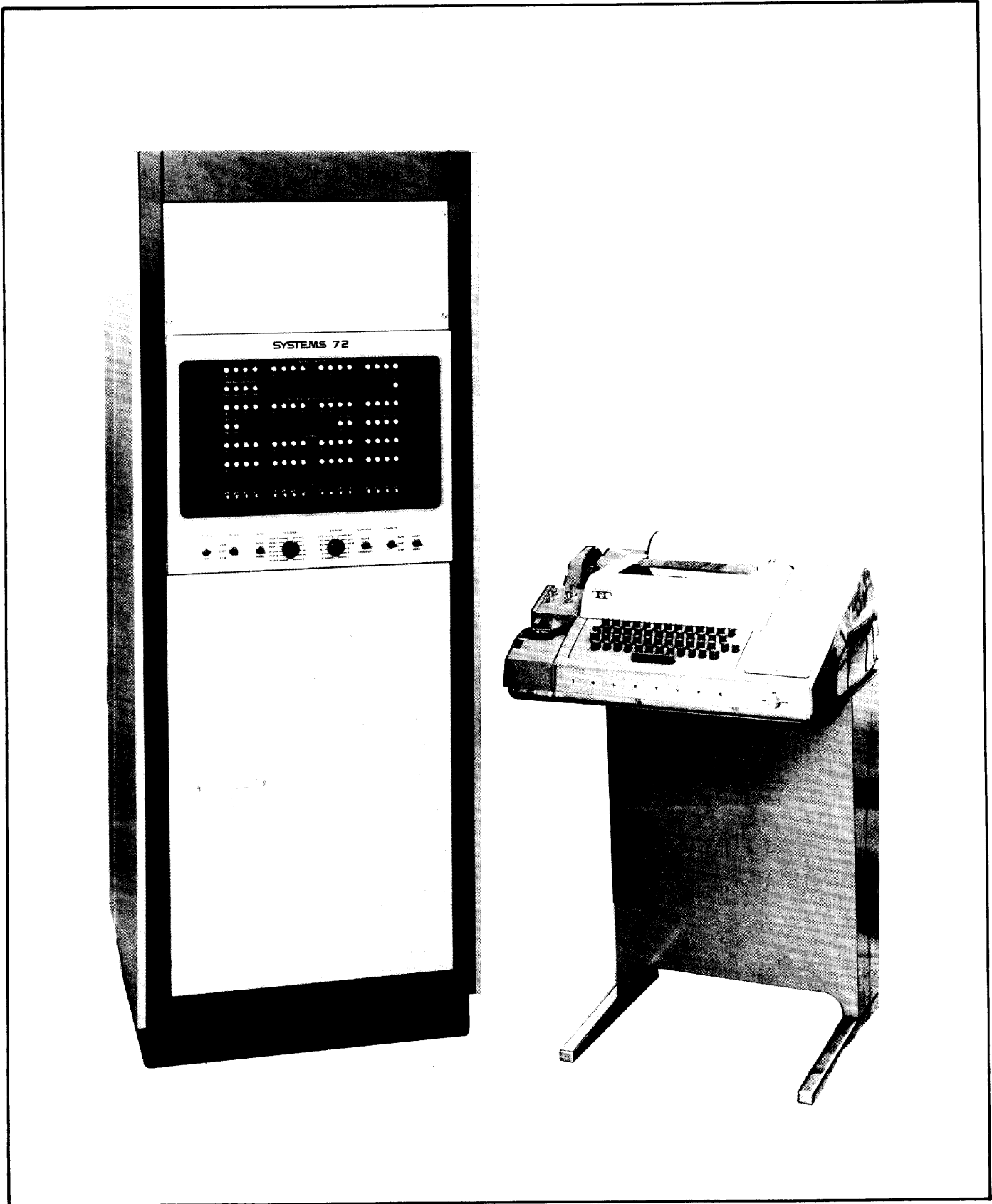


**Computer Reference Manual**  
**SYSTEMS 72**  
**Digital Computer**  
**April 1972**



Typical SYSTEMS 72 Digital Computer and Teletype

## LIST OF EFFECTIVE PAGES

The total number of pages in this manual is 184, consisting of the following pages:

Page Number	Issue
Title	Original
A	Original
i through iv	Original
1-1 through 1-12	Original
2-1 through 2-54	Original
3-1 through 3-10	Original
4-1 through 4-8	Original
A-1 through A-4	Original
B-1 through B-42	Original
C-1 through C-6	Original
D-1 through D-14	Original
E-1 through E-2	Original
F-1 through F-2	Original
G-1 through G-2	Original
H-1 through H-10	Original
I-1 through I-8	Original

## TABLES OF CONTENTS

Section	Title	Page
SECTION I	INTRODUCTION AND DESCRIPTION	
	Introduction	1-1
	Purpose of Equipment	1-1
	Optional Equipment	1-2
	Physical Configuration	1-2
	Functional Configuration	1-2
	Control Panel	1-8
	Power Turn on Procedure	1-8
	Loading Procedures	1-10
	Loading - No Optional Loader	1-10
	Loading - Optional Loader	1-11
	Modifying Memory From the Control Panel	1-11
	Displaying Memory at the Control Panel	1-12
	Hexadecimal Notation	1-12
SECTION II	CENTRAL PROCESSOR UNIT	
	Introduction	2-1
	CPU Operational Phases	2-1
	Instruction Format	2-3
	Addressable Registers	2-4
	Program Status Doubleword	2-5
	Address Calculation	2-9
	Relative Addressing	2-9
	Relative Base Addressing	2-9
	Indirect Addressing	2-12
	Post-Indexing	2-12
	Absolute Addressing	2-12
	Instruction Repertoire	2-12
	Branch Instructions	2-14
	Load Instructions	2-21
	Store Instructions	2-28
	Arithmetic Instructions	2-32
	Logical Instructions	2-36
	Compare Instruction	2-40
	Shift Instructions	2-42
	Call Instructions	2-46
	Input/Output Instructions	2-51
	Other Instructions	2-54
SECTION III	CORE/DISC MEMORY SYSTEM	
	Introduction	3-1
	Basic Core/Disc Memory System	3-1
	Memory System Functions	3-1
	Core Memory	3-2
	Dedicated Core Locations	3-2
	Data Guard	3-3
	Memory Parity Check	3-3
	Memory Ports	3-3
	Disc Memory	3-4

## TABLE OF CONTENTS (CONT'D)

		Page
SECTION III	CORE/DISC MEMORY SYSTEM	
(Cont'd)		
	Actual/Virtual Memory Configuration	3-4
	Memory Map Format	3-6
	Memory Map Operation	3-6
	Memory Map Update	3-7
	Memory Traps	3-7
	Sense Disc Rotational Position	3-8
	Output Disc Track and Sector Address	3-9
	Output Core Bank and Page Address	3-9
	Page Transfer Terminator	3-9
SECTION IV	INPUT/OUTPUT SYSTEM	
	Introduction	4-1
	Dual Level Interrupt System	4-1
	I/O Priority	4-2
	Interrupt Inhibit	4-2
	Interrupt Clear	4-2
	System Interrupts	4-2
	Set System Interrupts	4-3
	Sense System Interrupts	4-4
	Memory Traps	4-4
	Input/Output Command List	4-4
	Input/Output Instructions	4-6
	Basic Input/Output Operation	4-7
	PIN/POT I/O Capability	4-8
APPENDIX A	Glossary of Terms	A-1
APPENDIX B	SYSTEMS 72 Derivative Instructions	B-1
APPENDIX C	Sample Input/Output Program	C-1
APPENDIX D	Reference Tables	D-1
APPENDIX E	ASCII Character Set and Hexadecimal Codes	E-1
APPENDIX F	SYSTEMS 72 Instructions - Alphabetical Listing	F-1
APPENDIX G	SYSTEMS 72 Instructions - Numerical Listing	G-1
APPENDIX H	Effective Address Calculation Times	H-1
APPENDIX I	SYSTEMS 72 Special Derivatives Instructions	I-1

