IBM Field Engineering

Manual of Instruction

2075 Processing Unit — Volume 2

Storage Bus Control

Instruction Preparation

FLT, Logout, MCW

Interrupts

223-2873-1

Form 223-2873-1
FES S26-7034

PREFACE

This is one of six Field Engineering manuals for the 2075 Processing Unit. These six manuals contain the unit theory of operation, reference diagrams to be used when troubleshooting, and maintenance procedures.

A basic knowledge of the IBM System/360 as contained in the IBM System/360 Principles of Operation, Form A22-6821 is considered a prerequisite for studying the unit theory of operation. The theory of operation is contained in a four volume manual identified as a Field Engineering Manual of Instruction (FEMI). Volume 1 is a prerequisite for the detailed information contained in volumes 2, 3, and 4. Volume 1 contains the introduction to the system and the processing unit and a description of the functional units (registers, adders, and decoders) of the processing unit. Volumes 2 and 3 contain detailed instruction analysis, and volume 4 contains detailed information on special features and power supplies and control.

The four volumes of theory of operation contain many references to the diagrams packaged in the associated Field Engineering Diagrams Manual (FEDM). All diagrams in the FEDM are identified by a four digit figure number and unless otherwise specified, all four digit figure references in the FEMI indicate that the figure is contained in the associated FEDM.

The complete titles and form numbers of the six 2075 Field Engineering Manuals are:

- 2075 Processing Unit--Volume 1, Comprehensive Introduction, Functional Units, Field Engineering Manual of Instruction, Form 223-2872
- 2075 Processing Unit--Volume 2, Theory of Operation: Storage Bus Control; Instruction Preparation; FLT, Logout, MCW; Interrupts, Field Engineering Manual of Instruction, Form 223-2873
- 2075 Processing Unit--Volume 3, Theory of Operation: Fixed Point, I Execute, Branch, Floating Point, Variable Field Length, Field Engineering Manual of Instruction, Form 223-2874
- 2075 Processing Unit--Volume 4, Special Features, Power Supply and Control, Appendix, Field Engineering Manual of Instruction, Form 223-2875
- 2075 Processing Unit, Field Engineering Diagrams Manual, Form 223-2876
- 2075 Processing Unit, Field Engineering Maintenance Manual, Form 223-2880

MAJOR REVISION (December 1965)

This edition, Form 223-2873-1, obsoletes Form 223-2873-0.
Principal change in this edition: Many figures have been changed due to engineering changes affecting storage timing.

Copies of this and other IBM publications can be obtained through IBM Branch Offices.
Address comments concerning the contents of this publication to:
IBM Systems Development Division, Product Publications, Dept. 520, CPO Box 120, Kingston, N.Y. 12401

CONTENTS

LIST OF ILLUSTRATIONS

STORAGE BUS CONTROL

- Prerequisite for study of this section is study of the 2075 Processing Unit, Volume 1, Field Engineering Manual of Instruction, Form 223-2872.

- This section covers operation of the basic storage bus control -- one, two, or four 2365 Model 3 Processor Storage Units.

- Operation of the 2075 with the LCS is in the "Features" section.

The Model 75 storage bus control (bus control unit) is the link between main storage and storage users. Because the operation of the bus control unit (BCU) is affected extensively by system optional features such as large capacity storage (LCS) and shared LCS, only the basic BCU is described in this section. The basic BCU operates with a main storage of either one, two, or four 2365 Model 3 Processor Storage Units. These storage configurations are reflected in the system model designations:

Model H75 - One 2365 Processor Storage Unit
Model I75 - Two 2365 Processor Storage Units
Model J75 - Four 2365 Processor Storage Units

Operation of the BCU with optional main storage configurations is described in the 2075 Processing Unit, Volume 4, Field Engineering Manual of Instruction, Form 223-2875.

INTRODUCTION

- BCU handles all system storage requests.

- BCU accepts storage requests, then frees the requester during the storage cycle.

- BCU returns fetched data to the proper register or to the channel SBO.

- Each 2365 contains two independently operating high-speed storages (HSS).

- A 2365 stores even addresses in one HSS, odd addresses in the other HSS. This addressing scheme is called interleaving.

Whenever the E unit, the I unit, any channel, or the system control panel requires a storage reference (a store or fetch to main storage) a storage request is sent to the BCU. The BCU honors these requests one at a time according to a fixed priority scheme. For each request, BCU must start (select) the proper storage unit, route the incoming storage address, and route the data either to or from the selected storage unit (Figure 1).

A storage request to the BCU is actually an order, or command, to store or fetch a 72-bit word (64-data bits plus eight-parity bits) from main storage. As soon as the BCU begins to execute this order, the requesting area is free to do other work. For example, consider a fetch request from the I unit (Figure 2). The BCU will attempt to honor this request every machine cycle. However, it may be several cycles before the request is actually honored. For example, all channels have a higher priority than the I unit. If a channel is ready to start storage, the I unit request is blocked. Also, storage may be busy, forcing the I unit to wait.

When there are no conflicts, BCU will select the storage requested by the I unit and inform the I unit by sending an accept pulse. The accept signal tells the I unit that its request is being serviced. The I unit is now free to drop its signals to the BCU and proceed with other work. When the storage unit delivers the 72-bit word from the requested address, the BCU routes this word into the J register.

In handling storage requests, it is not necessary for the BCU to consider the E unit and the I unit as separate users. The I unit establishes priorities for E and I so that only one request from the CPU (E and I) is sent to the BCU at a time.

STORAGE WORDS AND STORAGE ADDRESSES

- Full storage address is 24 bits.

- Model 75 storage words contain eight bytes (double word).

- Bytes changed on a store operation are controlled by mark bits.

- Two-way address interleaving is employed on Models H75 and IH75.

- Four-way address interleaving is employed on Models I75 and J75.

All System/360 processors use a 24-bit address to select a byte within main storage. The Model 75 main storage, however, reads out eight bytes on every storage selection. On fetch operations, all eight bytes are delivered to the storage bus out (SBO) latch

register in the BCU and simultaneously written back into the cores. The CPU (or the channel) picks the proper bytes from among the eight bytes received if something less than a double word is required. On store operations, main storage again reads out eight bytes of data. However, one or more bytes of new data is substituted for the data read out before the double word is written back into the cores.

The substitution of bytes on a store operation is controlled by mark bits. There are eight mark bits, one for each of the eight bytes in a storage word. A mark 1 bit tells storage to replace the corresponding byte with the same byte of the storage bus in (SBI) latch register. A mark 0 bit tells storage to leave the corresponding byte unchanged (regenerate). The eight-byte storage words are on double word boundaries. The address of the first (lowest) byte has three low-order zeros (Figure 3). Because the BCU and storage handle only double words, the three low-order address bits are not used by either the BCU or storage.

Main storage consists of one, two, three, or four 2365 Processor Storage Units (Figure 4). Each 2365 has two independently operating high-speed storages (HSS). These HSS units are sometimes called M-4's. A system has from two to eight HSS, depending on the model. The addressing scheme is such that one HSS within a 2365 contains even addresses and the other HSS contains odd addresses. Even and odd addresses are in relation to double word addresses (Figure 3). A double word address is address bits 0-20. If bit 20 is a 0, the address is even; if bit 20 is a 1, the address is odd. The HSS have a two-letter designation. The first letter gives the 2365 (A, B, C, or D); the second letter designates even or odd addresses (E or O).

In Models H75 and IH75, double word addresses alternate between the two available HSS; this scheme is known as two-way interleaving (Figure 5). In Models I75 and J75, double word addresses progress through a ring of four HSS; this scheme is known as four-way interleaving (Figure 6). In Models I75 and J75, double word addresses progress through a ring of four HSS; this scheme is known as four-way interleaving (Figure 6).

Storage Address Protection

- Each 2365 has one storage protection unit.
- Storage address protection prevents inadvertent use of a storage location.
- Storage is protected in blocks of 256 storage words (2048 bytes).
- The key from the PSW or from the channel must match a prestored key in the SP unit.

Each of the 2365 processor storages contains a small storage and logic unit for storage address protection. This unit called the storage protection feature (SPF) or SP unit, serves both even and odd HSS within the 2365. The purpose of storage address protection is to prevent inadvertent use of a storage location.

For protection purposes, storage is divided into 256 storage word (2048 bytes) blocks. These blocks are pointed to by address bits 12-0. In other words, address bit 12 changes once for every 2048 consecutive byte addresses. Each SPF unit has one address location for each block of storage addresses within the associated HSS (Figure 7). In Models I75 and J75, two SPF units contain identical protection words because each storage block is spread among four HSS.

Each SPF address location contains five bits plus a parity bit (Figure 8). Initially, a series of set key instructions loads the five bits of each SPF word into the SPF unit(s). This initial loading establishes a protection pattern for each corresponding block of storage addresses.

Whenever a HSS is selected for a store or a fetch operation, a protection key (four bits) either from the PSW or from a channel is sent to the appropriate SP unit. If the CPU made the storage request, the protection key comes from PSW bits 8-11; if a channel made the request, the channel sends the protection key. This channel protection key was previously sent to the channel in a channel address word (CAW) bits 0-3.

The selected SP reads out the address location corresponding to the block of processor storage addresses that includes the address of this particular request. A comparison is made between the prestored key and the key sent to the SPF on this operation if either:

1. This is a store operation, or
2. This is a fetch operation and the read-protect bit in the SPF location is a 1 bit.

The comparison is for a bit-by-bit match of the two keys; also, the keys are considered to match if a key of all 0s was sent to the SP for this operation. If the keys match, the requested storage operation proceeds normally; a mismatch, however, prevents changing storage on a store operation or inhibits data delivery on a fetch operation. In case of key mismatch, a storage address protect (SAP) error is

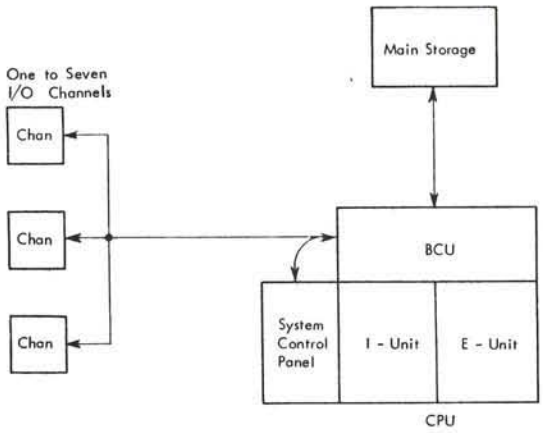


FIGURE 1. SYSTEM/360, MODEL 75

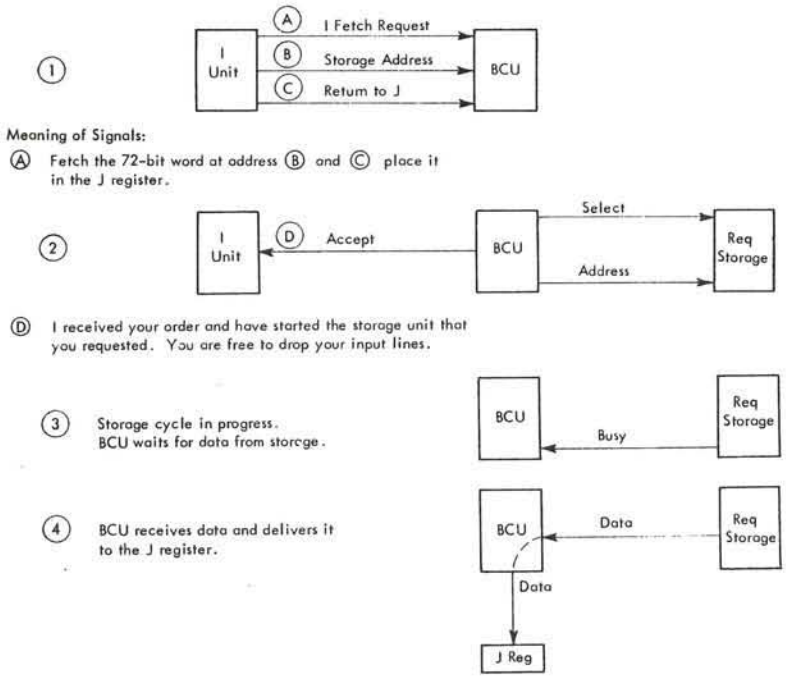


FIGURE 2. I UNIT FETCH

Low-Order Address Bits						Decimal Address	Doubleword Address (bits 0 through 20)
18	19	20	21	22	23		
1	0	0	1	1	1	39	4
1	0	0	1	1	0	38	
1	0	0	1	0	1	37	
1	0	0	1	0	0	36	
1	0	0	0	1	1	35	
1	0	0	0	1	0	34	
1	0	0	0	0	1	33	
1	0	0	0	0	0	32	
0	1	1	1	1	1	31	3
0	1	1	1	1	0	30	
0	1	1	1	0	1	29	
0	1	1	1	0	0	28	
0	1	1	0	1	1	27	
0	1	1	0	1	0	26	
0	1	1	0	0	1	25	
0	1	1	0	0	0	24	
0	1	0	1	1	1	23	2
0	1	0	1	1	0	22	
0	1	0	1	0	1	21	
0	1	0	1	0	0	20	
0	1	0	0	1	1	19	
0	1	0	0	1	0	18	
0	1	0	0	0	1	17	
0	1	0	0	0	0	16	
0	0	1	1	1	1	15	1
0	0	1	1	1	0	14	
0	0	1	1	0	1	13	
0	0	1	1	0	0	12	
0	0	1	0	1	1	11	
0	0	1	0	1	0	10	
0	0	1	0	0	1	9	
0	0	1	0	0	0	8	
0	0	0	1	1	1	7	0
0	0	0	1	1	0	6	
0	0	0	1	0	1	5	
0	0	0	1	0	0	4	
0	0	0	0	1	1	3	
0	0	0	0	1	0	2	
0	0	0	0	0	1	1	
0	0	0	0	0	0	0	

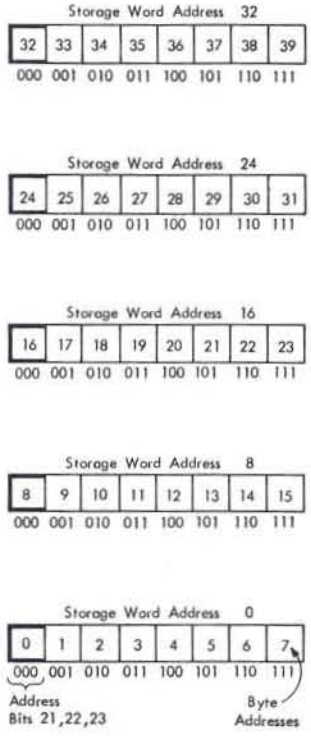
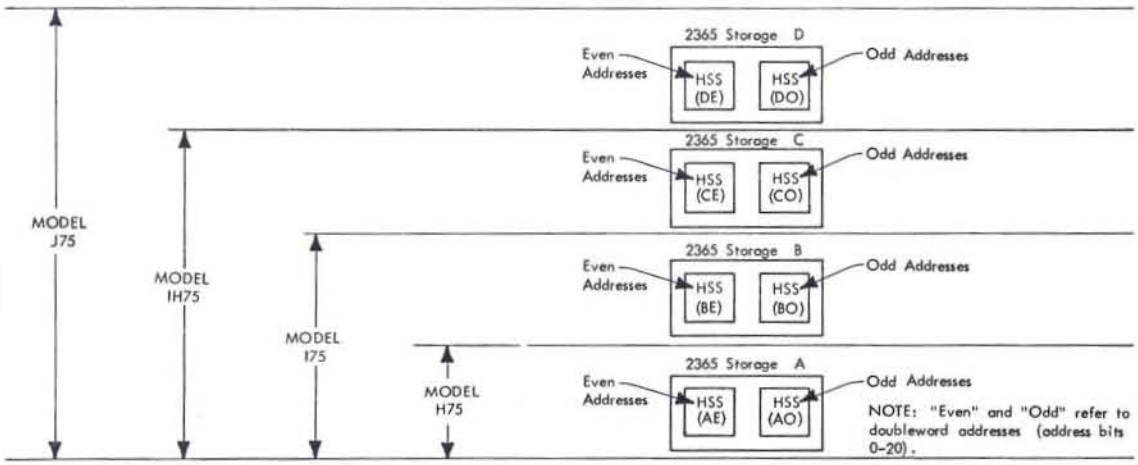


FIGURE 3. STORAGE WORDS



● FIGURE 4. MODEL 75 MAIN STORAGE

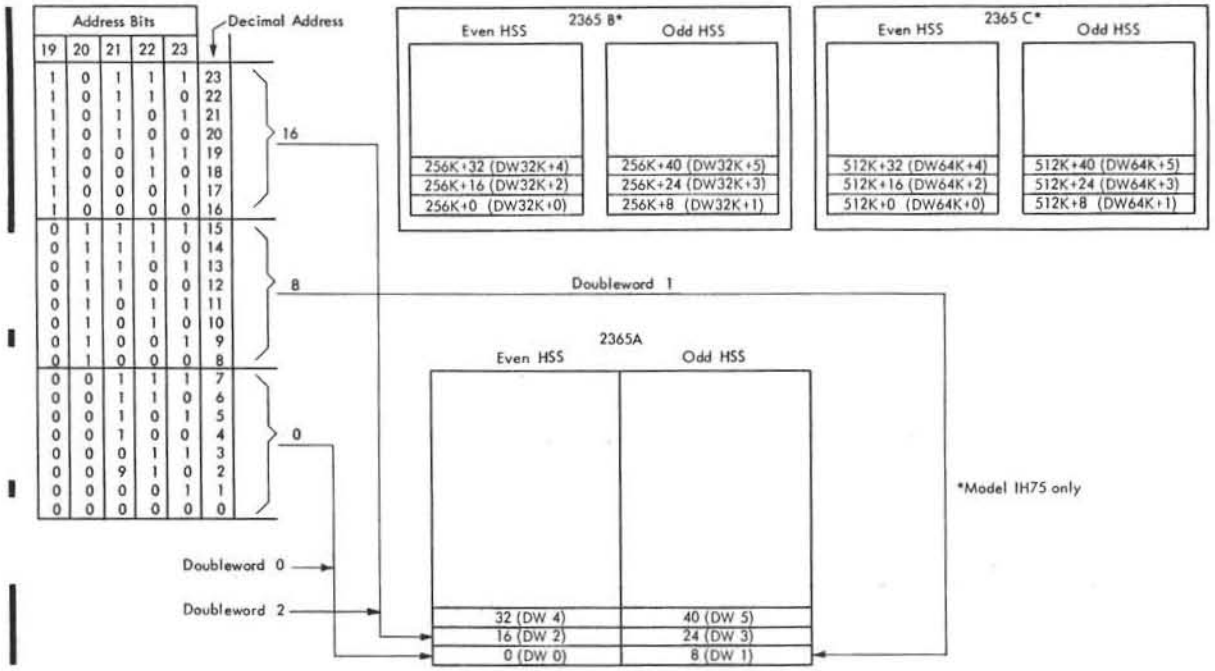
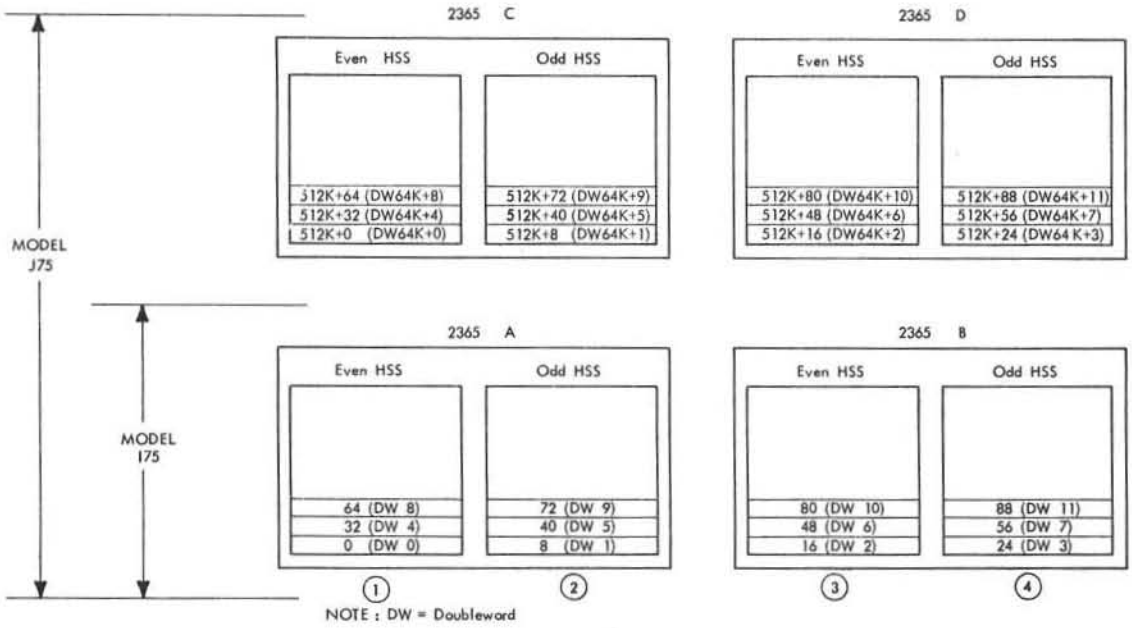
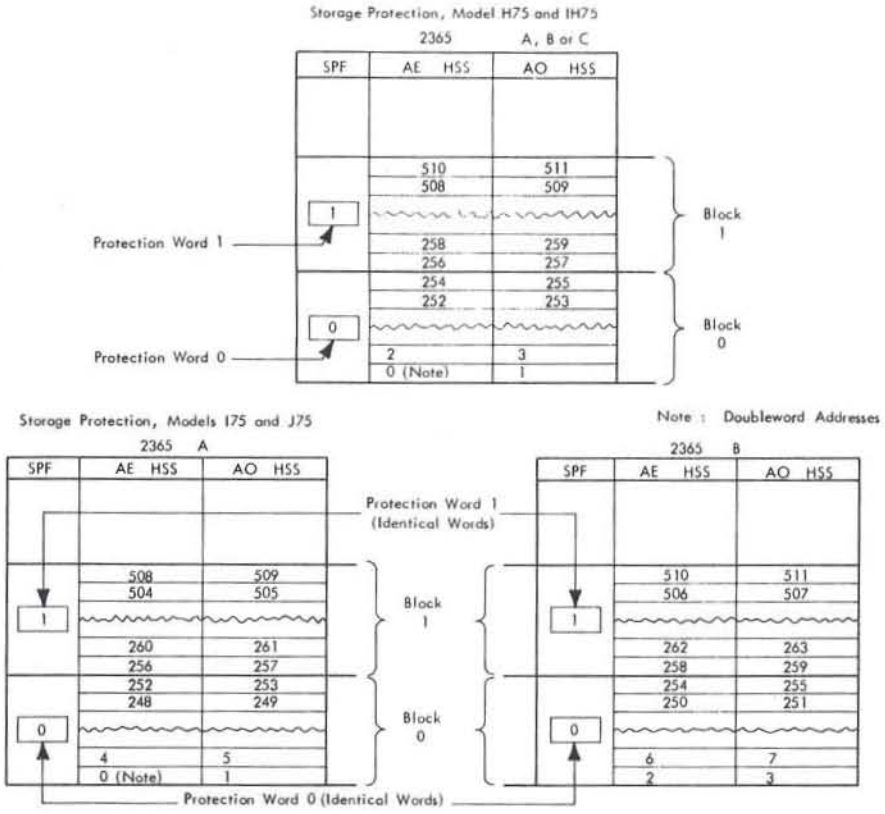


FIGURE 5. TWO-WAY INTERLEAVING, MODEL H75 AND MODEL IH75



● FIGURE 6. FOUR-WAY INTERLEAVING, MODELS I75 AND J75



● FIGURE 7. STORAGE ADDRESS PROTECTION

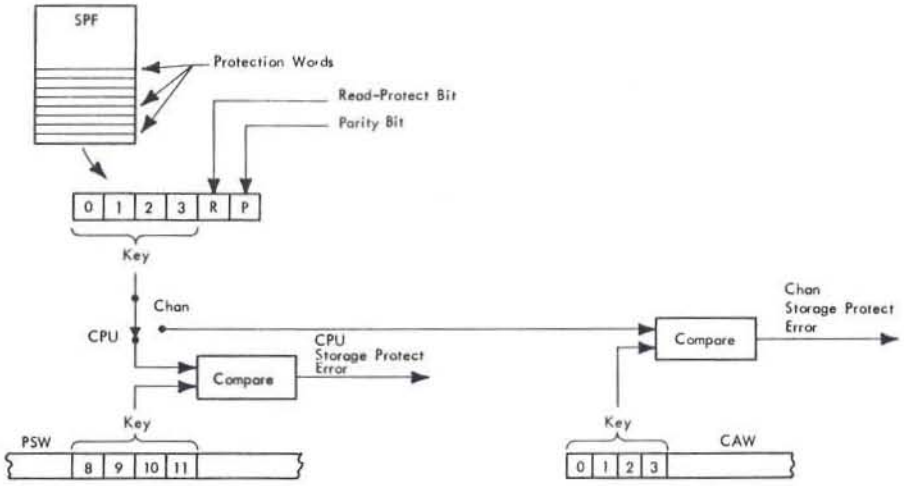


FIGURE 8. SPF PROTECTION WORD

signaled. This error causes a program interrupt if the storage request was from the CPU or a channel interrupt if the request was from a channel.

The contents of any SP location can be set into a general register by an insert key instruction.

DATA FLOW

- Storage address is set into SAR for CPU requests.
- Address bits needed to select a particular HSS are duplicated in the duplicate SAR.
- CPU store data is set into K.
- SBI latch is set from K on CPU store operations; SBI latch is set from channel SBI on channel store operations.
- All channels use a common storage address bus (channel SAB).
- Fetched data is temporarily stored in the SBO latch.
- X and Y return address registers route fetched data (and some error indications) to the proper destination.

The job of the BCU is to:

1. Grant priority to one storage request at a time.
2. Select the proper storage unit by examining the address bits.
3. Route address, protection key, store data, and mark bits to the selected storage unit.
4. Route fetched data returning from storage to the requester.
5. Handle error conditions detected by BCU, storage, and SPF.

The data flow to accomplish these jobs is explained in the following text. Refer to Figure 2005.

For CPU requests, the storage address is set into the storage address register (SAR) from either the incrementer or the addressing adder. The address for any CPU request is always in the SAR when the request is made or is being set into the SAR simultaneously with the request. This address remains in the SAR until BCU honors the request. When BCU honors a CPU request, SAR is gated through the address OR to storage. Note that some bits of SAR are duplicated in the duplicate (dup) SAR. These are bits needed by BCU to select a particular storage unit (HSS). The select bits are duplicated to make them available to BCU controls as soon as possible.

For a CPU request, the protection key field of the PSW is gated through the key gate and the key OR to storage. On a set key instruction, five bits of general register R1 are substituted for the PSW storage protection key field.

If the CPU requests a store operation, the data to be stored is in the K register at the time of the request (or will be set into K one cycle following the request), and will remain there until BCU honors the request and sets the data into the storage bus in (SBI) latch register. The SBI latch is fed directly to all storage units. A store operation also requires eight mark bits to be sent to storage along with the address bits. For a CPU store operation, the mark register has the proper mark bits at the time the request is made. The BCU gates the mark register through the mark OR to storage, then resets the mark register.

All channels and the system control panel use a single set of lines to deliver store data to the SBI latch register. This set of 64 lines (plus eight parity lines) is the channel storage bus in (channel SBI). In the same way, all channels and the system console deliver addresses on one set of lines (CAB or channel SAB), storage protection keys on the channel key bus, and mark bits on the channel mark bus. When BCU selects a storage for a channel, the CAB is gated through the address OR to storage, and the channel protection key is gated through the key OR to storage. On channel store operations, the SBI latch is set from the channel SBI and the channel mark bits are gated through the mark OR to storage.

BCU temporarily stores all fetched data in a 64-bit (plus eight-parity bit) register (the storage bus out (SBO) latch register). This register has outputs to the A, B, and J registers in the CPU and also feeds the channel SBO. It is the job of the BCU to see that a fetched storage word is delivered to the register specified by the user when the request was made. The BCU does this by sending an advance pulse to one of the receiving areas. For example, if the data is destined for the A register, BCU sends an advance signal which gates the setting of the A register from the SBO latch register. The proper advance pulse is generated by the two return address registers, X and Y. These registers are used alternately because of overlapped storage operations.

STORAGE SELECTION AND CONTROL

- The BCU maximum storage selection rate is one selection every two machine cycles.
- Any two HSS cycles can be overlapped.
- BCU keeps track of busy HSS.
- Channel bus priority circuits grant priority to one channel at a time.
- Channel requests have priority over CPU requests.
- CPU has a three machine-cycle access time to data in main storage.
- Maximum channel rate for main storage data is about 1 microsecond per double word.

The BCU can select storage units (HSS) at the rate of one selection every two machine cycles. This means that any two storage units can be simultaneously busy (overlapped, Figure 9). BCU maintains a busy trigger for each storage unit: two for Model H75, four for Model I75, six for Model IH75, and eight for Model J75. By examining certain address bits, the BCU determines which unit is being requested; if the BCU is not busy and the requested storage unit is not busy, the selection is made.

With each selection, the BCU is considered busy for two machine cycles (Figure 10). Alternate selections can be made to one HSS without interference from a busy trigger. Two consecutive requests for the same HSS, however, require a delay of four machine cycles between the first and second selections. Therefore, to achieve the maximum storage selection rate of one selection every two machine cycles, consecutive requests must not be for the same HSS and the BCU must always have a storage request waiting to be serviced. Actual storage requests will not always fulfill these requirements. However, interleaved addressing means that more requests are for a nonbusy storage than would be the case if consecutive addresses were in the same storage array: Instructions are generally fetched from sequential addresses; a channel works into and out of sequential addresses; and arithmetic operations often store or fetch into sequential addresses.

Requests are honored according to a fixed priority scheme. This scheme has two levels of priority. On the first level, the channel bus priority circuits grant priority to one channel (or the system control panel) at a time. The channel bus priority circuits are necessary to prevent interference on the channel buses (CAB, channel SBI, and channel SBO). Because the seven channels and the system control panel operate independently of each other and of the CPU, any number of these eight storage users may simultaneously request storage. When there are simultaneous requests, the BCU must allocate the use of the channel buses to one user at a time. The BCU allocates the use of the channel buses in a fixed priority scheme; channel 0 has the highest priority, channel 1 next-to-highest, and so forth. For example, channel 6 cannot access storage if any other channel is making a request. The system control panel, or maintenance channel, has the lowest priority and can access storage only when none of the channels are making a request.

Once a channel gains bus priority, it puts the storage address on the channel SBI (CAB). The entire process of recognizing a channel request, granting priority to the requesting channel, and receiving the storage address on CAB requires about 1 microsecond. During this time, the BCU will honor CPU requests even though channels have priority over the CPU. A channel is not considered to have a valid request until its storage address arrives in BCU and a line, address valid, is generated.

The second level of priority is to grant any channel that has generated address valid (and is requesting a nonbusy HSS) priority over the CPU.

The CPU is said to have a three-cycle access to storage (Figure 11). Assuming no interference from channels and a request for a nonbusy HSS, a fetched storage word will be set into A, B, or J three cycles after the BCU recognizes the request.

A channel access requires a minimum of about 1-1/2 microseconds from the time that BCU recognizes the request until the fetched word is in the channel registers. Most of this time is consumed in signal travel time to and from the channel. About 300 nanoseconds is required for a signal to travel to or from a channel. Although a single channel fetch requires about 1-1/2 microseconds, channels can access storage at a 1-microsecond rate. This 1-microsecond rate is possible because BCU overlaps the preparation for a new channel operation with the storage cycle and data return of the previous channel operation. As soon as a HSS is selected for a channel, the BCU tells the channel to remove its storage address from the CAB, and (assuming another channel request is pending) at the same time, tells the new channel to put its storage address on the CAB.

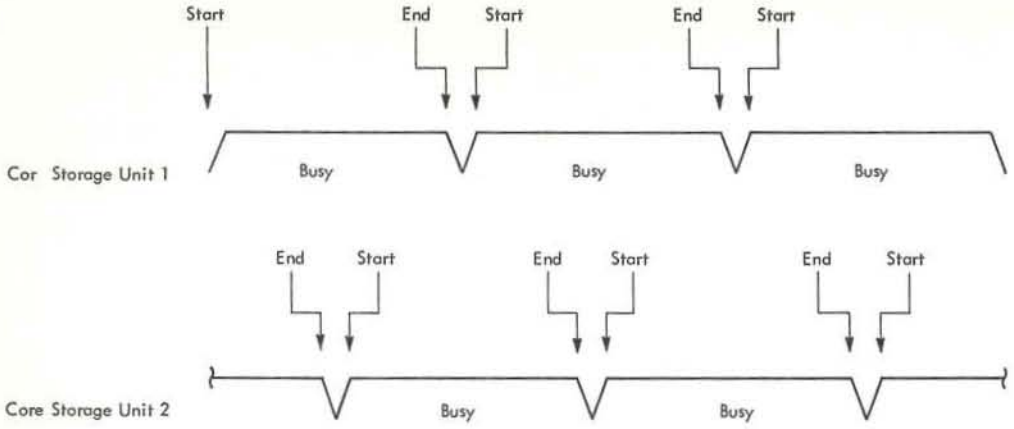


FIGURE 9. OVERLAPPED STORAGE CYCLES

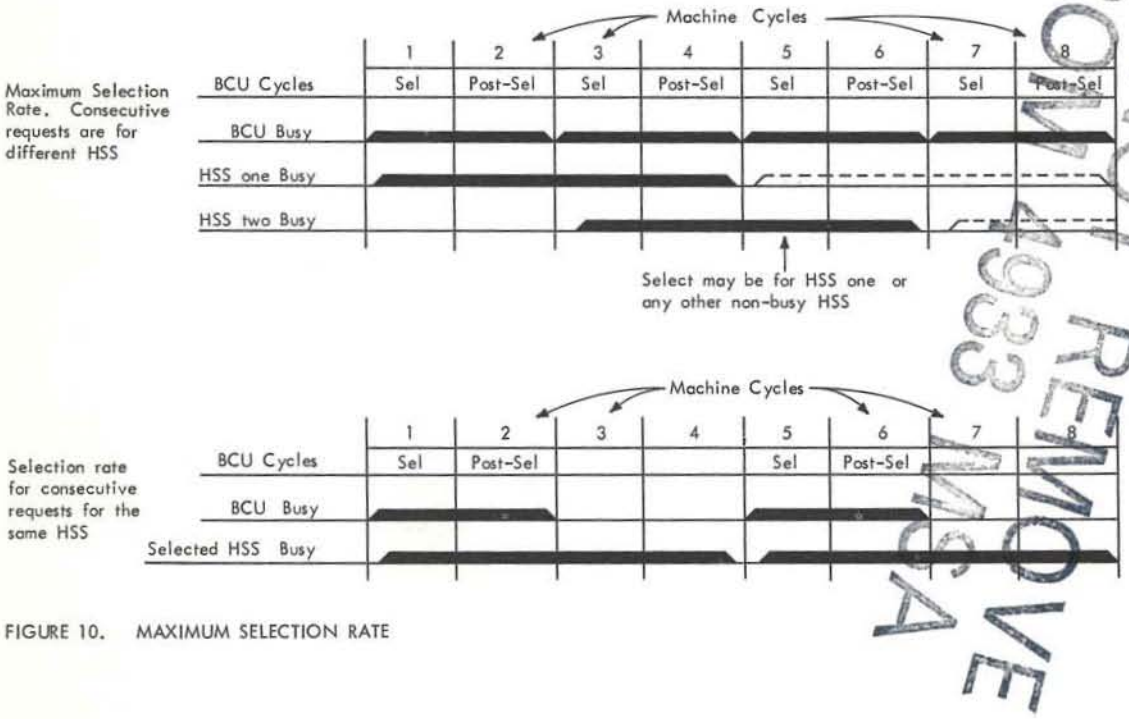


FIGURE 10. MAXIMUM SELECTION RATE

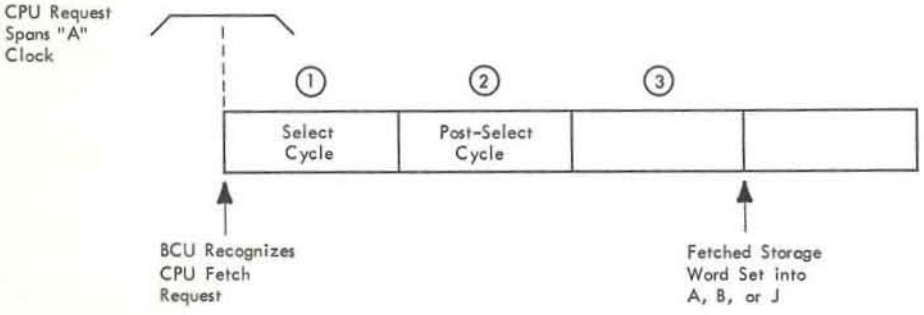


FIGURE 11. MACHINE CYCLES FOR CPU ACCESS

CRITICAL TIMING LOOP

- CPU is sometimes forced to wait for data from storage.
- Waiting periods are minimized by using buffer registers: A, B, and J.
- CPU access time consists of two parts: communication time and storage time.
- A primary job of the BCU is to minimize communication time.

In executing a program, the CPU is sometimes forced to wait for data from storage before continuing. In the Model 75, these periods of waiting are minimized by using instruction buffer registers (A-B), an operand buffer register (J), and by requesting storage fetches as soon as possible. There are times, however, when the CPU must await the return of data from storage. For example, a successful branch instruction cannot be terminated until a new instruction double word is in A or B. Also, on many sequences of instructions such as two consecutive E cycle RX instructions, the CPU must wait for operand deliveries from storage. Because of these times when the CPU is forced to wait for fetched data, the performance of the CPU depends to a large extent on the CPU storage access time.

Two time elements make up the CPU storage access time (Figure 12): access time of the storage unit and speed of CPU communication with the unit.

Fetched data is available at the storage frame about 450 nanoseconds after the start of the storage cycle. The time required to get a storage unit started and the time required to move fetched data from a storage into a CPU register is the communication or BCU time. Stringent time requirements are imposed on the BCU to minimize this communication time enough to achieve the three machine-cycle CPU access time (Figure 11). The storage selection circuits, for example, are designed for maximum speed rather than for the minimum number of modules required to accomplish the logical objectives. Where minimum delay is essential, the BCU uses parallel cabling and parallel logic (Figure 13).

The philosophy used to achieve minimum CPU storage access time is to consider that a CPU-storage-CPU loop exists, starting from A-B and ending at A-B (Figure 14). This critical timing loop is used on branches. The timings throughout this loop are tuned for maximum speed. The release of SAR, for example, is adjusted to coincide (approximately) with the time when the output of the AA begins to rise. Therefore, as the AA output rises, it passes through a released SAR and is on its way to storage long before SAR is locked. This philosophy in timing the release of SAR is used throughout the loop to allow the address to flow to storage and the data to flow from storage without being blocked by a locked register. All CPU operations do not use all of this critical timing loop. By using this path as a criterion, or "worst case," however, the timing of other storage paths is noncritical.

CPU FETCH, THE BASIC OPERATION

- CPU fetch is initiated by the I or E unit.
- BCU delivers fetched storage word to the A, B, or J register.
- BCU sends 14 address bits from SAR to a HSS.
- BCU sends accept to the CPU when select is sent to HSS.
- Selected HSS sends advance to BCU just prior to data delivery.

The basic BCU operation is a CPU fetch. All other operations performed by the BCU are variations of the CPU fetch operation. A CPU fetch is a storage fetch requested by either the I or the E unit. The BCU starts storage to get the 64-bit plus eight-parity bit word requested, then delivers this to the A, B, or J register.

The CPU requests a fetch operation by sending I fetch request or E fetch request to the BCU (Figure 15). Along with the request, the CPU sets the desired address into the SAR and the duplicate SAR, and specifies that the data be returned to the A, B, or J register.

The BCU examines duplicate SAR bits to determine which HSS is being requested. If the requested HSS is not busy, the BCU generates a select pulse for this HSS. If the CPU specifies that the data is to be returned to A-B, the BCU uses duplicate SAR bit 20 to choose the proper register. Bit 20, when on, means that an odd address is being requested, and the fetched data is to be returned to the B register. When off, bit 20 means an even address; even instruction words go to the A register.

On CPU fetch operations, BCU gates the SAR through the address OR to all storages. Each 2365 has two memory address registers (MAR): one for the even HSS and one for the odd HSS. The 14-bit plus two-parity bit address gated from the address OR is set into the MAR of the HSS that receives the select pulse.

Part of the 14-bit storage address is sent to the storage protect (SP) unit associated with the selected HSS to address the SP location that corresponds to the requested HSS address. The protection key from the PSW is routed to the selected SP unit.

When the select is sent to storage, the BCU sends accept back to the CPU. This signal tells the CPU that its request has been honored and it can now drop its request and change SAR. It can make a new request if one is pending.

About two cycles after select, the selected HSS sends an advance signal to the BCU (Figure 16). Half a cycle after advance, the selected HSS has the fetched 72-bit word gated from its memory data register (MDR) to the SBO latch register. Shortly thereafter, BCU releases the SBO latch. After releasing the SBO latch, BCU sends an A, B, or J advance signal to the CPU. The CPU uses the advance signal as a gate to set the receiving register with the next A clock. (A and B are actually released with a late B running (LBR) clock of the previous cycle.)

CPU Fetch Error Detection

- Storage address parity is checked by BCU and by storage.
- If BCU detects an error, BCU cancels the selected HSS.
- When a HSS is cancelled, the selected address is regenerated and BCU does a panel key fetch.
- BCU checks for an invalid storage address.
- Storage checks the parity of the fetched word; however, BCU ignores a data check from storage on CPU fetches.
- The SP unit checks for a storage address protection violation.

On every storage operation, the BCU checks the storage address for good parity. The check is made off the address OR and is on the full 24-bit address. If the address has bad parity, a cancel signal is sent to storage. The cancel condition causes storage to regenerate the selected address without delivering the data brought out. Storage suppresses any errors that it detects if cancel is on.

The cancel condition causes BCU to do a panel key fetch operation. This operation consists of setting the system control panel data keys into the SBO latch register so that the receiving register gets a good parity word. The receiving register gets meaningless data but an erroneous data parity check is prevented. If the receiving register were not set(left all zeros including parities), the CPU would generate a parity error when it checks the parity of the data it receives. Any address parity error detected on a CPU initiated storage cycle is a machine check, and it initiates a CPU logout followed by a machine check interrupt.

The BCU also checks for an invalid storage address on every storage access. An invalid address is an address outside (higher than) the available storage locations on a particular system. If an invalid address is detected, the BCU again cancels storage and does a panel key fetch operation. An invalid address error detected on a CPU initiated storage cycle is a program error and it causes a program interrupt.

On fetch operations, the selected storage checks the parity of the 14-bit address that it receives and checks the parity of the 64-data bits read out of the addressed location. If an address error is detected, the fetched word is not delivered: storage delivers a data word of all 0s with good parity (all parity bits are 1s). Storage also sends a storage address error line to the BCU. For a CPU initiated operation, the storage address error line brings up the machine check condition (logout followed by a machine check interrupt).

If storage detects a parity error on the fetched data word, a storage data error line is sent to the BCU. For CPU fetch operations, however, the data error line from storage is ignored; the CPU sometimes fetches data that will not be used. Any fetched data actually used by the CPU is parity-checked prior to use.

A storage address protect (SAP) error is another programming error that can be generated on a CPU fetch operation. This error is generated by the selected SP unit only if the read-protect bit in the addressed SP location is a 1 and there is a mismatch between the SP key (match bits) and the key bits from the PSW.

The SP unit parity-checks the address bits that it receives and parity-checks any protection key data that it uses. If the SP unit finds a parity error, it generates a storage address error which is handled exactly the same as an address error detected by the selected HSS.

CPU STORE OPERATION

- BCU routes SAR, mark register, and PSW key to storage.
- BCU sends select and store to storage.
- BCU sets K into SBI latch register and sends SBI latch register to storage.

The CPU requests a store operation by sending I store request or E store request to the BCU (Figure 17). Along with the request, CPU sets the desired address into SAR and duplicate SAR just as on a CPU fetch operation. On a store operation, however, the CPU must set the mark register along with the request or sometime prior to the request. The mark register is used only on CPU storage operations and is reset by the BCU after its contents have been sent to the selected storage. The CPU, therefore, is free to load the mark register anytime prior to a store operation.

The BCU generates a select to the requested HSS exactly as on a CPU fetch operation. The select is generated when neither the BCU nor the requested HSS is busy. The address (14 bits plus two parity bits) is gated from SAR through the address OR to storage and the CPU storage protect key (PSW 8-11) is gated through the key OR just as on a CPU fetch operation. The eight-mark bits (plus a parity bit) are gated from the mark register through the mark OR to the storage. Along with select, the BCU sends store to the selected 2365.

Just as on a CPU fetch operation, BCU sends accept back to the CPU to tell the CPU that its request has been honored and that it can now drop its request and change SAR. It can make a new request if one is pending.

When CPU initiates a store request, the data to be stored are either in the K register or will be set into K on the next cycle. On the post-selection cycle, the BCU sets the SBI latch register from K (Figure 18). All 72 bits are set into the SBI latch and sent to storage regardless of the number of bytes to be stored. The selected storage unit uses the mark bits to gate the corresponding bytes into its MDR. Those bytes of the MDR not set from the SBI are set by sense amplifiers at the end of the read portion of the storage read/write cycle. Then, the modified 72-bit word in the MDR is written back into the selected address on the write cycle.

CPU Store Error Detection

- The BCU makes the same checks as on a CPU fetch: address parity and invalid address.
- Storage checks address parity and data parity.
- The SP unit checks for mismatch of protection keys.

Just as on a CPU fetch operation, the BCU checks the parity of the storage address and checks for an invalid address. If either of these conditions are found, a cancel signal is sent to the selected HSS to cause the selected address to be regenerated without change. A bad parity address causes a machine check interrupt; an invalid address causes a program interrupt.

The selected storage checks the parity of the 14-bit address that it receives and checks the parity of the mark bits. If an error is found, storage regenerates the selected address and signals an address error to the BCU. Just as on a CPU fetch operation, a storage address error causes a machine check interrupt.

Storage also checks the parity of the data in the MDR at the time when the MDR contains the new 72-bit word to be written back into the array. Bad parity does not alter the store operation, but a storage data error line is sent to the BCU where a machine check condition is generated. A machine check occurs if any of the eight bytes have bad parity. The bad parity byte may have come from K or it may have been read out of the selected double word location in the array.

The SP unit checks for a match of the protection keys on all store operations. If there is a mismatch, a SAP error is signaled to cause a program interrupt.

Just as on a CPU fetch operation, the SP unit checks the parity of the address bits that it receives and the parity of the protection key information that it uses. If bad parity is found, the storage address check line is raised; action taken for an SP parity error is identical to the action taken for a HSS address error.

CHANNEL BUS PRIORITY

- Channel storage requests occur at random; two or more requests may occur simultaneously.
- Channel operations are done in two steps: priority and storage selection.
- Priority circuits operate independently of storage selection circuits.
- BCU grants priority to a channel by sending BCU response to the priority channel.
- When granted priority, a channel puts the storage address on the CAB.

Channel operations consist of two parts: channel bus priority and channel storage selection.

The BCU handles these two parts of channel operations independent of each other (Figure 19). The channel bus priority circuits examine channel requests,

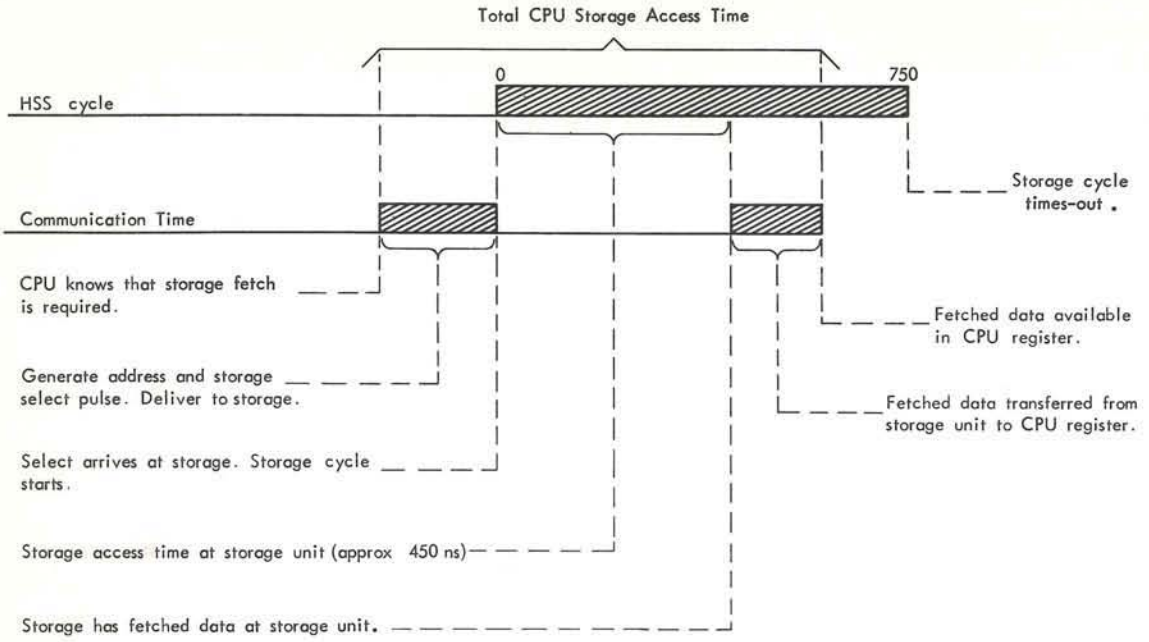


FIGURE 12. CPU STORAGE ACCESS TIME

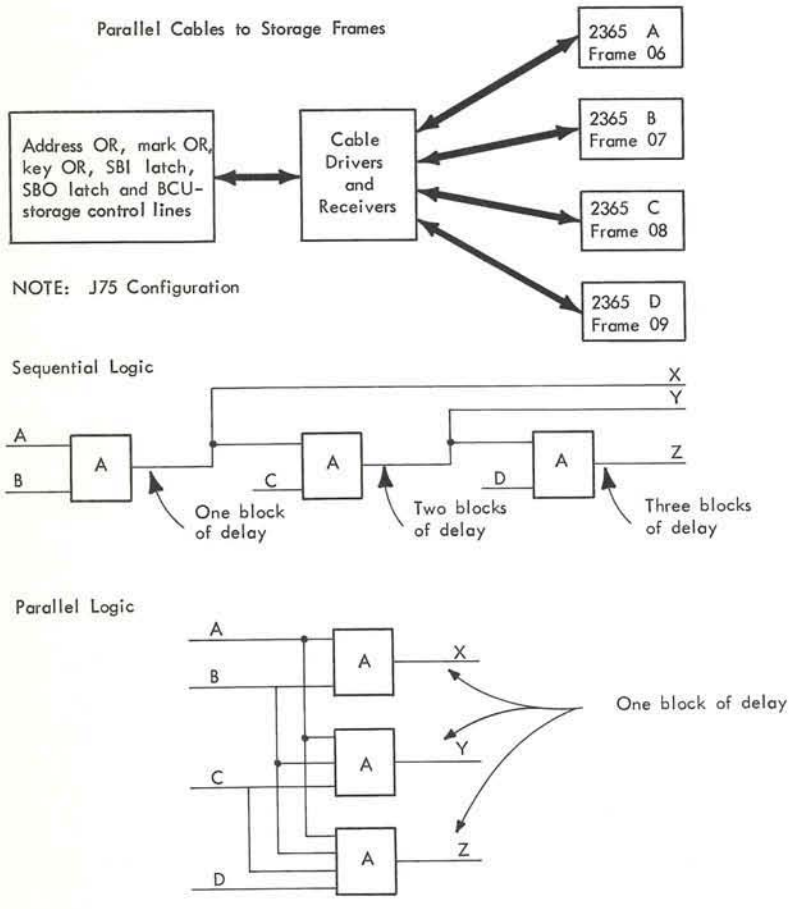


FIGURE 13. PARALLEL CABLES AND PARALLEL LOGIC

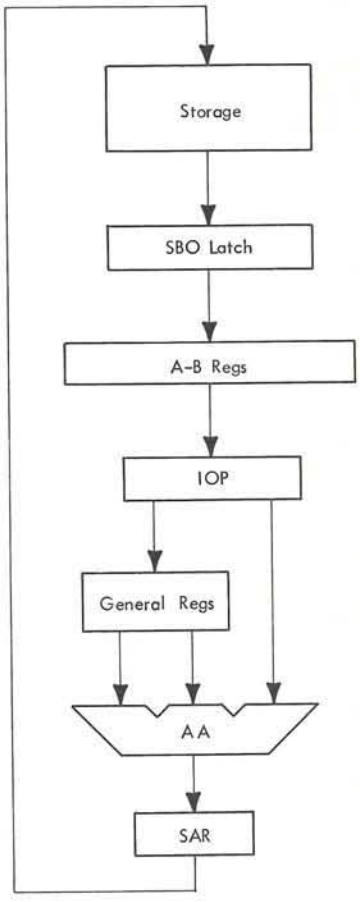


FIGURE 14. CRITICAL TIMING LOOP

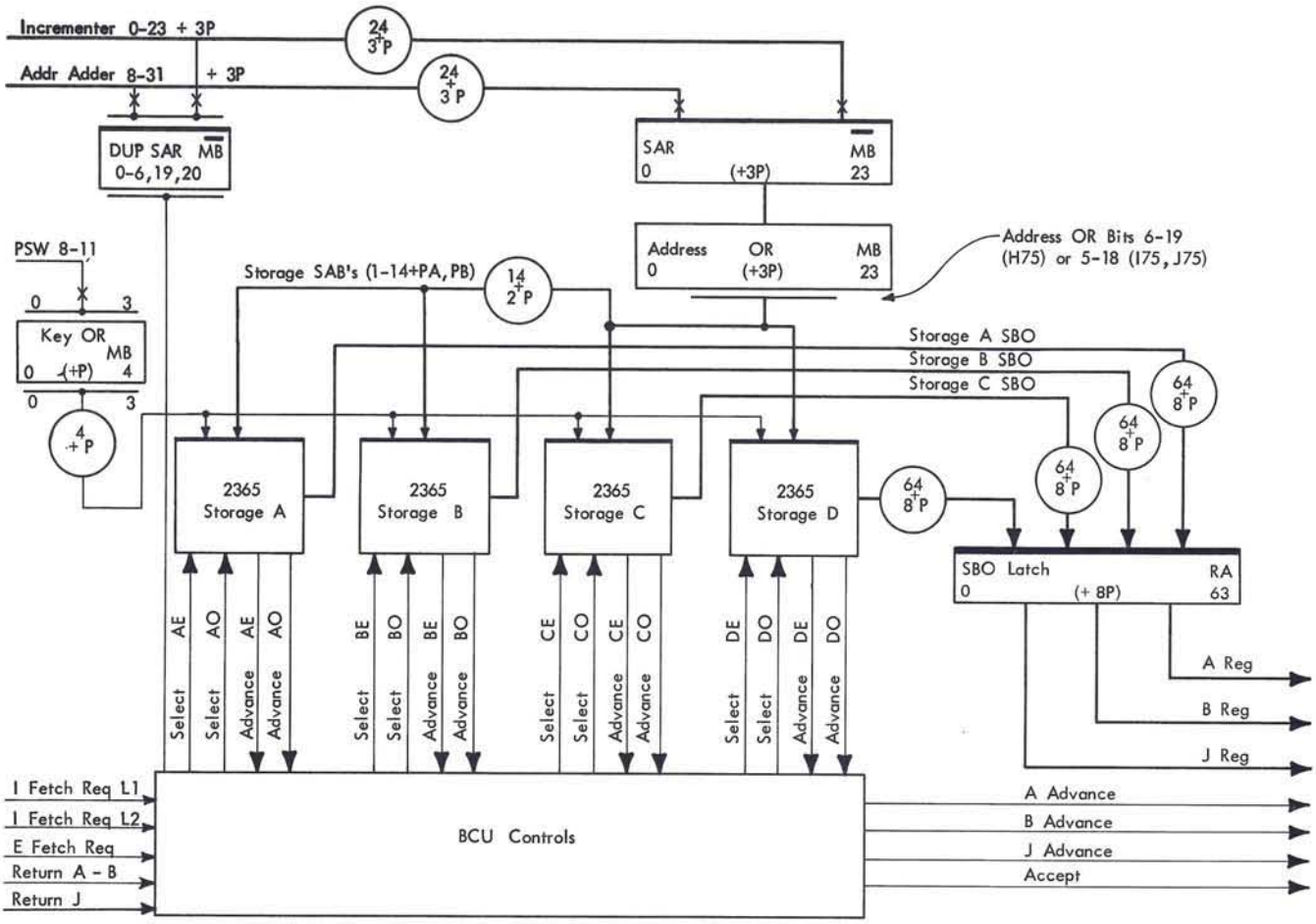


FIGURE 15. DATA FLOW, CPU FETCH

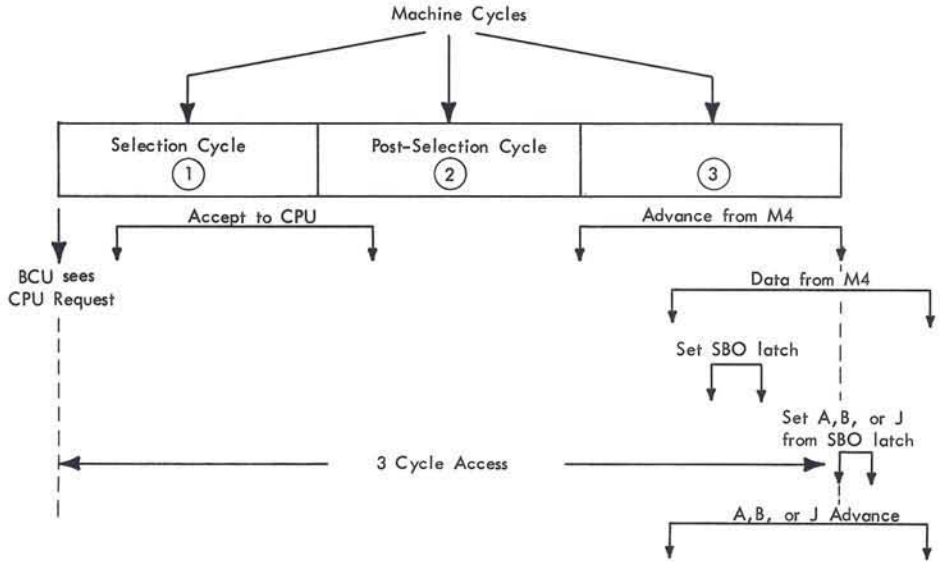


FIGURE 16. CPU FETCH TIMING

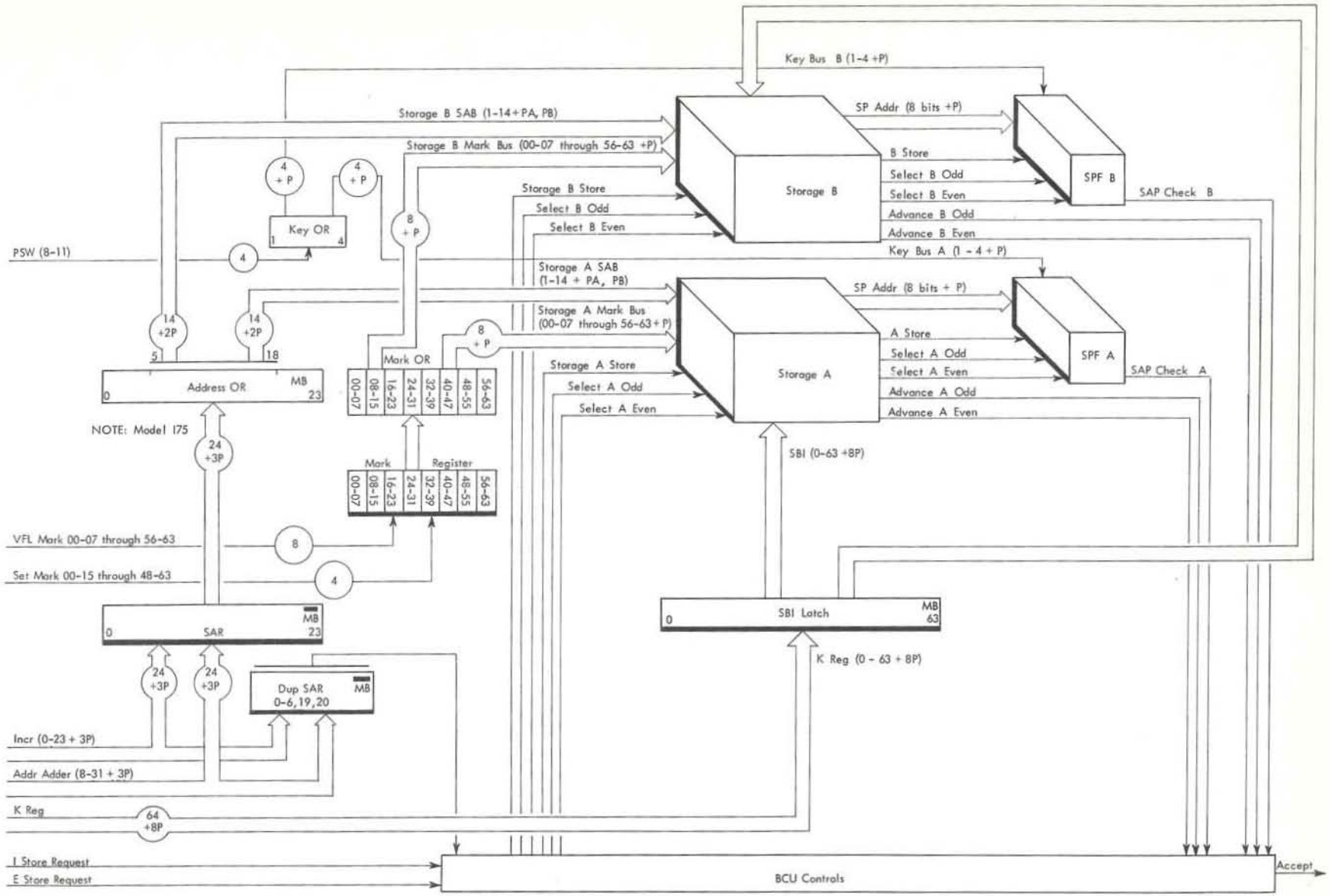


FIGURE 17. DATA FLOW, CPU STORE

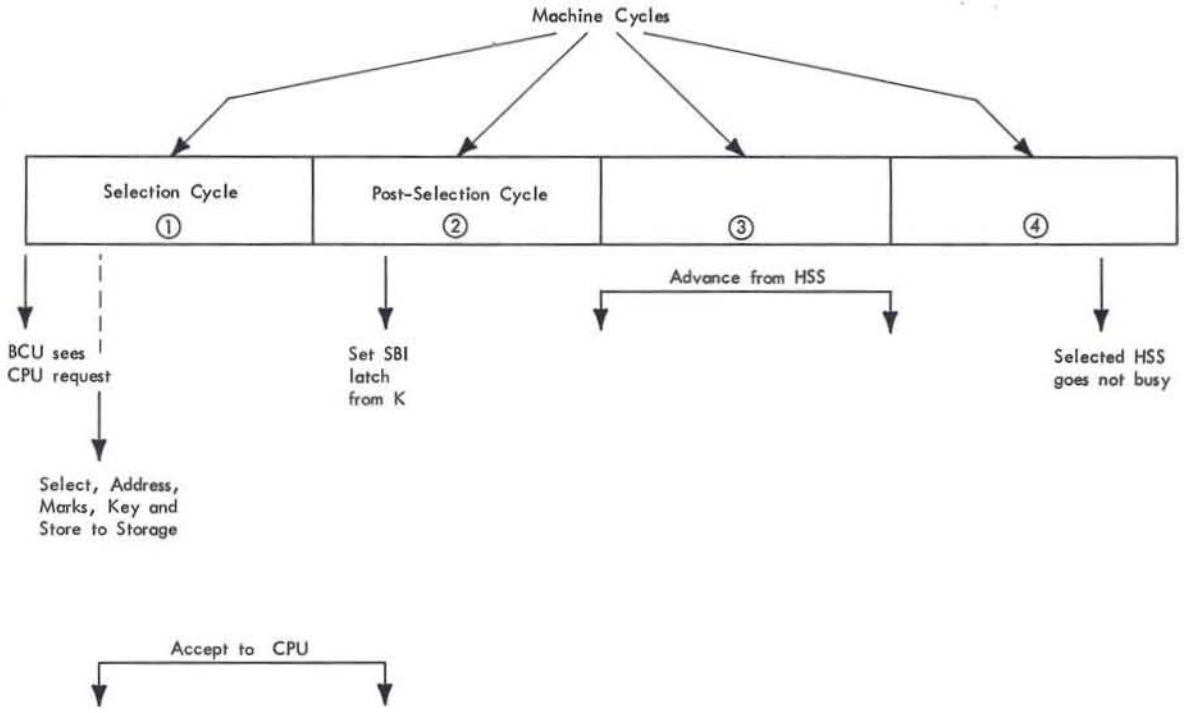


FIGURE 18. CPU STORE TIMING

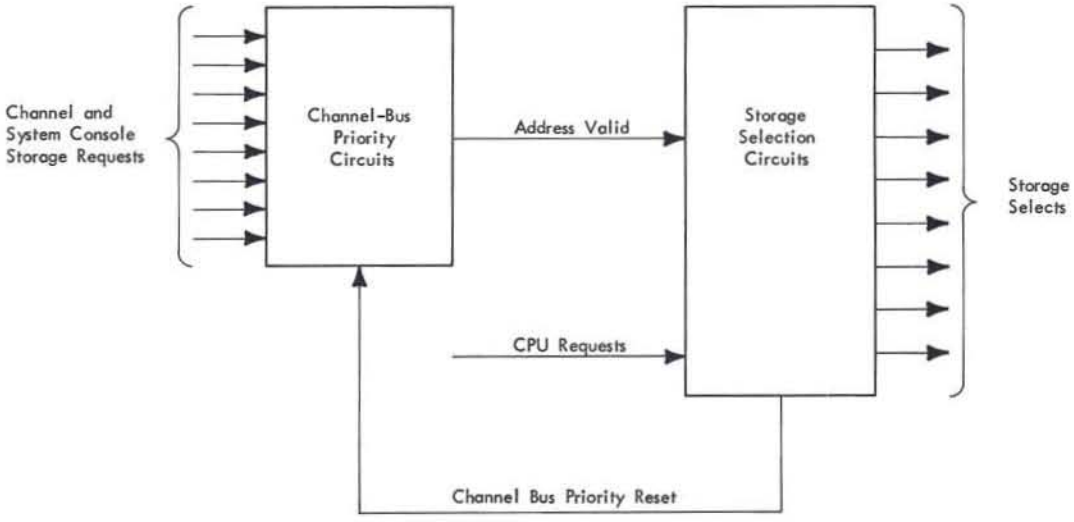


FIGURE 19. BCU HANDLING OF CHANNEL REQUESTS

grant priority to one channel at a time, and obtain the storage address from the requesting channel. When the channel bus priority circuits obtain a channel address, an address valid line is sent to the selection circuits. The address valid signal blocks CPU requests when the HSS requested by the channel is available.

