multidata
multidata
multidata
multidata
multidata
multidata
multidata
multidata
multidata
multidata
multidata
multidata
multidata
multidata
multidata
multidata
multidata
multidata
multidata
multidata

SYSTEMS
multidata division

Computer Reference Manual
SYSTEMS 72

COMPUTER REFERENCE MANUAL

SYSTEMS 72

October 1970

SYSTEMS ENGINEERING LABORATORIES, INC.
6901 West Sunrise Boulevard
Fort Lauderdale, Florida 33313

SYSTEMS MULTIDATA DIVISION

7300 Bolsa Avenue (213) 598-1377
Westminster, California (714) 892-8347

Publication No. 31101 Price: $10

CONTENTS

SECTION 1

INTRODUCTION

The salient characteristics of the SYSTEMS 72 are:

* 32,768 words of programmable memory, expandable to 65,536 words
* memory map
* automatic program fragmentation
* dynamic program relocation
* memory write protection
* 880-nanosecond core memory cycle time
* 4096-word core memory, expandable to 65,536 words
* 32,768-word memory extension disc, expandable to 131,072 words

* rapid context switching
* eight addressable registers
* double indexing
* displacement indexing
* single-level indirect addressing
* privileged instructions
* five-bit operation field
* only single-word instructions
* user-defined instructions
* relative addressing, forward and backward

* IOP-oriented input/output system
* multilevel interrupt system
* asynchronous, demand-multiplexed input/output
* data chaining
* command chaining
* device independent input/output
* up to 384 individually armed, enabled, triggered, sensed, and set interrupts
* multiple real-time clocks

* remote control panel

SECTION 3

MEMORY

Through memory mapping, the SYSTEMS 72 executes programs that exceed its core memory capacity.

In the basic configuration the Memory Access Controller maps the 32,768-word Memory Extension Disc into the 4096-word core memory in such a way that the effective capacity of core is equal to that of the disc.

Programmable memory increases to 65,536 words -- the range of the 16-bit effective address -- in a system with an expanded disc, even if core memory remains at 4096 words.

3.1 MEMORY ACCESS CONTROLLER

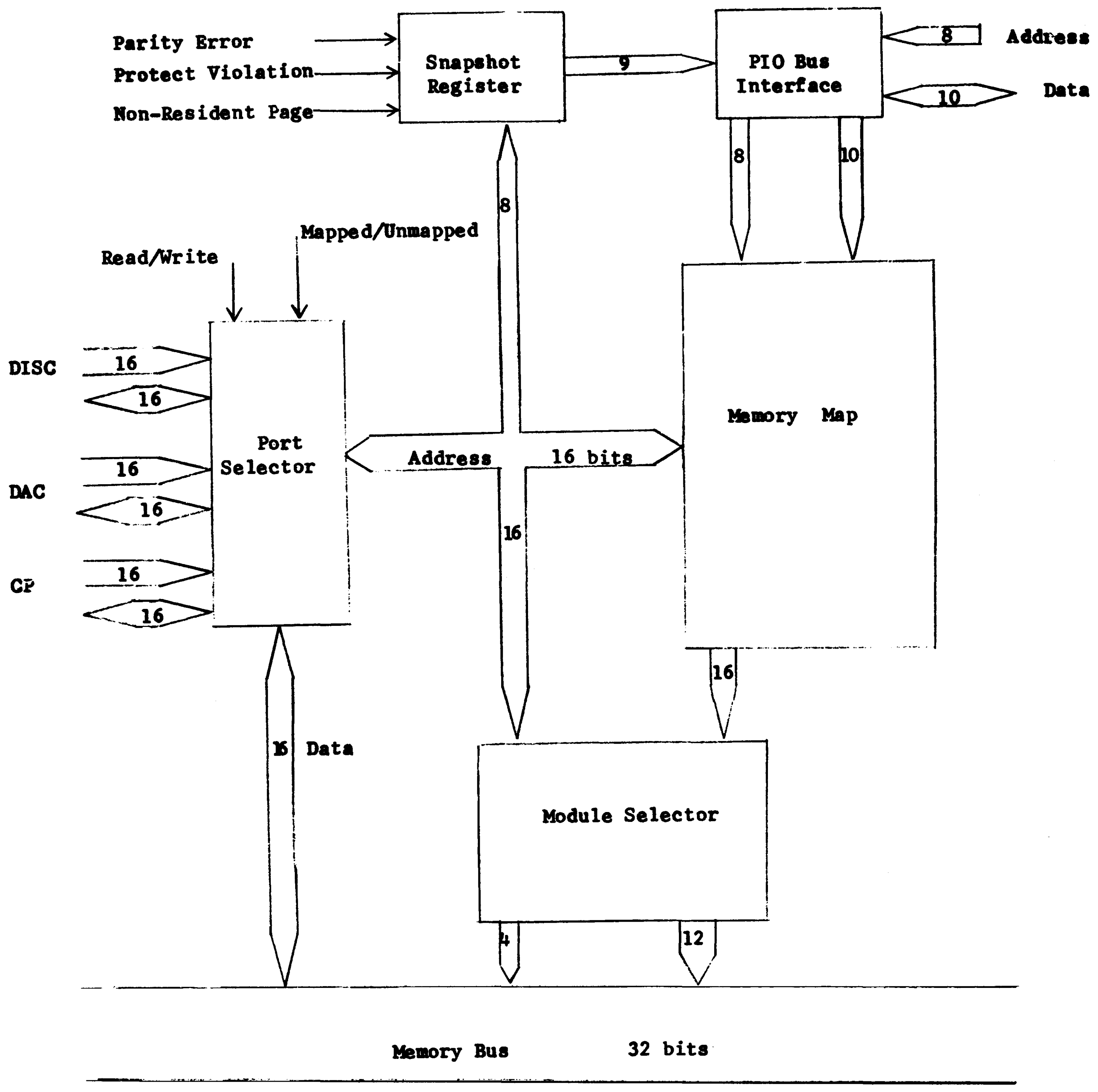
The Memory Access Controller relieves the programmer of all memory management tasks. It employs a Memory Map to dynamically relocate memory references. And when another program segment needs to be transferred into core, the Kernel initiates the transfer and modifies the Memory Map accordingly. (The Kernel is a small, core-resident program that is always transparent to the user's program.)

As shown in Figure 3-1, the controller provides three dedicated ports to core memory. Data transfers between any port and the memory bus are direct. An address reaches the bus through the Module Selector, which uses the four most significant bits to select the memory module and then loads the twelve least significant bits into the address register of that module.

The Memory Map translates virtual addresses into actual addresses.

The Snapshot Register monitors the eight most significant bits of each address from the processor port for use in error detection.

The interface to the Programmed Input/Output Bus enables the processor to modify the Memory Map and to read the Snapshot Register.



Memory Access Controller
Figure 3-1

3.1.1 Port Selection

The Memory Extension Disc, the Direct Access Channel, and the Central Processor each has an independent port to core memory. Each port accommodates 16 address lines and 16 data lines. The Port Selector responds to a core memory access request by switching the lines from the requesting port into the Memory Access Controller. It resolves simultaneous requests, granting highest priority to the disc port, next highest to the channel port, and lowest to the processor port. Each request indicates whether the access is to be a read or a write and whether the address is to be mapped or unmapped. Thus the disc, the channel, and the processor exercise independent control over the Mapped/Unmapped line; PSW2 controls mapping only while the processor port is selected. (Virtual addresses are always mapped; actual addresses are never mapped.)

3.1.2 Mapping

The Memory Access Controller divides core memory into 256-word pages and transfers program segments a page at a time from disc into core. This puts instructions in core, ready for execution. But the program refers to them by their virtual addresses, which do not agree, in general, with their actual addresses. The Memory Map solves this problem by keeping track of the correspondence between virtual and actual pages. (Only the eight most significant bits in the virtual address need be translated; since the pages always contain 256 words, the eight least significant bits remain intact.) As a virtual address enters the controller, the Memory Map substitutes the actual page corresponding to the virtual page and thereby produces the actual address.

The Memory Map is a complement of 128 (optionally 256) 10-bit integrated circuit registers, one for each page of virtual memory. Each register uses the format shown in Figure 3-2 to denote the status of its virtual page.

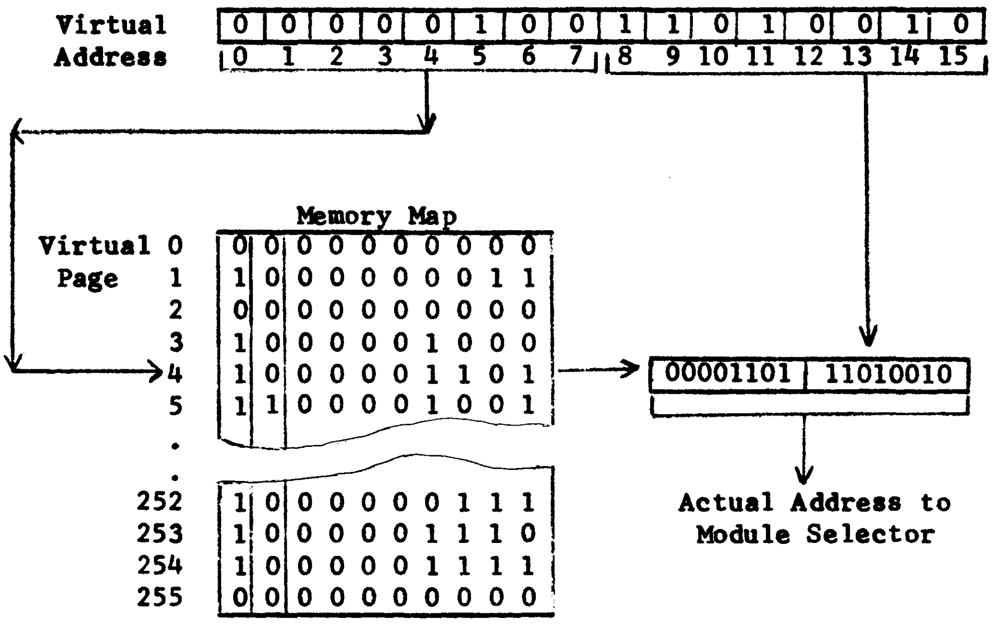
```
Virtual page n   | R | P |       Actual Page        |
                   0   1   2   3   4   5   6   7   8   9
```

Field	Function
R	Indicates, if set, that the virtual page currently occupies the core locations specified by Actual Page.
P	Protects the actual page from being written into, if set.

Field	Function
Actual Page	Replaces the eight most significant bits of any reference to virtual page n.

Memory Map Register
Figure 3-2

Figure 3-3 depicts the operation of the Memory Map. This example shows a virtual address calling for a page that is core-resident (R set); if R were reset, the attempted translation would fail and an error condition would arise, as discussed in 3.1.3.



Memory Map Operation
Figure 3-3

The Memory Map needs updating every time a virtual page changes status. To modify the contents of a map register, the Kernel sets up the desired configuration in the ten least significant bits of the D Register (in the format shown in Figure 3-2). Then it executes a POT instruction (defined in 4.5.9) with an effective address of X'OFRR'. The eight least significant bits of this effective address designate the map register to receive the contents of the D Register.

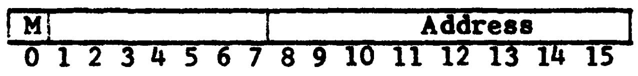
3.1.3 Error Detection

The Memory Access Controller detects an attempt to access a non-resident page, an attempt to write into a protected page, and (if the parity option is installed) a parity error that occurs during a read operation.

If an error is detected while the processor port is selected, the program traps to actual location X'0042', aborting the memory access. (In the master mode the processor may write into a protected page without triggering an error condition.) The trap sets the condition code indicators as follows:

Condition	CC1	CC2
Parity Error	0	0
Protect Violation	0	1
Non-Resident Page	1	0

The program should (and the Kernel does) now execute a PIN instruction (defined in 4.5.9) with an effective address of X'0FXX'. This Input Snapshot Register instruction loads the D Register according to the format in Figure 3-4. (Until unloaded by this PIN instruction, the Snapshot Register will not accept a new configuration.)



Field	Function
M	Indicates, if set, that the address required mapping.
Address	Holds the eight most significant bits of the effective address sent by the processor.

Snapshot Register
Figure 3-4

Disc and channel operations also respond to the detection of memory access errors. Error handling for disc operations is discussed in 3.3.2, for channel operations in 5.3.3.

3.2 CORE MEMORY

As many as 16 core memory modules operate independently on an asynchronous memory bus. The full memory cycle takes 880 nanoseconds. Each module contains 4096 words and has its own address and data registers. These registers allow the bus to release early and thereby overlap successive memory accesses to different modules.

3.2.1 Data Guard

Data Guard (a standard feature) assures that the contents of core memory remain undisturbed during system shutdown (including power failure) and restart. It prevents read or write signals during the power on/off sequence by holding the memory bus at ground. (The power off sequence does not begin until the end of the current instruction.) There is a Power Fail Safe option, described in 5.4.4, that provides additional capabilities.

3.2.2 Memory Parity Checking (Optional)

The Memory Parity option generates odd parity on each write operation and checks it on each read operation. (An attempt to read from a nonexistent core location causes a parity error.) The detection of a parity error while the processor is accessing memory initiates a trap, as discussed in 3.1.3. Trapping has the advantage of aborting the current operation, thereby preventing an error in the memory system from affecting the instruction register or one of the addressable registers.

