SPERRY UNIVAC
1100/80 Systems

4x4 Capability Processor and Storage

Programmer Reference

A

SPERRY UNIVAC
COMPUTER SYSTEMS

UP-8604 Rev. 1

Page Status Summary

Issue: UP-8604 Rev. 1

Section	Pages	Update
Cover/Disclaimer		
PSS	1	
Contents	1 thru 14	
1	1 thru 11	
2	1 thru 4	
3	1 thru 21	
4	1 thru 26	
5	1 thru 78	
6	1 thru 71	
7	1 thru 35	
8	1 thru 13	
Appendix A	1 thru 10	
Appendix B	1 thru 14	
Appendix C	1 thru 21	
Appendix D	1 thru 6	
Appendix E	1 thru 77	
Index	1 thru 16	
User Comment Sheet		
Total: 420 pages and cover		

Section	Pages	Update

Section	Pages	Update

Contents

Page Status Summary

Contents

Index

User Comment Sheet

Figures

Tables

1. Introduction

1.1. General

This manual provides information on the SPERRY UNIVAC 1100/80 Systems central processor unit (CPU), main storage unit (MSU), buffer storage interface unit (SIU), and input/output unit (IOU).

The SPERRY UNIVAC 1100/80 Systems are high-performance, software compatible, extensions to the proven SPERRY UNIVAC Series 1100 Systems. The 1100/80 Systems enhance the efficiency of the Series 1100 Systems by offering dependable and highly effective processing in real-time, demand, and batch modes and excel in multiprocessing applications.

Although the Series 1100 Systems may differ in hardware design, software compatibility is maintained. All components of the 1100/80 Systems (CPUs, IOUs, MSUs, SIUs, and peripherals) are controlled by the SPERRY UNIVAC Series 1100 Operating System. Industry standard language processors and application software are provided. The flexibility of the 1100/80 Systems allows the user to select a system to best meet his individual requirements.

1.2. System Components and Configurations

The 1100/80 Systems configurations range from a 1x1 (one CPU and one IOU) system up to a 4x4 (four CPUs and four IOUs) system. Other system components include: main storage units, storage interface units, system console, system transition unit, system maintenance unit, and motor/alternators. Table 1-1 lists all fully supported configurations. Processor organization is basically that of a multitask processor that operates in a multiprogramming environment. A 2x2 configuration is shown in Figure 1-1 and Figure 1-2. Figure 1-3 shows a 4x4 configuration.

The expansion capabilities of the 1100/80 Systems units are:

- Main storage - expandable in 524K word increments to a maximum of 4194K words.
- Storage interface unit - expandable in 4K word increments to a maximum of 16K words. Two SIUs are the maximum for any configuration, one per cluster.
- Central processor unit - one to three CPUs may be added to the system to form clusters of two CPUs maximum. Each cluster has access to main storage through its own SIU. Each CPU in a cluster can access only the IOUs within the same cluster (i.e., no ring configurations).
- Input/Output unit - one to three IOUs may be added to the system to form clusters with the CPUs. No more than two IOUs may be present in a cluster. Each IOU can contain up to eight channel modules which gives a total of 32 channel modules for a maximum system.

- System transition unit – the system transition unit (STU) may be expanded to control and partition SIU segments, MSU banks, CPUs, and IOUs, into two smaller independent systems called applications, or offline for maintenance.

- System console – one or more system consoles may be added to the system, depending on configuration requirements and user requirements.

- System maintenance unit – up to two system maintenance units (SMUs) may be configured in a 2x2 system, and up to four SMUs may be configured in a 4x4 system.

- Motor/alternator – one to three motor/alternators may be used with the system, depending upon system power requirements and upon degraded mode operation requirements.

The 1100/80 System can also include a subsystem availability unit which, when used in conjunction with a byte channel transfer switch, can be used for partitioning both byte and word subsystems from a remote location.

1.2.1. Central Processor Unit

The CPU is the central element of a large–scale system that is capable of serving both business and scientific applications in batch, demand, and real–time environments. The CPU provides compatibility with prior Series 1100 Systems at the user object code level, depending on internal code selections, peripheral configurations, and software implementation of hardware enhancements and user interfaces.

The basic CPU consists of the following components:

- A control and arithmetic section that includes fixed–point and floating–point arithmetic; byte–oriented instruction handling; logical data manipulation; instruction, interrupt, and arithmetic control and control storage.

- A maintenance section which acts as a device and a control during offline maintenance procedures initiated by the maintenance processor.

- Interfaces for two IOUs, one SIU, one SMU, one STU, and the system interrupt network.

The CPU has the following general characteristics:

- A complete set of arithmetic, logical, manipulative, data transfer, and sequence control instructions.

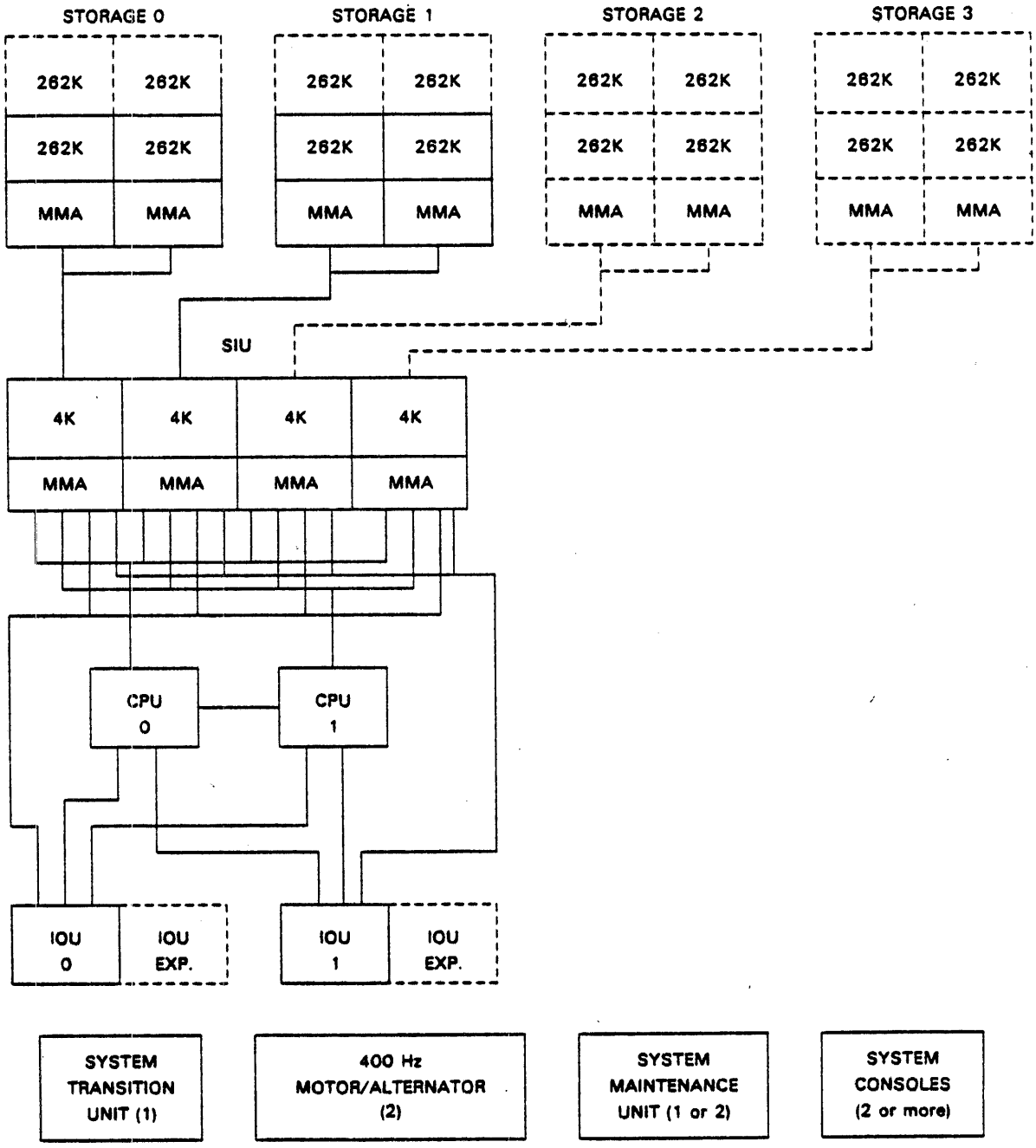
- A comprehensive relative addressing mechanism providing program segmentation and storage protection.

- An absolute addressing range of 16 million 36–bit words.

- A basic instruction fetch period of 200 nanoseconds.

- A general purpose microprogrammed arithmetic section.

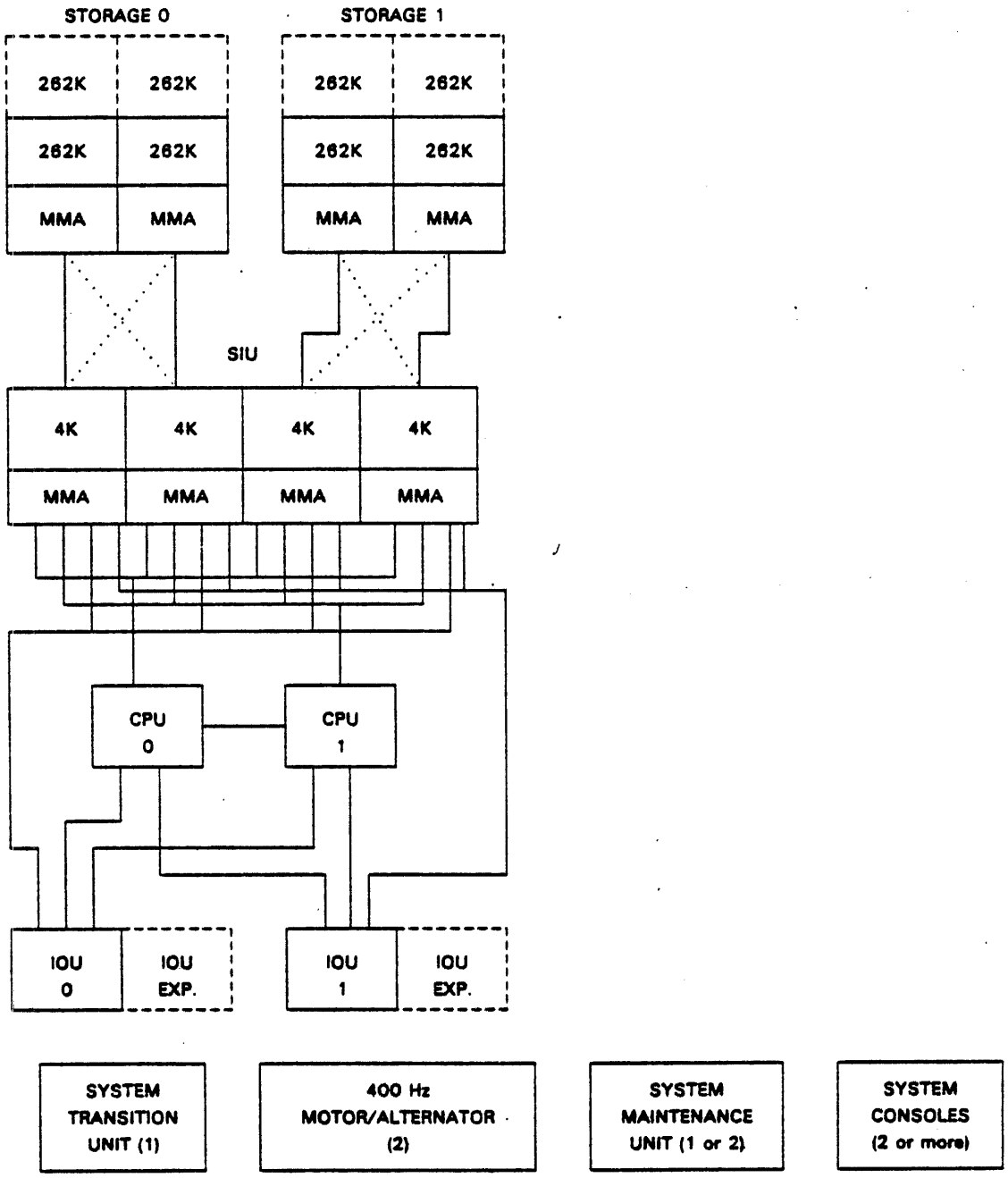
- A scientific accelerator module (SAM) feature that increases the execution speed of scientific floating–point type programs.



NOTES:

1. - - - - *Optional expansion.*
2. *MMA = Multi-module access unit.*

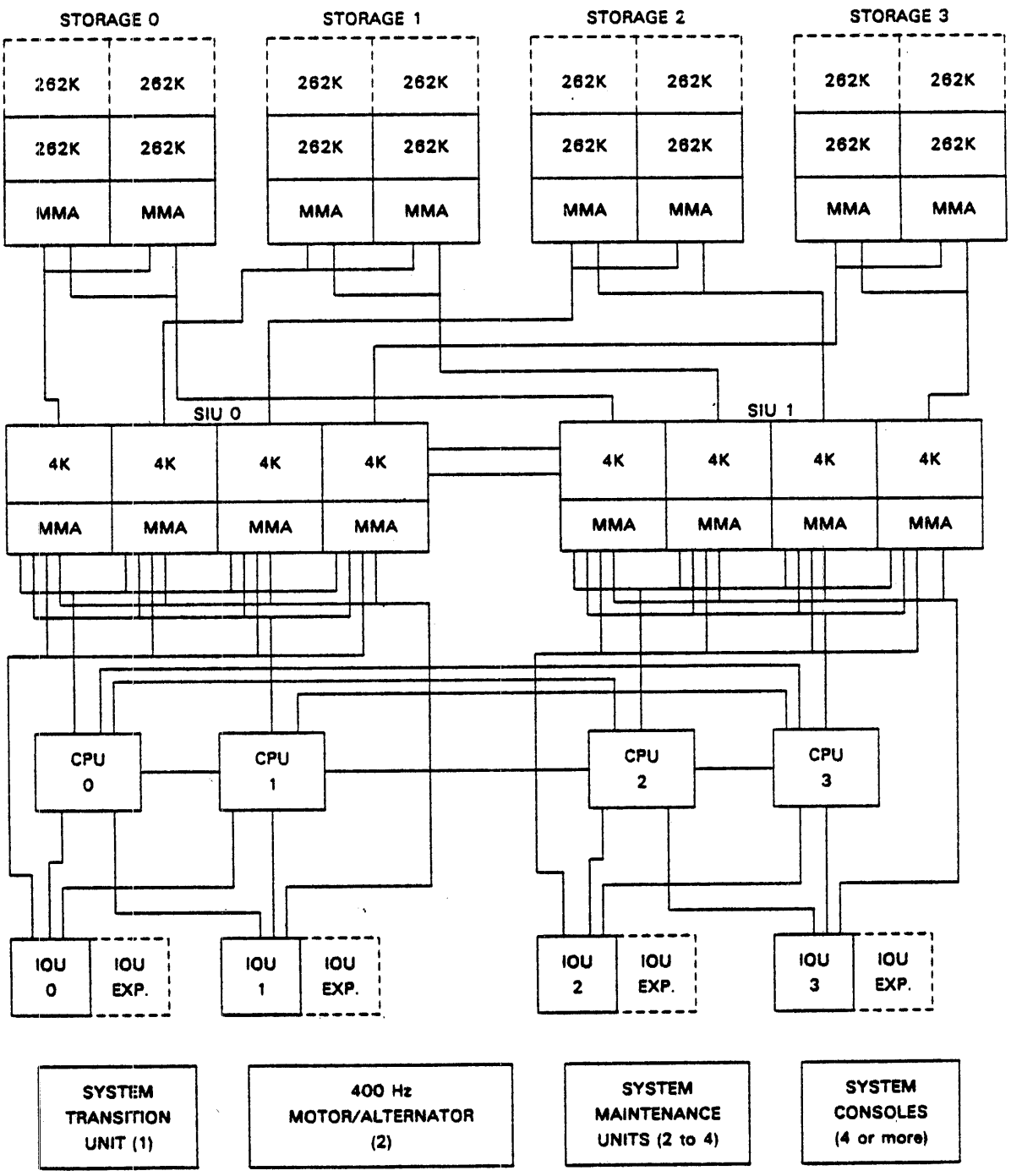
Figure 1-1. SPERRY UNIVAC 1100/82 2x2 System Segment/Cabinet Configuration



NOTES:

1. - - - - *Optional expansion.*
2. *MMA = Multi-module access unit.*
3. *Segment/Bank configurations are restricted to single cluster systems only.*
4. . . . . *Redundant interfaces only activated if the other segment is missing from the application.*

Figure 1-2. SPERRY UNIVAC 1100/82 2x2 System Segment/Bank Configuration



NOTES:

1. - - - - *Optional expansion.*
2. *MMA = Multi-module access unit.*
3. *Storage interface units must each contain an equal amount of storage.*

Figure 1-3. SPERRY UNIVAC 1100/84 4x4 System Segment/Cabinet Configuration

Table 1-1. Minimum/Maximum Functional Configurations

System Components	Configurations												
	1x1	1x2	2x1	2x2	2(1x1)	2x3	2x4	3x2	3x3	3x4	4x2	4x3	4x4
CPU	1	1	2	2	2	2	2	3	3	3	4	4	4
IOU	1	2	1	2	2	3	4	2	3	4	2	3	4
Main Storage Words	262K-4194K	262K-4194K	524K-4194K	524K-4194K	524K-4194K	524K-4194K	524K-4194K	524K-4194K	524K-4194K	524K-4194K	524K-4194K	524K-4194K	524K-4194K
SIU	1	1	1	1	2	2	2	2	2	2	2	2	2
SIU Words	4K-16K	4K-16K	4K-16K	4K-16K	8K-32K	8K-32K	8K-32K	8K-32K	8K-32K	8K-32K	8K-32K	8K-32K	8K-32K
System Console	1-N	2-N	1-N	2-N	2-N	3-N	4-N	2-N	3-N	4-N	2-N	3-N	4-N
STU	1	1	1	1	1	1	1	1	1	1	1	1	1
SMU	1	1	1	1-2	2	2	2	2-3	2-3	2-3	2-4	2-4	2-4
Motor/Alt.	1-2*	1-2*	1-2*	2-3*	2-3*	2-3*	2-3*	2-3*	2-3*	2-3*	2-3*	2-3*	2-3*

NOTES:

** A redundant motor/alternator may be configured only as a static unit.*

1. This chart includes the minimum configurations which can be supported in degraded operations.
2. N equals any number required.
3. Configurations greater than a 2x2 must be configured as two clusters (e.g., a 2x3 is configured as a 1x1 and a 1x2).
4. The number of motor/alternators configured depends on system load and motor/alternator power rating and onsite degraded mode operating requirements.

1.2.2. Storage System

The 1100/80 Systems storage system consists of large capacity, low cost, main storage units (MSUs); interfaced by moderate capacity, high speed storage buffers located in a separate cabinet called a storage interface unit (SIU). The high-speed storage buffers are used to achieve increased performance from the relatively low-speed MSUs. Each MSU can be divided into two separate banks. Each SIU contains from one to four logically independent storage buffers called segments. A main storage bank has a 2-port multimodule access (MMA) unit which allows access by two SIU segments, and provides a means for common accessing of main storage. The SIUs also provide the main storage interface to the system requesters (IOUs and CPUs). This interface is provided by 8-port MMA units associated with each segment of the SIU.

An SIU buffer segment is a set associative buffer having four words per block and four blocks per set. The interface between an SIU and main storage provides a 4-word wide transfer on a single read request, and a 1-word wide transfer on a write request. When a word is required from a block of data not already resident in the SIU, the request is made to the MSU and the new 4-word block of data is brought in. During the wait for this new block, the SIU segment is free to service other requests. On a write operation, the storing of a full word or half word consists of modifying the word in the buffer segment (if present), and also storing the word in the MSU. On a partial-word write, the word in the buffer segment (if present) is invalidated, and the partial write is made into the MSU. A next request for this invalidated word will cause its 4-word block to be brought in from the MSU.

Operation of the SIU buffers is transparent to software, although performance of the main storage is dependent on the organization of the software. This is true to the extent that software organization affects the miss rate, or percentage of instructions or operands not located in the buffer when first requested.

The minimum system contains 524K words of storage located in a single cabinet. This can be expanded to 1048K words in the cabinet, and up to four storage cabinets may be employed to provide a total of 4194K words of main storage. Three or more MSUs are required in a system larger than a 2x2.

The basic SIU contains 4K words of buffer storage and can be expanded by adding one to three logically independent 4K buffers, giving a maximum buffer size of 16K in the SIU. Two SIUs are required for all systems having more than two CPUs, allowing a maximum of 32K words of buffer storage in the system.

A detailed description of the SIU/MSU configuration assignments is given in Section 3. Figures 1-1, 1-2, and 1-3 show the different configurations.

1.2.3. Input/Output Unit

An 1100/80 Systems configuration includes at least one IOU. The IOU controls all transfers of data between the peripheral devices and main storage. Transfers are initiated by a CPU under program control. The IOU includes independent control paths to the CPU and data paths to main storage. The mode of I/O transmission is through either byte channels or word channels.

The IOU consists of two sections: a control section and a section containing three or four input/output channel modules. An IOU expansion allows up to four additional channel modules to be added to the IOU. The word I/O channel module provides four independent word I/O channel interfaces and occupies one channel module position. A second IOU with identical expansion capabilities can be added to each CPU cluster.

The control section includes all logic associated with the transfer of function, data, and status words between main storage and the subsystems. It also services I/O requests from either one or both of

the CPUs (in a multiprocessor system) and routes interrupts to one of the two CPUs. Interrupt routing may be specified by program.

Some capabilities of the IOU are:

- I/O transmission through byte channels or word channels
- Channel transfer rates of:
  - 1.67×10^6 bytes per second (maximum) on a block multiplexer channel;
  - 200×10^3 bytes per second (maximum) on a byte multiplexer channel; or
  - 500×10^3 words per second aggregate for a word channel module.
- Externally specified index (ESI) and internally specified index (ISI) transfer modes on the word channels.
- Channel buffering
- Interrupt tabling
- Parity generation/checking capability on all ISI channels and byte channels.

1.2.4. System Console

The system console provides the means for operator communications with the Executive System. The basic console consists of the following major components:

- The CRT/keyboard consists of a UNISCOPE 200 Display Terminal. The display format is 24 lines, with 64 characters per line. The 7-bit ASCII character set, consisting of 95 characters plus the space, is used. The keyboard provides all of the operator controls to use the UNISCOPE display terminal and communicate with the CPU.
- The incremental printer operates at 30 characters per second and provides a hard copy of console messages. (Five additional incremental printers may be connected to a console.)
- Maintenance interface for remote console operation by means of the system maintenance unit and a remote maintenance system.
- The fault indicator, located on the incremental printer, provides the operator with a visual indication of a fault condition in a major system component. The actual component and nature of the fault may then be determined from indicators on the operator/maintenance panel on the system transition unit.
- A standard byte multiplexer channel interface.

1.2.5. System Transition Unit (STU)

The STU contains the controls and indicators required for control and assignment of the system units. The functions controlled by the STU are:

- Partitioning

  The STU partitioning function provides the ability to assign the individual central complex units (CPUs, IOUs, SIU segments, and MSU banks) of a system to either one or two independent smaller systems (applications 0 or 1), or to isolate a unit from either application for offline concurrent maintenance.

  The partitioning function also indicates the operational status of each central complex unit. These status conditions are available to system software for configuration control.

  The ability to partition peripheral subsystems is provided by the subsystem availability unit (SAU), controls on the individual subsystems and, optionally, for some subsystems by software command. In addition, a form of partitioning is provided in that the STU has the capability of partitioning at the channel module level in the IOU. Each channel module may be placed offline or online by toggling a switch on the STU.

- Initial Load

  Initial load provides the ability to set module select register (MSR) values, select initial load paths, and initiate the initial load operation for either one of two applications. The MSR selects the section of main storage where the instruction execution sequence is initiated on an initial load.

- Automatic Recovery

  Automatic recovery provides the system, specified in Application 0 or 1, with an automatic system recovery capability. When automatic recovery is enabled, and the system software does not reset the automatic recovery timer within the preset time interval, the STU clears, reloads, and reinitiates the system. The system provides two recovery paths. If the two recovery paths are within the same application and the first recovery path fails, the alternate recovery path is automatically initiated. If only one recovery path is in the application, and the first recovery attempt fails, the same path is tried once more. If recovery again fails, an indicator on the STU panel is lit and no more recovery attempts are made. The recovery function provides for software resetting of the automatic recovery timer and for selection of the automatic recovery path to be used by the next recovery attempt (two paths in the application).

- Processor and Input/Output Unit Controls

  This panel provides the controls and indicators required for manual control of up to four CPUs, four IOUs, eight SIU segments, and eight MSU banks.

- Interface with SAU

  The interface with the SAU is used to convey IOU application information to the SAU for the purpose of word channel subsystems partitioning.

1.2.6. Subsystem Availability Unit (SAU)

The SAU provides the 1100/80 System with the ability to partition word subsystems and byte subsystem strings by means of an operator–controlled panel. Word subsystems and byte subsystem strings may also be placed offline. The SAU also provides subsystem partitioning status. Partitioning of major system components is still accomplished at the partitioning panel on the STU.

The SAU is the primary device used to control partitioning of peripheral subsystems within a multiprocessor system. The SAU contains the controls for partitioning word control units (CUs) by application. The CUs must have a UNIVAC Shared Peripheral Interface (SPI) for the SAU to exercise this control. The SAU allows the word subsystems with SPIs, to be assigned to Application 0, Application 1, or offline. The SAU provides a status word to the Operating System, through one word channel per IOU, which contains CU path partitioning selection information.

1.2.7. System Maintenance Unit (SMU)

The SMU provides for diagnostic checkout and fault isolation of the CPU, IOU, and SIU by the automatic comparison of logic status against known correct data. The SMU includes a maintenance processor, card tester, communications capability, UNISCOPE 200 Display Terminal, and peripherals.

1.2.8. Auxiliary Storage and Peripheral Subsystems

The 1100/80 Systems offer a full range of auxiliary storage and peripheral subsystems to provide the capability to satisfy many requirements. The standard SPERRY UNIVAC Subsystems include:

- SPERRY UNIVAC 8405/8430/8433/8434/8450/8470 Disk Subsystems
- SPERRY UNIVAC 8425 Disk Subsystem
- UNIVAC FH–432/FH–1782 Drum Subsystem
- UNISERVO 30 Group Magnetic Tape Subsystems
- UNISERVO 22/24 Magnetic Tape Subsystem
- UNISERVO 14 Magnetic Tape Subsystem
- UNISERVO 20 Magnetic Tape Subsystem
- UNIVAC 0716 Card Reader Subsystem
- UNIVAC 0604 Card Punch Subsystem
- UNIVAC 0768 Printer Subsystem
- UNIVAC 0770 Printer Subsystem
- SPERRY UNIVAC 0776 Printer Subsystem
- SPERRY UNIVAC Telcon System
- SPERRY UNIVAC General Communications Subsystem (GCS)

- SPERRY UNIVAC UTS 400 Display Terminal
- SPERRY UNIVAC UTS 400 Text Editor
- SPERRY UNIVAC BC/7 System
- UNISCOPE 200 Display Terminal
- UNISCOPE 100 Display Terminal
- UNIVAC DCT 500 Series Data Communications Terminals
- Remote SPERRY UNIVAC 90/30 Subsystem

1.2.9. Destandardized Subsystems

Some auxiliary storage and peripheral subsystems used on earlier model Series 1100 Systems can be configured, but Sperry Univac will not enhance any of the existing Series 1100 software for these destandardized subsystems.

1.2.10. Minimum Peripheral Complement

The following list of peripheral equipment is the minimum for the 1100/80 Systems. This minimum has been established to ensure an adequate complement for Sperry Univac customer engineering and software support.

	Minimum Complement	Alternate
1.	One 0716 Card Reader and one 0770 High Speed Printer Subsystem	One 0716 Card Reader and one 0776 or 0768 High Speed Printer Subsystem
2.	8450 Disk Subsystem with one control unit and two 8450 Disk Storage Units	8470 Disk Subsystem with one control unit and two 8470 Disk Storage Units; or 8430/8433/8434 Disk Subsystem with one control unit and two 8430, 8433, or 8434 Disk Storage Units; or 8425 Disk Subsystem with one control unit and two 8425 Disk Storage Units
3.	Magnetic tape subsystem with one control unit and four UNISERVO 30, 32, 34, or 36 Magnetic Tape Units	Magnetic tape subsystem with one control unit and four UNISERVO 22, 24 Magnetic Tape Units, or one control unit and four UNISERVO 20 Magnetic Tape Units; or one control unit and four UNISERVO 14 Magnetic Tape Units

2. Processing Unit

2.1. General

The 1100/80 Systems central processor unit (CPU) contains a control section, an arithmetic section, a maintenance section, a general register stack, and interfaces through which it is connected to other equipment. The IOU controls all data transfers between peripheral devices and storage. Transfers are initiated by a CPU under program control.

2.2. Control Section

The control section of the CPU interprets instructions and directs all processor operations except certain I/O operations. It is discussed briefly in this section and in more detail in 4.2.

2.2.1. Control Section Operation

The program instruction words are sequentially loaded into the control section. Each instruction word is interpreted by the control section which generates the signals necessary to perform the instruction. The instruction words are located in main storage and the data words (operands) are located either in main storage or in the addressable control registers which are part of the control section. The control section includes an address formation segment which generates the absolute main storage addresses to obtain the instruction words.

The instruction word is divided into fields. These fields specify to the control section the function to be performed, which portion of the operand is to be used, a control register, indexing, index register modification, indirect addressing, and an operand address.

2.2.2. Instruction Repertoire

The instruction repertoire includes fixed-point and floating-point arithmetic, logical functions, byte operations, block transfers, comparisons, tests, I/O control, and special purpose instructions. There are over 200 basic instructions in the repertoire. Partial-word data transfers and repetitive operations are included in the instruction repertoire. Indexing capability is provided with all instructions. Indirect addressing capability is also provided and is usable to any level with full indexing capability at each level.

Instructions such as data transfers, single-precision fixed-point adds, and certain logical functions, require less than 250 nanoseconds for complete execution. Indexing (18-bit) does not add to the execution time of an instruction. Details of the instruction repertoire are found in Section 5.

2.2.3. Control Registers

The 128 addressable control registers in the general register stack (GRS) of the control section are integrated–circuit registers. These control registers are addressed either explicitly or implicitly by the instructions. They fall into four categories: index registers, arithmetic registers, special registers, and unassigned registers.

The control registers are discussed in detail in Section 3.

2.2.4. Data Shift/Complement/Store Operation

The CPU includes circuitry which permits the various store instructions to bypass the arithmetic section. This circuitry includes the shifting capability needed for storing partial words in main storage, the sign testing capability needed for the Store Magnitude A instruction, and the complementing capability needed for the Store Negative A and Store Negative Magnitude A instructions.

2.3. Arithmetic Section

All arithmetic computation is microprogram controlled and is performed using the nonaddressable registers of the arithmetic section. These arithmetic processes can be performed in either fixed–point or floating–point mode. Fixed–point arithmetic instructions provide for single–precision, double–precision, half–word, and third–word addition and subtraction, and for fraction and integer multiplication and division. Floating–point instructions provide for both single–precision and double–precision operation. The arithmetic section also performs certain logical operations such as shifting and comparisons. The instruction word may be used to specify the transfer of any chosen portion of a word (half, third, quarter, or sixth) to the arithmetic section. The ability to transfer only the selected portion of a word minimizes the number of masking and shifting operations required.

A shift matrix in the arithmetic section permits the completion of an entire single word shift operation in one main storage cycle time. By use of the matrix, the shift operation can shift a single or double word operand in either direction up to 72 bit positions.

A scientific accelerator module (SAM) will enhance the performance of the following arithmetic instructions:

Add

Multiply (Fixed point)

Divide (Fixed point)

All floating–point instructions, both single and double precision

Details on the operation of the arithmetic section are found in 4.1.

2.4. Maintenance Section

The maintenance section performs all diagnostic tests using its own repertoire of commands. It operates only when the CPU is in maintenance mode. In this mode the processing system can be operating either online or offline. When online, the processing system and the maintenance system operate concurrently. In this case the maintenance system is connected to and operating on the byte bus, and the processing system operates normally, except that the processing operation is suspended

whenever the maintenance system needs to use the processor data and control paths for executing a maintenance function.

2.5. Input/Output Unit (IOU)

The IOU is a separate functional entity. I/O activity is initiated when the interpretation of certain instructions by the CPU causes signals to be sent to the IOU. Once an I/O operation is initiated, the IOU and the subsystem control the input and output transfers. The IOU operates with a wide variety of peripheral devices, and it requires minimal attention from the CPU.

Once an I/O operation is initiated by the program, I/O activity is independent of program control. The I/O data flows between main storage and the peripheral subsystem through an I/O channel. Each I/O channel consists of 36 input data lines, 2 input parity lines, 36 output data lines, 2 output parity lines, and various control signal lines. All data word bits are transmitted in parallel to or from the subsystem.

The IOU has five interfaces: a storage interface, a processor interface, a control unit–peripheral interface, a system transition unit interface, and a maintenance interface.

Details of the IOU are presented in Section 6.

2.6. System Status Word

The system status word, which originates in the system transistion unit (STU), indicates to the system software the partitioning status of each unit and also system status. This status information is transferred to each storage interface unit (SIU) segment where it can be read by a CPU through a special command code. The status information is not interpreted by the SIU. The power status in each unit will be indicated on the partitioning panel of the STU, and loss of dc power in a unit will force the status of that unit to be offline.

The system status word is divided into two 36–bit words (System Status Word 0 and System Status Word 1). Figure 2–1 shows how the status words are divided into fields. Each word contains two fields of data, one for application information and the other for load path information. Each system component has two application bits that are translated as follows:

Application 0	Application 1	
0	0	= Offline
0	1	= Application 0
1	0	= Application 1
1	1	= No System Transition Unit

System Status Word 0

	R/TM	NOT USED	LOAD PATH 0						APPLICATION 0																			
									AUTO			MAINT	MSU								SIU				IOU				PROC			
			FAULT	CLUS	SIU	IOU	PROC		NLP	PH1	PH0		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	3	2	1	0	3	2	1	0
**	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

System Status Word 1

	NOT USED	LOAD PATH 1						APPLICATION 1																			
								AUTO			MAINT	MSU								SIU				IOU				PROC			
		FAULT	CLUS	SIU	IOU	PROC		NLP	PH1	PH0		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	3	2	1	0	3	2	1	0
**	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

MAINT = Maintenance Mode 1 = One or more system components in that application are in maintenance mode.

R/T M = Real Time Mode 1 = Real Time Mode

AUTO = Auto Recovery

NLP = 1 = Load Path 1 next if both load paths in the same word are in auto recovery mode.
PH1 = 1 = Load Path 1 is enabled as an auto recovery load path for that application.
PH0 = 1 = Load Path 0 is enabled as an auto recovery load path for that application.

LOAD PATH

FAULT = 1 = Indicates that load path has failed.
*CLUS = 1 = Indicates that the following three components are in Cluster 1.
*SIU = 1 = Indicates SIU upper is in the load path.
*IOU = 1 = Indicates the odd IOU is in the load path.
*PROC = 1 = Indicates the odd PROC is in the load path.

NOTES:

* *These logic levels result after a signal inversion at the SIU.*
** *The bit position of the status word as sent by the SIU segment to the CPU.*

Figure 2–1. System Status Word Format

3. Storage System

3.1. General

The storage system comprises up to 4,194,304 words of main storage, up to 32,768 words of high-speed buffer storage, and 128 addressable control registers.

Main storage consists of one to four main storage units, depending on user requirements. Main storage provides the storage for the instruction and data words. The high-speed buffer storage consists of one or two storage interface units, depending on user requirements. The high speed buffer storage provides accelerated response between main storage and the central processor units, and between main storage and the input/output units. The 128 addressable control registers are located in the control section of each central processor unit. These registers provide fast access storage for data and control words.

The storage interface unit (SIU) high-speed storage buffers are grouped in sets of four blocks of four words each. A set is addressed directly by selected bits of a 24-bit absolute address. The remaining bits are used to make a comparison with the 4 block addresses to determine if the required block, and hence word, is present in the set. When a word is required from a block of data not resident in the SIU buffer, the SIU makes a request to main storage and the new block of data is loaded into the buffer.

3.2. Main Storage Unit

The main storage unit (MSU) consists of two independent banks. Each bank contains 262K 43-bit words consisting of 36 data bits and 7 error correction code (ECC) bits. Each bank is expandable to a maximum of 524K words. The MSU bank is accessible via a two port multi-module access (MMA) module.

The MSU contains an internal exerciser for offline operation in the maintenance mode. The exerciser uses much of the logic contained in the interface section of the MSU, and is capable of operating in an offline mode with either bank while the other bank is online. The maintenance mode is switch selectable at the MSU. Online availability is indicated to the SIU by the presence of the MSU AVAILABLE signal.

The MSU is capable of performing read, write, and partial write operations. Data is transferred from the MSU on a read operation and to the MSU on a write or partial write operation. A description of the operations performed within the MSU follows:

- Read – The MSU reads a full block of data consisting of four contiguous words (172 bits, 144 data and 28 ECC bits) from storage and transfers the information to the SIU via a 4–word interface in one data transfer.

- Write – The MSU writes one word (36 data and 7 ECC bits) of new information into the specified storage location.

- Partial Write – The MSU reads a data block (four contiguous words) from storage. Only that portion of the addressed word (one of the four words) which is enabled by the write control lines is modified with new write data. A new ECC is generated for the combined read and new write data. Then both the new word and the ECC bits are stored.

- Refresh – The MSU is a volatile storage device and requires a refresh cycle to maintain the data. Each MSU bank generates its own refresh cycle. Refresh signals are generated at 24 microsecond intervals. The refresh cycle is continually sequenced through the storage bank such that 1/128 of the bank is refreshed during each refresh cycle.

3.2.1. Write Data Error Detection

The requester (the SIU) sends 36 bits of write data plus a 7–bit ECC to the MSU. The MSU generates a 7–bit ECC from the 36 data bits received and then compares that code with the 7–bit code received. If a single bit error or multiple error is detected, the reference is executed, the special ECC is stored in the addressed location to indicate corrupt data on future requests to that location, and a WRITE DATA CHECK signal is sent to the requester. All 144 read data lines and all 28 ECC lines from the MSU to the requester remain inactive during a write cycle.

3.2.2. Partial Write Error Detection

On a partial write request, the MSU retrieves the data word to be modified from storage and compares the 7–bit ECC retrieved from storage with the 7–bit ECC generated from the read data by the MSU. If no errors are detected on the data word retrieved from storage or on the write data to be written, the word is modified and a new ECC is stored. If errors are detected on the data during a partial write, the MSU responds as follows:

- If a single bit error is detected on the data word retrieved from storage, the data bit in error is corrected, the word modified, and a new ECC is stored.

- If a multiple bit uncorrectable error or the special ECC is detected on the data word retrieved from storage, the word is not modified and the stored ECC is retained.

- If the MSU detects a single bit error or a multiple bit uncorrectable error on write data, the data word is modified and the special ECC is stored in the ECC bits. The MSU sends a WRITE DATA CHECK signal to the requester.

- If the MSU detects both a single bit error on the data word retrieved from storage and a single bit error or multiple bit uncorrectable error on the write data, the word is modified and the special ECC stored in the ECC bits. The MSU sends a WRITE DATA CHECK signal to the requester.

- If the MSU detects both a multiple bit uncorrectable error or the special ECC on the data word retrieved from storage and a single bit error or multiple bit uncorrectable error on the write data to be written, the word is not modified and the ECC is retained as stored. The MSU sends a WRITE DATA CHECK signal to the requester.

3.2.3. ECC Write Check Disable

When the ECC WRITE CHECK DISABLE signal is activated by the requester, the MSU writes the data and the ECC as received for either a full-word write or a partial write. The WRITE DATA CHECK signal is not transferred and the special ECC is not stored in the ECC bits if a write data error is detected. The special ECC is stored, however, if the MSU detects a write-control parity error. When the ECC write-check disable operation is performed, the stored data must be restored to correct the ECC by either software or by the MSU offline exerciser before returning the bank to an online condition.

3.2.4. Write Control Parity Checking

The requester sends nine write-control plus one write-control parity signals. The MSU checks for odd parity. If a write-control parity error is detected, the MSU performs a partial write function and the special ECC is stored in the location to indicate corrupt data on future requests to that location. The MSU sends a WRITE CONTROL CHECK signal to the requester.

3.2.5. Address Parity Checking

The requester sends 21 address bits plus one address parity bit. The MSU checks for odd parity. If the MSU detects an address parity error, the MSU responds as follows:

- If the request is for a read, the data block is transferred and the MSU sends an ADDRESS CHECK signal to the requester. Future requests are accepted by the MSU.
- If the request is for a write, the word is stored with the new ECC bits at the location selected. The MSU notifies the requester via the ADDRESS CHECK signal and the system transition unit (STU) via the write address check line. The MSU does not accept further requests to the bank in which the address parity error was detected until any requester partitioned to that bank sends a system CLEAR signal. The WRITE ADDRESS CHECK signal sent to the STU is maintained by the MSU until cleared manually at the MSU.
- If the request is for a partial write, the data word selected is modified and a new ECC is stored. The MSU notifies the requester via the ADDRESS CHECK signal and the STU via the write address check line. The MSU does not accept further requests to the bank in which the address parity error was detected until a requester partitioned to that bank sends a system CLEAR signal. The WRITE ADDRESS CHECK signal sent to the STU is maintained by the MSU until cleared manually at the MSU.

3.2.6. Refresh Fault

If a refresh fault is detected, the MSU completes the cycle upon which it is operating. Upon completion of the cycle, the MSU stops and no further requests are honored. A signal is sent to the STU to indicate an error; also, the error is indicated at the MSU. To clear a refresh fault from a bank of storage requires one of the following:

- powering down that bank and powering back up,
- going from an offline condition to an online condition at the STU, or
- performing a check reset at the MSU maintenance panel while the bank is offline.

3.2.7. Fixed Address Assignments

The interrupt subroutine entrances and certain status words are assigned fixed locations in main storage as shown in Tables 3–1 and 3–2. The listed addresses are relative to the contents of the 7–bit module select register (MSR) and the position of the MSR ACTIVE UPPER/LOWER switch. MSR may be manually loaded by setting the desired combination of the seven MSR switches and the MSR ACTIVE UPPER/LOWER switch on the STU partitioning panel. When an initial load operation is performed, the value in the MSR identifies the main storage area in which the incoming data is to be stored. During an externally specified index input/output (ESI–I/O) operation the value in the MSR identifies the high order bits of the address of the main storage locations from which the ESI access control words and chain pointer words are obtained.

Table 3–1. Fixed Address Assignments 0200–0237

Octal	Decimal	Assignment
200	128	Reserved for Hardware Default
201	129	Unassigned
202	130	Even Number I/O Interrupt Error
203	131	Odd Number I/O Interrupt Error
204	132	I/O Normal Status Interrupt
205	133	I/O Tabled Status Interrupt
206	134	I/O Machine Check Interrupt
207	135	Unassigned
210	136	Quantum Timer Interrupt
211	137	Real Time Clock Interrupt
212	138	Unassigned
213	139	Unassigned
214	140	Unassigned
215	141	Unassigned
216	142	Dayclock Value
217	143	Dayclock Interrupt
220	144	Immediate Storage Check Interrupt
221	145	Invalid Instruction
222	146	Executive Request Interrupt
223	147	Guard Mode Interrupt
224	148	Test and Set Interrupt
225	149	Characteristic Underflow Interrupt
226	150	Characteristic Overflow Interrupt
227	151	Divide Check Interrupt
230	152	Addressing Exception Interrupt
231	153	Breakpoint Interrupt
232	154	Interprocessor Interrupt
233	155	Power Check Interrupt
234	156	Delayed Storage Check Interrupt
235	157	Jump History Stack Interrupt
236	158	Emulation Interrupt
237	159	Unassigned

NOTE:

All fixed addresses are relative to the MSR.

Table 3-2. Fixed Address Assignments 0240-0277

Octal	Decimal	Assignment
240	160	Processor 0 Channel Address Word 0
241	161	Processor 0 Channel Address Word 1
242	162	Unassigned
243	163	Unassigned
244	164	Processor 1 Channel Address Word 0
245	165	Processor 1 Channel Address Word 1
246	166	Unassigned
247	167	Unassigned
250	168	Processor 2 Channel Address Word 0
251	169	Processor 2 Channel Address Word 1
252	170	Unassigned
253	171	Unassigned
254	172	Processor 3 Channel Address Word 0
255	173	Processor 3 Channel Address Word 1
256	174	Unassigned
257	175	Unassigned
260	176	Processor 0 Interrupt Address Word
261	177	Processor 0 Channel Status Word 0
262	178	Processor 0 Channel Status Word 1
263	179	Processor 0 Channel Status Word 2
264	180	Processor 1 Interrupt Address Word
265	181	Processor 1 Channel Status Word 0
266	182	Processor 1 Channel Status Word 1
267	183	Processor 1 Channel Status Word 2
270	184	Processor 2 Interrupt Address Word
271	185	Processor 2 Channel Status Word 0
272	186	Processor 2 Channel Status Word 1
273	187	Processor 2 Channel Status Word 2
274	188	Processor 3 Interrupt Address Word
275	189	Processor 3 Channel Status Word 0
276	190	Processor 3 Channel Status Word 1
277	191	Processor 3 Channel Status Word 2

NOTE:

All fixed addresses are relative to the MSR.

3.3. Storage Interface Unit

The storage interface unit (SIU) is composed of two identical and independent halves, an upper half and a lower half. Each SIU half consists of one or two high-speed buffer segments of 4,096 36-bit words each. Each SIU half is capable of addressing 2 million 36-bit words in two MSUs. Each SIU segment interfaces with at least one MSU bank, but not more than two MSU banks. The MSU interfaces with at least one SIU segment, but not more than two SIU segments.

The SIU high-speed buffer storage reduces the overall storage access time by automatically allowing the majority of storage references to be fetched from the SIU rather than the MSU. Storage requests from the central processor unit (CPU) and the input/output unit (IOU) are usually satisfied from data resident in the SIU buffer. These requests are categorized as "hits" and the storage access time is that of the buffer. A request for nonresident data generates a "miss" and requires the SIU to make a reference to main storage. The reference to main storage transfers a 4-word block of data into

the SIU from the MSU when a word from that block is read by the CPU or IOU. The block becomes one of many blocks remaining resident in the SIU until being displaced by a more current block.

3.3.1. Functional Characteristics

The SIU employs a set associative buffer. The 4K minimum buffer is organized into 256 sets, each set consisting of four blocks with four data words per block. The main storage is also divided into 256 sets. If the main storage contains 524K words, each set in main storage would contain 512 blocks of four words. For a particular set, any one of the blocks of four words in the main storage may be placed in any one of the four blocks in the corresponding SIU set. For a read operation, a 4-word block of data on fixed 4-word boundaries is transferred from main storage to the SIU during a single main storage cycle. For a write operation, one word of data is transferred from the SIU to the main storage during a single main storage cycle.

The storage requests made by the CPUs and IOUs are seen only by the SIU. Direct address selection is used to address the one of 256 sets in which the required word is located. This causes the SIU to simultaneously read the address of each of the four blocks currently resident in the set and to compare each with the requested block address. If one of the four block addresses matches the requested address, the appropriate word is read from that block and presented to the requester or written into the appropriate word of the matched block. If none of the four block addresses match on a read operation, a request is made by the SIU to main storage for the entire block which contains the required word. A request is always made by the SIU to main storage on a write operation. When read data arrives from main storage, it is stored in the least recently used block location as determined by the SIU, and the appropriate word is sent by the SIU to the original requester to complete the cycle. Each time a match reference (hit) is made in the SIU, the SIU reorders the block ages in the set by making the referenced block age 0 (most recently used) and the remaining three blocks one age older, unless an unmatched block was older than the matched block. Priority not only depends on history between IOUs and between CPUs (rotational), it also depends on first in/first out between ports for a CPU.

Each SIU segment uses an 8-port MMA with pre-emptive priority on ports 4 through 7 and non pre-emptive priority on ports 0 through 3. Within a given priority type, rotational priority exists. Ports 4 and 5 are IOU ports; ports 0 through 3 are CPU ports; and port 6 is reserved for future expansion. Port 7 is used for communications between two SIUs in a two cluster system. Specifically, that port is used by a segment which is altering a word within a given main storage address range to notify the corresponding other segment using that address range that the copy of that word, if resident in the other segment, is no longer valid and the validity bit for the corresponding block should be cleared. Then, if a subsequent request for a word in that invalidated block is made, the SIU segment must replace the invalidated copy with fresh valid data from the main storage.

The interface between the SIU and main storage provides a 4-word-wide read transfer on a single request and a 1-word-wide write transfer. When a word is required from a block of data not resident in the SIU, the request is made to main storage and the segment is free to handle other requests. On a read request, all four words and error correction code (ECC) bits are transmitted from the main storage to the requesting SIU segment intact. The requesting SIU segment will then correct any single bit errors and detect any double bit errors with indications of the uncorrectable error provided to the SIU segments requester. The ECC bits are not stored in the SIU, but parity bits are generated and stored. The ECC error syndrome bits (code bits produced by decoding the comparison of stored ECC bits and the ECC generated from the stored data) are sent to the CPUs via storage check interrupts.

When a new block is brought in from main storage and the SIU segment is full, an existing block in the corresponding segment must be displaced. The displacement algorithm used is least recently used (LRU). The storing of full- or half-word modified data is accomplished by using a store-through algorithm which modifies the word plus parity in the SIU, generates the ECC for the half or full word,

and initiates a write to main storage (36 data bits plus 7 ECC bits). The storing of a partial-word (excluding half-word) modified data uses a store-past algorithm which invalidates the word in the buffer, generates the ECC for the partial word received from the requester, and initiates a partial write to main storage. In both cases, full- and partial-word writes, the segment must send a message (systems with two clusters) to the corresponding segment of the other cluster to inform it that the copy, if resident, is no longer valid. Requester writes (except writes with segment lock) are early acknowledged to allow the requester to continue on without waiting for the actual write to take place.

On write data requests, only one word (36-data bits and 7 ECC bits) is transmitted from the SIU to main storage. Full- and half-word write requests are write-throughs (written in the SIU and main storage), and partial-word write requests, other than half-word writes, are write-past (invalidated in the SIU and written in main storage). On full- and half-word writes, the addressed word is updated if the address for that request is resident in the SIU. For partial- word writes, other than half-word writes, the addressed word is invalidated if the address for that request is resident in the SIU. For partial-word writes, a data block comprising four contiguous words is read from storage. Only the portion of the addressed word (one of the four) that is enabled is modified with new write data. A new ECC is generated for the combined read and new write data. Both the new word and the ECC bits are stored in main storage.

The minimum SIU consists of 4096 words. This unit interfaces with up to 1048K words of main storage. An SIU may be expanded by adding one to three more logically independent 4K word segments (for a maximum of 16K words) which may interface to an additional 3146K words of main storage. With the addition of CPUs and IOUs to configure greater than a 2x2, more 4K word segments must be added to the system which will provide dual access with the other SIU to the main storage. Thus, a maximum system can contain 32,768 words of SIU storage, 16,384 words of which are dedicated to each 2x2 cluster. The maximum main storage is 4194K words which can be accessed by either of the SIUs. Each 2x2 cluster must have an equal amount of SIU storage.

The SIU allows read-by/write-by degraded operation. The SIU array failures at a 4-word block level are reported and dynamically degraded by the hardware. Software control for degradation of the SIU at a 4-word block level is also available. This is a fault tolerant capability which allows an SIU segment to continue operations even though it has interrupt storage faults.

3.3.2. Tag and Data Buffer

The SIU segments have storage capacity for a data buffer of 4096 words (38 bits). A parity bit is stored with each 18-bit half word. The 4096 data storage words are handled as 1024 four-word blocks. There are four blocks per set and 256 sets per segment. Each data set is associated with a specific portion of main storage. For the maximum addressing range there are 4096 blocks associated with each set. At any one time only four of the 4096 blocks associated with a set can be resident in the SIU data buffer. A request to a word in one of these four resident sets results in a "hit."

In addition to data storage, each SIU segment has a tag buffer. The tag buffer has storage capacity for 256 double words (72 bits). Eighteen bits of a tag buffer word define the block address, invalid bit, degrade bit, and the two age bits for one block. There is a direct relationship between a data buffer address and a tag buffer address. The set address, requester address bits 10-03, references all of the tag buffer and the upper eight bits of the data buffer. The lower two bits of the data buffer address come from the requester address bits 01-00. This is a total of 10 address bits for the data buffer. Ten address bits can address a maximum of 1024 words, but the data buffer contains 4096 words. The data buffer, however, has a 4-word interface. Therefore, each of the 1024 addresses reference a 4-word block, one word from each of the four blocks represented by the set address.

The word address, bits 01-00 of the requester address, determines which of the four words of each block is presented at the data buffer interface.

The SIU uses the write through algorithm for store requests. This means that all store requests to the SIU will initiate a store request to the MSU via the main store interface stack (see 3.3.3). The SIU will replace resident data blocks based on the least recently used (LRU) algorithm.

3.3.3. Main Store Interface Stack

If the SIU receives a read or write request for nonresident data, the request will be placed on the SIU/MSU interface. The response time of the MSU to a request from the SIU is approximately 700 nanoseconds. If the SIU waited in an idle state until the request to the MSU was completed, the SIU would waste time which might otherwise be used to service other requests. The main store interface (MSI) stack frees the SIU to service other requests during the MSU response time.

The MSI stack is four words deep and 86 bits wide. Request data for the requests that must be presented to the MSU are transferred from the requester interface to the MSI stack by the SIU. Control is then directed to the requester interface while the MSU is responding to the request in the MSI stack. In the event that other requests are present at the requester interface, the SIU will accept the highest priority request and service it. These other requests may require that a request be made to the MSU. These requests will be loaded into the MSI stack to wait for completion of the request to the MSU that is at the top of the MSI stack. Up to three such requests will be accepted by the MSI stack, at which time the requester interface will be disabled until a response is received from the MSU.

3.3.4. Invalidate Interface

In systems configured in two clusters, the common point for storage references made by the two clusters is at the MSU. This means that any write request from either cluster will result in a change of data only at the MSU. If that data were resident in the other SIU, it will no longer be valid, having been superseded by the write. The invalidate interface insures the validity of data in the system. The invalidate interface is between SIU segment pairs, such as 0 and 1, or 2 and 3, etc. The invalidate interface invalidates data which may have been resident in the other segment of a segment pair. Segment pairs are those segments which interface to the same MSU address range. There are two invalidate interfaces per segment pair. Write requests that go through segment 0 must invalidate data resident in the same MSU address in segment 1, and write requests that go through segment 1 must invalidate data resident in the same MSU address in segment 0. A two–word–deep 25–bit–wide invalidate stack is associated with each invalidate interface.

3.3.5. Error Detection and Reporting

The SIU error detection and reporting scheme helps detect and isolate failing components. Detected faults are immediately isolated and are not propagated through the SIU–MSU–requester complex. When an error is detected, the error information is stored in one of four registers reserved for this purpose and a storage check interrupt status request (SCISR) is generated. The error information is then passed via the read data bus to the processor that acknowledges the interrupt request.

SCISR has a low prioity at the processor. The SCISR stack is four words deep and 37 bits wide. The SCISR stack will hold up to four status words, 36 data (status) bits, and one occupied bit for transmission to a processor. Status words will be placed on the requester interface in the same order as they are entered into the SCISR stack. The status word at the top of the SCISR stack will be placed on the requester read data lines if the requester interface operator (storage check status reference) is true during a request to the SIU.

3.3.6. Storage Interleave

Storage interleave is the alternating of requests to storage. The 1100/80 Systems have two levels of storage interleave. This interleave depends on the configuration assignments of the SIU segments and the MSU banks (see Appendix E). The two levels of interleave operate independently of each other. A particular system may have both levels of interleave, either one of the two levels of interleave, or no level of interleave.

The first level of interleave exists when the two segments in an SIU half are in the same application. When this condition exists, the requests from CPUs and IOUs alternate to the segments, depending on the setting of address bit 2. When address bit 2 is not set, the request goes to the lower numbered segment in the SIU half. When address bit 2 is set, the request goes to the higher numbered segment in the SIU half. This means that adjacent 4-word blocks are handled by alternate SIU segments. When the two segments in an SIU half are not in the same application, there is no storage interleave. Figure 3-1 shows first level configurations for interleave and non-interleave applications.

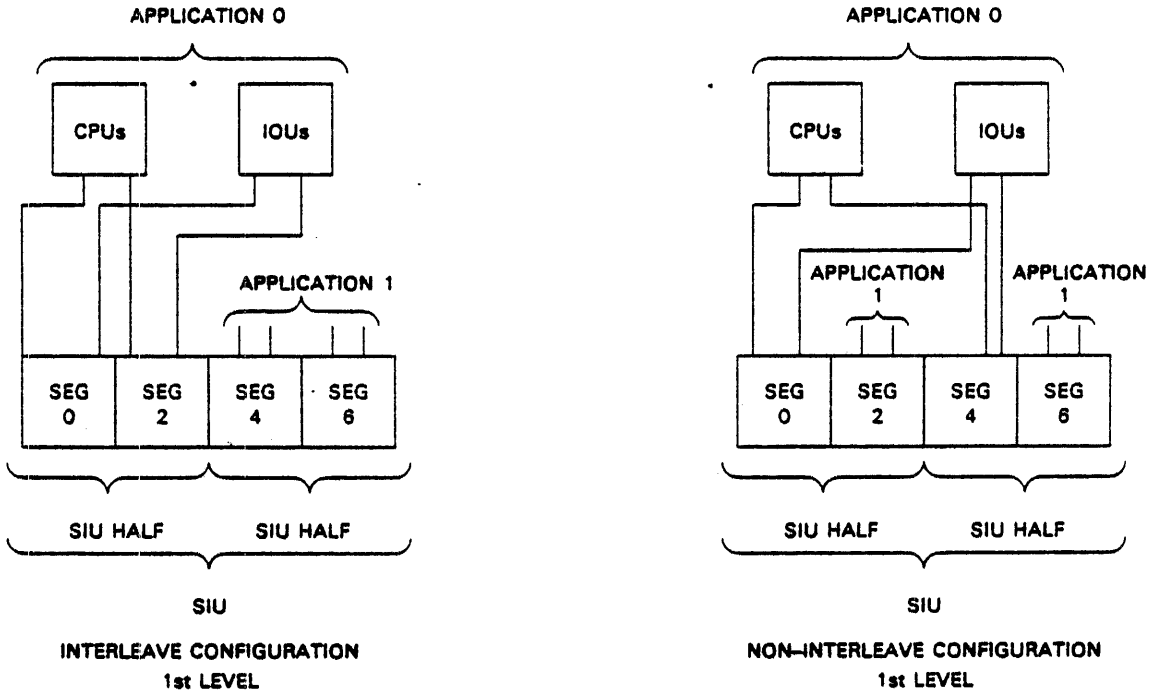


Figure 3-1. First Level Storage Interleave

The second level of interleave exists when the two storage banks in an MSU are in the same application. When this condition exists, the SIU segment directs requests to the banks, depending on the setting of address bit 3. When bit 3 is not set, the request is directed to the lower (even numbered) bank in the MSU. When bit 3 is set, the request is directed to the higher (odd numbered) bank in the MSU. Figure 3-2 shows configurations of second level interleave and non-interleave applications. When the two banks of an MSU are not in the same application, there is no storage interleave.

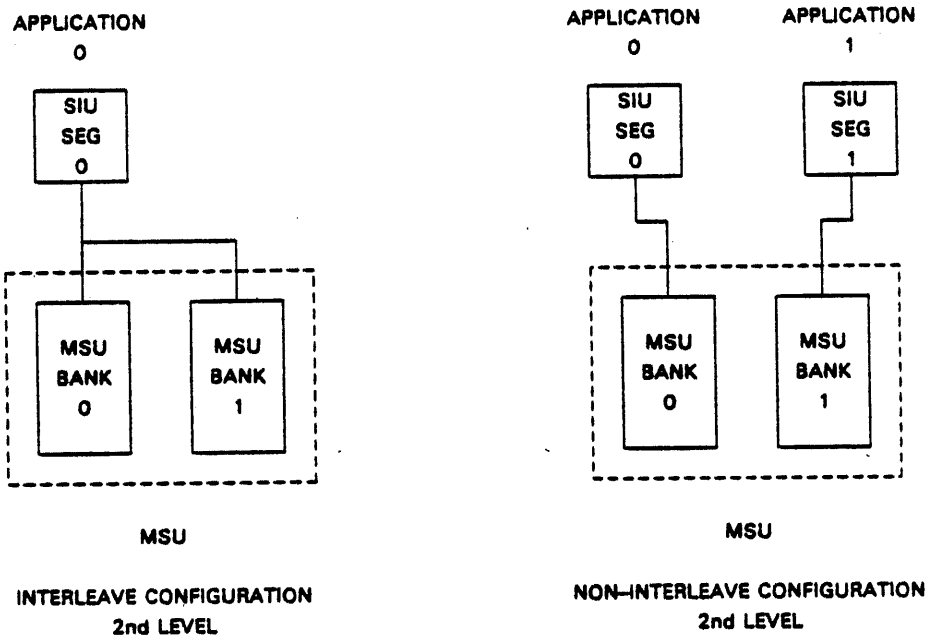
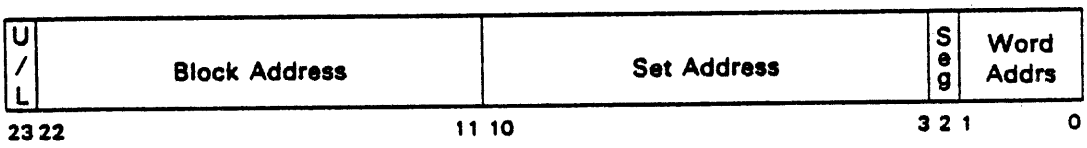


Figure 3-2. Second Level Storage Interleave

3.3.6.1. Addressing Modes

The interleave of addressing as described takes place above address 8,388K or below address 8,388K or both above and below. Address bit 23 is used by the requesters to direct requests to the MSU above or below address 8,388K. When bit 23 is not set, the requests go to the lower SIU half; when bit 23 is set, the requests are directed to the upper SIU half, which is the upper address range. The requester address format is as follows:



Bit Position	Description
23	Selects upper or lower SIU half
22 thru 11	Selects a 4-word block address
10 thru 3	Selects 1 of 256 sets
2	Selects SIU segment
0,1	Determines which of the four words of each block is presented at the data buffer interface.

Four addressing modes are possible in the address ranges above and below address 8,388K. The modes are determined by the levels of interleave within the address ranges. Table 3-3 and Figure 3-3 define the four modes.

Table 3-3. System Addressing Modes

Addressing Mode*	Level of Interleave		SIU Segments Per SIU Half	MSU Banks Per SIU Segment
	1st Level	2nd Level		
1**	Yes	Yes	2	2
2**	No	Yes	1	2
3	Yes	No	2	1
4	No	No	1	1

* These modes depend on the configuration as partitioned and running at any given time.

** Not available when configured in the limited "segment/bank" configuration.

Addresses to an SIU half are interleaved across the four MSUs in blocks of four words. Addresses $0–3_8$ are located in MSU 0; $4–7_8$ in MSU 2; $10–13_8$ in MSU 1; and $14–17_8$ in MSU 3. Therefore, SIU segment 0 will handle addresses $0–3_8$ and $10–13_8$; SIU segment 2 will handle addresses $4–7_8$ and $14–17_8$.

For the maximum configuration, requester address bit 2 is the SIU segment select bit, and bit 3 is the MSU select bit. Due to this preselection, bits 2 and 3 are not used as part of the MSU address. SIU halves, however, may be configured with other SIU/MSU combinations.

Manipulation of the requester address bits sent to the MSU is based on the addressing mode. Tables 3-4 and 3-5 show where the different requester address bits (particularly bits 2 and 3) are inserted in the MSU address for each addressing mode.

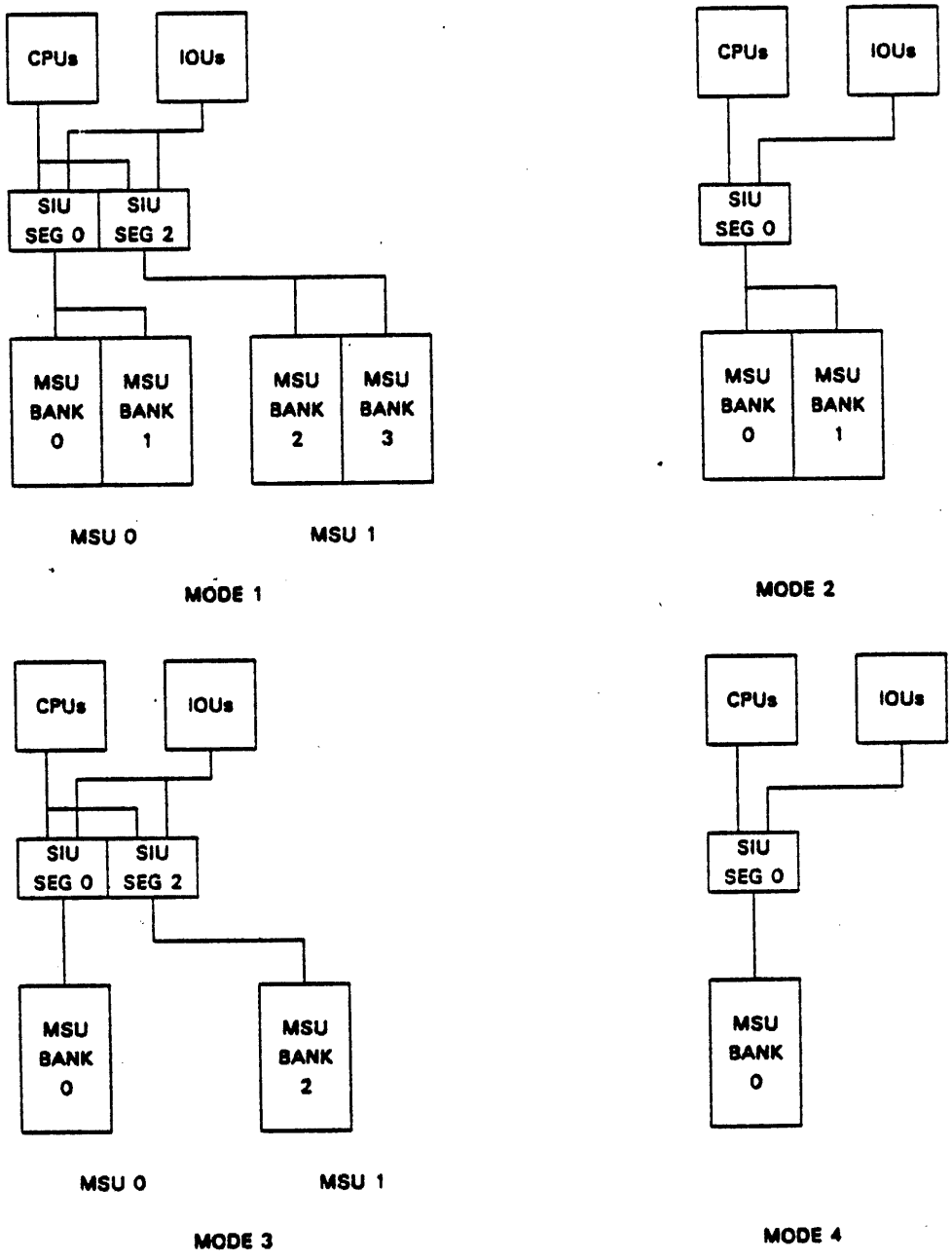


Figure 3-3. Configurations for Addressing Modes

Table 3–4. MSU Address Generation

Requester Address Bit	X SIUs X MSUs	MSU Address Bits			
		Mode 1	Mode 2	Mode 3	Mode 4
23*		X	X	X	X
22		20	X	X	X
21		19	20	20	X
20		18	19	19	20
19		17	18	18	19
18		16	17	16	18
17	15				
16	14				
15	13				
14	12				
13	11				
12	10				
11	09				
10	08				
09	07				
08	06				
07	05				
06	04				
05	03				
04	02				
03		X	X	17	17
02		X	16	X	16
01	01				
00	00				

NOTES:

In this table, the requester address bits are listed in the column on the left. The MSU address bits that use each requester address bit for each of the four addressing modes (see Table 3–3) are shown in the columns on the right. The MSU address bits shown in the column under "X SIU's X MSU's" use a particular requester address bit independent of the addressing modes. An "X" in a column after a requester address bit indicates that the requester address bit is not used in formulating the MSU address in the addressing mode.

** When bit 23 is not set, MSU address bits 17–20 are complemented in the SIU.*

Table 3-5. SIU/MSU Address Bit Manipulation

MSU Address Bit	X SIUs X MSUs	Requester Address Bit			
		Mode 1	Mode 2	Mode 3	Mode 4
20*		22	21	21	20
19*		21	20	20	19
18*		20	19	19	18
17*		19	18	03	03
16		18	02	18	02
15	17				
14	16				
13	15				
12	14				
11	13				
10	12				
09	11				
08	10				
07	09				
06	08				
05	07				
04	06				
03	05				
02	04				
01	01				
00	00				

NOTES:

In this table, the MSU address bits are listed in the column on the left. The requester address bits that are used to create the MSU address for each of the four addressing modes (see Table 3-3) are shown in the columns on the right. The requester address bits shown in the column under "X SIUs X MSUs" are transferred to a particular MSU address bit independent of the addressing mode.

** Bits 17-20 will be complemented when using SIU lower.*

3.3.6.2. Partitioning of Storage Configurations

The SIUs are partitioned on a segment level. A segment can be assigned to either application 0, application 1, or offline. When an application resides in two clusters, the assignment of segments to that application must be the same in both SIUs. If the first two lower numbered segments in SIU 0 are in application 0, then the first two segments in SIU 1 must be in application 0. For an application which resides only in one cluster, segments are assigned to it from only one SIU.

Main storage is partitionable on an MSU bank level. The MSU banks can be assigned to either application 0, application 1, or offline. The changing of assignment of any given MSU bank can have an effect on the other banks associated with that half of main storage because of the interleaving of addressing which can exist between banks. If an MSU bank is added to an application to which another bank in the same half of storage is assigned, interleave of addresses in that SIU half changes. If an MSU bank is removed from an application to which another bank in the same half of storage is assigned, interleave of addresses in that SIU half changes. To make the change in assignment, it is necessary to temporarily bring the SIU half offline, make the desired change, then bring the SIU half back online. The procedures for partitioning are defined in 1100/80 Systems, 4x4 Capability, Central Group, Operator Reference, UP-8599 (current version).

3.3.7. Storage Configurations

The 1100/80 Systems have two types of configuration assignments for the SIU segments and MSU banks: segment/bank configuration applicable to single cluster systems only, and segment/cabinet configuration applicable to single and double cluster systems. (Refer to Appendix E for storage configurations.)

The segment/bank configuration consists of unique cabling between the SIU and MSUs and allows for configuring with fewer MSU cabinets for a given amount of SIU segments. The segment/bank configuration is restricted to single cluster systems. The segment/bank configuration can be converted to a segment/cabinet configuration through a cabling change.

Segment/bank configurations, because of the MSU/SIU relationship, limit the main storage to a maximum of two cabinets. Segment/cabinet configurations have a different MSU/SIU relationship and therefore allow up to four MSU cabinets in a system. The MSU/SIU relationship for the segment/bank configuration is one MSU bank interfacing with one SIU segment. The segment/cabinet configuration is one MSU cabinet interfacing with one SIU segment. The relationship is controlled by cabling options which are possible between the MSUs and SIUs.

The directing of storage requests in the segment/bank configuration is essentially the same as for the segment/cabinet configuration. The directing of requests to upper and lower SIU storage is based on the condition of address bit 23, and the directing of requests between SIU segments within a half is based on address bit 2. This is the same as in the segment/cabinet configurations.

The directing of requests between the SIUs and MSUs is different. In the segment/cabinet configuration, requests are directed to both banks, and bit 3 determines which bank responds. In the segment/bank configuration, requests from the SIU segments to MSUs are directed to one MSU bank only. In the segment/bank configuration, bit 3 is not used to direct requests to MSU banks, since each SIU segment directs the requests to just one MSU bank.

One situation exists for the segment/bank configuration where address bit 3 is used to direct requests to both banks in a MSU cabinet. The situation is a degraded mode operation in which one SIU segment is unavailable to the system. In this case, the companion segment in the SIU half directs requests to both banks in the MSU cabinet, alternating between the two banks based on the condition of bit 3. This is performed the same way in the segment/cabinet configurations.

3.4. Control Storage

The control section of the CPU includes a general register stack (GRS) comprising 128 addressable control registers that can be independently referenced in parallel with main storage. Each control register stores a word consisting of 36 information bits. The control registers are addressable by the a- and x-fields of the instruction word, and by the value U developed in the index subsection of the CPU's control section. The details of control register addressing are explained in Section 4. Tables 3-6 and 3-7 summarize the control register address assignments.

3.4.1. Control Register Selection Designator

The 128 addressable control registers include one set of registers for use by the user program, and another set for use by the Executive program. The general register selection designator (D6) in the designator register defines which set of registers is addressed by the a- and x-designators of an instruction word. When D6 = 0, the user program set of control registers is addressed; when D6 = 1, the Executive program set of control registers is addressed. The contents of D6 has no effect on the choice of a control register for any particular value of U.

3.4.2. Control Register Address Assignments

Operand addresses 0_8 through 177_8 are assigned to the control registers. The following paragraphs define the various uses and related address assignments for the control registers.

3.4.2.1. Storage for MSR Value – 0143

During initial load of the system the value in the module select register (MSR) is loaded into GRS by the hardware. This one time load makes the MSR value available for referencing by software.

Table 3-6. GRS Register Assignments 0 Through 63

Octal	Decimal	Register Assignment
0000	0	Unassigned
0001	1	User X1
|	|	
0011	9	User X9
0012	10	User X10
0013	11	User X11
0014	12	User X12/A0
0015	13	User X13/A1
0016	14	User X14/A2
0017	15	User X15/A3
0020	16	User A4
0021	17	User A5
|	|	
0033	27	User A15
0034	28	Unassigned
|	|	
0037	31	Unassigned
0040	32	Executive BDT Pointer
0041	33	Immediate Storage Check Program Return Address
0042	34	Immediate Storage Check Designator Register
0043	35	Normal Program Return Address
0044	36	Normal Designator Register
0045	37	User BDT Pointer
0046	38	E00—0 BDI0 E20—0 BDI2
0047	39	E10—0 BDI1 E30—0 BDI3
0050	40	Quantum Timer
0051	41	Guard Mode Program Return Address
0052	42	Guard Mode Designator Register
0053	43	Guard Mode Interrupt Status
0054	44	Immediate Storage Check Status
0055	45	Normal Status
0056	46	P-Capture (IOU Error Interrupt)
0057	47	Designator Capture (IOU Error Interrupt)
0060	48	*
|	|	*
0067	55	*
0070	56	Jump History Stack
|	|	
0077	63	Jump History Stack

* Note that locations 60_8 through 67_8 are used as temporary working storage locations by the processor, and their contents are therefore unpredictable. Delayed storage checks are classed as normal interrupts (043, 044).

Table 3-7. GRS Register Assignments 64 Through 127

Octal	Decimal	Register Assignment
0100	64	Real-Time Clock
0101	65	User R1/Repeat Count
0102	66	User R2/Mask Register
0103	67	User R3/Staging Register 1
0104	68	User R4/Staging Register 2
0105	69	User R5/Staging Register 3
0106	70	User R6/J0
0107	71	User R7/J1
0110	72	User R8/J2
0111	73	User R9/J3
0112	74	User R10
|	|	
0117	79	User R15
0120	80	Executive R0
0121	81	Executive R1/Repeat Count
0122	82	Executive R2/Mask Register
0123	83	Executive R3/Staging Register 1
|	|	
0137	95	Executive R15
0140	96	Unassigned
0141	97	Executive X1
|	|	
0143	99	Executive X3 Initial Load MSR Value
|	|	
0153	107	Executive X11
0154	108	Executive X12/A0
|	|	
0157	111	Executive X15/A3
0160	112	Executive A4
|	|	
0173	123	Executive A15
0174	124	Unassigned
|	|	
0177	127	Unassigned

3.4.2.2. User Index (X) Registers - 0001-0017

The index registers, referred to as X-registers, provide the programmer with address modification capability (indexing).

An index register contains a modifier field (Xm), which is used to modify the operand address (indexing), and an increment field (Xi), which is used to modify the modifier field (automatic incrementation). If the relocation and storage suppression designator register bit 7 (D7) and the i-bit of an instruction are one, 24-bit index register mode is specified. In this mode, Xm is the lower 24 bits of the index register (bits 23-0), and Xi is the upper 12 bits of the index register (bits 35-24). In all other cases, 18-bit index register mode is selected. In this mode, Xm is the lower 18 bits of the index register (bits 17-0), and Xi is the upper 18 bits of the index register (bits 35-18).

3.4.2.3. User Accumulator (A) Registers – 0014–0033

The A–registers store arithmetic operands and results. The actual computation or logical function is performed in the arithmetic section, and the results are stored in the A–register or registers specified by the instruction. Four of the A–registers (addresses 14_8 – 17_8) overlap registers assigned as X–registers. This affords additional versatility in the use of A–registers and X–registers.

3.4.2.4. User Unassigned Registers – 0034–0037

Two of these unassigned registers (34_8 and 35_8) serve as an extension of the set of user A–registers when D6 = 0 and an instruction which requires more than one user A–register is being performed. All four of these unassigned registers can serve as general purpose registers.

3.4.2.5. Executive Bank Descriptor Table Pointer Register – 0040

The word at this location is read when the Executive bank descriptor pointer is specified.

3.4.2.6. Immediate Storage Check Interrupts – 0041–0042

When an interrupt occurs, these registers temporarily store the captured program return address and designators, respectively.

3.4.2.7. Normal Interrupts – 0043–0044

When an interrupt occurs, these registers store the normal captured program return address and designators, respectively.

3.4.2.8. User Bank Descriptor Table Pointer Register – 0045

The word at this location is read when the user bank descriptor pointer is specified.

3.4.2.9. Bank Descriptor Index Registers – 0046–0047

The control register at address 46_8 is used as a holding register for bank descriptors 0 and 2. The register at address 47_8 is used as a holding register for bank descriptors 1 and 3.

3.4.2.10. Quantum Timer – 0050

When an interrupt occurs, the captured quantum timer value is stored in this register.

3.4.2.11. Guard Mode – 0051–0053

When a guard mode fault interrupt occurs; the program return address is captured in address 51_8, the designators are captured at address 52_8, and the status is captured at address 53_8.

3.4.2.12. Immediate Storage Check Status - 0054

When an Immediate Storage Check interrupt occurs, the status is stored in this register.

3.4.2.13. Normal Status - 0055

Address 55_8 stores all processor–generated interrupt status except Immediate Storage Check status and Guard Mode status.

3.4.2.14. IOU Error Interrupts - 0056–0057

Address 56_8 provides a location in GRS for P–Capture, and address 57_8 provides a location in GRS for Designator Capture. These two address locations in the GRS allow software to recover when one of these IOU errors occurs.

3.4.2.15. Unassigned Registers - 0060–0067

The unassigned registers are not assigned to a specific use and may be used as temporary storage locations. Some of the unassigned registers may be used in the performance of the instructions that operate with a double word or triple word operand.

3.4.2.16. Jump History Stack - 0070–0077

The jump history stack consists of eight general register locations (70_8 to 77_8) that hold recent 24–bit absolute jump instruction addresses. Bit 35 of each entry contains a pass flag indicating whether the entry was stored on an odd or even pass through the stack. Entry stacking is activated by a Load Breakpoint Register (LBRX) instruction, according to the conditions specified in the breakpoint register. Unless terminated by one of these conditions, the process continues in a wraparound manner, and older entries are subsequently overwritten by new entries.

3.4.2.17. Real–Time Clock Register (RO) - 0100

The contents of the lower half (bit positions 17–00) of the real–time clock (RTC) register is decreased by one every 200 microseconds, independent of program control or supervision. The +0 is decremented to –1. A Real–Time Clock interrupt occurs if the RTC value in the lower half of the RTC register is zero when a decrementation cycle is initiated. The upper half (bit positions 35–18) of the RTC register should not be used.

3.4.2.18. User (R1) Repeat Count Register - 0101

The contents of the repeat counter register define the number of times a repeated instruction is executed. During execution of a repeated instruction, the contents of the lower half of the repeat count register is decreased by one each time the repeated instruction is executed. If an interrupt occurs during the sequence of repeated executions of an instruction, the repeat sequence is suspended to process the interrupt, and the current count is left in R1. The repeated sequence may be resumed after the interrupt has been processed. The final value of the count after the repeat sequence terminates is always available in R1. If the contents of the repeat count register is zero, the repeated instruction is not executed and the execution of the next instruction is initiated. Zero is defined as all zeros or all ones in the lower half of the word (bit positions 17–00); the upper half (bit positions 35–18) of the repeat count register should not be used.

3.4.2.19. User (R2)/Mask Register - 0102

The bits in the mask register specify the fields of operands to be operated upon in certain instructions. A logical AND is performed with the operand and the mask and/or its complement. The portions of the operand so selected are then used in the instruction operation.

3.4.2.20. User (R2-R5)/Staging Registers (SR1-SR3) - 0103-0105

The three staging registers are used for holding operand information and operation status for byte instruction execution.

3.4.2.21. User (R6-R9)/J-Registers (J0-J3) - 0106-0111

When the character addressing mode designator register bit 4 (D4) is one, j-field values of four through seven specify registers R6 through R9, respectively, instead of partial-word selections. These registers provide character addressing and indexing in a manner that is similar to, and in addition to, the word indexing function of the X-registers.

3.4.2.22. User R-Registers (R10-R15) - 0112-0117

These registers are unassigned and serve as general purpose registers. When D6 = 0, each of these registers can be implicitly addressed by one of the values 12_8 through 17_8 in the a-field of a Load R or Store R instruction.

3.4.2.23. Executive (R0) R-Register - 0120

This register is unassigned and serves as a general purpose register. When D6 = 1, this register is implicitly addressed when the a-field of a Load R or Store R instruction equals zero.

3.4.2.24. Executive (R1) Repeat Count Register - 0121

When D6 = 1, this register has the same function and format as the user R1 repeat count register.

3.4.2.25. Executive (R2)/Mask Register - 0122

When D6 = 1, this register performs the same function as the user R2 mask register.

3.4.2.26. Executive (R3-R5)/Staging Registers (SR1-SR3) - 0123-0125

When D6 = 1, these registers perform the same function as the user SR1-SR3 staging registers.

3.4.2.27. Executive (R6-R9)/J-Registers (J0-J3) - 0126-0131

When D6 = 1, these registers perform the same function as the user J0-J3 registers.

3.4.2.28. Executive R-Registers (R10-R15) - 0132-0137

These registers are unassigned and serve as general purpose registers. When D6 = 1, each of these registers can be implicitly addressed by one of the values 12_8 through 17_8 in the a-field of a Load R or Store R instruction.

3.4.2.29. Executive Index Registers (X1-X15) - 0141-0157

When D6 = 1, these registers perform the same functions as the user index registers.

3.4.2.30. Executive Accumulator Registers (A0-A15) - 0154-0173

When D6 = 1, these registers perform the same function as the user A-registers.

3.4.2.31. Executive Unassigned Registers - 0140, 0174-0177

When D6 = 1, these registers are used in the same manner as the unassigned registers at addresses 34_8-37_8.

3.4.2.32. Control Register Protection

When operating in guard mode (D2 = 1), a Guard Mode interrupt will occur if an attempt is made to execute a privileged (Executive) instruction or to store data into an Executive GRS location.

4. CPU Arithmetic and Control

4.1. General

The 1100/80 Systems central processor unit (CPU) comprises an arithmetic section, control section, maintenance section, general register stack, and interfaces for communicating with other units in the system.

The arithmetic and control sections are discussed in this section. The general register stack (GRS) is discussed in Section 3. A brief discussion of the maintenance section is in Section 2.

4.2. Arithmetic Section

4.2.1. General Operation

During the execution of logical and arithmetic instructions, the following steps are performed:

1. Transfer input data from instruction-word-specified storage locations or control registers to input registers in the arithmetic section. During the transfer, the input data are processed by the main control section to provide absolute values.

2. Perform the arithmetic operations of addition, subtraction (add negative), multiplication, division, byte manipulation, skip detection, etc., as specified by the instruction word.

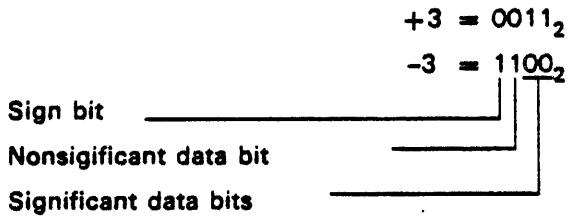
3. Transfer final results from the arithmetic section to temporary holding registers, general register storage, or indicate skip condition.

4.2.1.1. Data Word

The highest order binary bit represents the sign of the value contained in the remaining bit positions. If the sign bit contains a zero, the word is positive and 1's in the remaining bit positions represent significant data. If the sign bit contains a one, the word is negative and 0's in the remaining bit positions represent significant data. A binary data word containing all zeros is referred to as positive zero (+0). A binary data word containing all ones is referred to as negative zero (–0).

Example:

(Assume a 4-bit word length.)



4.2.1.2. Data Word Complement

The ones complement of any binary arithmetic data word is obtained when all zeros in the word are changed to ones, and all the ones are changed to zeros. An arithmetic data word of positive value, when complemented, becomes a negative value; and a negative value, when complemented, becomes a positive value.

4.2.1.3. Absolute Values

The absolute value of an arithmetic number is the magnitude of the number regardless of the sign.

Example:

Binary Value	Absolute Value
001110 (+14)	001110 (14)
110001 (-14)	001110 (14)

4.2.2. Microprogrammed Control

The 1100/80 Systems arithmetic section consists of three major parallel data paths. The main adder, the shifter, and the multiplier. The cycle time of the main adder and the shifter are both 150 nanoseconds. The multiplier is designed to have a 4-bit add and shift cycle time of 100 nanseconds. The main adder and shifter are extensively microprogram-controlled; however, the multiplier repeat cycles are self-initiated.

4.2.3. Main Adder Characteristics

The main adder of the CPU arithmetic section performs single- or double-precision adds or subtracts, or logical operations.

4.2.4. Fixed-Point Single- or Double-Precision Add or Subtract Overflow and Carry

In fixed-point arithmetic, the execution of certain instructions can result in an overflow or a carry condition. During execution, the overflow designator (D1) and the carry designator (D0) bits are cleared to zeros; the overflow and carry conditions set bits D1 and D0, respectively, in the designator register. These bits can be sensed by certain other instructions. Each of these designators, when set to one, remains in the set condition until the next time any one of the instructions in Table 4-1 is executed or until the Load Designator Register instruction is executed.

4.2.4.1. Overflow

An overflow condition is detected when one of the 10 instructions in Table 4-1 is executed and the numeric value of the result obtained exceeds the maximum numeric value that can be contained in the register holding the final result. Under this condition the resulting sign will be incorrect, an overflow enable is generated and sent to control, and D1 is set.

Table 4-1. Instructions That Condition the Carry and Overflow Designators

Function Code (Octal)	Instruction
f = 14, j = 00-17	Add to A
f = 15, j = 00-17	Add Negative to A
f = 16, j = 00-17	Add Magnitude to A
f = 17, j = 00-17	Add Negative Magnitude to A
f = 20, j = 00-17	Add Upper
f = 21, j = 00-17	Add Negative Upper
f = 24, j = 00-17	Add to X
f = 25, j = 00-17	Add Negative to X
f = 71, j = 10	Double-Precision Fixed-Point Add
f = 71, j = 11	Double-Precision Fixed-Point Add Negative

4.2.4.2. Carry

A carry condition is detected when an end-around carry occurs during the execution of an instruction listed in Table 4-1. The detection of a carry condition indicates that a carry was propagated out of the sign bit position and automatically added into the low-order bit position. The detection of the carry condition is significant when programming multiple-precision routines. In ones complement subtractive arithmetic, the carry condition can be equated to the no borrow condition, and the no carry condition to the borrow condition.

The condition of the carry designator can be tested by executing either the Jump Carry or Jump No Carry instructions. Table 4-2 lists the sign combinations for which the carry designator would be set to 1 indicating that a carry has occurred.

Table 4–2. Sign Bit Combinations Which Set Carry Designator

Operation	Input Operand Sign		Resultant Sign
	Augend	Addend	
Addition	+	-	+
	-	+	+
	-	-	+
	-	-	-
	Minuend	Subtrahend	
Subtraction (add negative)	+	+	+
	-	-	+
	-	+	+
	-	+	-

4.2.4.3. Arithmetic Interrupt

The arithmetic section cannot cause a system interrupt. But, when an arithmetic fault occurs, it generates a fault condition signal which allows the control section to set the appropriate designator bit. Other processor conditions in conjunction with those arithmetic fault conditions determine whether or not control generates an interrupt.

4.2.5. Fixed–Point Division

The process of dividing one fixed–point number by another consists of transferring the numbers to the arithmetic section, performing a series of trial subtractions to form a quotient and a remainder, transferring the properly signed quotient to a register and, if the remainder is to be saved, transferring the properly signed remainder to another register. All divide operations use the main adder and shifter.

4.2.6. Fixed–Point Multiplication

The arithmetic section contains a fast multiplier unit to handle multiplications. The main adder and shifter are used only in the beginning and ending cycles for input and output data adjustments.

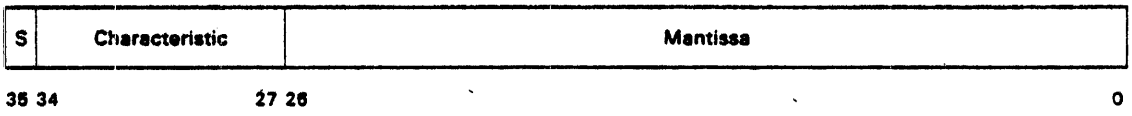
4.2.7. Floating–Point Arithmetic

Floating–point arithmetic handles the scaling problems which arise in computations involving numbers which vary widely in range. In floating–point arithmetic, the numbers are represented in a special format so that the computer can automatically handle the scaling.

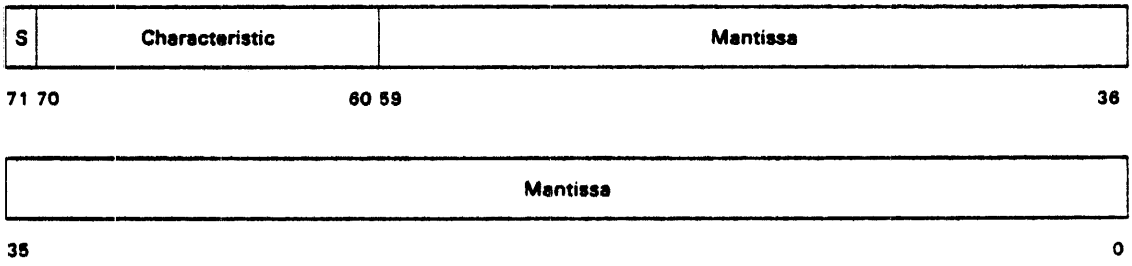
4.2.8. Floating-Point Numbers and Word Formats

Floating-point numbers in the instructions are represented in single-precision format as a 27-bit fractional quantity multiplied by the appropriate power of two, or in the double-precision format as a 60-bit fractional quantity multiplied by the appropriate power of two. The power of two is called the exponent. In machine representation, the exponents are biased to make them lie in the range of positive numbers or zero. These biased exponents are called characteristics. The fractional part is referred to as the mantissa. The two format types, single-precision and double-precision, are as follows.

Single-Precision Floating-Point Format



Double-Precision Floating-Point Format



An explanation of the sign bit, characteristic, and mantissa follows:

- Sign - The sign bit expresses the sign (S) of the numerical quantity represented by the floating-point number.

  If S = 0, the numerical quantity is positive (+).

  If S = 1, the numerical quantity is negative (-).

- Characteristic - The characteristic represents both the numerical value and the sign of the exponent.

  1. Single-Precision Characteristic - The 8-bit characteristic of a single-precision floating-point number represents an exponent value in the range +127 through -128. The characteristic is formed by adding a bias of +128 (200_8) to the exponent. Table 4-3 shows the range of characteristic values and corresponding exponent values.

Table 4–3. Single–Precision Floating–Point Characteristic Values and Exponent Values

Decimal Values		Octal Values	
Characteristic	Unbiased Exponent	Characteristic	Unbiased Exponent
255	+127	377	+177
128	000	200	000
000	-128	000	-200

2. Double–Precision Characteristic – The 11–bit characteristic of a double–precision floating–point number represents an exponent value in the range +1023 through -1024. The characteristic is formed by adding a bias of +1024 (2000_8) to the exponent. Table 4–4 shows the range of characteristic values and the corresponding exponent values.

Table 4–4. Double–Precision Floating–Point Characteristic Values and Exponent Values

Decimal Values		Octal Values	
Characteristic	Unbiased Exponent	Characteristic	Unbiased Exponent
2047	+1023	3777	+1777
1024	0000	2000	0000
0000	-1024	0000	-2000

- Mantissa – The mantissa portion of a floating–point number represents the fractional part of the number. In the instructions, the fractional part is normalized so that the absolute values represented are greater than or equal to 1/2, but less than one. Zero cannot be represented in this range; it is considered to be normalized as it stands. The binary point of a floating–point number is assumed to lie between the last bit of the characteristic and the first bit of the mantissa. The mantissa of a single–precision floating–point number contains 27 bits; for a double–precision floating–point number, the mantissa contains 60 bits. The mantissa need not be normalized for all instructions.

4.2.8.1. Single–Precision Floating–Point Numbers

A single–precision floating–point number can be derived from a positive decimal number as follows:

Example:

Given number = $+12_{10}$

$+12_{10} = 1100_2 = .1100_2 \times 10_2 + 4$

- Sign = + = 0
- Characteristic = exponent + bias
  $= 00\ 000\ 100_2 + 10\ 000\ 000_2$
  $= 10\ 000\ 100_2$
- Mantissa = $.110\ 000\ \ldots\ 000_2$
- The format for the floating–point number is as shown (sign included):

Sign	Characteristic	Mantissa
0	10 000 100	1100 0

35 34 27 26 0

$= 204600000000_8$

4.2.8.2. Double–Precision Floating–Point Numbers

A double–precision floating–point number can be derived from a positive decimal number following the same steps that were used for single–precision with these two exceptions:

- A bias value of 2000_8 is added to the exponent to form the characteristic. For single–precision the value is 200_8.
- The mantissa is 60 bits instead of 27 bits.

4.2.8.3. Negative Floating–Point Numbers

A floating–point number can be derived to represent a given negative number as follows:

- Represent the given number as a positive floating–point number.
- Form the ones complement of the entire positive floating–point number.

Example:

Given number = -12_{10}

- The single–precision floating–point number for $+12_{10}$ (including sign) is 204 600 000 000_8.
- The single–precision floating–point number for -12_{10} (including sign) is 573 177 777 777_8.

4.2.8.4. Residue

During single–precision floating–point Add or Add Negative, the bits shifted off the right end of the register during alignment of the mantissas is not included in the addition but saved, becoming the residue. After the addition is performed, the sum and the residue are each packed into floating–point format, the sum is stored, and the residue is stored if the floating–point residue store enable designator (D17) is one.

When the two 36–bit input operands for an Add or Add Negative instruction are transferred to the arithmetic section, their characteristics are examined, and the mantissa of the input operand with the smaller characteristic is right–shifted a number of bit positions equal to the difference between the characteristics. The bits shifted out of the 36–bit arithmetic register are saved in an auxiliary register. The portion of the mantissa saved in the auxiliary register is used to form the residue and is not included in the algebraic addition. After completion of the addition and any shifting necessary to normalize the sum, the sum and the residue are packed into single–precision floating–point format and transferred to two consecutive A–registers.

4.2.9. Normalized/Unnormalized Floating–Point Numbers

A floating–point number is normalized when the leftmost bit of the mantissa is not identical to the sign bit or when all bits of the mantissa are identical to the sign bit. A floating–point number is not normalized when all bits of the mantissa are not sign bits and the leftmost bit of the mantissa is identical to the sign bit.

All floating–point operations produce a normalized result when the input operands are normalized. The sums produced by Floating Add and Floating Add Negative instructions and the result produced by the Load and Convert to Floating instruction are always normalized, regardless of whether or not the input operands are normalized. When either or both input operands are not normalized, the result obtained may be less accurate than if normalized input operands had been used.

Normalized input operands must be used for the Floating Multiply, Divide, Compress and Load, and Expand and Load instructions. If normalized input operands are not used for these instructions, the results are undefined.

4.2.10. Floating–Point Characteristic Overflow/Underflow

Floating–point characteristic overflow/underflow occurs when the characteristic does not lie in the range represented in the number of bits allowed for the characteristic.

When any of the Floating–Point Add, Add Negative, Multiply, Divide, or Load and Convert instructions, or the Compress and Load instruction are performed, overflow or underflow may occur.

4.2.10.1. Floating–Point Characteristic Overflow

Single–precision floating–point characteristic overflow occurs when the 8–bit characteristic of the resultant most significant single–precision floating–point word represents a number greater than 377_8 and the associated mantissa is not zero.

Double-precision floating-point characteristic overflow occurs when the 11-bit characteristic of the resultant double-precision floating-point number represents a number greater than 3777_8 and the associated mantissa is not zero.

When overflow is detected, the action taken depends on the arithmetic exception interrupt designator (D20). The characteristic overflow designator (D22) is always set.

4.2.10.2. Floating-Point Characteristic Underflow

Single-precision floating-point characteristic underflow occurs when the resultant floating-point word represents a negative number and the associated mantissa of the result is not zero. This means that the exponent of the result is less than -200_8, thus the attached sign (positive – because absolute value is used) changes due to the borrow. If the characteristic of the residue (Floating Add, Floating Add Negative), remainder (Floating Divide), or the least significant single-precision word of the product (Floating Multiply) represents a negative number, this fact by itself does not result in underflow. Instead, the residue, remainder, or least significant word of the product is cleared to all zero bits or set to all one bits (to reflect the appropriate sign).

Double-precision floating-point characteristic underflow occurs when the 11-bit characteristic of the result represents a negative number, i.e., the exponent of the result is less than 2000_8, the mantissa of the result is not zero, and the double-precision underflow designator (D5) is cleared.

When underflow is detected, the characteristic underflow designator (D21) is always set and the action taken by the CPU depends on the state of D20.

4.2.10.3. Floating-Point Divide Fault

For single- or double-precision floating-point division, a divide fault condition will be detected when the mantissa of the divisor is zero. The action taken depends on the floating-point zero format selection designator (D8, for single-precision floating-point division only) and D20. The divide check designator (D23) is always set.

4.2.11. Fixed-Point to Floating-Point Conversion

Conversion of a fixed-point number to floating-point number is performed in the arithmetic section. The first input operand contains a characteristic (biased exponent) which defines the location of the binary point for the fixed-point number with respect to the standard position of the binary point for a floating-point number. The second input operand is the signed fixed-point number to be converted.

The conversion process consists of transferring the two operands to the arithmetic section, shifting the fixed-point number, if necessary, to position its bits as the mantissa for a normalized floating-point number. Modify the characteristic to reflect the magnitude and direction of the normalizing shift. Pack the shifted fixed-point number (mantissa) and the modified characteristic in floating-point format. Load the packed results in a register (conversion to single-precision floating-point format) or into two consecutive registers (conversion to double-precision floating-point format).

4.2.12. Floating-Point Addition

The process of adding two floating-point numbers consists of loading the numbers into the arithmetic section, determining the difference between the characteristics of the two numbers, shifting (right) the mantissa of the number having the smaller characteristic, adding the mantissas, combining the results in floating-point format, and transferring the resulting floating-point numbers to GRS.

The input operands for floating-point addition need not be normalized numbers. For single-precision addition, the sum (most significant word produced) is always a normalized number. The residue word may or may not be a normalized number. For double-precision addition, the sum is always a normalized number.

4.2.13. Double-Precision Floating-Point Addition

The steps performed for double-precision floating-point addition are similar to those for the single-precision addition with these six differences:

1. Each of the two operands occupy two 36-bit registers in the arithmetic section. In single-precision addition both operands are contained in two 36-bit registers.

2. The mantissa sum can contain a maximum of 60 bits in double-precision addition, instead of 27 bits as in single-precision addition.

3. The bits that are shifted out of the right end of the 36-bit register when the operands are lined up prior to addition are lost. There is no residue.

4. Double-precision characteristic overflow occurs when the characteristic is greater than 3777_8 and the mantissa is not zero.

5. Double-precision underflow occurs when the exponent is less than -2000_8 and the mantissa is not zero. In single-precision the value is -200_8.

6. The sum is stored in two consecutive registers, A and A+1. No residue is stored.

4.2.14. Floating-Point Subtraction (Add Negative)

Floating-point subtraction (both single-precision and double-precision) uses the same routine as for the Floating-point Add operation.

4.2.15. Floating-Point Multiplication

The process of multiplying two floating-point numbers consists of loading normalized input operands into the arithmetic section, unpacking, multiplying the mantissas, adding the characteristics, packing the results into floating-point format, and transferring the result to GRS. The results obtained for all cases in which either or both input operands are not normalized numbers are undefined.

4.2.16. Floating-Point Division

The process of dividing one floating-point number by another consists of loading the normalized input operands into the arithmetic section, unpacking, dividing one mantissa by the other, subtracting the characteristics, packing the results into floating-point format, and transferring the result to GRS. The results obtained for all cases in which either or both input operands are not normalized numbers are undefined.

4.2.17. Floating-Point Zero

Floating-point zero can be defined as a floating-point number having all mantissa bits identical to the sign bit.

4.2.18. Byte Instructions

This class of instructions is designed to permit transference, translation, comparison, and arithmetic computation of data in the form of predetermined bit patterns (e.g., half words, third words, quarter words, and sixth words) referred to as bytes.

There are a total of 15 distinct instructions that perform the various multiword (byte string) operations noted above. These instructions may be arranged under three functional groups:

1. instructions that involve byte transfers and manipulations between one storage location and another;

2. instructions that permit the mutual transference and manipulation of data among storage and various control and arithmetic registers; and

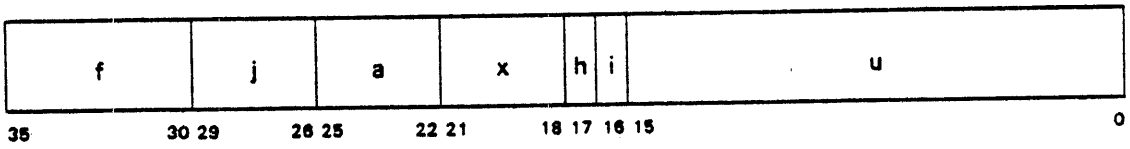
3. instructions that perform decimal arithmetic addition and subtraction operations.

Twelve of the byte instructions are performed in the arithmetic section. The remaining three instructions (33,00 - Byte Move; 33,01 - Byte Move with Translate; and 33,07 - Edit) are performed in the main control section.

4.3. Control Section

4.3.1. Instruction Word Format

During the running of a program in the 1100/80 Systems CPU, instructions are transferred from main storage locations to the control section of the CPU. The instructions are transferred from sequentially addressed main storage locations until the sequence is broken by the program or interrupted by the control section's reaction to some special condition or event. Each instruction is a coded directive to the control section; the control section initiates a sequence of steps necessary to perform the particular operation prescribed by the instruction. The 36-bit instruction word, illustrated below, is subdivided into seven fields.



where:

f = Function Code

j = Operand Qualifier, Character Addressing, Partial Control Register Address, or Minor Function Code

a = A–, X–, or R–register; Channel Number, Jump Key or Stop Keys Number; Minor Function Code; Partial Control Register Address

x = Index Register

h = Index Register Incrementation Control

i = Indirect Addressing Control, Base Register Suppression Control, 24–Bit Indexing Control, or Operand Basing Selector

u = Operand Address or Operand Base

4.3.2. Instruction Word Fields

The following paragraphs describe the manner in which the CPU's control section reacts to the contents of each of the seven fields of an instruction word.

4.3.2.1. Use of the f–Field

The f–field is used to define the basic operation to be performed for all legal values of f less than or equal to 70_8 (except 07_8, 33_8, and 37_8). When the f–field is 07_8, 33_8, 37_8 or greater than 70_8, the f– and j–fields are combined to form a 10–bit field used to define the basic operation. For 11 of these f, j combinations, the value in the a–field is used to define variations of the basic operation. All function codes are defined in Section 5 and listed in Appendix C.

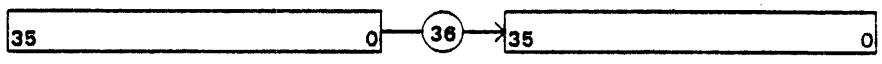
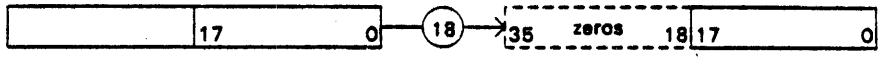
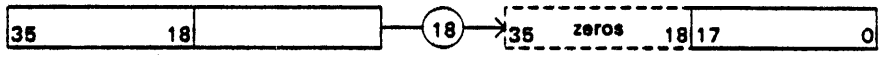
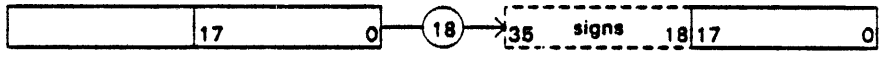
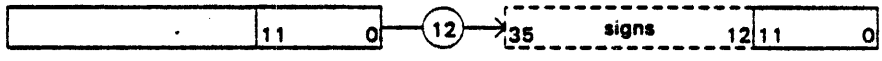
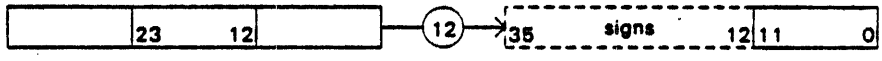
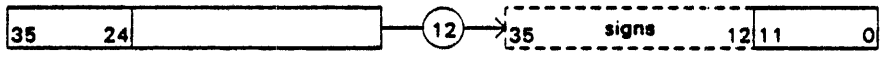
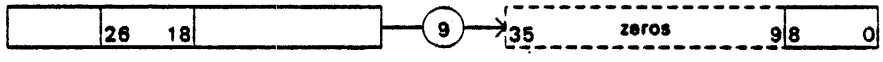
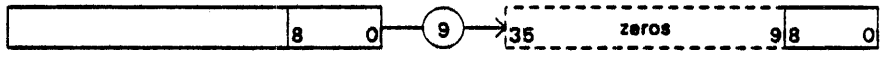
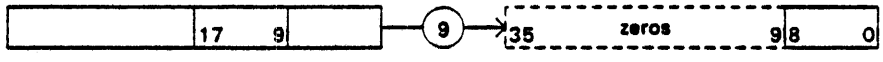
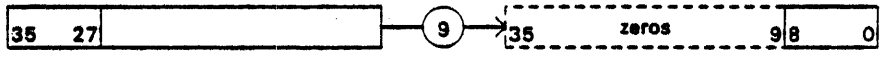
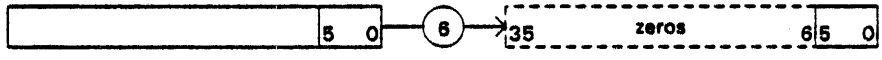
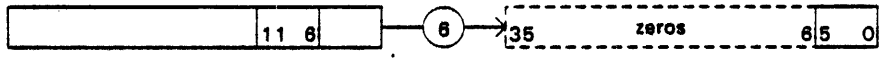
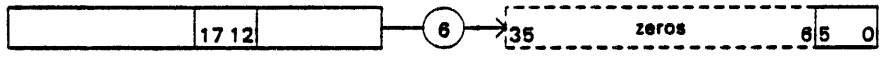
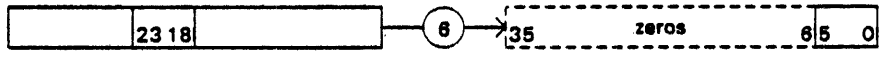
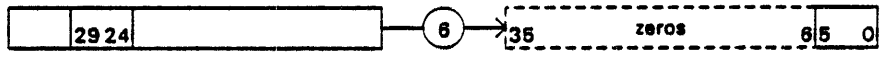
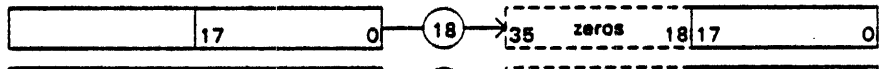
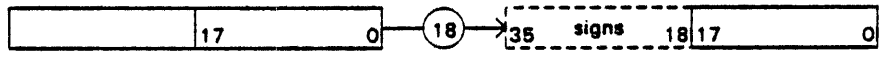
4.3.2.2. Description of the j–Field

When f is less than 70_8 (except 07_8, 33_8, and 37_8), the j–field is used as an operand qualifier or to identify a J–register used in the character addressing mode. When f is equal to 70_8, the j–field is used as part of a control register address. When f is 07_8, 33_8, 37_8 or greater than 70_8, the j–field and the f–field are used to define a basic operation; in this instance, the j–field operates as a minor function code.

