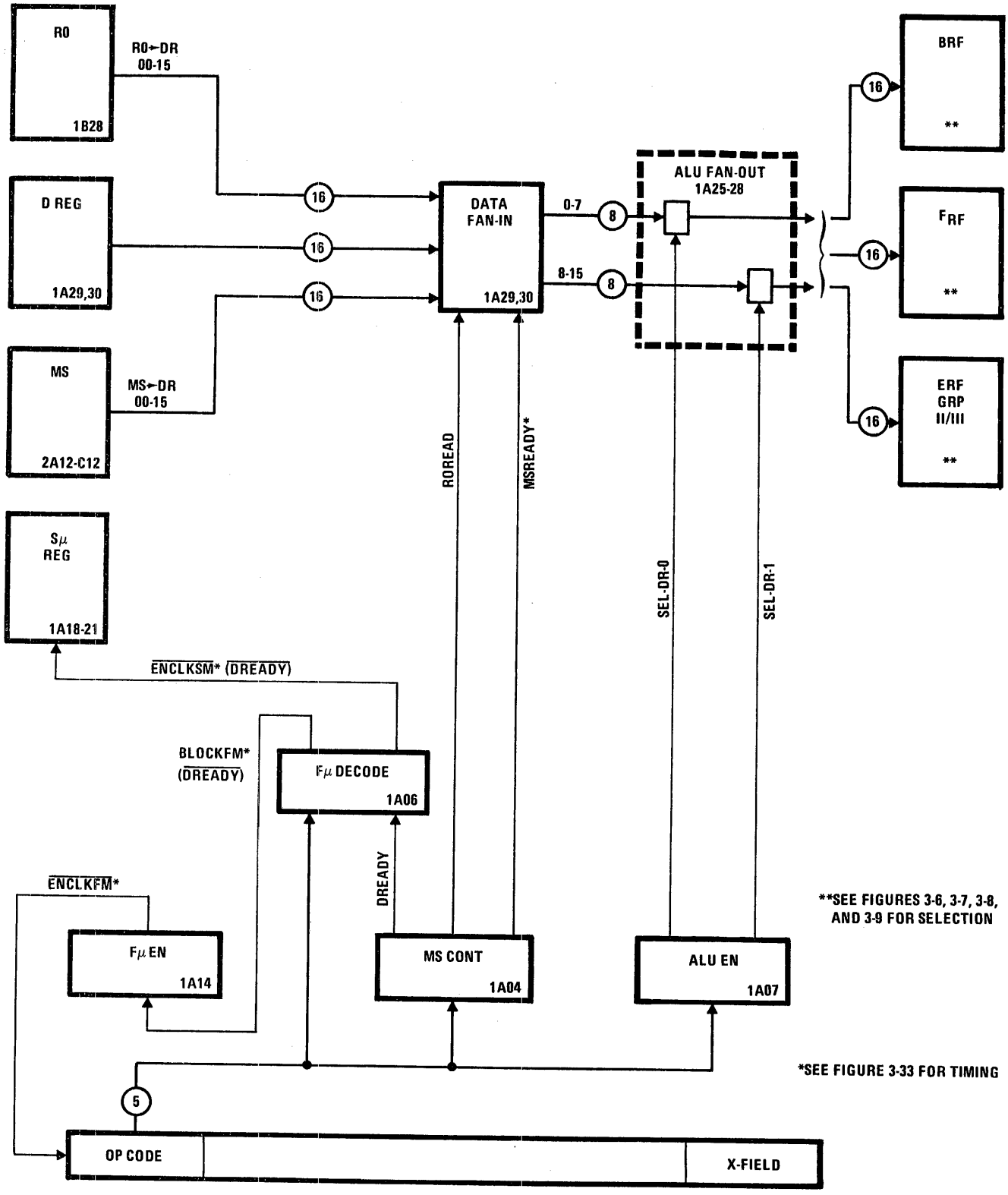


Figure 3-13. SDW μI

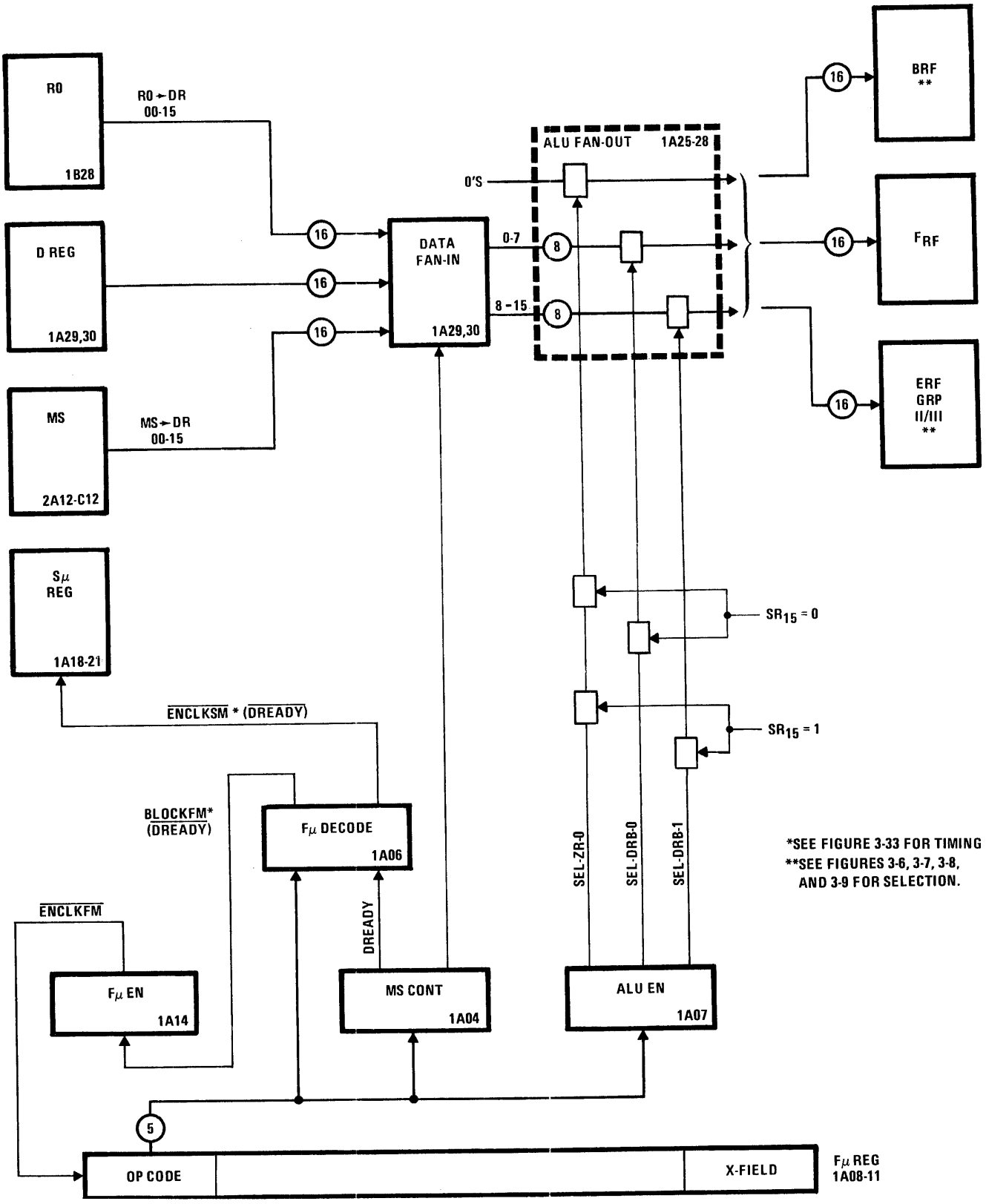


Figure 3-14. SDB μI

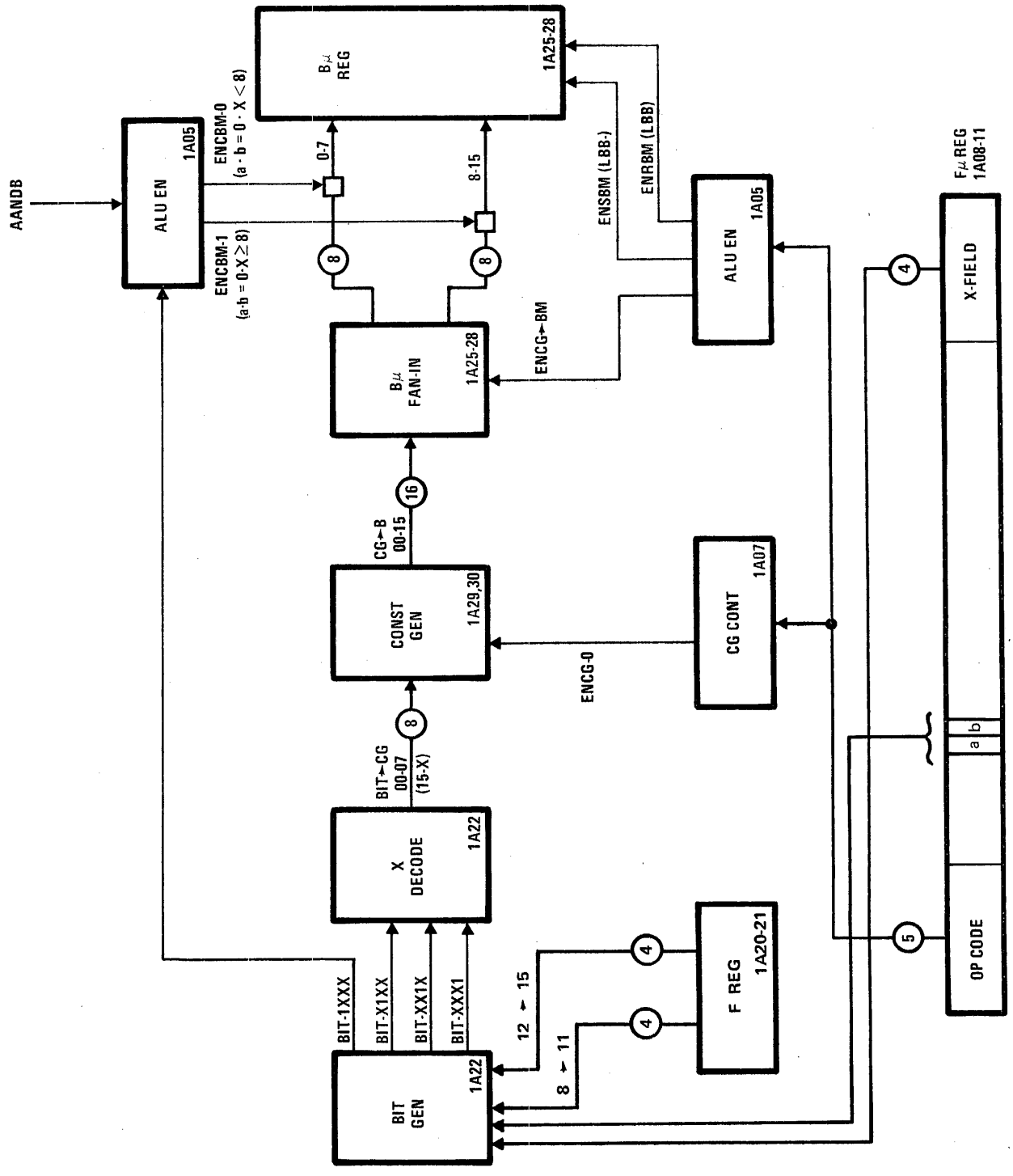


Figure 3-15. LBB and LBB- μI's (a·b ≠ 1.1)

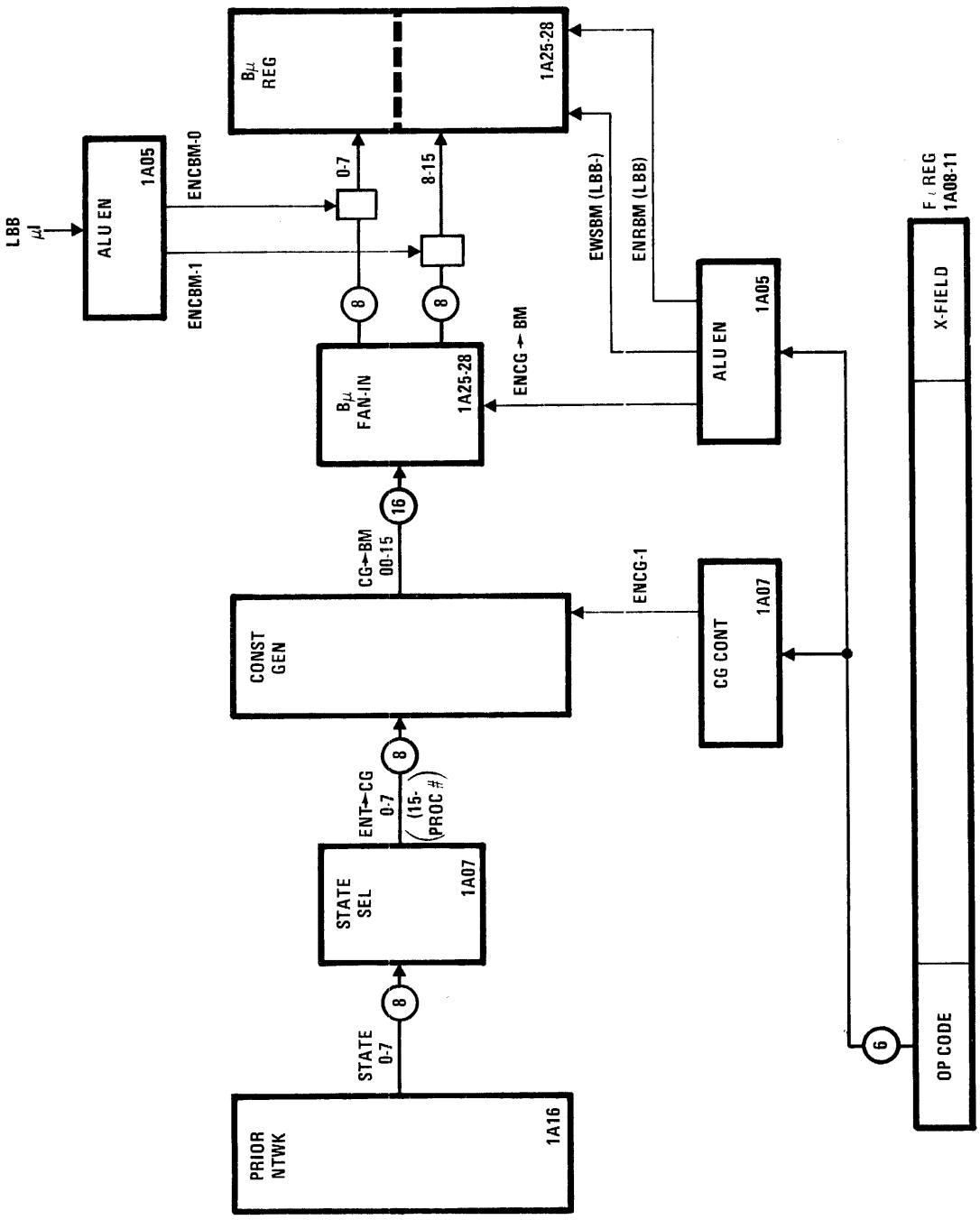


Figure 3-16. LBB and LBB-μI's (a · b = 1 · 1)