3.2.3 Dedicated Locations

Actual locations 0-7 serve as the eight addressable registers; crossover circuits nullify attempts to map virtual addresses 0-7. If the system includes the optional set of integrated circuit registers, they intercept memory accesses to locations 0-7 (and render those locations unavailable). A complete list of dedicated locations follows:

Decimal Address	Hexadecimal Address	Reserved For
0-7	0-7	Addressable Registers
8-63	8-3F	Input/Output Service Interrupts
64	40	Power-Off Interrupt

Decimal Address	Hexadecimal Address	Reserved For
65	41	Power-On Interrupt
66	42	Memory Access Controller Trap
67	43	Memory Extension Disc Interrupt
68*	44*	Direct Access Channel
69	45	Call 1
70*	46*	Call 2
71*	47*	Call 3
72*	48*	Real-Time Clock 1
73*	49*	Real-Time Clock 2
74*	4A*	Real-Time Clock 3
75*	4B*	Real-Time Clock 4
76*	4C*	Console Interrupt
128-511*	80-1FF*	System Interrupts**

* May address actual or virtual memory.
** Only as many locations as needed are reserved.

3.3 MEMORY EXTENSION DISC

Programs written for virtual memory are stored on the Memory Extension Disc. These programs may occupy any combination of 256-word pages, but they would normally take contiguous pages in program sequence. Disc capacity may be 32,768 words, 65,536 words, or 131,072 words. A single program may not exceed 65,536 words -- the range of the 16-bit effective address -- but a disc may hold programs to the limit of its capacity.

In its basic configuration the disc has 16 tracks, with 8 sectors/track and 256 words/sector. Expansion simply increases the number of tracks (to either 32 or 64) on the same side of a single disc.

A fixed-head, head/track design holds the average access time to 16.67 milliseconds (one-half of the rotational period of the disc).

The disc controller generates odd parity while writing a sector on disc and checks it while reading.

3.3.1 Page Transfers

Transfers between core and disc always occur a page at a time. (The 256-word page equals the capacity of a sector.) To set up a page transfer, the Kernel:

1) loads bits 7-15 of the D Register with the disc page address. (This 9-bit address covers the full 512-page range.)

2) executes a POT instruction with an effective address of X'0E04'. (If the disc controller is busy, it will set CC1 to indicate that it did not accept the page address.)

3) loads bits 8-15 of the D Register with the core memory page address. (This 8-bit address covers the full 256-word page range; it may specify either a virtual or an actual page.)

The program then initiates the transfer by executing a POT instruction with an effective address from among the following:

Effective Address	Specification
X'0E00'	Core to Disc, Mapped
X'0E01'	Disc to Core, Mapped
X'0E02'	Core to Disc, Unmapped
X'0E03'	Disc to Core, Unmapped

(As before, the disc controller sets CC1 if it is busy and did not accept the page address.)

Once started, the page transfer continues to completion -- unless halted by the disc controller or by the program.

3.3.2 Terminating a Page Transfer

When the disc controller completes a page transfer or halts the transfer because an error has occurred, it causes the program to interrupt to actual location X'0043' and sets the condition codes as follows:

Condition	CC1	CC2
Transfer incomplete, no error	0	0
Transfer incomplete, error	0	1
Transfer complete, no error	1	0
Transfer complete, error	1	1

If an error has occurred, the program can determine its origin by executing a PIN instruction with an effective address of X'0E04'. Bits 12 through 15 of the D Register then identify the error as follows:

Error	Bit Position Set
Non-Resident Page	12
Protect Violation	13
Memory Parity	14
Disc Parity	15

The program can halt a page transfer by executing a POT instruction with an effective address of X'0E05'. (The disc controller sets CC1 to indicate that a page transfer had been in progress.)

3.3.3 Position Sensing

So that it can be used efficiently by programs (such as the Kernel) in swapping, the disc allows its rotational position to be sensed to within a quarter of a sector. The execution of a PIN instruction with an effective address of X'0E05' produces the sector address (which ranges between zero and seven) in bit positions 13 through 15 of the D Register. The condition code settings then indicate the position within that sector:

Condition	CC1	CC2
First Quarter	0	0
Second Quarter	0	1
Third Quarter	1	0
Fourth Quarter	1	1

SECTION 4

CENTRAL PROCESSOR

The Central Processor provides the programmer with the means to make full use of a comprehensive instruction repertoire: addressable registers, condition codes, operating modes, double indexing, displacement indexing, indirect addressing, and relative addressing forward or backward.

Rapid context switching assures efficient handling of interrupts and smooth changes in operating mode.

4.1 ADDRESSABLE REGISTERS

The memory reference instructions operate directly on eight full-word registers. These addressable registers occupy either the first eight core memory locations or (optionally) 128 integrated circuit flip flops. Either way, they are always addressable as absolute locations 0-7. (Effective addresses in this range are not mapped.)

All eight registers have general utility and several have special capabilities. A brief summary of the registers follows:

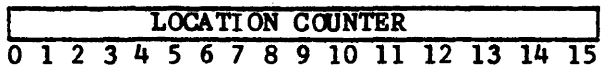
Register	Address	Name	Special Function
A	01	Accumulator	Holds the results of arithmetic, logical, compare, and shift operations
B	04	Base Register	Pre-indexing
D	00	Data Register	Holds 16-bit data words going to and coming from the Programmed Input/Output Bus.
E	02	Extended Accumulator	Becomes the low-order extension of the Accumulator on double register shift operations.
X	03	Index Register	Post-Indexing; looping
R1	05	Utility Register	None

Register	Address	Name	Special Function
R2	06	Utility Register	None
R3	07	Utility Register	None

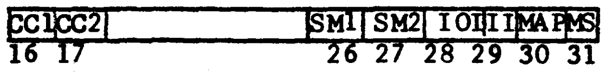
4.2 PROGRAM STATUS DOUBLEWORD

By keeping a summary of the program environment in two 16-bit integrated circuit registers, the Program Status Doubleword enables the SYSTEMS 72 to transfer control quickly from one program to another. This ability to turn suddenly to a new program and provide it with its correct operating environment, called rapid context switching, makes the Model A especially efficient in handling interrupts and in transferring control back and forth between user programs and the operating system.

Rapid context switching is implicit in traps, interrupts, and the execution of a Call 1 or a Call 2 instruction. The occurrence of any of these events will automatically cause the current Program Status Doubleword to be swapped for the one pertaining to the subroutine being called. Later, in exiting from this subroutine, the Branch Return and Clear instruction will put the previous Program Status Doubleword back in control.



PSW1



PSW2

Program Status Doubleword
Figure 4-1

As shown in Figure 4-1, the first program status word (PSW1) points to the location of the next instruction to be executed. It steps sequentially with every execution unless redirected by a branch or a call instruction or by a trap or an interrupt.

The second program status word (PSW2) sets the program operating environment through a series of status indicators. The next six paragraphs discuss the use of these indicators.

4.2.1 Condition Code Indicators (CC1 and CC2)

Most instructions (all except the branch, store, and call instructions) configure the condition code indicators with each execution. The interrupts and the trap also affect condition codes. The conditional branch instructions use these settings to discover, for example, that overflow has occurred, or that a device is ready, or that one register holds a higher number than another.

4.2.2 Compare Sequence Mode (SM1 and SM2)

In the compare sequence mode (SM1 set) the SYSTEMS 72 uses the compare instructions to detect whether all of the numbers in a group bear the same relationship to a given value. For example, input data entering the D Register could be compared with the contents of the A Register. At the end of the block transfer the condition code settings would reveal whether the data consistently exceeded the specified value. The setting of SM2 (defined in 4.5.6) helps determine the kind of comparison to be made.

4.2.3 Input/Output Inhibit (IOI)

Although the SYSTEMS 72 does not have an external input/output processor, its system design is IOP-oriented. Accordingly, input/output service requests from peripheral equipment take precedence over all interrupt service requests. Input/Output Inhibit (IOI set) blocks the input/output service requests but does not affect any other part of the system. (Section 5 describes the input/output system in detail.)

4.2.4 Interrupt Inhibit (II)

Interrupt Inhibit (II set) blocks all interrupt service requests but not the input/output service requests. This blocking prevents an interrupt from becoming active, but does not prevent an armed interrupt from entering the waiting state. (Section 5 describes the interrupt system in detail.)

4.2.5 Mapped Mode (MAP)

In the mapped mode (MAP set) the SYSTEMS 72 uses the Memory Map to translate program addresses into core memory addresses. (Section 3 describes the operation of the Memory Map.) The Kernel, a core-resident program that manages memory allocation, is never mapped; all other programs are usually mapped.

4.2.6 Master/Slave Mode (MS)

Privileged instructions will execute only while the SYSTEMS 72 is in the master mode (MS reset). The Kernel always has MS reset; user programs normally do not.

4.3 INSTRUCTION FORMAT

All SYSTEMS 72 instructions have this single-word format:

R	I	X	Operation					S		Displacement					
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Field	Function
R	Specifies addressing relative to PSW1, if set.
I	Specifies single-level indirect addressing, if set.
X	Specifies post-indexing, if set.
Operation	Specifies the operation to be executed.
S	Specifies the sign of the displacement, if set while R is set; specifies pre-indexing if set while R is reset.
Displacement	Specifies the displacement, an integer used in the effective address calculation.

Although conforming to the format, several of the branch instructions put certain fields to special use. The conditional branch instructions are incapable of indirect addressing and post-indexing; for them the I and X fields define a mask. Similarly, the Branch Return and Clear instruction does not include post-indexing; here the X field denotes whether the highest active interrupt is to be cleared.

4.4 ADDRESSING

The effective address calculation proceeds in exactly the same manner for each instruction. This is true regardless of whether the effective address will be used to specify a memory location, as in a load instruction, or to specify further details of the operation, as in a shift instruction.

A three-step sequence describes the process:

1. If relative addressing is specified (R set), the sign and the displacement form a 16-bit two's complement number (with the sign in bits 0-8), which adds to PSW1 to form the partial address.

   If pre-indexing if specified instead of relative addressing (R reset, S set), the displacement (with bits 0-8 reset) adds to the contents of the B Register to form the partial address.

   If neither relative addressing nor pre-indexing is specified (R and S both reset), the displacement (with bits 0-8 reset) becomes the partial address.

2. If indirect addressing is specified (I set), the partial address points to a memory location, the contents of which now become the partial address.

   If indirect addressing is not specified (I reset), the partial address remains unchanged.

3. If post-indexing is specified (X set), the contents of the X Register add to the partial address to form the effective address. (On byte instructions the X Register holds a byte count, which adds to the partial address to form a 17-bit effective byte address.)

   If post-indexing is not specified (X reset), the partial address becomes the effective address.

This process does not vary. For those branch instructions that are incapable of post-indexing or of both indirect addressing and post-indexing, the effective address calculation simply treats the pertinent fields as being reset. For call instructions the effective address calculation treats R, I, X, and S as being reset.

4.5 INSTRUCTION REPERTOIRE

The five-bit operation field yields a comprehensive instruction repertoire, ranging from basic arithmetic instructions to sophisticated context-switching call instructions. Further enhancing this power is the ability of the memory reference instructions to operate on all eight addressable registers.

By recognizing derivative instructions such as the register-to-register instructions, the conditional branch instructions, and the shift instructions, the assembler makes the programming task easier.

Not all of the available operation codes have been implemented; several have been set aside for (optional) user-specified instructions. An undefined operation code will execute as a "No Operation".

Because the SYSTEMS 72 operates asynchronously, the duration of an instruction depends on several parameters. The method of calculating this duration is too lengthy to include after each instruction in this section, Accordingly, timing information may be found in Appendix C.

The number in the operation field of each instruction is in the hexadecimal notation. (Elsewhere in this manual hexadecimal numbers are placed in quotation marks preceded by the letter X, as in X'0045'.)

4.5.1 Branch Instructions

The branch instructions exercise program control by forcing PSW1 out of its usual sequence, by setting up linkages to subroutines, by counting the iterations through a loop, by testing the condition codes, and by restoring the Program Status Doubleword.

Branch instructions do not affect the condition code indicators.

B Branch

R	I	X	12	S	Displacement
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The effective address replaces PSW1.

Affected: PSW1

BAL Branch and Link

R	I	X	13	S	Displacement
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The address immediately following the current address replaces the contents of the B Register, and the effective address replaces PSW1. (Pre-indexing is possible because the effective address calculation occurs before the contents of the B Register are altered.)

Affected: PSW1
B Register

BAL is the normal choice for branching to a subroutine. The subroutine can then use pre-indexing both to pass arguments and to set up the return address. By putting the linking address in the B Register instead of in the effective location, BAL is especially useful in branching to reentrant subroutines and subroutines with multiple entry points.

BIX Branch and Increment Index

```
|R|I|X|    14     |S|   Displacement      |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The contents of the X Register increment by one. If the contents of the X Register do not equal zero, the effective address replaces PSW1; if they do equal zero, program control passes to the location immediately following the current location. (The X field should normally be left reset; if set, post-indexing will occur before the X Register is incremented.)

Affected: PSW1
X Register

BCS Branch on Conditions Set

```
|R|I|X|    16     |S|   Displacement      |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The effective address replaces PSW1 if a condition code indicator and its corresponding mask bit are both set. (CC1 is masked by the I field, CC2 by the X field; the effective address calculation does not include indirect addressing or post-indexing.) Table 4-1 arrays all configurations of the condition codes and the mask, with a "B" marking every combination that will cause branching.