## LIST OF ILLUSTRATIONS

Figure	Title	Page
1-1	Basic SYSTEMS 72 Digital Computer, Front View	1-6
1-2	Simplified Block Diagram - SYSTEMS 72 Digital Computer	1-7
1-3	Front Panel Controls and Indicators	1-9
2-1	Block Diagram - CPU Operational Phases	2-2
2-2	Sequence Diagram - Effective Address Calculation	2-10
2-3	Calculation Diagram - Effective Address	2-11
3-1	Actual/Virtual Memory Configuration (Basic System)	3-5
4-1	Simplified Diagram - Input/Output System	4-5

## LIST OF TABLES

Table	Title	Page
1-1	SYSTEMS 72 Options	1-3
1-2	SYSTEMS 72 Software	1-4
2-1	Condition Code Configurations for Basic Instructions	2-7
2-2	CPU Basic Instructions	2-13

## LIST OF RELATED PUBLICATIONS

The following publications contain information not included in this manual, but necessary for a complete understanding of the SYSTEMS 72 Digital Computer.

<u>Publication Title</u>	<u>Publication No.</u>
MA/10 Operating System Reference Manual	323-720001
Math Library Reference Manual	323-720002
FORTTRAN IV Reference Manual	323-720003
MA/20 Reference Manual	*323-720004
MAP Assembler Reference Manual	323-720006
BASIC Reference Manual	323-720007
Interface Design Manual	310-770000
CPU/Memory Maintenance Guide	303-780000

### NOTE

(\* Preliminary)

## SECTION I

### INTRODUCTION AND DESCRIPTION

#### INTRODUCTION

This manual consists of four sections and three appendices that describe the SYSTEMS 72 Digital Computer. Detailed reference information is provided for the major functional areas of the equipment, such as the central processor, core/disc memory system, and input/output system. The detailed information describes the instruction set used by the central processor, the actual/virtual memory concept used by the core/disc memory system, and the input/output programming used by the input/output system.

#### PURPOSE OF EQUIPMENT

The purpose of SYSTEMS 72 is to offer the user a low-cost computer capable of executing large-scale programs while operating with a variety of input/output (I/O) devices. Modular expansion of core memory from 4096 sixteen-bit words for the basic configuration of the SYSTEMS 72, to 65,536 words in 4096-word increments, offers the user an effective trade-off in the higher cost of core versus the speed and efficiency at which the user's programs are executed.

Major features offered by the SYSTEMS 72 are as follows:

- 4096 word core memory expandable to 65,536 words
- 880-nanosecond core memory cycle time
- 32,768-words of programmable memory, expandable to 65,536 words
- 65,536-word memory extension disc, expandable to 131,072 words or 262,144 words.
- Virtual programmable memory to actual memory map
- Core memory write protection
- Automatic program fragmentation
- Dynamic program relocation
- Power fail safe
- Only single-word instructions
- User-defined instructions
- Privileged instructions
- Five-bit operation field
- Eight addressable registers
- Relative addressing, forward and backward
- Base-relative addressing
- Single-level indirect addressing
- Post-indexing
- Multilevel interrupt system
- Rapid context switching
- IOP-oriented input/output system
- Aysnchronous, demand-multiplexed input/output
- Data and command chaining
- Wide variety of hardware options and I/O devices



OPTIONAL  
EQUIPMENT

A wide variety of hardware options and I/O devices are provided for use with the SYSTEMS 72. Hardware options include priority interrupts, real-time clocks, memory parity, high speed registers, automatic bootstrap loader, high speed multiply/divide, console interrupt and direct access to memory. Peripheral I/O devices include paper tape punch/reader, card reader, various teletypes impact printers, disc drives, 7- and 9-track tape transports, and asynchronous data sets.