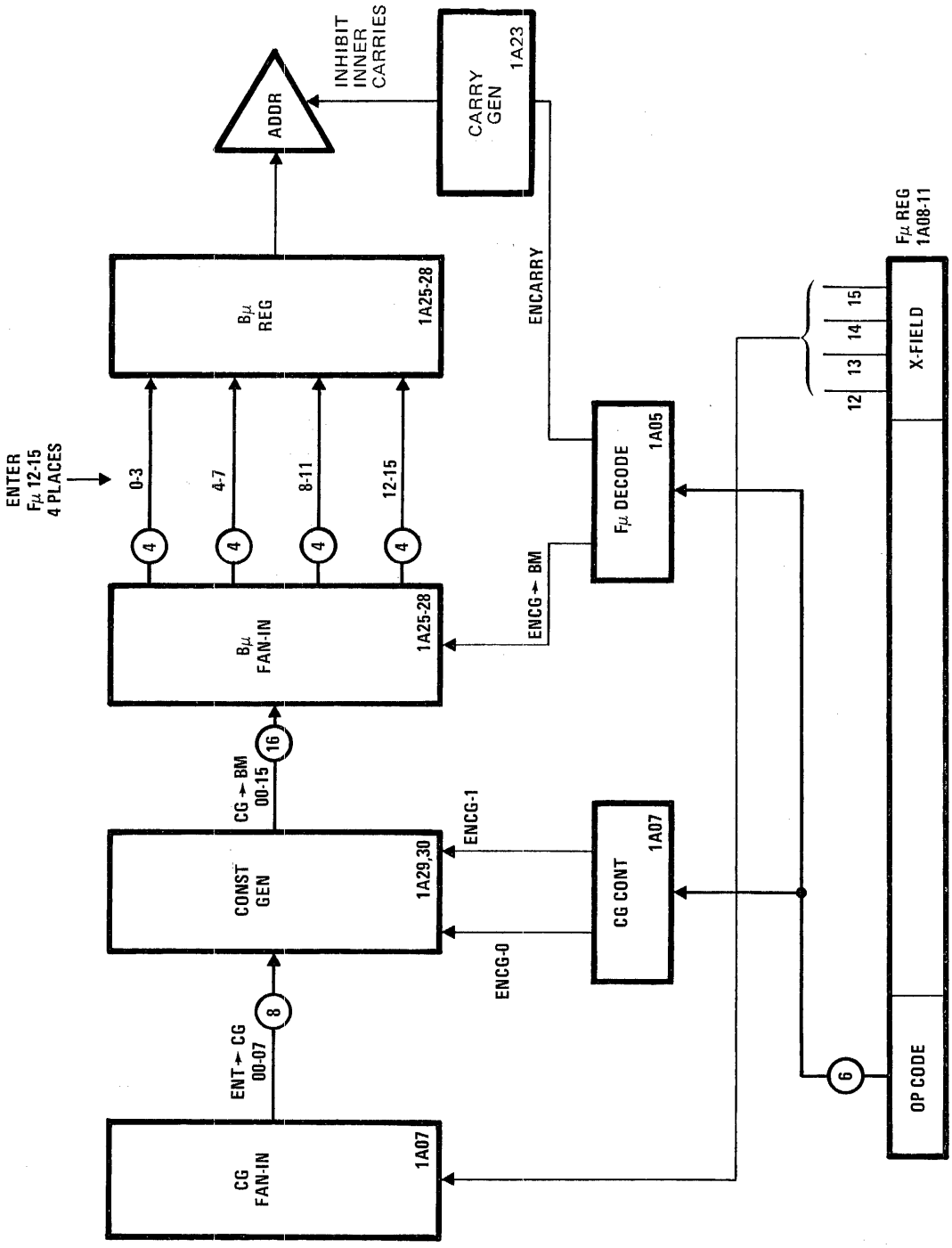


Figure 3-17. DIG μI

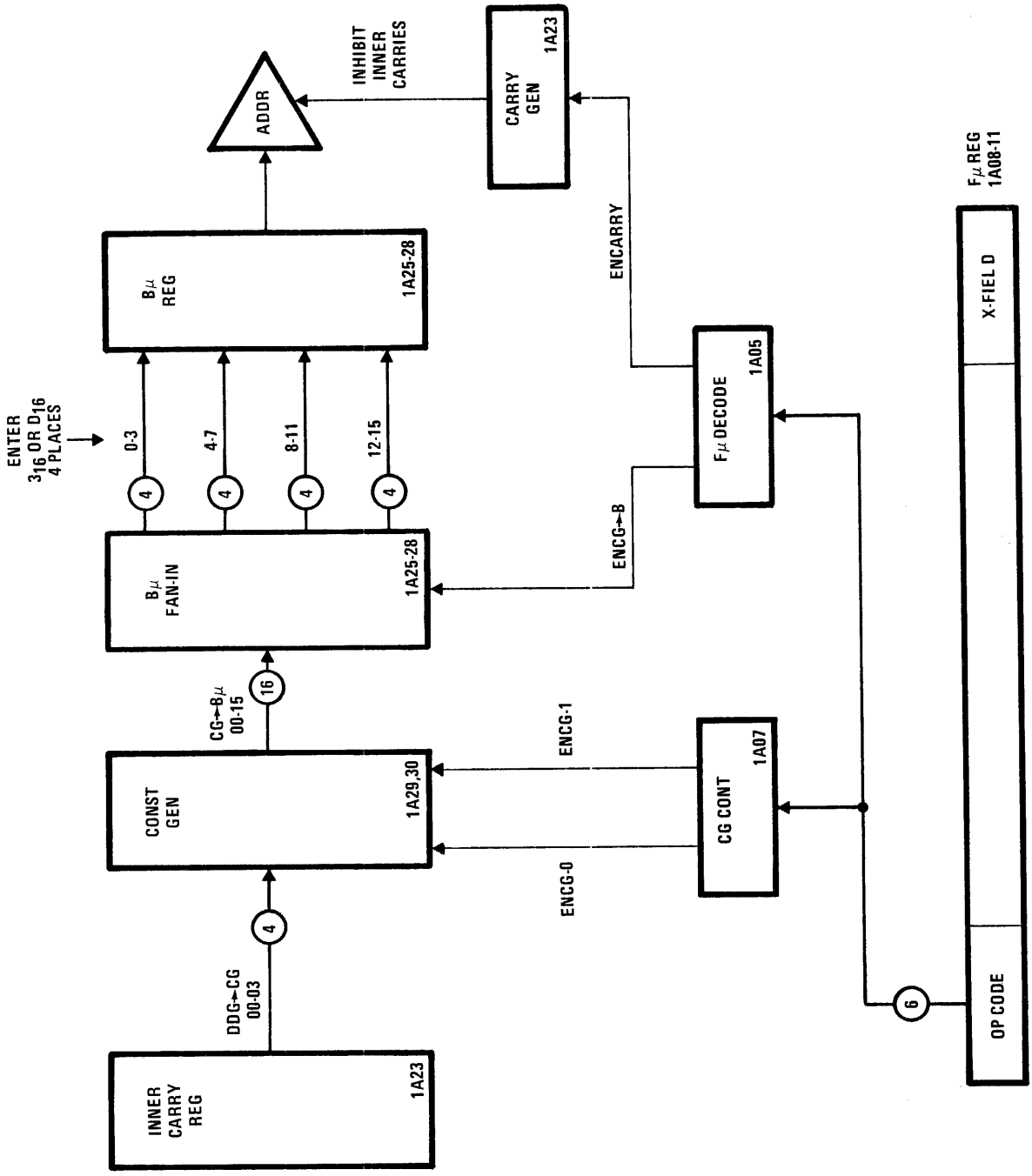


Figure 3-18. CORC μI

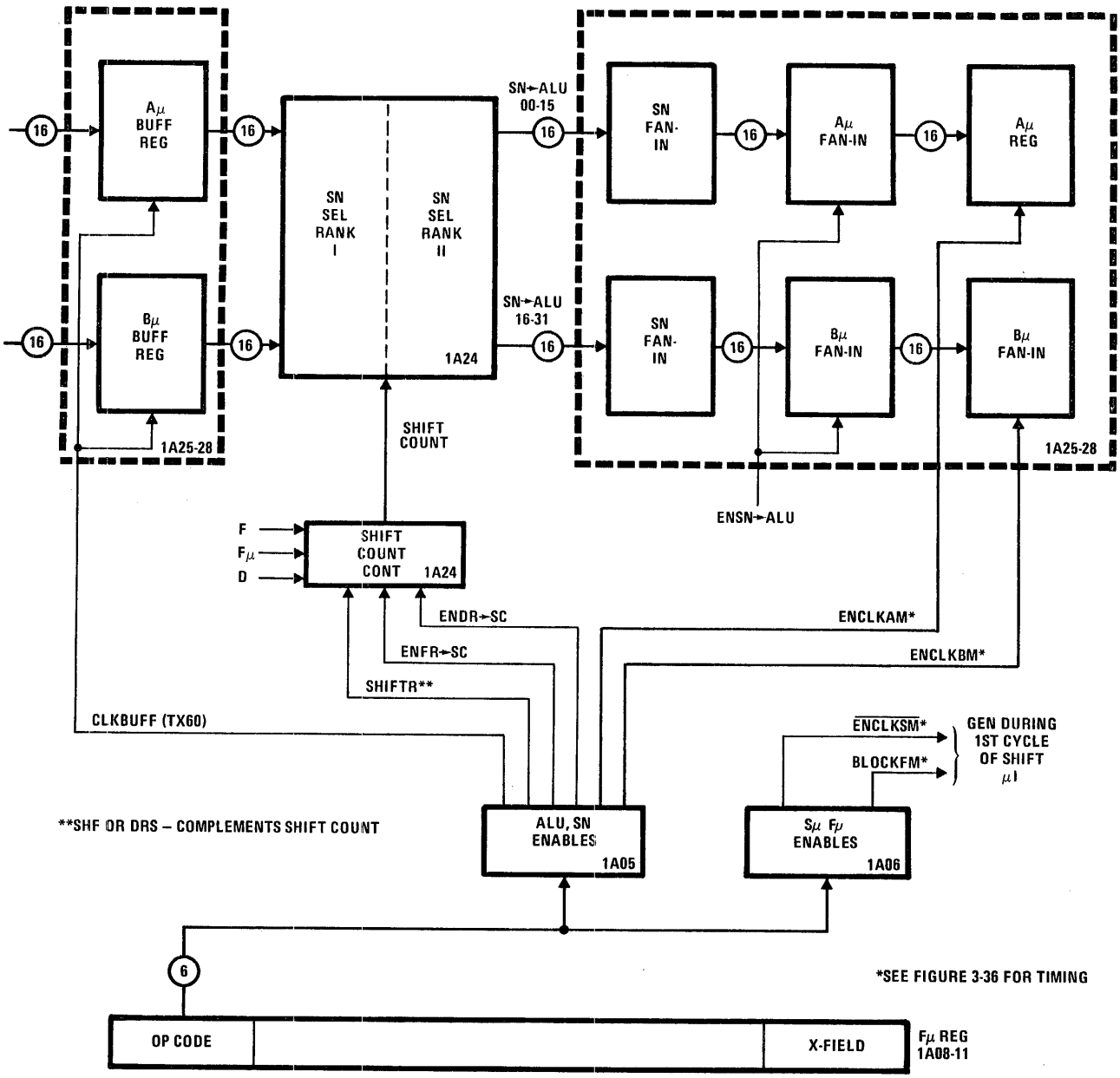


Figure 3-19. Shift

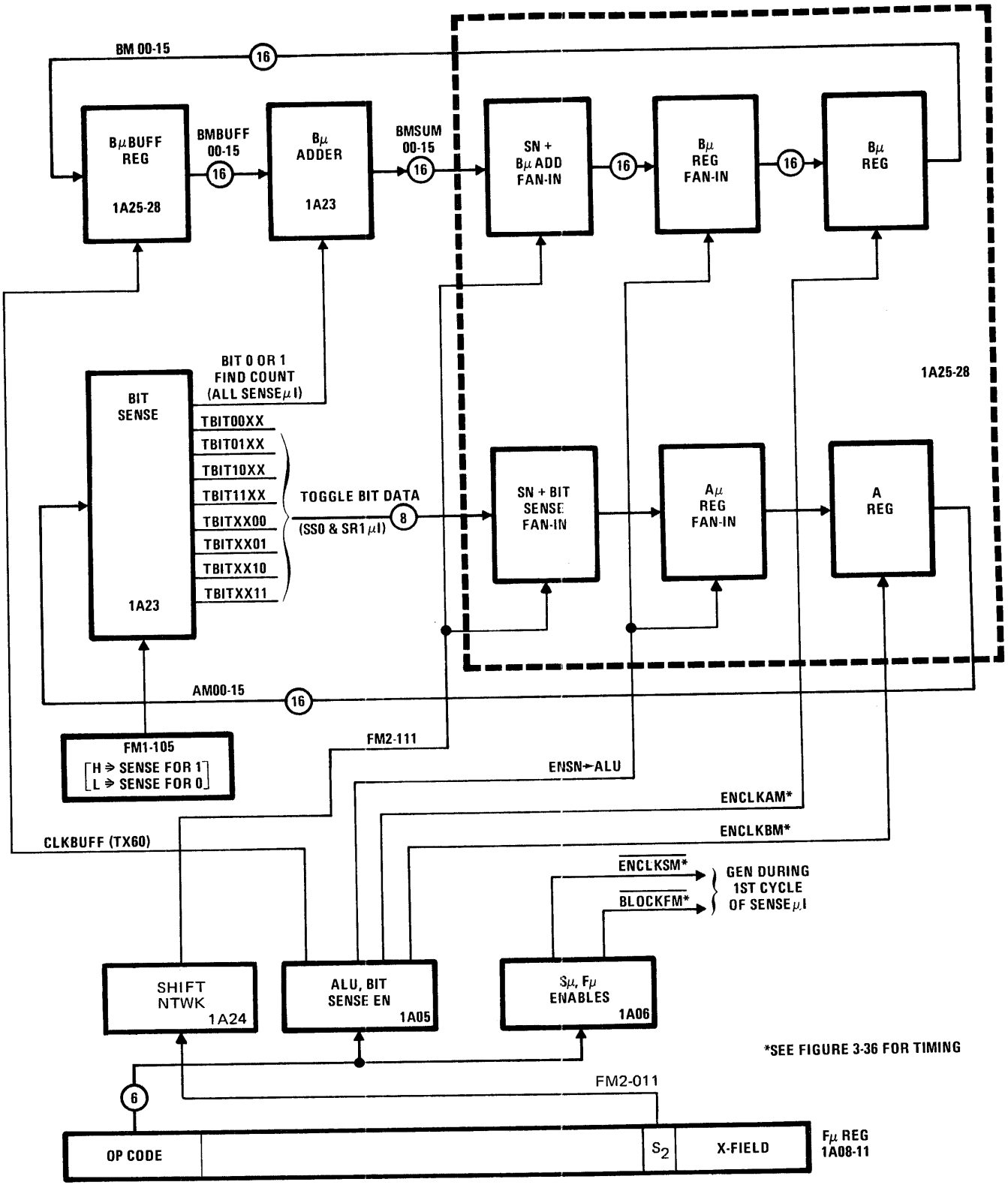