		IX	IX	IX	IX
CC1	CC2	00	01	10	11
0	0	-	-	-	-
0	1	-	B	-	B
1	0	-	-	B	B
1	1	-	B	B	B

Table 4-1

If the requirements for branching are not met, PSW1 advances normally.

Affected: PSW1

The assembler recognizes the following derivatives of BCS:

BNEZ Branch if Not Equal to Zero

```
|R|1|0|    16     |S|   Displacement      |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The effective address replaces PSW1 if the contents of the pertinent register do not equal zero. BNEZ is appropriate immediately after a load instruction (the receiving register is pertinent) or a logical instruction (the A Register is pertinent).

BLZ Branch if Less than Zero

R	0	1	16					S	Displacement						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The effective address replaces PSW1 if the contents of the pertinent register are less than zero. BLZ is appropriate immediately after a load instruction (the receiving register is pertinent) or a logical instruction (the A Register is pertinent).

BNE Branch if Not Equal

R	1	0	16					S	Displacement						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The effective address replaces PSW1 if the contents of the A Register and the other operand are not equal. BNE is appropriate immediately after a compare instruction.

BL Branch if Less Than

R	0	1	16					S	Displacement						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The effective address replaces PSW1 if the contents of the A Register are less than the other operand. BL is appropriate immediately after a compare instruction.

BC Branch on Carry

R	1	0	16					S	Displacement						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The effective address replaces PSW1 if the last arithmetic operation caused a carry. BC is appropriate immediately after an arithmetic instruction.

BO Branch on Overflow

R	0	1	16	S	Displacement
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The effective address replaces PSW1 if the last arithmetic operation caused an overflow. BO is appropriate immediately after an arithmetic instruction.

BCR Branch on Conditions Reset

R	I	X	15	S	Displacement
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The effective address replaces PSW1 unless a condition code indicator and its corresponding mask bit are both set. (CC1 is masked by the I field, CC2 by the X field; the effective address calculation does not include indirect addressing or post-indexing.) Table 4-2 arrays all configurations of the condition codes and the mask, with a "B" marking every combination that will cause branching.

CC1	CC2	IX 00	IX 01	IX 10	IX 11
0	0	B	B	B	B
0	1	B	-	B	-
1	0	B	B	-	-
1	1	B	-	-	-

Table 4-2

If the requirements for branching are not met, PSW1 advances normally.

Affected: PSW1

The assembler recognizes the following derivatives of BCR:

BEZ Branch if Equal to Zero

R	1	1	15	S	Displacement
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The effective address replaces PSW1 if the contents of the pertinent register are equal to zero. BEZ is appropriate immediately after a load instruction (the receiving register is pertinent) or a logical instruction (the A Register is pertinent).

BGEZ   Branch if Greater than or Equal to Zero

```
|R|0|1|     15      |S|     Displacement      |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The effective address replaces PSW1 if the contents of the pertinent register are greater than or equal to zero. BGEZ is appropriate immediately after a load instruction (the receiving register is pertinent) or a logical instruction (the A Register is pertinent).

BE   Branch if Equal

```
|R|1|1|     15      |S|     Displacement      |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The effective address replaces PSW1 if the contents of the A Register and the other operand are equal. BE is appropriate immediately after a compare instruction.

BGE   Branch if Greater than or Equal

```
|R|0|1|     15      |S|     Displacement      |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The effective address replaces PSW1 if the contents of the A Register are greater than or equal to the other operand. BGE is appropriate immediately after a compare instruction.

BNC   Branch on No Carry

```
|R|1|0|     15      |S|     Displacement      |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The effective address replaces PSW1 if the last arithmetic operation did not cause a carry. BNC is appropriate immediately after an arithmetic instruction.

BNO   Branch on No Overflow

```
|R|0|1|     15      |S|     Displacement      |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The effective address replaces PSW1 if the last arithmetic operation did not cause an overflow. BNO is appropriate immediately after an arithmetic instruction.

BRC Branch Return and Clear

R	I	X	17	S	Displacement
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The contents of the effective address and that of its succeeding location replace the Program Status Doubleword. The effective address calculation does not include post-indexing. Instead, a one in the X field indicates that the highest active interrupt is to be cleared. A privileged instruction, BRC has no effect when the processor is in the slave mode.

Affected:	PSW1
	PSW2
	Highest Active Interrupt

Call 1 and Call 2 instructions, as well as traps and interrupts, store the Program Status Doubleword before branching to a service routine. The new PSW2 normally puts the SYSTEMS 72 in the master mode, which enables these routines to use BRC upon returning to the main program. (Returns from the call instructions would normally be with the X field reset.)

4.5.2 Load Instructions

Registers A, B, D, and X are each capable of receiving the contents of any register or any memory location. In addition, the A Register may receive either the high- or low-order byte from any memory location or any other register.

Load instructions set the condition code indicators as follows:

Condition	CC1	CC2
Contents of receiving register less than zero	1	1
Contents of receiving register equal to zero	0	0
Contents of receiving register greater than zero	1	0

(For the byte instructions the receiving register can never be less than zero because bits 0-7 are always cleared.)

The conditional branch instructions listed below are appropriate immediately after a load instruction.

Mnemonic	Meaning
BEZ	Branch if contents of receiving register are equal to zero
BNEZ	Branch if contents of receiving register are not equal to zero
BGEZ	Branch if contents of receiving register are greater than or equal to zero
BLZ	Branch if contents of receiving register are less than zero

LDA Load A Register

```
|R|I|X|    09     |S|    Displacement     |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The effective word replaces the contents of the A Register.

Affected: A Register
CC1
CC2

The assembler recognizes the following derivatives of LDA:

LDAB Load A from B

```
|0|0|0|    09     |0|          04          |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The contents of the B Register replace the contents of the A Register.

LDAD Load A from D

```
|0|0|0|    09     |0|          00          |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The contents of the D Register replace the contents of the A Register.

LDAE Load A from E

```
|0|0|0|    09     |0|          02          |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The contents of the E Register replace the contents of the A Register.

LDAX   Load A from X

```
|0|0|0|      09       |0|         03           |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The contents of the X Register replace the contents of the A Register.

LDA1   Load A from R1

```
|0|0|0|      09       |0|         05           |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The contents of the R1 Register replace the contents of the A Register.

LDA2   Load A from R2

```
|0|0|0|      09       |0|         06           |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The contents of the R2 Register replace the contents of the A Register.

LDA3   Load A from R3

```
|0|0|0|      09       |0|         07           |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The contents of the R3 Register replace the contents of the A Register.

LDB   Load B Register

```
|R|I|X|      OC       |S|    Displacement      |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The effective word replaces the contents of the B Register.

Affected: B Register
          CC1
          CC2

The assembler recognizes the following derivatives of LDB:

LDBA Load B from A

0	0	0	0C					0	01						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The contents of the A Register replace the contents of the B Register.

LDBD Load B from D

0	0	0	0C					0	00						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The contents of the D Register replace the contents of the B Register.

LDBE Load B from E

0	0	0	0C					0	02						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The contents of the E Register replace the contents of the B Register.

LDBX Load B from X

0	0	0	0C					0	03						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The contents of the X Register replace the contents of the B Register.

LDB1 Load B from R1

0	0	0	0C					0	05						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The contents of the R1 Register replace the contents of the B Register.

LDB2 Load B from R2

0	0	0	0C					0	06						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The contents of the R2 Register replace the contents of the B Register.

LDB3 Load B from R3

```
|0|0|0|     0C     |0|        07           |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The contents of the R3 Register replace the contents of the B Register.

LDD Load D Register

```
|R|I|X|     08     |S|   Displacement      |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The effective word replaces the contents of the D Register.

Affected: D Register
CC1
CC2

The assembler recognizes the following derivatives of LDD:

LDDA Load D from A

```
|0|0|0|     08     |0|        01           |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The contents of the A Register replace the contents of the D Register.

LDDB Load D from B

```
|0|0|0|     08     |0|        04           |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The contents of the B Register replace the contents of the D Register.

LDDE Load D from E

```
|0|0|0|     08     |0|        02           |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The contents of the E Register replace the contents of the D Register.

LDDX Load D from X

```
|0|0|0|     08     |0|        03           |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The contents of the X Register replace the contents of the D Register.

LDD1 Load D from R1

0	0	0	08	0	05
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The contents of the R1 Register replace the contents of the D Register.

LDD2 Load D from R2

0	0	0	08	0	06
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The contents of the R2 Register replace the contents of the D Register.

LDD3 Load D from R3

0	0	0	08	0	07
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The contents of the R3 Register replace the contents of the D Register.

LDX Load X Register

R	I	X	0B	S	Displacement
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The effective word replaces the contents of the X Register.

Affected: X Register
CC1
CC2

The assembler recognizes the following derivatives of LDX:

LDXA Load X from A

0	0	0	0B	0	01
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The contents of the A Register replace the contents of the X Register.

LDXB Load X from B

0	0	0	0B	0	04
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The contents of the B Register replace the contents of the X Register.

LDXD Load X from D

0	0	0	0B	0	00
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The contents of the D Register replace the contents of the X Register.

LDXE Load X from E

0	0	0	0B	0	02
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The contents of the E Register replace the contents of the X Register.

LDX1 Load X from R1

0	0	0	0B	0	05
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The contents of the R1 Register replace the contents of the X Register.

LDX2 Load X from R2

0	0	0	0B	0	06
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The contents of the R2 Register replace the contents of the X Register.

LDX3 Load X from R3

0	0	0	0B	0	07
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The contents of the R3 Register replace the contents of the X Register.

LBY Load Byte into A Register

R	I	X	0A					S	Displacement						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The effective byte replaces bits 8-15 of the A Register (clearing bits 0-7).

If post-indexing is not specified, the effective byte is bits 0-7 of the effective word.

If post-indexing is specified, the X Register holds a byte count, and the effective address calculation produces a 17-bit effective byte address. The sixteen most significant bits point to the effective word, which contains an even-numbered byte in bits 0-7, an odd-numbered byte in bits 8-15.

Affected: A Register
CC1
CC2

The assembler recognizes the following derivatives of LBY:

LBYB Load Byte into A from B

0	0	X	0A					0	04						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The effective byte replaces bits 8-15 of the A Register (clearing bits 0-7). If post-indexing is specified, and the contents of the X Register equal one (a higher count will change the instruction), the effective byte is bits 8-15 of the B Register. Otherwise, the effective byte is bits 0-7 of the B Register.

LBYD Load Byte into A from D

0	0	X	0A					0	00						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The effective byte replaces bits 8-15 of the A Register (clearing bits 0-7). If post-indexing is specified, and the contents of the X Register equal one (a higher count will change the instruction), the effective byte is bits 8-15 of the D Register. Otherwise, the effective byte is bits 0-7 of the D Register.

LBYE Load Byte into A from E

0	0	0	0A	0	02
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The effective byte replaces bits 8-15 of the A Register (clearing bits 0-7). If post-indexing is specified, and the contents of the X Register equal one (a higher count will change the instruction), the effective byte is bits 8-15 of the E Register. Otherwise, the effective byte is bits 0-7 of the E Register.

LBYX Load Byte into A from X

0	0	0	0A	0	03
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The effective byte replaces bits 8-15 of the A Register (clearing bits 0-7). If post-indexing is specified, and the contents of the X Register equal one (a higher count will change the instruction), the effective byte is bits 8-15 of the X Register. Otherwise, the effective byte is bits 0-7 of the X Register.

LBY1 Load Byte into A from R1

0	0	0	0A	0	05
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The effective byte replaces bits 8-15 of the A Register (clearing bits 0-7). If post-indexing is specified, and the contents of the X Register equal one (a higher count will change the instruction), the effective byte is bits 8-15 of the R1 Register. Otherwise, the effective byte is bits 0-7 of the R1 Register.

LBY2 Load Byte into A from R2

0	0	0	0A	0	06
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The effective byte replaces bits 8-15 of the A Register (clearing bits 0-7). If post-indexing is specified, and the contents of the X Register equal one (a higher count will change the instruction), the effective byte is bits 8-15 of the R2 Register. Otherwise, the effective byte is bits 0-7 of the R2 Register.

LBY3 Load Byte into A from R3

```
|0|0|0|    0A     |0|          07          |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The effective byte replaces bits 8-15 of the A Register (clearing bits 0-7). If post-indexing is specified, and the contents of the X Register equal one (a higher count will change the instruction), the effective byte is bits 8-15 of the R3 Register. Otherwise, the effective byte is bits 0-7 of the R3 Register.

4.5.3 Store Instructions

The A Register and the D Register are both capable of storing their contents into any register or memory location. In addition, the A Register can also store bits 8-15 into either the high- or low-order byte of any register or memory location.

Store instructions do not affect the condition code indicators.

STA Store A Register

```
|R|I|X|    06     |S|  Displacement        |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The contents of the A Register replace the effective word.

Affected: Effective word

The assembler recognizes the following derivatives of STA:

STAB Store A into B

```
|0|0|0|    06     |0|          04          |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The contents of the A Register replace the contents of the B Register.