The channel bus priority circuits operate independently of the selection circuits until address valid is generated. At this point, the channel bus priority circuits are locked-up until the selection circuits select the storage requested by the waiting channel. When the selection circuits generate this storage select, a channel bus priority reset line is generated to free the channel bus priority circuits. Once free, the channel bus priority circuits continuously search for a channel storage request until one is found.

The channels bring up storage request any time they require a storage access. (Channel includes the system console or maintenance channel.) Unlike CPU requests, channel requests are completely asynchronous with each other and with BCU and CPU cycles; therefore, two or more channel requests may occur simultaneously (Figure 20). Whenever the channel buses are not in use (as indicated by the buffer trigger being off), the BCU checks for channel requests once per machine cycle. The checking is done from channel 0 through channel 6 and then the maintenance channel in that order. The first request that is found (highest priority) is honored, and any other requests are ignored.

BCU honors a channel request by sending that channel a BCU response signal. This signal tells the channel that it gained channel bus priority and should respond by putting the address of the desired storage location on the channel storage address bus (CAB). Along with the storage address, the channel sends an address valid signal to the BCU. After a delay to allow the storage address to travel from the channel to the BCU, the BCU gates the address valid signal from the channel to the storage selection circuits. The selection circuits also receive CAB bits 4, 19, and 20 to determine which HSS is being requested.

CHANNEL FETCH

- A series of signals is exchanged between the BCU and a channel that makes a storage access.
- The first signal is a simplex request line from the channel to the BCU.
- Second and third signals are simplex BCU response and BCU data request from the BCU to the channel.
- On channel fetch operations, the channel responds to the BCU with a multiplex storage address, a protection key, and an address valid signal.
- BCU sends the channel a multiplex channel accept signal when a storage selection is made for the channel.
- BCU sends the channel a multiplex channel advance signal just prior to data.
- BCU sends fetched data to the channel on a multiplex channel SBO.

A channel operation is controlled by a series of signals exchanged between the BCU and the channel being serviced. Some of these control signals use simplex lines; a simplex line is unique to or from a particular channel. Other signals use multiplex lines; a multiplex line goes to or comes from all channels. Channels share the use of multiplex lines; for example, all channels use a single set of address and data lines as previously explained.

The channel operation signal exchange begins with a channel that sends storage request to the BCU (Figures 21 and 22). Each channel has a storage request line (simplex) because any number of channels may simultaneously need an access to storage. When BCU grants priority to a channel, it responds to that particular channel with BCU response. The BCU response lines are also simplex because the BCU must tell one specific channel to put its address on the channel SAB (CAB). Because of signal travel time, the channel sends an address valid signal to tell the BCU controls when the address has arrived in the BCU. The channel also sends the storage protection key to the BCU on the channel key bus.

The BCU follows BCU reponse with another simplex line, BCU data request. The primary purpose of BCU data request is to tell the channel to put the incoming data on the channel storage bus in (SBI) if this is to be a store operation. At request time, however, BCU does not know if a channel is requesting a store or a fetch; therefore, the BCU data request signal is always sent, but the channel will not put data on the channel SBI for fetch operations. The channel does use BCU data request to drop its storage request to the BCU.

When BCU starts storage for the channel that has its address on the CAB, the BCU drops BCU response and sends channel accept. The channel accept signal is a multiplex line but channels condition this line with BCU data request and not BCU response. This conditioning ensures that only one channel will recognize channel accept when it is sent out. Channel accept tells the channel that its request has been honored

and that the data it required will be on the channel storage bus out (channel SBO) following the next channel advance pulse. The fall of BCU response tells the channel to take its storage address off the channel SAB.

When the selected storage sends advance, the BCU routes this signal to the channel. The channel delays channel advance to gate the requested data from the channel SBO into its registers.

CHANNEL STORE

- For a channel store operation, the initial sequence through BCU data request is identical to a channel fetch operation.
- The channel sends multiplex store line and multiplex mark bits along with storage address.
- The channel sends a double word of store data on the multiplex channel SBI in response to BCU data request.
- Just as on a channel fetch operation, BCU sends multiplex channel accept and channel advance signals to the channel; however, no data is delivered on the channel SBO.

A channel starts a channel store operation with a request exactly the same as it starts a channel fetch operation (Figure 23). The BCU grants priority and sends BCU response and BCU data request without knowing whether the channel desires to fetch or store.

The channel, upon receiving BCU response, puts the storage address on the channel SAB just as it does for a channel fetch operation. Along with the storage address and the storage protection key, however, the channel raises the store line to the BCU. Also, the channel puts the eight-mark bits (plus a parity bit) on the channel mark bus.

When the channel receives BCU data request, it not only turns off its request, but also puts the 72-bit store data word on the channel SBI.

After BCU response and BCU data request, the BCU waits for address valid just as on a channel fetch operation. With address valid, the BCU generates a select when BCU and the requested storage are free. Because the channel store line is up, the BCU also sends store to storage and gates the channel marks to storage along with the channel address bits (Figure 23).

On the post-selection cycle, the BCU sets the SBI latch register from the channel SBI. All 72 bits are sent to storage; the selected storage takes only the bytes which have corresponding mark bits, just as on a CPU store operation.

The channel accept and channel advance signals are sent back to the channel, just as on a channel fetch operation. On a store operation, however, no data is returned to the channel; and, therefore, the channel does not use the channel advance signal to gate the channel SBO.

RETURN ADDRESS CIRCUITS

- Return address registers remember the requested destination of fetched data from select until advance.
- Two return address registers are necessary because of overlapped storage cycles.
- Each register has six positions (no parity): A, B, J, channel, diagnose, and invalid.
- A, B, and J positions route fetched data to the corresponding CPU registers.
- Channel position is set for both channel fetch and channel store operations.
- Diagnose routes fetched data to the MCW register for diagnose instructions.
- Invalid routes invalid address error indication on CPU fetches and channel operations.
- X/Y binary trigger gates inputs to one return address register at a time.
- W/Z binary trigger gates outputs from one return address register at a time: W gates out X; Z gates out Y.
- X/Y trigger is switched by delayed select: W/Z trigger is switched by delayed advance.

The BCU must remember where to return fetched data. The requesting unit tells the BCU where to return the data when a fetch request is made. Several machine cycles later, the BCU uses this information to route the returning data.

Remembering where to return fetched data would be a simple task if the BCU always waited for the end of a storage cycle before initiating a new cycle. However, the BCU overlaps the operation of any two HSS units within main storage. To overlap storages, the BCU must remember two return addresses and associate these addresses with the correct storage cycle.

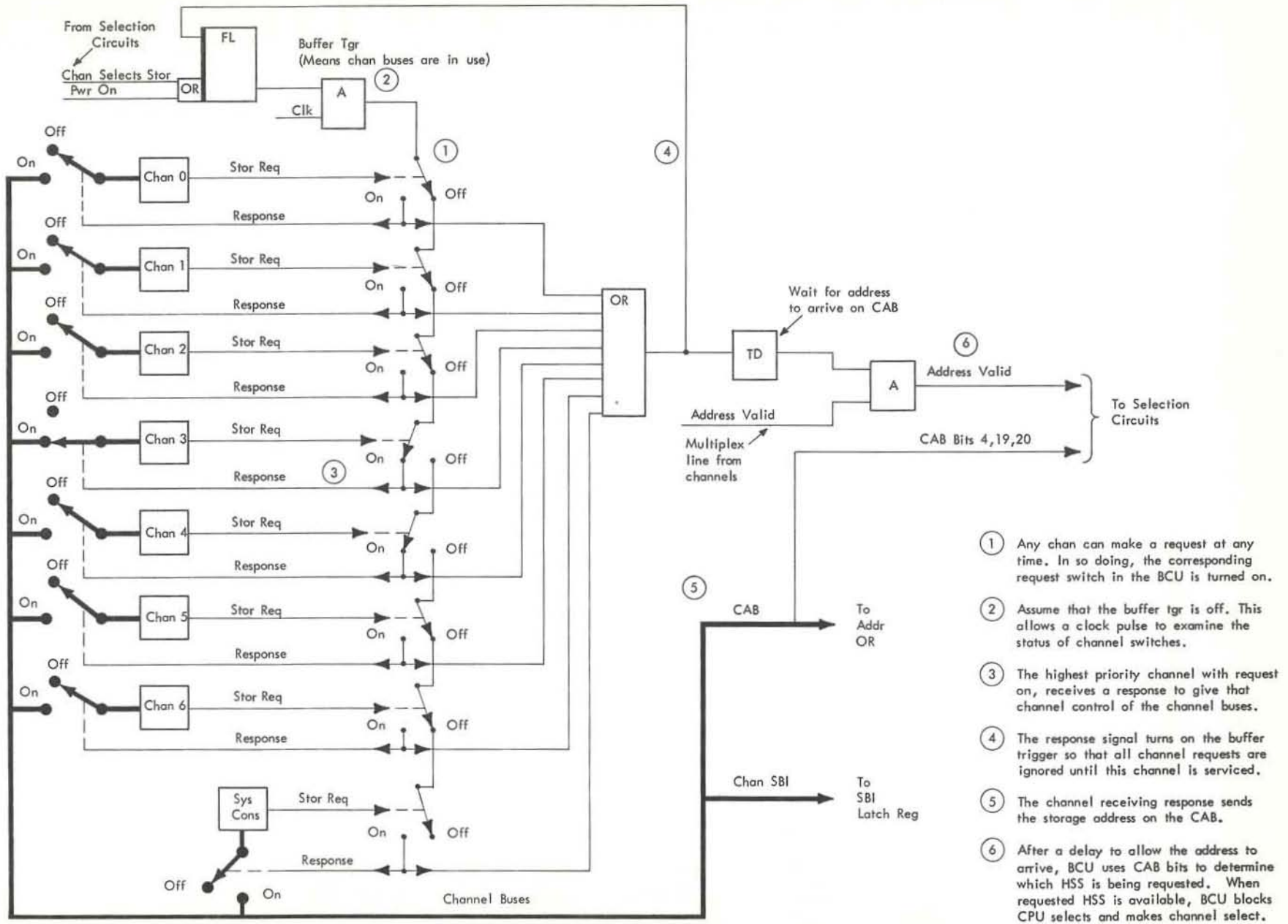


FIGURE 20. CHANNEL BUS PRIORITY

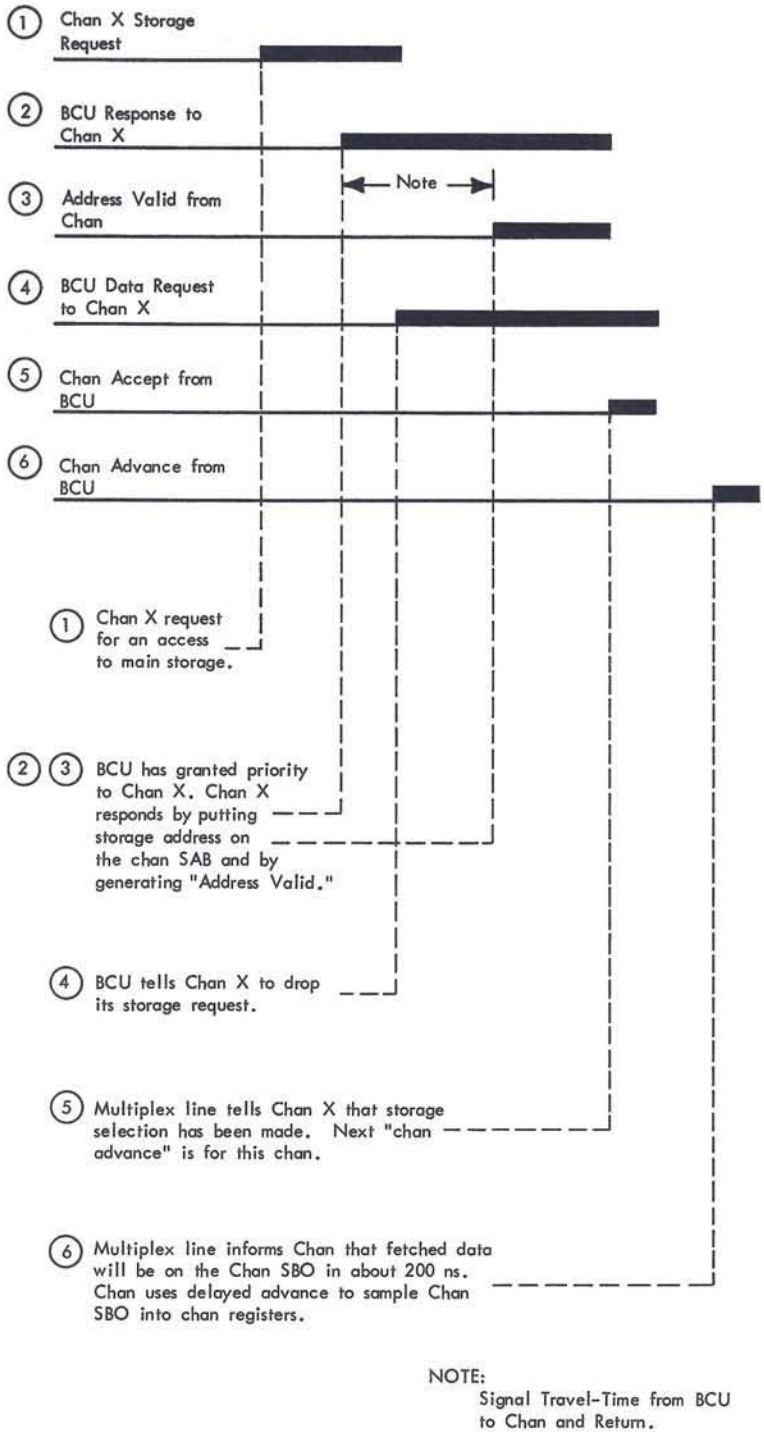
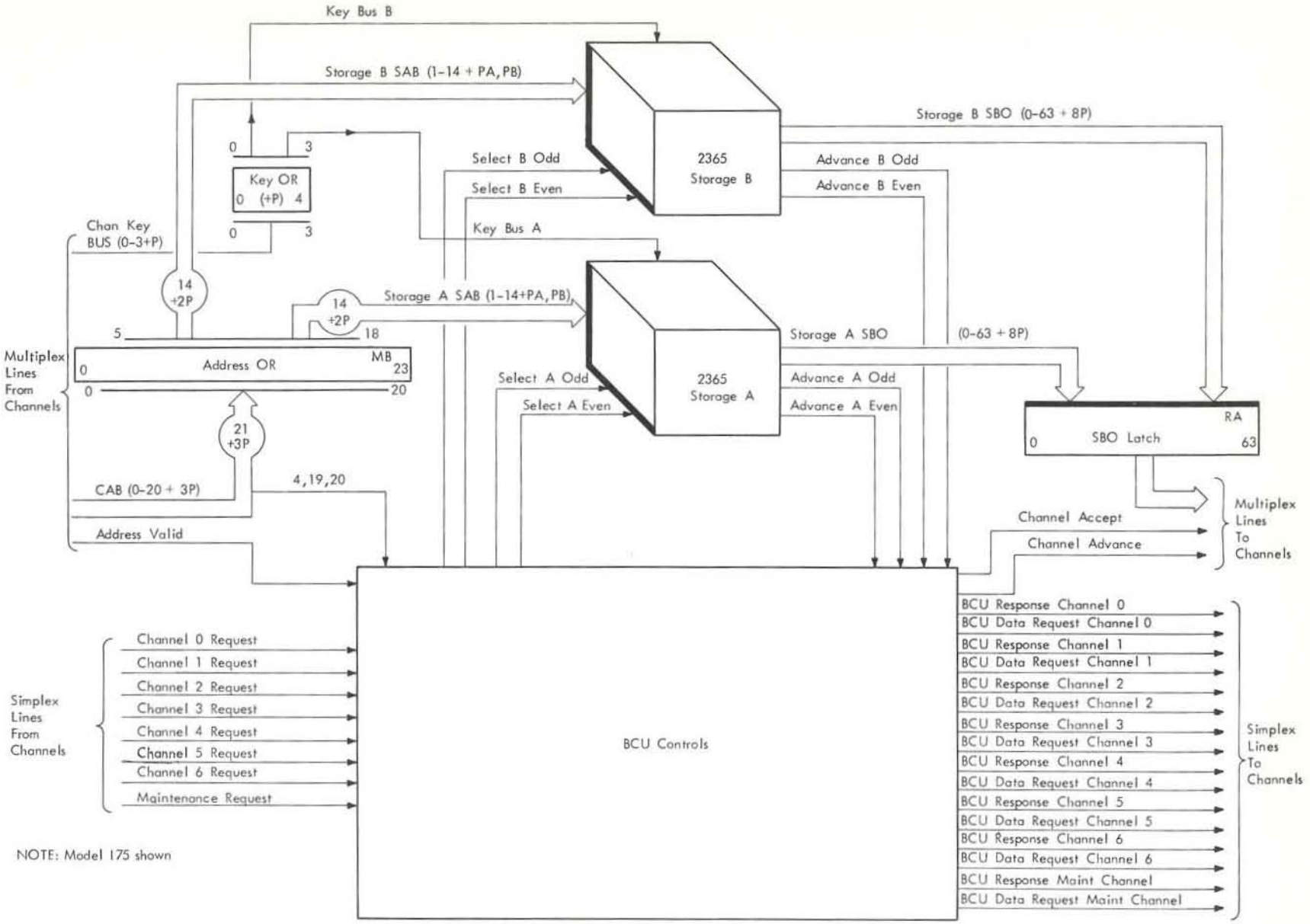


FIGURE 21. CHANNEL FETCH SIGNAL EXCHANGE



NOTE: Model 175 shown

FIGURE 22. DATA FLOW, CHANNEL FETCH

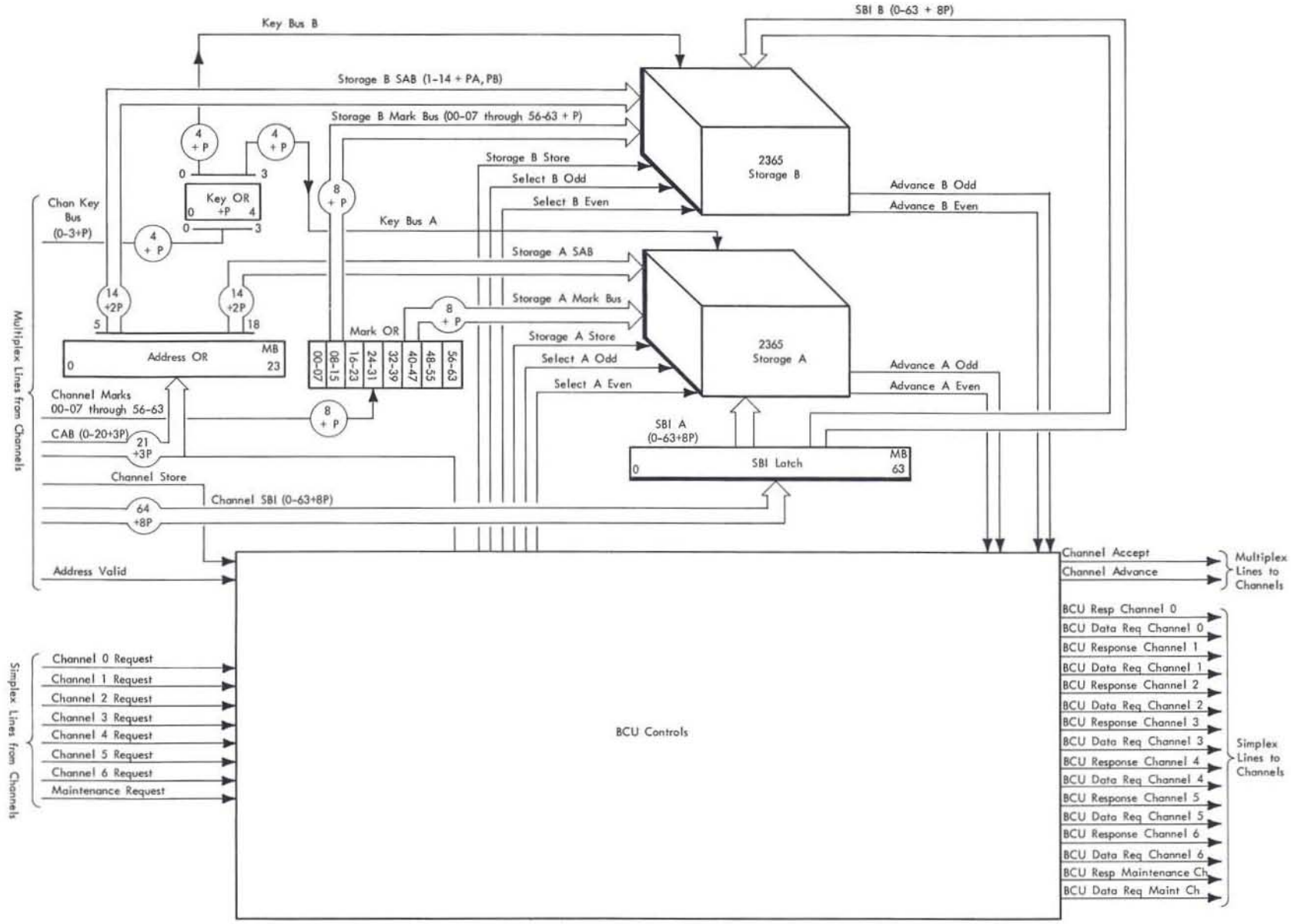
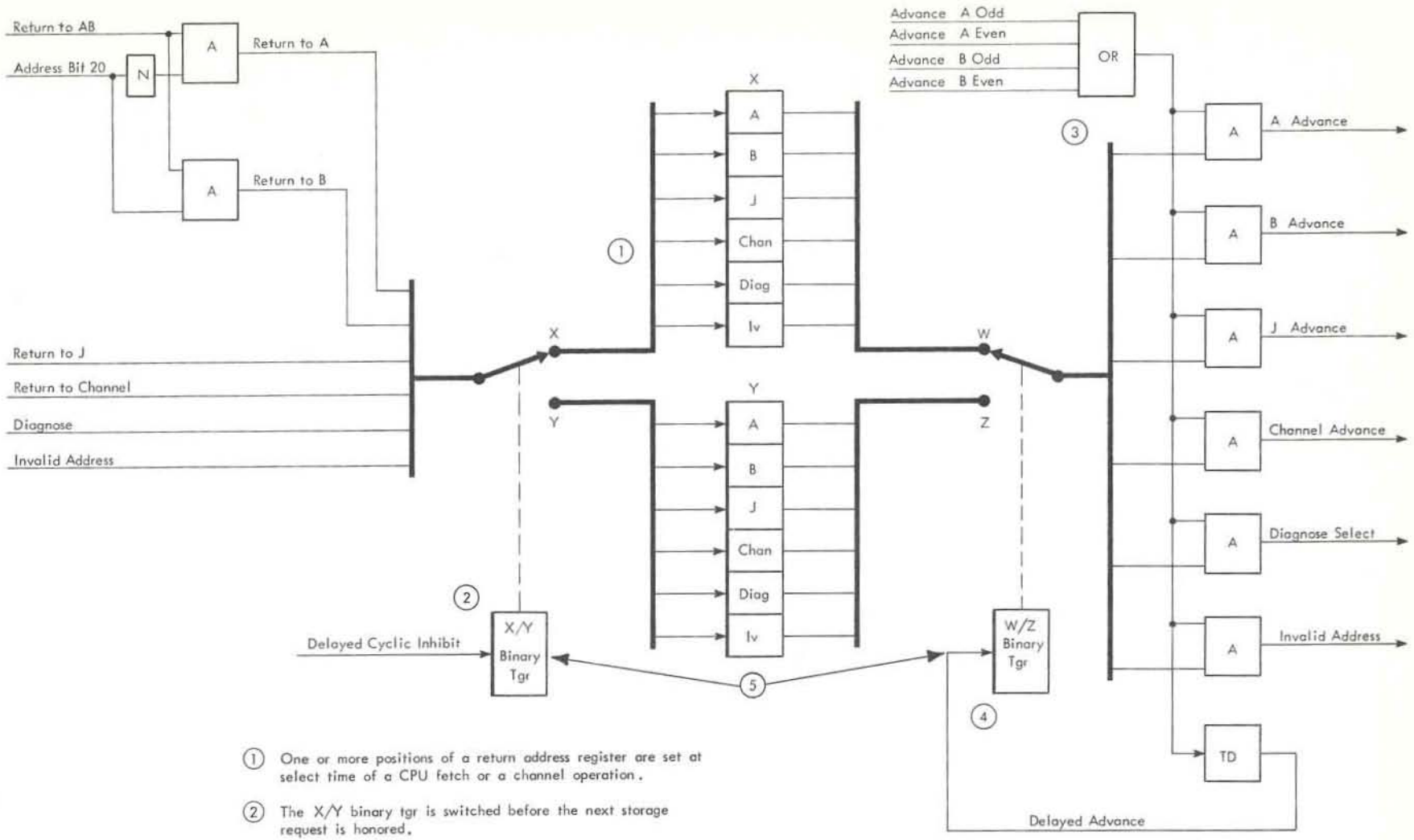


FIGURE 23. DATA FLOW, CHANNEL STORE



(1) One or more positions of a return address register are set at select time of a CPU fetch or a channel operation.

(2) The X/Y binary tgr is switched before the next storage request is honored.

(3) At advance time, the W/Z binary tgr gates-out the return address register set at select time.

(4) After the contents of a return address register have been used, the W/Z binary tgr is switched in preparation for the next advance.

(5) The X/Y and W/Z tgrs are switched during every storage cycle (stores as well as fetches).

FIGURE 24. RETURN ADDRESS

Whenever CPU makes a fetch request (I fetch request or E fetch request), one of two control signals is sent to the BCU to tell BCU where to return the fetch data. The two control signals are: return to J and return to A-B.

The BCU must remember the control signal associated with each request so that it can return fetched data to the correct destination. Because the BCU overlaps the operation of storage units, two registers are required to store return addresses. A fetch to A-odd storage, for example, may be initiated two machine cycles after a fetch to A-even storage. In this case, data from A even have not returned at the time the select to A odd is generated. Therefore, the return address for both fetches must be remembered. Note that because of the time relation of machine cycles to storage cycles, fetched data from the first fetch are back before the BCU can make a third consecutive fetch; therefore, two return address registers are sufficient (Figure 24).

Each of the two return address registers has six positions:

1. A
2. B
3. J
4. Channel
5. Diagnose
6. Invalid

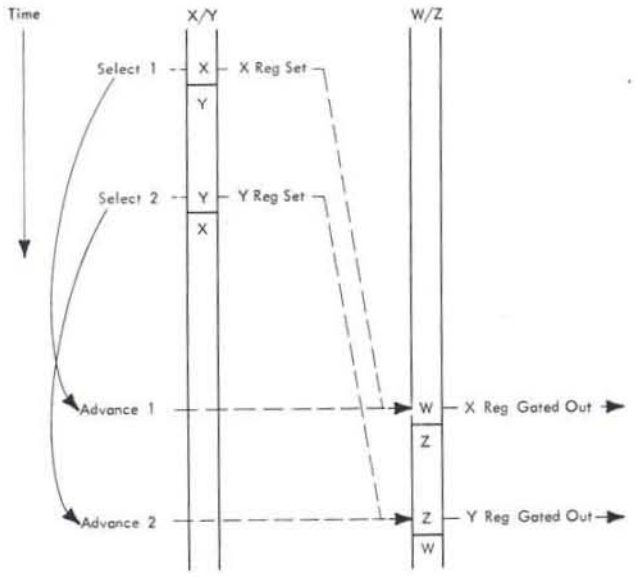
The return to A-B signal from the CPU is divided into return to A and return to B signals, using address bit 20. Data fetched from even addresses (not bit 20) go to the A register. Data fetched from odd addresses (bit 20) go to the B register.

A fourth position is set for channel operations. Unlike CPU, a return signal is sent to the channel on both store and fetch operations; therefore, a return signal from the channel is unnecessary. The channel select signal that is generated within the BCU sets the channel position of the return address registers.

Another position in the return address registers is for the diagnose instruction. This instruction sets the maintenance control word (MCW) register with the contents of a specified storage location. The diagnose instruction is unique because CPU makes the request but the fetched word is set into the MCW from the channel SBO. The diagnose position in a return address register sends a diagnose select signal to control circuits which perform this special gating.

The last position in the return address register is for an invalid address error. An invalid address is an address which is not within the main storage of this particular system. It is necessary to route the invalid address indication through the return address registers because the action taken for the invalid address depends on what the system was doing when it generated the erroneous address. For example, if instructions are located in the highest storage word with a branch back to a lower address, the I unit would generate an invalid address in attempting to fill an empty instruction buffer. In this case, the invalid address is not an error. The BCU must send the invalid address indication along with the A advance or B advance signal so that the CPU knows that this invalid address was generated for an IC fetch and, therefore, may not be an error condition.

Gating into and out of the return address registers is controlled by two binary triggers (Figure 24). The X/Y trigger gates the input, and the W/Z trigger gates the output. The W condition of the W/Z trigger always gates out the X return address register. The X/Y trigger gates inputs to one register at select time and then switches to be ready for the next select (Figure 25). The W/Z trigger gates the output of one register at advance time, then switches to be ready for the next advance.



NOTE: X/Y and W/Y have no fixed relation to odd/even storages.

FIGURE 25. RETURN ADDRESS REGISTER GATING

The advance signal gates fetched data into the proper register(s). Outputs from SBO latch go to the A register, the B register, the J register, and the channel SBO; but the data is set only into that register (or those registers) that receive an advance pulse. An example of two registers that receive an advance pulse, and therefore, the same data, is a manual load A-B. This operation causes BCU to fetch the 64-bit word (plus eight-parity bits) stored in the system control panel data keys (panel keys) and deliver this word to both the A and B registers.

THEORY OF OPERATION

- The major operations performed by the BCU are: CPU fetch, CPU store, channel fetch, and channel store.

- In addition to these, the BCU does special operations to fetch or store a storage protect key, load the maintenance control word register (MCW), handle system console manual operations, and handle errors detected by the BCU and main storage.

The major operations performed by the BCU are:

1. CPU fetch: BCU fetches the addressed double word from main storage and delivers the fetched data to the A, B, or J register.
2. CPU store: BCU delivers the double word in the K register and the contents of the mark register to main storage. Storage substitutes the K register data bytes indicated by the mark bits for corresponding bytes in the addressed location.
3. Channel fetch: BCU fetches the addressed double word and delivers the fetched data to the channel on the channel SBO.
4. Channel store: BCU delivers the double word on the channel SBI and the channel mark bits to main storage.

In addition to these four major operations, the BCU handles the storage access portion of the set and insert key instructions, the diagnose instruction, the test and set instruction, and system console manual operations. BCU also handles errors detected within the BCU and within main storage. Figure 9100 summarizes the operations performed by the BCU.

ADDRESS BIT FUNCTIONS

- Each HSS holds 16K double words.

- Address range within a HSS depends on the mode of interleaving.

- Fourteen address bits are used to address double words within a HSS: Models H75 and IH75 use bits 6-19; Models I75 and J75 use bits 5-18.

- SPF uses a portion of the HSS address bits.

Each HSS holds 16,384 (16K) double words or 131,072 (128K) bytes. The address range of a HSS, however, is 32K for two-way interleaving or 64K for four-way interleaving. In other words, the span of addresses from the lowest to the highest within a HSS depends on the mode of interleaving (Figure 26).

Fourteen bits are required to address the 16K double words within a HSS. Any group of 14 bits has 16K combinations; the particular address bits used, however, depend on the span of addresses within that HSS:

Address Bits	Number of Bits	Double Word Address Span
7-20	14	16K
6-20	15	32K
5-20	16	64K
4-20	17	128K

In the Model H75, address bits 6-19 constitute a HSS address; bit 20 is used to select one of the two HSS (Figure 27). In the Model IH75, address bits 6-19 constitute a HSS address; bit 20 is used to select odd or even HSS. Bits 4 and 5 are used to select A, B or C Memory Group (Figure 28A). For Models I75 and J75, address bits 5-18 constitute a HSS address; bits 19 and 20 select one of four HSS (Figure 28). In the Model J75, address bit 4 selects the lower or upper group of HSS (Figure 29).

A portion of the 14-bit address sent to a HSS is relayed to an SPF unit. Address bit 12 is the low-order SPF address bit; this bit changes once every 2048 consecutive byte addresses (256 double word addresses). In Models H75 and IH75, the single SPF unit stores 128 storage protection words (Figure 30). These 128 words, each protecting a block of 256 (double word) addresses, span the 32K storage words within the single 2365. Address bits 6-12 are used to address the 128 protection words in Models H75 and IH75.

In Models I75 and J75, each SPF unit stores 256 storage protection words because the span of addresses within each 2365 is twice (64K) that of Model H75 and Model IH75 (32K). Address bits 5-12 are required to address the 256 storage protection words (Figure 31).

In addition to locations within main storage, the BCU can fetch the system control panel data keys (panel keys). The panel keys are fetched for certain manual operations, for certain errors, and when the programmer specifies the panel key address. The panel key address is specified by a 1 bit in address

position 0 if the enable panel key address switch on the control panel is on.

Figures 9101, 9102, and 9103 summarize the address bit functions for Models H75, I75, IH75, and J75.

Address Switching

- Address bits can be switched to allow a diagnostic program to run with a failing HSS.
- Switching can be done manually or by setting bit 14 or bit 15 in the MCW.
- On Model H75 and Model IH75, address bit 6 is interchanged with address bit 20 or address bit 6 is inverted, then interchanged with bit 20.
- On Models I75 and J75, address bits 5 and 19 are used for bit switching instead of bits 6 and 20.
- On Model H75 and Model IH75, address switching defeats interleaving: on Models I75 and J75, address switching replaces four-way interleaving with two-way interleaving.

The Model 75 has a scheme for switching certain address bits as a diagnostic aid. The purpose of this scheme is to allow a diagnostic program to run when a HSS is failing. The program can then analyze the pattern of failing addresses to further localize the failing circuits within the defective HSS. The bits that are interchanged and the resulting effect on the physical location of addresses, depends on the system storage configuration; note, however, that the purpose of address switching is to arrange the addresses so that a sizeable program can be loaded in such a way as to avoid a failing HSS.

Address switching can be done manually from the system control panel or by changing bit 14 or bit 15 in the MCW with a diagnose instruction. The control panel switch has three positions:

1. Up - interchange storage address bits
2. Center - normal
3. Down - interchange and invert storage address bits

MCW bit 14 duplicates the up position of the switch; MCW bit 15 duplicates the down position of the switch. A priority scheme exists so that MCW bits have priority over the switch setting. The two MCW bits are interlocked so that if both are on, neither is effective (Figure 32).

For Models H75 and IH75, the interchange storage address bits condition interchanges bits 6 and 20. Reversing these bits, defeats the interleaved address scheme so that consecutive addresses are in one HSS (Figure 33). With normal addressing, consecutive addresses are in alternate HSS. When bits 6 and 20 are interchanged, the first 16K of consecutive addresses are in the HSS that normally contains only even addresses. Note that consecutive addresses within a HSS jump by 8K as bit 20 is used in place of bit 6. When, after 16K addresses, bit 6 goes to a 1, the HSS which normally contained odd addresses is selected because bit 6 is being used by the BCU in place of bit 20. The first 16K addresses are in the original even HSS; the second 16K addresses are in the original odd HSS.

When bit 20 is exchanged with the inverted condition of bit 6 (interchange and invert storage address bits, Model H75 and Model IH75), the effect is to place the first 16K addresses in the original odd HSS and the second 16K addresses in the original even HSS. In the Model H75 and Model IH75, a diagnostic program of 16K, or less, can be loaded into either of the two HSS to analyze the remaining HSS.

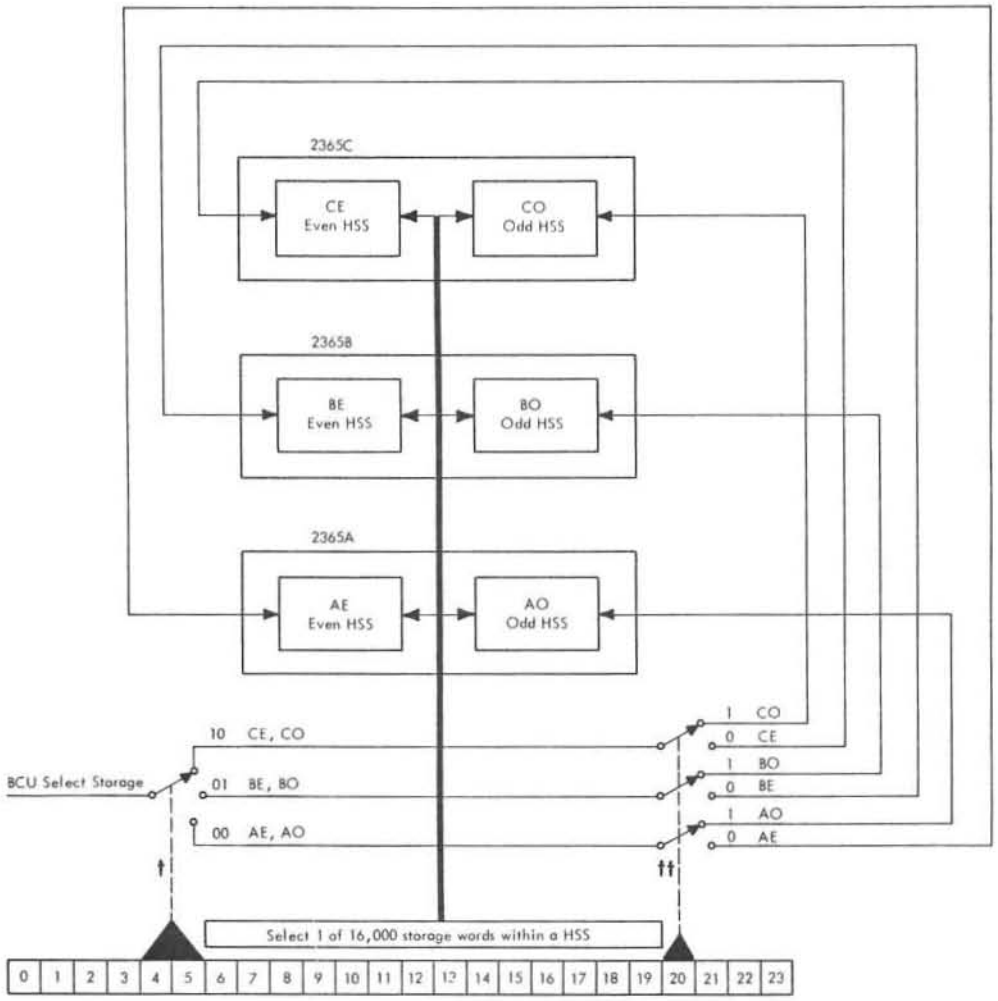
For Models I75 and J75, the interchange storage address bits condition interchanges bit 5 and 19. Reversing these bits, changes the addressing scheme from four-way interleaving to two-way interleaving (Figure 34). Every third consecutive address jumps by 8K compared to the normal addressing scheme; however, consecutive addresses stay within a pair of HSS (one 2365) until the 32K available locations are exhausted. With bits 5 and 19 interchanged, 2365 A contains the first 32K addresses, 2365 B contains the second 32K of addresses. In Model J75, 2365 C contains the third 32K of addresses and 2365 D contains the fourth 32K of addresses.

When bit 19 is interchanged with the inverted condition of bit 5 (interchange and invert storage address bits), the effect is to place the first 32K addresses in 2365 B and the second 32K addresses in 2365 A. In Model J75, 2365 D contains the third 32K addresses and 2365 C contains the fourth 32K addresses.

CPU FETCH

- I or E unit can initiate a CPU fetch.
- BCU selects storage to get requested storage word.
- When the fetched word returns from storage, BCU routes the word into the A, B, or J register.

A CPU fetch is a storage fetch requested by either the I unit or the E unit. The BCU starts storage to get the 64-bit plus eight-parity bit word requested, and then delivers this word to the A, B, or J register.



LEGEND:

† Used for additional optional storages

†† Byte address, not used by BCU or storage

●FIGURE 28A. HSS SELECTION, MODEL IH75

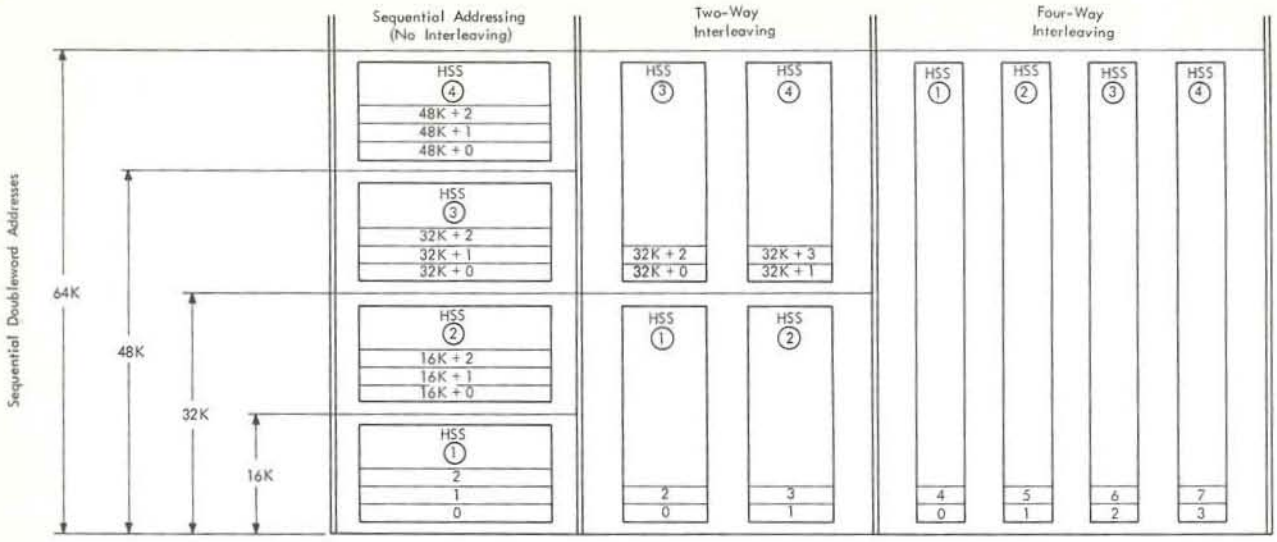
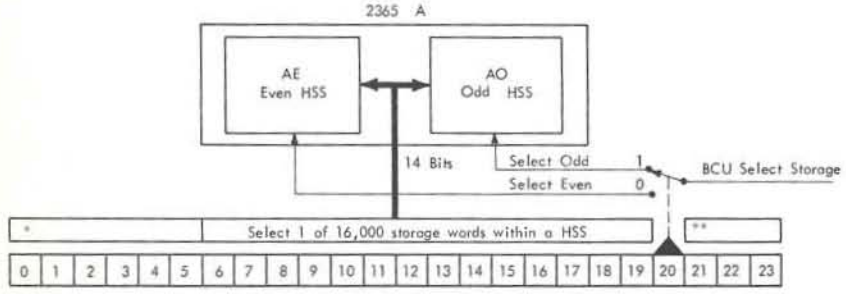
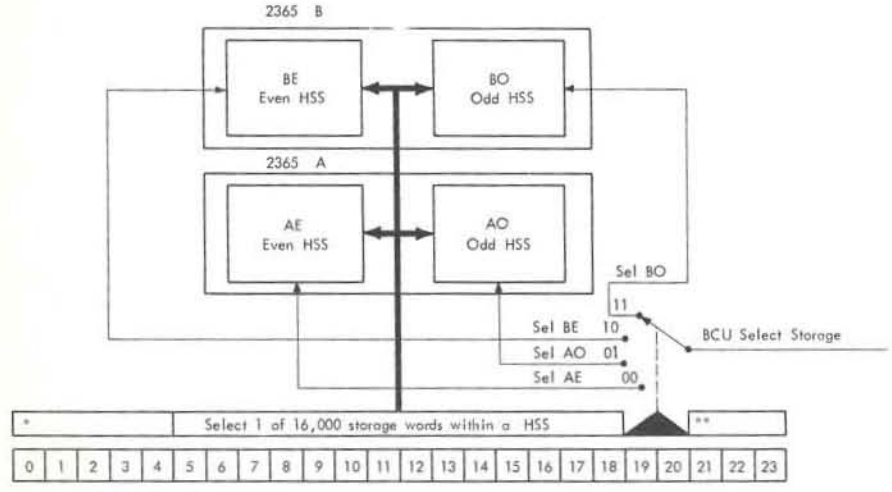


FIGURE 26. ADDRESS SPAN VERSUS INTERLEAVING MODE



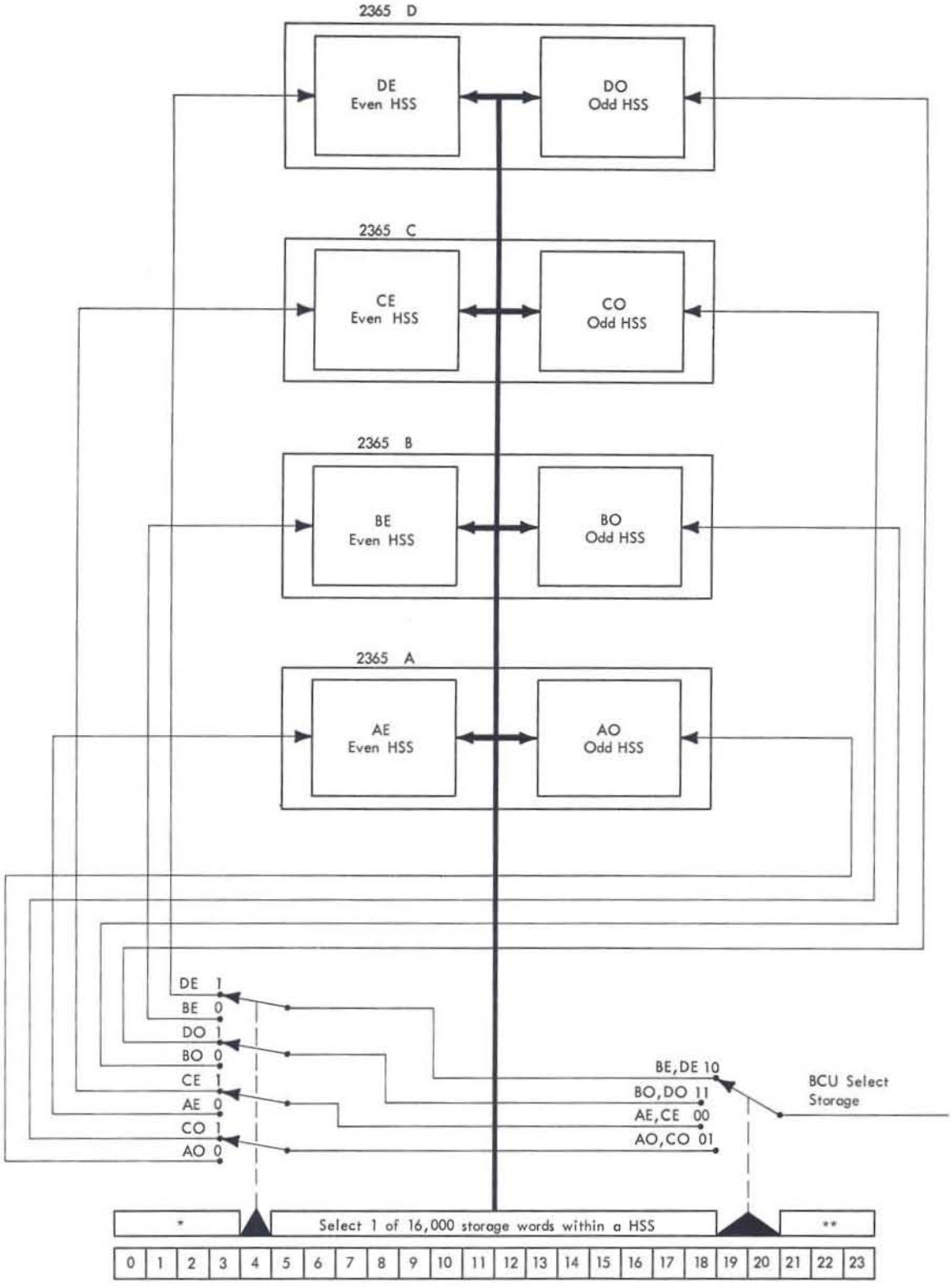
* Used for additional optional storages.
** Byte address. Not used by BCU or storage.

FIGURE 27. HSS SELECTION, MODEL H75



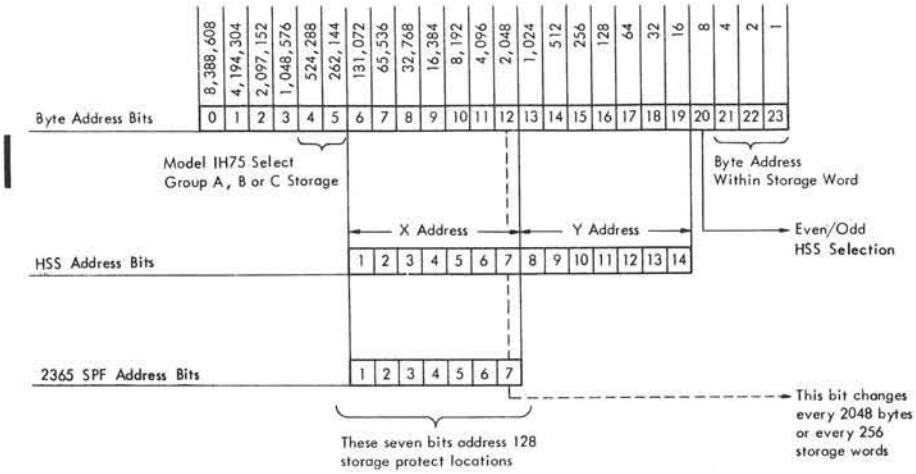
* Used for additional optional storages.
** Byte address. Not used by BCU or storages.

FIGURE 28. HSS SELECTION, MODEL I75



* Used for additional optional storages.
** Byte address. Not used by BCU or storage.

FIGURE 29. HSS SELECTION, MODEL J75



● FIGURE 30. SPF STORAGE BLOCK ADDRESSING, MODEL H75 AND MODEL IH75

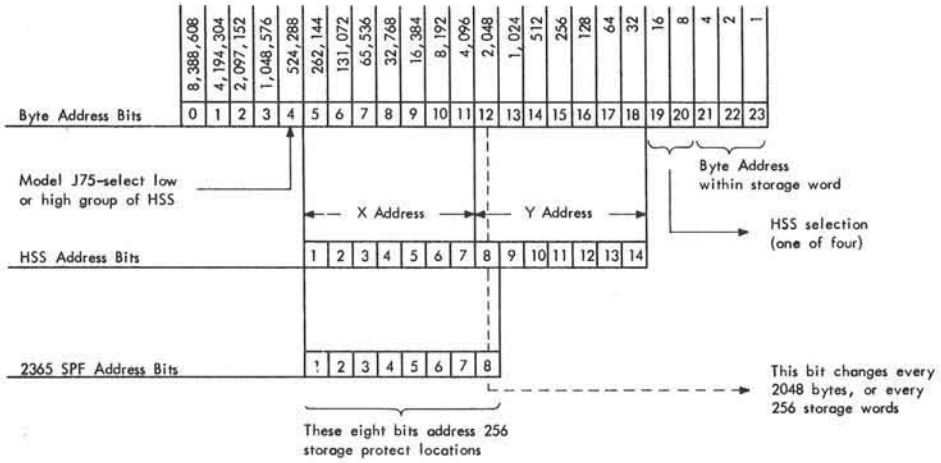


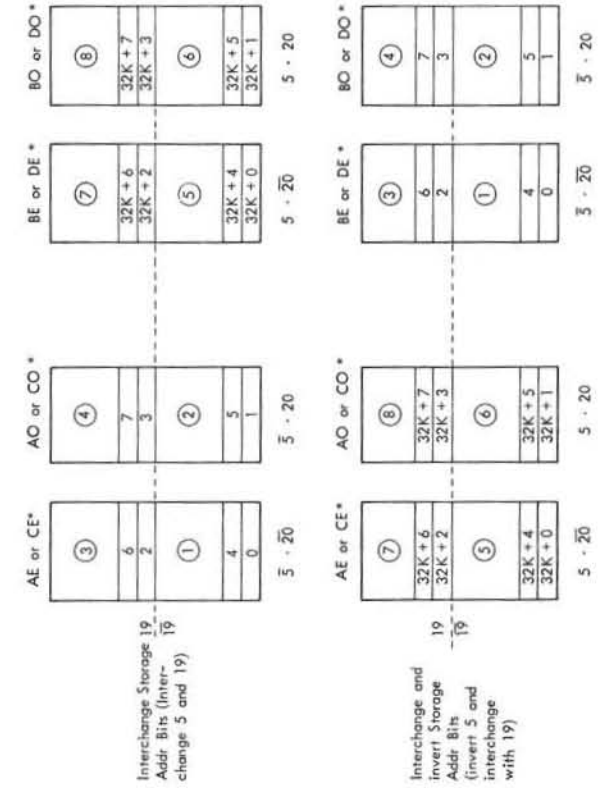
FIGURE 31. SPF STORAGE BLOCK ADDRESSING, MODELS I75 AND J75

MCW Bits		Switch Setting	Result
14	15		
0	0	Center	Normal addressing
1	1	Any	
0	0	Up	Interchange storage address bits
1	0	Any	
0	0	Down	Interchange and invert storage address bits
0	1	Any	

FIGURE 32. ADDRESS BIT SWITCHING

* Address Switching, Model IH75.
Addresses are 32K higher within C and are 64K higher within D than shown.

FIGURE 33. ADDRESS SWITCHING, MODEL H75



* Addresses within C and D HSS are 64K higher than the addresses shown.

FIGURE 34. ADDRESS SWITCHING, MODELS I75 AND J75

Data Flow

- BCU gets I fetch request or E fetch request.
- Address is in SAR (or is being set into SA[D]) at the time of request.
- BCU generates select to the proper storage unit.
- Selected HSS sends advance to forewarn BCU of returning data.
- BCU sets returning storage word into SBO latch.
- BCU sends A, B, or J advance to the CPU.

The CPU requests a fetch operation by sending I fetch request or E fetch request to the BCU (Figure 6100). Along with the request, the CPU sets the desired address into SAR and into duplicate SAR. SAR and the storage protection key from the PSW are gated through the address OR and the key OR unless a channel storage select is to be made on this machine cycle.

The requested CPU selection is made if there is no pending channel select blocking the CPU selects, the BCU is not busy (cyclic inhibit off), and the requested HSS is not busy. The BCU decodes address bits set into duplicate SAR to generate a select to the proper HSS. Note that all HSS receive the storage address and the protection key; only the HSS that receives select, however, sets the storage address into its MAR and uses the protection key.

When select is sent to storage, the busy trigger for the selected HSS is set to block further selections of this HSS until it ends its cycle. The A, B, or J position in the return address register that is pointed to by the X/Y binary trigger is set. The position set is determined by a return to line from the CPU and, in the case of A-B, address bit 20.

Shortly after select, an accept signal is sent back to the CPU. This signal tells the CPU that its request has been honored and that it can now drop its request and change SAR. It can make a new request if one is pending.

About two cycles after select, the selected HSS sends an advance signal to the BCU. Half-a-cycle after advance, the selected HSS has the fetched 72-bit word gated from its memory data register (MDR) to the SBO latch register. Shortly thereafter, the BCU releases the SBO latch to temporarily store the fetched word. After setting the SBO latch, the BCU sends an A, B, or J advance signal to the CPU. The CPU uses the advance as a gate to set the receiving register with the next A clock.

Control

- CPU interlocks all I and E storage request lines.
- BCU is busy with each selection for two machine cycles.
- A CPU request is blocked if BCU is busy, a channel selection is to be made on this cycle, a CDA signal is present from a channel, or if the requested HSS is busy.
- When BCU sends a select to HSS, accept is sent back to the CPU.
- BCU sets a return address register and switches the X/Y binary trigger.
- Advance from the selected HSS samples the return address register.
- Advance also gates the setting of the SBO latch.

The CPU interlocks all fetch and store request lines so that only one request can be made at a time. One of the two CPU return to lines always accompanies a CPU fetch request and the SAR and duplicate SAR either contain the desired storage address or these registers will be set at approximately the same time that the request signal is brought up.

The BCU checks for a CPU request each L time. The request is blocked, however, if the BCU is busy with another storage request (cyclic inhibit is on), if a channel is ready to select a nonbusy HSS, or if the HSS requested by the CPU is busy. If a CPU request is blocked, it is completely ignored until the next L time, when another attempt will be made. Meanwhile, the CPU is free to drop its request or substitute another request. For example, the I unit may substitute an operand fetch request for an unaccepted IC fetch request because the operand fetch has a higher priority. An unaccepted request is a request for which BCU has not generated an accept pulse.

From the time that BCU recognizes a CPU request, BCU is busy making the storage selection for two machine cycles (Figure 35). The approximate timing of the major control signals is shown in Figure 35. The generation of each control signal is shown in Figure 5200 and is described in the following text.

All CPU storage request lines are ORed, then combined with an L clock . not accept, and not single cycle to produce a fast request signal. This signal is routed to one set of HSS set select latch ANDs. In addition to bringing up fast request, the CPU fetch request lines are routed to set the CPU request latch if it is not being blocked by accept.

The output of the CPU request latch is routed to a second set of set select latch ANDs. The first set of set select latch ANDs ensures that a select latch is set as early as possible when a selection is made (minimum circuit delay). The second set of set select latch ANDs ensures a set pulse of sufficient duration to complete the latch-back path through the select latches.

A CPU set select signal, timed by an A clock, is a second condition necessary for setting a select latch. This signal is blocked if BCU is busy (cyclic inhibit on), a channel select is to be made on this cycle, or if a chain data address (CDA) signal is present from one of the channels. A channel brings up the CDA signal under certain conditions which require a rapid storage access. By blocking all CPU selects while CDA is on, the channel will probably be serviced more rapidly because neither the BCU nor the requested HSS will be busy when the channel request arrives.

The final condition that can block the setting of a select latch is a busy trigger that is on for the requested HSS. Duplicate SAR bits are decoded in the set select latch ANDs to pick a single HSS select latch with any one combination of bits in the duplicate SAR. If the busy trigger for this requested HSS is on, the setting of the select latch is blocked. If the busy trigger is off, the requested HSS select latch is set and the busy trigger for the selected HSS is turned on.

When any select latch is set, the positive select out latch is turned on to indicate that a storage selection was made.

The positive select out signal and an EB clock turn on the accept and pulse accept triggers. These two triggers are set in parallel, and under normal conditions, are also simultaneously reset. The pulse accept trigger is reset by a running clock, and the accept trigger is reset by a controlled clock. The two triggers are necessary for single-cycle operation to maintain the accept signal to the CPU for one controlled clock cycle even though the storage cycle has been completed. The accept and pulse accept triggers distinguish CPU from channel storage selects; they are not set when a channel selection is made.

The positive select out signal and a B clock set the cyclic inhibit latch to inhibit all storage selections on the following machine cycle.

The positive select out signal also controls the setting of a position in a return address register (Figure 5080). The A, B, or J position is set into the return address register that is pointed to by the X/Y binary trigger with a B clock during the selection cycle.

The positive select out signal is delayed to switch the X/Y binary trigger after a register position has been set.

After two machine cycles, the BCU is finished with the selection and can service any other request that is pending. Handling of the returning data is completely independent of the selection circuits. The only information that pertains to the selection just completed that is retained by the BCU is the bit set into a return address register. This information will be used when data returns from the selected HSS. The timing of events for returning fetched data and completing the fetch operation depends entirely on the selected HSS. The BCU simply waits for advance from storage and the number of machine cycles between select and advance is of no consequence to the BCU.

The storage advance pulse tells the BCU that a storage cycle is nearing completion. The BCU uses advance to sample the return address register that is pointed to by the W/Z trigger. For a CPU fetch operation, the sampling yields the appropriate advance signal to the CPU (A, B, or J). The A, B, or J advance signal, however, is delayed to bracket the A clock which sets the J register or the LBR clock which sets the A or B register.

The BCU also uses advance to gate the setting of the SBO latch: storage times advance to precede fetch data by about 100 nanoseconds.

The final two steps of a CPU fetch operation are:

1. Switching the X/Y trigger, and
2. Resetting the selected HSS busy trigger.

The BCU delays the storage advance to switch the W/Z trigger after advance has sampled the return address register.

The busy trigger is reset by a reset busy signal from the selected HSS nearly one machine cycle before the storage times-out. This timing allows the BCU to reselect this HSS at the same time that it is ending its previous cycle.

Detailed timings of consecutive CPU fetches to AO HSS are shown in Figure 8100.

CPU STORE

- CPU store is initiated by the I unit or the E unit.
- BCU selects storage and delivers the double word in K to storage.
- Only K bytes masked by mark bits are stored. Other bytes are regenerated.
- Protection key in PSW much match key in SPF storage.