Figure 3-20. Bit Sense and Sense/Toggle μI's

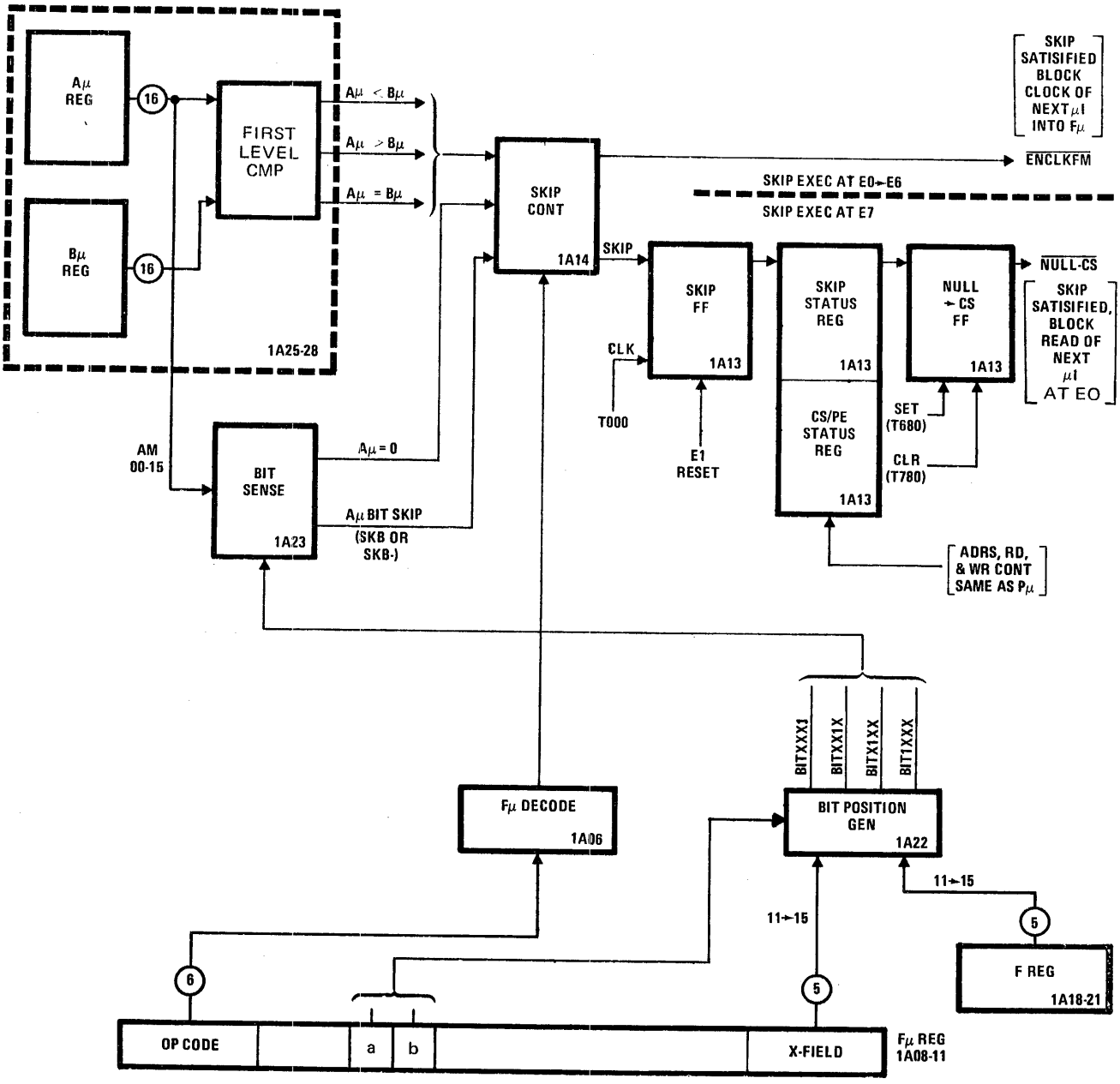


Figure 3-21. Skip μI's

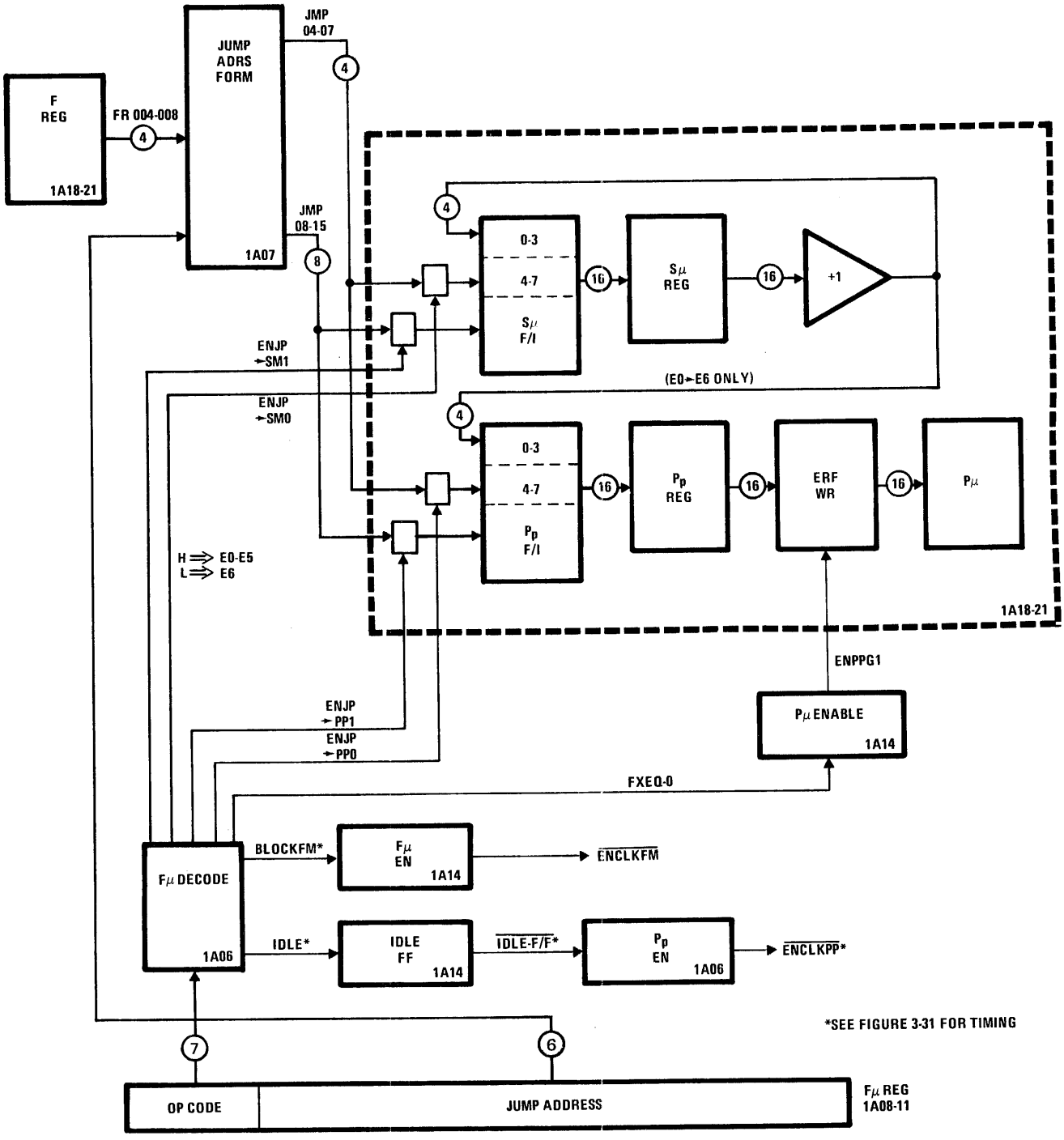


Figure 3-22. FNJ μI's (E0–E6)

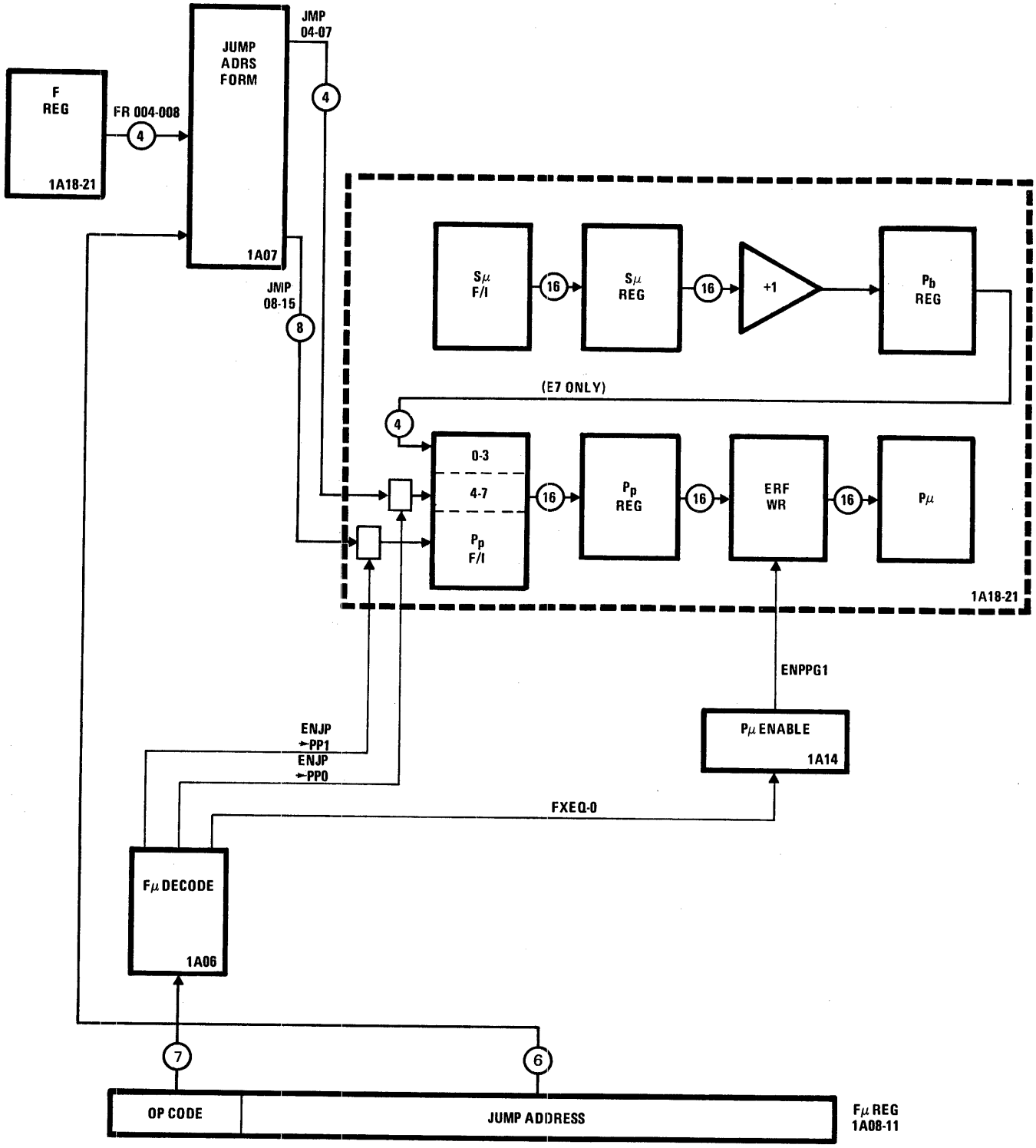


Figure 3-23. FNJ μI's (E7)