Affected: B Register

STAD Store A into D

```
|0|0|0|    06     |0|          00          |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The contents of the A Register replace the contents of the D Register.

Affected: D Register.

STAE Store A into E

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	06					0	02						

The contents of the A Register replace the contents of the E Register.

Affected: E Register

STAX Store A into X

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	06					0	03						

The contents of the A Register replace the contents of the X Register.

Affected: X Register

STA1 Store A into R1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	06					0	05						

The contents of the A Register replace the contents of the R1 Register.

Affected: R1 Register

STA2 Store A into R2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	06					0	06						

The contents of the A Register replace the contents of the R2 Register.

Affected: R2 Register

STA3 Store A into R3

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	06					0	07						

The contents of the A Register replace the contents of the R3 Register.

Affected: R3 Register

STD Store D Register

R	I	X	05	S	Displacement
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The contents of the D Register replace the effective word.

Affected: Effective word

The assembler recognizes the following derivatives of STD:

STDB Store D into B

0	0	0	05	0	04
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The contents of the D Register replace the contents of the B Register.

Affected: B Register

STDE Store D into E

0	0	0	05	0	02
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The contents of the D Register replace the contents of the E Register.

Affected: E Register

STDX Store D into X

0	0	0	05	0	03
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The contents of the D Register replace the contents of the X Register.

Affected: X Register

STD1 Store D into R1

0	0	0	05	0	05
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The contents of the D Register replace the contents of the R1 Register.

Affected: R1 Register

STD2 Store D into R2

0	0	0	05	0	06
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The contents of the D Register replace the contents of the R2 Register.

Affected: R2 Register

STD3 Store D into R3

0	0	0	05	0	07
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The contents of the D Register replace the contents of the R3 Register.

Affected: R3 Register

SBY Store Byte from A Register

R	I	X	07	S	Displacement
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

Bits 8-15 of the A Register replace the effective byte (leaving the other byte unaffected).

If Post-indexing is not specified, the effective byte is bits 0-7 of the effective word.

If post indexing is specified, the X Register holds a byte count, and the effective address calculation produces a 17-bit effective byte address. The 16 most significant bits point to the effective word, which contains an even-numbered byte in bits 0-7, an odd-numbered byte in bits 8-15.

Affected: Effective byte

The assembler recognizes the following derivatives of SBY:

SBYB Store Byte from A into B

```
|0|0|0|    07     |0|        04          |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

Bits 8-15 of the A Register replace the effective byte (leaving the other byte unaffected). If post-indexing is specified, and the contents of the X Register equal one (a higher count will change the instruction), the effective byte is bits 8-15 of the B Register. Otherwise, the effective byte is bits 0-7 of the B Register.

Affected: Effective byte of B Register

SBYD Store Byte from A into D

```
|0|0|0|    07     |0|        04          |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

Bits 8-15 of the A Register replace the effective byte (leaving the other byte unaffected). If post-indexing is specified, and the contents of the X Register equal one (a higher count will change the instruction), the effective byte is bits 8-15 of the D Register. Otherwise, the effective byte is bits 0-7 of the D Register.

Affected: Effective byte of D Register

SBYE Store Byte from A into E

```
|0|0|0|    07     |0|        02          |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

Bits 8-15 of the A Register replace the effective byte (leaving the other byte unaffected). If post-indexing is specified, and the contents of the X

Register equal one (a higher count will change the instruction), the effective byte is bits 8-15 of the X Register. Otherwise, the effective byte is bits 0-7 of the E Register.

Affected: Effective byte of the E Register

SBYX Store Byte from A into X

0	0	0			07			0				03			
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Bits 8-15 of the A Register replace the effective byte (leaving the other byte unaffected). If post-indexing is specified, and the contents of the X Register equal one (a higher count will change the instruction), the effective byte is bits 8-15 of the X Register. Otherwise, the effective byte is bits 0-7 of the X Register.

Affected: Effective Byte of the X Register

SBY1 Store Byte from A into R1

0	0	0			07			0				05			
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Bits 8-15 of the A Register replace the effective byte (leaving the other byte unaffected). If post-indexing is specified, and the contents of the X Register equal one (a higher count will change the instruction), the effective byte is bits 8-15 of the X Register. Otherwise, the effective byte is bits 0-7 of the R1 Register.

Affected: Effective Byte of the R1 Register

SBY2 Store Byte from A into R2

0	0	0	07	0	06
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

Bits 8-15 of the A Register replace the effective byte (leaving the other byte unaffected). If post-indexing is specified, and the contents of the X Register equal one (a higher count will change the instruction), the effective byte is bits 8-15 of the R2 Register. Otherwise, the effective byte is bits 0-7 of the R2 Register.

Affected: Effective byte of the R2 Register

SBY3 Store Byte from A into R3

0	0	0	07	0	07
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

Bits 8-15 of the A Register replace the effective byte (leaving the other byte unaffected). If post-indexing is specified, and the contents of the X Register equal one (a higher count will change the instruction), the effective byte is bits 8-15 of the R3 Register. Otherwise, the effective byte is bits 0-7 of the R3 Register.

Affected: Effective byte of the R3 Register

4.5.4 Arithmetic Instructions

The arithmetic instructions are capable of operating on the contents of any register or memory location. Should a carry or an overflow occur, the condition code indicators are set as follows:

Condition	CC1	CC2
No Overflow, No Carry	0	0
Overflow , No Carry	0	1
Carry, No Overflow	1	0
Carry and Overflow	1	1

The conditional branch instructions listed below are appropriate immediately after an arithmetic instruction.

Mnemonic	Meaning
BO	Branch on overflow
BNO	Branch on no overflow
BC	Branch on carry
BNC	Branch on no carry

ADD Add Memory to A Register

R	I	X	OD	S	Displacement
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The effective word plus the contents of the A Register replace the contents of the A Register.

Affected: A Register
CC1
CC2

The assembler recognizes the following derivatives of ADD:

ADDB Add B to A

0	0	0	OD	0	04
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The contents of the B Register plus the contents of the A Register replace the contents of the A Register.

ADDD Add D to A

0	0	0	OD	0	00
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The contents of the D Register plus the contents of the A Register replace the contents of the A Register.

ADDE Add E to A

0	0	0	OD	0	02
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The contents of the E Register plus the contents of the A Register replace the contents of the A Register.

ADDX Add X to A

0	0	0	OD	0	03
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The contents of the X Register plus the contents of the A Register replace the contents of the A Register.

ADD1 Add R1 to A

```
|0|0|0|      0D     |0|           05           |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The contents of the R1 Register plus the contents of the A Register replace the contents of the A Register.

ADD2 Add R2 to A

```
|0|0|0|      0D     |0|           06           |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The contents of the R2 Register plus the contents of the A Register replace the contents of the A Register.

ADD3 Add R3 to A

```
|0|0|0|      0D     |0|           07           |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The contents of the R3 Register plus the contents of the A Register replace the contents of the A Register.

SUB Subtract Memory from A Register

```
|R|I|X|      0E     |S|      Displacement      |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The contents of the A Register minus the effective word replace the contents of the A Register.

Affected: A Register
CC1
CC2

The assembler recognizes the following derivatives of SUB:

SUBB Subtract B from A

```
|0|0|0|      0E     |0|           04           |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The contents of the A Register minus the contents of the B Register replace the contents of the A Register.

SUBD Subtract D from A

0	0	0	0E	0	00
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The contents of the A Register minus the contents of the D Register replace the contents of the A Register.

SUBE Subtract E from A

0	0	0	0E	0	02
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The contents of the A Register minus the contents of the E Register replace the contents of the A Register.

SUBX Subtract X from A

0	0	0	0E	0	03
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The contents of the A Register minus the contents of the X Register replace the contents of the A Register.

SUB1 Subtract R1 from A

0	0	0	0E	0	05
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The contents of the A Register minus the contents of the R1 Register replace the contents of the A Register.

SUB2 Subtract R2 from A

0	0	0	0E	0	06
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The contents of the A Register minus the contents of the R2 Register replace the contents of the A Register.

SUB3 Subtract R3 from A

0	0	0	0E	0	07
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The contents of the A Register minus the contents of the R3 Register replace the contents of the A Register.

INC Increment Memory

R	I	X	0F					S	Displacement						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The effective word increases by one.

Affected:	Effective word
	CC1
	CC2

The assembler recognizes the following derivatives of INC:

INCA Increment A

0	0	0	0F					0	01						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The contents of the A Register increase by one.

Affected:	A Register
	CC1
	CC2

INCB Increment B

0	0	0	0F					0	04						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The contents of the B Register increase by one.

Affected:	B Register
	CC1
	CC2

INCD Increment D

0	0	0	0F					0	00						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The contents of the D Register increase by one.

Affected:	D Register
	CC1
	CC2

INCE Increment E

0	0	0	0F					0	02						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The contents of the E Register increase by one.

Affected:	E Register CC1 CC2

INCX Increment X

0	0	0	0F					0	03						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The contents of the X Register increase by one.

Affected:	X Register CC1 CC2

INC1 Increment R1

0	0	0	0F					0	05						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The contents of the R1 Register increase by one.

Affected:	R1 Register CC1 CC2

INC2 Increment R2

0	0	0	0F					0	06						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The contents of the R2 Register increase by one.

Affected:	R2 Register CC1 CC2

INC3 Increment R3

0	0	0	0F					0	07						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The contents of the R3 Register increase by one.

Affected:	R3 Register
	CC1
	CC2

4.5.5 Logical Instructions

The contents of the A Register are capable of forming the logical product, the logical sum, or the logical difference with the contents of any register or memory location.

The logical product contains ones only in those bit positions in which both operands contain ones.

The logical sum contains ones only in those bit positions in which either or both operands contain ones.

The logical difference contains ones only in those bit positions in which either but not both operands contain ones.

Logical instructions set the condition code indicators as follows:

Condition	CC1	CC2
Contents of A Register less than zero	1	1
Contents of A Register equal to zero	0	0
Contents of A Register greater than zero	1	0

The conditional branch instructions listed below are appropriate immediately after a logical instruction.

Mnemonic	Meaning
BEZ	Branch if contents of A Register are equal to zero
BNEZ	Branch if contents of A Register are not equal to zero
BGEZ	Branch if contents of A Register are greater than or equal to zero
BLZ	Branch if contents of A Register are less than zero

AND AND Memory into A Register

```
|R|I|X|      02       |S|        Displacement         |
 0 1 2 3  4  5  6  7  8  9  10  11  12  13  14  15
```

The logical product of the effective word and the contents of the A Register replaces the contents of the A Register.

Affected: A Register
CC1
CC2

The assembler recognizes the following derivatives of AND:

ANDB AND B into A

```
|0|0|0|      02       |0|             04              |
 0 1 2 3  4  5  6  7  8  9  10  11  12  13  14  15
```

The logical product of the contents of the A Register and the contents of the B Register replaces the contents of the A Register.

ANDD AND D into A

```
|0|0|0|      02       |0|             00              |
 0 1 2 3  4  5  6  7  8  9  10  11  12  13  14  15
```

The logical product of the contents of the A Register and the contents of the D Register replaces the contents of the A Register.

ANDE AND E into A

```
|0|0|0|      02       |0|             02              |
 0 1 2 3  4  5  6  7  8  9  10  11  12  13  14  15
```

The logical product of the contents of the A Register and the contents of the E Register replaces the contents of the A Register.

ANDX AND X into A

```
|0|0|0|      02       |0|             03              |
 0 1 2 3  4  5  6  7  8  9  10  11  12  13  14  15
```

The logical product of the contents of the A Register and the contents of the X Register replaces the contents of the A Register.

AND1 AND R1 into A

0	0	0	02					0	05						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The logical product of the contents of the A Register and the contents of the R1 Register replaces the contents of the A Register.

AND2 AND R2 into A

0	0	0	02					0	06						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The logical product of the contents of the A Register and the contents of the R2 Register replaces the contents of the A Register.

AND3 AND R3 into A

0	0	0	02					0	07						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The logical product of the contents of the A Register and the contents of the R3 Register replaces the contents of the A Register.

LOR OR Memory into A Register

R	I	X	03					S	Displacement						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The logical sum of the effective word and the contents of the A Register replaces the contents of the A Register.

Affected: A Register
CC1
CC2

The assembler recognizes the following derivatives of LOR:

LORB OR B into A

0	0	0	03					0	04						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The logical sum of the contents of the B Register and the contents of the A Register replaces the contents of the A Register.

LORD OR D into A

0	0	0	03	0	00
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The logical sum of the contents of the D Register and the contents of the A Register replaces the contents of the A Register.

LORE OR E into A

0	0	0	03	0	02
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The logical sum of the contents of the E Register and the contents of the A Register replaces the contents of the A Register.

LORX OR X into A

0	0	0	03	0	03
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The logical sum of the contents of the X Register and the contents of the A Register replaces the contents of the A Register.

LOR1 OR R1 into A

0	0	0	03	0	05
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The logical sum of the contents of the R1 Register and the contents of the A Register replaces the contents of the A Register.

LOR2 OR R2 into A

0	0	0	03	0	06
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The logical sum of the contents of the R2 Register and the contents of the A Register replaces the contents of the A Register.

LOR3 OR R3 Register into A

```
|0|0|0|    03     |0|          07           |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The logical sum of the contents of the R3 Register and the contents of the A Register replaces the contents of the A Register.

EOR Exclusive OR Memory into A Register

```
|R|I|X|    04     |S|     Displacement      |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The logical difference of the effective word and the contents of the A Register replaces the contents of the A Register.

Affected: A Register
CC1
CC2

The assembler recognizes the following derivatives of EOR:

EORB Exclusive OR B into A

```
|0|0|0|    04     |0|          04           |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The logical difference of the contents of the B Register and the contents of the A Register replaces the contents of the A Register.