4.3.2.2.1. Use of the j–Field as an Operand Qualifier

When the f–field of an instruction contains a value in the range 01_8 through 67_8 (except 07_8, 33_8, and 37_8) and the character addressing mode designator (D4) = 0, the j–field is used as an operand qualifier which specifies the data transfer pattern to or from main storage, except as specified in 4.3.2.2:2.

The j–field can contain values ranging from 0 through 17_8. Each value, except 4_8 through 7_8, determines a specific data transfer pattern. Each of the j–field values 4_8 through 7_8 may specify either of two different data transfer patterns, or character addressing with the choice dependent on the contents of the quarter word mode selector (D10) and D4 of the designator register (see 8.2.1). If D4 = 1, character addressing is specified and each of the j–field values 4_8 through 7_8 specify a J–register as explained in 4.3.2.2.2. Figures 4–1 and 4–2 illustrate all the possible data transfer patterns which can be specified by the j–field when D4 = 0.

J (Octal)	Quarter-Word Designator*	Storage Location → Arithmetic Register
0	0 or 1	
1	0 or 1	
2	0 or 1	
3	0 or 1	
4**	0	
5**	0	
6**	0	
7**	0	
4**	1	
5**	1	
6**	1	
7**	1	
10	0 or 1	
11	0 or 1	
12	0 or 1	
13	0 or 1	
14	0 or 1	
15	0 or 1	
16***	0 or 1	
17***	0 or 1	

* The Quarter-Word Mode Designator bit (D10) is held in the designator register.

** Character Addressing Mode Designator bit (D4) will imply J-register usage for instruction codes less than f = 70 (except 07, 33, 37) for character or byte manipulation. D4 overrides D10.

*** If x = 0, the h, i, and u are transferred. If x is not equal to zero, then u + (X_x) is transferred.

Figure 4-1. Data Transfers from Storage

J (Octal)	Quarter-Word Designator*	Arithmetic Register → Storage Location
0	0 or 1	35 0 → 36 → 35 0
1	0 or 1	17 0 → 18 → 17 0
2	0 or 1	17 0 → 18 → 35 18
3	0 or 1	17 0 → 18 → 17 0
4**	0	17 0 → 18 → 35 18
5**	0	11 0 → 12 → 11 0
6**	0	11 0 → 12 → 23 12
7**	0	11 0 → 12 → 35 24
4**	1	8 0 → 9 → 26 18
5**	1	8 0 → 9 → 8 0
6**	1	8 0 → 9 → 17 9
7**	1	8 0 → 9 → 35 27
10	0 or 1	5 0 → 6 → 5 0
11	0 or 1	5 0 → 6 → 11 6
12	0 or 1	5 0 → 6 → 17 12
13	0 or 1	5 0 → 6 → 23 18
14	0 or 1	5 0 → 6 → 29 24
15	0 or 1	5 0 → 6 → 35 30
16	0 or 1	No Data Transfer
17	0 or 1	No Data Transfer

* The Quarter-Word Mode Designator bit (D10) is held in the designator register.

** Character Addressing Mode Designator bit (D4) will imply J-register usage for instruction codes less than f = 70 (except 07, 33, 37) for character or byte manipulation. D4 overrides D10.

Figure 4-2. Data Transfers to Storage

- Operand qualification when f = 10_8 through 67_8 (except 33_8 and 37_8)

  These instructions require the transfer of a full 36–bit word or a partial word to the arithmetic section.

  1. If j = 0_8, the full 36–bit word addressed by U is transferred to the arithmetic section.

  2. If j = 01_8 through 15_8 and U specifies a main storage location (U $\geqslant$ 200_8), a partial word is transferred to the arithmetic section. In the arithmetic section, the partial word is extended to a full 36–bit word, either by zero fill or by sign bit fill from the leftmost bit position of the partial word, as illustrated in Figure 4–1.

  3. If j = 16_8 or 17_8, an 18–bit partial word is transferred to the arithmetic section. Details on the formation of this partial word and its extension are given in 4.3.2.8.2.

When j = 01_8 through 15_8 and U specifies a control register (U $\leqslant$ 177_8), the j–field is treated as if it contained 0_8 and the full 36–bit word is transferred from the control register to the arithmetic section.

- Operand qualification for store and block transfer instructions

  The full 36–bit word in the control register specified by the a–field (see 4.3.2.3.1) is transferred to a nonaddressable register in the data shift/complement/store section (f=01_8 through 04_8 and 06_8). The nonaddressable register is cleared to 0 when f=05_8.

  1. If j = 0_8, the full 36–bit word is transferred from the nonaddressable register to the location (main storage or control register) specified by U.

  2. If j = 01_8 through 15_8 and U specifies a main storage location (U $\geqslant$ 200_8), a partial word is transferred from the least significant bit positions of the nonaddressable register to the main storage location as shown in Figure 4–2. The contents of the remaining bit positions of the main storage location are not changed. Partial word writes of a third word, quarter word, or sixth word increase the storage cycle time to 200 nanoseconds.

  3. If j = 16_8 or 17_8, data is never transferred from the nonaddressable register to any storage location (main storage or control register).

When j = 01_8 through 15_8 and U specifies a control register (U $\leqslant$ 177_8), the j–field is treated as if it contained 0_8, and the full 36–bit word is transferred to the control register.

4.3.2.2.2. Use of the j–Field to Specify Character Addressing

When the f–field of an instruction contains a value in the range 01 through 67_8 (except 07_8, 33_8, and 37_8), D4 (the character addressing mode selector) = 1, and j = 4_8, 5_8, 6_8, or 7_8, the character addressing mode is specified. When the character addressing mode is specified, a j–field value of 4, 5, 6, or 7 specifies J0, J1, J2, or J3, respectively, in the GRS, as the register defining character or byte size, the position of the byte within a word, and other details. When the GRS selection designator (D6) = 0, the J–register is selected from the set of four J–registers at GRS locations 106 through 111_8. When D6 = 1, the J–register is selected from the set of four J–registers at GRS locations 126 through 131_8. The format of a J–register word as used in the character addressing mode is shown in Figure 4–3 and explained in Table 4–5.

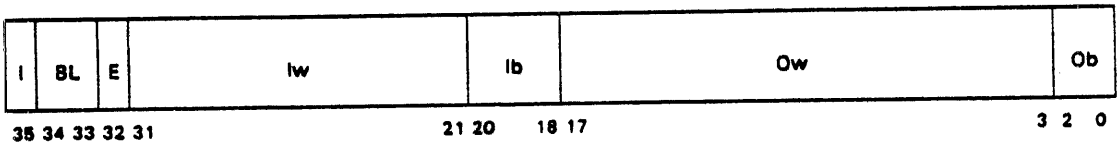


Figure 4–3. J–Register Format for Character Addressing Mode

Table 4–5. Explanation of J–Register Fields for Character Addressing Mode

Bit Positions	J–Register Field Identifier	Interpretation
35	I	The I–bit of the J–Register, in conjunction with the h–bit of the instruction, specifies whether or not the contents of the Ow– and Ob–fields are modified when the instruction is performed as follows: ■ I = 0 or h = 0 specifies no J–register modification* ■ I = h = 1 specifies modification of Ow and Ob by Iw and Ib, respectively.
34,33	BL	Specifies the byte length, as follows: ■ BL = 0 specifies a 9–bit byte ■ BL = 1 specifies an 18–bit byte ■ BL = 2 specifies a 6–bit byte ■ BL = 3 specifies a 12–bit byte
32	E	Specifies the bit used to extend the byte to 36 bits, if necessary, as follows: ■ E = 0 specifies extension with 0 bits ■ E = 1 specifies extension with the high order bit of the byte
31–21	Iw	Iw specifies the increment (or decrement) in words. Ib specifies the increment (or decrement) in bytes. If I = 0, or h = 0 these two values are ignored.
20–18	Ib	If I = 1 and h = 1, the values in the Iw– and Ib–fields are added to the values in the Ow– and Ob–fields, and the sums are stored in the Ow– and Ob–fields after the initial values in these two fields are used to form the absolute address of a word and select a byte within the word.

Table 4–5. Explanation of J–Register Fields for Character Addressing Mode (continued)

Bit Positions	J–Register Field Identifier	Interpretation
17–3	Ow	The offset in words. This value is used to form the relative address U and the absolute addresses SI and SD.
2–0	Ob	The offset in bytes. This value is used to select a particular byte within the selected word. The valid values of Ob for the possible values of BL are shown in Figure 4–4. Other byte selections are not defined.

* If I = 0, h = 1 in the instruction word specifies index register modification when x ≠ 0.

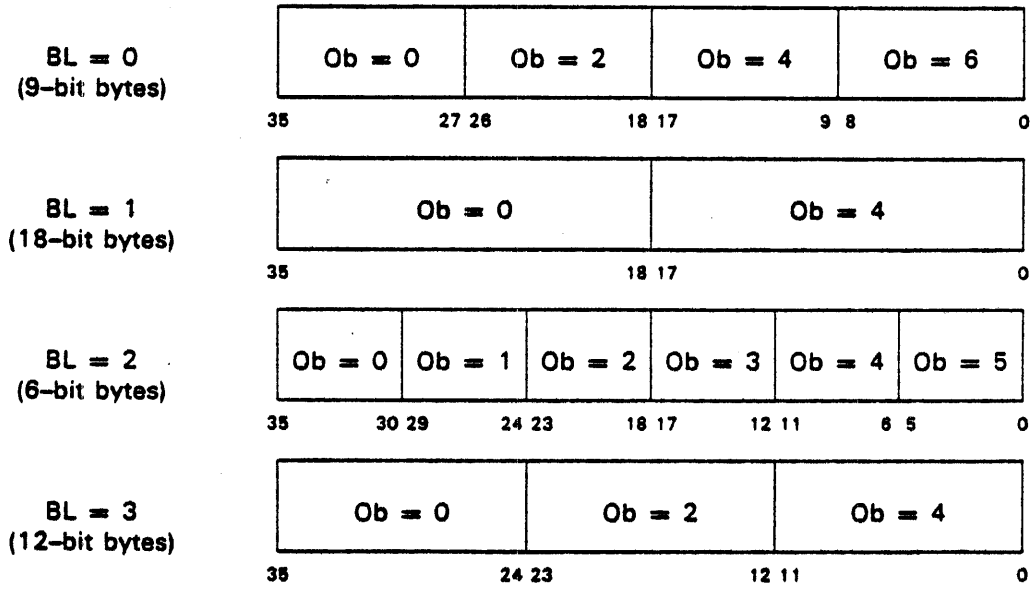


Figure 4–4. Byte Selected for Valid Combinations of BL and Ob Field Values

The additions performed when I = 1 and the h–bit of the instruction = 1 are symbolized by

$$Ob + Ib \rightarrow Ob$$

and

$$Ow + Iw \rightarrow Ow$$

The values in the Ow– and Ob–fields are always treated as positive values in these additions. The high order bit in the Iw–field (bit 31 of the specified J–register) is applied as the sign of both the Iw– and Ib–fields. If this sign is a zero bit, forward modification of Ow/Ob is performed. Forward modification permits incrementing the Ow– or Ob–field value (or both) to produce new Ow– and

Ob-field values to select any desired byte in lower order bit positions of the same word or select any desired byte in any word having a higher address. If the sign bit applied to the Iw- and Ib-fields is a one bit, backward modification of Ow/Ob is performed. Backward modification permits decrementing the Ow- and Ob-field value (or both) to produce new Ow- and Ob-field values to select any desired byte in higher order bit position of the same word or select any desired byte in any word having a lower address.

The result produced for the addition Ob + Ib → Ob is dependent on the two values used as inputs, the sign in the Iw-field, and the value in the BL- field, as shown in Tables 4-6 through 4-9. The valid combinations of Ob and Ib are shown in these tables. The result produced when any other combination is used is undefined.

The addition Ow + Iw → Ow is performed in an 18-bit ones complement subtractive adder after extending the 15-bit Ow-field to 18 bits with three high order zero bits and extending the 11-bit Iw-field to 18 bits with seven high order bits identical to the sign bit in the Iw-field. A carry or borrow generated in the addition Ob + Ib → Ob also enters the Ow + Iw → Ow addition. The sum is stored in the Ow-field of the specified J-register after the initial value in the Ow-field is used to form the relative and absolute addresses needed for the instruction.

If the value in the Ow-field is modified by adding a positive Iw value to produce an 18-bit sum greater than 077777_8 or by adding a negative Iw value to produce a negative 18-bit sum, the 15-bit value stored in Ow is undefined. Producing a negative 18-bit sum is a common programming error which can be avoided by choosing an artificially high-initial value for Ow and reducing the initial value of Xm by a like amount.

If the value U produced by the addition u + Xm + Ow (see 4.3.2.5) for an instruction which specifies the character addressing mode is less than 200_8, a register in the GRS is not referenced. Instead, the values SI and SD are produced, and if U passes the storage limits test, an attempt is made to reference main storage.

Table 4-6. Output Ob Values Produced When BL = 0

			Valid Input Ob Values			
BL = 0 (9-Bit Bytes)	Number of Bytes Forward or Backward	Valid Ib Value	0	2	4	6
Forward Modification (J_{31} = 0)	0	0	0	2	4	6
	1	2	2	4	6	0_C
	2	4	4	6	0_C	2_C
	3	6	6	0_C	2_C	4_C
Backward Modification (J_{31} = 1)	0	7	0	2	4	6
	1	5	6_B	0	2	4
	2	3	4_B	6_B	0	2
	3	1	2_B	4_B	6_B	0

For valid Ob/Ib input combinations

C = Carry (+1) to Ow + Iw → Ow addition

B = Borrow (-1) to Ow + Iw → Ow addition

Table 4-7. Output Ob Values Produced When BL = 1

			Valid Input Ob Values	
BL = 1 (18-Bit Bytes)	Number of Bytes Forward or Backward	Valid Ib Value	0	4
Forward Modification (J_{31} = 0)	0	0	0	4
	1	4	4	0_C
Backward Modification (J_{31} = 1)	0	7	0	4
	1	3	4_B	0

For Valid Ob/Ib Input Combinations

C = Carry (+1) to Ow + Iw → Ow addition

B = Borrow (-1) to Ow + Iw → Ow addition

Table 4-8. Output Ob Values Produced When BL = 2

			Valid Input Ob Values					
BL = 2 (6-Bit Bytes)	Number of Bytes Forward or Backward	Valid Ib Value	0	1	2	3	4	5
Forward Modification (J_{31} = 0)	0	0	0	1	2	3	4	5
	1	1	1	2	3	4	5	0_C
	2	2	2	3	4	5	0_C	1_C
	3	3	3	4	5	0_C	1_C	2_C
	4	4	4	5	0_C	1_C	2_C	3_C
	5	5	5	0_C	1_C	2_C	3_C	4_C
Backward Modification (J_{31} = 1)	0	7	0	1	2	3	4	5
	1	6	5_B	0	1	2	3	4
	2	5	4_B	5_B	0	1	2	3
	3	4	3_B	4_B	5_B	0	1	2
	4	3	2_B	3_B	4_B	5_B	0	1
	5	2	1_B	2_B	3_B	4_B	5_B	0

For valid Ob/Ib combinations

C = Carry (+1) to Ow Iw → Ow addition

B = Borrow (-1) to Ow Iw → Ow addition

Table 4-9. Output Ob Values Produced When BL = 3

			Valid Input Ob Values		
BL = 3 (12-Bit Bytes)	Number of Bytes Forward or Backward	Valid Ib Value	0	2	4
Forward Modification ($J_{31} = 0$)	0	0	0	2	4
	1	2	2	4	0_C
	2	4	4	0_C	2_C
Backward Modification ($J_{31} = 1$)	0	7	0	2	4
	1	5	4_B	0	2
	2	3	2_B	4_B	0

For valid Ob/Ib combinations

C = Carry (+1) to Ow Iw → Ow addition

B = Borrow (-1) to Ow Iw → Ow addition

4.3.2.2.3. Use of j-Field as Partial Control Register Address

When f = 70_8, the most significant bit of the j-field is ignored by the hardware, and the three low-order bits are combined with the contents of the a-field to form a 7-bit control register address.

4.3.2.2.4. Use of j-Field as Minor Function Code

When f = 07_8, 33_8, 37_8, or 71_8 through 76_8, the value in the j-field is a minor function code designator. An explanation of the details of each of these instructions is given in Section 5. They are summarized in Appendix C.

4.3.2.3. Uses of the a-Field

The contents of the a-field of an instruction word has a number of uses. The exact use is dependent on the instruction being performed and, in many cases, on the contents of the designator register.

4.3.2.3.1. Use of the a-Field to Reference an A-Register

For most of the instructions, the value in the a-field references one of the A-registers. When the A-, X-, and R-register set selector, D6, is equal to 0, each value in the range 0 through 17_8 in the a-field references one of the user A-registers in the range of control register addresses 14_8 through 33_8, respectively. When D6 = 1, each value in the range 0 through 17_8 in the a-field references one of the Executive A-registers in the range of control register addresses 154_8 through 173_8, respectively. In some instructions, the value in the a-field references two or three A-registers. When two or three A-registers are referenced, the value in the a-field explicitly references register Aa, and implicitly references registers Aa+1 and Aa+2.

The unassigned control registers at addresses 34_8, 35_8, 174_8, and 175_8 can be used as extensions of the two sets of 16 A-registers. For example, when a = 17_8 and the instruction requires referencing three A-registers (Aa, Aa+1, and Aa+2) then:

- If D6 = 0, the last user A-register (address 33_8) is referenced for Aa, the first user unassigned control register at address 34_8 is referenced for Aa+1, and address 35_8 is referenced for Aa+2.
- If D6 = 1, the last Executive A-register at address 173_8 is referenced for Aa, the following Executive unassigned control register at address 174_8 is referenced for Aa+1, and address 175_8 is referenced for Aa+2.

4.3.2.3.2. Use of the a-Field to Reference an X-Register

For certain instructions, the value in the a-field references one of the X-registers. When D6 = 0, each value in the range of 01_8 through 17_8 in the a-field references one of the user X-registers in the range of control register addresses 01_8 through 17_8, respectively; if a = 0, the user nonindexing X-register at control register address 0 is referenced. When D6 = 1, each value in the range of 01_8 through 17_8 in the a-field references one of the Executive X-registers in the range of control register addresses 141_8 through 157_8, respectively; if a = 0, the Executive nonindexing X-register at control register address 140_8 is referenced.

4.3.2.3.3. Use of the a-Field to Reference an R-Register

For certain instructions, the value in the a-field references one of the R-registers. When D6 = 0, each value in the range of 0 through 17_8 in the a-field references one of the user R-registers at control register addresses 100_8 through 117_8, respectively. When D6 = 1, each value in the range of 0 through 17_8 in the a-field references one of the Executive R-registers at control register addresses 120_8 through 137_8, respectively.

4.3.2.3.4. Use of the a-Field to Reference a Jump Key

For a Jump Key instruction, each value in the range of 01_8 through 17_8 in the a-field references one of the 15 select jump control circuits in the CPU. These circuits may be individually set and cleared via switches on the operator/maintenance panel on the system transition unit (STU).

4.3.2.3.5. Use of the a-Field to Reference Halt Keys

For a Halt Keys and Jump instruction, each of the four bit positions in the a-field references one of the four select stop control circuits in the CPU. These circuits may be individually set and cleared via switches on the STU.

4.3.2.3.6. Use of the a-Field as Minor Function Code

The value in the a-field specifies a particular variation of the basic operation initiated by the f, j combination of the following instructions:

- Load Breakpoint Register/Store Jump Stack
- Load Processor State Register
- Initiate Interprocessor Interrupt/Enable Second Day Clock/Enable Day Clock/Disable Day Clock
- Test and Set/Test and Set and Skip/Test and Clear and Skip

- Jump Overflow/Jump Floating Underflow/Jump Floating Overflow/Jump Divide Fault
- Jump No Overflow/Jump No Floating Underflow/Jump No Floating Overflow/Jump No Divide Fault

4.3.2.4. Use of the j– and a–Fields to Specify GRS Control Register Address

For the Jump on Greater and Decrement instruction, the values in the j–field and a–field combine to form a 7–bit address (the leftmost bit of the j–field is ignored). The 7–bit address specifies which one of the 128 addressable GRS control registers is to be used as the counter for the instruction.

4.3.2.5. Use of the x–Field

An indexing operation which utilizes a ones complement subtractive adder occurs for every instruction. If the A–, X–, and R–register set selector, D6, is equal to 0, each x–field value in the range 01_8 through 17_8 references one of the user X–registers at control register addresses 01_8 through 17_8, respectively. If D6 = 1, each x–field value in the range 01_8 through 17_8 references one of the Executive X–registers at control register addresses 141_8 through 157_8, respectively. When the value in the x–field is not zero, the value in the Xm–field of the X–register specified by the x–field is added to the extended contents of the u–field to form the relative operand address or an operand. This indexing operation is symbolized by the notation: u + Xm = U except for instructions which specify the character addressing mode (see 4.3.2.2.2) and for most byte instructions (see 4.2.18). In these cases it is symbolized by the notation u + Xm + Ow = U. Xm is an 18–bit field unless 24–bit indexing is specified.

When the value in the x–field is zero, no index register is referenced. An indexing operation, however, does occur. It consists of adding an 18–bit half word of all zero bits to the extended u–field value to form the relative operand address or operand. This indexing operation is symbolized by the notation: u + 0 = U or u + 0 + 0w = U.

An indexing operation never produces a U value consisting of all one bits. This applies when U is a relative address and also when U is extended with zero bits (j = 16_8) or with sign bits (j = 17_8) for use as an immediate operand.

Example:

If j ≠ 16 or 17_8, u = 000001_8, and Xm = 777776_8

then

u + Xm = U = 000001_8 + 777776_8 = 000000_8

Example:

If f = $10–67_8$ (except 33 and 37_8), j = 16 or 17_8

and

h = i = 0, u = 177777_8, and Xm = 600000_8,

then

u + Xm = U = 177777_8 + 600000_8 = 000000_8

Example:

If f = 10–67$_8$ (except 33 and 37$_8$), j = 16 or 17$_8$,

and

h = i = 1, u = 177777$_8$, and x = 0,

then

h, i, u + 0 = U = 777777$_8$ = 0 = 000000$_8$

4.3.2.6. Use of the h–Field

If the x–field of an instruction contains a nonzero value, the h–bit determines whether or not the contents of the X–register specified by the x–field of the instruction or the Ow– and Ob–fields of an instruction specifying the character addressing mode are modified.

After the indexing operation is complete, if h = 1, and x ≠ 0 for an instruction which does not specify the character addressing mode, or an instruction which specifies the character addressing mode and a J–register containing I = 0 (see 4.3.2.2.2), the contents of the Xi–field of the specified X–register is added to the contents of the Xm–field of the same register, and the sum is stored back in the Xm–field. The process is Xm + Xi → Xm. If the relocation and storage suppression designator (D7) or i = 0, the addition is performed in an 18–bit ones complement subtractive adder. If D7 = i = 1, the addition is performed in a 24–bit ones complement subtractive adder.

The only time index register modification produces an output of –0 occurs when both inputs are –0; that is, –0 + –0 = –0.

After the indexing operation is complete for an instruction which specifies the character addressing mode and a J–register containing I = 1, if h = 1, the contents of the Iw– and Ib–fields of the J–register are added to the contents of the Ow– and Ob–fields, respectively, as explained in 4.3.2.2.2. In this case Xm is not modified.

The modification of Xm or of Ow and Ob is performed without increasing instruction execution time.

If h = 0, neither Xm nor the Ow– or Ob–field is modified; the I–bit is ignored in this case.

4.3.2.7. Use of the i–Field

The i–field can be used to specify normal addressing, indirect addressing, absolute addressing, or to extend the u–field of an instruction.

If i = 1 and D7 = 0, indirect addressing occurs for all instructions except when f = 01$_8$ through 06$_8$, 10$_8$ through 32$_8$, 34$_8$ through 36$_8$, 40$_8$ through 67$_8$, and j = 16$_8$ or 17$_8$. For the exception, x ≠ 0 is also a required condition for indirect addressing.

Indirect addressing will not occur if:

- i = 0
- f = 01$_8$ through 06$_8$, 10$_8$ through 32$_8$, 34$_8$ through 36$_8$, 40$_8$ through 67$_8$, j = 16$_8$ or 17$_8$, and x = 0. (Then the i–field is used as an extension of the u–field.)

- D7 = 1 (Then i = 1 specifies absolute addressing, i = 0 specifies normal address generation of u + Xm.)

The above cases are summarized in Table 4-10.

Table 4-10. Summary of Use of i-Field

i	D7	All Instructions	Exceptions f is less than 70 (Except 07, 33 and 37) and j=16 or 17 and:	
			x ≠ 0	x = 0
0	0	Normal Addressing	Operand is (u + Xm)	Operand is h, i, u
0	1	Normal Addressing		
1	0	Indirect Addressing	Indirect Addressing	
1	1	Absolute Addressing (Xm = 24 Bits)	Operand is (u + Xm)	

When indirect addressing is specified, it is initiated after calculating the relative address and the absolute address in the index subsection, even if $U \leqslant 177_8$. The contents of bit positions 21 through 0 of the main storage location addressed is transferred to the control section of the CPU, replacing the x-, h-, i-, and u-field values of the current instruction. The modified instruction is then performed just as if the whole instruction word were initially obtained in its modified form from main storage. Indexing and index register incrementation (if specified) are performed in the normal manner for both the original and the modified instruction. If the modified instruction also specifies indirect addressing, the whole process of indirect addressing is repeated. The repetition or cascading of indirect addressing continues until the modified instruction contains a 0 bit in the i-field, or contains all 0 bits in the x-field for the f, j combinations which lead to the use of i to extend the u-field, at which time indirect addressing ceases and the modified instruction is performed.

If $f = 01_8$ through 67_8 (except 07_8, 33_8, and 37_8), $j = 16_8$ or 17_8, and x = 0 in an instruction as it is initially obtained from main storage or as it is modified as a result of an indirect addressing operation, indirect addressing does not occur even if i = 1. In this case, the i-field is used as an extension of the u-field.

4.3.2.8. Description of the u-Field

The ultimate use of the u-field depends on the values in the f and j-fields of the instruction.

For most f, j combinations, u is used to form an operand address. The indexed extension of the value in the u-field of the instruction is used as the relative address of a main storage location or as the address of a GRS location.

For certain f, j combinations, the indexed extension of the value in the u-field of the instruction (or of a modified instruction in the case of indirect addressing) is used as the operand for some

instructions, or as a count in the case of shift instructions. For other f, j combinations, the value in the u-field has no effect on the result of the instruction.

4.3.2.8.1. Use of the u-Field as an Operand Address Designator

When the value in the u-field of an instruction is an operand address designator because of the f, j combination or the specifying of indirect addressing, the 16-bit u value is extended to 18 bits with two high order zero bits to form one input to the index adder. Xm is the other input. U, the 18-bit output of the index adder, is used as the relative address of a main storage location if $U \geqslant 200_8$.

If $U < 200_8$, U is normally used as the absolute address of a GRS location. If $U < 200_8$ and the instruction specifies indirect addressing, a jump to address, or the address for an Execute instruction (see 5.13.3), U is the relative address of a main storage location rather than the absolute address of a GRS location.

For any given u-field value, a value can be chosen for the Xm portion of the specified index register which will produce any desired value of U in the range 000000_8 through 777776_8. (It is not possible to produce the value 777777_8.)

Certain instructions use U to reference both U and U+1 as a double-length (72-bit) word. In this case, U is the address of the most significant 36 bits and U+1 is the address of the least significant 36 bits.

4.3.2.8.2. Use of the u-Field as an Operand Designator

The value in the u-field of an instruction (or a modified instruction) is an operand ingredient rather than an operand address ingredient if indirect addressing is not specified and:

- $f = 07_8$, and $j = 14_8$;
- $f = 10_8$ through 67_8 (except 33_8 and 37_8), and $j = 16_8$ or 17_8; or
- 73_8 and $j = 0_8$ through 05_8 or 10_8 through 13_8 (all shift instructions).

When the value in the u-field of an instruction (or a modified instruction resulting from an indirect addressing sequence) is an operand designator, the 16-bit value in the u-field is extended to 18 bits to provide one of the inputs to the index adder for an indexing operation. This 18-bit value normally consists of 0 bits in the two leftmost bit positions and the 16-bit value from the u-field in the remaining bit positions. If $f = 10_8$ through 67_8 (except 33_8 and 37_8), $j = 16_8$ or 17_8, and $x = 0$; however, the bits in the h and i-fields are used in the two leftmost bit positions in place of the 0 bits. When h and i are both 1 bits and they are used to extend a u-field whose value is all 1 bits, the output of the index adder is all 0 bits rather than all 1 bits.

The 18-bit index adder output is normally sent to the arithmetic section where it is extended to become a 36-bit operand by 0-bit fill ($j = 16_8$) or by filling with bits identical to the leftmost bit of the index adder output ($j = 17_8$).

4.3.2.8.3. Use of the u-Field as a Shift Count Designator

The value in the u-field of an instruction (or a modified instruction) is a shift count designator if $f = 73_8$, and $j = 0$ through 05_8 or 10 through 13_8. In these cases the 16-bit u-value is extended to 18 bits with high order zero bits and added to Xm to form the 18-bit value U. The appropriate low order bits of U are used as the shift count.

4.3.2.8.4. Restrictions on the Use of the u-Field

When indirect addressing is not specified, certain instructions require the value in the u-field to be zero. These instructions are:

- Enable Day Clock
- Disable Day Clock

If this restriction is violated, the results produced are undefined.

5. Instruction Repertoire

5.1. General

This section describes the operation performed by each instruction in the 1100/80 Systems user repertoire. These descriptions are grouped by types of instructions.

An introduction to each group presents information that is common to all instructions in the group. The detailed descriptions of the individual instruction have the following format:

- Instruction name – Mnemonic code Octal function code
- Symbolic description of the operation performed by the instruction. The symbology used is defined in Appendix A.
- Textual description of the operation performed by the instruction.
- Sequentially numbered notes which provide special information related to the instruction, if appropriate.

For all instructions, any possible value may be used in the a–, x–, h–, i–, and u–fields unless an exception to this rule is stated in the notes. Any possible value may be used in the j–field except when j is a minor–function–code designator or when an exception is stated in the notes.

If the value of the j–field is 016 and 017 (an immediate operand specification) and the value of the x–field is zero, the h–bit, i–bit, and u–field make up the 18–bit operand. If the h– and i–bits are one and the value of the u–field is 0177777, however, the resulting operand is zero, not all ones. A negative zero can be generated as an immediate operand only by load negative instructions using x–, h–, i–, and u–fields of zero.

If the value of the a–field of the instruction is 017 (A15) and the instruction makes use of more than one arithmetic register (A+1 or A+2), those registers are located at general register stack (GRS) location 034 and 035, or 0174 and 0175, depending on the value of the GRS selection designator (D6). If automatic index register incrementation occurs, the value of Aa or Xa is not affected. The value of U or U+1 (if U < 0200) or A+1 (for two pass instructions, which require both U and U+1) may be affected; however, if Xx is referenced as one of these operands, the updated index value is used.

5.2. Load Instructions

The single-precision load instructions transfer data to the arithmetic section where a 36-bit word is always formed. The 36-bit word is then transferred to the register specified by the a-field of the instruction. Single-precision data-word transfers from storage to the arithmetic section are controlled by the value in the j-field.

For the double-precision load instructions, the j-field is a minor function code and full 72-bit data transfers result.

5.2.1. Load A - L,LA 10

(U) → A

The contents of U is transferred under j-field control to the arithmetic section and then to Aa.

5.2.2. Load Negative A - LN,LNA 11

-(U) → A

The contents of U is transferred under j-field control to the arithmetic section. The ones complement of the value in the arithmetic section is transferred to Aa.

5.2.3. Load Magnitude A - LM,LMA 12

| (U) | → A

The contents of U is transferred under j-field control to the arithmetic section. If the sign bit (bit 35) of the value in the arithmetic section is a 1 bit, it is complemented; if the sign bit is a 0 bit, it is not complemented. The final value (always positive) is transferred from the arithmetic section to Aa.

For j-field values 0, 3-7 (quarter word not set) and 17, sign bit 35 is controlled by sign extension.

1. This instruction is the same as Load A (see 5.2.1) for j = H1, H2, Q1-Q4, or S1-S6.

5.2.4. Load Negative Magnitude A - LNMA 13

-| (U) | → A

The contents of U is transferred under j-field control to the arithmetic section. If the sign bit (bit 35) of the value in the arithmetic section is a 0 bit, it is complemented; if the sign bit is a 1 bit, it is not complemented. The final value (always negative) is transferred from the arithmetic section to Aa.

For j-field values 3-7 (quarter word not set) and 17, sign bit 35 is controlled by sign extension.

1. This instruction may be used to load -0 into an A-register by using j = 16_8 or 17_8, and x = h = i = u = 0.

2. This instruction is the same as Load Negative A (see 5.2.2) for j = H1, H2, Q1-Q4, or S1-S6.

5.2.5. Load R - L,LR 23

(U) → Ra

The contents of U is transferred under j-field control to the arithmetic section and then to the Ra-register specified by the a-field.

1. If the processor is in user mode, an attempt to Load R0 causes a Guard Mode interrupt.

5.2.6. Load X Modifier - LXM 26

(U) → Xa_{17-0}; Xa_{35-18} unchanged

The contents of U is transferred under j-field control to the arithmetic section; the low-order 18 bits of the value in the arithmetic section is transferred to the lower half (bits 17-0) of the X-register specified by the a-field; the upper half (bits 35 through 18) of the X-register remains unchanged.

1. This instruction loads only the low-order 18 bits of the specified X-register, even if the relocation and storage suppression designator (D7) = i = 1 to specify 24-bit indexing.

5.2.7. Load X - L,LX 27

(U) → Xa

The contents of U is transferred under j-field control to the arithmetic section and then to the X-register specified by the a-field.

5.2.8. Load X Increment - LXI 46

(U) → Xa_{35-18}; Xa_{17-0} unchanged

The contents of U is transferred under j-field control to the arithmetic section; the low-order 18 bits of the value in the arithmetic section are transferred to the upper half (bits 35-18) of the X-register specified by the a-field. The lower half (bits 17-0) of the X-register remains unchanged.

1. This instruction loads the full high-order 18 bits of the specified X-register, even if the relocation and storage suppression designator (D7) = i = 1 to specify 24-bit indexing.

5.2.9. Double Load A - DL 71,13

(U,U+1) → A,A+1

The contents of U and U+1 are transferred to the arithmetic section and then to Aa and Aa+1, respectively.

5.2.10. Double Load Negative A - DLN 71,14

-(U,U+1) → A,A+1

The contents of U and U+1 are transferred to the arithmetic section where the 72-bit value is complemented and then transferred to Aa and Aa+1, respectively.

5.2.11. Double Load Magnitude A - DLM 71,15

$|(U,U+1)| \rightarrow A,A+1$

The contents of U and U+1 are transferred to the arithmetic section. If the sign bit (bit 35) of U is a 1 bit, the 72-bit value in the arithmetic section is complemented; if the sign bit is a 0 bit, the 72-bit value is not complemented. The final value (always positive) is transferred from the arithmetic section to Aa and Aa+1.

5.3. Store Instructions

The single-length store instructions transfer data from a control register specified by the a-field to the storage location or control register addressed by U. Exceptions to this are the Store Constant instructions. (See 5.3.5.)

Single-length data-word transfers to storage are controlled by the j-field. If $j = 16_8$ or 17_8, no data is stored. A Guard Mode interrupt will occur, however, if $U < 200_8$ and an Executive register is specified in user mode, or if $U \geqslant 0200_8$ and a storage-limits or write-protection violation occurs.

Indexing, index incrementation/decrementation, and indirect addressing function normally in all cases.

5.3.1. Store A - S,SA 01

$(A) \rightarrow U$

The contents of Aa is transferred under j-field control to location U.

1. If $j = 16_8$ or 17_8, no data is stored.

5.3.2. Store Negative A - SN,SNA 02

$-(A) \rightarrow U$

The complement of the value of Aa is transferred under j-field control to location U.

1. If $j = 16_8$ or 17_8, no data is stored.

5.3.3. Store Magnitude A - SM,SMA 03

$|(A)| \rightarrow U$

If the sign bit (bit 35) of the value of Aa is one, the value is complemented. The final value (always positive) is transferred under j-field control to location U.

1. If $j = 16_8$ or 17_8, no data is stored.

5.3.4. Store R - S,SR 04

(Ra) → U

The contents of the R-register specified by the a-field is transferred under j-field control to location U.

1. If j = 16_8 or 17_8, no data is stored.

5.3.5. Store Constant Instructions - XX 05; a = 00-07

Constant → U

A constant value specified by the a-field is transferred under j-field control to location U. The following octal constant values may be stored:

SZ	a = 0	000000 000000	Zero
SNZ	a = 1	777777 777777	Ones
SP1	a = 2	000000 000001	Plus One
SN1	a = 3	777777 777776	Minus One
SFS	a = 4	050505 050505	Fieldata Blanks
SFZ	a = 5	606060 606060	Fieldata Zeros
SAS	a = 6	040040 040040	ASCII Blanks
SAZ	a = 7	060060 060060	ASCII Zeros

5.3.6. Store X - S,SX 06

(Xa) → U

The contents of the X-register specified by the a-field is transferred under j-field control to location U.

1. If j = 16_8 or 17_8, no data is stored.

5.3.7. Double Store A - DS 71,12

(A,A+1) → U,U+1

The contents of Aa and Aa+1 are transferred to locations U and U+1, respectively.

5.3.8. Block Transfer - BT 22

(Xx + u) → Xa + u, repeat k times; k = the initial count in the repeat count register

A source word is transferred under j-field control to the arithmetic section, and then under j-field control to a destination word-location. The repeat count is decreased by 1. The source-to-destination transfer step is repetitively performed until the repeat count has been decreased to 0. The x-field specifies the X-register used with the u-field to determine the effective source word-address. The a-field specifies the X-register used in determining the effective destination word-address.

1. A word containing the desired repeat count in the rightmost 18-bit positions must be loaded in the repeat count register (R1) before performing the Block Transfer instruction.

2. If the initial repeat count is ±0, no data is transferred. If -0, then +0 is written into bits 17-0 of the repeat count.

3. If j = 16_8 or 17_8, no data is transferred; however, the repeat count is decreased to zero.

4. If the x-field is zero, no data is transferred. The contents of the X-register specified by the a-field remain unchanged, regardless of the contents of the a- and h-fields.

5. If an interrupt occurs before the repeat count has decreased to zero, the termination pass occurs at the conclusion of the currently active data transfer. The remnant repeat count is stored in R1. When the interrupt is honored, the captured P value is the address of the Block Transfer instruction or the address of the Execute instruction which led to the Block Transfer instruction. Thus, this address can be preserved and, when the interrupt has been processed, it is possible to return to the Block Transfer instruction and continue executing this instruction at the point where it was terminated for the interrupt. If the Block Transfer instruction was entered by means of an Execute instruction, the h-field of the Execute instruction must be zero so that, when the program returns to the Execute instruction, the effective U address will again lead to the Block Transfer instruction. If the Block Transfer instruction specifies indirect addressing (i = 1), the h-field must be zero to enable the program to return to the same effective U address and complete the Block Transfer instruction in the event of an interrupt.

6. If there is no indirect addressing (i = 0), the h-field is normally one. If h = 0, no incrementation/decrementation of the index registers occurs. When h = 0, the source and destination addresses are the initial contents of the index registers used repetitively for every transfer performed. Thus, no more than one data transfer is effectively performed.

7. If the x-field is not zero, but the a-field is zero, the a-field references index register zero (X0), and proper operation occurs.

5.4. Fixed-Point Arithmetic Instructions

The fixed-point arithmetic instructions perform integer or fractional addition, subtraction, multiplication, and division. In a single-precision arithmetic instruction, the transfer of data from location U in storage to the arithmetic section is under the control of the contents of the j-field of the instruction. For double-precision and parallel half-word and third-word arithmetic operations, the value in the j-field is a minor function code.

For all arithmetic instructions, indexing, index incrementation/decrementation, and indirect addressing function normally.

The overflow and carry designators are set according to the results of the operation for all add and add-negative instructions except add and add-negative halves and thirds.

The sign of the result is determined by the rules of algebra except for add and add-negative instructions where both operands are zero. In this case, the result is positive zero, except for add instructions where both operands are negative zero, and add-negative instructions where the minuend (Aa) is negative zero and the subtrahend (U) is positive zero.

5.4.1. Add to A - A,AA 14

(A) + (U) → A

The contents of U is transferred under j-field control to the arithmetic section. The 36-bit value in the arithmetic section is added algebraically to the contents of Aa. The sum is stored in Aa.

5.4.2. Add Negative to A - AN,ANA 15

(A) - (U) → A

The contents of U is transferred under j-field control to the arithmetic section. The 36-bit value in the arithmetic section is subtracted algebraically from the contents of Aa. The difference is stored in Aa.

5.4.3. Add Magnitude to A - AM,AMA 16

(A) + | (U) | → A

The contents of U is transferred under j-field control to the arithmetic section. If the sign bit (bit 35) of the 36-bit value in the arithmetic section is one, the value is complemented; if the sign bit is zero, the value is not complemented. The final 36-bit value in the arithmetic section (always positive) is added algebraically to the contents of Aa. The sum is stored in Aa.

Only valid for j = 3-7, 17.

1. This instruction is the same as Add to A (see 5.4.1) for j = H1, H2, Q1-Q4, or S1-S6.

5.4.4. Add Negative Magnitude to A - ANM,ANMA 17

(A) - | (U) | → A

The contents of U is transferred under j-field control to the arithmetic section. If the sign bit (bit 35) of the 36-bit value in the arithmetic section is one, the value is complemented; if the sign bit is zero, the value is not complemented. The final 36-bit value in the arithmetic section (always positive) is subtracted algebraically from the contents of Aa. The difference is stored in Aa.

Only valid for j = 3-7, 17.

1. This instruction is the same as Add Negative to A (see 5.4.2) for j = H1, H2, Q1-Q4, or S1-S6.

5.4.5. Add Upper - AU 20

(A) + (U) → A+1

The contents of U is transferred under j-field control to the arithmetic section. The 36-bit value in the arithmetic section is added algebraically to the contents of Aa. The sum is stored in Aa+1. The contents of U and Aa remain unchanged.

5.4.6. Add Negative Upper - ANU 21

(A) - (U) → A+1

The contents of U is transferred under j-field control to the arithmetic section. The 36-bit value in the arithmetic section is subtracted algebraically from the contents of Aa. The difference is stored in Aa+1. The contents of U and Aa remain unchanged.

5.4.7. Add to X - A,AX 24

(Xa) + (U) → Xa

The contents of U is transferred under j-field control to the arithmetic section. The 36-bit value in the arithmetic section is added algebraically to the contents of the X-register specified by the a-field. The sum is stored in the X-register specified by the a-field.

5.4.8. Add Negative to X - AN,ANX 25

(Xa) - (U) → Xa

The contents of U is transferred under j-field control to the arithmetic section. The 36-bit value in the arithmetic section is subtracted algebraically from the contents of the X-register specified by the a-field. The difference is stored in the X-register specified by the a-field.

5.4.9. Multiply Integer - MI 30

(A) x (U) → A,A+1

The contents of U is transferred under j-field control to the arithmetic section. The contents of Aa is multiplied algebraically by the 36-bit value in the arithmetic section, producing a 72-bit product. The most significant 36 bits of the product (including sign bits) are stored in Aa. The least significant 36 bits of the product are stored in Aa+1.

1. Bit positions 71 and 70 of the product are always sign bits. The product of any two 35-bit positive integers cannot exceed a 70-bit positive integer.

5.4.10. Multiply Single Integer - MSI 31

(A) x (U) → A

The contents of U is transferred under j-field control to the arithmetic section. The contents of Aa is multiplied algebraically by the 36-bit value in the arithmetic section, producing a 72-bit product. The least significant 36 bits of the product are stored in Aa. The most significant 36 bits of the product are lost.

1. The 36-bit result stored in Aa does not represent the product as a signed number if the leftmost 37 bits of the 72-bit product formed in the arithmetic section are not identical.

5.4.11. Multiply Fractional - MF 32

(A) x (U) → A,A+1

The contents of U is transferred under j-field control to the arithmetic section. The contents of Aa is multiplied algebraically by the 36-bit value in the arithmetic section, producing a 72-bit product which is shifted left circularly one bit position. The leftmost 36 bits of the shifted product, including the sign bit, are stored in Aa. The rightmost 36 bits are stored in Aa+1.

1. This instruction performs an operation identical to the Multiply Integer instruction (see 5.4.9) except that the 72-bit result of the multiplication process is shifted left circularly one bit position prior to storing it in Aa and Aa+1.

2. The rightmost bit of the result in Aa+1 is a sign bit and it is identical to the leftmost bit of the result in Aa.

5.4.12. Divide Integer - DI 34

(A,A+1) ÷ (U) → A; remainder → A+1

The contents of U is transferred under j-field control to the arithmetic section. The 72-bit signed number in Aa and Aa+1 is divided algebraically by the 36-bit value in the arithmetic section. The 36-bit signed quotient is stored in Aa. The remainder retains the sign of the dividend (the leftmost bit of the initial contents of Aa) and is stored in Aa+1.

1. The absolute value of the 72-bit signed dividend (Aa,Aa+1) should be less than the absolute value of the divisor (j-determined portion of U) multiplied by 2^{35}. If this relationship is not satisfied and the arithmetic exception interrupt designator (D20) is zero, Aa and Aa+1 are cleared to zero and the divide check designator (D23) is set to one. If this relationship is not satisfied and D20 is one, Aa and Aa+1 remain unchanged, the divide check designator (D23) is set to one, and a Divide Check interrupt results. This includes the case in which the divisor equals zero.

5.4.13. Divide Single Fractional - DSF 35

(A) ÷ (U) → A+1

The contents of U is transferred under j-field control to the arithmetic section. The contents of Aa is divided algebraically by the 36-bit value in the arithmetic section. The 36-bit signed quotient is stored in Aa+1. The remainder is lost. The contents of Aa remains unchanged.

1. The absolute value of the dividend (Aa) should be less than the absolute value of the divisor (j-determined portion of U). If this relationship is not satisfied and the arithmetic exception interrupt designator (D20) is zero, Aa+1 is cleared to zero and the divide check designator (D23) is set to one. If this relationship is not satisfied and D20 is one, Aa+1 remains unchanged, D23 is set to one, and a Divide Check interrupt results. This includes the case in which the divisor equals zero.

2. This instruction performs an operation like that of divide integer, except that the quotient appears to be shifted one bit to the right.

5.4.14. Divide Fractional - DF 36

$(A,A+1) \div (U) \rightarrow A$; remainder $\rightarrow A+1$

The contents of U is transferred under j-field control to the arithmetic section. The 72-bit signed number in Aa and Aa+1 is divided algebraically by the 36-bit value in the arithmetic section. The 36-bit signed quotient is stored in Aa. The remainder retains the sign of the dividend (the leftmost bit of the original contents of Aa) and is stored in Aa+1.

1. The absolute value of the leftmost half of the dividend (Aa) should be less than the absolute value of the divisor (j-determined portion of U). If this relationship is not satisfied and the arithmetic exception interrupt designator (D20) is zero, Aa and Aa+1 are cleared to zero and the divide check designator (D23) is set to one. If this relationship is not satisfied and D20 is one, Aa and Aa+1 remain unchanged, D23 is set to one, and a Divide Check interrupt results. This includes the case in which the divisor equals zero.

2. This instruction performs an operation identical to divide integer, except that the quotient appears to be shifted one bit to the right.

5.4.15. Double-Precision Fixed-Point Add - DA 71,10

$(A,A+1) + (U,U+1) \rightarrow A,A+1$

The 72-bit signed number from U and U+1 is added algebraically to the 72-bit signed number from Aa and Aa+1. The 72-bit sum is stored in Aa and Aa+1.

5.4.16. Double-Precision Fixed-Point Add Negative - DAN 71,11

$(A,A+1) - (U,U+1) \rightarrow A,A+1$

The 72-bit signed number from U and U+1 is subtracted algebraically from the 72-bit signed number from Aa and Aa+1. The 72-bit difference is stored in Aa and Aa+1.

5.4.17. Add Halves - AH 72,04

$(A)_{35-18} + (U)_{35-18} \rightarrow A_{35-18}$;

$(A)_{17-0} + (U)_{17-0} \rightarrow A_{17-0}$

The contents of each half (18-bit portion) of U is added algebraically to the contents of the corresponding half of Aa. The sums are stored in the corresponding halves of Aa.

1. There is no interaction between the upper and lower halves of the operands. A carry from bit position 17 is propagated to bit 0, rather than bit 18. A carry from bit position 35 is propagated to bit 18, rather than bit 0.

5.4.18. Add Negative Halves - ANH 72,05

$(A)_{35-18} - (U)_{35-18} \rightarrow A_{35-18}$;

$(A)_{17-0} - (U)_{17-0} \rightarrow A_{17-0}$

The contents of each half (18-bit portion) of U is subtracted algebraically from the contents of the corresponding half of Aa. The differences are stored in the corresponding halves of Aa.

1. There is no interaction between the upper and lower halves of the operands. A borrow from bit position 17 is propagated to bit 0, rather than bit 18. A borrow from bit position 35 is propagated to bit 18, rather than bit 0.

5.4.19. Add Thirds - AT 72,06

$(A)_{35-24} + (U)_{35-24} \rightarrow A_{35-24}$;

$(A)_{23-12} + (U)_{23-12} \rightarrow A_{23-12}$;

$(A)_{11-0} + (U)_{11-0} \rightarrow A_{11-0}$

The contents of each third (12-bit portion) of U is added algebraically to the contents of the corresponding third of Aa. The sums are stored in the corresponding thirds of Aa.

1. A carry from bit position 11, 23, or 35 is propagated to bit 0, 12, or 24, respectively, rather than to bit 12, 24, or 0.

5.4.20. Add Negative Thirds - ANT 72,07

$(A)_{35-24} - (U)_{35-24} \rightarrow A_{35-24}$;

$(A)_{23-12} - (U)_{23-12} \rightarrow A_{23-12}$;

$(A)_{11-0} - (U)_{11-0} \rightarrow A_{11-0}$

The contents of each third (12-bit portion) of U is subtracted algebraically from the contents of the corresponding third of Aa. The differences are stored in the corresponding thirds of Aa.

1. A borrow from bit position 11, 23, or 35 is propagated to bit 0, 12, or 24, respectively, rather than to bit 12, 24, or 0.

5.5. Floating-Point Arithmetic Instructions

Floating-point arithmetic operations allow for efficient computation involving numerical data with a wide range of magnitudes. Indexing, index incrementation/decrementation, and indirect addressing function normally in all floating-point arithmetic instructions.

The greatest precision is obtained in floating-point arithmetic operations when the floating-point input operands are normalized numbers. Certain floating-point operations produce undefined results if normalized input operands are not used. The supporting notes indicate which instructions are affected.

5.5.1. Floating Add - FA 76,00

$(A) + (U) \rightarrow A$; residue $\rightarrow A+1$ if D17 = 1

The single-precision floating-point number from location U is added to the single-precision floating-point number from Aa. The resulting sum is normalized and then stored in single-precision

floating-point format in Aa. If the floating-point residue store enable designator (D17) = 1, the residue in single-precision floating-point format is stored in Aa+1.

1. The result stored in Aa is a normalized number, even if either or both of the input operands are not normalized. No attempt is made to normalize the residue stored in Aa+1.

2. A Floating-Point Characteristic Overflow/Underflow interrupt may occur.

3. If the mantissa of the most significant word of the result is ±0, the word stored depends on the floating-point zero format selection designator (D8).

4. The sign of the most significant word of the result is the sign of the large input operand. The sign of the other operand is assigned to the residue.

5.5.2. Floating Add Negative - FAN 76,01

(A) - (U) → A; residue → A+1 if D17 = 1

The single-precision floating-point number from location U is subtracted from the single-precision floating-point number from Aa. The resulting difference is normalized and then stored in single-precision floating-point format in Aa. If the floating-point residue store enable designator (D17) = 1, the residue in single-precision floating-point format is stored in Aa+1.

1. The result stored in Aa is a normalized number, even if either or both of the input operands are not normalized. No attempt is made to normalize the residue stored in Aa+1.

2. A Floating-Point Characteristic Overflow/Underflow interrupt may occur.

3. If the mantissa of the most significant word of the result is ±0, the word stored depends on the floating-point zero format selection designator (D8).

4. The Floating Add Negative operation is identical to the Floating Add operation described in 5.5.1, except that the ones complement of the contents of location U is used as the second operand.

5. The sign of the most significant word of the result is the sign of the large input operand. The sign of the other operand is assigned to the residue.

5.5.3. Double-Precision Floating Add - DFA 76,10

(A,A+1) + (U,U+1) → A,A+1

The double-precision floating-point number from locations U and U+1 are added to the double-precision floating-point number from Aa and Aa+1. The resulting sum is normalized and then stored in double-precision floating-point format in Aa and Aa+1.

1. The result stored is a normalized number, even if either or both of the input operands are not normalized.

2. A Floating-Point Characteristic Overflow/Underflow interrupt may occur.

3. If the exponent value of the sum is less than -1024 and the double-precision underflow designator (D5) and the arithmetic exception interrupt designator (D20) are one, a Floating-Point Characteristic Underflow interrupt does not occur. Instead, +0 is stored in Aa and Aa+1. If D20 is zero, D5 is ignored.

4. If the mantissa produced is floating-point zero, the result stored is +0 regardless of the signs and characteristics of the input operands.

5.5.4. Double-Precision Floating Add Negative - DFAN 76,11

(A,A+1) - (U,U+1) → A,A+1

The double-precision floating-point number from locations U and U+1 are subtracted from the double-precision floating-point number from Aa and Aa+1. The resulting difference is normalized and then stored in double-precision floating-point format in Aa and Aa+1.

1. The result stored is a normalized number, even if either or both of the input operands are not normalized.

2. A Floating-Point Characteristic Overflow/Underflow interrupt may occur.

3. If the exponent value of the difference is less than -1024 and the double-precision underflow designator (D5) and the arithmetic exception interrupt designator (D20) are one, a Floating-Point Characteristic Underflow interrupt does not occur. Instead, +0 is stored in Aa and Aa+1. If D20 is one and D5 is zero, the interrupt occurs. If D20 is zero, D5 is ignored.

4. The Double-Precision Floating Add Negative operation is identical to the Double-Precision Floating Add process described in 5.5.3, except that the ones complement of the contents of U and U+1 is used as the second operand.

5. If the mantissa produced is floating-point zero, the result stored is +0, regardless of the signs and characteristics of the input operands.

5.5.5. Floating Multiply - FM 76,02

(A) x (U) → A (and A+1 if D17 = 1)

The single-precision floating-point number from Aa is multiplied by the single-precision floating-point number from location U. The resulting double-length product is packed into two single-precision floating-point numbers. The most significant portion of the product in single-precision floating-point format is stored in Aa. If the floating-point residue store enable designator (D17) = 1, the least significant portion of the product in single-precision floating-point format is stored in Aa+1.

1. If either or both input operands are not normalized numbers, the results are undefined. The following notes apply only if both input operands are normalized numbers.

2. A Floating-Point Characteristic Overflow/Underflow interrupt may occur.

3. The portion of the product stored in Aa is a normalized number. No attempt is made to normalize the number stored in Aa+1.

4. The algebraic rule for signs applies to the portions of the product stored in Aa and Aa+1.

5. If the mantissa of either or both input operands is zero, the following applies:

   a. A Floating-Point Characteristic Overflow/Underflow interrupt never occurs, regardless of the values of the characteristics of the input operands.

b. If the floating-point zero format selection designator (D8) is zero, the result stored in Aa is +0, regardless of the signs of the input operands.

c. If D8 is one and if the exponent value is in the range -128 through +127, the most significant product-word will reflect the magnitude of the characteristic produced and the sign produced by the mantissa arithmetic.

d. If the exponent value of the most significant product-word is greater than +127 or less than -128, the result stored in Aa is ±0, whichever would reflect the signs of the input operands.

6. The value of D8 has no effect on the least significant product-word. When the mantissa for the least significant product-word is zero, it is packed with the appropriate characteristic. If the characteristic of the residue is less than -128, the result stored in Aa+1 is ±0, whichever would reflect the signs of the operands.

   A characteristic overflow of the most significant word can occur; however, the characteristic of the residue could be in the range 000 through 377. In this case, the result stored in Aa is ±0 depending on the algebraic rule of the sign, and the residue is packed with the appropriate characteristic and stored in Aa+1.

7. If the characteristic of the number stored in Aa is greater than or equal to 27, the characteristic of the number stored in Aa+1 is 27 less than the characteristic in Aa.

5.5.6. Double-Precision Floating Multiply - DFM 76,12

(A,A+1) x (U,U+1) → A,A+1

The double-precision floating-point number from Aa and Aa+1 is multiplied by the double-precision floating-point number from locations U and U+1. The product is normalized and stored in double-precision floating-point format in Aa and Aa+1.

1. If either or both input operands are not normalized numbers, the results are undefined. The following notes apply only if both operands are normalized numbers.

2. A Floating-Point Characteristic Overflow/Underflow interrupt may occur.

3. The result stored in Aa and Aa+1 is always a normalized number.

4. The algebraic rule for signs applies except for the special cases covered in notes 5b and 6.

5. If the mantissa of either or both input operands are zero, the following applies:

   a. A Floating-Point Characteristic Overflow/Underflow interrupt never occurs, regardless of the values of the characteristics of the input operands.

   b. The result stored in Aa and Aa+1 is +0 regardless of the signs of the input operands.

6. If the exponent value of the product is less than -1024 and the double-precision underflow designator (D5) and the arithmetic exception interrupt designator (D20) are one, a Floating-Point Characteristic Underflow interrupt does not occur. Instead +0, regardless of the signs of the input operands, is stored in Aa and Aa+1. If D20 = 1 and D5 = 0, the interrupt occurs. If D20 = 0, D5 is ignored.

5.5.7. Floating Divide - FD 76,03

(A) ÷ (U) → A; remainder → A+1 if D17 = 1

The single-precision floating-point number from Aa is divided by the single-precision floating-point number from location U. The quotient is stored in Aa in single-precision floating-point format. If the floating-point residue store enable designator (D17) = 1, the remainder is stored in Aa+1 in single-precision floating-point format.

1. If either or both input operands are not normalized numbers, the results are not defined. The following notes apply only if both operands are normalized numbers.

2. A Floating-Point Characteristic Overflow/Underflow interrupt may occur.

3. If the mantissa of the divisor (U) is zero, a Divide Check interrupt occurs.

4. The quotient stored in Aa is a normalized number. No attempt is made to normalize the remainder that is stored in Aa+1 when D17 = 1.

5. The algebraic rule for signs applies to the quotient stored in Aa. The sign of the dividend is assigned to the remainder stored in Aa+1.

6. If the mantissa of the dividend (Aa) is zero but not the divisor (U), the following applies:

   a. A Floating-Point Characteristic Overflow/Underflow interrupt never occurs, regardless of the characteristics of the operands.

   b. If the floating-point zero format selection designator (D8) = 0, the quotient stored in Aa is +0, regardless of the signs of the operands.

   c. If D8 = 1 and the exponent value of the quotient is greater than +128 or less than -128, the quotient stored in Aa is ±0, whichever would reflect the signs of the input operands.

7. If the exponent value of the remainder is less than -128, the remainder stored in Aa+1 is ±0, whichever would reflect the sign of the dividend from Aa.

8. If the characteristic of the dividend from Aa is greater than or equal to 27, then the characteristic of the number stored in Aa+1 for the remainder is 27 or 26 less than the characteristic of the dividend.

5.5.8. Double-Precision Floating Divide - DFD 76,13

(A,A+1) ÷ (U,U+1) → A,A+1

The double-precision floating-point number from Aa and Aa+1 is divided by the double-precision floating-point number from locations U and U+1. The quotient is stored in Aa and Aa+1 in double-precision floating-point format. The remainder is not retained.

1. If either or both of the input operands are not normalized numbers, the results are undefined. The following notes apply only if both operands are normalized numbers.

2. A Floating-Point Characteristic Overflow/Underflow interrupt may occur.

3. If the mantissa of the divisor is zero, a Divide Check interrupt occurs.

4. The result stored in Aa and Aa+1 is always a normalized number.

5. The algebraic rule for signs applies, except for the special cases explained in notes 6b and 7.

6. If the dividend mantissa (Aa, Aa+1) is zero and the divisor mantissa (U,U+1) is not zero, the following applies:

   a. A Floating-Point Characteristic Overflow/Underflow interrupt never occurs, regardless of the values of the characteristics of the input operands.

   b. The result stored in Aa and Aa+1 is +0, regardless of the signs of the operands.

7. If the exponent value of the quotient is less than -1024, and the double-precision underflow designator (D5) and the arithmetic exception interrupt designator (D20) are one, a Floating-Point Characteristic Underflow interrupt does not occur. Instead +0, regardless of the signs of the input operands, is stored in Aa and Aa+1. If D20 = 1 and D5 = 0, the interrupt occurs. If D20 = 0, D5 is ignored.

5.5.9. Load and Unpack Floating - LUF 76,04

$|(U)|_{34-27} \rightarrow A_{7-0}$, zero fill;

$(U)_{26-0} \rightarrow A+1_{26-0}$, sign fill

The single-precision floating-point number from location U is transferred to the arithmetic section and unpacked. The absolute value of the biased characteristic of the input operand is transferred to bits 7 through 0 of Aa; bits 35 through 8 of the Aa are filled with 0 bits. The mantissa of the input operand is transferred to bits 26 through 0 of Aa+1; bits 35 through 27 of Aa+1 are filled with bits identical to the sign of the floating-point number in U.

1. No attempt is made to normalize the operand.

5.5.10. Double Load and Unpack Floating - DFU 76,14

$|(U,U+1)_{70-60}| \rightarrow A_{10-0}$, zero fill;

$(U,U+1)_{59-36} \rightarrow A+1_{23-0}$, sign fill;

$(U,U+1)_{35-0} \rightarrow A+2$

The double-precision floating-point number from locations U and U+1 is transferred to the arithmetic section and unpacked. The absolute value of the biased characteristic of the input operand is transferred to bits 10 through 0 of Aa; bits 35 through 11 of Aa are filled with 0 bits. The leftmost 24 bits of the mantissa, $(U)_{23-0}$, are transferred to bits 23 through 0 of Aa+1; bits 35 through 24 of Aa+1 are filled with bits identical to the sign of the floating-point number in locations U and U+1. The rightmost 36 bits of the mantissas (U+1) are transferred to Aa+2.

1. No attempt is made to normalize the operand.

5.5.11. Load and Convert to Floating - LCF 76,05

$(U)_{35} \rightarrow A+1_{35}$; [normalized $(U)]_{26-0} \rightarrow A+1_{26-0}$;

if $(U)_{35} = 0$: $(A)_{7-0} \pm$ normalizing count $\rightarrow A+1_{34-27}$;

if $(U)_{35} = 1$: ones complement of $[(A)_{7-0} \pm$ normalizing count$] \rightarrow A+1_{34-27}$

The fixed-point number from location U is sent to the arithmetic section where it is shifted right or left, as required, to normalize it. The normalizing shift count is added to the characteristic from the rightmost eight bits of Aa if a normalizing right-shift is required. It is subtracted from the characteristic if a normalizing left-shift is required. The adjusted characteristic (complemented if U is negative) is packed with the normalized value from U to form a single-precision floating-point number. The packed result is stored in Aa+1. The contents of Aa remain unchanged.

1. A Floating-Point Characteristic Overflow/Underflow interrupt may occur.
2. The 28 leftmost bits from Aa are ignored: $(Aa)_{7-0}$ must be prebiased.
3. If the resultant mantissa is zero, the following applies:
   a. If the floating-point zero format selection designator (D8) = 0, the result stored in Aa is +0.
   b. If D8 = 1, and the resultant characteristic is in the range 000 through 377, the characteristic is packed with the zero mantissa and stored in Aa.
   c. If D8 = 1, and the resultant characteristic is a negative number, ±0 is stored in Aa, depending on the sign of the input operand.

5.5.12. Double Load and Convert to Floating - DFP, DLCF 76,15

$(U)_{35} \rightarrow A+1_{35}$; [normalized $(U,U+1)]_{59-0} \rightarrow A+1_{23-0}$ and A+2;

if $(U)_{35} = 0$: $(A)_{10-0} \pm$ normalizing count $\rightarrow A+1_{34-24}$;

if $(U)_{35} = 1$: ones complement of $[(A)_{10-0} \pm$ normalizing count$] \rightarrow A+1_{34-24}$

The double-precision fixed-point number from locations U and U+1 is sent to the arithmetic section where it is shifted right or left, if necessary, to normalize it. The normalizing shift count is added to the characteristic from the rightmost 11 bits of Aa if a normalizing right-shift is required. It is subtracted from the characteristic if a normalizing left-shift is required. The adjusted characteristic (complemented if U is negative) is packed with the normalized value from U and U+1 to form a double-precision floating-point number, and the packed result is stored in Aa+1 and Aa+2. The contents of Aa remain unchanged.

1. A Floating-Point Characteristic Overflow/Underflow interrupt may occur.
2. The 25 leftmost bits from Aa are ignored; $(Aa)_{10-0}$ must be prebiased.
3. If the 72-bit input operand from U and U+1 is ±0, the result stored is +0, regardless of the sign of the 72-bit operand.
4. If the adjusted characteristic represents a negative number when the arithmetic exception interrupt designator (D20) and the double-precision underflow designator (D5) = 1, a

Floating-Point Characteristic Underflow interrupt does not occur. Instead +0, regardless of the sign of the 72-bit operand, is stored.

5.5.13. Floating Expand and Load - FEL 76,16

If $(U)_{35} = 0$: $(U)_{35-27} + 1600_8 \rightarrow A_{35-24}$;

if $(U)_{35} = 1$: $(U)_{35-27} - 1600_8 \rightarrow A_{35-24}$;

$(U)_{26-3} \rightarrow A_{23-0}$;

$(U)_{2-0} \rightarrow A+1_{35-33}$;

$(U)_{35} \rightarrow A+1_{32-0}$

The single-precision floating-point input operand from location U is transferred to the arithmetic section. The three fields of the operand are expanded to form a double-precision floating-point number as follows:

- The sign bit is stored in bits 71 and 32 through 0.
- The 8-bit characteristic, which includes a bias of 200_8, is modified to an 11-bit characteristic. which includes a bias of 2000_8, and is stored in bits 70 through 60.
- The 27-bit mantissa is stored in bits 59 through 33.

The result is transferred to Aa and Aa+1.

1. If the operand is not in the normalized single-precision floating-point format, the result stored is undefined. The following notes apply only if the input operand is a normalized number.
2. If the mantissa of the input operand is ±0, the result stored in Aa and Aa+1 is +0, regardless of the sign of the operand.
3. A Floating-Point Characteristic Overflow/Underflow interrupt will not occur as a result of this instruction.

5.5.14. Floating Compress and Load - FCL 76,17

If $(U)_{35} = 0$: $(U)_{35-24} - 1600_8 \rightarrow A_{35-27}$;

if $(U)_{35} = 1$: $(U)_{35-24} + 1600_8 \rightarrow A_{35-27}$;

$(U)_{23-0} \rightarrow A_{26-3}$;

$(U+1)_{35-33} \rightarrow A_{2-0}$

The double-precision floating-point operand from locations U and U+1 is transferred to the arithmetic section. The three fields of the operand are compressed to form a single-precision floating-point number as follows:

- The sign bit is stored in bit 35.
- The 11-bit characteristic, which includes a bias of 2000_8, is modified to an 8-bit characteristic, which includes a bias of 200_8, and is stored in bits 34 through 27.
- The 27 leftmost bits of the mantissa (bits 23 through 0 from location U, and bits 35 through 33 from location U+1) are stored in bits 26 through 0.

The result is transferred to Aa.

1. The following notes apply only if the operand is a normalized number.
2. If the arithmetic exception interrupt designator (D20) = 1, a Floating-Point Characteristic Overflow interrupt occurs if the characteristic of the operand is greater than +127, and a Floating-Point Characteristic Underflow interrupt occurs if the characteristic of the operand is less than -128. The characteristic underflow designator (D21) is set when an underflow condition is detected, and the characteristic overflow designator (D22) is set when an overflow condition is detected.
3. The contents of $U+1_{32-0}$ is ignored.
4. If the operand is not a normalized number or is equal to ±0, the result stored in Aa is +0, regardless of the characteristic of the input operand.

5.5.15. Magnitude of Characteristic Difference to Upper - MCDU 76,06

$$\left| \, |(A)|_{35-27} - |(U)|_{35-27} \, \right| \rightarrow A+1_{8-0};\ \text{zeros} \rightarrow A+1_{35-9}$$

The absolute value of the characteristic of the single-precision floating-point number from location U is subtracted from the absolute value of the characteristic of the single-precision floating-point number from Aa.

The absolute value of the 9-bit difference is stored in bits 8 through 0 of Aa+1. Bits 35 through 9 of Aa+1 are zero filled. The contents of Aa is not changed.

1. The mantissas from location U and from Aa are ignored.

5.5.16. Characteristic Difference to Upper - CDU 76,07

$$|(A)|_{35-27} - |(U)|_{35-27} \rightarrow A+1_{8-0};\ \text{sign bits to } A+1_{35-9}$$

The absolute value of the characteristic of the single-precision floating-point number from location U is subtracted from the absolute value of the characteristic of the single-precision floating-point number from Aa. The 9-bit signed difference is stored in bits 8 through 0 of Aa+1. Bits 35 through 9 of Aa+1 are filled with bits identical to the sign of the difference. The contents of Aa is not changed.

1. The mantissas from location U and from Aa are ignored.

5.6. Search and Masked-Search Instructions

There are six search instructions, each of which compares the contents of either one or two A-registers with the contents of storage locations or control registers. There are eight masked-search instructions, each of which compares contents of predefined bit positions of either one or two A-registers with the contents of the corresponding bit positions of storage locations or control registers.

These are all multistage instructions. The various stages required to perform these instructions are as follows:

- An initial stage
- Repeated test stages ($1_8 \leq$ repeat count $\leq 777776_8$)
- Termination stage

If indirect addressing is specified, it proceeds prior to initiation of the first test stage.

The initial stage prepares the control section and the arithmetic section for the test stages. The following steps are performed during the initial stage:

- The contents of the repeat count register (R1) is read from GRS.
- The contents of the specified A-registers are transferred to the arithmetic section.
- The contents of the mask register (R2) is transferred to the arithmetic section for a masked-search instruction.

These steps are performed only during the initial stage and are not repeated during the test stages.

The rightmost 18 bit positions of R1 contain the repeat count; that is, the maximum number of test stages to be performed. R1 must be loaded with the desired repeat count prior to initiating a search or masked-search instruction. If the initial repeat count is ±0, the R1 location is written to +0 and proceeds to the next instruction. If the initial repeat count is not ±0, the search instruction continues.

During each test stage, the value U is formed in the index subsection. For the search instructions, an input operand is transferred to the arithmetic section under j-field control. The inputs to the test process are the values obtained using the effective U address and the A-register or registers specified by the instruction.

For the masked-search instructions, the contents of the j-field is a minor function code. The inputs to the test process are:

- the logical product of the mask from R2 and the input operand addressed by U and,
- the logical products of the mask and the specified A-registers.

Each bit of the logical product is the logical product of the contents of corresponding bit positions of the two words. The logical product of two bits gives the same results as the Logical AND.

The search and masked-search instructions include algebraic and alphanumeric comparisons. During an algebraic comparison, the leftmost bit of each of the 36-bit values is considered to be a sign bit; a positive number is always recognized as being greater than a negative number. During an alphanumeric comparison, the leftmost bit of each of the 36-bit values is considered to be a numeric bit rather than a sign bit.

For each test process, the repeat count is decreased by one. If the test process shows that the specified conditions are met, the termination stage is initiated and the next instruction is skipped. If the specified conditions are not met and the repeat count is not zero, another test stage is normally initiated. If the repeat count is zero, the search instruction is terminated. It should be noted that if Xn = 0, Xi (the increment portion of the X-register) = 0, or h = 0, the value of U will not change.

Interrupts are detected during the test stages if the repeat count is greater than 1 and the instruction does not indicate a skip condition. The instruction terminates and stores the remaining repeat count in R1. The correct return address is captured.

If the search or masked-search instruction is entered by means of an Execute instruction, the h-field of the Execute instruction should be zero (that is, no incrementation) so that when the program returns to the Execute instruction after an interrupt, the effective U address will again lead to the search or masked-search instruction.

If the search or masked-search instruction specifies indirect addressing (i-field = 1), the h-field should be zero to enable the program to return to the same effective U address and resume the search or masked search after an interrupt.

For equality searches (SE, SNE, MSE, MSNE), +0 does not equal -0; for arithmetic searches (SLE, SG, SW, SNW, MSLE, MSE, MSW, MSNW), +0 is greater than -0; for alphanumeric searches (MASL, MASG), -0 is greater than +0.

When a search or masked-search is resumed after an interrupt, the initial stage is again performed to prepare the control section for the remaining test stages and to transfer the contents of the specified A-register to the arithmetic section for the comparisons performed in the test stages. When h = 1 (that is, index register incrementation is specified), if the a- and x-fields reference the same control register, the contents of that register will have been altered by the index incrementation which occurred before the search or masked search was interrupted. As a result, when the search or masked search is resumed, the value referenced by the a-field to be used in the test stages is no longer the original test value used before the interrupt occurred. Therefore, when h = 1, the a-field and x-field should not specify the same control register so that the search or masked-search instruction can be resumed in the event of an interrupt.

5.6.1. Search Equal - SE 62

Skip NI if (U) = (A), else repeat

During the initial stage, the contents of the repeat count register (R1) is transferred to the index subsection, the contents of Aa is transferred to the arithmetic section, and the P-register is incremented.

If the initial repeat count is zero, the next instruction (NI) is initiated.

If the initial repeat count is not zero, the first test stage is initiated. During each test stage, the repeat count is decreased and the contents of U is transferred to the arithmetic section under j-field control. This value from U is compared with the value from Aa and:

- If (U) = (Aa), the termination stage is initiated. This stage stores the remnant repeat count and increments the P-register. (Skip to NI.)
- If (U) ≠ (Aa) and the repeat count is not zero, another test stage is initiated.
- If (U) ≠ (Aa) and the repeat count is zero, the termination stage stores zero as the remnant repeat count and the P-register is not incremented.

1. +0 is not equal to –0.

5.6.2. Search Not Equal - SNE 63

Skip NI if (U) ≠ (A), else repeat

During the initial stage, the contents of the repeat count register (R1) is transferred to the index subsection, the contents of Aa is transferred to the arithmetic section, and the P–register is incremented.

If the initial repeat count is zero, the next instruction (NI) is initiated.

If the initial repeat count is not zero, the first test stage is initiated. During each test stage, the repeat count is decreased and the contents of U is transferred to the arithmetic section under j–field control. The value from U is compared with the value from Aa and:

- If (U) ≠ (Aa), the termination stage is initiated. The termination stage stores the remnant repeat count and increments the P–register. (Skip NI.)
- If (U) = (Aa) and the repeat count is not zero, another test stage is initiated.
- If (U) = (Aa) and the repeat count is zero, the termination stage is initiated. The termination stage stores zero as the remnant repeat count and the P–register is not incremented.

1. +0 is not equal to –0.

5.6.3. Search Less Than or Equal/Search Not Greater - SLE,SNG 64

Skip NI if (U) ≤ (A), else repeat

During the initial stage, the contents of the repeat count register (R1) is transferred to the index subsection, the contents of Aa transferred to the arithmetic section, and the P–register is incremented.

If the initial repeat count is zero, the next instruction (NI) is initiated.

If the initial repeat count is not zero, the first test stage is initiated. During each test stage, the repeat count is decreased and the contents of U is transferred to the arithmetic section under j–field control. The value from U is compared with the value from Aa and:

- If (U) ≤ (Aa), the termination stage is initiated. The termination stage stores the remnant repeat count and increments the P–register. (Skip NI.)
- If (U) > (Aa) and the repeat count is not zero, another test stage is initiated.
- If (U) > (Aa) and the repeat count is zero, the termination stage is initiated. The termination stage stores zero as the remnant repeat count and the P–register is not incremented.

1. +0 is greater than –0.

5.6.4. Search Greater - SG 65

Skip NI if (U) > (A), else repeat

During the initial stage, the contents of the repeat count register (R1) is transferred to the index subsection, the contents of Aa is transferred to the arithmetic section, and the P-register is incremented.

If the initial repeat count is zero, the next instruction (NI) is initiated.

If the initial repeat count is not zero, the first test stage is initiated. During each test stage, the repeat count is decreased and the contents of U is transferred to the arithmetic section under j-field control. The value from U is compared with the value from Aa and:

- If (U) > (Aa), the termination stage is initiated. The termination stage stores the remnant repeat count and increments the P-register. (Skip NI.)
- If (U) $\leqslant$ (Aa) and the repeat count is not zero, another test stage is initiated.
- If (U) $\leqslant$ (Aa) and the repeat count is zero, the termination stage is initiated. The termination stage stores zero as the remnant repeat count and the P-register is not incremented.

1. +0 is greater than -0.

5.6.5. Search Within Range - SW 66

Skip NI if (A) < (U) $\leqslant$ (A+1), else repeat

During the initial stage, the contents of the repeat count register (R1) is transferred to the index subsection, the contents of Aa and Aa+1 are transferred to the arithmetic section, and the P-register is incremented.

If the initial repeat count is zero, the next instruction (NI) is initiated.

If the initial repeat count is not zero, the first test stage is initiated. During each test stage, the repeat count is decreased and the contents of U is transferred to the arithmetic section under j-field control. The value from U is compared with the value from Aa and:

- If (U) > (Aa) and (U) $\leqslant$ (Aa+1), the termination stage is initiated. The termination stage stores the remnant repeat count and increments the P-register. (Skip NI.)
- If (U) $\leqslant$ (Aa) or (U) > (Aa+1), and the repeat count is not zero, another test stage is initiated.
- If (U) $\leqslant$ (Aa) or (U) > (Aa+1), and the repeat count is zero, the termination stage is initiated. The termination stage stores zero as the remnant repeat count and the P-register is not incremented.

1. +0 is greater than -0.
2. Normally, (Aa) < (Aa+1). However, if (Aa) $\geqslant$ (Aa+1), there is no value from U which can satisfy the conditions (Aa) < (U) $\leqslant$ (Aa+1).

5.6.6. Search Not Within Range - SNW 67

Skip NI if (U) $\leqslant$ (A) or (U) $>$ (A+1), else repeat

During the initial stage, the contents of the repeat count register (R1) is transferred to the index subsection, the contents of Aa and Aa+1 are transferred to the arithmetic section, and the P-register is incremented.

If the initial repeat count is zero, the next instruction (NI) is initiated.

If the initial repeat count is not zero, the first test stage is initiated. During each test stage, the repeat count is decreased and the contents of U is transferred to the arithmetic section under j-field control. The value from U is compared with the value from Aa and:

- If (U) $\leqslant$ (Aa) or (U) $>$ (Aa+1), the termination stage is initiated. The termination stage stores the remnant repeat count and increments the P-register. (Skip NI.)
- If (U) $>$ (Aa) and (U) $\leqslant$ (Aa+1), and the repeat count is not zero, another test stage is initiated.
- If (U) $>$ (Aa) and (U) $\leqslant$ (Aa+1), and the repeat count is zero, the termination stage is initiated. The termination stage stores zero as the remnant repeat count and the P-register is not incremented.

1. Normally, (Aa) $<$ (Aa+1). If, however, (Aa) $\geqslant$ (Aa+1), there is no value from U which will not satisfy the conditions (U) $\leqslant$ (Aa) or (U) $>$ (Aa+1).
2. +0 is greater than -0.

5.6.7. Masked Search Equal - MSE 71,00

Skip NI if (U) AND (R2) = (A) AND (R2), else repeat

During the initial stage, the contents of the repeat count register (R1) is transferred to the index subsection, the contents of Aa and R2 are transferred to the arithmetic section, the logical product of the values from Aa and R2 is formed, and the P-register is incremented.

If the initial repeat count is zero, the next instruction (NI) is initiated.

If the initial repeat count is not zero, the first test stage is initiated. During each test stage, the repeat count is decreased and the contents of U is transferred to the arithmetic section. (U) AND (R2) is compared to (Aa) AND (R2) and:

- If (U) AND (R2) = (Aa) AND (R2), the termination stage is initiated. This stage stores the remnant repeat count and increments the P-register (skip NI).
- If (U) AND (R2) $\neq$ (Aa) AND (R2) and the repeat count is not zero, another test stage is initiated.
- If (U) AND (R2) $\neq$ (Aa) AND (R2) and the repeat count is zero, the termination stage stores zero as the remnant repeat count and the P-register is not incremented.

1. +0 is not equal to -0.

5.6.8. Masked Search Not Equal - MSNE 71,01

Skip NI if (U) AND (R2) ≠ (A) AND (R2), else repeat.

During the initial stage, the contents of the repeat count register (R1) is transferred to the index subsection, the contents of Aa and R2 are transferred to the arithmetic section, the logical product of the values from Aa and R2 is formed, and the P-register is incremented.

If the initial repeat count is zero, the next instruction (NI) is initiated.

If the initial repeat count is not zero, the first test stage is initiated. During each test stage, the repeat count is decreased and the contents of U is transferred to the arithmetic section. (U) AND (R2) is compared to (Aa) AND (R2) and:

- If (U) AND (R2) ≠ (Aa) AND (R2), the termination stage is initiated. This stage stores the remnant repeat count and increments the P-register (skip NI).
- If (U) AND (R2) = (Aa) AND (R2) and the repeat count is not zero, another test stage is initiated.
- If (U) AND (R2) = (Aa) AND (R2) and the repeat count is zero, the termination stage stores zero as the remnant repeat count and the P-register is not incremented.

1. +0 is not equal to -0.

5.6.9. Masked Search Less Than or Equal/Not Greater - MSLE,MSNG 71,02

Skip NI if (U) AND (R2) ≤ (A) AND (R2), else repeat

During the initial stage, the contents of the repeat count register (R1) is transferred to the index subsection, the contents of Aa and R2 are transferred to the arithmetic section, the logical product of the values from Aa and R2 is formed, and the P-register is incremented.

If the initial repeat count is zero, the next instruction (NI) is initiated.

If the initial repeat count is not zero, the first test stage is initiated. During each test stage, the repeat count is decreased and the contents of U is transferred to the arithmetic section. (U) AND (R2) is compared to (Aa) AND (R2) and:

- If (U) AND (R2) ≤ (Aa) AND (R2), the termination stage is initiated. This stage stores the remnant repeat count and increments the P-register (skip NI).
- If (U) AND (R2) > (Aa) AND (R2) and the repeat count is not zero, another test stage is initiated.
- If (U) AND (R2) > (Aa) AND (R2) and the repeat count is zero, the termination stage stores zero as the remnant repeat count and the P-register is not incremented.

1. +0 is greater than -0.

5.6.10. Masked Search Greater - MSG 71,03

Skip NI if (U) AND (R2) > (A) AND (R2), else repeat

During the initial stage, the contents of the repeat count register (R1) is transferred to the index subsection, the contents of Aa and R2 are transferred to the arithmetic section, the logical product of the values from Aa and R2 is formed, and the P-register is incremented.

If the initial repeat count is zero, the next instruction (NI) is initiated.

If the initial repeat count is not zero, the first test stage is initiated. During each test stage, the repeat count is decreased and the contents of U is transferred to the arithmetic section. (U) AND (R2) is compared to (Aa) AND (R2) and:

- If (U) AND (R2) > (Aa) AND (R2), the termination stage is initiated. This stage stores the remnant repeat count and increments the P-register. (Skip NI.)
- If (U) AND (R2) $\leqslant$ (Aa) AND (R2) and the repeat count is not zero, another test stage is initiated.
- If (U) AND (R2) $\leqslant$ (Aa) AND (R2) and the repeat count is zero, the termination stage stores zero as the remnant repeat count and the P-register is not incremented.

1. +0 is greater than -0.

5.6.11. Masked Search Within Range - MSW 71,04

Skip NI if (A) AND (R2) < (U) AND (R2) $\leqslant$ (A+1) AND (R2), else repeat.

During the initial stage, the contents of the repeat count register (R1) is transferred to the index subsection, the contents of Aa, Aa+1, and R2 are transferred to the arithmetic section, the logical products of the values from Aa and R2 and the values from Aa+1 and R2 are formed, and the P-register is incremented. If the initial repeat count is zero, the next instruction (NI) is initiated.

If the initial repeat count is not zero, the first test stage is initiated. During each test stage, the repeat count is decreased and the contents of U is transferred to the arithmetic section. The logical products are compared and:

- If (U) AND (R2) > (Aa) AND (R2) and (U) AND (R2) $\leqslant$ (Aa+1) AND (R2) the termination stage is initiated. This stage stores the remnant repeat count and increments the P-register. (Skip NI.)
- If (U) AND (R2) $\leqslant$ (Aa) AND (R2) or (U) AND (R2) > (Aa+1) AND (R2) and the repeat count is not zero, another test stage is initiated.
- If (U) AND (R2) $\leqslant$ (Aa) AND (R2) or (U) AND (R2) > (Aa+1) AND (R2) and the repeat count is zero, the termination stage stores zero as the remnant repeat count and the P-register is not incremented.

1. Normally, (Aa) AND (R2) < (Aa+1) AND (R2). If, however, (Aa) AND (R2) $\geqslant$ (Aa+1) AND (R2), no possible value of U will satisify the search condition.
2. +0 is greater than -0.

5.6.12. Masked Search Not Within Range - MSNW 71,05

Skip NI if (U) AND (R2) $\leqslant$ (A) AND (R2) or (U) AND (R2) > (A+1) AND (R2), else repeat

During the initial stage, the contents of the repeat count register (R1) is transferred to the index subsection, the contents of Aa and R2 are transferred to the arithmetic section, the logical products of the values from Aa and R2 and the values from Aa+1 and R2 are formed, and the P-register is incremented.

If the initial repeat count is zero, the next instruction (NI) is initiated.

If the initial repeat count is not zero, the first test stage is initiated. During each test stage, the repeat count is decreased and the contents of U is transferred to the arithmetic section. The logical products are compared and:

- If (U) AND (R2) $\leqslant$ (Aa) AND (R2) or (U) AND (R2) $>$ (Aa+1) AND (R2) the termination stage is initiated. This stage stores the remnant repeat count and increments the P-register. (Skip NI.)
- If (U) AND (R2) $>$ (Aa) AND (R2) and (U) AND (R2) $\leqslant$ (Aa+1) AND (R2) and the repeat count is not zero, another test stage is initiated.
- If (U) AND (R2) $>$ (Aa) AND (R2) and (U) AND (R2) $\leqslant$ (Aa-1) AND (R2) and the repeat count is zero, the termination stage stores zero as the remnant repeat count and the P-register is not incremented.

1. Normally, (Aa) AND (R2) $<$ (Aa+1) AND (R2). If, however, (Aa) AND (R2) $\geqslant$ (Aa+1) AND (R2), every possible value of U will satisfy at least one of the following conditions:

   (U) AND (R2) $\leqslant$ (Aa) AND (R2)

   (U) AND (R2) $>$ (Aa+1) AND (R2)

2. +0 is greater than -0.

5.6.13. Masked Alphanumeric Search Less Than or Equal - MASL 71,06

Skip NI if (U) AND (R2) $\leqslant$ (A) AND (R2), else repeat

During the initial stage, the contents of the repeat count register (R1) is transferred to the index subsection, the contents of Aa and R2 are transferred to the arithmetic section, the logical product of the values from Aa and R2 is formed, and the P-register is incremented.

If the initial repeat count is zero, the next instruction (NI) is initiated.

If the initial repeat count is not zero, the first test stage is initiated. During each test stage, the repeat count is decreased and the contents of U is transferred to the arithmetic section. (U) AND (R2) is compared alphanumerically to (Aa) AND (R2), and:

- If (U) AND (R2) $\leqslant$ (Aa) AND (R2), the termination stage is initiated. This stage stores the remnant repeat count and increments the P-register. (Skip NI.)
- If (U) AND (R2) $>$ (Aa) AND (R2) and the repeat count is not zero, another test stage is initiated.
- If (U) AND (R2) $>$ (Aa) AND (R2) and the repeat count is zero, the termination stage stores zero as the remnant repeat count and the P-register is not incremented.

1. -0 is greater than +0.

5.6.14. Masked Alphanumeric Search Greater - MASG 71,07

Skip NI if (U) AND (R2) > (A) AND (R2), else repeat

During the initial stage, the contents of the repeat count register (R1) is transferred to the index subsection, the contents of Aa and R2 are transferred to the arithmetic section, the logical product of the values from Aa and R2 is formed, and the P-register is incremented.

If the initial repeat count is zero, the next instruction (NI) is initiated.

If the initial repeat count is not zero, the first test stage is initiated. During each test stage, the repeat count is decreased and the contents of U is transferred to the arithmetic section. (U) AND (R2) is compared alphanumerically to (Aa) AND (R2), and:

- If (U) AND (R2) > (Aa) AND (R2), the termination stage is initiated. This stage stores the remnant repeat count and increments the P-register (skip NI).
- If (U) AND (R2) $\leqslant$ (Aa) AND (R2) and the repeat count is not zero, another test stage is initiated.
- If (U) AND (R2) $\leqslant$ (Aa) AND (R2) and the repeat count is zero, the termination stage stores zero as the remnant repeat count and the P-register is not incremented.

1. -0 is greater than +0.

5.7. Test (or Skip) Instructions

Test instructions are used to read one or more words from storage or control registers and test for certain conditions. The result of the test is used to determine whether the instruction addressed by the incremented contents of the P-register (next instruction) should be performed or skipped.

The next instruction (NI) is always read from storage. If the decision is made to skip NI, it is discarded, the P-register is incremented a second time, and the contents of the P-register is then used to address the following instruction.

Indirect addressing, indexing, and index register incrementation/decrementation operate normally.

5.7.1. Test Even Parity - TEP 44

Skip NI if (U) AND (A) has even parity.

The value from U is transferred to the arithmetic section under j-field control, where it is used with the contents of Aa to form a 36-bit logical product.

If (U) AND (Aa) has an even number of 1 bits, the next instruction (NI) is skipped and the instruction following NI is performed.

If (U) AND (Aa) has an odd number of 1 bits, NI is performed.

5.7.2. Test Odd Parity - TOP 45

Skip NI if (A) AND (U) has odd parity.

The contents of U is transferred to the arithmetic section under j-field control, where it is used with the contents of Aa to form a 36-bit logical product.

If (U) AND (Aa) has an odd number of 1 bits, the next instruction (NI) is skipped and the instruction following NI is performed.

If (U) AND (Aa) has an even number of 1 bits, NI is performed.

5.7.3. Test Less Than or Equal/Test Not Greater Than Modifier - TLEM,TNGM 47

Skip NI if $(U)_{17-0} \leqslant (Xa)_{17-0}$; always $(Xa)_{17-0} + (Xa)_{35-18} \rightarrow Xa_{17-0}$

The contents of U is transferred to the arithmetic section under j-field control. The contents of the index register addressed by the a-field (Xa) is transferred to the arithmetic section. The rightmost 18 bits of the value from U is subtracted from the rightmost 18 bits of the value from Xa (this is performed as if the leftmost 18 bits of each operand were zeros).

If $(U)_{17-0} \leqslant (Xa)_{17-0}$ (the sign of the difference is positive), the next instruction is skipped and the instruction following NI is performed.

If $(U)_{17-0} > (Xa)_{17-0}$ (the sign of the difference is negative), NI is performed.

In either case, the leftmost 18 bits from Xa are added to the rightmost 18 bits from Xa, and the sum is stored in the rightmost 18 bit positions of Xa. The leftmost 18 bit positions of Xa are not changed.

1. If a = 0, index register zero (XO) is referenced.
2. +0 is less than -0.
3. Both Xa_{17-0} and the value from U are considered to be 18-bit numeric values with a positive sign implied.
4. Only the rightmost 18 bits of the value from U are involved in the operation. Values of 0, 1, or 3 in the j-field yield the same results. Values of 16_8 or 17_8 in the j-field yield the same result.
5. If h = 1 and a = x, the specified index register is incremented or modified only once.

5.7.4. Test Zero - TZ 50

Skip NI if (U) = ±0.

The contents of U is transferred to the arithmetic section under j-field control.

If the value transferred is ±0, the next instruction is skipped and the instruction following NI is performed.

If the value transferred is not ±0, NI is performed.

1. The contents of the a-field is ignored.

5.7.5. Test Nonzero - TNZ 51

Skip NI if (U) ≠ ±0.

The contents of U is transferred to the arithmetic section under j-field control.

If the value transferred is not ±0, the next instruction is skipped and the instruction following NI is performed.

If the value transferred is ±0, NI is performed.

1. The contents of the a-field is ignored.

5.7.6. Test Equal - TE 52

Skip NI if (U) = (A).

The contents of U is transferred to the arithmetic section under j-field control. The contents of Aa is also transferred to the arithmetic section.

If (U) = (Aa), the next instruction is skipped and the instruction following NI is performed.

If (U) ≠ (Aa), NI is performed.

1. +0 is not equal to -0.

5.7.7. Test Not Equal - TNE 53

Skip NI if (U) ≠ (A).

The contents of U is transferred to the arithmetic section under j-field control. The contents of Aa is also transferred to the arithmetic section.

If (U) ≠ (Aa), the next instruction is skipped and the instruction following NI is performed.

If (U) = (Aa), NI is performed.

1. +0 is not equal to -0.

5.7.8. Test Less Than or Equal/Test Not Greater - TLE,TNG 54

Skip NI if (U) ≤ (A).

The contents of U is transferred to the arithmetic section under j-field control. The contents of Aa is also transferred to the arithmetic section.

If (U) ≤ (Aa), the next instruction is skipped and the instruction following NI is performed.

If (U) > (Aa), NI is performed.

1. +0 is greater than -0.

5.7.9. Test Greater - TG 55

Skip NI if (U) > (A).

The contents of U is transferred to the arithmetic section under j-field control. The contents of Aa is also transferred to the arithmetic section.

If (U) > (Aa), the next instruction is skipped and the instruction following NI is performed.

If (U) $\leqslant$ (Aa), NI is performed.

1. +0 is greater than -0.

5.7.10. Test Within Range - TW 56

Skip NI if (A) < (U) $\leqslant$ (A+1).

The contents of U is transferred to the arithmetic section under j-field control. The contents of Aa and Aa+1 are also transferred to the arithmetic section.

If (Aa) < (U) $\leqslant$ (Aa+1), the next instruction is skipped and the instruction following NI is performed.

If (U) $\leqslant$ (Aa) or (U) > (Aa+1), NI is performed.

1. +0 is greater than -0.
2. Normally, (Aa) < (Aa+1). If, however, (Aa) $\geqslant$ (Aa+1), there is no value of U that can satisfy the condition (Aa) < (U) $\leqslant$ (Aa+1).

5.7.11. Test Not Within Range - TNW 57

Skip NI if (U) $\leqslant$ (A) or (U) > (A+1).

The contents of U is transferred to the arithmetic section under j-field control. The contents of Aa and Aa+1 are also transferred to the arithmetic section.

If (U) $\leqslant$ (Aa) or (U) > (Aa+1), the next instruction is skipped and the instruction following NI is performed.

If (U) > (Aa) and (U) $\leqslant$ (Aa+1), NI is performed.

1. +0 is greater than -0.
2. Normally, (Aa) < (Aa+1). If, however, (Aa) $\geqslant$ (Aa+1), every possible value of U will satisfy at least one of the following conditions:

   (U) $\leqslant$ (Aa)

   or

   (U) > (Aa+1)

5.7.12. Test Positive - TP 60

Skip NI if $(U)_{35} = 0$.

The contents of U is transferred to the arithmetic section under j-field control.

If the sign bit (bit 35) of the value from U is a 0 bit, the next instruction is skipped and the instruction following NI is performed.

If the sign bit is a 1 bit, NI is performed.

1. The contents of the a-field is ignored.
2. Always skip when j = H1, H2, Q1-Q4, or S1-S6.

5.7.13. Test Negative - TN 61

Skip NI if $(U)_{35} = 1$.

The contents of U is transferred to the arithmetic section under j-field control.

If the sign bit (bit 35) of the value from U is a 1 bit, the next instruction is skipped and the instruction following NI is performed.

If the sign bit is a 0 bit, NI is performed.

1. The contents of the a-field is ignored.
2. Never skip when j = H1, H2, Q1-Q4, or S1-S6.

5.7.14. Double-Precision Test Equal - DTE 71,17

Skip NI if (U, U+1) = (A, A+1).

The contents of U, U+1, Aa, and Aa+1 are transferred to the arithmetic section. U, U+1 and Aa, Aa+1 are 72-bit operands.

If (U, U+1) = (Aa, Aa+1), the next instruction is skipped and the instruction following NI is performed.

If (U, U+1) ≠ (Aa, Aa+1), NI is performed.

1. +0 is not equal to -0.

5.8. Shift Instructions

Each shift instruction transfers either one or two words to the arithmetic section, moves or shifts the bits of the words, and stores the shifted word or words in one or two control registers.

The following basic types of shifts are provided for both single-word (36-bit input operand) and double-word (two 36-bit words treated as a 72-bit input operand) operations:

- Right circular

  For a right-circular shift, a shift count of n moves the contents of all bit positions of the register holding the input operand n bit positions to the right. Bits shifted out the right end of the register appear in the leftmost bit positions vacated by the shift.

- Left circular

  For a left-circular shift, a shift count of n moves the contents of all bit positions of the register holding the input operand n places to the left. Bits shifted out the left end of the register appear in the rightmost bit positions vacated by the shift.

  For example: A shift count of 6 for a right-circular shift applied to 765432101234_8 as the input operand produces 347654321012_8 as the result. The same result is produced using a shift count of 30 for a left-circular shift.

  For a single-word circular shift, a shift count of 72 or 36 produces the same result as a shift count of 0 (no shift). A shift count of 37 produces the same effect as a shift count of 1, a shift count of 38 produces the same effect as a shift count of 2, and so on.

- Right logical

  For a right-logical shift, a shift count of n moves the contents of all bit positions of the register holding the input operand n places to the right. Bits shifted out the right end of the register are lost. The leftmost bit positions vacated by the shift are zero filled.

  For example: A shift count of 6 for a right-logical shift applied to 765432101234_8 as the input operand produces 007654321012_8 as the result.

- Left logical

  For a left-logical shift, a shift count of n moves the contents of all bit positions of the input operand register n places to the left. Bits shifted out the left end of the register are lost. The rightmost bit positions vacated by the shift are zero filled.

  For example: A shift count of 6 for a left-logical shift applies to 765432101234_8 as the input operand produces 543210123400_8 as the result.

- Right algebraic

  For an algebraic shift (right only, since no left algebraic shift is provided), a shift count of n moves the contents of all bit positions of the register holding the input operand n places to the right. Bits shifted out the right end of the register are lost. The bit positions vacated by the shift are filled with bits identical to the leftmost bit (sign bit) of the original input operand.

  For example: A shift count of 6 for an algebraic shift applied to 765432101234_8 as the input operand produces 777654321012_8 as the result.

The two Load Shift and Count instructions are basically left circular shift instructions. The shift count is determined by the configuration of the bits of the input operand. If the two leftmost bits are not identical, the shift count is zero. If the two leftmost bits are identical, the operand is shifted left circular by the minimum amount to position the bits of the input operand so that the two leftmost bits are not identical. The shift count is the count of the number of bit positions shifted. If all bits of an input operand are identical, no amount of circular shifting will position its bits so that the two left-most bits are not identical. In this instance, the shift count is 35 (single-word operand) or 71 (double-word operand). The shift count is stored in a control register.

For all shift instructions, except the two Load Shift and Count instructions, the input operands are specified by one or two A-registers, and the shift count is specified by bits 6 through 0 of the effective U. Indirect addressing, indexing, and index register incrementation/decrementation operate normally for all shift instructions.

The shift count can be any number between 0 and 72. If a shift count of 73 to 127 (111_8 through 177_8) is specified, the result produced is undefined. The value in the u-field of the shift instruction and the value of Xm (if x ≠ 0) must be chosen accordingly.

For the two Load Shift and Count instructions, the effective U specifies the input operand address just as for the other load instructions. The scaled result is loaded in the specified A-register (A, A+1 for Double Load Shift and Count instruction). The number of shifts required for scaling is stored in the next consecutive register A+1 (or A+2 for Double Load Shift and Count instruction).

5.8.1. Single Shift Circular - SSC 73,00

Shift (A) right circularly U places.

The contents of Aa is transferred to the arithmetic section. The shift count from bits 6 through 0 of U is transferred to the arithmetic section. The value from Aa is shifted right circularly by the number of bit positions specified by the shift count. The shifted value is stored in Aa.

1. The result stored is not defined for shift counts greater than 72.
2. If $36 \leqslant n \leqslant 72$, a shift count of n produces the same result as a shift count of n-36.

5.8.2. Double Shift Circular - DSC 73,01

Shift (A, A+1) right circularly U places.

The contents of Aa and Aa+1 are transferred to the arithmetic section. The shift count from bits 6 through 0 of U is transferred to the arithmetic section. The 72-bit value from Aa and Aa+1 is shifted right circularly the number of bit positions specified by the shift count. The shifted value is stored in Aa and Aa+1.

1. The result stored is not defined for shift counts greater than 72.

5.8.3. Single Shift Logical - SSL 73,02

Shift (A) right U places, zero fill.

The contents of Aa is transferred to the arithmetic section. The shift count from bits 6 through 0 of U is transferred to the arithmetic section. The value from Aa is right shifted the number of bit positions specified by the shift count. Bits shifted out of the rightmost bit positions are lost; the vacated leftmost bit positions are zero filled. The shifted value is stored in Aa.

1. The result stored is not defined for shift counts greater than 72.
2. If $36 \leqslant U \leqslant 72$, the result stored in Aa is +0.

5.8.4. Double Shift Logical - DSL 73,03

Shift (A,A+1) right U places, zero fill.

The contents of Aa and Aa+1 are transferred to the arithmetic section. The shift count from bits 6 through 0 of U is transferred to the arithmetic section. The 72-bit value from Aa and Aa+1 is right shifted the number of bit positions specified by the shift count. Bits shifted out of the rightmost bit positions are lost; the vacated leftmost bit positions are zero filled.

1. The result stored is not defined for shift counts greater than 72.

5.8.5. Single Shift Algebraic - SSA 73,04

Shift (A) right U places, sign fill.

The contents of Aa is transferred to the arithmetic section. The shift count from bits 6 through 0 of U is transferred to the arithmetic section. The value from Aa is right shifted the number of bit positions specified by the shift count. Bits shifted out of the rightmost bit positions are lost; bits identical to the content of bit 35 of the initial value from Aa appear in the vacated leftmost bit positions. The shifted count is stored in Aa.

1. The result stored is not defined for shift counts greater than 72.

2. If $35 \leqslant U \leqslant 72$, all bits of the result stored in Aa are identical to the leftmost bit of the input operand from Aa.

5.8.6. Double Shift Algebraic - DSA 73,05

Shift (A, A+1) right U places, sign fill.

The contents of Aa and Aa+1 are transferred to the arithmetic section. The shift count from bits 6 through 0 of U is transferred to the arithmetic section. The 72-bit value from Aa and Aa+1 is right shifted the number of bit positions specified by the shift count. Bits shifted out of the rightmost bit positions are lost; bits identical to the contents of bit 35 of the initial value from Aa appear in the vacated leftmost bit positions. The shifted value is stored in Aa and Aa+1.

1. The result stored is not defined for shift counts greater than 72.

5.8.7. Load Shift and Count - LSC 73,06

(U) → A; shift (A) left circularly until $(A)_{35} \neq (A)_{34}$; number of shifts → A+1.

The contents of location U is transferred to a nonaddressable 36-bit register in the arithmetic section and then shifted left circularly the minimum number of bit positions which will make bit 35 unequal to bit 34. The resultant scaled number is transferred to Aa and the shift count to Aa+1.

1. If bit 35 of the value from location U is not equal to bit 34, the number is already scaled and no shift occurs: (U) → Aa; +0 → Aa+1.

2. If the value from location U is ±0: (U) → Aa, the shift count is 35, and 43_8 → Aa+1.

5.8.8. Double Load Shift and Count - DLSC 73,07

(U, U+1) → A, A+1; shift (A, A+1) left circularly until $(A, A+1)_{71} \neq (A,A+1)_{70}$; number of shifts → A+2.

The contents of U and U+1 are transferred to a nonaddressable 72-bit register in the arithmetic section and then shifted left circularly the minimum number of bit positions which will make bit 71 unequal to bit 70. The resultant scaled number is transferred to Aa and Aa+1 and the shift count to Aa+2.

1. If bit 71 of the value from U and U+1 is not equal to bit 70, the double length number is already scaled and no shift occurs: (U) → Aa; (U+1) → Aa+1; +0 → Aa+2.

2. If the double-length value from locations U and U+1 is ±0: (U) → Aa; (U+1) → Aa+1; the shift count is 71; 107_8 → Aa+2.

5.8.9. Left Single Shift Circular - LSSC 73,10

Shift (A) left circularly U places.

The contents of Aa is transferred to the arithmetic section. The shift count from bits 6 through 0 of U is transferred to the arithmetic section. The value from Aa is shifted left circularly the number of bit positions specified by the shift count. The shifted value is stored in Aa.

1. The result stored is undefined for shift counts greater than 72.

2. If $36 \leqslant n \leqslant 72$, a shift count of n produces the same result as a shift count of n-36.

5.8.10. Left Double Shift Circular - LDSC 73,11

Shift (A, A+1) left circularly U places.

The contents of Aa and Aa+1 are transferred to the arithmetic section. The shift count from bits 6 through 0 of U is transferred to the arithmetic section. The 72-bit value from Aa and Aa+1 is shifted left circularly the number of bit positions specified by the shift count. The shifted value is stored in Aa and Aa+1.

1. The result stored is undefined for shift counts greater than 72.

5.8.11. Left Single Shift Logical - LSSL 73,12

Shift (A) left U places, zero fill.

The contents of Aa is transferred to the arithmetic section. The shift count from bits 6 through 0 of U is transferred to the arithmetic section. The value from Aa is left shifted the number of bit positions specified by the shift count. Bits shifted out of the leftmost bit positions are lost; the vacated rightmost bit positions are zero filled. The shifted value is stored in Aa.

1. The result stored is undefined for shift counts greater than 72.

2. If $36 \leqslant U \leqslant 72$, the result stored in Aa is +0.

5.8.12. Left Double Shift Logical - LDSL 73,13

Shift (A, A+1) left U places, zero fill.

The contents of Aa and Aa+1 are transferred to the arithmetic section. The shift count from bits 6 through 0 of U is transferred to the arithmetic section. The 72-bit value from Aa and Aa+1 is left shifted the number of bit positions specified by the shift count. Bits shifted out of the leftmost bit positions are lost; the vacated rightmost bit positions are zero filled. The shifted value is stored in Aa and Aa+1.

1. The result stored is undefined for shift counts greater than 72.

5.9. Unconditional Jump Instructions

A jump is a change in the sequence in which instructions are executed. It is accomplished by placing a new value in the program address register (P-register). Each unconditional jump instruction performs a unique operation in addition to the common operation of placing a new value in the P-register.

If the relative "jump to" address is less than 200_8, the next instruction is taken from the storage location addressed by the value rather than from a control register.

The Jump Keys instruction can be used to specify either a conditional or an unconditional jump. The Halt Jump/Halt Keys and Jump instruction specifies an unconditional jump, but the halt portion is conditional. Both of these instructions are included in the section on conditional jump instructions (see 5.11).

5.9.1. Store Location and Jump - SLJ 72,01

Relative P+1 $\rightarrow U_{17-0}$; jump to U+1

The P-register is incremented. An 18-bit relative return address is stored in the rightmost 18 bits of the location specified by the operand address. The value of the operand address plus one is transferred to the P-register as the "jump to" address. The upper half of the operand is unchanged.

1. The contents of the a-field is ignored.

2. If U < 200_8, the 18-bit relative return address is stored in the rightmost 18 bits of the control register addressed by U, and the leftmost 18 bit positions of that control register are unchanged.

3. The 18-bit address always represents a relative address. Therefore, executing this instruction in absolute addressing mode may produce erroneous results.

4. The relative return address is stored in the low-order 18 bits of a word. If this 18-bit relative return address is larger than 16 bits, the two high-order bits will be interpreted as h and i bits if the address is used in an instruction. The instruction may produce erroneous results.