A CPU store is a store-to-main storage requested by the I unit or the E unit. The BCU starts storage and sends the 64-bit plus eight-parity bit word in the K register to storage. Only those bytes of the K register that have a corresponding bit in the mark register are stored; bytes represented by a zero-mark bit are regenerated by storage.

The BCU sends the CPU storage protection key (PSW bits 8-11) to the SPF where this key is compared to the key for this address previously stored in SPF. If the keys match, the store operation proceeds. If the keys do not match, the SPF cancels the store operation (all bytes are regenerated) and signals a storage address protect (SAP) error.

Data Flow

- BCU routes SAR, mark register, and PSW key to storage.
- BCU sends select and store to storage.
- BCU sets K into SBI latch register and sends SBI latch register to storage.

The CPU requests a store operation by sending I store request or E store request to the BCU (Figure 6101). Along with the request, the CPU sets the desired address into SAR and the duplicate SAR and sets the mark register. The mark register is set in one of two ways:

1. Single bits set by VFL circuits.
2. Twin bits corresponding to halfword boundaries for storing halfwords, words, and double words.

The BCU generates a select pulse to start the correct HSS. The address (14 bits plus two-parity bits) is gated from SAR through the address OR to all HSS. The eight-mark bits (plus a parity bit) are gated from the mark register through the mark OR to the storage. The CPU storage protect key (PSW 8-11) is gated through the key OR and a parity bit is added. Along with select, the BCU sends store to all 2365 storage frames.

Just as on a CPU fetch operation, BCU sends accept back to the CPU about the time select is sent to storage. The accept signal tells the CPU that its request has been honored and that it can now drop its request and change SAR. It can make a new request if one is pending.

On the post-selection cycle, the SBI latch register is set from K. All 72-bits are set into the SBI latch and sent to storage regardless of the number of bytes to be stored. The selected HSS uses the mark bits to gate the corresponding bytes into its MDR. Those bytes of the MDR not set from SBI are set by the sense amplifiers at the end of the read portion of the storage read/write cycle. Then, the modified 72-bit word in the MDR is written back into the selected address on the write cycle.

After the post-selection cycle, the BCU is finished with the CPU store operation, except for the housekeeping jobs of turning off the storage busy trigger set by select and changing the W/Z trigger to keep it in step with the X/Y trigger. Just as on a CPU fetch operation, these jobs are done near the end of the storage cycle after the selected HSS sends advance.

Control

- CPU request latch and blocking conditions operate the same as for a CPU fetch.
- BCU sends select and store to storage along with SAR, marks, and key.
- No return address register bits are set.
- X/Y, W/Z, and HSS busy triggers operate the same as on a CPU fetch.

The CPU interlocks all fetch and store request lines so that only one request can be made at a time. The CPU may have loaded SAR, duplicate SAR, and the mark register prior to raising store request; or it may set these registers at approximately the same time that store request is brought up. In either event, BCU will have this input data at the L clock when the CPU request latch is set (Figure 5200).

The same blocking conditions described for a CPU fetch operation may block a CPU store operation. These blocks are:

1. A channel ready to select a nonbusy HSS on this cycle or a CDA line from a channel.
2. Cyclic inhibit on (BCU is busy).
3. Requested HSS is busy.

If the request is blocked, it is completely ignored until the next L time, when another attempt will be made if the request is still present. Meanwhile, the CPU is free to drop its request or substitute another request.

If the store request is not blocked, BCU decodes duplicate SAR bits and sends a select pulse to the requested HSS. The BCU gates the address from SAR, the mark bits from the mark register, and the SPF key from the PSW to the storage. The BCU also sends store to all 2365 storage frames. Also, during the selection cycle, BCU sends accept back to the CPU, turns on cyclic inhibit, and sets the appropriate HSS busy trigger.

A return address register is reset and no positions in the register are turned back on even if the specified store address is invalid (Figure 5080). A separate

error trigger is set for invalid CPU store address; details of all invalid address conditions are described under "Program Checks."

At the beginning of the post-selection cycle, the X/Y binary trigger is changed, even though no return address register bits are set for a CPU store operation. The SBI latch register is also released (reset and set) in the post-selection cycle. The CPU can (and often does) wait until A time of the post-selection cycle to set the K register; however, K can be set at any time prior to the post-selection cycle. The output of the SBI latch is ungated to storage; the selected HSS sets its MDR from the SBI about one-third of the way through its cycle.

The selected HSS sends advance back to the BCU, even though no data returns on a store operation (Figure 5080). Advance is delayed to switch the W/Z trigger to keep it in synchronization with the X/Y trigger.

The selected HSS sends reset busy to the BCU to reset the busy trigger turned on by select in time to allow a new select shortly after the end of the storage cycle (Figure 5200). A HSS receiving consecutive selects is idle about 30 nanoseconds (difference between four machine cycles and a 750-nanosecond storage cycle).

Detailed timings of consecutive CPU store operations to alternate storage addresses are shown in Figure 8101.

CHANNEL FETCH

- Channel bus priority circuits allocate the use of the channel buses to one channel at a time.
- BCU gives channel storage requests priority over CPU storage requests.
- Unlike CPU requests, a channel request remains pending if the requested HSS is busy.

The channel bus priority circuits grant priority to one channel (or the system control panel) at a time. The channel bus priority circuits are necessary to prevent interference on the channel buses (CAB, channel SBI, and channel SBO). Because the seven channels and the system control panel operate independently of each other and of the CPU, any number of these eight storage users may simultaneously request storage. When there are simultaneous requests, the BCU must allocate the use of the channel buses to one user at a time. The BCU allocates the use of the channel buses in a fixed priority scheme: channel 0 has the highest priority, channel 1 next-to-highest, and so forth. For example, channel 6 cannot access storage if any other channel is making a request. The system control panel, or maintenance channel, has the lowest priority and can access storage only when none of the channels are making a request.

Once a channel gains channel bus priority, it puts the storage address on the channel SAB (CAB). The entire process of recognizing a channel request, granting priority to the requesting channel, and receiving the storage address on CAB requires about 1 microsecond. During this time, the BCU will honor CPU requests even though channels have priority over the CPU.

A channel is not considered to have a valid request until its storage address arrives in BCU and a line, address valid, is generated. At this point, a channel select signal is generated at the first B time when the BCU and the HSS requested by the CAB bits are not busy. Any channel select blocks all CPU selects. By bringing up channel select at B time, a CPU request that would have been honored on the next cycle is blocked and the channel selection is made instead.

The turn-off of the HSS busy triggers is timed to allow a channel select signal to be generated at the B time of the cycle that precedes the machine cycle in which a new select can be sent to a HSS ending a previous cycle.

Data Flow

- Channel requests are asynchronous with each other and with the CPU and the BCU.
- BCU gives priority to a channel by sending that channel a BCU response.
- A channel puts its storage address on the channel SAB (CAB) when it gets priority.
- When BCU has a valid address and the BCU and the requested HSS are not busy, CPU requests are blocked and storage is selected for the channel.
- Advance from storage is sent directly to the channel.
- With advance, the BCU sets fetched data into the SBO latch.
- The channel uses advance to sample a fetched word from channel SBO into the channel registers.

The channels bring up storage request anytime they require a storage access. (Channel includes the system control panel which is considered the maintenance channel.) Unlike CPU requests, channel requests are completely random with respect to each other and with respect to the BCU and CPU cycles (Figure 6102).

Whenever the channel buses are not in use (as indicated by the buffer trigger being off), the BCU checks for channel requests once per machine cycle. The checking is done from channel 0 through channel 6 and then the maintenance channel in that order. The first request that is found (highest priority) is honored, and any other requests are ignored.

BCU honors a channel request by sending that channel a BCU response signal. This signal tells the channel that it gained channel bus priority and should respond by putting the address of the desired storage location on the channel SAB (CAB) and the protection key on the channel key bus. The CAB positions, 0-20 plus three parity bits, feed into corresponding positions of the BCU address OR.

About 300 nanoseconds travel time is required for BCU response to go out to a channel. Another 300 nanoseconds is required for the storage address to come back to the BCU. Because of this signal travel time, the channel sends an address valid signal to tell the BCU controls when the address has arrived in the BCU. Meanwhile, the BCU follows the BCU response signal with a BCU data request signal. The primary purpose of BCU data request is to tell the channel to put the incoming data on the channel storage bus in (SBI) if this is to be a store operation. At request time, however, BCU does not know if a channel is requesting a store or a fetch; therefore, the data request signal is always sent, but the channel will not put data on the channel SBI for fetch operations. The channel does use BCU data request to drop its storage request to the BCU.

Once the BCU receives address valid, it examines CAB bits to start the proper HSS in the same way duplicate SAR bits are examined to select a storage unit for CPU requests. Channel selects are decoded at B time of a machine cycle to block an L time CPU request (if present). Successful decoding of a channel select depends on the BCU and the requested HSS being not busy.

When the BCU decodes a channel select (B time), the address OR (and the key OR) is switched to gate 14 bits (plus two-parity bits) from the CAB to storage. After sending select to storage, the BCU sends accept to the channel. Accept tells the channel that its request has been honored and that the data it requested will be on the channel storage bus out (channel SBO) following the next channel advance pulse. Just prior to select, the BCU resets the channel bus priority circuits and turns off BCU response. The fall of BCU response tells the channel to take its storage address off the channel SAB.

When the selected HSS sends advance, the BCU routes this signal to the channel. The channel delays advance to gate the requested data from the channel SBO into its registers.

Control

- BCU samples for channel requests once per machine cycle until a request is found.
- A channel request turns on the buffer trigger which prevents further checking for requests until this channel is serviced.
- BCU sends BCU response and BCU data request to the requesting channel.
- The channel sends address valid when address on CAB is good.
- The channel waits if BCU or requested storage is busy.
- The channel bus priority circuits are released (reset) just prior to select.
- The maximum channel selection rate is one select per five machine cycles (about 1 microsecond).

A channel fetch operation begins when the BCU recognizes a channel request signal (Figure 5200). Note that the buffer trigger must be off in order for BCU to examine the channel request latches. The buffer trigger means that some channel has its address on the CAB (channel SAB). If more than one channel request latch is on, the BCU recognizes only the one which has the highest priority.

When the BCU recognizes a channel request, it immediately turns on the buffer trigger to prevent further checking of the request latches until this request is fulfilled. The buffer trigger output is combined with the request latches in the priority ANDs to send BCU response to the channel. The BCU response signal tells the channel to gate its storage address onto the CAB. BCU response remains on until the buffer trigger is turned off; the buffer trigger is turned off just prior to the generation of storage select for the channel. Because 600 nanoseconds travel time is required to remove the storage address from CAB, ample time is allowed for the HSS to set its address register (MAR) from CAB (through the address OR).

After a fixed time delay following BCU response, the BCU sends data request to the requesting channel. For a fetch operation, the channel uses data request only to turn off its storage request.

The BCU waits about 600 nanoseconds from BCU response to allow the storage address to arrive on the CAB. After this time delay, delay full comes on and the address valid line from the channel is tested. The address valid signal is timed to arrive at BCU slightly later than the slowest CAB bit; therefore, address valid tells BCU that all CAB bits are true (in their final state).

Once the address valid 2 trigger (Figure 5200) is on, the channel select ANDs are allowed to decode a channel select if the BCU is not busy (inhibit off). One of the channel select ANDs is activated when the HSS that is pointed to by the CAB selection bits is not busy. Unlike a CPU request, a channel request waits and remains pending if the BCU or the requested HSS is busy. As soon as neither is busy, channel select HSS is generated to block CPU requests. Another function of channel select HSS is to set the gating latch which switches the address OR and the key OR to gate the CAB and the channel keys to storage.

When select is sent to storage for a channel, the appropriate busy trigger, the positive select trigger, and the cyclic inhibit latch are turned on in the same way that these conditions are set for CPU storage selections.

At the LBR preceding select, the buffer trigger is reset to allow the channel bus priority circuits to examine for a new channel request. Resetting the buffer trigger also turns off BCU response to tell the channel to remove the storage address from CAB.

At the same time that the buffer trigger is reset, accept is sent to the channel. This signal tells the channel to use the next channel advance to gate the channel SBO into the channel. For 2365 storage units, accept is unnecessary; the fall of BCU response could have the same meaning. However, accept is necessary when LCS units are used on the system. In this case, it is sometimes necessary for BCU to drop BCU response to a channel without having honored its request. Therefore, the fall of BCU response, alone, cannot tell a channel that the next data on the channel SBO is the data that is requested. The operation of the BCU with LCS units attached to the system is in the "Features" section.

The action taken by BCU at select time appears, at first, to be too rapid. The channel priority circuits are released to look for a new channel request and, at the same time, BCU response is dropped. However, consider the travel time of signals to and from the channels. The requested address remains on CAB for about 600 nanoseconds after BCU response is dropped by the BCU. This gives plenty of time for the selected storage to set its MAR with the CAB address. In the same way, if the channel bus priority circuits immediately recognize a new request, it will require 600 nanoseconds for this new channel to fulfill the address valid condition within the BCU. Although the BCU can initiate a new BCU response almost immediately after dropping a prior BCU response, the signal travel time to and from the channels slows the over-all channel storage access rate to about 1 microsecond; that is, the maximum rate at which channels (one channel or a combination of channels) can access storage is one selection per five machine cycles (about 1 microsecond).

Like CPU operations, the terminating sequence of a channel operation is completely independent of the initiating sequence (Figure 5080). When the selected storage sends advance, this signal is routed to the channel by a return address register. The BCU also uses advance to gate the SBO latch and to switch the W/Z trigger. The channel delays the channel advance signal to gate the data from the channel SBO into the channel registers. The operation is completed when the selected HSS sends reset busy to the BCU to reset the busy trigger that is turned on at select time (Figure 5200).

Detailed timings of consecutive channel fetches to alternate storage addresses are shown in Figure 8102.

CHANNEL STORE

- BCU starts storage and stores the channel SBI double word.

A channel store is a storage store operation requested by any of the channels or the system console. The BCU takes the desired storage address from the CAB, starts storage for the channel, and delivers the store data (72 bits) on the channel SBI to the selected storage. The BCU also routes the eight-mark bits plus a parity bit and the four-bit (plus a parity bit) storage protection key from the channel to storage.

Data Flow

- Request, priority, and BCU response are the same as on a channel fetch operation.
- The channel puts store data on channel SBI when BCU sends BCU data request.
- BCU sets the SBI latch register from the channel SBI.

A channel starts a channel store operation with a request exactly the same as it starts a channel fetch operation (Figure 6103). The BCU grants priority and sends BCU response and data request without knowing whether the channel desires to fetch or store.

The channel, upon receiving BCU response, puts the storage address on CAB just as it does for a channel fetch. Along with the 21-bit plus three-parity bit storage address and the four-bit (plus a parity bit) protection key, the channel raises the store line to the BCU. Also, the channel puts the eight-mark bits (plus a parity bit) on the channel mark bus.

When the channel receives data request, it not only turns off its request, but also puts the 72-bit store data word on the channel SBI.

After BCU response and BCU data request, the BCU waits for address valid just as on a channel fetch operation. With address valid, the BCU generates a select when BCU and the requested storage

are free. Because the channel store line is up, the BCU also sends store to storage and gates the channel marks and the SPF key to storage along with the channel address bits.

On the post-selection cycle, the BCU sets the SBI latch register from the channel SBI. All 72 bits are sent to storage; the selected storage takes only the bytes which have corresponding mark bits just as on a CPU store operation.

After the post-selection cycle, the BCU is finished with the store operation except for the housekeeping jobs of turning off the storage busy trigger set by select and changing the W/Z trigger to keep it in synchronization with the X/Y trigger. These jobs are done near the end of a storage cycle after the selected storage sends advance.

Control

- Along with address from CAB, the BCU sends channel marks, channel key, and store to the selected storage.
- On post-selection cycle, BCU sets SBI latch register from channel SBI.
- Unlike a CPU store, a return address register is set for a channel store.
- Channel advance is sent to the channel even though no data is delivered.

The BCU control of a channel store operation is similar to that for a channel fetch operation (Figure 5200). The BCU recognizes a channel request through the channel bus priority circuits and responds with BCU response and BCU data request. When the channel receives BCU response, it puts the storage address on the CAB, puts the mark bits on the channel mark bus, puts the protection key on the key bus, and brings up the store line. When the channel receives BCU data request, it drops its request to the BCU and puts the 72-bit word to be stored on the channel SBI.

The BCU waits for address valid, then sets the address valid 1 trigger followed by the address valid 2 trigger. A further delay is necessary if either the BCU or the requested storage is busy. When both are free, the BCU uses CAB bits to generate a channel select which blocks CPU selects and sets the gating latch. The gating latch gates the channel address, marks, and key to all storages; only the selected HSS takes these inputs.

Just prior to select time, the channel bus priority circuits are released by resetting the request latches and the buffer trigger. Resetting the buffer trigger also turns off BCU response and, after about a 200-nanosecond delay, BCU data request is turned off.

During the selection cycle, BCU sends accept to the channel. The BCU also turns on cyclic inhibit and the appropriate storage busy trigger as it does during every selection cycle. The channel position of one of the return address registers is set and the X/Y binary trigger is switched.

During the post-selection cycle, the BCU releases the SBI latch register to set this register from the channel SBI.

After the post-selection cycle, the BCU is finished with the channel store operation until the selected storage sends advance (Figure 5080). When advance arrives, the BCU samples the return address register that is pointed to by the W/Z trigger. This sampling yields a channel advance signal which is sent to the channel. The channel uses this signal for housekeeping functions associated with an end-of-storage cycle. After sending channel advance, the BCU switches the W/Z trigger. The operation is completed when the selected HSS sends reset busy turn off the busy trigger which was set at select time.

Detailed timings of a channel store operation are shown in Figure 8103.

PANEL KEY FETCH

- BCU can fetch the control panel data keys instead of a storage location.
- A panel key fetch is made:
  1. When the panel keys are addressed, and
  2. On some errors

The BCU can fetch the system control panel data keys (panel keys) instead of a storage location. It does so when a storage user specifies the address of the panel keys instead of a storage location, and when an error condition prevents the return of data from the requested address. The panel keys are returned on error operations to put good parity in the receiving register. If the receiving register is loaded with all 0s, an erroneous machine check is generated because the receiving register has bad parity.

The panel key address is a 1 bit in address position 0. In addition to this address bit, the enable panel key address line must be active for the BCU to recognize address bit 0 as the panel key address. This line is brought up by the enable panel key address switch on the CE panel and for manual operations that use the panel key data.

Data Flow

- Panel keys are gated through the word switch matrix to the SBO latch register.

- SBO latch goes to normal destination.

When the BCU determines that a panel key fetch is to be made, it sends a panel key fetch signal to the maintenance channel (Figure 5201). This signal is delayed to gate the system control panel data keys into the SBO latch register. The delay is set to deliver the panel key data to the SBO latch at the time when data would have arrived from storage on a normal fetch. The SBO latch is released as a result of advance from a selected, but cancelled storage.

Good parity (8 bits) is always generated for the 64 data keys and the full 72-bit word is set into the SBO latch. The BCU sends an advance signal to the receiving register as if a storage fetch had been made.

In the case where BCU makes a panel key fetch because of an error, the receiving register gets a good parity word, and thus, an erroneous parity error is avoided. For example, an invalid address causes the BCU to make a panel key fetch. An invalid address is a programming error; if a parity error is allowed to occur, it would cause a machine check and would erroneously indicate a machine malfunction.

Control

- A HSS is selected, then cancelled.

- A cancelled HSS controls BCU.

Any of eight conditions cause the BCU to make a panel key fetch:

1. Enable panel key address switch on and any fetch request that has a 1 bit in address bit 0.
2. Load FLT control word.
3. Manual load AB.
4. Manual store to GP or FP register.
5. Manual set IC.
6. Manual set PSW.
7. Invalid address.
8. Address check detected in BCU.

In six of these operations, the panel key data is needed; the two error conditions cause a panel key fetch only as a means of getting a good parity word into the receiving register.

A storage is always started, then cancelled, for a panel key fetch (Figure 5201). BCU starts storage and sets a return address register in the normal way. In the case of an invalid address or an address parity check, select has already been sent to storage before the error condition is recognized.

At B time of the select cycle, BCU sends cancel to storage. The cancel signal causes storage to regenerate the selected address without delivering data to the SBO latch. Any errors detected by storage are ignored.

Even though the selected storage is cancelled, it still sends advance to the BCU. The BCU uses advances for all of the normal functions, including the gating of the SBO latch. The panel key fetch signal is delayed to time the delivery of the panel key data to the SBO latch as if it were data from the cancelled storage.

ADDRESS COMPARE

- An address compare circuit compares bit positions 0-20 of the address OR with 0-20 control panel address keys.

- The compare signal is used for scope synchronization and for stop on address compare.

The BCU contains an address compare circuit that checks every address sent to storage against the address setting in the system control panel address keys (Figure 5202). Bit positions 0 through 20 of the address OR are compared with the corresponding address keys.

The address compare circuit generates a scope synchronization (sync) signal for troubleshooting and stops the CPU when a particular address in storage is accessed. Two switches on the system control panel determine the results of an address compare. A three-position switch selects the CPU, the channel, or both. In the up (CPU) position, a scope sync signal is generated only when an address compare occurs on a CPU generated address. In the down (channel) position, a scope sync signal is generated only if the address compare is the result of a channel-generated address. In the center (unlabeled) position, any address compare generates a scope sync signal. The scope sync signal can be used to halt the CPU under control of an address compare stop switch. If a halt is generated, the CPU finishes the instruction in progress, then goes to the stopped state.

The output of the address compare circuit causes an address compare trigger to turn on for one machine cycle, beginning at B time of a BCU select cycle and ending at B time of the post-selection cycle (Figure 5202). The output of the address compare trigger is controlled by the address compare select switch on the system control panel. When this switch is in the center position, the address compare trigger

output is routed unconditionally to the address compare sync points. A sync point is located on each CPU gate and in each channel. When the select switch is set to the CPU position, the compare trigger output is not gated to the sync points unless the BCU gating latch is off (gate CPU). When the select switch is set to the channel position, the compare trigger output is gated to the sync points only if the gating latch is on (gate channel). The address compare sync pulse generates a CPU halt if the compare stop switch is in the stop position.

SPECIAL OPERATIONS

- Normal fetch and store operations are varied slightly for certain special operations.
- Set key, insert key, diagnose, and test and set instructions require a variation from the normal BCU operations.
- Single-cycle mode and some manual operations vary the operation of the BCU.

The normal fetch and store operations of the BCU are varied slightly to handle the storage access portion of the set and insert key instructions, the diagnose instruction, the test and set instruction, and some system control panel manual operations. The set key instruction is a store operation to one or two SP units. The insert key instruction is a fetch from an SP unit. The diagnose instruction is a CPU fetch with data delivery to the maintenance control word (MCW) register via the channel SBO. The test and set instruction is a combination CPU fetch and store operation. Several manual operations require the BCU to do a panel key fetch. Examples are: set PSW, load A-B, and store GP or FP registers. Placing the CPU in single-cycle mode also varies the operation of the BCU.

Set Key

- The set key instruction stores a five-bit key into SPF storage.
- The BCU handles errors on a set key operation as if it were a CPU store operation.
- A HSS is selected, but it is cancelled by the set key control line.
- On Models I75 and J75, the BCU performs two set key operations for each set key instruction.

The set storage key instruction is the means by which a configuration of key bits for a block of storage is set into the storage of the storage protection feature (SPF). The SPF storage holds a five-bit key (plus a parity bit) for each block of 256 main storage words (double words). On the set key instruction, SPF storage is addressed by the contents of general register R2 and the key set into SPF is taken from bits 24-28 of general register R1.

On a set key instruction, the BCU receives three control lines from the CPU: CPU fetch request, return to J, and set key. The set key line alters the operation of the BCU. Errors are handled as if they had occurred on a store operation and no data is returned. A return address register is not set even though return to J is active. A set key line is sent to storage. This line causes the selected HSS to cancel its operation and tells the SPF to store the incoming key. The BCU generates a parity bit for the incoming key and gates the six bits to the SPF.

For a Model H75, the BCU selection and post-selection cycles on a set key are the same as on a CPU store operation, except that the SBI latch is not set. (The H configuration has a single SP unit.)

For the Models I75 and J75, four HSS are interleaved. Two of these HSS are in one 2365 and the other two are in a second 2365. Each 2365 has one SP unit; therefore, two SP units must hold identical keys for a particular block of addresses. This means that on a set key operation, the new key must be set into two SP units. The BCU performs a set key operation to one 2365, then does a second set key to the other 2365 of the interleaved pair (pair of 2365's containing the four interleaved HSS).

The BCU has two triggers to control set key instructions that must set a key into two SP units (Figure 5205). When positive select turns on for the first set key operation, key trigger 1 is set to inhibit the accept signal to the CPU. As soon as positive select goes off, key trigger 2 is turned on. Key trigger 2 reverses address bit 19 to set the key into the other SP unit of the interleaved pair.

The BCU can make one or more channel storage accesses between the first and second set key operations. The CPU, however, cannot access storage because it is still waiting for an accept response to its set key request.

When positive select turns on for the second set key operation, key trigger 1 is turned off to allow the accept signal to be generated. The accept reset trigger turns off key trigger 2, restoring address bit 19 to normal.

Insert Key

- The insert key instruction fetches a five-bit key from SPF storage.

- The BCU handles errors on an insert key operation as if it were a CPU store operation.

- A HSS is selected, but it is cancelled by the insert key control line.

- Unlike a set key operation, the BCU fetches a key from a single SP unit on all models.

The insert storage key instruction is the means by which a programmer can examine a previously stored protection key for a particular block of storage. The instruction fetches the SPF key addressed by GR R2 and sets it into GR R1 bits 24-28.

General register R2, bits 8-31, is routed through the AA and is set into SAR 0-23. As on any CPU fetch or store operation, the BCU sends 14 bits of SAR to storage and either seven bits (H75) or eight bits (I75, J75) are routed to the SP unit to address the key to be fetched. The SP unit delivers the addressed key to the BCU key buffer register. The key (five bits plus a parity bit) is routed from the key buffer to the AOE mask where three 0s are added to make a full byte.

While the BCU is fetching the key, the CPU routes GR R1 through RBL to M and from M through the main adder to K. The AOE mask byte, which contains the fetched key, is set into K 24-31. This byte replaces the corresponding byte from GR R1. Bits 0-31 of K are then set back into GR R1 to complete the instruction.

At the end of an insert key instruction, bits 24-28 of GR R1 contain the fetched key. Bits 29-31 of GR R1 contain 0s, and the remainder of GR R1 is unchanged.

The BCU handles the insert storage key instruction almost identical to the way it handles the set storage key instruction for a Model H75. Unlike the set key instruction, the insert key is not affected by four-way interleaving. On an insert key instruction, the key is fetched from whichever SP unit is addressed by R2. On Models I75 and J75, the other SP unit containing identical information is not involved in the operation.

On an insert key instruction, the BCU receives I fetch request, return to J, and insert storage key control lines from the CPU. The insert storage key line causes the BCU to treat the fetch as a store operation; no return address positions are set even though return to J is active and any errors detected during the operation are handled as if they occurred during a CPU store operation.

The insert key line to the selected HSS causes a cancel and causes the SP unit to fetch the addressed key. The BCU uses an SPF advance signal from the SP unit to gate the fetched key into the key buffer register and to signal the E unit that it can proceed from the first fixed-point (FXP) cycles to a halfword logical cycle.

The advance and reset busy lines from the selected (and cancelled) HSS are used by the BCU as if the operation were a CPU store.

Diagnose

- The diagnose instruction loads the MCW register from a specified storage location.

- Diagnose is a CPU fetch with data return via the channel SBO to the MCW.

The diagnose instruction is handled uniquely by the BCU because it consists of a CPU fetch, but the fetched data is returned via the channel SBO to the MCW register. The CPU sets the calculated storage address into SAR and sends the I fetch request and diagnose signals to the BCU. The BCU starts storage and relays the storage address from SAR through the address OR to storage. At select time, the BCU sets the diagnose position in a return address register.

When the fetched 72-bit word returns from storage, the BCU sets it into the SBO latch register. From the SBO latch, the storage word goes out on the channel SBO. The BCU uses the advance pulse from the selected HSS to generate a diagnose select signal. The signal is generated by sampling the return address register set at select time. The diagnose select signal gates 0-31 of the channel SBO into the MCW register.

Test and Set

- Storage performs a unique operation for the test and set instruction; it does a combination fetch and store.

- The BCU performs a normal CPU fetch operation, except for a unique mark register reset.

The storage unit performs a unique operation for the test and set instruction. The addressed location is fetched and sent unaltered to the SBO latch register the same as on a fetch operation. Unlike a normal fetch, however, the storage uses a mark bit supplied by the CPU to designate a single byte to be changed in storage. The storage unit sets the designated byte to all 1s then regenerates the 72-bit word; thus, the storage unit performs a combination store and fetch.

To cause a test and set, the BCU performs a normal CPU fetch but sends a test and set line to the selected storage unit. No special gating is required

for the mark bit. The CPU sets a bit into the mark register; the mark register is gated to storage on any CPU operation. A unique mark-register reset, however, is required for the test and set instruction. The mark register is reset after any CPU store operation and after a test and set operation.

The storage protection unit is active on a test and set instruction. A SAP check causes the original word to be regenerated in storage and, instead of the fetched word, the storage unit delivers all 0s with good parity bits to the SBO latch register. This protects the CPU from taking a machine check caused by a SAP error.

SINGLE CYCLE

- Single cycle applies to CPU operations only.
- On single cycle, running clock pulses are continuous; controlled clock pulses are released one set per depression of the start key.
- Storage units are not affected by single cycle.

The single cycle mode is applicable only to CPU operations. The BCU handles channel requests with running clock pulses available whenever power is on, except during a system reset.

Single cycle mode does not affect the storage units. A storage cycle, once started, runs to completion in a fixed amount of time. A fetch request honored during single cycle causes a complete storage cycle with one depression of the start key. The fetched data is set into the receiving register (A, B, or J) by a running clock pulse. Similarly, a CPU store operation is completely executed with one set of control clock pulses. For correct operation, however, certain circuits within the BCU must operate differently during single cycle than during normal operation.

CPU Fetch

- CPU request latch is set with a control clock pulse.
- When a request is honored, the request latch is turned off and blocked during the following controlled clock cycle.
- There are two CPU accept triggers: pulse accept is reset by the running clock and normal accept is reset by the controlled clock.

The CPU request latch in the BCU is set with control clock pulses. For a single cycle CPU fetch, the request latch is set with the same conditions as those used for normal operation (a CPU fetch request and a control L clock). For single cycle, however, the request latch must be turned off before the next depression of start. To prevent honoring the same request twice, the request latch must be blocked so that it will not be set on the next depression of start.

To properly control the request latch, the CPU accept trigger is duplicated (Figure 36). These two accept triggers are set in parallel, but one is reset by a control clock while the other is reset by a running clock. The CPU accept signal is taken from the normal accept, so that accept to the CPU remains on until the next set of control clock pulses. This simulates normal operation to sequence CPU requests to the BCU. The pulse accept trigger causes the CPU request latch to be reset on the running clock cycle following a CPU select.

The CPU request and accept conditions are the only area of BCU changed for single cycle mode. All other areas are controlled by running clock pulses.

CPU Store

- To ensure that K is set, the BCU delays one controlled clock cycle before honoring a single cycle CPU store.

To perform a single cycle CPU store, the BCU must delay one cycle before honoring the request. Note that on a CPU store, the CPU often brings up store request one cycle before loading the K register. If BCU honors the request on the first cycle following the request, the K register is actually set with the A clock of the post-selection cycle. On single cycle, this A clock will not occur until the next time start is depressed.

Two latches are used to delay a single cycle store request for one cycle (Figure 37). This one-cycle delay ensures that the K register contains the data to be stored when BCU selects storage.

The accept and pulse accept triggers operate the same on a single cycle store as they do on a single cycle fetch.

ERROR HANDLING

- BCU handles two kinds of errors: parity checks and program checks.
- Most parity checks cause a machine check.

Two kinds of errors are handled by the BCU: parity check errors and programming check errors. The action taken for one of these errors depends not only on the type of error, but also on the user (channel or CPU) and whether the error occurred on a fetch or a store operation (Figure 9104).

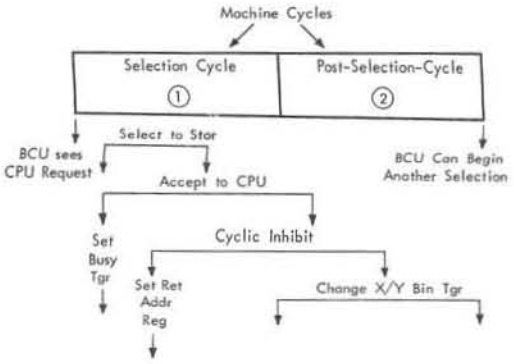


FIGURE 35. SELECTION AND POST-SELECTION CYCLES: CPU FETCH

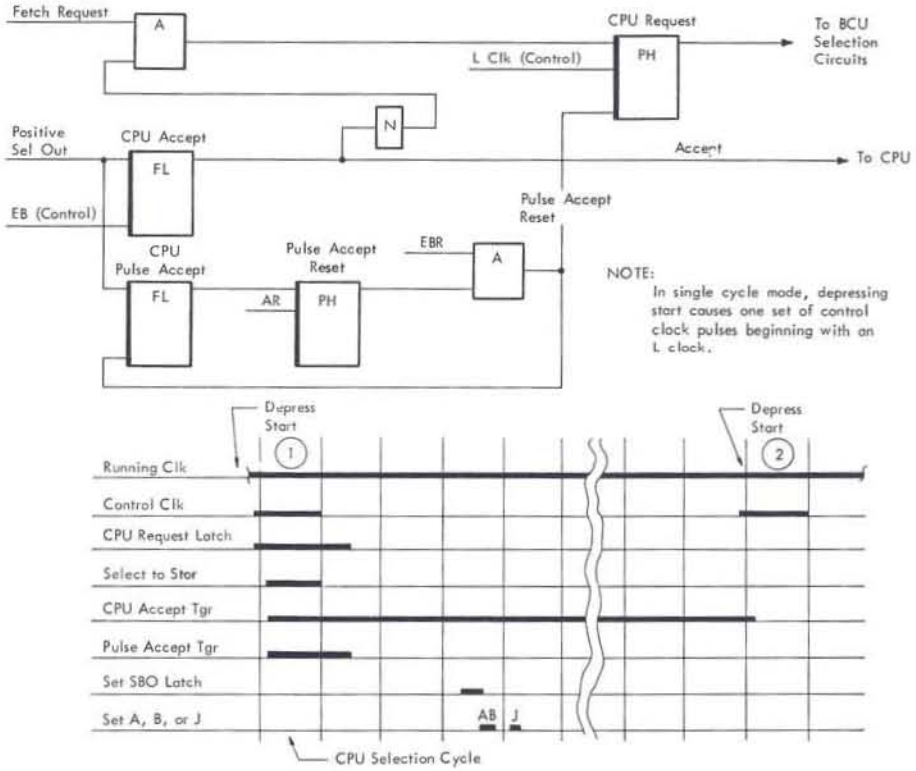


FIGURE 36. SINGLE-CYCLE CPU FETCH

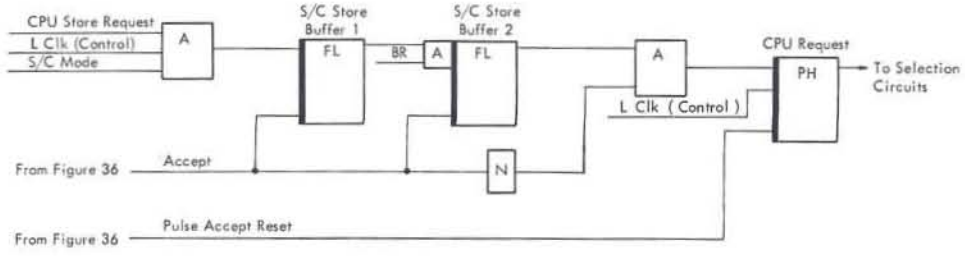


FIGURE 37. SINGLE-CYCLE CPU STORE

Parity check errors indicate a hardware malfunction and result in a logout followed by a machine check interrupt if they occur on a CPU operation. Parity errors that occur on channel operations are sent to the channel, where a channel interrupt will be initiated. Programming errors are caused by asking the machine for an illegal or an impossible operation. Programming errors result in a program interrupt.

Parity Checks

- BCU checks parity of all addresses off of the address OR.
- Each HSS parity-checks addresses and marks.
- SPF parity-checks addresses and keys.
- Each HSS parity-checks all data bytes in MDR on both fetch and store operations.

Address and Mark Parity

The BCU parity-checks all addresses that pass through the address OR (Figure 5203). This 24-bit address check is the only parity checking done within the BCU. When a bad parity address is detected, the BCU cancels the selected HSS so that the selected address will be regenerated without change and will not be delivered on the SBO. The HSS suppresses any errors found to prevent other error indications caused by the faulty address.

Each HSS parity-checks the 14-bit addresses that it receives. On store operations, the HSS also checks the parity of mark bits. The SPF parity-checks the address bits that it receives and parity-checks all keys that it uses: the parity of the two keys used for bit match, the parity of the incoming key on a set key instruction, and the parity of the outgoing key on an insert key instruction. A parity error detected at MAR, the mark register, or within the SPF sends storage address check to the BCU if the BCU has not sent cancel. A storage address check causes the selected address to be regenerated without change and prevents data delivery on the SBO.

The BCU address check and storage address check lines are ORed within the BCU to produce a single address check line. This line sets either the CPU address check latch or the channel address check latch, depending on the channel bit of the return address register associated with this operation. If the CPU address check latch is turned on, a machine check interrupt is initiated. If the channel address check latch is turned on, a channel address check signal is sent to the channel.

Storage Data Check

Each HSS checks all data bytes in its MDR on both store and fetch operations. A data byte parity error brings up the storage data error line to the BCU if cancel is not on. A storage data check does not alter the storage operation (fetch or store). The storage data error line is sent directly to the channel where it is recognized by channel advance on a channel operation. For CPU operations, the storage data error line is combined with the return address register outputs to set the X or Y CPU data check latch on a CPU store. Bad parity on a CPU fetch is not an error; the CPU may not use the portion of the storage word that has bad parity. The CPU checks the parity of fetched data that is uses.

Program Checks

- Two program checks are handled by the BCU: invalid address and storage address protect (SAP).
- BCU detects invalid addresses.
- SPF generates SAP.

Programming checks handled by the BCU are:

1. Invalid address (address not within main storage of this system).
2. Storage address protect (SAP) error (mismatch of storage protect keys).

Invalid Address

On every operation, the BCU checks for an invalid address (Figure 5204). An invalid address is an address which is beyond the range of storage addresses available on a particular system. Whenever an invalid address is detected, BCU cancels the selected HSS.

An invalid address usually causes an interrupt; however, the type of interrupt and the time at which it is taken depend on the operation being performed when the invalid address occurs. For channel operations and CPU fetches, the BCU stores the invalid condition in a return address register and sends it to the channel, A-B, or J along with the corresponding advance. A channel that receives invalid will initiate a channel interrupt. For fetches, the CPU does not initiate an interrupt until it determines whether the data from the invalid address is required.

For a CPU store operation, the invalid address condition is set into the CPU invalid store buffer trigger. This trigger initiates a program interrupt.

Storage Address Protect (SAP)

The SPF generates a SAP error on any store operation if the storage protect keys do not match (unless the in key is all 0s). On fetch operations, the SPF generates a SAP error if the keys do not match and the read-protect bit is on.

On store operations, the SAP error cancels the selected HSS and sends SAP to the BCU (Figure 5204).

On fetch operations, the SAP error suppresses data delivery on the SBO and sends SAP to the BCU.

For any channel operation, the BCU sets the channel SAP trigger if SAP is received from the SPF. The channel SAP error is recognized by channel (with channel advance) and causes a channel interrupt.

For CPU store operations, the BCU sets the CPU SAP trigger if SAP is received from the SPF. The CPU SAP trigger feeds the interrupt circuits to cause a program interrupt.

For CPU fetch operations, the BCU must handle SAP from SPF similar to the way an invalid address is handled on a CPU fetch operation. A program interrupt is not desirable until CPU determines that the fetched data is to be used. The BCU has an X and a Y SAP fetch buffer trigger; one of these triggers is set when SPF generates a SAP error on a CPU fetch. The output of these triggers is ORed to feed the A SAP, the B SAP, and the J SAP circuits; these circuits determine whether the data from the error fetch is actually required in the operation being performed. If this data is necessary, either the A SAP, the B SAP, or the J SAP trigger is turned on to cause a program interrupt.

Cancel

- Cancel causes storage to regenerate the selected address.

- No data is delivered to the SBO.

A cancel signal causes a selected HSS to regenerate the selected address without change. On a cancelled operation, data to the SBO is suppressed and any errors detected by the HSS are suppressed.

The cancel latch within a HSS is set in one of three ways:

1. Cancel from the BCU.
2. SAP signal from the SPF.
3. Set or insert key line from the BCU.

The BCU cancel signal is brought up on a BCU-detected address check, on an invalid address, and for a panel key fetch (Figure 5201).

Communicate and CPU Storage Busy

- The CPU communicate line controls two error latches in each HSS and one error latch in each SPF.

- The purpose of these error latches is to light an an indicator if the specified error occurs on a CPU operation.

- HSS indicators are address (eight lights) and data (eight lights).

- SPF indicator is SPF (four lights).

- The CPU storage busy line prevents entry into an interrupt sequence while BCU is handling a CPU request.

Two error latches in each HSS and one error latch in each SPF are controlled by a CPU communicate line from the BCU. The purpose of these latches is to light an error indicator if an error occurs on an operation requested by the CPU (Figure 5203). These indicators isolate a bad address or bad data as having come from either the CPU or a channel. The HSS indicators controlled by CPU communicate are address and data. The SPF indicator controlled by CPU communicate is SPF.

The CPU interrupt controls require BCU to generate a CPU storage busy line. This signal prevents entrance into the interrupt sequence while the BCU is busy with a CPU request.

The BCU generates the CPU communicate line from the CPU pulse accept condition (Figure 5206). The CPU storage busy signal is held up for two cycles by ORing the pulse accept and CPU communicate conditions to control an EBR PH.

MACHINE CHECKS

- The BCU generates a machine check signal for any of three error conditions:
  1. Address parity check detected by the BCU or storage on a CPU operation.
  2. Storage data parity check detected by storage on a CPU store operation.
  3. X/Y and W/Z return address triggers are out-of-synchronization (any operation).

The three error conditions are ORed to produce stop clock which turns off the control clock and initiates a logout followed by a machine check interrupt (Figure 5203).

Address Parity Check

- An address parity check is detected by BCU or storage.
- BCU sets channel address check or CPU address check.

The address OR parity check within the BCU and the storage address check are described under "Parity Errors." These two error conditions are ORed within the BCU, then set into either the channel address check or the CPU address check error latch. The CPU address check latch is one input to the stop clock OR.

Store Data Parity Check

- The HSS units check MDR parity on every operation.
- BCU ignores data check from HSS on CPU fetch operations.
- Data check is sent to the channel on channel operations.
- Data check causes a machine check on a CPU store operation.

The 2365 storage units check parity on the MDR on every operation as described under "Parity Errors." Unless cancel is on, the storage sends storage data check whenever it detects an error (Figure 5203). The BCU ignores a storage data check on CPU fetch operations, but sets either the data check X CPU or data check Y CPU latch if the error occurs on a CPU store operation. Both of these latches feed the stop clock OR.

Return Synchronization Check

- X/Y and W/Z binary triggers must stay in synchronization.
- Synchronization is checked only when no HSS are busy.
- W and not X, or Z and not Y, causes machine check.

The X/Y binary trigger gates return addresses into one of the two return address registers; the W/Z binary trigger gates the output of one of the two return address registers. The X condition of X/Y and the W condition of W/Z point to the X return address register; the Y condition of X/Y and the Z condition of W/Z point to the Y return address register. This relationship, or synchronization, must be maintained in order to return fetched data to the correct register.

When no HSS are busy, the synchronization of X/Y with W/Z can be checked; either X and W or Y and Z should be on. The checking circuit tests for W and not X, or Z and not Y (Figure 5203). Either of these conditions signals a return address synchronization error. This error condition brings up the stop clock OR to signal a machine check error.

INSTRUCTION PREPARATION

INTRODUCTION

- Simultaneous execution and preparation of instructions gains processing speed.
- Execution is program dependent.
- Preparation is automatic.
- Executions start with operation code registers loaded and operands delivered.
- Executions are guided by sequencers.
- Executions are done one at a time.
- Main objective of preparation is to make every machine cycle and execution cycle.
- Preparations are guided by sequencers T1 and T2.
- Instruction fetching is not T1 and T2 controlled.
- T1 and T2 cycles, instruction fetching, and executions are all done simultaneously.

The 2075 prepares instructions for execution in one unit and executes them in another unit, and thus greatly speeds up the running of any program. The use of different units enables simultaneous preparation and execution. While instruction one is being executed, instruction two is being prepared, see Figure 38.

The instruction preparation unit (I unit) performs all functions that are not directly dependent on the particular instruction being processed. For instance, preparation includes instruction fetching, no matter what the instruction, it must be fetched from storage. The execution unit (E unit) performs the specific operation called for by the instruction being executed. For instance, on a divide instruction the E unit divides, on an add instruction it adds.

The functions performed by the E unit are determined by the stored program. The I unit, however, without control from the stored program performs automatically all of those functions which are necessary to the running of any program. This distinction between automatic (or built-in) control of preparation as opposed to program (or external) control of execution is of primary importance for an understanding of 2075 operation. The preparation functions performed by the I unit under built-in control serve as a foundation for all operations performed by the 2075. A preparation failure can affect all executions.

Before being more specific about what the I unit does, we must look more closely at E unit operation. Some characteristics of a typical execution are shown on Figure 39.

Before an execution starts, the operation code of the instruction to be executed is in an E unit operation register (EOP). For most instructions, operands are delivered to RBL or the J register before the execution starts. During the execution, the operands are taken from RBL or J, the required operations are performed, and the results are delivered to specified locations via the K register.

On each machine cycle of an execution, the data flow and the operation performed are guided by a trigger called a sequencer. A series of sequencers is used for each execution. Many sets of sequencers are available. The sequencers to be used for any execution are determined by the instruction to be executed.

An execution may be a simple move from one general register to another requiring two E cycles, or it may be as complex as a VFL divide in which both operands come from storage and the result is returned to storage. This latter execution may require hundreds of E cycles. In all cases some selected sequencer defines the first cycle and another sequencer defines the last cycle of each execution. Executions are done one at a time; that is, the first cycle sequencer (EI) for an execution may not come on until the last cycle sequencer (ELC) for the preceding execution has been turned off.

For any program the shortest running time is achieved when every machine cycle is an execution cycle. The I unit's job is to perform all preparation functions in a way that enables continuous executions. This ideal performance is shown at the bottom of Figure 39.

The distinction between preparation and execution functions and a first breakdown of the preparation functions performed by the I unit are shown on Figure 40.

Most preparation functions are sequencer controlled as are executions. Unlike executions, one set of two sequencers is used for the preparation of all instructions. Before the preparation sequence can begin, instructions must be brought from storage to the processor. The I unit contains instruction buffers and a set of mechanisms aimed at keeping instructions always available in the buffers. The mechanisms that control instruction fetches monitor many conditions.

Instruction fetches are not made when they will interfere with executions or other preparation functions. Generally, however, new instructions are fetched before all buffered instructions are used. With the instructions available in the buffers, the sequencer controlled preparations are started. Under ideal conditions, sequencer controlled preparations are completed in two machine cycles. The preparation sequencers T1 and T2 guide the delivery of operands to the execution unit, the setting of the operation register in the E unit (EOP), and the sending of a start signal to the E unit. The "keep track" functions such as updating the instruction counter and controlling the gates from the instruction buffers to the operation register are also guided by T1 and T2. The start signal is sent to the E unit as the T2 functions are completed.

Most execution sequences are longer than two machine cycles but some require only two cycles. Assuming two cycle executions and otherwise ideal conditions, instruction fetches, sequencer controlled preparations, and executions will proceed simultaneously and without interfering with each other as shown at the bottom of Figure 40.

The 2075 processor as described consists of the instruction preparation unit and execution unit. A further breakdown by functional section is shown on Figure 41.

Three different sections perform executions. The execution unit (E unit) performs on executions requiring arithmetic or logical manipulation of data. The instruction execution unit (IE unit) performs on executions that are closely associated with I unit functions or mechanisms. The branch unit (Br unit) performs on executions that may result in a branch to a new instruction address. Each of these units executes certain instructions independently of the other execution units. Each receives its own start signal from the I unit and uses its own sequencers to control its operation as has been described for the E unit.

The use of three independent execution units does not enable more than one instruction to be executed at any one time. The only simultaneous operation of execution units occurs on some instructions which require the use of two units for their execution. On these instructions, the E unit operates simultaneously with either the Br unit or the IE unit. For example, on branch on index high (BXH) the E unit does arithmetic to determine the success of the branch, and the Br unit fetches instructions from the branch address.

The bus control unit (BCU) is another functional section of the 2075 processor. BCU contains the data flow paths and the controls for storage operation with the channels as well as with the processor. A main function of the BCU is to service near simultaneous storage requests from the different channels and the CPU in the order of their assigned priorities.

CONTROL AND FUNCTIONS OF T1 AND T2

- Dependencies between data paths and between functions require variations in cycling.
- Required variations are achieved by three IS control signals: TN T1, TN T2, and I to E transfer.
- Interference between executions and T1 cycles or T2 cycles is prevented by block T1-M and block T2-M.
- Interference between one execution and the next is prevented by busy triggers and last cycle sequencers.
- Operation registers are set by IS control.
- T1 cycles compute effective addresses and initiate fetches for storage operands.
- T2 cycles deliver operands from the registers.
- I to E transfer times the start of all executions.
- ICR contains the storage address of the instruction that is being prepared.

Note that the 2075 uses specific units to perform various jobs and thereby gains speed. Similarly, the I unit gains speed by keeping the various jobs that it must do as independent as possible. Figure 42 shows some of the main flow paths through the I unit. Mechanisms which are a part of units other than I are shown by dotted lines and their unit is given at the upper right corner.

Instruction addresses must be delivered to the storage address register (SAR) before instructions can be fetched. When a program is initially loaded, the program status word (PSW) comes from storage on the storage bus out (SBO). The PSW is gated from SBO to the J register and then to the PSW register in the I unit. When instructions are to be fetched, the portion of the PSW that contains the instruction address (ICR) is gated to the incrementer (Incr). The incrementer can deliver the address unchanged to SAR, or it can add appropriate increments when instructions in advance of the ICR value are to be fetched. The incrementer is the only arithmetic unit needed to generate instruction fetch addresses from the ICR value.

Instructions must be delivered to five operation registers. Instructions come from storage on SBO and are gated by BCU into either the A or B instruction buffer registers. A gate select mechanism gates the proper halfwords for the next instruction to be prepared to an I unit operation register (IOP). The setting of IOP and of the other operation registers fed by IOP is done under control of preparation sequencers T1 and T2. The exact cycle on which each operation register is set depends on many things which will be discussed later. The general rule is that each is set as early as possible without interfering with its use by the preceding instruction.

Addresses for operands from storage are generated on the T1 preparation cycle. An address may require the addition of three quantities: the base and the index which are in the general purpose registers (GPR), and the displacement (D) field of the instruction. On T1, IOP contains the instruction to be prepared. IOP decoding gates the required address components to the addressing adder (AA) and sets the output of the AA to SAR. The base register is gated to the general bus left (GBL) and the index register is gated to the general bus right (GBR). GBL and GBR go through the AA ORs and into the AA without further gating. The AA receives inputs and generates a sum on every T1 cycle. On instructions which do not require an address, the sum is not set to SAR.

Operands from GPR are delivered to the register bus latch (RBL) in the E unit on T2 preparation cycles. On every T2 cycle, two registers are selected for gating to GBR and GBL by decoding the R1 and R2 fields of IOP. Except when IOP decoding indicates a floating-point instruction, GBL and GBR are gated to RBL after going through the AA ORs. On floating-point instructions a selected floating-point register (FPR) is gated to RBL. RBL, therefore, receives information from either a GPR or a FPR on every T2 cycle. If the gated registers are not required for the execution of the instruction, the contents of RBL are ignored.

The units and mechanisms described as independent are not completely independent. Instruction fetching and operand fetching have been described as various functions performed by specific mechanisms. Figure 42 shows storage addresses for instructions and for operands independently generated, but both being delivered to the SAR. The BCU (and storage) can process only one storage address on any one machine cycle; therefore, nothing would be gained if two storage addresses were delivered from the 2075. On any machine cycle the conditions of the instruction buffers can call for an instruction fetch. On any T1 cycle, depending on the instruction and sequencers T1 and T2, an operand fetch may be initiated. The need for the operand fetch is determined during the T1 cycle. Both addresses are generated; if the operand fetch is required, the operand address is delivered to the SAR and the instruction address is blocked. Only if the operand fetch proves unnecessary is the instruction fetch made. Under normal conditions, operand fetches have priority over instruction fetches. Whenever an instruction fetch will interfere in any way with the preparation or execution of an instruction already available in the buffers, instruction fetching is blocked.

Instruction fetches must sometimes be blocked; the preparation cycles T1 and T2 must also sometimes be blocked and for the same reason, dependencies exist between the units and mechanisms described as independent. Note on Figure 42 that on T2 cycles, operands from the registers are delivered to RBL. On certain instructions, the E unit makes additional use of RBL as a data path during execution cycles. For these instructions, T2 cycles are blocked until RBL is no longer required by the E unit. Figure 42 also shows that on T1 cycles, GPR is gated out as components of the operand addresses. On certain instructions, the E unit requires that GPR be gated out during execution cycles. For these instructions, T1 cycles are blocked until GPR out gating is no longer required by the E unit.

The situations described in the preceding text require that T1 or T2 cycles should not start until the execution unit has finished using some shared mechanism. Sometimes even though the T1 or T2 sequencer is on, its functions cannot be completed in one cycle; the cycle must be repeated. T1 is allowed to come on even though the instruction to be prepared has not yet been delivered to the instruction buffers. T1 must be repeated until the instruction has been loaded to IOP. T2 also must be repeated sometimes. If the instruction being prepared requires an operand fetch, the fetch is initiated during T1 and maintained during T2. Should BCU be busy and not accept the request immediately, T2 must be repeated until the request is accepted. When all preparation functions have been completed, the start of the execution may be further delayed because the previous execution is still in progress. Some possible variations in cycling that result from these delays are shown on Figure 43.

Note that either T1 or T2 cycles may be repeated and that on some machine cycles neither T1 nor T2 is on. However, time is lost only when E1 does not occur on the machine cycle following ELC.

The variations in preparation sequencing do not affect execution sequencing. When the start signal is sent for the execution of any particular instruction, the appropriate first cycle sequencer is turned on and execution proceeds completely under control of the executing unit.

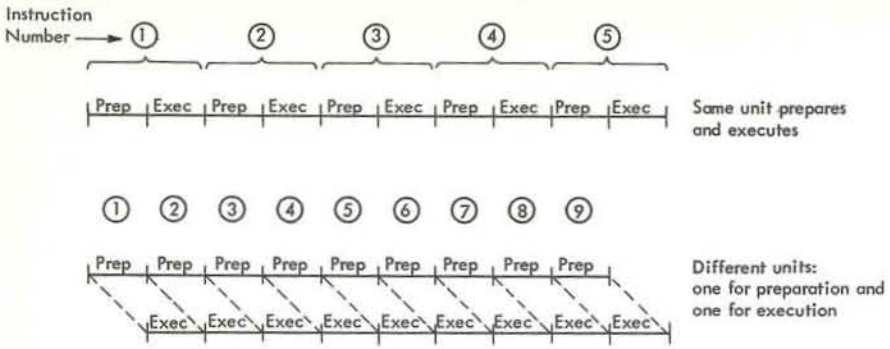


FIGURE 38. SIMULTANEOUS PREPARATION AND EXECUTION SAVES TIME

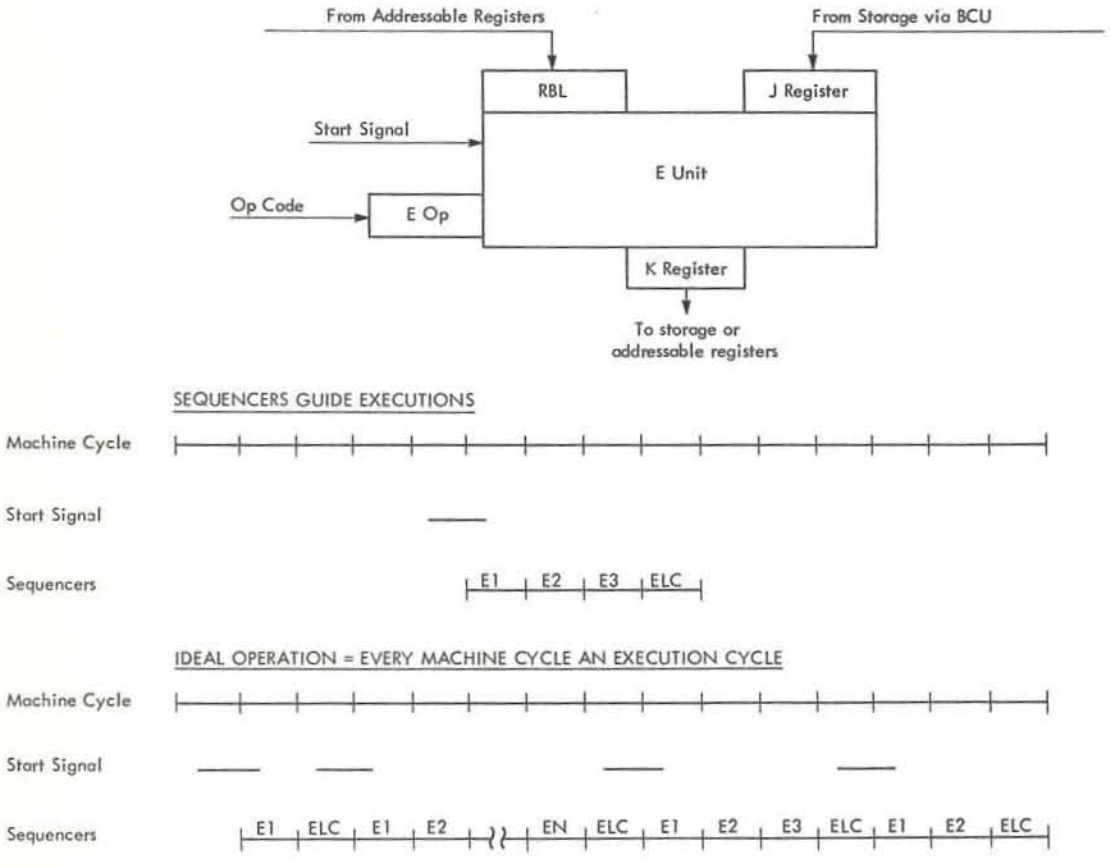


FIGURE 39. EXECUTIONS, SEQUENCERS, AND MACHINE CYCLES

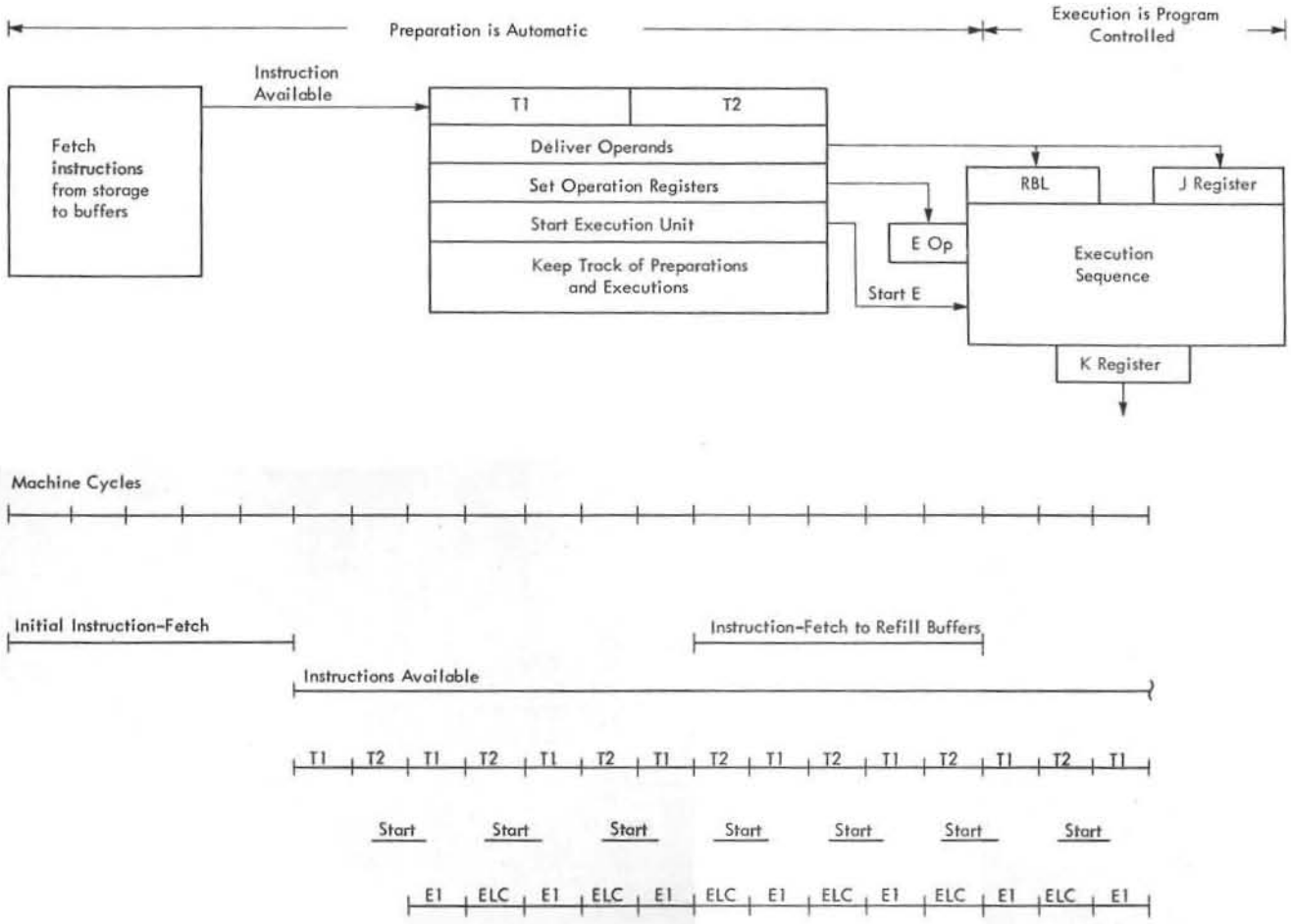


FIGURE 40. SIMULTANEOUS INSTRUCTION-FETCH, PREPARATION, AND EXECUTION

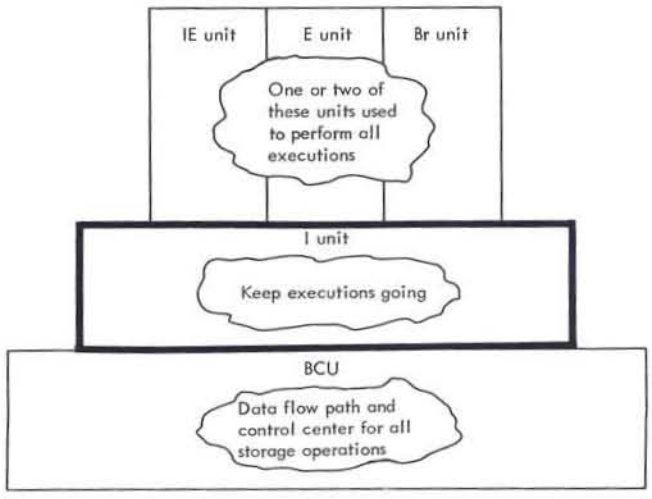
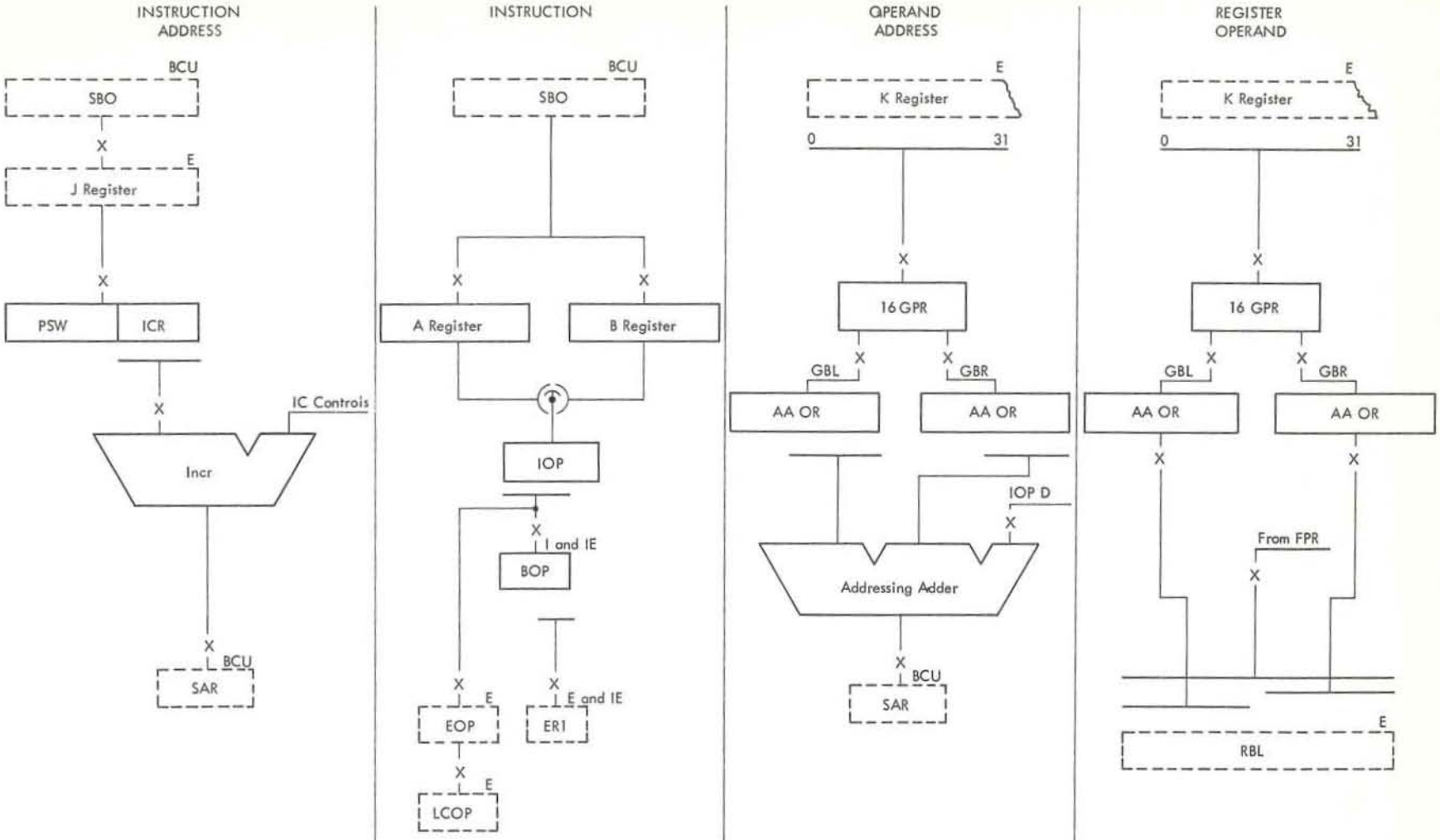


FIGURE 41. FUNCTIONAL SECTIONS OF 2075



Instruction addresses are always generated without preparation or execution sequencer control and are delivered to SAR when needed.