EORD Exclusive OR D into A

```
|0|0|0|    04     |0|          00           |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The logical difference of the contents of the D Register and the contents of the A Register replaces the contents of the A Register.

EORE Exclusive OR E into A

```
|0|0|0|    04     |0|          02           |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The logical difference of the contents of the E Register and the contents of the A Register replaces the contents of the A Register.

EORX Exclusive OR X into A

0	0	0	04					0	03						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The logical difference of the contents of the X Register and the contents of the A Register replaces the contents of the A Register.

EOR1 Exclusive OR R1 into A

0	0	0	04					0	05						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The logical difference of the contents of the R1 Register and the contents of the A Register replaces the contents of the A Register.

EOR2 Exclusive OR R2 into A

0	0	0	04					0	06						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The logical difference of the contents of the R2 Register and the contents of the A Register replaces the contents of the A Register.

EOR3 Exclusive OR R3 into A

0	0	0	04					0	07						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The logical difference of the contents of the R3 Register and the contents of the A Register replaces the contents of the A Register.

4.5.6 Compare Instructions

The compare instructions determine whether the contents of the A Register are less than, equal to, or greater than that of a particular memory location or register, and set the condition code indicators accordingly. These instructions treat both operands as signed integers in the two's complement format and leave the operands unchanged.

The condition code settings are:

Condition	CC1	CC2
Contents of A Register less than other operand	1	1
Contents of A Register equal to other operand	0	0
Contents of A Register greater than other operand	1	0

The conditional branch instructions listed below are appropriate immediately after a compare instruction.

Mnemonic	Meaning
BE	Branch if contents of A Register are equal to other operand
BNE	Branch if contents of A Register are not equal to other operand
BGE	Branch if contents of A Register are greater than or equal to other operand
BL	Branch if contents of A Register are less than other operand

If the SYSTEMS 72 is in the compare sequence mode (SM1 set) the result of each comparison combines logically with the current condition code settings instead of replacing them. The bit destined for CC1 forms the logical sum (SM2 reset) or the logical product (SM2 set) with the current setting of CC1. CC2 is set similarly. (The compare sequence mode inhibits condition code settings from all but the compare instructions.)

CMP   Compare A Register with Memory

```
|R|I|X|    10     |S|     Displacement      |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The condition code indicators depict the comparison between the contents of the A Register and the effective word.

Affected: CC1
          CC2

The assembler recognizes the following derivatives of CMP:

CMPB   Compare A with B

```
|0|0|0|    10     |0|        04          |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The condition code indicators depict the comparison between the contents of the A Register and the contents of the B Register.

CMPD   Compare A with D

```
|0|0|0|    10     |0|        00          |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The condition code indicators depict the comparison between the contents of the A Register and the contents of the D Register.

CMPE   Compare A with E

```
|0|0|0|    10     |0|        02          |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The condition code indicators depict the comparison between the contents of the A Register and the contents of the E Register.

CMPX   Compare A with X

```
|0|0|0|    10     |0|        03          |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The condition code indicators depict the comparison between the contents of the A Register and the contents of the X Register.

CMP1   Compare A with R1

```
|0|0|0|    10     |0|        05          |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The condition code indicators depict the comparison between the contents of the A Register and the contents of the R1 Register.

CMP2 Compare A with R2

0	0	0	10					0	06						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The condition code indicators depict the comparison between the contents of the A Register and the contents of the R2 Register.

CMP3 Compare A with R3

0	0	0	10					0	07						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The condition code indicators depict the comparison between the contents of the A Register and the contents of the R3 Register.

4.5.7 Shift Instructions

A full complement of shift operations derive from a single instruction.

Single register shifts take place in the A Register. Double Register shifts treat the A and E Registers as a single 32-bit register with the E Register holding the sixteen least significant bits.

Logical left shifts introduce zeros into the least significant bit position and lose bits leaving the most significant bit position. Logical right shifts introduce zeros into the most significant bit position and lose bits out of the least significant bit position. (In double register logical shifts the least significant position is bit position 15 of the E Register, the most significant position is bit position 0 of the A Register.)

Arithmetic left shifts introduce zeros into the least significant bit position and lose bits leaving bit position 1 of the A Register. Arithmetic right shifts propagate the contents of bit position 0 of the A Register and lose bits leaving the least significant bit position. (In double register arithmetic shifts the least significant bit position is bit position 15 of the E Register.)

Circular shifts take the bits leaving one end of the register and introduce them into the other end. (In double register circular shifts the two ends are bit position 0 of the A Register and bit position 15 of the E Register.)

After a left shift the condition code settings are:

Condition	CC1	CC2
Bit position 0 of the A Register contained a zero initially and has received only zeros	0	0
Bit position 0 of the A Register contained a one initially and has received only zeros	0	1
Bit position 0 of the A Register contained a one initially and has received only ones	1	0
Bit position 0 of the A Register has received a one at least once and has changed polarity at least once	1	1

After a right shift the condition code settings are:

Condition	CC1	CC2
Bit position 15 of the A Register has received only zeros	0	0
Bit position 15 of the A Register has received a one at least once	0	1

S Shift

R	I	X	11					S	Displacement						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Bits 8-15 of the effective address specify the shift completely:

S/D	TYPE		R/L	COUNT			
8	9	10	11	12	13	14	15

Bit Positions	Setting	Specification
8	0	Single register shift
	1	Double register shift
9, 10	00	Not allowed
	01	Circular Shift
	10	Logical shift
	11	Arithmetic Shift
11	0	Right shift
	1	Left shift
12-15		Number of places to shift from 1-16 (zero means 16 places)

Affected: A Register
E Register
CC1
CC2

A single register shift would ordinarily keep the R, I, X, and S fields set to zero and specify the entire operation in the Displacement field. A double register shift must develop its effective address through indirect addressing or indexing.

The assembler recognizes the following derivatives of S:

Mnemonic	Meaning	Bits 8-15 of Effective Address				
		8	9	10	11	12 13 14 15
SLL	Shift Logical Left	0	1	0	1	COUNT
SLR	Shift Logical Right	0	1	0	0	COUNT
SAL	Shift Arithmetic Left	0	0	1	1	COUNT
SAR	Shift Arithmetic Right	0	0	1	0	COUNT
SCL	Shift Circular Left	0	1	1	1	COUNT
SCR	Shift Circular Right	0	1	1	0	COUNT
SLLD	Shift Logical Left Double	1	1	0	1	COUNT
SLRD	Shift Logical Right Double	1	1	0	0	COUNT
SALD	Shift Arithmetic Left Double	1	0	1	1	COUNT
SARD	Shift Arithmetic Right Double	1	0	1	0	COUNT
SCLD	Shift Circular Left Double	1	1	1	1	COUNT
SCRD	Shift Circular Right Double	1	1	1	0	COUNT

4.5.8 Call Instructions

Call 1 operates only while the SYSTEMS 72 is in the master mode and always switches program context, making it ideal for operating system calls to the Kernel. Call 2 also switches program context, but is not privileged, making it ideal for user program calls to the operating system. Call 3 does not switch program context and is not privileged, making it a convenient link to subroutines within the user program. Call instructions ignore the R, I, X, and S fields in calculating their effective addresses.

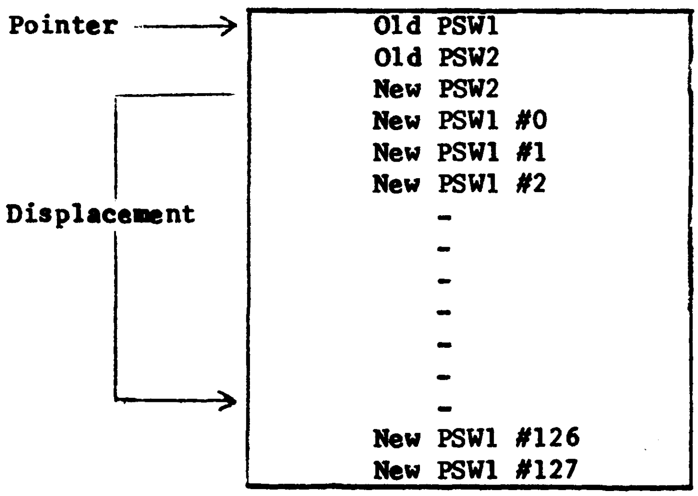
Call instructions do not affect the condition code indicators.

CAL1 Call 1

0	0	0	18	0	Displacement
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The contents of X'0045' point to the first location of a 131-word table. As shown in Figure 4-2, PSW1 and PSW2 go into the first two locations. The contents of the third location replace PSW2. The effective address becomes a displacement into the remaining 128 locations. The contents of the designated location replace PSW1.

Affected: PSW1
PSW2



Call 1/Call 2 Table
Figure 4-2

To return to the calling routine, the called routine executes a BRC instruction with an effective address equal to the contents of the pointer in X'0045'.

CAL1 is a privileged instruction. (It functions as a "No Operation" when the SYSTEMS 72 is in the slave mode.) It also presumes the mapped mode indicator to be reset and, therefore, treats X'0045' and the contents of X'0045' as actual addresses.

CAL2 Call 2

```
|0|0|0|     19     |0|     Displacement     |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The contents of X'0045' point to the first location of a 131-word table. As shown in Figure 4-2, PSW1 and PSW2 go into the first two locations. The contents of the third location replace PSW2. The effective address becomes a displacement into the remaining 128 locations. The contents of the designated location replace PSW1.

Affected:	PSW1
	PSW2

To return to the calling routine, the called routine executes a BRC instruction with an effective address equal to the contents of the pointer in X'0046'.

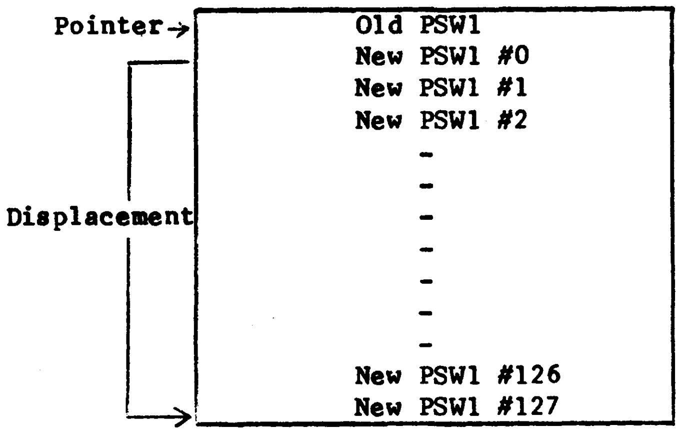
CAL2 is not a privileged instruction. Furthermore, it treats X'0046' and the contents of X'0046' as virtual addresses if the mapped mode indicator is set.

CAL3 Call 3

```
|0|0|0|     1A     |0|     Displacement     |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The contents of X'0047' point to the first location of a 129-word table. As shown in Figure 4-3, PSW1 goes into that first location. The effective address becomes a displacement into the remaining 128 locations. The contents of the designated location replace PSW1.

Affected: PSW1



Call 3 Table
Figure 4-3

To return to the calling routine, the called routine branches to the address contained in the first location of the table.

Unlike CAL1 and CAL2, CAL3 does not switch program context. Like CAL2, CAL3 is not a privileged instruction. It treats X'0047' and the contents of X'0047' as virtual addresses if MAP is set.

4.5.9 Input/Output Instructions

The input/output instructions operate on all devices connected to the Programmed Input/Output Bus. This bus runs 16 address lines, 16 bidirectional data lines, and six control lines to the Direct Access Channel, the Memory Map, the disc controller, PSW2, and the control panel, as well as to the standard peripheral equipment.

Programmed Input and Programmed Output are generic instructions. The effective address specifies both the device and the operation to be performed. (Those instructions that are defined and listed in Section 5 pertain to standard peripheral equipment and have dedicated effective addresses; all addresses with X'F' in the most significant hexadecimal digit are reserved for instructions pertaining to special user-oriented equipment.)

Condition code settings are defined for each instruction instead of for the input/output class of instructions.

All input/output instructions are privileged; while the SYSTEMS 72 is in the slave mode, they function as "No Operations".

PIN Programmed Input

R	I	X	01	S	Displacement
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The effective address drives the 16 address lines. Then the addressed device loads the D Register via the 16 data lines.

Affected:	D Register
	CC1
	CC2

The assembler recognizes the following derivatives of PIN:

RDS Read Data Switches

1	1	0	01	S	Displacement
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The contents of the D Register represent the Data Switch settings. (The assembler configures the displacement and the indirectly addressed location such that X'0401' becomes the effective address.)

Affected: D Register

RPS2 Read PSW2

1	1	0	01	S	Displacement
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The contents of PSW2 replace the contents of the D Register. (The assembler configures the displacement and the indirectly addressed location such that X'0402' becomes the effective address.)

Affected: D Register

POT Programmed Output

R	I	X	00	S	Displacement
0	1	2	3 4 5 6 7	8	9 10 11 12 13 14 15

The effective address drives the 16 address lines. Then the addressed device accepts the contents of the D Register via the 16 data lines.

Affected:	CC1
	CC2

The assembler recognizes the following derivatives of POT:

LDR Load Display Register

```
|1|1|0|    0    |S|    Displacement    |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The contents of the D Register replace the contents of the Display Register. (The assembler configures the displacement and the indirectly addressed location such that X'0401' becomes the effective address.)

Affected: None

LPS2 Load PSW2

```
|1|1|0|    0    |S|    Displacement    |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The contents of the D Register replace the contents of PSW2. (The assembler configures the displacement and the indirectly addressed location such that X'0402' becomes the effective address.)