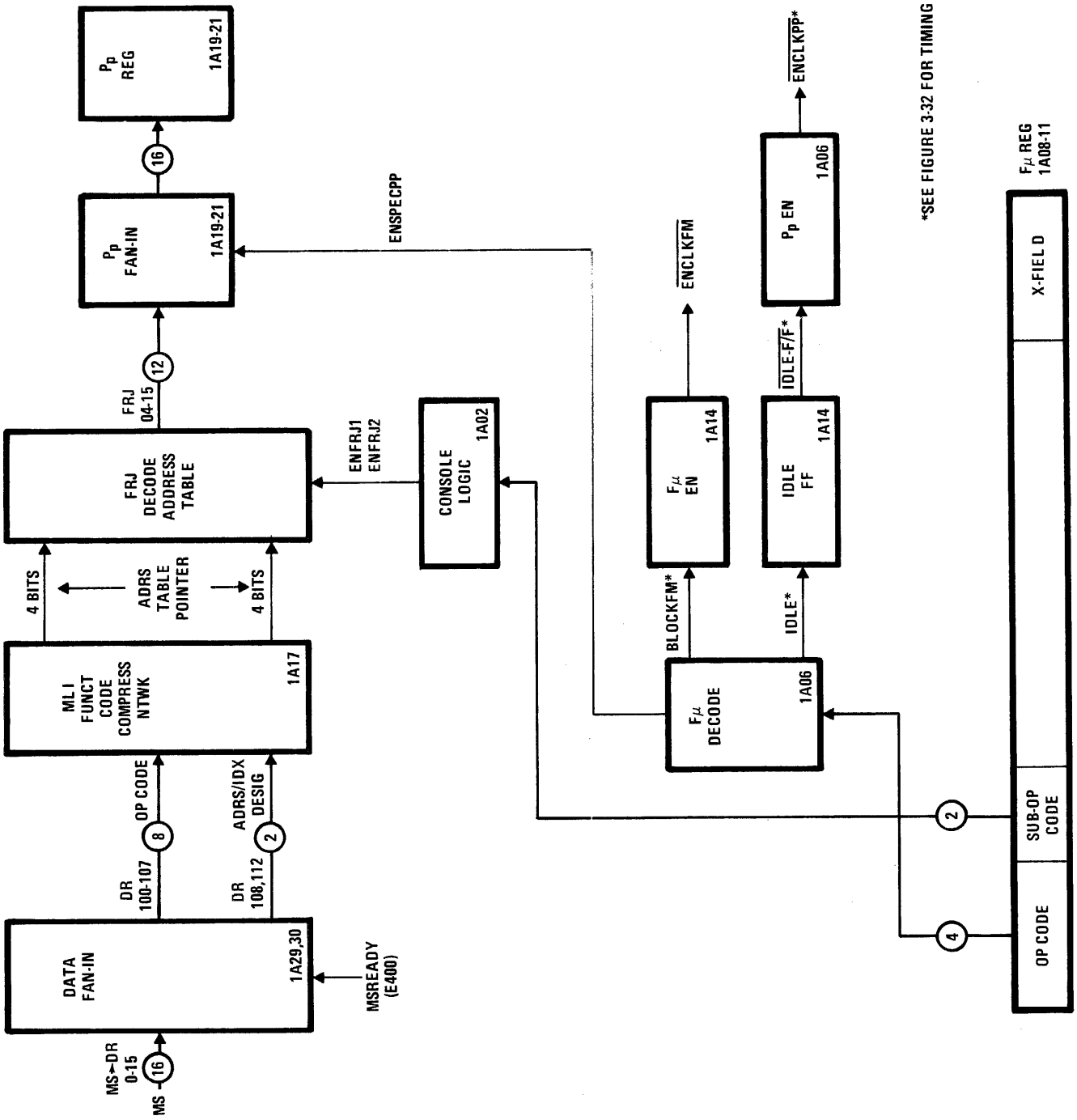


Figure 3-24. FRJ μI

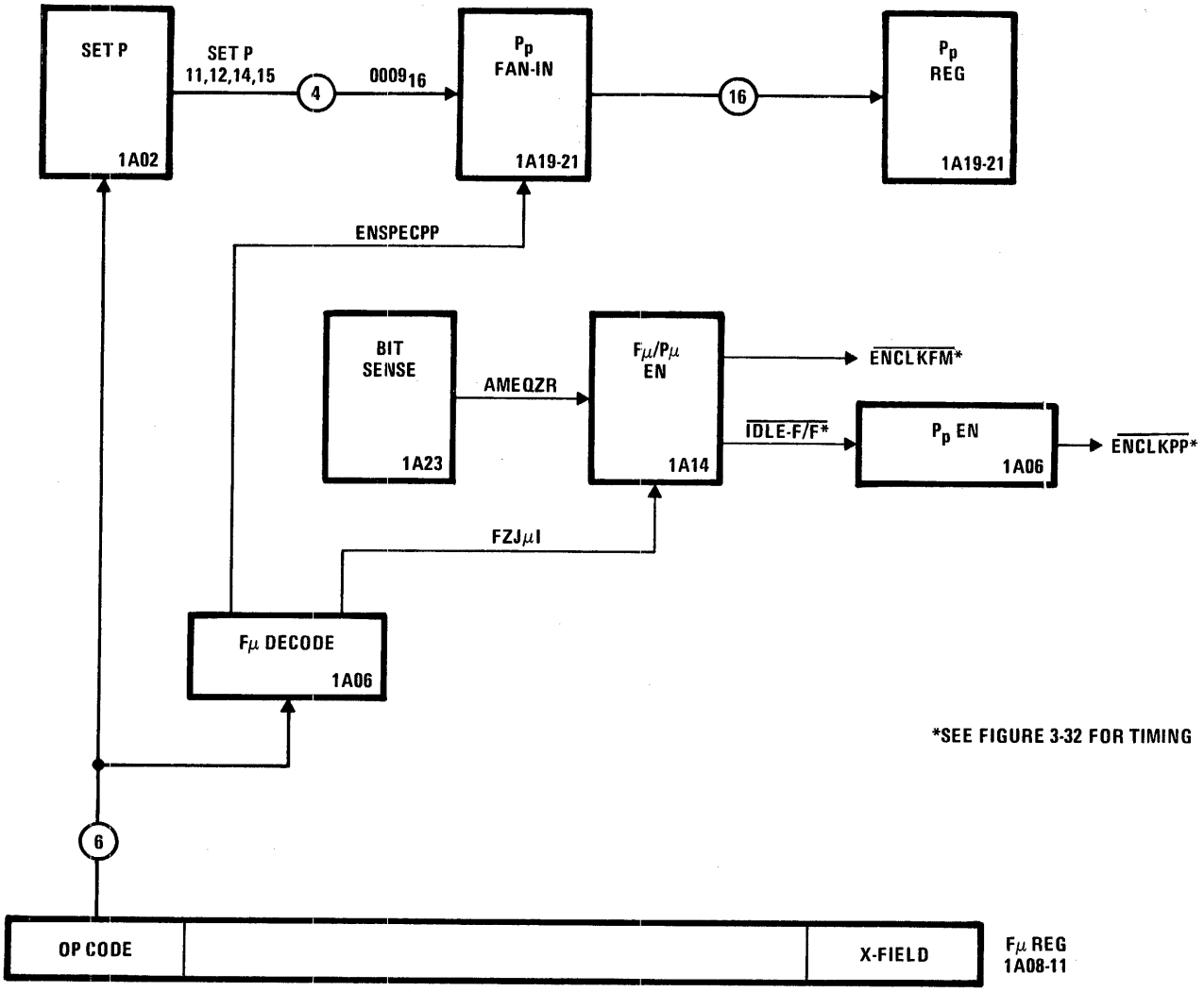


Figure 3-25. FZJ μI

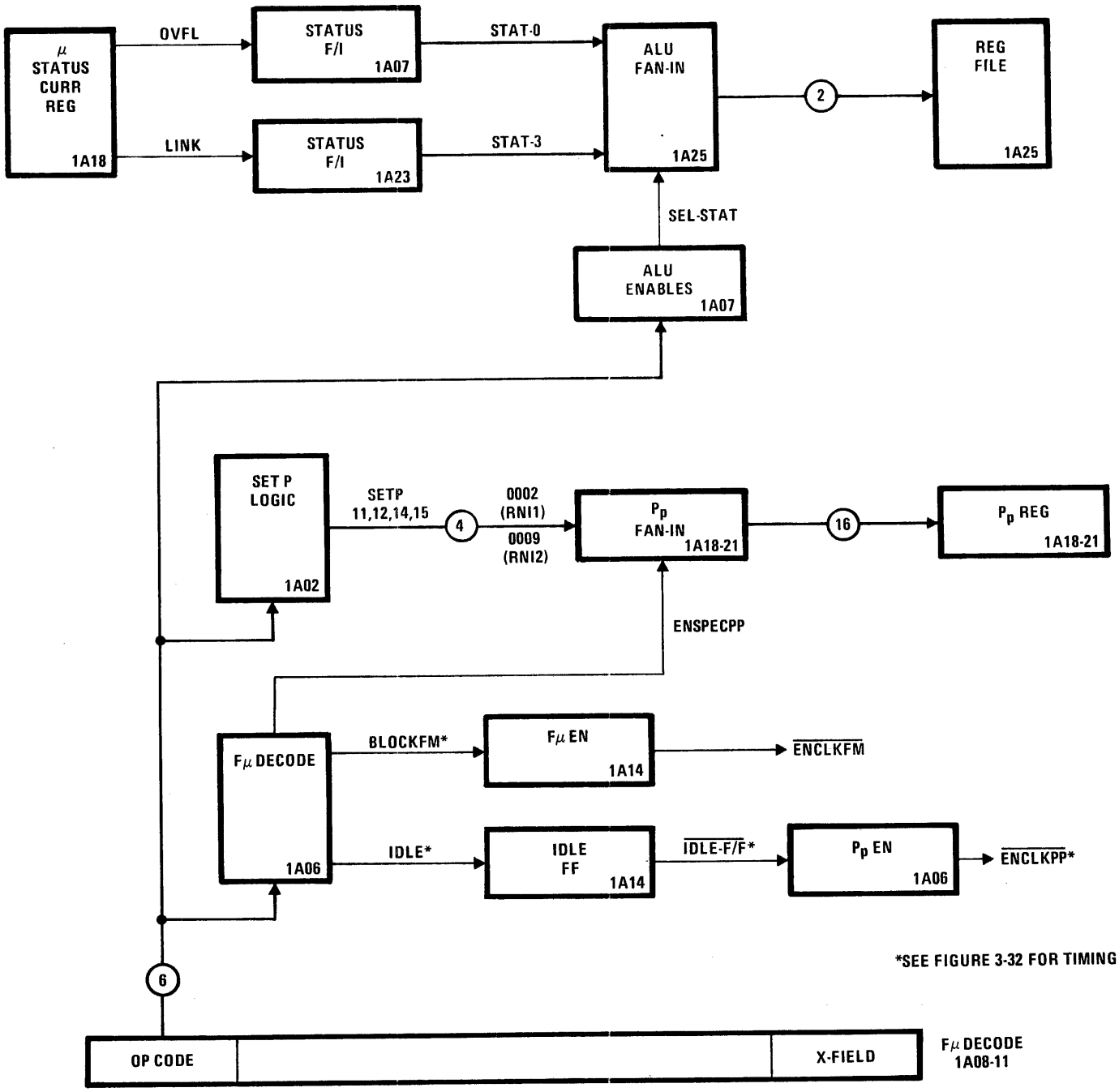


Figure 3-26. RNI1 and RNI2 μI's

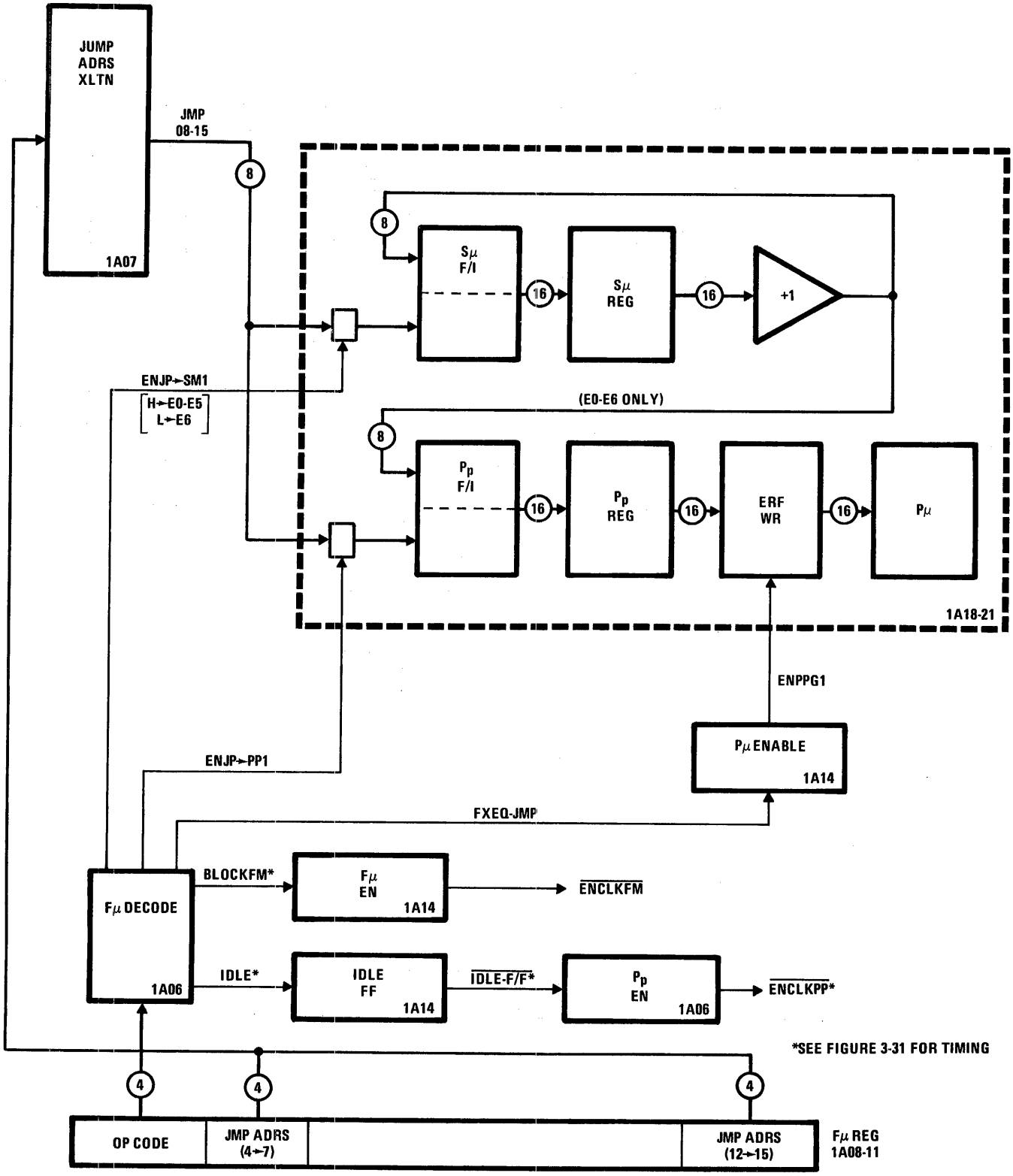


Figure 3-27. JMP μI (E0—E6)

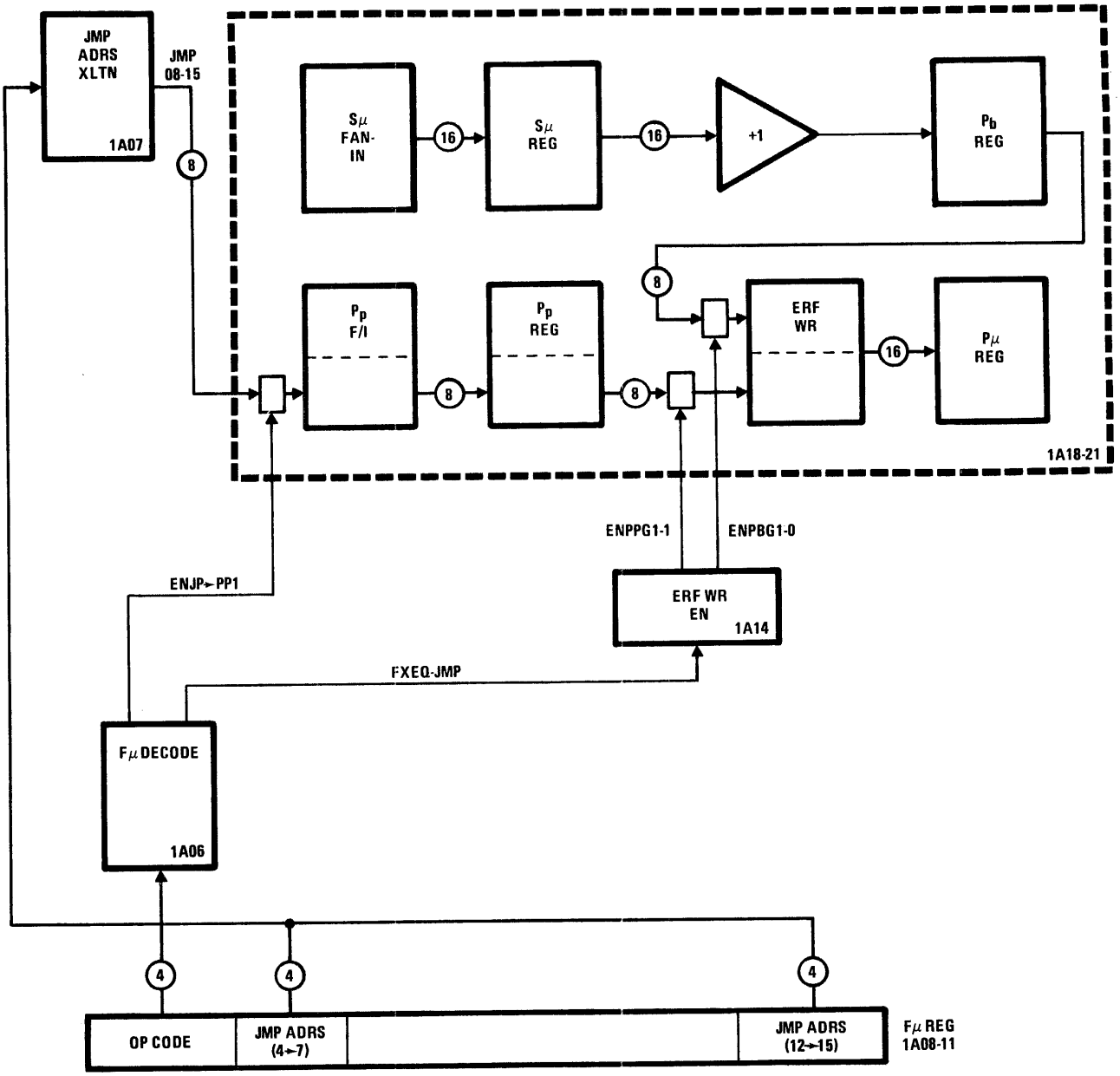


Figure 3-28. JMP μI (E7)