Instructions are moved from the buffers to the operation registers under control of preparation sequencers T1 and T2.

On T1 preparation cycles operand addresses are always generated and are gated to SAR when needed.

On T2 preparation cycles operands from addressable registers are always gated to RBL. They are used by E unit only when needed.

FIGURE 42. MAIN FLOW PATHS THROUGH I UNIT

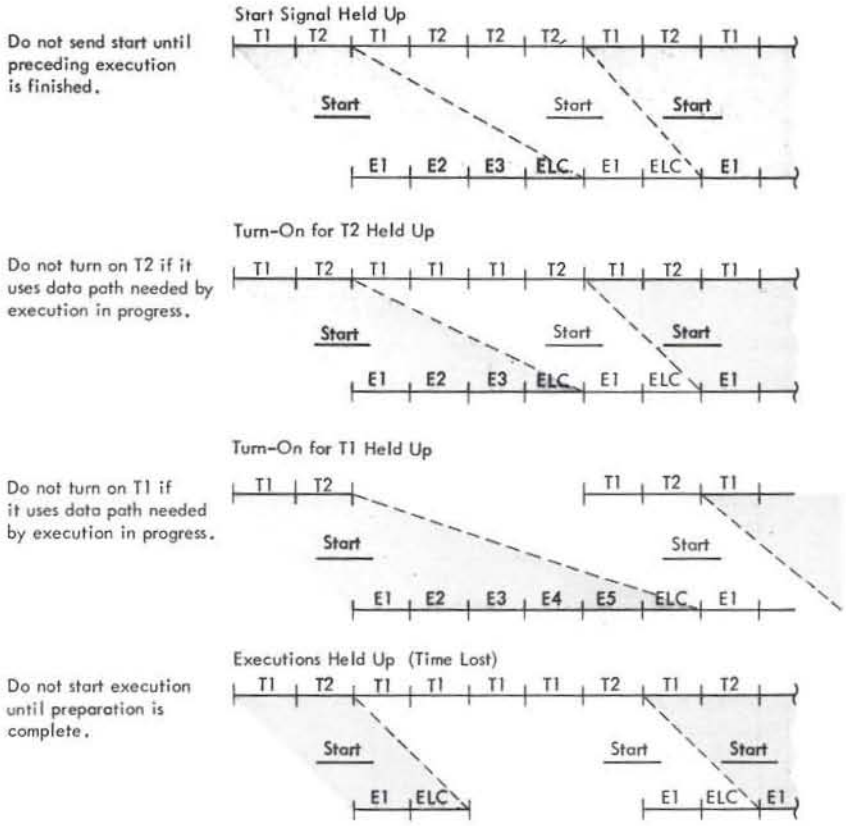


FIGURE 43. VARIATIONS IN PREPARATION CYCLING

Instruction Sequencing Controls

- Three control signals give all cycling variations: TN T1, TN T2, and I to E transfer.
- Each signal occurs once for each instruction.

There are two major control jobs in the I unit. The instruction counter (IC) controls fetch instructions to the buffers. The instruction sequencing (IS) controls turn T1 and T2 on and off and start the execution units. Both sets of controls operate to perform their assigned tasks as early as possible without slowing down the executions. Since the progress of preparations and executions determines the rate at which instructions are used and thus the need for instruction fetches, the sequencing controls are described first.

Three major control signals are basic to the IS controls. All of the required variations in preparation sequencing and starting execution units are obtained by controlling turn-on T1 (TN T1), turn-on T2 (TN T2), and I to E transfer. The functions of the first two signals are self-explanatory: I to E transfer signals the start of every execution.

Figure 44 shows, in simplified form, the development of each of these signals; Figures 5250, 5251, and 5252 show the detailed logic for each signal. The relationship of each signal to clock cycles and to the sequencer it controls is also shown.

By developing these signals at the proper time any required variation in preparation sequencing may be achieved; however, the signals are used for much more than simply setting their respective sequencers. For the preparation of a single instruction, any number of T1 or T2 cycles may occur; each of the three control signals, however, occurs only once. By using the control signals. as well as the sequencers. selected operations may be performed on the first or the last of a series of cycles controlled by the same sequencer. TN T1 is present at the beginning of the first T1 cycle only. TN T2 spans the last T1 and the first T2 cycle. I to E transfer marks the last T2 cycle. Figure 45 shows selective control by means of these signals.

Block T1-M and Block T2-M

- Block T1-M and block T2-M time TN T1 and TN T2.
- Block T1-M and block T2-M set at I to E transfer if T1 or T2 cycles will interfere with execution.
- Reset during execution when interference will no longer occur.

In the development of TN T1 and TN T2 one of the major considerations is the interference problem; that is, T1 or T2 must not be turned on if it uses any mechanism that is required by an execution in progress. This problem is solved by the use of two blocking triggers: block T1-memorized and block T2-memorized. If one of the triggers is on, it prevents the turn-on signal for the preparation sequencer to which it relates. The triggers are set at I to E transfer under control of a decode line from an I unit operation register, BOP. The line BOP decode block T1 (BD Blk T1) comes up during T2 for any instruction the execution of which requires that T1 cycles for the next instruction be prevented. When the reason for the block no longer exists; that is, when T1 cycles will no longer interfere with execution, the unit executing the instruction sends a signal that turns off the trigger and allows TN T1 to be developed. Blocking of T2 is done in the same way.

Figures 46 and 47 show the timings of these blocks. Note that for T1 the actual block anticipates the turn-on of the trigger. This is necessary because the normal case is to turn on T1 at I to E transfer the same time that the blocking trigger is set. Also note that in both cases the actual block is dropped when the turn-off signal is received and does not wait for the turn off of the blocking trigger. Figures 5253 and 5254 show the complete logic for turning block T1-M and block T2-M on and off.

Busy Triggers and Last Cycle Sequencers

- I to E transfer is held up if:
  1. T2 is not finished (OPF and not accept).
  2. Last execution is not finished (unit busy trigger and not same unit last cycle trigger).

I to E transfer is developed when T2 functions have been completed and the previous execution is complete or in its last cycle. With T2 on, the only delay in its completion occurs when the I unit is making an operand fetch and must wait for an accept from the BCU. The operand fetch (OPF) trigger is set when the I unit is to make a fetch and it is turned off when the request is accepted. Therefore, either OPF off or OPF on and accept indicate the completion of T2. Busy triggers and last cycle triggers that indicate the condition of each of the execution units must be monitored. As each execution is started, the E busy trigger is set if the E unit is used; the IE busy trigger is set if either the branch or the IE unit is to be used. This double use of the IE busy trigger is possible because the branch and the IE units are never used on the same instruction. An execution unit completes its part of an execution when its last cycle sequencer is on. The last cycle sequencer turns off

the busy trigger for the unit. The line last cycle memorized is brought up only when all execution units are not busy or are in their last cycle; and if T2 is complete, it allows I to E transfer (Figure 5252).

Setting Operation Registers

- Decoding from operation registers along with sequencers controls operation on each cycle.
- Preparation unit controls the setting of all operation registers.
- Each operation register is set as early as possible without interfering with use on the previous instruction.

Just as the setting of the T1 and T2 blocking triggers depends on decoding from an operation register so most other preparation and execution functions are directed by decode lines as well as sequencers. Operation decoding is performed from five registers. IOP and BOP are used to direct preparation. BOP is also used by the branch unit and the IE unit to direct executions. EOP and LCOP are used on E unit executions. ER1 is used by both IE and E to direct put-aways to general purpose or floating-point registers. All operation registers are set by the I unit using the execution unit busy triggers and last cycle information from each unit as part of the control circuits.

Figure 48 shows the operation register sets that occur as a string of instructions involving both E and IE executions is processed. The cycles during which each register must be correctly loaded are also shown.

IOP: Must contain the instruction being prepared for at least one T1 cycle before T2 is turned on, and for all T2 cycles. Normally IOP is set from the instruction buffers at TN T1 for each instruction and is not changed until TN T1 for the next instruction. Three conditions, however, require variations of this procedure. The logic for setting IOP under all conditions is shown on Figure 5256.

On occasion T1 is turned on before the instruction to be prepared is present in the instruction buffer. To take care of this situation, IOP is set at the start of each T1 cycle and TN T2 is not allowed until IOP has been correctly loaded for at least one T1 cycle. The line which allows T2 to be turned on is IOP loaded, which is generated by the IC controls. This line will be described in detail later. IOP loaded will be up during any cycle only if the instruction to be processed has been present in the instruction buffers a sufficient time to have been loaded into IOP at the beginning of the cycle. Since IOP loaded is needed for TN T2, IOP will contain the correct instruction for at least one T1 cycle.

The second variation to the normal setting of IOP occurs during T1 of the subject instruction of an execute instruction. At this time IOP must be set with bits ORed from two sources, the instruction buffers and a general register. IOP loaded indicates that the instruction buffers have been set to IOP and the execution sequence latch indicates that the information from the general register has been set to IOP. By setting IOP on every T1 cycle until both of these conditions are present, the required setting is assured.

The third variation to the setting of IOP occurs during the execution of SS instructions. An SS instruction consists of three halfwords. IOP holds only two halfwords. The third halfword is not required during instruction preparation but must be available in IOP for address calculation during execution. This is accomplished by setting IOP on every execution cycle of SS instructions and by allowing the E unit to control the gates from the instruction buffers to IOP during execution of these instructions.

BOP: Must contain each instruction for at least one T2 cycle before the I to E transfer, for all T1 cycles of the next instruction, and for all execution cycles of branch or IE executes until the branch or IE last cycle triggers are set. Normally BOP is set from IOP at TN T2 and not changed until the next TN T2. Two conditions require variation of this procedure. The logic for setting BOP under all conditions is shown on Figure 5257.

On some IE executes, T2 is turned on before the IE unit has reached its last cycle. For these instructions BOP is set on the same T2 cycle that the IE last cycle trigger is set. This set does not interfere with the use of BOP by IE since last cycle decoding is not required by the IE unit. The set meets the requirement that BOP be good for at least one T2 cycle before I to E transfer because I to E transfer cannot come up until the last cycle of any execution.

The second variation of the normal set of BOP occurs on the IE instruction store multiple. On this instruction the last four positions of BOP are incremented by the IE unit during execution to keep track of the registers that are stored. On these instructions BOP is not needed during T1 cycles of the following instructions.

EOP: Must contain the operation code of any instruction to be executed by the E unit from one cycle before the first E cycle until EOP decoding is no longer required (normally the turn-on of ELC). EOP is set

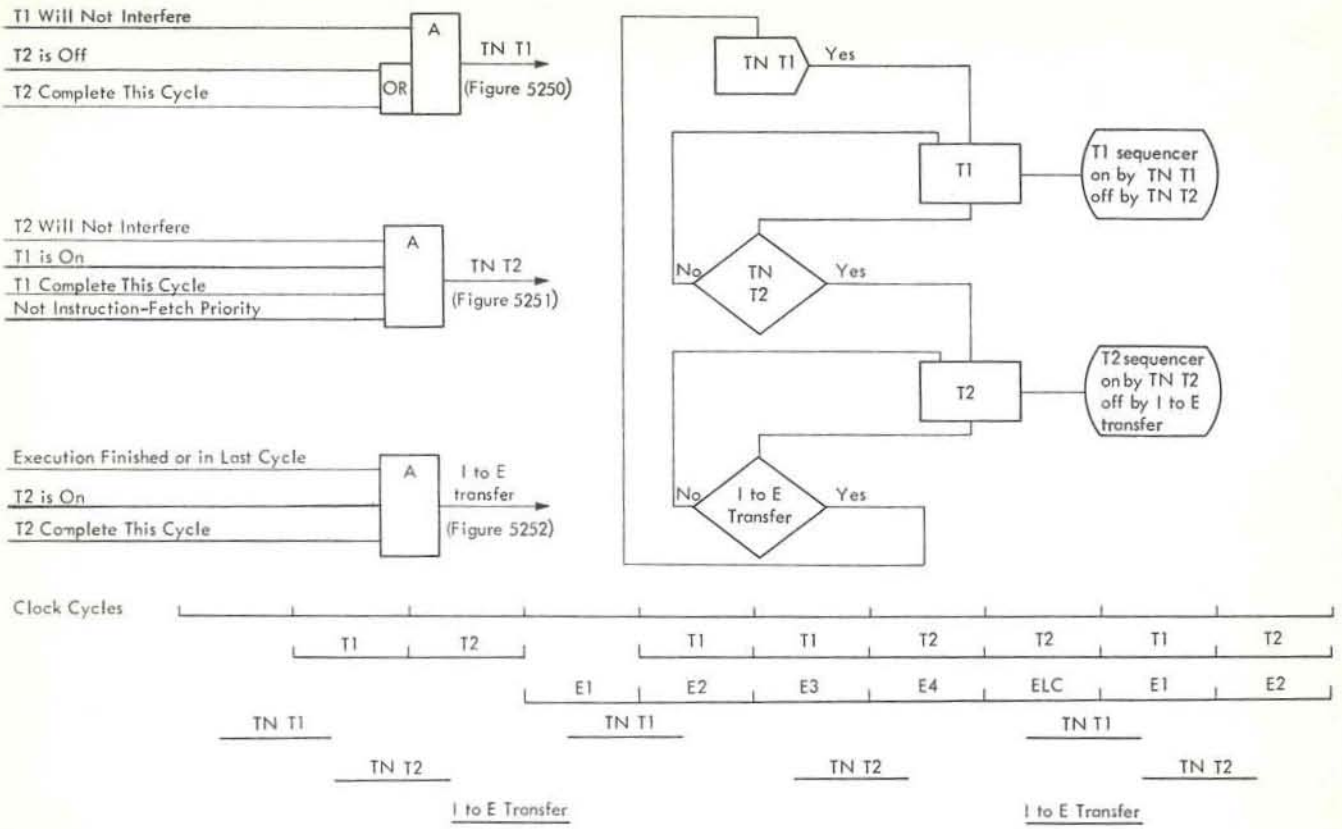


FIGURE 44. THREE INSTRUCTION SEQUENCING CONTROL SIGNALS

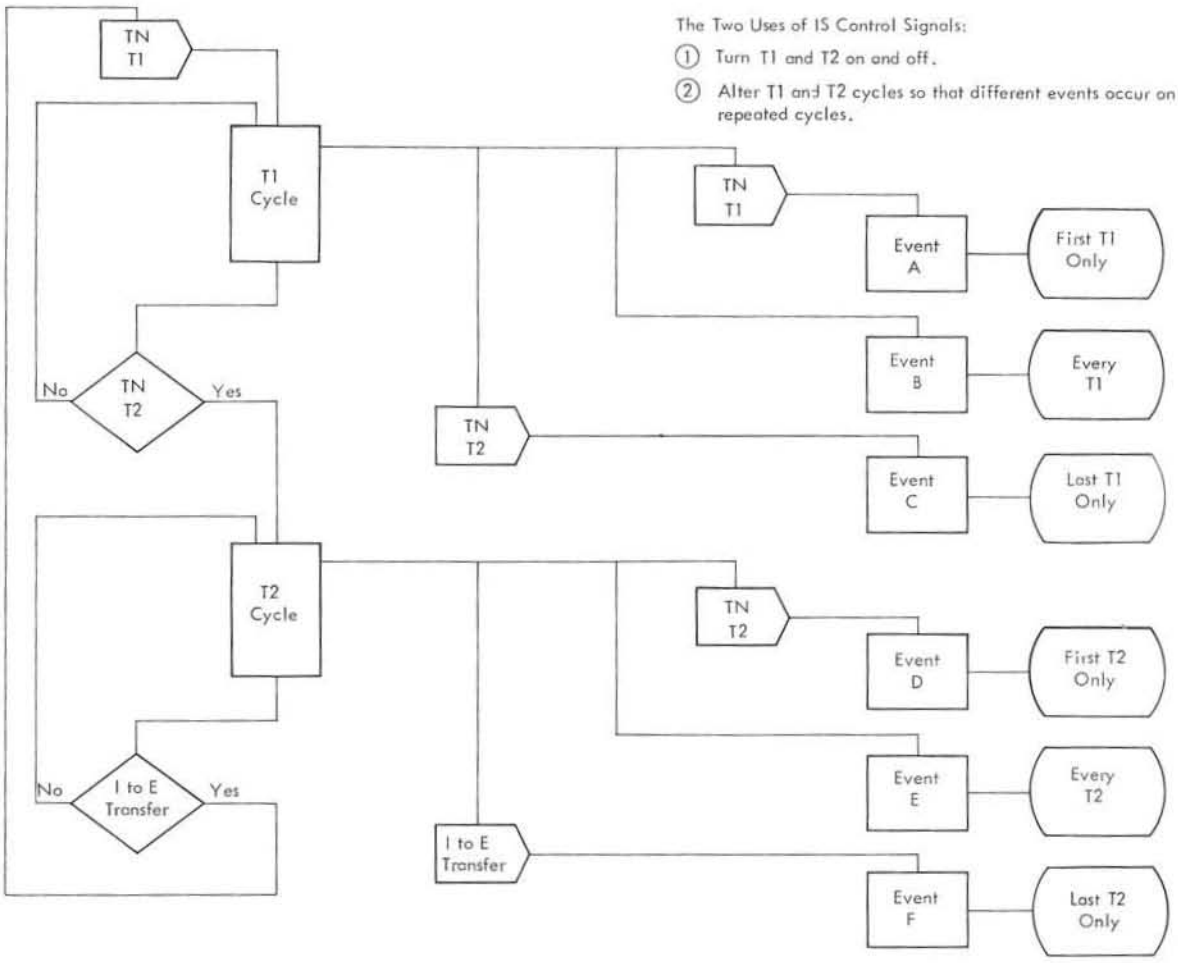


FIGURE 45. INSTRUCTION SEQUENCING CONTROL SIGNALS USED TWO WAYS

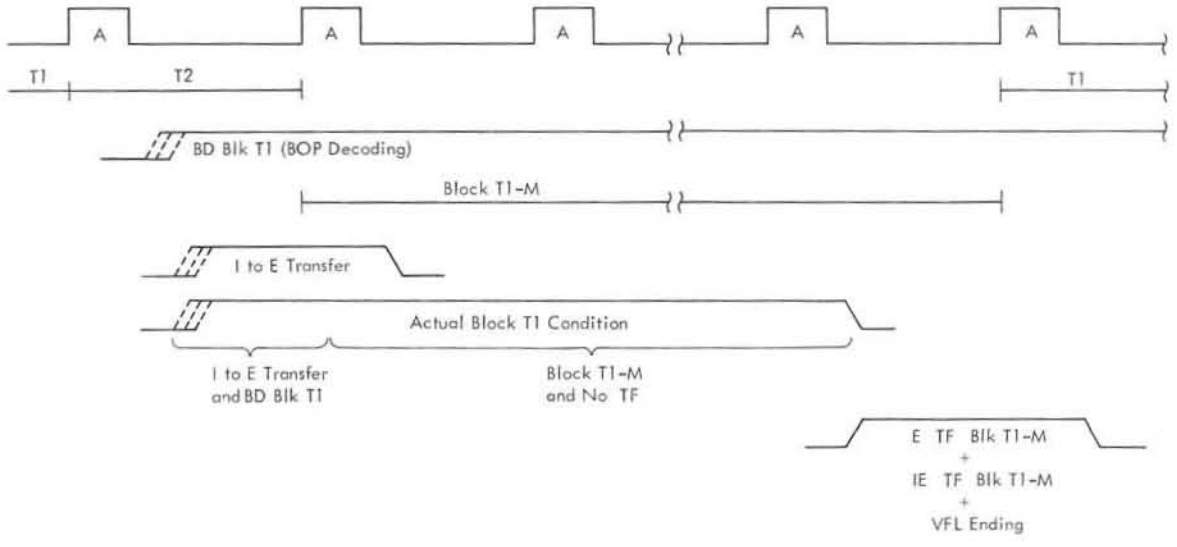


FIGURE 46. BLOCKING OF T1

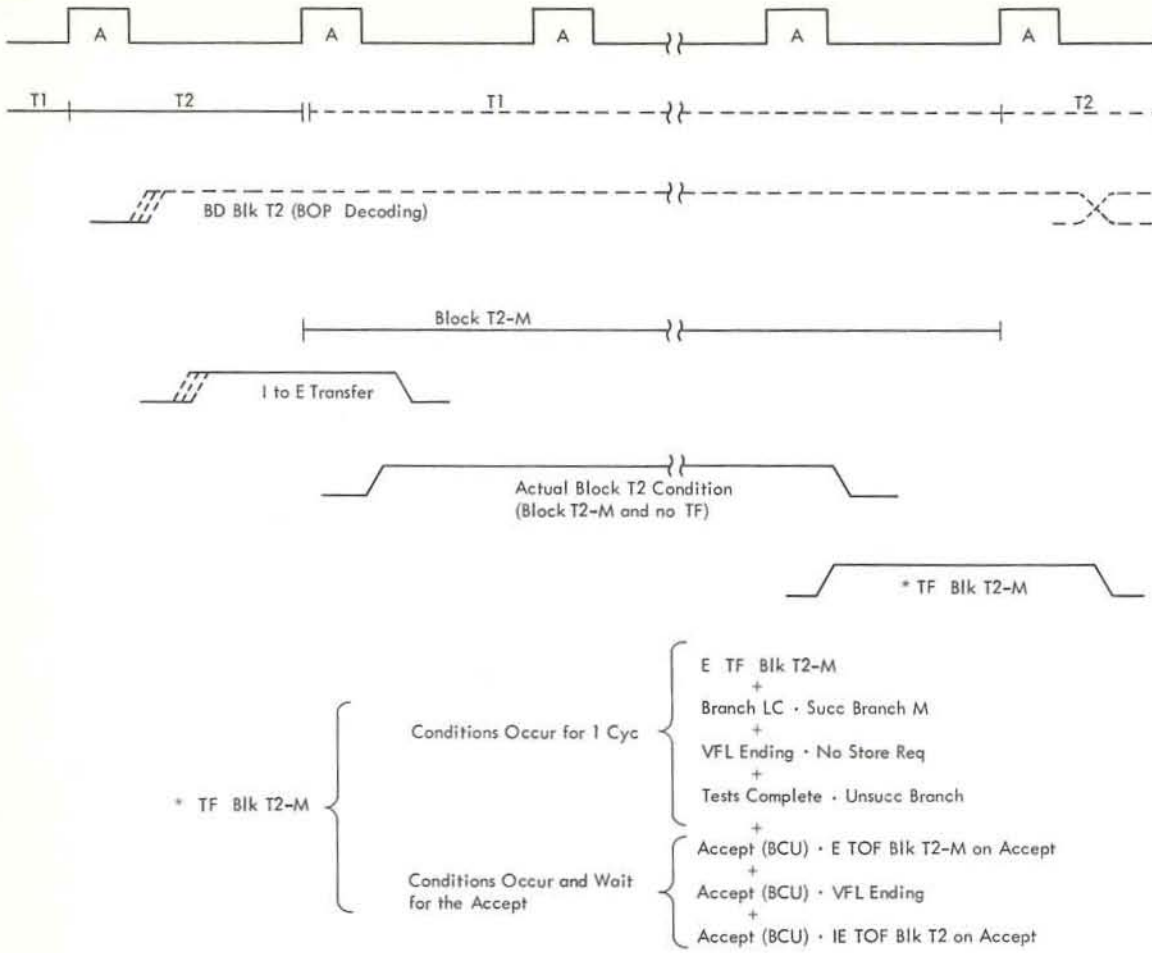


FIGURE 47. BLOCKING OF T2

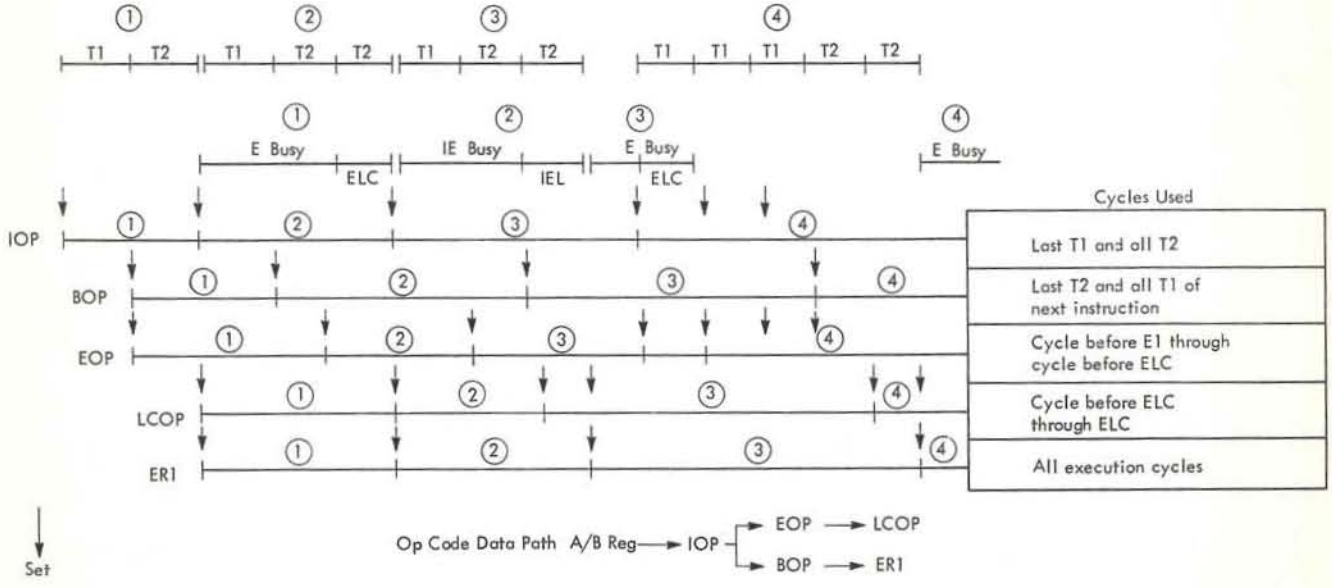


FIGURE 48. SETTING OF OPERATION REGISTERS

from IOP to fulfill these conditions under all circumstances by the use of four set timings as shown on Figure 5258.

The set timed by E not busy ensures a correct set when the preceding instruction was not an E execute or when the preceding instruction was an E execute but ELC occurred before T1 came on. The set timed by the ELC latch ensures a correct set when ELC and T1 coincide. These two sets require the T1 latch line so that EOP will not be set at the end of a last T2 cycle when IOP will be changing. The set timed by set ELC gives a proper set when the preceding instruction was an E execute for which ELC coincides with the last T2 cycle of the instruction about to be executed. The fourth set timed by set put-away (set PA) trigger is used on certain operations where set ELC is data dependent and may come up too late in the cycle to be used to set EOP because of circuit path length. On these instructions set PA always comes on a cycle before set ELC or it coincides with set ELC.

LCOP: Must contain the operation code of any instruction to be executed by the E unit from one cycle before ELC through ELC. These conditions are fulfilled by setting LCOP from EOP using two different sets as shown on Figure 5259.

The first set takes care of all situations where the first E cycle of the execution is not directly preceded by an ELC. The second set takes care of the situation when E1 follows ELC.

ER1: Is set from BOP at every I to E transfer. It, therefore, contains the R1 field of any instruction during all execution cycles. ER1, however, is incremented on the load and store multiple instructions under IE unit control. This is done to direct put-away to general registers and to determine when the last register has been loaded or stored. The logic for setting ER1 is shown on Figure 5260.

T1 and T2 Cycle Automatic Functions

- T1 cycles compute operand storage addresses and when required initiate fetches.
- T2 cycles deliver operands from the registers to the E unit.
- Loose decoding allows T1 and T2 functions to be performed when they are not needed.

With the three major IS control signals directing T1 and T2 cycles and the start of executions, and with all operation registers set so that they contain the required instruction, the I unit's job of delivering operands is accomplished automatically for every instruction.

On Every T1 Cycle: Decoding from IOP directs the selected data to the addressing adder. The adder generates a sum and makes it available at SAR and the H register at the end of each T1 cycle. Only on the last T1 cycle, as signaled by TN T2, is the adder output set in H. Only if IOP decode indicates the quantity generated is to be used as a storage address is it set in SAR. This allows SAR to be set with an instruction fetch address on all cycles not requiring its use for an operand address. The logic that accomplishes this T1 operation is shown on Figure 5261.

On the last T1 cycle decoding from IOP initiates a fetch request to BCU, if required. At A clock of the next cycle (first T2), the operand fetch (OPF) trigger is set. OPF maintains the request to BCU until it is accepted. At the same time, and only if the adder output is to be used, decoding from IOP causes the adder error checking circuits to be sampled. The logic for making the fetch and causing the AA error to be sampled is shown on Figure 5262.

On Every T2 Cycle: Selected addressable registers (GPR or FPR) are gated to the RBL in the E unit as shown on Figure 5264. Note that for all operation codes, except floating-point, two GPRs are gated to RBL. On floating-point operations, a selected FPR is gated to RBL on every T2 cycle and on every E cycle (after I to E transfer) until the E unit turns off FLOUT.

Loose Decoding: A principle that is used throughout the 2075 is illustrated by the gatings that occur on T1 and T2 cycles. Loose decoding occurs during T1 cycles of all SS instructions. On SS instructions all operand storage addresses are calculated during execution cycles. On T1 cycles, however, the register designated by the B1 field of the instruction and the D field of IOP are gated to the AA. The sum generated is set in H at TN T2. The contents of H are not used. The decoding that brings this about is useful on other instructions and does no harm on SS instructions. Stopping the unnecessary transfers on SS instructions would require additional logic and serve no useful purpose.

Start Execution Units

- I to E transfer times the start of all executions.
- I go (same timing as I to E transfer) sets the first IE unit sequencers.

- E go and enable first E cycle (same timing as I to E transfer) set the first E unit sequencers.

- First branch unit sequencer is set at TN T2 at least one cycle before start of branch executions.

With the setting of the operation register and the delivery of operands taken care of, the required execution unit or units may be started. Figure 5265 shows how this is accomplished for all instructions.

Execution is started by setting a selected sequencer which defines the first cycle of the instruction to be executed. The sequencer is selected by decoding from the operation registers, and the set is timed by I to E transfer. The I to E transfer line is not used, however, in setting the IE or E unit sequencers. The proper timing is achieved for the IE first cycle sequencer by setting the sequencer with I go which is timed by the same conditions that time I to E transfer. The proper timing is achieved for the E first cycle sequencer by using E go and enable first E cycle to time the set. All conditions for I to E transfer are contained in these two lines. Enable first E cycle originates in the E unit and contains conditions required for I to E transfer that are sometimes not available until late in the cycle. By ANDing them with E go in the E unit, and using the output of the AND to set the sequencer, the long circuit path to the I unit and back to the E unit is avoided.

The first sequencer for the branch executions is set at TN T2 at least one cycle before the other first cycle execution sequencers. The reason is that the first branch unit sequencer is used to control operation during T2 as well as during the branch execution. This operation is described in detail under branch instructions in 2075 Processing Unit, Volume 3, Field Engineering Manual of Instruction, Form 223-2874.

Keeping Track of Instruction Preparation and Execution

- During T1 and T2. ICR holds the address of the instruction being prepared.

- I to E transfer changes ICR to the address of the next instruction to be prepared.

- LO address is updated in the GSA during T1.

- LO address is set in GSR at TN T2.

- GSR gates the next instruction to IOP during T2 cycles.

- LO address is set to ICR at I to E transfer.

- If GSA carry, HO address is updated in the incrementer on the cycle after I to E transfer.

- On program interrupts the address of the instruction following the one causing the interrupt is always set to ICR to be stored with the old PSW.

During T1 and T2 cycles the instruction counter register (ICR) contains the storage address of the instruction being prepared. At the end of preparation (I to E transfer), the ICR is changed to contain the address of the next instruction to be prepared. The mechanisms used to do this are the gate select adder (GSA), the gate select register (GSR), and the incrementer (Incr) as shown on Figure 5266.

GSA: During normal processing the GSA always receives two inputs: the low-order positions of the ICR (20-22) and the instruction length of the instruction being prepared which is decoded from IOP. At TN T2 the output of the GSA is set in the GSR. The complete logic for all sets of the GSR is shown on Figure 5267.

GSR: In addition to storing the updated value of the ICR, between the time that it is generated in the GSA (TN T2) and the time that it must be set in the ICR (I to E transfer), the GSR is used to gate the proper instruction from the instruction buffers to IOP. The ungated output of the GSR controls the gates from the instruction buffers to IOP. Normally each instruction is, therefore, available at the input to IOP for at least one cycle before it is set in IOP.

Incrementer: As shown on Figure 5266, the incrementer is used in updating the ICR only when the GSA gives a carry out as it delivers the low-order bits to the GSR. The new high-order value for the ICR is generated in the incrementer on the cycle following I to E transfer and is set to the ICR at the beginning of the next cycle. The use of the incrementer does not interfere with its use in the generation of instruction fetch addresses. The incrementer can generate the proper value for a high-order advance and for an instruction fetch both on the same cycle. Some instruction executions use the incrementer as a data path during execution. On these instructions, the incrementer may not be used until at least one cycle after I to E transfer.

On Program Interrupts: The value of the ICR is stored with the PSW. The updating of the ICR is controlled by interrupt circuits so that the value stored is always the address of the instruction following the instruction on which the interrupt occurred. This necessitates

that the updating of ICR be blocked when an E time interrupt occurs simultaneously with an I to E transfer. I time interrupts are serviced at I to E transfer and in this case the normal updating of ICR must be allowed to take place.

CONTROL OF INSTRUCTION FETCHES

- Instructions in advance of the one being processed are normally available in A-B registers.
- IOP loaded signals the availability of the next instruction.
- An empty instruction buffer register (A-B) is recognized at TN T2.
- Conflict between instruction fetching and T1, T2, or execution cycles of an instruction blocks instruction fetching at TN T2 of the conflicting instruction.
- The address for the next necessary fetch to A-B is automatically generated and set in SAR if the instruction fetches are not blocked.
- An instruction fetch is made if either A or B is empty and instruction fetches are not blocked.
- Instruction fetching is given priority over instruction processing only if both A and B are in danger of being emptied.
- Detecting the need for a recovery causes both A and B to be filled, starting at the address in the ICR.

The instruction counter fetch controls (IC controls) are introduced by eight figures. Figure 49 shows how the IC controls fit into the 2075, states the main objectives of the IC controls, and references Figures 5268, 5269, 5270, 5271, 5273, and 5274 that show how the major objectives of the IC controls are accomplished.

THEORY OF OPERATION

Included in this chapter are flow diagrams that cover instruction preparation for every instruction and detailed discussions of the instruction sequencing controls and instruction fetching controls.

I TIME FOR ALL INSTRUCTIONS

- Every execution starts at I to E transfer.
- For any particular instruction, the conditions affecting execution are always the same at I to E transfer.
- Variations in the preparation sequencing have no effect upon the conditions existing at I to E transfer.

For each instruction a flow diagram in the 2075 Processing Unit, Field Engineering Maintenance Diagram Manual, Form 223-2876 shows all things done during preparation specifically in support of execution. The following list of all instructions references the preparation flow chart for each:

Name	Mnemonic	Figure
Add	AR	6153
Add	A	6152
Add Decimal	AP	6154
Add Halfword	AH	6152
Add Logical	ALR	6153
Add Logical	AL	6152
Add Normalized (Long)	ADR	6150
Add Normalized (Long)	AD	6151
Add Normalized (Short)	AER	6150
Add Normalized (Short)	AE	6151
Add Unnormalized (Long)	AWR	6150
Add Unnormalized (Long)	AW	6151
Add Unnormalized (Short)	AUR	6150
Add Unnormalized (Short)	AU	6151
AND	NR	6153
AND	N	6152
AND	NI	6158
AND	NC	6154
Branch and Link	BALR	6377
Branch and Link	BAL	6377
Branch on Condition	BCR	6375
Branch on Condition	BC	6375
Branch on Count	BCTR	6378
Branch on Count	BCT	6378
Branch on Index High	BXH	6379
Branch on Index Low or Equal	BXLE	6379
Compare	CR	6153
Compare	C	6152
Compare Decimal	CP	6154
Compare Halfword	CH	6152
Compare Logical	CLR	6153
Compare Logical	CL	6152
Compare Logical	CLI	6162
Compare Logical	CLC	6154

Name	Mnemonic	Figure
Compare (Long)	CDR	6150
Compare (Long)	CD	6151
Compare (Short)	CER	6150
Compare (Short)	CE	6151
Convert to Binary	CVB	6152
Convert to Decimal	CVD	6156
Diagnose	---	6174
Divide	DR	6161
Divide	D	6159
Divide Decimal	DP	6154
Divide (Long)	DDR	6150
Divide (Long)	DD	6151
Divide (Short)	DER	6150
Divide (Short)	DE	6151
Edit	ED	6154
Edit and Mark	EDMK	6154
Exclusive OR	XR	6153
Exclusive OR	X	6152
Exclusive OR	XI	6158
Exclusive OR	XC	6154
Execute	EX	6375
Halt I/O	HIO	6156
Halve (Long)	HDR	6150
Halve (Short)	HER	6150
Insert Character	IC	6152
Insert Storage Key	ISK	6154
Load	LR	6153
Load	L	6152
Load Address	LA	6167
Load and Test	LTR	6153
Load and Test (Long)	LTDR	6150
Load and Test (Short)	LTER	6150
Load Complement	LCR	6153
Load Complement (Long)	LCDR	6150
Load Complement (Short)	LCER	6150
Load Halfword	LH	6152
Load (Long)	LDR	6150
Load (Long)	LD	6151
Load Multiple	LM	6169
Load Negative	LNR	6153
Load Negative (Long)	LNDR	6150
Load Negative (Short)	LNER	6150
Load Positive	LPR	6153
Load Positive (Long)	LPDR	6150
Load Positive (Short)	LPER	6150
Load PSW	LPSW	6168
Load (Short)	LER	6150
Load (Short)	LE	6151
Move	MVI	6170
Move	MVC	6154
Move Numerics	MVN	6154
Move with Offset	MVO	6154
Move Zones	MVZ	6154
Multiply	MR	6161
Multiply	M	6159
Multiply Decimal	MP	6154
Multiply Halfword	MH	6159
Multiply (Long)	MDR	6150
Multiply (Long)	MD	6151
Multiply (Short)	MER	6150
Multiply (Short)	ME	6151
OR	OR	6153
OR	O	6152
OR	OI	6158
OR	OC	6154
Pack	PACK	6154
Read Direct	RDD	6172
Set Program Mask	SPM	6166
Set Storage Key	SSK	6165
Set System Mask	SSM	6175
Shift Left Double	SLDA	6155
Shift Left Double Logical	SLDL	6155
Shift Left Single	SLA	6155
Shift Left Single Logical	SLL	6155
Shift Right Double	SRDA	6155
Shift Right Double Logical	SRDL	6155
Shift Right Single	SRA	6155
Shift Right Single Logical	SRL	6155
Start I/O	SIO	6156
Store	ST	6157
Store Character	STC	6157
Store Halfword	STH	6157
Store (Long)	STD	6160
Store Multiple	STM	6171
Store (Short)	STE	6160
Subtract	SR	6153
Subtract	S	6152
Subtract Decimal	SP	6154
Subtract Halfword	SH	6152
Subtract Logical	SLR	6153
Subtract Logical	SL	6153
Subtract Normalized (Long)	SDR	6150
Subtract Normalized (Long)	SD	6151
Subtract Normalized (Short)	SER	6150
Subtract Normalized (Short)	SE	6151
Subtract Unnormalized (Long)	SWR	6150
Subtract Unnormalized (Long)	SW	6151
Subtract Unnormalized (Short)	SUR	6150
Subtract Unnormalized (Short)	SU	6151
Supervisor Call	SVC	6163
Test and Set	TS	6493
Test Channel	TCH	6156
Test I/O	TIO	6156
Test Under Mask	TM	6162
Translate	TR	6154
Translate and Test	TRT	6154
Unpack	UNPK	6154
Write Direct	WRD	6173
Zero and Add	ZAP	6154

INSTRUCTION SEQUENCING CONTROLS

T1 Cycle

The basic function of the T1 cycle is to gate to the addressing adder those fields required to form an effective address. See Figures 5261 and 50.

TN T1

If an instruction belongs to the class of instruction for which T1 of the next instruction may cause interference with execution of the first instruction, T1 of the next instruction is always blocked. At the I to E transfer of the first instruction, the trigger block T1-M is set if the instruction belongs to the above class. This blocks T1 of the next instruction until the unit executing the first instruction generates a signal that cancels the block. Similarly, if an instruction belongs to the class of instruction for which T2 of the next instruction may cause interference with execution of the first instruction, the T2 of the next instruction is blocked. A trigger block T2-M is operated in a method analogous to block T1-M. See Figures 51 and 52.

The T1 cycle is always taken as the first cycle of an instruction. It is also performed if no effective address is required. In some instructions, it procures only a single field to be used as an address. All fields are obtained from the general register(s) and/or the IOP register.

The turn-on for T1 called TN T1 is composed of many logic lines. The usual case is to turn it on with the I to E transfer of the previous instruction. Before the I to E transfer occurs, BOP decoding examines the class of instruction and determines if blocking of T1 of the next instruction is required. If BOP decodes, blocking of T1 is required (BD block T1) then the I to E transfer is not allowed to turn on T1, but instead turns on the block T1-M trigger. This trigger maintains the blocking during successive cycles. When the E time has progressed to the point of no longer requiring this block, it will generate a signal to turn-off the block T1-M trigger. This signal is also used to generate the TN T1 condition.

T1 may also be turned on by $\overline{\text{BLK T1-M}}^{L}$. This is the turn-on which allows the system to start up after it has been manually halted.

Instructions That Generate Blocking of the Next T1 Cycle:

Instructions	Condition
All SS Instructions--SPM, SSK, MR, DR, D, BXH, BXLE, and STM	Require gating out of general registers during the E time of the instruction. The execution unit is required to turn off the block trigger when this gating is no longer required.
All I/O Instructions	The channel and unit addresses are sent to the I/O devices from the H register. Blocking of T2 would prevent H from being changed; however, the release line from the channel (used to end the instruction) could extend into the next instruction and, if it were another I/O instruction, cause permature termination. This may occur because the release line is multiplexed and uses slow circuits. Thus, T1 is blocked to allow the extra start up cycle.
LA	The effective address must be saved in the H register until the E unit can transfer it to a general register. Although blocking of T2 could accomplish this, packaging requirements caused T1 to be used. No additional delays resulted.
LM	In the LM instruction many general registers are loaded. The compare block is not capable of detecting a compare from more than two registers.
LPSW	In LPSW the ICR is changed. Thus, new instructions will need fetching to the buffers before processing can continue.
DIAG	Until the MCW is loaded and the error status lines have settled down, any further instruction execution must be blocked.
SVC	This instruction requires blocking of T1 to prevent the TN T1 from changing IOP. IOP is required during the setting of the interrupt code into the PSW.

The program halt trigger can also prevent TN T1. This trigger, located in the maintenance console, is

used for the maintenance functions: manual halt, single operation, etc. An output of the program halt trigger is latched on the sequence control board and deconditions TN T1.

Set IOP (Figures 5256 and 52)

The IOP register contains two instruction halfwords. The first halfword contains an operation code and, for multiple halfword instructions (all but RR), the second halfword is used for addressing. Handling of the third halfword, required by SS instructions, is described in the "Instruction Fetching" section. By means of the gate select mechanism any two successive halfwords of the eight halfwords contained in the instruction buffers (A-B registers) are selected and sent to the IOP register. The proper selection is normally made during the previous T2 cycle. The gate select mechanism causes the length of the instruction as decoded from the first two bits of IOP to be added to the ICR and this value to be stored in the gate select register with TN T2. The output of the gate select register selects the gate for IOP. With the I to E transfer, the gate select register is sent back to the ICR thus updating the ICR.

The T1 function can be accomplished in one machine cycle. However, if the conditions to generate TN T2 do not exist the T1 cycle continues to repeat itself until TN T2 occurs. IOP is set to the gated contents of A-B at the beginning of every T1 cycle. The logic line TN T1 sets IOP for the first T1 cycle. If T1 repeats itself, the set IOP occurs at the beginning of each machine cycle until TN T2 occurs. There is one exception to the later set condition. During the execute instruction, the setting of IOP has taken place. This special block is represented by the expression:

$$\overline{\text{IOP LOADED}^{L} \cdot \text{XEQ SEQ}^{L}}$$

A special set $\text{SSOP}^{L} \cdot \text{VFL ADR}^{L}$ is also provided for setting IOP on SS instructions; therefore, the total expression for setting IOP is:

$$\text{TON T1} + \text{T1}^{L} \cdot \overline{\text{TON T2}} \cdot \text{Not (IOP LOADED Lth} \cdot \text{XEQ SEQ Lth)} + \text{SSOP}^{L} \cdot \text{VFL ADR}^{L}$$

I time (that is, T1) may start even though the selected instruction has not yet returned to the instruction buffer. This could occur, for example, after successful branches or recoveries. Since T1 cannot be turned off (see "TN T2") until the selected instruction has returned to IOP, there will be at least one T1 cycle during which IOP has contained the correct instruction.

The selected instruction will arrive at IOP during the same cycle that it returns to A-B since set IOP has occurred for each T1 cycle. Note that if the selected instruction is contained entirely within one register, waiting is required only until this register is loaded. The timing of A-B in relation to IOP and the problems encountered in single cycle are discussed in the "Instruction Fetching Controls" section.

Effective Addressing (Figure 5261)

The T1 cycle conditions input gating to the addressing adder for up to three fields. The fields determined by the instruction are selected by IOP. The contents of general register R2 (X2) are gated to the addressing adder via GBR, and the contents of B2 are gated via GBL. D2 is gated directly to the addressing adder from IOP. The general register buses have a width of 32 bits plus parity. However, only 24 bits participate in the add. The D field has a width of 12 bits. When general register 0 is specified as one of the fields, that adder input receives all zero data with a correct parity for that field. The instructions in the RR format are exceptions in that they make no distinctions for GR 0.

The effective address formed has a width of 24 bits. With the TN T2 it is always gated into the H register, and may also be sent to SAR. When the effective address is sent to SAR, the error checking of the AA is sampled. All of the preceding conditions are decoded by IOP. The RS shift instructions, which use the effective address only as a shift amount, are unique in that they sample the AA error but they do not send the result to SAR.

T1 Cycle Additional Functions

The T1 control trigger has outputs used in other areas of the machine, for example, it is used during the maintenance feature single operation to turn-on the block T1-M trigger and it is also used by the E unit for setting EOP.

The T1 cycle during which TN T2 occurs can be considered the good T1 cycle. For the TN T2 to occur every condition for the T1, functions must be available. In addition, every blocking condition that prevents the second part of I time must have been removed.

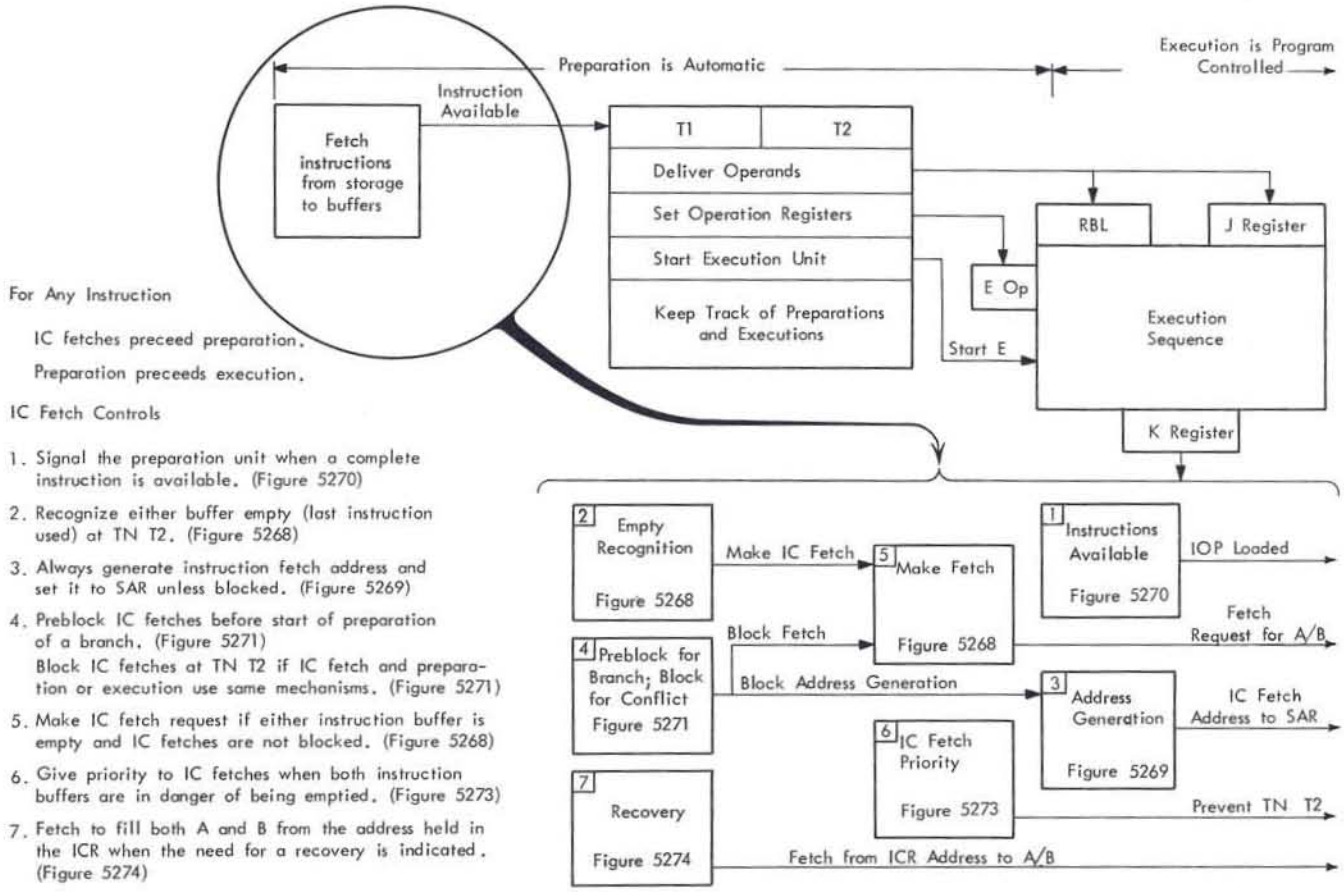


FIGURE 49. INSTRUCTION COUNTER FETCH CONTROLS

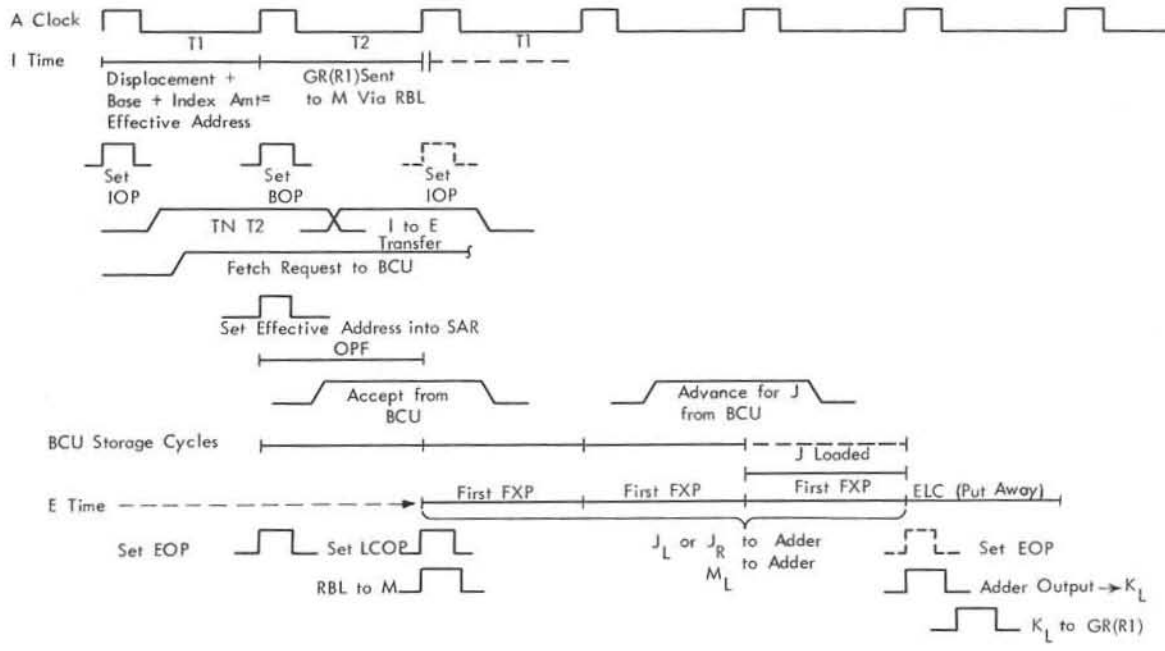


FIGURE 50. I TIME AND E TIME FOR AN RX-FXP-ADD (FULL WORD)

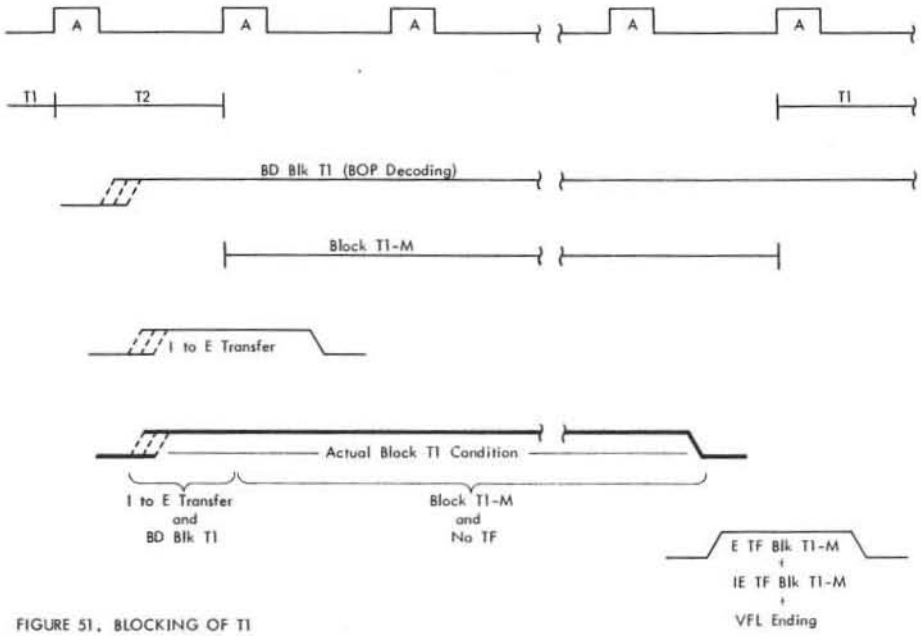


FIGURE 51. BLOCKING OF T1

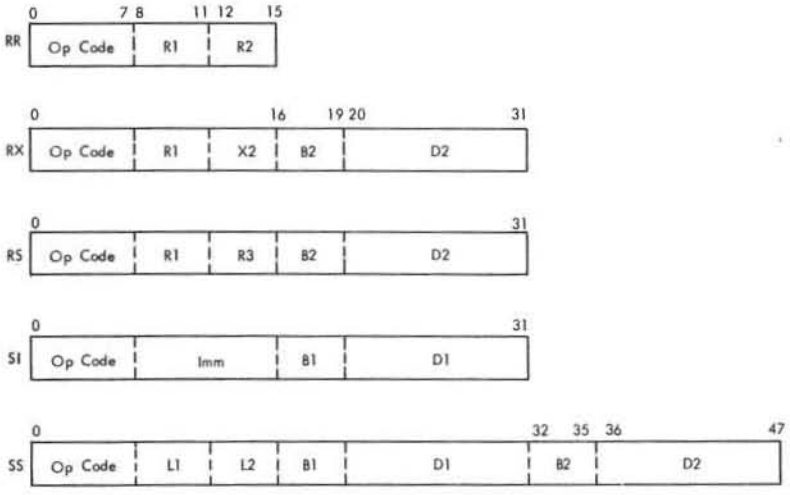


FIGURE 52. INSTRUCTION WORD FORMATS

TN T2

The following are the conditions necessary to generate the logic level TN T2.

IOP Loaded: The IC controls generate this condition if the selected gate (from A-B to IOP) is pointing to a loaded instruction in the A-B register(s). See Figure 5270.

NO COMPARE BLOCK + $\overline{\text{E BUSY}}$: Buffer operation register (BOP) decodes those E unit instructions that will perform put-aways to the general registers. A compare occurs when the put-away is to any of the general registers used by T1 of the next instruction. T1 must subsequently wait until the new value has been put-away. This waiting is accomplished by blocking TN T2. The block exists until the E busy trigger is turned off. See Figure 5255.

No Instruction Fetch Priority: When the processing has emptied one instruction buffer and is approaching the point of emptying the second buffer (without having allowed the first to be reloaded), the IC controls may generate a block TN T2 until an IC fetch is made. This gives the IC controls access to SAR. (See "Instruction Fetching Controls" section and Figure 5273.)

$\overline{\text{BLOCK T2-M}}$ + TOF BLK T2-M: This condition is available if the previous instruction processing has not set the BLOCK T2-M trigger or the execution unit is turning it off.

Block T2-M (Figures 53 and 5254)

The need for blocking of T2 is determined in the same manner that T1 blocking is determined. BOP decoding generates BD block T2. This decode line and the I to E transfer turns on the block T2-M trigger. Since the TN T2 cannot occur with the I to E transfer, no additional logic is needed to anticipate the block T2 as was required for blocking T1.

Some instructions will generate both BD block T1 and BD block T2 lines. In these cases both block triggers are set. When the processing no longer requires the blocking of T1, the execution unit turns

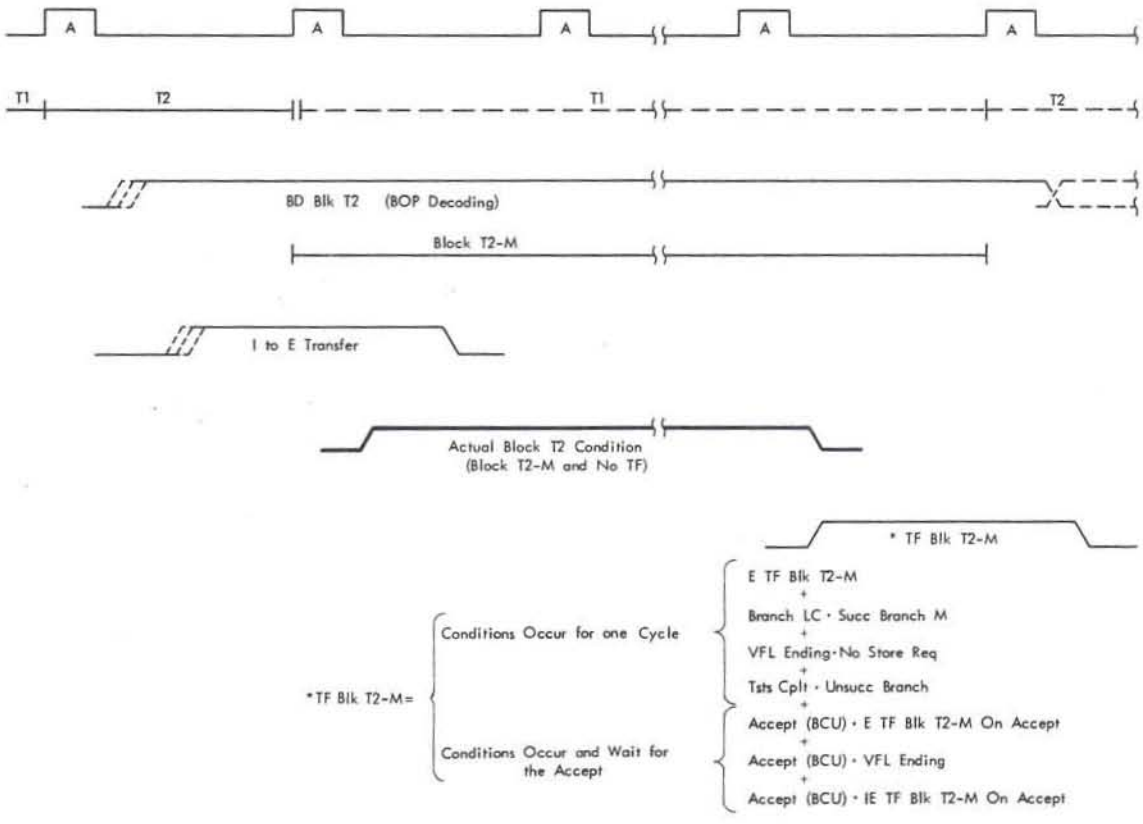


FIGURE 53. BLOCKING OF T2

it off. Processing continues and the overlap is started on the next instruction; however, T2 is blocked until the execution unit turns off block T2-M.