Affected: None

4.5.10 Other Instructions

The assembler recognizes several derivative instructions that have general utility.

NOP No Operation

```
|1|0|0|   12    |0|         01         |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

PSW1 advances normally, but no addressable register, memory location, or condition code indicator is affected.

Affected: None

CLA Clear A Register

```
|0|0|0|   04    |0|         01         |
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

CC1, CC2, and all 16 bit positions of the A Register are reset.

Affected:	A Register
	CC1
	CC2

HLT Halt

1	1	0			0			S			Displacement				
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Program execution stops. If the operator places the COMPUTE switch in IDLE and then back to RUN, execution will begin with the next instruction in sequence. An interrupt becoming active will restart the program at the address contained in the new PSW1. The return from the interrupt subroutine will be to the location immediately following the Halt instruction. Both CC1 and CC2 are reset. A privileged instruction, HLT functions as a "No Operation" while the processor is in the slave mode.

Affected:	CC1
	CC2

The assembler configures the displacement and the indirectly addressed location such that X'0401' becomes the effective address.

SECTION 5

INPUT/OUTPUT SYSTEM

In keeping with its large-scale architecture, the SYSTEMS 72 provides IOP capability in a demand-multiplexed input/output system. It also provides a convenient interface to special devices, either through the Programmed Input/Output Bus or through the (optional) Direct Access Channel. A multilevel interrupt system responds to service requests on a priority basis.

5.1 INPUT/OUTPUT PROCESSOR

The (programmed) Input/Output Processor frees the SYSTEMS 72 from the usual small computer limitations that restrict software growth. In handling all peripheral device operations as a separate activity, the IOP at once provides flexible, comprehensive control without imposing a requirement for intricate, painstaking programming. (Although an integral part of the Central Processor, the IOP differs functionally from an external IOP only in that it requires more frequent servicing by the operating system.)

The IOP drives the Programmed Input/Output Bus, consisting of sixteen address lines, sixteen bidirectional data lines, and six control lines. Because service requests from peripheral devices are demand multiplexed, instead of being continuously scanned in a fixed priority pattern, the devices can use essentially all of the 200 KB bandwidth of the bus.

Peripheral devices operate independently and asynchronously because their controllers require minimal direction from the IOP and because each controller has its own data buffer. (These buffers vary in size with the data rates of the devices, ranging from one byte for the teletypewriter to sixteen bytes for the nine-track magnetic tape transport.) Thus a device takes only as much bandwidth as needed for data to move between its buffer and the IOP; it will not seize the bus on block transfers (except in burst mode). Controllers keep device addresses in six toggle switch settings instead of in fixed circuitry. This means that device addressing is flexible; arbitrary limits have not been imposed on the number of devices of a given type that may be in a system.

The operating system, conceived concurrently with SYSTEMS 72 architecture and designed to make full use of the IOP, relieves the programmer of all tasks associated with using the IOP. The result is an input/output system that is both powerful and easy to program.

5.1.1 IOP Operation

Although the IOP is software-implemented and not physically distinct from the Central Processor, it appears to be a separate entity -- both to the programmer and to the operating system. Accordingly, the convention listed below will be observed throughout the IOP discussion:

The Central Processor executes instructions.
The Input/Output Processor executes commands.
The Peripheral Device executes orders.

Figure 5-1 lists the instructions that the processor executes to initiate and monitor input/output operations.

Instruction	Effective Address	Use
Start I/O (SIO)	0000 0101 00DD DDDD	Starts an I/O operation
Halt I/O (HIO)	0000 0101 01DD DDDD	Halts an I/O operation
Test Device (TDV)	0000 0101 00DD DDDD	Inputs device status
Test I/O (TIO)	0000 0101 10DD DDDD	Sets CC1 if device is busy
Reset I/O (RIO)	0000 0100 0000 0011	Halts all devices

Notes: 1) Bits marked "D" in the effective address specify the device address.

2) All of these instructions set CC1 if the addressed device is busy (or, for RIO, if any device is busy).

3) SIO, HIO, and RIO are POT instructions. TDV and TIO are PIN instructions.

Input/Output Instructions
Figure 5-1

The IOP uses an Input/Output Command List (IOCL) to control peripheral devices. Figure 5-2 shows the IOCL format.

Word 1	Order	Flag
Word 2	Interrupt Address	
Word 3	Data Address	
Word 4	Byte Count	
	0-7	8-15

Order Field:

Bit Configuration 0 1 2 3 4 5 6 7	Order
M M M M M M 0 1	Write
M M M M M M 1 0	Read
M M M M M M 1 1	Control

0 1 2 3 4 5 6 7	Order
M M M M M 1 0 0	Sense
M M M M 1 1 0 0	Read Backward
M M M M 1 0 0 0	Transfer in Channel
0 0 0 0 0 0 0 0	Stop

(The letter "M" designates bits to be used to further specify the operation and adapt it to the needs of a particular device.)

Flag Field:	Bit	Meaning (If Bit is Set)
	8	Map the data addresses and terminal interrupt address
	9	Interrupt on Zero Byte Count
	10	Interrupt on Transmission Error
	11	Suppress Incorrect Length
	12	Interrupt on Unusual End
	13	Command Chain
	14	Data Chain
	15	Interrupt on Channel End

Interrupt Address:	In terminating an operation, the device interrupts to this address.
Data Address:	The address of the first word of the data block.
Byte Count:	The number of bytes to be transferred.

Input/Output Command List
Figure 5-2

A program initiates an input/output operation by making available to the operating system the starting address of the data, the byte count, and certain details about the direction and manner of the data transfer. The operating system responds by setting up an IOCL and then activating an appropriate device. (Since input/output is device independent, it is only at this point -- run time -- that the device is specified.

The device interrupts to the IOP, which sends the first word of the IOCL to the device and disconnects. This word specifies the operation, how it is to be performed, and how it is to be terminated. When the device is ready to begin the data transfer, it interrupts to the IOP again. The IOP then executes a data transfer command and disconnects, incrementing the data address (third word of the IOCL) and decrementing the byte count (fourth word of the IOCL). The device executes a data transfer order and interrupts to the IOP when ready for another. (For high-speed data transfers, the device may keep the IOP connected while it empties its data buffer.) This cycle continues until the byte count goes to zero; the IOP then sends the device its terminal interrupt address (which will cause the device to interrupt back to the operating system instead of to the IOP) and disconnects. The device then carries out the termination sequence specified to it earlier by the first word of the IOCL.

5.1.2 Chaining

The device termination sequence may call for another IOCL and not for an interrupt back to the operating system. In this event the IOP receives the interrupt; it responds by sending the first word of the next IOCL -- a process called chaining.

In data chaining the new IOCL specifies a new termination sequence (at least the final IOCL must be different, or the operation would not have a programmed conclusion), a new data address, and a new byte count; it does not change the operation itself. Data chaining thus provides scatter-read, gather-write capability; for example, a single instruction, with appropriate IOCL's can initiate a series of operations that gather information from several areas of core memory and print it on a single line of the console teletypewriter.

Command chaining brings in a totally new IOCL, one that specifies a different operation, as well as a new data address, a new byte count, and a new termination sequence. This enables a single instruction to initiate a series of different operations. For example, command chaining could have the IOP read 16 records from magnetic tape, rewind the tape, and reposition the tape 12 records from the load point.

5.1.3 Special Devices

The Interface Manual explains how special devices may be operated under IOP control.

5.2 PROGRAMMED INPUT/OUTPUT BUS

Special devices that do not need the block transfer capability of the IOP may be interfaced to the Programmed Input/Output Bus and operate under direct program control.

The bus consists of 16 address lines, 16 bidirectional data lines, and 6 control lines. The Programmed Input (PIN) and Programmed Output (POT) instructions (defined in 4.5.9) execute single-word transfers between the D Register and devices on the bus. The effective address of the PIN or the POT specifies both the device and the operation to be performed.

The operating system reserves all effective addresses with X'F' in the most significant hexadecimal digit for special user-oriented equipment.

5.3 DIRECT ACCESS CHANNEL

The (optional) Direct Access Channel interfaces special systems-oriented devices directly to core memory. Each device controls its own data transfer rate, which may approach the 1,000,000 words/second limit of the Memory Access Controller.

5.3.1 Preparing the Channel

The typical channel operation will be a high-speed block transfer. To set up the starting address, the program loads this address into the D Register and then executes a POT instruction with an effective address of X'0405'. If both condition code indicators are reset, the POT was accepted and the address register of the channel now holds the starting address; if CC1 is set, the channel was busy and the POT had no effect on it.

A similar sequence (with an effective address of X'0406' for the POT) puts the word count in the word count register. Again, the condition codes indicate whether the POT was effective.

To set the status of the channel, initially and also when recovering from a power loss (described in 5.4.4), the program executes a POT instruction with an effective address of X'0409'. At this time the contents of the D Register specify the following:

Bit	Meaning (If Bit is Set)
0	Makes bits 2, 5, 6, 7, 8, 9, 10, and 11 effective.
1	Makes bits 2 and 3 effective even if bit 0 is reset.
2	Arms the channel interrupt.
3	Enables the channel interrupt.
5	Indicates interrupt address is to be mapped.
6	Sets interrupt to waiting state.
7	Sets interrupt to active state.
8	Indicates the contents of the address register are to be mapped.
9	Sets channel to run mode.

All other bits are unused, except 10 and 11, which specify the operating mode:

Bit 10	Bit 11	Specification
0	0	Data transfer shall be an output
0	1	Direction of transfer specified by device
1	0	Data transfer shall be an input
1	1	Direction of transfer specified by device

5.3.2 Channel Operation

When the channel has received its starting address, word count, and status, it is ready to transfer data. The transfer begins when the program executes a POT instruction with an effective address of X'0407'. (The contents of the D Register are irrelevant.) If the channel was busy, it will reject the POT and set CC1; either way, the device controls CC2. If the POT was accepted, the channel waits for the device to request a data transfer. (The external system controls the data transfer, even to the point of selecting the device; the program addresses only the channel, not the device.)

The channel automatically increments the address register and decrements the word count with each data transfer. When the word count reaches zero, or if an error is detected, the channel interrupts to location X'0044' (actual or virtual, as specified in the channel status) if the channel interrupt is armed and enabled.

The condition code indicators report the following:

CC1	CC2	Specification
0	0	Transfer complete, no error
0	1	Transfer incomplete, no error
1	0	Transfer complete, error
1	1	Transfer incomplete, error

The program can stop a data transfer by executing a POT instruction with an effective address of X'0408'. If the channel had been busy, it will set CC1; the device controls CC2. Stopping does not affect the channel registers or cause an interrupt; the same POT used for start is suitable for restart.

5.3.3 Sensing

The program can read the address register by executing a PIN instruction with an effective address of X'0405'. This loads the D Register with the contents of the address register, without affecting channel operation. CC1, if set, indicates that the channel was busy; CC2 is always reset.

Similarly, a PIN with an effective address of X'0406' reads the word count. Again, CC1 is set if the channel was busy; CC2 is always reset.

To sense the current status of the channel, the program executes a PIN instruction with an effective address of X'0409'. (The condition codes defined for the interrupt in 5.3.2 apply to this PIN.) The contents of the D Register then describe status, as follows:

Bit	Meaning (If Bit is Set)
0	Unused, always set
1	Unused, always reset
2	Interrupt Armed
3	Interrupt Enabled
4	Unused, always reset
5	Interrupt Address Mapped

Bit	Meaning (If Bit is Set)
6	Interrupt in Waiting State
7	Interrupt in Active State
8	Contents of Address Register to be Mapped
9	Channel is Active (no stopped, word count not zero)
10, 11	Operating Mode (defined below)
12	Unused, always Reset
13	Memory Parity Error
14	Memory Protect Violation
15	Non-Resident Page

The operating mode (bits 10 and 11) specifies whether the direction of the data transfers is under the control of the channel or the device:

Bit 10	Bit 11	Specification
0	0	Output
0	1	As defined by device
1	0	Input
1	1	As defined by device