Table 1-1 lists the optional equipment available for use with SYSTEMS 72.

PHYSICAL  
CONFIGURATION

The basic configuration of SYSTEMS 72 is illustrated in figure 1-1. A card frame assembly mounted behind the control panel houses the circuit cards that comprise the central processor unit (CPU), core/disc memory system, power fail safe, and teletype controller. The first 4096-word core memory module, which is supplied with the basic system, is also installed in the card frame assembly. The card frame assembly will accommodate an additional Model 7230 Core Memory Module; however, further core memory expansion requires the Model 7231 Memory Extender Chassis.

In addition to the teletype controller and power fail safe, five I/O slots are provided in the card frame assembly to accommodate hardware options or controllers for the various I/O devices. If more than five slots are required for the hardware options or device controllers, the basic system capability is expanded by the Model 7271 I/O Extender Chassis.

The Model 7220 High Speed Multiply/Divide is installed in a dedicated slot in the card frame assembly. The same slot may also be used for other customer-required optional instructions. Another dedicated slot is provided for the Model 7240 Direct Access Channel or the Model 7241 Multiplexed Input/Output Processor.

Disc memory capacity of the SYSTEMS 72 may be increased by installing one of the Model 7235 through 7236 Disc Assemblies in place of the 65,536-word disc supplied with the basic system. Use of a larger disc memory in conjunction with the Model 7232 Memory Map Extension increases the programmable memory capacity of the system to 65,536 words.

The exact rack-mounted configuration of a particular system depends upon the number of rack-mounted peripheral devices, and whether or not the system requires the Model 7231 Memory Extender Chassis and Model 7271 I/O Extender Chassis.

FUNCTIONAL  
CONFIGURATION

The functional configuration of the basic SYSTEMS 72 is illustrated in figure 1-2. Data processing is controlled by the CPU, which receives input data from the control panel, teletype, or other I/O device. The CPU subsequently transfers the information to the core memory by way of the core/disc memory system.

Table 1-1. SYSTEMS 72 Options

Model	Description	Model	Description
4005	61-Inch Cabinet	4531	9-Track Tape Transport
4021	86-Pin Card Extender	4535	7-Track Tape Transport and Controller
4022	60-Pin Card Edge Connector	4536	7-Track Tape Transport
4023	-Conductor Cable	4590	Magnetic Tape Reel
4024	60/25-Pin Connector Cable Assembly	4701	Asynchronous Data Set and Controller
4103	Paper Tape Punch/Reader and Controller	4760	General Purpose I/O Interface
4211	Card Reader and Controller	7200	SYSTEMS 72 Basic Computer
4302	ASR-33 Teletype and Controller	7210	Memory Parity Trap Card Assembly
4303	KSR-33 Teletype and Controller	7211	Real-Time Clock Card Assembly
4313	ASR-35 Teletype and Controller	7212	High Speed Register Card Assembly
4314	KSR-35 Teletype and Controller	7214	Automatic Bootstrap Loader Card Assembly
4361	Buffered Impact Printer and Controller	7220	High Speed Multiply/Divide Card Assembly
4412	24 Megabyte Disc Drive and Controller	7230	4K Core Memory Card Assembly
4414	48 Megabyte Disc Drive and Controller	7231	Memory Extension Chassis
4415	2.4 Megabyte Disc Drive and Controller	7232	Long Map Card Assembly
4417	24 Megabyte Disc Drive	7235	131K-Word Disc Assembly
4419	48 Megabyte Disc Drive	7236	262K-Word Disc Assembly
4420	2.4 Megabyte Disc Drive	7241	Multiplexer Input/Output Processor
4481	2.4 Megabyte Disc Pack	7242	MIOP Expansion
4485	48 Megabyte Disc Pack	7245	Direct Access Interface Card Assembly
4499	24 Megabyte Disc Pack	7250	Console Interrupt Card Assembly
4530	9-Track Tape Transport and Controller	7251	Priority Interrupt Pair Card Assembly
		7271	I/O Extender Chassis
		72910	MA/20