Instructions That Generate BD Block T2:

Instruction	Condition
ISK, ST, STH, STC, CVD, STD, STE, RD, STM, MVI, NI, OI, XI, and all SS instructions	Operand fetches are generated by the execution unit, or operands are to be stored in external storage. For these operand fetches the SAR is required to provide the storage address(es). SAR must not be changed by TN T2 of the overlapped instruction.
LH, CH, AH, and SH	The E unit requires the RBL as part of the data path to expand the fields to 32 bits. T2 of overlapped instructions is blocked to prevent the gating of operands into the RBL.
CDR, ADR, SDR, AWR, SWR, SER, AER, SER, AUR, SUR, CD, AD, SD, AW, SW, CE, AE, SE, AU, SU	The E unit requires the contents of the BOP REG for selection of the FP REG to supply operand 1 exponent on fractions add cycle.
MR, DR, MDR, DDR, MER, DER, and MD	The E unit requires the J register as part of the data path during E time. Overlap of the next instructions could result in a getch of an operand to J if T2 were not blocked.
BCR, BC, BCTR, BCT, BXH, BXLE, BALR, BAL, and EX	For branch instructions the block T2-M trigger is turned on by BRANCH OPL. I to E XFER. This blocks the second part of I time on overlapped instructions until the branch success is determined. Blocking T2 protects the H register which contains the branch-to-address.

Logic is provided to remove the block T2 condition at the earliest possible time. Each turn-off for the block trigger also enters TN T2 as a conditioning level. The only exception occurs during the execute instruction. In this instruction the block T2-M trigger is turned off one cycle before T2 is allowed to turn on. In this case the latched output of block T2-M maintains the block for this cycle.

When blocking is employed to protect SAR, the execution unit conditions the turn-off of block T2-M with the control line TF block T2-M on accept at the appropriate time during instruction execution. This control line and accept (from BCU) turns off block T2-M. See Figure 53.

If the blocking was protecting the J register, the control line E TF block T2 is generated by the E unit to turn off block T2-M. This line is always generated during the last cycle of relevant instructions; and if the system is not in single-cycle mode, the line is also generated a number of cycles before J is free. T2 can be allowed to turn on earlier because any operand fetches it may initiate would not return to J for at least four cycles. The turn-off may be anticipated for up to three cycles before J is free.

If RBL usage required blocking, the E unit generates TF block T2-M when this usage is no longer required.

Compare Block (Figure 5255)

A compare block condition with the E busy trigger on causes a deconditioning level to TN T2. If the previous instruction is doing a put-away into a general register which is being used to compute an effective address during T1 of the next instruction, then T1 of the next instruction must be held up until the operands have been put-away. This condition is detected by comparing both IX2 and IB2 to BR1. IX2 and IB2 contain the address of the general registers which could be used to form the T1 cycle effective address, and BR1 contains the address of the put-away general register for the previous instruction.

BOP remains unchanged from the previous instruction until TN T2 of the next instruction and therefore provides the operation code of the previous instruction during T1 of the next instruction. On the basis of BOP, it is determined whether or not the previous instruction requires a put-away to the general registers. It is also determined if there is a pair of put-aways to an even/odd pair of general registers.

If BOP indicates a single put-away (BD compare request) then BR1 is compared to IX2 and IB2 according to their usage in the effective address. IX2 (or IB2) is not used in the effective address if it references GR 0. If BOP indicates a double put-away to an even/odd register (BD double register compare), then the same compares are made as for a single put-away; but the low-order bit is ignored in the compare. If either compare is satisfied and if E busy is on, then compare block prevents T2 from turning on.

All put-aways to the general register are completed before the end of the last cycle of an instruction. The usage of E busy in compare block ensures that if a compare situation occurs, at least one correct effective addressing cycle is taken after the execution of the previous instruction is completed.

In the discussion of T2 no line comparable to compare block is required in the gating of internal operands to the E unit. No block is necessary because all put-aways are completed by B time of ELC. In the remaining half-cycle, any general register may be gated to the E unit before the I to E transfer.

TN T2 Functions Controlled by IOP

The TN T2 indicates a good T1 cycle. In addition to providing the set condition for the T2 cycle, it performs many functions. IOP decoding steers the

TN T2 condition. The following lines are described with respect to their functions at TN T2.

ID AA to SAR: With TN T2 the effective address formed by the AA is gated from the AA latches into SAR.

ID Sample AA Error: Tests the error checking lines of the AA during TN T2. If an error occurs, it causes a machine check condition.

ID Fetch Class: Indicates that operand fetching is required. With TN T2 it initiates the request to BCU. See "Operand Fetching."

ID Block IC: At TN T2 this forms the third blocking condition. It prevents IC fetches from interfering with instruction processing. It also forms the TN for block IC-M, a control trigger used to maintain the IC blocking condition.

ID Floating Point: Used in the controls that gate out the floating-point registers. It sets the FLOUT trigger with TN T2.

ID RR: Used during TN T2 to set the FR2 trigger. FR2 determines the order in which floating-point operands are gated to the E unit. See the "T2 Cycle."

ID SS: ID SS conditions the sequencing controls to process the SS format instructions by turning on the SSOP control trigger with TN T2.

ID Branch Operation: Sets the branch operation control trigger with TN T2. Branch operation is a status trigger used during branches. ID branch operation is also used to override any blocking of TN T2 generated by the IC controls.

TN T2 Additional Functions

In addition to those functions decoded by IOP, TN T2 also conditions the following functions.

Setting the Gate Select Register: During T1 the instruction in IOP is examined to determine its length (in halfwords). This length is added to the ICR low order. With TN T2 the adder output is gated into the GSR. Gate selection now changes and a new gate for A-B to IOP is selected. The next instruction is thus gated to the IOP input.

Setting of IOP: Since TN T2 indicates a good T1 cycle, the TN T2 logic line deconditions the setting of IOP from the T1 triggers latched output.

Setting of BOP: The setting of BOP usually occurs with TN T2. For a detailed description, see the "Buffer Operation Register" section and Figure 5257.

Instruction Buffer Empty Condition: When the last instruction located in one of the A-B registers has been selected and set into IOP, TN T2 indicates the empty condition to the IC fetch controls and turns off the appropriate loaded trigger. See Figure 5268.

IOP Error: TN T2 is used to sample for a parity error on bits 8-15 of IOP. If an error exists, the trigger IOP error is turned on.

Interrupts: If the instruction being executed is from an invalid storage address, if bit 23 of the address of the instruction is 1, or if the operation code of the instruction is invalid, then I program interrupt is set at TN T2. The ID code for the type of interrupt is set into the PSW with TN T2.

H Register: The output of the addressing adder is set into H at TN T2.

Buffer Operation Register (BOP) (Figure 5257)

The buffer operation register is a 12-bit operation register set from IOP 0-11. By taking advantage of the way E time and I time are overlapped, three areas are able to share usage of BOP. BOP is used by T2 during the cycle of the I to E transfer, and it is used by branch executions and IE executions during E time.

Branch executions and IE executions have been implemented in a way such that no operation code is required by the execution unit during the last cycle. Since the I to E transfer of the next instruction cannot occur until the end of the previous instruction, there is a time to set BOP which accomplishes all of the following:

1. BOP is not updated until one cycle after IOP is correctly set.
2. BOP is updated at least one cycle before the I to E transfer.
3. BOP is available to the branch and IE units for the cycle before the I to E transfer and for every execution unit cycle except the last cycle.
4. No performance penalty is taken because of the shared usage of BOP.

The specific logic implementation to control the setting of BOP is:

BOP is set at TN T2 if the IE busy trigger is off, indicating that neither the IE unit nor the branch unit are operating. BOP is set at TN T2 if either IE unit last cycle (IEL) trigger or branch unit last cycle (BRLC) trigger is on, indicating that the relevant unit is completing execution. These set conditions

are not sufficient to ensure that BOP is set at least one cycle before an I to E transfer if the previous instruction is an I execution or an unsuccessful branch. In the case of IE unit execution, TN-IEL (TN T2 + $T2^L$) is used to obtain a further set condition for BOP. Thus, BOP is set one cycle early even if IEL causes an I to E transfer on the next cycle.

For unsuccessful branches, T2 can be turned on with the turn-on of BRLC but not before. Therefore, TN T2 is ANDed with the unsuccessful branch turn-on of BRLC to obtain another set for BOP. Thus, for unsuccessful branches BOP is set one cycle early even if BRLC causes an I to E transfer on the next cycle. For successful branches, T2 is never turned on until BRLC is turned off. Therefore, no special set is required for successful branches.

The logic statement is:

Set BOP = TON T2 ($BRLC^L$ + IEL^L + $\overline{IE\ BSY}^L$ + TON IEL + Unsuccessful Branch Turn-On of BRLC) + $T2^L$ · TON IEL

BOP decoding is used during the T2 cycle to specify which internal registers are to be gated to the RBL as operands, the execution unit(s) required to perform the E time of the instruction, and at what point overlap of the next instruction is to be blocked, if necessary. BOP decode lines also define many functions of the store instructions. A list of the BOP decode lines is shown on Figure 9068. When BOP is still maintained during T1 of the next instruction it allows usage in generating the block of TN T2 called compare block.

T2 Cycle

The TN-T2 logic level indicates that the T1 cycle has been properly performed and thus initiates the T2 cycle. The T2 cycle has as one of its functions the gating of internal operands. It is this function that required placement of T2 near the general registers. T2, as well as T1, are located on the G-C2 board.

Internal Operand Gating (Figure 5264 and 50)

Internal operands are gated to RBL, a 64-bit latch located in the E unit. RBL is latched during every L clock and unlatched during every not L clock. Operands from the general registers are sent via GBL or GBR. If the instruction is in the floating-point class, operands are gated from the floating-point registers.

General register operands are gated to RBL by T2 if IOP2 = 0 (that is, not FP). Operands are gated simultaneously to RBL by T2 on general bus left (GBL) and general bus right (GBR). General register BR1 is gated to GBL and either general register IR2 or general register BR1 + 1 is gated to GBR. BR1 + 1 is gated to GBR for shift and RX instructions and MR. For all other instructions, IR2 is gated to GBR. The execution unit ignores those operands gated by T2 which are not needed to complete the instruction.

Certain instructions require that gating be continued after the I to E transfer or that additional operands be sent. This gating is accomplished by the control triggers GROUT or FLOUT. For a detailed description of GROUT, see the "GROUT" section. FLOUT is set with TN T2 for any instruction in the floating-point class regardless of format. It opens the gate from the floating-point registers to the RBL. Floating-point register gating is performed on the full 64-data bits of the registers. Thus, each addressed register fills the RBL. For instructions in the RR format, this full gating means that the two operands must be sent sequentially. With TN T2 and ID RR, a control trigger FR2 is conditioned. The FR2 trigger causes floating-point register gating to be addressed by the IR2 field (the second operand). The turn-on for FR2 is maintained during subsequent T2 cycles (if they occur) by T2 and the BOP decode line BD RR. With the I to E transfer, the turn-on FR2 condition is removed. With no turn-on, the FR2 will turn off. When FR2 is off the first operand, addressed by the BR1 field, is selected for gating to the RBL by FLOUT. FLOUT stays on after the I to E transfer and the E unit provides a turn-off when gating is no longer required. Normally, the turn-off is generated during the first E unit cycle. The exceptions are the divide instructions. Divide instructions require prenormalization of the divisor to be completed before the second operand, the dividend, can be accepted. For the divide instructions, block T2-M is set and this prevents the overlap of another instruction from setting the FR2 trigger prematurely. See Figures 5264 and 54.

Incrementer Gating

The effective address formed by T1 is, during branches, the address of the branch-to instruction. Since two instruction buffers are provided within the CPU, two effective addresses are required in order to fill both buffers. H and SAR are set to the effective address by TN T2 which also initiates the first fetch. The second address is formed by adding a control amount (1 in position 20) to the contents of the H register. The T2 branch operation gates the H register and the control amount to the incrementer. With the I to E transfer, the output of the incrementer is sent to SAR. Branch controls initiate the second fetch.

The LM instruction requires that the effective address of H and SAR be updated by one storage word after the first operand fetch has been made. The BOP decode line BD multiple LD gates the contents of the H register and a control amount (+1 in incrementer position 20) to the incrementer. With I to E transfer, the incrementer output is sent to SAR and H.

Any time the output of the incrementer is gated into a register, the checking circuits in the incrementer are tested. If there is an error the incrementer error trigger is turned on. Therefore, the incrementer error line is tested at the I to E transfer of branches and LM.

Operand Fetching (Figures 5261, 5262, and 55)

Most features of a T2 operand fetch request are similar to the features of all CPU fetch requests. If you understand T2 operand fetches, it should be possible to understand any CPU fetch.

If IOP contains an instruction requiring an operand fetch, the TN T2 ID fetch class line generates a fetch request. This turns on the CPU request trigger at A time in the BCU. This trigger in BCU actually initiates storage operation. TN T2 ID fetch class also turns on the operand fetch trigger (OPF) in the I unit controls. Whenever a fetch request is made, SAR must be loaded with the correct fetch address at time of the request (or earlier). For all instructions requiring a fetch during I time, the logic ID gate addressing adder to SAR TN T2 sets SAR from the addressing adder with A clock.

The CPU request trigger in the BCU is normally kept on until the request is accepted. However, if the request line from CPU is cancelled, the CPU request turns off on the next A clock. This feature is very useful, as it enables fetch requests which have not been honored by BCU and which are no longer needed to be cancelled. This ability is used extensively during IC fetching, and is also used to cancel fetch requests when entering an interrupt sequence. In the case of T2 operand fetches, the request line is held up by the output of OPF.

Whenever the CPU request trigger is on, a requested storage is not busy, and the channel does not have priority, then the requested storage is started at about B time. The BCU then generates a line, called accept, which straddles the next A clock. This is distributed throughout the CPU. If OPF is on, accept turns the OPF trigger off, thus dropping the request. The request does not drop early enough to prevent setting of CPU request again at the A clock staddled by accept. The two cycles of BCU prevent any further action being taken by the request, and CPU request is immediately reset with a BR clock.

Accept rises with an early BR clock and remains until the next early B clock. Thus, A clock is straddled by accept even if the clock is being single cycled. This line is used in connection with conventional CPU request logic. There is another accept line, called pulsed accept, which rises at the same time as accept, but which falls on the next early BR clock. Thus, pulsed accept straddles the next AR clock only. Pulsed accept is used primarily in the branch and IC controls.

OPF is also used to indicate that the fetch should be returned to the J register. The unlatched output of OPF is ORed with certain other fetch sequencers and is sent to the BCU. This line is used by BCU to set up a return address register for a return to J. When the storage gates the requested word out onto the SBO, the storage also sends a line called advance to the BCU. The advance and BR clock are used to gate the storage data into the SBO latch. If the appropriate return address register indicates a return to J, the advance is gated out to the CPU as J advance. The J advance AR clock is used to gate the SBO latch data into the J register and to turn on J loaded. The trigger J loaded indicates to the various execution units that the fetch has returned and data is in the J register. J loaded is normally turned off when the data is transferred out of J.

In the standard 2075 configuration, data is returned three cycles after CPU request is set (assuming the storage was available). The storage times out in four cycles and is available at that time for another operation. A fetch of an operand from an odd address (as per address bit 20) can be overlapped with a fetch of an operand from an even address. Fetches from addresses with bit 20 identical cannot be overlapped. Because of the overlapped operation of BCU, it is possible for two operand fetches for two different instructions to be outstanding at the same time. This does not cause difficulty because the relevant execution unit clears out the contents of J before the second fetch returns. At the time the J register is cleared out, J loaded is also turned off and the controls are thus prepared to receive the second fetch. In the few cases where the execution unit cannot clear out J in time (for example, multiply or divide), T2 and the associated fetch of the next instruction are blocked until such time as a J free condition may be anticipated.

I to E Transfer

The I to E transfer is a logic level generated during the last T2 cycle. The principal function of the I to E transfer is to initiate the start of E time. When all the T2 cycle functions have been completed and the E time of the previous instruction has also been

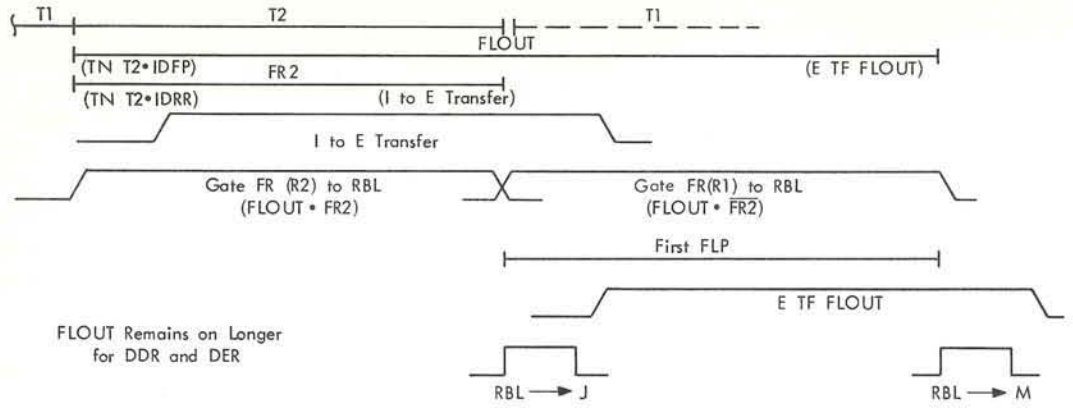


FIGURE 54. RR FLOATING-POINT OPERAND GATING

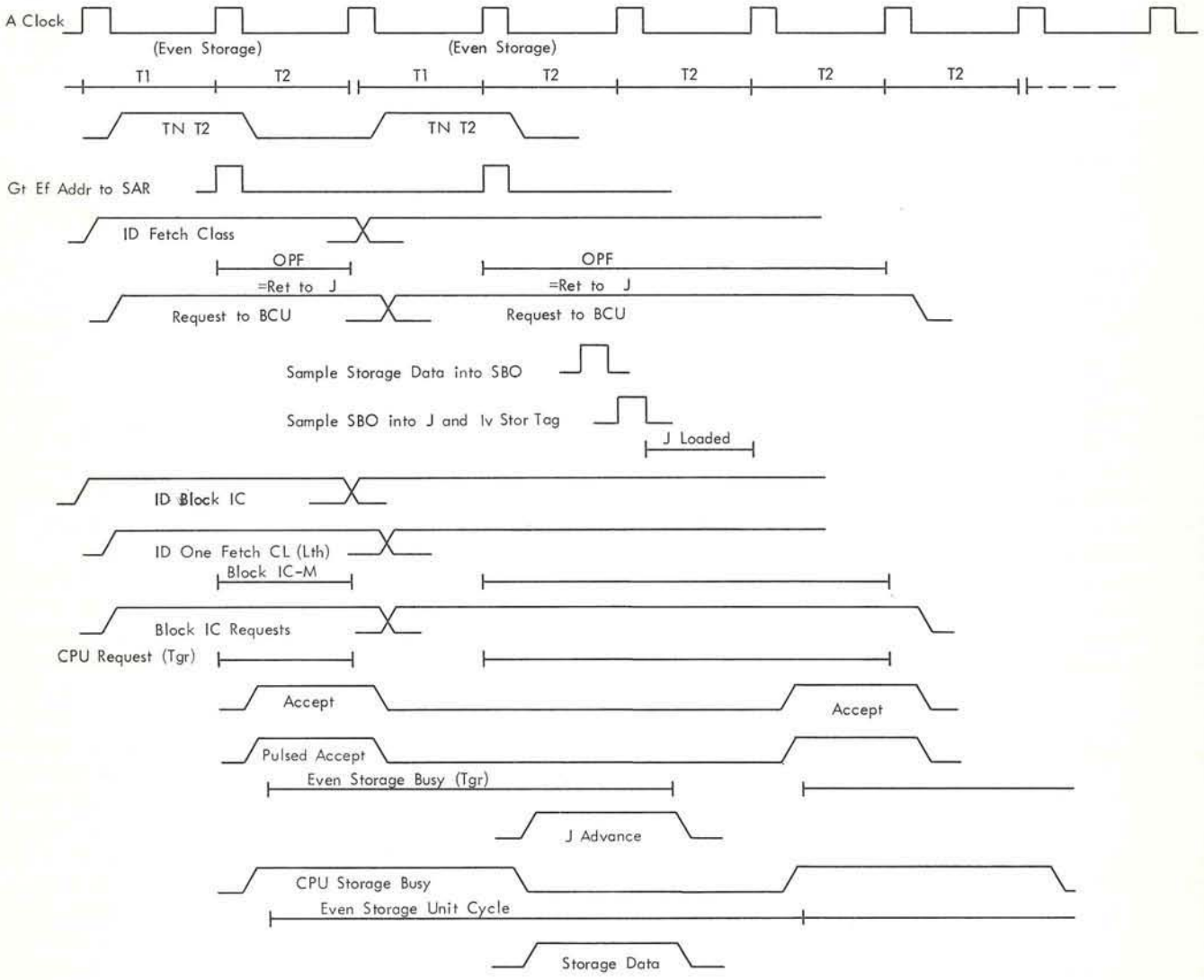


FIGURE 55. OPERAND FETCH TIMING

completed, the I to E transfer occurs. Lines from the interrupt mechanism further condition the I to E transfer. If an interrupt is outstanding, the E time is not started, but instead the interrupt mechanism begins an interrupt sequence.

Generation of I to E Transfer (Figure 5252)

The logic for the I to E transfer is $T2^L \cdot (\overline{OPF}^L +$ ACCEPT) $\cdot$ LCM. The first two quantities indicate that I time of the instruction to be transfered is completed. LCM indicates that E time of the previous instruction has been completed. If OPF is not on, either no operand fetch was required or the fetch request already has been honored by BCU (accept turns off OPF). Since accept enters the logic for the I to E transfer directly, the transfer may be made at the time the request is honored.

Last cycle memorized (LCM) part of the I to E transfer statement occurs when a previous E time reaches its last cycle or when all execution units are idle. The execution units are idle when both E busy and IE busy are off. E busy is turned on at the I to E transfer of any instruction starting the E unit. ELC turns E busy off unless it is being turned on by the next instruction. IE busy is turned on at the I to E transfer of any instruction starting the branch or IE unit. BRLC + IEL turns IE busy off unless it is being turned on by the next instruction.

The last cycle of E time must be allowed to condition the I to E transfer if successive E time without idle machine cycles are desired. Since two execution units may be required to accomplish an instruction's E time, further complications arise in determining the true last cycle of E time. The last cycle logic from each of the execution units cannot alone indicate the true last cycle. Only when the last unit completes its operation, or when both end simultaneoulsy, can a true last cycle be generated.

No instructions require that the branch unit and the IE unit operate simultaneously. It is this fact that allows these units to share the same busy trigger (IE busy). See Figures 56 and 5265.

Instruction	Branch Unit	E Unit	IE Unit
ISK		X	X
SSM		X	X
LM		X	X
STM		X	X
BALR	X (if R2≠0)	X	
BCTR	X (if R2≠0)	X	
BCT	X	X	
BAL	X	X	
BXH	X	X	
BXLE	X	X	

FIGURE 56. INSTRUCTIONS REQUIRING TWO EXECUTION UNITS

Testing for the true last cycle is performed in the following manner. When any unit sends its last cycle, this last cycle is compared against the busy trigger of the other unit. If the busy trigger for the other unit is off, the last cycle condition is the true last cycle. If the other busy trigger is still on, the last cycle of the first unit only provides a turn-off for its own busy trigger. Logic is also provided to test for the simultaneous ending of the two units. The simultaneous occurrence of both last cycle conditions generates a true last cycle.

The following statement defines how the true last cycle is determined:

$$(IEL + BRLC) \cdot \overline{E\ BUSY} + ELC \cdot \overline{IE\ BUSY} + (IEL + BRLC) \cdot ELC$$

The logic level generated by the above function is called control last cycle. It is used in generating LCM and is also sent to the interrupt mechanism. There it is used to indicate that E time is ending so that the interrupt mechansim may test for E time interrupt conditions.

The complete statement for LCM is:

$$LCM = (\overline{IE\ BUSY}^L \cdot \overline{E\ BUSY}^L) + CTRL\ LAST\ CYCLE^L$$
$$= (IEL^L + BRLC^L + \overline{IE\ BUSY}^L)(ELC^L + \overline{E\ BUSY}^L)$$

Interrupts (Figure 57)

Since interrupts are so important to the I to E transfer, they will be discussed before going further into the functions performed at that time.

There are two classes of interrupts: I time interrupts and E time interrupts. I time and E time interrupts are distinguished by the time that the interrupt is processed. I time interrupt sequences are entered at the I to E transfer, and E time interrupts are entered after the last cycle of execution time.

I time interrupts may be detected during T1 or T2. Interrupts resulting from instructions located in an invalid storage address, from instructions located on a non-halfword boundary, or from instructions with invalid operation codes are detected during T1. If any of these interrupts occur, the I program interrupt trigger is turned on with TN T2. At the same time, the interruption code is set up for the type of interrupt which has occurred. An I time interrupt is taken at the time of the I to E transfer and no execution unit is started. Interrupts resulting from an invalid operand specification, from an attempt to process an execute instruction while in execute mode, from decoding the instruction SVC, or from decoding of a privileged instruction while not in monitor mode are all detected during T2. If any of these interrupts occur, except SVC, the I program interrupt trigger is

turned on with the I to E transfer. If I program interrupt was not on previously, the interruption code is set up at the I to E transfer for the type of interrupt which has occurred. An I time interrupt is taken and no execution unit is started.

I time interrupts are entered at I to E transfer time by turning on the I interrupt end trigger in the interrupt sequencing controls. The essential line which does all the conditioning for I time interrupts is I interrupt from I.

E time interrupts are normally detected during the last cycle of instruction execution. Certain E time interrupts are also detected during interrupt last cycle or while the system is in wait status.

E time interrupts are generated in two locations: the I unit and the E unit. Interrupts which are detected in the E unit during the course of an instruction are set into a buffer register in the E unit. These interrupts are program interrupts such as divide check, exponent overflow, etc. The detection of such an interrupt causes the E interrupt from E line to rise. A second group of E time interrupts consists of interrupts which are detected asynchronously in the I unit. These are interrupts such as I/O interrupts, timer advance requests, external interrupts, etc. These interrupts, which are buffered in the I unit, cause the E interrupt from I line to rise.

If an E time interrupt is detected, then during the last cycle of instruction execution an E time interrupt sequence is started by turning on EXIT with an A clock. The interruption code is set from the interrupt buffers during the interrupt sequencing. It is possible that the I to E transfer of the next instruction is occurring at the same time as the turn-on for EXIT. If this is true, the execution unit is blocked from starting by the E time interrupt lines.

It is possible for an E time interrupt to occur at the same time as an I time interrupt for the next instruction. Since the E time interrupt is associated with the earlier instruction, it takes priority over the I time interrupt. Thus, even though the interruption code has already been set for the I time interrupt, the E time interrupt is processed and the interruption code is set again.

Both EXIT and I interrupt end generate a line called interrupt reset. This line resets any sequencers in the I unit which might still be on and thus effectively terminates all normal instruction processing.

Starting of Execution Units (Figures 5265 and 58)

Execution units are started at I to E transfer time. The branch unit is started by I to E TRANSFER · NO IRPTS · BROP^{L}. Branch operation is turned on at TN T2 for all branch instructions (provided R2 $\neq 0$ in an RR branch) and, therefore, may be used to condition the branch unit. The IE unit is started by I GO = I to E TRANSFER · NO INRPTS · BD I EXEC · $\overline{\text{BROP}}$. BROP must be excluded because the instruction BCR is included in BD I execute. For BCR the IE unit is started if R2 = 0 in which case the IE unit does a no operation. Branch operation is turned on for this instruction only if R2 $\neq$ 0 and hence the exclusion in I go.

E go is complicated by the fact that certain lines from the E unit may be late in arriving at the I unit. The logic for E go is:

$$T2^{L}\,(\overline{\text{OPF}}^{L} + \text{ACCEPT}) \cdot (\overline{\overline{\text{IE BUSY}}^{L} + \text{BRLC}^{L} + \text{IEL}^{L}}) \cdot \overline{\text{I IRPT FROM I}} \cdot \overline{\text{E IRPT FROM I}} \cdot \text{BD E EXEC}$$

The expression $(\text{ELC}^{L} + \overline{\text{E BUSY}}^{L}) \cdot (\overline{\text{E IRPT FROM E}})$ is effectively generated in the E unit. This expression is ANDed with E go to obtain a signal to start the E unit. It can be seen that the expression $\text{EGO} \cdot (\text{ELC}^{L} + \overline{\text{E BUSY}}^{L}) \cdot \overline{\text{E IRPT FROM E}}$ is analogous to the expression for I go. It is necessary to do part of the ANDing in the E unit because the line E interrupt from E is too late to send to the I unit and then back to the E unit. $\text{ELC}^{L} + \overline{\text{E BUSY}}^{L}$ is also ANDed with E go in the E unit as protection against a late ELC line.

Stores (Figures 59 and 5263)

Stores which are made at the I to E transfer by the I unit are described in this section. Most features of this type of store are similar to the features of any CPU store. Thus, an understanding of all stores may be obtained by reading this section.

For instructions such as ST, STE, and STD, store requests are made at the I to E transfer if no interrupts are outstanding. BD store request, no interrupts, and I to E transfer set the CPU request and the CPU store triggers in the BCU and the store request trigger in the I unit. For all stores in this type, the effective address is set into SAR and H at TN T2. Block T2-M is turned on by the I to E transfer, thus preventing TN T2 of the next instruction from altering SAR until an accept is received. The E unit sends, for these instructions, the line TF block T2-M on accept which is ANDed with accept to turn off block T2-M and to allow setting of SAR by TN T2 of the next instruction.

CPU request starts a storage unit and generates accepts just as for a fetch. The store request line to the BCU is held up by the latched output of store request until store request is turned off by accept.

The CPU store trigger indicates to the BCU that a store rather than a fetch is being requested. CPU store remains on until SAR is again set.

There are eight mark bits in BCU which must be set to indicate which bytes in external storage are to be altered. These bits are set at the I to E transfer of stores of the type being discussed. The bits are set on the basis of the output of the BOP decoder and bits 21 and 22 of the H register. The output of the BOP decoder indicates the length of the field being stored, and bits 21 and 22 indicate where the field being altered starts in a storage word. The mark bit corresponding to any byte in the storage word to be altered is set to a 1 at this time.

NOTE: It is possible to set the mark bits individually on various cycles. This facility is used extensively in the SS instructions for setting up store fields.

The E unit is started at the I to E transfer of the instructions being discussed. The E unit places the operand to be stored into the K register at the end of the first E unit cycle. K is gated into the SBI latch on the early BR clock immediately following the AR clock that is straddled by pulsed accept. ELC is set by A clock accept for this class of instructions.

If the address for the store is for a nonexistent storage, CPU store invalid is set with the late BR clock just before the AR clock is straddled by the pulsed accept. The invalid address interrupt is then taken as an E time interrupt at the end of ELC.

For all stores by CPU (except stores by the interrupt mechanism), bits 8-11 of the PSW are gated out to a storage protection feature. If a compare is generated in this feature, a storage address protection (SAP) signal is sent back to the CPU. This signal is sampled into the SAP latch at the AR clock, that follows accept. This causes an interrupt which is treated as an E time interrupt. However, the interrupt may return too late to be processed after the instruction that caused the interrupt. Therefore, the interrupt will not be processed until the next instruction is completed.

The store request trigger gates the ICR into the incrementer and adds 1 into position 20 of the incrementer. Compares are made between H and the ICR, and between H and the output of the incrementer. The two compare lines are handled as described in the "Instruction Fetching Controls" section to generate the program store compare (PSC) line. This signal ANDed with store request^{L} turns on the PSC trigger in the interrupt mechanism. The PSC line indicates that a store is being made into an address which may have been prefetched into the A or B register for instruction processing. The PSC trigger will cause an E time interrupt to be taken which initiates an IC recovery only. This results in the A and B registers being refetched after the store is completed.

If the CPU is being single cycled, the store request must be delayed for one cycle to allow the E unit time to set the K register. The one cycle delay is generated by a pair of single-cycle store triggers in the BCU. These triggers are turned on in place of CPU request when the store request is first made. On the next A clock, the single-cycle store triggers turn on CPU request, and the store proceeds at high speed. See Figure 60.

Additional Functions of I to E Transfer

Updating the PSW Length Code: Bits 32 and 33 of the PSW, the length code, are set to the length of the instruction by the I to E transfer. The number of halfwords required by the instruction word determine its length. Updating the LC does not occur if an E time interrupt is detected.

Updating the ICR: As defined in the "Instruction Fetching Controls" section, the ICR is considered in two parts: high order and low order. With the I to E transfer, the contents of the gate select register are set into the ICR low order. If the IC high order needs advancing, IC high-order advance is turned on with the I to E transfer. Neither action takes place if an E time interrupt is outstanding.

Ending T2: The I to E transfer turns T2 off.

Turn-On Block Triggers: When BOP decoding indicates that blocking of the next T1 or T2 cycle is required, block T1-M and/or block T2-M triggers are turned on with the I to E transfer.

Setting of Busy Triggers: The I to E transfer in generating the start condition for execution units also sets their respective busy triggers (IE busy and E busy).

SS Instructions: The SSOP status trigger is set by TN T2, and the I to E transfer conditions the VFL address trigger if there is no interrupt.

Decondition FR2 (Floating-Point R2): The I to E transfer prevents any further setting of FR2.

Set SAR and H Registers: SAR is set from the incrementer at I to E transfer time for branches and LM. H is also updated for LM. The incrementer error is sampled at this time.

ER1: The contents of the BOP R1 field are set into the ER1 register with the I to E transfer. ER1 is used by the E unit for put-aways to internal registers.

BR + 1 Request: The BR + 1 fetch for branches is generally made at I to E transfer time if there is no interrupt.

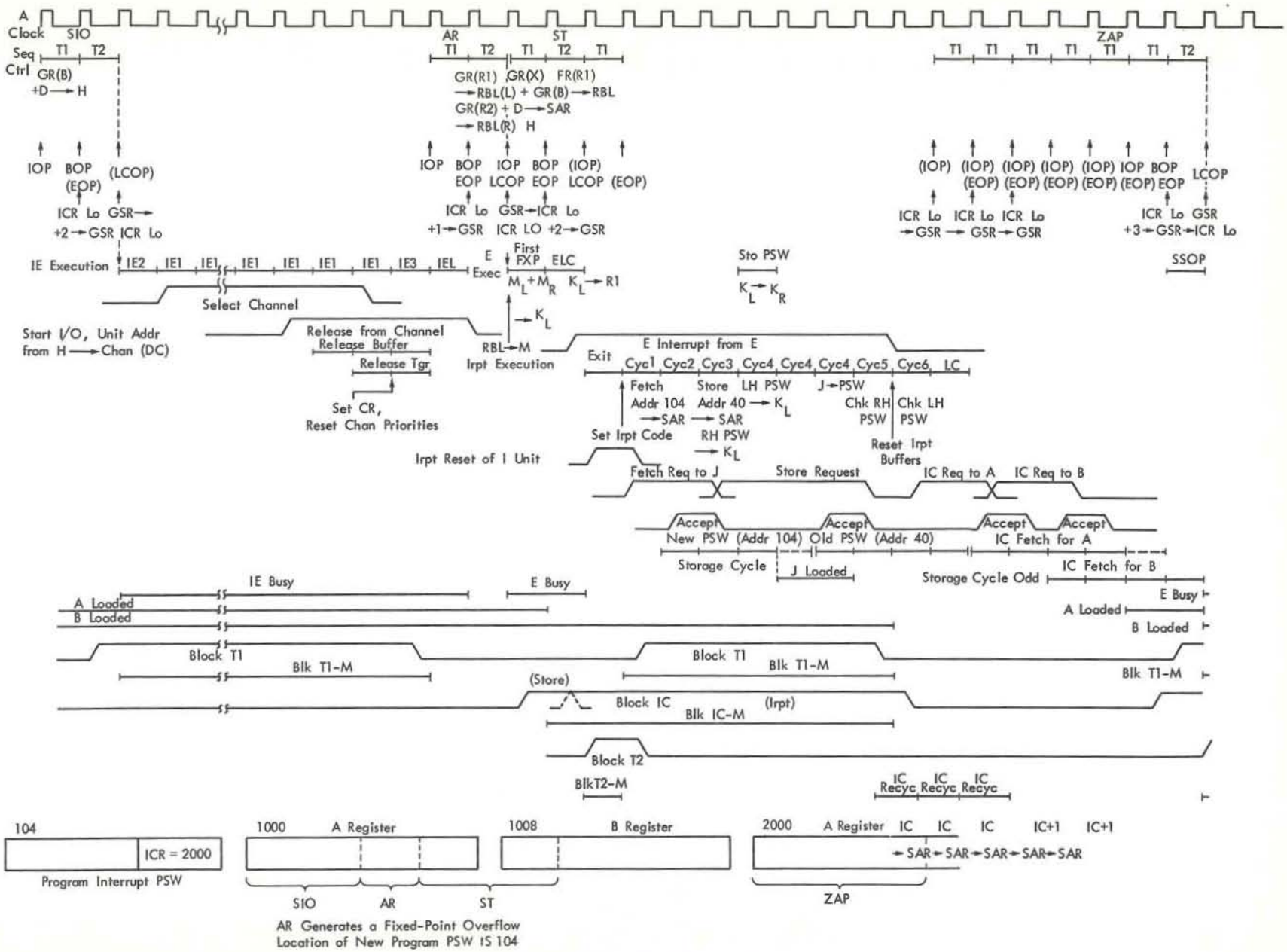


FIGURE 57. INSTRUCTION SEQUENCING FOR START I/O AND INTERRUPT

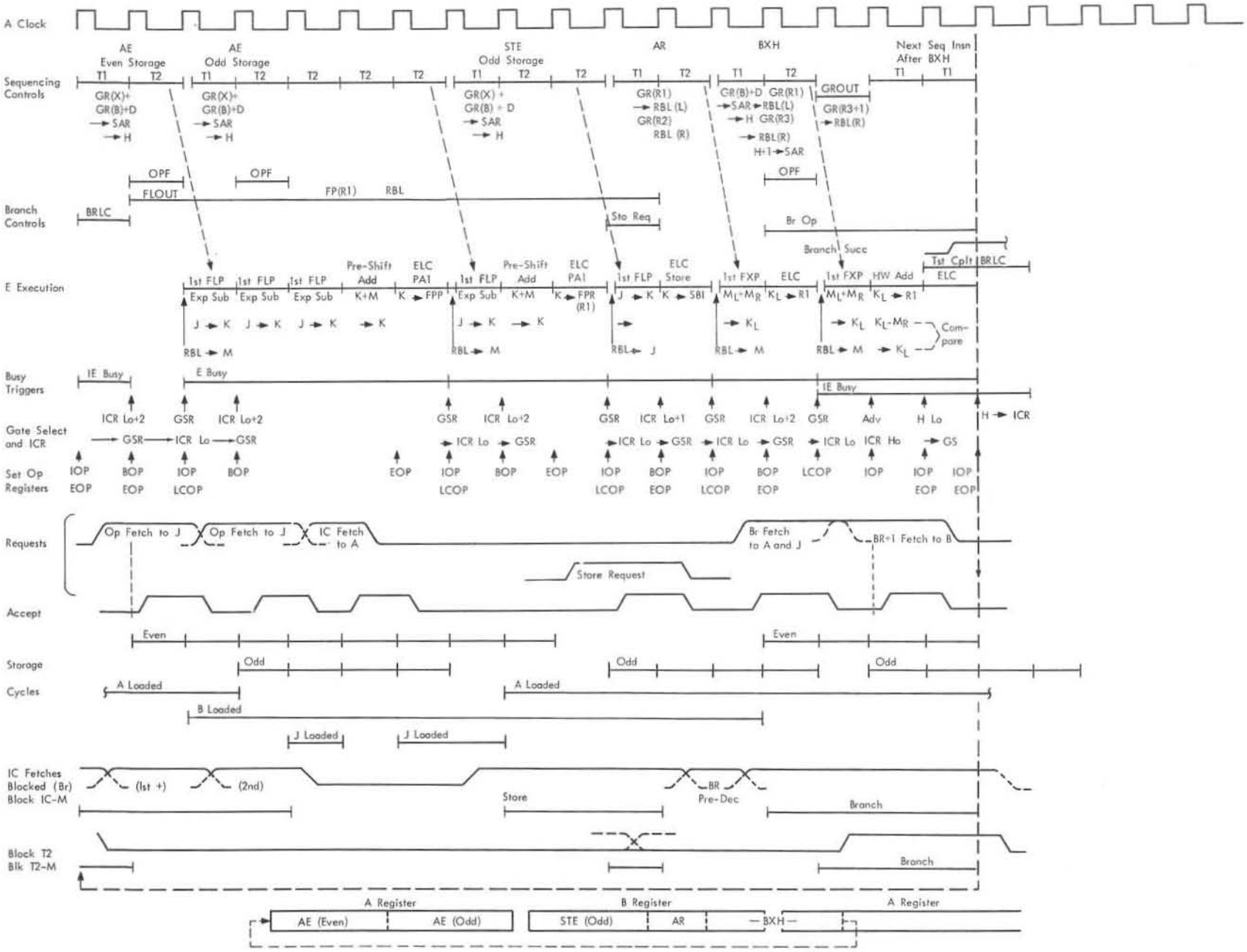


FIGURE 58. INSTRUCTION SEQUENCING FOR E EXECUTIONS AND BRANCH

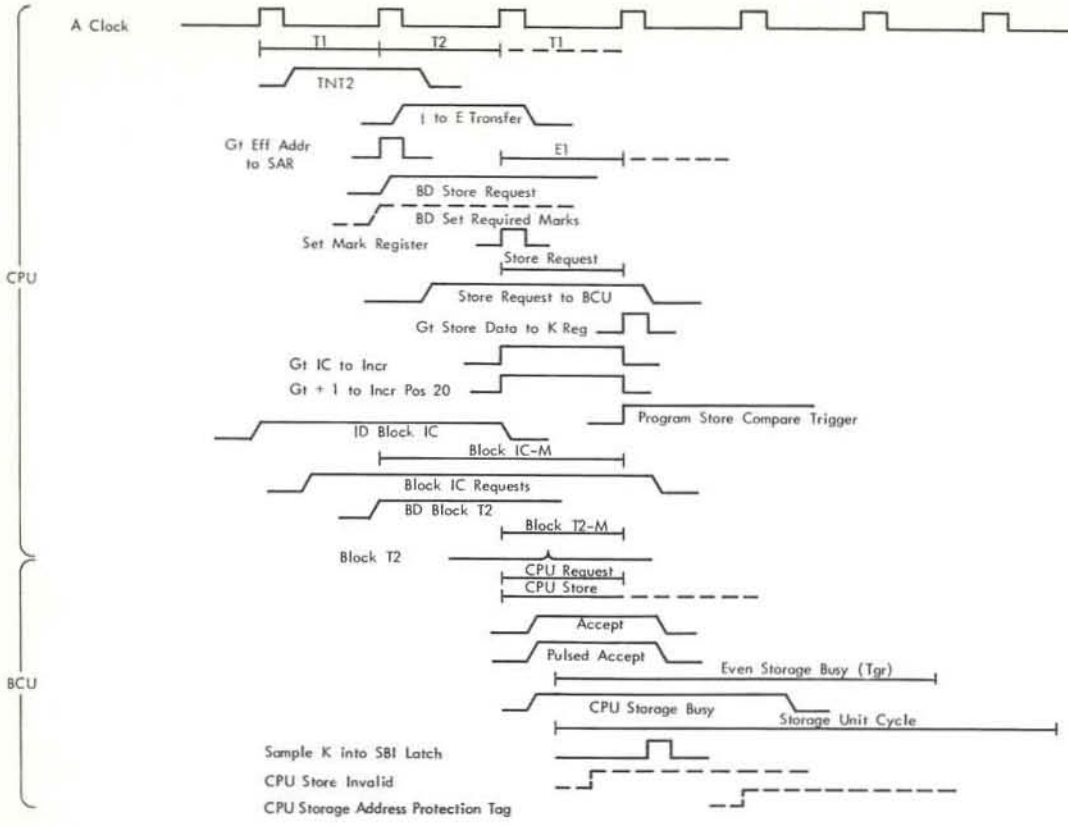


FIGURE 59. I UNIT STORE REQUESTS

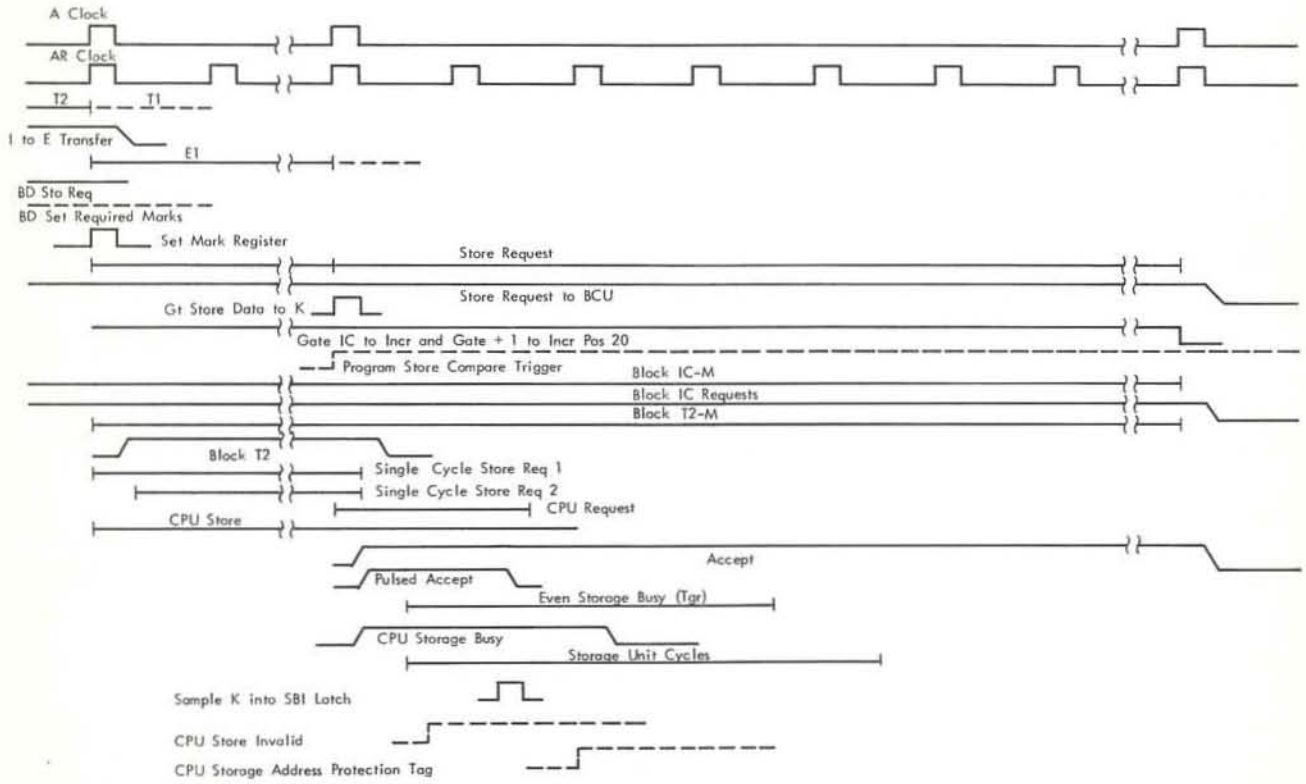


FIGURE 60. I UNIT STORE REQUESTS--SINGLE CYCLE

T1 and IOP: The I to E transfer is used in the logic for setting T1 and IOP.

General Register Out (GROUT): GROUT is turned on for certain instructions to gate out GPR during E cycles.

BOP Error: A parity check is performed on the first byte of BOP at the I to E transfer. BOP error is turned on at this time if there is a parity error.

Decoders

The E unit contains two operation registers: execution operation register (EOP) and last cycle operation register (LCOP). Both EOP and LCOP are eight-bit registers which are loaded with the first eight bits of an instruction. EOP is set from IOP, and LCOP is set from EOP. With the use of two operation registers, the E unit is able to overlap the last cycle of one instruction with the decode cycle for the next instruction.

EOP is set by the following logic (Figure 5258):

$$T1^{L} \cdot (\overline{E\ BUSY}^{L} + ELC^{L}) + TON\ ELC$$

This logic ensures that EOP is set one cycle before the start of an E unit instruction. If the E unit is not operating or if ELC is on during the last T1 cycle, EOP is set correctly at TN T2; therefore, EOP is set at least a cycle before the I to E transfer. However, if the E unit does not finish the previous instruction until after T1 of the next instruction, TN ELC ensures that ELC is set at least one cycle before the I to E transfer. Note that EOP is not available to the E unit during the last cycle of an instruction, therefore, decoding is done from LCOP during this time.

LCOP is set by the following logic (Figure 5259):

$$T2^{L} \cdot \overline{(E\ BUSY}^{L} + ELC^{L})$$

This logic has the effect of setting LCOP at the I to E transfer, perhaps along with many unnecessary earlier settings.

Instruction Executions

There are a few instructions which have unique features that are relevant to this section.

GROUT Class Instructions (Figure 61)

There are a number of instructions which require that the general register operands be gated out during E time. This gating is accomplished by GROUT, a trigger located on G-C2 near the general registers. Since GROUT gates out operands from the general registers, T1, which also gates out general registers, cannot be allowed to turn on (except in EX). This is accomplished by turning on block T1-M.

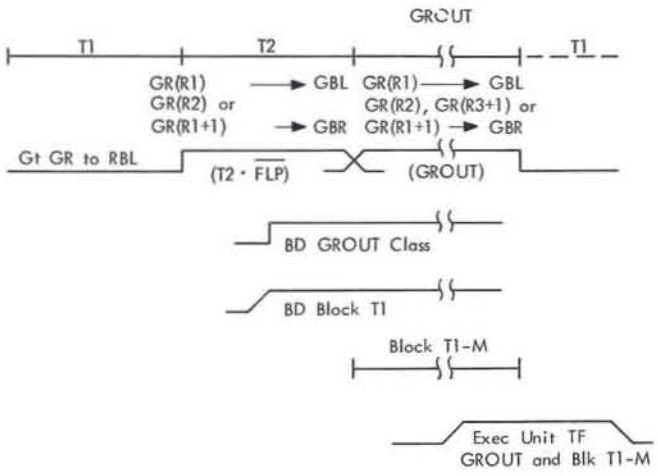


FIGURE 61. GROUT TIMING

Those instructions which use GROUT are: SPM, SSK, MR, DR, EX, D, BXH, BXLE, and STM. For the SPM, SSK, and STM instructions, GROUT gates BR1 onto GBL for use by the execution unit. GROUT also gates general register BR1 onto GBL for EX. T1 and GROUT are on simultaneously for EX. GROUT overrides T1 to the extent of preventing the GR specified by IB2 from being gated onto GBL.

For the D and DR instructions, the R1 field must be even or an invalid specification interrupt is taken. If R1 is even, GROUT gates general register BR1 onto GBL and general register BR1 + 1 onto GBR. For the MR instruction, GROUT gates general register IR2 onto GBR.

For the BXH and BXLE instructions, GROUT gates general register IR3 onto GBR also. In this case GROUT forces a 1 into the low-order position of the decoder so that the odd register of an even/odd pair is always obtained (R3 + 1).

For all these instructions (except EX), the execution unit sends a signal to turn off GROUT. For the EX instruction, GROUT is turned off by the I unit controls after bits 24-31 of the general register, addressed by BR1, have been ORed into bits 8-15 of IOP.

Note that operands are generally gated out by GROUT to the unspecified half of RBL. However, these operands are ignored by the execution unit.

Supervisor Call (SVC)

The instruction SVC is unique in that it is treated as a T2 interrupt and no normal E time is taken. At I to E transfer, the SVC interrupt competes with any other existing interrupts for priority, and the interrupt sequence is entered. See Figure 6163.

Shift Instructions

The shift amount for shift instructions is obtained from bits 18-23 of the effective address which is placed in H at TN T2. The H register is available to the E unit.

WD, RD, and Immediate Instructions

For these instructions, IOP bits 8-15 are transferred to the Y and Z registers in VFL. The Y and Z registers are set by VFL controls at B time. They are set by essentially the same logic that sets EOP. Thus, the I to E transfer may be allowed to change IOP.

INSTRUCTION FETCHING CONTROLS

The instruction counter controls in the 2075 control the normal advancing of the instruction counter and the normal fetching of instructions.

Instructions returned from storage are buffered in either the A or the B register before being sent into the IOP register for initial execution. The instruction counter register (ICR) contains 24 bits (numbered 0-23) and is advanced by means of two adders: the gate select adder which advances the IC low order (bits 20-22) and the incrementer which advances the IC high order (the remaining bits). The gate select adder works in conjunction with the gate select register (GSR) to select gates from the A and B registers to the IOP register. The instruction counter controls, while advancing the GSR, also maintains the ICR with a proper address for interrupt purposes.

The IC controls also make normal IC fetches and generate the instruction fetch addresses. The addresses are generated by adding in the incrementer an appropriate small increment amount to the ICR. The IC controls attempt to make an IC fetch as soon as an empty instruction buffer condition is detected, but any instruction in the process of execution may block out IC fetches if an IC fetch would cause interferences with the instruction execution. If the IC fetches are blocked continuously by instruction executions, the I unit ultimately will exhaust all instructions in the buffers. At this time, the IC block will drop, thus allowing IC fetches to be made and instruction execution to resume. Normally, the instruction buffers will not be exhausted before IC fetches are made. However, special logic has been incorporated to ensure that, except in unique situations, both buffers are not emptied.

Physical Description of Data Flow

The IC is a 24-bit register used for addressing instructions in external storage during normal linear sequencing. Though the register is located in positions 40-63 of the PSW, it is numbered 0-23 in the 2075. The ICR is gated into a 24-bit adder called the incrementer. The input on the other side of the incrementer is at positions 19 and 20 only. These two lines are generated by controls. Thus, addresses at the incrementer input can be advanced 0, 1, 2, or 3 double words. The output of the incrementer is latched with an L clock, and this latch register has outputs to SAR and ICR (in addition to other outputs). While the gate to SAR is a full 24 bits in width, the gate to ICR lacks positions 20-22 and, therefore, consists of only 21 bits. A full 24-bit path from the H register to the incrementer is also provided.

Note that the incrementer is not a true general adder because certain special inputs can occur only if there is a machine malfunction. If there is a bit in incrementer data input position 20 at the same time as there are bits on both control inputs, an incorrect sum results. This appears as:

	Position							
Data	16	17	18	19	20	21	22	23
Incrementer data input	-	-	-	b	1	-	-	-
Control input				1	1			
Incrementer output	-	-	-	$\bar{b}$	0	-	-	-
Sum	-	-	-	b	0	-	-	-

For this case result, position 19 is erroneous; and if b is 0, the carry which normally would go into position 18 is not generated. For any other inputs, the incrementer gives a correct sum. The parity prediction and checker for incrementer P16-23 (called P2) is designed for the incrementer as implemented--not a true adder. The details of the incrementer implementation are described in the "Functional Units" section of 2075 Processing Unit, Volume 1, Field Engineering Manual of Instruction, Form 223-2872.

A, B, and J Registers

There are two 64-bit instruction buffers (A and B). The A register is associated with double words in storage for which the address contains a 0 in position 20. The B register is associated with double words for which the address contains a 1 in position 20. Except for this one consideration, registers A and B are symmetrical in usage. Bit 63 of register A is considered as being adjacent to bit 0 of register B, and bit 63 of register B is considered as being adjacent to bit 0 of register A. There is also one 64-bit operand buffer J. All of these buffers are fed by the 64-bit SBO which is fed from external storage. There is also a 64-bit bus from J to AB. There is a 32-bit operand

register called IOP which can be loaded directly from any 32-bit field in AB, starting at a halfword address. The field in AB that is selected for gating to IOP is determined by a full decoder off a three-bit gate select register.

Checking (Figure 62)

The A, B, J, IOP, SBO, H, and ICR registers have one parity bit for each byte. The incrementer has a checker which checks input parity and the internal operation of the incrementer. The incrementer also generates normal parities for each of the three output bytes. These are generated by independent circuitry. A unique parity bit (P1) is also generated within the incrementer. This parity is generated by incrementer inputs 16-19, control input 19, and the input parity. Assume that there are an even (odd) number of bits in incrementer inputs 16-19. If control input 19 is 1, and if adding a 1 in position 19 of the incrementer input would result in a sum with an odd (even) number of bits in positions 16-19, P1 is set opposite to the input parity. If the result has an even (odd) number of bits in positions 16-19, P1 is set equal to the input parity. The parity bit P1 is gated into ICR P16-23 instead of the conventional parity bit for 16-23. However, the normal incrementer parity for 16-23 is gated to SAR P16-23. The following is an example of the operation of the incrementer:

	Position									
Data	16	17	18	19	20	21	22	23	P	P1
Incrementer input	0	0	1	1	0	0	1	0	0	
Control input				1	1					
Incrementer output	0	1	0	0	1	0	1	0	0	1

Gate Select Mechanism (Figure 5018)

A three-bit path is provided from ICR 20-22 to the gate select added pre-latch. A second three-bit input to the gate select adder pre-latch comes from the gate select register. The output of this latch feeds into the gate select adder. The gate select adder is a three position adder that is implemented in two levels. Control inputs are provided at positions 21-22 on the second side of the gate select adder. The output of the gate select adder can be gated into the three position gate select register. This register also has a gated input from positions 20-22 of the H register which is used during branches. There is also a carry position in the gate select register which is fed by the carry output of the gate select adder. The decoder off the gate select register selects 32-bit fields in the A and B registers for gating to IOP:

GSR	Field Selected
000	A 00-31
001	A 16-47
010	A 32-63
011	A 48-63, B 00-15
100	B 00-31
101	B 16-47
110	B 32-63
111	B 48-63, A 00-15

The gate select register has a corrected parity position which is used in updating ICR P16-23 whenever ICR 20-22 is updated from the GSR. The corrected parity is generated from the gate select adder data inputs and a parity bit that is sent with the data. Normally this parity comes from ICR P16-23, but if the ICR is being set from the incrementer at the same time as the GSR is being set from the ICR, the input parity is taken from P1 of the incrementer.

The expression for generation of the corrected parity bit is:

$$\text{Input Parity} \veebar \text{Adv 1 Halfword} \cdot \overline{\text{Adv 2 Halfwords}} \cdot \text{GS Adder Input 21} + \overline{\text{Adv 2 Halfwords}} \cdot \overline{\text{GS Adder Input 22}} + \text{GS Adder Input 21} \cdot \overline{\text{GS Adder Input 22}} + \overline{\text{Adv 1 Halfword}} \cdot \text{Adv 2 Halfwords} \cdot \overline{\text{GS Adder Input 21}}$$

GS adder input 21 and GS adder input 22 are data inputs into the GS adder, and advance 2 halfwords and advance 1 halfword are the corresponding control inputs.

This expression predicts the new ICR P16-23 that results from the add in the gate select adder. If any single error occurs in the gate select adder or the parity generation circuitry, the corrected parity bit is incorrectly set. When the GSR is returned to the ICR, the last byte of the ICR contains incorrect parity. Whenever the ICR is gated into the incrementer for an IC high-order advance or an instruction fetch, the parity is checked. At this time the erroneous parity results in a data check.

Addressing of Instructions

At the start of each T1 cycle, the instruction being executed by the I unit is placed into the IOP register. The leftmost halfword of the instruction is placed into IOP 0-15, and the next halfword is placed into IOP 16-31. If the instruction is an RR instruction, IOP 16-31 is not involved in the instruction execution. The basic unit of data in external storage is the double word = two words = eight bytes = 64 bits. Addresses, which are 24 bits in length, address bytes. Therefore, bits 0-20 of any address specify a double word in storage and bits 21-23 specify a byte within a double

word. Instructions always start at halfword addresses, and any address for an instruction should have a 0 in position 23. To obtain an instruction at a given address, it is necessary to first fetch the double word that is addressed by bits 0-20. If bit 20 is 0, the double word is buffered in the A register. If bit 20 is 1, the double word is buffered in the B register. If the instruction is located in a position which overlaps a double word boundary, it is necessary to fetch the next double word. This second double word would be buffered in the B register if the first double word was in the A register, and vice versa.

Before any instruction that is contained in the A-B registers can be properly set into IOP, one of the gates out of the A-B registers must be selected. This is accomplished by setting bits 20-22 of the instruction counter register into the GSR. On the basis of the GSR, one of the eight gates from the A-B registers to IOP is selected. The specific decoding was described in the "Data Flow" section.

The first two bits of any instruction, which are placed in IOP 0, 1 (and BOP 0, 1), contain a code that identifies the number of halfwords in the instruction. There is a length decoder off of BOP 0, 1 which provides the actual length of instruction. The output of the length decoder is fed to PSW 32, 33. The coding is:

IOP 0	1	Input to PSW 32	33
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	1

ICR Advancing

An example of ICR advancing is shown in Figure 63. ICR 20-22 is gated continuously into the gate select pre-latch, except during special circumstances. Normally this latch is not latched, but acts as a flush path. The two-bit instruction length, which is obtained from the IOP decoder, is gated into the control inputs of the gate select adder.

At the start of any instruction, bits 20-22 of the address for that instruction are contained in ICR 20-22 and the GSR. During the last T1, the instruction being executed is located in IOP; and, therefore, bits 20-22 of the address of the next instruction are available at the output of the gate select adder. This output and the carry and corrected parity are gated into the corresponding GSR position with TON T2 A CLK. During T2, the GSR output selects the proper gate from the A-B registers for the next instruction. Thus, the gates are selected in time for setting IOP with the next instruction in the A-B registers on the clock cycle following TN T2.

When an instruction is transferred from the I unit to the E unit, the IC low order is advanced by gating the GSR to ICR 20-22. The corrected parity is also set back into ICR P16-23 at the I to E transfer. The length of the instruction being transferred is gated from the BOP decoder to PSW 32, 33 with the I to E transfer. If a carry was generated by the gate select adder, the ICR will not be advanced completely until positions 0-20 are advanced. Therefore, in order to advance the IC high order, the IC high-order advance trigger is turned on by I to E transfer and GS carry. IC high-order advance gates the complete ICR into the incrementer, adds a 1 in control input position 19, and gates the sum and the new parities back into the ICR 0-19 and 23. Thus, within one cycle of the I to E transfer, the whole ICR is updated.

As will be described under "Instruction Fetching," the IC controls have the ability, under certain circumstances, to force the incrementer control input to generate IC fetch addresses. The fetch controls are not necessarily blocked during an IC high-order advance. Therefore, the possibility exists that the IC fetch controls may be activating incrementer control input 20 (also, incrementer control input 19) at the same time as an IC high-order advance is activating incrementer control input 19. ICR 20 is always 0 whenever an IC high-order advance is made; therefore, there can be no carry into position 19 of the incrementer. During IC high-order advance, incrementer sum positions 0-19 and 23 are the correct bits for updating the IC high-order even though IC fetch addresses are being generated. Incrementer parity bit P1 16-23, which is generated by predicting the effect of the incrementer control input 19 on the input parity, is the correct parity for the updated ICR and is gated back to ICR 16-23.

When updating the IC low order, the quantity to be updated should be taken from the fully advanced previous ICR value. It is possible for an IC high-order advance to coincide with a good or last T1 cycle. In this case, the IC high order is being advanced for one update at the same time as the updated IC low order is being set into the GSR for the next update. This generates no problems (because of the partial incrementer to ICR gate), except for the setting of the corrected parity into the GSR. Since the parity to be modified in the gate select adder has not yet been set into the ICR, the parity is taken directly from P1 16-23 of the incrementer output. Thus, the corrected parity is set correctly at TN T2 for transfer to the ICR P16-23 on the next I to E transfer.