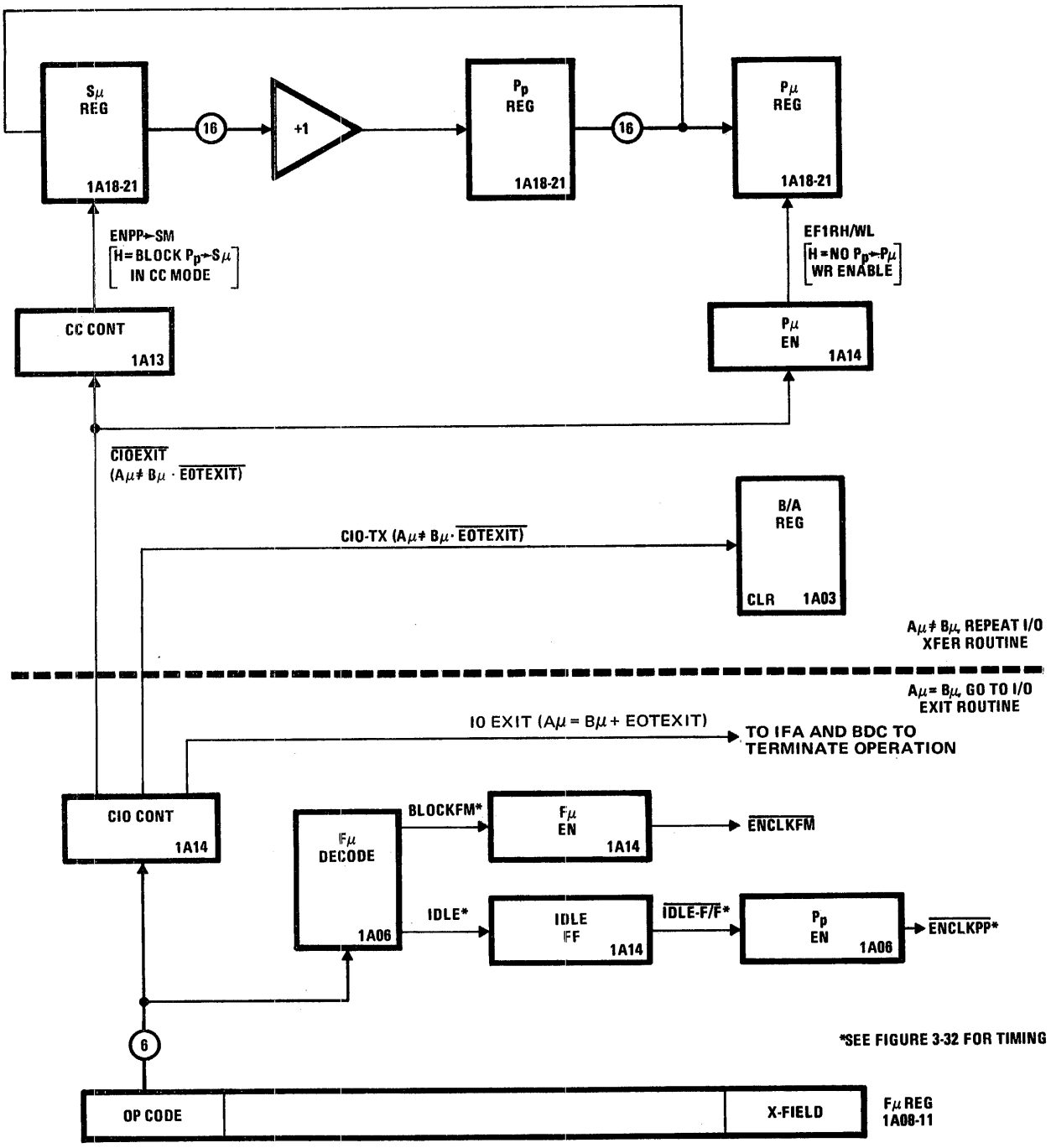


Figure 3-29. CIO1

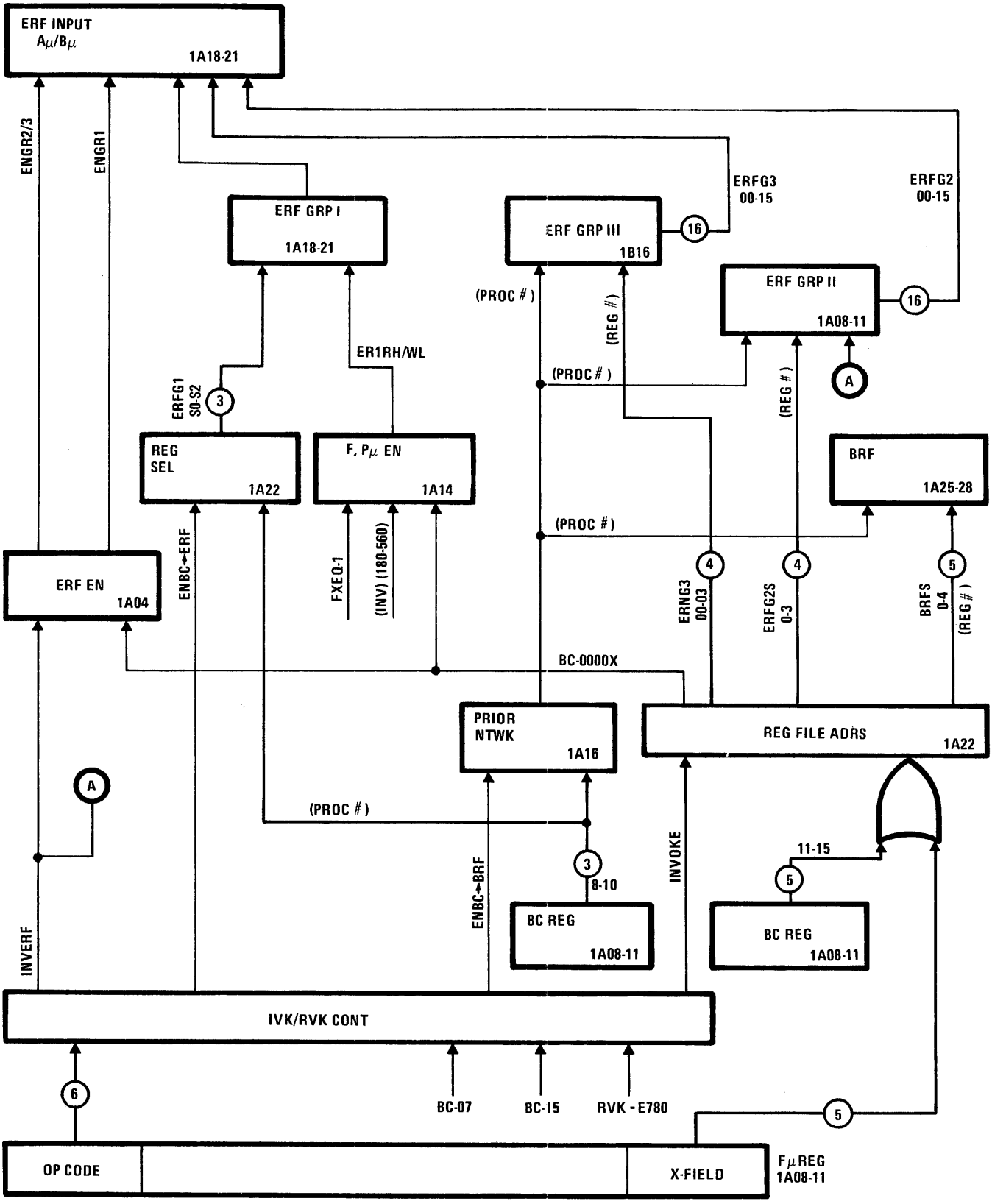
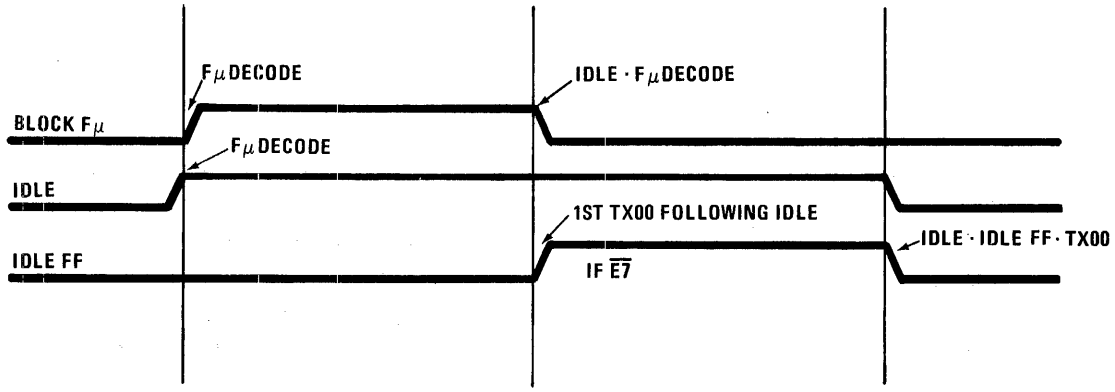


Figure 3-30. INV and RVk μI



NOTE: IF INSTRUCTION IS DECODED AT E7 TIME, BLOCK Fμ IS NEGATED BY E0-E TIME AND Fμ IS CLOCKED AS USUAL AT THE END OF E7 TIME. IN THAT CASE THE INSTRUCTION TAKES ONLY 100 NS.