TABLE 1-2. SYSTEMS 72 SOFTWARE

CODE	NUMBER	TITLE
AAA	8000000	
AAB	8000001	MA/10 Kernel
AAC	8000002	MA/10 Programmed IOP
AAD	8000003	Loader TextLister
AAE	8000004	Symbolic Editor
AAF	8000005	MA/CC1
AAG	8000006	Symbol Concordance
AAH	8000007	MA/10 Linking Loader
AAI	8000008	MA/10 System Generator
AAJ	8000009	MAP Assembler
AAK	8000010	BASIC Compiler
AAL	8000011	FORTRAN IV Compiler
AAM	8000012	MA/10 Virtual Non-Linking Loader (32K)
AAN	8000013	Debug - Basic
AAO	8000014	Debug- EXTENDED
AAP	8000015	MA/10 System Loader (32K)
AAQ	8000016	MA/10 IOCS
AAR	8000017	Card Reader Handler
AAS	8000018	Line Printer Handler
AAT	8000019	MA/10 Fixed Head Disc
AAU	8000020	MA/10 System INITIALIZE
AAV	8000021	
AAW	8000022	
AAX	8000023	MA/10 Virtual Non-Linking Loader (65K)
AAY	8000024	MA/10 System Loader (65K)
AAZ	8000025	MA/10 Movable Head Disc Handler
ABA	8000026	Monitor Services
ABB	8000027	MA/10 SYSPAGE
ABC	8000028	MA/10 IOCS Dibs/Cibs
ABD	8000029	MA/10 Teletype Handler
ABE	8000030	MA/10 Cassette & Magnetic Tape
ABF	8000031	MEDIA
ABG	8000032	D.P. Fixed Pt. Two's Complement and D.P. Fixed Pt. Absolute
ABH	8000033	Fl. Pt. Load and Fl. PTStore
ABI	8000034	D.P. Fixed Pt. Load & D.P. Fixed Pt.Store
ABJ	8000035	Floating Point Normalize
ABK	8000036	Double Precision Fixed point Add, Subtract
ABL	8000037	Single Precision Fixed Point Multiply
ABM	8000038	Single Precision DIV
ABN	8000039	Floating Point Overflow
ABO	8000040	Floating Point Add, Subtract
ABP	8000041	Fl. Pt. Comp. and D.P. Fixed Pt. Comp
ABQ	8000042	Fl. Pt. Multiply and D.P. Fixed Pt. Multiply
ABR	8000043	Fl. Pt. Divide and D.P. Fixed Pt. Divide
ABS	8000044	Binary Integer to ASCII Conversion
ABT	8000045	ASCII to Binary Integer Conversion

TABLE 1-2. SYSTEMS 72 SOFTWARE (Cont'd)

CODE	NUMBER	TITLE
ABU	8000046	ASCII to Floating Point Conversion
ABV	8000047	Floating Point to ASCII Conversion
ABW	8000048	Floating Point Square Root
ABX	8000049	Floating Point Natural Logarithm
ABY	8000050	Floating Point Common Logarithm
ABZ	8000051	Floating Point Exponential ( $e^x$ )
ACA	8000052	Floating Point Tangent
ACB	8000053	Floating Point Sine, Cosine
ACC	8000054	Floating Point Arc Tangent
ACD	8000055	Floating Point Hyperbolic Tangent
ACE	8000056	Random Number Generator
ACF	8000057	Floating Point to Fixed Point Conversion
ACG	8000058	Fixed Point to Floating Point Conversion
ACH	8000059	CPU Confidence Test
ACI	8000060	Full Memory Test
ACJ	8000061	TTY Read/Punch Test
ACK	8000062	Disc File Test
ACL	8000063	Map Register Test
ACM	8000064	Upper 4K Memory Test
ACN	8000065	DAC Test
ACO	8000066	Absolute Dumper
LIB	801000	IOCS' Library
LIB	801001	Math Library