Advancing the IC high order during the first cycle of E time, creates problems with the execution of only two instructions. These are LA and LPSW. In

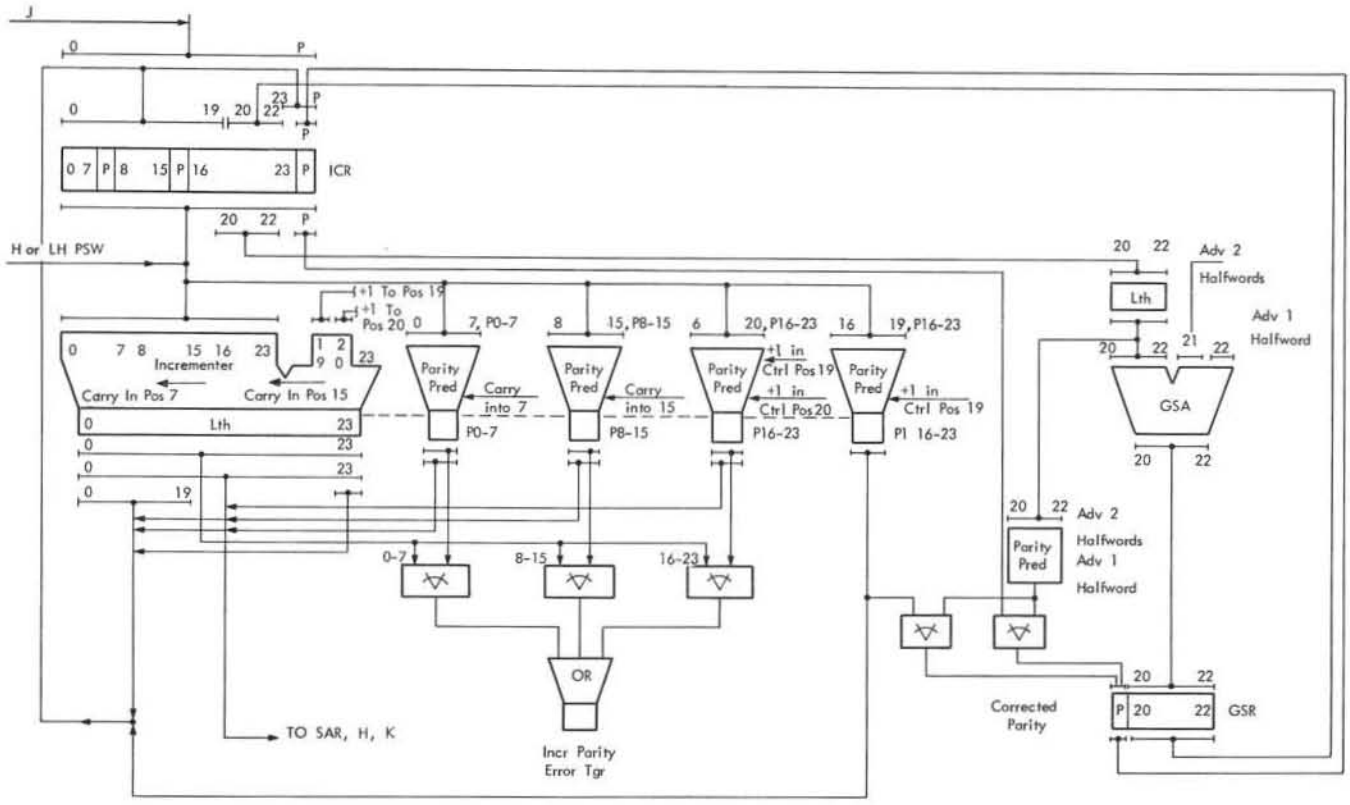


FIGURE 62. PARITY HANDLING IN THE INCREMENTER AND GATE SELECT ADDER

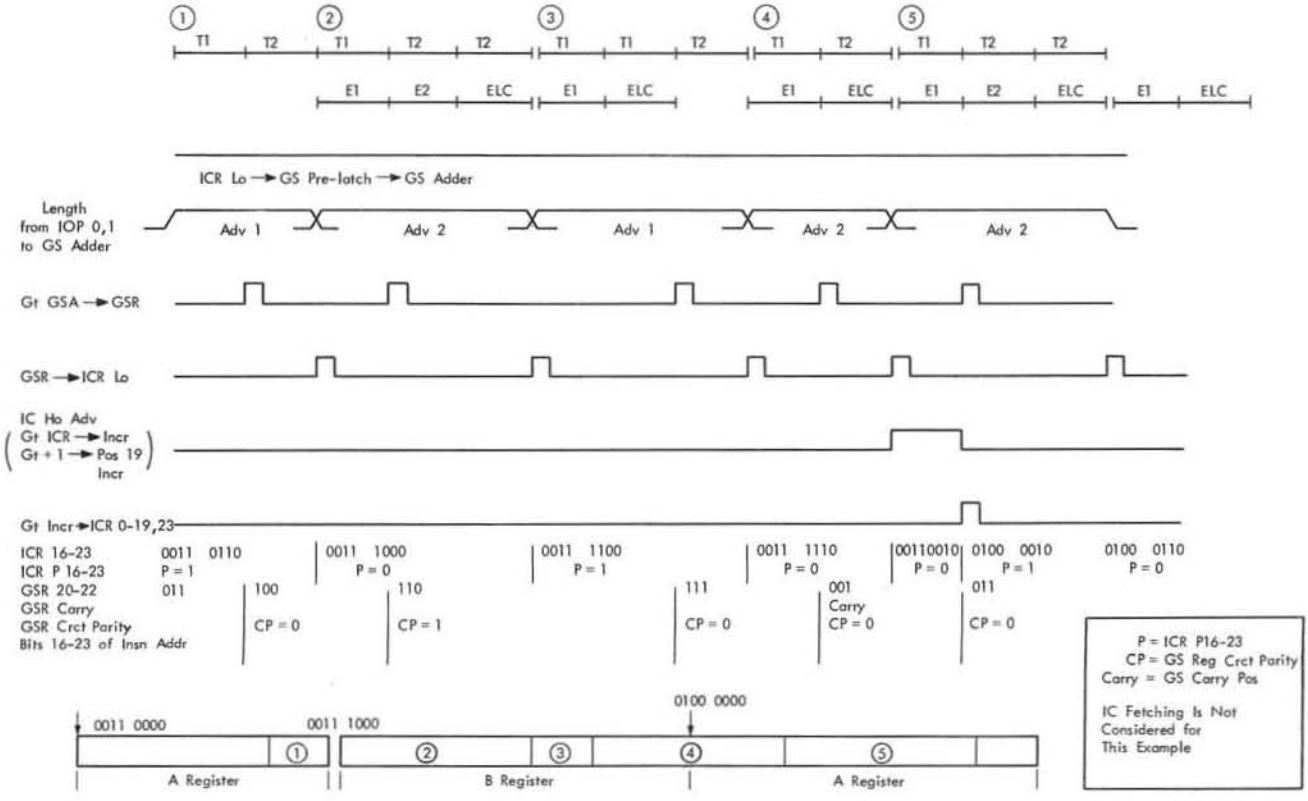


FIGURE 63. EXAMPLE OF NORMAL TIMING OF IC ADVANCES

LA, the E unit makes use of the incrementer to transfer the effective address to the K register. The transfer is made on the second E time cycle instead of the first E time cycle to avoid conflict with an IC high-order advance. In LPSW, there is a possibility that an IC high-order advance is gating the incrementer output into the ICR at the same time as J is being gated into the ICR by the instruction execution. To avoid this multiple gating, the IE unit blocks the incrementer to ICR gate whenever the PSW is being loaded from J.

Interrupts

If an interrupt occurs, the PSW which is stored must contain the address of the next instruction. There are two types of interrupts: E interrupts and I interrupts. E interrupts take place after the last cycle of instruction execution. This class includes the majority of interrupt types. I interrupts take place at the I to E transfer in place of normal instruction execution. It is possible for an E interrupt and an I interrupt to occur simultaneously. In this case the E interrupt is processed and the I interrupt is suppressed.

In the case of an E time interrupt, the ICR has already been advanced to the next instruction. Therefore, the I to E transfer of the next instruction is blocked from having any further effect on the ICR, the IC high-order advance trigger, or the PSW length code. In the case of an I time interrupt, the ICR must be advanced to the next instruction. At the time of the I to E transfer, ICR 20-22 is advanced normally, IC high-order advance is turned on if required, and the PSW length code is set.

SS Instructions

The ICR is advanced normally at the start of any SS instruction. During SS instructions the execution unit must alternately have access to the second and third halfwords of the SS instruction. Access to these halfwords is obtained through special controls to the gate select mechanism. An example of an SS instruction timing is shown in Figure 64.

At TN T2 of SS instructions, the SSOP status trigger is turned on. The output of SSOP latches the gate select pre-latch, thus holding bits 20-22 of the address of the SS instruction at the gate select adder input. When SSOP is latched, it also inhibits the normal gating of the instruction length code into the control side of the gate select adder. By this time the GSR has been updated normally at TN T2. At the I to E transfer the GSR is transferred to ICR 20-22. An IC high-order advance can also take place, if needed. The ICR is advanced for SS instructions as in any other instruction, and the ICR then has the proper value for handling any interrupts during the SS instruction.

At TN T2, block IC-M is turned on. At the I to E transfer, block T1-M and block T2-M are turned on. These three triggers guarantee that neither IC fetches nor I time of the next instruction can interfere with the SS execution.

The VFL address trigger is turned on with I to E transfer in an SS instruction. The output of this trigger is ANDed with an early B clock to set the output of the gate select adder into the GSR. In this case the early B clock is timed so that it does not overlap an A clock. The use of the B clock on the GSR serves as a latch. The GSR is guaranteed not to change until the IC low order has been set on the I to E transfer.

During execution of the SS instruction, the VFL logic has control over a special line--VFL address advance. This line provides an advance 1 halfword input to the gate select adder. Since SSOP and VFL address are both on, the address of the SS instruction is held at the gate select adder input by the gate select adder pre-latch and the normal length code addition is suppressed. With each early B clock, the GSR is set with the address of the SS instruction (if VFL address advance is not on) or is set to the address of the second halfword of the SS instruction (if VFL address advance is on).

The IOP register is set from the A-B registers by VFL ADR^{L} · SSOP^{L} ON EVERY A clock. The SS instruction is still in the A-B registers because IC fetching is blocked during SS instruction executions. IOP is set according to the quantity which was placed in the GSR by the previous early B clock. Depending on the VFL address advance line, IOP is loaded on every A clock with either the first two halfwords or the second two halfwords of the SS instruction. The SSOP trigger gates the IOP D field and the general register addressed by the IOP B field to the addressing adder. On the cycle after IOP is set, either the address (contents B1 + D1) or the address (contents B2 + D2) is available at the input to SAR and H. VFL has controls for setting the effective address into SAR and H, as desired.

At the end of an SS instruction, the VFL thru line is sent to the I unit to return the controls to normal operation. VFL thru turns off SSOP with an A clock, thus unlatching the gate select adder pre-latch. VFL now holds the VFL address advance line down, and VFL ADR · $\overline{T1}$ · $\overline{T1}^{L}$ now blocks the length code from entering the gate select adder. ICR 20-22 flushes into the gate select adder and is set into the GSR on the next early B clock. The GSR now contains the proper value for the next instruction. Block T1-M is turned off on the next A clock, and T1 of the next

instruction is turned on. Block T2-M, block IC-M, and VFL address are turned off whenever SAR is not needed by the SS instruction. When T1 turns on, VFL ADR · $\overline{T1}$ · $\overline{T1^L}$ no longer blocks the length code input to the gate select adder. The normal add is made in the gate select adder at TN T2.

Repeat Instruction (Figure 65)

Repeat instruction provides continuous execution of an instruction located in the left end of the A register. In order to utilize repeat instruction, it is necessary to first place the desired instruction in the left end of the A register.

All instructions may be processed in repeat instruction mode except branches, load PSW,or execute. Also, interrupts or IC recoveries may not be taken while in repeat instruction. Any of the above exceptions initiates instruction fetches to A and B, thus destroying the instruction being repeated. The DC line repeat instruction suppresses the following gates in the gate select mechanism:

1. ICR 20-22 to gate select adder input
2. Length code to gate select adder input
3. GSR to ICR 20-22

REPEAT INSTR · $\overline{\text{VFL ADR}}$ also holds the gate to GSR open continuously.

The operation of repeat instruction is:

Except during SS instructions, the suppression ot gating forces 0s to be flushed into the gate select adder. The 0s are set into the GSR by REPEAT INSTR · $\overline{\text{VFL ADR}}$. The GSR holds 0s at all times and continuously selects the first halfword in the A register. At no time is the GSR able to make an ordinary advance. The gate back to ICR 20-22 is suppressed in order to prevent setting the ICR with incorrect parity. If the original parity in the ICR was correct, it will remain correct throughout the operation of repeat instruction. During SS instructions VFL address is on and, therefore, REPEAT INSTRUCTION · $\overline{\text{VFL ADR}}$ no longer sets the GSR. The SS controls now operate and are able to utilize the gate select mechanism in the usual manner during instruction execution. Upon completion of the SS instruction, 0s are returned to the GSR. The interlocks to prevent IC fetching are discussed in the "Empty Rule" section.

Instruction Fetching (Figures 66, 67, and 68)

In the instruction counter scheme of the 2075, the interlocks between IC advancing and IC fetching are indirect in nature. Advancing the IC and fetching the instructions takes place at times which are not closely interlocked. In order to generate correct IC fetch addresses, the instruction fetching mechanism monitors the ICR. On the basis of the ICR and the loaded status of AB, the ICR is modified in the incrementer to obtain an instruction fetch address. The rules of IC fetching are:

IOP Loaded Rule: T2 of any instruction is not turned on before all halfwords of the instruction to be executed have been available in the A-B registers for at least one good T1 cycle.

Empty Rule: The fact that an instruction is emptying the A or B register and that an IC fetch is therefore required is recognized at TN T2 of the instruction that is emptying the register.

IC Fetch Rule: If either A or B register is empty and IC fetches are not blocked, an IC fetch request is sent to the BCU. The fetch request is maintained until accept is returned from the BCU or until IC fetching is blocked.

IC Address Rule: Addresses for IC fetches are generated automatically whenever IC fetching is not blocked by block IC-M. The address is generated on the basis of the ICR, the empty status of A, and the empty status of B.

Block IC Rule: Any instruction which has been completely loaded into the AB register and which is in the process of execution may block IC fetches in order to ensure that no interference arises in the usage of the incrementer or of the SAR. The block is started with TN T2 and continues to a point uniquely determined for each instruction.

Pre-Block IC Rule: Upon advancing the GSR at TN T2, the next instruction is available at the input to IOP. If this instruction is BXH, BXLE, BALR, BAL, BCTR, BCT, or EX, and is completely loaded in the A-B registers, instruction fetching is blocked. This rule prevents unnecessary IC fetches from interferring with branch fetches.

IC Fetch Priority Rule: In certain cases where A and B are both close to being emptied, TN T2 of the instruction about to be executed is blocked until the IC fetch mechanism fetches an instruction word. By decreasing the chances of both buffers being emptied, this rule results in a net speed-up of the system.

IOP Loaded Rule (Figure 5230)

To turn on T2 of an instruction, the instruction must have been in IOP for at least one good T1 cycle. The GSR is set with bits 20-22 of the instruction address no later than the cycle that precedes the start of T1.

The GSR is set with an A clock for normal ICR advancing; but in other cases, such as at the termination of SS instructions or branching, the GSR is set with an early B clock. In either case, the GSR output has at least a three-fourths cycle to select a gate from A-B to IOP. Any words that return to A or B are loaded with a late BR clock and an advance from BCU. BCU generates an A advance to gate in A and a B advance to gate in B. The contents of A or B have, at the very least, a one-fourth cycle to get to IOP. Usually the required instruction will have been available in the A-B registers for a number of cycles.

On the AR clock immediately following the late BR clock on which a word is returned to the A or B register, the trigger A loaded or B loaded, respectively, is set. A loaded stays on until the last instruction is removed from the A register. B loaded is treated similarly. A loaded and B loaded indicate when the A register and B register, respectively, contain unprocessed instructions. With the start of every T1 cycle, IOP is set from the output of the selected AB gate.

The IOP loaded line must rise before T2 can be turned on. This line, which indicates when an unprocessed instruction is available at the output of the AB ORs, is:

$$\text{A LOADED} \cdot \text{B LOADED} + (\overline{\text{GS21} \cdot \text{GS22}\,(\text{P0} + \text{P1})} + \overline{\text{GS21} \cdot \text{P0} \cdot \text{P1}}) \cdot (\text{A LOADED} \cdot \overline{\text{GS20}} + \text{B LOADED} \cdot \text{GS20})$$

GS20, GS21, and GS22 are positions of the GSR. P0 and P1 are positions of the selected word from the A-B registers. P0 goes to IOP 0 and P1 goes to IOP 1. The above line is latched with a not B clock and is then used to condition TN T2.

If A and B are loaded, a good instruction is available in IOP no matter which AB gate is selected. If the instruction selected is contained entirely within the A register and A is loaded, a good instruction is available in IOP. The instruction addressed is entirely within the A register if GSR position 20 is 0 and if none of the following are satisfied:

1. The instruction starts at the last halfword of a register (GS21 = 1, GS22 = 1) and is not an RR instruction (that is, P0 = 1 or P1 = 1).
2. The instruction starts in the last half of a register (GS21 = 1) and is an SS instruction (that is, if P0 = 1 and P1 = 1).

Similar logic is used to determine whether the instruction is entirely within the B register.

The IOP loaded line is latched with a not B clock because of a problem that arises during single cycling. A fetch may return to A or B on a running B clock a few cycles after the last controlled clock pulse has been given. IOP is loaded on an A clock. If T1 is on and if the turn-on of T2 is waiting for IOP loaded, this line must not be allowed to rise before IOP has been set. Immediately upon setting the A (or B) register and the associated A loaded trigger (or B loaded trigger), the raw IOP loaded line rises. The line IOP loaded^{L} does not rise because it was latched after the last B clock. IOP is set with the next A clock and IOP loaded^{L} rises with the next corresponding B clock. A good T1 cycle will occur before T2 is turned on by IOP loaded^{L}.

Empty Rule (Figure 5268)

If an instruction is emptying the A register, the A loaded trigger is turned off at TN T2. Though this section discusses the A register, an exactly analogous discussion holds for the B register. The logic, called ID A empty, for recognizing the last instruction in A is $\overline{\text{GS20}}$ (GS20 for B register) and any of the following:

$$\text{GS21} \cdot \overline{\text{IDRR}}$$
$$\text{GS21} \cdot \text{GS22}$$
$$\text{GS22} \cdot \text{IDSS}$$

IDRR is the ANDed output of not IOP 0 and not IOP 1. IDSS is the ANDed output of IOP 0 and IOP 1. The first term covers any two or three halfword instructions, starting in the last half of A. The second term covers any instruction, starting in the last quarter of A. The last term covers SS instructions, starting in the second quarter of A.

Not A loaded (ANDed with no outstanding fetch to A) is the normal method for indicating the necessity for a fetch to A. If IC fetches are not blocked and if a fetch to B does not have priority, a fetch request to A is made on the next A clock. As described previously, A loaded is turned on when the fetch returns to A. An auxiliary trigger, instruction counter fetch to A memorized (ICAM), is turned on when the BCU accepts the request to A. ICAM (and its turn-on) serves to indicate an outstanding fetch. ICAM (and its turn-on) prevents not A loaded from indicating a fetch required condition for A.

If an instruction is emptying the A register, the conditions which turn off A loaded at TN T2 also enter directly into the fetch request circuitry. If IC fetches are not blocked and if a fetch to B does not have priority, a fetch request to A is made at TN T2. For a number of instructions, including most RRs, this anticipation circuitry allows early IC fetches to be made.

Note that if the system is in repeat instruction, the GSR never leaves the first halfword of the A register. No instruction buffer is ever made emtpy and no IC fetch is every made. Thus, the contents of the

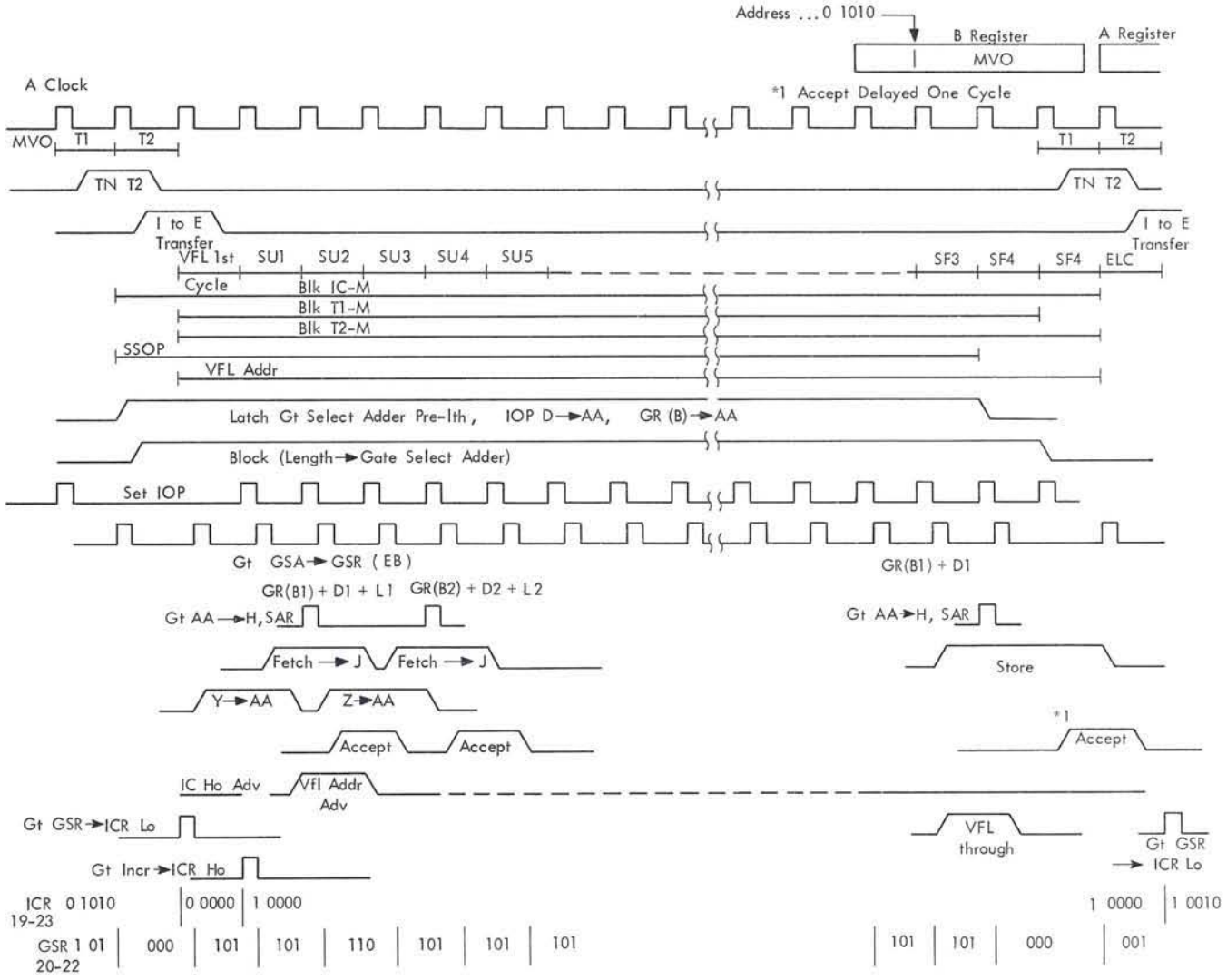
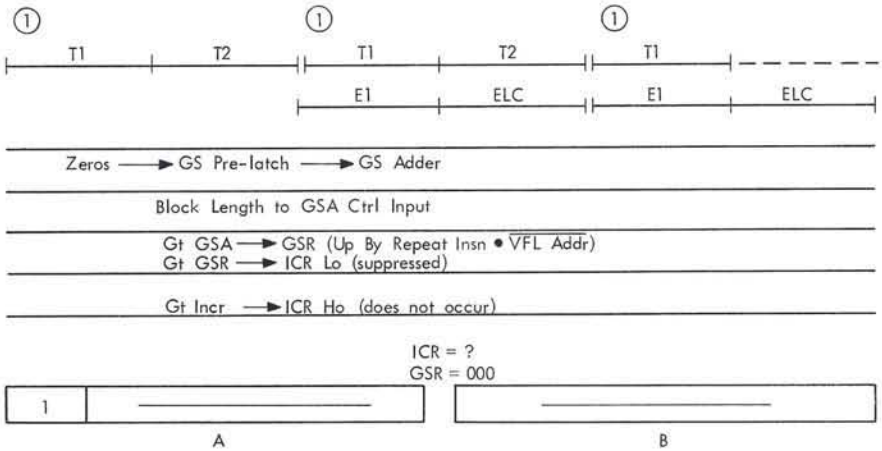


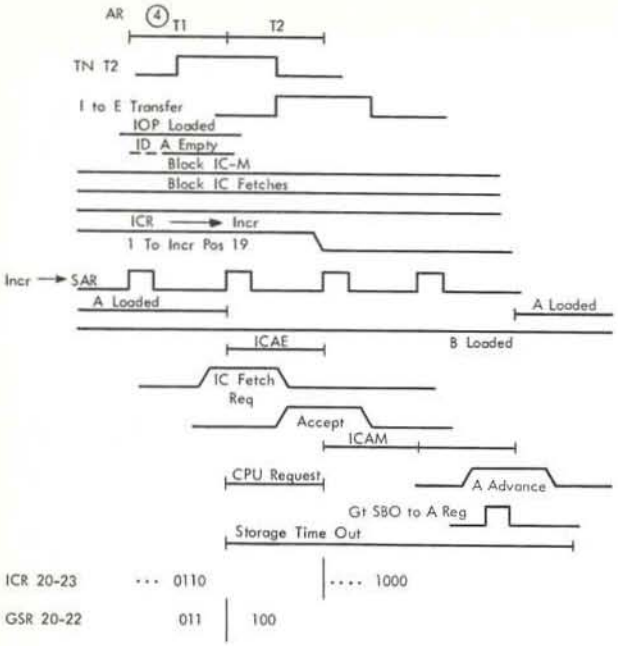
FIGURE 64. EXAMPLE OF I UNIT TIMING OF AN SS INSTRUCTION



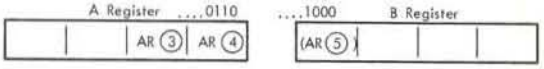
Since the ICR never advances, empty conditions cannot occur; therefore no IC fetches are made.

FIGURE 65. EXAMPLE OF TIMING OF REPEAT INSTRUCTION

ICR LO



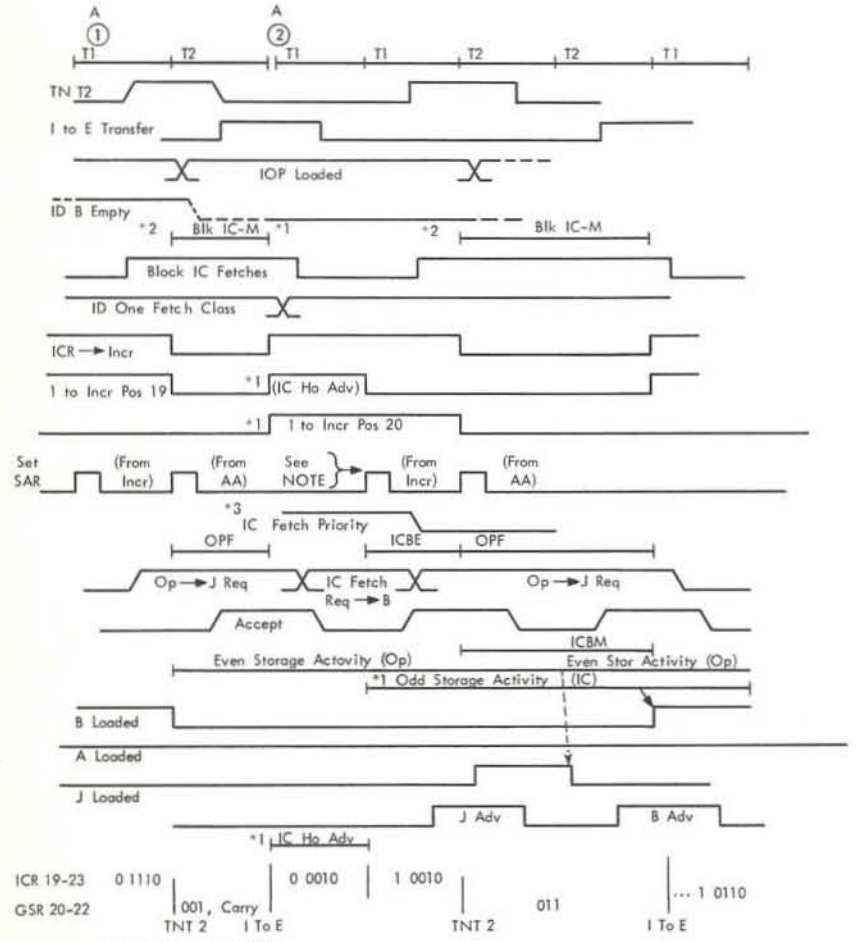
Note that IC fetch address for A is correctly generated in anticipation of A being emptied.



This instruction stream has the property of not interfering with IC fetches.

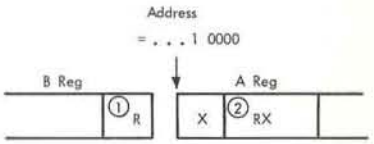
Assume both A and B loaded at start of instruction ④.

FIGURE 66. EXAMPLE OF TIMING OF AN IC FETCH



Following Features are Used:

1. Simultaneous IC Ho Adv and IC fetch address generation
2. Block IC-M
3. IC Fetch Priority Blocks T2



1. RX instructions are FXP ADDs to even storage.
2. Assumed both A and B loaded at start of instruction ①.
3. No IC interference from instructions not shown.

NOTE: On fourth sample SAR is set to ....1 1010.

FIGURE 67. EXAMPLE OF TIMING

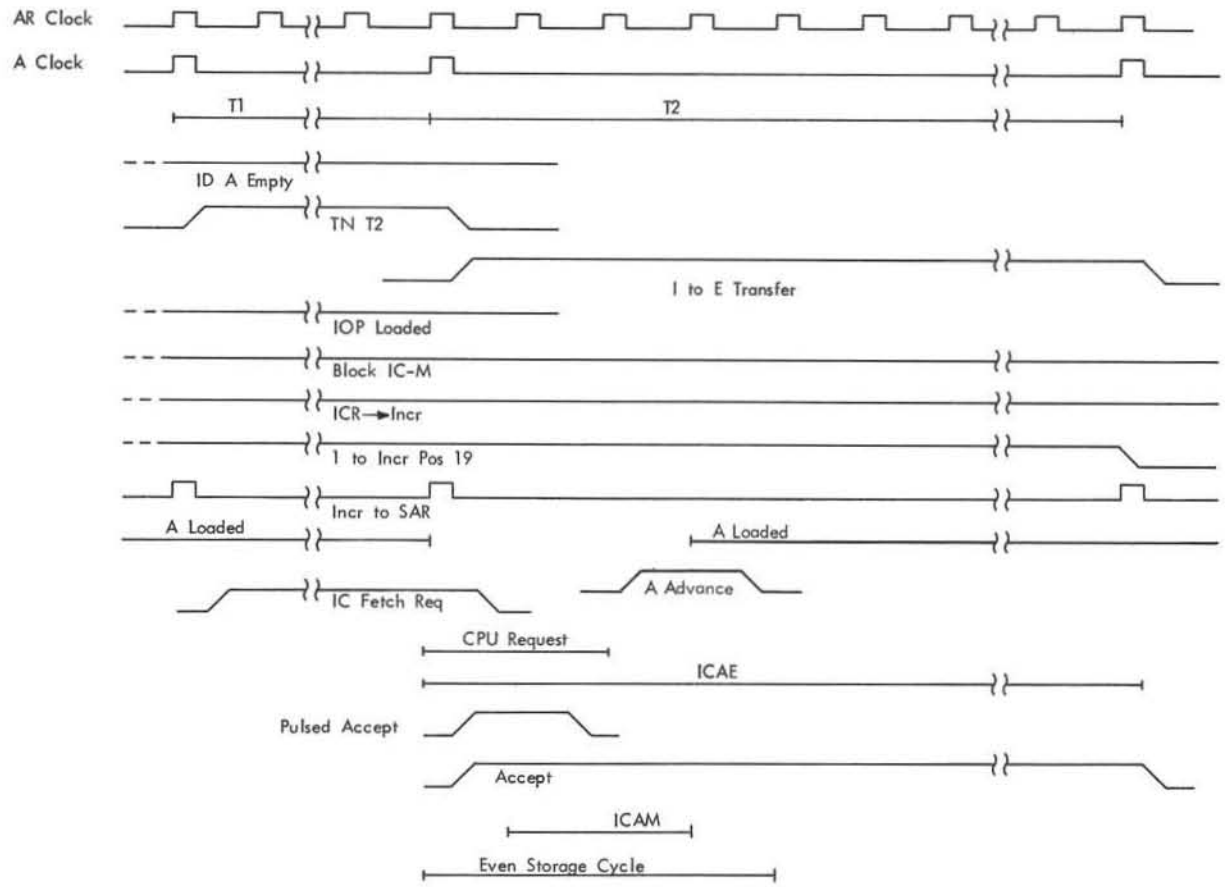


FIGURE 68. EXAMPLE OF IC FETCH--SINGLE CYCLE

ICR Positions			AE · $\overline{BE}$				AE · BE				$\overline{AE}$ · BE				$\overline{AE}$ · $\overline{BE}$			
			Incr	Incr	Reg	Red	Incr	Incr	Reg	Red	Incr	Incr	Reg	Red	Incr	Incr	Reg	Red
20	21	22	19	20			19	20			19	20			19	20		
0	0	0	0	0	A	R	0	0	A		0	1	B		0	0	A	R
0	0	1	0	0	A	R	0	1	A	R	0	1	B		0	0	A	R
0	1	0	1	0	A		0	1	B		0	1	B		1	0	A	
0	1	1	1	0	A		0	1	B		0	1	B		1	0	A	
1	0	0	0	1	A		0	0	B		0	0	B	R	0	0	B	R
1	0	1	0	1	A		0	1	B	R	0	0	B	R	0	0	B	R
1	1	0	0	1	A		0	1	A		1	0	B		1	0	B	
1	1	1	0	1	A		0	1	A		1	0	B		1	0	B	

AE = ICAM + A Loaded

BE = ICBM + B Loaded

Incr 19 = Ctrl Input to Incrementer Pos 19.

Incr 20 = Ctrl Input to Incrementer Pos 20.

A,B Indicates Register Which Is Fetched.

R Indicates Inputs Cannot Occur and Therefore Entry Is Redundant.

FIGURE 69. CHART FOR IC FETCH ADDRESS GENERATION

instruction buffers remain unchanged throughout the repeat instruction.

IC Fetch Rule (Figure 5268)

If either A or B is empty and if no block exists, an IC fetch request is made. If both A and B are empty, the priority circuitry described in the next section determines which register is requested first. This section describes fetches to the A register only: similar logic is used to make fetches to the B register.

At the same time as an IC request is made to the BCU, the trigger instruction counter fetch to A execute (ICAE) is turned on. On any A clock on which an IC request is made, SAR is set from the incrementer. This is controlled by not block IC-M, and is described under "Block IC Rule." ICAE turns off unless it is turned on. It is turned on each cycle a request for A is made, and it turns off on any cycle that a request for A is not made. ICAE has two functions. It is ORed with ICBE to indicate that the fetch is to be returned to the A or B register. If a fetch to A-B is indicated, BCU examines bit 20 of SAR and sets the A return trigger in BCU if SAR20 = 0; and it sets the B return trigger in BCU if SAR20 = 1.

When the accept is received from BCU, ICAE steers the accept to turn on ICAM. ICAM is the IC fetch to A outstanding trigger. At this time A advance turns off ICAM and turns on A loaded.

The full logic for generating the A empty condition of the previous section is described here. A appears empty to the fetch logic if the following expression is 1:

$$A\ \overline{LOADED}^{L} \cdot \overline{ICAM\ Lth} \cdot (\overline{ICAE\ Lth \cdot ACCEPT})$$

$$+ ID\ A\ EMPTY \cdot TON\ T2$$

The first term represents the normal empty condition and the second term is the anticipatory term. Note that $\overline{ICAE}^{L} \cdot \overline{ACCEPT}$ is included in the first term. This quantity makes the register look full immediately upon receipt of the accept for an IC fetch to A. Therefore, ICAE will not stay on after the accept has been generated. The normal fetching is shown in Figure 66.

If the B register is empty at the same time as a fetch to A is being made, it is possible that the fetch request to A is followed by a fetch request to B. The request to B in this case occurs on the cycle following the cycle of the accept for the first fetch. The request to B is not made on the cycle of the accept because the address circuitry (see "IC Address Rule") is not able to respond until ICAM is turned on. This causes no deterioration in performance because of the two cycle bus rate of the BCU.

If the CPU is being single cycled, the sequencing of the IC fetch sequencers is slightly modified. At this time, single clock pulses are generated upon each depression of the start button. All running clocks continue to operate, however, in the normal high-speed manner.

Pulsed accept is a 200-nanosecond latched line which straddles the AR clock following the start of storage activity. Accept rises at the same time as pulsed accept and falls after the first A clock following the start of storage.

The request trigger in BCU (CPU request) turns off with each A clock unless it is turned on by a request from the CPU at a time when accept is not on. The request trigger is also turned off by PULSED ACCEPT · AR CLK. Since single-cycle pulses can be issued at a very slow rate only, any word requested on one A clock will be returned to the CPU before the next A clock. The BCU request trigger is turned off by pulsed accept and cannot be turned on with the next A clock.

ICAE is turned on and off with an A clock. ICAM is turned on and off with an AR clock. A loaded is turned on with an AR clock and is turned off with an A clock. If an IC request is made while single cycling, ICAE will turn on and stay on until the next A clock. ICAM will turn on and off at high speed, and A loaded will turn on when ICAM turns off. See Figure 68.

IC Address Rule (Figures 5269 and 69)

Whenever block IC-M is off, the ICR is gated to the incrementer and an increment amount is added into positions 19 and 20. The amount to be added to the ICR is shown in the chart in Figure 69. The output of the incrementer is gated into SAR if no block IC condition exists. The chart also indicates whether the generated address is for the A register or the B register. This chart is not used for branch fetches or for the first IC fetch during an IC recovery.

This chart indicates how addresses are generated for any other IC fetch situation. AE is equal to the expression $\overline{A\ LOADED} \cdot \overline{ICAM}$, and BE is equal to the expression $\overline{B\ LOADED} \cdot \overline{ICBM}$. AE (BE), therefore, is an unlatched line which is related to the need for a fetch to A(B). AE and BE are used in conjunction with ICR 20-22 to generate the increment amount for the incrementer and are the five coordinates of the chart. Whenever an R appears in the table, that entry represents a situation which can never happen if the machine operates correctly. An analysis of the various entries is:

Consider those entries for which (ICR 20 = 0) · $\overline{AE}$ · BE = 1. This indicates that the ICR is in the A

register, A does not need to be fetched, and B does not need to be fetched. Therefore, good instructions in A are currently being processed, but the B register must be fetched if instruction processing is to proceed uninterrupted. Since instructions are fetched sequentially and the ICR addresses an instruction in the A register, bits 0-20 of the proper address for a fetch to B can be obtained by adding a 1 into position 20 of the ICR. Since SAR bits 21-23 are ignored by the BCU, no problem is created if the output of incrementer positions 21-23 (= ICR 21-23) is set into SAR 21-23 along with the rest of the address.

If (ICR 20 = 0) · AE · $\overline{BE}$, then A must be fetched and B does not need to be fetched. This situation can arise while processing the last instruction in the A register. At least one cycle occurs after the AE condition has been recognized and before the ICR has been updated to the B register. During these cycles (ICR 20 = 0) · AE · $\overline{BE}$ = 1.

(ICR 20 = 0) · AE · $\overline{BE}$ cannot occur until at least one good T1 cycle has been done on an instruction located in the A register. This can be illustrated by considering the situation with (ICR 20 = 0) · AE and with no instructions processed in the A register. The ICR has therefore just left the B register, thereby emptying that register without any fetches being made. If AE = 1, then the A register has been empty for an extended period. The situation is now (ICR 20 = 0) · AE · BE. As will be shown, the A register is fetched at this time. The situation now becomes (ICR 20 = 0) · $\overline{AE}$ · BE. Thus. (ICR 20 = 0) · AE · $\overline{BE}$ cannot occur until an instruction in A is processed.

If the ICR addresses the first halfword of A, A cannot be emptied by an instruction in A because it would take a four-halfword instruction to empty A and no instruction exceeds three halfwords. If the ICR addresses the second halfword of A, A can be emptied only through an SS instruction. For SS instructions, however, IC fetching is always blocked during the period between the recognition of the empty condition and the advance of the ICR and this situation cannot occur. Therefore, the top two entries in the situation now being considered are marked with an R for redundant.

For the other two cases, since B is already loaded, it is necessary to fetch the A register two storage words ahead of the ICR. Therefore, a bit is added to the ICR in position 19.

If (ICR 20 = 0) · AE · $\overline{BE}$, both A and B must be filled. If the ICR addresses any but the first halfword of A, the AE condition occurs as just described for (ICR 20 = 0) · AE · $\overline{BE}$. The only difference arises from the fact that the previously emptied B register has not yet been filled; therefore, the B register is the register to be fetched. The address for B is obtained by adding a bit into position 20 of the incrementer. Again, the situation for the ICR addressing the second halfword of A is redundant because SS instructions block the IC.

If the ICR addresses the first halfword of the A register and AE· BE, there are no loaded instructions available in the instruction buffers. Before the instruction at the address specified by the ICR can be executed, that instruction must be fetched to the A register and the ICR without any modification is used as a fetch address.

The entries for (ICR 20 = 0) · $\overline{AE}$ · $\overline{BE}$ have meaning because a fetch request can be made on the same sample as A loaded is turned off; therefore, it is necessary to generate a correct fetch address in anticipation of an empty condition. An example of this situation is shown in Figure 66. This column is the same as the column for (ICR 20 = 0) · AE · $\overline{BE}$ because the only new empty condition which can be generated while ICR 20 = 0 is the A empty condition.

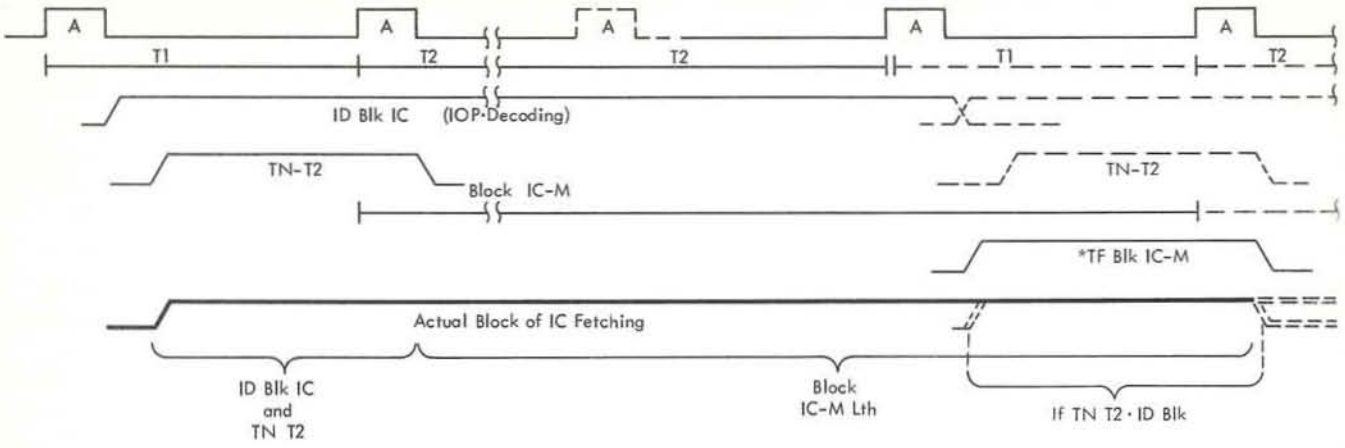
All of the above reasoning holds for entries in the chart for which ICR 20 = 1. For these entries, AE and BE must be interchanged in the discussion.

As described under "ICR Advancing," the generation of IC fetch addresses is not inhibited if there is an IC high-order advance. At this time the ICR address is incorrect because a carry from the ICR low-order must be entered into position 19. While gating the ICR to the incrementer, IC high-order advance forces a 1 into position 19. If an IC high-order advance is taking place, ICR 20-22 = 000 or 001, or 010. If ICR 20-22 = 010, an SS instruction is being executed and IC fetches are blocked by the SS execution. If ICR 20-22 = 000 or 001, the chart indicates that either 0 or 1 in position 20 must be added to the ICR. If the entry is 0, the correct updated ICR (old ICR + 1 in position 19) is obtained at the output of the incrementer for IC fetching. If the chart entry is 1, the (old ICR + 1 in position 19 + 1 in position 20) is obtained at the output of the incrementer. This is the correct value for making an IC fetch. As described under "ICR Advancing," bits 0-20 and 23 of the incrementer contain the correct updated IC high-order value for all cases and are gated back into the ICR at the end of the ICR high-order advance.

During IC high-order advance. IC fetch addresses may also be computed and no special interlocks are necessary to inhibit the IC fetching during this time.

Block IC Rule (Figures 5271, 5272, and 70)

Many instructions block IC fetches to ensure that no interference in the usage of SAR or the incrementer occurs. For all instructions which block the IC, the



NOTE:
In case of branches or the execute instruction, actual blocking may be anticipated by separate logic.

* TF Blk IC-M= IE TF Blk IC-M
+ VFL Ending · No Store Request
+ E TF Blk IC-M · No Branch Op
+ Tsts Cplt · Branch Unsucc
+ Branch LC · Branch Succ M
+ Accept · ID One Fetch Class
+ Accept · E TF Blk T2-M on Accept
+ Accept · VFL Ending
+ Accept · IE TF Blk T2-M on Accept

FIGURE 70. BLOCKING OF IC FETCHING

trigger block IC-M is turned on at TN T2. Block IC-M is turned on by TON T2 · ID BLK IC. IC fetch requests and ICR (from the incrementer) to SAR gating are inhibited by TON T2 · ID BLK IC + BLOCK IC-M^{L}. The ICR, along with the appropriate increment amounts, is gated into the incrementer for IC fetch address computation whenever block IC-M is off. When block IC-M is turned on, this gating is suppressed.

The timing for block IC-M is shown in Figure 70. An example of an instruction stream utilizing block IC-M is shown in Figure 67. The uses of block IC-M by instruction classes are described below.

One Fetch Class: For this class an operand is fetched at TN T2. During the operand fetch the IC controls are denied access to SAR. Upon generation of accept the IC fetch mechanism is again allowed to operate. ID ONE FETCH CLASS^{L} · OPF^{L} · ACCEPT · $\overline{\text{TON BLK IC-M}}$ turns off block IC-M. An IC fetch address is generated which can be set into SAR on the next A clock. This is the earliest time that BCU can accept another storage request because of the two cycle bus rate. Therefore, the I unit controls are able to operate BCU at its maximum rate in this situation, if needed.

The instructions which belong to one fetch class are most of the external fetch instructions for which no storage requests are made during E time. These include instructions such as A, SD, DE, and TM.

STH, STC, CVD, ST, STD, STE, SSK, ISK, RD, STM, MVI, NI, OI, XI, LM and SS Instructions: For these instructions the execution unit make use of SAR for storage fetches and stores. Therefore, the IC controls are inhibited from making any requests until the last storage request of E time has been accepted. At this time the execution unit (I unit controls in the case of SS instructions) generates a line TF block T2-M on accept. This line, which conditions the turn-off of block T2-M is also used to condition the turn-off of block IC-M. The condition which turns block IC-M off is TOF BLK T2-M ON ACCEPT · ACCEPT · $\overline{\text{TON BLK IC-M}}$, except for SS instructions. For SS instructions, ($\overline{\text{SSOP}}^{L}$ · VFL ADR^{L}) · ($\overline{\text{STORE REQUEST}}^{L}$ + ACCEPT) turns off block IC-M. As described under "SS Instructions," $\overline{\text{SSOP}}^{L}$. VFL ADR^{L} occurs at the end of the SS instruction. At this time, if there is no store request outstanding, block IC-M is turned off immediately. If there is a store request outstanding, block IC-M is turned off upon receipt of the accept.

Branches: For branch instructions and EX, IC fetches are blocked until the end of the branch instructions. This gives the branch controls access to both the incrementer and SAR.

BALR (R2 = 0), LA: For these instructions (and certain other instructions), instruction execution makes use of the incrementer. Therefore, block IC-M is turned on to prevent IC fetch address generation from interfering with the instruction execution use of the incrementer. The E unit turns off block IC-M when it has finished using the incrementer.

LPSW: Block IC-M is turned on for this instruction because a fetch is made at TN T2. It is turned off when the new PSW has been loaded and an IC recovery has been started.

Diagnose: Block IC-M is turned on because the IE unit makes a fetch during E time. Block IC-M is turned off by a sequencer (IE 3); it is turned on by the proceed from the PDU. This ensures that the IC fetches will always start at a known time after a diagnose.

Block IC-M is also turned on during interrupt routines. In this case the turn-on of block IC-M does not block IC fetches. A block is not necessary because the interrupt lines directly block IC fetches at this time. Block IC-M is turned off when an IC recovery is started.

Block IC-M is turned on by a computer reset. If the reset resulted from an IPL or a machine check, block IC-M is turned off at the end of the appropriate routine. If the reset resulted from a manual reset, block IC-M is turned off by set IC or set PSW.

Pre-Block IC Rule (Figure 5271)

Whenever the GSR is advanced to a new instruction, the new instruction (if in A-B) is available at the output of the eight-way OR off the A-B registers. If a BALR, BAL, BCTR, BCT, EX, BXH, or BXLE instruction is decoded at this point, IC fetching is blocked. This block does not prevent generation of IC fetch addresses and their placement into the SAR. The block inhibits only the IC fetch request line to the BCU. The line which generates the block is IOP LOADED^{L} · [PD (BALR + BAL + BCTR + BCT + EX + BXH + BXLE)] ^{L}.

The effect of this rule is to block unnecessary IC fetches before branches which have a high probability of being successful branches. Thus, storage activity before the branch fetch is decreased by blocking the IC. This increases the probability that the branch fetch can be made without running into storage conflicts. BCR and BC are excluded from this block because they are not considered to be branches with a high probability of success.

IC Fetch Priority Rule (Figures 5273 and 67)

The action of the previous rules is summarized in the following text.

The empty status of a register is recognized at TN T2 of the instruction that empties that register. IC fetch requests are made to that register unless some instruction in the process of execution generates a block to inhibit IC fetching. This block may occur through the action of block IC-M or through the action of pre-block IC. IC fetch requests may or may not be accepted by the BCU. A fetch request which has not yet been accepted by the BCU may be cancelled at any time by a block that is generated by a new instruction. It is possible for the IC fetch to be made very early, and it is also possible for IC fetches to be blocked out continuously by instruction executions until both registers are emptied. If both registers are emptied, all IC blocks fall and both registers are fetched in proper order.

These rules give low priority to IC fetches. Whenever an IC fetch request is recognized to cause interference with instruction execution, the instruction execution is given priority. This has two advantages: IC fetches cause minimum interference with instruction executions; if IC fetches are not forced prematurely, they will be recognized to be unnecessary if a successful branch is encountered before both registers are emptied. Therefore, an unnecessary storage cycle is not taken. For normal instruction sequences (that is, sequences not containing successful branches), a net performance loss is suffered if both registers are allowed to empty.

With both registers empty, it is necessary to make an IC fetch while the I unit remains idle. This results in a significant number of lost cycles. It is better to take a smaller penalty by forcing IC fetches at some earlier time when instructions are still available for processing. As a result IC fetch priority is incorporated to allow IC fetches to be made as late as possible while minimizing the chances of the I unit being idled by lack of instructions. The action of this rule is accomplished by blocking TN T2 at times which are described in the following text.

TN T2 is blocked by:

$\overline{\text{ID (BROP + RR)}}$ · (GSR 21 = 1 + GSR 22 = 1)

· (GSR 20 = 0 · B FTCH REQUIRED + GSR 20 = 1 ·

A FTCH REQUIRED)

where A FTCH REQUIRED = $\overline{\text{ICAE Lth · ACCEPT}}$ ·

$\overline{\text{ICAM}}^{L}$ · $\overline{\text{A LOADED}}^{L}$

and B FTCH REQUIRED = $\overline{\text{ICBE Lth · ACCEPT}}$ ·

$\overline{\text{ICBM}}^{L}$ · $\overline{\text{B LOADED}}^{L}$.

When we consider the A fetch required situation, note that no block of TN T2 can be generated by this rule until the GSR addresses the second, third, or fourth halfword of B. This incorporates the concept of not forcing IC fetches prematurely.

If the GSR addresses one of the above halfwords and the instruction is an RR instruction, again no block is generated. A block is not generated because no interference is possible between RR instructions and IC fetches (except for ISK, SSK, and the branches). In all probability the IC fetch will be made without any special block to hold up RR execution.

If the GSR addresses a branch instruction (RS, RX branch, or RR branch with R2 $\neq$ 0), again IC fetching is not given special priority. This is for the same reason that the pre-block IC rule was incorporated. It is undesirable to have IC fetches, which might prove unnecessary, made before a branch instruction. If branches were not excluded from bringing up IC fetch priority, there would be a conflict between this rule and the pre-block IC rule.

If the GSR addresses any instruction which starts in the last three halfwords of B and which is neither a branch nor an RR instruction, the IC fetch priority holds up TN T2 until a fetch to A has been made. Since RX instructions are assumed to be the instructions with the highest rate of occurrence, and if A is empty, an instruction starting in halfwords two, three, or four of the B register is probably the last complete instruction available in the buffers. It is the effect of this rule to force any unmade IC fetches before TN T2 of the last probable instruction in the IC buffers. As a result, a new instruction will probably be available in the buffers when needed for execution.

Instruction Fetching--Special Cases

Program Store Compare (Figures 5263 and 71)

If a store is made by the CPU to an address which has been prefetched to the A-B register, it is necessary to refetch that register. In the 2075 refetching is made whenever a store is made to either the storage word addressed by the ICR or to the storage word located in the next location. To implement this feature, two compares are provided which enable comparison between the store address and the ICR, and between the store address and the next address after the ICR. Whenever a store takes place (except stores made by interrupts), these compares are tested without regard to whether or not the instruction buffers have been loaded. If a compare occurs, an IC recovery is taken as an E time interrupt and the buffers are refetched.

	Bits Compared	
Compare	$\overline{\text{IC Ho Adv}}$	IC Ho Adv
H: ICR	0-20	—
H: Incr	0-20	0-19

Store Request Gates → { ICR → Incr; +1 → Incr Position 20 }

IC Ho Adv Gates → { ICR → Incr; +1 → Incr Position 19 }

If all bits in any of the indicated Compares are satisfied when store request is on, the trigger Prgm Stor Cmpr is turned on.

FIGURE 71. CHART FOR PROGRAM STORE COMPARE

Whenever a CPU store (except an interrupt store) is made, the store request trigger (Figure 5263) is turned on. This trigger gates the ICR into the incrementer and adds a 1 into position 20. At this time block IC-M is always on.

At the time that a store address is set into the SAR, the address is also set into H. There are two compares off of the H register. The usage of the compares is shown in Figure 71. The first compare is a full compare of the H bits 0-20 to ICR bits 0-20. The second compare is a full compare of H bits 0-19 to the incrementer output bits 0-19. This second compare can be modified to include bit 20.

If IC high-order advance is off, 0-20 of H is compared to 0-20 of the ICR and 0-20 of H is compared to 0-20 of the incrementer output. Bits 21-23 are ignored because they are not relevant to external storage addresses. If either compare is satisfied while store request is on, program store compare is turned on and an IC recovery will result at the end of the instruction. If IC high-order advance is on, the ICR is two full storage words behind its proper value; therefore, the H to ICR compare is blocked by IC high-order advance. Since +1 is added into incrementer position 19 by the IC high-order advance and +1 is added into incrementer position 20 by store request, the incrementer output is one full storage word ahead of the correct ICR value. Whenever IC high-order advance is turned on, bit 20 of ICR is 0. If the H bits 0-19 are compared to incrementer outputs 0-19, the following compares are effectively generated:

1. 0-20 of H to 0-20 of the correct ICR.
2. 0-20 of H to 0-20 of the next address that follows the correct ICR.

If the H to incrementer compare is satisfied, program store compare is again turned on.

Interrupt Entry

Whenever an interrupt is detected and one of the three interrupt lines into the I unit sequencing controls rises, IC fetches are immediately blocked. The IC fetch controls may still attempt to make IC fetches, but since no storage unit can be started, it is impossible for the request to be acknowledged. All IC fetching is terminated by the first interrupt sequences.

At this time the IC controls are reset and block IC-M is turned on (unless the interrupt is an IC recovery only).

IC Recoveries (Figures 5274 and 72)

During an interrupt routine, the instruction LPSW, or the manual operation set IC or set PSW, a new value is placed into the instruction counter. It is necessary to refetch the A and B registers. At the appropriate time after the IC has been loaded, a signal is generated by the unit that loads the IC. This signal starts an IC recovery. A typical example of an IC recovery is shown in Figure 72. This signal resets A loaded, B loaded, block IC-M, and block T1-M; and it also sets IC recovery.

IC recovery^{L} blocks the control inputs into the gate select adder and gates the gate select adder into the GSR with an A clock. Thus, the GSR is correctly loaded for the first instruction.

IC recovery destroys the normal method for generating an IC fetch address and for establishing priority between the A and B registers. The ICR is gated directly through the incrementer to the SAR. Since block IC-M is off, an IC fetch is now made. This fetch is from the location that is addressed by the ICR and it goes to the instruction buffer indicated by ICR 20.

IC recovery^{L} resets J loaded to ensure that it is not left on if a previous instruction failed to reset it.

IC recovery is reset by PULSED ACCEPT · (ICAE^{L} + ICBE^{L}) · AR CLK; therefore, IC recovery turns off when the first fetch is accepted. The IC fetch controls now operate normally and they correctly fetch the next storage word for the second instruction buffer. T1 turns on a cycle after block T1-M is turned off and instruction sequencing now proceeds normally.

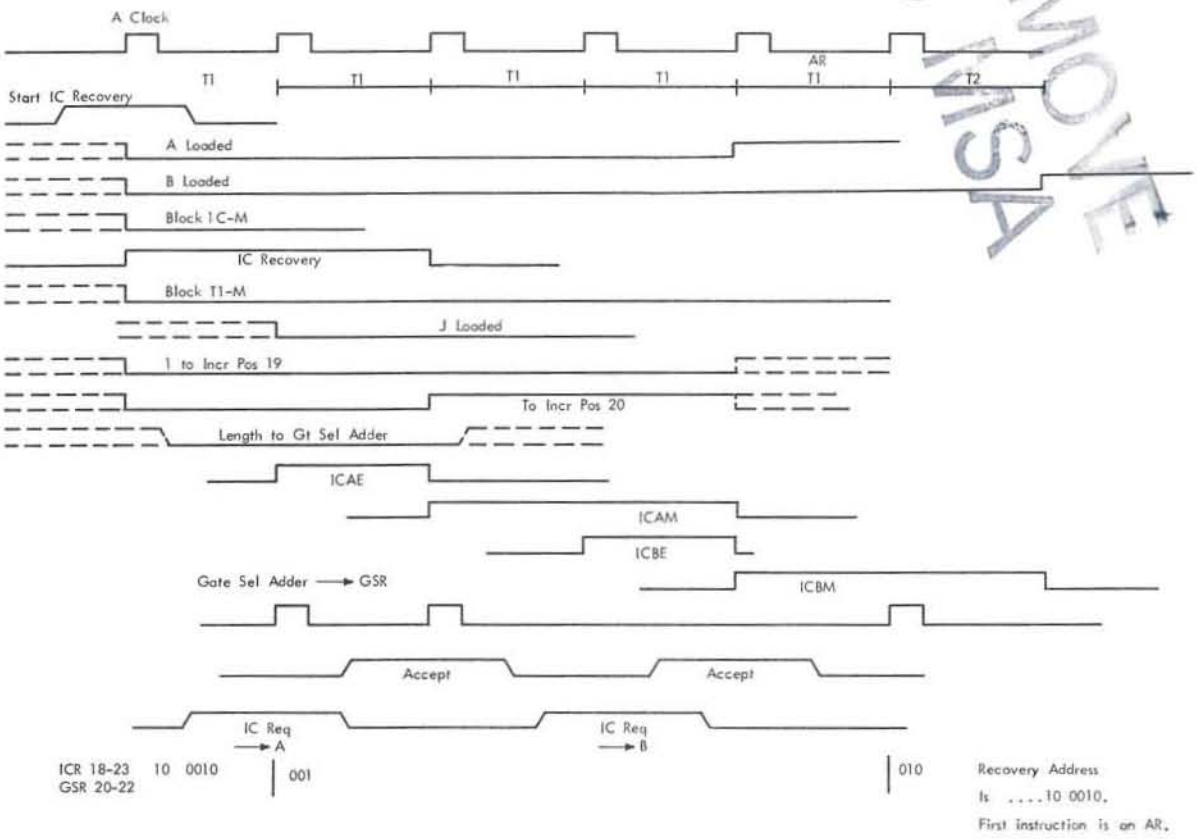


FIGURE 72. EXAMPLE OF TIMING FOR AN IC RECOVERY

INTRODUCTION, FLT

- Tests are executed as they are read from magnetic tape.
- Each test checks a single CPU element or a small group of circuits.
- The status of a single CPU indicator shows the result of the test.
- Each test is executed eight times before the next test is tried.
- A hardware failure stops the testing; a diagnostic index reference is displayed in the test register.
- Each test consists of five major steps:
  1. Load the test into storage.
  2. Scan into the CPU triggers.
  3. Advance the control clock (optional).
  4. Compare a trigger status to an expected result.
  5. Execute the next test or terminate the testing.

Fault location testing (FLT) on the 2075 is a maintenance aid. Special FLT hardware in the CPU, in conjunction with test words read from tape, enables the customer engineer to check out the CPU circuits. The tests run until a failing circuit is found; the failing circuit stops the testing and displays in the test register the number of the test that failed.

The customer engineer then looks up the failing test number on a listing; the listing shows the test points of the circuits that could cause the failure, and the voltage levels (up or down) that should appear at these points. He may replace the questionable cards directly, or he may repetitively run the failing test and scope the test points to locate the failure.

The FLTs do not test the CPU functionally as do diagnostic programs. For example, a diagnostic program tests the multiply circuits by actually multiplying two numbers and comparing the product to a predetermined result. The FLTs, however, test the multiply circuits (and all other circuits) one element at a time: each trigger is checked to see that it will turn on and off; where practicable, the inputs of each AND circuit are conditioned and a check is made to see that the output is correct, etc. The entire CPU is checked in this manner.

There are approximately 100,000 tests on the tape. Each tests a single element or a small group of elements for a single condition or output. For example, 72 separate tests from the tape check the 72 triggers of the J register to see that they can turn on. Seventy-two other tests check the same triggers to see that they can turn off. After all of the triggers are tested (control triggers as well as register positions), logical groups of circuits are tested, such as the adders, decoders, etc.

NOTE: Although in the preceding and following descriptions, the advance portion of the FLT operation is considered to be the number of cycles that the particular test calls for, all of the FLTs used currently call for no advance or for an advance of only one machine cycle.

To summarize, FLT will:

1. Read a test from tape and load into storage
2. Scan into the CPU triggers from the test words just loaded into storage
3. Advance the controlled clock a predetermined number of cycles
4. Compare a single selected indicator with the expected result bit
5. Run the next test in sequence or terminate the testing

To test a group of logic, the inputs to the logic must be properly conditioned so that a test of the output will be meaningful. Each test on the tape, therefore, sets up the CPU to the proper status (certain triggers turned on) that results in a particular output of the logic in question. This output is one of the indicators on the system control panel. Any one of 1216 indicators on the panel can be selected as the indicator to reflect the status of the logic being tested. After the CPU is set up for a test (scan in), the proper indicator is selected and its status is compared with an expected result bit in a register that was also set as part of the CPU set-up. If it is an equal compare, the test passed; if it is an unequal compare, the test failed.