Figure 3-31. JMP and FNJ μI Timing Diagram

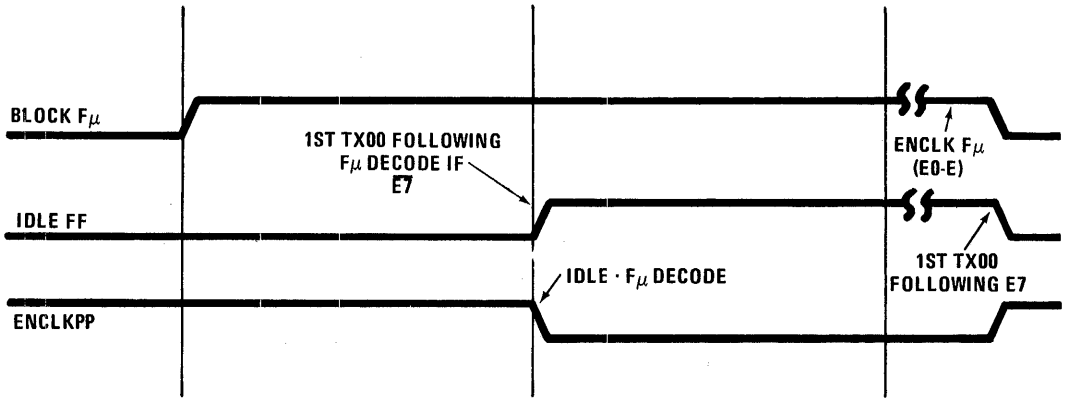


Figure 3-32. (FZJ · Aμ = 0), FRJ, RNI1, RNI2, CIO1, CIO2, ROM, and SYNC μI's

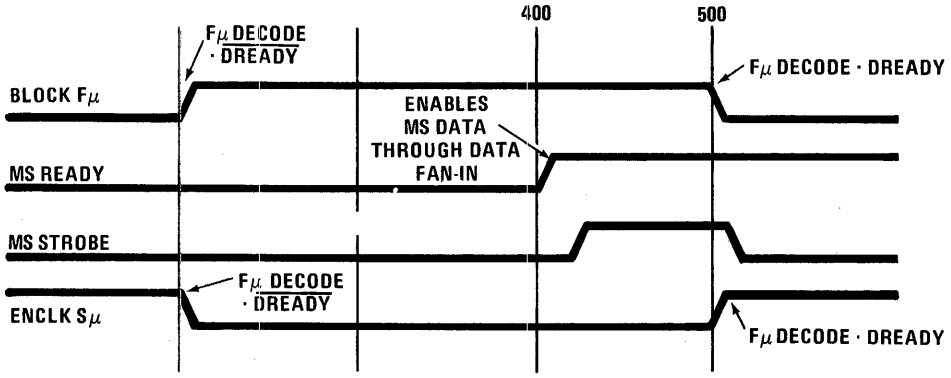


Figure 3-33. SDW and SDB

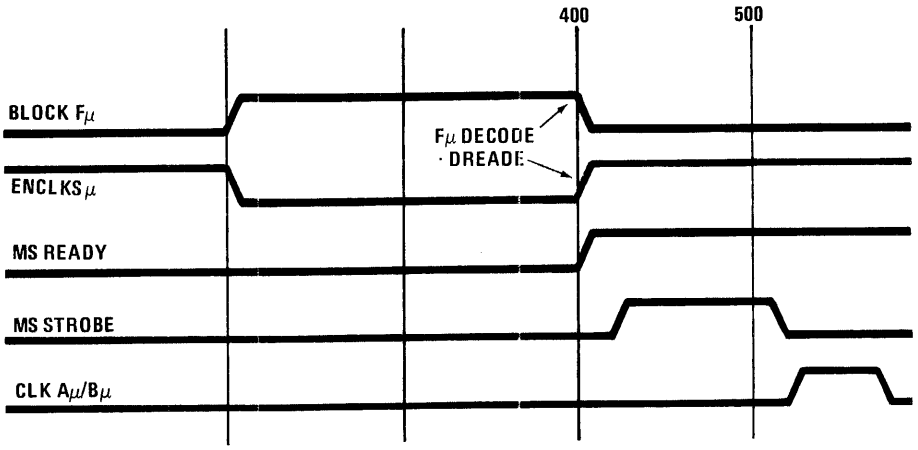


Figure 3-34. DTA, DTA-, IDX, and DFA μI's

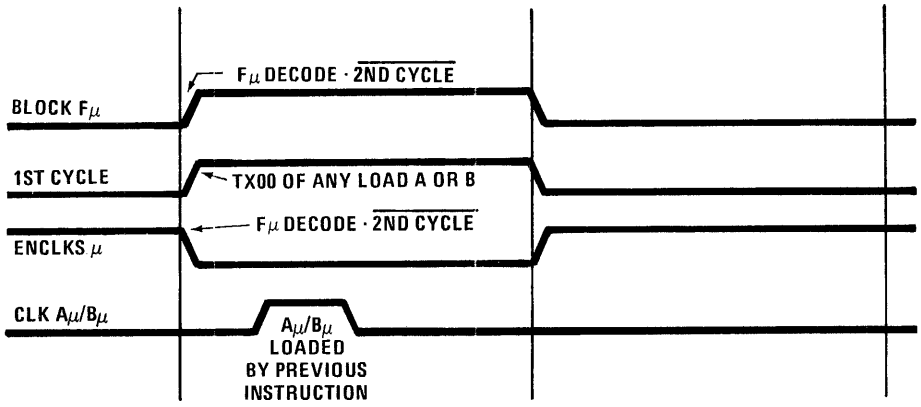


Figure 3-35. SUM, DSUM, CMP, and CMU μI's

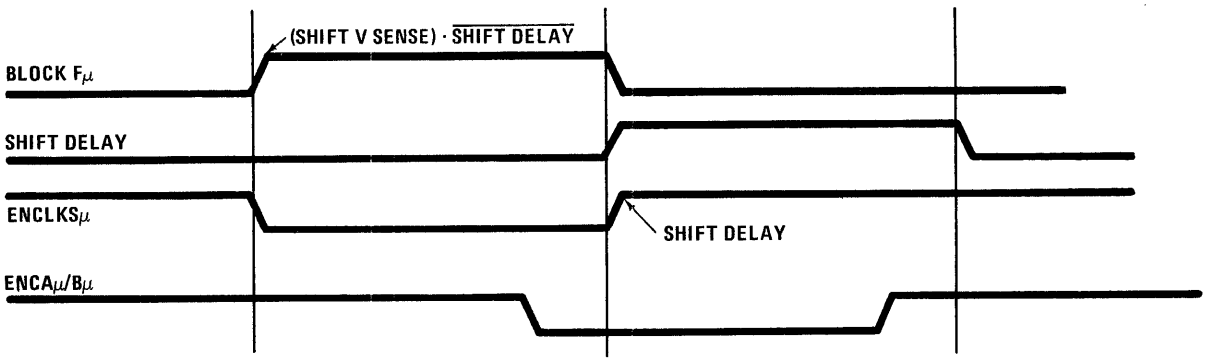


Figure 3-36. All Shift or Sense Instructions

IMPLEMENTING MACHINE LANGUAGE INSTRUCTION BY MICRO INSTRUCTIONS

This section describes how micro-instructions (μI's) are linked together in routines to execute (implement) machine-language instructions (MLI's). This is done by means of showing certain MLI's in flow diagram form as examples, listing all the μI's necessary to implement each MLI. Emphasis is placed on the concept of using individual μI's as "building blocks" to form routines, routines to form a complete MLI. Use of the μI assembly listing in Appendix 3A to locate the routines comprising a MLI is also described.

BASIC IMPLEMENTATION SCHEME

Machine language instructions (MLI's) executed by the system are done so by micro-instructions (μI's) arranged in a particular order as required by the MLI. This arrangement of μI's necessary to execute a MLI can be divided into at least three, and sometimes more, individual sequences as shown in Figure 3-37. The first sequence is called the Read Next Instruction (RNI) sequence. This sequence reads the first word of the MLI from main storage (MS), inserts this word in the F register for subsequent translation, and performs a first-level, or *Format Jump,* decode of the MLI to determine its format, that is, its length (2-, 4-, 6-, or 8-byte) and the type of operand addressing specified (direct, indirect, or indexing). Upon obtaining this information, a *jump* is made to a corresponding FRJ sequence common to all MLI's of this format. The FRJ sequence reads the first operand to be processed by the MLI from either a file register or from MS, depending on the addressing mode specified. This sequence also performs a second-level, or Function Jump (FNJ) decode of the MLI to determine its function, that is whether the MLI will perform an add, shift, move, compare, or other type of function. At this point, the MLI is uniquely defined.

Upon determining the MLI function, a *jump* is made to a corresponding FNJ sequence that reads the second operand, performs the specified operation on the two operands, and stores the result. After completing the store, a *jump* is made back to the RNI sequence to execute the following MLI. Each sequence takes at least one time slice to perform. More complex MLI's, such as multiply, divide, or I/O instructions may require more than one time slice to complete the FRJ sequence and several execute sequences apart from the FNJ sequence. However, all MLI's are basically executed in the manner just described, where the FRJ decode performs a gross translation of the MLI and isolates it to a group of several MLI's, and the FNJ decode performs a final translation to uniquely define the MLI.

An example of how the RNI, FRJ, and FNJ sequences are used to execute a two-byte MLI, specifically the ADDR (26) MLI, is shown in Figure 3-38. This figure shows the functional operations making up each sequence and the address of the corresponding μI in CS required to perform each function. (Refer to the CS assembly listing in Appendix 3A for a listing and description of each μI at the address listed in Figure 3-38.) Execution of the MLI can begin with either the RNI0, RNI1, or RNI2 sequence, depending on when in the program the MLI is executed as described in the paragraph titled Set P_p Logic. If the ADDR MLI is the first MLI of the program, the RNI0 sequence is entered at address 0000* and the MLI address is obtained from the processor's assigned P register in the Basic Register File (BRF). If the ADDR MLI follows another MLI in the same program, that previous MLI will have terminated with either an RNI1 or RNI2 μI to cause a *jump* back to either the RNI1 sequence (address 0002) or the RNI2 sequence (address 0009). The choice of terminating with either an RNI1 or RNI2 μI will depend on whether the previous MLI had time to form the ADDR MLI address, as discussed in the paragraph titled Set P_p Logic. Depending on which of the two RNI sequences is entered, the MLI address is obtained from either register Q1 (P+2) or register Q2 (P+4). Except for the source of obtaining the MLI address, the RNI0 and RNI1 sequences are identical; therefore, they combine at address 0004. Both RNI0/1 and RNI2 sequences terminate at addresses 0008 and 000F, respectively, with an FRJ μI to implement the FRJ decode operation.

*All CS addresses are represented in hexadecimal form.

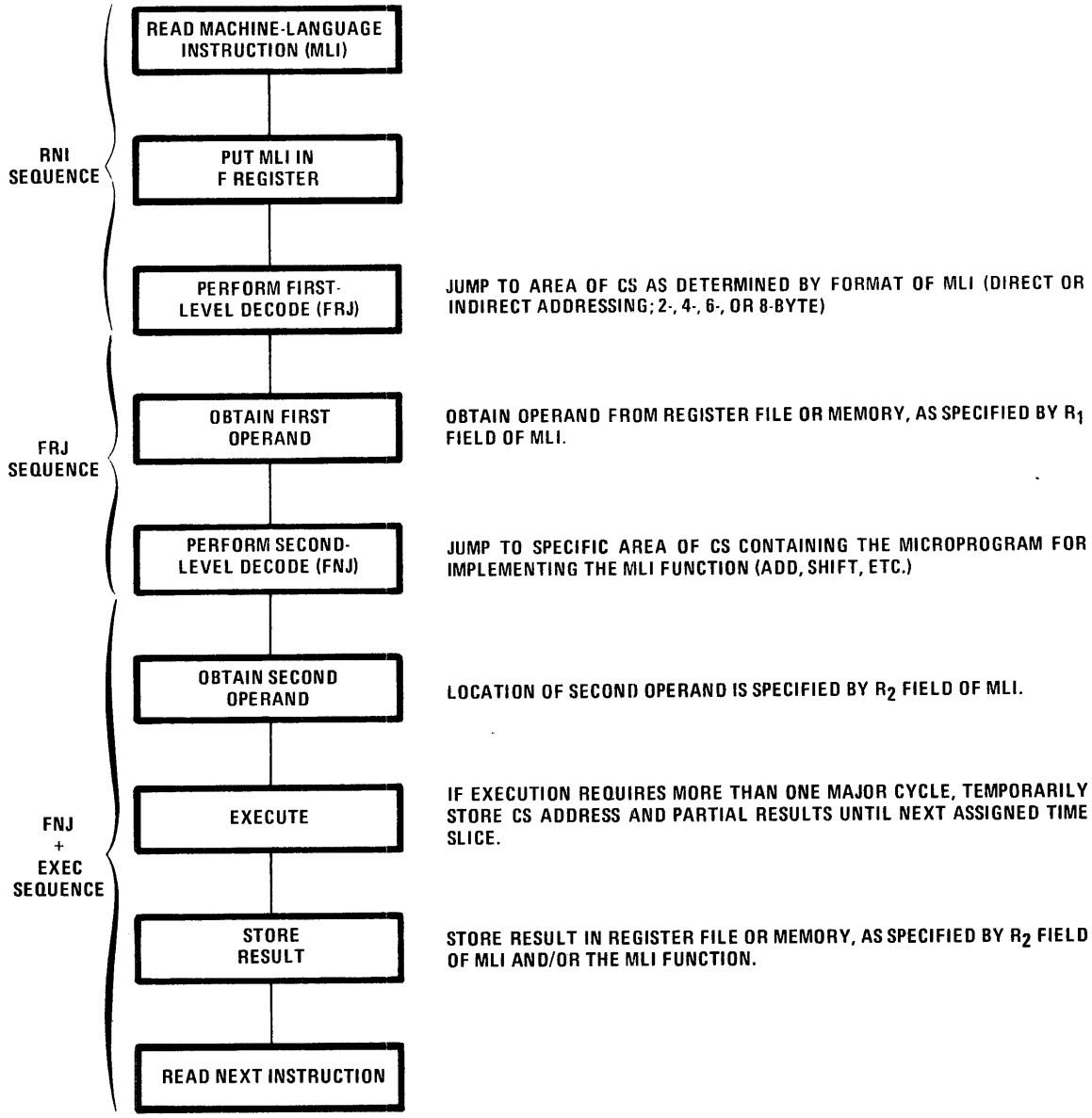


Figure 3-37. Basic Microcode Implementation of MLI

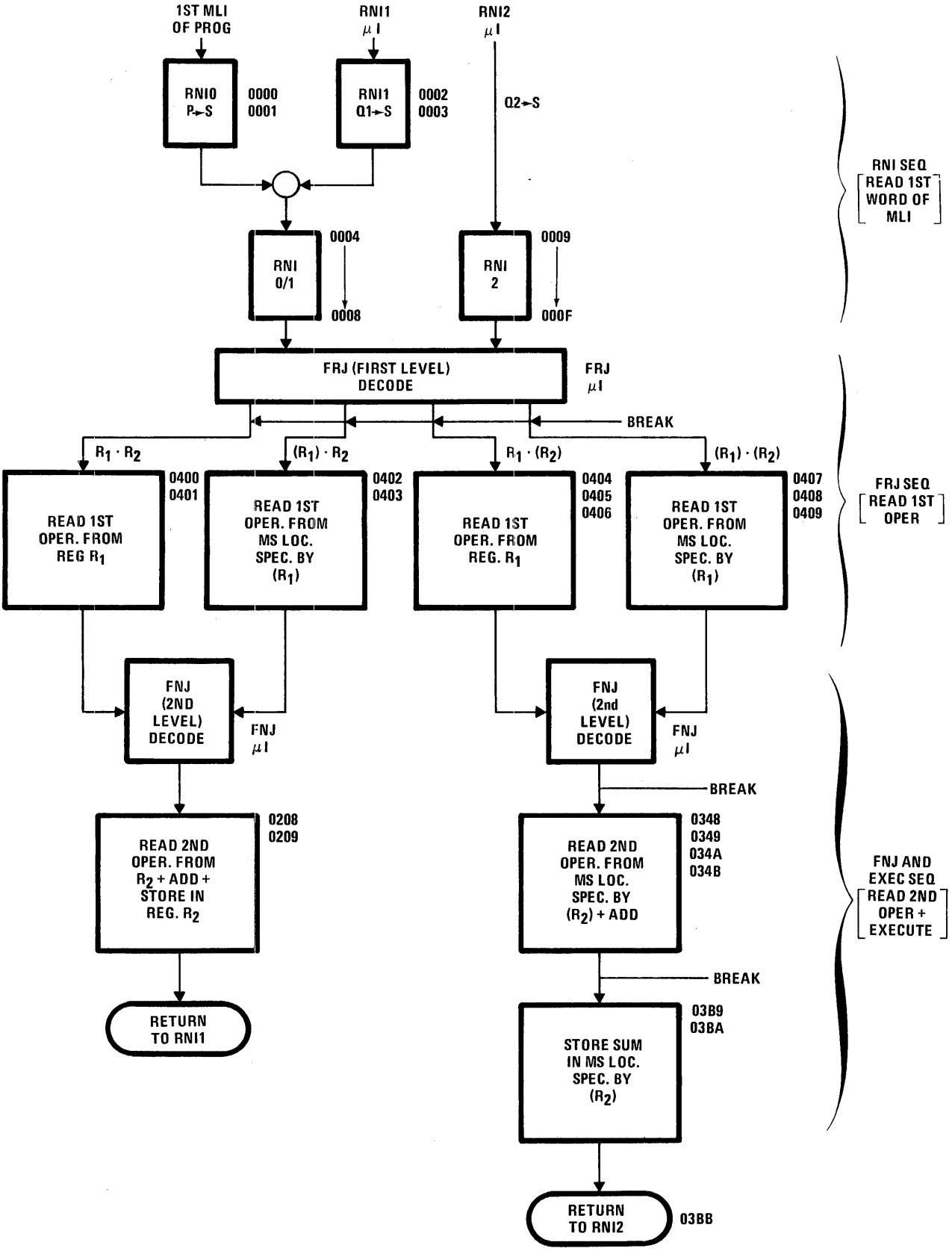
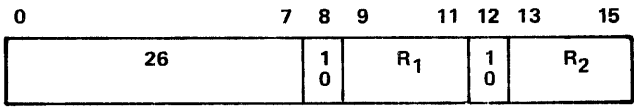


Figure 3-38. Two-Byte (ADDR) MLI Flow Diagram

The FRJ decode translates the MLI function code and indirect designators to determine the MLI format. An examination of the FRJ decode address table determines that the ADDR MLI is a two-byte MLI in the 20-29 range of function codes, defining it as a register/register MLI. The result is a *jump* to one of four FRJ sequence starting addresses: 0400, 0402, 0404, or 0407; depending on the addressing mode specified. This *jump* is made at the beginning of a new time slice assigned to the processor, as indicated by the word "BREAK" in Figure 3-38. For this example, the FRJ sequence for all four addressing modes is shown. (During actual execution, of course, only one of the four could be specified.) For convenience, these four possible addressing modes are summarized below:



ADDR MLI FORMAT

$R_1 \cdot R_2$ – first operand read direct/second operand read direct

$(R_1) \cdot R_2$ – first operand read indirect/second operand read direct

$R_1 \cdot (R_2)$ – first operand read direct/second operand read indirect

$(R_1) \cdot (R_2)$ – first operand read indirect/second operand read indirect

The FRJ sequences for the $R_1 \cdot R_2$ and $(R_1) \cdot R_2$ mode start at different addresses since the first mode requires reading the first operand via direct addressing and the second mode requires reading the first operand via indirect addressing. The FNJ and execute sequence for these two conditions, however, are identical because the second operand for both conditions is read via direct addressing. The FNJ decode, therefore, generates FNJ sequence beginning address 0208 for both the $R_1 \cdot R_2$ and $(R_1) \cdot R_2$ modes. Upon completing the *Sum* and *Store* operation, the routine ends with an RNI1 μI, which causes a jump back to the RNI1 sequence to read the next MLI. For both these addressing modes, both FRJ and FNJ sequences are executed in the same time slice. If executed in either of these two addressing modes, therefore, the ADDR MLI takes two time slices to execute: one for the RNI sequence and one for the combined FRJ and FNJ/execute sequence.