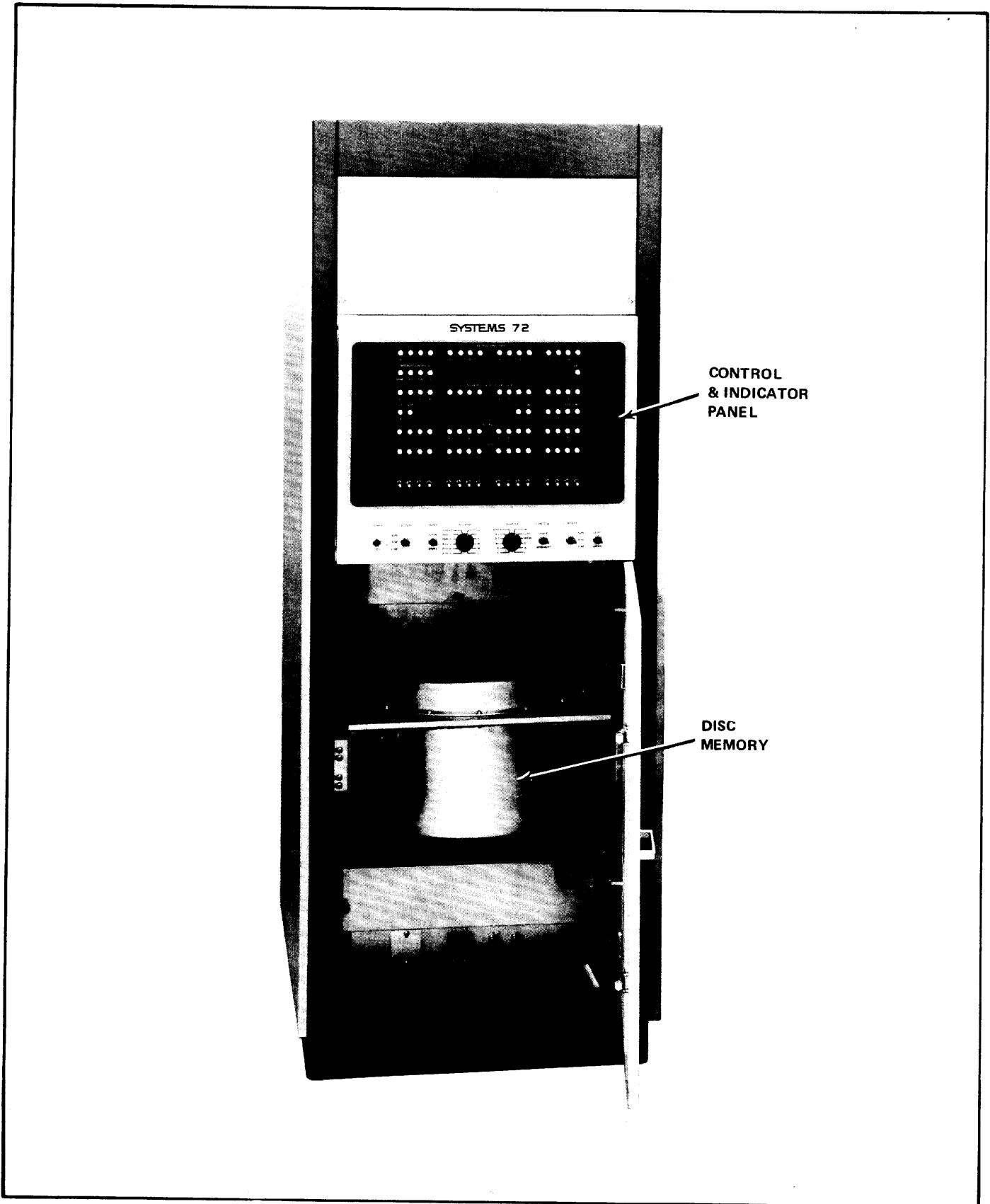


Figure 1-1. Basic SYSTEMS 72 Digital Computer, Front View

Information to be processed and transferred to the control panel, teletype, or other I/O device is read from the core memory under control of the core/disc memory system. The information is subsequently transferred to the CPU for output to the control panel or I/O device.