In testing many of the circuit groups, it is not enough merely to condition their inputs and then have the output checked. Some logic requires that after the set-up a number of machine cycles must occur before the logic reaches a state that can be tested. Therefore, after the set-up (scan in), the controlled clock is started and allowed to run for a predetermined number of cycles (advance cycles). Then the comparison is made between the selected indicator and the expected result bit.

Test Tape Format

The test tape contains, in order: one IPL control record consisting of three channel command words (CCWs); one channel control record consisting of five CCWs; a succession of 17-word tests blocked with approximately 50 tests per record (Figure 73).

The IPL record performs exactly as in the initial program load procedure: the first three double words (CCWs) are read from tape into double word storage locations 0, 1, and 2. The channel obtains the CCW from location 1 and uses it to direct the reading of the next record, the channel control record (five CCWs), into storage, starting at double word location 16. The channel control record consists of the CCWs that will cause the channel to read the succeeding 17-word tests alternately into the storage buffer areas.

The first few test records on the tape, hard core tests, check out the FLT circuits. These tests check the FLT hardware in three levels. The first-level tests are executed manually by single cycle, the results being observed in various indicators; they check the FLT sequencing triggers and the scan-in paths to the J register.

The second and third-level tests are executed automatically; the second level checks out the FLT seek circuits; the third level checks out the FLT switch matrices.

Following the hard core tests are the CPU FLTs that require no advance cycles (zero-cycle tests), which test the CPU triggers for turn-on and turn-off. Following these tests are all the other CPU FLTs, ones that require an advance of one cycle or more.

The last test in each record is a dummy test. Its function is to keep the channel from reading across the interrecord gap before the result of the last real test is known so that after a last real test is executed, a halt or a backspace of tape can be executed before the interrecord gap is reached.

Each test consists of seventeen 64-bit words, organized in the following manner:

Word 0, Test Number: Bits 46-63 contain an 18-bit number which is used to identify the test. Bits 0-45 are not used.

Words 1-15, Test Input Pattern: This is a 960-bit field which is used to specify the initial state of the CPU. Each bit in this field corresponds to a trigger in the CPU, and the triggers are set on or off depending on whether the bits are 1s or 0s.

Word 16, FLT Control Word: This word contains the following fields:

BITS 19-21, CLOCK PULSE SELECT: These bits select the kinds of control clock pulses that will be emitted during the clock advance portion of the test. Bit 19 selects the A pulses, bit 20 selects the early B pulses, and bit 21 selects the B and late B pulses. These three bits may be in any combination.

BITS 22-25, CLOCK ADVANCE FIELD: This field contains the number of control clock cycles that must be executed before the trigger specified by the comparison bit address is compared to the expected result bit.

BIT 32, UNCONDITIONAL TERMINATION BIT: This bit causes the testing to stop after the current test, regardless of whether the test passed or failed.

BIT 33, CONDITIONAL TERMINATION BIT: This bit causes the testing to stop after the current test if the trigger specified by the comparison bit address does not match the expected result bit.

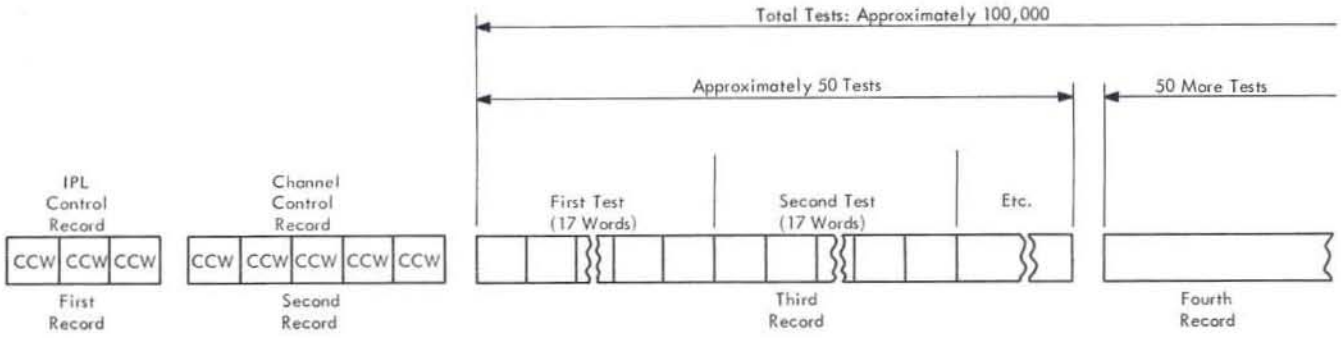


FIGURE 73 TEST TAPE FORMAT

BIT 34, EXPECTED RESULT: This bit is compared to the state of the trigger specified by the comparison bit address. If they match, the test passed; if they do not match, the test failed.

BITS 34-35, COMPARISON BIT ADDRESS: This field specifies which of the possible 1216 triggers will be compared to the expected result bit.

BITS 46-63, DIAGNOSTIC INDEX NUMBER OR ALTERNATE TEST NUMBER: Normally, this field contains the number of the current test; therefore, on a stop, the failing test number is displayed in the test register.

If desired, on the passing of a test, certain succeeding tests on the tape may be skipped, the next test to be executed being selected by the alternate test number. This method of operation requires that no termination bits be present and that the fail trigger be off after the eighth execution of a test. The FLT next-test-selection circuits have the capability for this mode of operation, but it is expected that this mode of operation will not be used and that all tests on the tape will be executed in sequence, with no skips.

FLT Sequence

Figures 6200 and 6201 show the sequence of events and the data flow of FLT. Throughout the discussion, refer to the flowchart and data flow diagram.

There are five major steps in an FLT operation:

1. Load the tests into storage
2. Scan the test pattern into the CPU triggers
3. Advance the control clock a number of cycles
4. Compare the status of a selected trigger to an expected result
5. Select the next test to be executed or terminate the testing

Load the Tests

To load the first record from tape into storage, the customer engineer follows a procedure similar to the one used on initial program load. The FLT mode switch is set to FLT, the rotary switches are set to the addresses of the channel and tape unit, and the load pushbutton is pressed. The system resets and the tape unit starts. The first record is read from tape (three double words) and placed in storage locations 0, 1, and 2 (double word locations).

The channel then obtains the command from location 1 (a read command) which causes the next 40 bytes, the second record on tape, to be read and placed in storage, starting at double word location 16. This record consists of the five CCWs that will load into storage all of the following test records on the tape.

After the 40 bytes are placed into storage, the channel obtains the next command from location 2 because of command chaining. This second command is a transfer-in-channel, which transfers control to the command at location 17. The command at location 17 reads the next 136 bytes (17 test words) into storage area A, which starts at double word location 16,384.

After the 136-byte read is completed, the channel fetches the command at location 18 because of data chaining. The command at location 18 transfers control to the command at location 19 and sends a transfer-in-channel (TIC) pulse to the FLT controls. The command at location 19 reads the next test into storage area B, which starts at double word location 16,416.

After storage area B is filled, channel control is transferred back to the command at location 17 and another TIC pulse is sent to the FLT controls. In this manner, the channel loads tests alternately into areas A and B as fast as data can be read from the tape.

Each TIC pulse signals the FLT controls that a buffer area in storage has just been filled with a test. On receipt of the TIC signal, the FLT controls execute the test eight times, while the next test is being loaded into the other buffer area. The CPU is fast enough so that even after executing a test eight times, the FLT controls are idle, waiting for the next TIC pulse.

Scan In

Scan in is the part of the FLT sequence that sets up the CPU triggers according to the 15 test pattern words in a storage buffer area. The control clock is not running during scan in; a special clock, the scan clock, creates the timed pulses necessary to make the storage requests and gate the data from storage into the CPU triggers. The scan clock, once started, goes through seven stages (0-6) and then stops.

The TIC pulse from the channel makes the initial storage request. The BCU takes the address for the storage operations from the channel SAB positions as follows:

- 0-5 - Always 0
- 6 - Always 1
- 7-14 - Always 0
- 15 - Storage section trigger
- 16-20 - Word control counter positions 0-4

Because the storage section trigger and the word control counter (WCC) are reset initially, the first location addressed in storage is double word location 16,384 which is the beginning of buffer area A and the location of a test-number word. This first double word is fetched from storage and placed on the channel SBO. Storage advance starts the scan clock. The WCC being 0 indicates that the word on the channel SBO is a test-number word; but since the test-number word is used only when seeking a new test to run, and the seek trigger off at this time indicates no seek, the test-number word is allowed to "die" on the channel SBO. The WCC is stepped to 1 and another storage request is made, this time by a scan clock pulse.

Because the WCC is now 1, the second storage fetch is from address 16,385, which is the first test-pattern word. Storage advance again starts the scan clock, and since this word is a test pattern word, it is returned to the J register. The WCC = 1 is decoded to gate the contents of the J register into a certain group of 64 CPU triggers. The WCC is stepped to 2 and another storage request is made.

This process continues, with the WCC addressing storage for the pattern-word fetches and also picking out the group of 64 triggers to put the pattern bits into, until all 15 pattern words have been scanned in. At the end of the fifteenth scan in, the WCC has been stepped to 16.

The scan in completed allows the FLT sequence to go to the advance control clock portion.

Advance Control Clock

The next storage request fetches the double word from the seventeenth double word location of the buffer area, which is the location of the FLT control word. The FLT control word designates how many machine cycles will now be taken (clock advance), which clock pulses are to be used, which trigger will be examined afterwards for its status, what the expected result is, and the number of the next test to be executed if the current test passes all eight times. Storage advance again starts the scan clock, and since the WCC = 16, the FLT control word now on the channel SBO is set into the FLT control word registers.

The WCC = 16 and a scan clock pulse start the CPU control clock. When the FLT control word was set into the FLT control word registers, one of the areas set up was the advance counter. The advance counter was set to correspond to the number of cycles that the control clock should run, and now as the control clock runs, the advance counter is stepped down until it reaches 0. When the advance counter reaches 0, it stops the control clock.

The advance completed allows the FLT sequence to go on to the compare portion.

Compare

Advance completed causes the FLT controls to enter the compare portion of the sequence by turning on the compare cycle trigger and starting the scan clock. During scan in, the WCC was used to address storage and to scan the pattern words into the proper groups of triggers. At the end of scan in, the FLT control word was set into the control word registers, two of the areas set being the word control counter and the bit control register. The WCC now is used as a register, not a counter, and in conjunction with the bit control register, picks one trigger out of the possible 1216 to be compared to the expected result bit.

The compare cycle trigger and a scan clock pulse make the comparison between the selected trigger and the expected result bit. If it is an equal compare, the pass trigger is set; if it is an unequal compare, the fail trigger is set. This concludes one of the eight times that the current test must be executed before it is decided what to do with the results of the testing.

A repeat counter is used to count the number of executions of a test; one execution scans in the 15 pattern words, sets the FLT control word registers, advances the control clock, and compares the selected trigger to the expected result. The repeat counter starts at count 0 and steps up 1 after each execution. After the eighth execution (denoted by the repeat counter at 7), the FLT operation is allowed to go on to the next portion of the sequence, which is the selection of the next test or the termination of the testing.

Select Next Test or Terminate

As soon as a comparison is made with the repeat counter equal to 7 (the eighth comparison), the fail trigger is examined along with the unconditional and conditional termination bits to decide upon a course of action.

The FLT seek circuits were designed to allow for automatically skipping certain lower-level tests if a prior higher-level test passed, and conversely, to execute the lower-level tests only if a higher-level test failed. This concept of testing is the reason for having conditional terminations.

The current concept of testing, however, is to execute all the tests on the tape, with no skips, and to halt the testing on the failure of any of the tests. To

implement this concept, it is likely that all tests will have a conditional termination bit. The conditional termination bit in conjunction with the fail trigger on stops the testing.

Transmission Checks During FLT

When the channel discovers a data check, it sends the scan data check signal to the FLT controls. Then the channel automatically backspaces one record (by transferring to location 16 in its control program) and starts reading into area A again.

The FLT controls will, upon receipt of the scan data check signal, step the retry counter, reset the CPU, and wait for a TIC pulse from the channel. When the TIC pulse is received, the FLT controls start the testing from storage area A, repeating the tests from the record that had the data check. If the checks persist, the record will be retried up to 64 times. If 64 attempts to complete the record are unsuccessful, testing stops and the manual intervention required indicator turns on (W32).

If a control check occurs in the channel, the channel stops transmission and sends the scan control check signal to the FLT controls. The signal stops the testing immediately and turns on the manual intervention required indicator.

If the BCU recognizes an address check, data check, or invalid address check, this information is sent to the FLT controls. On any of these checks, a stop scan signal is sent to the channel to stop it from transmitting data, followed by a start scan signal to make it backspace the record and start reading into storage area A again. The CPU and the FLT controls are reset, and the retry counter is stepped. The FLT controls then wait for a TIC pulse from the channel. When this pulse is received, the FLT controls start the testing from storage area A, repeating the tests from the record involved. After 64 unsuccessful attempts to complete the test record, automatic testing stops with the manual intervention required indicator on.

Manual Controls for FLT

- A single test can be run repetitively.
- A test can be single-cycled.
- The FLT control word registers can be set up from the data keys.
- Testing can be restarted at any point on the tape.

FLT Mode Switch: This switch enables the system to run fault locating tests. In FLT mode, CPU storage requests and CPU select channel lines are blocked, the initial program load function is modified, and the channel specified by the rotary switches is conditioned for FLT operation.

Repeat FLT Switch: When the system is in FLT mode, this switch allows the fault locating test specified by the test register to be executed repetitively, independent of normal branch and stop conditions. The system reset or stop FLT pushbutton will halt the testing.

Single Cycle FLT/Log Switch: This switch prevents the normal turn-on of the scan clock, but allows the scan clock to run through its seven stages whenever the start FLT/log pushbutton is pressed; thus, an FLT or a logout operation may be stepped by increments of the total scan clock.

Load FLT Control Word Pushbutton: When the FLT mode switch is active, this pushbutton causes the contents of data keys 19-63 to be set into the FLT control word registers:

Data Keys	FLT Control Word Field
19-21	Clock pulse select
22-25	MCW counter (clock advance field)
32	Unconditional termination trigger
33	Conditional termination trigger
34	Bit-compare trigger
35-39	Word control counter
40-45	Bit control register
46-63	Test register

In addition, seek mode is set. If testing is then restarted via restart FLT I/O, the first test to be executed will be the one whose test number matches the contents of the test register.

Stop FLT Pushbutton: When the system is in FLT mode, this pushbutton halts the FLT controls at the end of the test currently being executed. Channel transmission is stopped and the channel disconnects at the end of the record.

Start FLT/Log Pushbutton: When the system is stopped in FLT mode, this pushbutton restarts the execution of the test last performed. The test will be repeated until either stop FLT or system reset is pressed. The test is reread from the A/B buffer area of storage; the channel and tape are not activated. If the single cycle FLT/log switch is on, the action of the start FLT/log pushbutton is as described under "Single Cycle FLT/Log Switch."

Restart FLT I/O Pushbutton: When the system is in the FLT mode, this pushbutton initiates a backspace record in the channel. The channel then rereads the record into storage area A, and the normal automatic test routine is resumed.

Indicators for FLT

Manual Intervention Required (W32) PR 031: This trigger turns on to indicate that the result of the test is not valid and that manual intervention is necessary. This situation can be brought about by persistent control checks or data checks from the channel, or persistent address, data or invalid address checks from the BCU.

The manual intervention trigger is turned off by system reset.

Pass, Fail (W34, 35) PR 201: During the compare cycle of each of the eight executions of a test, the pass trigger or the fail trigger is set to indicate whether the execution passed or failed. The triggers are turned off after the eight executions and after the decision is made whether to select the next test, seek the next test, or stop the testing.

Seek (W36) PR 271: This trigger turns on during the select-next-test portion of the FLT sequence (after the eighth execution of a test) to cause a seek of the next test. A seek of the next test is called for if there are no termination bits and the current test passes, or if the FLT control word is entered manually.

The seek trigger causes each succeeding test to be rejected until a test is found whose number is equal to the number in the test register. An equal comparison indicates that the next test to be executed is now in storage buffer A or B and is to be scanned in.

Loop On Test (W37) PR 271: This trigger causes the same test to be executed repetitively, without stopping. Loop on test is activated when the repeat FLT switch is on or when the start FLT/log pushbutton is used for starting. System reset or the stop FLT pushbutton will halt the testing.

Storage Section (W39) PR 271: This trigger determines whether storage buffer area A or storage buffer area B is addressed when the FLT controls send an address to the BCU. The FLT operation starts with the storage section trigger off, which causes the first scan in to be from area A; the trigger is complemented after the eighth execution of each test to cause the next scan in to be from the alternate storage area.

Advance Cycle (W41) PR 241: This trigger turns on as the FLT control word is reading out of storage (a scan clock pulse and WCC = 16) to set the word into the FLT control word register and to start the control clock. The advance cycle trigger also starts up D1, D2, and D3 sequence; these three triggers monitor the advance counter so that the control clock can be stopped when the count is reduced to 0. The advance cycle trigger turns off when the compare cycle trigger turns on.

Compare Cycle (W42) PR 251: This trigger is on for one cycle following the advance portion of the FLT operation to make the comparison between the selected CPU trigger and the expected result bit. The pass trigger or the fail trigger is set as a result of the comparison.

Scan In Cycle (W43) PR 241: The TIC pulse from the channel makes a storage request. Storage advance turns on the scan clock. Scan clock A2 pulse turns on the scan in cycle trigger. The scan in cycle trigger and scan clock A2 pulses gate the 15 test pattern words from the J register into the CPU triggers. The scan in cycle trigger turns off when the advance portion of the test is entered.

Intermittent (X32) PR 201: This trigger turns on when the pass and fail triggers are on at the same time and indicates that the current test both passed and failed sometime during the eight executions; the machine malfunction, therefore, is intermittent.

The intermittent trigger serves only to turn on the indicator. The trigger is reset by the start FLT/log pushbutton (not in single cycle), the restart FLT I/O pushbutton, system reset, or when the retry counter is stepped.

Stop (X34) PR 151: This trigger stops the testing by blocking further storage requests and sends a signal to the channel to stop the tape. The trigger turns on by (1) normal stop conditions, that is, an unconditional termination or a conditional termination with a failure, (2) by the stop FLT pushbutton, or (3) by any condition that brings about manual intervention required.

No Compare (X36) PN 181: This indicator is from the exclusive OR circuit that compares the status of the selected CPU trigger (the output of the bit switch matrix) with the expected result bit (the bit compare latch). The no compare indicator turns on when the comparison is unequal.

Clock Select (X37-39) PR 061: These triggers are a part of the FLT control word register. The triggers are set from channel SBO positions 19, 20, and 21 by the pulse set FLT control word.

Trigger A (X37), when on, allows the emission of A pulses during the clock advance portion of the FLT sequence; trigger EB (X38) allows emission of the early B pulses; and trigger B (X39) allows emission of the B pulses.

Storage Request (X40) PR 151: This trigger makes the storage requests of the BCU that are needed by the FLT operation. The trigger turns off when the BCU returns the accept pulse.

Logout (X42) PR 301: The logout trigger turns on by (1) the log-on-check signal, which results from any machine check that is not disabled, or (2) by the logout pushbutton when the CPU is stopped. The logout trigger stays on until the 19 double words of indicator information have been stored.

Log Complete (X43) PR 301: This trigger turns on as the last double word of log information is stored and causes one of the following:

1. If the logout was the result of a log-on-check, the CPU is reset and a machine check interrupt is initiated.
2. If the logout was the result of pressing the logout pushbutton, the CPU stops. No reset occurs.

INTRODUCTION, LOGOUT

- Logout operation stores the status of 1216 CPU triggers into storage.
- Logout can be automatic following a machine check or can be caused manually by a pushbutton.

The state of the CPU is expressed in terms of the binary states of the various triggers that make up its registers and controls. Logout is the way in which this state of the CPU is stored into fixed locations of storage. During logout, the binary states of 1216 triggers are stored as the bits of 19 double words; the words are stored starting at storage location 16. Figure 74 shows the data flow of logout.

The logout process is executed by the scan controls in the PDU. Most of this hardware is the same as that used on FLT, such as the word control counter, the word switch matrix, the scan clock, and so on.

For a logout to have meaning, the control clock in the CPU must be stopped, which will always be the case. Logout can be initiated two ways:

1. By a log-on-check signal from the CPU. The log-on-check signal results from any machine check that is not disabled.
2. By pressing the logout pushbutton when the CPU is stopped.

Either of these conditions starts the logout process by making a store request of the BCU. The BCU takes the storage address from the channel SAB positions as follows:

0-14 - Always 0
15 - WCC position 0
16 - Not WCC position 0
17 - WCC position 1
18 - WCC position 2
19 - WCC position 3
20 - WCC position 4

Since the WCC starts out at all 0s, the first address taken by the BCU is for double word location 16. Also, the word control counter, through a decoder, selects word 0 of the word switch matrix. The word switch matrix is fed by 19 groups of 64 indicator drivers, and the selection by the WCC of word 0 picks out one of the groups to be gated from the output of the matrix. These 64 trigger indications are placed on the SBI and are taken by storage into location 16 whenever the storage store cycle occurs.

Storage advance turns on the scan clock whose pulses step the WCC to 1 and make another storage request. Storage address 17 is now sent to the channel SAB and word 1 is selected at the word switch matrix (another group of 64 indicator drivers). The storage cycle stores the second group of 64 trigger indications, and storage advance again turns on the scan clock, which again steps the WCC and makes a storage request. This process continues until all 19 groups of trigger indications are stored.

The storage advance signal, as the nineteenth word is stored, generates a log complete signal. The log complete signal causes one of the following:

1. If the logout was the result of a log-on-check (a machine check occurred), the CPU is reset and a machine check interrupt is taken.
2. If the logout was the result of pressing the logout pushbutton, the CPU stops. No reset or interrupt occurs.

The two logout indicators (logout and log complete) are described in the "Indicators for FLT" section.

Logout may be single cycled by the following procedure:

1. Turn on the single-cycle FLT/log switch
2. Press the logout pushbutton
3. Press the start FLT/log pushbutton for each run of the scan clock

INTRODUCTION, MAINTENANCE CONTROL WORD

The maintenance control word register (MCW) is primarily a test register, whose main purpose is to dynamically test the various error checking stations in the CPU or in a selected channel.

The diagnose instruction loads the MCW register from the storage location specified in the address portion of the instruction (Figure 75). Once the instruction is executed, certain operations of the CPU and/or a selected channel are directly affected by the bits in the MCW. These bits control their corresponding operations in the CPU or channel until the bits are removed from the MCW by another diagnose instruction, or until the MCW is cleared by system reset.

NOTE: MCW position 3 is reset not only by system reset but also by computer reset when the CPU is selected (position 7 off).

A feature of MCW control is the automatic stop and logout of the CPU at a predetermined number of machine cycles after the execution of the diagnose instruction. A portion of the MCW is a cycle counter, which is set initially on the MCW load to the desired number of cycles. Following the diagnose instruction execution, normal instruction processing continues, but with the MCW bits controlling their corresponding operations and with the cycle counter being decreased by 1 each machine cycle. When the counter reaches 0, a pseudo machine check is forced which stops the CPU and initiates the logout. If the proper bits are initially loaded into the MCW, and if proper operands are selected for the instructions that follow the diagnose instruction, errors can be forced in the CPU or the selected channel and the results of the errors recorded in the storage logout area. In this manner, the error detection circuits can be tested.

The diagnose instruction is a privileged instruction; that is, it is valid only in the monitor state. Because of its effect on system operation, the status of the system is considered as having been switched to a diagnostic state after the execution of the diagnose instruction.

The effective address of the diagnose instruction must specify a double word storage location. The leftmost single word of this location loads the MCW register.

MCW Control

The MCW register indicators are at console locations M16-50.

MCW 0, 1 and 2, Channel Decode: These three positions are decoded to select one of the seven channels to be placed under MCW control. The bits and their corresponding selection are:

MCW Positions			Selection
0	1	2	
0	0	0	Channel 0
0	0	1	Channel 1
0	1	0	Channel 2
0	1	1	Channel 3
1	0	0	Channel 4
1	0	1	Channel 5
1	1	0	Channel 6

Any channel selected causes a diagnose select channel signal (a simplex line) to be sent to that channel, which causes the channel to simulate the I/O interface. A bit in MCW 7 gates the multiplex diagnostic lines from the MCW into the selected channel. The multiplex diagnostic lines to the channels originate at MCW positions 3, 4, and 5.

MCW 3, Force Carry: This bit has two functions, one for channels and one for the CPU. If channels are selected (by MCW 7), bit 3 activates a reverse data parity line in the channel specified by MCW 0, 1, and 2 that (1) causes a reversal of the parity bit that comes out of the channel's simulate I/O register (which feeds the bus in), and (2) blocks the parity checking of the bus in. For the channel, therefore, bit 3 simulates the channel's reading out of a bad byte from an I/O device.

If the CPU is selected (by not MCW 7), bit 3 causes a reversal of the half-sum parities in the main adder and the exponent adder. This causes an adder error only if the carry circuits do not call for a reversal of the half-sum parities; therefore, by performing an addition with selected operands, and with MCW bit 3 on, the choice can be made between correct and incorrect parity bits for each byte in the data word.

MCW 4, Reverse Parity: This bit, enabled by MCW 7, reverses the parity at the selected channel's byte counter, causing the byte counter parity to be incorrect when the byte counter is updated. This bit provides a means of testing the byte counter check circuits.

MCW 5, Block Set: This bit, enabled by MCW 7, causes the channel to block the setting of a control check or data check due to incorrect parity on a CCW or a data word fetched from storage. This

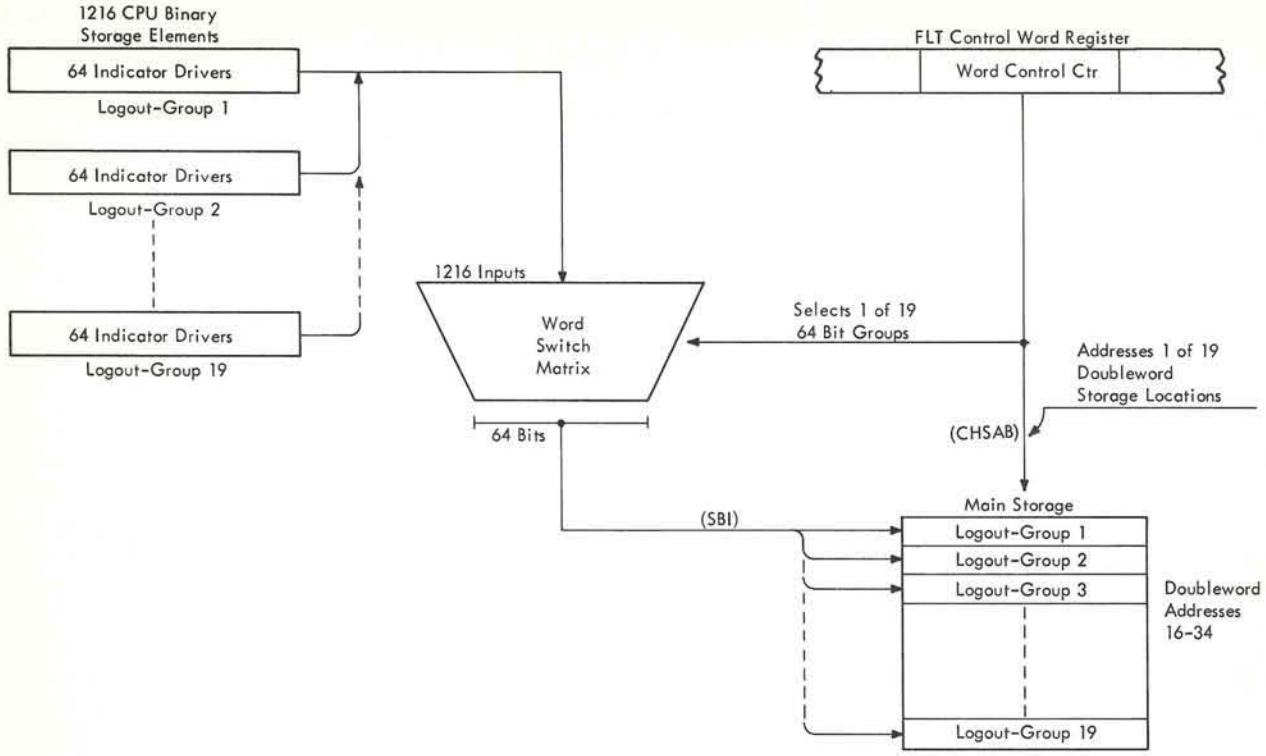


FIGURE 74. LOGOUT DATA FLOW

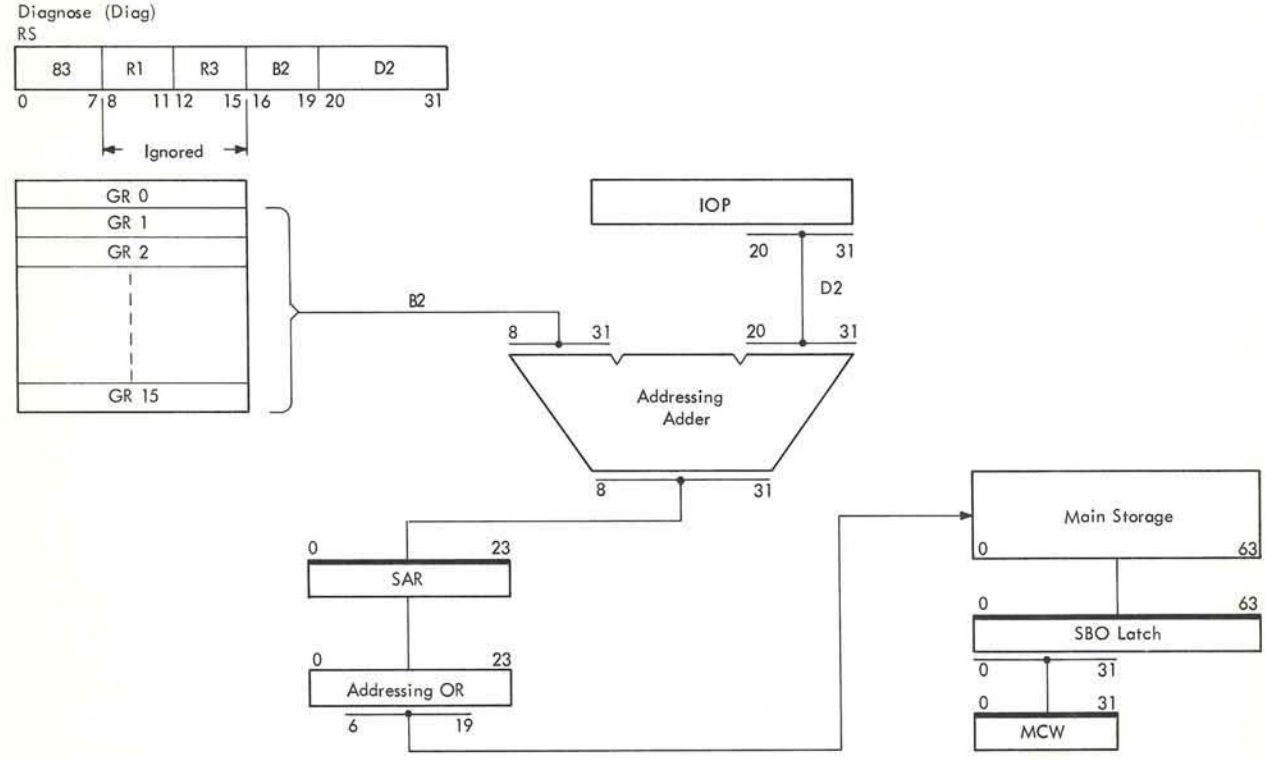


FIGURE 75. DIAGNOSE INSTRUCTION

function allows invalid CCWs to be brought into the channel to test sections of the channel's check circuits.

MCW 6, Send Stop: This bit conditions the diagnostic stop and logout feature of the MCW. If CPU is selected, (by not MCW 7) and position 6 is on, the CPU stops and a logout occurs at a predetermined number of machine cycles after the execution of the diagnose instruction. The number of machine cycles that occur is determined by the count set into the MCW count field by the diagnose instruction. This automatic stop and logout function allows the CPU to progress through an instruction (or instructions) only to the extent specified by the initial MCW count.

MCW 7, Select Channel: This position, if on, selects channels for MCW control; if off, the position selects the CPU.

MCW 9, Channel Mark Parity to 1: This position is independent of any selected unit. Bit 9 causes the channel mark parity sent to the BCU by any channel to be a 1, thereby allowing the storage mark parity checking station to be tested.

MCW 10, Channel SAB Parity to 1: This position is independent of any selected unit. Bit 10 causes the channel SAB parity bits for bytes 0-7 and 8-15 to be 1s, thereby allowing testing of the parity checking station at the BCU's addressing OR.

MCW 11, Enable Address Check: This position is independent of any selected unit. It allows testing of the address error checking stations in the storage units. Bit 11 disables address checking in the BCU for both CPU and channel operations; therefore, the BCU will not recognize address errors and, consequently, will not cancel storage if the BCU receives an address with bad parity from the CPU or a channel. If the address checking station in the storage works properly, the error will be detected there and forwarded to the BCU; thus, a means is provided for testing the address checking circuits in the storage units.

MCW 12, Stop Timer: This position is independent of any selected unit. Bit 12 blocks the updating of the interval timer.

MCW 14-15, Interchange Address: These bits are independent of any selected unit. The bits duplicate the function of the interchange storage address switch located at Z28 on the system control panel; position 14 duplicates the up position of the switch and position 15 duplicates the down position. (For a description of address switching, see 2075 Processing Unit, Volume 2, Field Engineering Manual of Instruction, Form 223-2873.) MCW positions 14 and 15 have priority over the console switch; that is, if bit 14 or 15 is a 1, the address interchange function called for by the bit will be effective regardless of the setting of the switch. There is no interlock, however, between the two MCW positions. Care must be taken to have only one of the positions on at a given time.

MCW 22-31, Count: These bits form a ten-bit count field, which is used with the diagnostic stop and logout feature of MCW control. The field is set to some initial count by the diagnose instruction, the count that specifies the number of machine cycles the CPU is to spend executing instructions (following the diagnose instruction) before the stop and logout occurs.

If the initial count is 0, the logout occurs immediately following the diagnose instruction execution. If the initial count is other than 0, the instructions that follow the diagnose instruction are executed for the number of machine cycles specified by the count, the count being reduced 1 on each machine cycle. When the count reaches 0, the machine stops and a logout occurs.

THEORY OF OPERATION, FLT

This section describes the details of the FLT circuits. A review of Figures 6200 and 6201 should precede a study of circuit details.

Word Switch Matrix (Figure 5300)

The word switch matrix has as its inputs 1216 indicator lines, 64 data keys, and 20 control lines (19 of which gate the indicators, and one which gates the data keys). Its output is 64 data lines numbered 0-63. The word switch matrix separates the indicator lines and the data keys into 20 groups of 64 lines each. The state of a particular group will be gated through and will be available at the output depending on which control line is up. The control lines are mutually exclusive; however, there will always be one line active.

If an indicator is on, the corresponding output line will be a logical 1 when gated through. A data key in the 1 position will be gated through as a logical 1. During FLT mode or during a logout, the control lines that gate the indicators through will be active; at all other times the data keys will be gated through the matrix.

Odd parity is generated on the eight output bytes from the word switch matrix. The 64 output data lines and the eight parity bits feed the channel SBI

and the SBO latches. Only the 64 data lines go to the input of the bit switch matrix.

Bit Switch Matrix (Figure 5301)

The bit switch matrix has as its inputs the 64 output data lines from the word switch matrix and 16 control lines. Its output is one data line. The bit switch matrix selects one of the 64 inputs and gates it through to the output. The one to be selected depends on the state of the control lines.

The 16 control lines are divided into eight byte lines, each corresponding to one of the eight input bytes, and eight bit lines, which pick the bit in the selected byte. During FLT, a pair of the 16 lines are always selected and this pair of lines determines which one of the input lines will appear at the output.

Word Control Counter (Figure 5302)

The word control counter is a five-position counter, consisting of a group of five indicated triggers and a group of five latches. The triggers feed through combinatorial logic into the latches; the latches in turn feed directly into the triggers. The control pulse step word control counter sets the contents of the triggers plus 1 into the latches. The control pulse set latches to triggers sets the value of the latches back into the triggers.

The five triggers are also fed from channel SBO positions 35-39. The SBO is gated into the WCC triggers with the control pulse set FLT control word.

The word control counter has the following functions:

1. It provides the five low-order positions (16-20) of the storage address during FLT and logout operations.
2. It provides, via a decoder, the select lines that gate the 64-bit groupings through the word switch matrix.
3. It provides, via the same decoder, the gating lines for the CPU that determine where the pattern words will be put during the scan in portion of FLT.

Bit Control Register (Figure 5301)

The bit control register consists of a group of six indicated triggers numbered 0-5. The six triggers are fed from the channel SBO positions 40-45. SBO is gated into the register by the control pulse set FLT control word. The bit control register generates the 16 gating lines to the bit switch matrix. The first three positions of the register feed a binary decoder that develops the eight byte select lines; the last three positions feed another binary decoder that develops the eight bit select lines.

Test Register

The test register consists of 18 indicated triggers. The 18 triggers are fed by channel SBO positions 46-63. The SBO is gated into the test register by the control pulse set FLT control word. The test register has two functions:

1. During termination, the register contains the index number that refers to the diagnostic index. The number is displayed on the system control panel at M67-86.
2. During the running of an FLT, if no termination bits are present, the register contains the alternate test number. If the test passes, the next test to be executed is the one whose test number matches the contents of the test register.

Repeat Counter (Figure 5303)

The repeat counter counts the number of times a test is executed, so that after eight executions, conditions will be established to select the next test or to terminate the operation.

The clock steps twice during each compare cycle, once when the compare cycle A5D1 pulse becomes active, and again when the pulse becomes inactive.

The first four executions step the clock through one complete cycle (1-8) and the next four executions step the clock through another complete cycle. Trigger 2, which is on only during the second clock cycle, and clock output 6 generate the RC 7 pulse. RC 7 conditions the terminate and the select next-test circuits.

The RC 0 output conditions the FLT repeat circuits and conditions the turn-off of the seek trigger.

Scan Clock (Figure 5304)

The output of the scan clock consists of eight 100-nanosecond pulses labeled A0 through A7. The clock is driven by two complementary inputs, which are created by the network of singleshots shown in the figure. Note in the timing sequence that singleshots A and B fire in tandem, each firing twice for a complete clock cycle. Singleshot C fires with A, and singleshot D fires with B, to generate the complement inputs to the clock. Details of the clock are shown on Systems PR 990.

The scan clock is started by turning on the run scan clock trigger. Scan clock pulse A5 turns the run scan clock trigger off. With the trigger off, singleshot B can no longer turn on singleshot A, so the clock completes its cycle (through A7) and stops.

Scan In

After a test is loaded from tape into area A or B, the TIC pulse from the channel makes the first storage request. When the storage word is available, the storage advance pulse starts the scan clock (Figure 5304). Scan clock pulse A2 turns on the scan in cycle trigger (Figure 5302), which stays on until the 15 pattern words are scanned in. Figure 5305 shows the scan in circuit.

As the fifteenth pattern word is scanning in, the WCC steps to 16. During the fifteenth scan in, scan clock pulse A4 requests storage again (to fetch the FLT control word), and scan clock pulse A6, with WCC 16, turns on the advance cycle trigger (Figure 5304).

Advance

The advance cycle trigger: (1) allows the FLT control word registers to receive the FLT control word from storage, and (2) enables the operation of triggers D1, D2, and D3 (Figure 5304). Triggers D1, D2, and D3 start and stop the CPU control clock so that the number of CPU cycles specified by the advance counter will occur during the advance portion of the FLT operation.

When the FLT control word returns from storage, storage advance sets the FLT control word into the FLT control word registers and starts the scan clock. During FLT advance, the only function of the scan clock is to start the D1, D2, and D3 sequence; a scan clock A3D2 pulse gates an AR clock pulse to turn on trigger D1. One cycle later, trigger D1 turns off and trigger D2 turns on. D2 stays on for two cycles and turns on D3. D3 stays on until the beginning of the compare portion of the FLT operation.

Trigger D1 turns on the start clock sync trigger in the CPU clock controls and at the same time examines the advance counter for an initial count of 0. If the advance counter is 0, which means that no CPU cycles are to take place, D1 also turns on the MCW pseudo check trigger, which blocks the control trigger; in this case, no controlled pulses are emitted by the CPU clock.

If the advance counter contains an initial count of 1, D1 turns on the start clock sync trigger the same as before and also turns on D2. D2 recognizes the 1-count in the advance counter and turns on the MCW pseudo check trigger, but before the control clock can be blocked, one set of controlled pulses will have been emitted by the CPU clock.

If the advance counter has an initial count of more than 1, D3 monitors the advance counter until the advance counter is reduced to 1; then D3 turns on the MCW pseudo check trigger to stop the CPU control clock. The control clock will have emitted a number of sets of controlled pulses equal to the initial count in the advance counter.

When the MCW pseudo check trigger stops the control clock, it also fires a 2-microsecond single-shot to turn on the compare cycle trigger. The compare cycle trigger denotes the end of the advance portion of the test and conditions the compare circuits. The 2-microsecond delay allows time for the CPU trigger lines to settle down before the comparison is made between the selected CPU trigger and the expected result bit.

Compare

Figures 5300 and 5301 show how one CPU trigger, out of the possible 1216, is selected and compared to the expected result bit. As soon as the FLT control word is loaded (at the beginning of the advance portion of the test), the word control counter and the bit control register select one CPU trigger, and the bit compare trigger is either left off or turned on to indicate what the status of the selected trigger should be after the advance is completed.

When the compare trigger turns on at the end of the advance, the scan clock is started and a scan clock A0 pulse samples the output of the compare circuit. If the status of the selected trigger and the expected result are equal, the pass trigger turns on; if they are unequal, the fail trigger turns on.

Each test is executed eight times, and if a test both passes and fails during the eight tries, the intermittent trigger turns on. The intermittent trigger only turns on an indicator.

THEORY OF OPERATION, MCW

MCW control of the CPU or the channels starts with the execution of the diagnose instruction. The instruction fetches a double word from storage and sets the left half of it into the MCW register. Figure 6210 shows the execution of the diagnose instruction, and Figure 5320 shows the CPU and channel functions that are controlled by the MCW positions.

The diagnose instruction is executed by the IE and D sequencers. During I time, the storage address

is calculated and set into the SAR. IE1 makes the fetch request and maintains the request until the BCU responds with accept. The IE1 cycles also send a diagnose signal to the BCU that sets the diagnose position of one of the return address registers.

After BCU generates accept, no sequencers are on until the advance pulse from the selected storage samples the return address register and generates diagnose select. The diagnose select signal (delayed approximately 150 nanoseconds) gates SBO 0-31 into the MCW register and at the same time turns on sequencer D1.

The timings of D1, D2, and D3 are shown in Figure 6210. If MCW 6 is off, the three sequencers turn on in order, and D3 generates a proceed signal. This signal turns on IE3, allowing the execution of the instruction to continue.

If MCW 6 is on and MCW 7 is off, D1, D2, and D3 turn on the same as before, but now they monitor the MCW counter (as shown in Figure 5320). If the MCW count field is 0, D1 turns on the MCW pseudo check trigger, which stops the clock and causes a logout before the proceed signal is generated. If the initial count is other than 0, D2 or D3 turn on the MCW pseudo check trigger at a time that allows the CPU to run, following proceed, a number of cycles that corresponds to the initial count in the MCW count field.

INTERRUPTS

INTRODUCTION

- An interrupt temporarily halts processing while the current PSW is stored and a new PSW is fetched.
- Processing resumes under control of the new PSW.
- The stored PSW shows what caused the interrupt.

Processing a job on the 2075 consists primarily of executing in sequence the instructions of the problem program. It is necessary at times, however, to halt temporarily the execution of the problem instructions and take care of one of a number of unusual or special conditions that can arise, such as a program error, an interval timer overflow, an I/O unit completing its current assignment, a machine error, and so on. The interrupt circuits monitor the system for all of the unusual or special conditions; when one of them occurs, the interrupt circuits cause the CPU to suspend execution of the program in process at the time and to transfer to another program that can analyze the interrupting condition and take some course of action. (Specific examples are given later.)

The interrupt process accomplishes the program transfer by replacing the current PSW register contents with a new PSW. The new PSW establishes the new status of the CPU and establishes the starting point of the new routine to be executed, which is normally in the operating system program. Before the current PSW is replaced, however, the interrupt code field of the PSW register is set according to the type of interrupt condition that was detected. Then the entire current PSW is stored away and the new PSW is fetched from storage and set into the PSW register. Next, an IC recovery occurs, fetching a new stream of instructions from the transferred-to program.

The interrupt hardware has now completed its job. From this point, the operating system program examines the stored-away PSW to determine what caused the interrupt and takes a course of action consistent with the cause. This action might be to process a real-time job because of an external signal from another computer; or to call in an analyzing program because of a machine check; or to terminate the problem because of a program check, and so on. After the necessary action by the operating system program, the interrupted problem may be resumed by the execution of the load PSW instruction, which loads the PSW register with the PSW that was stored away at the time of the interrupt.

Interrupt Classes

- There are five classes of interrupts:
  1. External
  2. Program
  3. Machine check
  4. Supervisor call
  5. Input/output

All of the interrupts (with two exceptions) fall into one of these classes. Each class is distinguished by the fixed-storage locations in which the current PSW is stored and from which the new PSW is fetched. The two exceptions are initial program load (IPL) and timer advance request. These two procedures use the interrupt circuits to accomplish their objectives, but do not result in the exchange of PSWs.

External

This class includes (1) timer word overflow, which occurs when the timer word goes from a positive to a negative value, (2) interrupt key on the system control panel, and (3) any of six external signals from an "outside" computer. When any of these three types of interrupt conditions occurs, the CPU takes an external interrupt, which results in the storing away of the current PSW in the external old storage location (24) and the fetching of a new PSW from the external new location (88).

Program

This class consists of 19 interrupt conditions, all of them being associated with either the interpretation or the execution of instructions. Any program interrupt results in the exchange of program old and program new PSWs. The fixed-storage locations for the program PSWs are: old, 40; new, 104.

Machine Check

This interrupt constitutes a class. It is always the result of a machine malfunction and results in the exchange of machine check old and new PSWs (old, 48; new, 112).

Supervisor Call

This class includes only the supervisor call interrupt, which occurs when the supervisor call instruction is decoded. The current PSW is stored in location 32 (old), and a PSW is fetched from location 96 (new).

is calculated and set into the SAR. IE1 makes the fetch request and maintains the request until the BCU responds with accept. The IE1 cycles also send a diagnose signal to the BCU that sets the diagnose position of one of the return address registers.

After BCU generates accept, no sequencers are on until the advance pulse from the selected storage samples the return address register and generates diagnose select. The diagnose select signal (delayed approximately 150 nanoseconds) gates SBO 0-31 into the MCW register and at the same time turns on sequencer D1.

The timings of D1, D2, and D3 are shown in Figure 6210. If MCW 6 is off, the three sequencers turn on in order, and D3 generates a proceed signal. This signal turns on IE3, allowing the execution of the instruction to continue.

If MCW 6 is on and MCW 7 is off, D1, D2, and D3 turn on the same as before, but now they monitor the MCW counter (as shown in Figure 5320). If the MCW count field is 0, D1 turns on the MCW pseudo check trigger, which stops the clock and causes a logout before the proceed signal is generated. If the initial count is other than 0, D2 or D3 turn on the MCW pseudo check trigger at a time that allows the CPU to run, following proceed, a number of cycles that corresponds to the initial count in the MCW count field.

INTERRUPTS

INTRODUCTION

- An interrupt temporarily halts processing while the current PSW is stored and a new PSW is fetched.
- Processing resumes under control of the new PSW.
- The stored PSW shows what caused the interrupt.

Processing a job on the 2075 consists primarily of executing in sequence the instructions of the problem program. It is necessary at times, however, to halt temporarily the execution of the problem instructions and take care of one of a number of unusual or special conditions that can arise, such as a program error, an interval timer overflow, an I/O unit completing its current assignment, a machine error, and so on. The interrupt circuits monitor the system for all of the unusual or special conditions; when one of them occurs, the interrupt circuits cause the CPU to suspend execution of the program in process at the time and to transfer to another program that can analyze the interrupting condition and take some course of action. (Specific examples are given later.)

The interrupt process accomplishes the program transfer by replacing the current PSW register contents with a new PSW. The new PSW establishes the new status of the CPU and establishes the starting point of the new routine to be executed, which is normally in the operating system program. Before the current PSW is replaced, however, the interrupt code field of the PSW register is set according to the type of interrupt condition that was detected. Then the entire current PSW is stored away and the new PSW is fetched from storage and set into the PSW register. Next, an IC recovery occurs, fetching a new stream of instructions from the transferred-to program.

The interrupt hardware has now completed its job. From this point, the operating system program examines the stored-away PSW to determine what caused the interrupt and takes a course of action consistent with the cause. This action might be to process a real-time job because of an external signal from another computer; or to call in an analyzing program because of a machine check; or to terminate the problem because of a program check, and so on. After the necessary action by the operating system program, the interrupted problem may be resumed by the execution of the load PSW instruction, which loads the PSW register with the PSW that was stored away at the time of the interrupt.

Interrupt Classes

- There are five classes of interrupts:
  1. External
  2. Program
  3. Machine check
  4. Supervisor call
  5. Input/output

All of the interrupts (with two exceptions) fall into one of these classes. Each class is distinguished by the fixed-storage locations in which the current PSW is stored and from which the new PSW is fetched. The two exceptions are initial program load (IPL) and timer advance request. These two procedures use the interrupt circuits to accomplish their objectives, but do not result in the exchange of PSWs.

External

This class includes (1) timer word overflow, which occurs when the timer word goes from a positive to a negative value, (2) interrupt key on the system control panel, and (3) any of six external signals from an "outside" computer. When any of these three types of interrupt conditions occurs, the CPU takes an external interrupt, which results in the storing away of the current PSW in the external old storage location (24) and the fetching of a new PSW from the external new location (88).

Program

This class consists of 19 interrupt conditions, all of them being associated with either the interpretation or the execution of instructions. Any program interrupt results in the exchange of program old and program new PSWs. The fixed-storage locations for the program PSWs are: old, 40; new, 104.

Machine Check

This interrupt constitutes a class. It is always the result of a machine malfunction and results in the exchange of machine check old and new PSWs (old, 48; new, 112).

Supervisor Call

This class includes only the supervisor call interrupt, which occurs when the supervisor call instruction is decoded. The current PSW is stored in location 32 (old), and a PSW is fetched from location 96 (new).

Input/Output

The I/O class includes the interrupts caused by signals from any of the six channels. An I/O interrupt causes the current PSW to be stored in location 56, the channel status word to be stored in location 64, and the new PSW to be fetched from location 120.

Figure 76 shows the permanently allocated storage locations. Note that in addition to the slots reserved for old and new PSWs, the first three double-word locations are reserved for IPL use; location 64 is reserved for the channel status word, location 72 for the channel address word (single word), and location 80 for the timer word (single word). The logout area starts in location 128 and has space for the storage of 19 double words.

Interruptable Status

- Certain PSW positions enable or mask off interrupt requests: a 1 bit enables the request; a 0 bit blocks the request.

Certain positions in the PSW, called masks, determine the interruptable status of the CPU (Figure 77). If a mask position is a 1, the corresponding interrupt source is allowed to interrupt the CPU; if a 0, the interrupt is said to be masked off and the interrupt does not occur. Some interrupt sources, although masked off, remain pending and will be taken should the mask position in question be changed by the introduction of a new PSW. Each new PSW that is introduced may, therefore, change the interruptable status of the CPU. Two of the mask fields, system mask and program mask, may be changed independently of the rest of the PSW by the instructions set system mask and set program mask. The mask fields and their effect on interrupts are:

System Mask, PSW 0-6

Each position either allows or masks off the I/O interrupt signals from its respective channel. If masked off, the I/O interrupt request remains pending.

PSW 7

This single position is also a part of the system mask field and either allows or masks off as a group all of the external interrupt class. If masked off, the external interrupts remain pending.

PSW 13

This position either allows or masks off machine check interrupts. If masked off, the machine check condition is ignored.

Program Mask, PSW 36-39

These four positions either allow or mask off, respectively: fixed-point overflow, decimal overflow, exponent underflow, and significance. If masked off, the interrupt conditions are ignored. Note that of the 19 conditions for program interrupts, only four of them are controllable by the PSW.

Interrupt Examples

An interrupt causes the storing of the current PSW and the fetching of a new PSW, using fixed-storage locations that correspond to the class of the interrupt. It is the programmer's responsibility to determine what the new PSW shall do for the situation that caused the interrupt. The following examples show how certain interrupt conditions might be used to accomplish various objectives. The associated diagrams are simplified and are not intended to show programming techniques.

External

If the CPU is in the wait state, and it is desired to switch to the running state, a new PSW must be introduced. Because no instructions are executed in the wait state, an interrupt must occur to change the PSW. The interrupt key could be used for this purpose (Figure 78), provided PSW position 7 is a 1. Depressing the key causes an external interrupt with the resultant exchange of PSWs. The new PSW would specify the running state by having position 14 to a 0; the IC portion would specify the address of the first instruction with which to start processing.

Program

Figure 79 shows how the program interrupt might be used to signal to the machine a program-caused situation such as a fixed-point overflow. In this particular case, the interrupt request would be honored only if PSW position 36 is a 1. After the interrupt, the new PSW could specify the supervisor state (bit 15 a 0) and the starting point of a subroutine. The subroutine would first examine the interrupt code field of location 40 (where the old PSW was stored) to determine the cause of the interrupt. The subroutine could then examine the instruction address (IC) and the instruction length code (ILC) of location 40. The IC points to the instruction that would have been executed next if the CPU had not been interrupted, and the ILC indicates the length of the last instruction executed, the one that caused the overflow. With this knowledge, the subroutine can locate the overflow-causing instruction and take whatever action is necessary.

At the end of the subroutine, the load PSW instruction might be executed, specifying location 40 as its operand. This action would place the original PSW in control again; the problem would continue from where it was interrupted.

Machine Check

Upon the occurrence of a machine error, with the CPU check switch on the system control panel in the process position, the sequence of events is as shown in Figure 80. The controlled clock is stopped immediately on detection of the error, blocking further change of any triggers in the CPU. The CPU is then logged out; that is, the status of 1216 CPU triggers is stored in 19 consecutive double words, starting at byte address 128. Next, the CPU is reset, the clock is restarted, and a machine check interrupt occurs, exchanging old and new machine check PSWs. The new PSW specifies the supervisor state and transfers to the operating system program. The operating system program then brings in a recording and retry program that analyzes the logged-out data to make a decision whether to retry the instruction on which the error occurred. Some instructions can be retried directly because the operands involved are still valid; other instructions, however, destroy their operands as they are executed, making a direct retry impossible.

If the retry is attempted and is successful, the recording and retry program notifies the operating system program which reloads the PSW register from location 48 (location 48 contains the interrupted PSW). The problem program continues from where it was interrupted.

If the retry is attempted but is unsuccessful, the retry program notifies the operating system program; the problem is aborted, the operator is notified by a printout, and the next stacked job is run.

If the retry is not attempted, control is returned to the operating system program which may restart the problem from some prior checkpoint. If the failure persists, the problem is aborted, the operator is notified by a printout, and the next stacked job is run.

Supervisor Call

The supervisor call interrupt may be used for changing the status of the CPU, for example, switching from problem to supervisor state. If a problem program needs to start an I/O operation, it would first switch to the supervisor state before an I/O instruction could be executed (I/O instructions are invalid in the problem state). Figure 81 shows the supervisor call interrupt being used for such a purpose.

The supervisor call instruction causes a supervisor call interrupt, resulting in the exchange of PSWs. The new PSW specifies the supervisor state and locates the start I/O instruction. The start I/O instruction is decoded, addressing a channel and an I/O device. At this point, the CPU hangs up until it receives a release from the channel. The release does not come, however, until the channel obtains the CAW from storage, then the first CCW from storage, then a signal from the I/O device indicating that the device can perform what the CCW specifies.

Once the channel releases the CPU, the CPU continues with the instruction that follows start I/O. This next instruction could reload the PSW register with the PSW from location 32 (the interrupted PSW) thus continuing the problem program from where it was interrupted.

Input/Output

Channels send I/O interrupt signals to the CPU upon the occurrence of various error conditions in the channel or the I/O device, and upon the normal end of an I/O operation. Figure 82 shows an I/O interrupt caused by the channel completing its operation with the I/O device.

An I/O interruption can occur only after the execution of the current instruction is completed and while the CPU is interruptable for the channel that presents the request. The I/O interrupt signal from the channel starts the interrupt sequence, during which time the channel stores a channel status word (CSW) in location 64. The interrupt sequence exchanges I/O old and new PSWs. The new PSW locates a program routine that examines the CSW just stored to determine whether the I/O operation was completed satisfactorily. The load PSW instruction could then be executed to transfer control back to the program that was interrupted.

Initial Program Load

Figure 83 shows the sequence of events of the IPL procedure. The load pushbutton on the system control panel causes the channel and device addressed by the rotary switches to read three words (the IPL PSW and two CCWs) and place them in the first three double word locations of storage. The channel then obtains the second word it just stored (a CCW) and uses it to read from the device and put the rest of the program into storage. After the read operation is completed, the channel sends a release signal to the CPU which starts the interrupt sequence.

Because IPL controls are on, the store portion of the interrupt sequence is blocked, but the fetch portion is allowed, fetching the PSW from location 0 and

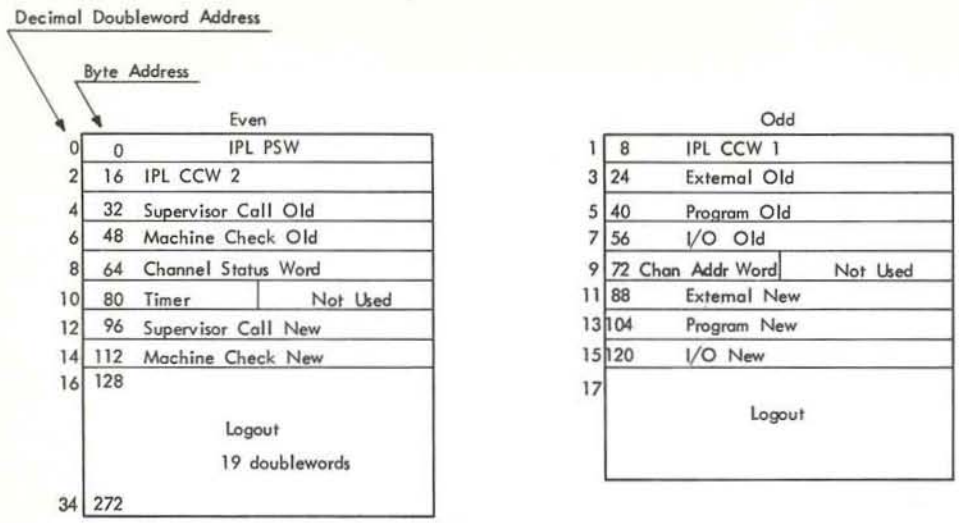
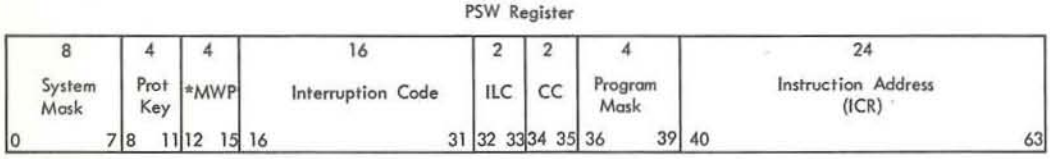


FIGURE 76. FIXED STORAGE LOCATIONS



Bit
- 0 Multiplexor Channel
- 1 Selector Channel 1
- 2 Selector Channel 2
- 3 Selector Channel 3
- 4 Selector Channel 4
- 5 Selector Channel 5
- 6 Selector Channel 6
- 7 External

8-11 Protection Key

- 12 VFL Code; 1=ASCII, 0=EBCDIC
- 13 Machine Check Mask (M)
- 14 Wait State (W); 1= wait, 0= running
- 15 Problem State (P); 1= problem, 0=supervisory

16-31 Interruption Code

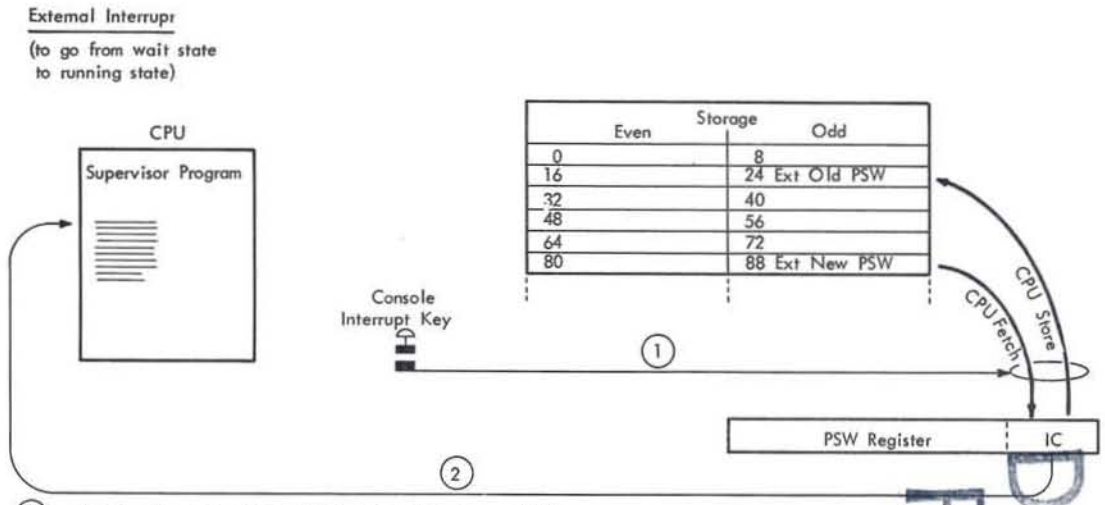
32-33 Instruction Length Code (in halfwords); 01 = 1 halfword, 10 = 2 halfwords, 11 = 3 halfwords

34-35 Condition Code

- 36 Fixed-Point Overflow
- 37 Decimal Overflow
- 38 Exponent Underflow
- 39 Significance

40-63 Instruction Address

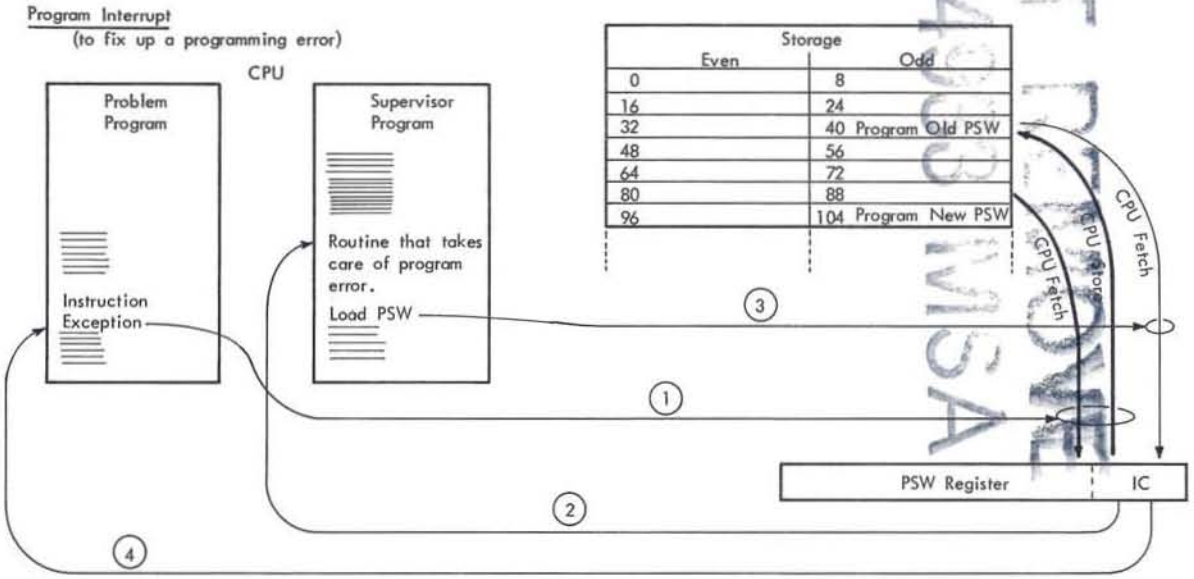
FIGURE 77. PROGRAM STATUS WORD



(1) Pressing the interrupt pushbutton stores the current PSW in the external old PSW location and fetches a new PSW.