5.9.2. Load Modifier and Jump - LMJ 74,13

Relative P+1 $\rightarrow Xa_{17-0}$; jump to U

The P-register is incremented. An 18-bit relative return address is stored in the rightmost 18 bits of the index register specified by the a-field. The leftmost 18 bits of that index register are not affected. The value of the operand is transferred to the P-register as the "jump to" address.

1. If the GRS selection designator (D6) = 0 and the value in the a-field is zero, the relative return address is stored in index register zero (X0).

2. If index register incrementation is specified, the relative return address is stored in the index register specified by the a-field after the new value for Xm is stored in the index register specified by the x-field. As a consequence, if the value in the a-field is not zero and it is the same as the value in the x-field, it makes no difference whether the value in the h-field is zero or one.

5.9.3. Allow All Interrupts and Jump - AAIJ 74,07

Allow all interrupts and jump to U.

This instruction allows interrupts prevented by the occurrence of an interrupt or the execution of a Prevent All Interrupts and Jump instruction.

1. The contents of the a-field is ignored.

2. The Allow All Interrupts and Jump instruction does not affect the Dayclock interrupt when it is disabled by the Disable Dayclock instruction and enabled by the Enable Dayclock instruction.

5.10. Bank Descriptor Selection Instructions

Each program may be composed of or associated with a large number of program or data segments; of these, up to four may be active at any given time. Bank Descriptor selection instructions allow a program to select which segments are among the four that are currently active.

5.10.1. Load Bank and Jump - LBJ 07,17

The LBJ instruction loads the bank descriptor register selected by bit position 34 and 33 of the index register specified by the a-field of the instruction word (Xa) with a new bank descriptor, stores the old bank descriptor indexing information and relative program address in Xa as return information, and then jumps to the location specified by the operand address. The new bank descriptor is located by adding the bank descriptor index contained in bit positions 18 through 29 of Xa to the bank descriptor table pointer selected by bit position 35 of Xa. If bit 35 is zero, the user pointer and table are selected; if bit 35 is one, the Executive pointer and table are selected. An Address Exception interrupt occurs when bit 35 is one and the EXEC bank descriptor table pointer enable designator (D19) is zero. An Address Exception interrupt also occurs if the bank descriptor index value exceeds the length of the table.

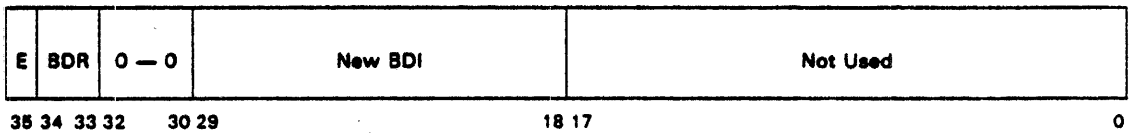
Before the new bank descriptor values are actually loaded, the old bank descriptor is located and the use-count field is decreased by one under storage lock. An Addressing Exception interrupt occurs if the C-flag of the old bank descriptor is one and the use count is decreased to zero, or if the use count is decreased from zero to all ones. The new bank descriptor is loaded in the bank descriptor register, the P-flag is transferred to the privileged instruction, GRS protect, and interrupt lockout detect designator (D2), and the W-flag of the new bank descriptor is placed in the appropriate write-protection bit of the bank descriptor register designator bits (D13 through D16).

When the new bank descriptor is located, the associated use count field is increased by one under storage lock, and an Addressing Exception interrupt occurs if the R-flag of the bank descriptor is one, if there is a V-flag violation, or if the use count field is increased from all ones to zero.

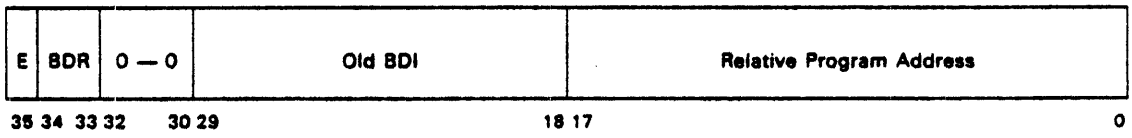
The appropriate bank descriptor indexing information is copied from the GRS processor state area into the upper half of Xa, the relative program address is copied into the lower half of Xa, and the new bank descriptor indexing information is stored in the appropriate half word of GRS 46_8 or 47_8. The operand address is formed and a jump to that location is effected. If both an address exception and jump address guard mode limits violation occur during the execution of this instruction, the address exception will be taken.

The following are the formats of Xa before and after execution of the instruction:

Xa Before Execution



Xa After Execution



5.10.2. Load I-Bank Base and Jump - LIJ 07,13

The LIJ instruction is executed as a special case of the LBJ instruction. Bit positions 34-33 of Xa are ignored; if the BDR selector designator (D12) is zero, BDR0 is loaded and the BDR field (bits 34-33) of the bank descriptor index is written to 0_8; and if D12 is one, BDR1 is loaded and the BDR field is written to 1_8.

5.10.3. Load D-Bank Base and Jump - LDJ 07,12

The LDJ instruction is executed as a special case of the LBJ instruction. Bit positions 34-33 of Xa are ignored; if the BDR selector designator (D12) is zero, BDR2 is loaded and the BDR field (bits 34-33) of the bank descriptor index is written to 2_8; and if D12 is one, BDR3 is loaded and the BDR field is written to 3_8.

5.11. Conditional Jump Instructions

Each of the conditional jump instructions performs a test for a specific condition (or set of conditions). If the condition is satisfied, the value U is transferred to the P-register, and the instruction addressed by U is performed next. If the condition is not satisfied, the next instruction (NI) is performed.

5.11.1. Jump Greater and Decrement - JGD 70

Jump to U if $(\text{control register})_{ja} > 0$; go to NI if $(\text{control register})_{ja} \leqslant 0$; always $(\text{control register})_{ja} - 1 \rightarrow \text{control register}_{ja}$.

If the 36–bit signed number in the control register addressed by the rightmost 7 bits of the ja–field is greater than zero (bit 35 contains a 0 bit and the number does not consist of all 0 bits), the instruction at location U is executed next. If the number is less than, or equal to, zero (bit 35 contains a 1 bit or the number consists of all 0 bits), the next instruction is performed. In either case, the number is decreased by one and the difference is stored in the control register addressed by the ja–field.

1. A Guard Mode interrupt occurs (if guard mode is set) when the ja–field specifies a value in the range 40_8 through 100_8, or 120_8 through 177_8 if the privileged instruction, GRS protect, and interrupt lockout detect designator (D2) = 1. This is true regardless of the value of the GRS selection designator (D6).

2. The leftmost bit in the j–field is ignored.

5.11.2. Double–Precision Jump Zero - DJZ 71,16

Jump to U if $(A,A+1) = \pm 0$; go to NI if $(A,A+1) \neq \pm 0$.

If the 72–bit operand contained in Aa and Aa+1 is ±0, the instruction at location U is performed next. If the operand is not ±0, the next instruction (NI) is performed.

5.11.3. Jump Positive and Shift - JPS 72,02

Jump to U if $(A)_{35} = 0$; go to NI if $(A)_{35} = 1$; always shift (A) left circularly one bit position.

If bit 35 of Aa contains a 0 bit, the instruction at location U is performed next. If bit 35 contains a 1 bit, the next instruction is performed. The contents of Aa is always shifted left circularly one bit position.

1. The bit shifted out of bit 35 of Aa is shifted to bit 0 of Aa.

5.11.4. Jump Negative and Shift - JNS 72,03

Jump to U if $(A)_{35} = 1$; go to NI if $(A)_{35} = 0$; always shift (A) left circularly one bit position.

If bit 35 of Aa is a 1 bit, the instruction at location U is performed next. If bit 35 is a 0 bit, the next instruction is performed. The contents of Aa is always shifted left circularly one bit position.

1. The bit shifted out of bit 35 of Aa is shifted to bit 0 of Aa.

5.11.5. Jump Zero - JZ 74,00

Jump to U if $(A) = \pm 0$; go to NI if $(A) \neq \pm 0$.

If (Aa) is ±0, the instruction at location U is performed next. If Aa does not contain ±0, the next instruction is performed.

5.11.6. Jump Nonzero - JNZ 74,01

Jump to U if (A) ≠ ±0; go to NI if (A) = ±0.

If (Aa) is not ±0, the instruction at location U is performed next. If (Aa) is ±0, the next instruction is performed.

5.11.7. Jump Positive - JP 74,02

Jump to U if $(A)_{35} = 0$; go to NI if $(A)_{35} = 1$.

If bit 35 of Aa is a 0 bit, the instruction at location U is performed next. If bit 35 is a 1 bit, the next instruction is performed.

5.11.8. Jump Negative - JN 74,03

Jump to U if $(A)_{35} = 1$; go to NI if $(A)_{35} = 0$.

If bit 35 of Aa is a 1 bit, the instruction at location U is performed next. If bit 35 is a 0 bit, the next instruction is performed.

5.11.9. Jump/Jump Keys - J,JK 74,04

Jump to U if a = 0 or if a = set JUMP SELECT switch; go to NI if neither is true.

If the a-field contains all 0 bits, the instruction at location U is performed next. If the a-field contains a value in the range of 1 through 15 (1_8 through 17_8) and the correspondingly numbered JUMP SELECT switch/indicator is set, the instruction at location U is performed next; if the correspondingly numbered JUMP SELECT switch/indicator is not set, the next instruction is performed.

1. The indicator for each of the 15 JUMP SELECT switch/indicators is turned on by pressing that JUMP SELECT switch/indicator. Each is turned off by pressing the associated clear switch. Either can be done while the central processor unit (CPU) is running.

2. Care should be exercised in using a value other than all 0 bits in the a-field if the program is to run concurrently with one or more other programs. Any other program may include a Jump Keys instruction with the same value in the a-field and specify that it is to be run with the corresponding JUMP SELECT switch/indicator set.

5.11.10. Halt Jump/Halt Keys and Jump - HJ,HKJ 74,05

Stop if [a=0 OR if (a AND set STOP SELECT switches) ≠ 0] AND D2 = 0; on restart or continuation jump to U.

If the a-field contains all 0 bits, the execution of program instruction halts. If the a-field contains a 1 bit in a bit position which corresponds to a lit STOP SELECT switch/indicator, the program halts. A manual restart causes the instruction at location U to be executed next unless the program address register was manually changed while the CPU was halted.

When the privileged instruction, GRS protect,and interrupt lockout detect designator (D2) = 1 and the halt conditions are satified, the jump is executed without halting.

1. The indicator for each of the four STOP SELECT switch/indicators is turned on by pressing one of the STOP SELECT switch/indicators. They are turned off by pressing the associated clear switch.

2. If the program address register is manually changed while the CPU is halted, program execution will resume at the new address when the CPU is restarted.

5.11.11. Jump No Low Bit - JNB 74,10

Jump to U if $(A)_0 = 0$; go to NI if $(A)_0 = 1$.

If bit 0 of Aa is a 0 bit, the instruction at location U is performed next. If bit 0 is a 1 bit, the next instruction is performed.

1. If the Jump No Low Bit instruction is used to determine whether the value in Aa is an even or an odd integer, consideration must be given to the sign of the value.

5.11.12. Jump Low Bit - JB 74,11

Jump to U if $(A)_0 = 1$; go to NI if $(A)_0 = 0$.

If bit 0 of Aa is a 1 bit, the instruction at location U is performed next. If bit 0 is a 0 bit, the next instruction is performed.

1. If a Jump Low Bit instruction is used to determine whether the value in Aa is an even or an odd integer, consideration must be given to the sign of the value.

5.11.13. Jump Modifier Greater and Increment - JMGI 74,12

Jump to U if $(Xa)_{17-0} > 0$; go to NI if $(Xa)_{17-0} \leqslant 0$; always $(Xa)_{17-0} + (Xa)_{35-18} \rightarrow Xa_{17-0}$.

If the signed number in bits 17 through 0 of the X-register specified by the a-field is greater than zero (bit 17 is a 0 bit and the number does not consist of all 0 bits), the instruction at location U is performed next. If the number is less than or equal to zero (bit 17 is a 1 bit or the number consists of all 0 bits), the next instruction is performed. In either case, the signed number in bits 35 through 18 of the X-register is added to the signed number in bits 17 through 0, and the sum is stored in bits 17 through 0 of the X-register.

1. The number in Xa_{17-0} before the addition is tested, rather than the number resulting from the addition.

2. If a = x and h = 1, the specified index register is effectively modified only once for each execution of the instruction.

5.11.14. Jump Overflow - JO 74,14; a = 0

Jump to U if D1 = 1; go to NI if D1 = 0.

Where the a-field is an extention of the f- and j-fields.

If the overflow designator (D1) is one, the instruction at location U is performed next. If D1 is zero, the next instruction is performed.

1. Performing the Jump Overflow instruction does not change D1.

5.11.15. Jump Floating Underflow - JFU 74,14; a = 1

Jump to U if D21 = 1, clear D21; go to NI if D21 = 0.

If the characteristic underflow designator (D21) is one, the instruction at location U is performed next and D21 is cleared by the instruction. If D21 is zero, the next instruction is performed.

5.11.16. Jump Floating Overflow - JFO 74,14; a = 2

Jump to U if D22 = 1, clear D22; go to NI if D22 = 0.

If the characteristic overflow designator (D22) is one, the instruction at location U is performed next and D22 is cleared by the instruction. If D22 is zero, the next instruction is performed.

5.11.17. Jump Divide Fault - JDF 74,14; a = 3

Jump to U if D23 = 1, clear D23; go to NI if D23 = 0.

If the divide fault designator (D23) is one, the instruction at location U is performed next and D23 is cleared by the instruction. If D23 is zero, the next instruction is performed.

5.11.18. Jump No Overflow - JNO 74,15; a = 0

Jump to U if D1 = 0; go to NI if D1 = 1.

If the overflow designator (D1) is zero, the instruction at location U is performed. If D1 is one, the next instruction is performed.

1. Executing the Jump No Overflow instruction does not change D1.

5.11.19. Jump No Floating Underflow - JNFU 74,15; a = 1

Jump to U if D21 = 0; go to NI if D21 = 1; clear D21.

If the characteristic underflow designator (D21) is zero, the instruction at location U is performed next. If D21 is one, the next instruction is performed. D21 is cleared by the instruction.

5.11.20. Jump No Floating Overflow - JNFO 74,15; a = 2

Jump to U if D22 = 0; go to NI if D22 = 1; clear D22.

If the characteristic overflow designator (D22) is zero, the instruction at location U is performed next. If D22 is one, the next instruction (NI) is performed. D22 is cleared by the instruction.

5.11.21. Jump No Divide Fault - JNDF 74,15; a = 3

Jump to U if D23 = 0; go to NI if D23 = 1; clear D23.

If the divide fault designator (D23) is zero, the instruction at location U is performed. If D23 is one, the next instruction is performed. D23 is cleared by the instruction.

5.11.22. Jump Carry - JC 74,16

Jump to U if D0 = 1; go to NI if D0 = 0.

If the carry designator (D0) is one, the instruction at location U is performed next. If D0 is zero, the next instruction is performed.

1. The contents of the a-field is ignored.
2. Performing the Jump Carry instruction does not change D0.

5.11.23. Jump No Carry - JNC 74,17

Jump to U if D0 = 0; go to NI if D0 = 1.

If the carry designator (D0) is zero, the instruction at location U is performed next. If D0 is one, the next instruction is performed.

1. The contents of the a-field is ignored.
2. Performing the Jump No Carry instruction does not change D0.

5.12. Logical Instructions

The three logical operations are the Logical Inclusive OR (referred to as the Logical OR and symbolized by OR), the Logical Exclusive OR (symbolized by XOR); and the Logical AND (symbolized by AND). Each of these instructions uses two input operands. One input operand is obtained from location U and the other from an A-register. Table 5-1 lists the four possible combinations of the two bits from any bit position of the two input operands and the result produced for that bit position for each of the three basic operations.

Table 5-1. Truth Table for Logical OR, XOR, and AND

Input Bits		Output (Result) Bit		
First Operand	Second Operand	OR	XOR	AND
0	0	0	0	0
0	1	1	1	0
1	0	1	1	0
1	1	1	0	1

The Masked Load Upper instruction performs a compound logical operation; the contents of selected bit positions of one operand are merged with the contents of the remaining bit positions of a second operand.

5.12.1. Logical OR - OR 40

(A) OR (U) → A+1

The contents of Aa is transferred to the arithmetic section. The contents of U is transferred to the arithmetic section under j-field control. A 36-bit result is formed in the arithmetic section, as follows:

- The result contains a 1 in each bit position for which the corresponding bit position of either (or both) of the input operands contains a 1.
- The result contains a 0 in each bit position for which the corresponding bit position of both input operands contains a 0.

The result is stored in Aa+1.

5.12.2. Logical Exclusive OR - XOR 41

(A) XOR (U) → A+1

The contents of Aa is transferred to the arithmetic section. The contents of U is transferred to the arithmetic section under j-field control. A 36-bit result is formed in the arithmetic section, as follows:

- The result contains a 1 in each bit position for which the corresponding bit position of either (but not both) of the input operands contains a 1.
- The result contains a 0 in each bit position for which the contents of the corresponding bit position of the input operands are both 0 or both 1.

The result is stored in Aa+1.

5.12.3. Logical AND - AND 42

(A) AND (U) → A+1

The contents of Aa is transferred to the arithmetic section. The contents of U is transferred to the arithmetic section under j-field control. A 36-bit result is formed in the arithmetic section, as follows:

- The result contains a 1 in each bit position for which the corresponding bit position of both input operands contains a 1.
- The result contains a 0 in each bit position for which the corresponding bit position of either (or both) of the input operands contains a 0.

The result is stored in Aa+1.

5.12.4. Masked Load Upper - MLU 43

[(U) AND (R2)] OR [(A) AND NOT (R2)] → A+1

The contents of Aa and R2 are transferred to the arithmetic section. The contents of U is transferred to the arithmetic section under j-field control. A 36-bit result is formed in the arithmetic section, as follows:

- The result contains a 1 in each bit position for which the corresponding bit position of the operand from U and the operand from R2 both contain 1 bits.
- The result contains a 1 in each bit position for which the corresponding bit position of the operand from Aa and the ones complement of the operand from R2 both contain 1 bits.
- The result contains 0 bits in the remaining bit positions.

The result is stored in Aa+1.

1. The desired value must be loaded in R2 (mask register) by an instruction preceding the Masked Load Upper instruction.

5.13. Miscellaneous Instructions

Each of the eight following instructions is classed as miscellaneous.

5.13.1. Load DR Designators - LPD 07,14

$U_{6,5,3-0}$ → Designator register:

Bit 0 → D4 Bit 3 → D10

Bit 1 → D5 Bit 5 → D17

Bit 2 → D8 Bit 6 → D20

Bits 0, 1, 2, 3, 5, and 6 of U are transferred to the designator register bits (see 8.2.1); D4, D5, D8, D10, D17, and D20, respectively. These are the only designator bits which can be changed by a user program.

5.13.2. Store DR Designators - SPD 07,15

Designator register bits → U_{6-0}; zeros → U_{17-7}

D4 → Bit 0 D12 → Bit 4

D5 → Bit 1 D17 → Bit 5

D8 → Bit 2 D20 → Bit 6

D10 → Bit 3

D20, D17, D12, D10, D8, D5, and D4 of the designator register (see 8.2.1) are transferred to bit positions 7–0 of U, respectively. The upper half of the operation location is unaffected.

5.13.3. Execute - EX 72,10

Execute the instruction at U.

The P–register is incremented provided the instruction was addressed by the contents of the P–register. The instruction at location U is transferred to the control section to replace the Execute instruction as the next instruction to be performed.

1. The contents of the a–field are ignored.
2. The remote instruction, specified by U, is always obtained from a storage location.
3. Execute instructions may be cascaded; that is, the instruction in the remote location may be an Execute instruction.
4. The P–register is incremented only once, when the original Execute instruction is obtained for execution.
5. Generally, an interrupt cannot occur between the time an Execute instruction is started and the instruction (or instructions) it leads to has been completed except when an Execute instruction leads to a repeated instruction (see 5.3.8 and 5.6). An interrupt cannot occur between the start of the Execute instruction and the completion of the initial stage of the repeated instruction. The interrupt, however, can cause initiation of a termination stage immediately following completion of the initial stage or any time thereafter in order to permit the interrupt to occur.
6. If an Execute instruction leads to a repeated instruction, index register incrementation should not be specified for the Execute instructions or for any indirect addressing sequence involved (see 5.3.8. note 6, and 5.6).

5.13.4. Executive Request - ER 72,11

Generate Executive Request interrupt

An Executive Request interrupt is generated.

1. A Guard Mode/Storage Limits interrupt will occur if indirect addressing is specified (i = 1, the relocation and storage suppression designator, D7 = 0) and the operand address causes a storage limits violation.

2. The contents of the a-field is ignored.

5.13.5. Test and Set - TS 73,17; a = 0

If $(U)_{30} = 1$, Generate Test and Set interrupt; if $(U)_{30} = 0$, go to NI, if $U \geq 200$, then $01_8 \rightarrow U_{35-30}$; $(U)_{29-0}$ unchanged.

An operand fetch (from GRS if $U < 200$; from storage if $U \geq 200$) is initiated to read the operand specified by the operand address. If bit 30 of the operand is zero, the next instruction is performed. If $U \geq 200$ and bit 30 of the operand is zero, then 01_8 is written into bits 35 through 30 of the storage operand. Bits 29 through 0 at location U are neither examined nor altered. When $U < 200$ no write back will occur.

5.13.6. Test and Set and Skip - TSS 73,17; a = 1

If $(U)_{30} = 1$, go to NI; if $(U)_{30} = 0$, skip NI, if $U \geq 200$, then $01_8 \rightarrow U_{35-30}$; $(U)_{29-0}$ unchanged.

An operand fetch (from GRS if $U < 200$; from storage if $U \geq 200$) is initiated to read the operand specified by the operand address. If bit 30 of the operand is zero, the next instruction is skipped. If bit 30 of the operand is one, the next instruction is performed. If $U \geq 200$ and bit 30 of the operand is zero, then 01_8 is written into bits 35 through 30 of the storage operand. Bits 29 through 0 at location U are neither examined or altered. When $U < 200$ no write back will occur.

5.13.7. Test and Clear and Skip - TCS 73,17; a = 2

If $(U)_{30} = 0$, perform NI; if $(U)_{30} = 1$, skip NI, if $U \geq 200$ clear $(U)_{35-30}$; $(U)_{29-0}$ unchanged.

An operand fetch (from GRS if $U < 200$; from storage if $U \geq 200$) is initiated to read the operand specified by the operand address. If bit 30 of the operand is zero, the next instruction is performed. If bit 30 of the operand is one, the next instruction is skipped. If $U \geq 200$ and bit 30 of the operand is one, then bits 35 through 30 of the storage operand are cleared. Bits 29 through 0 at location U are neither examined or altered. When $U < 200$ no write back will occur.

5.13.8. Test and Set Alternate - TSA 73,17; a = 4

The TSA instruction is intended to allow access to data under test and set. The instruction is like Test and Set except that bit position 14 is tested, and bits 0 through 14 are set to one. An interrupt occurs if bit 14 is one when the test is performed. If $U < 200_8$, bit 14 of the GRS word is tested as above; however, the word is not modified. This instruction is used only with 494 mode capability.

5.13.9. Test and Set and Skip Alternate - TSSA 73,17; a = 5

The TSSA instruction is like Test and Set Alternate except that the next instruction is skipped if bit 14 is zero. The next instruction is not skipped if bit 14 is one, rather than causing an interrupt. In this respect, the TSSA instruction is like the Test and Set and Skip instruction. If $U < 200_8$, bit 14 of the GRS word is tested as above; however, the word is not modified. This instruction is used only with 494 mode capability.

5.13.10. No Operation - NOP 74,06

Proceed to next instruction.

The NOP instruction ensures that there is an interval between the end of the instruction that precedes it and the start of the one that follows it.

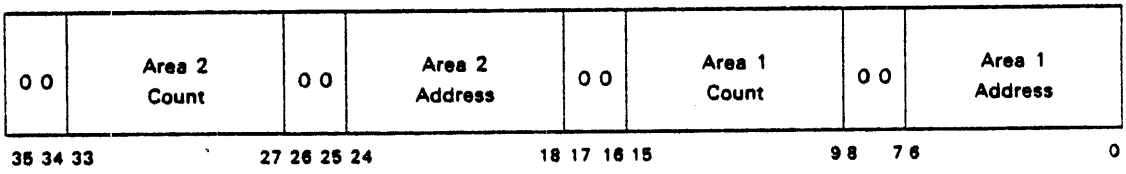
1. The contents of the a-field is ignored.

2. The only effects that the values in the x-, h-, i-, and u-fields can have on the operation is the index register incrementation obtained when x ≠ 0 and h = 1, and the indirect addressing delay introduced when i = 1 and the relocation and storage suppression designator (D7) = 0. Indirect addressing can cause a guard mode if the relative address generated from X and U does not fall within limits on any of the bank descriptors.

5.13.11. Store Register Set - SRS 72,16

Aa contains an address and count for each of two GRS areas. These areas are stored consecutively, starting at the location specified by the operand address of the instruction. If either or both count values are zero, no transfer occurs to the respective area(s). Relative addresses less than 200_8 are treated as storage addresses, not GRS addresses.

The following is the format of Aa for this instruction:



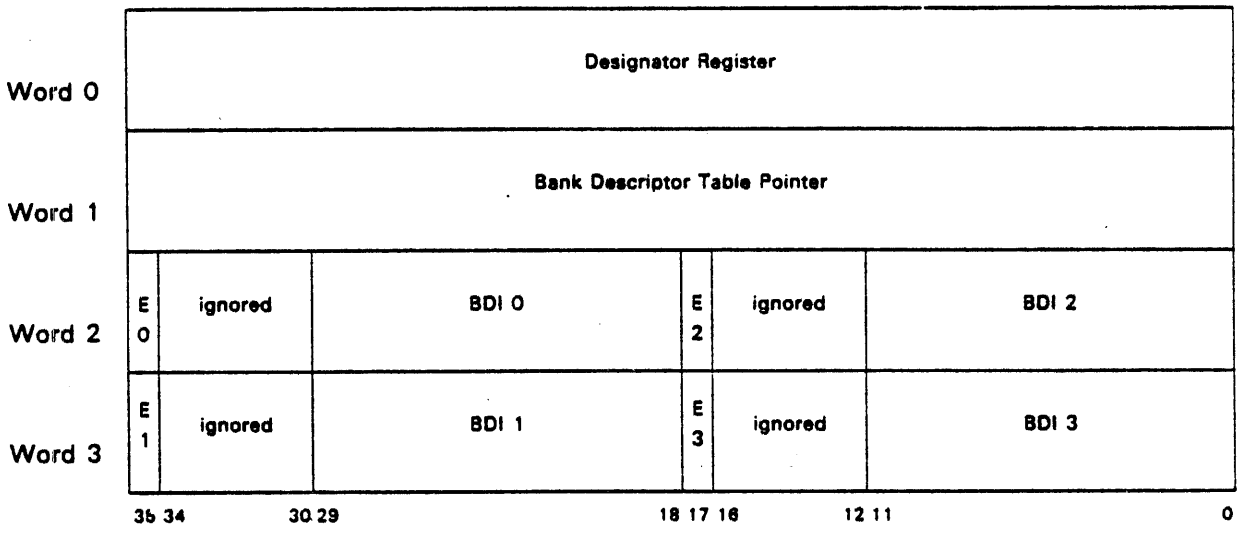
5.13.12. Load Register Set - LRS 72,17

The format of Aa and the operation of the instruction are like that of SRS, except that information is transferred from the location specified by the operand address to the area specified by Aa. Relative addresses less than 200_8 are treated as storage addresses, not GRS addresses.

5.13.13. Test Relative Address - TRA 72,15

The TRA instruction provides a means to determine whether a specific relative address is within a given relative addressing range. The operand address is the first word of a 4-word packet defining

an addressing environment to be used in testing the relative address. The packet contains a designator register, bank descriptor table pointer, four bank descriptor indexes, and E bits, in the following format:



The four E bits within the packet determine whether the BDT pointer in the packet (E = 0) or the EXEC BDT pointer (E = 1) is to be used with the appropriate BDI to reference the respective bank descriptor.

The relative address to be tested is contained in Xa_{17-0}. This relative address is translated into an absolute address within the addressing environment specified by the above packet. Relative addresses less than 200_8 are treated as storage addresses, not GRS addresses. There is no check for table length violation.

To determine the order of descriptor usage during testing, the bank descriptor register (BDR) selector designator (D12) is used. When D12 = 1, the order is 1, 3, 0, 2. When D12 = 0, the order is 0, 2, 1, 3. These orders are followed regardless of D35. D12 is obtained from the current D12 in the hardware designator register if the address for the designator register is 44_8 (U = 44_8); otherwise, D12 is obtained from the designator register in the packet.

The results of this instruction are stored in Xa and indicated by skip or no skip. If the relative address tested is within limits, the number of the bank descriptor register within whose limits the relative address exists, is stored in Xa_{34-33}, and the absolute address produced is stored in Xa_{23-00}. If the relative address does not fall within any limits, Xa is cleared to zero and the next instruction is executed. If the relative address tested is within limits, the write protect bit of the bank descriptor within whose limits the relative address exists is tested. If it is zero, the next instruction is skipped; if it is one, the next instruction is executed.

The bank descriptors which are fetched from storage are loaded, one at a time, into hardware BDR0 where the limits check is done. Since these test bank descriptors are loaded into BDR0, the resident bank descriptor will be destroyed. The BDR0 must be returned to the original values once all the testing is done. To do this the hardware will fetch the current BD0 from storage using the current index values and pointers in the GRS at the start of the TRA. Once the current BD0 comes from storage, it is loaded into GRS addresses 66 and 67. Then at the end of the TRA, these GRS addresses are read up and loaded into BDR0.

5.13.14. Increase Instructions - XX 05; a = 10-17

The operand specified by the operand address is transferred under j-field control to the arithmetic section, increased by a value specified by the a-field control to the arithmetic section, and stored under j-field control in the location specified by the operand address; the operation is performed under storage lock (test and set). If the initial operand or the result is zero, the next instruction is executed; otherwise, the next instruction is skipped. If 077 +1 (per j transfer) is not equal to zero, a skip does not occur. The following values may be selected by the a-field:

mnemonic	a-field	increase value
INC	10	+1
DEC	11	-1
INC2	12	+2
DEC2	13	-2
ENZ	14-17	0 (-0 is changed to +0 for sign-extended operands)

The increase and zero test operations depend on the j-field values to some degree. Certain j-field values extend or interpret the sign of the operand (W, XH1-XH2, T1-T3); for these values, the increase is a ones complement, sign-extended operation, and either positive zero or negative zero satisfies the zero test. The remaining j-field values do not consider the sign of the operand (H1-H2, Q1-Q4, S1-S6); for these values, the increase is a twos complement, field-size operation, and only positive zero satisfies the zero test.

5.14. Byte Instructions

This class of instructions is designed to permit transference, translation, comparison, testing, and arithmetic computation of data in the form of predetermined bit patterns (e.g., half words, third words, quarter words, and sixth words) referred to as bytes.

There are a total of 15 distinct instructions that perform the various multiword (byte string) operations noted above. These instructions may be arranged under three functional groups:

1. Instructions that involve byte transfers and manipulations between one storage location and another.

2. Instructions that permit the mutual transference and manipulation of data among storage and various control and arithmetic registers.

3. Instructions that perform decimal arithmetic addition and subtraction operations.

These instructions operate on strings of characters (byte strings) under control of J-registers and staging registers. The J-registers are implicitly addressed by the instruction and are used to index through the byte strings. One J-register is provided for each of four possible byte strings used by an instruction. These registers, J0 through J3, are located in GRS addresses 106_8 - 111_8 for user programs, or 126_8 - 131_8 for Executive programs. Figure 5-1 shows the J-register format, including the function of the various fields.

Staging and control information necessary to handle the byte strings are held in staging registers. Three R-registers (R_3, R_4, and R_5 of GRS), designated as SR1, SR2, and SR3, respectively, are used

for this purpose. The information stored in these registers provides the capability of interrupting the performance of certain instructions. The actual information stored may vary from one instruction to the next. See the individual instructions for use of the staging registers.

Byte string addressing is accomplished through use of the instruction's u-field, index registers specified by the x-field of the instruction, and the Ow (offset in words) field of the appropriate J-register. The address of byte string 0 (designated SJ0), for example, is given by summing the contents of u, Xx, and the Ow field of J0 ($U + Xx + J0_{Ow}$). The address of byte string 1 (SJ1) would be given by ($U + Xx + 1 + J1_{Ow}$), etc. A particular byte within the word of a byte string is pointed to by the Ob-field (offset in bytes) of the J-register. Byte strings may begin on any word-fraction boundary compatible with byte size; i.e., strings of 6-bit bytes must be located on sixth-word boundaries, 9-bit bytes on quarter-word boundaries, etc. The length of a byte string, in number of bytes, is stored in staging register SR3. The length of byte string 0 (designated LJ0) is stored in bit locations 35-27 of SR3, the length of byte string 1 (LJ1) in bit locations 26-18 of SR3, and the length of byte string 2 (LJ2) in bit locations 17-9 of SR3. Any, all, or none of these values may apply for a particular instruction.

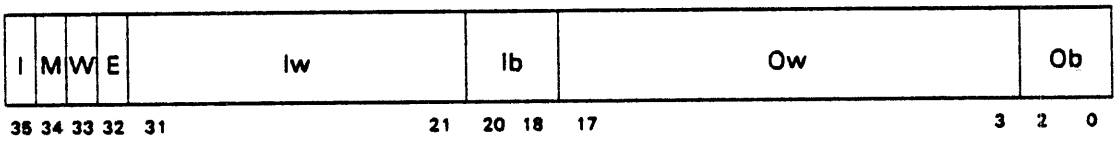


Figure 5-1. J-Register Format

Field	Function
I	J-register modifier bit; used with the h-bit of the instruction to control J (I=1) or X(I=0) register modification.
M	Mode 6/9-bit modulus: 0 = 9-bit mode (ASCII) 1 = 6-bit mode (Fieldata)
W	Width 6-12 or 9-18 bits: 0 = 6/9 bits 1 = 12/18 bits
E	a) for 33,03 E = Translate: 0 = translation 1 = no translation b) for all other byte instructions E must be zero. c) for character addressing (non-byte instructions): 0 = no sign extension 1 = sign extension

Iw	Increment in words
Ib	Increment in bytes
Ow	Offset in words
Ob	Offset in bytes

The direction in which each instruction progresses through its operand byte strings is specified per instruction in the J-register. The increment word (Iw) and increment byte (Ib) fields of the J-register are used during instruction execution to update the effective byte address. The effective value of Iw and Ib may be either ±1, the actual value loaded into the register by the program depends on the byte length being used. The value of Iw must be ±0 and have the same sign as Ib. Therefore, Iw is effectively Ib sign-extended (Iwb).

Table 5-2 gives the values of Iwb for +1 and -1 effective increments for 6, 9, 12, and 18-bit bytes.

Some of the extended-sequence byte-manipulation instructions are designed to permit their interruption during their execution. However, interrupts are accepted only following the store or compare phase of the instruction. As the instruction comoletes each of these phases, a check is made to see if an interrupt is waiting to be processed. If an interrupt request is current, it is acknowledged and processed immediately. When the instruction is again activated, the interrupt control bits are decoded, and control is returned to the appropriate phase of execution. There are three bits (29-27) in SR1 that are available for interrupt control, thus providing up to 7 types of interrupt classification within an instruction.

Table 5-2. J-Register Increment Field Values

For a Byte Length of	And an Effective Increment of	The Value of Ib Must be
6 bits	+1	+1
6	-1	-1
9	+1	+2
9	-1	-2
12	+1	+2
12	-1	-2
18	+1	+4
18	-1	-4

There are seven restrictions on byte addressing that should be noted:

1. The result of an instruction performed on overlapping byte strings is undefined.
2. A byte string may not wrap around its J-register offset field; i.e., Ow cannot be incremented through its maximum value of 77777_8 or decremented through its minimum value of 00000_8.
3. Normal address limits violation detection and interrupt will be in effect.
4. All instructions utilizing the J-registers must have their operands located in storage.
5. The h-field of all byte instructions must be set to 1. This is set automatically by the assembler when a byte mnemonic is encountered.

6. The I-field of all J-registers used must be set to 1.

7. The E-field of all J-registers used must be set to 0 for all byte instructions except 33, 03.

The 33, 03-04; 33, 10-11; 33, 14-15; and 37, 06-07 instructions will store a 7-bit status word in $SR3_{17-9}$ either upon successful completion of the instruction or upon detection of an error condition which prevents completion of the instruction. A definition of the 7 status bits is contained in Table 5-3.

Successful completion of an instruction will result in the storing of an all-zero word, except for the cases of a decimal-add overflow (37, 06-07) or a missing mantissa field (33, 14-15) or roundup status (33, 16-17).

When dealing with 9-bit bytes, the ASCII format shall be accepted and only ASCII is generated when used for operations involving signed numeric-byte strings. An ASCII byte is the eight lowest-order bits in a quarter word. The byte is divided into a 4-bit zone and a 4-bit digit; the zone is the most significant part of the byte. The sign convention adopted for a byte string is called "trailing-included sign format," i.e., the sign of the byte string is contained in the zone (Z) portion of the least significant byte.

There are three exceptions to the "trailing-included sign convention". The Byte-to-Single Floating Conversion (fj = 33, 14) and Byte-to-Double Floating Conversion (fj = 33, 15) instructions use a separate non-included sign byte with the byte string. This byte is simply a "+" or "-" character.

Table 5-4 gives the binary coding for the plus and minus signs to be used in ASCII and Fieldata coding. The hardware checks for a minus sign in arithmetic operations. If the sign of the arithmetic operations is not minus, then the result is assumed to be plus. The types of signs accepted and generated by each of the byte-manipulation instructions are listed in the table.

5.14.1. Byte Move - BM 33,00

Transfer LJ1 bytes from source string to receiving string. Truncate or fill.

The BM instruction transfers LJ1 bytes from a source string starting at address SJ0 to a receiving string starting at address SJ1. The byte string at address SJ0 contains LJ0 bytes; the byte string at address SJ1 contains LJ1 bytes. If LJ1 is less than LJ0, the move will be truncated when LJ1 bytes have been transferred. If LJ1 is greater than LJ0, then (LJ1-LJ0) fill bytes will be added in the trailing positions of the byte string located at address SJ1. The contents of $SR2_{17-0}$ are used as the fill byte.

When byte strings of different byte size are transferred, the receiving string determines how many bits from each source string byte will be accepted. For example, if SJ1 is in the 9-bit mode and SJ0 is in 6-bit mode, the three leading bits of the SJ1 byte are made zero. If SJ1 is in the 6-bit mode and SJ0 is in the 9-bit mode, only the six least significant bits of the SJ0 byte are accepted, the rest being lost.

Table 5-3. Byte Status Word

Status Bit	Type of Error	Instruction	Condition Detected
Bit 0 Set	Format Error	33-10,11	Byte not digit or blank (checked on all but last byte) or least significant 4 bits of last byte greater than 9.
		37-06,07	Byte not digit (checked on all but first byte) or least significant 4 bits or first bytes greater than 9.
		33-14,15	a. Two signs in string not separated by at least one non-blank character. b. Two decimal points in mantissa c. Significant character not found. d. Illegal character in string. e. Illegal character in exponent. f. Decimal point last character and no digit in string.
Bit 1 Set	Underflow	33-14	Magnitude of input too small to represent in single-precision floating-point number.
		33-15	Magnitude of input too small to represent in double-precision floating-point number.
		33-14,15	Exponent negative and power of ten too small to represent double-precision floating-point format.
Bit 2 Set	Overflow	33-10	Magnitude of input too large to represent in 35 binary bits.
		33-11	Magnitude of input too large to represent in 71 binary bits.
		37-06,07	Decimal-add overflow.
		33-14	Magnitude of input too large to represent in single-precision floating-point.
		33-15	Magnitude of input too large to represent in double-precision floating-point.
		33-14,15	Mantissa interpreted as integer too large to represent in 60 binary bits.
Bit 3 Set	Decimal Point Error	33-14,15	a. Decimal point count greater than 31. b. Two decimal points in mantissa. c. Decimal point last character and no digit in string.
Bit 4 Set	No Significant Character Found	33-14,15	a. Bits 0, 3 set and significant character not read yet. b. Mantissa field does not contain at least one digit (note that a blank following a decimal point is considered a digit). c. String does not contain at least one nonblank and nonsign character.
Bit 5 Set	Exponent Found or Byte Roundup	33-14,15	a. Bits 0, 1, 2, 3 set and exponent field detected.
		33-16,17	b. Byte 10 (33,16) or 19 (33,17) is greater than four.
Bit 6 Set	Mode Error	33-10,11	6- or 9-bit mode not selected (W bit) on one of the following instructions.
		33-14,15	
Bit 7 Set	Byte Compare	33-03,04	Set if non-compare encountered during instruction.

Both the values LJ1 and LJ0 are reduced by one following each byte move. This instruction is terminated when the value of LJ1 equals zero (LJ1 = 0).

1. This instruction is interruptible after each store operation.
2. The lwb fields in J0 and J1 must be loaded with effective values of ± 1, depending on mode and width.
3. The desired fill byte must be loaded in $SR2_{17-0}$.

Table 5-4. Byte String Sign Codes

Character Code Formats	Sign Conventions	
	+	-
1. ASCII Included (Zone portion)	1010	1011
2. Fieldata Included	11	10
3. ASCII Separated (Entire byte)	00101011	00101101
4. Fieldata Separate	100010	100001

5.14.2. Byte Move With Translate - BMT 33,01

Translate and transfer LJ1 bytes from source string to receiving string. Truncate or fill.

The BMT instruction translates and transfers LJ1 byte from byte string SJ0 to byte string SJ1. The translation and transfer process uses byte string SJ2 as a translation table for byte string SJ0. That is, each byte of the string SJ0 is used as an index to a byte in string SJ2. The SJ2 byte thus addressed is transferred to the byte string SJ1. If the value of LJ1 is less than LJ0, the transfer terminates when LJ1 bytes have been processed. If the value LJ1 is greater than LJ0, then (LJ1-LJ0) translated fill bytes are placed in the trailing positions of SJ1. The contents of $SR2_{17-0}$ are used to index the fill byte.

When byte strings of different byte size are transferred, the receiving string determines how many bits from each byte of the source string will be accepted. If SJ1 is in the 6-bit mode and SJ2 is in the 9-bit mode, only the six least significant bits of SJ2 byte are accepted, the rest being lost.

The translation table pointer register, J2, must be in the 9- or 18-bit mode. This restriction does not prevent Fieldata translations, but it requires that the translation table bytes are either 9- or 18-bit entries. Both the values LJ1 and LJ0 are decreased by one following each byte translation. When the value of LJ1 is equal to zero (LJ1 = 0), the instruction is terminated.

The fill byte referenced by $SR2_{17-0}$ must be preloaded left shifted one bit if MW = 0 in J0 (see Figure 5-1), indicating the source string is 9-bit bytes, and must be preloaded left shifted two bits if BL = 1 in J0, indicating the source string is 18-bit bytes.

1. This instruction is interruptible after each byte store operation.
2. The lwb fields of J0 and J1 must be loaded with effective values of ± 1, depending on mode and width.

5.14.3. Byte Translate and Compare - BTC 33,03

Optionally, translate and compare LJ0 bytes from SJ0 with LJ1 bytes from SJ1; terminate the instruction on not equal or when both LJ0 and LJ1 equal zero; when:

(A) > 0; string SJ0 > SJ1

(A) = 0; string SJ0 = SJ1

(A) < 0; string SJ0 < SJ1

The BTC instruction optionally translates and compares LJ0 bytes of string SJ0 with the optionally translated LJ1 bytes of string SJ1. String SJ2, starting at address $(u+(X+2)+J2_{ow})$, is used as the translation table for strings SJ0 and SJ1 when the corresponding E-bit is zero. A one in the corresponding E-bit inhibits translation. Thus, a translation can be made on either or both strings. If no translation is desired, the Byte Compare instruction (33,04) should be used. The comparison is made by subtracting the optionally translated SJ1 byte from the optionally translated SJ0 byte and storing the result in register Aa. If the contents of Aa is zero (Aa = 0), then the next pair of bytes are translated or not, according to the content of (E) and compared. If the contents of Aa is not zero (Aa ≠ 0), or if both of the strings SJ0 and SJ1 have a value of LJ1 = 0 and LJ0 = 0, then the instruction is terminated. The values of LJ1 and LJ0 are always decreased by one, and the J0- and J1-registers are increased or decreased by one, depending upon the direction addressed.

When the instruction termination occurs, the relative value of an SJ0 string in respect to the value of an SJ1 string may be determined as follows:

- If the contents of the Aa-register is positive (A > 0), then the SJ0 string is greater than the SJ1 string (after optional translations).
- If the contents of the Aa-register is zero (Aa = 0), then the SJ0 string is equal to the SJ1 string (after optional translations).
- If the contents of the Aa-register is negative (Aa < 0), then the SJ0 string is less than the SJ1 string (after optional translations).

1. If either string SJ0 or string SJ1 is depleted before the other, trailing fill characters are added to the shorter string.
2. The fill byte for string SJ0 is contained in $SR1_{17-0}$ and the fill byte for string SJ1 is contained in $SR2_{35-18}$.
3. The fill bytes in SR2 must be preloaded left shifted one bit if MW = 0 in J0 indicating the source string is 9-bit bytes and must be preloaded left shifted two bits if MW = 1 in J0 indicating the source string is 18-bit bytes (see Figure 5-1).
4. This instruction is interruptible after each test.
5. The lwb-fields of J0 and J1 must be loaded with effective values of ± 1, depending on mode and width (see Table 5-2).
6. If the byte strings do not compare, then bit 7 of the byte status word is set (see Table 5-3).

5.14.4. Byte Compare - BC 33,04

Compare LJ0 bytes from string SJ0 with LJ1 bytes from string SJ1; terminate instruction on not equal or when both LJ0 and LJ1 are zero.

The corresponding string SJ1 byte is subtracted from the string SJ0 byte, the result is stored in Aa, and a zero test is performed. The value of LJ1 and LJ0 are always decreased by one, and the J0– and J1–registers are updated. If the contents of the Aa is zero, the next pair of bytes are tested. If the value of Aa is nonzero, or both LJ1 and LJ0 are zero (i.e., the longer string has been depleted), the BC instruction is terminated.

When the instruction termination occurs, the relative value of an SJ0 string in respect to the value of an SJ1 string may be determined as follows:

- If the contents of the Aa–register is positive (Aa > 0), then the SJ0 string is greater than the SJ1 string.
- If the contents of the Aa–register is zero (Aa = 0), then the SJ0 string is equal to the SJ1 string.
- If the contents of the Aa–register is negative (Aa < 0), then the SJ0 string is less than the SJ1 string.

1. If either string SJ0 or string SJ1 are depleted before the other, trailing fill characters are added to the shorter string. The fill byte for the string SJ0 is contained in $SR2_{17-0}$ and the fill byte for string SJ1 is contained in $SR2_{35-18}$.
2. This instruction is interruptible after each compare.
3. The lwb–fields of J0 and J1 must be loaded with effective values of ± 1, depending on mode and width (see Table 5–2).
4. If the byte strings do not compare, then bit 7 of the byte status word is set (see Table 5–3).

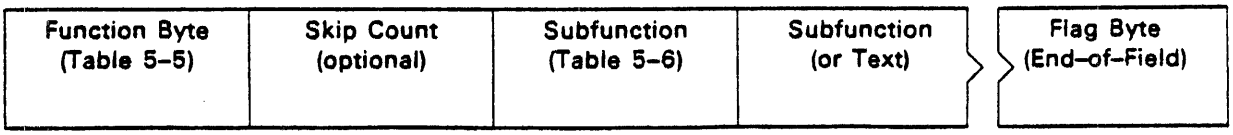
5.14.5. Edit - EDIT 33,07

Edit byte string SJ0 and transfer to string SJ1 under the control of string SJ2.

A source byte string (string SJ0) specified by the u–field of the Edit instruction, using registers Xx and J0, is edited into a receiving byte string (string SJ1) specified by the u–field of the instruction, X(x+1), and J1. Specific editing commands are coded within a control byte string (string SJ2) whose location is designated by the J2 register, the u–field of the instruction, and the register X(x+2). The control stream commands are designed to duplicate all of the functions of the PICTURE clause of the COBOL compiler. Therefore, the main use of the Edit instruction is to make the appropriate editing changes to a numeric byte string for output to the printer. For example, blanking–out the leading zeros, adding a "$" character or the appropriate sign code, inserting commas or a decimal point within the number, or appending a descriptor word such as "CR" or "DB."

The following information describes and summarizes the basic operational steps of the Edit instruction.

A typical field in the control stream (string SJ2) will contain the following elements:



The function byte specifies control information for the whole field following it (see Table 5–5). One function of this byte is to specify whether there is a skip count or not. If there is a skip count, it is given in the next byte. The rest of the field contains a series of subfunction bytes and text bytes. The subfunction bytes are those described in Table 5–6 and specify operations to be performed as the source string is edited into the receiving string. The text bytes are bytes similar to the source string bytes which may be edited into the receiving string. The last subfunction byte in the field is the flag byte which establishes the end–of–file action. The flag byte may be followed by another field starting with a function byte, or a second flag byte indicating termination of the Edit instruction.

Operation of the Edit instruction is based on performing a sequence of field "microprograms" defined by the control string. A field scan is established when the instruction is initiated or when the initiation of a new field occurs and results in certain "function initiation" actions based on the contents of the function byte. The first control stream byte must be the function byte. It will be stored in staging register $SR1_{26-18}$. If a skip count is required, as indicated by the function byte, the second control stream byte contains the skip count and is transferred from the control stream to staging register $SR1_{17-9}$. Next, the J1–register is saved in J3. This saves the position of the first byte of the receiving string for use if the "blank–if–zero" command (bit 0 of function byte is set) is required when the end of the field is encountered. Finally, the skip count ($SR1_{17-9}$) is used to skip the indicated number of bytes in the receiving string. This is done by updating J1 position 0wb as many times as the value in $SR1_{17-9}$ (skip count).

When the function initiation actions are completed for this field, the first subfunction byte is transferred from the control stream to $SR1_{8-0}$ for interrogation. The subfunction byte is transferred from the control stream to $SR1_{8-0}$ for interrogation. The subfunction and text bytes are sequentially interrogated until a "flag" (end–of–field) subfunction byte is encountered. At this point the end–of–field action is completed and another function is initiated. This process continues until two "flag" bytes are encountered together indicating termination of the instruction. A detailed description of the function byte, subfunction, and text bytes follows.

5.14.5.1. Function Byte

The interpretation of the function byte is given in Table 5–5. A more detailed descripton of each bit position follows:

Table 5–5. Function Byte Interpretation

Function Byte		
Bit	0	1
5	No Skip Count	Skip Count Follows
4	Fixed Sign	Floating Sign
3	Fixed Symbol	Floating Symbol
2	Sign = Minus or Fill	Sign = Minus or Plus
1	Edit – No Sign Action	Sign Action on Edit
0	Normal Edit	Blank if Zero

- Function Bit 5 – A skip mechanism is included that allows the programmer the option of ignoring a series of bytes in the receiving string. The skip count is placed in position $SR1_{17-9}$ during function initiation and specifies the number of bytes to be skipped in the receiving string before the first subfunction byte is interrogated. The maximum value allowed is 63_{10} for either the 6- or 9-bit mode.

- Function Bit 4 – If fixed sign is indicated, an appropriate sign byte (as specified by function bit 2) is placed in the receiving string position specified by $SR2_{35-18}$. $SR2_{35-18}$ must be loaded by the subfunction "sign-position indicator" discussed in 5.14.5.2. If floating sign is indicated, an appropriate sign byte is placed in the receiving string position specified by $SR2_{17-0}$. $SR2_{17-0}$ is loaded by the subfunctions "digit select" or "significance start indicator" as discussed in 5.14.5.2. In either case, a fill byte is transferred to the receiving string where the sign will be. The sign bytes are specified by the programmer in SR3 and are transferred to the receiving string when a "flag" subfunction byte is interrogated (end-of-field action). This bit has no meaning unless function bit 2 (sign action on edit) is set to 1.

- Function Bit 3 – If this bit is a 1 bit, the symbol specified by the programmer in $SR3_{8-0}$ is transferrred to the receiving string at the position specified by $SR2_{17-0}$. Position $SR2_{17-0}$ is loaded the same way as for "floating sign" above. If both function bits 3 and 4 are set to one, the floating sign will be inserted and the floating symbol ignored. As with the sign codes, the symbol is actually inserted during end-of-field action. If function bit 3 is a zero there is no symbol inserted during end-of-field action. A symbol may still be inserted into the receiving string during subfunction interrogation with a "symbol-position indicator" subfunction (described in 5.14.5.2).

- Function Bit 2 – If function bit 2 is a 0, the sign code inserted into the receiving string is either a minus or a fill, as appropriate. If function bit 2 is a 1 bit, the sign code is either a minus or a plus. The plus, minus, and fill bytes are specified by the programmer in $SR3_{17-9}$, $SR3_{26-18}$, or $SR3_{35-27}$, respectively. This bit has no meaning unless function bit 1 (sign action on edit) is set to one.

- Function Bit 1 – If function bit 1 is a 1, the sign action indicated by bits 4, 2, and the sign of the source string is taken (i.e., a plus, minus, or fill byte is inserted into the receiving string). If bit 1 is zero, no plus or minus bytes are inserted into the receiving string. The only effect bit 4 (floating sign) would have is to insert a fill byte in the position where the floating sign byte should be.

- Function Bit 0 – The programmer has the option of leaving an all-zero receiving field or replacing it with fill bytes. The field is considered to be all-zero until a nonzero byte has been transferred from the source string by a "digit select" subfunction. Position $SR1_{31}$ is set to one when the first nonzero digit is transferred. $SR1_{31}$ is not set to one by any subfunction other than "digit select." The entire receiving string field is replaced with fill bytes during end-of-field action (see "flag" subfunction in 5.14.5.2) if function bit 0 and $SR1_{31}$ are both set to 1. The start and end of the field are indicated by J3 (loaded during function initiation) and J1, respectively.

5.14.5.2. Subfunction Byte

The interpretation of the subfunction byte is given in Table 5-6 and discussed in the following paragraphs.

Table 5-6. Subfunction Byte Interpretaion

Byte	Subfunction	Fieldata Symbol
000 000 011	Pass Byte	#
000 101 111	Significance Start Indicator	\
000 100 110	Digit Select	&
000 111 110	Symbol Position Indicator	¤
000 111 010	Sign Position Indicator	' (apostrophe)
000 101 001	Trailing Text Start Indicator	(
000 111 111	Flag (End-of-Field)	≠

- Pass Byte – If the subfunction of the control stream byte is a "pass byte," the byte currently pointed at by the source field pointer is transferred to the receiving string intact.

- Significance Start Indicator – If the subfunction of the control stream is a "significance start byte," the "significance trigger" $SR1_{34}$ is set to one. Also, if either floating sign (function bit 4 set) or a floating symbol (function bit 3 set) will be required, then the receiving field pointer, J1-register, is stored in $SR2_{17-0}$, and a fill byte $SR3_{35-27}$ is inserted in the receiving field. This fill byte is replaced by the appropriate sign byte in end-of-field processing as indicated by the function byte bit settings. If the "significant trigger" has already been set to one, this byte is ignored.

- Digit Select – If the subfunction of the control stream is a "digit select byte," the byte currently pointed at by the source field pointer and the significance trigger are examined, according to the following criteria:

  1. If the significance trigger is off, and the source byte has a zero digit portion, the receiving field will have a fill character ($SR3_{35-27}$) inserted into it.

  2. If the significance trigger is off and the source byte is not a zero,

     a. the significance trigger is set on, and

     b. if either floating sign or floating symbol will be required, the receiving field pointer (J1) is stored in $SR2_{17-0}$, and a fill byte, ($SR3_{35-27}$) is inserted in the receiving string. This fill byte is replaced by the appropriate sign byte in end-of-field processing as indicated by the function byte bit settings. The receiving field pointer (J1) is incremented, and

     c. the source byte is transferred to the receiving string.

  3. If the significance trigger is "on," then the source byte is transferred to the receiving field.

  4. If this is the first nonzero digit to be transferred from the source string, the C bit, $SR1_{31}$ is set to one for use when interrogating the "blank-if-zero" function bit in end-of-field processing.

  5. The appropriate zone code, as prescribed by the receiving string pointer (J1), is always written into the receiving string.

An exception to the "digit-select" transmission exists if the zone portion of the byte is negative sign (sign overpunch). In this case, the "N" bit is turned "on" ($SR1_{32}$) and the negative sign bits are replaced by the appropriate zone code.

- Symbol Position Indicator – If the subfunction of the control stream is a "symbol position indicator," the symbol byte ($SR3_{8-0}$) is stored in the receiving field. The setting of function bit 3 does not affect the operation of this subfunction byte.

- Sign Position Indicator – If the subfunction of the control stream is a "sign position indicator," the receiving field pointer, contained in the J1-register, is copied into the fixed-sign position pointer ($SR2_{35-18}$), and the fill character ($SR3_{35-27}$) is transmitted to the receiving field. This fill byte is replaced by the appropriate sign byte in end-of-field processing as indicated by the function byte bit settings. This subfunction byte must be used to indicate the position of the fixed sign if the function bit 4 is 0 (fixed sign).

- Trailing Text Start Indicator – If the subfunction of the control stream is a "trailing text start" byte, the trailing text trigger $SR1_{33}$ is set to one. If a negative sign has been detected in the source string scan ($SR1_{32}$ set to one), any text information encountered in the control string is now transferred to the receiving string. If $SR1_{32}$ equals zero, fill bytes ($SR3_{35-27}$) are transferred to the receiving field rather than the text bytes.

- Flag Byte (End-of-Field) – If the subfunction of the control stream is a "flag" byte, then the end-of-field action will be established. At this point the appropriate sign insertion and "blank-if-zero" command actions are done as indicated by the function byte. The net control stream byte is either a function byte starting a new field or another flag byte terminating the Edit instruction.

If the subfunction of the control stream is none of the subfunction bytes of Table 5-6, it is assumed to be a text byte. If either $SR1_{33}$ and $SR1_{32}$ are set (see Table 5-7), or $SR1_{34}$ is set and $SR1_{33}$ is not set, the text will be transferred to the receiving field. In all other cases, the fill byte ($SR3_{35-27}$) will be transferred to the receiving field.

A summary of staging register (SR1-SR3) and J-register (J0-J3) usage are given in Table 5-7.

1. The lwb-fields of J0, J1, and J2 must be loaded with effective values of +1, depending on mode and width (see Table 5-2).

2. SR3 must be loaded with the desired codes.

3. This instruction is interruptible.

Table 5–7. Summary of Staging Register and J–Register Fields

Field	Position	Function
CMP (Complement Mode)	$SR1_{35}$ (BT0)	No complement if 0, complement if 1.
ST (Significance Trigger)	$SR1_{34}$ (BT1)	Set to 1 if the control byte is a "significant start byte."
T (Trailing Text Trigger)	$SR1_{33}$ (BT2)	Set to 1 when a trailing text start indicator has been detected.
N (Negative Bit)	$SR1_{32}$ (BT3)	Set to 1 when a negative sign has been detected.
C (Control Bit)	$SR1_{31}$ (BT4)	Set to 1 when the first nonzero digit is transferred from the source string by the "digit select" subfunction.
L (Skip Bit)	$SR1_{30}$ (BT5)	Set to 1 if a skip is in progress.
I (Interrupt Bits)	$SR1_{29-27}$ (BT6–8)	Controls return after instruction interrupt.
Function	$SR1_{26-18}$ (BS2)	Contains active Edit field function.
Skip Count	$SR1_{17-9}$ (BS3)	Contains skip count to bypass receiving field.
Subfunction	$SR1_{8-0}$ (BS4)	Contains active Edit subfield function or text.
Fixed–Position Pointer	$SR2_{35-18}$ (BH0)	Acts as index modifier for pointing to byte in receiving string which will receive fixed sign or symbol. Receives contents of J1–register position 0wb.
Floating–Position Pointer	$SR2_{17-0}$ (BH1)	Acts as index modifier for pointing to byte in receiving string which will receive floating sign or symbol. Receives contents of J1–register position 0wb.
Fill Byte	$SR3_{35-27}$ (BB0)	Byte used when fill is called for.
Negative–Sign Byte	$SR3_{26-18}$ (BB1)	Byte used when negative sign insertion is specified.
Plus–Sign Byte	$SR3_{17-9}$ (BB2)	Byte used for positive sign insertion.
Symbol Byte	$SR3_{8-0}$ (BB3)	Byte used for symbol insertion.
Source Pointer	J0	Points at source byte for Edit action.
Receiving Pointer	J1	Points at byte to receive edited byte.
Control Pointer	J2	Acts as index modifier for pointing to control string (byte).
Start–of–Field Pointer	J3	Copy of contents of J1–register at start of field. Used to control blank–if–zero action.

5.14.6. Byte to Binary Single Integer Convert - BI 33,10

Convert LJO byte in string SJO into a signed binary integer in register A.

The BI instruction converts byte string SJO composed of LJO bytes, coded in either ASCII or Fieldata, into a signed binary integer in the Aa-register. The JO-register initially points to the leftmost byte in the string and is set for left-to-right incrementation. The sign must be represented in the zone of the least significant byte.

A 7-bit status word is stored in the low-order bits of $SR3_{17-9}$. An all zero word indicates successful completion of the instruction. Bit 0 set indicates a format error and is set if one of the input bytes is not a digit or a blank (checked on all but the last byte), or the least significant 4 bits of the last byte are greater than 9. Bit 2 set indicates an overflow condition and is set if the magnitude of the input string SJO is too large to be represented by 35 binary bits.

1. This instruction is not interruptible.

2. The lwb-field of JO must be loaded with effective value of +1, depending on mode and width (see Table 5-2).

3. J1, J2, and J3 are not used in this instruction.

5.14.7. Byte to Binary Double Integer Convert - BDI 33,11

Convert LJO bytes in string SJO into a signed binary integer in registers A and A+1.

The BDI instruction converts byte string SJO composed of LJO bytes, coded in either ASCII or Fieldata, into a signed binary integer in the Aa- and Aa+1-register. The JO-register initially points to the leftmost byte in the string and is set for left-to-right incrementation. The sign must be represented in the zone of the least significant byte.

A 7-bit status word is stored in the low-order bits of $SR3_{17-9}$. An all zero word indicates successful completion of the instruction. Bit 0 set indicates a format error and is set if one of the input bytes is not a digit or a blank (checked on all but the last byte), or the least significant 4 bits of the last byte are greater than 9. Bit 2 set indicates an overflow condition and is set if the magnitude of the input string SJO is too large to be represented by 72 binary bits.

1. This instruction is not interruptible.

2. The lwb-field of JO must be loaded with effective value of +1, depending on mode and width (see Table 5-2).

3. J1, J2, and J3 are not used in this instruction.

5.14.8. Binary Single Integer to Byte Convert - IB 33,12

Convert the binary integer in A to byte format and store in string SJO.

The IB instruction converts the binary integer contained in the Aa-register to a byte format and stores the results in string SJO. String SJO is LJO bytes long and the rightmost byte has the sign in the zone portion.

The converted number is right-justified and zero filled (60_8 - Fieldata 0) in the string. If string SJO is not long enough to accommodate the converted number, the remaining bytes will be truncated. The JO-register must be set for negative incrementation and point to the rightmost byte.

1. The lwb-field of J0 must be loaded with effective value of -1, depending on mode and width (see Table 5-2).

2. J1, J2, and J3 are not used in this instruction.

3. This instruction is not interruptible.

5.14.9. Binary Double Integer to Byte Convert - DIB 33,13

Convert the binary integer in A and A+1 to byte format and store in string SJ0.

The DIB instruction converts the binary integer contained in the Aa- and A+1-registers to a byte format in string SJ0. String SJ0 is LJ0 bytes long, and the rightmost byte has the sign in its zone portion.

The converted number is right-justified and zero filled (60_8 - Fieldata 0) in the string. If string SJ0 is not long enough to accommodate the converted number, the remaining bytes will be truncated. The J0-registers must be set for negative incrementation and point to the rightmost byte of string SJ0.

1. lwb-field of J0 must be loaded with effective value of -1, depending on mode and width (see Table 5-2).

2. J1, J2, and J3 are not used in this instruction.

3. This instruction is not interruptible.

5.14.10. Byte to Single Floating Convert - BF 33,14

Convert LJ0 bytes in string SJ0 into a single-length floating-point format in register A.

The BF instruction converts byte string SJ0 composed of LJ0 bytes, coded in either ASCII or Fieldata, into a single floating-point number in register Aa.

String SJ0 may have either a leading plus-sign character (+) or a leading minus-sign character (-) that indicates the sign of the mantissa. In the absence of either a plus or minus sign, the mantissa is assumed to be positive. The mantissa must be representable in 30 binary bits if interpreted as an integer and may or may not contain a decimal point character. If the mantissa does not contain a decimal point character, this character is assumed and its position is contained in $SR3_{26-18}$. If the decimal point character is present, this condition overrides the effect of $SR3_{26-18}$. The general input format is shown in Table 5-8.

The exponent, if present, follows the least significant digit of the mantissa. An exponent is indicated by an E or D character followed by a minus sign and then the digits, if the exponent is negative. If the exponent is positive, the E or D character is followed by the digits alone or by a plus sign followed by the digits. The E or D character may be optionally omitted. In this case, either a plus sign or a minus sign must precede the digits of the exponent. The exponent must be limited to two digits. If an exponent is not present, 10^0 will be assumed.

A 7-bit status word is stored in the low-order bits of $SR3_{19-7}$. An all zero word indicates successful completion of the instruction. For possible error conditions and status word indications see Table 5-3.

1. This instruction is not interruptible.

2. lwb–field of J0 must be loaded with effective value of +1, depending on mode and width (see Table 5–2).

3. J1, J2, and J3 are not used in this instruction.

4. SR3 positions 26–18 are the number of digits to the right of the decimal point.

Table 5–8. General Input Format for Byte–to–Floating Instructions

Byte String						
Fields:	B	MS	M	ED	ES	E
Valid Characters:	ΔΔΔ...	±	Digit, Decimal Point, or ΔΔΔ...	DΔΔΔ... or EΔΔΔ...	±	Digits or ΔΔΔ...

where:

B Leading blank (Δ) characters. Blanks in this field will be ignored.

MS Mantissa sign: field may include one plus (+) or minus (–) character.

M Mantissa: first digit or decimal point character indicates start of field. Blanks in this field will be interpreted as zeros.

ED Exponent delineator: field may include either a D or E character followed by blanks. Blanks in this field will be ignored.

ES Exponent sign: field may include one plus (+) or minus (–) character.

E Exponent: field may include digits or blanks. Blanks in this field will be interpreted as zeros.

NOTE:

Any of the fields may be included or omitted in the input byte string subject only to the limitations listed below:

1. The valid characters indicated for each field are the only allowable characters.

2. The mantissa sign (MS) and the exponent sign (ES) must be separated by at least one nonblank character.

3. The last character in the string cannot be a sign character.

4. Overflow will occur if the number is too large to represent in single–precision format for the Byte to Single Floating Convert (f,j=33,14) instruction or in double–precision format for the Byte to Double Floating Convert (f,j=33,15) instruction.

5. Underflow will occur if the number is too small to represent in single–precision format for the Byte to Single Floating Convert (f,j=33,14) instruction or in double–precision format for the Byte to Double Floating Convert (f,j=33,15) instruction.

6. Underflow will occur if the exponent alone is too small to represent in double-precision floating-point format.

7. The mantissa must be representable in 60 binary bits when it is interpreted as an integer; i.e., ignoring the decimal point.

8. The decimal point count (number of digits or blanks to the right of the decimal point) must not be greater than 31.

9. Two decimal points in the mantissa will be detected as an error.

10. At least one nonblank and one nonsign character must be included in the string.

11. If the last character is a decimal point, it must be preceded by at least one nonblank and one nonsign character.

5.14.11. Byte to Double Floating Convert - BDF 33,15

Convert LJO bytes in string SJO into a double-length floating-point format in registers A and A+1.

The BDF instruction converts byte string SJO composed of LJO bytes, coded in either ASCII or Fieldata, into a double-precision floating-point number in registers Aa and Aa+1. The general input format is shown in Table 5-8.

The SJO string may have either a leading plus-sign character (+) or a leading minus-sign character (-) that indicates the sign of the mantissa. In the absence of either a plus or minus sign, the mantissa is assumed to be positive. The mantissa must be representable in 60 binary bits if interpreted as an integer and may or may not contain a decimal point character. If the mantissa does not contain a decimal point character, this character is assumed and its position is contained in $SR3_{26-18}$. If the decimal point character is present, this condition overrides the effect of $SR3_{26-18}$.

If the exponent is present, it follows the least significant digit of the mantissa. The exponent is formed according to the same rules that apply to the Byte to Single Floating-Point instruction (see 5.14.10), except that there may be up to three digits in the exponent.

A 7-bit status word is stored in the low-order bits of $SR3_{19-7}$. An all zero word indicates successful completion of the instruction. For possible error conditions and status word indications see Table 5-3.

1. Floating-point interrupts (characteristic underflow/overflow) may occur during the instruction.

2. lwb-field of J0 must be loaded with effective value of +1, depending on mode and width. (See Table 5-2.)

3. J1, J2, and J3 are not used in this instruction.

4. SR3 position 26-18 is the number of digits to the right of the decimal point.

5.14.12. Single Floating to Byte Convert - FB 33,16

Convert the single-length floating-point number in A to byte format and store in string SJO.

The FB instruction converts a single-length floating-point number contained in the Aa-register to a byte string starting at address SJO. The format of the resulting string SJO contains two numbers.

The first number is a 9-byte decimal fraction that has its sign in the zone part of the least significant byte. The second number is a 2-byte exponent with its sign in the zone portion of the least significant byte. No error termination is possible for this instruction. Bit 5 of the byte status word is set to one if the tenth byte of the calculated mantissa would have been five or greater (see Table 5-3).

1. The lwb-field of J0 must be loaded with effective value of +1, depending on mode and width. (See Table 5-2.)
2. J1, J2, and J3 are not used in this instruction.
3. This instruction is not interruptible.

5.14.13. Double Floating to Byte Convert - DFB 33,17

Convert the double-length floating-point number in A and A+1 to byte format and store in string SJ0.

The DFB instruction converts a double-length floating-point number contained in the Aa- and Aa+1-registers to a byte string starting at address SJ0. The format of the resulting string is similar to that of the Single Floating to Byte Convert (see 5.14.12.) instruction except that the first number is equivalent to an 18-byte string and the second number is equivalent to a 3-byte string. The first number is the mantissa and the second number is the exponent. Each number has its sign in the zone portion of the least significant byte. No error termination is possible for this instruction. Bit 5 of the byte status word is set to one if the nineteenth byte of the calculated mantissa would have been five or greater (see Table 5-3).

1. The lwb-field of J0 must be loaded with effective value of +1, depending on mode and width. (See Table 5-2.)
2. J1, J2, and J3 are not used in this instruction.

5.14.14. Byte Add - BA 37,06

Add the LJ0 bytes in string SJ0 to the LJ1 bytes in string SJ1 and place the results in string SJ2.

The BA instruction adds byte string SJ0 (of length LJ0) to byte string SJ1 (of length LJ1) and stores the results in byte string SJ2 (of length LJ2). Only 6-bit Fieldata or 9-bit ASCII formats may be used. The sign of the SJ0 and SJ1 strings must be stored in the zone portion of the least significant byte. If the length of the resultant byte string is smaller that LJ2 digits, then the SJ2 string will be zero filled. The J-registers must point to the least significant digit and should be set for right-to-left incrementation.

1. This instruction is interruptible.
2. The lwb-fields of J0, J1, and J2 must be loaded with the effective value of -1, depending on mode and width. (See Table 5-2.)

5.14.15. Byte Add Negative - BAN 37,07

Subtract the LJ0 bytes in string SJ0 from the LJ1 bytes in string SJ1 and place the results in string SJ2.

The BAN instruction subtracts byte string SJ0 (of length LJ0) from byte string SJ1 (of length LJ1) and stores the results in byte string SJ2 (of length LJ2). Only 6-bit Fieldata or 9-bit ASCII format may be used. The sign of the SJ0 and SJ1 strings should be stored in the zone portion of the least significant byte. If the length of the resultant byte string is smaller that LJ2 digits, then string SJ2 will be zero filled. The J-registers must point to the least significant digit and should be set for right-to-left incrementation.

1. This instruction is interruptible.
2. The lwb-fields of J0, J1, and J2 must be loaded with effective value of -1, depending on mode and width. (See Table 5-2.)

5.15. Executive Instructions

The instructions in this group are intended for use by the Executive system. When designator register bits 35 and 2 are zero, the Executive repertoire is selected. This allows execution of all Executive (privileged) instructions in addition to those of the user repertoire. The Executive repertoire includes instructions for control of the processor state, interrupts, input/output, and instrumentation.

The Executive control instructions defined for the CPU are described in the following paragraphs.

5.15.1. Prevent All Interrupts and Jump - PAIJ 72,13

The CPU will not recognize certain interrupt requests received following the completion of the PAIJ instruction nor will it react to interrupt requests received following the start of the execution of the instruction.

The following interrupts may be prevented by the PAIJ instruction:

- All I/O interrupts, including those for Normal Status, Tabled Status, and Machine Check interrupts.
- Jump History Stack interrupts.
- Interprocessor interrupts.
- Dayclock and Real-Time Clock interrupts.
- Storage Check interrupts caused as the result of transfers from the storage interface unit (SIU) to Storage Units.
- Power Check interrupts.

NOTE:

This instruction does not cause the dayclock register to be replaced.

5.15.2. Load Dayclock - LDC 73,14,10

The LDC instruction causes the internal dayclock register value of the CPU to be replaced at the start of the next update cycle with the value in the dayclock location in fixed storage. (See 8.2.2 for a description of the dayclock.)

5.15.3. Enable/Disable Dayclock - EDC,DDC 73,14, 11-12

The EDC and DDC instructions enable and disable, respectively, the internal dayclock of the CPU. When a dayclock is enabled, if the dayclock is also selected, the dayclock value is stored in the dayclock location in fixed storage during each update cycle, and a dayclock interrupt request may be generated by the dayclock.

5.15.4. Select Dayclock - SDC 73,14, 13

Each CPU contains an internal dayclock. One dayclock in each application may be selected at any given time to store its value in the dayclock location in fixed storage. The operand address of the SDC instruction specifies the CPU number whose dayclock is selected for this function; note that the selected dayclock must also be enabled (via EDC).

5.15.5. Select Interrupt Locations - SIL 73,15, 00

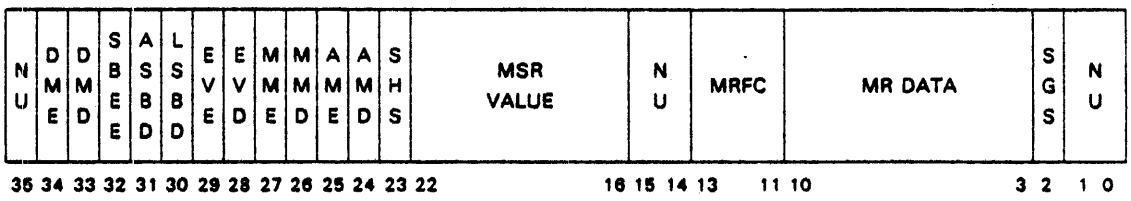
Bits 22 through 16 of the operand are transferred to the module select register (MSR) specified by bit 23. If bit 23=1, transfer is to the MSR in SIU upper half; if bit 23=0, transfer is to the MSR in SIU lower half.

The MSR is used as the base for all fixed address assignment references, and there is a separate MSR for the lower half of the addressing range (0-8M) and the upper half of the addressing range (8-16M); the load path selection of the system transition unit determines which is to be used for fixed references.

In addition to the normal function of modification of the MSR value and activation/deactivation of the MSR ACTIVE signal, the SIU recognizes the remaining bits in the SIL data word for various error reporting modes, error injection, and diagnostic functions.

For the basic function of modifying the interrupt location (changing the MSR value), it must be remembered that in the SIU each segment is completely independant and, therefore, has its own MSR register and MSR ACTIVE signal. This means that if two segments are operating together (interleaving) within a half, an SIL instruction must be directed to each segment in order to activate/deactivate the MSR in that segment. This is accomplished by using bits 23 and 2 of the SIL data word (see Figure 5-2) as the segment select bits (bit 23 = upper/lower half select and bit 2 = segment select within a half). These bits in the SIL data word must always reflect the segment desired for the SIL function. The following is a breakdown of all SIL functions:

- If SIL data bit 25 is set, and SIL data bit 24 is set, the addressed segment loads SIL data bits 22-16 as an MSR value and activates the MSR ACTIVE control signal for that segment.
- If SIL data bit 25 is set and SIL data bit 24 is clear, the MSR ACTIVE control signal for the addressed segment is deactivated.
- If SIL data bit 27 is set, and SIL data bit 26 is set, a SCISR interrupt is generated with maintenance read miss mode status bit set each time a read miss operation is executed.



35	NU	NOT USED
34	DME	DUMMY MSU ENABLE
33	DMD	DUMMY MSU DATA
32	SBEE	ALL SINGLE BIT ERROR DATA
31	ASBD	ALL SINGLE BIT ERROR DATA
30	LSBD	LIMITED SINGLE BIT ERROR DATA
29	EVE	ECC VERIFICATION ENABLE
28	EVD	ECC VERIFICATION DATA
27	MME	MAINTENANCE READ MISS ENABLE
26	MMD	MAINTENANCE READ MISS DATA
25	AME	ACTIVATE MSR ENABLE
24	AMD	ACTIVATE MSR DATA
23	SHS	SIU HALF SELECT
22-16		MSR VALUE
15-14	NU	NOT USED
13-11	MRFC	MAINTENANCE AND RELIABILITY FUNCTION CODE
10-03	MR DATA	MAINTENANCE AND RELIABILITY DATA
02	SGS	SIU SEGMENT SELECT
01-00	NU	NOT USED

Figure 5-2. Select Interrupt Locations

- If SIL data bit 27 is set, and SIL data bit 26 is clear, the maintenance read miss mode is deactivated in the addressed SIU segment.
- If SIL data bit 29 is set, and SIL data bit 28 is set, the ECC verification mode is activated in the addressed SIU segment. In error correction code (ECC) verification mode the main storage unit (MSU) on each write request disables the write data ECC check and stores the ECC code presented by the SIU.
- If SIL data bit 29 is set, and SIL data bit 28 is clear, the ECC verification mode is deactivated in the addressed SIU segment.
- If SIL data bit 32 is set, and SIL data bit 31 is set, the reporting of all single bit errors on MSU read operations via the SCISR interrupt is activated.
- If SIL data bit 32 is set, SIL data bit 31 is clear, and bit 30 is set, then the reporting of limited single bit errors on MSU read operations via SCISR interrupt is activated.
- If SIL data bit 34 is set, and SIL data bit 33 is clear, then dummy MSU mode is activated. Dummy MSU mode disables the SIU/MSU interface and provides verification of SIU functional operation without an MSU by turning the last four words of write data and the associated ECC codes around as read data and the associated ECC codes.

- If SIL data bit 34 is set, and SIL data bit 33 is clear, then dummy MSU mode is deactivated.
- If SIL data bits 13–11 equals 0_8, no maintainability and reliability is performed.
- If SIL data bits 13–11 equal 1_8, SIL data bits 09–03 are saved and any bits which are set will corrupt (invert) the corresponding ECC bit (06–00) that will be loaded into the main storage interface registers on the next write request. If the SIU is not in dummy MSU mode, the ECC syndrome will be, checked by the MSU during execution of the next write request. If the SIU is in dummy MSU mode, the ECC syndrome will be checked by the SIU during the next read miss request.
- If SIL data bits 13–11 equal 2_8, then SIL data bit 03 being set forces an address parity error on the invalidate interface on the next invalidate request.
- If SIL data bits 13–11 equal 2_8, then SIL data bit 04 being set forces the next tag address that is loaded into the tag buffer to be loaded with incorrect parity. This is done only when a read miss has been completed without encountering an error.
- If SIL data bits 13–11 equal 2_8, then SIL data bit 05 being set forces a tag control parity error the next time a read miss has been completed without encountering an error.
- If SIL data bits 13–11 equals 2_8, then SIL data bit 06 being set forces a write control parity error on the next main storage request.
- If SIL data bits 13–11 equal 2_8, then SIL data bit 07 being set forces an address parity error on the next main storage request.
- If SIL data bits 13–11 equals 2_8, then SIL data bits 09–08 force the four data words loaded into data buffer on the next read miss operation to be loaded with incorrect parity according to the following:

  1. SIL data bit 08 being set forces the lower half word of all four data words on a read miss operation to be loaded into the buffer with incorrect parity.
  2. SIL data bit 09 being set forces the upper half word of all four data words on the read miss operation to be loaded into the buffer with incorrect parity.

NOTE:

The following two functions require the CPU → SIU address buss to be preconditioned to the SET ADDRESS (10–03) desired for this operation. This is accomplished by storing and executing the SIL data word in main store rather than GRS. The set address affected will then correspond to the set address at which the SIL data word was stored.

- If SIL data bits 13–11 equal 4_8, SIL data bits 10–03 are loaded as the age bits on the next read or write hit or read miss, according to the following:

  1. SIL data bit 03 replaces the age bit for block A, bit 0.
  2. SIL data bit 04 replaces the age bit for block A, bit 1.
  3. SIL data bit 05 replaces the age bit for block B, bit 0.
  4. SIL data bit 06 replaces the age bit for block B, bit 1.

5. SIL data bit 07 replaces the age bit for block C, bit 0.

6. SIL data bit 08 replaces the age bit for block C, bit 1.

8. SIL data bit 10 replaces the age bit for block D, bit 1.

- If SIL data bits 13–11 equal 5_8, SIL data bits 10–03 control the upgrading and degrading of the SIU blocks according to the following:

1. If SIL data bit 03 is set, block A is degraded.

2. If SIL data bit 03 is clear, block A is upgraded (set to the invalid state).

3. If SIL data bit 04 is set, block B is degraded.

4. If SIL data bit 04 is clear, block B is upgraded (set to the invalid state).

5. If SIL data bit 05 is set, block C is degraded.

6. If SIL data bit 05 is clear, block C is upgraded (set to the invalid state).

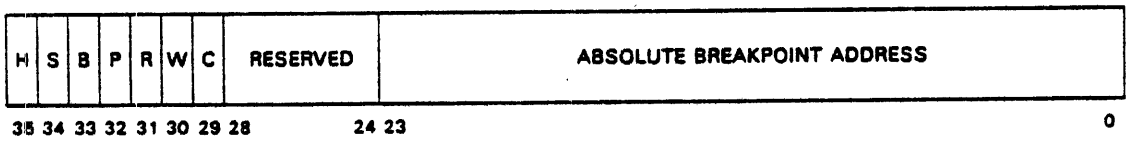
7. If SIL data bit 06 is set, block D is degraded.

8. If SIL data bit 06 is clear, block D is upgraded (set to the invalid state).

The set being degraded is selected by bits 10–03 of the data address presented with the MSR reference request.

5.15.6. Load Breakpoint Register - LBRX 73,15, 02

The operand specified by the operand address is transferred to the breakpoint register. This establishes the modes of operation for the breakpoint mechanism, and activates and establishes the modes of operation for the jump history stack.



Bit 35 The H–bit specifies that the CPU will stop on a breakpoint match condition; an interrupt request will not be generated. If H is zero, a Breakpoint interrupt will occur on a breakpoint match condition. The H–bit is ignored in real time mode and does not affect interrupts caused by the jump history stack.

Bit 34 The S–bit specifies that when the jump history stack is full (an entry is made in the last location), entry stacking is disabled and a Jump History Stack interrupt is generated. If the S–bit is zero, entry stacking wraps around from the last to first location in the stack without causing an interrupt. The S–bit should not be set unless the C–bit is also set.

Bit 33 The B–bit specifies that entry stacking is disabled when any interrupt occurs. If the B–bit is zero, entry stacking is not affected by interrupts, except as provided by the S–bit.

Bit 32 The P-bit allows a breakpoint match to occur on an instruction address produced from the program address register or a jump or EXECUTE operand instruction.

Bit 31 The R-bit allows a breakpoint match to occur on an operand address during a read operation.

Bit 30 The W-bit allows a breakpoint match to occur on an operand address during a write operation.

Bit 29 The C-bit specifies that the GRS locations 070-077 be cleared to zero and sequential jump history stacking begin at 070.

Bits 28-24 Reserved for future use.

Bits 23-0 The absolute breakpoint address is the value compared to the 24-bit absolute instruction or operand address.

5.15.7. Store Processor ID - SPID 73,15, 05

The binary serial number is stored in the first third of the operand, the 2-character Fieldata revision level is stored in the second third of the operand, the CPU features provided are stored in the fifth sixth of the operand (bit 9 for 4x4 STU, SIU, and MSU; bit 8 for byte-oriented instructions; bit 7 for floating-point instructions) and the binary CPU number is stored in the last sixth of the operand.

5.15.8. Load Quantum Timer - LQT 73,15, 03

The full-word operand specified by the operand address is placed in the quantum timer.

5.15.9. Load Base - LB 73,15,10

Bits 17 through 0 of the operand specified by the operand address are placed in the base-value field of the bank descriptor register specified by bits 34 and 33 of Xx. If the x-field of the instruction is zero, BDR0 is implicitly specified.

NOTE:

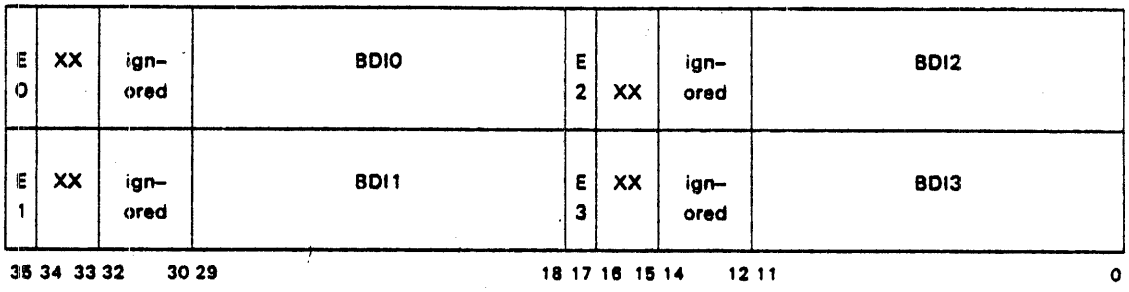
Execution of an LB instruction invalidates the Bank Descriptor specification in GRS 046 or 047 and the BDT pointer in GRS 040 or 045 for the specified base.

5.15.10. Load Limits - LL 73,15, 11

Bits 35 through 24 and 23 through 15 of the operand specified by the operand address are placed in the upper and lower limits fields, respectively, of the bank descriptor register specified by bits 34-33 of Xx. If the X-field of the instruction is zero, BDR0 is implicitly specified.

5.15.11. Load Addressing Environment - LAE 73,15, 12

The double-word operand specified by the operand address contains four bank descriptor specifications in the following format:



This operand is placed in GRS locations 046 and 047, and the limits and base values of the four bank descriptors specified by this operand are placed in the respective bank descriptor registers. The bank descriptor table length check is not performed on the bank descriptor index supplied by the instruction. Bank descriptor flags and use counts are neither interpreted nor altered by LAE.

5.15.12. Store Quantum Time - SQT 73,15, 13

The current value of the quantum timer is stored at the operand address, which may be in GRS or storage. Execution of this instruction has no effect on D29.

5.15.13. Load Designator Register - LD 73,15, 14

The full-word operand specified by the operand address is placed in the designator register. All designator register specifications are in effect at the completion of this instruction.

5.15.14. Store Designator Register - SD 73,15, 15

The contents of the designator register is stored at the location by the operand address.

5.15.15. User Return - UR 73,15, 16

This instruction provides an orderly mechanism for returning to a user program. The instruction effectively combines LD and jump, except that the component operations are performed with the correct repertoire, addressing, and register set.

The double-word operand specified by the operand address contains the relative program address and designator register value that establish the user operating state.

The second word of the operand is placed in the designator register, and all specifications are put in effect. The lower 24 bits of the first word of the operand then becomes the relative program address. If the relative program address is subsequently found to be out of limits, the interrupt will capture the new P-value.

Bit positions 23 through 18 of the relative address and the A-flag (bit position 35 of the same word) should be zero, unless base register suppression (D35 = 0, D7 = i = 1) is intended or was in effect when the address was stored as the result of an interrupt.

5.15.16. Reset Auto-Recovery Timer - RAT 73,15, 06

This instruction resets the timer in the auto-recovery section of the STU. This must be done within an interval specified by the auto-recovery timer in order to prevent an automatic initial load from being initiated, if auto recovery is enabled in the STU. The timer interval can be set from 1 to 15 seconds in increments of 1 second.

5.15.17. Toggle Auto-Recovery Path - TAP 73,15, 07

The system allows for two auto-recovery paths (CPU/IOU/SIU Half combinations). Each time an auto-recovery is attempted, the path selection is toggled. When a successful recovery does occur, this instruction allows the software to return the auto-recovery selection to the successful path.

5.15.18. Store System Status - SSS 73,15, 17

This instruction stores two words of system status at the location specified by the operand address. System status includes partitioning information relating to each CPU, IOU, SIU, and MSU. (See 2.6)

5.15.19. Initiate Interprocessor Interrupt - IIIX 73,15, 04

The operand address value specifies the number of the processor to be interrupted. Any processor in the application, including the one generating the interrupt, may be interrupted.

5.15.20. Diagnostics - 73,14, 14 - 17

The MDA (73,14,14) and MDB (73,14,15) instructions test a large portion of the arithmetic hardware and a smaller portion of the control section hardware. They generate specific operands, cause the arithmetic section to manipulate these operands, and store the arithmetic results into specific locations in GRS. Since the results should always be the same, these results can be compared against known good data. The operands generated and the results stored for the MDA instruction are as follows:

Operand A	= 070707070707	
Operand A+1	= 070707070707	
Operand U	= 070707070622	
Operand U+1	= 252525252525	
Result A	= 000372706711	Stored in GRS 62
Result A+1	= 256171354400	Stored in GRS 63

The operands generated and the results stored for the MDB instruction are as follows:

Operand A	= 070707070707	
Operand A+1	= 070707070701	
Operand U	= 070707070600	
Operand U+1	= 070707070707	
Result A	= 771177117711	Stored in GRS 62
Result A+1	= 777777776677	Stored in GRS 63

The result of each arithmetic iteration of the MDA instruction and final result of the MDB instruction are internally checked in the hardware of the processor against a known result. If a miscompare occurs, the processor will enable an internal flag indicating a MDA or a MDB failure, and an arithmetic parity PROM error will be generated. Otherwise, the result of the MDA and MDB instruction operations are stored in GRS locations 62 and 63, and are available for software compare.

If a software test of the results, which are stored in GRS 62 & 63, yields a miscompare, a faulty arithmetic section is indicated and further diagnostics should be performed.

Function codes 73,14,16, and 17 are undefined and will no-op.

5.15.21. Initiate Maintenance Interrupt - IMI 72,00

Send an attention interrupt to the maintenance processor if in maintenance mode, otherwise no-op.

5.15.22. Input/Output Instructions

The I/O instructions are described in detail in Section 6.

For each I/O instruction, the operand address specifies the IOU, channel, and device number, if applicable. For certain instructions, the index register specified by the a-field of the instruction (Xa) contains a parameter associated with the operation, generally an address. Each I/O instruction skips the next instruction if the operation was initiated properly, and a condition code of zero is stored in the upper sixth-word of register Xa.

If the next instruction is executed, the upper sixth-word of register Xa contains a code that describes one of three conditions: 020 = status is available, 040 = busy, and 060 = not operational; the remainder of Xa is not disturbed.

The Input/Output instructions are:

- Start I/O Fast Release (SIOF-75, 01): Xa contains the first channel command word (CCW) address.
- Test Subchannel (TSC-75, 03)
- Halt Device (HDV-75, 04)
- Halt Channel (HCH-75, 05)
- Load Channel Register (LCR-75, 10): Xa contains the value to be loaded.
- Load Table Control Words (LTCW-75, 11): Xa contains the first CCW address.

NOTE:

Because of the single interrupt address word (IAW) and channel status word (CSW) locations for a CPU, I/O instructions that may alter these locations must be executed with interrupts locked out.

5.16. Invalid Function Codes

A number of function codes are invalid and cause an Invalid Instruction Fault interrupt to address 221_8 to occur. These function codes are listed in Table 5-9.

Table 5-9. Invalid Function Codes

Function Code (Octal)		Result
f	j	
00	00–17	Invalid codes cause Invalid Instruction Fault interrupt to address MSR + 221_8.
07	00–11, 16	
37	10–17	
72	12, 14	
73	14 (a = 00–07)	
73	15 (a = 01)	
73	16	
73	17 (a = 03, 06–17)	
74	14 (a = 04–17)	
74	15 (a = 04–17)	
75	06, 07, 12–17	
77	00–17	

6. Input/Output

6.1. General

The input/output unit (IOU) provides a means of communications between the central processor unit (CPU) and its external media. Under CPU control, the IOU handles all transfer of data and status between peripherals and main storage. It minimizes CPU involvement in input/output operations, yet provides a flexible method of controlling and interrogating input/output activity. This section describes the IOU's system philosophy, functional characteristics, various hardware and software options, and overall operation.

6.2. Functional Characteristics

The IOU has five interfaces: a storage interface unit (SIU) interface, a system transition unit (STU) interface, a CPU interface, a system maintenance unit (SMU) interface, and a control unit or peripheral interface. Each IOU consists of one control module and from three to eight channel modules. (See Figure 6-1.) The control module handles all the interfacing with the storage unit STU control lines and either one or two CPUs. The control module to processor(s) interface initiates instructions and handles interrupts. The control module to storage interface transfers data, input/output control words, and status between the channel modules and the storage interface. The control module establishes a data handling and interrupt priority among the eight channel modules.

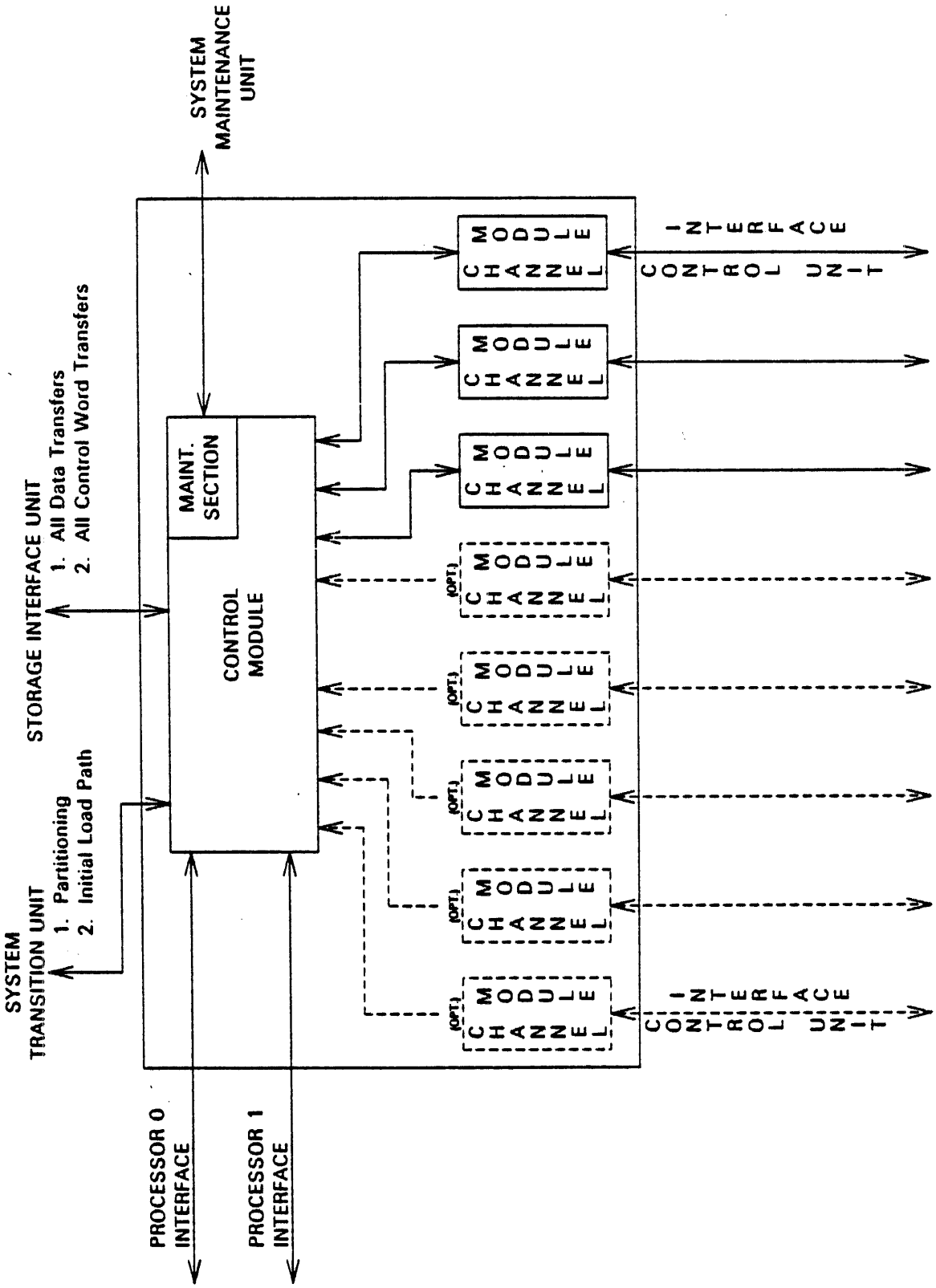


Figure 6-1. 1100/80 Input/Output Unit

6.2.1. Channels

The channel handles all interfacing with the control units. The channel executes input/output instructions, formats and transfers data, generates interrupts and status, and establishes priority among input/output instructions, data transfers, and interrupts. Each channel is designated by feature to be either a byte multiplexer channel, a block multiplexer channel, or a Series 1100 compatible word channel.

A channel provides a standard interface for communicating with control units. A control unit provides the logical capability necessary to adapt the standard form of control provided by the channel to the characteristics of an input/output device. A control unit may be housed separately and connected to one or many devices, or it may be physically and logically integrated with an input/output device. Input/output devices provide external storage and a means of communications for a processing system. Magnetic tape units, printers, storage devices such as disks and drums, consoles, and card readers are examples of input/output devices.

Priority among devices is established by the control units. Priority among control units is determined by the logical connection to the channel. Each word channel has a maximum of four parallel input/output interfaces, each of which is the equivalent of one Series 1100 word channel on another Series 1100 System, each with an assigned priority. (See Figure 6–2.)

Each word channel can connect to up to four control units, one to each of the four parallel interfaces. The interfaces may be either externally specified index (ESI) or internally specified index (ISI) in the following combinations:

- 0 ESI and 4 ISI
- 0 ESI and 3 ISI
- 0 ESI and 2 ISI
- 0 ESI and 1 ISI
- 1 ESI and 2 ISI
- 1 ESI and 1 ISI
- 1 ESI and 0 ISI
- 2 ESI and 2 ISI
- 2 ESI and 1 ISI
- 2 ESI and 0 ISI

Each byte or block multiplexer channel has one input/output interface, and up to eight control units can be connected to the interface in daisy chain fashion, but only one control unit at a time is logically connected to the channel. (See Figure 6–2.) The channel polls the control units serially, and the highest priority control unit requiring service logically connects to the channel. A byte multiplexer channel connects to a control unit for the length of time to transfer one byte of data. A block multiplexer channel connects to a control unit for the length of time to transfer a block of data. No other device can communicate over the interface during the time a block is being transferred.

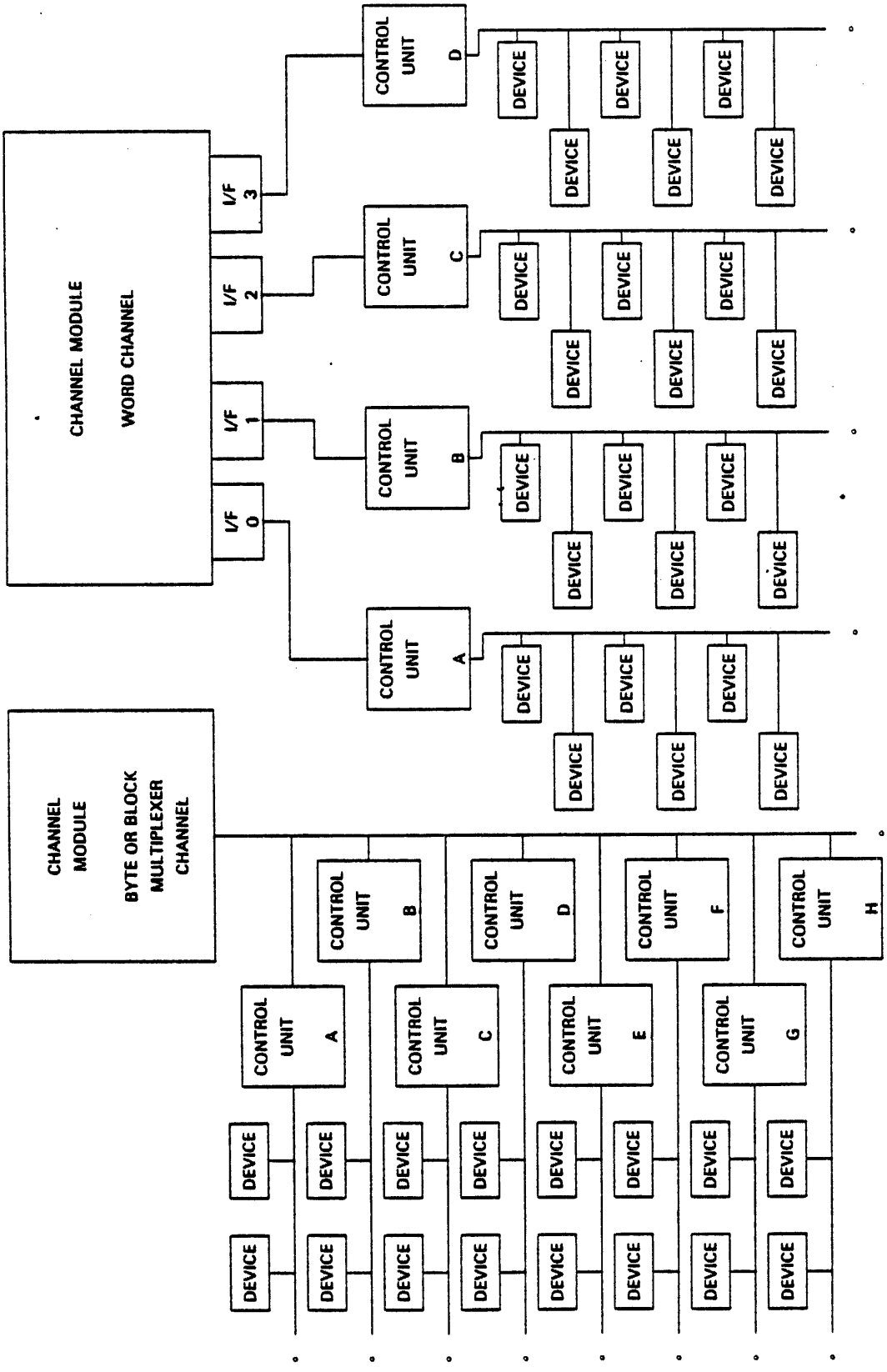


Figure 6-2. Byte or Block Multiplexer Channel and Word Channel Configuration

6.2.2. Subchannels

A subchannel is defined as a set of control words that manages input/output operations. Each set of control words contains a data address, a data count, the mode of the subchannel, the storage address of the next control word, and special flags. Subchannels may be either shared or nonshared. A subchannel is referred to as shared if two or more devices use the same subchannel for input/output operations. On a shared subchannel only one device at a time can transfer data. A subchannel is referred to as nonshared if it is associated with and can be used only with a single input/output device. On a word channel, ISI subchannels are shared and ESI subchannels are nonshared. On a byte multiplexer channel, all subchannels are shared; while on a block multiplexer channel, all subchannels are nonshared.

An IOU channel has the capability of maintaining eight resident subchannels. The basic IOU channel provides that all eight resident subchannels are shared. In word channel modules with the subchannel expansion feature (F1654–00) and option 0 (C1655–00), there are two resident ISI shared subchannels, four resident ESI nonshared subchannels, and 124 nonresident, nonshared subchannels. With the subchannel expansion feature and option 1 (C1655–01), there are eight resident ESI nonshared subchannels and 120 nonresident, nonshared subchannels. Nonresident, nonshared subchannels are kept in main storage. With the subchannel expansion feature and option 1, the eight most recently active nonshared subchannels are held in the channel. The remaining 120 subchannels are held in main storage. If the channel receives a request for a nonshared subchannel that is not resident in the channel, the least recently used resident nonshared subchannel is determined and then moved into main storage. The requested nonshared subchannel is then moved from main storage to the channel, and the request is handled. With the subchannel expansion feature and option 0, each channel has four shared subchannels and 128 nonshared subchannels. The four most recently active nonshared subchannels are kept resident in the channel, and the remaining 124 nonshared subchannels are held in main storage.

6.3. Control of Input/Output Devices

The CPU controls I/O operations by means of six I/O instructions: Start I/O Fast Release (SIOF), Test Subchannel (TSC), Halt Device (HDV), Halt Channel (HCH), Load Channel Register (LCR), and Load Table Control Words (LTCW). The LCR instruction addresses either the control module or a channel. The HCH and LTCW instructions address only a channel; they do not address an I/O device. All other instructions address a channel and subchannel.

6.3.1. Input/Output Device Addressing

An I/O device and its associated channel module and control module are designated by a 13–bit I/O address. The I/O address has an 8–bit device address in bits 00–07, a channel address in bits 08–11, and an IOU number in bit 12. Because the maximum configuration allows for only eight channel modules, bit 11 of the channel address is ignored and bits 08–10 are used to select a channel module. Of the 8–bit device address for a word channel, bit 07 specifies whether the selected subchannel is shared or nonshared. Device addresses with bit 7 equal to zero specify nonshared subchannels, and device addresses with bit 7 equal to one specify shared subchannels. Each nonshared subchannel is identified by a unique device address allowing a maximum of 128 nonshared subchannels per channel. On a word nonshared subchannel, bit 06 of the device address specifies the ESI interface. If bit 06 equals zero, ESI interface 0 is selected; and if bit 6 equals one, ESI interface 1 is selected. Byte multiplexer channels are limited to shared subchannel operation; therefore, bit 07 must be a one. All 256 subchannels on a block multiplexer channel are nonshared.

For shared subchannels on a byte or block multiplexer channel, bits 04–06 of the device address select one of eight shared subchannels and its associated control unit. Bits 00–03 select one of a maximum of 16 devices. This allows a maximum of eight shared subchannels and 128 devices per

byte or block multiplexer channel. There is a maximum of four shared subchannels and four associated ISI interfaces on a word channel. Bits 05–06 of the device address select the subchannel and the ISI interface. Bit 04 must be zero, and bits 00–03 are ignored. On word channels, the device address selects only a subchannel and an interface. The device is selected by an external function word.

Each channel can accommodate a different number of devices, depending upon the type of channel (byte or block multiplexer or word) and the option selected (all shared, subchannel expansion feature – option 0, or subchannel expansion feature – option 1) (See Table 6–1). Except for the rules described, the assignment of channel and device addresses is arbitrary.

6.3.2. States of the Input/Output System

The result of an I/O instruction is determined by the collective state of the channel, subchannel, and device selected by the I/O address. Depending on the type of channel and the I/O instruction being executed, different combinations of the channel, subchannel, and device will be interrogated to determine the response to an I/O instruction. When the response to an I/O instruction is determined by the state of the channel, the subchannel and device are not interrogated. If the response to an I/O instruction is determined by the state of the subchannel, the device is not interrogated. On a word channel the device is never interrogated to determine an I/O instruction response.

The channel, subchannel, and device can each be in one of four states. (See Table 6–2.) There are 13 composite states that cover all the conditions detected by an I/O instruction. In the following paragraphs each composite state is identified by three letters. The three letters indicate the state of the channel, subchannel, and device selected by the I/O address of the I/O instruction. There are two exceptions:

1. For the LTCW instruction, the second letter indicates the state of the status table subchannel.

2. For the LCR instruction, the second letter indicates whether the channel has the feature installed to handle the LCR instruction.

The symbol X in place of a letter indicates that the state of the corresponding component is not significant for the execution of an instruction. Unless specifically noted, the composite state applies to any type of channel. A description of each composite state follows:

- Channel Available (AXX): (Byte and block multiplexer channels only.) The channel is available. The states of the subchannel and device are not significant. This condition is detected only by a HCH instruction.