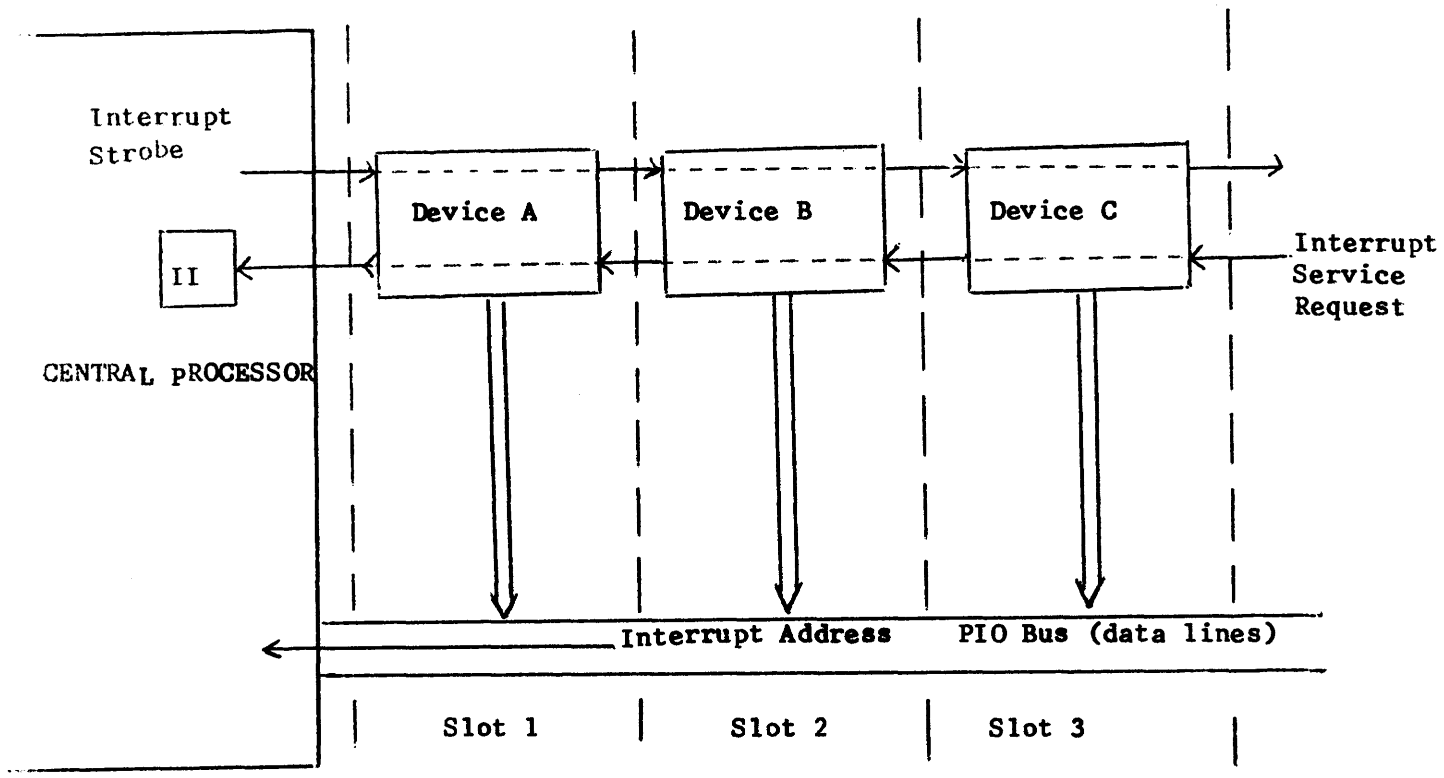
The PIN resets the error indicators (bits 13, 14, and 15) in the channel.

The Power Fail Safe option (described in 5.4.4) senses channel status during its power-off sequence.

5.4 INTERRUPT SYSTEM

The Interrupt System allows devices on the Programmed Input/Output Bus to force the Central Processor to transfer program control to interrupt service routines.

The cycle begins when an external stimulus causes an interrupt to enter the waiting state. If none of the interrupts in I/O slots nearer to the processor is in the waiting state of the active state, the waiting interrupt emits a service request. This request takes precedence over interrupts farther from the processor, interrupting those in the active state and blocking service requests from those in the waiting state. (Thus I/O slot position determines priority in the interrupt queue. But it does not determine interrupt address; each interrupt bears its own address.)



Interrupt System
Figure 5-3

As shown in Figure 5-3, the request proceeds to the Interrupt Inhibit (the II field of PSW2). If II is reset, the request causes the processor to drive an interrupt strobe from one I/O slot to the next until the device making the request receives the strobe. The device then places its 16-bit interrupt address on the PIO Bus data lines, drives the Mapped line if the address is to be mapped, and configures the condition code indicators. The interrupt is now in the active state.

When the processor completes its current instruction, it accesses the location specified by the interrupt address. The contents of this location point to a four-word table. (If the interrupt address was mapped, the pointer is also mapped.) The processor then stores PSW1 in the first word of the table and PSW2 in the second. After loading PSW1 from the third and PSW2 from the fourth, it turns program control over to the new Program Status Doubleword. (The procedure just described is rapid context switching -- also mentioned in 4.3.2.)

At the conclusion of the interrupt subroutine, the execution of a Branch Return and Clear instruction (BRC) with the X field set will restore the previous Program Status Doubleword and clear the highest active interrupt. (Clearing takes an interrupt out of the active state but does not affect the waiting state.)

System reset (described in 6.1.12) disarms and disables all interrupts except the Power Fail Safe option interrupts. It also takes them out of the waiting or active states.

5.4.1 The Trap

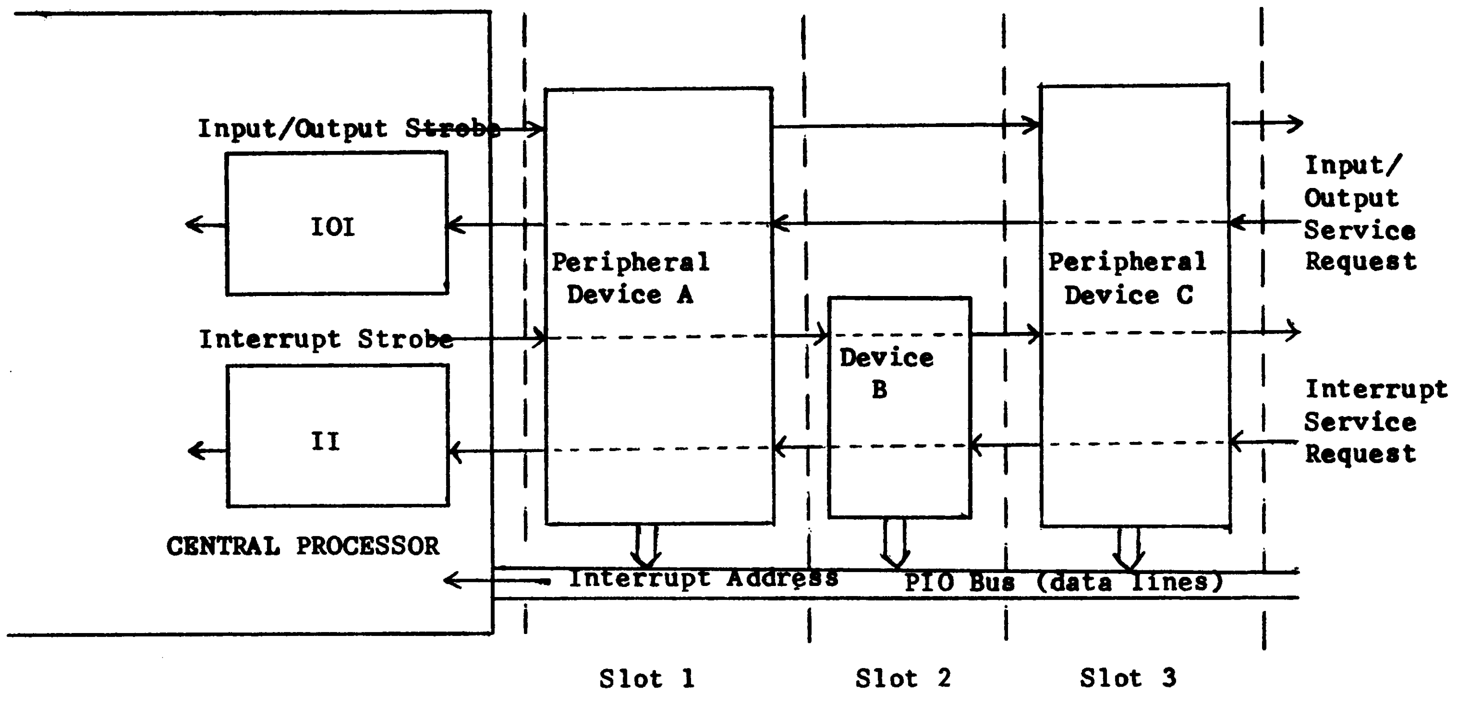
Although not classified as an interrupt, the Memory Access Controller Trap resembles an interrupt of the highest priority.

Like an interrupt, it has a pointer location (always actual location X'0042') that points to a four-word table where rapid context switching takes place. It also configures the condition code indicators. The return from its service routine does not have the X field of BRC set because the Trap does not require clearing.

The Trap differs from an interrupt in that it cannot be inhibited and also in that it takes effect immediately, aborting the current instruction.

5.4.2 Input/Output Interrupts

Peripheral devices require two interrupt levels each, a higher one for handling (rate-sensitive) data transfers and a lower one for signalling the end of a transmission. Accordingly, the interrupt system provides input/output service requests and an input/output strobe. It also includes an Input/Output Inhibit (the IOI field of PSW2), which determines whether an input/output service request can trigger an input/output strobe. The result is the two-tiered system depicted in Figure 5-4.



Dual-Level Interrupts
Figure 5-4

An impending data transfer causes a device to put its higher-level interrupt into the waiting state. This interrupt then follows the procedure described earlier for interrupts generally. Here, however, its input/output service request takes predence over all activity on the lower tier, even that emanating from I/O slots nearer to the processor. Another difference is that the interrupt address is restricted to actual locations 0-63 (the eight addressable registers are not excluded) and is, therefore, not mapped. (The setting of six toggle switches in the device controller specifies both the device address and the interrupt address.) In exiting the input/output service routine, BRC clears the input/output interrupt (which was the highest active interrupt, regardless of its I/O slot).

As a device reaches its end of transmission, it puts its lower-level interrupt into the waiting state. The interrupt cycle proceeds in the usual manner, the IOCL (defined in 5.1.1) having previously supplied a 16-bit interrupt address and a mapped/unmapped indicator. (This is the sole instance in which the device address is not also the interrupt address.)

5.4.3 System Interrupts

For use in special real-time systems, the System Interrupts provide extreme versatility. They can be individually armed/disarmed, enabled/disabled, triggered, sensed, or set.

> Disarming turns an interrupt off. A program uses this capability to reassign a stimulus to a different priority level or to remove it altogether.
>
> Disabling an armed interrupt prevents that interrupt from requesting service, but not from acknowledging a stimulus. Thus, a program can defer response to a stimulus without losing track of it.
>
> Triggering is the means by which a program can initiate an interrupt stimulus of its own. These program-generated interrupts are useful in simulating external system elements during program checkout. They are also useful in putting portions of a program into the external stimuli queue.
>
> Sensing finds out whether an interrupt is armed or unarmed, enabled or disabled, whether it is waiting or not, whether it is active or not, and whether its interrupt address is to be mapped or unmapped. This capability is essential to the power-off routine, which must record interrupt status during a power fail shutdown (described in 5.4.4).
>
> Setting is the ability to configure interrupt status under program control. An interrupt needs to be set initially and during power restart; it may also require a change in status during the course of program execution.

System Interrupts follow the procedure described earlier for interrupts generally. Their service requests take the lower tier shown in Figure 5-4, which makes them subject to Interrupt Inhibit. Since the Interrupt Pair option provides two System Interrupts on a printed circuit card, each card carries an even- and an odd-numbered interrupt. System Interrupt addresses begin with X'0080' and go to X'01FF' (a range of 384). Eight toggle switch settings determine the eight most significant bits in the address; circuitry on the card distinguishes the even- and odd-numbered addresses (the least significant bit in the nine-bit interrupt address).

A program senses interrupt status by executing a PIN instruction with an effective address of X'01TT', where "TT" represents the toggle switch address of the interrupt pair. The contents of the D Register then report status in the format shown in Figure 5-5.

Bit	Meaning (If Bit is Set)
0	Not used, always set
1	Not used, always reset
2	Even Interrupt Armed
3	Even Interrupt Enabled
4	Not used, always reset
5	Even Interrupt Address Mapped
6	Even Interrupt in Waiting State
7	Even Interrupt in Active State
8	Not used, always set
9	Not used, always reset
10	Odd Interrupt Armed
11	Odd Interrupt Enabled
12	Not used, always reset
13	Odd Interrupt Address Mapped
14	Odd Interrupt in Waiting State
15	Odd Interrupt in Active State

D Register Format (Sensing Interrupt)
Figure 5-5

To set the status of an interrupt, the program first loads the D Register with a bit pattern consistent with the format in Figure 5-6. It then executes a POT instruction with an effective address of X'01TT', where "TT" represents the toggle switch address of the interrupt pair.

Bit	Meaning (If Bit is Set)
0	Makes bits 2, 3, 5, 6, and 7 effective
1	Makes bits 2, 3, 4, and 5 effective even if bit 0 is reset
2	Arms the Even Interrupt
3	Enables the Even Interrupt
4	Triggers the Even Interrupt
5	Causes Even Interrupt Address to be mapped
6	Sets Even Interrupt to the Waiting State
7	Sets Even Interrupt to the Active State
8	Makes bits 10, 11, 13, 14, and 15 effective
9	Makes bits 10, 11, 12, and 13 effective even if bit 8 is reset

Bit	Meaning (If Bit is Set)
10	Arms the Odd Interrupt
11	Enables the Odd Interrupt
12	Triggers the Odd Interrupt
13	Causes the Odd Interrupt Address to be mapped
14	Sets the Odd Interrupt to the Waiting State
15	Sets the Odd Interrupt to the Active State

D Register Format (Setting Interrupt)
Figure 5-6

SECTION 6

CONTROL PANEL

The operator control panel contains the controls and indicators necessary to display the current status of the computer, to change that status, and to make changes or insertions into registers and memory. There are also many maintenance functions provided on the control panel to allow easy isolation of hardware errors.

The control panel is depicted in Figure 6-1.