(2) New external PSW can specify running state and locate the instruction with which to start processing.

FIGURE 78. EXTERNAL INTERRUPT



(1) Some exceptional condition arises concerning an instruction in the problem program (for example, an overflow); exceptions can occur in either the I unit or the E unit. CPU stops instruction execution, stores problem PSW in program old PSW location, and fetches new PSW.

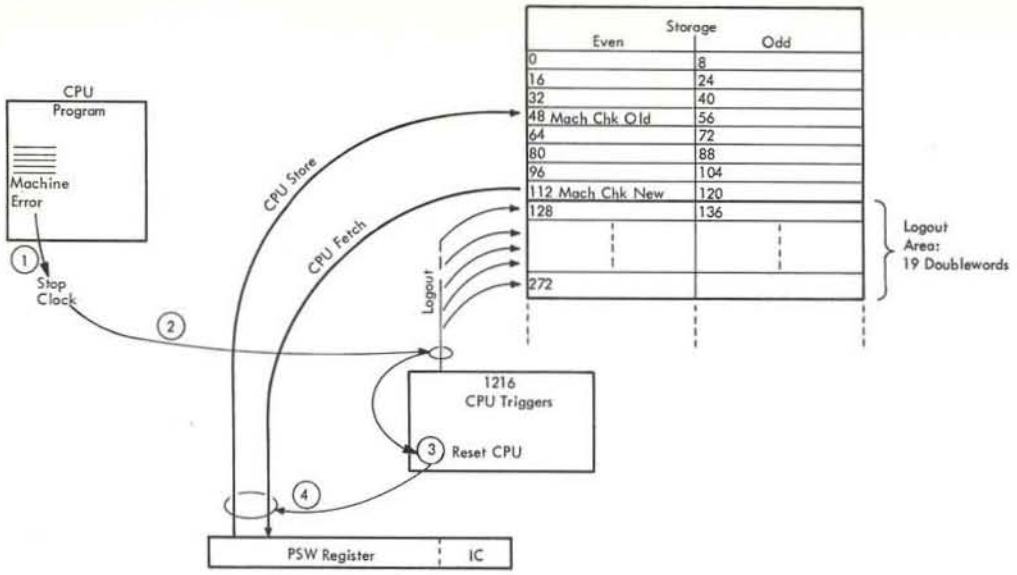
(2) New program PSW locates routine that handles the instruction exception that just occurred.

(3) After exception-fixup routine is finished, load PSW instruction reloads problem PSW back into PSW register.

(4) Problem PSW causes problem program to continue on from where it was interrupted.

FIGURE 79. PROGRAM INTERRUPT

Machine Check Interrupt



(1) Machine makes error, which stops clock.

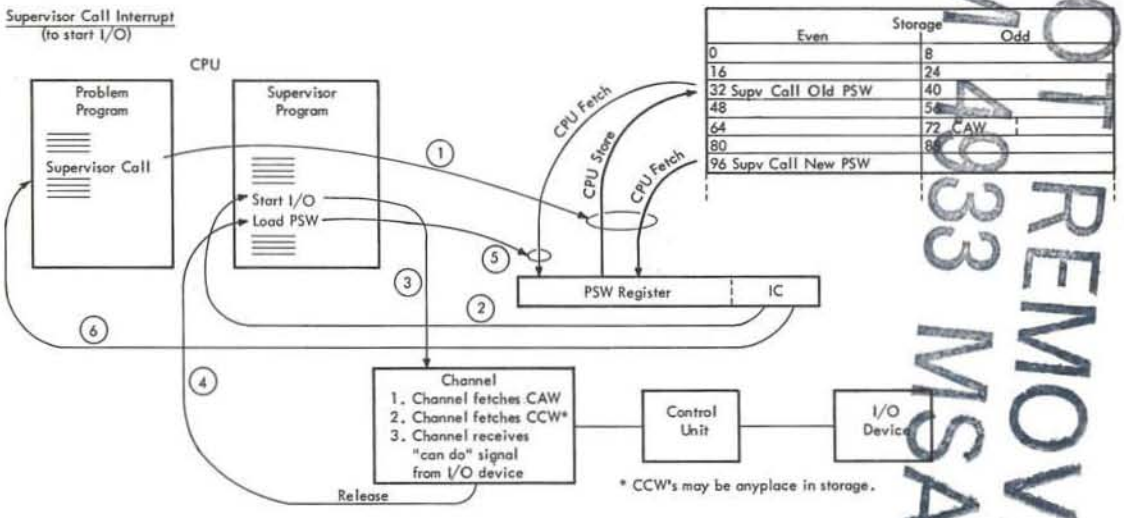
(2) Logout circuits store 19 doublewords that represent the status of 1216 CPU triggers.

(3) CPU reset.

(4) Machine check interrupt sequence stores current PSW in machine check old PSW location and fetches machine check new PSW. New PSW locates program that takes corrective action.

FIGURE 80. MACHINE CHECK INTERRUPT

Supervisor Call Interrupt (to start I/O)



(1) Supervisor call instruction causes supervisor call interrupt which stores problem PSW in supervisor call old location and fetches a new PSW.

(2) New supervisor call PSW locates start I/O instruction.

(3) Start I/O instruction starts up channel and causes CPU to hold up further instruction executions.

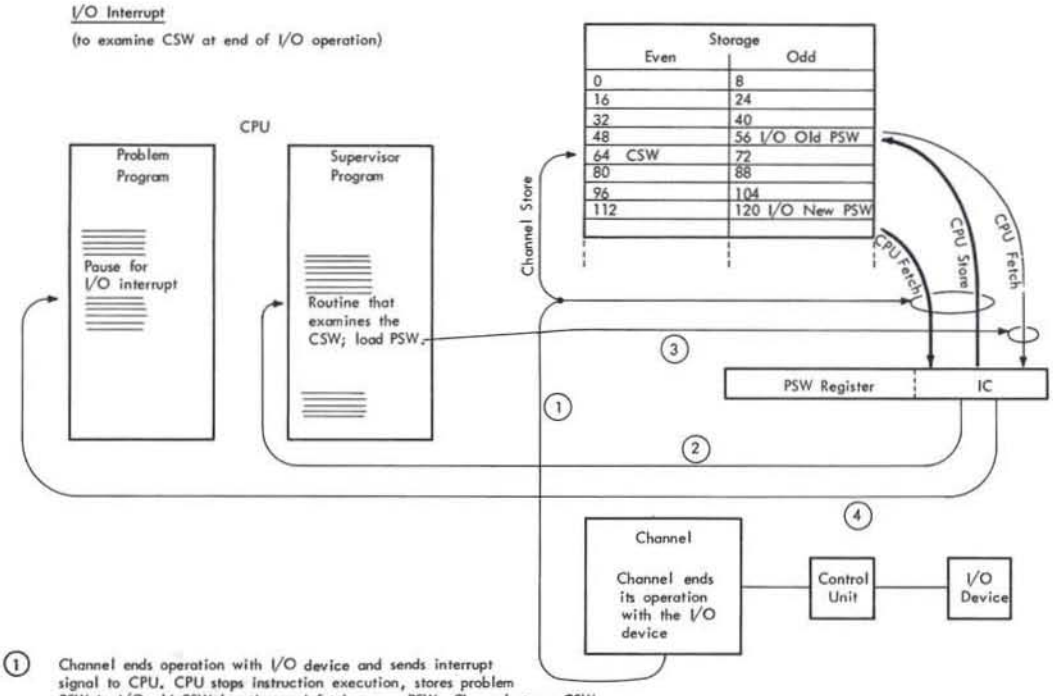
(4) Channel sends release to CPU. CPU executes next instruction (load PSW).

(5) Load PSW instruction reloads problem PSW back into PSW register.

(6) Problem PSW causes problem program to continue with next instruction.

NOTE: Once channel sends release to CPU, channel performs operation with I/O device independent of CPU.

FIGURE 81. SUPERVISOR CALL INTERRUPT



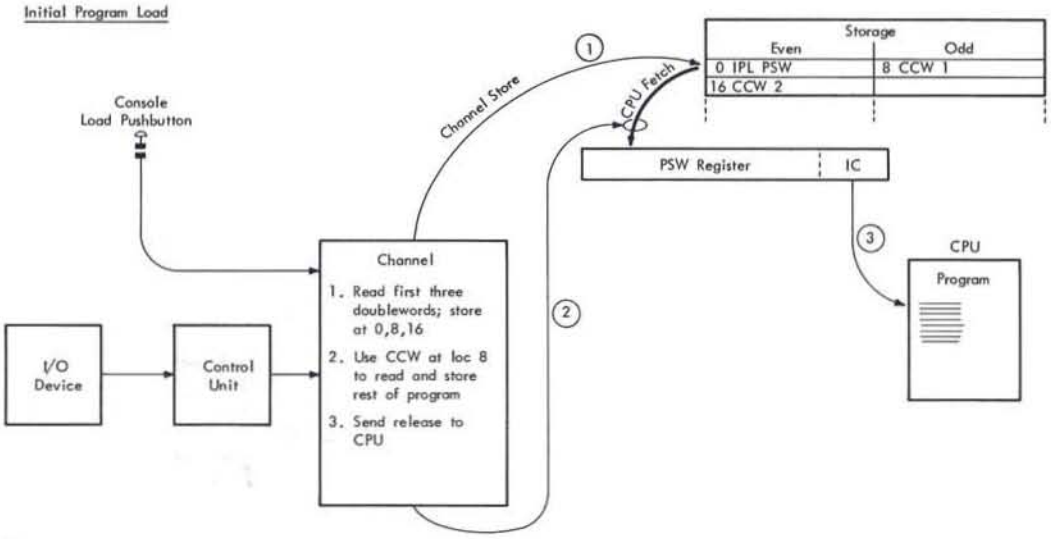
(1) Channel ends operation with I/O device and sends interrupt signal to CPU. CPU stops instruction execution, stores problem PSW in I/O old PSW location and fetches new PSW. Channel stores CSW while PSW swapping is going on.

(2) New I/O PSW locates routine that examines the CSW just stored by channel.

(3) After CSW-examining routine is finished, load PSW instruction reloads problem PSW back into PSW register.

(4) Problem PSW causes problem program to continue on from where it was interrupted.

FIGURE 82. I/O INTERRUPT



(1) Load pushbutton causes channel to read first three doublewords from I/O device and to place them in storage locations 0,8, and 16. The channel then uses the CCW at location 8 to store the rest of the program.

(2) At the end of the read operation, channel sends release to CPU which causes an IPL load PSW interrupt. This interrupt sequence loads the PSW register from location 0.

(3) The IPL PSW, now in the PSW register, locates the start of the program to be executed.

FIGURE 83. IPL INTERRUPT

setting it in the PSW register. An IC recovery then occurs, fetching a stream of instructions as dictated by the IC of the PSW. The CPU starts executing these instructions.

Interrupt Priorities

- Priorities in general are: (1) those caused by instruction execution, (2) those caused by external or I/O signals, and (3) those caused by I unit processing.

- Machine check and IPL share top priority.

Interrupt requests can come from various sources at the same time, such as from the I unit while an instruction is being fetched or decoded and from the E unit while an instruction is being executed. All interrupts are assigned a priority; and when two or more interrupt requests occur at the same time, the priority circuits choose the interrupt request with the highest priority and execute that interrupt.

The general priority scheme is that interrupts should be taken in the same order as the instructions with which they are associated. Interrupt conditions resulting from instruction execution, therefore, are given priority over interrupt conditions resulting from I unit processing. In addition to the instruction execution interrupts and the I time processing interrupts, there are interrupts that may occur asynchronously with CPU operation, such as the external or I/O interrupt. These interrupts have a lower priority than the execution interrupts but higher than the I time interrupts. The general order of interrupt priority becomes: those caused by instruction execution, those caused by external or I/O, and those caused by I unit processing.

These three classes of interrupts are E from E, E from I, and I from I. The E from E class of interrupts are caused by instruction execution that occurs in the E unit. The E from I class of interrupts are caused by instruction execution that occurs outside the E unit. The I from I class of interrupts are caused by instruction preparation that occurs in the I unit (during T1 or T2).

An exception to the general order of interrupts stated above is that two interrupts of the E from I class, SAP and invalid store address, take priority over the E from E class.

Two interrupts are special, not belonging to any of the three classes; they are machine check and IPL.

Figure 9250 lists all of the interrupts in the order of descending priority. A description of each interrupt follows.

Machine Check

On a machine error, a logout takes place, then a CPU reset, then a machine check interrupt request. The machine check interrupt request forces the start of the interrupt sequence directly, without first having to be detected during an E last cycle or a T1 or T2, as do other interrupts. The machine check interrupt has priority over all other interrupts.

Initial Program Load

The IPL procedure consists of loading storage from an I/O device and then loading the PSW register from storage location 0. The PSW loading portion of the IPL is performed by a modified interrupt sequence (PSW fetch, but no store).

The channel signals the CPU when the I/O to storage operation is completed by sending a release signal. This signal forces the start of an interrupt sequence, during which the PSW is loaded from storage location 0. Because neither the E unit nor the I unit is running at the time of the release signal, no other interrupt can be detected. IPL, therefore, has top priority along with machine check.

Storage Address Protect

Of all the program-caused interrupts, the storage address protect (SAP) has highest priority. A SAP check occurs when a store is attempted in a location whose key in the storage protect memory does not match the protection key in the PSW.

The storage cycle in which the store is attempted is in progress when the mismatch of keys is detected. The storage cycle is completed without actually storing anything and a mismatch signal is sent to the bus control unit (BCU), turning on the SAP check trigger. This trigger in turn requests a storage address protect interrupt.

The request is checked only during an E last cycle, but because the instruction execution calling for the store might be completed before the mismatch of the keys is detected, the interrupt request might come too late to be sampled during the E last cycle of the execution that caused it. The interrupt request, in this case, is detected at the next E last cycle, which is on the instruction following the one that called for the bad store.

It is possible, however, that on the execution of the bad-store instruction or on the concurrent I time processing of the next instruction another interrupt condition is present; this other interrupt condition would be detected, causing entrance into an interrupt sequence. The SAP check signal would arrive at the interrupt controls as this sequence for the other

interrupt is started, and would force the sequence already started to become a SAP check interrupt sequence.

Note that a SAP check interrupt can be taken at two intervals, immediately after the bad-store instruction or after the next instruction, depending on whether another interrupt occurred on the bad-store instruction. An analyzing program would have difficulty establishing which instruction caused the SAP check; therefore, a retry is not attempted if a SAP check occurs.

Invalid Store Address

This interrupt condition occurs when the I unit specifies a store address that is outside the available storage.

The bad-address condition is detected in the BCU, which sends an interrupt request to CPU along with the accept pulse for the storage cycle that was to perform the store operation. The storage cycle is completed without anything being stored. Because the interrupt condition is detected early in the storage cycle, the request for this interrupt is present during the E last cycle of the instruction execution associated with it. The request is sampled during this E last cycle, and the interrupt sequence is started.

E Program

The interrupts in this section are all in the E from E class; that is, they originate in the E unit during an instruction execution and they are detected during E time (specifically, E last cycle).

The occurrence of any of the following 12 conditions turns on a trigger in the E unit called the E interrupt trigger, which in turn requests an E program interrupt (exchange of program PSWs). During the interrupt sequence, the interrupt code field of the old PSW is set according to the particular condition that caused the interrupt. If more than one of the 12 conditions occur on the same execution, the condition that occurs first turns on the E interrupt trigger, which, in addition to requesting an E program interrupt, blocks the recognition of any of the other E program interrupt conditions. The 12 E program interrupts, therefore, are mutually exclusive; there are no priority considerations among them:

The E program conditions are:

Invalid Data: This interrupt may be caused by any of the following:

1. The sign or digit codes of operands are incorrect in decimal arithmetic, convert to binary, or editing operations.
2. In decimal arithmetic, the fields overlap incorrectly.
3. In decimal arithmetic, the multiplicand has too many high-order significant digits.

Fixed-Point Overflow: A high-order carry occurs or high-order significant bits are lost in fixed-point addition, subtraction, shifting, or sign control operations.

Fixed-Point Divide: The quotient exceeds its register size (31 bits plus sign), or the result of convert to binary exceeds 31 bits.

Invalid Address: An operand address is outside available storage.

E SAP: A SAP violation occurs on an operand fetch of a read-protected storage location.

Specification: In multiply or divide decimal, one of the following occurs:

1. The multiplier or the divisor exceeds 15 digits and sign ($L2 > 7$).
2. The multiplier is not shorter than the multiplicand ($L2 \geqq L1$); the divisor is not shorter than the dividend ($L2 \geqq L1$).

Decimal Overflow: In a decimal operation, the destination field is too small to contain the result.

Decimal Divide: The quotient exceeds the specified data field size.

Exponent Overflow: In floating-point arithmetic, the result characteristic exceeds 127.

Exponent Underflow: In floating-point arithmetic, the result characteristic goes less than 0.

Significance: In floating-point addition or subtraction, the result has an all zero fraction.

Floating-Point Divide: An attempt is made to divide by 0.

External

Any of the following three interrupts cause the exchange of the external PSWs; each interrupt, however, has its own identifying digit set into the interrupt code field of the old PSW. If two or more requests occur simultaneously, one interrupt is taken, but with an interrupt code bit set for each request.

Console: Initiated by the interrupt key on the system control panel.

Timer: The timer word is stepped from a positive to a negative value.

External Signal: The signal on any or all of six external lines from another computer.

Timer Advance Request

Timer advance requests are initiated by a 50-cycle or 60-cycle signal from the power supply. On any of the requests, if no higher-priority request is outstanding, a timer advance interrupt sequence occurs which fetches the timer word from location 80, setting it in the J register. The E unit then updates the word by subtracting 5 from it, and stores the word back into location 80.

Subtracting 5 at a 60 cps rate is equivalent to subtracting 1 at a 300 cps rate, or 1 every 3.33 milliseconds. The same reduction rate is achieved on 50-cycle machines by subtracting 6 instead of 5, at the 50 cps rate.

If reduction causes the count to go negative, a timer overflow condition is detected and a timer overflow interrupt request is made. The timer overflow interrupt will occur immediately after the timer advance interrupt, provided the external signal mask bit is on in the PSW.

Input/Output

An I/O interrupt is caused by an I/O request signal from one of the six channels. If requests from several channels occur simultaneously, a separate priority circuit for channel requests selects one of the requests for servicing; the other requests remain pending.

If I/O interrupts have priority, a response is sent to the selected channel, and the interrupt sequence is started. The sequence is stopped one cycle after it starts, however, to wait on the channel that received the response to store its channel status word. After the CSW is stored, the channel sends a release signal to the CPU which allows the interrupt sequence to continue, exchanging I/O PSWs.

If requests are pending from other channels, the requests are examined during the last cycle of the interrupt sequence; one is selected, and if I/O interrupts still have priority, the I/O interrupt process is repeated.

Program Store Compare Recovery

During the I time processing of a store type instruction, the I unit calculates the effective address of the store and places the address in SAR and the H register. As the I unit transfers the instruction to the E unit, a storage request is made (for the store) and the instruction counter register (ICR) is updated to the address of the next instruction. This is normal I unit processing.

While the store trigger is on (from store request to store accept), a comparison is made between the H register (the address to be stored into) and the ICR (the address of the next instruction); if the comparison is equal, the E unit is about to make a store into the location of the next storage words to be processed, words that probably have been prefetched into the A or B register. This situation is remedied by taking an IC recovery, fetching again the instructions that start at the updated setting of the ICR; the fetch is not made, however, until the store is completed.

The program store compare (PSC) signal (from the equal condition of the H register and the ICR) turns on the IC recovery required trigger, which in turn makes an interrupt request. At the end of the store instruction execution (E last cycle), this request is considered along with any other interrupt requests that are present; if the PSC gets priority, the IC recovery trigger turns on which starts the prefetch sequence in the I unit.

The PSC recovery takes priority over the interrupts listed below it because the lower-priority interrupts are concerned with the instruction that follows the store instruction; this next instruction is the one that might be changed by the store.

The PSC recovery interrupt does not use the regular interrupt sequencers because no PSWs are exchanged. Once priority for PSC recovery is established, the IC recovery trigger turns on which causes the necessary fetches to A and B.

I Program

The interrupts in this category all result from conditions that occur during the I time processing of an instruction, which is the instruction following the current one being executed. Because the I unit and the E unit overlap, priority is granted to I program interrupts only if no interrupts associated with the current instruction are outstanding.

All of the following I program interrupts result in the exchange of program PSWs.

Invalid Address: The I unit calculates an instruction address (to fill the A-B register) that is outside available storage.

A-B SAP: A SAP violation occurs on an IC fetch of a read-protected storage location.

Specification: The I unit calculates an instruction address that does not fall on a halfword boundary.

Operation: The bit configuration of the operation code field is not that of an assigned instruction.

Privileged Operation: Any of the following instructions are encountered in the problem state:

Start I/O
Halt I/O
Test I/O
Test channel
Insert storage key
Set storage key
Set system mask
Load PSW
Read direct
Write direct
Diagnose

Execute: The object instruction of an execute is another execute.

Specification: Any of the following conditions occur:

1. The I unit calculates an operand address that does not fall on an integral boundary.
2. The R1 field of an instruction specifies an odd general register when a pair of general registers are to be used for an operand.
3. A floating-point register other than 0. 2, 4, or 6 is specified.
4. The block address specified in the set storage key or insert storage key instruction has the four low-order bits not all zero.

Supervisor Call Instruction

The supervisor call interrupt exchanges supervisor call PSWs. The interrupt request is made when the instruction is decoded.

As the supervisor call interrupt sequence is entered, the interrupt code field of the current PSW is set to the bit configuration of the R1 and R2 fields of the supervisor call instruction. The R1 and R2 fields of the instruction do not specify general registers, as they do in other instructions: they are used only as a means to convey some message to the operating system program via the old PSW.

Execute Operation Recovery

A recovery only sequence is initiated at the completion of the object instruction of the instruction execute, provided the object instruction is not a successful branch. Because the object instruction was interjected into the normal sequence of instructions, disturbing the prefetched instructions in the A-B registers, it is necessary to recover to the instruction following the execute instruction so that the normal instruction sequence can be resumed.

The execute operation recovery has the lowest priority in the interrupt system's priority scheme; in the event of any other interrupt associated with execute's object instruction, that interrupt would have to be serviced before returning to the normal instruction stream.

As with the program store compare recovery described earlier, the execute operation recovery does not use the regular interrupt sequencers because no PSWs are exchanged. Once priority for execute operation recovery is established, the IC recovery trigger turns on which causes the necessary fetch to A or B.

If the object instruction of the execute is a successful branch, the execute operation recovery is not taken because the branch instruction is purposely changing the normal sequence of instructions.

Interrupt Sequence Initiation

There are four paths from which the interrupt sequence can be entered: (1) from the I unit, (2) from the E unit, (3) by direct entrance, and (4) from the interrupt sequence.

Entrance from the I Unit

- Interrupt condition detected during T1 or T2.
- Entrance trigger is I interrupt end.
- E unit is blocked; interrupt sequence occurs.

Certain I program interrupts are detected during T1 and certain I program interrupts are detected during T2. Regardless on which cycle, T1 or T2, the interrupt request is detected, the I unit stops at the end of the good T2 and turns on the entrance trigger for the interrupt sequence, I interrupt end (Figure 84).

At the end of T2, as I interrupt end comes on, the I to E transfer line causes the IC and ILC to be updated just as if no interrupt were going to occur; however, no execution unit is allowed to start and storage requests are blocked.

The I interrupt end trigger stays on for one cycle and then turns on the first interrupt sequencer, interrupt cycle 1, at the same time resetting the I unit. The interrupt sequencers then come on in succession and perform the interrupt.

As stated previously, requests can be detected during either T1 or T2. It is possible that two interrupts may occur in the I unit, one during T1 and the other during T2. If this happens, only the T1 interrupt is detected; the T2 interrupt is ignored.

Entrance from the E Unit

- Interrupt condition detected during control last cycle.
- Entrance trigger is EXIT.
- I unit is blocked; interrupt sequence occurs immediately after control last cycle.

All interrupts other than the I program group, machine check, and IPL load PSW, are detected in the E unit during control last cycle. If one or more interrupt requests are present during control last cycle, the execution interrupt trigger, EXIT, turns on following control last cycle (Figure 84). One of the requests is given priority, and EXIT, which is on for one cycle, turns on the first interrupt sequencer, interrupt cycle 1. The interrupt sequencers then come on in succession and perform the interrupt.

NOTE: Control last cycle is defined as (ELC · $\overline{\text{IE Busy}}$) + (IELC · $\overline{\text{E Busy}}$) + ($\overline{\text{IE Busy}}$ · $\overline{\text{E Busy}}$).

At the time of control last cycle, the I unit may be in any of its own cycles. If an interrupt request is detected during control last cycle, and if a good T2 coincides with this control last cycle, the detected request blocks the updating of the IC and ILC and blocks the start of an execution unit. The EXIT trigger blocks storage requests. If a good T2 coincides with control last cycle, and if there is an I unit interrupt request along with the E unit request, I interrupt end and EXIT will attempt to come on at the same time; EXIT will turn on, holding off I interrupt end. EXIT then turns on interrupt cycle 1, at the same time causing a reset of the I unit.

If there is no coincidence of control last cycle and good T2, there is no conflict between the I unit and E unit interrupts; EXIT turns on interrupt cycle 1 and resets the I unit.

Direct Entrance

- Machine check or IPL.
- No entrance trigger is used; interrupt sequencers turned on directly.

The machine check and IPL load PSW interrupts do not turn on either entrance trigger (EXIT or I interrupt end), but instead cause a direct entrance into the interrupt sequence by forcing on interrupt cycle 1. Because these two interrupt requests follow a CPU reset, there are no testing cycles present, that is, no control last cycle or T1-T2; no other interrupt can be detected, therefore, at the same time as machine check or IPL load PSW.

Interrupt Last Cycle

During an interrupt sequence, none of the program caused interrupts can arise because the I unit and E unit are stopped. An asynchronous signal, however, can arise, such as a channel interrupt request, a timer advance request, and so on. Also, an interrupt sequence might load a new PSW that has a different system mask than the one it replaced, now enabling a channel or external interrupt that might have been pending.

At the end of each interrupt sequence, during interrupt last cycle, a test is made for outstanding interrupt requests. If any are present, EXIT is turned on immediately after interrupt last cycle and another interrupt sequence is taken for the request that has the highest priority.

Interrupt Sequencing

- Seven sequencing triggers:
  1 and 2 -- fetch new PSW
  3 and 4 -- store current PSW
  5 and 6 -- parity check new PSW
  Interrupt last cycle -- tests for other outstanding interrupts

The interrupt requests are checked during T1 or T2 and during E last cycle. The request with the highest priority is selected, the I unit and E unit are stopped, and the interrupt sequence is entered.

Most of the interrupts use the same fixed sequence, which is the turning on in succession of seven sequencing triggers. Each sequencer does a specific job, but the result of all of them is the exchange of the PSWs.

The sequence starts by the turn-on of the first sequencer, interrupt cycle 1 (Figure 85). The first sequencer stays on for only one cycle and then turns on the second sequencer, interrupt cycle 2. The combination of interrupt cycles 1 and 2 sends the new PSW address to SAR and makes a storage request for the fetch of the new PSW. Interrupt cycle 2 then stays on until the fetch accept comes back from storage. When the accept is received, the sequence advances to interrupt cycle 3.

While storage is timing out for the fetch, interrupt cycle 3 turns on for one cycle and then turns on interrupt cycle 4. The combination of interrupt cycles 3 and 4 makes a storage request and sends the PSW register contents to the K register and on to the SBI. At this point, storage is fetching the new PSW, the old PSW is on the SBI, and store request is on. Interrupt cycle 4 stays on until storage is obtained for the store, but since the fetch and the store reference the same storage, the store cannot start until the fetch is completed.

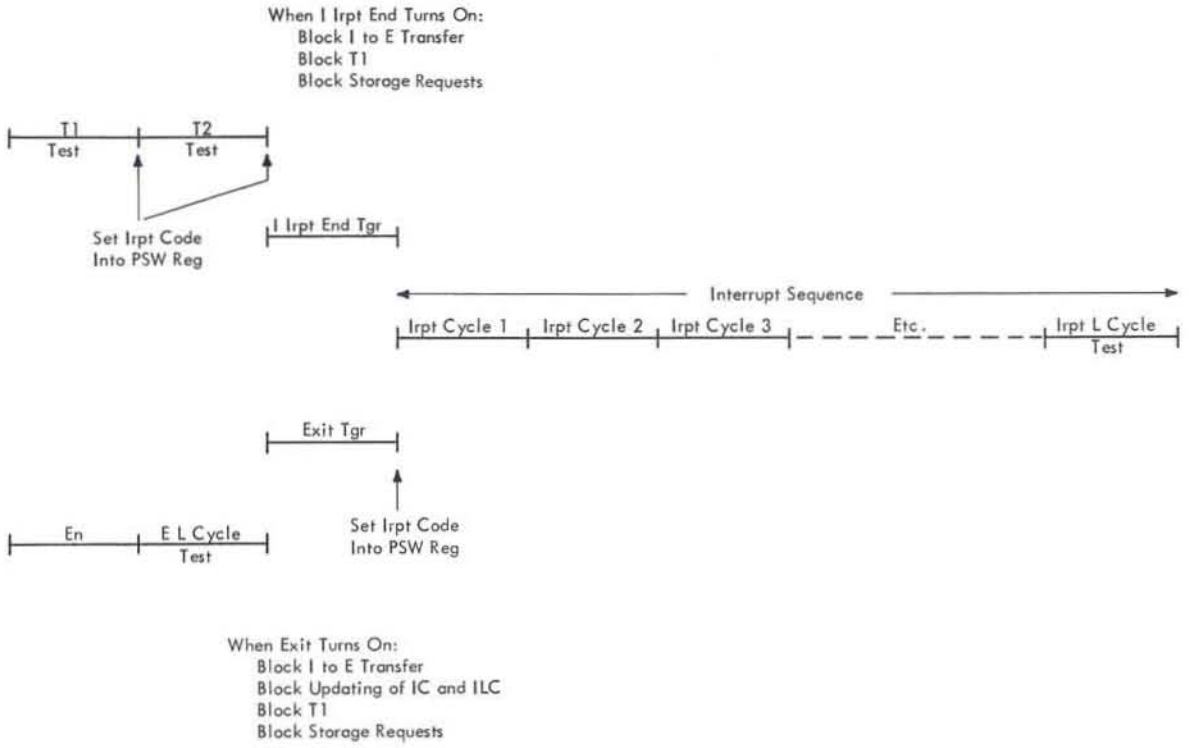


FIGURE 84. INTERRUPT SEQUENCE INITIATION

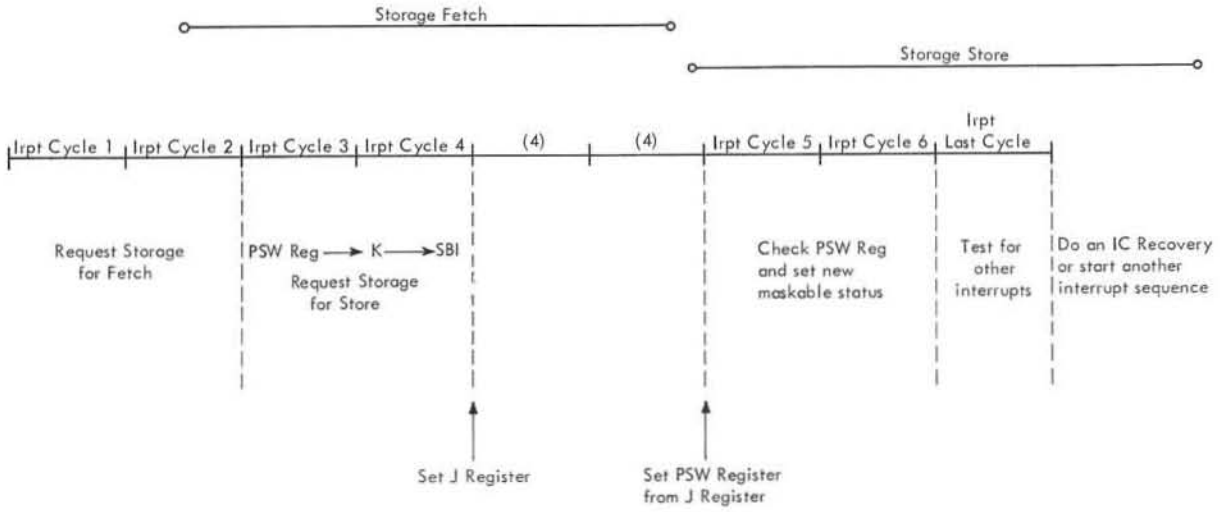


FIGURE 85. BASIC INTERRUPT SEQUENCE

Storage advance is received for the fetch. The advance pulse sets the double word from storage (the new PSW) into the J register. As soon as storage times-out for the fetch, it is immediately started again because store request is on. With interrupt cycle 4 on, the sequencing waits until the accept is received for the store; when this occurs, the contents of J are set into the PSW register and the sequence advances to interrupt cycle 5. At this point, the new PSW is in the PSW register, and the storage is on its way for the store (the old PSW is on the SBI). The interrupt sequence is now finished with storage.

The only jobs to be completed are to check the PSW register for parity and to test for any other outstanding interrupts. The checking is accomplished by sequencers interrupt cycle 5 and interrupt cycle 6; each stays on only one cycle and each checks a half of the PSW register. After interrupt cycle 6, one more cycle occurs, interrupt last cycle, at the beginning of which IC recovery is turned on. During this cycle, any interrupt request that is present is recognized, EXIT is turned on, IC recovery is reset, and entrance is made again to another interrupt sequence. If during interrupt last cycle there are no outstanding requests, the IC recovery continues, prefetching the instructions located by the IC of the new PSW.

The interrupt sequence just described is the normal fixed sequence. The interrupts that use this pattern are:

1. I program
2. E program
3. BCU (invalid store address and SAP)
4. External (timer, console, and external signal).
5. Supervisor call instruction

Modified Sequences

Some interrupts, for one reason or another, require modification of the fixed sequence. These modified sequences are described separately in this section.

Input/Output Sequence

- Interrupt cycle 1 stays on until the channel stores the CSW.

The processing of channel interrupts departs from normal sequencing only in that a pause is injected between interrupt cycle 1 and interrupt cycle 2 (before the fetch request is made); during this pause, the channel that originates the interrupt request performs its interrupt routine, which includes storing its channel status word. The unit address bits, which along with the channel address are set into the old PSW instead of an interrupt code, are not available from the channel until the channel has completed its processing.

At the same time that interrupt cycle 1 turns on to start the I/O interrupt sequence, an interrupt response signal is sent to the channel; this signal serves as a "go" signal for the channel to start its processing. When the channel has completed this processing, it sends a release signal to the interrupt controls which results in turning on interrupt cycle 2; the channel and unit addresses are set into the PSW register at this time, and the interrupt sequence continues the rest of the way the same as the regular fixed sequence.

Timer Advance Request Sequence

- Sequence consists only of interrupt cycles 1, 2, and interrupt last cycle.
- E unit decrements the timer word.

The timer advance request sequence starts like the regular fixed sequence with interrupt cycles 1 and 2 making a fetch request. The address sent to storage is that of the timer word, location 80. The PSWs are not involved in a timer advance; consequently, the timer advance sequence does not use interrupt cycles 3 through 6.

After interrupt cycle 2, while storage is timing out for the fetch, an E unit control trigger turns on (timer J to M trigger) which controls the updating of the timer word. The sequencing now waits until the timer word returns to the J register. When this occurs, the E unit decrements the word by using the M register and the adder. The updated word is sent to K and on to the SBI. Meanwhile, the E unit makes a storage request to store away the updated timer word into location 80.

When the E unit receives the accept for the storage store cycle, the interrupt controls are again activated, turning on interrupt last cycle. The actions during this cycle are the same as with the regular fixed sequence: outstanding requests, if there are any, are recognized, EXIT is turned on, and another interrupt sequence is entered; if there are no requests, processing resumes with an IC recovery.

Recovery-Only Sequence

During E last cycle, recovery-only requests (program store compare or execute operation) are considered along with other interrupt requests. The entrance trigger EXIT is turned on, and if recovery only receives priority, the triggers blocking the I unit are turned off and the IC recovery trigger is turned on. The machine proceeds directly into an IC recovery without any of the interrupt sequencers turning on.

IPL Load PSW Sequence

- Sequence consists of interrupt cycles 1, 2, 5, 6, and interrupt last cycle.

The IPL procedure starts with the load pushbutton which resets the system. Next, the channel and unit addresses in the rotaries are sent to the channel along with an IPL pulse, which starts the loading of storage by the channel.

When the loading of storage is completed, the channel sends a release signal to CPU; the release signal results in the turn-on of interrupt cycle 1.

The ensuing interrupt sequence consists of interrupt cycles 1, 2, 5, 6, and interrupt last cycle. Cycles 1 and 2 make a fetch request of storage location 0, which contains the initial PSW. Interrupt cycle 2 stays on until the storage accept comes back. When accept is received, interrupt cycle 2 turns off, but cycles 3 and 4 do not come on because no store is to be made; instead, the interrupt IPL buffer trigger turns on. This trigger is used to fill the storage time-out gap; it stays on until the word at location 0 returns from storage and is set into the J register.

The interrupt IPL buffer trigger then turns off and interrupt cycle 5 turns on; at the same time J goes to the PSW register. At this point, the interrupt sequence proceeds to completion the same as a normal fixed sequence, ending with an IC recovery.

Machine Check Sequence

The machine check interrupt sequence is the same as the normal fixed sequence except for the manner in which the sequence is entered. At the completion of logout, which is caused by a machine error, the CPU is reset. A reset line, in conjunction with the log complete and restart triggers, turns on the machine check control trigger in the interrupt controls. When the reset is completed, the control clock is started and this trigger, with a clock pulse, turns on interrupt cycle 1, starting the interrupt sequence.

Special Conditions

Wait Status

- Console, timer overflow, external signals, and I/O interrupts may occur during wait status.

Position 14 of the PSW indicates whether the CPU is in the wait state (1 bit) or the running state (0 bit). In the wait state, the CPU does not execute instructions, but remains suspended until an interrupt occurs. The interrupts that may occur are timer advance, external (console, timer overflow, and external signals), and I/O. These requests start the interrupt sequence, which except for timer advance, result in the loading of a new PSW. The new PSW may or may not change the running status of the CPU, depending on the new value of position 14. In the wait state, the timer is advanced normally without effecting the wait status.

A wait trigger within the interrupt controls is associated with the status bit in the PSW. This trigger reflects the true status of the CPU since there is a delay between the setting of the PSW wait bit and the halt of processing, and a delay between the resetting of the wait bit and the resumption of processing.

After every setting of a new PSW, the wait bit is observed before normal processing is continued. At the end of the load PSW instruction or at the end of every interrupt sequence that fetches a PSW, the wait bit prevents the instruction fetching for the new IC value, and the wait trigger is set if no interrupts require servicing. Any of the interrupt signals (console, timer advance, external signals, or I/O) that arise in the wait state will reset the wait trigger and set EXIT, which leads into the interrupt sequence.

If the CPU is in the wait state and the halt trigger is set (by the stop pushbutton), all processing is inhibited. This includes the external and I/O interrupts and timer advances. All operations are deferred until the halt trigger is reset (by the start pushbutton). When processing resumes, the highest priority interrupt that is outstanding will be serviced.

Storage Interlock

- Interrupt sequence cannot start until storage goes not busy.

An interlock is provided at the entrance to interrupt sequencing for two reasons:

1. If a store operation is in process, the interrupt sequence must be delayed for a sufficient time to allow for a possible SAP check to return. This avoids the possibility of missing a protection interrupt.

2. If a fetch is outstanding, the interrupt sequence must be delayed for a sufficient time to allow for the effects of the returning word to be nullified. (An unwanted effect might be the turn-on of the J loaded trigger.)

The interlock is accomplished by the CPU storage busy line from the BCU, which designates that the CPU has a storage operation in process. The interlock is provided during the set of EXIT or I interrupt end; whichever trigger is used remains on and prevents the turn-on of interrupt cycle 1 until the CPU storage busy line drops.

THEORY OF OPERATION

This section describes the detection and entrance circuits for interrupts and details of the various interrupt sequences. The interrupts described in this section are grouped according to the kind of sequence they cause, not necessarily in the order of their priority.

CPU SAP, Invalid Store Address, E Program, and External

Detection and Entrance Logic

Figure 5350 shows the interrupt requests that are detected during E last cycle, during interrupt last cycle, or during the wait state, resulting in the turn-on of the entrance trigger EXIT. The requests entering this figure, with the exception of channel, timer advance, and recovery only, cause the normal fixed sequence shown. This group of requests includes:

1. CPU SAP check
2. CPU invalid store
3. E from E group (12 E program interrupts)
4. Console
5. Timer overflow
6. External signals

The three exceptions, channel, timer advance, and recovery only, also turn on EXIT, but they result in modified sequences. These three exceptions are described later.

In Figure 5350 note that the requests SAP, invalid store, recovery only, and the E program group (E from E), are detected only during control last cycle. If any of these requests occur, they do so because of the I unit processing or the E unit execution of an instruction; in either case, they are detected at the end of the execution of the instruction in question so that an interrupt sequence can immediately follow.

The remaining requests shown in the figure, timer advance request, channel interrupts, and the external group (timer, console, and external signals), may occur at anytime with respect to CPU operation. These interrupt requests, therefore, are observed not only during E last cycle along with the group of requests previously mentioned, but also during interrupt last cycle because they might occur while an interrupt sequence is in process, and also during the time that the CPU is in wait status. This asynchronous group of requests, therefore, may cause an interrupt sequence to start immediately after an instruction execution, immediately after another interrupt sequence, or during a CPU wait, depending on when the request occurs.

Each of the incoming requests in Figure 5350 comes from a request trigger and latch arrangement similar to that shown for one of the external signal requests. A description of the turn-on and turn-off conditions of the request triggers follows.

The CPU SAP and CPU invalid store request triggers are in the BCU. They are turned on by a SAP or invalid store condition detected in the BCU, and they are turned off by interrupt end reset, which occurs at the beginning of interrupt cycle 6 of any interrupt sequence.

The E interrupt from E request trigger is the E interrupt trigger in the E unit. It is turned on by any of the 12 E program conditions that may occur during the execution of an instruction, and it is turned off by interrupt end reset.

The request trigger for console interrupts is the console interrupt trigger. It is turned on by a signal from the interrupt pushbutton, and it is turned off by interrupt cycle 4 of a console external interrupt sequence.

The request trigger for timer overflow is the timer interrupt trigger. It is turned on when the timer word goes from positive to negative during a timer advance interrupt sequence, and it is turned off by interrupt cycle 4 of a timer external interrupt sequence.

The external signal request triggers are shown in Figure 5350. They are turned on by signals from an outside source and turned off by interrupt cycle 4 of an external interrupt sequence.

The request triggers for recovery only, timer advance request, and channel interrupts are described later.

Most of the request triggers are set with an A pulse and are followed by latches that are released at not L time; at not L time the request trigger outputs are transferred to the detection circuits, waiting there for E last cycle to occur so that they can be detected.

The asynchronous requests, however, consisting of external signal, timer advance request, console, and channel, have special conditions imposed on their request trigger latches, as stated in the note in Figure 5350. This group of requests must be handled in a special manner because of the following:

Once an interrupt request becomes outstanding, it blocks the start of the execution units. If one of the asynchronous interrupt requests were allowed to become outstanding (the latch reflecting the trigger status) as soon as the request occurred, it might happen while none of the execution units were running, such as between instructions during an instruction step operation. The outstanding request would block the start of the execution unit, and without an execution unit running, there would be no E last cycle to detect the request; the CPU would hang up. To prevent this condition, the latches of the asynchronous request triggers are released only if an execution unit is already in operation (E or IE busy).

Also, if the stop pushbutton is pressed while in the wait state, all interrupts are to be blocked, including those that normally could occur during wait. The stop condition of CPU turns on the interrupt priority hold trigger, an output of which prevents the release of the asynchronous request latches. In the stopped condition, therefore, the asynchronous interrupts cannot be detected even though the wait trigger is on.

Note in Figure 5350 that any of the requests entering the diagram can turn on EXIT, resulting in the start of an interrupt sequence. All requests present at the time also feed a priority generating circuit, which grants priority to the highest request. It is the priority line generated that determines whether the sequence that follows is to be a normal sequence, as in Figure 5350, or one of the modified sequences (see "Modified Sequences").

The priority generated also allows its corresponding request to set the interrupt code field to the proper configuration, and determines the addresses to be used for the PSW fetch and store. For example, a CPU SAP request turns on EXIT, starting the sequence; the SAP priority (it is the highest of the requests in Figure 5350) gates the SAP request to set a bit into position 29 of the interrupt code field and generates address 104 for the PSW fetch and address 40 for the PSW store. The addresses are those of the program PSWs.

Interrupt Sequence

Figure 5350 shows the details of the interrupt fixed sequence that occurs as a result of any of the E program, SAP, invalid store address, or external interrupts. Note that at the turn-on of EXIT, various blocks are brought up so that the current PSW will not be disturbed (it has already been updated for the instruction just completed) and so that no new storage cycles may be started by the CPU, such as by the I unit; storage is now available for the exchange of the PSWs.

Also at the turn-on of EXIT, the interrupt priority hold trigger turns on and remains on throughout the interrupt sequence; it is turned off at the end of interrupt cycle 5 by interrupt end reset. The interrupt priority hold trigger prevents another turn-on of EXIT, thus making sure that another interrupt sequence cannot start until the current one is completed.

Interrupt reset comes up during EXIT, resetting the I unit, which is now in T1 of the next instruction. The A pulse following the turn-on of EXIT turns on interrupt cycle 1, provided storage is not busy from a prior CPU operation. If storage is busy, an interlock CPU storage busy line blocks the turn-on of interrupt cycle 1 until storage goes not busy; EXIT, therefore, stays on until storage is available for the interrupt sequence.

The A pulse following the turn-on of EXIT sets the interrupt code bits into 16-31 of the PSW register.

Interrupt cycle 1 (always one cycle) sets up SAR with the address of the new PSW to be fetched. Interrupt cycle 2 makes the fetch request, staying on until accept is received. When the accept is received, the sequence advances to interrupt cycle 3.

As storage starts timing out for the fetch, interrupt cycles 3 and 4 start storing the current PSW. Interrupt cycle 3 (always one cycle) sets up SAR with the store address and gates the right-half of the PSW through the incrementer to the left-half of K. The A pulse at the end of interrupt cycle 3 latched sets K and brings up store request.

During interrupt cycle 4, the left-half of K is sent through the shifter and back to the right-half of K, under control of the store PSW trigger in the E unit. (This data was originally the right-half of the PSW.) At the same time, the left-half of the PSW is gated through the incrementer into the left-half of K; the following A pulse sets K. The BCU sets the contents of K into the SBI latches because store request is up. Interrupt cycle 4 stays on until the accept comes from BCU for the store. The current PSW is now on the SBI and will be stored as the old PSW whenever the storage store occurs.

In due time, the fetched word returns from storage and sets into J. Storage times-out for the fetch and is immediately started again for the store, provided CPU has storage priority. On the advent of store accept, the PSW register is set from J, interrupt cycle 4 turns off, and interrupt cycle 5 turns on.

Interrupt cycle 5 gates the right-half of the PSW through the incrementer for parity checking; interrupt cycle 5 also brings up interrupt end reset, which turns off the request triggers for CPU SAP, invalid store address, and E program. If the interrupt sequence was caused by one of the external group of requests (console, timer, or external signals), the request triggers for these interrupts would be turned off at interrupt cycle 4.

Interrupt cycle 6 gates the left-half of the PSW through the incrementer for parity checking. At the end of interrupt cycle 6, the block T1-M and block IC-M triggers are turned off and interrupt last cycle is turned on. If the new PSW position 14 does not specify wait status, the IC recovery trigger is turned on.

The I unit now starts an IC fetch to recover to the setting of the IC in the new PSW, but at the same time, the interrupt last cycle tests for other outstanding interrupts. If an outstanding interrupt is found (it could be only channel, timer advance request, console, or external signal), EXIT is immediately turned on, the I unit is again reset, and another sequence is taken.

I Program

Detection and Entrance Logic

Figure 5351 shows the interrupt requests that are detected in the I unit during T1 and T2, resulting in the turn-on of the entrance trigger I interrupt end. The interrupt conditions entering this figure cause the normal fixed sequence shown in the figure. The sequence is similar to the fixed sequence described in the preceding section, the main differences being the manner in which the sequence is entered and the setting of the interrupt code field.

Of the eight interrupt requests entering Figure 5351, four of them, invalid address, SAP, invalid operation, and address specification, occur during T1. The four requests enter a priority generating circuit (not shown) which selects the request that has highest priority. The priority generated allows its corresponding request to set the interrupt code field to the proper configuration; the actual set occurs at the turn-on of T2.

Also at the turn-on of T2, any of the four requests turn on the I program interrupt trigger; this trigger stays on until the completion of the interrupt sequence, generating the fetch and store addresses for the PSW exchange (program PSWs). With the I program interrupt trigger on, the entrance trigger, I interrupt end, turns on at the normal I to E transfer time, provided no interrupt request, detected during E last cycle, turned on EXIT.

If storage is not busy, I interrupt end stays on for one cycle and then turns on interrupt cycle 1; if storage is busy, I interrupt end stays on until the A pulse following the fall of the storage interlock line, then turns on interrupt cycle 1. Interrupt cycle 1 starts the fixed interrupt sequence.

At the same time that I interrupt end turns on, the interrupt priority hold trigger also turns on. This trigger stays on throughout the interrupt sequence and prevents the start of another sequence until the current one is completed.

The other four requests entering Figure 5351, privileged operation, execute to execute, specification, and supervisor call, occur during T2. These four requests enter a priority generating circuit (not shown) which selects the request that has highest priority. The priority generated allows its corresponding request to set the interrupt code field to the proper configuration; the actual set occurs at the normal I to E transfer time. Also at this time, both I program interrupt (except for supervisor call) and I interrupt end are turned on, which do the same jobs as described previously.

If interrupt requests occur during T1 and T2, the T1 request turns on I program interrupt and sets the interrupt code as previously described. The I program interrupt trigger is on by the time the T2 request is detected (I to E transfer time), thus blocking another set of the interrupt code field. The T1 request has effectively taken priority over the T2 request.

Interrupt Sequence

Figure 5351 shows the fixed sequence that results from any of the following interrupt requests:

1. Invalid instruction address
2. SAP violation
3. Instruction address specification
4. Invalid operation code
5. Privileged operation
6. Execute to execute
7. Specification
8. Supervisor call instruction

The first four requests occur during T1; the last four occur during T2. Any of the eight requests result in the turn-on of the entrance trigger, I interrupt end.

Because an interrupt request is outstanding at the normal I to E transfer time, the start of the execution units is prevented. The interrupt reset that occurs as a result of I interrupt end being on resets the I unit, thereby blocking further I unit processing. All CPU operations, therefore, are halted in favor of the interrupt sequence.

Subsequently, the interrupt sequencing triggers come on in succession, performing the exchange of the program PSWs. The detailed action of each sequencer is the same as in the fixed sequence described for CPU SAP, invalid store address, E program, and external.

Input/Output

Figure 5352 shows the detection logic and the interrupt sequence for I/O interrupts.

An interrupt request from a channel turns on its priority A trigger, provided the request is enabled by a system mask bit in the PSW. (More than one channel may send a request at the same time.) The priority A triggers feed a channel priority circuit which selects the one channel that has highest priority of those making requests. The request that is granted priority turns on the channel interrupt trigger and also blocks further sets of the priority A triggers. One particular channel will now get the next channel interrupt sequence that is taken.

At E last cycle or interrupt last cycle (or during wait status), any of several outstanding interrupt

requests, including the request by the channel interrupt trigger, may turn on EXIT, which turns on interrupt cycle 1 to start an interrupt sequence. The channel request, along with other outstanding interrupt requests, also feeds a priority generating circuit; and if the channel has priority over the other requests, the interrupt sequence just started becomes a channel interrupt sequence.

Unlike the description of other interrupt sequences, interrupt cycle 1 does not lead directly into interrupt cycle 2; interrupt cycle 1 goes off after one cycle and the turn-on of interrupt cycle 2 depends on a release signal from the channel to CPU. This is accomplished as follows:

When interrupt cycle 1 turns on, the channel interrupt response trigger also turns on. The channel interrupt response trigger (1) sends a response signal to the channel that was granted priority over the other channels, and (2) prevents the turn-on of interrupt cycle 2 until release is received from the channel. The response signal causes the channel to proceed with whatever processing it has to perform, which includes storing the channel status word.

When the channel completes its processing, it sends a release signal to the CPU which turns on the channel interrupt release trigger. This trigger turns off the interrupt response trigger and turns on interrupt cycle 2. The interrupt sequence now continues just like the fixed sequences described previously, except that since channel priority is up, I/O PSWs are addressed for the exchange.

During interrupt cycle 5, interrupt end reset comes on, as it does for other interrupt sequences, and resets all the request triggers, including the channel interrupt trigger. It also resets the channel priority A triggers, dropping the priority granted line that was blocking the inputs to the priority A triggers. This allows a new sampling of the request lines from the channels according to a new system mask that may have been introduced as part of the new PSW.

The enabled requests again compete with one another for priority with the highest one obtaining priority, again turning on channel interrupt. During interrupt last cycle, the outstanding channel request is again ORed with other possible outstanding interrupt requests to turn on EXIT; and if the channel is granted priority over the other requests, another channel interrupt occurs.

Note that at the turn-on of interrupt cycle 1, no interrupt code is set into the PSW as in other interrupt sequences. Instead, a three-bit code that identifies the channel is set into PSW 21-23. This code is generated from an encoder that has as its input a line corresponding to the channel that receives the interrupt. When the channel sends release to the CPU, allowing the interrupt sequence to continue, the unit address is available from the channel on the unit address bus in. This unit address is set into PSW 24-31 at the same time as the turn-on of interrupt cycle 2.

Timer Advance

The timer advance detection logic, the timer advance interrupt sequence, and the updating data flow are shown in Figure 5353.

A fixed frequency signal from the power supply, through a 500-nanosecond singleshot, initiates the timer advance request. The frequency corresponds to the line frequency of the machine: 60 cps for 60-cycle machines; 50 cps for 50-cycle machines.

The singleshot output, after being synchronized with CPU timing, turns on the timer advance request trigger. Note that the turn-on of this trigger can be blocked by the disable interval timer switch (on the system control panel) or by a bit in position 12 of the MCW. Once the timer advance request trigger is on, however, it will remain on until a timer advance interrupt sequence is taken or until the CPU is reset.

During E last cycle or interrupt last cycle (or during wait status), any of several outstanding interrupt requests, including the request by the timer advance request trigger, may turn on the EXIT trigger, which turns on interrupt cycle 1 to start an interrupt sequence. The timer advance request, along with other outstanding requests, also feeds a priority generating circuit; if timer advance receives priority over the other requests, the interrupt sequence just started becomes a timer advance interrupt.

The timer advance interrupt sequence starts like any other interrupt sequence, that is, interrupt cycle 1 sets up the address of the fetch and interrupt cycle 2 makes the storage request. Interrupt cycle 2 stays on until storage is obtained and accept is received. At the end of the first interrupt cycle 2, however, the timer J to M trigger turns on due to interrupt cycle 2 and timer advance priority. The timer J to M trigger will cause the fixed-point arithmetic circuits in the E unit to decrement the timer word whenever the word returns to the J register.

When fetch accept is returned by the BCU, interrupt cycle 2, timer advance request, and interrupt priority hold turn off. The sequence does not continue with interrupt cycles 3 through 6, as it does with other interrupts, because there are no PSWs to be exchanged. Instead, the E unit is conditioned by

the timer J to M trigger to take over and decrement the timer word.

J advance sets the timer word into the J register; the word always fetches to the left-half of J because of the fixed-storage location from where it came (single word location 80). The timer J to M trigger and J loaded cause the left halfword of J to flush through the adder to the left-half of M. At the same time, bits are forced into position 53 and 55 of M (the right-half); these bits constitute the subtrahend to be used for the subtraction on the next cycle. (If 50 cps machine, bits are forced into 53 and 54.)

On the next cycle, timer J to M latched and an A clock turn on the timer subtract trigger which gates both halves of M into the adder, subtracting the forced bits from the timer word. The difference is set into K. The subtrahend bits being in M 53 and 55 amount to a reduction of 5 from the timer word, considering the units position of the timer word (0-31) to be position 23. Bits in the rightmost byte of the timer word, 24-31, would be used for decrementing if a more refined count were desired, with the lesser decrementing-amount occurring at a proportionately higher rate.

After the subtraction cycle, the updated timer word in K goes to the SBI and waits there to be sent back to storage. The storage request for the store is made while the subtraction is being performed; the subtraction is completed and the result is on the SBI in time to be stored. The store request is brought up by the timer advance trigger (turned on by J advance and the timer J to M trigger). The timer advance trigger also conditions the return back to the interrupt sequence that was interrupted after interrupt cycle 2. The return is made by store accept turning on interrupt last cycle. Concurrent with the turn-on of interrupt last cycle is the turn-on of IC recovery and T1.

If no interrupt requests are outstanding during interrupt last cycle, an IC recovery takes place and the program continues; if an outstanding request is detected, EXIT is turned on again, the I unit is reset, and another interrupt is taken.

Recovery Only

Figure 5354 shows the entrance logic and sequence for recovery only. Recovery only is not truly an interrupt because no interrupt sequence occurs. But recovery-only conditions are considered as interrupt requests because they must obtain priority over other interrupt requests that may be outstanding before the recovery-only operation can occur.

A recovery-only condition occurs when:

1. An instruction stores into the location of a possible next instruction to be executed.

2. The object instruction of an execute is something other than a successful branch.

The first of these two conditions is a program store compare condition, detected in the I unit. The condition turns on the IC recovery required trigger. At E last cycle, any of several interrupt requests, including the request from the IC recovery required trigger, may turn on EXIT. Also, the IC recovery request competes with the other requests for priority. If another request is granted priority over IC recovery, that interrupt is started by the turn-on of interrupt cycle 1; but if IC recovery is granted priority, the turn-on of interrupt cycle 1 is blocked and EXIT turns on the IC recovery trigger.

The second of the two recovery-only conditions is an execute instruction that does not branch. This condition has the lowest priority of all the interrupt requests. At the I to E transfer time that begins the object instruction's execution, the IC recovery required trigger turns on; but if the execution is a successful branch, the IC recovery required trigger is turned off before E last cycle arrives. If not turned off by a successful branch, the IC recovery required trigger turns on EXIT and tries to get priority the same as described for program store compare.

Note that the requests that turn on EXIT have priority over those that turn on I interrupt end, and that execute operation turns on EXIT but does not have priority over anything. This apparent conflict is solved by the following:

Note in Figure 5354 the IC recovery required trigger turn-on consisting of execute operation latch, I to E transfer, and no interrupt from I. The no interrupt from I leg is actually a part of the priority system since it allows the execute operation to request an IC recovery only if there are no I program or supervisor call requests.

Once the IC recovery trigger turns on, the fetch to A or B is as shown in the sequence in Figure 5354. This is the same as normal I unit prefetching and is described in detail in "Instruction Preparation."

Machine Check

The machine check interrupt sequence is the same as the full fixed sequence shown in Figure 5350 except that interrupt cycle 1 is turned on directly; EXIT is not used.

The turn-on of interrupt cycle 1 is accomplished by using the logic shown in Figure 5355. First, a machine error stops the clock and causes a logout operation. The logout, in addition to storing the status of the CPU triggers, turns on the restart trigger. When the logout is complete, denoted by the word counter at 18 (19 words have been stored), the log complete trigger turns on. This trigger, in conjunction with restart, brings up CPU reset, which

turns on the machine check control trigger and resets the CPU. When the CPU reset ends, the clock starts and interrupt cycle 1 turns on.

The machine check control trigger resets the interrupt code field to all 0s and generates the PSW addresses.

IPL (Load PSW)

The IPL operation loads storage and then causes an interrupt sequence, during which the PSW register is loaded from location 0.

Figure 5356 shows the entrance logic and the interrupt sequence.

The IPL procedure starts by depressing the load button on the system control panel or on the operator's console. The system is immediately reset; the reset pulse turns on the IPL trigger. The IPL trigger gates the unit address in the rotary switches to all channels and turns on the IE1 trigger. IE1 gates the channel address in the rotary switch to the channels; this selects one of the channels. At the same time, the IE1 trigger turns on the IE2 trigger, which sends an IPL pulse to the channels. The IPL pulse is accepted by the selected channel. IE2 stays on for only one cycle and turns on the IE3 trigger, the only purpose of IE3 being to prevent another turn-on of IE2.

When the channel receives the IPL pulse, it proceeds to load storage. When the load operation is complete, the channel sends a release signal to the CPU. The release signal turns on the release buffer trigger, which synchronizes the release signal with CPU timing. The release buffer trigger turns on the release trigger. The release trigger turns on interrupt cycle 1, starting the interrupt sequence.

Interrupt cycle 1 and interrupt cycle 2 address storage and make a fetch request, the same as in other interrupt sequences. Interrupt cycle 2 stays on, as usual, until accept is received. Because no store is to be made, there are no interrupt cycles 3 and 4; instead, the interrupt IPL buffer trigger turns on, whose sole purpose is to turn on interrupt cycle 5 as soon as the word from location 0 is returned to the J register. The interrupt IPL buffer ANDed with J loaded turns on interrupt cycle 5 and sets the contents of J into the PSW register.

The rest of the sequence is the same as described for other sequences; that is, interrupt cycles 5 and 6 check the PSW for parity, and an IC recovery is started. Interrupt last cycle occurs as in other interrupt sequences, but at this point there should not be any outstanding interrupt requests.

STORAGE BUS CONTROL

INSTRUCTION PREPARATION

FLT, LOGOUT, MCW

INTERRUPTS

IBM FE Supplement

System/Unit	2075
Re: Form No.	223-2873-1
This Supplement No.	S26-7034
Date	January 1968
Previous Supplement Nos.	None

This supplement revises and updates Volume 2 of the Field Engineering Manual of Instruction on the IBM 2075 Processing Unit, Form 223-2873-1. This supplement incorporates the floating point changes released under Engineering change EC 705848E and the 2075 Model IH 75 configuration.

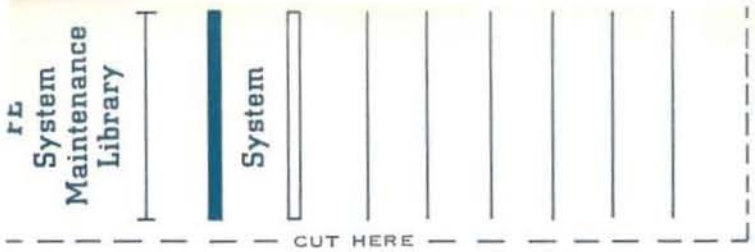
Incorporate this supplement by replacing Title page, Preface page, List of Illustrations page, pages 5, 6, 8, 9, 10, 12, 29, 30, 33, 34, 70, and 71 with corresponding pages attached to this notice.

Changes to text are indicated by a vertical bar to the left of the affected material. Revised diagrams are identified by a bullet (•) to the left of the figure caption. (In addition, changes that are not readily apparent are indicated by a vertical bar to the left of the changed area.

File this cover letter at the back of the publication. It will then serve as a record of the changes received and incorporated.

International Business Machines Corp., Product Publications Dept., Neighborhood Road, Kingston, N.Y. 12401

PRINTED IN U.S.A.



223-2873-1

Printed in U.S.A. 223-2873-1

IBM

International Business Machines Corporation
Field Engineering Division
112 East Post Road, White Plains, N.Y. 10601