- Subchannel Available (AAX): The addressed channel and subchannel are operational, not busy executing a previous command, and not holding status. The state of the device is not significant. On a word channel a device is never interrogated to determine the response to an I/O instruction.

- Device Available (AAA): (Byte and block multiplexer channels only) The addressed channel, subchannel, and device are operational, not busy executing a previous command, and not holding status.

- Interrupt Pending in Device (AAI) or Device Working (AAW): (Byte and block multiplexer channels only) The addressed channel and subchannel are available. The addressed control unit or I/O device is executing a previously initiated operation or is holding status. The following situations are possible:

    1. The control unit is executing an operation on the addressed device or on another device associated with the same control unit.

2. The device or control unit is executing an operation on another channel or subchannel.

3. The device or control unit is holding status for the addressed device or another device associated with the same control unit.

- Device Not Operational (AAN): (Byte and block multiplexer channels only) The addressed channel and subchannel are available. The addressed I/O device is not operational. This occurs when the control unit for the addressed device is not installed or not online.

- Interrupt Pending in Subchannel (AIX): The addressed channel is available. The addressed subchannel is holding status from either a previously initiated operation or the present instruction attempting to be initiated. The subchannel is ready to store its status in a channel status word (CSW). The status can be for the addressed device or another device on the subchannel. The state of the addressed device is not significant.

- Subchannel Working (AWX): The addressed channel is available. The addressed subchannel is busy executing a previously executed operation. The state of the device is not significant.

- Subchannel Not Operational (ANX): The addressed channel is available. The addressed subchannel is not operational. This occurs when the channel is not equipped to handle that subchannel because of the particular type of channel and features selected.

- Interrupt Pending in Channel (IXX): This condition is never detected because channel status is reported by an independent interrupt mechanism. Channel status is not detectable or retrievable by way of I/O instruction. (See 7.4.1, Machine Check Interrupts.)

- Channel Working (WXX): (Block multiplexer channel only) The addressed channel is operating in burst mode (transferring a block of data). The states of the subchannel and device are not significant. The TSC and LCR instructions do not penetrate a channel in a working state and are not executed. A HDV instruction penetrates a working channel only if the channel is working with the addressed device. The HCH instruction always penetrates a working channel and halts the device that has control of the channel interface at the time that the HCH instruction is received. The SIOF instruction always penetrates a working channel. The response to the SIOF instruction is determined by the state of the subchannel.

- Channel Not Operational (NXX): An addressed channel is not operational when it is not installed in the system or is not online. The states of the addressed subchannel and device are not significant.

- Hardware Fault (XXX): If the IOU detects a hardware fault before the channel is selected, the instruction is terminated and a machine check interrupt is generated. The state of the I/O system is insignificant.

Table 6-1. Device Addressing

Device Addresses (Hexadecimal)	Byte Multiplexer Channel	Block Multiplexer Channel	Word Channel		
	Eight Shared Subchannels	256 Nonshared Resident Subchannels	Four Shared Subchannels	Subchannel Expansion Feature Option 0	Subchannel Expansion Feature Option 1
00–3F	Not used	Nonshared	Not Used	Nonshared ESI Interface A*	Nonshared ESI Interface A*
40–7F	Not Used	Nonshared	Not Used	Nonshared ESI Interface B*	Nonshared ESI Interface B*
80–8F	Shared 0**	Nonshared	Shared 0 ISI Interface A	Not Used	Not Used
90–9F	Shared 1	Nonshared	Not Used	Not Used	Not Used
A0–AF	Shared 2	Nonshared	Shared 2 ISI Interface B	Not Used	Not Used
B0–BF	Shared 3	Nonshared	Not Used	Not Used	Not Used
C0–CF	Shared 4	Nonshared	Shared 4 ISI Interface C	Shared 4 ISI Interface C	Not Used
D0–DF	Shared 5	Nonshared	Not Used	Not Used	Not Used
E0–EF	Shared 6	Nonshared	Shared 6 ISI Interface D	Shared 6 ISI Interface D	Not Used
F0–FF	Shared 7	Nonshared	Not Used	Not Used	Not Used
Number and type of Subchannels	8 Shared 0 Nonshared	0 Shared 256 Nonshared	4 Shared 0 Nonshared	2 Shared 128 Nonshared	0 Shared 128 Nonshared

* ESI Interface A has 64 device addresses 00–3F, and ESI interface B has 64 device addresses 40–7F.

** This number designates which of the eight channel hardware registers are associated with which device addresses. For nonshared subchannels, option 0 uses hardware registers 0, 1, 2, and 3; option 1 uses hardware registers 0, 1, 2, 3, 4, 5, 6, and 7.

Table 6-2. Channel, Subchannel, and Device States

State	Abbreviation	Definition
Channel		
Available	A	Ready to accept a nonpenetrating or penetrating instruction.
Interrupt Pending	I	Not defined.
Working	W	Operating in burst mode (block multiplexer channel only), and can accept and execute only penetrating instructions.
Not Operational	N	Not installed or offline.
Subchannel		
Available	A	Ready to accept new command.
Interrupt Pending	I	Holding status.
Working	W	Busy executing previous command.
Not Operational	N	Not installed.
Device		
Available	A	Ready to accept new command.
Interrupt Pending	I	Holding status.
Working	W	Busy executing previous command.
Not Operational	N	Not installed or not operational.

6.3.3. Condition Codes

The result of an I/O instruction is reported by a 2-bit condition code. The condition code is stored by the processor in bits 34–35 of the Xa register at the time the execution of the instruction is completed. The condition code is determined by the composite state of the I/O system selected by the I/O instruction. Tables 6–3, 6–4, and 6–5 show the relationship between the condition code for each instruction and the composite state of the I/O system. Special conditions affecting the condition code are also indicated.

Table 6-3. I/O System Composite State vs Condition Codes

Composite State	Condition Code	Byte Multiplexer Channel	Block Multiplexer Channel	Word Channel
AAA*	0, 1	HDV	HDV	(1)
AAI	1	HDV	HDV	(1)
AAW	1	HDV	HDV	(1)
AAN	3	HDV	HDV	(1)
AAX	0, 0, 1	LCR, LTCW, TSC	LCR, TSC, SIOF	HDV, LCR, LTCW, TSC, SIOF
AIX	1, 2 (a)	HDV, LTCW, SIOF, TSC	HDV, SIOF, TSC	HDV, LTCW, SIOF, TSC
AWX	0, 1, 2	HDV, LTCW, HDV, SIOF, TSC	HDV, HDV, SIOF, TSC	HDV, LTCW, SIOF, TSC
ANX	3	HDV, LCR, LTCW, SIOF, TSC	DHV, LCR, SIOF, TSC	HDV, LCR, LTCW, SIOF, TIO, TSC
AXX	0	HCH	HCH	(2)
IXX	-	(2)	(2)	(2)
WXX	0, 2	(3)	HCH, HCV, TSC, LCR	(4)
WAX	0	(3)	SIOF	(3)
WIX	1, 2	(3)	SIOF, SIOF, HEV, TSC	(3)
WWX	0, 2	(3)	HDV, HDV, SIOF, TSC	(3)
WNX	3	(3)	HDV, LCR, SIOF, TSC	(3)
NXX	3	HCH, HDV, LCR, LTCW, SIOF, TSC	HCH, HDV, LCR, LTCW, SIOF, TSC	HCH, HDV, LCR, LTCW, SIOF, TSC
XXX	2, 3	HCH, HDV, LCR, LTCW, SIOF, TSC,	HCH, HDV, LCR, LTCW, SIOF, TSC,	HCH, HDV, LCR, LTCW, SIOF, TSC

A = Available I = Interrupt Pending W = Working N = Not Available

X = Any of the above states

*The three letters, from left to right, indicate the state of the channel, subchannel, and device selected by the I/O address with two exceptions: a) For the LCR instruction, the second letter indicates whether the channel has the feature installed to handle the LCR instruction, b) For the LTCW instruction, the second letter indicates the state of the status table subchannel.

(a) Special conditions in the channel determine whether a condition code of 1 or 2 will be presented. These special conditions are covered in Table 6-4.

(1) A word channel never interrogates a device to determine a response to an I/O instruction.

(2) This condition is not detectable by any I/O instruction.

(3) Byte channels and word channels do not transfer blocks of data, only bytes or words, and thus can never be in the working state.

(4) The word channel is never in the working state.

Table 6-4. I/O Instruction Condition Codes for Byte or Block Multiplexer Channels

State*	Channel	Conditions	SIOF (a)	TSC	HDV	HCH	LCR	LTCW
AAA	Block	–	–	–	0	–	–	–
AAI	Byte, Block	–	–	–	1	–	–	–
AAW	Byte, Block	–	–	–	1	–	–	–
AAN	Byte, Block	–	–	–	3	–	–	–
AAX	Byte, Block	–	0/1(c)	0	0	–	0	0
AIX	Byte, Block	(1) and (3)	1	1	1	–	–	1
AIX	Byte, Block	(1) and (4)	2	2	2	–	0	2
AIX	Byte, Block	(2)	2	1	2	–	–	–
AWX	Byte, Block	–	2	2	1	–	–	0
ANX	Byte, Block	–	3	3	3	–	3	3
AXX	Byte, Block	–	–	–	–	0	–	–
IXX	Byte, Block	–	–	–	–	–	–	–
WXX	Block	–	–	2	–	0	–	–
WAX	Block	–	0/1 (c)	–	2	–	2	–
WIX	Block	(1) and (3)	1	–	2	–	–	–
WIX	Block	(2) or (4)	2	–	2	–	–	–
WWX	Block	–	2	–	1/2 (b)	–	–	–
WNX	Block	–	3	3	3	–	3	–
NXX	Byte, Block	–	3	3	3	3	3	3
XXX	Byte, Block	(5)	2	2	2	2	2	2
XXX	Block	(6)	–	–	–	–	–	–
	Byte, Block	(7)	2	–	–	–	–	–

States: A = Available I = Interrupt Pending W = Working N = Not Available X = Any of the states

* The three letters, from left to right, indicate the state of the channel, subchannel, and device selected by the I/O address with two exceptions: a) for the LCR instruction, the second letter indicates whether the channel has the feature installed to handle the LCR instruction, b) for the LTCW instruction, the second letter indicates the state of the status table subchannel.

Table 6–4. I/O Instruction Condition Codes for Byte or Block Multiplexer Channels (continued)

Footnotes for Table 6–4 are as follows:

(a) If the SIOF queue is full, or the channel is in working state, the SIOF instruction will unconditionally receive a condition code of 2.

(b) If the channel is working (operating in burst mode) with the addressed device, the operation is terminated and the condition code equals 1. If the channel is working, but not with the addressed device, the condition code equals 2.

(c) If a hardware or software error is detected when retrieving the second word of the channel address word (CAW), a channel status word (CSW) is stored and the instruction receives a condition code of 1. If no hardware or software error is detected, the instruction receives a condition code of 0.

(1) Subchannel is holding status for addressed device.

(2) Subchannel is holding status for a device other than the addressed device.

(3) Interrupt cancellation was not attempted or was attempted and completed successfully.

(4) Interrupt cancellation was attempted and was unsuccessful.

(5) Hardware fault was detected when reading first word of the CAW and Machine Check interrupt was generated.

(6) Unit Check status had not been presented via interrupt or instruction.

(7) The channel is in working state.

Table 6-5. I/O Instruction Condition Codes for Word Channels

State*	Channel	Conditions	SIOF	TSC	HDV	HCH	LCR	LTCW
AAX	Word	–	0	0	0	–	0	0
AIX	Word	(1)	1	1	1	–	–	1
AIX	Word	(2)	2	2	2	–	–	2
AWX	Word	–	2	2	0	–	0	0
ANX	Word	–	3	3	3	–	3	3
AXX	Word	–	–	–	–	–	–	–
IXX	Word	–	–	–	–	–	–	–
WXX	Word	–	–	–	–	–	–	–
NXX	Word	–	3	3	3	–	3	3
XXX	Word	(3)	2	2	2	2	2	2
XXX	Word	(4)	–	–	–	3	–	–

States: A = Available I = Interrupt Pending W = Working N = Not Available X = Any of the states

* The three letters, from left to right, indicate the state of the channel, subchannel, and device, respectively.

(1) Interrupt cancellation was not attempted or was attempted and completed successfully.

(2) Interrupt cancellation was attempted unsuccessfully.

(3) Hardware fault was detected when reading first word of the CAW and Machine Check interrupt was generated.

(4) The Halt Channel instruction is not accepted on a word channel.

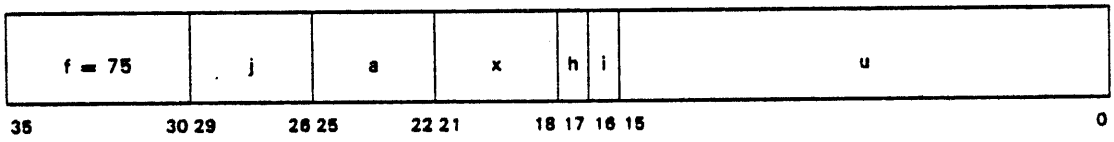
6.3.4. Instruction Format and Channel Address Word

All I/O instructions have an f = 75_8. The j value specifies the particular I/O instruction to be initiated:

j = 01_8 = Start I/O Fast Release	(SIOF)
j = 03_8 = Test Subchannel	(TSC)
j = 04_8 = Halt Device	(HDV)
j = 05_8 = Halt Channel	(HCH)
j = 10_8 = Load Channel Register	(LCR)
j = 11_8 = Load Table Control Words	(LTCW)

Bits 00–12 of u + Xm specify the I/O address (the IOU, channel, and device numbers). Bits 00–23 of Xa consist of either the starting address of the channel command word (CCW) or status table control word (STCW) list or the register input data for the LCR instruction. Upon detection of an I/O instruction, the CPU builds a channel address word (CAW) in a fixed location of low-order storage. The CAW is for hardware use only and consists of two 36-bit words, CAW 0 and CAW 1. The j-value is stored in CAW 0, bits 26–29. Bits 00–12 of u + Xm (the I/O address) are stored in CAW 0, bits 00–12. Bits 00–23 of Xa (the first CCW or STCW address or register input data) are stored in CAW 1, bits 00–23. The IOU refers to the CAW only during the execution of an I/O instruction. The pertinent information thereafter is stored in the channel. Instruction word and CAW format is as follows:

I/O Instruction Format

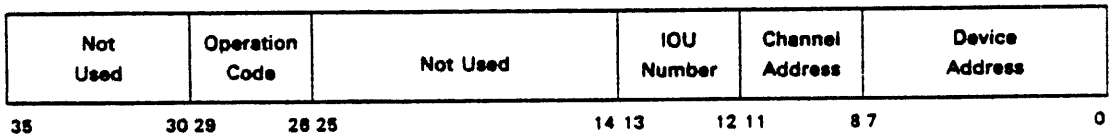


j Specifies I/O instruction

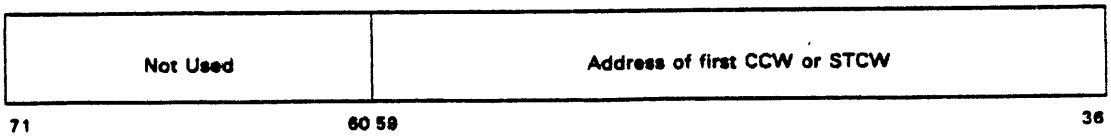
a Address of register holding first CCW address

x u + Xm bits 0–12 equal I/O address

CAW 0



CAW 1



where bits:

71-60 Must be zeros except when operation code specifies LCR; then it may contain data to be transferred to the register.

59-36 Specifies the double word address storage location which contains the first CCW in the channel program if the operation code specifies SIOF (operation code 01), or the double word storage location of the first STCW in the Status Table channel program if the operation code specifies LTCW (operation code 11), or the data to be transferred to the channel if the operation code specifies LCR (operation code 10).

35-30 Must be zeros

29-26 Operation code (octal); specifies the I/O operation to be performed:

- 01 Start I/O Fast Release
- 03 Test Subchannel
- 04 Halt Device
- 05 Halt Channel
- 10 Load Channel Register
- 11 Load Table Control Words

25-14 Must be zeros

13-12 IOU Number - bits 13 and 12 select the IOU:

Bit 13	Bit 12	
0	0	= IOU 0
0	1	= IOU 1
1	0	= IOU 2
1	1	= IOU 3

11-8 Channel Address - since eight channel modules is the maximum allowed, bit 11 is ignored and bits 10 through 8 select the channel module.

7-0 Device Address - specifies address of the device, if any, on which the operation is to be performed.

6.3.5. Instruction Operation

Upon detection of an I/O instruction, the CPU builds the CAW and then initiates the IOU via the CPU/IOU interface. The IOU establishes instruction priority between the two CPUs (CPU 0 has priority over CPU 1) and reads CAW 0. If the IOU detects a hardware fault in reading CAW 0, the instruction is immediately terminated, a Machine Check interrupt is generated, and a condition code of 2 is presented to the CPU

If no hardware fault is detected, the IOU then executes the instruction. Depending on the I/O instruction, the IOU uses the state of the channel, the states of the channel and subchannel, or the states of the channel, subchannel, and device to determine the condition code. When the IOU has

completed the I/O instruction, the CPU clears bits 30–33 of Xa and stores the condition code in bits 34–35 of Xa. Bits 00–29 of Xa are left unchanged. If the condition code equals 0, the CPU skips the next instruction in the program. If the condition code equals 1, 2, or 3, the CPU executes the next instruction.

NOTE:

An I/O instruction may cause a channel status word (CSW) to be stored. To prevent the contents of the CSW stored by the instruction from being destroyed by an immediately following I/O interrupt, interrupts must be locked out before issuing the I/O instruction and must remain locked out until the information in the CSW provided by the instruction has been acted upon or stored elsewhere for later use.

6.4. I/O Instructions

The I/O instructions can be classified as penetrating or nonpenetrating. A penetrating I/O instruction is always executed even if the channel is in a working state. (Note that only a block multiplexer channel can be in a working state.) A nonpenetrating I/O instruction always receives a busy response (condition code = 2) when attempted on a channel in a working state. The Start I/O Fast Release and Halt Channel instructions are penetrating instructions and are always executed even on working channels. A Halt Device instruction is executed only if: a) the channel is working with the addressed device, or b) the channel is in the available state.

For any I/O instruction a condition code of 0 indicates that the instruction was completed successfully. A condition code of 1 always indicates that a valid CSW has been written into low-order storage and the instruction was not executed. A subchannel is always returned to the available state after being relieved of status by an instruction or an interrupt. A condition code of 2 indicates that the I/O instruction was not executed. A condition code of 3 indicates that the instruction was not executed because either the channel, subchannel, or device was not operational. Use Tables 6–4 and 6–5 to determine what the condition code response to an I/O instruction means.

After each instruction, the condition codes and the conditions causing each condition code are listed.

6.4.1. Operation Code – 75,00

An I/O instruction with an operation code of 00 is not available on the byte multiplexer channel, the block multiplexer channel, or the word channel.

- Condition Code = 0

  Not Used

- Condition Code = 1

  Not Used

- Condition Code = 2

  Operation terminated due to machine check

- Condition Code = 3

  Instruction not available

6.4.2. Start I/O Fast Release – SIOF 75,01

If the addressed subchannel of an SIOF instruction is available, the second word of the CAW that contains the address of the first CCW is retrieved and stored in the subchannel. If the channel is not in the working state and no hardware or software errors are detected, the device address is placed in an SIOF queue, and a condition code of 0 is presented to the CPU. If a software or hardware error is detected during retrieval of the address of the first CCW, subchannel status is generated, a CSW is stored, and a condition code of 1 is presented to the CPU.

Whenever the channel becomes available, the device addresses of successfully executed SIOFs (condition code of 0) are fetched from the queue on a first in/first out basis. When a device address is fetched from the queue, the first CCW specified for that device address is retrieved from storage and the operation specified by that CCW is initiated at the device if the device is available. If the device is not available, the operation is not initiated, and the software is notified via interrupt.

The subchannel is set to an active state at the time that the device address associated with that subchannel is placed in the SIOF queue. The subchannel remains in an active state until either termination status is detected or the operation is terminated by an I/O instruction. Only one SIOF instruction per subchannel and 64 SIOF instructions per channel can be held in the queue at one time.

6.4.2.1. Byte or Block Multiplexer Channel Operation

- Condition Code = 0

  1. The channel was operational, and the subchannel was available. The first CCW address was read from the CAW and stored in the subchannel successfully, and the device address was loaded in the SIOF queue.

- Condition Code = 1

  1. The subchannel was holding status from a previous operation on the addressed device (check the device and subchannel status fields of the CSW).

  2. A software or hardware error was detected when attempting to fetch the address of the first CCW from the CAW (check the subchannel status field of the CSW).

- Condition Code = 2

  1. The subchannel was holding status for the addressed device, but was busy presenting the status by way of interrupt.

  2. The subchannel was busy holding status for or executing a previously initiated operation on a device other than the addressed device.

  3. The subchannel was busy executing a previously initiated operation with the addressed device.

  4. The SIOF queue was full. The channel had 64 pending SIOFs to be initiated at the device level.

  5. A hardware fault was detected when fetching the first word of the CAW.

- Condition Code = 3

  1. The addressed channel or subchannel was not operational.

6.4.2.2. Word Channel Operation

- Condition Code = 0

  1. The channel was operational, and the subchannel was available. The first CCW address was read from the CAW and stored in the subchannel successfully, and the subchannel address was loaded in the SIOF queue.

- Condition Code = 1

  1. A software or hardware error was detected when attempting to fetch address of the first CCW from the CAW (check the subchannel status field of the CSW).

- Condition Code = 2

  1. The subchannel was holding status, but was busy presenting the status by way of interrupt.
  2. The subchannel was busy executing a previously initiated operation.
  3. The SIOF queue was full. The channel had 64 pending SIOFs to be initiated at the device level.
  4. A hardware fault was detected when fetching the first word of the CAW.
  5. The subchannel was holding status from a previous operation (check the device and subchannel status fields of the CSW).

- Condition Code = 3

  1. The channel or subchannel was not operational.

6.4.3. Operation Code - 75,02

An I/O instruction with an operation code of 02 is not available on the byte multiplexer channel, the block multiplexer channel, or the word channel.

- Condition Code = 0

  Not Used

- Condition Code = 1

  Not Used

- Condition Code = 2

  Operation terminated due to machine check

- Condition Code = 3

  Instruction not available

6.4.4. Test Subchannel - TSC 75,03

The Test Subchannel instruction interrogates the channel and subchannel specified by the I/O address. The addressed device is not affected. The subchannel is not interrogated if the channel is in a working state.

6.4.4.1. Byte or Block Multiplexer Channel

- Condition Code = 0

  1. The channel and subchannel are available.

- Condition Code = 1

  1. The subchannel was holding status from a previously initiated operation (check the device and subchannel status fields of the CSW).

- Condition Code = 2

  1. The subchannel was holding status, but was busy presenting the status by way of interrupt.
  2. The subchannel was busy executing a previously initiated operation.
  3. On a block multiplexer channel only, the channel was busy operating in burst mode.
  4. A hardware fault was detected in fetching the first word of the CAW.

- Condition Code = 3

  1. The channel or subchannel was not operational.

6.4.4.2. Word Channel Operation

- Condition Code = 0

  1. The channel and subchannel are available.

- Condition Code = 1

  1. The subchannel was holding status from a previously initiated operation (check the device and subchannel status fields of the CSW).

- Condition Code = 2

  1. The subchannel was holding status, but was busy presenting the status by way of interrupt.
  2. The subchannel was busy executing a previously initiated operation.
  3. On a block multiplexer channel only, the channel was busy operating in burst mode.
  4. A hardware fault was detected in fetching the first word of the CAW.

- Condition Code = 3

  1. The channel or subchannel was not operational.

6.4.5. Halt Device – HDV 75,04

The Halt Device instruction terminates the current operation on the channel and subchannel specified by the I/O address. The operation on the specified subchannel is terminated immediately, and the device is notified of the termination when the device references the channel.

NOTE:

The Halt Device instruction is intended for use only as a recovery mechanism for software or hardware faults because the capability of an HDV instruction is limited for two reasons:

1. *A Halt Device instruction does not penetrate a working block multiplexer channel and is rejected unless the channel is working with the addressed device.*

2. *The resultant state of the device on a byte or block multiplexer channel after receiving a Halt Device instruction is unpredictable. The device may or may not still be active and may or may not eventually present status via interrupt or instruction.*

6.4.5.1. Byte or Block Multiplexer Channel Operation

- Condition Code = 0

  1. The channel, subchannel, and device were in the available state.

  2. The channel was available. The operation that was being executed by the subchannel with either the addressed device or another device on that subchannel has been terminated. The device has been signaled to terminate the operation. At a later time the device, after having completed termination of the operation, may present status by way of interrupt or instruction. The subchannel has been returned to the available state.

- Condition Code = 1

  1. The channel and subchannel were available. The control unit was busy executing a previously initiated operation on either the addressed device or another device (check the device status field of the CSW).

  2. The subchannel was holding status from a previous operation on the addressed device (check the device and subchannel status fields of the CSW).

  3. The channel and subchannel were busy operating in burst mode with the addressed device. The operation has been terminated and the device has been signaled to terminate the operation. At a later time the device, after having completed termination of the operation, may present status by way of interrupt or instruction. The subchannel has been returned to the available state.

- Condition Code = 2

  1. The subchannel was holding status for the addressed device, but was busy presenting the status by way of interrupt.

2. The subchannel was holding status for a device other than the addressed device.

3. On a block multiplexer channel, the channel was operating in burst mode with a device other than the addressed device.

4. A hardware fault was detected when fetching the first word of the CAW.

- Condition Code = 3

1. The channel, subchannel, or device was not operational.

6.4.5.2. Word Channel Operation

- Condition Code = 0

1. The addressed channel was operational, and the addressed subchannel was in the available state.

2. The addressed channel was operational. The operation that was being executed by the addressed subchannel has been terminated. The subchannel has been returned to the available state.

- Condition Code = 1

1. The subchannel was holding status from a previously initiated operation (check the device and subchannel status fields of the CSW).

- Condition Code = 2

1. The subchannel was holding status, but was busy presenting the status by way of interrupt.

2. A hardware fault was detected when fetching the first word of the CAW.

- Condition Code = 3

1. The channel or subchannel was not operational.

6.4.6. Halt Channel - HCH 75,05

The Halt Channel instruction terminates the current operation on the channel specified by the I/O address. The Halt Channel instruction is intended to be used to recover a byte or block multiplexer channel that has become inoperative during peripheral interface sequencing because of a control unit interface error. The Halt Channel instruction is not accepted on a word channel.

6.4.6.1. Byte or Block Multiplexer Channel Operation

- Condition Code = 0

1. The channel was available.

2. The channel was busy operating in burst mode or was in a hung condition because an interface error has resulted in the suspension of normal sequencing. The operation on the subchannel that had control of the channel interface has been terminated, and the device

has been signaled to terminate the operation. At a later time, when the device has completed terminating the operation, it may present status by way of interrupt. The subchannel has been returned to the available state.

- Condition Code = 1

  Impossible

- Condition Code = 2

  1. A hardware fault was detected when fetching the first word of the CAW.

- Condition Code = 3

  1. The channel was not operational.

6.4.6.2. Word Channel Operation

- Condition Code = 0

  Impossible

- Condition Code = 1

  Impossible

- Condition Code = 2

  1. A hardware fault was detected when fetching the first word of the CAW.

- Condition Code = 3

  1. The channel was a word channel.

6.4.7. Load Channel Register - LCR 75,10

The Load Channel Register instruction is used to load the interrupt mask register in the IOU control module or the channel base register in the channel specified by the I/O address. The operation to be performed is specified by bit 00 of the I/O address field. If bit 00 of the CAW is a zero, the channel base register of the addressed channel is loaded with the contents of bits 36 through 50 of the CAW. If bit 00 of the CAW is a one, the interrupt mask register is loaded with the contents of bits 36–71 of the CAW. The use of the channel base register is described in 6.17, and the use of the interrupt mask register is described in 6.18.

6.4.7.1. Byte and Block Multiplexer Channel

- Condition Code = 0

  1. The interrupt mask register or channel base register was loaded successfully.

- Condition Code = 1

Impossible

- Condition Code = 2

  1. A hardware fault was detected when fetching the first word of the CAW.

  2. The channel was operating in burst mode (loading the channel base register on a block multiplexer channel only).

- Condition Code = 3

  1. The channel was not operational or did not have the subchannel expansion feature installed (loading the channel base register only).

6.4.7.2. Word Channel Operation

- Condition Code = 0

  1. The interrupt mask register or channel base register was loaded successfully.

- Condition Code = 1

  Impossible

- Condition Code = 2

  1. A hardware fault was detected when fetching the first word of the CAW.

- Condition Code = 3

  1. The channel was not operational or did not have the subchannel expansion feature installed (loading the channel base register only).

6.4.8. Load Table Control Words - LTCW 75,11

The Load Table Control Words instruction loads the status table subchannel in the IOU and channel specified by the I/O address. The status table subchannel controls the tabling of communication interrupt status as described in 6.12. The LTCW instruction initiates the execution of a CCW list by the status table subchannel on the IOU and channel selected by the I/O address. Bits 36-59 of the CAW contain the address of the first status table control word (STCW) of the STCW list. A STCW contains the data count and the starting address for the status table. Even if the status table subchannel is active (has a valid data count), a new LTCW instruction is accepted, a new STCW list is initiated, and the status table subchannel is loaded with the first STCW of the new list.

6.4.8.1. Byte and Block Multiplexer Channel

- Condition Code = 0

  1. The addressed channel was operational, and the status table subchannel was available or active. The first STCW has been fetched, and the status table subchannel has been initiated successfully.

- Condition Code = 1

  1. The status table subchannel was holding status because of a software or hardware error on a previous operation (check the subchannel status field of the CSW).

  2. A software or hardware error was detected when attempting to fetch either the address of the first STCW from the CAW or the first STCW from storage (check the subchannel status field of the CSW).

- Condition Code = 2

  1. The status table subchannel was holding status from a previous operation, but was busy presenting the status by way of interrupt.

  2. A hardware fault was detected when fetching the first word of the CAW.

- Condition Code = 3

  1. The channel was not operational or did not have the feature installed allowing it to handle communication interrupt status or it was a block multiplexer channel.

6.4.8.2. Word Channel Operation

- Condition Code = 0

  1. The addressed channel was operational and the status table subchannel was available or active. The first STCW has been fetched and the status table subchannel has been initiated successfully.

- Condition Code = 1

  1. The status table subchannel was holding status because of a software or hardware error on a previous operation (check the subchannel status field of the CSW).

  2. A software or hardware error was detected when attempting to fetch either the address of the first STCW from the CAW or the first STCW from storage (check the subchannel status field of the CSW).

- Condition Code = 2

  1. The status table subchannel was holding status from a previous operation, but was busy presenting the status by way of interrupt.

  2. A hardware fault was detected when fetching the first word of the CAW.

- Condition Code = 3

  1. The channel was not operational or did not have the feature installed allowing it to handle communication interrupt status or it was a block multiplexer channel.

6.5. Execution of I/O Operations

A channel can execute seven commands: Write, Read, Read Backward (only on a byte or block multiplexer channel), Sense (only on a byte or block multiplexer channel), Control, Transfer in Channel (TIC), and Store Subchannel Status. Each command except Transfer in Channel and Store Subchannel Status initiates a corresponding I/O operation. The initiation and execution of a command issued to a subchannel and device is termed an "I/O operation."

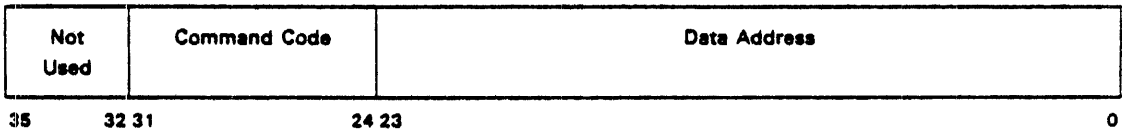
A series of I/O operations on the same device (byte or block multiplexer channels) or on the same subchannel (word channel) is executed under control of a set of channel command words (CCWs). The execution of a set of channel command words (CCW list) is initiated by a SIOF instruction. For an SIOF instruction, the address of the first CCW is stored in the channel, and a condition code is presented to the CPU. At an idle time in the channel sequencing, the first CCW is fetched and the specified I/O operation is initiated. The CCWs can be located anywhere in main storage. Fetching of a CCW by the channel does not affect the contents of the location in main storage. Each additional CCW in the CCW list is obtained when the operation has progressed to the point where the additional CCW is needed.

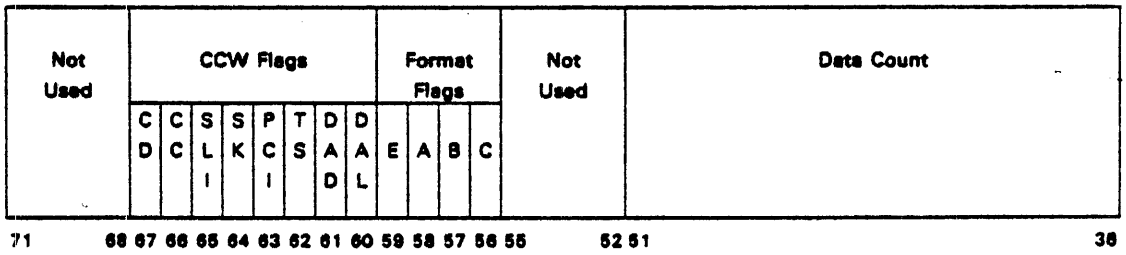
6.5.1. Channel Command Word

A channel command word specifies the command to be executed, and for commands initiating I/O operations it designates the storage area associated with the operation, the amount of data to be transferred, the formatting of data that is to be done, and the action to be taken when the operation is completed.

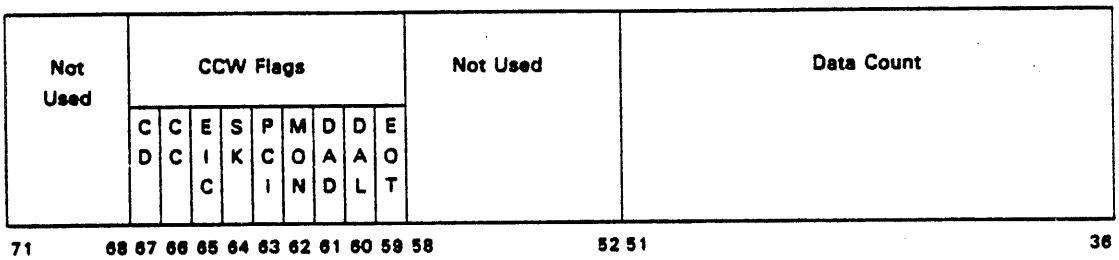
The CCW has the following format:

Byte or Block Multiplexer Channel CCW





Word Channel CCW



where bits:

0–23 Data address bits contain the storage address of the first data word to be transferred unless the command code is Transfer in Channel or Store Subchannel Status. For the Transfer in Channel command, the field contains the new CCW address; and for the Store Subchannel Status command, the field contains the address where status is to be stored.

24–31 Command Code bits specity the operation to be performed by the device or channel.

28–29 Format Control bits are interpreted only on ESI word subchannels. In quarter-word mode format control bits 29 and 28 specify the location of the first quarter word within the first word of data for each CCW.

Bits 29 and 28	Description
00	Specifies that the first quarter word be selected from bits 27 through 35 of the first data word.
01	Specifies that the first quarter word be selected from bits 18 through 26 of the first data word.
10	Specifies that the first quarter word be selected from bits 09 through 17 of the first data word.
11	Specifies that the first quarter word be selected from bit 00 through 08 of the first data word.

In half-word mode format control bit 28 specifies the location of the first half word within the first word of data for each CCW.

Bit 28	Description
0	Specifies that the first half word be selected from bits 00 through 17 of the first data word.
1	Specifies that the first half word be selected from bits 18 through 35 of the first data word.

32–35 Not used.

36–51 Data Count bits specify the number of bytes, words, half words, or quarter words transferred to or from storage.

52-55 Not used.

56-67 Bits 56-67 contain flags that specify data formats, special handling of an operation by the channel, and the action to be taken once the present operation is completed.

67 Chain Data (CD) specifies that upon completion of the portion of a data transfer operation being controlled by the current CCW, a new CCW is to be read from storage and the operation is to be continued under control of the new CCW.

66 Chain Command (CC) specifies that upon completion of the operation, a new CCW is to be read from storage and the operation specified by the new command code is to be initiated. If the chain data flag is set, the chain command flag is ignored.

65 Suppress Length Indication (SLI), for byte and block multiplexer channels only, specifies that if command chaining conditions are present, a command chain operation be initiated regardless of the residual byte count. The absence of the suppress length indication bit specifies that if command chaining conditions are present, a command chain operation be initiated only if the residual byte count equals zero, or the previous command was an immediate command. If these conditions are not met, the execution of the CCW list is terminated and an interrupt is generated. The suppress length indication flag is ignored if the Truncated Search flag is set in the same CCW.

NOTE:

Command chaining conditions are defined as Channel End and Device End status, the command chain flag set, and the data chain flag clear in the active CCW.

External Interrupt Chain (EIC), for word channel and ESI subchannels only, specifies that upon completion of the operation at the ESI word channel device, the operation specified by the new command is to be initiated. The external interrupt presented by the device is stored in the status table prior to chaining, and a tabled interrupt request is presented to the CPU.

The external interrupt chain is not executed if:

1. The status table subchannel is not active,
2. A hardware error is detected when entering the external interrupt in the status table,
3. A hardware or software error is detected during the retrieval of the new CCW from storage, or
4. The subchannel was not in an active state before receiving the external interrupt.

The EI chain flag is not interpreted on ISI word channels.

64 Skip Data (SK) specifies that data will not be written in storage for input operations. The device and subchannel, however, are handled in the same manner as during conventional input operation. For all other operations, the skip data flag is ignored.

63 Program Controlled Interrupt (PCI) specifies that the channel shall store a PCI subchannel status indication and generate an interrupt as soon as possible after a CCW containing this flag is obtained.

62 Truncated Search (TS), for block multiplexer channels only, specifies that a special chaining operation is to be executed. The channel saves the command in the CCW with the truncated search flag and also saves the command from the preceding CCW. These two commands are then reissued to the control unit during a truncated search operation. See 6.8.4. for a detailed description of truncated search operations.

Monitor (MON), for word channels only, specifies that the channel shall store subchannel status and generate an interrupt when the data count in the final CCW has been exhausted.

61 Data Address Decrement (DAD) specifies that the data address be decreased by one for each full data word transferred under control of the current CCW. This flag is ignored if the data address lock (DAL) flag is set. If neither the DAD or DAL flags are set, the data address will be incremented by one for each full data word transferred.

60 Data Address Lock (DAL) specifies that the contents of the data address field remain unchanged for each word transferred under control of the current CCW.

59 End of Transmission (EOT), for word channels only, specifies that an EOT bit (a one-bit in bit position 9 of the output data word) be transmitted with the last byte that is transferred under control of that CCW. The EOT bit is not set and not transmitted to the peripheral if the data chain flag in the CCW is set. The EOT flag is used only on ESI quarter word channel interfaces and is ignored on all other channels including ISI word channel interfaces.

Emulation Mode (E), for byte and block multiplexer channels only, when equal to zero, specifies Series 1100 mode for data transfer operations and the 36-bit data packing format is selected. E equals one if invalid.

On word channels the E-bit is ignored and the mode of operation is specified by patch wire. Each word channel Series 1100 ESI interface operates via patch wire in either quarter-word mode or half-word mode.

58 Format Flags A, B, and C (byte and block multiplexer channels only) specify the
57 packing format of data bytes. (See Tables 6-6, 6-7, and 6-8.) Format A or
56 format B must be selected on the byte multiplexer channel. The format flags are ignored on a word channel.

68-71 Not used.

6.5.2. CCW Completion

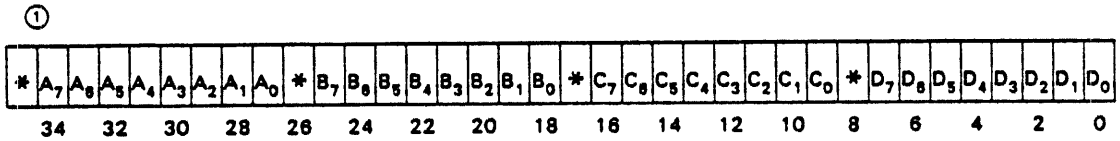
A CCW operation can be terminated by the channel or by the device. A CCW operation may also be terminated by the HDV or HCH instructions. Termination by the HCH and HDV instructions is covered in the instruction descriptions. A channel terminates the operation when the data count is exhausted. A device terminates the operation by presenting status. When a CCW operation is terminated, either a new CCW is fetched and a new operation is initiated, or an interrupt is generated. Unfortunately, a subchannel on a word channel may, instead of fetching a new CCW or generating an interrupt, return to the available state with no other action being taken. This occurs when the data count for the current operation is exhausted. the data chain, command chain, and monitor flags are

all cleared, and the device does not present an external interrupt. This condition can be prevented by setting the monitor flag. The monitor flag set and the data chain and command chain flags cleared specify that an interrupt is to be generated when the data count of the present operation is exhausted.

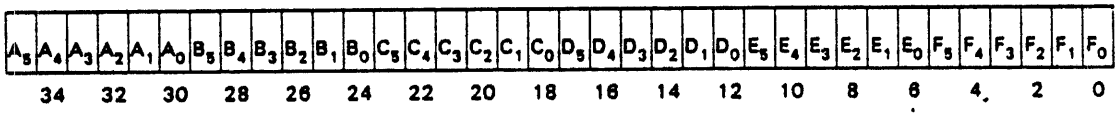
When an operation is terminated, the action taken by the subchannel is determined by the condition that caused the operation to be terminated and by the chain data, chain command, truncated search, SLI, EI chain, PCI, and monitor flags of the CCW flag field. Tables 6-9 and 6-10 illustrate the relationship between the CCW flags and the action taken.

Table 6–6. MSU Data Format – 36–Bit Format, Forward Operation

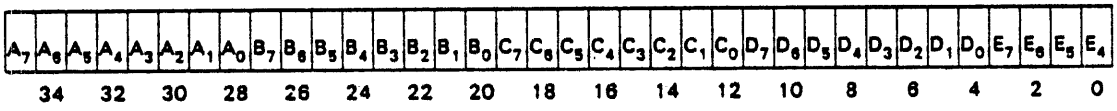
Format A



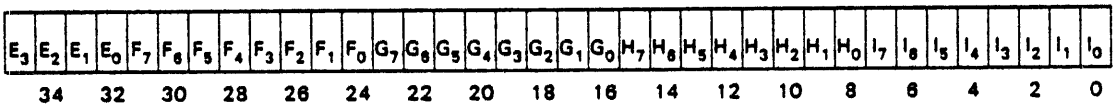
Format B



Format C



Format C (continued)

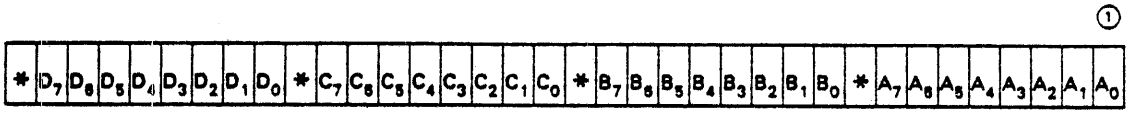


* For input operations, bits with an asterisk will be written to zero by the IOU. For output operations, bits with an asterisk are ignored.

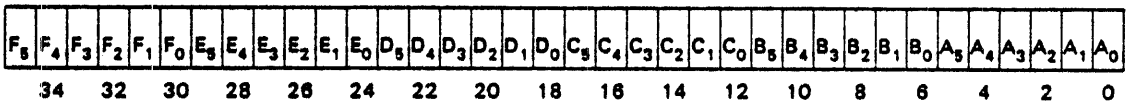
① The letters indicate the order in which bits are transferred. A indicates the first byte transferred, B indicates the second byte transferred, C indicates the third byte transferred, etc. The subscripts indicate the bit position in the byte. A 7 is the most significant bit in the byte, a 6 is the second most significant bit in the byte, a 5 is the third most significant bit in the byte, etc.

Table 6-7. MSU Data Format – 36-Bit Format, Backward Operation

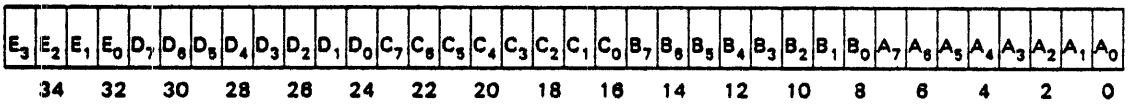
Format A



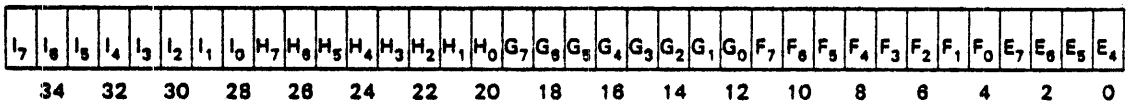
Format B



Format C



Format C (continued)



* For input operations, bits with an asterisk will be written to zero by the IOU. For output operations, bits with an asterisk are ignored.

① The letters indicate the order in which are transferred. A indicates the first byte transferred, B indicates the second byte transferred, C indicates the third byte transferred, etc. The subscripts indicate the bit position in the byte. A 7 is the most significant bit in the byte, a 6 is the second most significant bit in the byte, a 5 is the third most significant bit in the byte, etc.

Table 6-8. Format Flags vs Type of Channel

Emulate	Format A	Format B	Format C	Type of Channel	Result
X	X	X	X	Word	The E, A, B, and C flags are ignored on a word channel. The mode (36-bit) is determined by hardware feature.
0	1	0	0	Byte/Block	36-bit quarter-word format (4 bytes per word)
0	0	1	0	Byte/Block	36-bit 6-bit packed format (6 bytes per word)
0	0	0	1	Block only	36-bit 8-bit packed format (4-1/2 bytes per word)
X	0	0	1	Byte only	The operation is not initiated and the Program Check subchannel status bit is set.
All other combinations				Byte/Block	The operation is not initiated and the Program Check subchannel status bit is set.

X - Can be either 0 or 1.

Table 6-9. CCW Flags vs Termination Conditions on Byte or Block Multiplexer Channel

CD Chain Data	CC Chain Command	TS Truncated Search	SLI Suppress Length Indication	Data Count Exhausted – No Device Status	Data Count Exhausted – Chaining Status	Data Count Exhausted – Terminate Status	Data Count Not Exhausted – Chaining Status	Data Count Not Exhausted – Terminate Status
0	0	0	0	Stop	End	End	End, IL	End, IL
0	0	0	1	Stop	End	End	End	End
0	0	1	0	Stop	End	End	Truncated Search	End, IL
0	0	1	1	Stop	End	End	Truncated Search	End
0	1	0	0	Wait	Command Chain	End	End, IL	End, IL
0	1	0	1	Wait	Command Chain	End	Command Chain	End
0	1	1	0	Wait	Command Chain	End	Truncated Search	End, IL
0	1	1	1	Wait	Command Chain	End	Truncated Search	End
1	0	0	0	Data Chain	*	*	End, IL	End, IL
1	0	0	1	Data Chain	*	*	End, IL	End, IL
1	0	1	0	Data Chain	*	*	Truncated Search	End, IL
1	0	1	1	Data Chain	*	*	Truncated Search	End, IL
1	1	0	0	Data Chain	*	*	End, II Search	End, IL
1	1	0	1	Data Chain	*	*	End, IL	End, IL
1	1	1	0	Data Chain	*	*	Truncated Search	End, IL
1	1	1	1	Data Chain	*	*	Truncated Search	End, IL

Table 6-9. CCW Flags vs Termination Conditions on Byte or Block Multiplexer Channel (continued)

End	The operation is terminated. If the operation is immediate and has been specified by the first CCW associated with a Sense Release instruction, a condition code of 1 is set, and the status portion of the CSW is stored as part of the execution of the Sense Release instruction. In all other cases, an interrupt is generated in the subchannel when the channel accepts device status.
Stop	The device is signaled to terminate transfer of data, but the subchannel remains in the working state until device status is accepted; at this time an interrupt is generated in the subchannel.
Wait	The device is signaled to terminate transfer of data, but the subchannel remains in the working state until device status is accepted; if the device status is chaining status, a command chain operation is initiated; if the device status is terminate status, an interrupt is generated.
IL	Incorrect length subchannel status is indicated with the interrupt or condition code of 1.
Chain Command	The channel initiates a command chain operation upon receipt of Device End.
Truncated Search	The channel initiates a truncated search operation.
Chain Data	The channel immediately fetches a new CCW for the same operation.
*	The situation where the count is zero but data chaining is indicated at the time the device provides status cannot validly occur. When data chaining is indicated, the channel fetches the new CCW after transferring the last byte of data designated by the current CCW but before the device provides the next request for data or status transfer. As a result, the channel recognizes the status from the device only after it has fetched the new CCW, which cannot contain a count of zero unless a programming error has been made.

Table 6-10. CCW Flags vs Termination Conditions on Word Channel

Flags					Termination Condition
CD	CC	E* IC	MON	PCI	
Data Count Exhausted					
0	0	X	0	0	(1)
0	0	X	X	1	(2)
0	0	X	1	X	(2)
1	X	X	X	X	Data Chain
0	1	X	X	X	Command Chain
Data Count Not Exhausted and External Interrupt**					
X	X	0	X	X	(2)
X	X	1	X	X	EI Chain and Tabled Interrupt

(1) No action is taken. The subchannel is returned to the available state.

(2) The operation is terminated and an interrupt is generated

* EI Chain Flag is valid only on ESI subchannels.

** The situation where the data count is zero and an external interrupt is detected cannot validly occur. When the data count is exhausted, the flags are immediately inspected and the appropriate action is taken before the external interrupt is accepted.

6.6. Command Code

The command code specifies to the channel, and on a byte or block multiplexer channel also to the device, the operation to be performed. The command code assignment is listed in Table 6-11. The symbol X indicates that the bit position is ignored by the channel; M identifies a modifier bit used by the control unit or device on the byte or block multiplexer channels. The M bits are ignored on a word channel.

Table 6-11. CCW Command Code

Byte or Block Multiplexer Channel		Word Channel
Command	Code	Command
Invalid	XXXX 0000	Invalid
Sense	MMMM 0100	Invalid
Transfer in Channel (TIC)	XXX0 1000	Transfer in Channel (TIC)
Store Subchannel Status	XXX1 1000	Store Subchannel Status
Read Backward	MMMM 1100	Invalid
Write	MMFF MM01	Write
Read	MMFF MM10	Read
Control	MMMM MM11	Forced External Function
X = Not Used	M = Not Used (Word Channel) M = Modifier (Byte or Block Multiplexer Channel) F = Format Control (Word Channel) F = Modifier (Byte or Block Multiplexer Channel)	

On a byte or block multiplexer channel, commands that initiate I/O operations (Write, Read, Read Backward, Control, and Sense) cause all eight bits of the command code to be transferred to the I/O device. The modifier bits specify to the device how the operation is to be performed.

Whenever the channel detects an invalid command code during the initiation of a command, the Program Check bit in the subchannel status field is set and the operation is terminated. If the first CCW designated by the CAW contains an invalid command, the operation is terminated, the Program Check subchannel status bit is set and reported by either a condition code of 1 and an associated CSW or an interrupt. When the invalid code is detected during command chaining, the new operation is not initiated, and an interrupt is presented with the Program Check bit in the subchannel status field set. The command code is ignored during data chaining, unless the Transfer in Channel command is specified.

6.6.1. Transfer in Channel Command – TIC

The Transfer in Channel command provides an unconditional branching function in the channel. The TIC command provides for chaining between CCWs not located in sequential storage locations. This allows command and buffer loops. A new CCW is fetched from the location designated by the data address field of the TIC command. A new CCW and CCW list are immediately initiated. The TIC command can occur during data chaining or command chaining. The data count field, the format flags, and the CCW flags of a TIC command are not interpreted. The data chain or command chain operation is carried through the Transfer in Channel CCW to the new CCW.

If consecutive TIC commands are detected or if the CCW address of a TIC command is not on a double word boundary, the execution of the CCW list is terminated, and an interrupt is presented with the Program Check bit of the subchannel status field set.

6.6.2. Store Subchannel Status Command - SST

The Store Subchannel Status command provides a means of obtaining the data count of a subchannel within a CCW list without having to terminate the execution of the CCW list. When an SST command is detected, a double word CSW is stored at the location specified by the data address field of the Store Subchannel Status CCW. The device and subchannel status fields of the CSW will always be invalid. The device address and next CCW address of the CSW will be valid, and the data count will be the residue data count from the previous CCW. After storing the CSW, the execution of the CCW list is continued.

The data count field, the format flags, and all the CCW flags of an SST command are not interpreted. The SST command is detected only during command chaining.

If the data address field does not specify a double word boundary, the execution of the CCW list is terminated. The Program Check bit of the subchannel status field is set and reported by either a condition code of 1 and an associated CSW or an interrupt.

6.7. Data Transfer

Data transfers are controlled by the data address and data count fields of the CCW. The data address field contains the storage address of the first data to be transferred. The data count field of a CCW specifies the number of bytes or words to be transferred. On a byte or block multiplexer channel, the data count specifies the number of bytes to be transferred. On a word channel ISI interface, the data count specifies the number of 36-bit words to be transferred. On a word channel ESI interface, the data count specifies the number of quarter words or half words to be transferred.

6.7.1. Format Flags (E, A, B, and C)

On a word channel the format flags E, A, B, and C are not interpreted. Emulation mode on a word channel is determined on a channel basis by hardware patch wire. No formatting of data is done on a word channel. All transfers of ISI and ESI data are compatible with Series 1100 I/O data transfers for 36-bit operations.

On a byte or block multiplexer channel, the emulation flag (E) and the format flags (A, B, and C) control the formatting of data. If the emulation flag is zero, the data word width is 36 bits. An emulation flag of one is invalid.

The format flags select either the quarter-word format (A), the 6-bit packed format (B), or the 8-bit-packed format (C). Tables 6-6 and 6-7 illustrate the data formats for forward and backward operations with the 36-bit mode.

Unused bits in format A are zero filled on input operations. With the 36-bit mode of operation, when bytes are transferred to bits 0-7, 9-16, 18-25, and 27-34, the respective bits 8, 17, 26, and 35 will be written to zero.

If an input operation is not completed on an address boundary, leftover bytes and bits within a word are not affected in format A. On the block multiplexer channel with formats B and C, leftover bytes and partial bytes within a full word are zero filled and only full words are written into storage. On the byte multiplexer channel with format B, leftover bytes and bits within a word are left unchanged. With format A, individual quarter words (9 bits) are written for the 36-bit mode of operation.

Formats A and B (36-bit mode) are the only valid formats on a byte multiplexer channel. If format C is selected on a byte multiplexer channel or if more than one format is selected, the operation is terminated and the Program Check bit in the subchannel status field is set.

6.7.2. Skip Data - SK

The skip data flag in the CCW when set specifies that no data is to be transferred to storage. The skip data flag is defined only for read, read backward, and sense operations. The skip data flag is ignored in all other operations. Skipping affects only the handling of data by the channel. The operation at the I/O device proceeds normally, and data is transferred to the channel. The channel keeps updating the data count, but does not place the information in main storage. When the data count is exhausted, a new CCW is obtained if either the command chain or the data chain flag is set.

Each CCW is controlled by its individual skip data flag. Thus skipping, when combined with data chaining, permits the program to place throughout storage selected portions of a block of data from an I/O device.

6.7.3. Data Address Decrement - DAD

The data address decrement flag in the CCW when set specifies that the data address when being updated is to be decremented rather than incremented. The DAD flag is valid for all data transfer operations. The DAD flag affects only the handling of data by the channel. The operation at the I/O device proceeds normally. When the data count is exhausted, a new CCW is obtained if either the command chain or the data chain flag is set, and the new CCW is under the control of its DAD flag.

6.7.4. Data Address Lock - DAL

The data address lock flag in the CCW when set specifies that the data address is never updated. The DAL flag is valid for all data transfer operations. The DAL flag affects only the handling of data by the channel. The operation at the I/O device proceeds normally. When the data count is exhausted, a new CCW is obtained if either the command chain or the data chain flag is set, and the new CCW is under the control of its DAL flag.

6.8. Chaining Operations

When a channel has performed the transfer of data specified by a CCW, it can continue the activity initiated by the SIOF instruction by fetching a new CCW (chaining). Chaining occurs only between CCWs located in successive double word locations in storage. A CCW list is executed in an ascending order of addresses. The address of a new CCW is obtained by adding two to the address of the current CCW. Two CCW lists in noncontiguous storage locations can be connected for chaining purposes by a TIC command. On a byte or block multiplexer channel, all CCWs of a CCW list apply to the I/O device specified in the original SIOF instruction. On a word channel, all CCWs of a CCW list apply to the subchannel specified in the original SIOF.

Three types of chaining are provided: data chaining, command chaining, and EI chaining (valid only on an ESI subchannel). The specification of chaining is effectively propagated through a TIC command. When in the process of chaining a TIC command is detected, the CCW designated by the TIC is used for the type of chaining specified in the CCW preceding the TIC CCW.

A chaining operation is initiated when the operation on the present CCW is completed. A CCW operation is completed when either the data count is exhausted or the device presents status. The combination of the data count, device status, chain data flag, chain command, and EI chain flags determine what type of chaining, if any, occurs. Tables 6-9 and 6-10 outline what action is taken under all the combinations of CCW flags and termination conditions.

6.8.1. Data Chaining

Data chaining provides software with the capability of changing the data address at any time during the transfer of a block of data. Data chaining may be used to rearrange information as it is transferred between main storage and an I/O device. Data chaining permits blocks of information to be transferred to or from noncontiguous areas of storage, and when used with the skip data flag, data chaining allows the software to place selected portions of a block of data in main storage. If an output data transfer in A format is terminated by a stop code bit, data chaining will not occur.

For data chaining, the new CCW fetched by the channel defines a new storage area for the original I/O operation. Execution of the operation at the I/O device is not affected. The contents of the command code field of the new CCW is ignored unless it specifies a TIC command.

Data chaining on the byte multiplexer channel and on word ESI subchannels is executed immediately after the last byte or partial word under control of the current CCW has been transferred to storage or to the device. The old CCW is replaced by the new CCW before another data request from the device is handled. If the device presents status after exhausting the count of the current CCW, but before transferring any data to or from the storage area designated by the new CCW, the action taken by the channel is controlled by the new CCW flags. If a hardware or software error is detected when fetching the new CCW, the operation is terminated and an interrupt is generated. The channel status word (CSW) or tabled status word (TSW) associated with the interrupt will indicate why the operation was terminated.

On the block multiplexer channel and on word ISI subchannels, data chain CCWs are prefetched by the channel hardware. Each block multiplexer channel and each word ISI subchannel has an eight-word data buffer in the channel hardware to decrease the probability of data overruns. During output operations, a data chain is executed immediately after the last byte or word under control of the current CCW has been transferred to the data buffer. The old CCW is replaced by the new CCW, and the data buffer is now kept full under control of the new CCW. During input operations, the channel hardware prefetches one data chain CCW ahead during channel idle time. After the last byte or word under control of the current CCW has been transferred from the device to the data buffer, the channel continues to accept data under control of the new CCW if the new CCW has been prefetched. If a hardware or software error is detected when fetching the new CCW, the operation is terminated and an interrupt is generated. The CSW associated with the interrupt will indicate why the operation was terminated.

During data chaining, the channel hardware prefetches only one CCW ahead and the data associated with that CCW. If the byte count or word count in a data chain CCW is smaller than the data buffer (8 words), the buffering provided is limited to the size of the byte count or word count. For example, if a data chain CCW contains a byte count of one, only one byte of data buffering will be provided for that CCW. When transferring data to or from time-dependent devices, the use of data counts smaller than the buffer size increases the probability of data overruns; the smaller the data count, the higher the probability of a data overrun.

NOTE:

Late acknowledges can occur from chaining short buffers on high-speed peripherals.

6.8.2. Command Chaining

A subchannel executes a command chain operation by retrieving a new CCW and beginning execution of the command specified by that CCW. The subchannel is activated and begins executing the new command. On a byte or block multiplexer channel, the new command is also passed directly to the device. An A format stop code termination does not prevent command chaining.

Command chaining makes it possible for software to initiate the transfer of multiple blocks of data by means of a single SIOF instruction. It also allows a subchannel to initiate the execution of auxiliary functions and data transfer operations without software interference at the end of each operation. On a byte or block multiplexer channel, command chaining, in conjunction with the status modifier condition, allows the channel to modify the normal sequence of operations in response to signals provided by the I/O device.

During command chaining, the new CCW fetched by the channel specifies a new I/O operation. On a byte or block multiplexer channel, command chaining occurs only when the I/O device presents chaining status, the command chain flag is set, and the data chain flag is not set, and either the byte count equals zero or the SLI flag is set. On a word channel, command chaining occurs only when the data count of the current operation is exhausted, the command chain flag is set, and the data chain flag is not set. When command chaining takes place, the completion of the current operation does not cause an interrupt. The data count indicating the amount of data transferred during the current operation can be obtained by using an SST command as part of the command chain.

Command chaining takes place and the new operation is initiated only if no hardware error has been detected during the current operation. If a hardware error has been detected, the operation is immediately terminated and an interrupt is generated. Also, if a hardware or software error is detected when trying to initiate the new CCW of a command chain, the operation is terminated and an interrupt is generated.

An exception to the sequential chaining of CCWs occurs on a byte or block multiplexer channel when the I/O device presents the status modifier condition along with chaining status. When command chaining is specified, the combination of chaining status and the status modifier condition causes the channel to fetch and chain to the CCW whose main storage address is four higher than that of the current CCW.

6.8.3. EI Chaining (ESI Word Interface Only)

The EI chaining flag is interpreted only on an ESI subchannel. A subchannel executes an EI chain operation by placing the external interrupt in the status table if the status table subchannel is active, retrieving a new CCW from a storage address four higher than the storage address of the current CCW, and executing the command specified by that CCW. An EI chain is executed only if all of the following conditions are met:

1. The EI chain CCW flag is set.
2. An ESI external interrupt is presented.
3. The status table subchannel is active.
4. No hardware error is detected when making an entry into the status table for the external interrupt.
5. The subchannel is active.

NOTE:

If the data count on a word subchannel is exhausted and neither data chaining nor command chaining is specified, the subchannel is immediately changed from active mode to either status pending or idle mode.

6. No hardware or software error is detected when retrieving the new CCW.

Extreme care must be taken if the EI chain flag is set along with the chain data and/or chain command flags. If the channel detects exhaustion of the data count first, the data chain or command chain will be performed, and any subsequent EI chain will be under the control of the EI chain flag in the new CCW. If the channel detects the external interrupt first, an EI chain will be performed if the EI chain flag in the current CCW is set. The new CCW used by the channel is different, depending on whether the chain is an EI chain, a data chain, or a command chain. If a data chain or command chain is executed, a new CCW whose storage address is two higher than that of the current CCW is executed. If an EI chain is executed, a new CCW whose storage address is four higher than that of the current CCW is executed.

If a software or hardware error is detected when trying to initiate the new CCW during an EI chain, the operation is terminated, a TSW indicating why the operation was terminated is stored in the status table, and a tabled interrupt request is generated. The subchannel is returned to the available state.

6.8.4. Truncated Search

The truncated search capability provides software with a simple yet effective method of reading or writing multiple records on a disk control unit. A truncated search operation is executed under control of the truncated search CCW flag. A block multiplexer channel executes a truncated search operation by reissuing the command in the preceding CCW and the command in the current CCW when proper status is received from the device before the data count is exhausted.

The following is an example of a truncated search operation:

CCW	COMMAND
1	Search command (This CCW has the CC flag set.)
2	TIC command (TIC back to CCW 1.)
3	Read or Write command (This and only this CCW must have the truncated search flag set.)

The channel retrieves CCW 1 and issues the Search command and search bytes to the device. If the device does make a compare on the search bytes, device status of Channel End and Device End is presented to the channel. The channel executes a command chain, retrieves the TIC command, and then retrieves the Search command and reissues the Search command to the device. This loop continues until the device makes a compare (finds the correct record) and presents device status of Channel End, Device End, and Status Modifier. Because of the special device status, the channel skips the next CCW (CCW 2) and executes CCW 3. The Read or Write command is issued to the device and the device begins transferring data.

When the device detects the end of a record, device status of Channel End and Device End is presented to the channel. The channel checks the device status, and detects the truncated search CCW flag. The channel begins the truncated search operation by reissuing the previous command (the Search command from CCW 1) and responding to the first request for data with the "command out" interface line. The device automatically makes a compare and presents device status of Channel End, Device End, and Status Modifier. The channel reissues the Read or Write command and data transfer is initiated starting with the residual data count and data address from the previous read/write operation. This procedure is repeated at the end of each record until the byte count is exhausted.

Data chaining after the byte count is exhausted is allowed; however, further truncated search operations are executed only if the active CCW has the truncated search flag set. Command chaining after the byte count is exhausted is also allowed. See Table 6–9 to determine what action the channel takes when either the byte count is exhausted or device status is presented.

The channel unconditionally sets the most significant bit (the M–bit) of the Search command every time the Search command is issued to the device as part of a truncated search operation. The M–bit set allows the disk device to switch heads when an index mark is detected.

6.8.5. Truncated Search Restrictions

There are several programming restrictions for truncated search operations:

1. The truncated search flag is valid only for the block multiplexer and bus extension block multiplexer channels. If the truncated search flag is set in a CCW for a byte multiplexer channel, the operation is terminated immediately and program check subchannel status is presented to the software.

2. Split status from the control unit (only Channel End for Device Status) immediately terminates execution of the subchannel program. If the truncated search flag is set and a control unit splits status, the channel does not execute the truncated search operation.

3. In any CCW the suppress length indication flag is ignored if the truncated search flag is set. When the truncated search flag is set in a CCW, a command chaining operation is initiated only if the control unit presents Channel End and Device End status, the chain data flag is clear, the chain command flag is set, and the byte count for that CC has been exhausted.

   NOTE:

   A command chaining operation retrieves a new CCW and begins executing the command specified by that CCW. A truncated search chaining operation reissues the previous search and read commands.

4. Data chaining during truncated search operations is allowed. The truncated search flag needs to be set only in each CCW containing the original read or write command and not in any subsequent CCW's connected to it by data chaining. Thus, the state of the truncated search flag in the original read or write CCW governs truncated search operations for all CCW's connected to it via data chaining.

5. One truncated search operation is defined by the CCW containing the search command, the CCW containing the TIC command (optional), the CCW containing the read or write command, and all the CCW's connected to the original read or write CCW via data chaining. For all CCW's defining a truncated search operation in 36–bit mode, the byte count for each CCW except the search command CCW record length for each record must be on a byte boundary. Otherwise the byte count for each CCW except the search command CCW must be on a word boundary and the record length must be on an even word boundary. Allowable byte and word boundary CCW byte counts and record lengths for each format are given in the following tables.

Allowable CCW Byte Counts

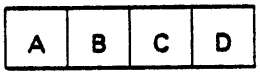
Format	Allowable Byte Boundary Byte Counts	Allowable Word Boundary Byte Counts
36–Bit Format A	Any Byte Count	4,8,12,16,20,24,28...
36–Bit Format B	Any Byte Count	6,12,18,24,30,36,42...
36–Bit Format C	Any Byte Count	5,9,14,18,23,27,32...

Allowable Record Lengths

Format	Allowable Byte Boundary Record Lengths	Allowable Word Boundary Record Length
36–Bit Format A	Any Byte Count	4,8,12,16,20,24,28...
36–Bit Format B	Any Byte Count	6,12,18,24,30,36,42...
36–Bit Format C	Any Byte Count	9,18,27,36,45...

6. In addition to the governing of byte boundary record lengths, the byte count of each record, added to the accumulative total of all previous records, must be on a byte boundary.

Example:



- Assume one CCW will be used to transfer data to all four (A–D) records.
- The length of Record A (in bytes) must be a byte boundary.
- The length of Record B (in bytes), added to the length of Record A (in bytes), must be a byte boundary.
- The length of Record C (in bytes), added to the length of Record B (in bytes), added to the length of Record A (in bytes), must be a byte boundary.

7. In order to insure adherence to Items 5 and 6, Data Chaining within record boundaries is restricted to word boundaries only.

NOTE:

Items 5, 6, and 7 apply to 36 bit format C only. Truncated search record lengths on mixed word and byte boundaries are allowed in 36 bit formats A and B.

6.9. Interrupt Generation Flags

The program controlled interrupt (PCI) flag and the monitor (MON) flag (word channel only) are interrupt generation flags that cause the subchannel to generate an interrupt. The PCI flag generates an interrupt with the PCI bit set in the subchannel status field and the MON flag generates an interrupt with the MON bit set in the subchannel status field.

6.9.1. Program Controlled Interrupt - PCI

The program controlled interrupt provides the software with a means of causing an I/O interrupt during the execution of a CCW list. The PCI flag can be in any CCW of a CCW list, but is ignored on a TIC command or an SST command. Neither the PCI flag nor the associated interrupt affects the execution of the CCW list.

The channel attempts to interrupt the program whenever the PCI bit of the CCW flags is detected during a data transfer. On all shared subchannels and nonshared block multiplexer subchannels, if the channel is presenting an interrupt for another subchannel, no action is taken. The channel will then interrupt the program when the PCI flag is detected during a data transfer and no channel interrupt request is outstanding. Nonshared subchannels on a byte multiplexer or word channel will attempt to table the interrupt after a data transfer. If the status table is valid, a TSW for that subchannel is stored in the status table. If the status table is not valid, no status is presented and the PCI flag is cleared.

A CSW containing the PCI bit may be stored by an interrupt while the operation is still proceeding or by a completion interrupt. Also, the PCI bit of the subchannel status field may accompany other valid subchannel and device status. The PCI condition cannot be detected by an instruction while the subchannel is in the working state.

If chaining occurs before the interrupt (due to the PCI flag) has been handled, the PCI condition is carried over to the new CCW. This carry-over occurs on both data and command chaining, and in either case, the condition is propagated through the TIC and SST commands. The PCI conditions are not stacked; if another CCW is fetched with the PCI flag set before the interrupt (due to the PCI flag of the previous CCW) has been handled, only one interrupt takes place. Thus, multiple PCI flags in a CCW list may result in only one interrupt.

6.9.2. Monitor - MON (Word Channel Only)

The monitor (MON) flag specifies that an interrupt be generated with the Monitor subchannel status bit set. The interrupt is not presented until the CCW operation with the MON flag is completed. The Monitor bit of the subchannel status field may accompany other valid subchannel and device status. The MON flag is interpreted only in the final CCW of a CCW list. If the data chain or command chain flag is set in a CCW, the MON flag is ignored and no interrupt is presented. If the EI chain and MON flags are set in a CCW or an ESI subchannel, the execution of the operation determines the subchannel's response. If the data count is exhausted before the external interrupt is received, an interrupt with Monitor subchannel status is generated. If an external interrupt is presented later, the external interrupt will be stored in the status table, but the EI chain will not be executed. If the data count is not exhausted when the external interrupt is received, the external interrupt will be stored in the status table and the EI chain will be executed.

6.10. Status

I/O status can be separated into the following four categories:

1. Channel Status – Hardware error that cannot be associated with a particular device or subchannel.

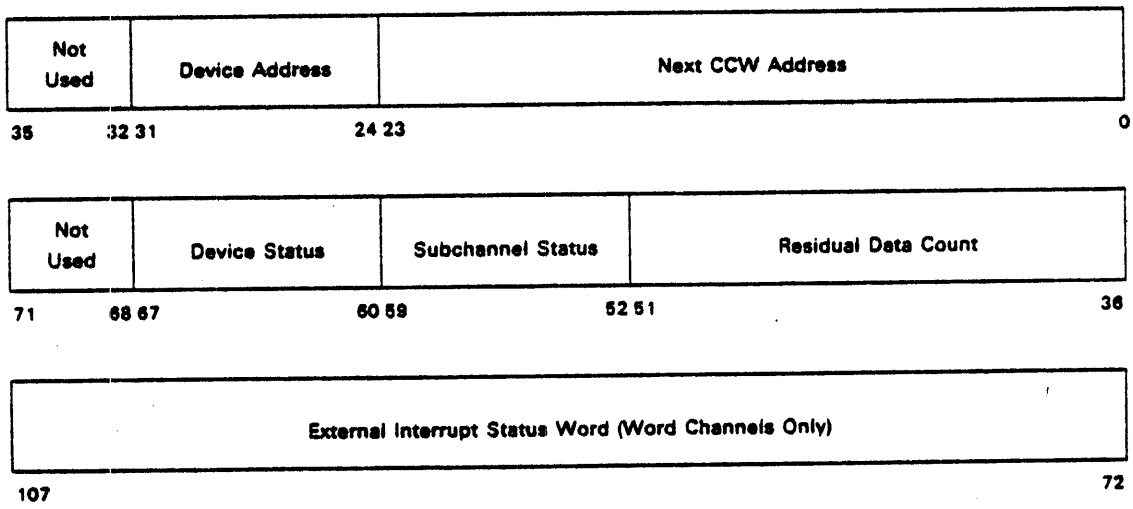
2. Status for Noncommunications Subchannels – Status caused by a device, CCW flags, or a hardware or software error on block multiplexer subchannels, byte multiplexer shared subchannels, and word ISI subchannels.

3. Status for Communications Subchannels – Status caused by a device, CCW flags, hardware error, or software error on byte multiplexer nonshared subchannels, and word ESI subchannels.

4. Status for Status Table Subchannel – Status caused by the PCI CCW flag, hardware error, or software error on the status table subchannel.

I/O status is presented either by instruction, SST command, status table, or interrupt. (See Table 6–12.) A standard channel status word (CSW) or tabled status word (TSW) is generated in all five cases. The format is:

CSW or TSW



where bits:

24–31 Contains the device number associated with the status information. On the status table subchannel, this number has no meaning.

0–23 Contains the value of the next CCW address field at the time the status information was stored. Because of prefetching conflicts, the next CCW address field of a CSW or TSW may point at a CCW specified by a TIC command. For example, assume a CCW list with a CCW at 20 data chained to a TIC CCW at address 22. The TIC points to a CCW at address 54. The next CCW address field of a CSW or TSW for this CCW list could contain 20, 22, 54, or 56. The next CCW address field of a CSW or TSW is invalid if either the program check subchannel status bit or channel control check subchannel status bit is set in that CSW or TSW.

Table 6–12. I/O Status

Type of Status	Status Generated	Status Presented to Software
Channel Status	1. Hardware error on the channel to device interface that cannot be associated with a particular device or subchannel. 2. Storage error that cannot be associated with a particular device or subchannel.	1. Machine Check interrupt
Status for Noncommunications Subchannels a. All block multiplexer subchannels b. Byte multiplexer subchannels c. Word channel ISI subchannel	1. Device 2. CCW flags (PCI or MON) 3. Software error 4. Hardware error	1. Instruction, or 2. Normal interrupt
	5. Store Subchannel Status command	3. Store subchannel status operation
Status for Communications Subchannels a. Byte multiplexer subchannels b. Word channel ESI subchannels	1. Device 2. CCW flags (PCI or MON) 3. Software error 4. Hardware error	1. Instruction, or 2. Status Table Entry and Table interrupt
	5. Store Subchannel Status command	3. Store subchannel status operation
Status for Status Table Subchannel	1. CCW flag (PCI) 2. Software error 3. Hardware error	1. LTCW Instruction, or 2. Normal interrupt with bit 24 of the IAW set

32-35 Not Used.

36-51 Contains the value of the data count existing at the time the status information was stored.

52-59 Contains a subchannel generated status byte if one or more of the following conditions is detected.

- 52 SIOF Check (Byte and block multiplexer channels)
  SIOF/EI Collision (Word channel)
- 53 Interface Control Check
- 54 Channel Control Check
- 55 Channel Data Check
- 56 Format A Stop Code (Block multiplexer channel only)
- 57 Program Check
- 58 Incorrect Length (Byte and block multiplexer channels)
  Monitor Interrupt (Word Channel)
- 59 Program Controlled Interrupt

60-67 Contains a device-generated status byte if one or more of the following conditions is detected and reported by the device.

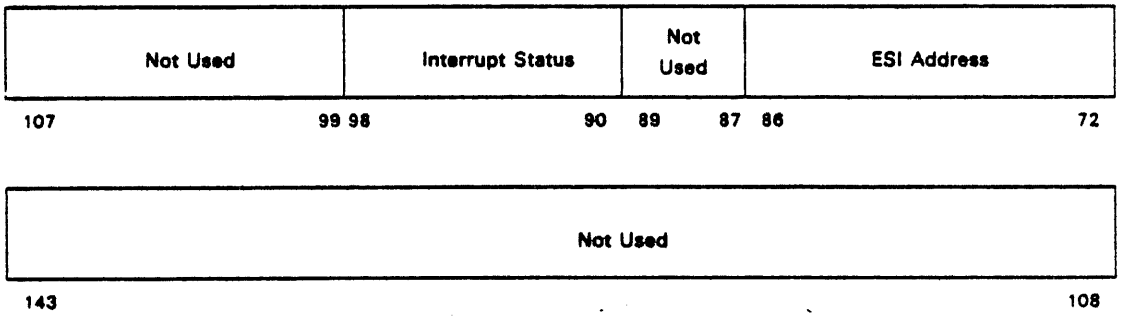
60	Unit Exception	Byte and block multiplexer channels only
61	Unit Check	Byte and block multiplexer channels only
62	Device End	Byte and block multiplexer channels only
63	Channel End	Byte and block multiplexer channels only
64	Busy	Byte and block multiplexer channels only
65	Control Unit End	Byte and block multiplexer channels only
66	Status Modifier	Byte and block multiplexer channels only
67	Attention	In word channels used to indicate the presence of external interrupt information in the third word of the CSW.

68-71 Not Used.

72-107 On a byte or block multiplexer channel, the device status field contains the actual status presented by the device. On a word channel, the only valid device status bit is the Attention bit (bit 67). On a word channel, the Attention bit set indicates that bits 72-107 of the CSW contain an external interrupt status word. The Attention bit cleared indicates that bits 72-107 of the CSW are invalid.

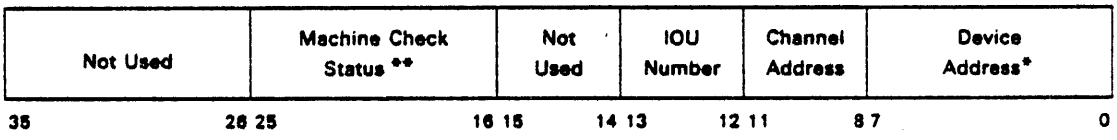
The TSW extension word contains the External Interrupt Status Word presented by the device. The format is:

TSW Extension (ESI Only)



When an I/O interrupt is acknowledged by the CPU, an interrupt address word (IAW) is stored in a fixed location of low order storage. For any I/O interrupt the standard IAW format is:

IAW



* For a machine check IAW, the Device Address is valid only if bit 21 in the status code field is set.

** Bit 24 of the field is meaningful on non-table status requests only. Bits 16 through 23 and 25 are used with Machine Check status requests only.

Each one of a possible four CPUs has one reserved address for the IAW and three reserved addresses for the CSW (See Table 6-13). Status is presented to the CPU that caused a CSW or IAW to be stored. For example, if CPU 0 executes an I/O instruction and receives a condition code of 1, the CSW is stored in the reserved CSW address locations of CPU 0. Or if CPU 3 acknowledges an I/O interrupt, the IAW and CSW are stored in the reserved IAW and CSW address locations of CPU 3. Status is reported only once and only through one path. If a subchannel that is holding status and presenting an interrupt request is then interrogated by an I/O instruction, it will present that status to either the interrupt or the I/O instruction, but not both. The channel attempts to cancel the interrupt. If the interrupt cancellation is successful, the status will be presented to the instruction and the condition code will equal 1. If the interrupt cancellation fails, the status will be presented to the interrupt. The I/O instruction will then receive a condition code of 2.

The Machine Check status bits indicate the following conditions.

25 Used with bit 21 – Indicates that multiple storage errors occurred when the channel module attempted to write one of the 128 nonshared subchannels into storage.

24 The CSW contains status information from the status table subchannel.

23 A storage error occurred when the IOU control module attempted to read the first word of the CAW.

22 A storage error occurred when the IOU control module attempted to read the second word of the CAW during a Load Channel register operation.

21 A storage error occurred when the channel module attempted to write one of the 128 nonshared subchannels into storage.

20 A storage error occurred when the channel was attempting to write the preceding Interrupt Address Word.

19 A storage error occurred when the channel attempted to write a Channel Status Word for a tabled interrupt.

18 A storage error occurred when the channel attempted to write a Channel Status Word for a non-tabled interrupt or an I/O Instruction.

17 A storage error occurred while attempting to read the second word of the CAW during a Load Channel register operation.

16 An interface control signal error or device address parity error prevented subchannel identification during a control unit initiated selection sequence. This bit is not used on a word channel.

Table 6–13. IOU Fixed Addresses

Octal	Decimal	Assignment
240	160	Processor 0 Channel Address Word 0
241	161	Processor 0 Channel Address Word 1
242	162	Unassigned
243	163	Unassigned
244	164	Processor 1 Channel Address Word 0
245	165	Processor 1 Channel Address Word 1
246	166	Unassigned
247	167	Unassigned
250	168	Processor 2 Channel Address Word 0
251	169	Processor 2 Channel Address Word 1
252	170	Unassigned
253	171	Unassigned
254	172	Processor 3 Channel Address Word 0
255	173	Processor 3 Channel Address Word 1
256	174	Unassigned
257	175	Unassigned
260	176	Processor 0 Interrupt Address Word
261	177	Processor 0 Channel Status Word 0
262	178	Processor 0 Channel Status Word 1
263	179	Processor 0 Channel Status Word 2
264	180	Processor 1 Interrupt Address Word
265	181	Processor 1 Channel Status Word 0
266	182	Processor 1 Channel Status Word 1
267	183	Processor 1 Channel Status Word 2
270	184	Processor 2 Interrupt Address Word
271	185	Processor 2 Channel Status Word 0
272	186	Processor 2 Channel Status Word 1
273	187	Processor 2 Channel Status Word 2

Table 6-13. IOU Fixed Addresses (continued)

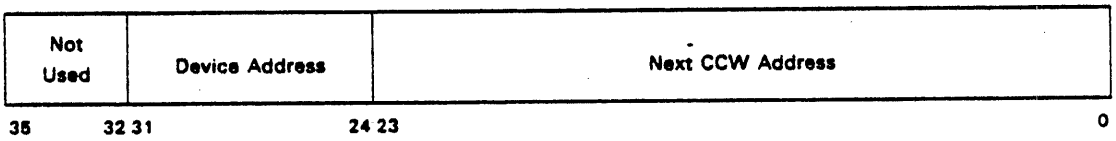
Octal	Decimal	Assignment
274	188	Processor 3 Interrupt Address Word
275	189	Processor 3 Channel Status Word 0
276	190	Processor 3 Channel Status Word 1
277	191	Processor 3 Channel Status Word 2

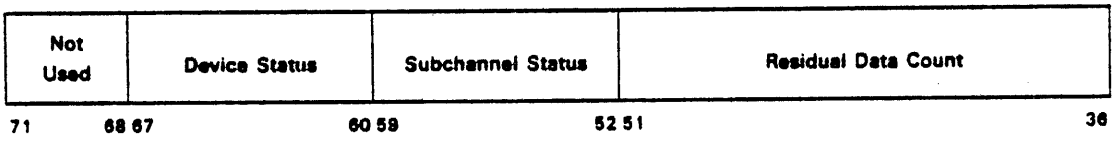
6.11. Instruction Status

If the addressed subchannel or, in the case of a HDV instruction, the addressed device is in the interrupt pending state for an I/O instruction, or if the execution of the I/O instruction generates status conditions, the channel stores a CSW in the proper reserved storage addresses and presents a condition code of 1. There are two exceptions: a CSW is not stored for a subchannel in the interrupt pending state if an interrupt request for that subchannel has been presented to the CPU and cancellation of that interrupt request was unsuccessful. Also, a CSW is not stored if the addressed subchannel is holding status for a device other than the addressed device. Note that a TSC instruction addresses only a subchannel and not a device.

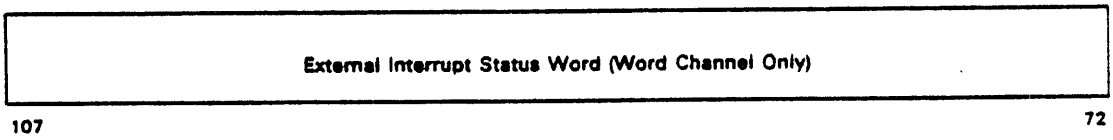
The response of a condition code of 1 to an I/O instruction always indicates that a CSW has been stored by the channel, and that the addressed subchannel has been set to the available state. The status in the CSW may pertain to either a previous operation or the attempted execution of the present instruction.

CSW for I/O Instruction





On word channels, bits 60–66 will be cleared. Bit 67 will be set only if bits 72–107 of the CSW contain an external interrupt.

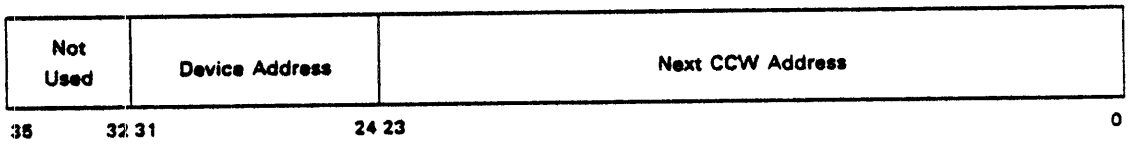


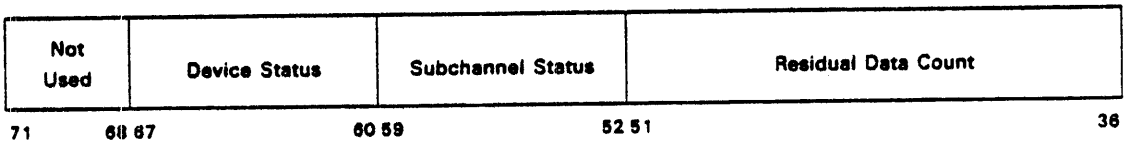
6.12. Status Table

The status table subchannel controls the status table. The status table subchannel is loaded by an LTCW instruction. An LTCW instruction initiates the execution of a status table control word (STCW) list by the status table subchannel. Bits 36–59 of the CAW contain the address of the first STCW of the STCW list. An STCW contains the data count and the starting address for the status table. The command code field is checked only for a TIC command. Any other command code value activates the status table subchannel. The chain data (CD) and program controlled interruption (PCI) flags are the only valid flags in an STCW. All the other STCW flags are ignored. Even if the status table subchannel is active, a new LTCW instruction is accepted, a new STCW list is initiated, and the status table subchannel is loaded with the first STCW of the new list.

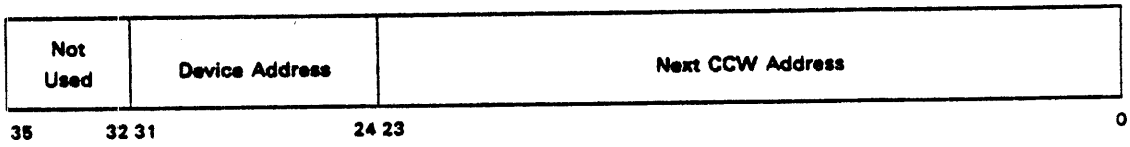
After each entry (TSW) stored in the status table, the data address field of the status table is incremented by two (byte multiplexer channel) or four (word channel), and the data count is decremented by two (byte multiplexer channel) or four (word channel). If the status table subchannel detects a data count of zero or a hardware error when attempting to make an entry in the status table, the operation of the status table subchannel is terminated and a normal interrupt request is generated.

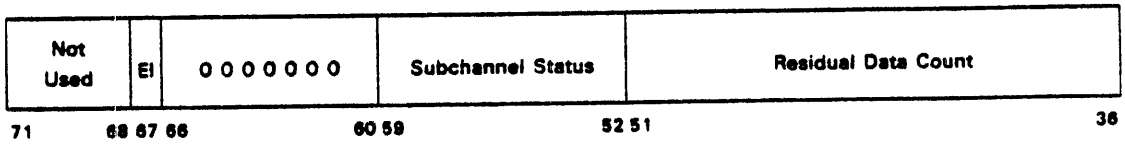
TSW for Nonshared Byte Multiplexer Subchannels





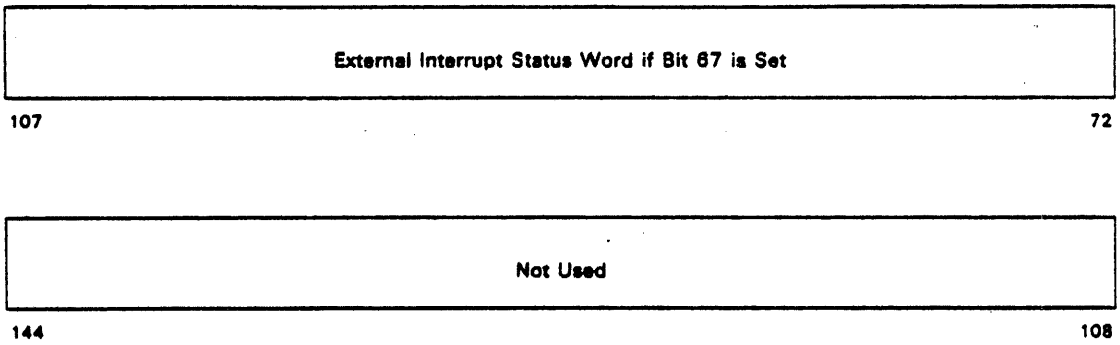
TSW for ESI Word Subchannels





EI = 0 means bits 72–107 are meaningless

= 1 means bits 72–107 contain external interrupt status word



On a word channel, the status table subchannel must be active before any ESI device requests for either data or status are handled. This restriction prevents ESI device requests from corrupting initial load operations.

Status from the status table subchannel is reported via I/O instruction or normal interrupt. The device status field of a status table subchannel CSW will always be zero filled. The only valid subchannel status bits and the conditions generating the particular subchannel status bit are the following:

1. Channel Control Check (Bit 54)

   a. A hardware fault was detected during retrieval of a status table control word.

2. Channel Data Check (Bit 55)

   a. A hardware fault was detected when making an entry in the status table.

3. Program Check (Bit 57)

   a. The status table CAW first STCW address did not specify a double word boundary.

   b. The original STCW data count field equaled zero and the command was not Transfer in Channel.

   c. The Transfer in Channel command was specified in successive STCWs.

   d. The STCW address specified by a TIC command is not on a double word boundary.

   e. The byte multiplexer channel STCW data address did not specify a double word boundary.

   f. The ESI word channel status table CCW data address did not specify a quadruple word boundary.

   g. The byte multiplexer channel STCW data count was not a multiple of two.

   h. The ESI word channel STCW data count was not a multiple of four.

   i. The channel attempted to read STCW from a location outside of available storage.

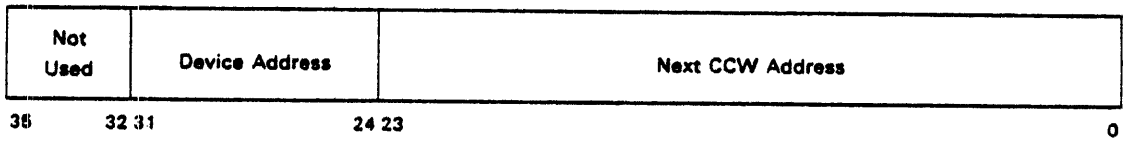
   j. The STCW data address specified a location outside the available storage.

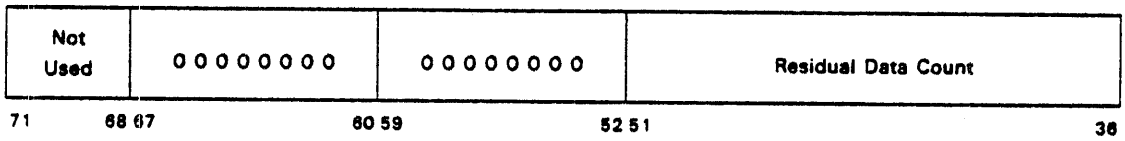
   k. The STCW data count field was decreased to zero and data chaining was not indicated.

6.13. Store Subchannel Status - SST

The only useful status resulting from an SST command will be the residual data count. The device and subchannel status fields will always be zero. Valid device status, subchannel status, or external interrupt status cannot be presented by way of an SST command. This status is reported only by way of interrupt, instruction, or the status table.

CSW for the Store Subchannel Status Command





6.14. Subchannel Status

The subchannel status bits are set when particular CCW flags are set or when a hardware or software error is detected by the channel. Subchannel status indications are presented to the software in the subchannel status field of a CSW or TSW.

6.14.1. SIOF Device Check (Byte or Block Multiplexer Channels Only)

The SIOF Device Check (bit 52) subchannel status bit indicates that a device that was to be initiated by a previously issued SIOF instruction is not operational (not installed or offline).

6.14.2. SIOF-EI Collision (Word Channel Only)

The SIOF/EI Collision (bit 52) subchannel status indication is provided only on subchannels controlling word peripheral interfaces. When set, this bit indicates that an external interrupt was received on a word interface whose subchannel was holding a pending channel program from a previously executed SIOF instruction. The queued operation has been removed from the channel hardware and will not be initiated at the device.

6.14.3. Interface Control Check

The Interface Control Check (bit 53) subchannel status bit indicates that a hardware error in the channel to device interface was detected by the channel. The hardware error was detected during one of the following operations:

1. A byte or block multiplexer channel detected a device interface parity error during the transfer of device status.

2. A word channel detected a device interface parity error during the transfer of an external interrupt status word.

3. A byte peripheral device responded with an address other than the address specified by the channel during a channel initiated selection sequence.

4. A byte peripheral device became non-operational during a command chaining.

5. A byte peripheral interface presents a combination of control signals that cause improper sequencing in the channel.

6.14.4. Channel Control Check

The Channel Control Check (bit 54) subchannel status bit indicates that a hardware error was detected by the channel when attempting to read or write a control word from storage. The hardware error was detected during one of the following operations:

1. The channel was attempting to read the second word of the CAW for an SIOF instruction.

2. The channel was attempting to read a CCW or STCW.

3. The channel was attempting to store a status word for the store subchannel status command.

4. The channel was attempting to read from main storage a control word for one of the 128 nonshared subchannels.

6.14.5. Channel Data Check

The Channel Data Check (bit 55) subchannel status bit indicates that a hardware error was detected during the transfer of data from the device to the channel, from the channel to main storage, or from main storage to the channel.

6.14.6. A Format Stop Code (Block Multiplexer Channel Only)

The A Format Stop Code (bit 56) subchannel status bit indicates that a one was detected in the most significant bit of an A format quarter word and the operation in progress was terminated. The residual byte count may or may not be zero. If data chaining was specifed in the active CCW, it will be ignored.

6.14.7. Program Check

The Program Check (bit 57) subchannel status bit is set in a CSW when the hardware detects an error in the information provided by software in a CCW, STCW or CAW. The program check bit is also set if the CCW, STCW or CAW to be fetched is in nonavailable storage. If a CSW is reported as a result of a program check, the residue count is unpredictable. The command address field of the CSW will only indicate which CAW or group of data chained CCWs or STCWs contained the word that caused the program check. The following is a list of conditions that will cause program checks:

1. The CAW first CCW address did not specify a double word boundary.

2. The status table CAW first STCW address did not specify a double word boundary.

3. A CCW or STCW contained an invalid command for an operation other than a data chain.

4. The CCW or STCW data count equaled zero and the command was neither Transfer in Channel nor Store Subchannel Status.

5. The Transfer in Channel command was specified in successive CCWs or STCWs.

6. An ESI word channel CCW contained the Forced EF command with a data count not equal to one.

7. The CCW or STCW address specified by a TIC command was not on a double word boundary.

8. The Store Subchannel Status command data address field did not specify a double word boundary.

9. The truncated search (TS) flag was specified in a byte multiplexer CCW.

10. Any byte multiplexer channel CCW or a block multiplexer channel command CCW did not specify only one format.

11. A byte multiplexer channel CCW specified format C.

12. The byte multiplexer channel STCW data address field did not specify a double word boundary.

13. The ESI word channel status table CCW data address field did not specify a quadruple word boundary.

14. The byte multiplexer channel STCW data count field was not a multiple of two.

15. The ESI word channel STCW data count field was not a multiple of four.

16. The channel attempted to read a CCW or STCW from a location outside of available storage.

17. A CCW or STCW data address specified a location outside the available storage.

18. The STCW data count field was decreased to zero and data chaining was not indicated.

6.14.8. Monitor (Word Channel Only)

The Monitor (bit 58) subchannel status bit indicates that a CCW list on a word subchannel has been completed. The Monitor subchannel status bit is set and an interrupt is generated when an operation is completed on a CCW that has the monitor CCW flag set and does not have either the data chain flag set or the command chain flag set. If a CCW has either the data chain or the command chain flag set, the monitor flag is not interpreted and no interrupt is generated.

6.14.9. Incorrect Length (Byte or Block Multiplexer Channels Only)

The Incorrect Length (bit 58) subchannel status bit indicates that the number of bytes specified in the CCWs for an I/O operation was not equal to the number of bytes requested or offered by the device. The incorrect length indication is presented only if the active CCW has neither the truncated search flag nor the suppress length indication flag set, and the data chain flag is not set. The incorrect length indication is also suppressed on immediate commands. Detection of an incorrect length condition causes the operation to be terminated and an interrupt to be generated. See Table 6-9 for the affect of the CD, CC, SLI, and TS flags on the indication of incorrect length.

6.14.10. Program Controlled Interrupt

When the channel detects the PCI flag during a data transfer, an interrupt and/or a table entry in the status table is attempted for that subchannel. The PCI (bit 59) subchannel status bit indicates that a PCI flag had been detected in a CCW or STCW list. Because the program controlled interruption does not affect the execution of a CCW or STCW list, the detection of several PCI flags in a CCW or STCW list may result in only one interrupt with the PCI subchannel status bit set. Each PCI flag, however, never generates more than one interrupt or one TSW entry.

6.15. Device Status

If the device presents termination status on a byte multiplexer shared subchannel or a block multiplexer subchannel, the operation is terminated and an interrupt is generated if an interrupt from another subchannel is not already being presented. If an interrupt is being presented, the status is stacked in the device. When the interrupt mechanism becomes available, the status is accepted from the device, and an interrupt is generated. If the device presents termination status on a byte multiplexer nonshared subchannel, the operation is terminated, and a status table entry is executed if the status table subchannel is active. If the status table subchannel is inactive, no status table entry is made, and the device status is lost.

When a device on a byte or a block multiplexer channel presents chaining status, a command chain is performed only if the command chain flag is set and the data chain flag is not set. Otherwise, the operation is terminated, and an interrupt is presented as previously explained.

On a word channel ISI interface, an external interrupt terminates the operation and generates an interrupt. On a word channel ESI interface, an external interrupt terminates the operation and generates a status table entry if the status table is active. If the status table is inactive, no table entry is made, and the external interrupt is lost. There is one exception for terminating an operation. An external interrupt will generate an entry into the status table and cause an EI chain to be performed if the operation is still active, the EI chain flag is set, and the status table subchannel is active.

The byte or block multiplexer channel device status bits are:

Status Codes	Status
1000 0000	Attention
0100 0000	Status Modifier
0010 0000	Control Unit End
0001 0000	Busy
0000 1000	Channel End
0000 0100	Device End
0000 0010	Unit Check
0000 0001	Unit Exception

For the word channel, the device status field will always be zero with the exception of bit 67. If bit 67 of the CSW or TSW is set, bits 72–107 contain an external interrupt status word. If bit 67 is cleared, bits 72–107 of the CSW or TSW are meaningless.

After presenting an interrupt with any device status or any subchannel status other than the PCI bit, the subchannel is returned to the available state. The associated device, however, may not be available because device status may have split or the device may have not presented status by the time the interrupt was acknowledged. After initiating a CCW list, the software may have to handle any number of interrupts before both the subchannel and device are available.

NOTE:

All bits of the subchannel status and device status fields must be investigated for each CSW or TSW, because several device status bits and/or several subchannel status bits may be set in one CSW or TSW.

6.16. Data Chaining Precautions

There are several precautions that should be taken during data chaining operations. Data chaining with small data counts on high-speed devices that are capable of data overruns should be avoided. The following precautions should also be noted when data chaining on the byte or block multiplexer channels:

1. On the byte multiplexer channel, switching of the byte packing formats (the E, A, B, and C CCW flags) is allowed between data chained CCWs and command chained CCWs. On the block multiplexer channel, the format flags in the command CCW specify the format for the entire data chained CCW list. On the block multiplexer channel, the format flag field in all data chained CCWs is ignored, and the format flags can only be changed by executing a command chained CCW.

2. Format A (36-bit mode) – If a CCW byte count for an operation is not completed on a full word boundary, leftover bytes in the final data word are unaffected.

3. Byte multiplexer channels – Format B (36-bit mode) – If a CCW byte count for an input operation is not completed on a full word boundary, leftover bytes in the final data word are unaffected.

4. Block multiplexer channels – Format B (36-bit mode) – If a CCW byte count for an input operation is not completed on a full word boundary, leftover bytes in the final data word are zero filled. Only full 36-bit words are transferred to main storage in format B on the block multiplexer channel.

5. Formats A and B (36-bit mode) – If a CCW byte count is not completed on a full word boundary and data chaining is indicated, the first byte transferred under control of the new CCW is stored in or read from the initial byte position for that format.

6. Format C (36-bit mode) – If a CCW byte count for an input operation is not completed on a full word boundary, the leftover bytes and partial bytes to complete only that word are zero filled. Only full words are transferred to storage in format C. The one exception is partial byte remainders. See item 7.

7. Format C (36-bit mode) – If a CCW byte count is exhausted on a partial byte boundary (a boundary that requires the final byte to be split into two words for input, or a boundary that requires part of the final byte to be taken from another word for output) and data chaining is not indicated, the partial byte remainder is thrown away on input. On output the final output byte is formed as described in Table 6-14 for the applicable 36-bit format C forward operation.

8. Format C (36-bit mode) – If a CCW byte count is not completed on a full word boundary and data chaining is indicated, the first byte transferred under control of the new CCW is stored or read from the initial byte position for format C. There is one exception. If a CCW byte count

is exhausted on a partial byte boundary and data chaining is indicated, the packing or unpacking of data continues as if the data chain never occured. For example, on input the first part of the final byte (the byte that is on the partial byte boundary) is transferred to storage under control of the original CCW. The partial byte remainder is then transferred to storage under control of the new CCW. It should be noted that when the final data chained CCW byte count is exhaused on a partial byte/9N+5 boundary, the partial byte is not transferred to storage on input. On output the final output byte is formed as described in Table 6-14 for the applicable 36-bit format C forward operation.

9. Block multiplexer channel and ISI word interfaces - Because the channel prefetches one CCW ahead during data chaining, there is one restriction on dynamically modifying CCW lists during input data chaining operations. If a channel is executing CCW A and data chaining to CCW B is specified, CCW B cannot be part of the data buffer being retrieved by CCW A because the channel may have prefetched CCW B before CCW B was transferred to storage as part of the data buffer of CCW A.

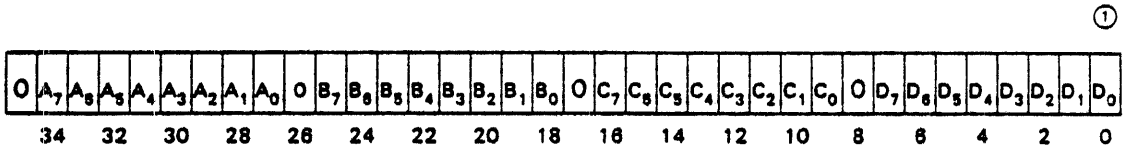
10. Word channels - Certain chaining operations can be executed by either a data chain or command chain. In these situations always use data chaining.

See Table 6-14 for examples of the above items.

Table 6-14. Byte Data Packing on Abnormal Boundaries

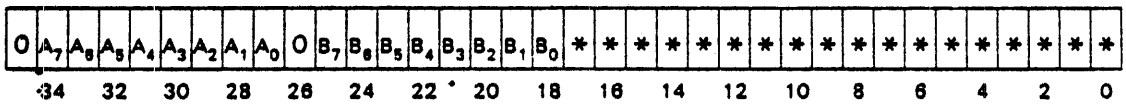
36-Bit Format A - Forward Operation

Storage Address Specified by CCW1

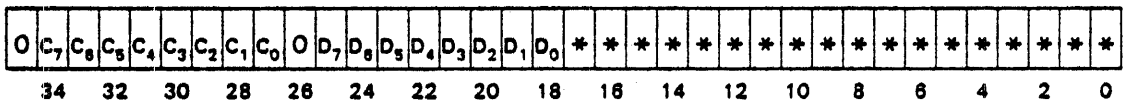


Byte count = 4

Storage Address Specified by CCW1

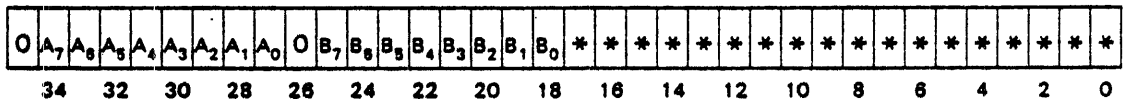


Storage Address Specified by CCW2



Byte count = 2 Data chained to a byte count of 2

Storage Address Specified by CCW1



Byte count = 2 or status after 2 bytes

* Bit positions with an asterisk will be written into on input and ignored on output.

0 These bit positions set to zero by hardware on input and ignored on output.

① The letters indicate the order in which bits are transferred. A indicates the first byte transferred, B indicates the second byte transferred, C indicates the third byte transferred, etc. Numbers indicate the bit position in the byte. A 7 is the most significant bit in the byte, a 6 is the second most significant bit in the byte, a 5 is the third most significant bit in the byte, etc.