I/O data transfers are accomplished under control of an Input/Output Processor (IOP). The Input/Output Processor (IOP) is provided in the basic system as a Programmed Input/Output Processor (PIOP) or optionally as a hardware Multiplexed Input/Output Processor (MIOP). The PIOP is software-implemented, it appears to be a separate entity to both the user's program and the operating system. To set up a data transfer, the user's program specifies the direction and size of transfer, the appropriate location in virtual memory, and the action to be taken upon termination of the transfer. The Multiplexed Input/Output Processor (MIOP) is a hardware option that can be used to replace the PIOP. It serves to increase the bandwidth to about 700,000 words per second. The addition of this option has no effect on programs already written.

Several busses are used to facilitate transfer of data and address information. Data transfers between the CPU and control panel or I/O device are accomplished by way of the program I/O data bus, while data transferred between the CPU and core/disc memory system is exchanged by way of the memory data bus. The source of the data input to the CPU, or destination of the data output from the CPU, is defined by an effective address, which is calculated by the CPU and output on the associated bus.

Programs stored in the core or disc memory control the processing sequence. The core/disc memory system, under control of a small core-resident program which retrieves information required by the current program from the disc memory and stores the information in core for use by the CPU during processing. Information not required by the current program is transferred from core to disc.

Operator intervention into the processing sequence is by means of the switches located on the control panel, teletype, or other I/O device.