The FRJ sequences for the $R_1 \cdot (R_2)$ and $(R_1) \cdot (R_2)$ mode also start at different addresses because of the different addressing mode required for reading the first operand. In addition, these two addressing modes cannot read the first operand using the $R_1 \cdot R_2$ and $(R_1) \cdot R_2$ mode first operand read routine because it is necessary to store the first operand read from MS in register T3 before reading the second operand. This additional store requires a SDW μI not available in the $R_1 \cdot R_2$ and $(R_1) \cdot R_2$ first operand read routines. Upon storing the first operand read in T3, the FNJ decode branches to a common FNJ and execute sequence to read the second operand using indirect addressing and add it to the first operand. After the addition, a *jump* is made to a separate store sequence to store the result in MS at location (R_2). If executed in either of these two addressing modes, therefore, the ADDR MLI takes four time slices to execute: one each for the RNI, FRJ, and FNJ sequences, and one for the final store sequence.

Execution of a six-byte MLI, that of the ADDM (62) MLI, is shown in Figure 3-39. Like the ADDR MLI flow diagram of Figure 3-38, the ADDM MLI flow diagram also shows the four possible modes of addressing. However, the ADDM MLI is more complex because of the indexing capability provided by this MLI. This indexing capability is provided by the R_1 and R_2 fields of the first MLI word, which add the contents of registers defined by these fields to the operand addresses designated by the M_1 and M_2 fields contained in the second and third words of the MLI. Indexing of this type is called post-indexing. (For more details regarding indexing, see the MEMOREX 7300 Processing Unit Reference Manual.) The figure assumes that the proper RNI sequence has been completed and that a *jump* has been made to one of four FRJ sequence starting addresses as a result of the FRJ decode operation. Because of its greater complexity, the ADDM MLI is able to take greater advantage of the divisible nature of the μI routines comprising a sequence, as evidenced by the numerous jumps within each sequence used to execute the MLI. The greater number of time slices required to execute this MLI reflects the two additional words of the MLI read from MS and the indexing operations required during the FRJ sequence. The number of time slices required per sequence for each of the four addressing modes is listed below:

	RNI	FRJ	FNJ/EXEC	STORE
$M_1 \cdot M_2$	1	3	1	1
$(M_1) \cdot M_2$	1	4	1	1
$M_1 \cdot (M_2)$	1	4	1	1
$(M_1) \cdot (M_2)$	1	5	1	1

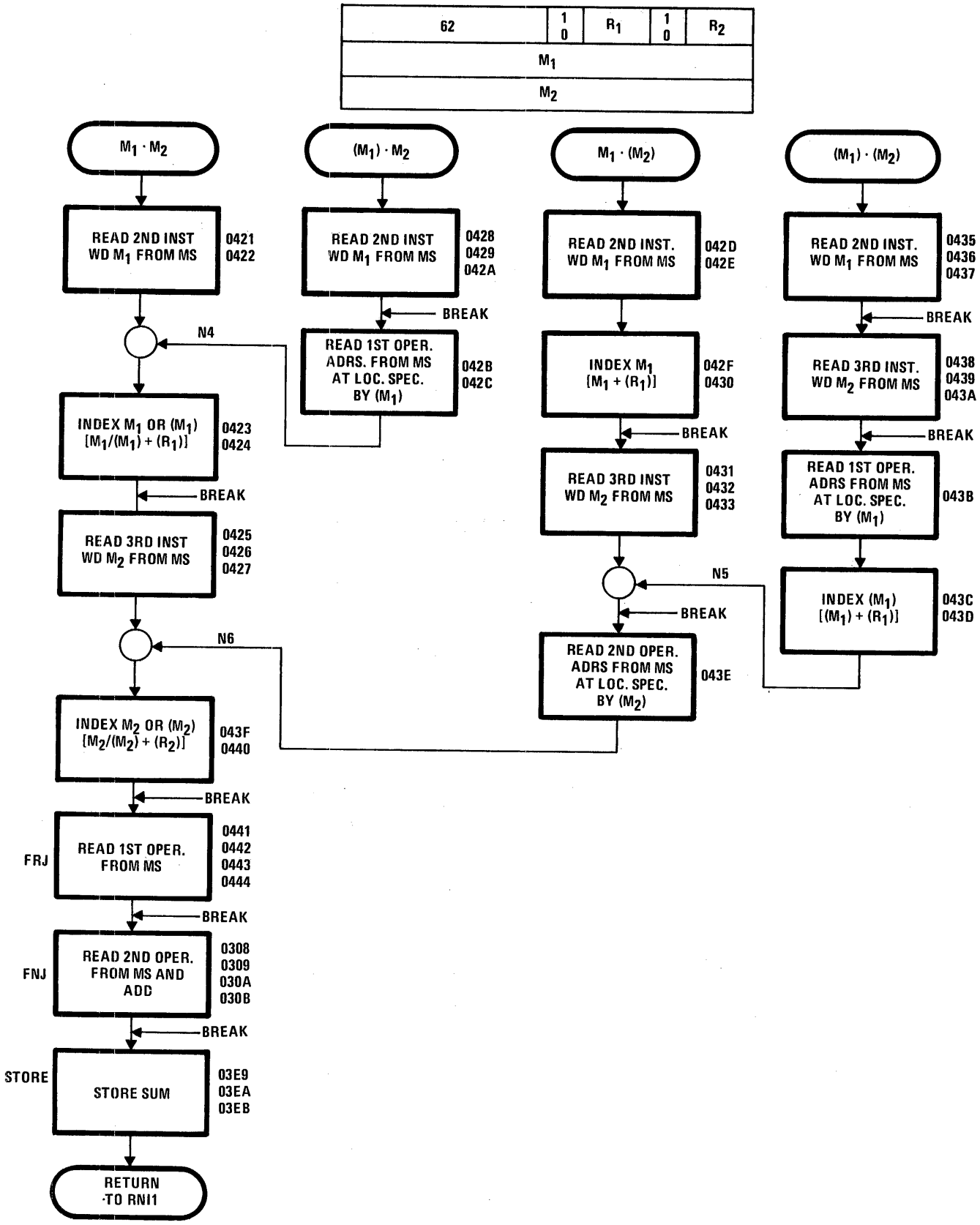
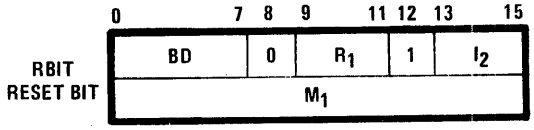


Figure 3-39. Six-Byte (ADDM) Instruction Flow Diagram

REPRESENTATIVE MLI FLOW DIAGRAMS

Flow diagrams for a number of other MLI's (RBIT, BOF, and CBYM) are shown in Figures 3-40, 3-41, and 3-42. The flow diagrams start at the beginning of the FRJ sequence, upon completing the FRJ μI of the RNI sequence. Note that the BOF MLI is completed during the FRJ sequence, while the others require an FNJ/execute sequence for their completion.



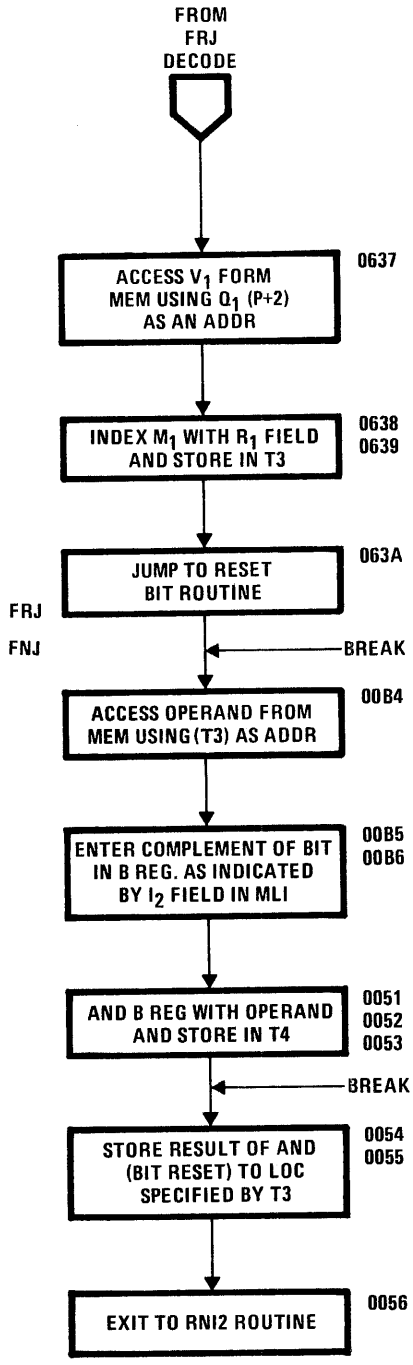
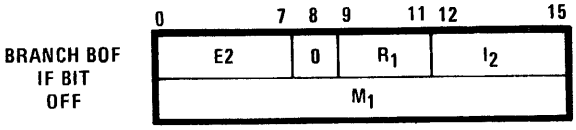


Figure 3-40. Two-Byte (RBIT) Instruction Flow Diagram



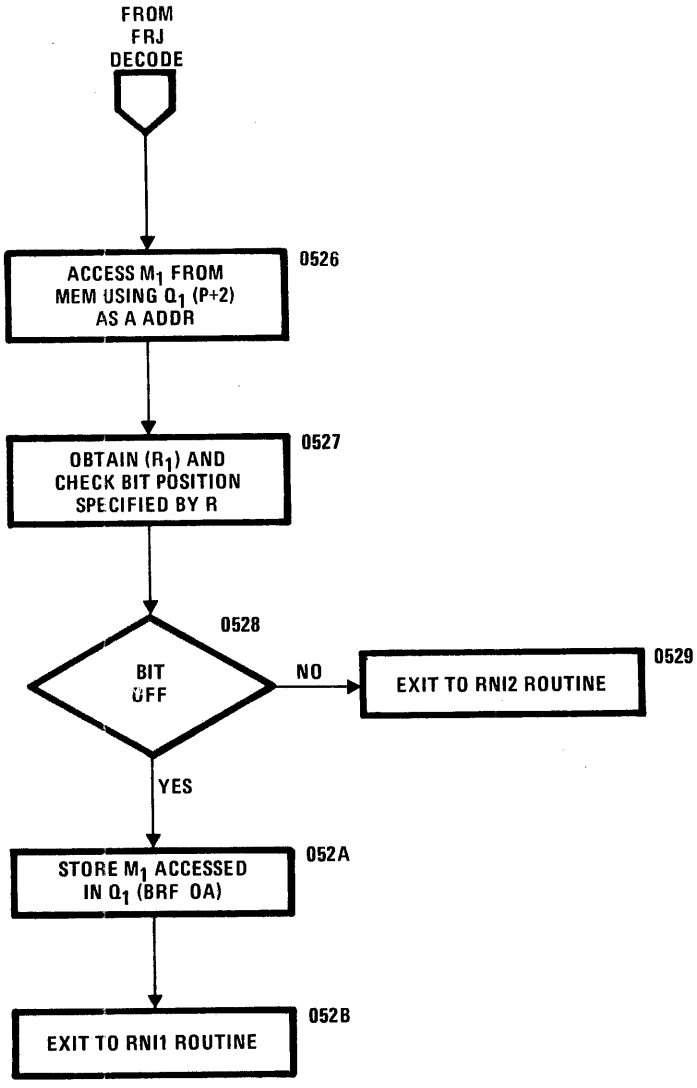


Figure 3-41. Two-Byte (BOF) Instruction Flow Diagram

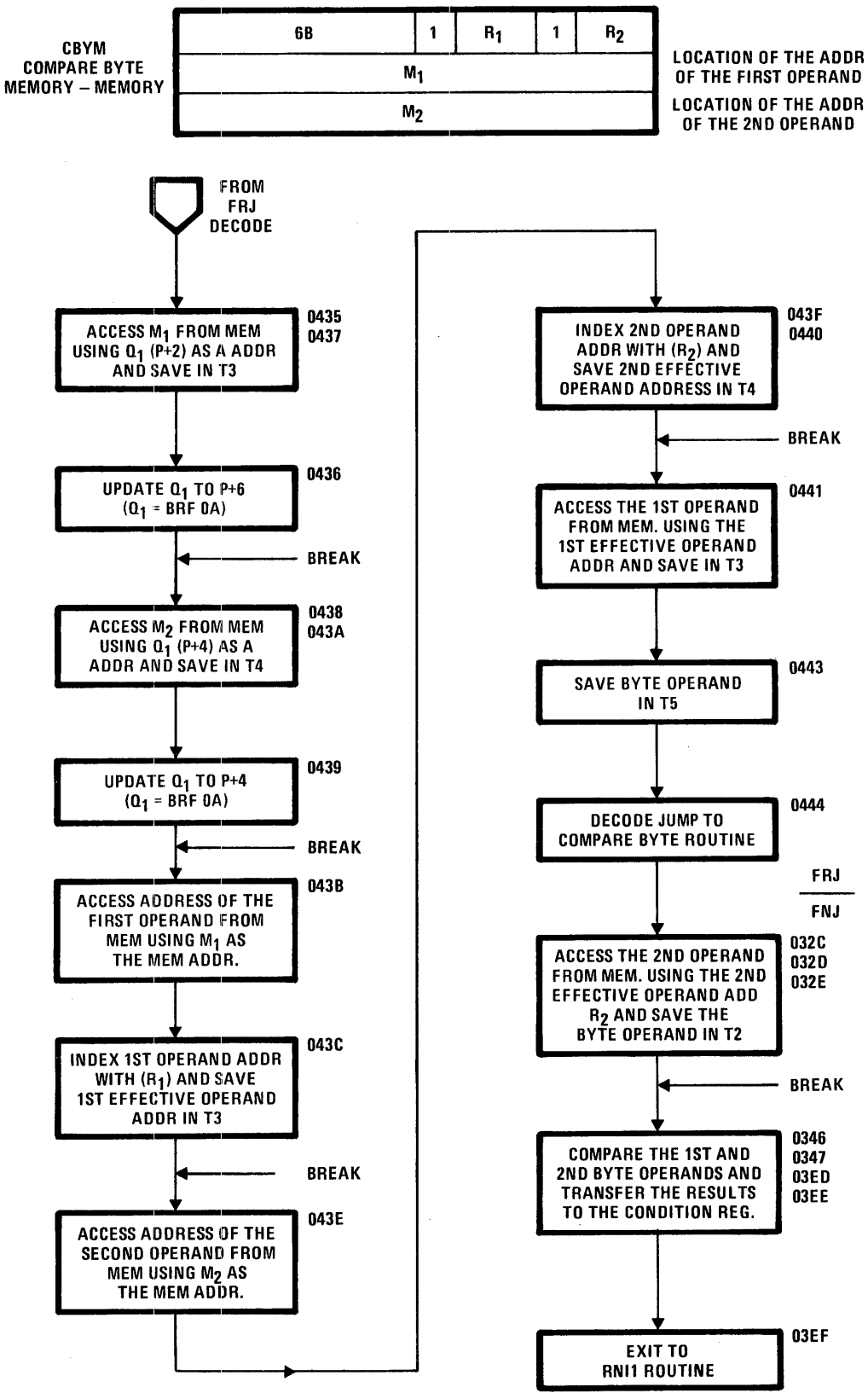


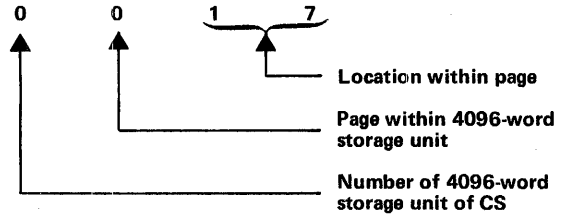
Figure 3-42. Six-Byte (CBYM) Instruction Flow Diagram

APPENDIX 3A
MICRO-INSTRUCTION ASSEMBLY LISTING

The micro-instruction (μI) assembly listing is a sequential printout of the contents of control storage (CS). It lists all μI's making up each machine language instruction (MLI), trap routine, and off-line function initiated from the System Control Panel. Each CS location is listed in the printout, even those which do not contain a μI. Such unused locations are loaded with "0's", except the parity bit (bit position 8), which is set to "1" to produce odd parity for the corresponding location. Directions for using the μI assembly listing to determine how a particular MLI or trap routine is implemented by μI's are contained in the paragraphs titled Implementing MLI's by μI's and System Reset Operations, respectively. This appendix discusses the symbols and conventions used in interpreting the listing. Refer to Figure 3A-1, which shows a typical page of the CS assembly listing, for the locations of the numbered items described below:

(1) PAGE -- This item defines the sheet page of the assembly listing (not the 256-word physical CS page). Each sheet page is numbered in consecutive order within each of the 16 physical pages of CS. Depending on the amount of explanatory matter on each sheet page, each physical page may have from 7 to 15 sheet pages.