Table 6–14. Byte Data Packing on Abnormal Boundaries (continued)

36–Bit Format B – Forward Operation

Storage Address Specified by CCW1

A_5	A_4	A_3	A_2	A_1	A_0	B_5	B_4	B_3	B_2	B_1	B_0	C_5	C_4	C_3	C_2	C_1	C_0	D_5	D_4	D_3	D_2	D_1	D_0	0	0	0	0	0	0	0	0	0	0	0	0
34		32		30		28		26		24		22		20		18		16		14		12		10		8		6		4		2		0	

Storage Address Specified by CCW2

E_5	E_4	E_3	E_2	E_1	E_0	F_5	F_4	F_3	F_2	F_1	F_0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
34		32		30		28		26		24		22		20		18		16		14		12		10		8		6		4		2		0	

Byte count = 4 Data chain to byte count = 2

Storage Address Specified by CCW1

A_5	A_4	A_3	A_2	A_1	A_0	B_5	B_4	B_3	B_2	B_1	B_0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
34		32		30		28		26		24		22		20		18		16		14		12		10		8		6		4		2		0	

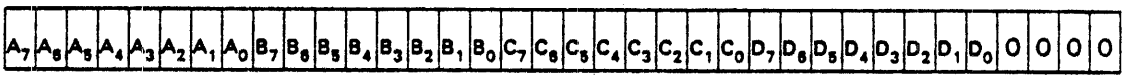
Byte count = 2 or status after 2 bytes

0 These bit positions are set to zero by hardware on input and ignored on output.

Table 6-14. Byte Data Packing on Abnormal Boundaries (continued)

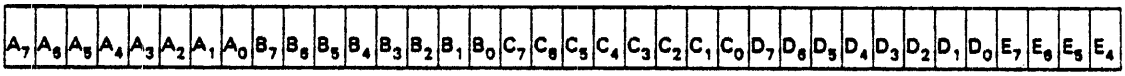
36-Bit Format C – Forward Operation

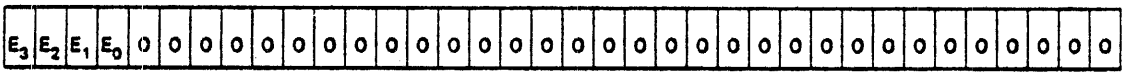
Storage Address Specified by CCW1



Data count = 4 or status after 4 bytes

Storage Address Specified by CCW1 (9N+5 Boundary)





Data count = 5 and status after five bytes

E_3 to E_0 are not transferred to storage on input

E_3 to E_0 are created from A_7 to A_4 on output (invalid data)

– OR –

Data count > 5 and status after five bytes

E_3 to E_0 are not transferred to storage on input

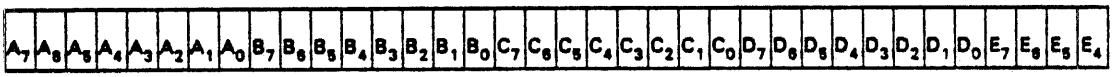
E_3 to E_0 are transfered to the device on output (valid data)

0 These bit positions set to zero by hardware on input and are ignored on output.

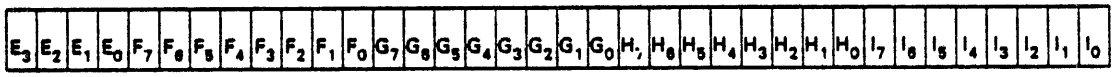
Table 6–14. Byte Data Packing on Abnormal Boundaries (continued)

36–Bit Format C – Forward Operation

Storage Address Specified by CCW1

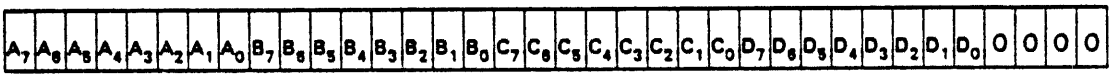


Storage Address Specified by CCW2

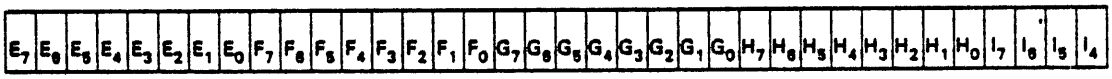


Data count = 9

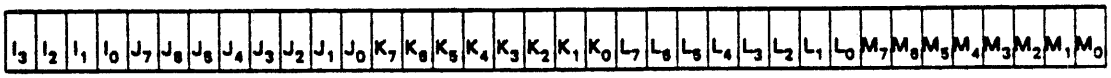
Storage Address Specified by CCW1



Storage Address Specified by CCW2



Storage Address Specified by CCW2



Data count = 4 Data chained to data count = 9

0 These bit positions are set to zero by hardware on input and ignored on output.

Table 6–14. Byte Data Packing on Abnormal Boundaries (continued)

36–Bit Format C – Forward Operation

Storage Address Specified by CCW1

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0	C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	E_7	E_6	E_5	E_4
34		32		30		28		26		24		22		20		18		16		14		12		10		8		6		4		2		0	

Storage Address Specified by CCW2

E_3	E_2	E_1	E_0	F_7	F_6	F_5	F_4	F_3	F_2	F_1	F_0	G_7	G_6	G_5	G_4	G_3	G_2	G_1	G_0	H_7	H_6	H_5	H_4	H_3	H_2	H_1	H_0	I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0
34		32		30		28		26		24		22		20		18		16		14		12		10		8		6		4		2		0	

Data count = 5 Data chained to data count = 4

Storage Address Specified by CCW1

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0	C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	E_7	E_6	E_5	E_4
34		32		30		28		26		24		22		20		18		16		14		12		10		8		6		4		2		0	

Storage Address Specified by CCW1

E_3	E_2	E_1	E_0	F_7	F_6	F_5	F_4	F_3	F_2	F_1	F_0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
34		32		30		28		26		24		22		20		18		16		14		12		10		8		6		4		2		0	

Storage Address Specified by CCW2

G_7	G_6	G_5	G_4	G_3	G_2	G_1	G_0	H_7	H_6	H_5	H_4	H_3	H_2	H_1	H_0	I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0	J_7	J_6	J_5	J_4	J_3	J_2	J_1	J_0	0	0	0	0
34		32		30		28		26		24		22		20		18		16		14		12		10		8		6		4		2		0	

Data count = 6 Data chained to data count = 4

0 These bit positions are set to zero by hardware on input and ignored on output.

6.17. Subchannel Expansion Feature and Channel Base Register

As described in 6.2.2 and 6.3.1, the subchannel expansion feature enables the channel to maintain the control words for the four or eight (depending on the option) most recently active subchannels in the channel while storing the control words of the remaining 120 or 124 subchannels in main storage. Four storage addresses per subchannel and 512 addresses per channel must be reserved for each channel with the subchannel expansion feature. These addresses are a hardware scratch pad and are for hardware use only and should not be interrogated or changed by software except during initialization or error recovery. During system initialization, the software must initialize these addresses by setting equal to 1_{16} bits 24–27 of every fourth address (each address with bits 0 and 1 equal to 0). The format of the control words held in storage is shown in Table 6–15.

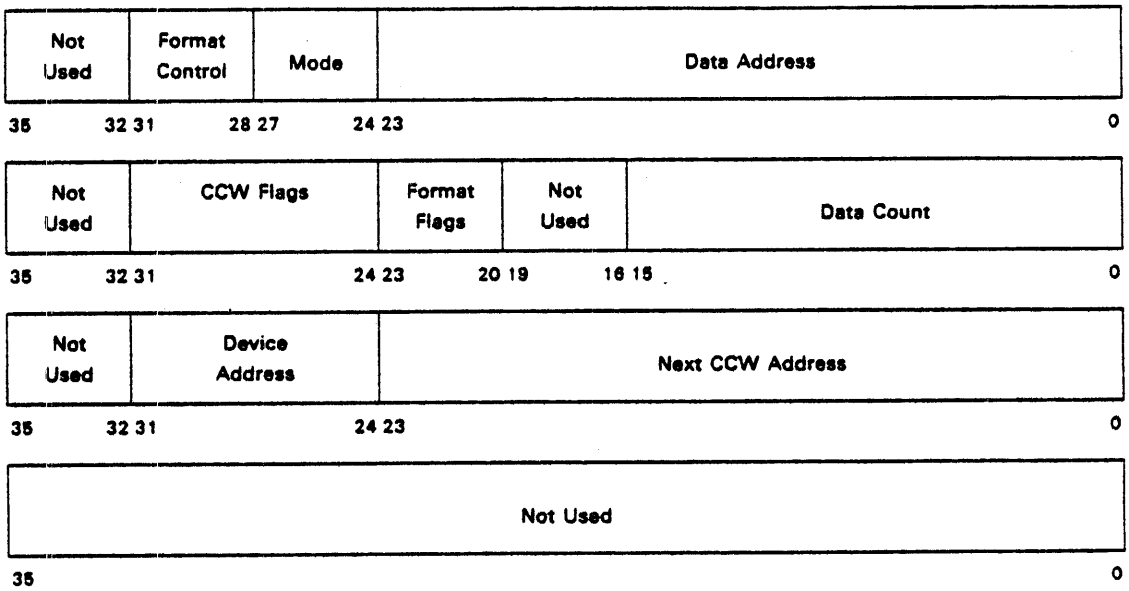
If the mode bits 27–24 = 0001_2, all four words are not used. All other values of the mode bits indicate a hardware fault.

Each channel that has the subchannel expansion feature has a channel base register that consists of 15 bits. The channel base register specifies the location of the 512 addresses that are the channel's scratch pad. The channel base register provides bits 09 through 23 of the storage addresses that are used when swapping subchannel control words between main storage and the channel.

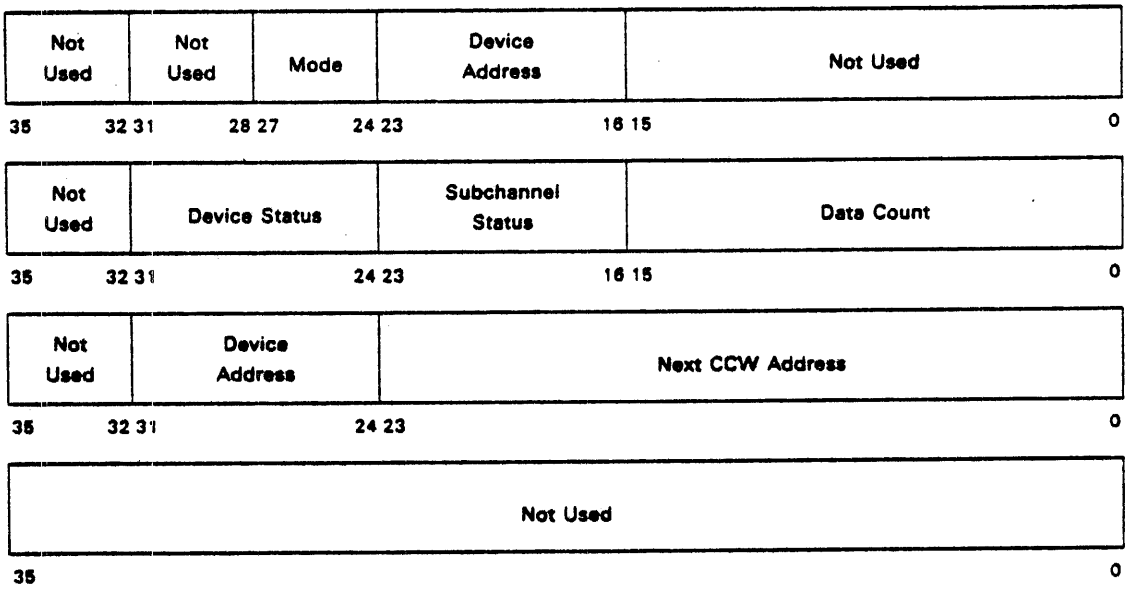
6.18. Interrupt Mask Register

The interrupt mask register is loaded with the contents of bits 36–71 of the CAW during an LCR instruction that has bit 0 of the CAW set. One interrupt mask register is provided in each IOU. This register provides the capability of determining which channel modules are allowed to present communications or noncommunications interrupts. It also provides a means of selecting to which CPU or CPUs the interrupts are to be sent. The register is divided into four bytes. Two bytes for each CPU. These two bytes are then separated into communications or noncommunications interrupts. The interrupt mask register has the format shown in Table 6–16. A set bit suppresses interrupts of the specified type from the corresponding channel module to the corresponding CPU; i.e., a one bit in position 39 suppresses the reporting of noncommunications interrupts on channel module 3 to CPU 0. If bit position 57 is also set, the reporting of noncommunications interrupts on channel module 3 is completely suppressed. An interrupt that is completely suppressed will be reported when either of the corresponding mask bits is later changed to 0. Any interrupts that are currently being presented to the CPU when an LCR instruction is received will be reported before the interrupt mask takes effect. In a unit CPU system, software must mask out both communications and noncommunications interrupts to the nonexistent CPU during system initialization and each time the LCR instruction is used.

Table 6-15. Scratch Pad Formats for Subchannel Expansion Feature



Format above for: Mode Bits 27-24 = 0010_2 or 1000_2 or 1001_2 or 1010_2 or 1100_2



Format above for: Mode Bits 27-24 = 0000_2 or 0011_2

Table 6-16. Interrupt Mask Register

Processors 1 and 3		
Bit Positions of the CAW	71 70 69 68 67 66 65 64 63	62 61 60 59 58 57 56 55 54
Channel Module Number	* 7 6 5 4 3 2 1 0	* 7 6 5 4 3 2 1 0
Interrupt Type	Communications	Noncommunications
Processors 0 and 2		
Bit Position of the CAW	53 52 51 50 49 48 47 46 45	44 43 42 41 40 39 38 37 36
Channel Module Number	* 7 6 5 4 3 2 1 0	* 7 6 5 4 3 2 1 0
Interrupt Type	Communications	Noncommunications

* *Not used*

6.19. Initial Load

The initial load capability is included in each channel. The initial load operation is performed in only the 36-bit mode of operation. On a byte or block multiplexer channel, format C (8-bit packed) is specified, the starting data address is set to MSR, and the byte count is set to 4608_{10}. On a word channel, the starting data address is set to MSR and the word count is set to 1024_{10}. The size of the initial load boot block is determined by the channel and the device. The channel will terminate the operation after 1024_{10} full 36-bit words have been loaded into storage. If the device presents status before 1024_{10} words have been loaded, the channel immediately terminates the operation and presents an interrupt to the CPU

6.20. Back-to-Back Operation (Word Channel Only)

The word channel back-to-back capability will allow the software to execute block transfers or scatter/gather operations via an I/O channel. Two ISI interfaces on the same channel are required, an output interface and an input interface. The back-to-back interfaces must be initialized after the IOU is master cleared, after any hardware or software error on the back-to-back interfaces, or after a back-to-back operation that did not have the output buffer data count equal to the input buffer data count. The back-to-back interfaces are initialized by using the following procedure:

1. To the output interface, issue an SIOF instruction that initiates the execution of one CCW. The CCW must have a forced external function command and a data count of one.

2. Handle the external interrupt from the input interface. This external interrupt was generated by the forced external function operation on the output interface.

To execute a back-to-back operation the following procedure must be followed:

1. Issue an SIOF instruction to the input interface to activate the input buffer.

2. Issue an SIOF instruction to the output interface to activate the output buffer.

Once the back-to-back interfaces are initialized, many back-to-back operations may be performed by following the above procedure for each operation. Data chaining is allowed on both the output and input interfaces for each back-to-back operation, but the total output buffer data count must equal the total input buffer data count. The use of the DAD, SK, DAL, PCI, and MON CCW flags is allowed, but command chaining is prohibited.

6.21. Priorities

The control module establishes data handling priority among the eight channel modules. Channel module 0 has the highest priority and channel module 7 has the lowest priority. The channel module gives highest priority to data transfers, second highest priority to interrupts, and lowest priority to instructions. Word channel data handling priority is a homing priority with interface A having the highest priority, then interfaces B, C, and D, with interface D having the lowest priority.

6.22. Basic Programming Procedure

The programmer should use the following basic procedure in order to execute a series of operations on a byte or block multiplexer device or a word subchannel:

1. Build the list of CCWs, making sure that the correct flags in each CCW are set, and build any necessary external function words or data buffers.

2. Load the address of the first CCW in Xa bits 0-23.

3. Load the u and Xm registers such that u + Xm bits 0-12 specifies the IOU, channel, and device numbers.

4. Disable I/O interrupts.

5. Issue the SIOF instruction.

6. Test the condition code to determine the result of the SIOF instruction. (Note that the next instruction is skipped if the condition code equaled 0 and the CPU did not timeout the instruction.)

7. If the instruction is not timed out by the CPU and a condition code of 0 is received, enable I/O interrupts and continue with the CPU program. If another condition code is received or the instruction is timed out, the appropriate action should be taken.

8. Wait for the I/O interrupt or interrupts. Use the resultant status to determine if the CCW list was executed successfully. If the CCW list was terminated before it was completed, the status will contain enough information to determine how much of the CCW list was executed, and why the CCW list was terminated before it was completed.

6.23. Programming Examples

For an example of the block multiplexer channel CCW list see Figure 6–3. The execution of this CCW list is initiated by an SIOF instruction with a CCW address of $B0_{16}$. The CCW list is executed as follows:

1. The channel reads the first CCW and issues the Read command to the device.
2. Nine bytes are transferred from the device to the channel, but none of the bytes are written into storage because the skip data flag is set.
3. The device presents Channel End and Device End status (chaining status).
4. The channel initiates the command chain and issues the Write command to the device.
5. Thirty–six bytes are transferred from the channel to the device. All the bytes are transferred from the same storage address because the data address lock flag is set.
6. The channel executes a data chain and transfers two bytes from storage address $F0_{16}$ and two bytes from storage address EF_{16} because the data address decrement flag is set.
7. The device presents Channel End and Device End status (chaining status).
8. The channel initiates the command chain and executes the Store Subchannel Status command. A two word CSW is stored at storage address 44_{16}. In the CSW, the next CCW address field equals 88, the data count field equals zero, and the subchannel and device status fields equal zero.
9. The channel continues the command chain and issues the read backward command to the device and then terminates the operation because an illegal combination of format flags is detected.
10. The device presents Channel End and Device End status.
11. The channel accepts the device status and presents an interrupt request to the CPU.
12. The interrupt is acknowledged and a CSW is written with the next CCW address field equal to BA_{16}, the data count field equal to 1_{16}, the Program Check and PCI subchannel status bits set, and the Channel End and Device End device status bits set.

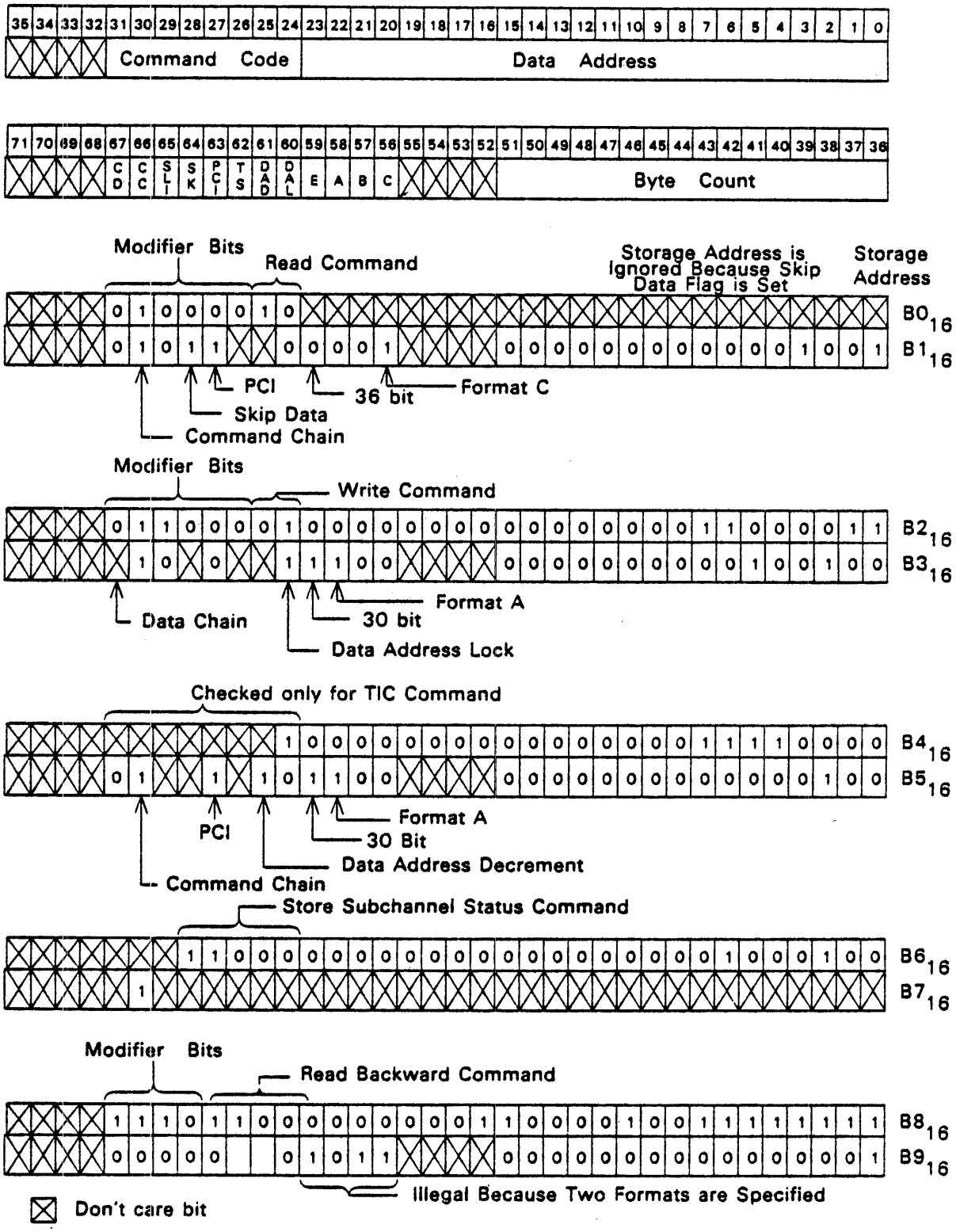


Figure 6-3. Block Multiplexer Channel Example CCW List

The interrupt mechanism was assumed to be busy during the execution of this CCW list, causing the PCIs to be overlaid.

For an example of the word channel ISI interface CCW list see Figure 6-4. The execution of this CCW list is initiated by an SIOF instruction with a CCW address of A2. The CCW list is executed as follows:

1. The channel reads the first CCW and issues the forced external function to the device. The forced external function contains a Write command for the device.

2. The channel then executes the command chain. The TIC command is executed and the CCW address field is changed to 54_{16}.

3. The channel continues the command chain and executes the Write command by activating the output data buffer.

4. Four words are transferred from the channel to the device. All the words are transferred from the same storage address because the data address lock flag is set.

5. The channel executes a data chain and transfers three words from storage addresses 29_{16}, 28_{16}, and 27_{16} because the data address decrement flag is set.

6. The channel executes a command chain and issues a forced external function to the device. The forced external function contains a Read command for the device.

7. The channel executes the Read command by activating the input buffer.

8. Two words are transferred from the device to the channel but not to storage because the skip data flag is set.

9. The channel terminates the operation and presents an interrupt request to the CPU when the interrupt mechanism is free.

10. The interrupt is acknowledged and a CSW is written with the next CCW address field equal to 62_{16}, the data count field equal to zero, and the Monitor and PCI subchannel status bits set.

The interrupt mechanism was assumed to be busy during the execution of the CCW list, causing the PCIs to be overlaid. The device was assumed to be non-interrupting. On an ISI interface, an external interrupt immediately terminates the execution of a CCW list.

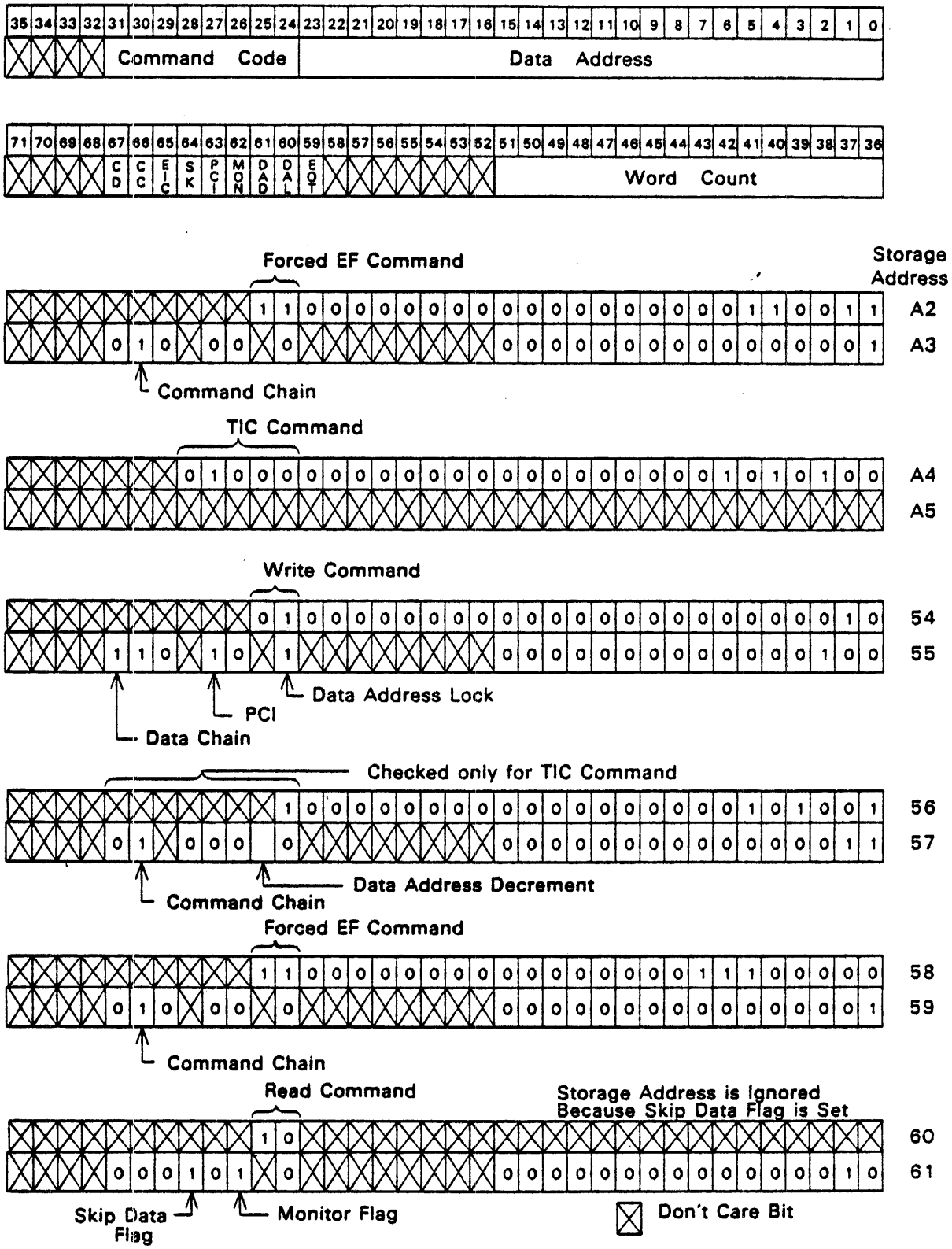


Figure 6–4. Word Channel ISI Interface Example CCW List

7. Interrupts

7.1. General

An interrupt causes the current instruction sequence to be suspended and an instruction sequence starting at a fixed storage location to be initiated; the fixed address replaces the value in the program address register. The fixed storage address is associated with the event or condition that caused the interrupt to be generated, and thereby allows switching to a program to respond to that condition or event. Excepting those instructions that are explicitly named as interruptible, such as repeated instructions like Block Transfer, the central processor units (CPU) honors interrupts only after the current instruction is completed and only if the interrupt to be honored is allowed. Table 7–1 shows the interrupts and their priorities. The interrupts are categorized by levels of priority. The following interrupts are always allowed:

- All program exception interrupts, including Guard Mode and Addressing Exception interrupts.
- All arithmetic exception interrupts, including Characteristic Overflow, Characteristic Underflow, and Divide Check interrupts.
- Certain program initiated interrupts, including Executive Request, Test and Set, Quantum Timer, Breakpoint, and Emulation interrupts.
- Storage Check interrupts caused as the result of transfers between the CPU and storage interface unit (SIU).

Certain interrupts are disallowed between the execution of a Prevent All Interrupts and Jump (PAIJ) instruction or the occurrence of an interrupt, and the execution of an Allow All Interrupts and Jump (AAIJ) instruction or User Return (UR) or Load Designator Register (LD) instruction that sets the allow interrupts designator (D3). These include the following:

- All I/O interrupts, including those for Normal status, Tabled status, and Machine Check interrupts.
- Power Check interrupts

Table 7-1. Interrupt Priority

Priority Level	Interrupt Type
0	Immediate Storage Check (oper port)
	Immediate Storage Check (inst port)
1	Guard Mode (oper port)
	Guard Mode (inst port)
	IOU 0 ERR
	IOU 1 ERR
2	Addressing Exception
	Invalid Instruction
	Executive Request
	Test and Set
	Characteristic Overflow
	Characteristic Underflow
	Divide Check
3	Emulation
4	Breakpoint
5	Quantum Timer
6	Jump History Stack
7	Power Restored
	Power Loss
8	Real Time Clock
9	Dayclock
10	Delayed Storage Check Upper
	Delayed Storage Check Lower
11	IOU 0 Machine Check
12	IOU 1 Machine Check
13	IOU 0 Normal Status
14	IOU 1 Normal Status
15	IOU 0 Tabled Status
16	IOU 1 Tabled Status
17	IPI P
18	IPI P+1 or P-3
19	IPI P+2 or P-2
20	IPI P+3 or P-1

NOTES:

1. *Priority levels 0 through 5 are internal interrupts, which can be neither locked out nor deferred (always allowed).*
2. *Priority levels 6 through 20 are external interrupts, which can be both locked out and deferred.*
3. *For interprocessor interrupts, P is the CPU number of the CPU being interrupted. For priority levels 18, 19, and 20 use the term that gives an IPI number from 0 to positive 3.*

- Interprocessor interrupts (IPIs)

NOTE:

If interrupts are locked out and the CPU is stopped via Halt Jump (HJ) instruction, Interprocessor interrupts and Power Check restoration interrupts are allowed; and if the CPU is stopped in the cleared state, in addition to Interprocessor interrupts, I/O normal status interrupts are allowed if the CPU has been selected for initial load. (The I/O normal status interrupts are not allowed until software has sent a function to the channel or word channel interface.)

- Dayclock and Real-Time Clock interrupts.
- Storage Check interrupts caused as the result of transfers from the SIU to storage units.
- Jump History Stack interrupts.

7.2. Interrupt Sequence

When the CPU honors an interrupt request, the following events occur:

- The processor state is stored in the general register stack (GRS) in three groupings: program status (041–044, 050–052, and 056–057), addressing status (040 and 045–047 not actually stored during interrupt), and interrupt status (053–055).
- All designator register bits are set to zero except the GRS selection designator (D6) and the relocation and storage suppression designator (D7), which are set to one, and the BDR selector designator (D12), which is not altered.
- External interrupts are prevented from occurring until allowed by an AAIJ, IJR, or LD instruction.
- Control is transferred to the associated interrupt location. Note that this instruction must be an unconditional jump, but need not be a Load Modifier and Jump (LMJ) instruction. LMJ may capture the wrong relative address because of the processor state change (e.g., LBJ interrupt).

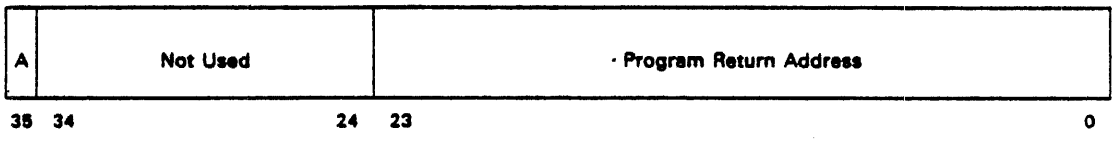
7.2.1. Program Status

Program status is stored for all interrupts, and includes the following information:

- Program Return Address – GRS Location 043 for Normal interrupts
  GRS Location 051 for Guard Modes
  GRS Location 041 for Immediate Storage Checks
  GRS Location 056 for IOU Error Interrupt
- Quantum Timer Value – GRS Location 050 for all interrupts.
- Designator Register Value – GRS Location 044 for Normal interrupts
  GRS Location 052 for Guard Modes
  GRS Location 042 for Immediate Storage Checks

GRS Location 057 for IOU Error Interrupt

A program return address is the address of the instruction following the last instruction that was fully executed; program control would normally be returned at this point for recoverable errors. The program return address stored in GRS locations 041, 043, 051, and 056 is in the following format:



The program return address value will vary depending on the operation being performed at the time of interrupt:

- If an incomplete block transfer, search instruction, or byte instruction is interrupted, the return address will be P.
- If a satisfied (the condition specified by the instruction exists) skip instruction is interrupted, the return address will be P+2.
- If a satisfied (the condition specified by the instruction exists) jump instruction is interrupted, the return address will be U.
- If an instruction other than the above is interrupted, the return address will be P+1.

The contents of the program address register is changed only by a jump instruction (including User Return) or interrupt. Instruction references following a jump instruction are made under the same addressing constraints that conditioned the operand address of the jump instruction that began the straight line instruction stream.

Bit position 35 of the first word of the 2-word program status packet contains a flag that identifies the correct program addressing mode. The two modes of program address generation include absolute (A=1), corresponding to D35=0 and D7=i=1, and relative (A=0), corresponding to D35=1, or D34, or i=0.

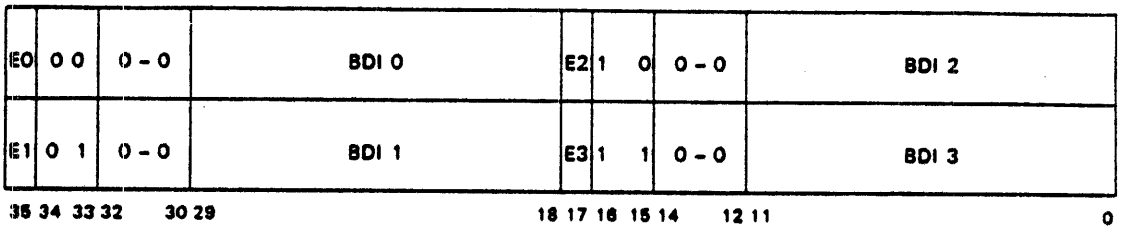
Bit positions 18 through 23 of the relative program address are zero unless absolute 24-bit indexing mode is selected. For straight line instruction sequencing, the relative program address is increased by one for each instruction that is executed or skipped. This increase is accomplished by twos complement addition with wraparound at 18 or 24 bits, depending on the value of the A-flag.

7.2.2. Addressing Status

Addressing status is not actually stored during the interrupt sequence. The information within this group is placed in GRS by the software either directly (load, store) or indirectly (Load Bank and Jump, LBJ; Load Addressing Environment, LAE) and is used from GRS by the CPU for addressing operations. Addressing status includes the following information:

- Executive bank descriptor table pointer.
- User bank descriptor table pointer.

- Bank descriptor specifications in the following format:



7.2.3. Interrupt Status

Interrupt status is information associated with a particular type of interrupt, and is stored only when its type of interrupt occurs. Immediate Storage Check status is stored in GRS location 054, Guard Mode status is stored in GRS location 053, all other CPU-generated interrupt status is stored in the Normal status location, GRS 055. Interrupt status is associated with the following types of interrupts:

- Immediate Storage Check interrupts
- Guard Mode interrupt
- Executive Request, Test and Set, and Invalid Instruction interrupts
- Delayed Storage Check interrupts
- Breakpoint and Emulation interrupts
- Power Check interrupt
- Addressing Exception interrupt
- Interprocessor interrupt

7.3. Interrupt Types

The CPU provides 20 interrupt priorities. The interrupt types are listed in Table 7-1.

7.3.1. Program Exception Interrupts

Invalid Instruction - This interrupt occurs when the CPU attempts to execute an instruction with an invalid function code. The operand address of the instruction (24 bits of U) is stored in GRS as interrupt status.

Guard Mode - This interrupt occurs in the following cases:

- When the privileged instruction, GRS protect, and interrupt lockout detect designator (D2) equals one, and the execution of a privileged instruction is attempted, or interrupt lockout period is exceeded.
- When the storage limit fails and either D7 is zero or the i-bit of the instruction word is zero.

- When any reference is made to an SIU that has its interface to the CPU disabled.
- When attempting to store in GRS locations other than those allowed for the user (40_8 through 100_8, and 120_8 through 177_8, when D2 is one).
- When attempting to write into a storage area specified by bank descriptor register (BDR0, 1, 2, or 3) for which the corresponding write protection designator bit (D13 through D16) is one.

See Figure 7-1 for the format of the Guard Mode interrupt status stored in GRS during the interrupt sequence.

Addressing Exception – This interrupt occurs in the following cases:

- E-bit violation – If the E-bit (bit 35) from Xa of an LBJ, Load I-Bank Base and Jump (LIJ) or Load D-Bank Base and Jump (LDJ) instruction is one and the EXEC bank descriptor table pointer enable designator (D19) is zero.
- Table length violation – If a bank descriptor index value from Xa of an LBJ, LIJ, or LDJ instruction is greater than the selected BDT pointer length value.
- Residency interrupt – If the R-flag of the new bank descriptor is one.
- Entry point violation – If the V-flag of the new bank descriptor is one.
- Use count overflow on LBJ, LIJ, or LDJ new bank descriptor.
- Use count underflow on LBJ, LIJ, or LDJ old bank descriptor.
- Use count decreased to zero and C-flag was one.

See Figure 7-2 for the format of the Addressing Exception interrupt stored in GRS during the interrupt sequence. The program return address stored for this interrupt is P+1 for E bit or table length violations, and the jump to address for all other violations.

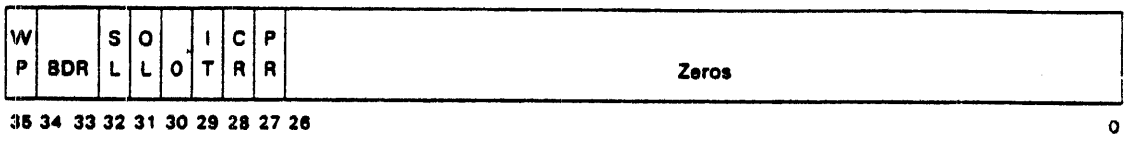
7.3.2. Arithmetic Exception Interrupts

An interrupt occurs in the following cases only if the arithmetic exception interrupt designator (D20) is one.

Characteristic Overflow – Occurs when the exponent value of a floating-point result is greater than $+127_{10}$ (single precision) or $+1023_{10}$ (double precision). When this condition is detected, the characteristic overflow designator (D22) is set to one.

Characteristic Underflow – Occurs when the exponent value of a floating-point result is less than -128_{10} (single precision) or -1024_{10} (double precision). When this condition is detected, the characteristic underflow designator (D21) is set to one.

Divide Check – Occurs when the magnitude of the quotient exceeds the range of the specified register. When this condition is detected, the divide check designator (D23) is set to one.



Bit 35 Write protection violation.

Bits 34–33 BDR number associated with write protection violation.

Bit 32 Storage limits violation.

Bit 31 Reference to disabled storage.

Bit 30 Is zero.

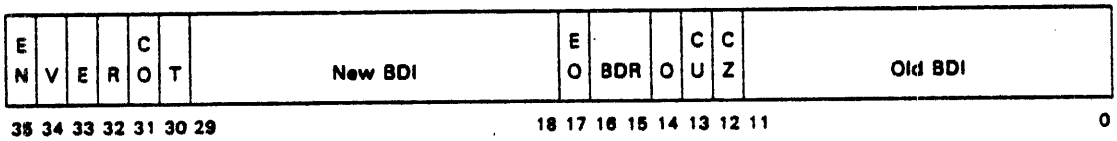
Bit 29 Interrupt lockout exceeded.

Bit 28 Control register violation.

Bit 27 Privileged instruction violation

Bits 26–0 Are zeros.

Figure 7–1. Format of Guard Mode Interrupt Status



Bit 35 New bank descriptor E–flag specification from Xa.

Bit 34 The V–flag indicates an entry point violation on the new bank descriptor.

Bit 33 The E–flag indicates an E–bit violation on the new bank descriptor.

Bit 32 The R–flag indicates the residency flag of the new bank descriptor was one.

Bit 31 The CO–flag indicates a use count overflow on the new bank descriptor.

Bit 30 The T–flag indicates a table length violation on the new bank descriptor.

Bits 29–18 New bank descriptor BDI specification from Xa.

Bit 17 Old bank descriptor E–flag specification from GRS.

Bits 16–15 The bank descriptor register specification from Xa.

Bit 14 Is zero.

Bit 13 The CU–flag indicates a use count underflow on the old bank descriptor.

Bit 12 The CZ–flag indicates the old bank descriptor use count was decreased from one to zero and the C–flag was one.

Bits 11–0 Old bank descriptor BDI specification from GRS.

NOTE:

This interrupt results only from the execution of an LBJ, LIJ, or LDJ instruction. The new bank descriptor specifications are contained in Xa before execution; the old bank descriptor specifications are contained in GRS locations 046 and 047, and are placed in Xa during execution .

Figure 7–2. Format of Addressing Exception Interrupt Status

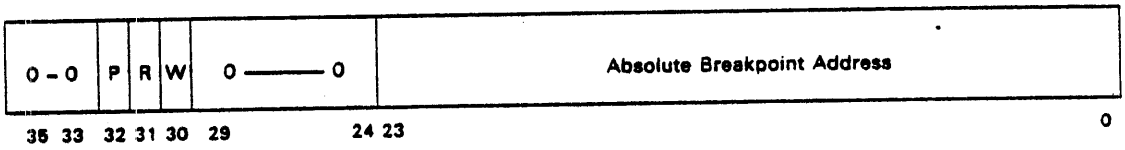
7.3.3. Program–Initiated Interrupts

Executive Request – This interrupt occurs as a result of executing an Executive Request (ER) instruction. This instruction allows a worker program to release control of the CPU to the Executive System. The operand address of the instruction (24 bits of U) is stored in GRS (055) as interrupt status. The CPU interrupts to fixed storage location MSR+222.

Test and Set – This interrupt occurs as a result of executing a Test and Set (TS) instruction if bit 30 of the operand is one, or as a result of executing a Test and Set Alternate (TSA) instruction. The operand address of the instruction (24 bits of U) is stored in GRS (055) as interrupt status. The CPU interrupts to fixed storage location MSR+224.

Breakpoint – This interrupt occurs when an equality comparison is made between the contents of the breakpoint register and an instruction or operand address. The breakpoint interrupt condition will be discarded if a higher priority internal interrupt occurs within the same instruction. See Figure 7–3 for the format of the Breakpoint interrupt status.

Jump History Stack – This interrupt occurs when the jump history stack is full, if the S–flag (bit 34) of the breakpoint register is one.



Bits 35–33 Are zeros.

Bit 32 The P–flag indicates an instruction address breakpoint.

Bit 31 The R–flag indicates an operand address breakpoint during a read operation.

Bit 30 The W–flag indicates an operand address breakpoint during a write operation.

Bit 29–24 Are zeros.

Bits 23–0 The absolute breakpoint address.

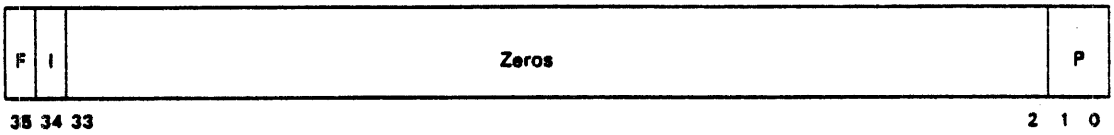
NOTE:

For both instruction address breakpoints (P–flag) and operand address breakpoints (R– or W–flags), the instruction is executed and the program return address is captured.

Figure 7–3. Format of Breakpoint Interrupt Status

7.3.4. Interprocessor Interrupt

This interrupt occurs when a CPU in a system executes an Initiate Interprocessor Interrupt (III) instruction. When a CPU detects a Function Programmable Read Only Memory (PROM) parity error or an Interrupt PROM parity error, an Interrupt Request is sent to: (1) itself if the error does not exist upon a second read of PROM data (the status word contains error history, bits 35 and 34, and the number of the requesting CPU), or (2) another CPU in the same application if the error exists upon a second read of PROM data. The interrupting CPU number can be determined from the status word stored in GRS address during the interrupt sequence. (See Figure 7–4 for format of the status word.)



Bit 35 The F-flag indicates Function PROM parity error detected.

Bit 34 The I-flag indicates Interrupt PROM parity error detected.

Bits 33 – 2 Are zeros.

Bits 1 – 0 Interrupting CPU Number

NOTE:

Bit 35 and/or 34 will be set only when a CPU sends a III to itself as a result of detecting PROM parity errors.

Figure 7–4. Format of Interprocessor Interrupt Status

7.3.5. Clock Interrupts

Quantum Timer – A Quantum Timer interrupt occurs when the quantum timer value reaches zero.

Real-Time Clock – This interrupt occurs when the contents of the lower 18 bits of the real-time clock (RTC) register (GRS address 100_8) is decreased through zero. The value of +0 is decremented to –1. The value contained in the RTC is decreased by one every 200 microseconds. The RTC oscillator is accurate to ±0.02 percent.

Dayclock – This interrupt request is made to all processors in the system once every 6.5536 seconds. Only one processor may honor each request. The dayclock value is increased by one every 200 microseconds. (See 8.2.2 for a description of the dayclock.)

7.3.6. Storage Check Interrupts

Storage Check interrupts are caused by conditions that are divided into two groups: immediate interrupt conditions, which are related to the current CPU storage reference, and delayed interrupt conditions, which may or may not be related to the current operation. Immediate interrupt conditions include check conditions that occur on transfers between the CPU and SIU. They may terminate the instruction and cannot be prevented by an interrupt or PAIJ instruction. Delayed interrupt conditions include internal SIU checks and check conditions that occur on transfers from the SIU to main storage units. These conditions do not affect the current instruction and may be deferred by an interrupt or PAIJ instruction.

7.3.6.1. Immediate Storage Checks

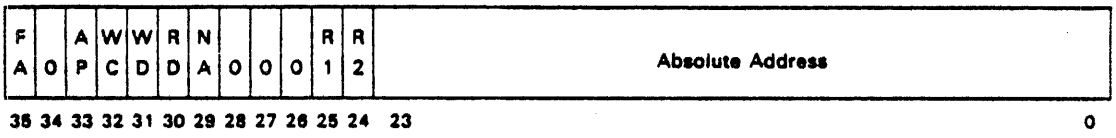
Immediate Storage Check interrupt status word is shown in Figure 7-5. An immediate check interrupt occurs:

- If the SIU detects a parity error on the address, controls, or write data from the CPU.
- If the CPU detects a parity error on the read data from the SIU.
- If the SIU detects an address that references a non-available storage location.

The Immediate Storage Check interrupt status word provides information to assist in performing software instruction retry (no retry is actually done by hardware). The retry information is provided only as a result of an Immediate Storage Check interrupt. On all Immediate Storage Checks the P value captured is P of the instruction having the error plus one, regardless of whether the error occurred on the instruction or on its operand(s). Two bits are used to define retry status, bit 25 defines whether the check occurred on an instruction fetch or an operand read/write. If bit 25 = 1, the check occurred on an instruction fetch, and retry can be done by decrementing P by one and returning to that point without further analysis. If bit 25 = 0, the check occurred on an operand read/write and further analysis is required.

In general, bit 25 = 0 analysis must determine if the instruction on whose operand cycle the check occurred had the indirect bit or incrementation bit set. If indirection was specified, retry will not be successful since one or more cycles of indirection may have occurred. If h was set, incrementation will have occurred, and will occur again if retry is attempted.

Bit 24 defines the state of execution at the point of immediate check on an operand cycle. If bit 24 = 0, the initial values are intact (except as described above for indirect and increment) and retry can be attempted. If bit 24 = 1, execution will have proceeded to a state where retry is not possible.



Bit 35 — Indicates that the Immediate Storage Check interrupt occurred while fetching an instruction from any interrupt entrance fixed address.

Bit 34 — Is zero.

Bit 33 — Indicates an address parity check.

Bit 32 — Indicates a control parity check.

Bit 31 — Indicates a write data parity check.

Bit 30 — Indicates a parity check on read data received by the CPU.

Bit 29 — Specifies that the addressed storage is not available.

Bits 28–26 — Are zero.

Bit 25 (Retry 1) — = 0 Immediate check on operand.

= 1 Immediate check on instruction.

Bit 24 (Retry 2) — = 0 Retry possible on operand check.

= 1 Retry not possible

Bits 23–0 * — Absolute address associated with check condition.

* When Bit 35 is a 1, the absolute address does not contain the MSR value, but is the fixed address relative to MSR.

Figure 7–5. Format of Immediate Storage Check Interrupt Status

7.3.6.2. Delayed Storage Check Interrupts

Delayed checks are associated with internal SIU, SIU/MSU or internal main storage unit (MSU) errors which are reported via a STORE CHECK INTERRUPT STATUS REQUEST (SCISR) signal. Delayed storage checks are a class of interrupts which results from the detection of fault conditions as follows:

1. During internal SIU operation.
2. During data transfer operations between an SIU and an MSU.
3. During internal MSU operation.

The SIU uses three SCISR formats to transfer error information to the processing and software level of the system.

7.3.6.2.1. Internal SIU Check

The internal SIU check format is shown in Figure 7-6, and Table 7-2 describes the bits.

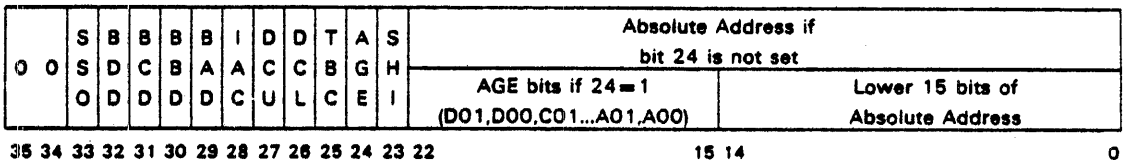


Figure 7-6. Internal SIU Check Format

Table 7-2. Internal SIU Check

Bits	Function	Description
35-34	Format Indicator	Specifies format for the status word and equals 00.
33 (SSO)	Storage Check Interrupt Status Request Stack Overflow	Common to all SCISR formats and indicates a SCISR stack overflow condition. Bit will be set when the SIU attempts to generate a fifth SCISR interrupt before the CPU acknowledges the first SCISR interrupt. Will stay set until the CPU accepts the first interrupt indicating to the software that one more SCISR status words have been lost.
32 (BDD) 31 (BCD) 30 (BBD) 29 (BAD)	Block D Degraded Block C Degraded Block B Degraded Block A Degraded	Bits 32-29 set to 1 indicate which particular block or blocks have been degraded as a result of errors detected by the SIU.
28 (IAC)	Invalidate Address Check	Set conditions indicate the SIU has detected an address parity check on the invalidate interface from the parallel SIU. So, the receiving SIU invalidates all resident data, since it cannot determine which address has been modified in the parallel SIU. This interface is only active in a 4x4 configuration where two SIUs are sharing and communicating with the same MSU. Any data altered in one SIU must be marked invalid in the other SIU if it is resident.
27 (DCU) 26 (DCL)	Data Check Upper Half Word Data Check Lower Half Word	Either bit set to 1 indicates the SIU has detected a parity check on data resident in the SIU and has degraded the blocks (bits 32-29) in which the data was stored. The bad data is passed to the requester and an Immediate Check indicating a read data error will be generated by that component also.

Table 7-2. Internal SIU Check (continued)

Bits	Function	Description
25 (TBC)	Tag Buffer Check	Set condition indicates that the SIU has detected on the requested set address either: bad parity on a block address that has been marked valid, or an illegal combination of the three control bits (Par, Invalid, Degraded) for a block address. Either error will also set the associated block degrade bits (32–29) of this SCISR, indicating the block on which the error occurred. This error prevents the SIU from getting a match on the block in error. Therefore, requests to other blocks proceed as normal; or if to the bad block, it proceeds as a "miss" operation.
24 (AGE)	Duplicate Age Check	Set condition indicates the SIU has detected one or more duplicate ages at the requested set address. All four block degrade bits (32–29) will set unless one or more of the blocks has been previously degraded. The failing age bits are bits 22–15 of the SCISR. The request will proceed as a normal read or write. All future requests to this set address will generate misses since all four blocks have been degraded.
23 (SHI)	SIU Half Indicator	Indicates which SIU half (0 – SIU lower, 1 – SIU upper) caused the error condition.
22–00 22–15 14–00	Absolute Address if bit 24 is not set AGE bits if bit 24 is set Absolute Address, lower 15 bits	Requester address at the time the SIU detected the error if bit 24 (AGE) is not set. If bit 24 is set, the failing ages are captured in bits 22–15 and the remainder of the address bits reflect the requester address.

7.3.6.2.2. SIU/MSU Interface Check

The SIU/MSU interface check format is shown in Figure 7–7, and Table 7–3 describes the bits.

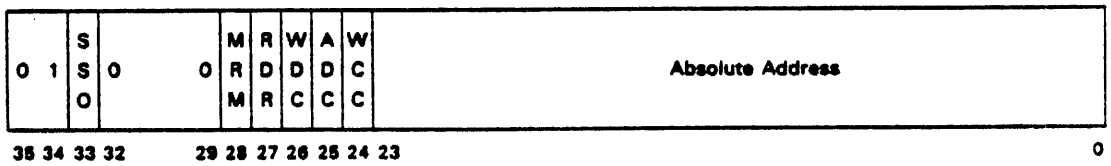


Figure 7-7. SIU/MSU Interface Check Format

Table 7-3. SIU/MSU Interface Check

Bits	Function	Description
35-34	Format Indicator	Specifies format for the status word and equals 01.
33 (SSO)	Storage Check Interrupt Status Request Stack Overflow	Common to all SCISR formats and indicates a SCISR stack overflow condition. Bit will be set when the SIU attempts to generate a fifth SCISR interrupt before the CPU acknowledges the first SCISR interrupt. Will stay set until the CPU accepts the first interrupt indicating to the software that one or more SCISR status words have been lost.
32-29	Not Used	Always 0
28 (MRM)	Maintenance Read Miss	Set condition indicates the MSU executed a read miss.
27 (RDR)	Read Request	Set condition indicates that the error causing the SCISR to be generated occurred during a read operation.
26 (WDC)	Write Data Check	Set condition indicates that the MSU detected a single or multiple bit error on the data and error correction code (ECC) bits the SIU was attempting to write. The MSU writes the Special ECC Syndrome at the addressed location denoting that address as containing known corrupted data. Any attempt to read that address will result in an immediate ADDRESS NOT AVAILABLE signal to the requester. The only way of clearing the Special ECC Syndrome is by successfully executing a full word write with good ECC to the particular address.
25 (ADC)	Address Check	Set condition indicates that the MSU detected a parity error on the address field sent by the SIU. In the case of a read request, the SIU will present an immediate ADDRESS NOT AVAILABLE signal to the requester and notify the software via this SCISR of the error. In the case of a write request, the MSU will fail to respond to a further request to protect against the undetected corruption of data. This condition will cause the system to freeze if more requests are pending for that MSU. This necessitates manually downing these units and rebooting.
24 (WCC)	Write Control Check	Set condition indicates that the MSU detected a parity error on the write control field sent by the SIU. The MSU writes the Special ECC Syndrome at the address location denoting it as containing known corrupted data. If the error occurs on a read, the requester will receive an immediate ADDRESS NOT AVAILABLE signal and the software will be notified

Table 7–3. SIU/MSU Interface Check (continued)

Bits	Function	Description
		via this SCISR. If the error occurred on a write, only the SCISR will be generated.
	Absolute Address	Address on which the error occurred.

7.3.6.2.3. SIU/MSU Read or Partial Write ECC Check

The SIU/MSU read or partial write ECC check format is in Figure 7–8, and Table 7–4 describes the bit functions.

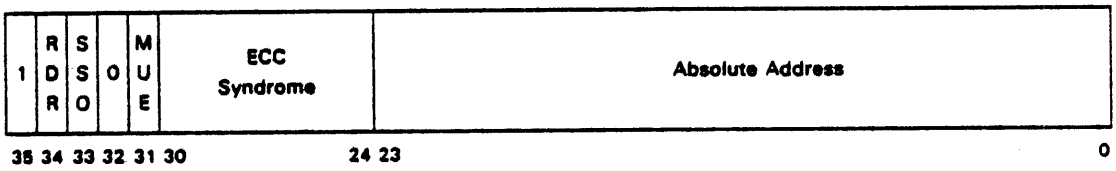


Figure 7–8. SIU/MSU Read or Partial Write ECC Check Format

Table 7–4. SIU/MSU Read or Partial Write ECC Check

Bits	Function	Description
35	Format Indicator	Set condition indicates the SIU/MSU read or partial write ECC check format.
34 (RDR)	Read Request	Set condition indicates that the information relates to errors encountered on a read request. Clear condition relates to errors on partial writes. On each write request, the MSU presents the word to be altered and the associated ECC code to the SIU. The SIU checks the word for single and multiple bit errors. If the SIU detects a multiple bit uncorrectable error (MUE), a SCISR is generated. At the same time the MSU will either correct a single bit error or detect an MUE. When it corrects the single bit error, the partial write is executed and the correct data word and ECC are stored back. If an MUE is detected, the MSU aborts the partial write and stores the special ECC code at the addressed location, flagging that location as having known corrupted data. Any read of that location will result in an immediate ADDRESS NOT AVAILABLE signal to the requester.

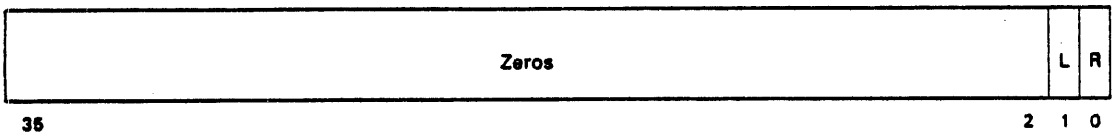
Table 7–4. SIU/MSU Read or Partial Write ECC Check (continued)

Bits	Function	Description
33 (SSO)	Storage Check Interrupt Status Request Stack Overflow	Common to all SCISR formats and indicates a SCISR stack overflow condition. Bit will set when the SIU attempts to generate a fifth SCISR interrupt before the CPU acknowledges the first SCISR interrupt. Will stay set until the CPU accepts the first interrupt indicating to the software that one or more SCISR status words have been lost.
32	Not Used	Always 0
31 (MUE)	Multiple Bit Uncorrectable Error	Set condition indicates that the ECC syndrome is a multiple bit uncorrectable error. Any MUE encountered by the SIU on a read miss is reported via this SCISR, and if the MUE occurs on the word requested by the requester, an immediate ADDRESS NOT AVAILABLE signal will be returned. If the request is for a word other than the word in error, the requester will be given the correct data and released. If the MUE occurs on a partial write, only the SCISR will be generated.
30–24	ECC Syndrome	Bits 30–24 contain the ECC syndrome detected by the SIU.
23–00	Absolute Address	Address on which the error occurred.

7.3.7. Power Check Interrupt

A Power Check occurs when a system or CPU power check condition is detected due to a power service interruption or failure. Power Check interrupt status is stored in GRS during the interrupt sequence.

The format of the Power Check interrupt status word is shown in Figure 7–9 and provides for early warning power loss detection and power restoration indication.



Bits 35–2 Are zeros.

Bit 1 The L-bit indicates that a power loss condition has been detected and CPU power will drop after a 100ms grace period, unless within that period the power loss condition is removed (power restoration has occurred).

Bit 0 The R-bit indicates that power has been restored following a power loss indication, but within the 100ms grace period. This interrupt condition is not locked out when the CPU is stopped following a Power Check power loss interrupt.

NOTE:

If both L and R are one, it means that another power loss that had not yet been reported occurred after a recovery.

Figure 7-9. Power Check Interrupt Status Word

7.3.8. Byte Status Code

A 7-bit status code is stored in the BB2 field of Staging Register 3 (SR3), either upon completion of the instruction or upon detection of an error condition which prevents completion during the execution of the following instructions:

- Byte-to-Binary Single Integer Convert (33, 10)
- Byte-to-Binary Double Integer Convert (33, 11)
- Byte-to-Single Floating Convert (33, 14)
- Byte-to-Double Floating Convert (33, 15)
- Byte Add (37, 06)
- Byte Add Negative (37, 07)

Successful completion of an instruction results in the storing of an all zero status word except for a decimal overflow for the byte Add and Add Negative instructions (37, 06 and 37, 07) or a missing mantissa field for the byte-to-floating instructions (33, 14 and 33, 15).

7.3.9. Multiprocessor Interrupt Synchronization

All external interrupt requests, for example those generated by IOUs, are presented to each CPU in the cluster where the interrupt was generated. Therefore, an interlocked synchronization mechanism is provided to assure that only one CPU actually accepts the interrupt request. This interlock is provided as part of the interprocessor interrupt network and consists of four lines: one incoming select line, one incoming select acknowledge, one outgoing select line, and one outgoing select acknowledge.

The outgoing lines of one CPU are connected to the incoming select lines of another CPU, forming a ring. A CPU is allowed to honor an interrupt request between the time the incoming select is

acknowledged and the time the outgoing select is propagated. The time available for honoring interrupt requests is sufficient to allow examination of requests but not so long that system interrupt response time is impaired. An acceptable value is between 800 and 2000 nanoseconds per CPU. If a decision is made by the CPU to honor an interrupt request, the propagation of the outgoing select is delayed until the interrupt request is acknowledged and the interrupt request is dropped.

A CPU will retain the interrupt interlock by not passing the interrupt select in a unit processor system; also in a multiprocessor system where the other CPU is offline or powered down, and if, during initial load, that CPU is selected for taking the initial load interrupt. The system transition unit will provide the information to each unit. Each cluster in a multiprocessor system has its own independent interrupt synchronization mechanism.

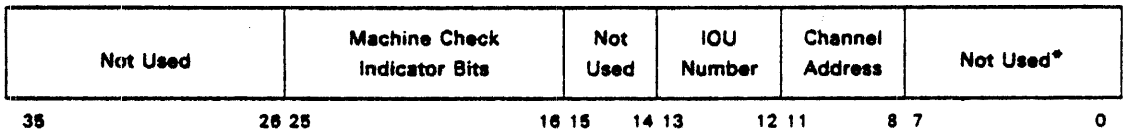
7.4. Input/Output Interrupts

There are three classes of I/O interrupts: normal interrupts, tabled interrupts, and machine check interrupts. Status conditions for the status table subchannel, all shared subchannels, and nonshared subchannels on a block multiplexer channel are reported by the normal interrupt mechanism. Status conditions for nonshared subchannels on a byte multiplexer channel or word channel are reported by the tabled interrupt mechanism. IOU or channel status conditions are reported by the Machine Check interrupt mechanism. IOU or channel status is any status not associated with a particular subchannel or device.

7.4.1. Machine Check Interrupts

If the control module or a channel detects status not associated with a particular device or subchannel, a Machine Check interrupt request is generated. When the Machine Check interrupt is acknowledged, an Interrupt Address Word (IAW) is stored in the fixed IAW address of the CPU that acknowledged the interrupt. Bits 8–13 of the IAW specify the channel module number and IOU number, respectively, associated with the Machine Check interrupt. Bits 16–25 of the IAW are master bitted to indicate the condition or conditions that caused the Machine Check interrupt to be generated. A description of the Machine Check IAW bit positions is is given in Table 7–5.

Machine Check IAW



* Bit positions 7–0, Device Address field, valid only when bit 21 is set to 1.

Table 7-5. Machine Check IAW Bit Description

Bit Position	Function	Description
35 - 26		Not used
25 - 16	Machine Check Indicator Bits	
25 - 24		Not used
23		A storage error occured when the IOU control module attempted to read the first word of the CAW.
22		A storage error occurred when the IOU control module attempted to read the second word of the CAW during a Load Channel Register operation.
21		A storage error occurred when the channel module attempted to write a control word for one of the 128 nonshared subchannels into storage.
20		A storage error occurred when the channel was attempting to write the preceding Interrupt Address Word.
19		A storage error occurred when the channel attempted to write a Channel Status Word for a Tabled Interrupt.
18		A storage error occurred when the channel attempted to write a Channel Status Word for a Nontabled Interrupt or an I/O instruction.
17		A storage error occurred when the channel attempted to read the second word of the channel address word (CAW) during a Load Channel Register operation.
16*		An interface control signal error or device address parity error prevented subchannel identification during a control-unit-initiated selection sequence. This bit is not used on a word channel.
15 - 14		Not used
13 - 12	IOU Number	Bit 13 12 0 0 = IOU 0 0 1 = IOU 1 1 0 = IOU 2 1 1 = IOU 3

Table 7-5. Machine Check IAW Bit Description (continued)

Bit Position	Function	Description
11 - 8	Channel Address	The IOU can contain up to 8 channel modules. Only bits 10-8 of this 4-bit field are used to designate one of eight modules. If bit 11 is ever set to a 1, the IOU has malfunctioned but it will not impact the system. Therefore, bit 11 can be ignored without compromising the system in any way.
7 - 0**	Device Address (only if bit 21 is set to 1)	Not used unless bit 21 is set to 1.

* If bit 16 of the Machine Check IAW is set, each device on the channel module must be label checked because the interrupt could be related to a device reporting attention status for a disk or tape changes.

** If bit 21 of the Machine Check IAW is set, the subchannel identified by bits 7-0 of the IAW must be halted by a Halt Device (HDV) instruction.

7.4.2. Normal Interrupts

If the status table subchannel, any shared subchannel, or a block multiplexer nonshared subchannel detects status conditions, a Normal interrupt request is generated. When the Normal interrupt is acknowledged, an IAW and Channel Status Word (CSW) are stored in the fixed IAW and CSW addresses of the CPU that acknowledged the interrupt. Bits 25-35 of the IAW are meaningless for a Normal interrupt. If bit 24 of the IAW is set, the interrupt is for the status table subchannel in the IOU and channel specified by bits 8-13 of the IAW. The device address field (bits 0-7) is not interpreted. The associated CSW contains status from the status table subchannel. If bit 24 of the IAW is not set, bits 8-13 contain the IOU and channel address associated with the interrupt. On a byte or block multiplexer channel, the device address field (bits 0-7) specifies the device. On a word channel, the device address field specifies only the subchannel. The associated CSW contains status for the device or subchannel specified by the device address field of the IAW. A description of the Normal Interrupt IAW and CSW bit positions is given in Tables 7-6 and 7-7, respectively.

Normal Interrupt IAW

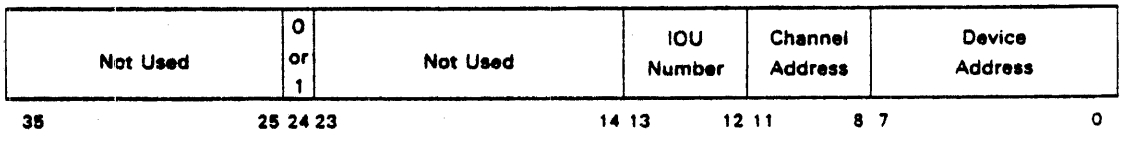


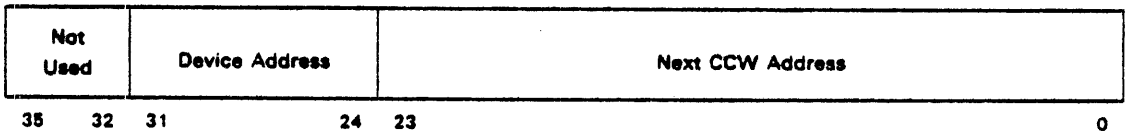
Table 7-6. Normal Interrupt IAW Bit Description

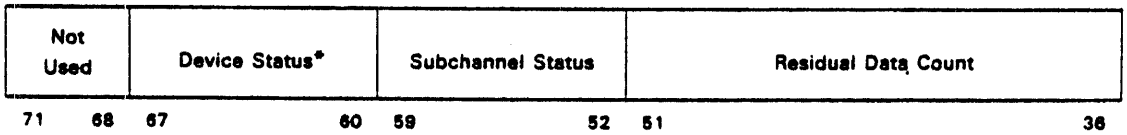
Bit Position	Function	Description
35 - 25		Not used
24		0 - Indicates a Normal Interrupt IAW. 1 - Indicates an interrupt for the status table subchannel. (See 7.4.3).
23 - 14		Not used
13 - 12	IOU Number	Bit 13 12 0 0 = IOU 0 0 1 = IOU 1 1 0 = IOU 2 1 1 = IOU 3
11 - 8	Channel Address	The IOU can contain up to 8 channel modules. Only bits 10-8 of this 4-bit field are used to designate one of eight modules. If bit 11 is ever set to a 1, the IOU has malfunctioned, but it will not impact system operation. Therefore, bit 11 can be ignored without compromising the system in any way.
7 - 0	Device Address	The field should be interpreted as follows: ■ If bit 24 of the IAW is equal to 1, the interrupt is for the Status Table Subchannel for the channel module indicated in this IAW and this field should be ignored. ■ If bit 24 of the IAW is zero, the interrupt is for a byte or block multiplexer channel or an internally specified index (ISI) word channel interface. ■ On byte or block multiplexer channels, bits 7-0 are used to indicate Device Address. ■ For ISI Word Channel interfaces, the Device Address field is broken down as follows: - Bits 7-4 indicate ISI interface (channel) presenting the interrupt. The valid values for this field are: BIT 7 6 5 4 Interface (Channel) A - 1 0 0 0

Table 7–6. Normal Interrupt IAW Bit Description (continued)

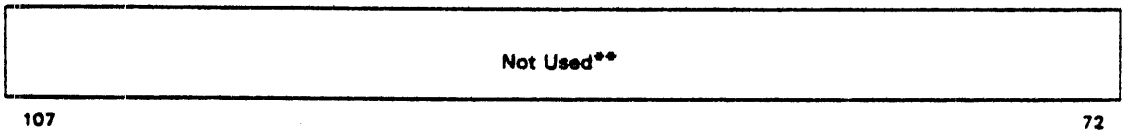
Bit Position	Function	Description
		Interface (Channel) B – 1 0 1 0 Interface (Channel) C – 1 1 0 0 Interface (Channel) D – 1 1 1 0 Bit 7 is always 1 and bit 4 is always 0. Bits 5 and 6 indicate the binary channel number (0–3). If any other than the above valid values are presented, they should be ignored since they indicate an IOU malfunction which will not affect operation in any way. – Bits 3–0 are meaningless for Word Channel Module interrupts.

Normal Interrupt CSW





* Table 7–7 explains differences in the use of this field.



** External Interrupt status word if bit 67 is set.

Table 7-7. Normal Interrupt CSW Bit Description

Bit Position	Function	Description
107 - 72		Bit positions 107–72 contain the External Interrupt status for word channel operation if bit 67 is set to 1. If bit 67 is 0 and the channel is a word interface channel, there is no External Interrupt status and the reason for the interrupt is contained in the Subchannel Status field. For a byte or block multiplexer channel, bit positions 107–72 are not used.
71 - 68		Not used
67 - 60	Device Status	Used for byte or block multiplexer channels only, with exceptions noted.

	Byte or Block Multiplexer Channel	Word Channel
67	Attention	Attention
	NOTE: *On word channels, if bit 67 equals 1, bits 107–72 will indicate an External Interrupt status word.*	
66	Status Modifier	Not Used
65	Control Unit End	Not Used
64	Busy	Not Used
63	Channel End	Not Used
62	Device End	Not Used
61	Unit Check	Not Used
60	Unit Exception	Not Used

NOTES:

1. *If Channel Control Check (CSW bit 54) or Interface Control Check (CSW bit 53) are set to one, the Device Status field is not valid; i.e., there may have been an error on reporting of the status by the CU or storing of the status by the IOU.*
2. *If there is a Subchannel Status, other than those bits in note 1, both fields, Subchannel Status and Device Status, are valid. On word channel modules the Attention bit (bit 67) always indicates whether bits 107–72 have valid EI status, except in the cases listed in note 1.*

Table 7-7. Normal Interrupt CSW Bit Description (continued)

Bit Position	*Function*	*Description*
		3. If the interrupt is for the Status Table Subchannel, the Device Status field is meaningless. *4. If the Device Status field and Subchannel Status field are both zero, the status is treated as a device error.*
59 - 52	Subchannel Status	
59	Program Controlled Interrupt	Indicates that a Program Controlled Interrupt flag was encountered by the channel module when processing a channel program.
58	Monitor/Incorrect Length	Word Channel Indicates that a channel program has been completed. *NOTE:* *In CCWs that have both chaining (command and data) bits set and monitor set, the monitor will be ignored and no interrupt will occur.* Byte and Block Multiplexer Channel This bit indicates that the number of bytes specifed in the CCW for the I/O operation on the subchannel indicated was not equal to the number of bytes requested or offered by the device. Detection of an incorrect length condition causes the operation to be terminated and an interrupt to be generated. The DC, CC, SLI, and TS CCW flags affect the incorrect length indication.
57	Program Check	This bit indicates that an error was detected by the channel, which indicates one of the following conditions: ■ The CAW first CCW address did not specify a double word boundary. ■ The status table CAW first STCW address did not specify a double word boundary. ■ A CCW or STCW contained an invalid command for, and operation other than, a data chain. ■ The CCW or STCW data count equalled zero and the command was neither a Transfer in Channel or a Store Subchannel Status.

Table 7-7. Normal Interrupt CSW Bit Description (continued)

Bit Position	Function	Description
		■ The Transfer in Channel command was specified in successive channel command words (CCWs) or status table control words (STCWs). ■ For an Start I/O Fast Release (SIOF) or STCW instruction, the channel attempted to read the second word of the CAW from a location outside of available storage. ■ Truncated search record lengths were specified on mixed word and byte boundries. ■ An externally specified index (ESI) word channel CCW contained the Forced FF command with a data count not equal to one. ■ The CCW or STCW address specified by a Transfer in Channel (TIC) command was not on a double word boundary. ■ The Store Subchannel Status command data address field did not specify a double word boundary. ■ The Truncated Search (TS) flag was specified in a byte multiplexer CCW. ■ Any byte multiplexer CCW or a block multiplexer command CCW did not specify only one format. ■ A byte multiplexer CCW specified Format C. ■ The ESI word channel status table CCW data address field did not specify a quadruple word boundary. ■ The ESI word channel STCW data count field was not a multiple of four. ■ The channel attempted to read a CCW or STCW from a location outside of available storage. ■ A hard channel control word (HCCW) or hard status table control word (HSTCW) data address specified a location outside the available storage. The HSTCW data count field was decreased to zero and data chaining was not indicated. *NOTE:*

Table 7-7. Normal Interrupt CSW Bit Description (continued)

Bit Position	*Function*	*Description*
		Address out-of-range conditions reported by this bit may be caused by hardware failures.
56		Not used
55	Channel Data Check	This bit indicates that an error has been detected during the transfer of data from the device to the channel, from the channel to storage, or from storage to channel.
54	Channel Control Check	This bit indicates that a hardware error was detected by the channel when attempting to read or write a control word from storage. The hardware error was detected during one of the following operations: ■ The channel was attempting to read the second word of the CAW for an SIOF instruction. ■ The channel was attempting to read a CCW or STCW. ■ The channel was attempting to store a status word for the store subchannel status command. ■ The channel was attempting to read from main storage a control word for one of the 128 nonshared subchannels. *NOTE:* *When this bit is set, the Device Status field is ignored (not used)*
53	Interface Control Check	This bit indicates that a hardware error in the channel to device interface was detected by the channel. The hardware error was detected during one of the following operations: ■ A byte or block multiplexer channel detected a device interface parity error during the transfer of device status. ■ A word channel detected a device interface parity error during the transfer of an External Interrupt status word. ■ A byte peripheral device responded with an address other than the address specified by the channel during a channel-initiated selection sequence. ■ A byte peripheral device became nonoperational during command chaining.

Table 7-7. Normal Interrupt CSW Bit Description (continued)

Bit Position	Function	Description
		■ A byte peripheral interface control signal sequence error was detected.
52	SIOF Device Check	This bit indicates that a device that was to be initiated by a previously issued SIOF instruction is not operational (not installed or offline).
51 - 36	Residual Data Count	A count of the number of words or bytes specified in the CCW which were not transferred by the channel.
35 - 32		Not used
31 - 24	Device Address	The Device Address field should be interpreted as follows: ■ If bit 24 of the IAW is equal to 1, the interrupt is for the Status Table Subchannel for the channel module indicated in this IAW and this field should be ignored. ■ If bit 24 of the IAW is zero, the interrupt is for a byte or block multiplexer channel or an ISI word channel interface. ■ On byte or block multiplexer channels, bits 31-24 are used to indicate Device Address. ■ For ISI Word Channel interfaces, the Device Address field is defined as follows: Bits 31-28 indicate ISI interface (channel) presenting the interrupt. The valid values for this field are: BIT 31 30 29 28 Interface (Channel) A - 1 0 0 0 Interface (Channel) B - 1 0 1 0 Interface (Channel) C - 1 1 0 0 Interface (Channel) D - 1 1 1 0 Bit 31 is always 1, and bit 28 is always 0. Bits 30-29 indicate the binary channel number (0-3). If any other than the above valid values are presented, they should be ignored since they indicate an IOU malfunction which will not affect operation in any way. Bits 27-24 are meaningless for Word Channel Module interrupts.

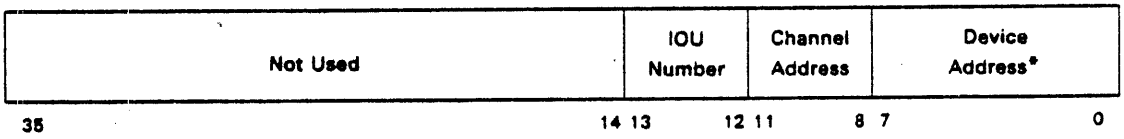
Table 7-7. Normal Interrupt CSW Bit Description (continued)

Bit Position	Function	Description
23 - 0	Next CCW Address	This field contains the 24-bit absolute address of the next CCW.

7.4.3. Tabled Interrupts

Status for a nonshared subchannel on a byte multiplexer channel or word channel (communications status) is stored in a status table under the control of the status table subchannel. There is one status table subchannel per channel. If the status table subchannel is not active when the nonshared subchannel status conditions are detected, the status is lost and the communications subchannel is returned to the available state. If the status table subchannel is active, a tabled status word (TSW) containing the communications subchannel status is stored at the address specified by the status table and a Tabled interrupt request is generated. If another communications subchannel detects status conditions before the Tabled interrupt is acknowledged, its status is stored in a TSW at the address specified by the status table. The Tabled interrupt request is reset. Thus, a single Tabled interrupt request may report several entries (TSWs) in the status table. When the Tabled interrupt is acknowledged, an IAW and CSW are stored in the fixed IAW and CSW addresses of the processor that acknowledged the interrupt. Bits 14-35 of the IAW are meaningless for a Tabled interrupt. Bits 8-13 specify the IOU and channel. Bits 0-7 specify the device address of the last entry in the status table. The associated CSW contains the status of the status table. The subchannel status field of the CSW is cleared. A description of the Tabled Interrupt IAW and CSW bit positions is given in Tables 7-8 and 7-9, respectively.

Tabled Interrupt IAW



* Device address is address of device that most recently made an entry in the status table.

Table 7-8. Tabled Interrupt IAW Bit Description

Bit Position	Function	Description
35 - 14		Not used.
13 - 12	IOU Number	Bit 13 12 0 0 = IOU 0 0 1 = IOU 1 1 0 = IOU 2 1 1 = IOU 3
11 - 8	Channel Address	The IOU can contain up to 8 channel modules. Only bits 10–8 of this 4–bit field are used to designate one of eight modules. If bit 11 is ever set to a 1, the IOU has malfunctioned, but it will not impact system operation. Therefore, bit 11 can be ignored without compromising the system in any way.
7 - 0	Device Address	The Device Address field should be interpreted as follows: ■ If bit 24 of the IAW is equal to 1, the interrupt is for the Status Table Subchannel for the channel module indicated in this IAW and this field should be ignored. ■ If bit 24 of the IAW is zero, the interrupt is for a byte or block multiplexer channel or an ISI word channel interface. ■ On byte or block multiplexer channels bits 7–0 are used to indicate Device Address. ■ For ISI Word Channel interfaces, the Device Address field is broken down as follows: – Bits 7–4 indicate ISI interface (channel) presenting the interrupt. The valid values for this field are: BIT 7 6 5 4 Interface (Channel) A - 1 0 0 0 Interface (Channel) B - 1 0 1 0 Interface (Channel) C - 1 1 0 0 Interface (Channel) D - 1 1 1 0 Bit 7 is always 1, and bit 4 is always 0. Bits 5 and 6 indicate the binary channel number (0–3). If any other than the above valid values are presented, they are to be

Table 7-8. Tabled Interrupt IAW Bit Description (continued)

Bit Position	Function	Description
		ignored. – Bits 3–0 are meaningless for Word Channel module interrupts.

Tabled Interrupt CSW

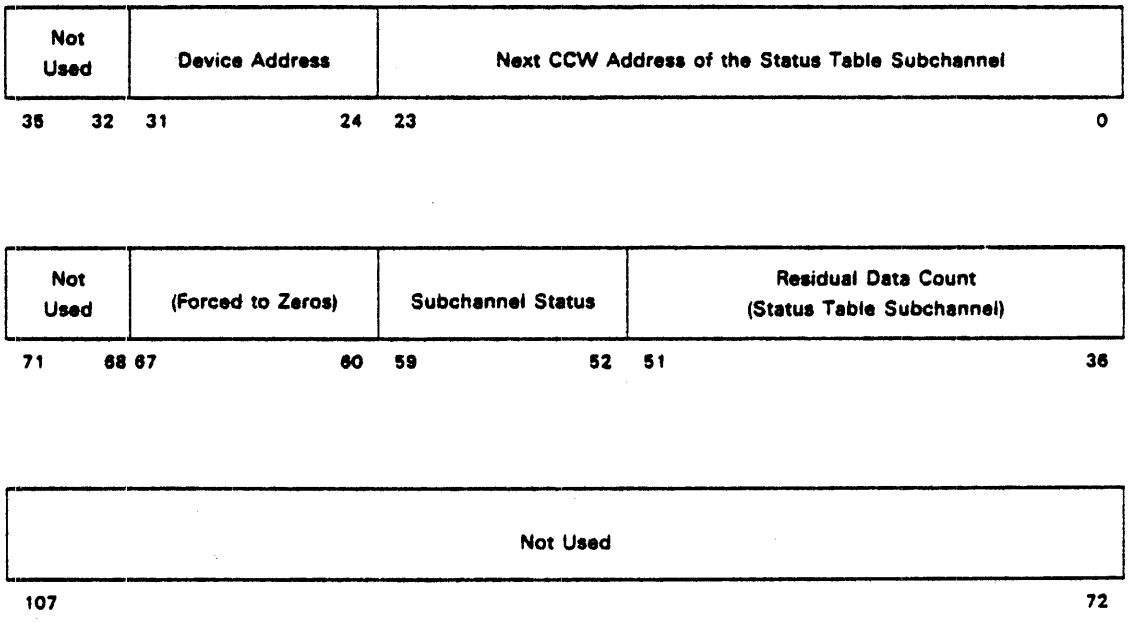


Table 7-9. Tabled Interrupt CSW Bit Description

Bit Position	Function	Description
107 - 72		Not used
71 - 68		Not used
67 - 60	(Forced to Zeros)	In normal CSWs, bits 67–60 comprise the Device Status field. For tabled interrupt status words, however, this field is forced to zeros by the hardware. It is therefore meaningless and should be ignored.
59 - 52	Subchannel Status	Contains a subchannel–generated status byte. (See Table 7–7 for a description of each of the Subchannel Status bits 59–52.)
51 - 36	Residual Data Count	Contains the value of the data count existing at the time status information was stored.
35 - 32		Not used
31 - 24	Device Address	Contains the device address associated with the status information. (See Table 7–7 for a description of the Device Address bits 31–24.)
23 - 0	Next CCW Address	Contains the value of the next CCW address existing at the time the status information was stored, if the CCW address check subchannel status bit is set to zero. If the CCW address check bit is set to 1, the value of the next CCW address plus two is stored.

7.5. Interrupt Errors

Interrupt errors are errors detected on interrupt status information. The type of errors detected are such that it is not possible for the system to process the data which was being reported when the error occurred. Errors can be detected on processor or Input/Output interrupts and when detected force the processor to specific fixed address locations.

7.5.1. Processor Interrupt Errors

Processor interrupt errors occur when the interrupt type information is lost during the execution of a processor interrupt. The condition is that the processor has been interrupted but the information indicating the type of interrupt, guard mode, etc., has been lost. This condition forces the processor to fixed address 200_8 (Hardware Default). This is an unrecoverable condition for the system and it is brought to a stop.

7.5.2. Input/Output Interrupt Errors

Input/Output (I/O) interrupt errors indicate that an error has been detected which was associated with an I/O interrupt or status for an I/O instruction. On the detection of this type of error, the I/O activates interrupt error interface lines to the processors. The interrupt error interface lines being activated cause the processors to go to fixed addresses 202_8 (Even Number I/O Interrupt Error) or 203_8 (Odd Number I/O Interrupt Error), depending on which IOU in the cluster had an error.

7.5.2.1. Cause of Even/Odd I/O Interrupt Errors

If the IOU attempts to store an IAW or CSW and that word is rejected by the SIU because of bad parity, then the IOU raises an interface signal to tell the CPU that one of the words does not contain good data. The SIU will not store the bad data with good parity.

As the IOU stores CSWs during both an I/O interrupt sequence and during a Condition Code = 1 situation when an I/O instruction is being executed, the CPU must detect and act upon the interface signal during two sequences which both have I/O interrupts locked out. For this reason, the Even/Odd I/O interrupt is penetrating. As penetrating interrupts are presented to both CPUs connected to the IOU, it is a possibility that the CPU not directly involved in the error will also pick up and act upon the interface signal because it happens to be in one of the sequences described.

7.5.2.2. Operation of Even/Odd I/O Interrupts

Captured P and D-bits for the Even/Odd I/O interrupt are saved in GRS locations 56 and 57. The P captured will depend upon the sequence being executed at the time that the interface signal is detected. The cases are as follows:

1. During another Interrupt

   Captured P = The fixed address for the interrupt.

2. During An I/O Instruction

   Captured P = The I/O instruction +1 if the condition code is nonzero.

   Captured P = The I/O instruction +2 if the condition code is zero.

NOTE:

In case 1, the original interrupt will already have saved P and D bits in its own GRS locations, so it will be possible for software to subsequently return control to the original interrupted activity.

7.5.2.3. Software Action on Even/Odd I/O Interrupts

Software action will depend upon the contents of GRS locations 56 and 57 to determine which sequence detected the error and to direct the subsequent recovery action. The cases are as follows:

1. P=204/205/206

   The error occurred during an I/O interrupt sequence, probably the one which caused the CSWs to be stored. As this data is suspect, control cannot be given to the I/O interrupt handlers. Details of the error will be logged and control will be given to the original interrupted activity whose P and D bits had been saved in GRS locations 43 and 44.

As this is effectively discarding an interrupt, the result may be a software "timeout" due to loss of that interrupt.

2. P=202/203

   This is a multiple Even/Odd I/O interrupt case where the CPU has detected the interface error while already handling one. As captured P and D bits are overwritten by the second interrupt, this is unrecoverable and the system will EXERR (Error Exit).

3. 200 ≤ P < 240 and P ≠ 202 thru 206

   The error signal was detected during an interrupt not associated with I/O, so control will be given back to the interrupt handler so that software processing of it may be initiated. The original environment is still good as P and D bits have been saved in:

   - GRS 41 and 42 Immediate Storage Check
   - GRS 51 and 52 Guard Mode Violation
   - GRS 43 and 44 Normal Interrupts

   Details of the error will be logged prior to returning to the interrupt handler.

4. P ≠ Fixed Memory and P–1 is an I/O Instruction

   The error occurred as a result of a Condition Code = 1 condition on this instruction. Details of the error, including the addressed subchannel, will be logged and the CSWs in storage will be corrupted to an illegal recognizable value. Control can then be returned to P and D bits to process the results of the I/O instruction.

5. P ≠ Fixed Memory and P–1 is not an I/O Instruction

   This is an I/O instruction sequence which was executed correctly, but just happened to pick up the error signal from the IOU. The error will be logged and control given to the captured P and D bits.

6. Captured D–bits have D7 = 0, D2 = 1

   As neither interrupt nor I/O sequences take place in this state, it is assumed that the CPU is malfunctioning and the environment is suspect. The system will EXERR.

7. P = 0

   This CPU does not have the necessary hardware changes to recover from the error. The original interrupt (if there was one) will have stored P in GRS 41, 43, or 51. The Even/Odd I/O interrupt will store its captured D–bits in GRS 44.

NOTE:

Since software doesn't know which of cases 1 through 5 it has, there is no way to return control from the Even/Odd I/O interrupt. The system will EXERR.

7.5.2.4. Logging

Details to be logged by I/O error interrupt handling will include the following:

- Captured P and D bits
- IAW and CSWs for the CPU – These are suspect, of course.
- Even (202) or (203) IOU
- Addressed subchannel in condition code 1 case

8. Executive Control

8.1. General

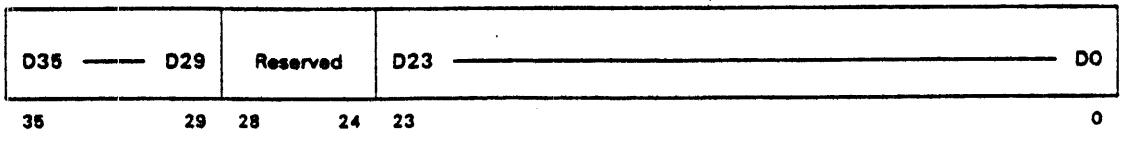
The 1100/80 Systems central processor unit (CPU) operates under the control of an Executive program which controls and coordinates the activities of the combined hardware and software systems and has exclusive use of certain control capabilities. By the use of bank descriptors, it can relocate any program in main storage. It provides storage protection through the use of storage limits registers. The bank descriptor specifications are contained in the general register stack (GRS). The bank descriptors are controlled by the Executive program for itself and for user programs. However, the user programs, through the Load Bank and Jump (LBJ), Load I-Bank Base and Jump (LIJ), and Load D-Bank Base and Jump (LDJ) instructions, are allowed to modify part of the designator register from a table prepared by the Executive program. The user programs are also allowed to modify several of the control bits by using the Load DR Designators instruction (LPD, see 5.13.1). The operations related to and affected by the contents of the bank descriptors are explained in this section.

8.2. Processor State

The processor state is defined as information contained in the CPU and required to describe a program activity. This includes the bank descriptor information, the designator register, the relative program address, and the GRS. The processor state is automatically saved in GRS when an interrupt occurs. The designator register and relative program address can also be stored by instruction. Each element of the processor state can be loaded by instruction, and certain elements may be loaded in groups to facilitate the orderly sequencing of program control.

8.2.1. Designator Register

The designator register contains information controlling functional characteristics of the CPU. The designator register may be loaded by instruction, although certain bit combinations are not valid or may not be available to the user. In general, no hardware checks are made for these invalid combinations. When an interrupt occurs, the current value of the designator register is stored in GRS, and the register is cleared, unless otherwise specified in the following paragraphs, to establish the proper interrupt handling environment. The format of the designator register is:



D35–D34 Must be zero

D33 Reserved

D32–D30 Must be zero

D29 Quantum Timer Enable

When this designator is one, the quantum timer value is decreased by one for every 100 nanoseconds period that the CPU is actually executing instructions. When the quantum timer value is zero, a Quantum Timer interrupt is generated. When D29 is zero, the quantum timer value is not altered.

D28–D24 Reserved

D23 Divide Check Designator

This designator is set to one when the magnitude of the quotient exceeds the range of the specified register.

D22 Characteristic Overflow Designator

This designator is set to one when the characteristic of a floating–point result is greater than 177_8 (single–precision) or 1777_8 (double–precision).

D21 Characteristic Underflow Designator

This designator is set to one when the characteristic of a floating–point result is less than -200_8 (single–precision) or -2000_8 (double–precision).

D20 Arithmetic Exception Interrupt Designator

When this designator is zero, if an arithmetic exception occurs (D23, D22, or D21 set to one), the specified A–registers are cleared to zero and no interrupt occurs. When D20 is one, if an arithmetic exception occurs, the specified A–registers are left unchanged (except as specified by D5), and an interrupt occurs. When D20 is one, all instructions that can cause an arithmetic exception are executed without instruction overlap (i.e., completely executed prior to beginning the execution of the next instruction).

D19 EXEC Bank Descriptor Table Pointer Enable

When this designator is zero, only the user bank descriptor table (BDT) pointer may be selected during execution of the LBJ, LIJ or LDJ instructions. If an attempt is made to reference the EXEC bank descriptor table, an Addressing Exception interrupt is generated. When D19 is one, either the EXEC or user BDT pointer may be selected during execution of an LBJ, LIJ, or LDJ instruction.

D18 Reserved

D17 Floating-Point Residue Store Enable

Enables residue store for single-precision floating-point instructions.

D16 BDR3 Write Protection

When this designator is one, a Guard Mode interrupt will occur if an attempt is made to write into the storage area specified by BDR3, and either D7 or the i-bit is zero.

D15 BDR1 Write Protection

When this designator is one, a Guard Mode interrupt will occur if an attempt is made to write into the storage area specified by BDR1, and either D7 or the i-bit is zero.

D14 BDR2 Write Protection

When this designator is one, a Guard Mode interrupt will occur if an attempt is made to write into the storage area specified by BDR2, and either D7 or the i-bit is zero.

D13 BDR0 Write Protection

When this designator is one, a Guard Mode interrupt will occur if an attempt is made to write into the storage area specified by BDR0, and either D7 or the i-bit is zero.

D12 BDR (Bank Descriptor Register) Selector

When this designator is one, BDR1 and BDR3 are selected as the primary pair of bank descriptor registers; when D12 is zero, BDR0 and BDR2 are selected as the primary pair. The primary pair is selected over the secondary pair if the storage limits overlap. D12 will be toggled during the execution of a jump instruction if the jump operand address falls exclusively within the limits of the secondary pair. D12 is not altered when an interrupt occurs.

D11 Ignored when D35 is zero.

D10 Quarter-Word Mode Designator

When this designator is zero, for instructions with function codes less than 70_8 (not including 07, 33, and 37), the j-field values of 4, 5, 6, and 7 are interpreted as follows:

- j = 4 Specifies half-word (18-bit) transfers to or from bits 35 through 18 of the operand.
- j = 5 Specifies third-word (12-bit) transfers to or from bits 11 through 0 of the operand.
- j = 6 Specifies third-word (12-bit) transfers to or from bits 23 through 12 of the operand.
- j = 7 Specifies third-word (12-bit) transfers to or from bits 35 through 24 of the operand.

When D10 is one, for instructions with function codes less than 70_8 (not including 07, 33, and 37), the j-field values of 4, 5, 6, and 7 are interpreted as follows:

j = 4 Specifies quarter-word (9-bit) transfers to or from bits 26 through 18 of the operand.

j = 5 Specifies quarter-word (9-bit) transfers to or from bits 8 through 0 of the operand.

j = 6 Specifies quarter-word (9-bit) transfers to or from bits 17 through 9 of the operand.

j = 7 Specifies quarter-word (9-bit) transfers to or from bits 35 through 27 of the operand.

The value of D10 has no effect on an instruction in the following circumstances:

- When the f-field of the instruction contains a value in the range 70_8 through 77_8, or 07, 33, or 37.
- When D4 is one.
- When the j-field contains a value other than 4, 5, 6, or 7.

D9 Reserved

D8 Floating-Point Zero Format Selection

This designator affects Floating Point Add, Floating Point Add Negative, Floating Point Multiply, Floating Point Divide, and Load and Convert to Floating (only single precision) instructions. When D8 is zero, and if the mantissa of the most significant word of a single-precision floating point result is ±0, the entire word is stored as all zeros. When D8 is one, and if the mantissa of the most significant word of a single-precision floating-point result is ±0, the most significant word is packed and stored with the appropriate characteristic.

D7 Relocation and Storage Suppression

D7 controls 1100 mode index register length, relocatability, and limit violation checking. If D7=0, index registers are 18 bits long, relocation is performed through basing, and a limits violation will cause a Guard Mode interrupt. If D7=1, the same functions are dependent upon the i-bit of the instruction currently executing: if i=0, operation proceeds as if D7=0; if i=1, index registers are 24 bits long, relocation is not performed (a base value is not added to the relative address), and relative addresses (which are not identical to absolute addresses) are not checked for limit violations.

Program addresses following a jump instruction are formed under the same D7 and i-bit conditions that were in effect for the jump instruction. That is, if an absolute jump occurred (D35=0, D7=i=1), subsequent instruction references will be absolute; if a relative jump occurred (D7 or i=0), subsequent instruction references will be relocated according to the current addressing control designators and BDR values. D7 is set to one on master clear or when an interrupt occurs.

D6 General Register Stack (GRS) Selection Designator

When D6 is zero, the GRS addresses below are assigned for use by the user program and can be referenced by the a- and x-fields of the instruction.

Index (X) Registers	$0001_8 - 0017_8$
Accumulators (A-Registers)	$0014_8 - 0033_8$
Special (R) Registers	$0101_8 - 0117_8$

When D6 is one, the GRS addresses below are assigned for use by the Executive program and can be referenced by the a- and x-fields of the instruction. D6 is set to one when an interrupt occurs.

Index (X) Registers	$0141_8 - 0157_8$
Accumulators (A-Registers)	$0154_8 - 0173_8$
Special (R) Registers	$0120_8 - 0137_8$

D5 Double-Precision Underflow Designator

When D5 is zero, a Floating-Point Characteristic Underflow interrupt occurs if characteristic underflow is detected during the execution of a double-precision floating-point instruction. The contents of the specified A-registers remain unchanged. When D5 is one, the interrupt does not occur; however, the contents of the specified A-registers are cleared to zeros and the normal instruction sequence is continued.

D4 Character Addressing Mode

When D4 is one, character addressing may be used by instructions for which the j-field is an operand qualifier (01-06, 10-32, 34-36, 40-67). Character addressing involves the utilization of J-registers which are selected by j-field values of 4, 5, 6, or 7. Note that character addressing for byte-oriented instructions that use J-registers (33 and 37, 06-07) does not depend on D4. The J-field functions specified by D34 and D10 are overridden by D4.

D3 Allow Interrupts Designator

When D3 is one, external interrupts are allowed; when D3 is zero, external interrupts are locked out. D3 may be altered by UR and LD, is set by AAIJ, and is cleared by PAIJ and the interrupt sequence.

D2 Privileged Instruction, GRS Protect, and Interrupt Lockout Detect

When D2 is one, a Guard Mode interrupt will occur if an attempt is made to execute a privileged (Executive) instruction, or to store into an Executive GRS location.

D2 equal to one also enables checking the length of the period during which interrupts are locked out by D3 equal to zero (whether D3 became zero by instruction execution or by taking an interrupt) or by a string of Execute Remotes or Indirects. A total of 256 storage references (only those made by the CPU are counted) are allowed during this locked-out period. Additional references will cause a Guard Mode interrupt. This checking occurs whether or not an interrupt is actually being locked out.

D1 & D0 Overflow Designator (D1) and Carry Designator (D0)

These designators are similar and so are defined together.

For the following instructions, D0 and D1 are cleared and then set according to the results of the operation:

A	(14)	Add to A
AN	(15)	Add Negative to A
AM	(16)	Add Magnitude to A
ANM	(17)	Add Negative Magnitude to A
AU	(20)	Add Upper
ANU	(21)	Add Negative Upper
AX	(24)	Add to X
ANX	(25)	Add Negative to X
DA	(71, 10)	Double–Precision Fixed Point Add
DAN	(71, 11)	Double–Precision Fixed Point Add Negative

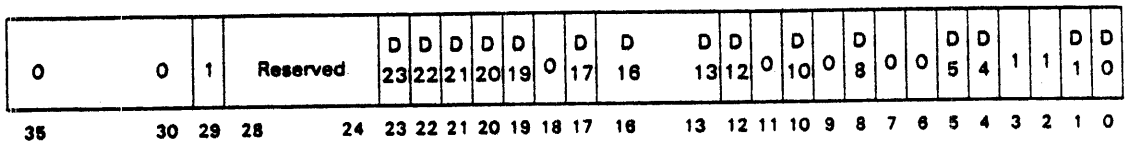
D1 is set to one if an overflow condition is detected during execution of any of the above instructions, and D0 is set to one if a carry condition is detected. When D0 or D1 are set, they remain so until another one of the above instructions is executed, or until the designator bits are directly altered by the program.

Figure 8–1 shows three basic conditions of the Designator Register.

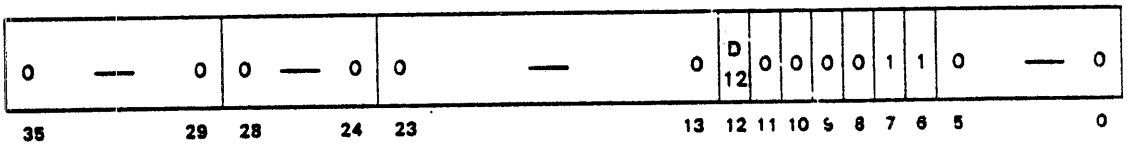
NOTE:

D16–D13 are independent of D2.

User Mode



Interrupt Mode



Executive Mode

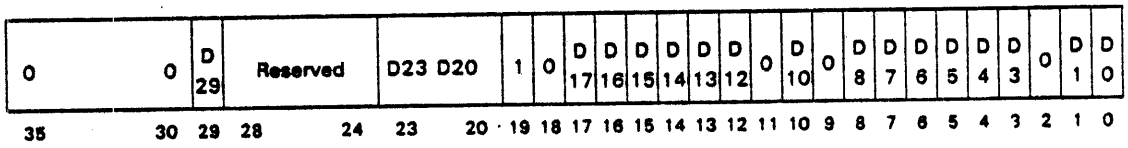


Figure 8–1. Basic Designator Register States

8.2.2. Dayclock

The internal dayclock register of each CPU is updated every 200 microseconds, whether the CPU is running or stopped (and power is applied). The dayclock is disabled, but not cleared, by master clear.

It is cleared only upon powerup or programmed reset to zero. Master clear prevents any dayclock updates until the processor is running and has been defined to have control of the dayclock. Once the processor has been defined to have control of the dayclock, it will continue to update the dayclock until it is mastercleared, even though it may be in a stop condition. The dayclock value is a single 36–bit binary number without separately identified digits of time.

The dayclock value is stored in a fixed storage location (module select register (MSR) + 216_8) when the dayclock has been both selected (via Select Dayclock instruction) and enabled (via Enable Dayclock instruction). If either condition is not true, the dayclock location is not altered by the dayclock. Only one dayclock can be selected in an application. That is, only one dayclock in a multiprocessor system can be selected unless the CPUs are partitioned into two applications.

The value in the dayclock location is kept current regardless of I/O waits or execution of extended instruction, e.g., block transfer, byte, etc.

The dayclock register value is replaced at the start of the next update cycle with the value in the dayclock location in fixed storage if the dayclock has been loaded (via the Load Dayclock instruction) since the last update cycle. If neither has occurred, the internal value is used for the update.

A dayclock interrupt request will be set every 6.5536 seconds if the dayclock has been both selected and enabled. The interrupt may be honored by any CPU in a multiprocessor system if in the same application, and it will occur only if interrupts are allowed and if the dayclock is enabled on that CPU.

8.3. Introduction to Addressing

The CPU's hardware provides for relocating the instructions and/or data for any program in main storage, and the ability to specify that all areas of main storage not assigned to a program are locked out to that program for read, write, and jump references. The main storage areas which may be assigned to a program are specified in 64-word granules beginning on any 512-word boundary and ending on any 64-word boundary.

8.3.1. Main Storage Organization

The 1100/80 Systems are designed as modular systems, permitting a variety of main storage configurations. The minimum main storage configuration comprises one 524K word basic cabinet containing two 262K banks. The storage capacity can be expanded to 4 MSUs in 524K word increments to a maximum of 4,194,304 words.

The 2-word bank descriptor register (BDR) provides the CPU with the flexibility for allocating storage for a program segment. The base value in conjunction with a relative address determines the absolute storage location. The upper and lower limits define the range of relative addresses within a program segment, with the upper limit specified in 64-word increments and the lower limit specified in 512-word increments.

8.3.2. Program Segmentation

A program may be written in segments which may be relocated in main storage. When the program is loaded into main storage, the Executive program determines the number of 64-word granules and assigns them in contiguous blocks. Any unfilled portion of a granule is unavailable to another program if it is to be run with guard mode/storage limits protection.

8.3.3. General Theory of 1100/80 Addressing

Normal 1100/80 Systems programs are constructed without consideration for the physical area of storage they will occupy during execution. As the program is constructed, each address is mapped into a set of addresses called relative addresses. A relative address is actually used in an instruction within the program which references other locations or words in the program. Proper conversion from these relative addresses to the physical locations of the program will occur during execution using the bank descriptor register mechanism of the 1100/80 Systems. The range of relative addresses is from 0 to 262,143.

Relative address (U) is composed of the sum of the u-field of the instruction, the modifier field of the Xx-register, and in certain cases, the word offset (Ow) field of the Jj-register. A negative zero (all ones) cannot be generated as a relative address. For shift instructions, I/O instructions, or immediate operands (j = 16 or 17), the relative address is generally used directly as an operand. If a relative address is less than 0200, it is generally used to reference GRS. If the relative address is greater than 0200, it is converted to an absolute address and used to reference storage.

An absolute address is normally composed of the sum of a relative address and a base value selected from one of the four available bank descriptor registers. It is also possible to have U generated as an absolute address and used directly to reference storage without being altered by addition of a base.

8.3.4. Bank Descriptor

A bank descriptor (BD) is a 2-word set of data defining storage allocation for a program segment. Bank descriptors are held in a bank descriptor table (BDT) which is located and defined by the bank descriptor table pointer (BDTP). The table address in the BDTP is the absolute address of the first word of the first BD in the BDT. The table length in the BDTP is in units of descriptors, not words. The BDs within the BDT are located by adding a bank descriptor index (BDI) to a BDTP without adding the bases. Therefore, the Guard Mode interrupt that may be produced is when the request is made to an SIU that has its interface disabled to the CPU. The BDI is also in units of descriptions. A table length value of zero defines a BDT containing only one BD. The BDT has a maximum length of 4K BDs (8K words). The BDT is expected to reside in storage (i.e., not in GRS). The BDTP and BD formats are shown in Figure 8-2.

8.3.5. Limits

The upper and lower limits define the range of relative addresses within a program segment. The check of a relative address against limits is inclusive, i.e., a relative address is within limits if it is greater than, or equal to, the lower limit; or less than, or equal to, the upper limit. The lower limit is in increments of 512 words, the upper limit is in increments of 64 words. Therefore, a program may contain any multiple of 64 words, beginning on any 512-word boundary and ending on any 64-word boundary.

8.3.6. Control Information

The flag and use count fields of the BD provide control pertaining to the relative space (segment) defined by the BD and a count of the activity or usage of this BD.

The R-flag indicates that an addressing exception interrupt is to occur if a reference is made to the segment through the new bank descriptor of an LIJ, LDJ, or LBJ instruction.

The W-flag indicates that a Guard Mode interrupt indicating write protection violation is to occur if a write is attempted into the segment.

The P-flag value is transferred to the privileged instruction, GRS protect, and interrupt lockout detect designator (D2) during the execution of an LBJ, LIJ, or LDJ instruction.

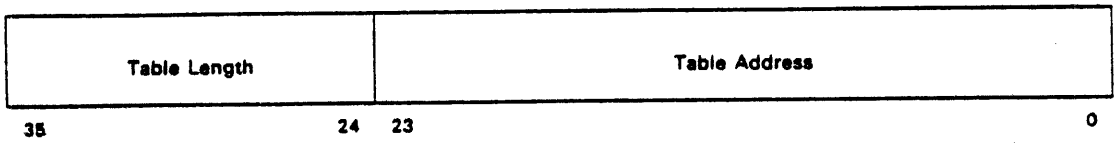
The V-flag indicates that entry point validation must be performed. This is accomplished by assuring that the relative operand address of the LBJ, LIJ, or LDJ instruction (jump address) that references the bank descriptor is equal to the bank descriptor lower limit value extended with low-order zeros (bits 8 through 0). This relative operand address must also select the BDR that is being loaded. If these conditions are not met and V is one, an addressing exception interrupt will occur. If the relative operand address is not within any limits, a Guard Mode interrupt will occur.

The C-flag indicates that an interrupt is to occur if the use count is decreased to zero.

8.3.7. Bank Descriptor Registers

A bank descriptor register (BDR) contains the upper limit, lower limit, and base of a bank descriptor; these allow relative address limits checking for protection, base selection, and absolute address formation. Four bank descriptor registers, BDR0, BDR1, BDR2, and BDR3, are provided in the CPU. The BDRs are loaded by the Load Addressing Environment (LAE), Load Limits (LL), or Load Base (LB) instructions. These instructions do not test the flags or change the use count.