6.1 CONTROL PANEL SWITCHES AND INDICATORS

6.1.1 Power

The POWER switch is a toggle switch which controls primary AC power to the system.

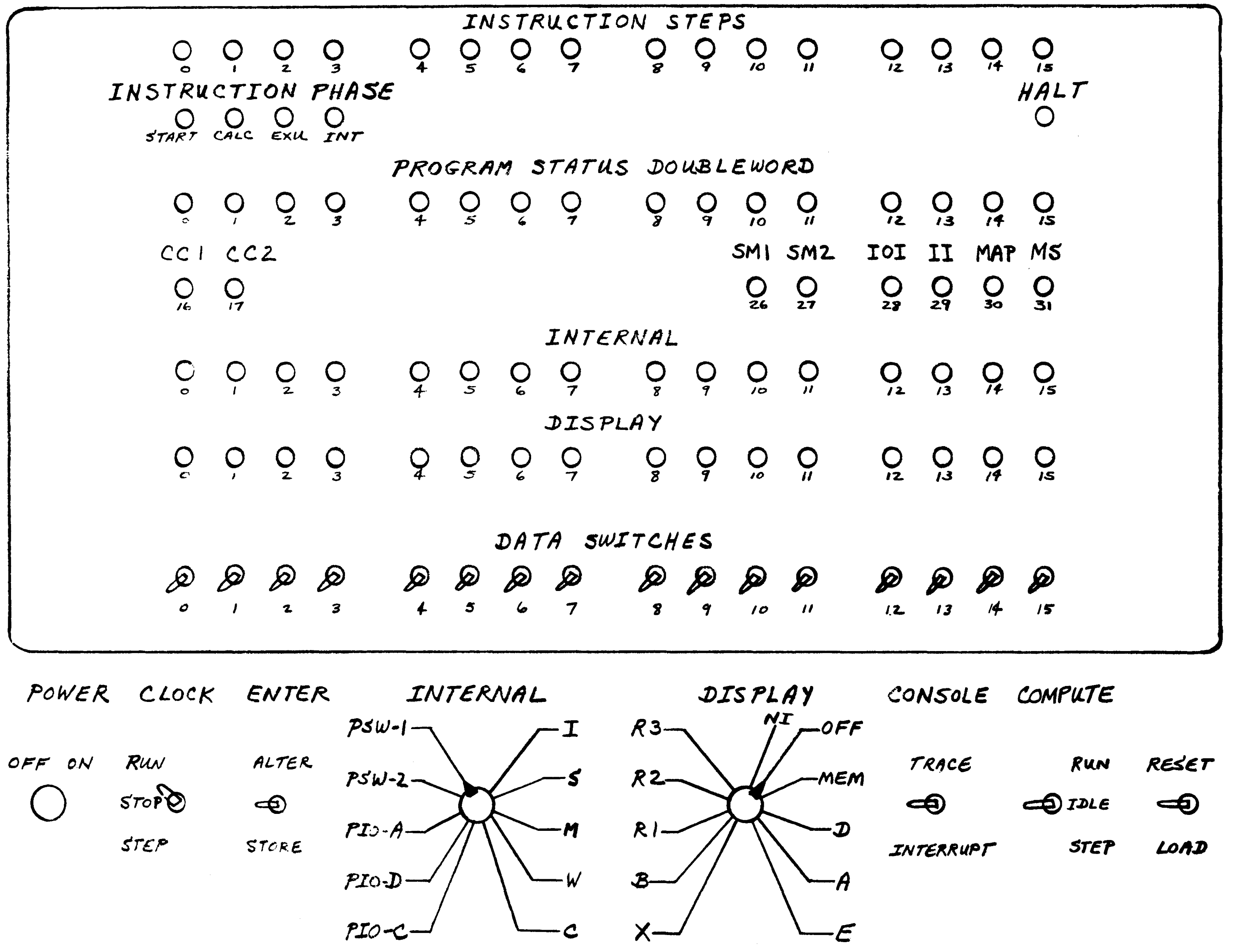
6.1.2 Instruction Steps

The INSTRUCTION STEPS lights indicate which instruction step the CPU is currently in for each instruction phase for each instruction executed. These indicators display the steps of instruction execution, I/O operation, and control panel operation. These indicators are primarily for use by maintenance personnel. Light indicator 0 is on at the beginning of an instruction execution phase and as the execution of the instruction proceeds, the current light goes out and the next step light goes on. This process continues until the next instruction phase begins at which time indicator light 0 begins again.

6.1.3 Instruction Phase

The INSTRUCTION PHASE lights indicate which phase of execution the CPU is in as it executes an instruction. The indicators specify the following:

FIGURE
CONTROL CONSOLE



1. START: The START light is on at the beginning of every instruction execution. All switches to the right of the CLOCK switch are operational when the START light is on. For example, when the CPU is set to the IDLE state, the START light is on and the INSTRUCTION STEPS light 0 is on.

2. CALC: The CALC light is on when the CPU is calculating the effective address of the instruction being executed.

3. EXU: The EXU light is on when the CPU is executing the current instruction.

4. INT: The INT light is on only during the time when an interrupt occurs and the old PSW1 and PSW2 are being stored and the new PSW1 and PSW2 are being accessed as specified in the interrupt service routine. The CPU will load in the interrupted routines defined PSW1 and PSW2 and then as soon as the CPU begins executing the interrupt service routine, the light will go out.

The INSTRUCTION PHASE lights are primarily for use by maintenance personnel.

6.1.4 Clock

The CLOCK RUN-IDLE-STEP switch allows the user to step through the instruction steps of an instruction while it is being executed by the CPU. This switch is latching in the RUN position and momentary in the STEP position. This switch is normally set to RUN when a program is being executed. The CPU will hang up when in IDLE and one clock timing will occur in an instruction execution each time the switch is moved to the STEP setting. The CLOCK switch is primarily for use by maintenance personnel.

6.1.5 Halt

The HALT light is on whenever the CPU is in the wait state.

6.1.6 Program Status Doubleword

The PROGRAM STATUS DOUBLEWORD displays the current operating status of the CPU and consists of two words as follows:

Word 1: Word 1 of the Program Status Doubleword (referred to as PSW1) contains the current program counter which is continually updated as the CPU executes a program to point at the next instruction location.

Word 2: Word 2 of the Program Status Doubleword (referred to as PSW2) contains the following indicators and associated lights on the operator's console:

Indicator	Meaning, If on
CC1	Condition Code 1 is set
CC2	Condition Code 2 is set
SM1	Processor is in the compare sequence mode
SM2	Further defines the compare sequence mode (described in 4.5.6)
IOI	Input/output service requests are inhibited
II	Interrupt service requests are inhibited
MAP	Processor is in the mapped mode
MS	Processor is in the slave mode

6.1.7 Internal Lights and Switch

The INTERNAL light indicators work in conjunction with the INTERNAL switch. The INTERNAL switch may be set to any of the following positions and each position represents an internal hardware register.

PSW1 refers to Program Status Word 1

PSW2 refers to Program Status Word 2

PIOA refers to the Programmed Input/Output address lines which contains the address going out to an I/O device

PIOD refers to the bidirectional Programmed Input/Output data lines which contains the data being transferred to/ from the addressed device. These indicators are only valid while single clocking through the execution of a Programmed Output or Programmed Input instruction.

PIOC refers to the Programmed Input/Output status lines which contains the condition codes (indicators 0 and 1) and the mapped bit (indicator 14) coming back to the CPU from the device. These indicators are only valid while single clocking through the execution of a Programmed Output or Programmed Input instruction.

I refers to the Instruction Register

S refers to the Effective Address Register

M refers to a CPU working register

W refers to a CPU working register

C refers to the output of the CPU's adder

For the most part, the INTERNAL lights and switch are used for maintenance personnel. The exception of this rule is whenever the user desires to modify PSW1 or PSW2. In all cases except for the PSW1 and PSW2, the setting of the INTERNAL switch causes the pertinent hardware register to be displayed in the INTERNAL lights. However, when the INTERNAL switch is set to PSW1 or PSW2 the pertinent register is not displayed in the INTERNAL lights as these registers are permanently displayed. These two settings are used in conjunction with the ENTER switch in the ALTER mode to modify the PROGRAM STATUS DOUBLEWORD word 1 or word 2 to the setting of the DATA SWITCHES.

6.1.8 Enter

The ENTER switch is a momentary switch which has two positions:

ALTER causes the DATA SWITCHES setting to enter the hardware register specified by the INTERNAL switch. This will allow registers PSW1, PSW2, I, S, M, W to be altered. However, the COMPUTE switch must be in the IDLE position.

STORE causes the DATA SWITCHES setting to enter the memory location specified in PSW1 and then automatically increments PSW1 by one. However, the COMPUTE switch must be in the IDLE position.

6.1.9 Display Lights and Switch

The DISPLAY light indicators work in conjunction with the DISPLAY switch. The DISPLAY switch may be set to any of the following positions and display the result in the DISPLAY lights:

R3 displays Utility Register 3

R2 displays Utility Register 2

R1 displays Utility Register 1

B displays Base Register

X displays the Index Register

E displays the Extended Arithmetic Register

A displays the Arithmetic Register

D displays the Data Register

MEM displays the contents of the location specified by the DATA SWITCHES

NI displays the next instruction to be executed as pointed to by PSW1

When the COMPUTE switch is in IDLE, any of the positions on the DISPLAY switch will display the appropriate value in the DISPLAY lights in a static fashion. When the COMPUTE switch is in RUN, the register or memory location specified by the DISPLAY switch is displayed in a dynamic fashion at a two-cycles-per-second rate. This means the user can dynamically follow a data pattern in any register or memory location as the program is executing.

6.1.10 Console

The CONSOLE switch is momentary in the INTERRUPT position and latching in the TRACE position. The TRACE position causes an interrupt to occur after every instruction executed such that a trace routine can follow the processing logic of a program as it is executing. The INTERRUPT position causes a momentary console interrupt to trigger in the CPU.

6.1.11 Compute

The COMPUTE switch is latching in the RUN position and momentary in the step position. The COMPUTE switch must be in IDLE for many of the other switches to be operational. When the RUN mode is selected, program execution begins as defined in PSW1 and PSW2. When placed in the STEP position, the CPU executes one instruction and halts.

6.1.12 Reset/Load

The RESET/LOAD switch is momentary in both positinns. This switch will operate only if the COMPUTE switch is set to IDLE.

When placed in the RESET position, I/O devices on the system are reset, PSW1 is set to hexadecimal 50, and PSW2 is set to 0.

When placed in the LOAD position, the program contained on the (optional) Bootstrap Loader will be placed into memory beginning at hexadecimal location 50.

6.1.13 Data Switches

The DATA SWITCHES are two position switches that are latching in the 1 (up) and 0 (down) positions. These switches allow the user to define 16-bit data words and memory addresses to be used in conjunction with other switches on the console.

6.2 LOADING PROCEDURE

The loading procedure is preformed as follows:

1. Place the COMPUTE switch to IDLE.
2. Enter the device number now being loaded from into the DATA SWITCHES right justified.
3. Place the RESET/LOAD switch in the RESET position -- which initializes the CPU.
4. Place the RESET/LOAD switch in the LOAD position -- which brings the (optional) Bootstrap Loader into core memory beginning at location hexadecimal 50.
5. Place the RESET/LOAD switch in the RESET position -- which resets the CPU and all devices on the system. PSW1 is also set to hexadecimal 50 and PSW2 is set to 0.
6. Place the COMPUTE switch to RUN and the CPU beings execution of the Bootstrap Loader beginning in memory location hexadecimal 50 which will boot in from the device specified in the DATA SWITCHES.

6.3 MODIFYING MEMORY FROM THE CONSOLE

Any core memory location or programmable register can be modified as follows:

1. Place the COMPUTE switch to IDLE.

2. Set PSW1 equal to the location to be modified. Set the DATA SWITCHES to the memory location desired, place the ENTER switch to the ALTER position.

3. Enter the value to be stored in the DATA SWITCHES.

4. Place the ENTER switch to the STORE position -- which causes the value in the data switches to be stored into the location pointed to by PSW1. PSW1 is automatically incremented by one. PSW1 is then assumed to be a virtual or actual address depending upon the setting of the MAP bit in PSW2 (bit 14).

5. Keep entering new values into the DATA SWITCHES and toggling the ENTER switch to STORE -- which stores data into sequential memory locations.

6.4 READING OUT MEMORY FROM THE CONSOLE

Any core location can be read out to the DISPLAY lights as follows:

1. Set the DISPLAY switch to MEM and the memory location desired into the DATA SWITCHES.

2. The contents of the location specified in the DATA SWITCHES is then automatically displayed in the DISPLAY lights.

   A toggle switch located on the inside of the control panel determines whether the location pointed to by the DATA SWITCHES is an actual or virtual address (normally a virtual address).

Setting the DISPLAY switch to MEM and the DATA SWITCHES to any location desired, that location will be displayed in the DISPLAY lights and updated at a two-cycles-per-second rate. The user may arbitrarily change the DATA SWITCHES to any address as the program is executing, and that memory cell will be displayed in the DISPLAY lights at a two-cycles-per-second rate -- even when the COMPUTE switch is in RUN and the program is dynamically changing core locations.

multidata
multidata
multidata
multidata
multidata
multidata
multidata
multidata
multidata
multidata
multidata
multidata
multidata
multidata
multidata
multidata
multidata
multidata
multidata
multidata

102687001