(2) LOCATN – This column lists each 16-bit CS location in hexadecimal form. Using location 0017_{16} as an example, each digit of the location number provides the following information:



(3) OBJECT – This column lists the contents of each CS location in machine coded hexadecimal form (μI object code).

(4) A/B – These two columns list the values of the a and b designators (bit positions 6 and 7) for each μI.

(5) ADDR – This column lists the rightmost two digits of the μI object code. Generally, these two digits specify either the address of a register specified by a RF read, or RF write μI, or a branch address of a branch μI. In some cases, however, the listed value indicates a constant to be operated on by the particular μI. If the ADDR value specifies a register, the register will be in either the BRF or ERF, depending on the values of the a and b designators.

(6) SOURCE STATEMENT – This column lists the μI in mnemonic form (source code) and a source code equivalent of the corresponding ADDR column entry (register number, jump address, or constant). To the right of each μI listed is a short description explaining the purpose of the μI. Where a register number is listed, it is indicated with an abbreviation as defined below:

C – Condition register (BRF)

P – P register (BRF)

Tn, Qn – Transient register n of BRF

M – Register defined by R_1 field of MLI

R – Register defined by R_2 field of MLI

Xn – Register n of ERF Group II

En – Register n of ERF Group III

(7) This column lists the branch address in source code form corresponding to the CS location in hexadecimal form as a means of defining the start of a branch routine. This information is used in conjunction with the table at the top of the page, which provides a cross-reference of branch addresses in both source code and object code form. The first sheet page of each CS physical page provides similar information regarding the branch addresses referenced within that physical page.

(8) This column lists the cumulative results of a longitudinal parity check performed during a CS scan operation. Each entry represents the results of the check up to the corresponding CS location, if no longitudinal parity errors were detected. This information is used during maintenance operations to determine the location at which incorrect data is stored.

(9) Refer to the FRJ decode address table in the front of the assembly listing to find the starting address of the FRJ sequence corresponding to the MLI under consideration.

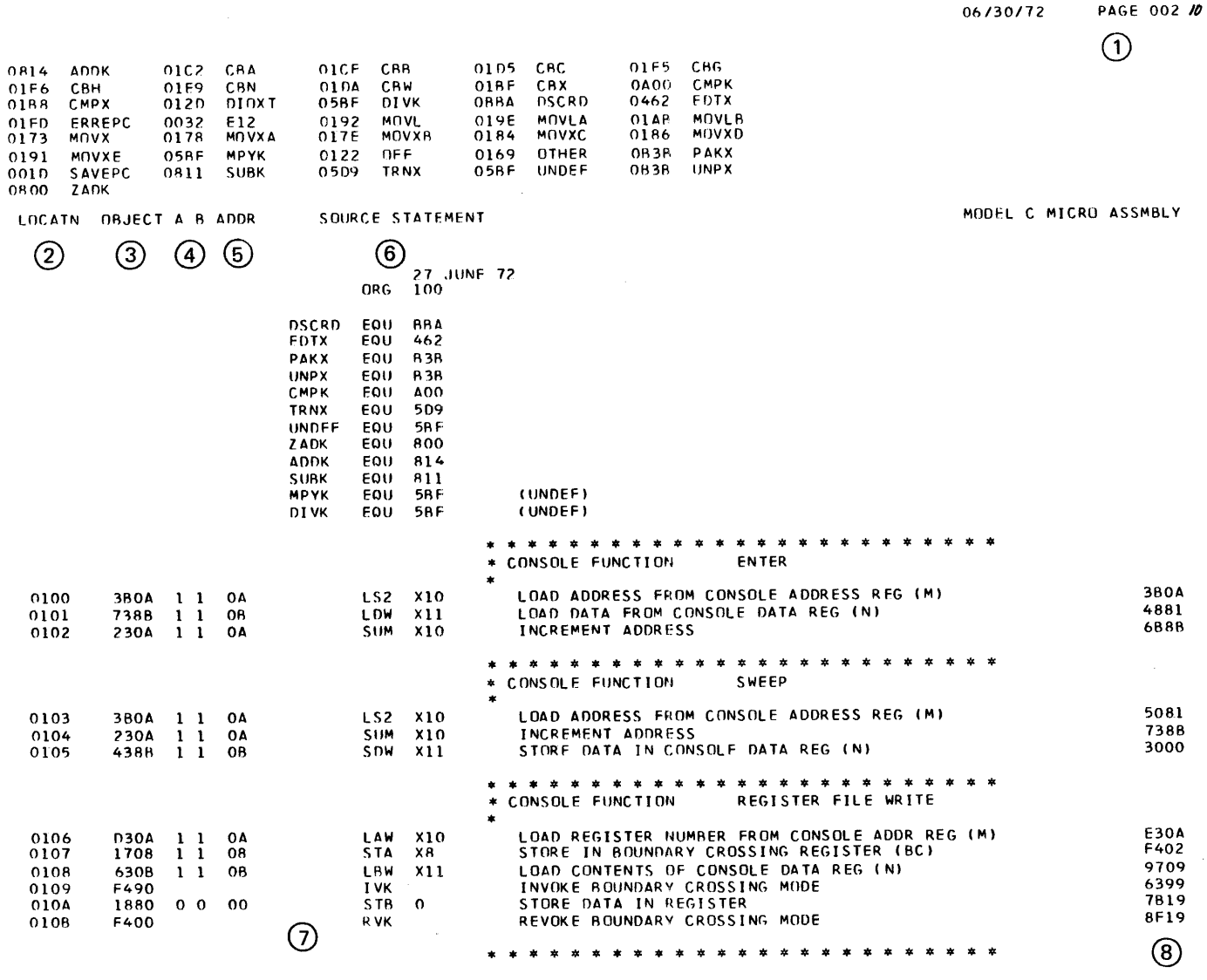
(10) These conventions specify the four possible operand addressing modes associated with these MLI's. (For more details about operand addressing, see the MEMOREX 7300 Processing Unit Reference Manual.) These are as follows:

A:B – first operand read direct, second operand read direct.

(A):B – first operand read indirect, second operand read direct.

A:(B) – first operand read direct, second operand read indirect.

(A):(B) – first operand read indirect, second operand read indirect.



```
                                                        06/30/72      PAGE 002 10
                                                                          (1)
0814  ADDK     01C2  CBA      01CF  CBB      01D5  CBC      01F5  CBG
01F6  CBH      01F9  CBN      01DA  CBW      01BF  CBX      0A00  CMPK
01B8  CMPX     0120  DIOXT    05BF  DIVK     0BBA  DSCRD    0462  FDTX
01FD  ERREPC   0032  E12      0192  MOVL     019E  MOVLA    01AB  MOVLB
0173  MOVX     0178  MOVXA    017E  MOVXB    0184  MOVXC    0186  MOVXD
0191  MOVXE    05BF  MPYK     0122  OFF      0169  OTHER    0B3B  PAKX
001D  SAVEPC   0811  SUBK     05D9  TRNX     05BF  UNDEF    0B3B  UNPX
0800  ZADK

LOCATN  OBJECT A B ADDR      SOURCE STATEMENT                  MODEL C MICRO ASSMBLY
  (2)     (3)  (4)(5)            (6)
                          27 JUNE 72
                    ORG   100

                DSCRD EQU   BBA
                FDTX  EQU   462
                PAKX  EQU   B3B
                UNPX  EQU   B3B
                CMPK  EQU   A00
                TRNX  EQU   5D9
                UNDEF EQU   5BF
                ZADK  EQU   800
                ADDK  EQU   814
                SUBK  EQU   811
                MPYK  EQU   5BF     (UNDEF)
                DIVK  EQU   5BF     (UNDEF)

                                    * * * * * * * * * * * * * * * * * * * * * * * * *
                                    * CONSOLE FUNCTION      ENTER
                                    *
0100    380A  1 1  0A       LS2   X10     LOAD ADDRESS FROM CONSOLE ADDRESS REG (M)        380A
0101    738B  1 1  0B       LDW   X11     LOAD DATA FROM CONSOLE DATA REG (N)              4881
0102    230A  1 1  0A       SUM   X10     INCREMENT ADDRESS                               6B8B

                                    * * * * * * * * * * * * * * * * * * * * * * * * *
                                    * CONSOLE FUNCTION      SWEEP
                                    *
0103    380A  1 1  0A       LS2   X10     LOAD ADDRESS FROM CONSOLE ADDRESS REG (M)        5081
0104    230A  1 1  0A       SUM   X10     INCREMENT ADDRESS                               738B
0105    438B  1 1  0B       SDW   X11     STORE DATA IN CONSOLE DATA REG (N)              3000

                                    * * * * * * * * * * * * * * * * * * * * * * * * *
                                    * CONSOLE FUNCTION      REGISTER FILE WRITE
                                    *
0106    D30A  1 1  0A       LAW   X10     LOAD REGISTER NUMBER FROM CONSOLE ADDR REG (M)   E30A
0107    1708  1 1  08       STA   X8      STORE IN BOUNDARY CROSSING REGISTER (BC)         F402
0108    630B  1 1  0B       LBW   X11     LOAD CONTENTS OF CONSOLE DATA REG (N)           9709
0109    F490               IVK           INVOKE BOUNDARY CROSSING MODE                   6399
010A    1880  0 0  00       STB   0       STORE DATA IN REGISTER                          7B19
010B    F400               RVK           REVOKE BOUNDARY CROSSING MODE                   8F19
          (7)
                                    * * * * * * * * * * * * * * * * * * * * * * * * *    (8)
```

Figure 3A-1. Typical Page of CS Assembly Listing

APPENDIX 3B
LOGIC SIGNAL NAME ABBREVIATIONS

ABAND	Abandon
ALU	Airthmetic-Logical Unit
AM	Aμ register
AT	Address Table
B/A	Busy/Active
BC	Boundary Crossing
BKPT	Breakpoint
BLK	Block
BM	Bμ register
BRF	Basic Register File
BRFS	Basic Register File Select
BUFF	Buffer
CC	Consecutive Cycles
CG	Constant Generator
CHK	Check
CIN	Carry In
CLK	Clock
CLR	Clear
CR	Control register
CRIN	Carry In
CS	Control Storage
CSS	Control Store Scan
DISPY	Display
DRB0	D register Byte 0
ECC	Error Correction Code
EN	Enable
ENCAM	Enable Clock Aμ
ENDO	End Out
ENRAM	Enable Reset Aμ
ENSAM	Enable Set Aμ
ENT	Enter
EOR	Exclusive – OR
EOT	End of Transmission
EQ	Equal
ENRSYCL	Enable Resync FF Clear
ERF	Extended register
ERFG2	Extended Register File Group II
EXEC	Execute
EOXX-E	EO Minor Cycle, Early
EOXX-L	EO Minor Cycle, Late
FB	F Buffer register
FF	Flip-flop
FM1	Fμ register, Rank 1
FM2	Fμ register, Rank 2
FR	F register
FRJ	Format Jump
FXEQ	Fμ Translation Equals (example: FXEQ-6 means translation of μI whose operation code is 6)
GC	Group Carry
GEN	Generate
GP	Group
GT	Greater Than
ICA	Integrated Communications Adapter
ICRA	Integrated Card Reader Adapter
IFA	Integrated File Adapter
INVERF	Invoke Extended Register File
IOR	Inclusive – OR
JMP	Jump
JP	Jump
LD	Load
LT	Less Than
MC	Master Clear
MR	Console Address register
MS	Main Storage
SMI	MS Interface
NR	Console Data register
OV	Overflow
PB	P Buffer register
PE	Parity Error
PM	Pμ register
P_p	P Pointer register
PR	Privileged register
PROP	Propagate
RD	Read
REQ	Request
RF	Register File (either BRF or ERF)
RNI	Read Next Instruction
RTC	Real Time Clock
RO	Register Option
SELFH/PL	Select F if High or Pμ if Low
SN	Shift Network
SOPXEQ	Sub-op Code Translation Equals (example: SOPXEQ-0 means translation of μI whose sub-op code is 0)
SPEC	Specified, Special
SR	S register
STDBYTE	Store D Byte
SW	Switch
S1, S2, S3	Select 1, Select 2, Select 3
TB	Tie Breaker
TBIT	Toggle Bit
WR	Write
XTAL	Crystal
ZR	Zero

INDEX

COMMENTS FORM

7300 Processing Unit Design Description Manual Volume II: Shared Resources (2501.002)

Please send us your comments, to help us produce better publications. Use the space below to qualify your responses to the following questions, if you wish, or to comment on other aspects of the publication. Please use specific page and paragraph/line references where appropriate. All comments become the property of the Memorex Corporation.

- Is the material:

	Yes	No
Easy to understand?	☐	☐
Conveniently organized?	☐	☐
Complete?	☐	☐
Well illustrated?	☐	☐
Accurate?	☐	☐
Suitable for its intended audience?	☐	☐
Adequately indexed?	☐	☐

- For what purpose did you use this publication (reference, general interest, etc.)?

  _____

- Please state your department's function: _____

  _____

Please use the space below to describe any specific comments which you feel will help us to produce a better publication. _____

MEMOREX

First Class

Permit No. 250
Santa Clara
California 95050

Business Reply Mail

No Postage Necessary if Mailed in the United States

Postage Will be Paid By

Memorex Corporation

Santa Clara Publications
Mail Stop 00-21
1200 Memorex Drive
Santa Clara, California 95052

Thank you for your information.....

Our goal is to provide better, more useful manuals, and your comments will help us to do so.

.....Memorex Publications