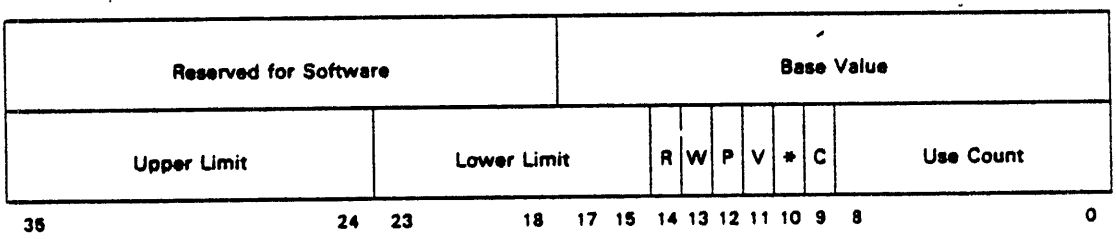
Bank Descriptor Table Pointer Format



Bits 35–24 Bank Descriptor Table Length (No. of Descriptors)

Bits 23–0 Bank Descriptor Table Address (Absolute)

Bank Descriptor Format



First Word:	Bits 35–18	Reserved for software
	Bits 17–0	Base value for relocation
Second Word:	Bits 35–24	Storage protection upper limit value
	Bits 23–15	Storage protection lower limit value
	Bit 14	Residency flag
	Bit 13	Write protection flag
	Bit 12	Privileged protection flag
	Bit 11	Validate entry point flag
	Bit 10	* Reserved for software
	Bit 9	Use count interrupt on zero flag
	Bits 8– 0	Use count value

Figure 8–2. Bank Descriptor and BDT Pointer Formats

8.3.8. Address Generation

If base suppress conditions exist, the relative address is used as the absolute address, otherwise an absolute address is generated. To generate an absolute address, a relative address is added to a base value selected from those available in the four bank descriptor registers. To select which of the base values to use, a limits check is made between the relative address and the upper and lower limits. BDR selector designator (D12) determines the order of BDR use as follows:

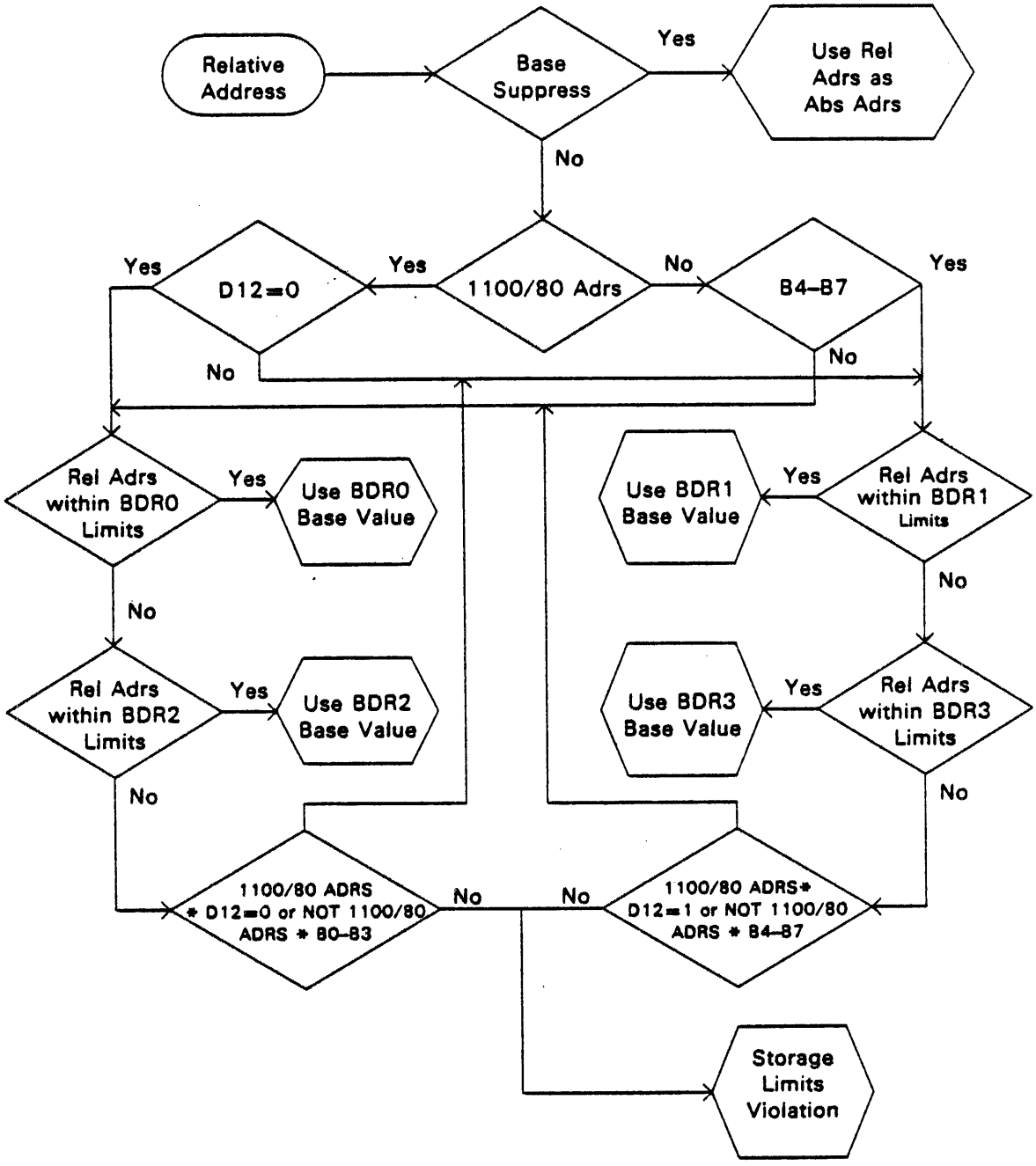
Selector	Use (in order of preference)
D12 = 0	BDR0, BDR2, BDR1, BDR3
D12 = 1	BDR1, BDR3, BDR0, BDR2

The lower limit (9 bits) of a BDR is checked against U_{17-9}, and the upper limit (12 bits) is checked against U_{17-6}. The first BDR passing the limits check is used for absolute address generation. Figure 8-3 shows the base value selection in flow chart form. If a relative address is not within limits of any BDR, a storage limits violation occurs. If a relative address is within limits, then the base corresponding to those limits is used to convert the relative address to an absolute address with which to reference storage. Base values may be assigned in 64-word increments.

The base addition is done with base and relative address alignments shown below:

Relative address	zeros		17 — 6	5 — 0	
Base value	17 — 12	11 — 0	zeros		
Absolute address	23 ——————————— 0				

The base addition is end off; i.e., a carry produced out of bit 23 is not propagated into bit 0.



Define B0–B3, B4–B7	as – (Used in 494 mode.)
Define BASE SUPPRESS	as – D35=0 * D7=1 * i=1 * NOT P-Fetch OR A-Flag * P-Fetch
Define 1100/80 ADRS	as – D35=0 NOT (D34=1 * D7=1 * i=0 * NOT inst ref)
Define 494 ADRS	as – NOT 1100/80 ADRS
Define INST REF	as – P-Fetch OR Jump Operand OR Execute Operand

Figure 8–3. Base Value Selection

8.3.9. P-Capturing Instructions

The program return address value (P) is the address of the instruction that is currently being executed. The P-capturing instructions are: Store Location and Jump (SLJ), and Load Modifier and Jump (LMJ).

A description of the P-capturing instructions (Store Location and Jump, and Load Modifier and Jump) is given in 5.9.1 and 5.9.2, respectively.

Appendix A. Abbreviations, Definitions, and Symbols

A	An arithmetic register. GRS addresses $14–33_8$ and $154–173_8$. Registers at addresses 34, 35, 174, and 175_8 can be used either as general purpose registers or as extensions of the sets of A-registers. In some cases A is used to mean Aa.
Aa	The A-register specified explicitly by the a-field of an instruction word.
A+1 Aa+1	An A-register having an address one greater than the address of the A-register specified by the a-field of an instruction word.
A+2 Aa+2	An A-register having an address of two greater than the address of the A-register specified by the a-field of an instruction word.
Absolute Address	A 36-bit address which identifies a specific location in main storage, as opposed to the relative address.
a-field	A-register designator (bits 25–22) of an instruction word. The a-field is interpreted in one of several ways, depending on the instruction word function code. The a-field may specify an A-register, an R-register, or an X-register. For the function code 70_8 (JGD instruction), the j-field and a-field are combined to specify a GRS address. The a-field also is used to specify the I/O channels, a jump key, stop keys, or as an extension of the function code of the instruction.
AND	Logical product
ASCII	American Standard Code for Information Interchange (seven bits)
Bank	A set of main storage locations having consecutive addresses. Defined by a bank descriptor word (BDW). Bank addressing is achieved by loading a base value in a designator register to be added to each bank relative address to produce the corresponding absolute address.
BD	Bank descriptor is a 2-word set of data defining storage allocation for a program segment.
BDI	Bank descriptor index. An integer value used as an index into a BDT.
BDI Registers	The two locations in the GRS which contain the BDIs (total of 4) for the banks currently addressable by the CPU. GRS locations 46 and 47_8.

BDR	Bank descriptor register.
BDT	Bank descriptor table.
BDTP	Bank descriptor table pointer
Block Multiplexer	A block multiplexer channel has multiple subchannels and always forces the I/O device to transfer data in multibyte mode.
Byte	A unit of information which consists of eight bits of data.
Byte Count	The number of bytes of data to be transferred to or from storage.
Byte Multiplexer	The byte multiplexer channel contains multiple subchannels and operates in either single or multi–byte mode.
CAW	The channel address word contains the instruction, IOU and CPU number, channel address, device address and the address of the first CCW.
CCW	Channel command word. A control word located anywhere in storage (location specified by the CAW) used for channel operations. The CCW specifies the device command, data address, CCW flags, format flag, and data count.
Channel	An I/O channel provides the hardware control and data paths required to direct the flow of data between a peripheral device and storage.
Channel Base Register	The contents of the channel base register are used to address control words in upper storage.
Characteristic	Biased exponent portion of a floating–point number.
Cluster	A group of units consisting of one SIU plus the CPUs and IOUs associated directly with it.
Condition Code	Indicates the channels response during the execution of an instruction.
Control Word	Refer to CAW and CCW.
Control Module	The control module handles all I/O instructions and resolves storage request and interrupt conflicts for up to eight channel modules.
Command Chaining	Allows execution of a new channel command word whenever the present operation is complete at the device level. This will result in the specification of a new operation with the same device without program intervention.
CPU	Central processor unit.
CSW	Channel status word. The channel stores status detected or received during execution of an I/O instruction and ending status associated with noncommunication subchannels and the status table subchannel in the CSW.
Data Chaining	Specifies a new buffer area in storage and permits continuous operation of the device without program intervention.
D–Bank	A bank based on BD.

Designator Bits
D bits

These bits are used to establish and provide control of the CPU operations and to report status. (See 8.2.1)

D0	carry designator
D1	overflow designator
D2	privileged instruction, GRS protect, and interrupt lockout detect
D3	allow interrupts designator
D4	character addressing mode
D5	double-precision underflow designator
D6	GRS selection designator
D7	relocation and storage suppression
D8	floating-point zero format selection
D9	reserved
D10	quarter-word mode designator
D11	ignored when D35 is zero
D12	BDR selector
D13	BDR0 write protection
D14	BDR2 write protection
D15	BDR1 write protection
D16	BDR3 write protection
D17	floating-point residue store enable
D18	reserved
D19	EXEC BDTP enable
D20	arithmetic exception interrupt designator
D21	characteristic underflow designator
D22	characteristic overflow designator
D23	divide check designator
D24-D28	reserved
D29	quantum timer enable

	D30–D32	must be zero
	D33	reserved
	D34–D35	must be zero

Device — A basic peripheral unit from or to which data is transferred in a system.

Device Address — An address generated in the CPU during an I/O instruction and by the control unit to indicate the address of the currently selected device. This is used to associate a particular device with a subchannel operation.

Double Word Boundary — Any even–numbered storage address.

E-bit — Bit 35 of the word in Xa for an LIJ/LDJ instruction and bits 35 and 17 of the BDI registers.

ECC — Error correction code.

EF — External function. A control signal sent by an IOU to a peripheral control unit to identify the word on the output data lines as a function word rather than an output data word.

EI — External interrupt. A control signal sent to an IOU by a peripheral control unit which identifies the word on the input word lines as a status word rather than an input data word.

ESI — Externally specified index.

ESI Interface — Word channel interface capable of addressing up to 64 communications devices on one I/O interface.

f-field — Function code designator (bits 35–30) of an instruction word. The f–field specifies the particular type of operation or function to be performed. The j– and a–fields serve as minor function codes on certain instructions.

Granule — Any group of 64 contiguous words in main storage having addresses in the range $XXXXX000_8$ through $XXXXX777_8$.

GRS — General register stack. A group of 112 addressable 36–bit control registers. The CPU uses these high–speed registers for holding intermediate results, indexing, and a variety of special functions such as repeat counting and holding status words.

HCCW — Hard channel control word.

h–field — Index register incrementation designator (bit 17) of an instruction word. The h–field controls index register modification and J–register modification.

HSTCW — Hard status table control word.

IAW — Interrupt address word

Ib — Increment in bytes. Bits 20–18 of a J–register. Used by byte instruction and other instructions operating in the character addressing mode (D4 = 1).

I-bank	A bank based on the I-bank base value of the bank descriptor register.
i-field	Indirect addressing designator (bit 16) of an instruction word. The i-field normally controls indirect addressing. It may be used instead to specify base register suppression/24-bit indexing or use of the utility base for operands, depending on the values of D7,
Immediate Command	An operation which will result in the subsystem generating an immediate status condition upon receipt of the command code.
Increment	The leftmost 18 bits (12 bits if D9 = D7 = i = 1) of an index register. Symbolized by Xi.
Instruction Word	A statement that specifies an operation and the values or locations of its operands.
I/O	Input/output
IOU	Input/output unit
ISI	Internally specified index
ISI Interface	A word channel interface which communicates with one peripheral control unit.
Iw	Increment in words. Bits 31-21 of a J-register. Used by byte instructions and by instructions operating in the character addressing mode (D4 = 1).
J	J-register (J0-J3) at GRS addresses $106-111_8$ or $126-131_8$. Used by byte instructions and by instructions operating in the character addressing mode (D4 = 1).
j-field	Operand qualifier, partial GRS address, or minor function code designator (bits 29-26) of an instruction word.
K	Used for notational convenience to replace the three low order digits of an integral power of 2 or an integral multiple thereof. Thus, 262K is used to represent $262,144(2^{18})$.
LJ0 LJ1 LJ2	Indicates the number of bytes in string SJ0, SJ1, and SJ2, respectively. Stored in $SR3_{35-27}$, $SR3_{26-18}$, and $SR3_{17-9}$, respectively. Maximum value is 511.
Main Storage Range (Half)	Consists of the main storage associated with upper or lower SIU segments but not both. It is the amount of main storage addressable above address 8 million or below address 8 million. Though it is referred to as a half, it is not necessarily half of main storage.
Main Storage Unit (MSU)	A free-standing storage cabinet made up of one to four 262K word modules of storage per unit. One to four units per system.
MSU Bank	Essentially one half of an MSU cabinet and made up of 1 or 2 modules of storage with common address and central logic. There are 1 to 8 MSU banks per system. A bank is a partitionable component on the STU.

MSU Module	The smallest increment of expansion of main storage, consisting of 262K words. There are from 1 to 16 MSU modules per system.
Major Function Code	The f-field of an instruction word.
Mantissa	The fractional part of a floating-point number.
Minor Function Code	A portion of an instruction word used with the f-field to specify the operation to be performed. For all instructions for which f = 07, 33, or 37_8 or for which f is greater than 70_8, the j-field contains a minor function code. For some instructions for which f is greater than 70_8, the a-field also contains a minor function code.
Modifier	The rightmost 18 bits (24 bits if D7 = i = 1) of an index register. Symbolized by Xm. It is added to the 16-bit address in the u-field of an instruction to produce a relative address (Xm is 18 bits) or absolute address (Xm is 24 bits).
MSR	Module select register
MUE	Multiple bit uncorrectable error.
Multi-Byte Mode	A type of operation available on the byte channel which permits a control unit to transfer several bytes of data before releasing the channel.
NI	Next instruction
Nonresident Subchannel	A set of control words held in reserve storage.
Nonshared Subchannel	A subchannel intended to operate with communications type peripheral devices. These subchannels allow concurrent access in an interleaving manner by a multiple number of devices through a multiplexing control unit to main storage.
Normalize	To normalize a number in floating point format, the mantissa is shifted left or right until the leftmost bit of the mantissa is not identical to the sign bit.
Ob	Offset in bytes. Bits 2-0 of a J-register. Used by byte instructions and by instructions operating in the character addressing mode (D4 = 1).
Option 0	Used with the subchannel expansion feature to provide four resident subchannels, four resident nonshared subchannels and 124 nonresident nonshared subchannels.
Option 1	Provides 128 nonshared subchannels. The eight most recently active are held in the channel. The remaining 120 subchannels are held in main storage.
OR	Logical inclusive OR
Ow	Offset in words. Bits 17-3 of a J-register. Used by byte instructions and by instructions operating in the character addressing mode (D4 = 1).
P	The program address or P-register

P-Register	The P-register contains the address of the instruction that is currently being executed.
Parity Bit	A binary digit appended to a group of bits to make the number of one bits always odd or always even.
PCI	Program controlled interrupt. (See 6.9.1.)
Program Controlled Interrupt	A program controlled bit in a CCW. When set, an interrupt and/or a table entry in the status table is made for that subchannel.
R	A special purpose control register specified explicitly or implicitly by an instruction word. GRS addresses 100_8 - 117_8 and 120_8 - 137_8.
Ra	The R-register specified by the a-field of an instruction word.
Relative Address	Normally, the address (U) formed by the addition of u, the address field of an instruction, and Xm, the modifier portion of the index register specified by the instruction (U = u + Xm). For byte instructions and instructions performed in the character addressing mode, the relative address is U = u + Xm + Ow. A relative address is not produced for instructions performed with base register suppression.
Relative P+1	An 18-bit relative address captured by certain jump instructions. Formed by subtracting the active PSRs BI or BD value which corresponds to the value used to develop the absolute jump to address for the most recent previous jump instruction from the address of the instruction following the current jump instruction.
Resident Subchannel	A set of control words held in the channel module.
Residue	The least significant result word produced by a single-precision Floating Add or Floating Add Negative instruction.
RTC	Real time clock
R0	Real time clock register at GRS address 100_8, or the control register at GRS address 120_8.
R1	Repeat count control registers at GRS addresses 101 and 121_8. They are used during Block Transfer, search, and masked search instructions.
R2	Mask control registers at GRS addresses 102 and 122_8. They are used during masked search instructions and the Masked Load Upper instruction.
R3-R5	Staging Register 1-3 (SR1-SR3). Used by byte instructions.
R6-R9	J-registers J0-J3. One or more of these registers are used by byte instructions and by instructions operating in the character addressing mode (D4 = 1).
S	Sign bit or bit position
SD	The 24-bit D-bank absolute address developed through addition: SD = (u + BD) + Xm or SD = (u + BD) + Xm + Ow.

Shared Subchannel	A subchannel is shared if two or more devices use the same subchannel for I/O operations. On a shared subchannel only one device at a time can transfer data.
SI	The 24-bit I-bank absolute address developed through addition: SI = (u + BI) + Xm or SI = (u + BI) + Xm + Ow.
SIOF Queue	Used for storing the device address for SIOF instructions presented by the CPU but not yet excuted by the IOU.
SJ0	A byte string whose starting word address is formed by summing the u-field of the instruction, the modifier of the index register specified by the instruction word, and the Ow-field of register J0. The Ob-field of J0 points to a byte within a word.
SJ1	A byte string based on J1, X+1, and Ow in the same manner as SJ0 is based on J0, X, and Ow.
SJ2	A byte string based on J2, X+2, and Ow in the same manner as SJ0 is based on J0, X, and Ow.
SK	Skip data (See 6.5.1)
SR1	Staging register 1 (R3), GRS addresses 103_8 or 123_8.
SR2	Staging register 2 (R4), GRS addresses 104_8 or 124_8.
SR3	Staging register 3 (R5), GRS addresses 105_8 or 125_8.
STCW	Status table control word
SIU	Storage interface unit, a free-standing cabinet made up of one to four 4K word segments of high-speed buffer storage. One unit required for each cluster configured.
SIU Half	The segments in an SIU which are associated with the lower address range or with the upper address range. There are 1 or 2 SIU segments in a half.
SIU Segment	A 4K word module of high-speed intermediate storage. There are 1 to 8 SIU segments per system. A segment is a partitionable component on the STU.
STU	System transition unit
Subchannel	A subchannel is an organization of uniquely addressable access paths which are capable of independently sustaining a single I/O operation concurrent with other I/O operations; i.e., a set of control words.
Subchannel Expansion Feature	Provides the capability for a channel module to operate with nonshared subchannels. (Refer to Option 0 and Option 1 for an explanation.)
Subsystem Clear	An I/O CLEAR signal originating at the IOU goes out on all 24 channels of that IOU.
System Reset	Clears all IOU registers and control designators, resets all peripheral subsystems and initializes all resident subchannels to idle mode.

TIC	Transfer in Channel. A command stored as part of the CCW to perform a branch between noncontiguous CCWs.
TSW	Tabled status word
U	The 18-bit value produced in the index subsection by adding the rightmost 18 bits (Xm-field) of the index register specified by the x-field of the instruction (or by adding 0 if X = 0) to the 16-bit value in the u-field of the instruction (u-field is extended to 18 bits). U = u + Xm or U = u + Xm + 0w.
u-field	The contents of bit positions 15-0 of an instruction word.
V-field	The relative address contained in bits 17-0 of an ISI or ESI access control word.
W-field	The count field of an access control word. For ISI operations, the W-field is bits 33-18. For half-word ESI operations, the W-field is bits 32-18. For quarter-word ESI operations, the W-field is bits 29-18.
Word Interface	A set of cable drivers and receivers for communicating with one peripheral control unit.
X	Index register. GRS addresses 1_8-17_8 and 141_8-157_8.
X+1	An index register having an address one greater than the address of the index register specified by the x-field of an instruction word.
X+2	An index register having an address two greater than the address of the index register specified by the x-field of an instruction word.
Xa	The X-register specified by the a-field of an instruction word.
Xi	Normally, bits 35-18 of an index register (bits 35-24 when D7 = i = 1). Used to increment or decrement Xm (the modifier) when specified by the instruction word.
x-field	Index register designator (bits 21-18) of an instruction word.
Xm	Normally, bits 17-0 of an index register (bits 23-0 when D9 = D7 = i = 1).
XOR	Logical exclusive OR
Xx	The X-register specified by the x-field of instruction word. In some cases X is used to mean Xx.
+0	Two words, one word, or a field consisting of all 0 bits.
-0	Two words, one word, or a field consisting of all 1 bits.
( )	The contents of the register or location identified by the symbol within the parentheses.
( )'	The ones complement of the register or location identified by the symbol within the parentheses.
$( )_n$	The contents of bit position n of the register or location identified by the symbol within the parentheses. For example, $(A)_{35}$ means the contents of bit position 35 of A.

$( )_{n-m}$	The contents of bit positions n through m of the register or location identified by the symbol within the parentheses. For example, $(X)_{17-0}$ means the contents of bit position 17 through 0 of X.
$\| \ \|$	Absolute value or magnitude
→	Direction of data flow

Appendix B. Summary of Word Formats

See 2.6 for the following:

System Status Word 0

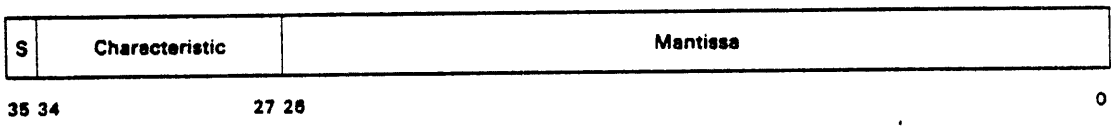
R/TM	Not Used	Load Path 0						Application 0																							
		Fault	CLUS		SIU	IOU	PROC	Auto			MAINT	MSU								SIU								IOU				PROC			
								NLP	PH1	PH0		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	3	2	1	0	3	2	1	0
35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

System Status Word 1

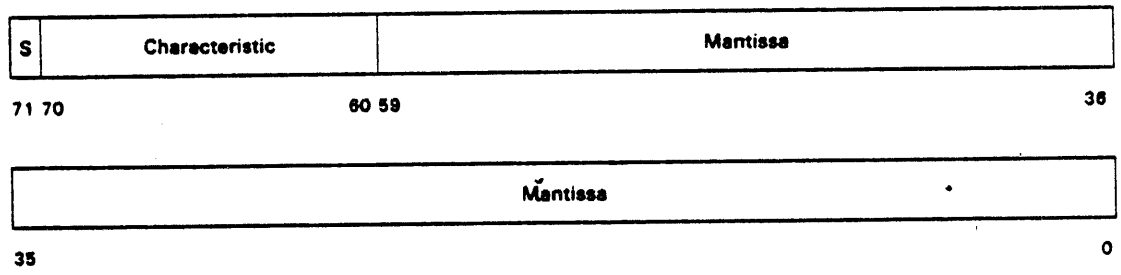
Not Used	Load Path 1						Application 1																								
	Fault	CLUS	SIU	IOU	PROC	Auto			MAINT	MSU								SIU								IOU				PROC			
						NLP	PH1	PH0		7	6	5	4	3	2	1	0	6	5	4	3	2	1	0	3	2	1	0	3	2	1	0
35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

See 4.2.8 for the following:

Single–Precision Floating–Point Format

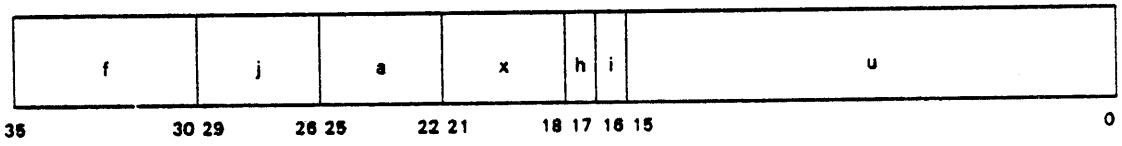


Double–Precision Floating–Point Format



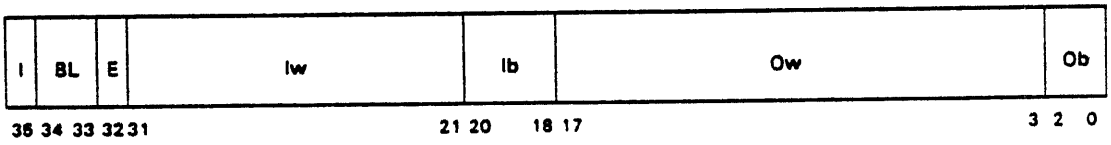
See 4.3.1 for the following:

Instruction Word Format



See 4.3.2.2.2 for the following:

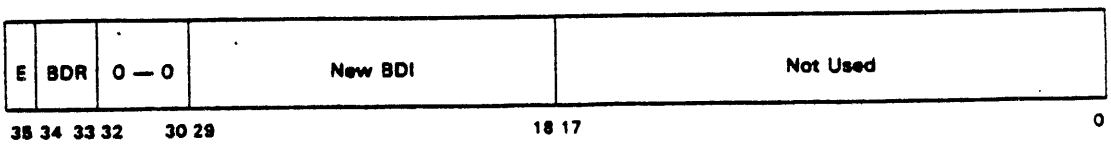
J–Register Format for Character Addressing Mode



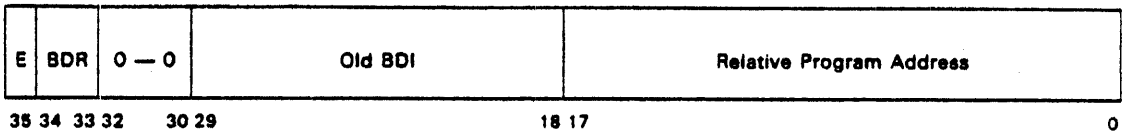
See 5.10.1 for the following:

Load Bank and Jump

Xa Before Execution

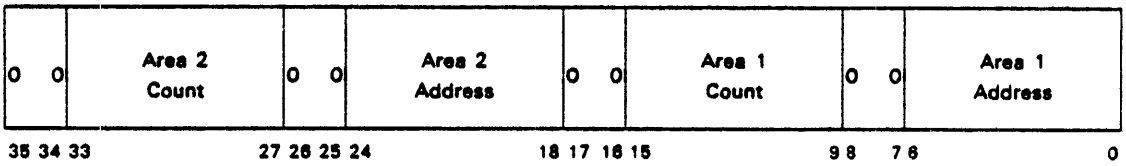


Xa After Execution



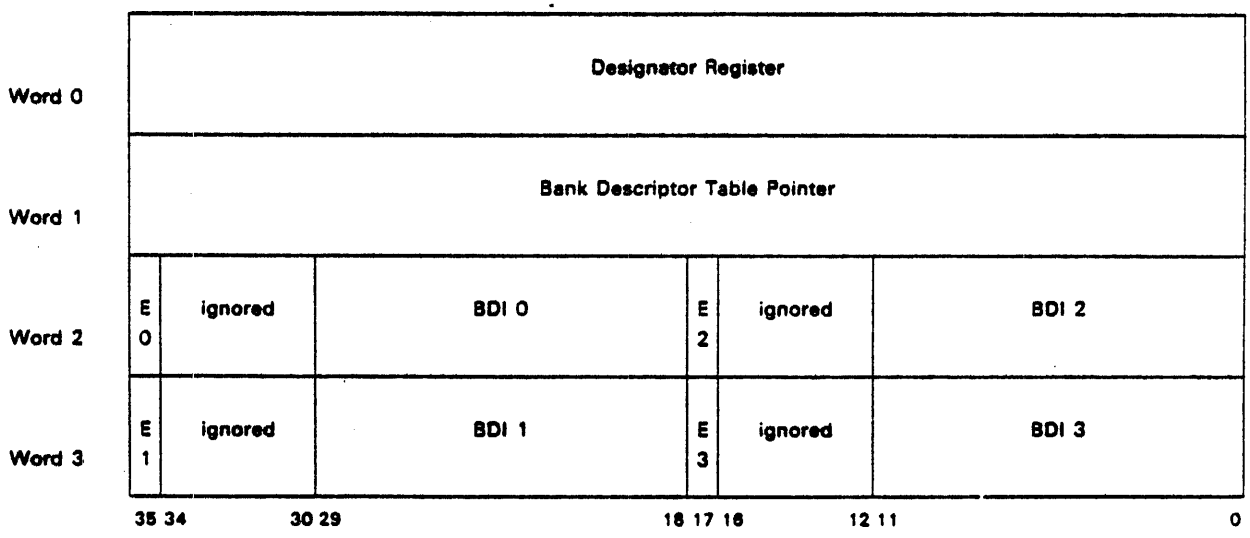
See 5.13.11 for the following:

Aa Format for Store Register Set



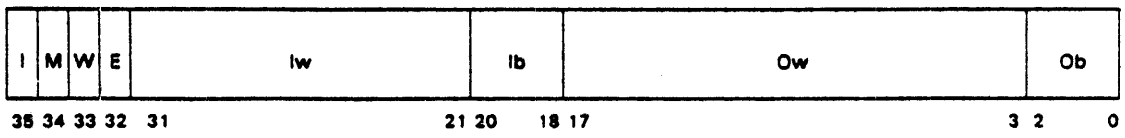
See 5.13.13 for the following:

Test Relative Address



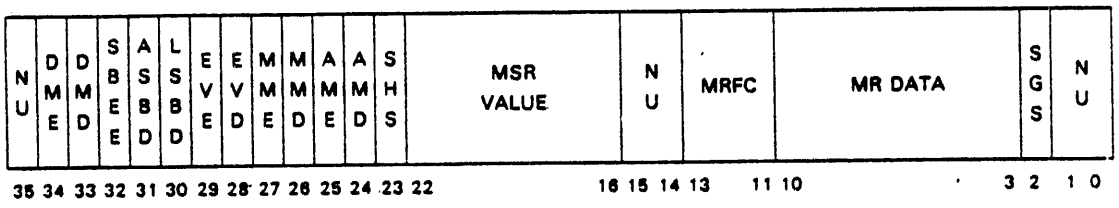
See 5.14 for the following:

J–Register Format



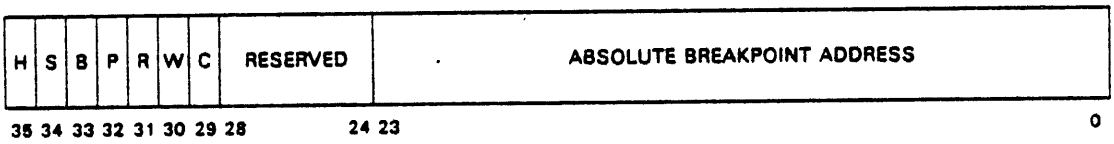
See 5.15.5 for the following:

Select Interrupt Locations



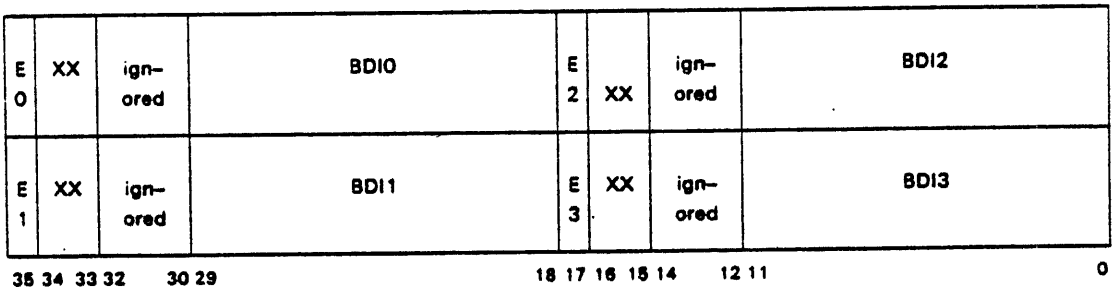
See 5.15.6 for the following:

Load Breakpoint Register



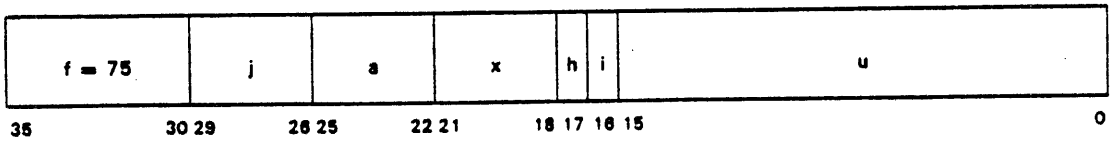
See 5.15.10 for the following:

Load Addressing Enviornment



See 6.3.4 for the following:

I/O Instruction Format



See 6.3.4 for the following:

CAW 0

Not Used	Operation Code	Not Used	IOU Number	Channel Address	Device Address
35 30	29 26	25 14	13 12	11 8	7 0

See 6.3.4 for the following:

CAW 1

Not Used	Address of first CCW or STCW
71 60	59 0

See 6.5.1 for the following:

Byte or Block Multiplexer Channel CCW

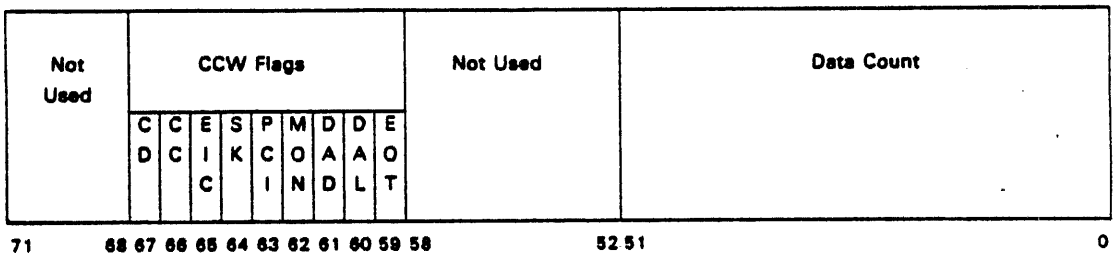
Not Used	Command Code	Data Address
35 32	31 24	23 0

Not Used	CCW Flags								Format Flags				Not Used	Data Count
	CD	CC	SLI	SK	PCI	TS	DAD	DAL	E	A	B	C		
71 68	67	66	65	64	63	62	61	60	59	58	57	56	55 52	51 0

See 6.5.1 for the following:

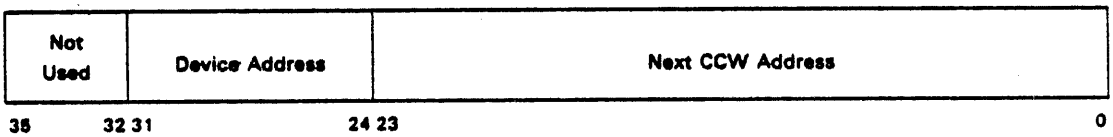
Word Channel CCW

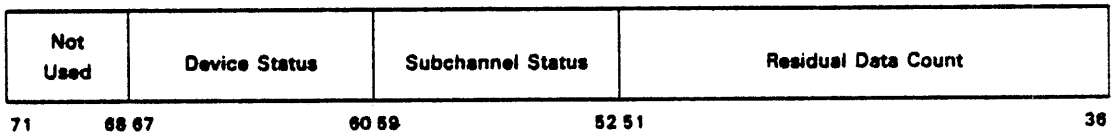
Not Used	Command Code			Data Address
		Format Control		
35 32	31 30	29 28	27 24	23 0

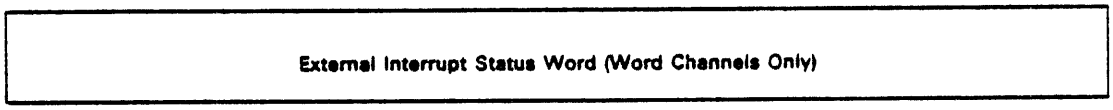


See 6.10 for the following:

CSW or TSW

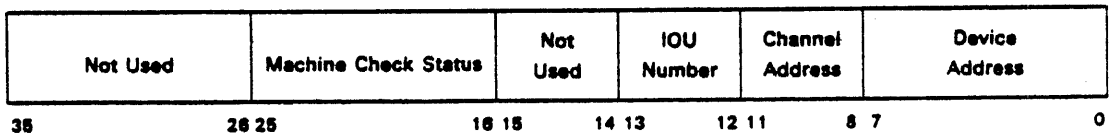






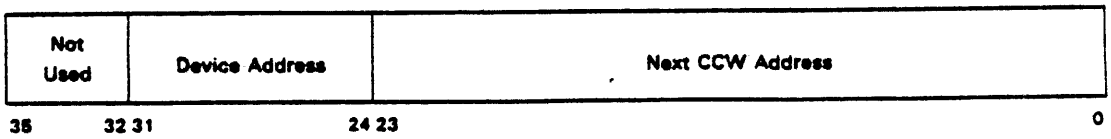
See 6.10 for the following:

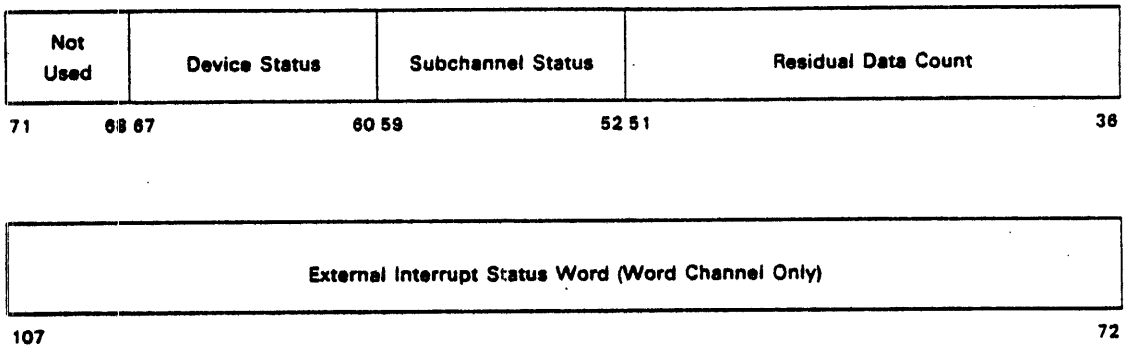
IAW



See 6.11 for the following:

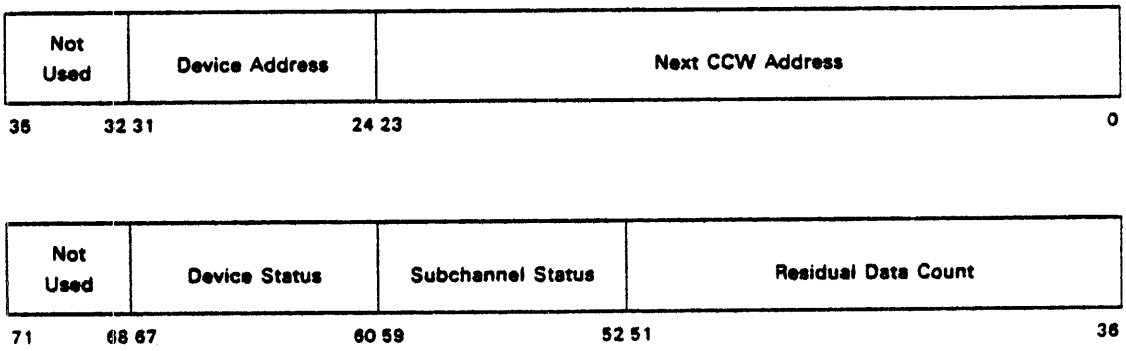
CSW for I/O Instruction





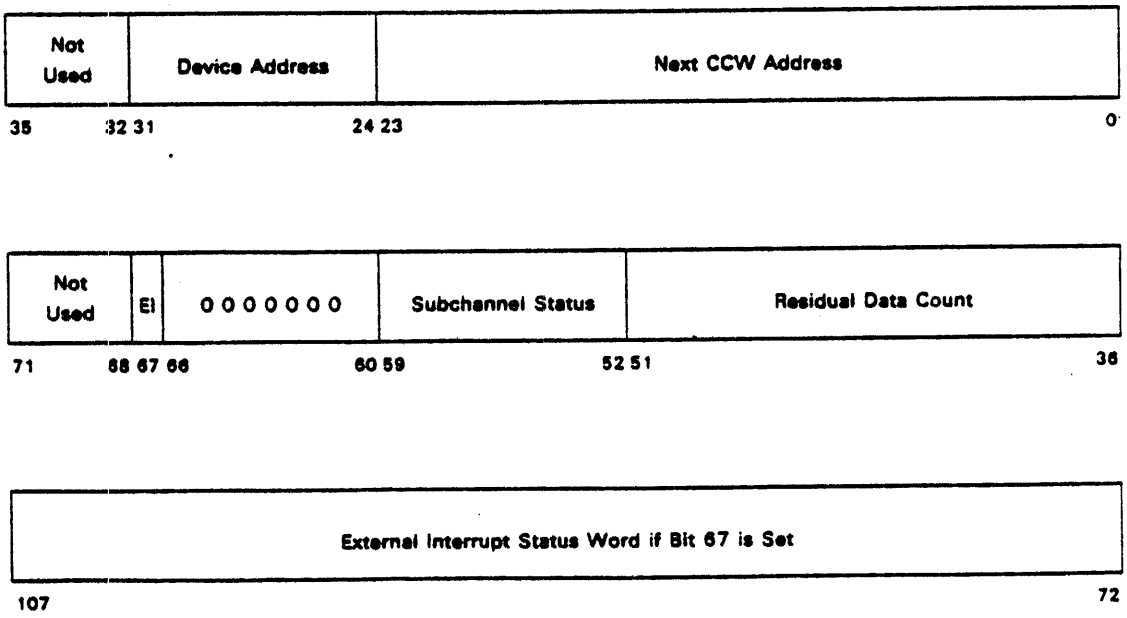
See 6.12 for the following:

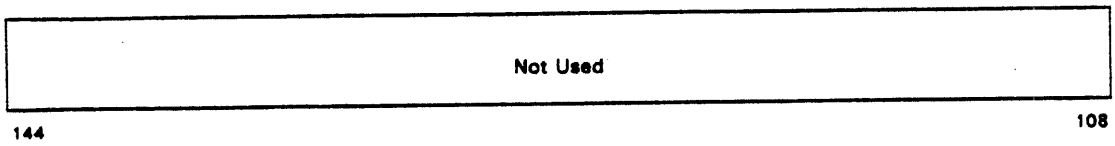
TSW for Nonshared Byte Multiplexer Subchannels



See 6.12 for the following:

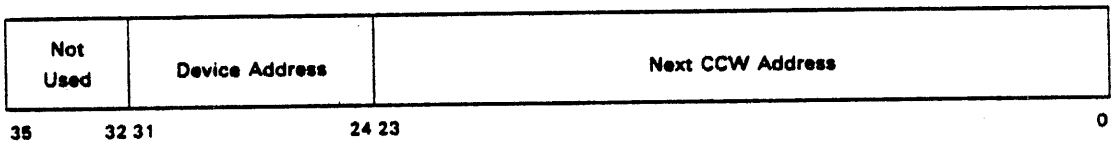
TSW for ESI Word Subchannels

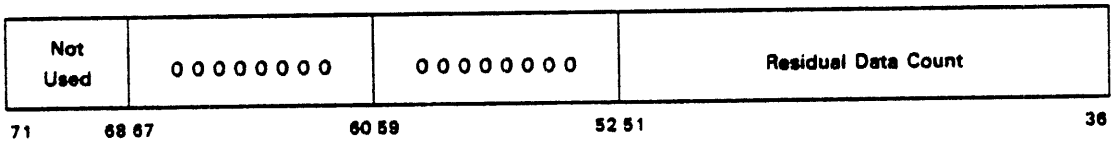




See 6.13 for the following:

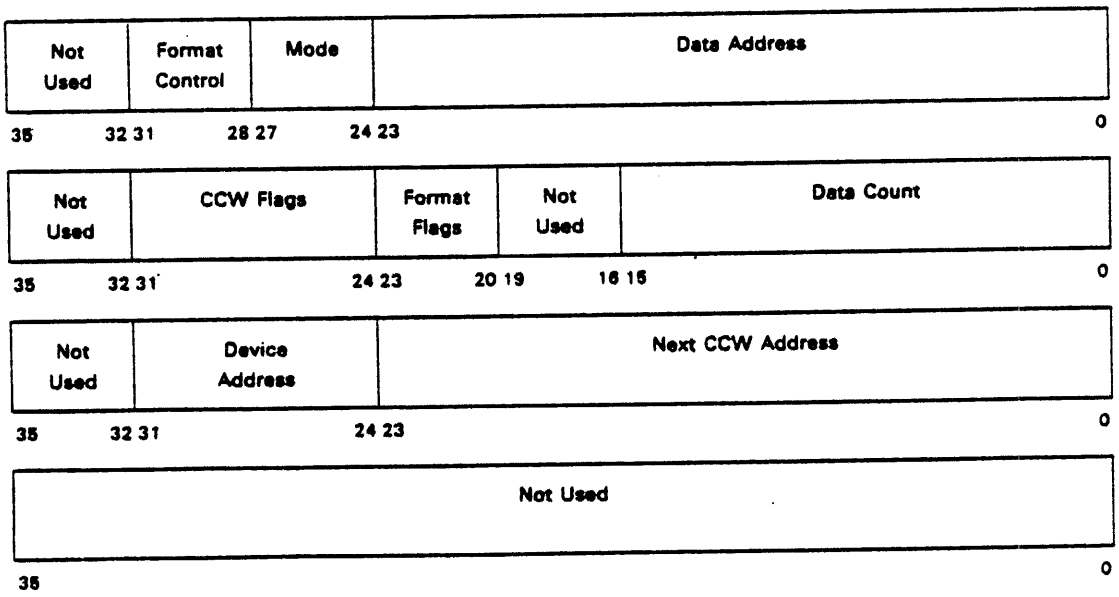
CSW for the Store Subchannel Status Command

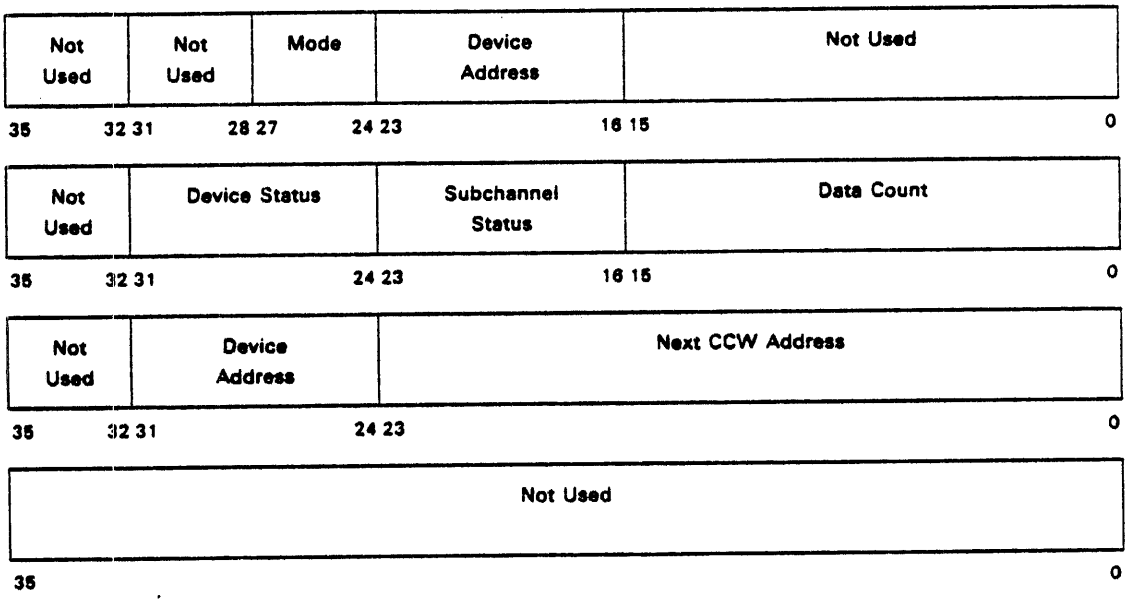




See 6.17 for the following:

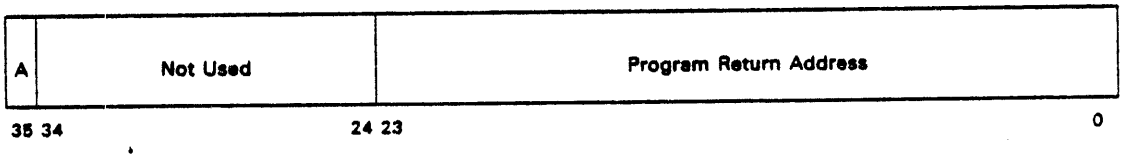
Scratch Pad Formats for Subchannel Expansion Feature





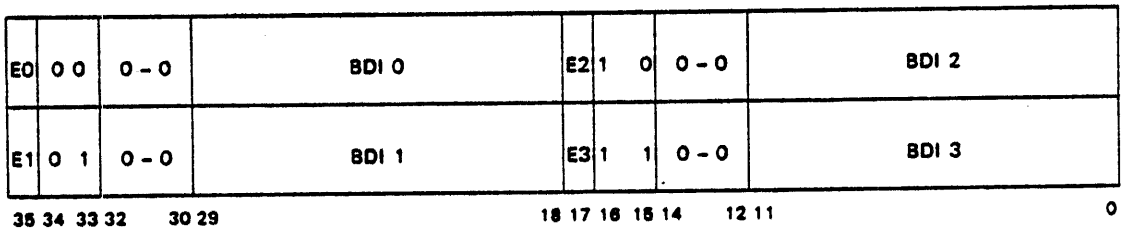
See 7.2.1 for the following:

Program Return Address



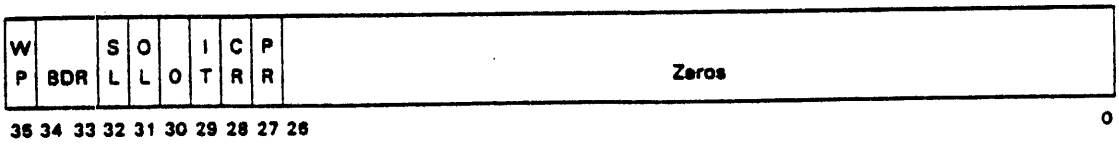
See 7.2.2 for the following:

Addressing Status

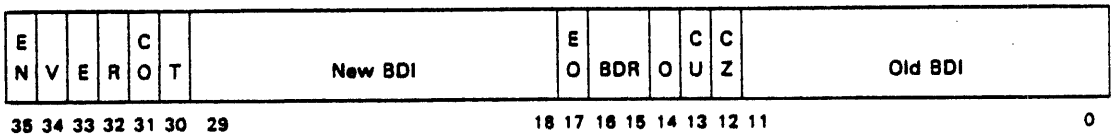


See 7.3.2 for the following:

Format of Guard Mode Interrupt Status

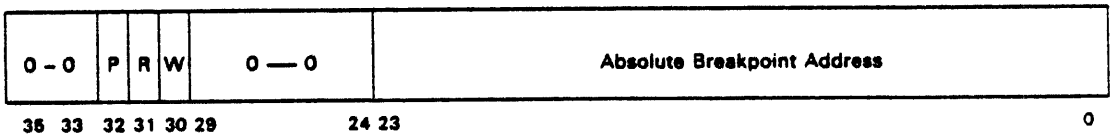


Format of Addressing Exception Interrupt Status



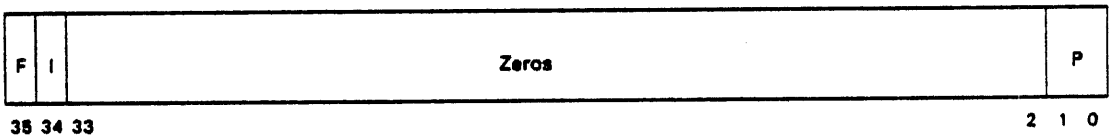
See 7.3.3 for the following:

Format of Breakpoint Interrupt Status



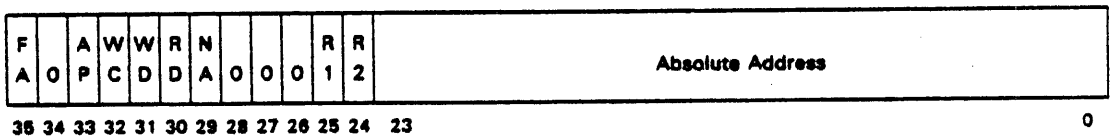
See 7.3.4 for the following:

Format of Interprocessor Interrupt Status



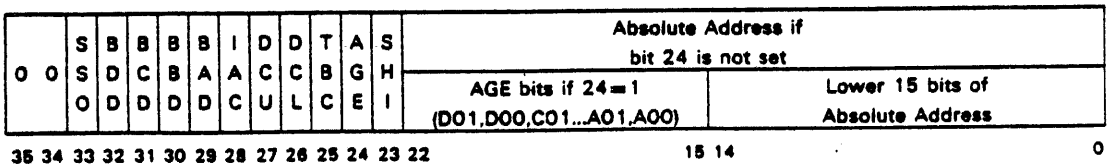
See 7.3.6.1 for the following:

Format of Immediate Storage Check Interrupt Status



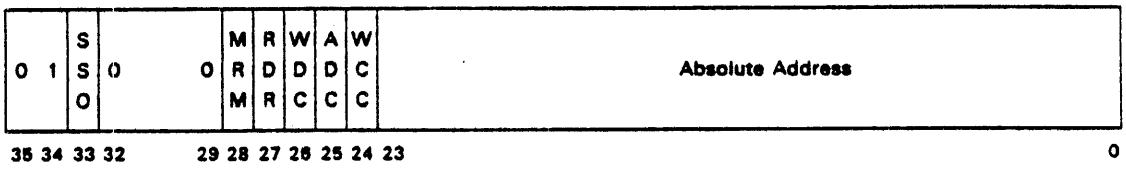
See 7.3.6.2.1 for the following:

Internal SIU Check Format



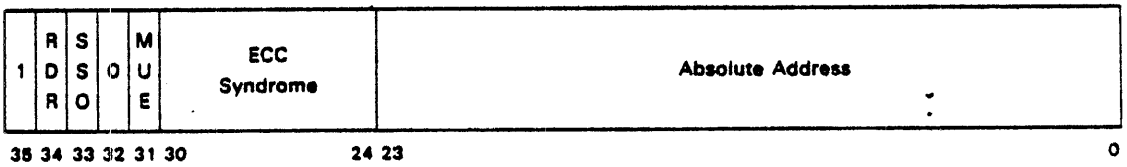
See 7.3.6.2.2 for the following:

SIU/MSU Interface Check Format



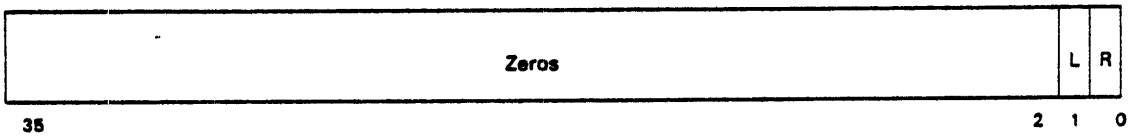
See 7.3.6.2.3 for the following:

SIU/MSU Read or Partial Write ECC Check Format



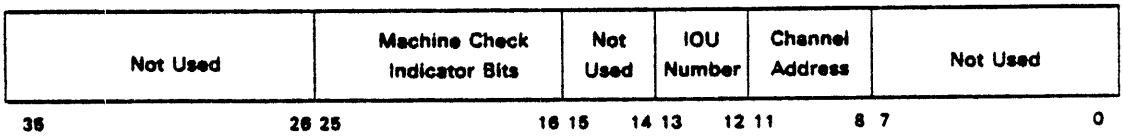
See 7.3.7 for the following:

Power Check Interrupt Status Word



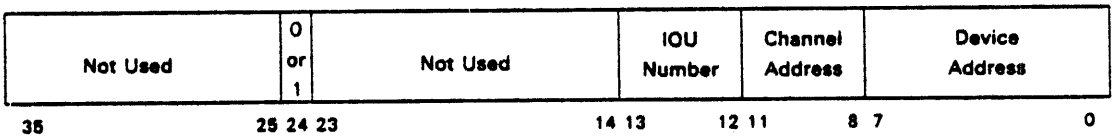
See 7.4.1 for the following:

Machine Check IAW

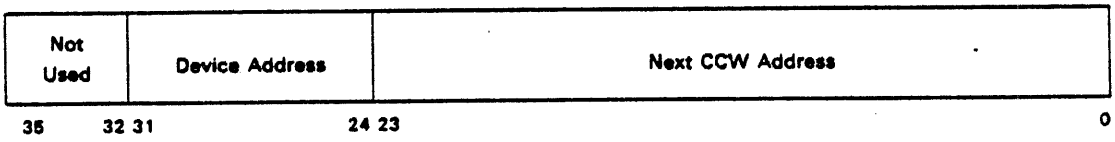


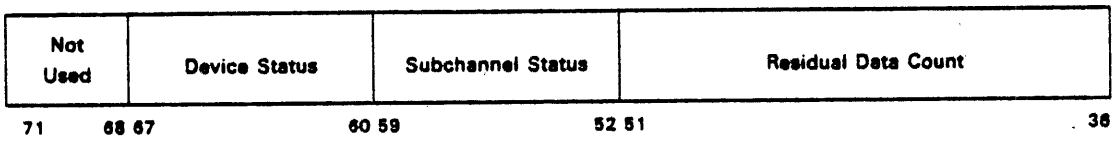
See 7.4.2 for the following:

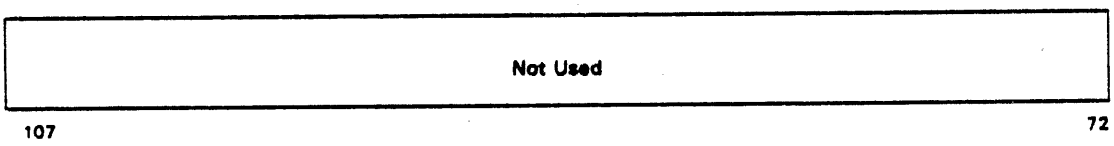
Normal Interrupt IAW



Normal Interrupt CSW

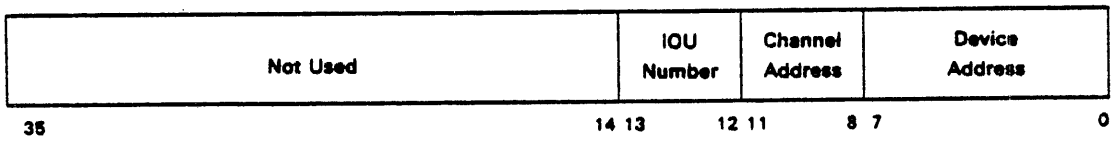




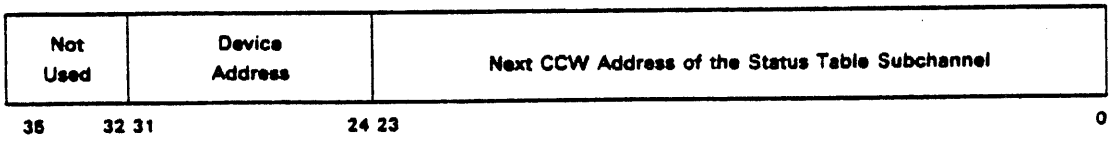


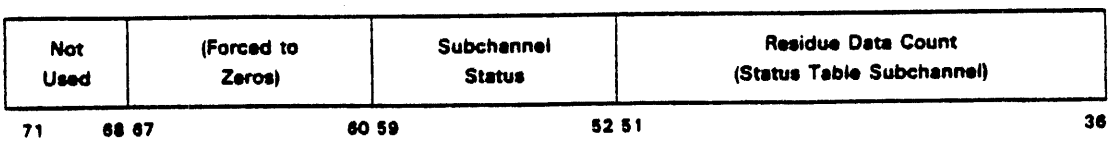
See 7.4.3 for the following:

Tabled Interrupt IAW



Tabled Interrupt CSW





Not Used

107 72

See 8.2.1 for the following:

Designator Register

D35	D29	Reserved	D23	D0

35 29 28 24 23 0

See 8.2.1 for the following:

Basic Designator Register States

User Mode

0	0	1	Reserved	D 23	D 22	D 21	D 20	D 19	0	D 17	D 16	D 13	D 12	0	D 10	0	D 8	0	0	D 5	D 4	1	1	D 1	D 0

35 30 29 28 24 23 22 21 20 19 18 17 16 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Interrupt Mode

0 — 0	0 — 0	0 — 0	D 12	0	0	0	0	1	1	0 — 0

35 29 28 24 23 13 12 11 10 9 8 7 6 5 0

Executive Mode

0	0	D 29	Reserved	D23 D20	1	0	D 17	D 16	D 15	D 14	D 13	D 12	0	D 10	0	D 8	D 7	D 6	D 5	D 4	D 3	0	D 1	D 0

35 30 29 28 24 23 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

See 8.3.4 for the following:

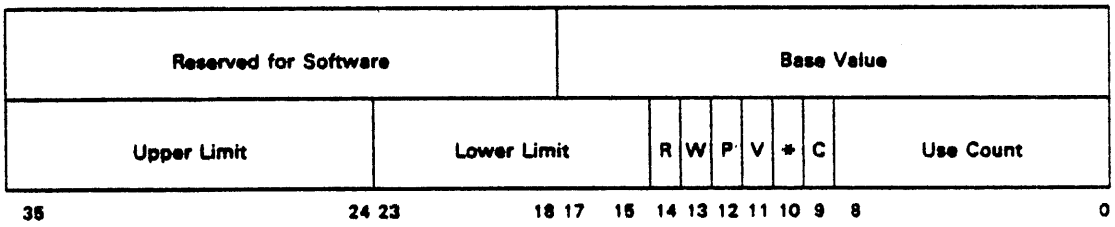
Bank Descriptor Table Pointer Format

Table Length	Table Address

35 24 23 0

See 8.3.4 for the following:

Bank Descriptor Format



Appendix C. Instruction Repertoire

Table C-1. Mnemonic/Function Code Cross-Reference

Mnemonic	Function Code (Octal) f	j	Paragraph Reference
A,AA	14		5.4.1
A,AX	24		5.4.7
AAIJ	74	07	5.9.3
AH	72	04	5.4.17
AM,AMA	16		5.4.3
AN,ANA	15		5.4.2
AN,ANX	25		5.4.8
AND	42		5.12.3
ANH	72	05	5.4.18
ANM,ANMA	17		5.4.4
ANT	72	07	5.4.20
ANU	21		5.4.6
AT	72	06	5.4.19
AU	20		5.4.5
BA	37	06	5.14.14
BAN	37	07	5.14.15
BC	33	04	5.14.4
BDF	33	15	5.14.11
BDI	33	11	5.14.7
BF	33	14	5.14.10
BI	33	10	5.14.6
BM	33	00	5.14.1
BMT	33	01	5.14.2
BT	22		5.3.8
BTC	33	03	5.14.3
CDU	76	07	5.5.16
DA	71	10	5.4.15
DAN	71	11	5.4.16
DDC	73	14	5.15.3
	a = 12		
DEC	05	00-17	5.13.14
	a = 11,13		
DF	36		5.4.14
DFA	76	10	5.5.3
DFAN	76	11	5.5.4
DFB	33	17	5.14.13
DFD	76	13	5.5.8
DFM	76	12	5.5.6
DFP,DLCF	76	15	5.5.12
DFU	76	14	5.5.10
DI	34		5.4.12
DIB	33	13	5.14.9
DJZ	71	16	5.11.2
DL	71	13	5.2.9
DLM	71	15	5.2.11
DLN	71	14	5.2.10
DLSC	73	07	5.8.8
DS	71	12	5.3.7
DSA	73	05	5.8.6
DSC	73	01	5.8.2
DSF	35		5.4.13
DSL	73	03	5.8.4
DTE	71	17	5.7.14
EDC	73	14	5.15.3
	a = 11		
EDIT	33	07	5.14.5

Table C-1. Mnemonic/Function Code Cross-Reference (continued)

Mnemonic	Function Code (Octal) f	j	Paragraph Reference
ENZ	05	00–17	5.13.14
	a = 14–17		
ER	72	11	5.13.4
EX	72	10	5.13.3
FA	76	00	5.5.1
FAN	76	01	5.5.2
FB	33	16	5.14.12
FCL	76	17	5.5.14
FD	76	03	5.5.7
FEL	76	16	5.5.13
FM	76	02	5.5.5
HCH	75	05	6.4.6
HDV	75	04	6.4.5
HJ,HKJ	74	05	5.11.10
IB	33	12	5.14.8
IIIX	73	15	5.15.19
	a = 04		
IMI	72	00	5.15.21
INC	05	00–17	5.13.14
	a = 10,12		
J,JK	74	04	5.11.9
JB	74	11	5.11.12
JC	74	16	5.11.22
JDF	74	14	5.11.17
	a = 03		
JFO	74	14	5.11.16
	a = 02		
JFU	74	14	5.11.15
	a = 01		
JGD	70		5.11.1
JMGI	74	12	5.11.13
JN	74	03	5.11.8
JNB	74	10	5.11.11
JNC	74	17	5.11.23
JNDF	74	15	5.11.21
	a = 03		
JNFO	74	15	5.11.20
	a = 02		
JNFU	74	15	5.11.19
	a = 01		
JNO	74	15	5.11.18
	a = 00		
JNS	72	03	5.11.4
JNZ	74	01	5.11.6
JO	74	14	5.11.14
	a = 00		
JP	74	02	5.11.7

Mnemonic	Function Code (Octal) f	j	Paragraph Reference
JPS	72	02	5.11.3
JZ	74	00	5.11.5
L,LA	10		5.2.1
L,LR	23		5.2.5
L,LX	27		5.2.7
LAE	73	15	5.15.11
	a = 12		
LB	73	15	5.15.9
	a = 10		
LBJ	07	17	5.10.1
LBRX	73	15	5.15.6
	a = 02		
LCF	76	05	5.5.11
LCR	75	10	6.4.7
LD	73	15	5.15.13
	a = 14		
LDC	73	14	5.15.2
	a = 10		
LDJ	07	12	5.10.3
LDSC	73	11	5.8.10
LDSL	73	13	5.8.12
LIJ	07	13	5.10.2
LL	73	15	5.15.10
	a = 11		
LM,LMA	12		5.2.3
LMJ	74	13	5.9.2
LN,LNA	11		5.2.2
LNMA	13		5.2.4
LPD	07	14	5.13.1
LQT	73	15	5.15.8
	a = 03		
LRS	72	17	5.13.12
LSC	73	06	5.8.7
LSSC	73	10	5.8.9
LSSL	73	12	5.8.11
LTCW	75	11	6.4.8
LUF	76	04	5.5.9
LXI	46		5.2.8
LXM	26		5.2.6
MASG	71	07	5.6.14
MASL	71	06	5.6.13
MCDU	76	06	5.5.15
MDA	73	14	5.15.20
	a = 14		
MDB	73	14	5.15.20
	a = 15		
MF	32		5.4.11

Table C-1. Mnemonic/Function Code Cross-Reference (continued)

Mnemonic	Function Code (Octal) f	j	Paragraph Reference
MI	30		5.4.9
MLU	43		5.12.4
MSE	71	00	5.6.7
MSG	71	03	5.6.10
MSI	31		5.4.10
MSLE,MSNG	71	02	5.6.9
MSNE	71	01	5.6.8
MSNW	71	05	5.6.12
MSW	71	04	5.6.11
NOP	74	06	5.13.10
OR	40		5.12.1
PAIJ	72	13	5.15.1
RAT	73	15	5.15.16
	a = 06		
S,SA	01		5.3.1
S,SR	04		5.3.4
S,SX	06		5.3.6
SAS	05	00-17	5.3.5
	a = 06		
SAZ	05	00-17	5.3.5
	a = 07		
SD	73	15	5.15.14
	a = 15		
SDC	73	14	5.15.4
	a = 13		
SE	62		5.6.1
SFS	05	00-17	5.3.5
	a = 04		
SFZ	05	00-17	5.3.5
	a = 05		
SG	65		5.6.4
SIL	73	15	5.15.5
	a = 00		
SIOF	75	01	6.4.2
SLE,SNG	64		5.6.3
SLJ	72	01	5.9.1
SM,SMA	03		5.3.3
SN,SNA	02		5.3.2
SNE	63		5.6.2
SNW	67		5.6.6
SNZ	05	00-17	5.3.5
	a = 01		
SN1	05	00-17	5.3.5
	a = 03		
SPD	07	15	5.13.2
SPID	73	15	5.15.7
	a = 05		
SP1	05	00-17	5.3.5
	a = 02		
SQT	73	15	5.15.12
	a = 13		
SRS	72	16	5.13.11
SSA	73	04	5.8.5
SSC	73	00	5.8.1
SSL	73	02	5.8.3
SSS	73	15	5.15.18
	a = 17		
SW	66		5.6.5
SZ	05	00-17	5.3.5
	a = 00		
TAP	73	15	5.15.17
	a = 07		
TCS	73	17	5.13.7
	a = 02		
TE	52		5.7.6
TEP	44		5.7.1
TG	55		5.7.9
TLE,TNG	54		5.7.8
TLEM,TNGM	47		5.7.3
TN	61		5.7.13
TNE	53		5.7.7
TNW	57		5.7.11
TNZ	51		5.7.5
TOP	45		5.7.2
TP	60		5.7.12
TRA	72	15	5.13.13
TS	73	17	5.13.5
	a = 00		
TSA	73	17	5.13.8
	a = 04		
TSC	75	03	6.4.4
TSS	73	17	5.13.6
	a = 01		
TSSA	73	17	5.13.9
	a = 05		
TW	56		5.7.10
TZ	50		5.7.4
UR	73	15	5.15.15
	a = 16		
XOR	41		5.12.2

Table C-2. Instruction Repertoire

Function Code (Octal)		Mnemonic	Instruction	Description
f	j			
00	0-17	-	Invalid Code	Causes Invalid Instruction Fault interrupt to MSR + 221_8
01	0-15	S,SA	Store A	(Aa) → U
02	0-15	SN,SNA	Store Negative A	- (Aa) → U
03	0-15	SM,SMA	Store Magnitude A	| (Aa) | → U
04	0-15	S,SR	Store R	(Ra) → U
05	0-17	SZ a = 00	Store Zero	Store constant 000000 000000, zeros, in location specified by operand address
05	0-17	SNZ a = 01	Store Negative Zero	Store constant 777777 777777, all ones, in location specified by operand address
05	0-17	SP1 a = 02	Store Postive One	Store constant 000000 000001, postive one, in location specified by operand address
05	0-17	SN1 a = 03	Store Negative One	Store constant 777777 777776, negative one, in location specified by operand address
05	0-17	SFS a = 04	Store Fieldata Spaces	Store constant 050505 050505, Fieldata spaces, in location specified by operand address
05	0-17	SFZ a = 05	Store Fieldata Zeros	Store constant 606060 606060, Fieldata zeros, in location specified by operand address
05	0-17	SAS a = 06	Store ASCII Spaces	Store constant 040040 040040, ASCII spaces, in location specified by operand address
05	0-17	SAZ a = 07	Store ASCII Zeros	Store constant 060060 060060, ASCII zeros, in location specified by operand address
05	0-17	INC a = 10	Increase Operand by One	Increase operand by one. If initial operand or result is zero, execute NI; if not zero, skip NI.
05	0-17	DEC a = 11	Decrease Operand by One	Decrease operand by one. If initial operand or result is zero, execute NI; if not zero, skip NI.
05	0-17	INC2 a = 12	Increase Operand by Two	Increase operand by two. If initial operand or result is zero, execute NI; if not zero, skip NI.

Table C-2. Instruction Repertoire (continued)

Function Code (Octal)		Mnemonic	Instruction	Description
f	j			
05	0–17	DEC2 a = 13	Decrease Operand by Two	Decrease operand by two. If initial operand or result is zero, execute NI; if not zero, skip NI.
05	0–17	ENZ a = 14–17	Increase Operand by Zero	Increase operand by zero. If initial operand or result is zero execute NI; if not zero, skip NI.
06	0–15	S,SX	Store X	(Xa) → U
07	0–11	-	Invalid Code	Causes Invalid Instruction Fault interrupt to MSR + 221_8
07	12	LDJ	Load D-Bank Base and Jump	Ignore Xa bit positions 34–33; if D12 = 0, select BDR2; if D12 = 1, select BDR3
07	13	LIJ	Load I-Bank Base and Jump	Ignore Xa bit positions 34–33; if D12 = 0, select BDR0; if D12 = 1, select BDR1
07	14	LPD	Load DR Designators	$U_{6,5,3-0}$ → Designator Register Bit 6 → D20 Bit 2 → D8 Bit 5 → D17 Bit 1 → D5 Bit 3 → D10 Bit 0 → D4
07	15	SPD	Store DR Designators	Designator Register D-bits → U_{6-0} D20 → Bit 6 D8 → Bit 2 D17 → Bit 5 D5 → Bit 1 D12 → Bit 4 D4 → Bit 0 D10 → Bit 3
07	16	-	Invalid Code	Causes Invalid Instruction Fault interrupt to MSR + 221_8
07	17	LBJ	Load Bank and Jump	Load BDR; jump to location specified by the operand address
10	0–17	L,LA	Load A	(U) → A
11	0–17	LN,LNA	Load Negative A	- (U) → A
12	0–17	LM,LMA	Load Magnitude A	| (U) | → A
13	0–17	LNMA	Load Negative Magnitude A	- | (U) | → A
14	0–17	A,AA	Add to A	(A) + (U) → A

Table C-2. Instruction Repertoire (continued)

Function Code (Octal)		Mnemonic	Instruction	Description
f	j			
15	0–17	AN,ANA	Add Negative To A	(A) – (U) → A
16	0–17	AM,AMA	Add Magnitude To A	(A) + | (U) | → A
17	0–17	ANM,ANMA	Add Negative Magnitude to A	(A) – | (U) | → A
20	0–17	AU	Add Upper	(A) + (U) → A+1
21	0–17	ANU	Add Negative Upper	(A) – (U) → A+1
22	0–15	BT	Block Transfer	(Xx + u) → Xa + u; repeat k times
23	0–17	L,LR	Load R	(U) → Ra
24	0–17	A,AX	Add to X	(Xa) + (U) → Xa
25	0–17	AN,ANX	Add Negative to X	(Xa) – (U) → Xa
26	0–17	LXM	Load X Modifier	(U) → Xa_{17-0}; Xa_{35-18} unchanged
27	0–17	L,LX	Load X	(U) → Xa
30	0–17	MI	Multiply Integer	(A) x (U) → A, A+1
31	0–17	MSI	Multiply Single Integer	(A) x (U) → A
32	0–17	MF	Multiply Fractional	(A) x (U) → A, A+1, left circular one bit
33	00	BM	Byte Move	Transfer LJO bytes from source string to receiving string. Truncate or fill receiving string as required
33	01	BMT	Byte Move With Translate	Translate and transfer LJO bytes from source string to receiving string. Truncate or fill receiving string as required
33	02	BTT	Byte Translate and Test	Translate and test N bytes against (A); if not equal, terminate instruction with J0 pointing to unequal byte and N ≠ 0. (This instruction simulated by library procedures and routines.)

Table C-2. Instruction Repertoire (continued)

Function Code (Octal)		Mnemonic	Instruction	Description
f	j			
33	03	BTC	Byte Translate and Compare	Translate and compare LJ0 bytes from string SJ0 to LJ1 bytes from string SJ1; terminate instruction on not equal or if both LJ0 and LJ1 are zero, when: (Aa) > 0; string SJ0 > SJ1 (Aa) = 0; string SJ0 = SJ1 (Aa) < 0; string SJ0 < SJ1
33	04	BC	Byte Compare	Compare LJ0 bytes from string SJ0 to LJ1 bytes from string SJ1; terminate instruction on not equal or if both LJ0 and LJ1 are zero
33	05	BPD	Byte to Packed Decimal Convert	Convert N bytes in string E to packed decimal in string F. (This instruction simulated by library procedures and routines.)
33	06	PDB	Packed Decimal to Byte Convert	Convert N packed decimal digits in string E to bytes in string F. (This instruction simulated by library procedures and routines.)
33	07	EDIT	Edit	Edit string SJ0 and transfer to string SJ1 under the control of string SJ2
33	10	BI	Byte to Binary Single Integer Convert	Convert LJ0 bytes in string SJ0 to a signed binary integer in register A
33	11	BDI	Byte to Binary Double Integer Convert	Convert LJ0 bytes in string SJ0 to a signed binary integer in registers A and A+1
33	12	IB	Binary Single Integer to Byte Convert	Convert signed binary integer in A to byte format and store in string SJ0
33	13	DIB	Binary Double Integer to Byte Convert	Convert the binary integer in A and A+1 to byte format and store in string SJ0
33	14	BF	Byte to Single Floating Convert	Convert LJ0 bytes in string SJ0 to a single length floating point format in register A
33	15	BDF	Byte to Double Floating Convert	Convert LJ0 bytes in string SJ0 to a double length floating point format in registers A and A+1

Table C-2. Instruction Repertoire (continued)

Function Code (Octal)		Mnemonic	Instruction	Description
f	j			
33	16	FB	Single Floating to Byte Convert	Convert the single length floating point number in A to byte format and store in string SJ0
33	17	DFB	Double Floating to Byte Convert	Convert double length floating point number in A and A+1 to byte format and store in string SJ0
34	0–17	DI	Divide Integer	(A, A+1) divided by (U) → A; REMAINDER → A+1
35	0–17	DSF	Divide Single Fractional	[(A, 36 sign bits) right algebraic shift 1 place] divided by (U) → A+1
36	0–17	DF	Divide Fractional	[(A, A+1) right algebraic shift 1 place] divided by (U) → A; REMAINDER → A+1
37	00	QB	Quarter Word Byte to Binary Compress	Discard $(A)_{35}$, $(A)_{26}$, $(A)_{17}$, and $(A)_{8}$; place the remaining bits in A_{31-0}; $(A)_{31} \rightarrow A_{35-32}$. (This instruction simulated by library procedures and routines.)
37	01	BQ	Binary to Quarter Word Byte Extend	Discard $(A)_{35-32}$; place the remaining bits in A_{34-27}, A_{25-18}, A_{16-9}, and A_{7-0}; zero fill A_{35}, A_{26}, A_{17}, and A_{8}. (This instruction simulated by library procedures and routines.)
37	02	QBH	Quarter Word Byte to Binary Halves Compress	Discard $(A)_{35}$, $(A)_{27}$, $(A)_{17}$, and $(A)_{8}$; place the remaining bits in A_{33-18} and A_{15-0}; $(A)_{33} \rightarrow A_{35-34}$; $(A)_{15} \rightarrow A_{17-16}$. (This instruction simulated by library procedures and routines.)
37	03	BHQ	Binary Halves to Quarter Word Byte Extend	Discard $(A)_{35-34}$ and $(A)_{17-16}$; place the remaining bits in A_{34-27}, A_{25-18}, A_{16-9}, and A_{7-0}; zero fill A_{35}, A_{26}, A_{17}, and A_{8}. (This instruction simulated by library procedures and routines.)
37	04	QDB	Quarter Word Byte to Double Binary Compress	Discard A_{35}, A_{26}, A_{17}, A_{8}, $A+1_{35}$, $A+1_{26}$, $A+1_{17}$, and $A+1_{8}$; place the remaining bits in A_{27-0} and A+1; $(A)_{27} \rightarrow A_{35-28}$. (This instruction simulated by library procedures and routines.)

Table C-2. Instruction Repertoire (continued)

Function Code (Octal)		Mnemonic	Instruction	Description
f	j			
37	05	DBQ	Double Binary to Quarter Word Byte Extend	Discard $(A)_{35-28}$; place the remaining bits from A and A+1 in A_{34-27}, A_{25-18}, A_{16-9}, and A_{7-0}, $A+1_{34-27}$, $A+1_{25-18}$, $A+1_{16-9}$, and $A+1_{7-0}$; zero fill A_{35}, A_{26}, A_{17}, A_8, $A+1_{35}$, $A+1_{26}$, $A+1_{17}$, and $A+1_8$. (This instruction simulated by library procedures and routines.)
37	06	BA	Byte Add	Add the LJ0 bytes in string SJ0 to the LJ1 bytes in string SJ1 and store the results in string SJ2
37	07	BAN	Byte Add Negative	Subtract the LJ0 bytes in string SJ0 from the LJ1 bytes in string SJ1 and store the results in string SJ2
37	10–17	-	Invalid Code	Causes Invalid Instruction Fault interrupt to MSR + 221_8
40	0–17	OR	Logical OR	(A) OR (U) → A+1
41	0–17	XOR	Logical Exclusive OR	(A) XOR (U) → A+1
42	0–17	AND	Logical AND	(A) AND (U) → A+1
43	0–17	MLU	Masked Load Upper	[(U) AND (R2)] OR [(A) AND NOT (R2)] → A+1
44	0–17	TEP	Test Even Parity	Skip NI if (U) AND (A) has even parity
45	0–17	TOP	Test Odd Parity	Skip NI if (U) AND (A) has odd parity
46	0–17	LXI	Load X Increment	(U) → $(Xa)_{35-18}$; $(Xa)_{17-0}$ unchanged
47	0–17	TLEM	Test Less Than or Equal to Modifier	Skip NI if $(U)_{17-0} \leqslant (Xa)_{17-0}$; always $(Xa)_{17-0}$ + $(Xa)_{35-18}$ → Xa_{17-0}
		TNGM	Test Not Greater Than Modifier	
50	0–17	TZ	Test Zero	Skip NI if (U) = ±0
51	0–17	TNZ	Test Nonzero	Skip NI if (U) ≠ ±0
52	0–17	TE	Test Equal	Skip NI if (U) = (A)

Table C-2. Instruction Repertoire (continued)

Function Code (Octal)		Mnemonic	Instruction	Description
f	j			
53	0–17	TNE	Test Not Equal	Skip NI if (U) ≠ (A)
54	0–17	TLE TNG	Test Less Than or Equal Test Not Greater	Skip NI if (U) ≤ (A)
55	0–17	TG	Test Greater	Skip NI if (U) > (A)
56	0–17	TW	Test Within Range	Skip NI if (A) < (U) ≤ (A+1)
57	0–17	TNW	Test Not Within Range	Skip NI if (U) ≤ (A) or (U) > (A+1)
60	0–17	TP	Test Positive	Skip NI if $(U)_{35} = 0$
61	0–17	TN	Test Negative	Skip NI if $(U)_{35} = 1$
62	0–17	SE	Search Equal	Skip NI if (U) = (A), else repeat
63	0–17	SNE	Search Not Equal	Skip NI if (U) ≠ (A), else repeat
64	0–17	SLE SNG	Search Less Than or Equal Search Not Greater	Skip NI if (U) ≤ (A), else repeat
65	0–17	SG	Search Greater	Skip NI if (U) > (A), else repeat
66	0–17	SW	Search Within Range	Skip NI if (A) < (U) ≤ (A+1), else repeat
67	0–17	SNW	Search Not Within Range	Skip NI if (U) ≤ (A) or (U) > (A+1), else repeat
70	0–17	JGD	Jump Greater and Decrement	Jump to U if $(\text{Control Register})_{ja} > 0$; go to NI if $(\text{Control Register})_{ja} \leq 0$; always $(\text{Control Register})_{ja} - 1 \rightarrow \text{Control Register}_{ja}$
71	00	MSE	Masked Search Equal	Skip NI if (U) AND (R2) = (A) AND (R2), else repeat
71	01	MSNE	Masked Search Not Equal	Skip NI if (U) AND (R2) ≠ (A) AND (R2), else repeat
71	02	MSLE	Masked Search Less Than or Equal	Skip NI if (U) AND (R2) ≤ (A) AND (R2), else repeat

Table C-2. Instruction Repertoire (continued)

Function Code (Octal)		Mnemonic	Instruction	Description
f	j			
		MSNG	Masked Search Not Greater	
71	03	MSG	Masked Search Greater	Skip NI if (U) AND (R2) > (A) AND (R2), else repeat
71	04	MSW	Masked Search Within Range	Skip NI if (A) AND (R2) < (U) AND (R2) ≤ (A+1) AND (R2), else repeat
71	05	MSNW	Masked Search Not Within Range	Skip NI if (U) AND (R2) ≤ (A) AND (R2) or (U) AND (R2) > (A+1) AND (R2), else repeat
71	06	MASL	Masked Alphanumeric Search Less Than or Equal	Skip NI if (U) AND (R2) ≤ (A) AND (R2), else repeat
71	07	MASG	Masked Alphanumeric Search Greater	Skip NI if (U) AND (R2) > (A) AND (R2), else repeat
71	10	DA	Double-Precision Fixed-Point Add	(A, A+1) + (U, U+1) → A, A+1
71	11	DAN	Double-Precision Fixed-Point Add Negative	(A, A+1) - (U, U+1) → A, A+1
71	12	DS	Double Store A	(A, A+1) → U, U+1
71	13	DL	Double Load A	(U, U+1) → A, A+1
71	14	DLN	Double Load Negative A	- (U, U+1) → A, A+1
71	15	DLM	Double Load Magnitude A	|(U, U+1)| → A, A+1
71	16	DJZ	Double-Precision Jump Zero	Jump to U if (A, A+1) = ± 0; go to NI if (A, A+1) ≠ ± 0
71	17	DTE	Double-Precision Test Equal	Skip NI if (U < U+1) = (A, A+1)
72	00	IMI	Initiate Maintenance Interrupt	Send Attention Interrupt to Maintenance Processor, if in Maintenance Mode, otherwise NO-OP

Table C-2. Instruction Repertoire (continued)

Function Code (Octal)		Mnemonic	Instruction	Description
f	j			
72	01	SLJ	Store Location and Jump	Relative P+1 → U_{17-0}; jump to U+1
72	02	JPS	Jump Positive and Shift	Jump to U if $(A)_{35} = 0$; go to NI if $(A)_{35} = 1$; always shift (A) left circularly one bit position
72	03	JNS	Jump Negative and Shift	Jump to U if $(A)_{35} = 1$; go to NI if $(A)_{35} = 0$; always shift (A) left circularly one bit position
72	04	AH	Add Halves	$(A)_{35-18} + (U)_{35-18}$; → $(A)_{35-18}$; $(A)_{17-0} + (U)_{17-0}$ → A_{17-0}
72	05	ANH	Add Negative Halves	$(A)_{35-18} - (U)_{35-18}$ → $(A)_{35-18}$; $(A)_{17-0} - (U)_{17-0}$ → A_{17-0}
72	06	AT	Add Thirds	$(A)_{35-24} + (U)_{35-24}$ → A_{35-24}; $(A)_{23-12} + (U)_{23-12}$ → A_{23-12}; $(A)_{11-0} + (U)_{11-0}$ → A_{11-0}
72	07	ANT	Add Negative Thirds	$(A)_{35-24} - (U)_{35-24}$ → A_{35-24}; $(A)_{23-12} - (U)_{23-12}$ → A_{23-12}; $(A)_{11-0} - (U)_{11-0}$ → A_{11-0}
72	10	EX	Execute	Execute the instruction at U
72	11	ER	Executive Request	Interrupt to MSR + 222_8
72	12	-	Invalid Code	Causes Invalid Code Fault interrupt to MSR + 221_8
72	13	PAIJ	Prevent All Interrupts and Jump	Prevent all interrupts and jump to U
72	14	-	Invalid Code	Causes Invalid Code Fault interrupt to MSR + 221_8
72	15	TRA	Test Relative Address	Used to determine whether a relative address is within a given relative addressing range
72	16	SRS	Store Register Set	Transfer GRS areas defined in Aa to storage starting at address U
72	17	LRS	Load Register Set	Transfer from storage starting at location U to areas defined in Aa
73	00	SSC	Single Shift Circular	Shift (A) right circularly U places

Table C-2. Instruction Repertoire (continued)

Function Code (Octal)		Mnemonic	Instruction	Description
f	j			
73	01	DSC	Double Shift Circular	Shift (A, A+1) right circularly U places
73	02	SSL	Single Shift Logical	Shift (A) right U places, zero fill
73	03	DSL	Double Shift Logical	Shift (A, A+1) right U places, zero fill
73	04	SSA	Single Shift Algebraic	Shift (A) right U places, sign fill
73	05	DSA	Double Shift Algebraic	Shift (A, A+1) right U places, sign fill
73	06	LSC	Load Shift and Count	(U) → A; shift (A) left circularly until $(A)_{35} \neq (A)_{34}$; number of shifts → A+1
73	07	DLSC	Double Load Shift and Count	(U, U+1) → A, A+1; shift (A, A+1) left circularly until $(A, A+1)_{71} \neq (A, A+1)_{70}$; number of shifts → A+2
73	10	LSSC	Left Single Shift Circular	Shift (A) left circularly U places
73	11	LDSC	Left Double Shift Circular	Shift (A, A+1) left circularly U places
73	12	LSSL	Left Single Shift Logical	Shift (A) left U places, zero fill
73	13	LDSL	Left Double Shift Logical	Shift (A, A+1) left U places, zero fill
73	14	a = 00–07	Invalid Code	Causes Invalid Instruction Fault interrupt to MSR $+221_8$
73	14	LDC a = 10	Load Dayclock	Replace dayclock register value with fixed storage value at start of next update cycle.
73	14	EDC a = 11	Enable Day Clock	Enable the internal dayclock of the processor.
73	14	DDC a = 12	Disable Day Clock	Disable the internal dayclock of the processor.

Table C-2. Instruction Repertoire (continued)

Function Code (Octal)		Mnemonic	Instruction	Description
f	j			
73	14	SDC a = 13	Select Day Clock	Select internal dayclock
73	14	MDA a = 14	Diagnostics	Generates A and A+1 operands, U operand, U+1 operand, Store results A, A+1 in GRS addresses 62 and 63.
73	14	MDB a = 15	Diagnostics	Generates A and A+1 operands, U operand, U+1 operand, Store results A, A+1 in GRS addresses 62 and 63.
73	14	a = 16,17	Diagnostics	Undefined, will NO-OP
73	15	SIL a = 00	Select Interrupt Locations	$(U)_{8-0} \rightarrow$ MSR
73	15	- a = 01	Invalid Code	Causes Invalid Instruction Fault interrupt to MSR + 221_8
73	15	LBRX a = 02	Load Breakpoint Register	Transfer operand to Breakpoint Register
73	15	LQT a = 03	Load Quantum Timer	Place full-word operand in Quantum Timer
73	15	IIIX a = 04	Initiate Interprocessor Interrupt	Interrupt processor specified by operand address value
73	15	SPID a = 05	Store Processor ID	Store: binary serial number in first third; 2-character Fieldata revision level in second third; processor features in the fifth sixth; processor number in last sixth of operand
73	15	RAT a = 06	Reset Auto-Recovery Timer	Reset auto-recovery timer in system transition unit
73	15	TAP a = 07	Toggle Auto-Recovery Path	Toggle path selection after each auto-recovery attempt

Table C-2. Instruction Repertoire (continued)

Function Code (Octal)		Mnemonic	Instruction	Description
f	j			
73	15	LB a = 10	Load Base	Place operand bits 0 through 17 in base value field of BDR specified by bits 33 and 34 of Xx
73	15	LL a = 11	Load Limits	Place operand bits 15 through 23 and 24 through 35 in BDR limits fields specified by Xx bits 33 and 34
73	15	LAE a = 12	Load Addressing Environment	Place the double-word operand in GRS location 046 and 047 and place the limits and base values of the four Bank Descriptors specified by this operand in four respective Bank Descriptor Registers
73	15	SQT a = 13	Store Quantum Time	Store Quantum Timer value at the operand address location. Executing this instruction has no effect on D29.
73	15	LD a = 14	Load Designator Register	Place full-word operand in Designator Register
73	15	SD a = 15	Store Designator Register	Store Designator Register contents at location specified by operand address
73	15	UR a = 16	User Return	(U + 1) → Designator Register; jump to address specified by $(U)_{23-0}$ using new register set
73	15	SSS a = 17	Store System Status	Store two system status words at the location specified by operand address
73	16	-	Invalid Code	Causes Invalid Instruction Fault interrupt to MSR $+221_8$
73	17	TS a = 00	Test and Set	If $(U)_{30} = 1$, Generate Test and Set interrupt; if $(U)_{30} = 0$, go to NI, if U ≥ 200, then $01_8 \rightarrow U_{35-30}$; $(U)_{29-0}$ unchanged
73	17	TSS a = 01	Test and Set and Skip	if $(U)_{30} = 1$, go to NI; if $(U)_{30} = 0$, skip NI, if U ≥ 200, then $01_8 \rightarrow U_{35-30}$; $(U)_{29-0}$ unchanged
73	17	TCS a = 02	Test and Clear and Skip	If $(U)_{30} = 0$, perform NI; if $(U)_{30} = 1$, skip NI, if U ≥ 200 clear $(U)_{35-30}$; $(U)_{29-0}$ unchanged

Table C-2. Instruction Repertoire (continued)

Function Code (Octal)		Mnemonic	Instruction	Description
f	j			
73	17	a = 03	Invalid Code	Causes Invalid Instruction Fault interrupt to MSR + 221_8
73	17	TSA a = 04	Test and Set Alternate	Test bit position 14; if $(U)_{14} = 1$, interrupt; if $(U)_{14} = 0$, take next instruction and set bits 00 through 14 to one.
73	17	TSSA a = 05	Test and Set and Skip Alternate	If $(U)_{14} = 1$, take next instruction; if $(U)_{14} = 0$, skip next instruction and set bits 00 through 14 to one.
73	17	a = 06–17	Invalid Code	Causes Invalid Instruction Operation Fault interrupt to MSR + 221_8
74	00	JZ	Jump Zero	Jump to U if (A) = ± 0 go to NI if (A) ≠ ± 0
74	01	JNZ	Jump Nonzero	Jump to U if (A) ≠ ± 0; go to NI if (A) = ± 0
74	02	JP	Jump Positive	Jump to U if $(A)_{35} = 0$; go to NI if $(A)_{35} = 1$
74	03	JN	Jump Negative	Jump to U if $(A)_{35} = 1$; go to NI if $(A)_{35} = 0$
74	04	J JK	Jump Jump Key	Jump to U if a = 0 or if a = set JUMP SELECT control circuit; go to NI if neither is true
74	05	HJ HKJ	Halt Jump Halt Keys and Jump	Stop if a = 0 or if [a-field AND set STOP SELECT control circuits] ≠ 0; on restart or continuation jump to U
74	06	NOP	No Operation	Proceed to next Instruction
74	07	AAIJ	Allow All Interrupts and Jump	Allow all interrupts and jump to U
74	10	JNB	Jump No Low Bit	Jump to U if $(A)_0 = 0$; go to NI if $(A)_0 = 1$
74	11	JB	Jump Low Bit	Jump to U if $(A)_0 = 1$; go to NI if $(A)_0 = 0$
74	12	JMGI	Jump Modifier Greater and Increment	Jump to U if $(Xa)_{17-0} > 0$; go to NI if $(Xa)_{17-0} \leqslant 0$; always $(Xa)_{17-0} + (Xa)_{35-18} \rightarrow Xa_{17-0}$
74	13	LMJ	Load Modifier and Jump	Relative P + 1 $\rightarrow (Xa)_{17-0}$; jump to U

Table C-2. Instruction Repertoire (continued)

Function Code (Octal)		Mnemonic	Instruction	Description
f	j			
74	14	JO a = 00	Jump Overflow	Jump to U if D1 = 1; go to NI if D1 = 0
74	14	JFU a = 01	Jump Floating Underflow	Jump to U if D21 = 1, clear D21; go to NI if D21 = 0
74	14	JFO a = 02	Jump Floating Overflow	Jump to U if D22 = 1, clear D22; go to NI if D22 = 0
74	14	JDF a = 03	Jump Divide Fault	Jump to U if D23 = 1, clear D23; go to NI if D23 = 0
74	14	a = 04–17	Invalid Code	Causes Invalid Instruction Fault interrupt to MSR + 221_8
74	15	JNO a = 00	Jump No Overflow	Jump to U if D1 = 0; go to NI if D1 = 1
74	15	JNFU a = 01	Jump No Floating Underflow	Jump to U if D21 = 0; go to NI if D21 = 1; clear D21
74	15	JNFO a = 02	Jump No Floating Overflow	Jump to U if D22 = 0; go to NI if D22 = 1; clear D22
74	15	JNDF a = 03	Jump No Divide Fault	Jump to U if D23 = 0; go to NI if D23 = 1; clear D23
74	15	a = 04–17	Invalid Code	Causes Invalid Instruction Fault interrupt to MSR + 221_8
74	16	JC	Jump Carry	Jump to U if D0 = 1; go to NI if D0 = 0
74	17	JNC	Jump No Carry	Jump to U if D0 = 0; go to NI if D0 = 1
75	00	–	Invalid Code	Causes IOU to return a condition code of 3 to the CPU, indicating instruction not available
75	01	SIOF	Start I/O Fast Release	Initiates operation on subchannel specified by bit 00 through 15 of CAW

Table C-2. Instruction Repertoire (continued)

Function Code (Octal) f	Function Code (Octal) j	Mnemonic	Instruction	Description
75	02	-	Invalid Code	Causes IOU to return a condition code of 3 to the CPU, indicating instruction not available
75	03	TSC	Test Subchannel	Interrogates the channel and subchannel
75	04	HDV	Halt Device	Terminates current operation on channel and subchannel
75	05	HCH	Halt Channel	Terminates current operation on channel
75	06,07	-	Invalid Code	Causes Invalid Instruction Fault interrupt to MSR + 221_8
75	10	LCR	Load Channel Register	Load the interrupt mask register or load the channel base register
75	11	LTCW	Load Control Words	Loads the status table subchannel
75	12-17	-	Invalid Code	Causes Invalid Instruction Fault interrupt to MSR + 221_8
76	00	FA	Floating Add	(A) + (U) → A; RESIDUE → A+1 if D17 = 1
76	01	FAN	Floating Add Negative	(A) - (U) → A; RESIDUE → A+1 if D17 = 1
76	02	FM	Floating Multiply	(A) x (U) → A (and A+1 if D17 = 1)
76	03	FD	Floating Divide	(A) divided by (U) → A; REMAINDER → A+1 if D17 = 1
76	04	LUF	Load and Unpack Floating	$\lvert (U) \rvert_{34-27} \rightarrow A_{7-0}$, zero fill $(U)_{26-00} \rightarrow A+1_{26-00}$, sign fill $(U)_{35} \rightarrow A+1_{35}$
76	05	LCF	Load and Convert to Floating	$(U)_{35} \rightarrow A+1_{35}$, [NORMALIZED $(U)]_{26-0} \rightarrow A+1_{26-0}$; if $(U)_{35} = 0$, $(A)_{7-0}$ ± NORMALIZING COUNT → $A+1_{34-27}$; if $(U)_{35} = 1$, ones complement of [$(A)_{7-0}$ ± NORMALIZING COUNT] → $A+1_{34-27}$
76	06	MCDU	Magnitude of Characteristic Difference to Upper	$\lvert \lvert (A) \rvert_{35-27} - \lvert (U) \rvert_{35-27} \rvert \rightarrow A+1_{8-0}$; zeros → $A+1_{35-9}$

Table C-2. Instruction Repertoire (continued)

Function Code (Octal)		Mnemonic	Instruction	Description
f	j			
76	07	CDU	Characteristic Difference to Upper	$\lvert (A) \rvert_{35-27} - \lvert (U) \rvert_{35-27} \rightarrow A+1_{8-0}$; sign bits $\rightarrow A+1_{35-9}$
76	10	DFA	Double-Precision Floating Add	(A, A+1) + (U, U+1) → A, A+1
76	11	DFAN	Double-Precision Floating Add Negative	(A, A+1) - (U, U+1) → A, A+1
76	12	DFM	Double-Precision Floating Multiply	(A, A+1) x (U, U+1) → A, A+1
76	13	DFD	Double-Precision Floating Divide	(A, A+1) divided by (U, U+1) → A, A+1
76	14	DFU	Double Load and Unpack Floating	$\lvert (U, U+1) \rvert_{70-60} \rightarrow A_{10-0}$, zero fill; $(U, U+1)_{59-36} \rightarrow A+1_{23-0}$, sign fill; $(U, U+1)_{35-0} \rightarrow A+2$
76	15	DFP, DLCF	Double Load and Convert to Floating	$(U)_{35} \rightarrow A+1_{35}$; [NORMALIZED $(U, U+1)]_{59-0} \rightarrow A+1_{23-0}$ and A+2; if $(U)_{35}$, $(A)_{10-0}$ ± NORMALIZING COUNT → $A+1_{34-24}$; if $(U)_{35}$ = 1, ones complement of [$(A)_{10-0}$ ± NORMALIZING COUNT] → $A+1_{34-24}$
76	16	FEL	Floating Expand and Load	If $(U)_{35} = 0$; $(U)_{35-27} + 1600_8 \rightarrow A_{35-24}$. If $(U)_{35} = 1$; $(U)_{35-27} - 1600_8 \rightarrow A_{35-24}$ $(U)_{26-3} \rightarrow A_{23-0}$; $(U)_{2-0} \rightarrow A+1_{35-33}$; $(U)_{35} \rightarrow A+1_{32-0}$
76	17	FCL	Floating Compress and Load	If $(U)_{35} = 0$; $(U)_{35-24} - 1600_8 \rightarrow A_{35-27}$. If $(U)_{35} = 1$; $(U)_{35-24} + 1600_8 \rightarrow A_{35-27}$ $(U)_{23-0} \rightarrow A_{26-3}$; $(U+1)_{35-33} \rightarrow A_{2-0}$
77	0-17	-	Invalid Code	Causes Invalid Instruction Fault interrupt to MSR + 221_8

Table C-3. Octal vs Mnemonic Instruction Code

First Digit	Function Code – Second Digit							
	0	1	2	3	4	5	6	7
0		S SA	SN SNA	SM SMA	S SR	(1)	S SX	(see below)
1	L LA	LN LNA	LM LMA	LNMA	A AA	AN ANA	AM AMA	ANM ANMA
2	AU	ANU	BT	L LR	A AX	AN ANX	LXM	L LX
3	MI	MSI	MF	Bytes (see below)	DI	DSF	DF	Bytes (see below)
4	OR	XOR	AND	MLU	TEP	TOP	LXI	TLEM TNGM
5	TZ	TNZ	TE	TNE	TLE TNG	TG	TW	TNW
6	TP	TN	SE	SNE	SLE SNG	SG	SW	SNW
7	JGD	See Below						

Funct. Code	First j Digit	Second j Digit							
		0	1	2	3	4	5	6	7
07	0 1			LDJ	LIJ	LPD	SPD		LBJ
33	0 1	BM BI	BMT BDI	BTT IB	BTC DIB	BC BF	BPD BDF	PDB FB	EDIT DFB
37	0 1	QB	BQ	QHB	BHQ	QDB	DBQ	BA	BAN
71	0 1	MSE DA	MSNE DAN	(2) DS	MSG DL	MSW DLN	MSNW DLM	MASL DJZ	MASG DTE
72	0 1	IMI EX	SLJ ER	JPS	JNS PAIJ	AH	ANH TRA	AT SRS	ANT LRS
73	0 1	SSC LSSC	DSC LDSC	SSL LSSL	DSL LDSL	SSA (3)	DSA (4)	LSC	DLSC (5)
74	0 1	JZ JNB	JNZ JB	JP JMGI	JN LMJ	J,JK (6)	HJ,HKJ (7)	NOP JC	AAIJ JNC
75	0 1	LCR	SIOF LTCW		TSC	HDV	HCH		
76	0 1	FA DFA	FAN DFAN	FM DFM	FD DFD	LUF DFU	LCF (8)	MCDU FEL	CDU FCL

NOTES:

1. *SZ, SNZ, SP1, SN1, SFS, SFZ, SAS, SAZ, INC, DEC, INC2, DEC2, ENZ*
2. *MSLE, MSNG*
3. *LDC, EDC, DDC, SDC, MDA, MDB*

4. *SIL, LBRX, LQT, IIIX, SPID, RAT, TAP, LB, LL, LAE, SQT, LD, SD, UR, SSS*

5. *TS, TSS, TCS, TSA, TSSA*

6. *JO, JFU, JFO, JDF*

7. *JNO, JNFU, JNFO, JNDF*

8. *DFP, DLCF*

Appendix D. Code Conversions

D.1. ASCII and Fieldata Code Conversion Tables

Codes, which also represent collating sequence, are given in octal in Tables D-1 and D-2.

ASCII codes from 00_8 to 37_8 are used for communications. They are format, separator, and control characters. These are not converted into Fieldata.

The ASCII symbols represented by codes 40_8 to 137_8 are converted into the identical Fieldata symbols, except that the quotation marks symbol (42_8) is converted into a lozenge (76_8), the circumflex (136_8) is converted into a delta (04_8), underscore (137_8) is converted into a not equal sign (77_8).

There are no remaining unique Fieldata symbols into which to convert the balance of the ASCII symbols, represented by codes 140_8 to 177_8, (these codes are shown boxed in Table D-2), so most of these codes are "folded" over codes 100_8 to 137_8 (by clearing bit 5, which amounts to subtracting 40_8). This means that ASCII codes 101_8 (A) and 141_8 (a), for example, are both translated as if they were code 101_8 (converted to Fieldata 06_8 for A). Two exceptions to this general rule are the ASCII opening brace (173_8) and closing brace (175_8) which are converted to Fieldata question mark (54_8) and exclamation point (55_8), respectively, to satisfy overpunch sign considerations. The Operating System folds all codes from 140_8 to 177_8.

Table D-1. Fieldata to ASCII Code Conversion

Fieldata Code (Octal)	Fieldata 80-Column Card Code	Symbol	ASCII	
			Octal Code	Symbol
00	7-8	@	100	@
01	12-5-8	[	133	[
02	11-5-8	]	135	]
03	12-7-8	#	43	#
04	11-7-8	Δ	136	^
05	(blank)	(space)	40	(space)
06	12-1	A	101	A
07	12-2	B	102	B
10	12-3	C	103	C
11	12-4	D	104	D
12	12-5	E	105	E
13	12-6	F	106	F
14	12-7	G	107	G
15	12-8	H	110	H
16	12-9	I	111	I
17	11-1	J	112	J
20	11-2	K	113	K
21	11-3	L	114	L
22	11-4	M	115	M
23	11-5	N	116	N
24	11-6	O	117	O
25	11-7	P	120	P
26	11-8	Q	121	Q
27	11-9	R	122	R
30	0-2	S	123	S
31	0-3	T	124	T
32	0-4	U	125	U
33	0-5	V	126	V
34	0-6	W	127	W
35	0-7	X	130	X
36	0-8	Y	131	Y
37	0-9	Z	132	Z
40	12-4-8	)	51	)
41	11	- (minus)	55	- (minus)
42	12	+	53	+
43	12-6-8	<	74	<
44	3-8	=	75	=
45	6-8	>	76	>

Table D–1. Fieldata to ASCII Code Conversion (continued)

Fieldata Code (Octal)	Fieldata 80–Column Card Code	Symbol	ASCII	
			Octal Code	Symbol
46	2–8	&	46	&
47	11–3–8	$	44	$
50	11–4–8	*	52	*
51	0–4–8	(	50	(
52	0–5–8	%	45	%
53	5–8	: (colon)	72	: (colon)
54	12–0	?	77	?
55	11–0	!	41	!
56	0–3–8	,(comma)	54	,(comma)
57	0–6–8	\	134	\
60	0	0	60	0
61	1	1	61	1
62	2	2	62	2
63	3	3	63	3
64	4	4	64	4
65	5	5	65	5
66	6	6	66	6
67	7	7	67	7
70	8	8	70	8
71	9	9	71	9
72	4–8	'(apostrophe)	47	'(apostrophe)
73	11–6–8	;	73	;
74	0–1	/	57	/
75	12–3–8	.(period)	56	.(period)
76	0–7–8	¤	42	"
77	0–2–8	≠ or stop	137	_

Table D–2. ASCII to Fieldata Code Conversion

ASCII		System Console			Fieldata	
Octal Code	Symbol	Keyboard Symbol	CRT Symbol	Incremental Printer Symbol	Octal Code	Symbol
40	SP	(space bar)	(space)	(space)	05	(space)
41	!	!	!	!	55	!
42	"	"	"	"	76	¤
43	#	#	#	#	03	#
44	$	$	$	$	47	$
45	%	%	%	%	52	%
46	&	&	&	&	46	&
47	' (apos.)	' (apos.)	' (apos.)	' (apos.)	72	' (apos.)
50	(	(	(	(	51	(
51	)	)	)	)	40	)
52	*	*	*	*	50	*
53	+	+	+	+	42	+
54	, (comma)	, (comma)	, (comma)	, (comma)	56	, (comma)
55	- (minus)	- (minus)	- (minus)	- (minus)	41	- (minus)
56	. (period)	. (period)	. (period)	. (period)	75	. (period)
57	/	/	/	/	74	/
60	0	0	0	0	60	0
61	1	1	1	1	61	1
62	2	2	2	2	62	2
63	3	3	3	3	63	3
64	4	4	4	4	64	4
65	5	5	5	5	65	5
66	6	6	6	6	66	6
67	7	7	7	7	67	7
70	8	8	8	8	70	8
71	9	9	9	9	71	9
72	: (colon)	: (colon)	: (colon)	: (colon)	53	: (colon)
73	;	;	;	;	73	;
74	<	<	<	<	43	<
75	=	=	=	=	44	=
76	>	>	>	>	45	>
77	?	?	?	?	54	?
100	@	@	@	@	00	@
101	A	A	A	A	06	A
102	B	B	B	B	07	B
103	C	C	C	C	10	C
104	D	D	D	D	11	D
105	E	E	E	E	12	E
106	F	F	F	F	13	F
107	G	G	G	G	14	G

Table D–2. ASCII to Fieldata Code Conversion (continued)

ASCII		System Console			Fieldata	
Octal Code	Symbol	Keyboard Symbol	CRT Symbol	Incremental Printer Symbol	Octal Code	Symbol
110	H	H	H	H	15	H
111	I	I	I	I	16	I
112	J	J	J	J	17	J
113	K	K	K	K	20	K
114	L	L	L	L	21	L
115	M	M	M	M	22	M
116	N	N	N	N	23	N
117	O	O	O	O	24	O
120	P	P	P	P	25	P
121	Q	Q	Q	Q	26	Q
122	R	R	R	R	27	R
123	S	S	S	S	30	S
124	T	T	T	T	31	T
125	U	U	U	U	32	U
126	V	V	V	V	33	V
127	W	W	W	W	34	W
130	X	X	X	X	35	X
131	Y	Y	Y	Y	36	Y
132	Z	Z	Z	Z	37	Z
133	[	[	[	[	01	[
134	\	\	\	\	57	\
135	]	]	]	]	60	]
136	^	^	^	^	04	Δ
137	—	—	—	—	77	≠
140				@	00	@
141	a*	A**	a*	A**	06	A**
through	through	through	through	through	through	through
172	z*	Z**	z*	Z**	37	Z**
173	|	|	|	[	54	?
174	:	:	:	\	57	\
175	}	}	}	]	55	!
176	~	~	~	^	04	Δ
177	DEL	(no key)	⁄.	_	77	≠

* Lowercase alphabet
** Uppercase alphabet

D.2. Special Characters in ASCII

The special characters in ASCII are:

SP designates space, which is normally nonprinting.
DEL designates delete, and has a code of all 1 bits. This code eliminates the previous character – even on paper tape or other nonerasable medium.

Definitions of the 32 ASCII control characters, codes 00_8 to 37_8:

00 NUL Null – all zero character which may serve as time fill
01 SOH Start of heading
02 STX Start of text
03 ETX End of text
04 EOT End of transmission
05 ENQ Enquire – "Who Are You?"
06 ACK Acknowledge – "Yes"
07 BEL Bell – human attention required
10 BS Backspace
11 HT Horizontal tabulation
12 LF Line feed
13 VT Vertical tabulation
14 FF Form feed
15 CR Carriage return
} format effectors for printing or punching
16 SO Shift out – nonstandard code follows
17 SI Shift in – return to standard code
20 DLE Data link escape – change limited data communication control
21 DC1
22 DC2
23 DC3
24 DC4
} Device control for turning on or off auxiliary devices
25 NAK Negative acknowledge – "No"
26 SYN Synchronous idle – from which to achieve synchronism
27 EBT End of transmission block – relates to physical communication block
30 CAN Cancel previous data
31 EM End of medium – end of used, or wanted, portion of information
32 SUB Substitute character for one in error
33 ESC Escape – for code extension – change some character interpretations
34 FS File separator
35 GS Group separator
36 RS Record separator
37 US Unit separator
} These information separators are ordered in descending hierarchy. They are followed by ASCII 40_8 (space), which can also be thought of as a word separator.

Appendix E. Storage Configurations

E.1. General

The 1100/80 Systems with 4x4 capability allow expansion to three or four processor configurations. These configurations require a system transition unit (STU) and a storage system having different characteristics than those used in systems whose maximum configuration cannot be expanded beyond two processors.

The maximum storage configuration consists of eight partitionable storage interface unit (SIU) segments and eight partitionable main storage unit (MSU) banks. With this number of components there is a wide variety of possible storage configurations. Illustrations and data presented in this Appendix provide a better understanding of the flexibility and restrictions of these configurations. The installation may use this information as a reference, as a guide to storage configuration options, as a guide to partitioning, and as an aid to reconfiguring for multiple applications or to recover after the loss of a storage component.

The 1100/80 Systems storage system consists of large capacity, low cost MSUs; interfaced by moderate capacity, high speed storage buffers located in the SIU. The high speed storage buffers are used to achieve increased performance from the relatively low speed MSUs. Each MSU can be divided into two separate banks. Each SIU contains from one to four logically independent storage buffers called segments. A main storage bank has a 2-port multimodule access (MMA) unit which allows access by two SIU segments and provides a means for common accessing of main storage.

All possible storage configurations are not shown in this subsection; however, any combination not shown can be derived from the information presented in this subsection.

E.2. Definition of Terms

Terms pertaining to the storage configurations are defined in the following listing.

Application	This term is used in the STU sense, i.e., it is an operable partition of the total configuration at a site.
Cluster	Standard system components are grouped into clusters 0 and 1. Cluster 0 components are central processor unit (CPU) 0, CPU 1, input/output unit (IOU) 0, IOU 1, SIU segments 0, 2, 4 and 6. Cluster 1 components are CPU 2, CPU 3, IOU 2, IOU 3 and SIU segments 1, 3, 5, and 7. MSUs are not cluster-oriented.

Failure	Failure of a storage component refers to a hardware condition such that the component is not usable in the application, i.e., it is offline or otherwise inaccessible.
Loss	Refers to either removal or failure of a component.
MSU	Main storage unit. A storage cabinet containing one or two banks.
MSU Bank (also Bank)	A portion of main storage which is partitionable at the STU.
Removal	Removal or addition of a storage component refers to a planned event.
Segment/bank	One SIU segment interfacing with one MSU bank. Applicable to single cluster system only.
Segment/cabinet	One SIU segment interfacing with one MSU cabinet. A cabinet contains two banks. Applicable to both single and double cluster systems.
SIU–Generated MSR	The system hardware allows the setting of module select register (MSR) to be hardware generated rather than manually set at the STU. This is not usable for bootstrap into SIU lower when unusable addresses exist in the lower half. MSR must then be set manually.
SIU Half (also Half)	This is two interleavable SIU segments (0 and 2, 4 and 6, 1 and 3, or 5 and 7). A half can exist, however, with only one of the SIU segments. An SIU half represents an address range, either those words above $40M_8$ (SIU upper) or below $40M_8$ (SIU lower).
SIU Segment (also Segment)	A unit of cache memory containing 4K words, partitionable at the STU, which may service one MSU which may have one or two MSU banks.
Unusable Addresses	It is possible to configure an application in which interleaved components are of unequal sizes, and addresses in one part are not satisfied by corresponding addresses in the other part of the interleave. This occurrence is referred to as unusable addresses.

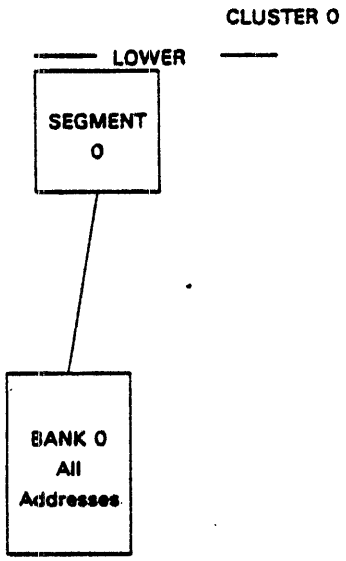
E.3. Address Interleave

The following examples illustrate the octal address distribution for various storage configurations. There are two basic options for storage configurations. The first is a segment/cabinet storage configuration with examples in E.3.1, and the second is a segment/bank storage configuration with examples in E.3.2.

The addresses shown in the MSU bank configurations are the lower absolute address bits, in octal, which represent the distribution of the storage addresses. Absolute addresses consist of 24 bits, labeled bit positions 23–0, in which bit 0 is the least significant. Address interleaving between the requesters and the SIU segments is controlled by the contents of bit position 2. Address interleaving between the SIU segments and the MSU banks is controlled by the contents of bit position 3. The contents of bit position 23 is used by the requesters to determine whether the request is sent to SIU upper or SIU lower. This divides the addressing range into addresses below $40M_8$ (SIU lower) and addresses above $40M_8$ (SIU upper). Note that there is no interleave between SIU halves, and that the address ranges do not have to be of equal size in two SIU halves.

E.3.1. Address Interleaving in Segment/Cabinet Storage Configurations

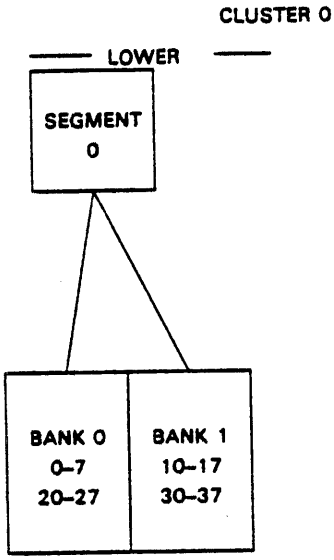
E.3.1.1. One Segment/One Bank



NOTES:

1. *No interleave.*
2. *Unusable addresses do not occur.*

E.3.1.2. One Segment/Two Banks

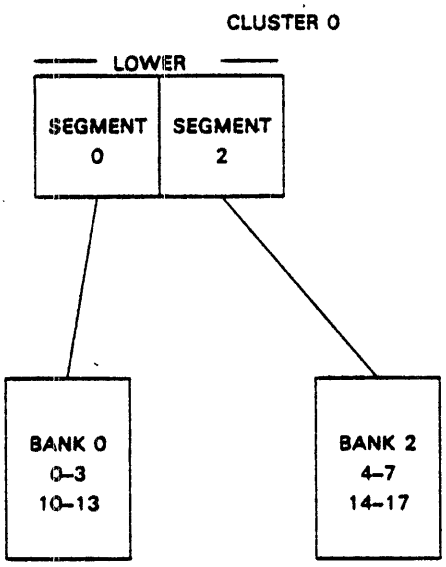


NOTES:

1. *Interleave between banks 0 and 1 on bit 3.*

2. *The banks must be equal in size or unusable addresses occur. If they are not equal in size and are in the lower half, MSR cannot be set to SIU generated for bootstrap into the lower half and would have to be manually set on the STU.*

E.3.1.3. Two Segments/Two Banks

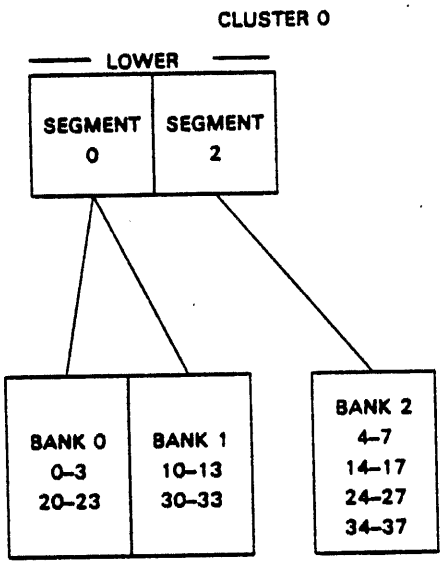


NOTES:

1. *Interleave between segments 0 and 2 on bit 2.*

2. *The banks must be equal in size or unusable addresses occur. If they are not equal in size and are in the lower half, MSR cannot be set to SIU generated for bootstrap into the lower half and would have to be manually set on the STU.*

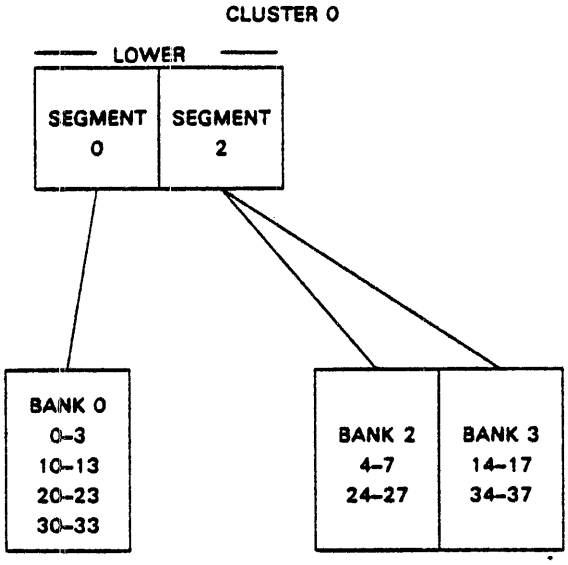
E.3.1.4. Two Segments/Three Banks – Basic



NOTES:

1. *Interleave between segments 0 and 2 on bit 2.*
2. *Interleave between banks 0 and 1 on bit 3.*
3. *The two banks connected to the same segment must be equal in size and the bank connected to the other segment must have a size equal to the total of the other two banks or unusable addresses occur. If unusable addresses occur and the banks are in the lower half, MSR cannot be set to SIU generated for bootstrap into the lower half and would have to be manually set on the STU.*

E.3.1.5. Two Segments/Three Banks – Alternate



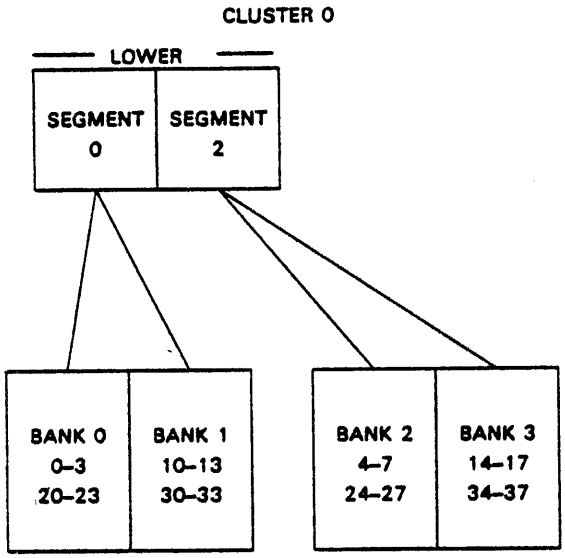
NOTES:

1. *Interleave between segments 0 and 2 on bit 2.*

2. *Interleave between banks 2 and 3 on bit 3.*

3. *The two banks connected to the same segment must be equal in size and the bank connected to the other segment must have a size equal to the total of the other two banks or unusable addresses occur. If unusable addresses occur and the banks are in the lower half, MSR cannot be set to SIU generated for bootstrap into the lower half and would have to be manually set on the STU.*

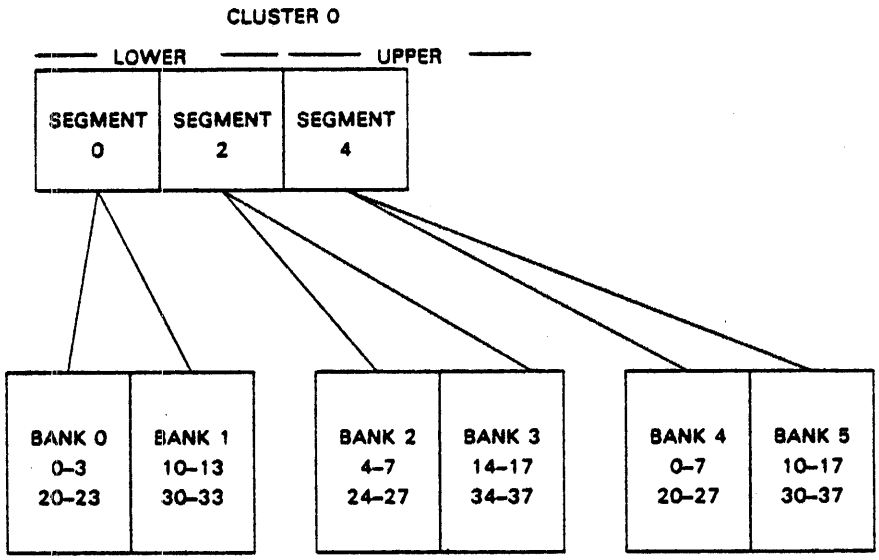
E.3.1.6. Two Segments/Four Banks



NOTES:

1. *Interleave between segments 0 and 2 on bit 2.*
2. *Interleave between banks 0 and 1, and 2 and 3 on bit 3.*
3. *All banks must be equal in size or unusable addresses occur. If unusable addresses occur and the banks are in the lower half, MSR cannot be set to SIU generated for bootstrap into the lower half and would have to be manually set on the STU.*

E.3.1.7. Three Segments/Six Banks



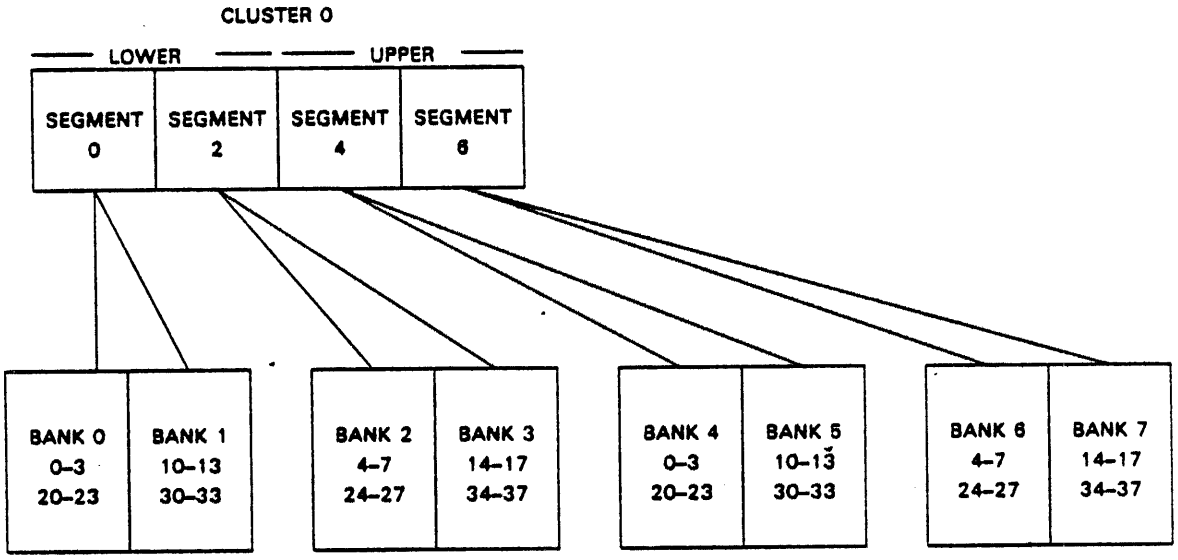
NOTES:

1. *Interleave between segments 0 and 2 on bit 2.*

2. *Interleave between banks 0 and 1, 2 and 3, and 4 and 5 on bit 3.*

3. *Banks 0, 1, 2, and 3 must be of equal size, and banks 4 and 5 must be of equal size or unusable addresses occur. It is acceptable for banks 4 and 5 to contain more or less storage than the other banks. If unusable addresses occur and the banks are in the lower half, MSR cannot be set to SIU generated for bootstrap into the lower half and would have to be manually set on the STU.*

E.3.1.8. Four Segments/Eight Banks



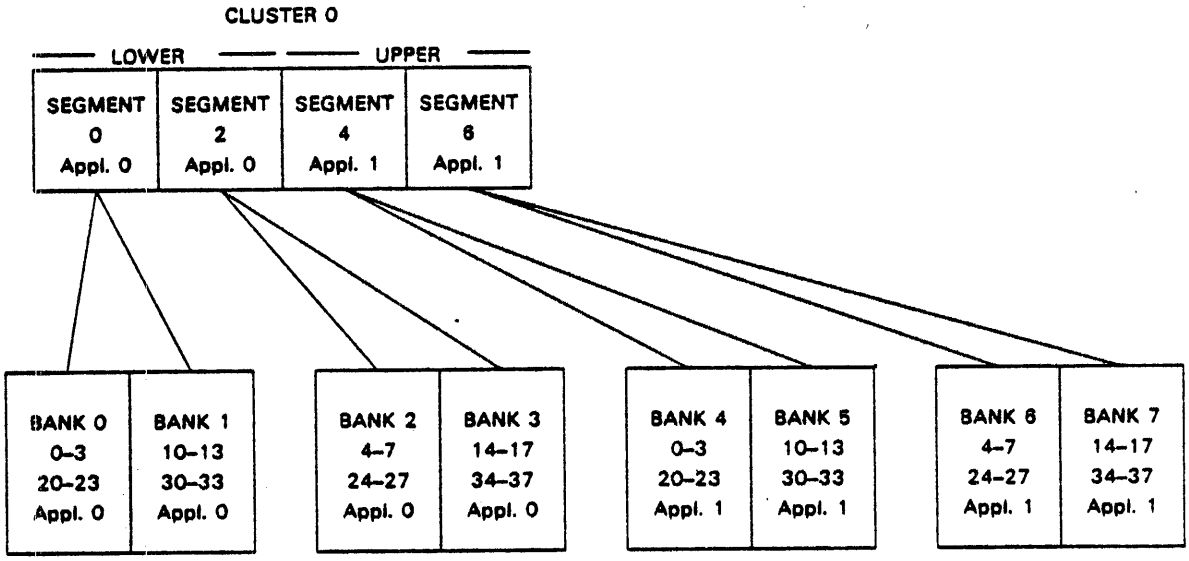
NOTES:

1. *Interleave between segments 0 and 2, and 4 and 6 on bit 2.*

2. *Interleave between banks 0 and 1, 2 and 3, 4 and 5, and 6 and 7 on bit 3.*

3. *Banks 0, 1, 2, and 3 must be of equal size and banks 4, 5, 6, and 7 must be of equal size or unusable addresses occur. If unusable addresses occur and the banks are in the lower half, MSR cannot be set to SIU generated for bootstrap into the lower half and would have to be manually set on the STU.*

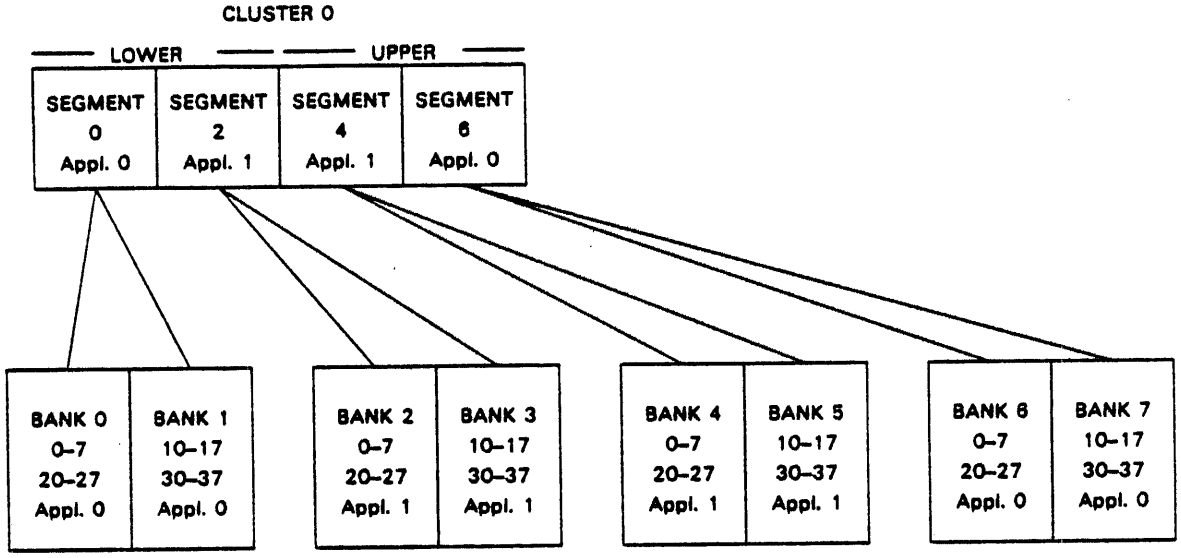
E.3.1.8.1. Partitioned by Storage Halves



NOTES:

1. *Interleave between segments 0 and 2 on bit 2 in Application 0.*
2. *Interleave between banks 0 and 1, and 2 and 3 on bit 3 in Application 0.*
3. *Interleave between segments 4 and 6 on bit 2 in Application 1.*
4. *Interleave between banks 4 and 5, and 6 and 7 on bit 3 in Application 1.*

E.3.1.8.2. Partitioned Across Storage Halves

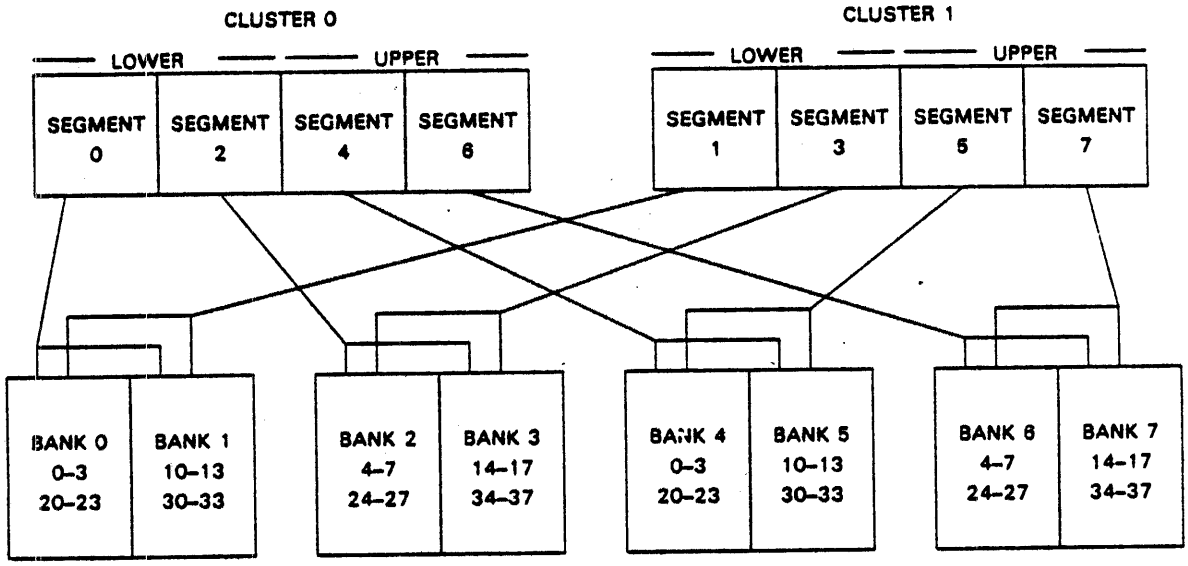


NOTES:

1. *Interleave between banks 0 and 1, and 6 and 7 on bit 3 in Application 0.*

2. *Interleave between banks 2 and 3, and 4 and 5 on bit 3 in Application 1.*

E.3.1.9. Eight Segments/Eight Banks



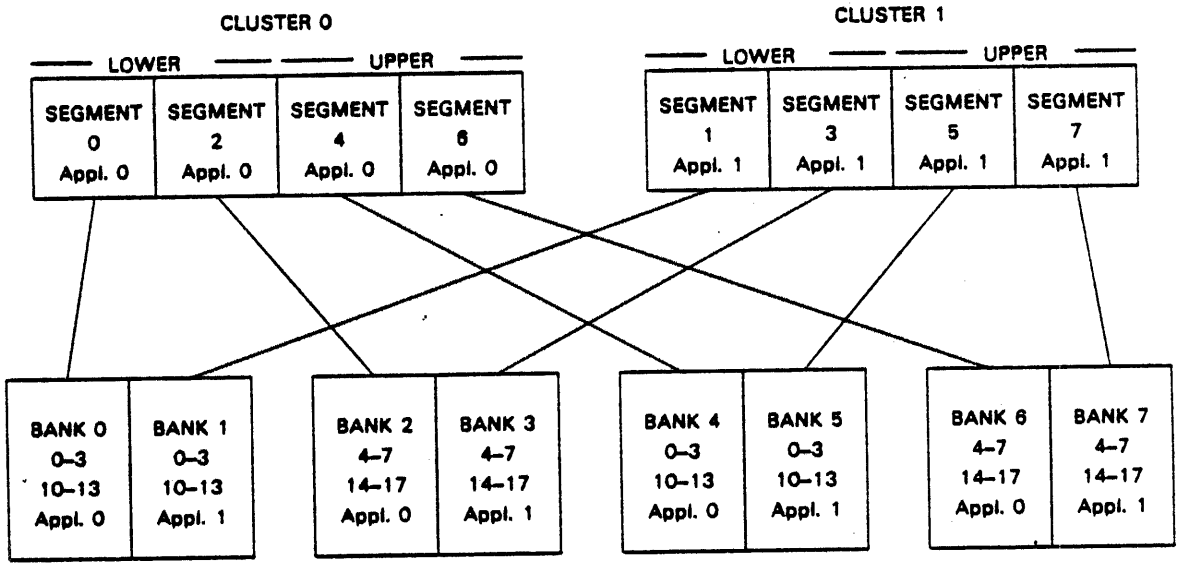
NOTES:

1. *Interleave between segments 0 and 2, 4 and 6, 1 and 3, and 5 and 7 on bit 2.*

2. *Interleave between banks 0 and 1, 2 and 3, 4 and 5, and 6 and 7 on bit 3.*

3. *Banks 0, 1, 2, and 3 must be of equal size and banks 4, 5, 6, and 7 must be of equal size or unusable addresses occur. If unusable addresses occur and the banks are in the lower half, MSR cannot be set to SIU generated for bootstrap into the lower half and would have to be manually set on the STU.*

The following configuration is an eight segment/eight bank configuration partitioned by cluster across MSU storage halves.



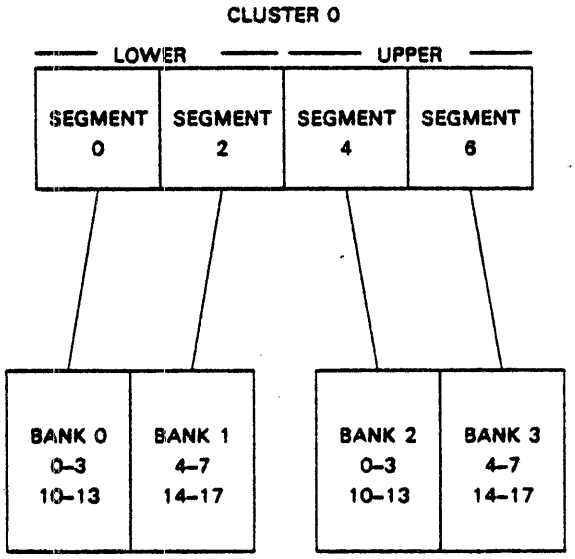
NOTES:

1. *Interleave between segments 0 and 2, and 4 and 6 on bit 2 in Application 0.*

2. *Interleave between segments 1 and 3, and 5 and 7 on bit 2 in Application 1.*

E.3.2. Address Interleaving in Segment/Bank Storage Configurations

E.3.2.1. Four Segments/Four Banks

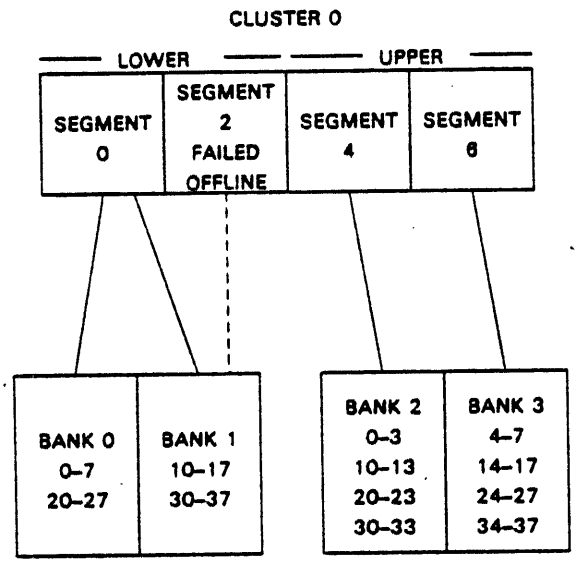


NOTES:

1. *Interleave between segments 0 and 2, and 4 and 6 on bit 2.*

2. *Banks 0 and 1 must be of equal size and banks 2 and 3 must be of equal size or unusable addresses occur. If they are not equal in size and are in the lower half, MSR cannot be set to SIU generated for bootstrap into the lower half and would have to be manually set on the STU.*

E.3.2.2. Degraded Mode – Failed Segment

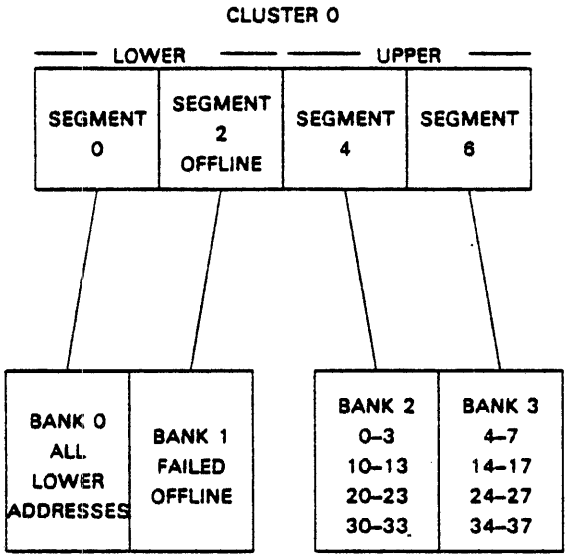


NOTES:

1. *Interleave between segments 4 and 6 on bit 2.*

2. *Interleave between banks 0 and 1 on bit 3.*

E.3.2.3. Degraded Mode – Failed Bank



NOTES:

1. *Interleave between segments 4 and 6 on bit 2.*

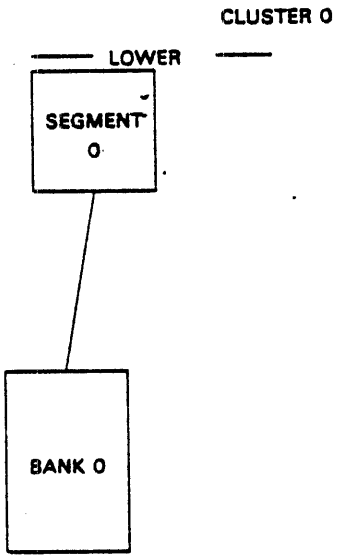
2. *Unusable addresses do not occur.*

E.4. Segment/Cabinet Storage Configurations

The following examples illustrate some of the possible segment/cabinet storage configurations. These configurations are not, in any way, being recommended over other usable configurations. They are configurations that could be used in a situation where a configuration is needed for partitioned operation or for degraded–mode operation. The data in this section provides a better understanding of system storage configuration.

E.4.1. One–Segment Configurations

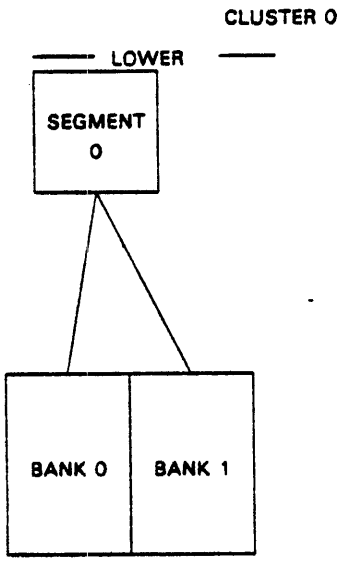
E.4.1.1. One Segment/One Bank



NOTES:

1. *No interleave.*
2. *Unusable addresses do not occur.*
3. *Loss of a segment or bank results in loss of the application. No reboot options.*

E.4.1.2. One Segment/Two Banks

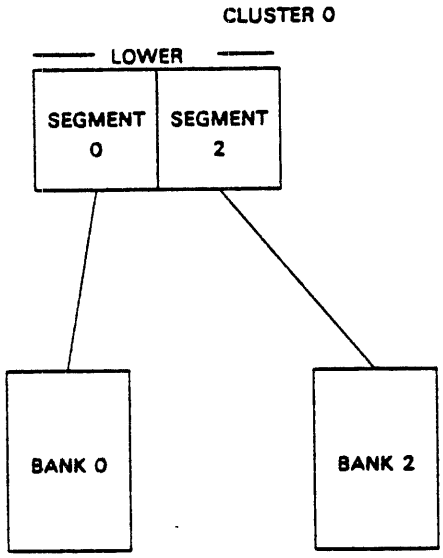


NOTES:

1. *Interleave between banks 0 and 1 on bit 3.*
2. *The banks must be equal in size or unusable addresses occur. If they are not equal in size and are in the lower half, MSR cannot be set to SIU generated for bootstrap into the lower half and would have to be manually set on the STU.*
3. *Loss of a segment results in loss of the application. No reboot options.*
4. *Loss of a bank results in bringing down the application. The application may be rebooted with the remaining bank.*

E.4.2. Two-Segment Configurations

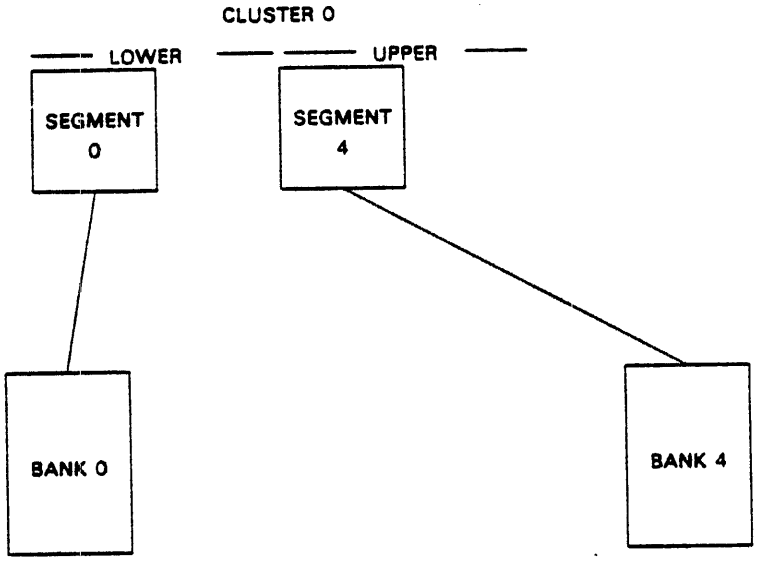
E.4.2.1. Two-Segments/Two Banks - Basic



NOTES:

1. *Interleave between segments 0 and 2 on bit 2.*
2. *The banks must be equal in size or unusable addresses occur. If they are not equal in size and are in the lower half, MSR cannot be set to SIU generated for bootstrap into the lower half and would have to be manually set on the STU.*
3. *Loss of a segment results in bringing down the application. The application may be rebooted with the remaining segment.*
4. *Loss of bank results in bringing down the application and loss of the bank's segment. The application may be rebooted with the remaining bank and segment.*

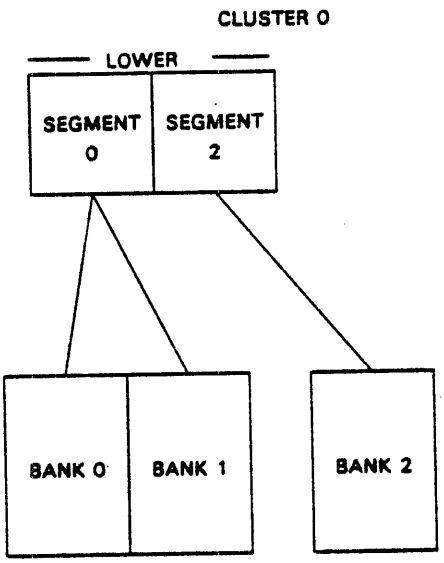
E.4.2.2. Two Segments/Two Banks – Alternate



NOTES:

1. *No interleave.*
2. *Unusable addresses do not occur.*
3. *Loss of a segment or bank in a half which contains the resident EXEC brings the application down. If the resident EXEC is restricted to one half of storage, removal of a segment or bank in the other half does not necessitate a reboot, but a failure in this other half will probably cause a reboot.*
4. *The application may be rebooted with the segment in the other half.*
5. *For loss of a bank, the application may be rebooted with the bank and segment in the unaffected half.*

E.4.2.3. Two Segments/Three Banks – Basic



NOTES:

1. *Interleave between segments 0 and 2 on bit 2.*

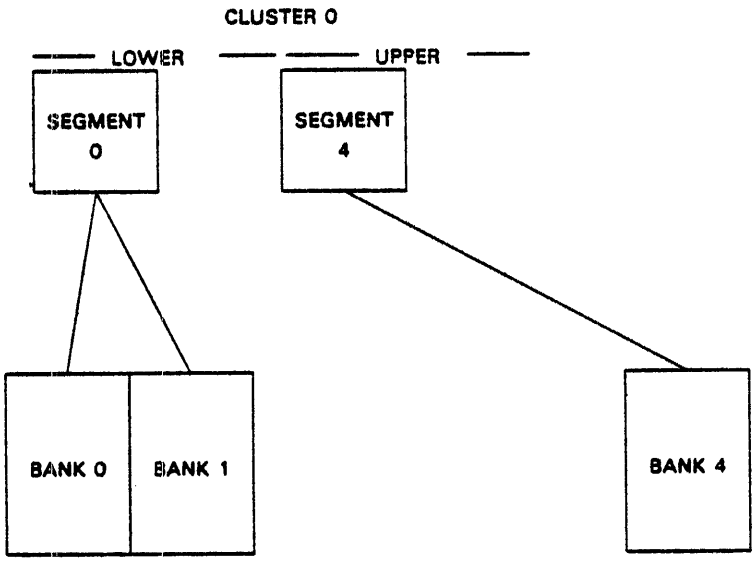
2. *Interleave between banks 0 and 1 on bit 3.*

3. *The two banks connected to the same segment must be equal in size and the bank connected to the other segment must have a size equal to the total of the other two banks or unusable addresses occur. If unusable addresses occur and the banks are in the lower half, MSR cannot be set to SIU generated for bootstrap into the lower half and would have to be manually set on the STU.*

4. *Loss of a segment results in bringing down the application. The application may be rebooted with the remaining segment.*

5. *Loss of a bank results in bringing down the application. If the bank is the only one connected to a segment, the application may be rebooted with the other segment and its banks. If the bank was one of two connected to a segment, the application may be rebooted with both segments and the remaining two banks.*

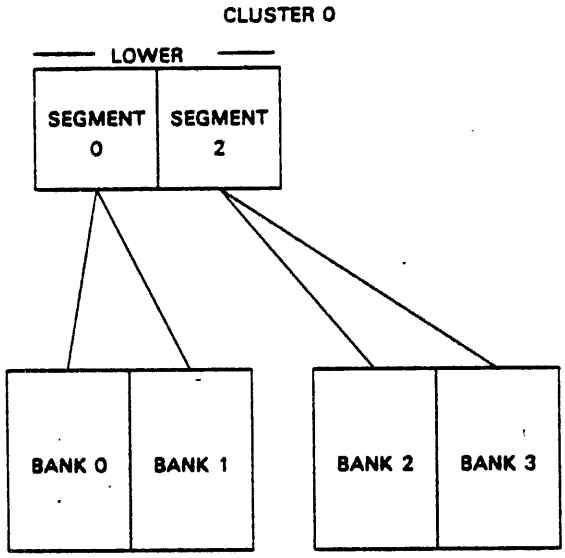
E.4.2.4. Two Segments/Three Banks – Alternate



NOTES:

1. *Interleave between banks 0 and 1 on bit 3.*
2. *Unusable addresses do not occur for one bank in a half.*
3. *For two banks in a half, the banks must be equal in size or unusable addresses occur. If they are not equal in size and are in the lower half, MSR cannot be set to SIU generated for bootstrap into the lower half and would have to be manually set on the STU.*
4. *Loss of a segment or bank in a half which contains the resident EXEC brings the application down. If the resident EXEC is restricted to one half of storage, removal of a segment or bank in the other half does not necessitate a reboot, but a failure in this other half will probably cause a reboot.*
5. *The application may be rebooted with the segment in the other half.*
6. *If a bank lost is the only one connected to a segment, the application may be rebooted with the other segment and its banks. If the bank was one of two connected to a segment, the application may be rebooted with both segments and the remaining two banks.*

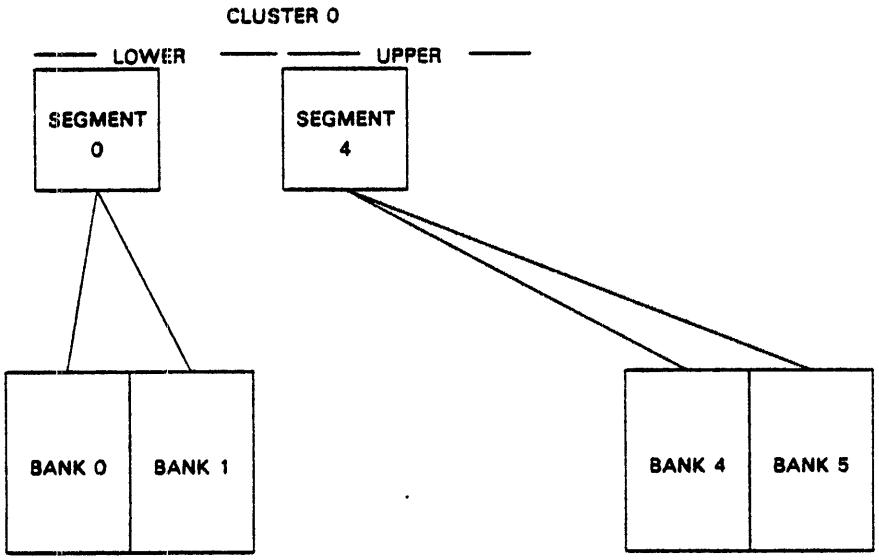
E.4.2.5. Two Segments/Four Banks – Basic



NOTES:

1. *Interleave between segments 0 and 2 on bit 2.*
2. *Interleave between banks 0 and 1, and 2 and 3 on bit 3.*
3. *All banks must be equal in size or unusable addresses occur. If unusable addresses occur and the banks are in the lower half, MSR cannot be set to SIU generated for bootstrap into the lower half and would have to be manually set on the STU.*
4. *Loss of a segment results in bringing down the application. The application may be rebooted with the remaining segment.*
5. *Loss of a bank results in bringing down the application. The application may be rebooted with the two segments and remaining three banks (see E.4.2.3, Note 3) or with two segments and two banks.*

E.4.2.6. Two Segments/Four Banks – Alternate



NOTES:

1. *Interleave between banks 0 and 1, and 4 and 5 on bit 3.*

2. *Banks 0 and 1 must be of equal size and banks 4 and 5 must be of equal size or unusable addresses occur. If they are not equal in size and are in the lower half, MSR cannot be set to SIU generated for bootstrap into the lower half and would have to be manually set on the STU.*

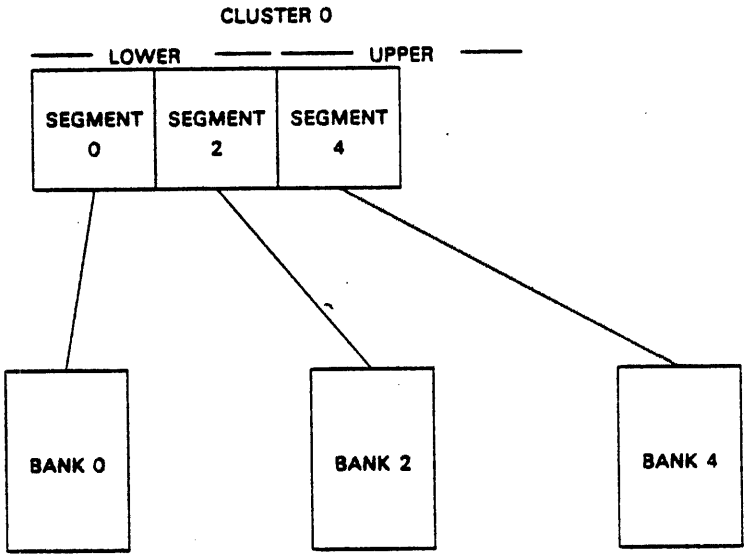
3. *Loss of a segment or bank in a half which contains the resident EXEC brings the application down. If the resident EXEC is restricted to one half of storage, removal of a segment or bank in the other half does not necessitate a reboot, but a failure in this other half will probably cause a reboot.*

4. *The application may be rebooted with the segment in the other half.*

5. *For loss of a bank, the application may be rebooted with two segments and the remaining three banks.*

E.4.3. Three–Segment Configurations

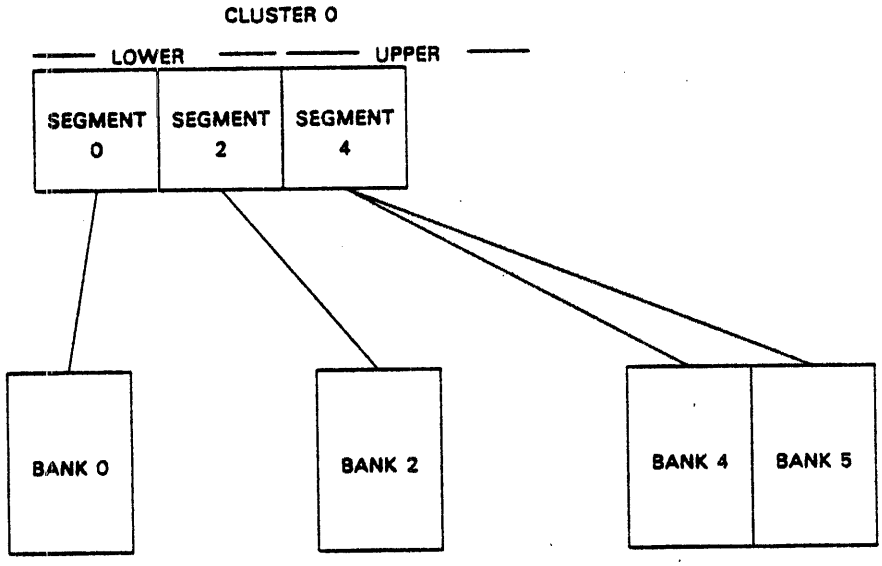
E.4.3.1. Three Segments/Three Banks



NOTES:

1. *Interleave between segments 0 and 2 on bit 2.*
2. *Unusable addresses do not occur for one bank in a half.*
3. *For two banks in a half, the banks must be equal in size or unusable addresses occur. If they are not equal in size and are in the lower half, MSR cannot be set to SIU generated for bootstrap into the lower half and would have to be manually set on the STU.*
4. *Loss of a segment or bank in a half which contains the resident EXEC brings the application down. If the resident EXEC is restricted to one half of storage, removal of a segment or bank in the other half does not necessitate a reboot, but a failure in this other half will probably cause a reboot.*
5. *The application may be rebooted with the remaining segments. If a two segment half was downed dynamically, it may be reentered into the application with the one remaining segment.*
6. *For loss of a bank, the application may be rebooted with the remaining two banks and their segments.*

E.4.3.2. Three Segments/Four Banks



NOTES:

1. *Interleave between segments 0 and 2 on bit 2.*

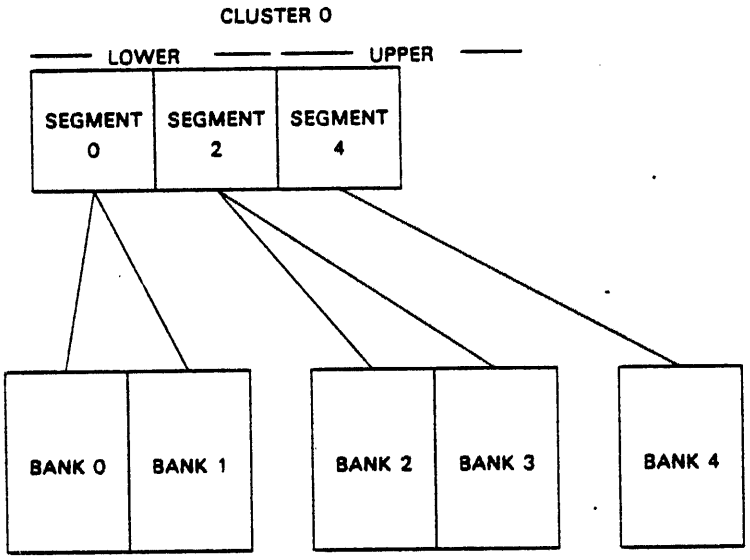
2. *Interleave between banks 4 and 5 on bit 3.*

3. *Banks 0 and 2 must be of equal size and banks 4 and 5 must be of equal size or unusable addresses occur. If they are not equal in size and are in the lower half, MSR cannot be set to SIU generated for bootstrap into the lower half and would have to be manually set on the STU.*

4. *For two banks in each half, loss of a bank in a half which contains the resident EXEC brings the application down. If the resident EXEC is restricted to one half of storage, removal of a bank in the other half does not necessitate a reboot, but a failure in this other half will probably cause a reboot.*

5. *If the bank lost was the only one connected to a segment, the application may be rebooted with the remaining segment and bank in that half and the unaffected half. If the bank lost was one of two connected to a segment, the application may be rebooted with the remaining three banks and three segments.*

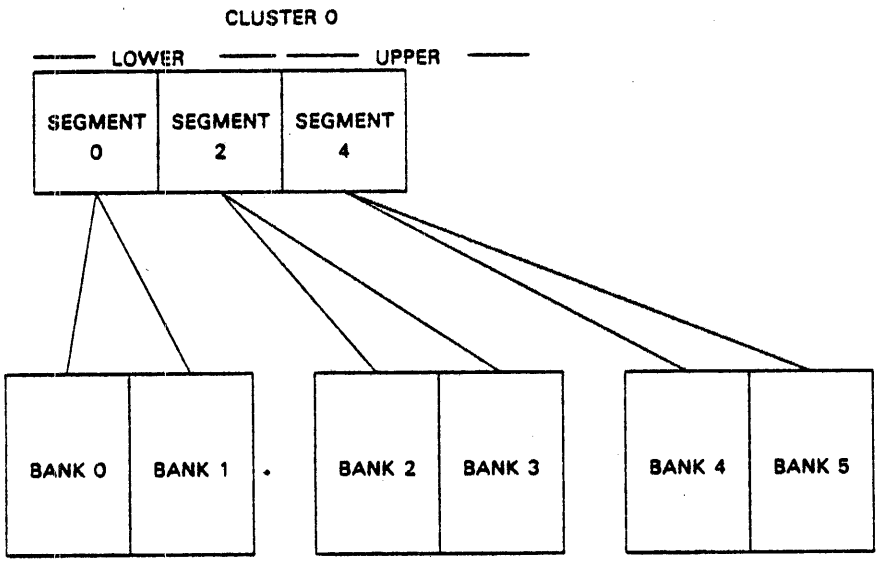
E.4.3.3. Three Segments/Five Banks



NOTES:

1. *Interleave between segments 0 and 2 on bit 2.*
2. *Interleave between banks 0 and 1, and 2 and 3 on bit 3.*
3. *Unusable addresses do not occur for one bank in a half.*
4. *For four banks in a half, all banks must be equal in size or unusable addresses occur. If unusable addresses occur and the banks are in the lower half, MSR cannot be set to SIU generated for bootstrap into the lower half and would have to be manually set on the STU.*
5. *Loss of a segment or bank in a half which contains the resident EXEC brings the application down. If the resident EXEC is restricted to one half of storage, removal of a segment or bank in the other half does not necessitate a reboot, but a failure in this other half will probably cause a reboot.*
6. *The application may be rebooted with the remaining segments. If a two segment half was downed dynamically, it may be reentered into the application with the one remaining segment.*
7. *If the bank lost was the only one in the half, the application may be rebooted with the other half. If the bank was one of four in a half, the application may be rebooted with the unaffected half and either the remaining three banks and two segments in the affected half (see E.4.2.3, Note 3) or two banks and two segments in the affected half.*

E.4.3.4. Three Segments/Six Banks



NOTES:

1. *Interleave between segments 0 and 2 on bit 2.*

2. *Interleave between banks 0 and 1, 2 and 3, and 4 and 5 on bit 3.*

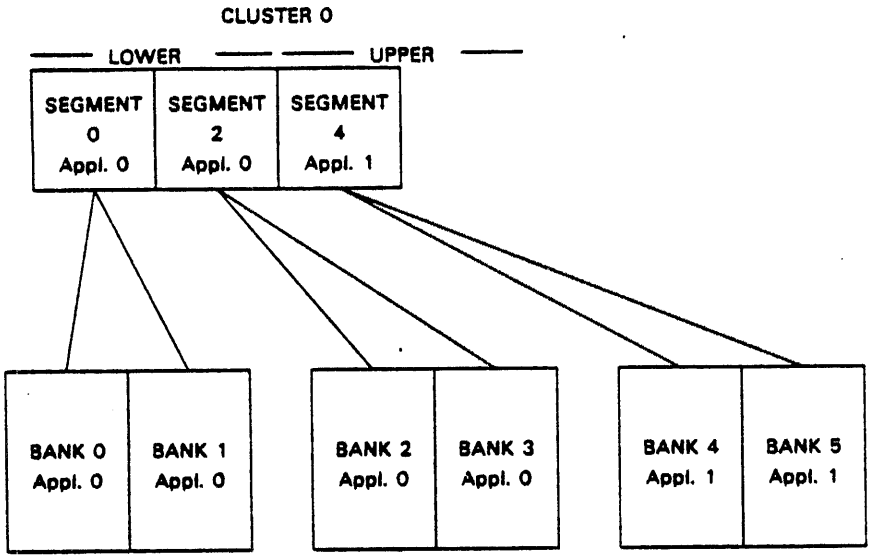
3. *Banks 0, 1, 2 and 3 must be of equal size and banks 4 and 5 must be of equal size or unusable addresses occur. If unusable addresses occur and the banks are in the lower half, MSR cannot be set to SIU generated for bootstrap into the lower half and would have to be manually set on the STU.*

4. *Loss of a segment or bank in a half which contains the resident EXEC brings the application down. If the resident EXEC is restricted to one half of storage, removal of a segment or bank in the other half does not necessitate a reboot, but a failure in this other half will probably cause a reboot.*

5. *The application may be rebooted with the remaining segments. If a two segment half was downed dynamically, it may be reentered into the application with the one remaining segment.*

6. *If the bank lost was one of two in a half, the application may be rebooted with the other half and the remaining bank and segment in the affected half. If the bank was one of four in a half, the application may be rebooted with the other half and either two segments and three banks in the affected half (see E.4.2.3, Note 3) or two segments and two banks.*

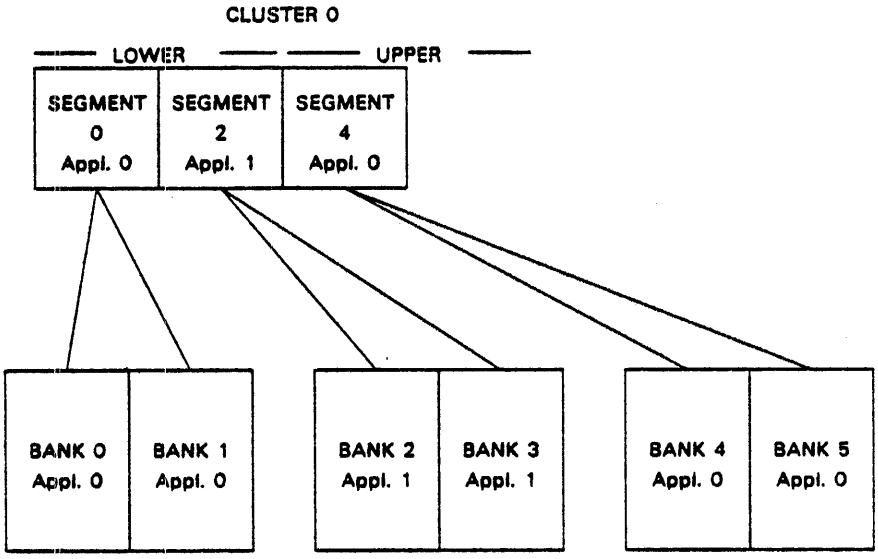
E.4.3.4.1. Partitioned by SIU halves



NOTES:

1. *Interleave between segments 0 and 2 on bit 2 in Application 0.*
2. *Interleave between banks 0 and 1, and 2 and 3 on bit 3 in Application 0.*
3. *Interleave between banks 4 and 5 on bit 3 in Application 1.*
4. *In Application 0, loss of a segment results in bringing down the application. The application may be rebooted with the remaining segment.*
5. *In Application 0, loss of a bank in a half which contains the resident EXEC brings the application down. If the resident EXEC is restricted to one half of storage, removal of a bank in the other half does not necessitate a reboot, but a failure in this other half will probably cause a reboot. The application may be rebooted with two segments and the remaining three banks.*
6. *In Application 1, loss of a segment results in loss of the application. No reboot options.*
7. *In Application 1, loss of a bank results in bringing down the application. The application may be rebooted with the remaining bank.*

E.4.3.4.2. Partitioned Across SIU Halves

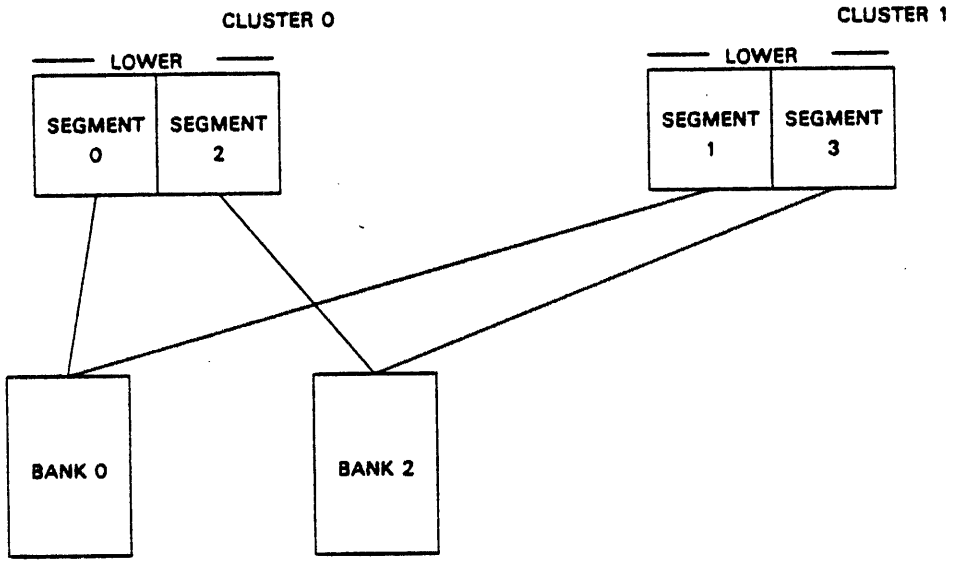


NOTES:

1. *Interleave between banks 0 and 1, and 4 and 5 on bit 3 in Application 0.*
2. *Interleave between banks 2 and 3 on bit 3 in Application 1.*
3. *In Application 0, loss of a segment or bank in a half which contains the resident EXEC brings the application down. If the resident EXEC is restricted to one half of storage, removal of a segment or bank in the other half does not necessitate a reboot, but a failure in this other half will probably cause a reboot.*
4. *In Application 0, the application may be rebooted with the segment in the other half. For loss of a bank, the application may be rebooted with two segments and the remaining three banks.*
5. *In Application 1, loss of a segment results in loss of the application. No reboot options.*
6. *In Application 1, loss of a bank results in bringing down the application. The application may be rebooted with the remaining bank.*

E.4.4. Four-Segment Configurations

E.4.4.1. Four Segments/Two Banks – Dual Cluster



NOTES:

1. *Interleave between segments 0 and 2, and 1 and 3 on bit 2.*

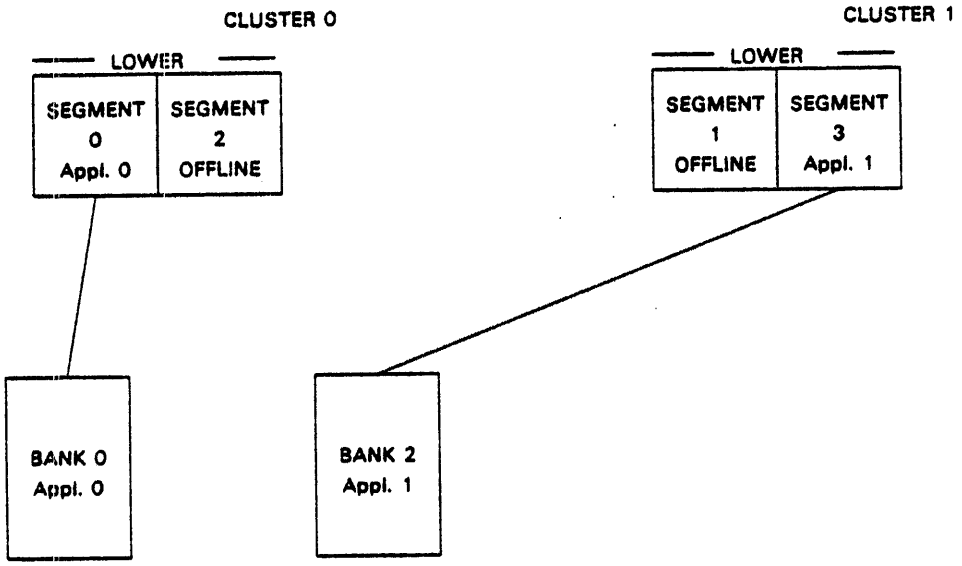
2. *The banks must be equal in size or unusable addresses occur. If they are not equal in size and are in the lower half, MSR cannot be set to SIU generated for bootstrap into the lower half and would have to be manually set on the STU.*

3. *Loss of a segment results in bringing down the application. The application may be rebooted in a single cluster with the two segments for the cluster, provided that this cluster has an IOU. If it is rebooted in this manner, all banks remain available for use but processing power is lessened.*

   The application may also be rebooted as dual cluster with one segment in each cluster. If it is rebooted in this manner, banks connected to the two segments removed are not available for use.

4. *Loss of a bank brings the application down. The application may be rebooted with the remaining bank and its two segments.*

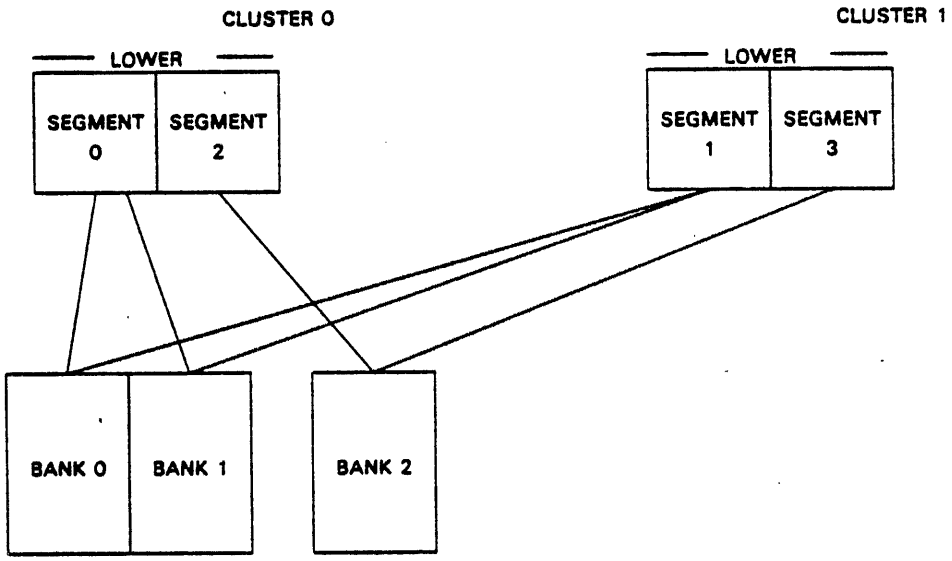
The following is an example of a partitioned configuration.



NOTES:

1. *No interleave.*
2. *Unusable addresses do not occur.*
3. *Loss of a segment or bank results in loss of the application. No reboot options.*

E.4.4.2. Four Segments/Three Banks – Dual Cluster



NOTES:

1. *Interleave between segments 0 and 2, and 1 and 3 on bit 2.*

2. *Interleave between banks 0 and 1 on bit 3.*

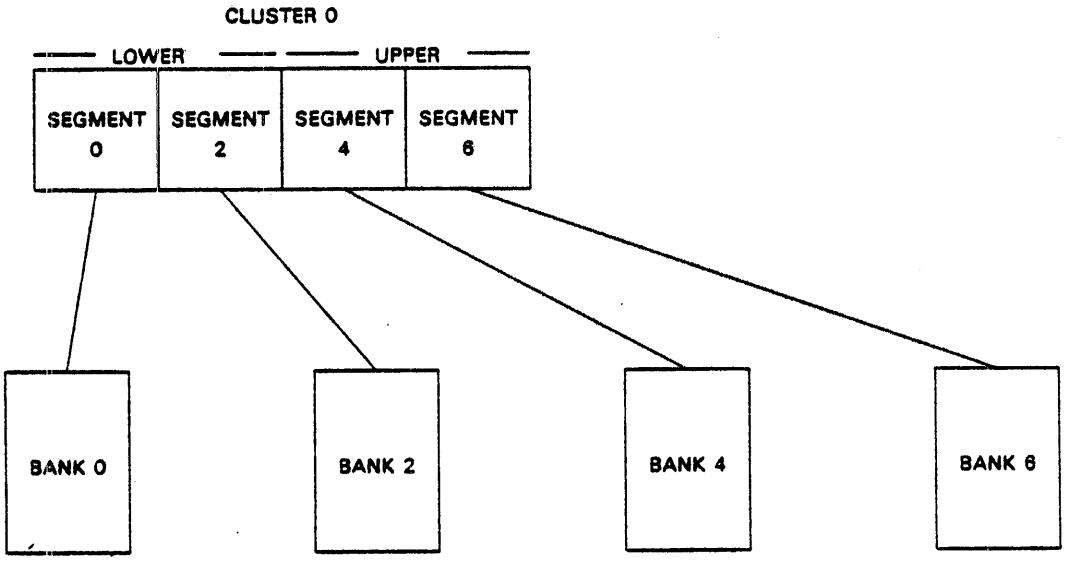
3. *The two banks connected to the same segment must be equal in size and the bank connected to the other segment must have a size equal to the total of the other two banks or unusable addresses occur. If unusable addresses occur and the banks are in the lower half, MSR cannot be set to SIU generated for bootstrap into the lower half and would have to be manually set on the STU.*

4. *Loss of a segment results in bringing down the application. The application may be rebooted in a single cluster with the two segments for the cluster, provided that this cluster has an IOU. If it is rebooted in this manner, all banks remain available for use but processing power is lessened.*

   The application may also be rebooted as dual cluster with one segment in each cluster. If it is rebooted in this manner, banks connected to the two segments removed are not available for use.

5. *Loss of a bank brings the application down. If the bank was the only one connected to two segments, the system may be rebooted with the remaining two banks and their two segments. If the bank was one of two connected to two segments, the segments may be retained with the other banks for the reboot. (Note that for loss of 1 of 4 banks, Note 3 in E.4.2.3 may apply.)*

E.4.4.3. Four Segments/Four Banks



NOTES:

1. *Interleave between segments 0 and 2, and 4 and 6 on bit 2.*

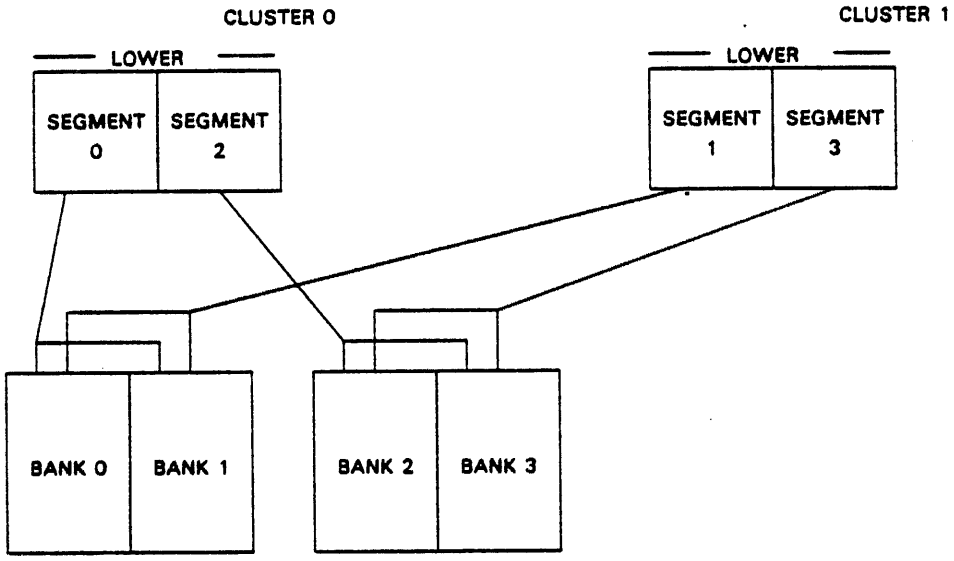
2. *Banks 0 and 2 must be of equal size and 4 and 6 must be of equal size or unusable addresses occur. If they are not equal in size and are in the lower half, MSR cannot be set to SIU generated for bootstrap into the lower half and would have to be manually set on the STU.*

3. *Loss of a segment or bank in a half which contains the resident EXEC brings the application down. If the resident EXEC is restricted to one half of storage, removal of a segment or bank in the other half does not necessitate a reboot, but a failure in this other half will probably cause a reboot.*

4. *The application may be rebooted with the remaining segments. If a two segment half was downed dynamically, it may be reentered into the application with the one remaining segment.*

5. *For loss of a bank, the application may be rebooted with the remaining three banks and their segments.*

E.4.4.4. Four Segments/Four Banks – Dual Cluster



NOTES:

1. *Interleave Between Segments 0 and 2, and 1 and 3 on bit 2.*

2. *Interleave between banks 0 and 1, and 2 and 3 on bit 3.*

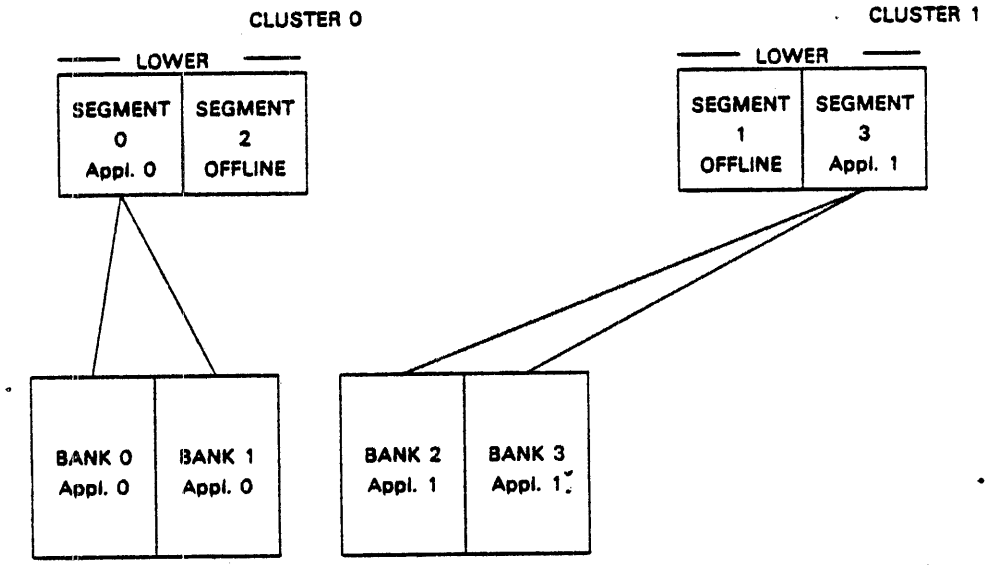
3. *All banks must be equal in size or unusable addresses occur. If unusable addresses occur and the banks are in the lower half, MSR cannot be set to SIU generated for bootstrap into the lower half and would have to be manually set on the STU.*

4. *Loss of a segment results in bringing down the application. The application may be rebooted in a single cluster with the two segments for the cluster, provided that this cluster has an IOU. If it is rebooted in this manner, all banks remain available for use but processing power is lessened.*

   The application may also be rebooted as dual cluster with one segment in each cluster. If it is rebooted in this manner, banks connected to the two segments removed are not available for use.

5. *Loss of a bank brings the application down. If the bank was the only one connected to two segments, the system may be rebooted with the remaining two banks and their two segments. If the bank was one of two connected to two segments, the segments may be retained with the other banks for the reboot. (Note that for loss of 1 of 4 banks, Note 3 in E.4.2.3 may apply.)*

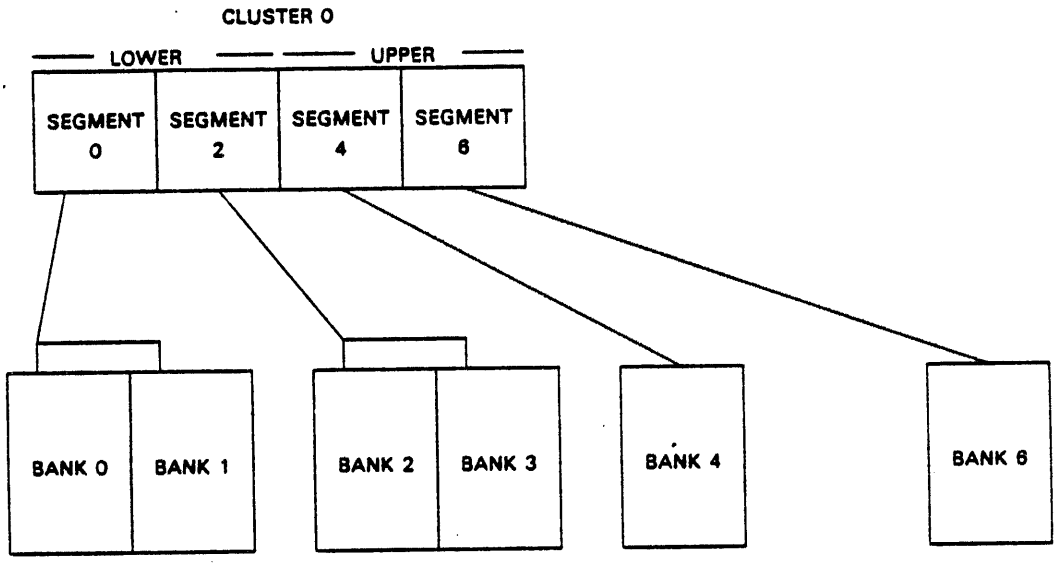
The following is an example of a configuration partitioned by cluster.



NOTES:

1. *Interleave between banks 0 and 1 on bit 3 in Application 0.*
2. *Interleave between banks 2 and 3 on bit 3 in Application 1.*
3. *Loss of a segment results in loss of the application. No reboot options.*
4. *Loss of a bank results in bringing down the application. The application may be rebooted with the remaining bank.*

E.4.4.5. Four Segments/Six Banks



NOTES:

1. *Interleave between segments 0 and 2, and 4 and 6 on bit 2.*

2. *Interleave between banks 0 and 1, and 2 and 3 on bit 3.*

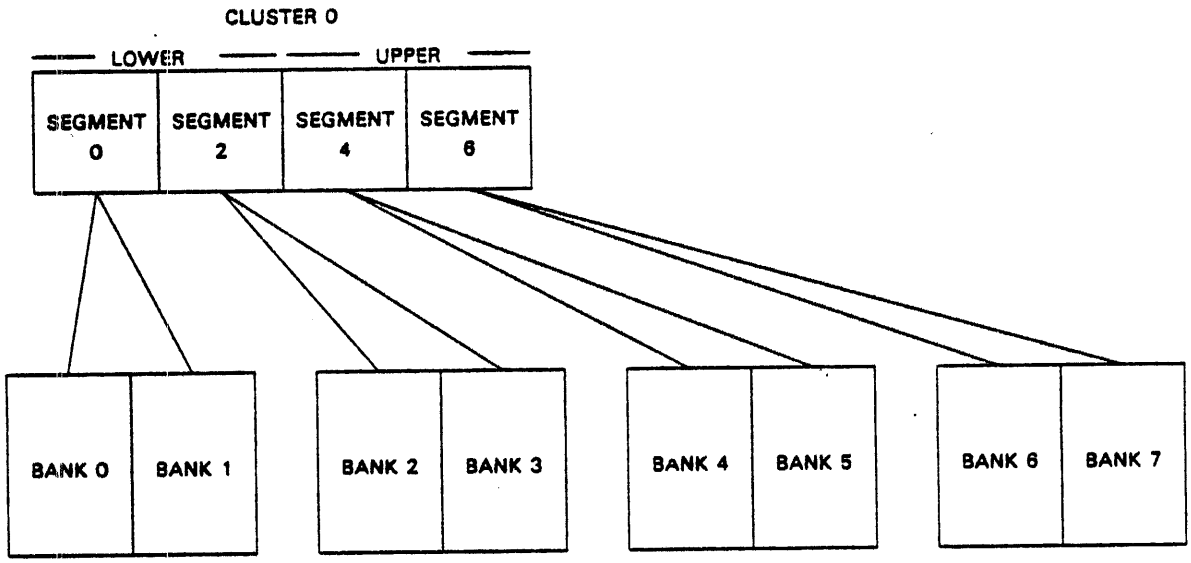
3. *Banks 0, 1, 2, and 3 must be of equal size and banks 4 and 6 must be of equal size or unusable addresses occur. If unusable addresses occur and the banks are in the lower half, MSR cannot be set to SIU generated for bootstrap into the lower half and would have to be manually set on the STU.*

4. *Loss of a segment or bank in a half which contains the resident EXEC brings the application down. If the resident EXEC is restricted to one half of storage, removal of a segment or bank in the other half does not necessitate a reboot, but a failure in this other half will probably cause a reboot.*

5. *The application may be rebooted with the remaining segments. If a two segment half was downed dynamically, it may be reentered into the application with the one remaining segment.*

6. *If the bank lost was one of two in a half, the application may be rebooted with the unaffected half and the remaining bank and its segment in the affected half. If the bank lost was one of four in a half, the application may be rebooted with the unaffected half and either three banks and two segments in the affected half (see E.4.2.3, Note 3) or two banks and two segments in the affected half.*

E.4.4.6. Four Segments/Eight Banks



NOTES:

1. *Interleave between segments 0 and 2, and 4 and 6 on bit 2.*

2. *Interleave between banks 0 and 1, 2 and 3, 4 and 5, and 6 and 7 on bit 3.*

3. *Banks 0, 1, 2, and 3 must be of equal size and banks 4, 5, 6, and 7 must be of equal size or unusable addresses occur. If unusable addresses occur and the banks are in the lower half, MSR cannot be set to SIU generated for bootstrap into the lower half and would have to be manually set on the STU.*

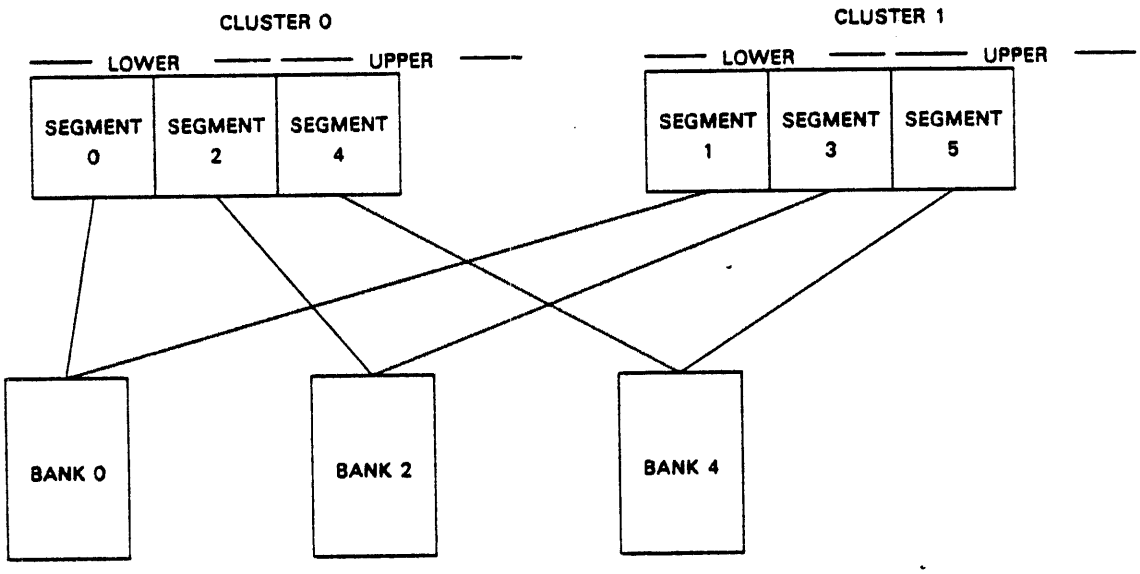
4. *Loss of a segment or bank in a half which contains the resident EXEC brings the application down. If the resident EXEC is restricted to one half of storage, removal of a segment or bank in the other half does not necessitate a reboot, but a failure in this other half will probably cause a reboot.*

5. *The application may be rebooted with the remaining segments. If a two segment half was downed dynamically, it may be reentered into the application with the one remaining segment.*

6. *For loss of a bank, the application may be rebooted with the other half and either two segments and three banks in the effected half (see E.4.2.3, Note 3) or two segments and two banks.*

E.4.5. Six-Segment Configurations

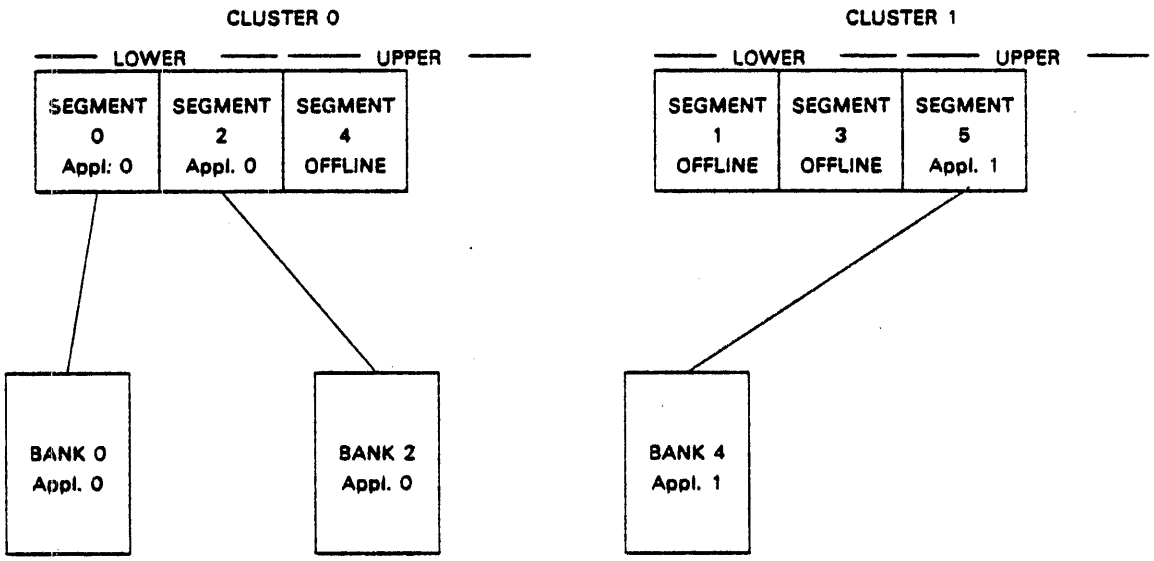
E.4.5.1. Six Segments/Three Banks - Dual Cluster



NOTES:

1. *Interleave between segments 0 and 2, and 1 and 3 on bit 2.*
2. *For one bank in a half, unusable addresses do not occur.*
3. *Banks 0 and 2 must be equal in size or unusable addresses occur. If they are not equal in size and are in the lower half, MSR cannot be set to SIU generated for bootstrap into the lower half and would have to be manually set on the STU.*
4. *Loss of a segment or bank in a half which contains the resident EXEC brings the application down. If the resident EXEC is restricted to one half of storage, removal of a segment or bank in the other half does not necessitate a reboot, but a failure in this other half will probably cause a reboot.*
5. *The application may be rebooted in a single cluster with the three segments for the cluster, provided that this cluster has an IOU. If it is rebooted in this manner, all banks remain available for use but processing power is lessened. The application may also be rebooted as dual cluster with two segments in each cluster. If it is rebooted in this manner, banks connected to the two segments removed are not available for use.*
6. *For loss of a bank, the application may be rebooted with the remaining two banks and their four segments.*

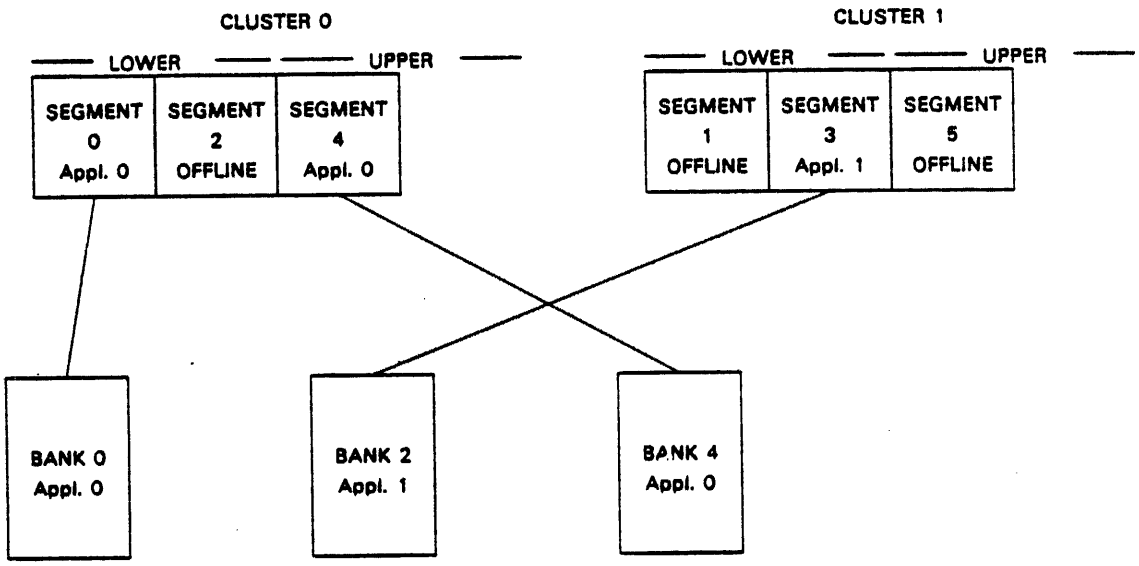
E.4.5.1.1. Partitioned by Cluster and SIU Halves



NOTES:

1. *Interleave between segments 0 and 2 on bit 2 in Application 0.*
2. *No interleave in Application 1.*
3. *In Application 0, loss of a segment results in bringing down the application. The application may be rebooted with the remaining segment.*
4. *In Application 0, loss of a bank results in bringing down the application and loss of the bank's segment. The application may be rebooted with the remaining bank and segment.*
5. *Unusable addresses do not occur in Application 1.*
6. *Loss of a segment or bank in Application 1 results in loss of the application. No reboot options.*

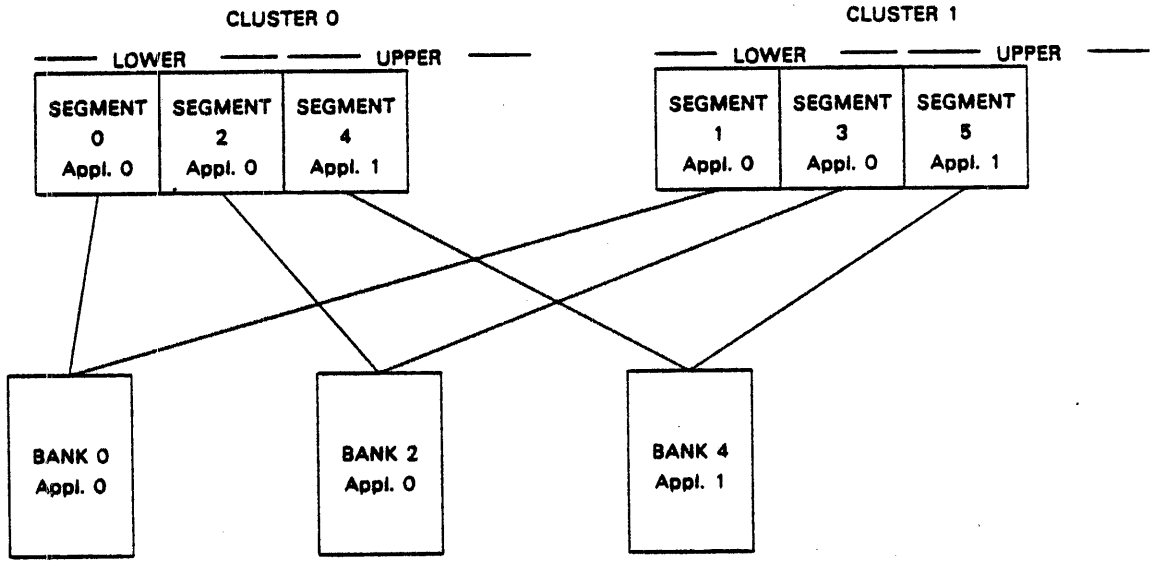
E.4.5.1.2. Partitioned by Cluster Across SIU Halves



NOTES:

1. *No interleave in either application.*
2. *Unusable addresses do not occur in either application.*
3. *Loss of a segment or bank in Application 0 in a half which contains the resident EXEC brings the application down. If the resident EXEC is restricted to one half of storage, removal of a segment or bank in the other half does not necessitate a reboot, but a failure in this other half will probably cause a reboot.*
4. *For loss of a segment, the application may be rebooted with the segment in the other half in Application 0.*
5. *For loss of a bank, the application may be rebooted with the bank and segment in the unaffected half in Application 0.*
6. *Loss of a segment or bank in Application 1 results in loss of the application. No reboot options.*

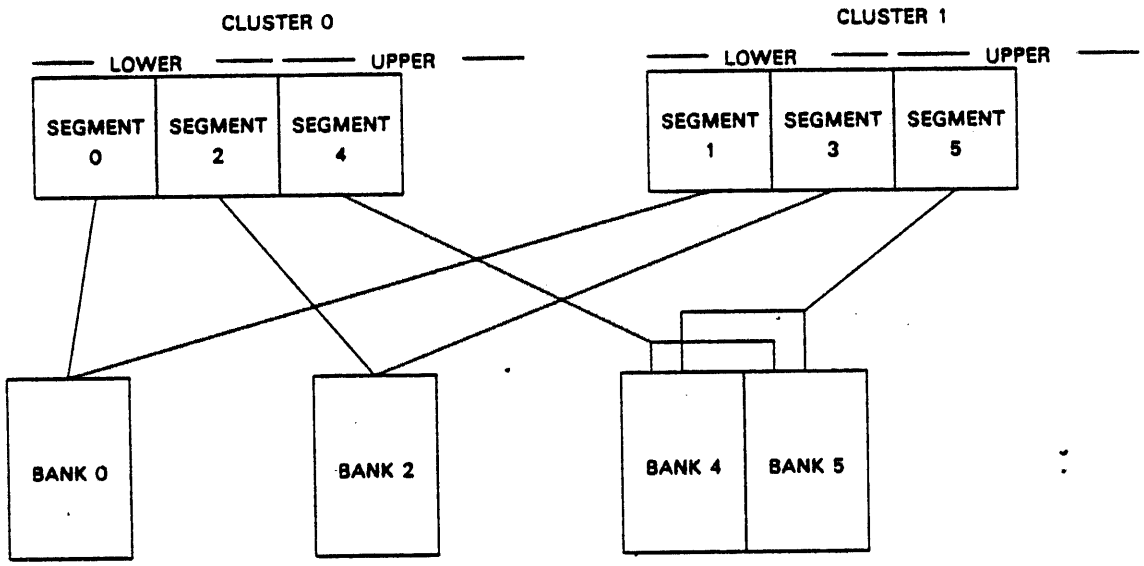
E.4.5.1.3. Partitioned Within Cluster and SIU Halves



NOTES:

1. *Interleave between segments 0 and 2, and 1 and 3 on bit 2 in Application 0.*
2. *No interleave in Application 1.*
3. *In Application 0, the banks must be equal in size or unusable addresses occur. If they are not equal in size and are in the lower half, MSR cannot be set to SIU generated for bootstrap into the lower half and would have to be manually set on the STU.*
4. *Loss of a segment or bank results in bringing down Application 0.*
5. *Application 0 may be rebooted in a single cluster with the two segments for the cluster, provided that this cluster has an IOU. If it is rebooted in this manner, all banks remain available for use but processing power is lessened. The application may also be rebooted as dual cluster with one segment in each cluster. If it is rebooted in this manner, banks connected to the two segments removed are not available for use.*
6. *If a bank is lost in Application 0, the system may be rebooted with the remaining bank and its two segments.*
7. *Unusable addresses do not occur in Application 1.*
8. *In Appliction 1, loss of a segment or bank results in bringing down the application. The application may be rebooted in a single cluster with the one remaining segment, provided that this cluster has an IOU.*

E.4.5.2. Six Segments/Four Banks - Dual Cluster



NOTES:

1. *Interleave between segments 0 and 2, and 1 and 3 on bit 2.*

2. *Interleave between banks 4 and 5 on bit 3.*

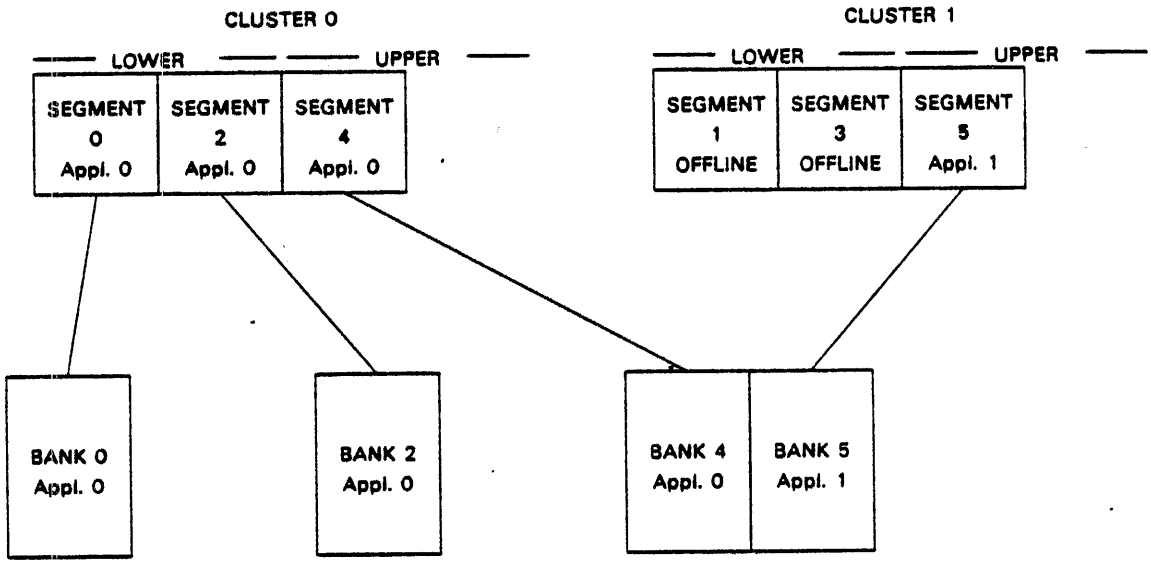
3. *Banks 0 and 2 must be of equal size and banks 4 and 5 must be of equal size or unusable addresses occur. If they are not equal in size and are in the lower half, MSR cannot be set to SIU generated for bootstrap into the lower half and would have to be manually set on the STU.*

4. *Loss of a segment or bank in a half which contains the resident EXEC brings the application down. If the resident EXEC is restricted to one half of storage, removal of a segment or bank in the other half does not necessitate a reboot, but a failure in this other half will probably cause a reboot.*

5. *The application may be rebooted in a single cluster with the three segments for the cluster, provided that this cluster has an IOU. If it is rebooted in this manner, all banks remain available for use but processing power is lessened. The application may also be rebooted as dual cluster with two segments in each cluster. If it is rebooted in this manner, banks connected to the two segments removed are not available for use.*

6. *If the bank lost was the only one connected to two segments, the application may be rebooted with the remaining three banks and four segments. If the bank was one of two connected to a segment, the application may be rebooted with the remaining three banks and all six segments.*

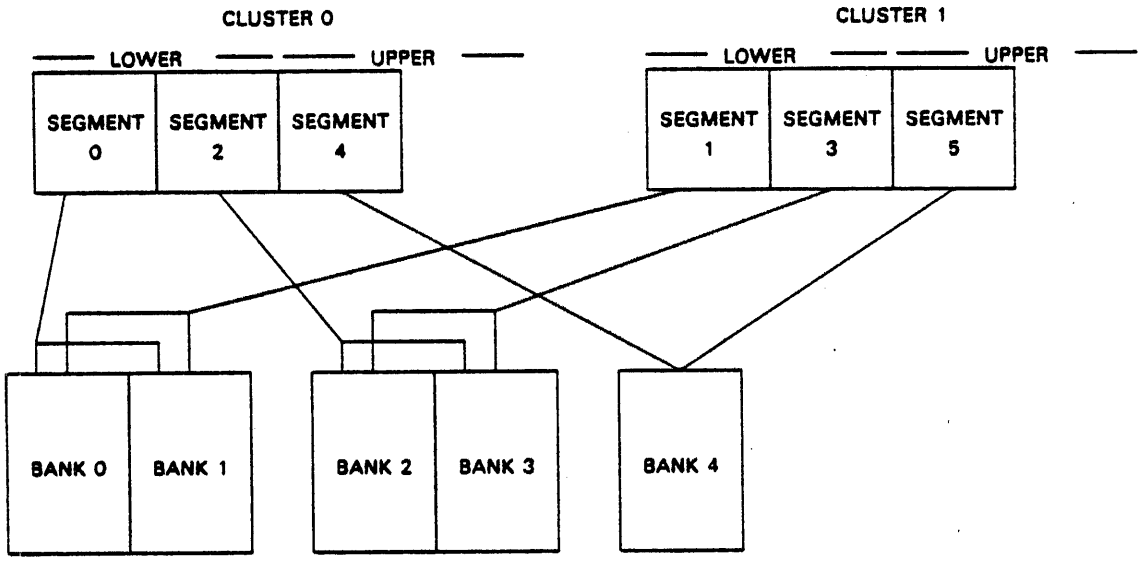
The following is an example of a configuration partitioned by cluster.



NOTES:

1. *Interleave between segments 0 and 2 on bit 2 in Application 0.*
2. *No interleave in Application 1.*
3. *Unusable addresses do not occur for one bank in a half.*
4. *Loss of a segment or bank in a half which contains the resident EXEC brings the application down. If the resident EXEC is restricted to one half of storage, removal of a segment or bank in the other half does not necessitate a reboot, but a failure in this other half will probably cause a reboot.*
5. *For loss of a segment, application 0 may be rebooted with the remaining segments. If a two segment half was downed dynamically, it may be reentered into the application with the one remaining segment. For loss of a bank, the application may be rebooted with the remaining two banks and their segments.*
6. *In Application 1, loss of a segment or bank results in loss of the application. No reboot options.*

E.4.5.3. Six Segments/Five Banks – Dual Cluster



NOTES:

1. *Interleave between segments 0 and 2, and 1 and 3 on bit 2.*

2. *Interleave between banks 0 and 1, and 2 and 3 on bit 3.*

3. *Unusable addresses do not occur for one bank in a half.*

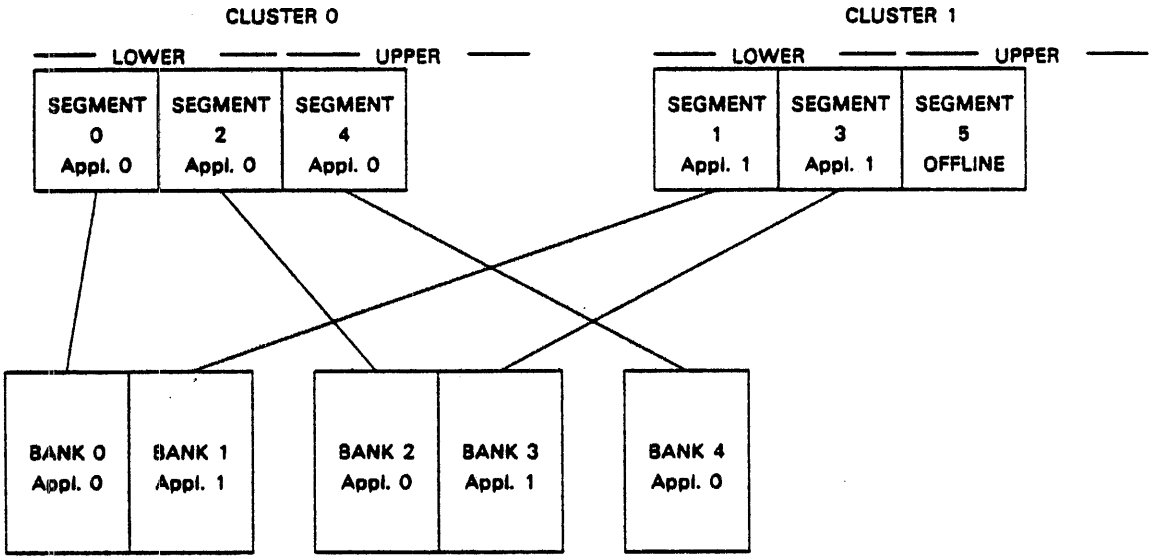
4. *Banks 0, 1, 2, and 3 must be of equal size or unusable addresses occur. If unusable addresses occur and the banks are in the lower half, MSR cannot be set to SIU generated for bootstrap into the lower half and would have to be manually set on the STU.*

5. *Loss of a segment or bank in a half which contains the resident EXEC brings the application down. If the resident EXEC is restricted to one half of storage, removal of a segment or bank in the other half does not necessitate a reboot, but a failure in this other half will probably cause a reboot.*

6. *The application may be rebooted in a single cluster with the three segments for the cluster, provided that this cluster has an IOU. If it is rebooted in this manner, all banks remain available for use but processing power is lessened. The application may also be rebooted as dual cluster with two segments in each cluster. If it is rebooted in this manner, banks connected to the two segments removed are not available for use.*

7. *If the bank lost was the only one in the half, the application may be rebooted with the unaffected half. If the bank was one of four in a half, the application may be rebooted with the unaffected half and either the remaining three banks and four segments in the affected half (see E.4.2.3, Note 3) or two banks and four segments in the affected half.*

E.4.5.3.1. Partitioned by Cluster



NOTES:

1. *Interleave between segments 0 and 2 on bit 2 in Application 0.*

2. *Interleave between segments 1 and 3 on bit 2 in Application 1.*

3. *In Application 0, loss of a segment or bank in a half which contains the resident EXEC brings the application down. If the resident EXEC is restricted to one half of storage, removal of a segment or bank in the other half does not necessitate a reboot, but a failure in this other half will probably cause a reboot.*

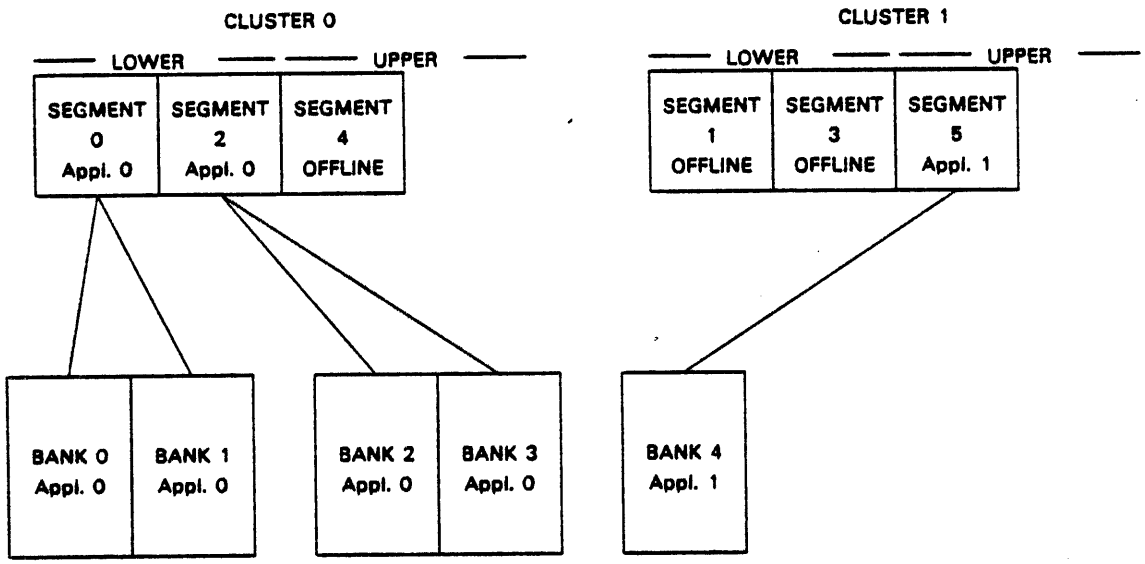
4. *The application may be rebooted with the remaining segments. If a two segment half was downed dynamically, it may be reentered into the application with the one remaining segment.*

5. *For loss of a bank in Application 0, the application may be rebooted with the remaining two banks and their segments.*

6. *In Application 1, loss of a segment results in loss of the application. The application may be rebooted with the remaining segment.*

7. *Loss of a bank results in bringing down the application and loss of the bank's segment. The application may be rebooted with the remaining bank and segment.*

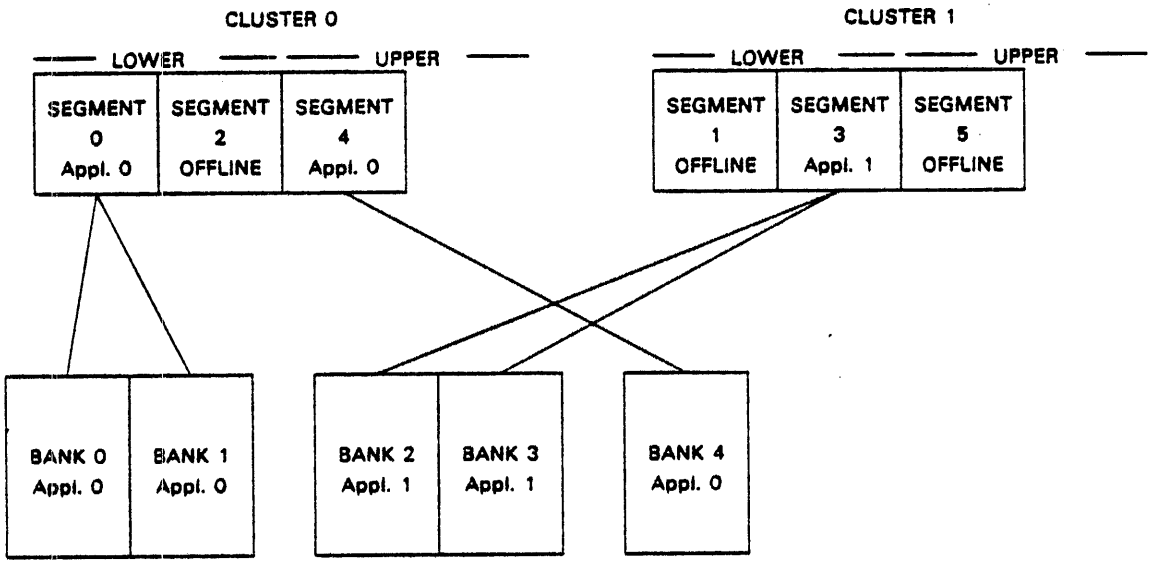
E.4.5.3.2. Partitioned by Cluster by SIU Halves



NOTES:

1. *Interleave between segments 0 and 2 on bit 2 in Application 0.*
2. *Interleave between banks 0 and 1, and 2 and 3 on bit 3 in Application 0.*
3. *Loss of a segment results in bringing down Application 0. The application may be rebooted with the remaining segments.*
4. *Loss of a bank results in bringing down the application. The application may be rebooted with the two segments and remaining three banks (see E.4.2.3, Note 3) or with two segments and two banks.*
5. *No interleave in Application 1. Unusable addresses do not occur.*
6. *Loss of a segment or bank in Application 1 results in loss of the application. No reboot options.*

E.4.5.3.3. Partitioned by Cluster Across SIU Halves



NOTES:

1. *Interleave between banks 0 and 1 on bit 3 in Application 0.*

3. *In Application 0, loss of a segment or bank in a half which contains the resident EXEC brings the aplication down. If the resident EXEC is restricted to one half of storage, removal of a segment or bank in the other half does not necessitate a reboot, but a failure in this other half will probably cause a reboot.*

4. *The application may be rebooted with the segment in the other half.*

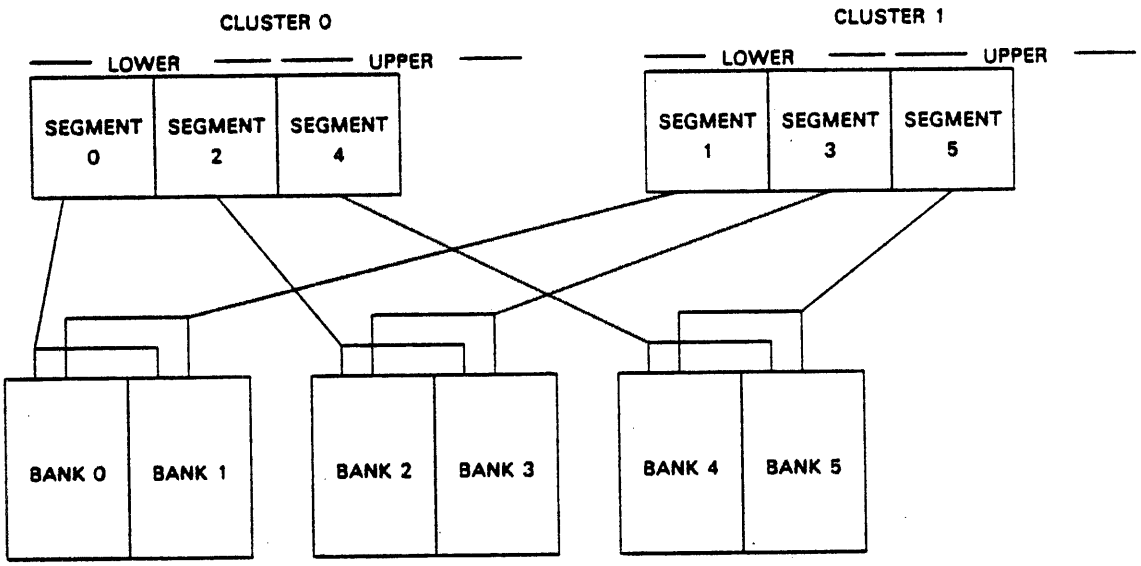
5. *If the bank lost is the only one connected to a segment, the application may be rebooted with the other segment and its banks. If the bank was one of two connected to a segment, the application may be rebooted with both segments and the remaining two banks.*

6. *Interleave between banks 2 and 3 on bit 3 in Application 1.*

7. *Loss of a segment in Application 1 results in loss of the application. No reboot options.*

8. *Loss of a bank in Application 1 results in loss of the application. The application may be rebooted with the remaining bank.*

E.4.5.4. Six Segments/Six Banks – Dual Cluster



NOTES:

1. *Interleave between segments 0 and 2, and 1 and 3 on bit 2.*

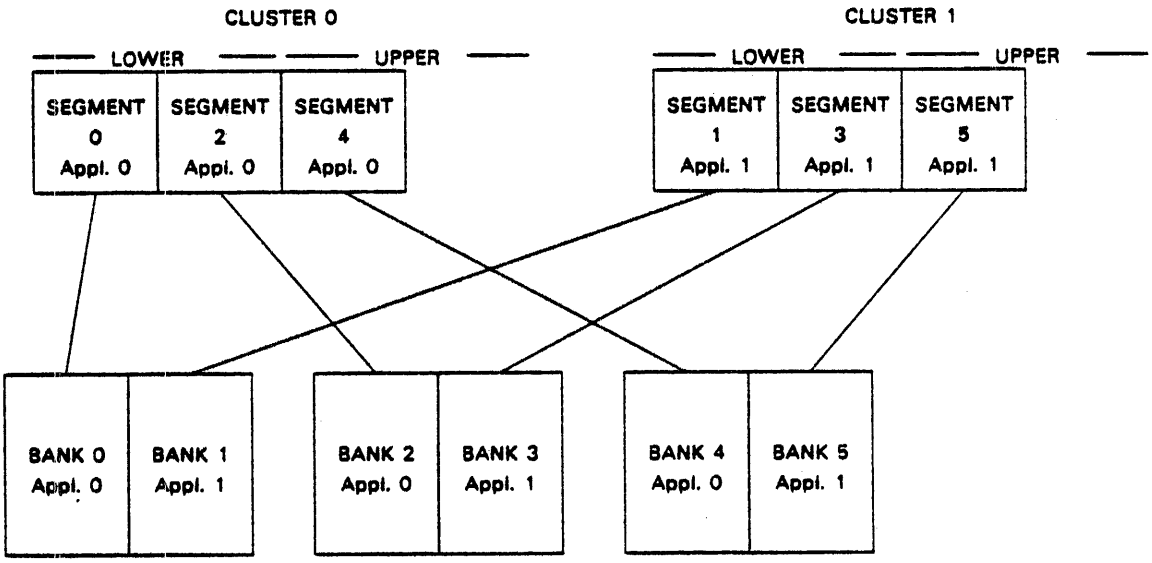
2. *Banks 0, 1, 2, 3 must be of equal size and banks 4 and 5 must be of equal size or unusable addresses occur. If unusable addresses occur and the banks are in the lower half, MSR cannot be set to SIU generated for bootstrap into the lower half and would have to be manually set on the STU.*

3. *Loss of a segment or bank in a half which contains the resident EXEC brings the application down. If the resident EXEC is restricted to one half of storage, removal of a segment or bank in the other half does not necessitate a reboot, but a failure in this other half will probably cause a reboot.*

4. *The application may be rebooted in a single cluster with the three segments for the cluster, provided that this cluster has an IOU. If it is rebooted in this manner, all banks remain available for use but processing power is lessened. The application may also be rebooted as dual cluster with two segments in each cluster. If it is rebooted in this manner, banks connected to the two segments removed are not available for use.*

5. *If the bank lost was one of two in a half, the application may be rebooted with the other half and the remaining bank and two segments in the effected half. If the bank was one of four in a half, the application may be rebooted with the unaffected half and either three banks and four segments in the affected half (see E.4.2.3, Note 3) or four segments and two banks.*

The following is an example of a configuration partitioned by cluster.



NOTES:

1. *Interleave between segments 0 and 2 on bit 2 in Application 0.*

2. *Interleave between segments 1 and 3 on bit 2 in Application 1.*

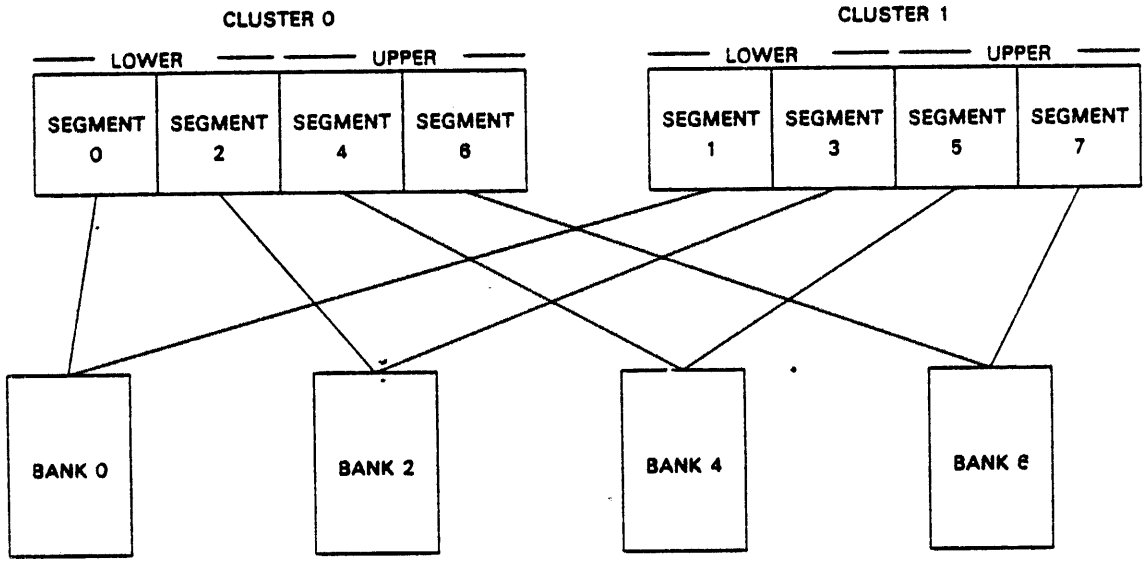
3. *Loss of a segment or bank in a half which contains the resident EXEC brings the application down. If the resident EXEC is restricted to one half of storage, removal of a segment or bank in the other half does not necessitate a reboot, but a failure in this other half will probably cause a reboot.*

4. *Either application may be rebooted with the remaining segments. If a two segment half was downed dynamically, it may be reentered into the application with the one remaining segment.*

5. *For loss of a bank, either application may be rebooted with the remaining two banks and their segments.*

E.4.6. Eight-Segment Configurations

E.4.6.1. Eight Segments/Four Banks – Dual Cluster



NOTES:

1. *Interleave between segments 0 and 2, 4 and 6, 1 and 3, and 5 and 7 on bit 2.*

2. *Banks 0 and 2 must be of equal size and banks 4 and 5 must be of equal size or unusable addresses occur. If they are not equal in size and are in the lower half, MSR cannot be set to SIU generated for bootstrap into the lower half and would have to be manually set on the STU.*

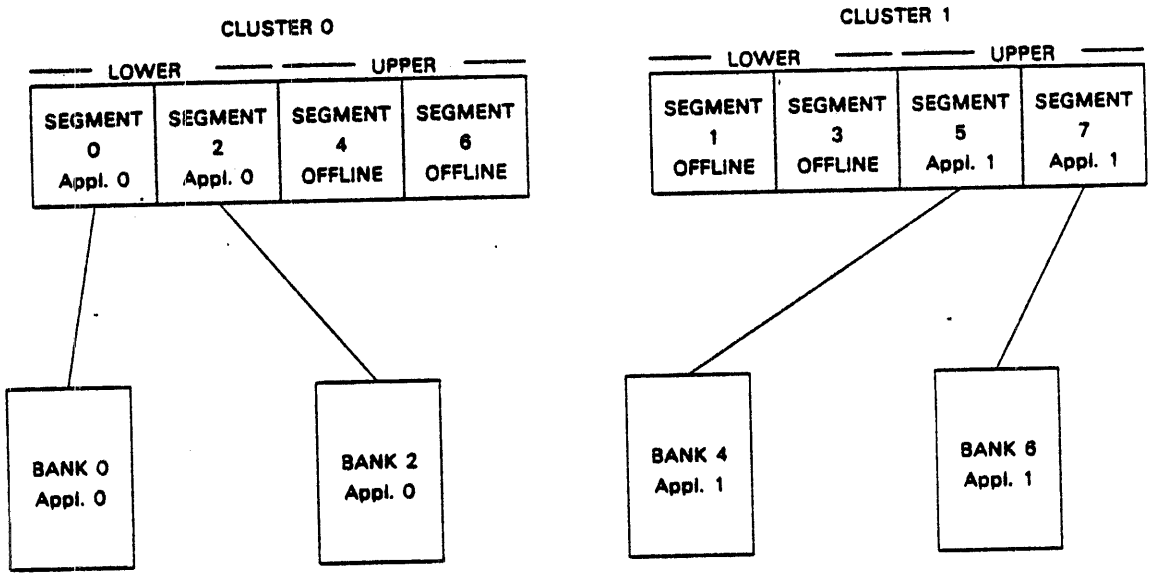
3. *Loss of a segment or bank in a half which contains the resident EXEC brings the application down. If the resident EXEC is restricted to one half of storage, removal of a segment or bank in the other half does not necessitate a reboot, but a failure in this other half will probably cause a reboot.*

4. *The application may be rebooted in a single cluster with the four segments for the cluster, provided that this cluster has an IOU. If it is rebooted in this manner, all banks remain available for use but processing power is lessened.*

   The application may also be rebooted as a dual cluster with three segments in each cluster. If it is rebooted in this manner, banks connected to the two segments removed are not available for use.

5. *For loss of a bank, the application may be rebooted with the unaffected half and the remaining bank and its two segments in the affected half.*

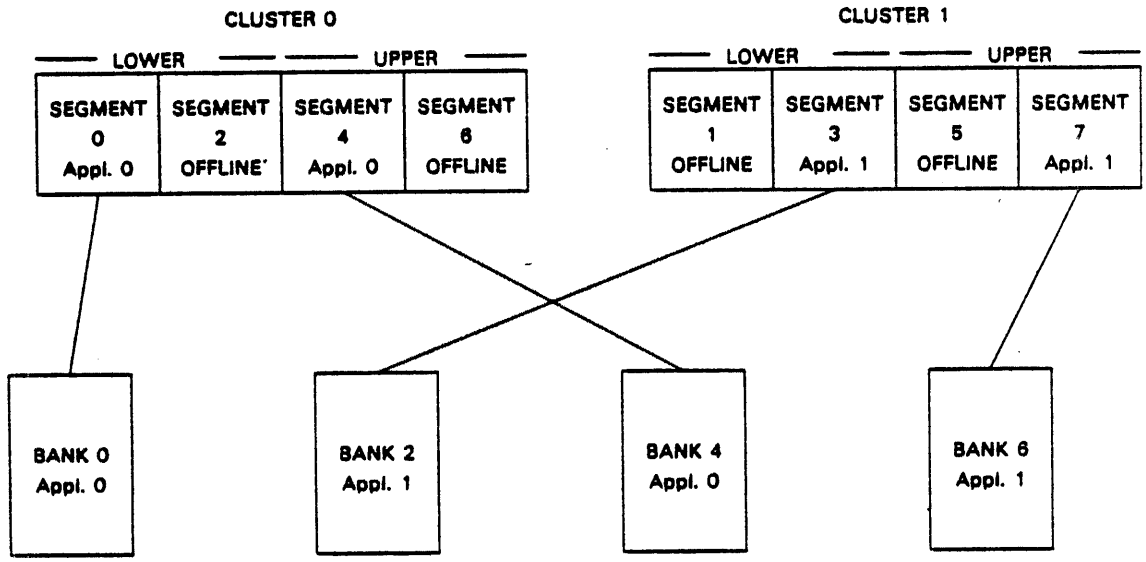
E.4.6.1.1. Partitioned by Cluster and SIU Halves



NOTES:

1. *Interleave between segments 0 and 2 on bit 2 in Application 0.*
2. *Interleave between segments 5 and 7 on bit 2 in Application 1.*
3. *Loss of a segment results in loss of the application. The application may be rebooted with the remaining segment.*
4. *Loss of a bank brings the application down. The application may be rebooted with the remaining bank and segment.*

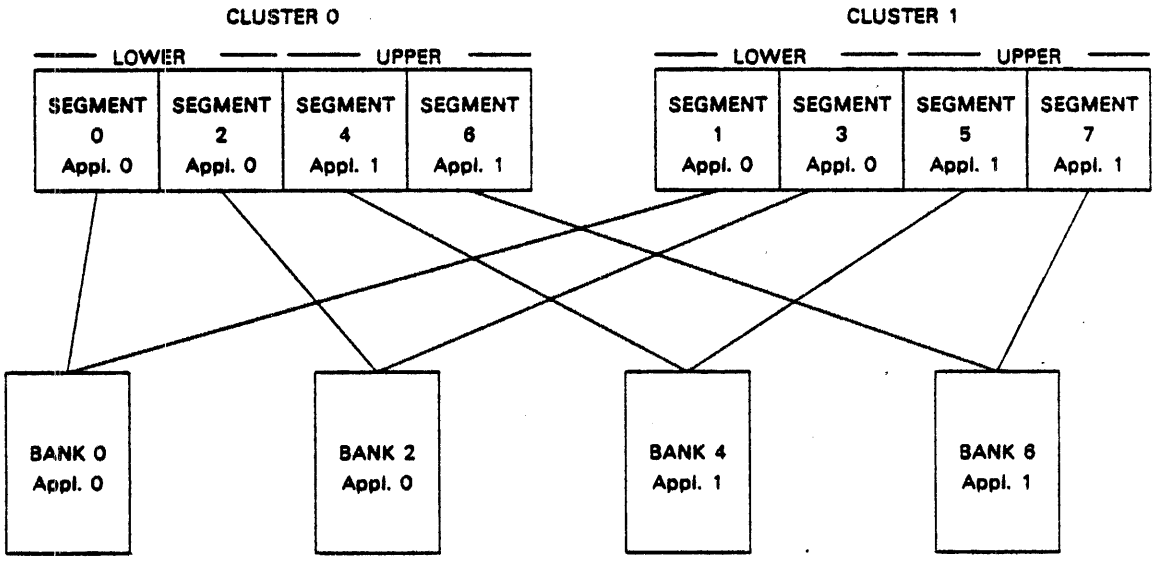
E.4.6.1.2. Partitioned by Cluster Across SIU Halves



NOTES:

1. *No interleave.*
2. *Unusable addresses do not occur.*
3. *Loss of a segment or bank in a half which contains the resident EXEC brings the application down. If the resident EXEC is restricted to one half of storage, removal of a segment or bank in the other half does not necessitate a reboot, but a failure in this other half will probably cause a reboot.*
4. *For loss of a segment, the application may be rebooted with the segment in the other half.*
5. *For loss of a bank, the application may be rebooted with the bank and segment in the unaffected half.*

E.4.6.1.3. Partitioned Within Cluster by SIU Halves



NOTES:

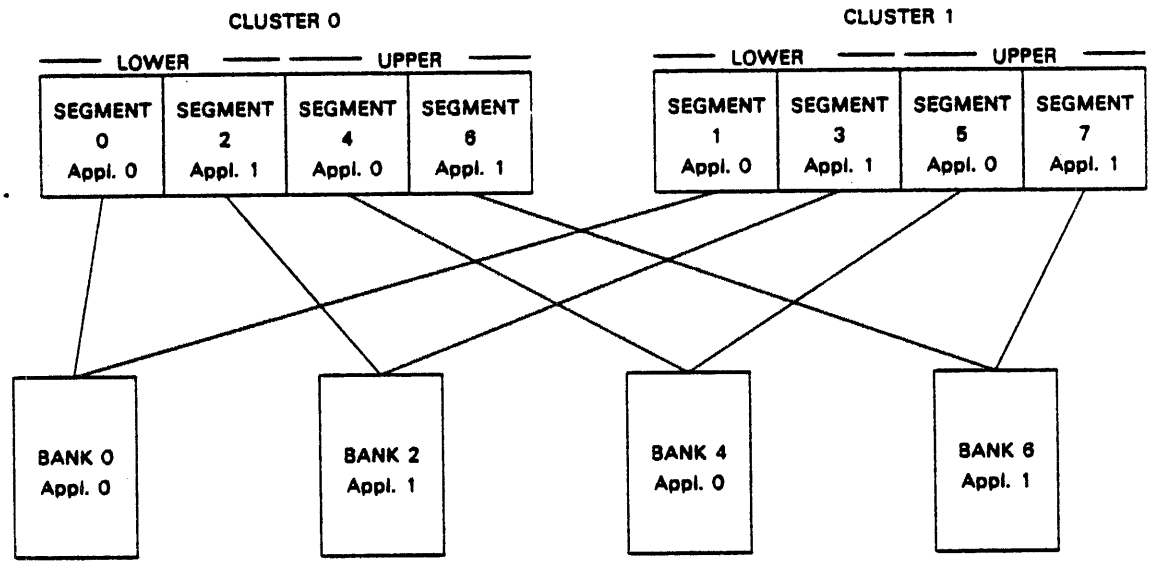
1. *Interleave between segments 0 and 2, and 1 and 3 on bit 2 in Application 0.*

2. *Interleave between segments 4 and 6, and 5 and 7 on bit 2 in Application 1.*

3. *Loss of a segment results in bringing down the application. The application may be rebooted in a single cluster with the two segments for the cluster, provided that this cluster has an IOU. If it is rebooted in this manner, all banks remain available for use but processing power is lessened. The application may also be rebooted as dual cluster with one segment in each cluster. If it is rebooted in this manner, banks connected to the two segments removed are not available for use.*

4. *Loss of a bank brings the application down. The application may be rebooted with the remaining bank and its two segments.*

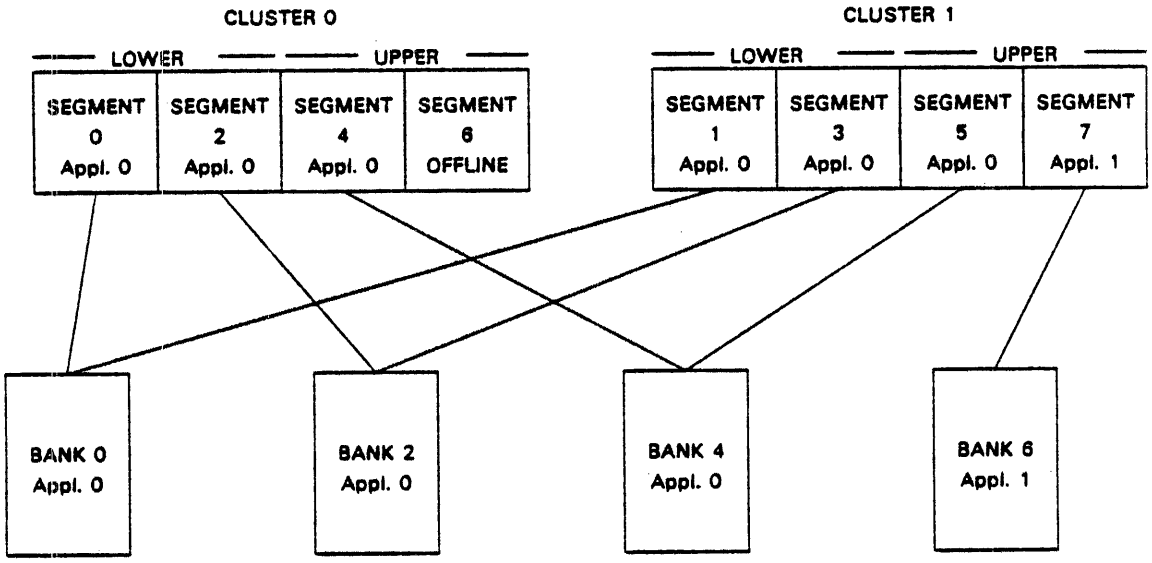
E.4.6.1.4. Partitioned Within Cluster Across SIU Halves



NOTES:

1. *No interleave.*
2. *Unusable addresses do not occur.*
3. *Loss of a segment or bank in a half which contains the resident EXEC brings the application down. If the resident EXEC is restricted to one half of storage, removal of a segment or bank in the other half does not necessitate a reboot, but a failure in this other half will probably cause a reboot.*
4. *The application may be rebooted in a single cluster with the two segments for the cluster, provided that this cluster has an IOU. If it is rebooted in this manner, all banks remain available for use but processing power is lessened. The application may also be rebooted as dual cluster with one segment in each cluster. If it is rebooted in this manner, banks connected to the two segments removed are not available for use.*
5. *For loss of a bank, the application may be rebooted with the remaining bank and its two segments.*

E.4.6.1.5. Minimal Storage Partitioned Out



NOTES:

1. *Interleave between segments 0 and 2, and 1 and 3 on bit 2 in Application 0.*

2. *Loss of a segment or bank in a half which contains the resident EXEC brings the application down. If the resident EXEC is restricted to one half of storage, removal of a segment or bank in the other half does not necessitate a reboot, but a failure in this other half will probably cause a reboot.*

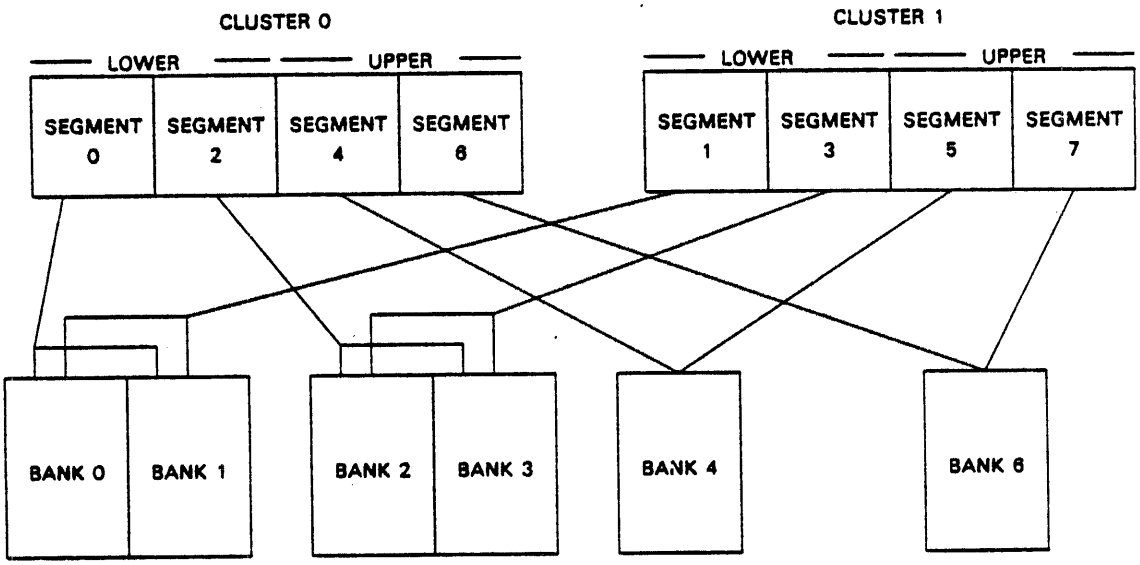
3. *Application 0 may be rebooted in a single cluster with the three segments for the cluster, provided that this cluster has an IOU. If it is rebooted in this manner, all banks remain available for use but processing power is lessened. The application may also be rebooted as dual cluster with two segments in each cluster. If it is rebooted in this manner, banks connected to the two segments removed are not available for use.*

4. *For loss of a bank, the application may be rebooted with the remaining two banks and their four segments.*

5. *No interleave in Application 1.*

6. *Loss of a segment or bank in Application 1 results in loss of the application. No reboot options.*

E.4.6.2. Eight Segments/Six Banks – Dual Cluster



NOTES:

1. *Interleave between segments 0 and 2, 4 and 6, 1 and 3, and 5 and 7 on bit 2.*

2. *Interleave between banks 0 and 1, and 2 and 3 on bit 3.*

3. *Banks 0, 1, 2, and 3 must be of equal size and banks 4 and 6 must be of equal size or unusable addresses occur. If unusable addresses occur and the banks are in the lower half, MSR cannot be set to SIU generated for bootstrap into the lower half and would have to be manually set on STU.*

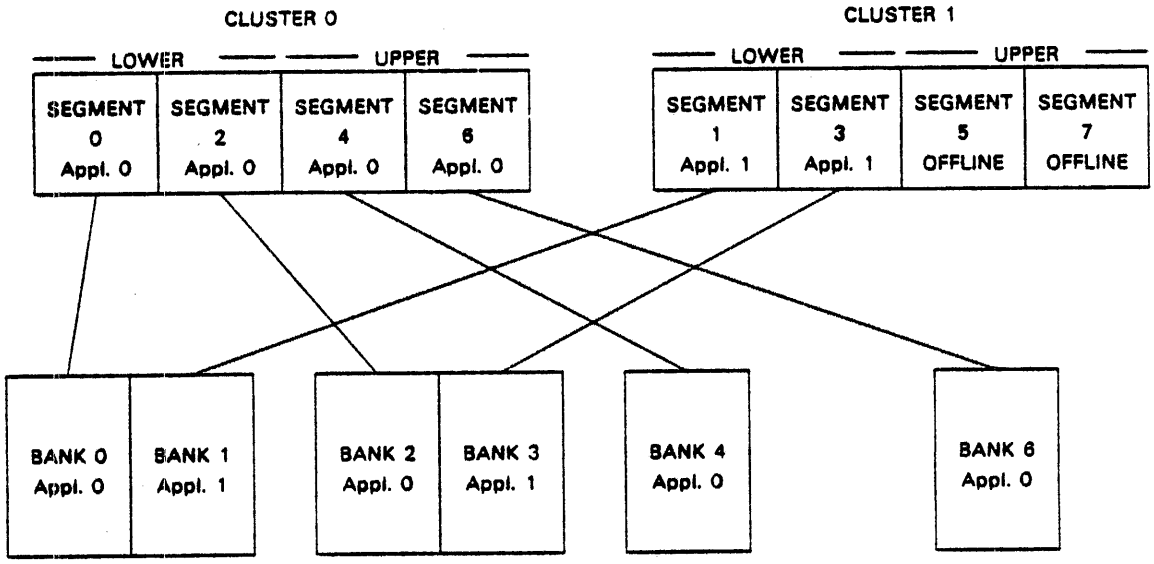
4. *Loss of a segment in a half which contains the resident EXEC brings the application down. If the resident EXEC is restricted to one half of storage, removal of a segment or bank in the other half does not necessitate a reboot, but a failure in this other half will probably cause a reboot.*

5. *The application may be rebooted in a single cluster with the four segments for the cluster, provided that this cluster has an IOU. If it is rebooted in this manner, all banks remain available for use but processing power is lessened. The application may also be rebooted as dual cluster with six segments in each cluster. If it is rebooted in this manner, banks connected to the two segments removed are not available for use.*

6. *Loss of a bank in a half which contains the resident EXEC brings the application down. If the resident EXEC is restricted to one half of storage, removal of a bank in the other half does not necessitate a reboot, but a failure in this other half will probably cause a reboot.*

7. *If the bank lost was one of two in a half, the application may be rebooted with the unaffected half and the remaining bank and its two segments in the affected half. If the bank was one of four in a half, the application may be rebooted with the unaffected half and either the remaining three banks and four segments (see E.4.2.3, Note 3) or two banks and four segments in the affected half.*

E.4.6.2.1. Partitioned by Cluster



NOTES:

1. *Interleave between segment 0 and 2, and 4 and 6 on bit 2 in Application 0.*

2. *Loss of a segment or bank in a half which contains the resident EXEC brings the application down. If the resident EXEC is restricted to one half of storage, removal of a segment or bank in the other half does not necessitate a reboot, but a failure in this other half will probably cause a reboot.*

3. *Application 0 may be rebooted with the remaining segments. If a two segment half was downed dynamically, it may be reentered into the application with the one remaining segment.*

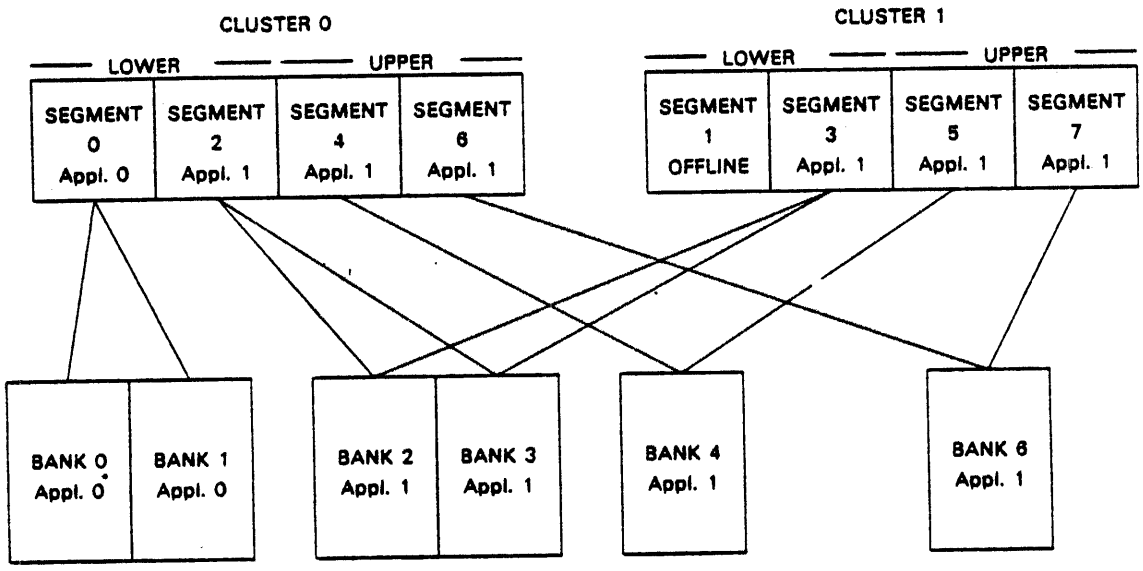
4. *For loss of a bank, the application may be rebooted with the unaffected half and the remaining bank and segment in the affected half.*

5. *Interleave between segments 1 and 3 on bit 2 in Application 1.*

6. *Loss of a segment results in bringing down the application. The application may be rebooted with the remaining segment.*

7. *Loss of a bank results in bringing down the application and loss of the bank's segment. The application may be rebooted with the remaining bank and segment.*

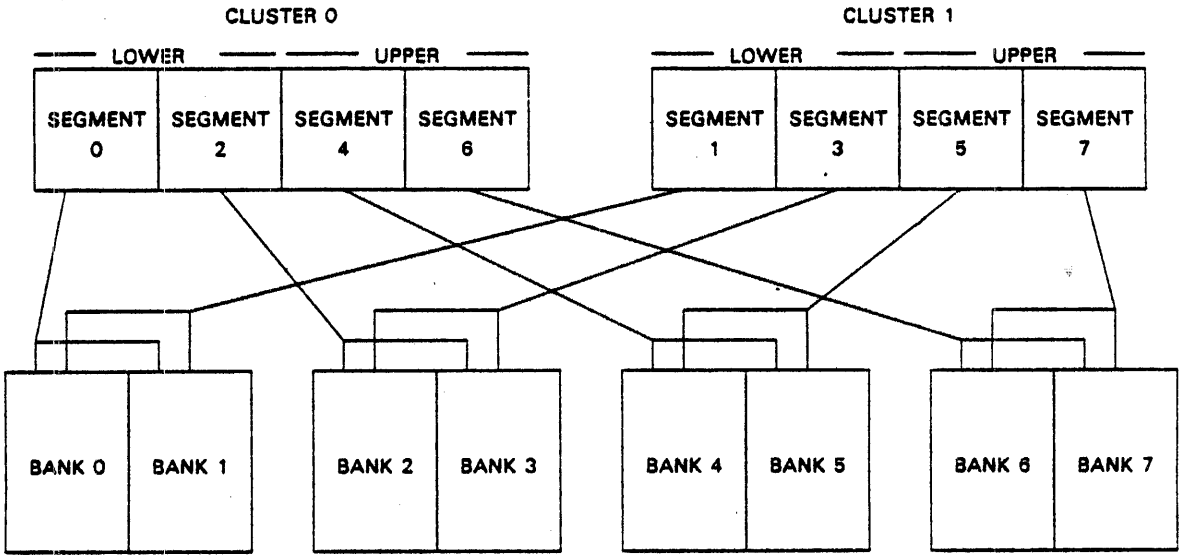
E.4.6.2.2. Partitioning One MSU Out



NOTES:

1. *Interleave between banks 0 and 1 on bit 3 in Application 0.*
2. *Loss of a segment in Application 0 results in loss of the application. No reboot options.*
3. *Loss of a bank results in bringing down the application. The application may be rebooted with the remaining bank.*
4. *Interleave between segments 4 and 6, and 5 and 7 on bit 2 in Application 1.*
5. *Interleave between banks 2 and 3 on bit 3 in Application 1.*
6. *In Application 1, loss of a segment or bank in a half which contains the resident EXEC brings the application down. If the resident EXEC is restricted to one half of storage, removal of a segment or bank in the other half does not necessitate a reboot, but a failure in this other half will probably cause a reboot.*
7. *The application may be rebooted in a single cluster with the three segments for the cluster, provided that this cluster has an IOU. If it is rebooted in this manner, all banks remain available for use but processing power is lessened. The application may also be rebooted as dual cluster with two segments in each cluster. If it is rebooted in this manner, banks connected to the two segments removed are not available for use.*
8. *If the bank lost was the only one connected to two segments, the application may be rebooted with the remaining three banks and four segments. If the bank was one of two connected to a segment, the application may be rebooted with the remaining three banks and all six segments.*

E.4.6.3. Eight Segments/Eight Banks – Dual Cluster



NOTES:

1. *Interleave between segments 0 and 2, 4 and 6, 1 and 3, and 5 and 7 on bit 2.*

2. *Interleave between banks 0 and 1, 2 and 3, 4 and 5, and 6 and 7 on bit 3.*

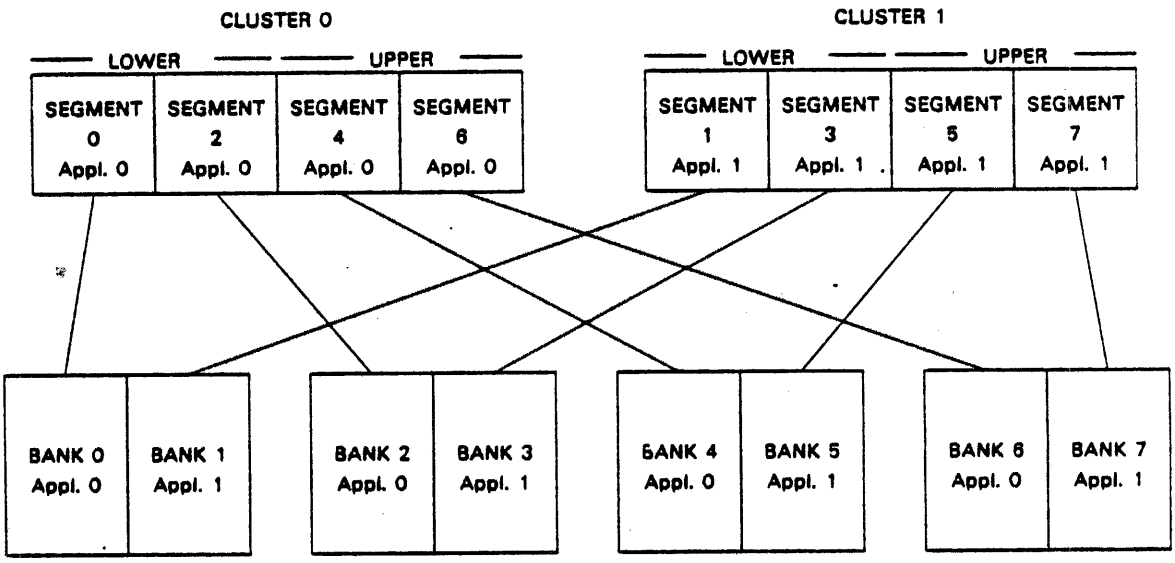
3. *Banks 0, 1, 2, and 3 must be of equal size and banks 4, 5, 6, and 7 must be of equal size or unusable addresses occur. If unusable addresses occur and the banks are in the lower half, MSR cannot be set to SIU generated for bootstrap into the lower half and would have to be manually set on the STU.*

4. *Loss of a segment or bank in a half which contains the resident EXEC brings the application down. If the resident EXEC is restricted to one half of storage, removal of a segment or bank in the other half does not necessitate a reboot, but a failure in this other half will probably cause a reboot.*

5. *The application may be rebooted in a single cluster with the four segments for the cluster, provided that this cluster has an IOU. If it is rebooted in this manner, all banks remain available for use but processing power is lessened. The application may also be rebooted as dual cluster with six segments in each cluster. If it is rebooted in this manner, banks connected to the two segments removed are not available for use.*

6. *For loss of a bank, the application may be rebooted with the unaffected half and either three banks and four segments (see E.4.2.3, Note 3) or four segments and two banks in the affected half.*

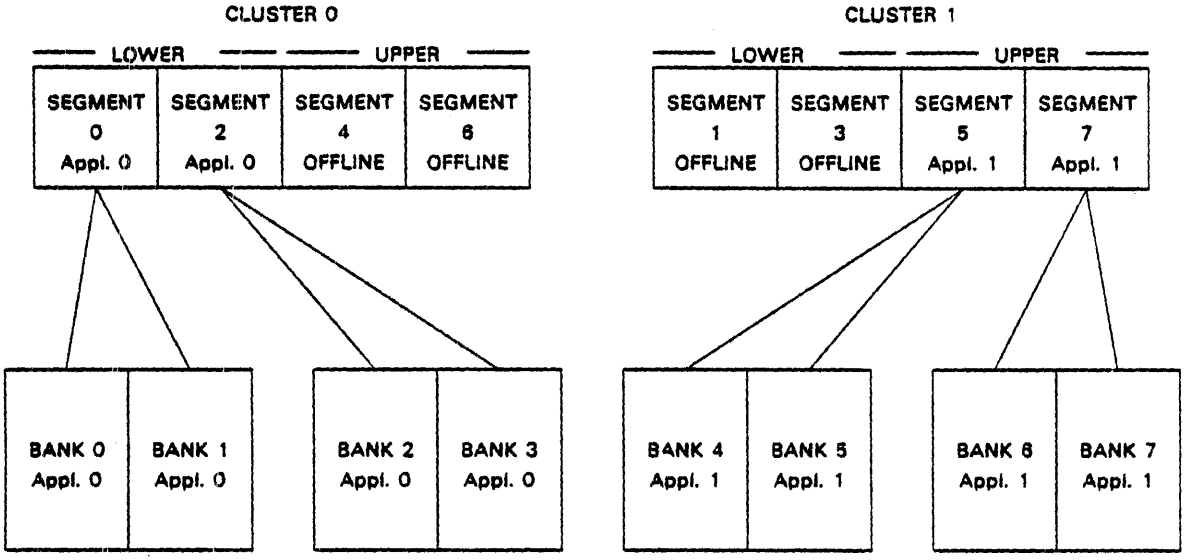
E.4.6.3.1. Partitioned by Cluster Across SIU Halves



NOTES:

1. *Interleave between segments 0 and 2, and 4 and 6 on bit 2 in Application 0.*
2. *Interleave between segments 1 and 3, and 5 and 7 on bit 2 in Application 1.*
3. *Loss of a segment or bank in a half which contains the resident EXEC brings the application down. If the resident EXEC is restricted to one half of storage, removal of a segment or bank in the other half does not necessitate a reboot, but a failure in this other half will probably cause a reboot.*
4. *The application may be rebooted with the remaining segments. If a two segment half was downed dynamically, it may be reentered into the application with the one remaining segment.*
5. *For loss of a bank, the application may be rebooted with the unaffected half and the remaining bank and segment in the affected half.*

E.4.6.3.2. Partitioned by Cluster by SIU Halves



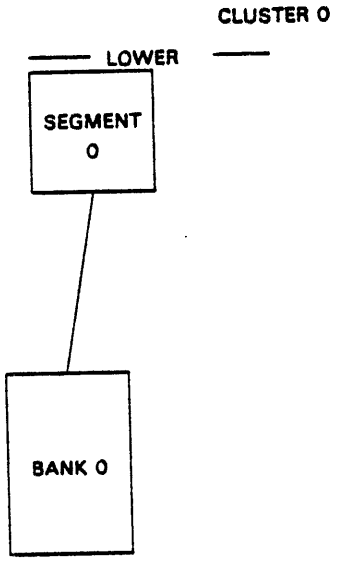
NOTES:

1. *Interleave between segments 0 and 2 on bit 2 in Application 0.*
2. *Interleave between banks 0 and 1, and 2 and 3 on bit 3 in Application 0.*
3. *Interleave between segments 5 and 7 on bit 2 in Application 1.*
4. *Interleave between banks 4 and 5, and 6 and 7 on bit 3 in Application 1.*
5. *Loss of a segment results in bringing down the application. The application may be rebooted with the remaining segment.*
6. *Loss of a bank results in bringing down the application. The application may be rebooted with the two segments and remaining three banks (see E.4.2.3, Note 3) or with two segments and two banks.*

E.5. Segment/Bank Storage Configurations

The following examples illustrate some of the possible segment/bank storage configurations which can be used only in systems involving single clusters. These configurations are different from segment/cabinet configurations in that they require different printed circuit card placement in the STU and different cable routing between the SIU and the MSUs. These configurations cannot be expanded beyond one SIU and two MSUs.

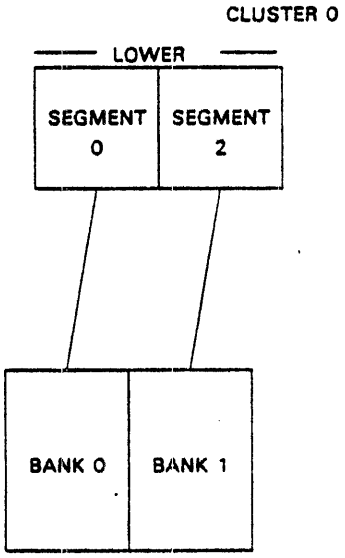
E.5.1. One Segment/One Bank



NOTES:

1. *No interleave.*
2. *Unusable addresses do not occur.*
3. *Loss of a segment or bank results in loss of the application. No reboot options.*

E.5.2. Two Segments/Two Banks



NOTES:

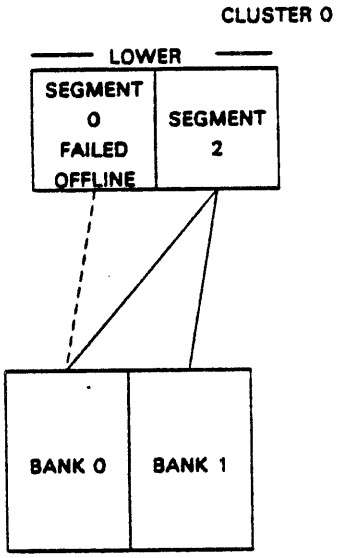
1. *Interleave between segments 0 and 2 on bit 2.*

2. *The banks must be equal in size or unusable addresses occur. If they are not equal in size and are in the lower half, MSR cannot be set to SIU generated for bootstrap into the lower half and would have to be manually set on the STU.*

3. *Loss of a segment results in bringing down the application. The application may be rebooted with the remaining segment and two banks.*

4. *Loss of a bank brings the application down. The application may be rebooted with the remaining bank and its segment.*

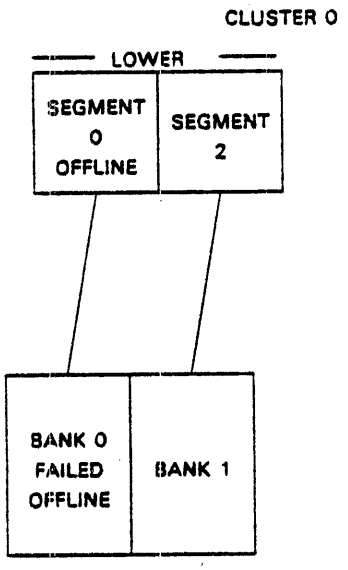
E.5.2.1. Degraded Mode – Failed Segment



NOTES:

1. *Interleave between banks 0 and 1 on bit 3.*
2. *The banks must be equal in size or unusable addresses occur. If they are not equal in size and are in the lower half, MSR cannot be set to SIU generated for bootstrap into the lower half and would have to be manually set on the STU.*
3. *Loss of a segment results in loss of the application. No reboot options.*
4. *Loss of bank brings the application down. The application may be rebooted with the remaining bank and the segment.*

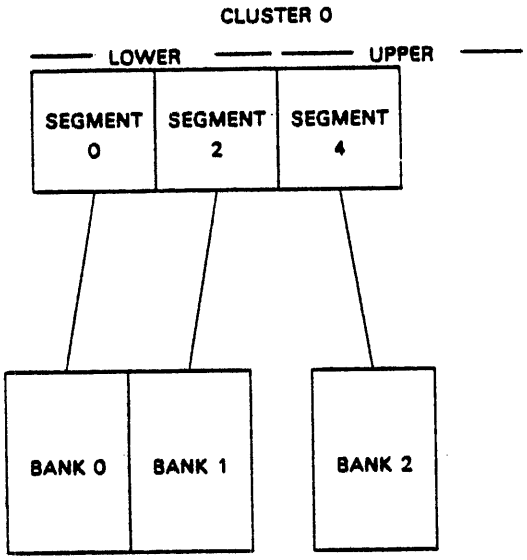
E.5.2.2. Degraded Mode – Failed Bank



NOTES:

1. *No interleave.*
2. *Unusable addresses do not occur.*
3. *Loss of a segment or bank results in loss of the application. No reboot options.*

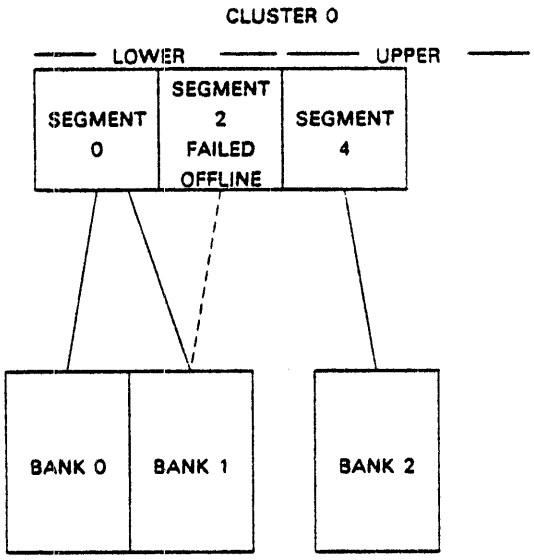
E.5.3. Three Segments/Three Banks



NOTES:

1. *Interleave between segments 0 and 2 on bit 2.*
2. *Unusable addresses do not occur for one bank in a half.*
3. *Banks 0 and 1 must be of equal size or unusable addresses occur. If they are not equal in size and are in the lower half, MSR cannot be set to SIU generated for bootstrap into the lower half and would have to be manually set on the STU.*
4. *Loss of a segment or bank in a half which contains the resident EXEC brings the application down. If the resident EXEC is restricted to one half of storage, removal of a segment or bank in the other half does not necessitate a reboot, but a failure in this other half will probably cause a reboot.*
5. *If the segment was the only one in that half, the application may be rebooted with the unaffected half. If the segment was one of two in a half, the system may be rebooted with the unaffected half and the remaining segment and two banks in the affected half.*
6. *For loss of a bank, the application may be rebooted with the remaining two banks and their segments.*

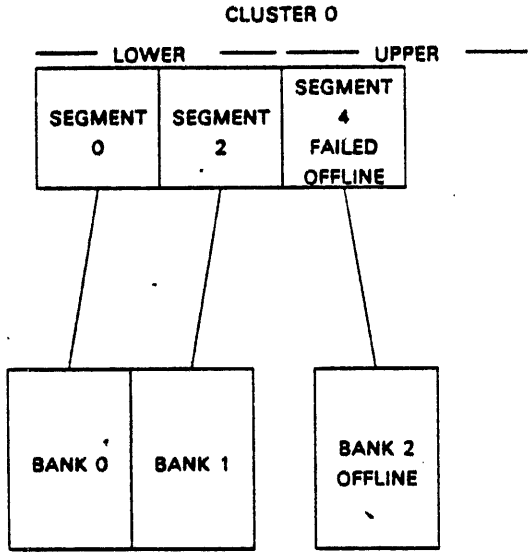
E.5.3.1. Degraded Mode – Failed Lower Segment



NOTES:

1. *Interleave between banks 0 and 1 on bit 3.*
2. *Unusable addresses do not occur for one bank in a half.*
3. *The banks must be equal in size or unusable addresses occur. If they are not equal in size and are in the lower half, MSR cannot be set to SIU generated for bootstrap into the lower half and would have to be manually set on the STU.*
4. *Loss of a segment or bank in a half which contains the resident EXEC brings the application down. If the resident EXEC is restricted to one half of storage, removal of a segment or bank in the other half does not necessitate a reboot, but a failure in this other half will probably cause a reboot.*
5. *For loss of a segment, the application may be rebooted with the unaffected half.*
6. *If the bank lost was the only one in a half, the application may be rebooted with the unaffected half. If the bank was one of two in a half, the application may be rebooted with the unaffected half and the remaining bank and segment in the affected half.*

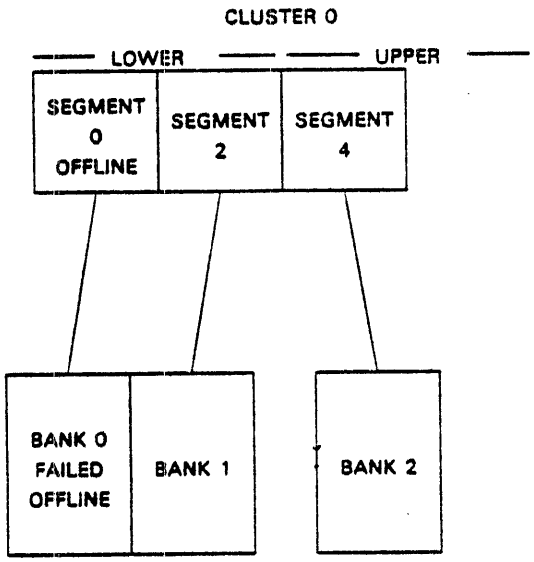
E.5.3.2. Degraded Mode – Failed Upper Segment



NOTES:

1. *Interleave between segments 0 and 2 on bit 2.*
2. *The banks must be equal in size or unusable addresses occur. If they are not equal in size and are in the lower half, MSR cannot be set to SIU generated for bootstrap into the lower half and would have to be manually set on the STU.*
3. *Loss of a segment results in bringing down the application. The application may be rebooted with the remaining segment and two banks.*
4. *Loss of a bank brings the application down. The application may be rebooted with the remaining bank and its segment.*

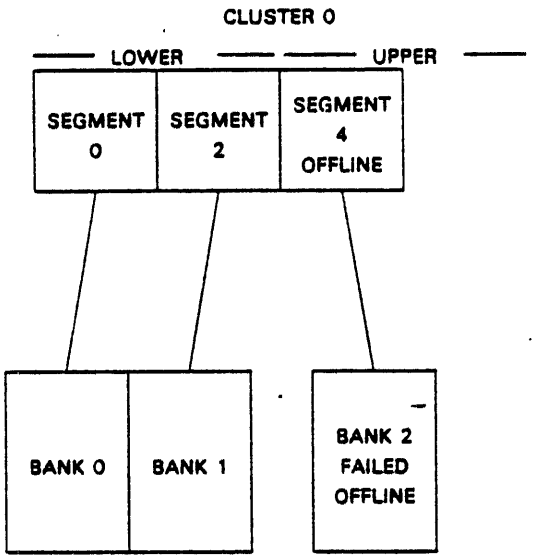
E.5.3.3. Degraded Mode – Failed Lower Bank



NOTES:

1. *No interleave.*
2. *Unusable addresses do not occur for one bank in a half.*
3. *Loss of a segment or bank in a half which contains the resident EXEC brings the application down. If the resident EXEC is restricted to one half of storage, removal of a segment or bank in the other half does not necessitate a reboot, but a failure in this other half will probably cause a reboot.*
4. *For loss of a bank, the application may be rebooted with the unaffected half.*

E.5.3.4. Degraded Mode – Failed Upper Bank



NOTES:

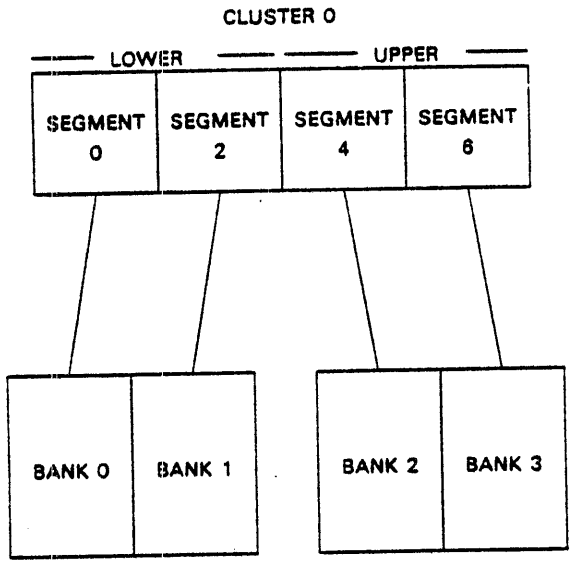
1. *Interleave between segments 0 and 2 on bit 2.*

2. *The banks must be equal in size or unusable addresses occur. If they are not equal in size and are in the lower half, MSR cannot be set to SIU generated for bootstrap into the lower half and would have to be manually set on the STU.*

3. *Loss of a segment results in bringing down the application. The application may be rebooted with the remaining segment and two banks.*

4. *Loss of a bank brings the application down. The application may be rebooted with the remaining bank and its segment.*

E.5.4. Four Segments/Four Banks



NOTES:

1. *Interleave between segments 0 and 2, and 4 and 6 on bit 2.*

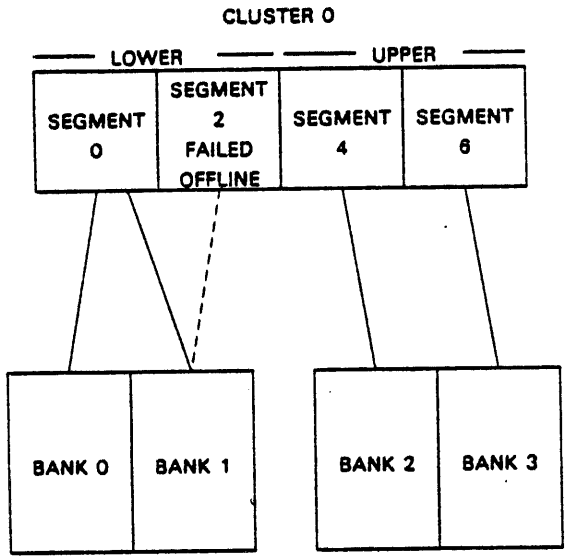
2. *Banks 0 and 1 must be of equal size and banks 2 and 3 must be of equal size or unusable addresses occur. If they are not equal in size and are in the lower half, MSR cannot be set to SIU generated for bootstrap into the lower half and would have to be manually set on the STU.*

3. *Loss of a segment or bank in a half which contains the resident EXEC brings the application down. If the resident EXEC is restricted to one half of storage, removal of a segment or bank in the other half does not necessitate a reboot, but a failure in this other half will probably cause a reboot.*

4. *For loss of a segment, the application may be rebooted with the unaffected half and the remaining segment and two banks in the affected half.*

5. *For loss of a bank, the application may be rebooted with the remaining three banks and their segments.*

E.5.4.1. Degraded Mode – Failed Segment



NOTES:

1. *Interleave between segments 4 and 6 on bit 2.*

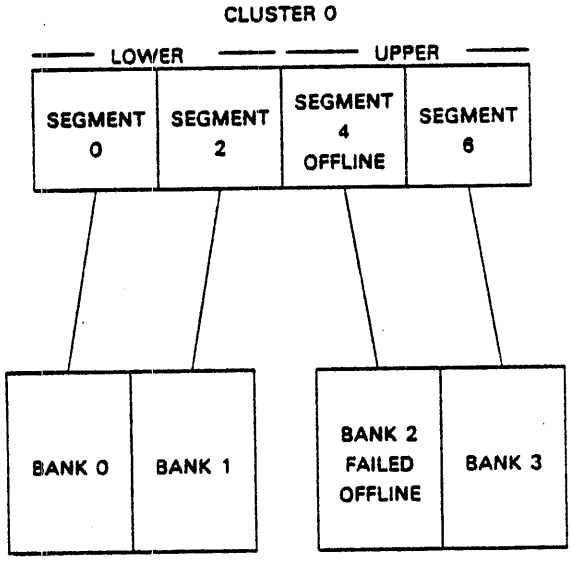
2. *The banks must be equal in size or unusable addresses occur. If they are not equal in size and are in the lower half, MSR cannot be set to SIU generated for bootstrap into the lower half and would have to be manually set on the STU.*

3. *Loss of a segment or bank in a half which contains the resident EXEC brings the application down. If the resident EXEC is restricted to one half of storage, removal of a segment or bank in the other half does not necessitate a reboot, but a failure in this other half will probably cause a reboot.*

4. *If the segment was the only one in that half, the application may be rebooted with the unaffected half. If the segment was one of two in a half, the system may be rebooted with the unaffected half and the remaining segment and two banks in the affected half.*

5. *If the bank lost was one of two connected to a segment, the application may be rebooted with the remaining three banks and all three segments. If the bank was the only one connected to a segment, the application may be rebooted with the remaining three banks and their two segments.*

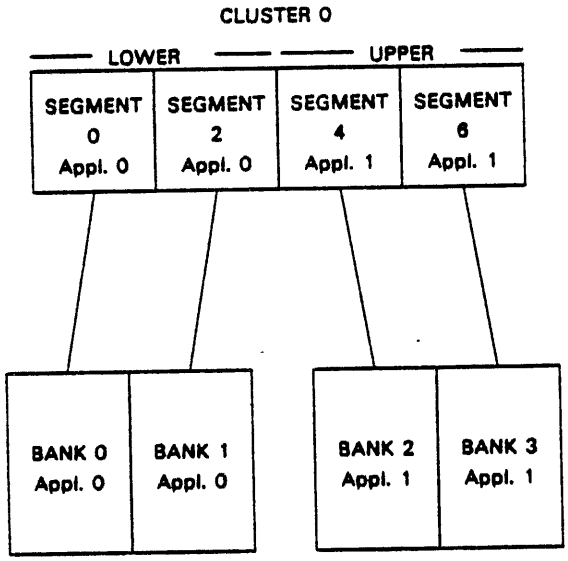
E.5.4.2. Degraded Mode – Failed Bank



NOTES:

1. *Interleave between segments 0 and 2 on bit 2.*
2. *Unusable addresses do not occur for one bank in a half.*
3. *The banks must be equal in size or unusable addresses occur. If they are not equal in size and are in the lower half, MSR cannot be set to SIU generated for bootstrap into the lower half and would have to be manually set on the STU.*
4. *Loss of a segment or bank in a half which contains the resident EXEC brings the application down. If the resident EXEC is restricted to one half of storage, removal of a segment or bank in the other half does not necessitate a reboot, but a failure in this other half will probably cause a reboot.*
5. *If the segment was the only one in that half, the application may be rebooted with the unaffected half. If the segment was one of two in a half, the system may be rebooted with the unaffected half and the remaining segment and two banks in the affected half.*
6. *For loss of a bank, the application may be rebooted with the remaining two banks and their segments.*

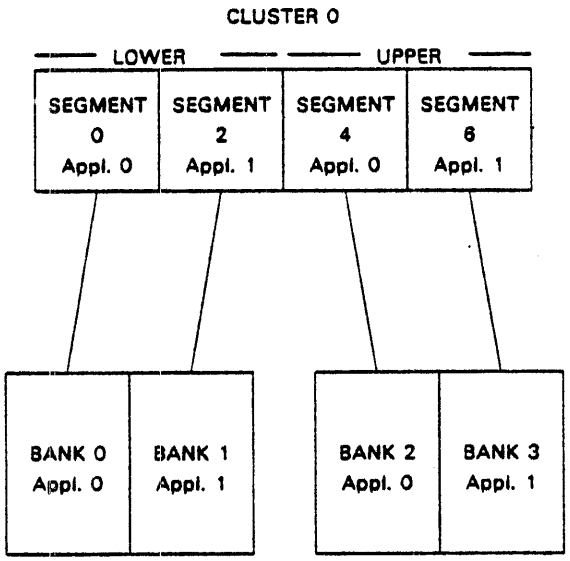
E.5.4.3. Partitioned by SIU Halves



NOTES:

1. *Interleave between segments 0 and 2 on bit 2 in Application 0.*
2. *Interleave between segments 4 and 6 on bit 2 in Application 1.*
3. *The banks must be equal in size or unusable addresses occur. If they are not equal in size and are in the lower half, MSR cannot be set to SIU generated for bootstrap into the lower half and would havve to be manually set on the STU.*
4. *Loss of a segment results in bringing down the application. The application may be rebooted with the remaining segment and two banks.*
5. *Loss of a bank brings the application down. The application may be rebooted with the remaining bank and its segment.*

E.5.4.4. Partitioned Across SIU Halves

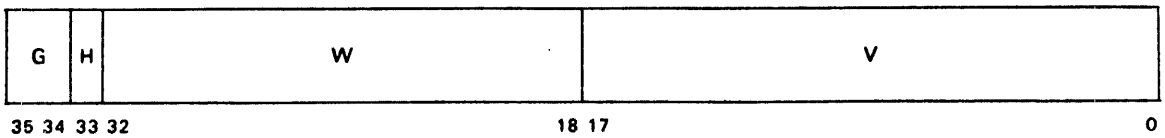


NOTES:

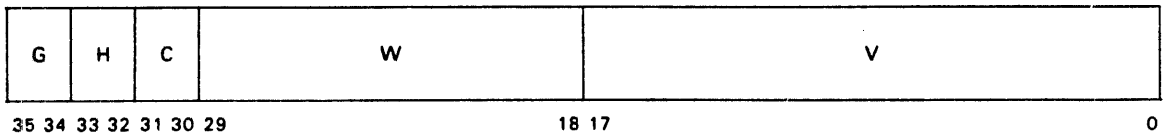
1. *No interleave.*
2. *Unusable addresses do not occur.*
3. *Loss of a segment or bank in a half which contains the resident EXEC brings the application down. If the resident EXEC is restricted to one half of storage, removal of a segment or bank in the other half does not necessitate a reboot, but a failure in this other half will probably cause a reboot.*
4. *The application may be rebooted with the unaffected half.*

Index

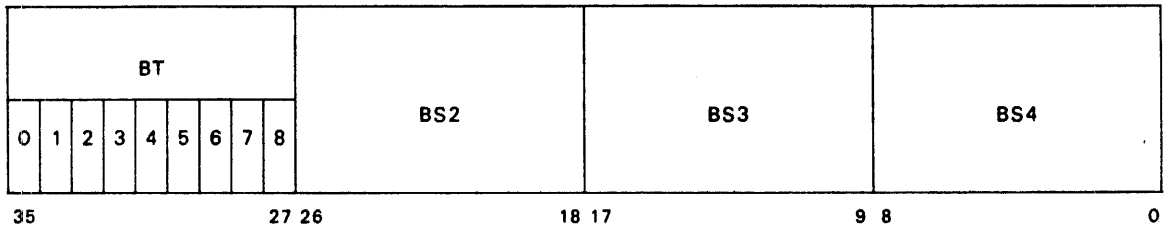
F



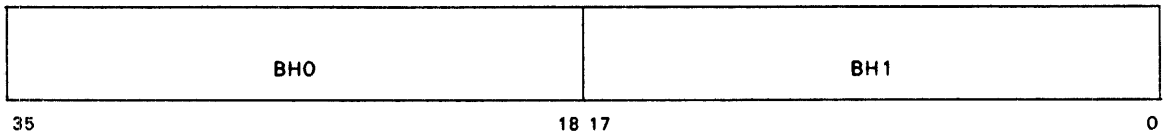
ESI ACCESS CONTROL WORD (QUARTER WORD)



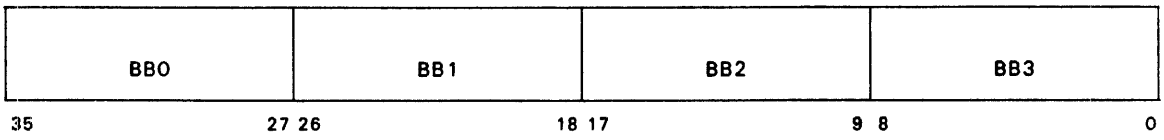
BYTE MANIPULATION STAGING REGISTER 1



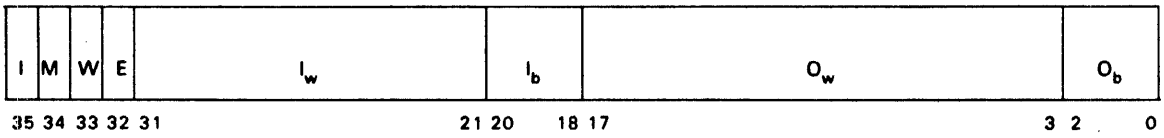
BYTE MANIPULATION STAGING REGISTER 2



BYTE MANIPULATION STAGING REGISTER 3



J REGISTER



Appendix C. Instruction Repertoire and Instruction Times

The estimated instruction times given below are a subset of the actual execution times. Actual execution times are a function of SIU conflicts, storage spigot conflicts, internal CPU instruction sequencing conflicts, register conflicts, timing between CPU and storage, and interrupt processing. Also, system parameters which affect actual execution times are instruction storage access, storage characteristics (i.e., cycle time, storage refresh time, access time, and service time) and cable lengths. See Appendix A for notational conventions used in the description column, and the notes at the end of the table for additional information on the instruction times. For the meaning of the symbol '*' (used in the Mnemonic column), and the symbols 'a' and 'b' (used in the Instruction Time column, see the end of the table.

Function Code (Octal)		Mnemonic	Instruction	Description	Instruction Time (in ns)
f	j				
00	0-17	-	Invalid Code	Causes Invalid Instruction interrupt to address 221_8	1150
01	0-17	S, SA	Store A	$(A_a) \rightarrow U$	200
02	0-17	SN, SNA	Store Negative A	$-(A_a) \rightarrow U$	200
03	0-17	SM, SMA	Store Magnitude A	$\lvert(A_a)\rvert \rightarrow U$	200
04	0-17	S, SR	Store R	$(R_a) \rightarrow U$	200
05	0-17	SZ (a=0)	Store Zero	000000 000000 $\rightarrow$ U	200

Function Code (Octal)		Mnemonic	Instruction	Description	Instruction Time (in ns)
f	j				
05	0-17	SNZ (a=1)	Store Negative Zero	777777 777777 → U	200
05	0-17	SP1 (a=2)	Store Positive One	000000 000001 → U	200
05	0-17	SN1 (a=3)	Store Negative One	777777 777776 → U	200
05	0-17	SFS (a=4)	Store Fieldata Blanks	050505 050505 → U	200
05	0-17	SFZ (a=5)	Store Fieldata Zeros	606060 606060 → U	200
05	0-17	SAS (a=6)	Store ASCII Blanks	040040 040040 → U	200
05	0-17	SAZ (a=7)	Store ASCII Zeros	060060 060060 → U	200
05	0-17	INC (a=10)	Increase	(U) + 1 → U. If initial or final value of U=0, execute N; else skip NI	550/750
05	0-17	DEC (a=11)	Decrease	(U) - 1 → U. If initial or final value of U=0, execute NI; else skip NI	550/750
05	0-17	INC 2 (a=12)	Increase by 2	(U) + 2 → U. If initial or final value of U=0, execute NI; else skip NI	550/750
05	0-17	DEC 2 (a=13)	Decrease by 2	(U) - 2 → U. If initial or final value of U=0, execute NI; else skip NI	550/750

- see p. 183

Function Code (Octal)		Mnemonic	Instruction	Description	Instruction Time (in ns)
f	j				
05	0–17	ENZ (a=14–17)	Eliminate Negative Zero	Negative 0 is changed to positive 0 for sign-extended operands	550
06	0–17	S, SX	Store X	$(X_a) \rightarrow U$	200
07	0–11	-	Invalid Code	Causes Invalid Instruction interrupt to address 221_8	1150
07	12	LDJ	Load D Bank Base and Jump	Select BDR specified by D12; if D12=0 select BDR2; if D12=1 select BDR3; load BDR with new BD located by BDI and BDTP specified in (X_a); old BDI and P+1 $\rightarrow X_a$; jump to address U	2550
07	13	LIJ	Load I Bank Base and Jump	Select BDR specified by D12; if D12=0 select BDR0; if D12=1 select BDR1; load BDR with new BD located by BDI and BDTP specified in (X_a); old BDI and P+1 $\rightarrow X_a$; jump to address U	2550
07	14	LPD	Load Designators	$(U)_{6,5,3-0} \rightarrow$ Designator Register: $(U)_0 \rightarrow$ D4 $(U)_3 \rightarrow$ D10 $(U)_1 \rightarrow$ D5 $(U)_5 \rightarrow$ D17 $(U)_2 \rightarrow$ D8 $(U)_6 \rightarrow$ D20	600
07	15	SPD (a=0)	Store Designators	Designator Register D-bits $\rightarrow U_{6-0}$, zeros $\rightarrow U_{17-7}$: D4 $\rightarrow U_0$ D12 $\rightarrow U_4$ D5 $\rightarrow U_1$ D17 $\rightarrow U_5$ D8 $\rightarrow U_2$ D20 $\rightarrow U_6$ D10 $\rightarrow U_3$	200
07	16		Invalid Code	Causes Invalid Instruction interrupt to address 221_8	1150
07	17	LBJ	Load Bank and Jump	Load BDR specified by $(X_a)_{34,33}$ with new BD located by BDI and BDTP specified in (X_a); old BDI and P+1 $\rightarrow X_a$; jump to address U	2550
10	0–17	L, LA	Load A	$(U) \rightarrow A_a$	200

Function Code (Octal)		Mnemonic	Instruction	Description	Instruction Time (in ns)
f	j				
11	0–17	LN, LNA	Load Negative A	$-(U) \rightarrow A_a$	200
12	0–17	LM, LMA	Load Magnitude A	$\lvert (U) \rvert \rightarrow A_a$	200
13	0–17	LNM, LNMA	Load Negative Magnitude A	$-\lvert (U) \rvert \rightarrow A_a$	200
14	0–17	A, AA	Add to A	$(A_a)+(U) \rightarrow A_a$	200
15	0–17	AN, ANA	Add Negative To A	$(A_a) - (U) \rightarrow A_a$	200
16	0–17	AM, AMA	Add Magnitude To A	$(A_a)+\lvert (U) \rvert \rightarrow A_a$	200
17	0–17	ANM, ANMA	Add Negative Magnitude To A	$(A_a) - \lvert (U) \rvert \rightarrow A_a$	200
20	0–17	AU	Add Upper	$(A_a)+(U) \rightarrow A_a+1$	250
21	0–17	ANU	Add Negative Upper	$(A_a) - (U) \rightarrow A_a+1$	250
22	0–15	BT	Block Transfer, Repeat	$(X_a+u) \rightarrow X_a+u$; repeat K times	1050 + 200K
23	0–17	L, LR	Load R	$(U) \rightarrow R_a$	200
24	0–17	A, AX	Add To X	$(X_a)+(U) \rightarrow X_a$	200
25	0–17	AN, ANX	Add Negative To X	$(X_a) - (U) \rightarrow X_a$	200
26	0–17	LXM	Load X Modifier	$(U) \rightarrow X_{a17-0}$; X_{a35-18} unchanged	200
27	0–17	L, LX	Load X	$(U) \rightarrow X_a$	200
30	0–17	MI	Multiply Integer	$(A_a)\cdot(U) \rightarrow A_a,A_a+1$	1150 + 100b
31	0–17	MSI	Multiply Single Integer	$(A_a)\cdot(U) \rightarrow A_a$	1100 + 100b
32	0–17	MF	Multiply Fractional	$(A_a)\cdot(U) \rightarrow A_a,A_a+1$	1250 + 100b

b = Number of arithmetic loops

Function Code (Octal)		Mnemonic	Instruction	Description	Instruction Time (in ns)
f	j				
33	00	BM	Byte Move	Transfer N bytes from source string to receiving string. Truncate or fill receiving string as required	2100 + 500N
33	01	BMT	Byte Move With Translate	Translate and transfer N bytes from source string to receiving string. Truncate or fill receiving string as required	2100 + 1200N
33	02	BBT*	Byte Translate and Test	Translate and test N bytes against (A); if not equal, terminate instruction with J0 pointing to unequal byte and N≠0	
33	03	BTC	Byte Translate and Compare	Translate and compare N bytes from string E to M bytes from string F; terminate instruction on not equal or when both M and N have been reduced to zero; when: $(A_0) > 0$; string E > F $(A_0) = 0$; string E = F $(A_0) < 0$; string E < F	1600 + 2000N
33	04	BC	Byte Compare	Compare N bytes from string E to M bytes from string F; terminate instruction on not equal or when both M and N are zero	1600 + 2000N
33	05	BPD*	Byte to Packed Decimal Convert	Convert N bytes in string E to packed decimal in string F	-
33	06	PDB*	Packed Decimal to Byte Convert	Convert N packed decimal digits in string E to bytes in string F	-
33	07	EDIT	Edit	Edit byte string E and transfer to byte string F under the control of string G (see Note 2 (b))	
				Skip	4600 + 500N_1
				Editing action	4600 + 2300N_2
				Blank if zero	4600 + 1000N_2

* = Simulated in software by the Executive System.

Function Code (Octal)		Mnemonic	Instruction	Description	Instruction Time (in ns)
f	j				
33	10	BI	Byte to Binary Single Integer Convert	Convert N bytes in string E into a signed binary integer in register A_a	1800 + 750N
33	11	BDI	Byte to Binary Double Integer Convert	Convert N bytes in string E into a signed binary integer in registers A_a and A_a+1	1950 + 750N
33	12	IB	Binary Single Integer to Byte Convert	Convert the binary integer in A_a to byte format and store in string E	8850 + 500N
33	13	DIB	Binary Double Integer to Byte Convert	Convert the binary integer in A_a and A_a+1 to byte format and store in string E	9100 + 500N
33	14	BF	Byte to Single Floating Convert	Convert N bytes in string E into a single length floating point format in register A_a	13,850+ 1050N
33	15	BDF	Byte to Double Floating Convert	Convert N bytes in string E into a double length floating point format in registers A_a, A_a+1	13,850+ 1050N
33	16	FB	Single Floating to Byte Convert	Convert the single length floating point number in A_a to byte format and store in string E	5150 + 1050N
33	17	DFB	Double Floating to Byte Convert	Convert the double length floating point number in A_a and A_a+1 to byte format and store in string E	5150 + 1050N
34	0-17	DI	Divide Integer	$(A_a, A_a+1) \div (U) \rightarrow A_a$; REMAINDER $\rightarrow A_a+1$	1400 + 100b
35	0-17	DSF	Divide Single Fractional	$(A_a) \div (U) \rightarrow A_a+1$	5800
36	0-17	DF	Divide Fractional	$(A_a, A_a+1) \div (U) \rightarrow A_a$; REMAINDER $\rightarrow A_a+1$	5800
37	00	QB*	Quarter-Word Byte	Discard $(A)_{35}$, $(A)_{26}$,$(A)_{17}$, and $(A)_8$; place the remaining bits in A_{31-0}; $(A)_{31} \rightarrow A_{35-32}$	-
37	01	BQ*	Binary to Quarter-Word Byte Extend	Discard $(A)_{35-32}$; place the remaining bits in A_{34-27}, A_{25-18}, A_{16-9}, and A_{7-0}; zero fill A_{35}, A_{26}, A_{17}, and A_8	-

* = Simulated in software by the Executive System.

b = Number of arithmetic loops

Function Code (Octal)		Mnemonic	Instruction	Description	Instruction Time (in ns)
f	j				
37	02	QBH*	Quarter-Word Byte to Binary Halves Compress	Discard $(A)_{35}$, $(A)_{26}$, $(A)_{17}$, and $(A)_{8}$; place the remaining bits in A_{33-18} and A_{15-0}; $(A)_{33} \rightarrow A_{35-34}$; $(A)_{15} \rightarrow A_{17-16}$	
37	03	BHQ*	Binary Halves to Quarter- Word Byte Extend	Discard $(A)_{35-34}$ and $(A)_{17-16}$; place the remaining bits in A_{34-27}, A_{25-18}, A_{16-9}, and A_{7-0}; zero fill A_{35}, A_{26}, A_{17}, and A_{8}	-
37	04	QDB*	Quarter-Word Byte to Double Binary Compress	Discard A_{35}, A_{26}, A_{17}, A_{8}, $A+1_{35}$, $A+1_{26}$, $A+1_{17}$, and $A+1_{8}$; place the remaining bits in A_{27-0} and $A+1$; $(A)_{27} \rightarrow A_{35-28}$	-
37	05	DBQ*	Double Binary to Quarter-Word Byte Extend	Discard $(A)_{35-28}$; place the remaining bits from A and $A+1$ in A_{34-27}, A_{25-18}, A_{16-9}, A_{7-0}, $A+1_{34-27}$, $A+1_{25-18}$, $A+1_{16-9}$, and $A+1_{7-0}$; zero fill A_{35}, A_{26}, A_{17}, A_{8}, $A+1_{35}$, $A+1_{26}$, $A+1_{17}$, and $A+1_{8}$	-
37	06	BA	Byte Add	Add the N bytes in string E to the M bytes in string F and place the result in string G (see Note 7)	1300 + 1500/ 2500N
37	07	BAN	Byte Add Negative	Subtract the N bytes in string E from the M bytes in string F and place the result in string G (see Note 7)	1300 + 1500/ 2500N
37	10–17	-	Invalid Code	Causes invalid instruction interrupt to address 221_{8}	-
40	0–17	OR	Logical OR	(A_a) OR (U) $\rightarrow A_a+1$	250
41	0–17	XOR	Logical Exclusive OR	(A_a) XOR (U) $\rightarrow A_a+1$	250
42	0–17	AND	Logical AND	(A_a) AND (U) $\rightarrow A_a+1$	250
43	0–17	MLU	Masked Load Upper	[(U) AND (R2)] OR [(A_a) AND (R2)'] $\rightarrow A_a+1$	550
44	0–17	TEP	Test Even Parity	Skip NI if (U) AND (A_a) has even parity	450/650
45	0–17	TOP	Test Odd Parity	Skip NI if (U) AND (A_a) has odd parity	450/650
46	0–17	LXI	Load X Increment	(U) $\rightarrow (X_a)_{35-18}$; $(X_a)_{17-0}$ unchanged	200

* = Simulated in software by the Executive System.

Function Code (Octal)		Mnemonic	Instruction	Description	Instruction Time (in ns)
f	j				
47	0–17	TLEM	Test Less Than or Equal to Modifier	Skip NI if (U) ≤ $(X_a)_{17-0}$; always $(X_a)_{17-0}$ + $(X_a)_{35-18} \rightarrow X_{a17-0}$	600/800
		TNGM	Test Not Greater Modifier	Alternate mnemonic for TLEM instruction	
50	0–17	TZ	Test Zero	Skip NI if (U)=±0	350/550
51	0–17	TNZ	Test Nonzero	Skip NI if (U) ≠±0	350/550
52	0–17	TE	Test Equal	Skip NI if (U)=(A_a)	350/550
53	0–17	TNE	Test Not Equal	Skip NI if (U) ≠ (A_a)	350/550
54	0–17	TLE	Test Less Than or Equal	Skip NI if (U) ≤ (A_a)	450/650
		TNG	Test Not Greater		
55	0–17	TG	Test Greater	Skip NI if (U)>(A_a)	450/650
56	0–17	TW	Test Within Range	Skip NI if (A_a)<(U) ≤ (A_a+1)	500/700
57	0–17	TNW	Test Not Within Range	Skip NI if (U) ≤ (A_a) or (U)>(A_a+1)	500/700
60	0–17	TP	Test Positive	Skip NI if $(U)_{35}=0$	350/550
61	0–17	TN	Test Negative	Skip NI if $(U)_{35}=1$	350/550
62	0–17	SE	Search Equal	Skip NI if (U)=(A_a), else repeat	1050 + 200a
63	0–17	SNE	Search Not Equal	Skip NI if (U) ≠ (A_a), else repeat	1050 + 200a
64	0–17	SLE	Search Less Than or Equal	Skip NI if (U) ≤ (A_a), else repeat	1050 + 200a
		SNG	Search Not Greater	Alternate mnemonic for SLE instruction	
65	0–17	SG	Search Greater	Skip NI if (U) > (A_a), else repeat	1050 + 200a

a = Number of words

Function Code (Octal)		Mnemonic	Instruction	Description	Instruction Time (in ns)
f	j				
66	0–17	SW	Search Within Range	Skip NI if $(A_a)<(U) \leq (A_a+1)$, else repeat	1100 + 200a
67	0–17	SNW	Search Not Within Range	Skip NI if $(U) \leq (A_a)$ or $(U)>(A_a+1)$, else repeat	1100 + 200a
70	0–17	JGD	Jump Greater and Decrement	Jump to U if $(\text{Control Register}_{ja})>0$; go to NI if $(\text{Control Register}_{ja}) \leq 0$; always $(\text{Control Register}_{ja})-1 \rightarrow \text{Control Register}_{ja}$	200/300
71	00	MSE	Mask Search Equal	Skip NI if (U) AND (R2) = (A_a) AND (R2), else repeat	1050 + 200a
71	01	MSNE	Mask Search Not Equal	Skip NI if (U) AND (R2) ≠ (A_a) AND (R2), else repeat	1050 + 200a
71	02	MSLE	Mask Search Less Than or Equal	Skip NI if (U) AND (R2) ≤ (A_a) AND (R2), else repeat	1050 + 200a
		MSNG	Mask Search Not Greater	Alternate mnemonic for MSLE instruction	
71	03	MSG	Mask Search Greater	Skip NI if (U) AND (R2) > (A_a) AND (R2), else repeat	1050 + 200a
71	04	MSW	Masked Search Within Range	Skip NI if (A_a) AND (R2) < (U) AND (R2) ≤ (A_a+1) AND (R2), else repeat	1100 + 200a
71	05	MSNW	Masked Search Not Within Range	Skip NI if (U) AND (R2) ≤ (A_a) AND (R2) or (U) AND (R2) > (A_a+1) AND (R2), else repeat	1100 + 200a
71	06	MASL	Masked Alpha-numeric Search Less Than or Equal	Skip NI if (U) AND (R2) ≤ (A_a) AND (R2), else repeat	1050 + 200a
71	07	MASG	Masked Alphanumeric Search Greater	Skip NI if (U) AND (R2) > (A_a) AND (R2), else repeat	1050 + 200a
71	10	DA	Double Precision Fixed-Point Add	$(A_a, A_a+1) + (U, U+1) \rightarrow A_a, A_a+1$	450

a = Number of words

Function Code (Octal)		Mnemonic	Instruction	Description	Instruction Time (in ns)
f	j				
71	11	DAN	Double Precision Fixed-Point Add Negative	$(A_a, A_a+1) - (U,U+1) \rightarrow A_a, A_a+1$	450
71	12	DS	Double Store A	$(A_a, A_a+1) \rightarrow U, U+1$	400
71	13	DL	Double Load A	$(U, U+1) \rightarrow A_a, A_a+1$	400
71	14	DLN	Double Load Negative A	$-(U, U+1) \rightarrow A_a, A_a+1$	400
71	15	DLM	Double Load Magnitude A	$\lvert (U, U+1) \rvert \rightarrow A_a, A_a+1$	400
71	16	DJZ	Double Precision Jump Zero	Jump to U if $(A_a, A_a+1) = \pm 0$; go to NI if $(A_a, A_a+1) \neq \pm 0$	250/350
71	17	DTE	Double Precision Test Equal	Skip NI if $(U, U+1) = (A_a, A_a+1)$	550/750
72	00	IMI	Initiate Maintenance Interrupt	Send Attention interrupt to Maintenance Processor if in maintenance mode, otherwise NOP	200
72	01	SLJ	Store Location and Jump	(P)–Base Address Modifier $\rightarrow U_{17-0}$; jump to U+1	400
72	02	JPS	Jump Positive and Shift	Jump to U if $(A_a)_{35}=0$; go to NI if $(A_a)_{35}=1$; always shift (A_a) left circularly one bit position	200/300
72	03	JNS	Jump Negative and Shift	Jump to U if $(A_a)_{35}=1$; go to NI if $(A_a)_{35}=0$; always shift (A_a) left circularly one bit position	200/300
72	04	AH	Add Halves	$(A_a)_{35-18}+(U)_{35-18} \rightarrow (A_a)_{35-18}$; $(A_a)_{17-0}+(U)_{17-0} \rightarrow A_{a17-0}$	500
72	05	ANH	Add Negative Halves	$(A_a)_{35-18}-(U)_{35-18} \rightarrow (A_a)_{35-18}$; $(A_a)_{17-0}-(U)_{17-0} \rightarrow A_{a17-0}$	500
72	06	AT	Add Thirds	$(A_a)_{35-24}+(U)_{35-24} \rightarrow A_{a35-24}$; $(A_a)_{23-12}+(U)_{23-12} \rightarrow A_{a23-12}$	800

Function Code (Octal)		Mnemonic	Instruction	Description	Instruction Time (in ns)
f	j				
				$(A_a)_{11-0} + (U)_{11-0} \rightarrow A_{a11-0}$	
72	07	ANT	Add Negative Thirds	$(A_a)_{35-24} - (U)_{35-24} \rightarrow A_{a35-24}$; $(A_a)_{23-12} - (U)_{23-12} \rightarrow A_{a23-12}$; $(A_a)_{11-0} - (U)_{11-0} \rightarrow A_{a11-0}$	850
72	10	EX	Execute	Execute the instruction at U	300
72	11	ER	Executive Request	Causes Executive Request interrupt to address 222_8	1250
72	12	-	Invalid Code	Causes Invalid Instruction interrupt to address 221_8	1150
72	13	PAIJ	Prevent all Interrupts and Jump	Disable all interrupts and jump to U	300
72	14	-	Invalid Code	Causes Invalid Instruction interrupt to address 221_8	1150
72	15	TRA	Test Relative Address	Test a given relative address against limits (see Note 8)	2050/ 2850
72	16	SRS	Store Register Set	Transfer GRS areas defined in A_a to storage starting at address U	800 + 200a
72	17	LRS	Load Register Set	Transfer from storage starting at location U to GRS areas defined in A_a	800 + 200a
73	00	SSC	Single Shift Circular	Shift (A_a) right circularly U places	350
73	01	DSC	Double Shift Circular	Shift (A_a, A_a+1) right circularly U places	400
73	02	SSL	Single Shift Logical	Shift (A_a) right U places; zero fill	200
73	03	DSL	Double Shift Logical	Shift (A_a, A_a+1) right U places; zero fill	300
73	04	SSA	Single Shift Algebraic	Shift (A_a) right U places; sign fill	200
73	05	DSA	Double Shift Algebraic	Shift (A_a, A_a+1) right U places; sign fill	300

a = Number of words

Function Code (Octal)		Mnemonic	Instruction	Description	Instruction Time (in ns)
f	j				
73	06	LSC	Load Shift and Count	(U) → A_a, shift (A_a) left circularly until $(A_a)_{35} \neq (A_a)_{34}$; NUMBER OF SHIFTS → A_a+1	550
73	07	DLSC	Double Load Shift and Count	(U,U+1) → A_a, A_a+1; shift (A_a, A_a+1) left circularly until $(A_a, A_a+1)_{71} \neq (A_a, A_a+1)_{70}$; NUMBER OF SHIFTS → A_a+2	1350
73	10	LSSC	Left Single Shift Circular	Shift (A_a) left circularly U places	350
73	11	LDSC	Left Double Shift Circular	Shift (A_a, A_a+1) left circularly U places	400
73	12	LSSL	Left Single Shift Logical	Shift (A_a) left U place; zero fill	200
73	13	LDSL	Left Double Shift Logical	Shift (A_a, A_a+1) left U places; zero fill	300
73	14	LDC (a=10)	Load Dayclock	(MSR + 216_8) → dayclock at start of next update cycle	200
73	14	EDC (a=11)	Enable Dayclock	Enable dayclock	200
73	14	DDC (a=12)	Disable Dayclock	Disable dayclock	200
73	14	SDC (a=13)	Select Dayclock	Select dayclock in processor number U	200
73	14	MDA (a=14)	Diagnostics A	Generates A, A+1 and U, U+1 operands to test arithmetic; stores results, A=00372706711 and A+1 = 256171354400 in GRS addresses 62_8 and 63_8	4850
73	14	MDB (a=15)	Diagnostics B	Generates A, A+1 and U, U+1 operands to test arithmetic; stores results, A=771177117711 and A+1=777777776677 in GRS addresses 62_8 and 63_8	4850

Function Code (Octal)		Mnemonic	Instruction	Description	Instruction Time (in ns)
f	j				
73	14	NOP (a=16,17)	No Operation	Proceed to NI	200
73	15	SIL (a=0)	Select Interrupt Location	$(U)_{22-16}$ → MSR specified by $(U)_{23}$	600
73	15	– (a=1)	Invalid Code	Causes Invalid Instruction interrupt to address 221_8	1150
73	15	LBRX (a=2)	Load Breakpoint Register	(U) → Breakpoint Register (see Note 9)	550/750
73	15	LQT (a=3)	Load Quantum Timer	(U) → Quantum Timer	500
73	15	III (a=4)	Initiate Interprocessor Interrupt	Interrupt CPU specified by U	200
73	15	SPID (a=5)	Store Processor ID	Store processor serial number, revision level, features provided, and system number in U	200
73	15	RAT (a=6)	Reset Auto-Recovery Timer	Set auto-recovery timer to zero	200
73	15	TAP (a=7)	Toggle Auto-Recovery Path	Toggle auto-recovery path selection after each attempt	200
73	15	LB (a=10)	Load Base	$(U)_{17-0}$ → BDR specified by $(X_x)_{34,33}$	600
73	15	LL (a=11)	Load Limits	$(U)_{35-24}$ and $(U)_{23-15}$ → BDR limits fields specified $(X_x)_{34,33}$	600

Function Code (Octal)		Mnemonic	Instruction	Description	Instruction Time (in ns)
f	j				
73	15	LAE (a=12)	Load Addressing Environment	(U,U+1) → BDI Registers in GRS locations 46_8 and 47_8; BD Limits and Base Values to BDRs.	1850
73	15	SQT (a=13)	Store Quantum Timer	(Quantum Timer) → U	200
73	15	LD (a=14)	Load Designator Register	(U) → Designator Register	650
73	15	SD (a=15)	Store Designator Register	(Designator Register) → U	450
73	15	UR (a=16)	User Return	(U+1) → Designator Register; jump to address specified by $(U)_{23-0}$ using new register set	850
73	15	SSS (a=17)	Store System Status	System Status → U,U+1	1050
73	16	-	Invalid Code	Causes Invalid Instruction interrupt to address 221_8	1150
73	17	TS (a=0)	Test and Set	If $(U)_{30}=1$, interrupt; if $(U)_{30}=0$, go to NI, and if U ⩾ 200 set $(U)_{35-30}$ to 01_8	1450/400
73	17	TSS (a=1)	Test and Set and Skip	If $(U)_{30}=1$, go to NI; if $(U)_{30}=0$, skip, and if U ⩾ 200 set $(U)_{35-30}$ to 01_8	400/600
73	17	TCS (a=2)	Test and Clear and Skip	If $(U)_{30}=0$, go to NI; if $(U)_{30}=1$, skip, and if U ⩾ 200 clear $(U)_{35-30}$	400/600
73	17	TSA (a=4)	Test and Set Alternate	If $(U)_{14}=1$ interrupt; if $(U)_{14}=0$ go to NI; always set $(U)_{14-0}$ to ones	1450/400

Function Code (Octal)		Mnemonic	Instruction	Description	Instruction Time (in ns)
f	j				
73	17	TSSA (a=5)	Test and Set and Skip Alternate	If $(U)_{14}=1$ go to NI, if $(U)_{14}=0$ skip NI; always set $(U)_{14-0}$ to ones	400/600
74	00	JZ	Jump Zero	Jump to U if $(A_a) = \pm 0$; go to NI if $(A_a) \neq \pm 0$	200/300
74	01	JNZ	Jump Nonzero	Jump to U if $(A_a) \neq \pm 0$; go to NI if $(A_a) = \pm 0$	200/300
74	02	JP	Jump Positive	Jump to U if $(A_a)_{35}=0$; go to NI if $(A_a)_{35}=1$	200/300
74	03	JN	Jump Negative	Jump to U if $(A_a)_{35}=1$; go to NI if $(A_a)_{35}=0$	200/300
74	04	J (a=0)	Jump	Jump to U	300
74	04	JK (a≠0)	Jump Keys	Jump to U if a=lit SELECT JUMP indicator	200/300
74	05	HJ (a=0)	Halt Jump	Jump to U and halt	Halt
74	05	HKJ (a≠0)	Halt Keys and Jump	Jump to U; stop if a AND lit SELECT STOP indicators ≠ 0	200/Halt
74	06	NOP	No Operation	Proceed to NI	200
74	07	AAIJ	Allow All Interrupts and Jump	Allow all interrupts and jump to U	300
74	10	JNB	Jump No Low Bit	Jump to U if $(A_a)_0=0$; go to NI if $(A_a)_0=1$	200/300
74	11	JB	Jump Low Bit	Jump to U if $(A_a)_0=1$; go to NI if $(A_a)_0=0$	200/300
74	12	JMGI	Jump Modifier Greater and Increment	Jump to U if $(X_a)_{17-0}>0$; go to NI if $(X_a)_{17-0} \leq 0$; always $(X_a)_{17-0} + (X_a)_{35-0} \rightarrow X_{a17-0}$	200/300
74	13	LMJ	Load Modifier and Jump	(P) - Base Address Modifier $\rightarrow (X_a)_{17-0}$; jump to U	300

Function Code (Octal)		Mnemonic	Instruction	Description	Instruction Time (in ns)
f	j				
74	14	JO (a=0)	Jump Overflow	Jump to U if D1=1; go to NI if D1=0	200/300
74	14	JFU (a=1)	Jump Floating Underflow	Jump to U if D21=1; go to NI if D21=0; clear D21	200/300
74	14	JFO (a=2)	Jump Floating Overflow	Jump to U if D22=1; go to NI if D22=0; clear D22	200/300
74	14	JDF (a=3)	Jump Divide Fault	Jump to U if D23=1; go to NI if D23=0; clear D23	200/300
74	15	JNO (a=0)	Jump No Overflow	Jump to U if D1=0; go to NI if D1=1	200/300
74	15	JNFU (a=1)	Jump No Floating Underflow	Jump to U if D21=0; go to NI if D21=1, clear D21	200/300
74	15	JNFO (a=2)	Jump No Floating Overflow	Jump to U if D22=0; go to NI if D22=1; clear D22	200/300
74	15	JNDF (a=3)	Jump No Divide Fault	Jump to U if D23=0; go to NI if D23=1; clear D23	200/300
74	16	JC	Jump Carry	Jump to U if D0=1; go to NI if D0=0	200/300
74	17	JNC	Jump No Carry	Jump to U if D0=0; go to NI if D0=1	200/300
75	00	-	Invalid Code	Causes IOU to return a condition code of 3 to the CPU indicating instruction not available	-
75	01	SIOF	Start I/O Fast Release	(U,U+1) → CAW; initiate operation defined in CCW at address $(CAW)_{59-36}$ on device specified by $(CAW)_{15-00}$	1150

Function Code (Octal)		Mnemonic	Instruction	Description	Instruction Time (in ns)
f	j				
75	02	-	Invalid Code	Causes IOU to return a condition code of 3 to the CPU indicating instruction not available	-
75	03	TSC	Test Subchannel	(U,U+1) → CAW; interrogate subchannel specified by $(CAW)_{15-0}$	1150
75	04	HDV	Halt Device	(U,U+1) → CAW; terminate current operation on subchannel and device specified by $(CAW)_{15-0}$	1150
75	05	HCH	Halt Channel	(U,U+1) → CAW; terminate current operation on channel specified by $(CAW)_{15-0}$	1150
75	06,07	-	Invalid Code	Causes Invalid Instruction interrupt to address 221_8	1150
75	10	LCR	Load Channel Register	(U,U+1) → CAW; if $(CAW)_0=0$, $(CAW)_{50-36}$ → CBR; if $(CAW)_0=1$, $(CAW)_{71-36}$ → interrupt mask register	1150
75	11	LTCW	Load Table Control Words	(U,U+1) → CAW; load HCAW and HSTCW in channel specified by $(CAW)_{15-8}$; $(CAW)_{59-36}$ → HCAW; STCW at address specified by $(CAW)_{59-36}$ → HSTCW	1150
75	12-17	-	Invalid Code	Causes Invalid Instruction interrupt to address 221_8	1150
76	00	FA	Floating Add	$(A_a)+(U)$ → A_a; RESIDUE → A_a+1 if D17=1	800/1050
76	01	FAN	Floating Add Negative	$(A_a)-(U)$ → A_a; RESIDUE→ A_a+1 if D17=1	800/1050
76	02	FM	Floating Multiply	$(A_a)\cdot(U)$ → A_a(and A_a+1 if D17=1)	1350 + 100b
76	03	FD	Floating Divide	$(A_a)\div(U)$ → A_a; REMAINDER → A_a+1 if D17=1	4800/ 4850
76	04	LUF	Load and Unpack Floating	$\|(U)\|_{34-27}$ → A_{a7-0},zero fill; $(U)_{26-0}$ → A_a+1 $_{26-0}$, sign fill	300

b = Number of arithmetic loops

Function Code (Octal)		Mnemonic	Instruction	Description	Instruction Time (in ns)
f	j				
76	05	LCF	Load and Convert to Floating	$(U)_{35} \rightarrow A_a+1_{35}$, [NORMALIZED $(U)]_{26-0} \rightarrow A_a+1_{26-0}$; if $(U)_{35}=0$, $(A_a)_{7-0}$± NORMALIZING COUNT] $\rightarrow$ A_a+1_{34-27}; if $(U)_{35}=1$, ones complement of [$(A_a)_{7-0}$ ± NORMALIZING COUNT] $\rightarrow$ A_a+1_{34-27}	550
76	06	MCDU	Magnitude of Characteristic Difference to Upper	$\lvert (A_a) \rvert_{35-27} - \lvert (U) \rvert_{35-27} \rvert \rightarrow (A_a+1)_{8-0}$ ZEROS $\rightarrow A_a+1_{35-9}$	350
76	07	CDU	Characteristic Difference to Upper	$\lvert (A_a) \rvert_{35-27} - \lvert (U) \rvert_{35-27} \rightarrow A_a+1_{8-0}$; SIGN BITS $\rightarrow A_a+1_{35-9}$	350
76	10	DFA	Double Precision Floating Add	$(A_a, A_a+1) + (U, U+1) \rightarrow A_a, A_a+1$	950
76	11	DFAN	Double Precision Floating Add Negative	$(A_a, A_a+1) - (U, U+1) \rightarrow A_a, A_a+1$	950
76	12	DFM	Double Precision Floating Multiply	$(A_a, A_a+1) \cdot (U, U+1) \rightarrow A_a, A_a+1$	2450
76	13	DFD	Double Precision Floating Divide	$(A_a, A_a+1) \div (U, U+1) \rightarrow A_a, A_a+1$	9850
76	14	DFU	Double Load and Unpack Floating	$\lvert (U,U+1)_{70-50} \rvert \rightarrow A_{a10-0}$, zero fill; $(U,U+1)_{59-38} \rightarrow A_a+1_{23-0}$, sign fill; $(U,U+1)_{35-0} \rightarrow A_a+2$	850
76	15	DLCF, DFP	Double Load and Convert to Floating	$(U)_{35} \rightarrow A_a+1_{35}$; [NORMALIZED $(U, U+1)]_{59-0} \rightarrow A_a+1_{23-0}$ and A_a+2; If $(U)_{35}=0$, $(A_a)_{10-0}$± NORMALIZING COUNT $\rightarrow$ A_a+1_{34-24}; If $(U)_{35}=1$, ones complement of [$(A_a)_{10-0}$ ±NORMALIZING COUNT] $\rightarrow A_a+1_{34-24}$ (see Note 10)	700/1100
76	16	FEL	Floating Expand and Load	$(U)_{26-3} \rightarrow A_{a23-0}$; $(U)_{2-0} \rightarrow A_a+1_{35-33}$;	450

Function Code (Octal)		Mnemonic	Instruction	Description	Instruction Time (in ns)
f	j				
				$(U)_{35} \rightarrow A_a+1_{32-0}$; If $(U)_{35} = 0$, $(U)_{35-27} + 1600_8 \rightarrow A_{a35-24}$; If $(U)_{35}=1$, $(U)_{35-27} - 1600_8 \rightarrow A_{a35-24}$	
76	17	FCL	Floating Compress and Load	$(U)_{23-0} \rightarrow A_{a26-3}$; $(U+1)_{35-33} \rightarrow A_{a2-0}$ if $(U)_{35,0}$ $(U)_{35-24} + 1600_8 \rightarrow A_{a35-27}$ if $(U)_{35}=1$, $(U)_{35-24}+1600_8 \rightarrow A_{a35-27}$	700
77	0–17	-	Invalid Code	Causes Invalid Instruction interrupt to address 221_8	1150

NOTES:

1. The timing estimations use only whole word stores with the exception of the byte instructions which must use partial stores.

2. Repeated instruction times can be estimated by these formulas:

   a. Total time = set up + number of repeats (K or N) x basic time.

   b. Edit in the byte instruction is estimated by N_1 = skip count, N_2 = number of control bytes.

3. Instruction set up times are special sequences to condition the control for repeated sequences and are added to the total time required for the repeated instruction.

4. Test or skip instruction times are stated as xxx/yyy. The first number is the time for skip not taken, the second is for skip taken.

5. Conditional jump times are stated as xxx/yyy. The first number is time for jump not taken, the second for jump taken.

6. Byte instruction times are, in general, data dependent as well as string length dependent; therefore, times given represent a given set of data but could change for different data.

7. For the 37-06, 07 (Byte Add and Add Negative) instructions, the greater time applies if a refetch is required.

8. For the 72-15 (Test Relative Address) instruction, the lesser time applies if the address tested is within the first pair of limits used in the test.

9. For the 73-15 (Load Breakpoint) instruction, the greater time applies when bit 29 of the operand specifies clear jump history stack.

10. For the 76-15 (Double Load and Convert to Floating) instruction, the greater time applies when D20=1 (no instruction overlap).

11. The times given for the I/O instructions are processor times. Add 200 ns if skip, and wait-time for IOU to respond.