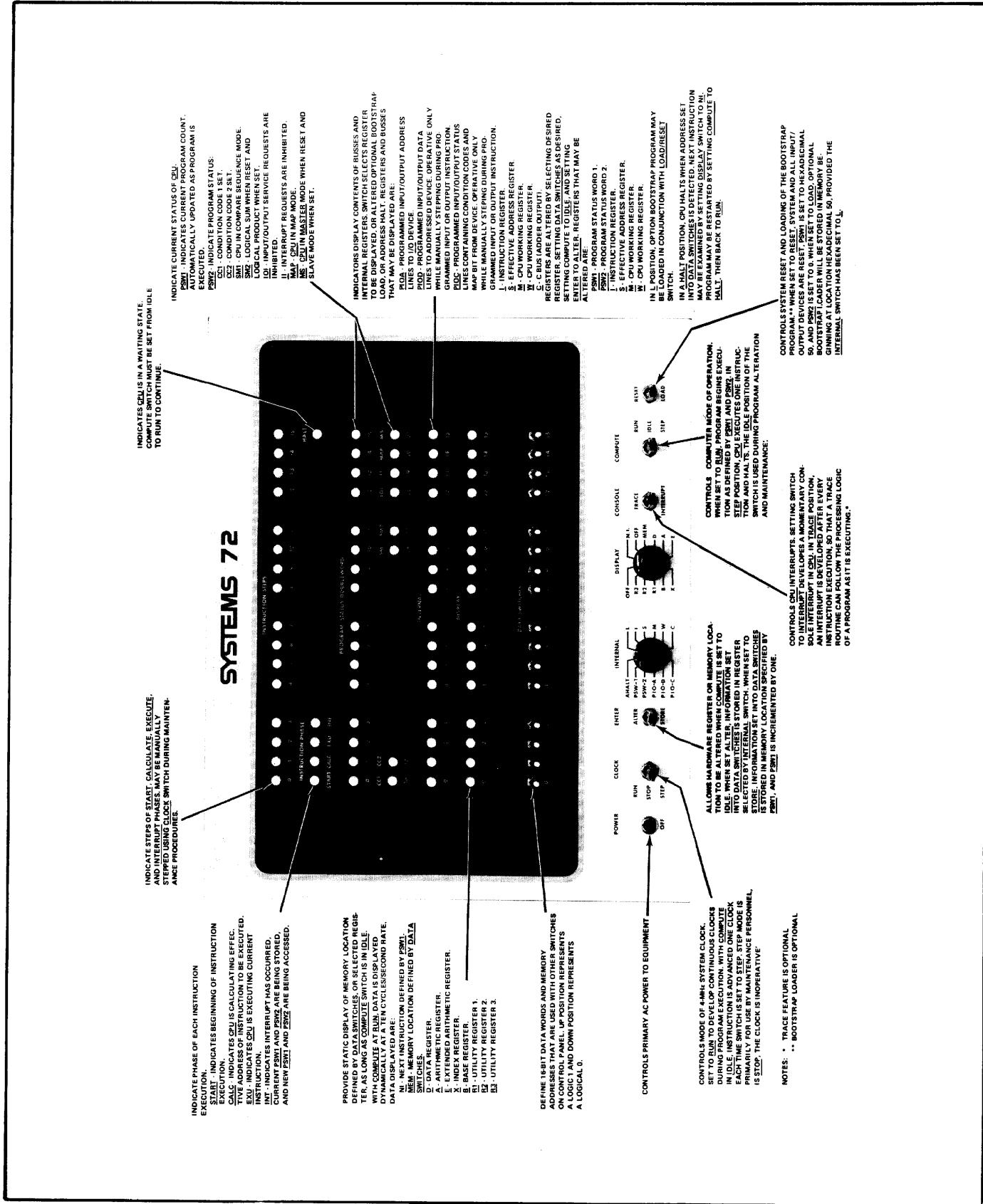
A Direct Access Channel (DAC), or a Multiplexed Input/Output Processor (MIOP) are provided as hardware options. The DAC permits direct data transfers between an I/O device and the core memory, so that the transfer is not under control of the CPU. This transfer rate is 1,000,000 words per second. The MIOP allows for the servicing of up to 64 devices concurrently. The bandwidth here is 700,000 words per second.

### CONTROL PANEL

The operator control panel contains the controls and indicators necessary to display the current status of the computer, change the status as required, and alter the various registers and memory. Certain switches and indicators are also provided to facilitate maintenance of the computer. Figure 1-3 illustrates the controls and indicators on the operator control panel and provides a brief description of the function of each control and indicator.

### POWER TURN ON PROCEDURE

To apply power to the basic configuration of SYSTEMS 72, connect the power cord from the system power supply to a source of 115-volt ac, single-phase power. Set the circuit breaker on the ac distribution panel in each rack to ON.



INDICATES CPU IS IN A WAITING STATE. COMPUTE SWITCH MUST BE SET FROM IDLE TO RUN TO CONTINUE.

INDICATES PHERS OF HEART, CALCULATE, EXECUTE, AND INTERRUPT PHASES. MAINTENANCE STEPPED USING CLOCK SWITCH DURING MAINTENANCE PROCEDURES.

INDICATE PHASE OF EACH INSTRUCTION START. INDICATES BEGINNING OF INSTRUCTION EXECUTION.  
 CALS. INDICATES CPU IS CALCULATING EFFEC. TIVE ADDRESS OF INSTRUCTION TO BE EXECUTED.  
 INT. INDICATES INTERRUPT HAS OCCURRED.  
 CURRENT PSW1 AND PSW2 ARE BEING STORED, AND NEW PSW1 AND PSW2 ARE BEING ACCESSED.

INDICATE CURRENT STATUS OF CPU.  
 PSW1. INDICATES CURRENT PROGRAM COUNT. ADDRESS MANUALLY UPDATED AS PROGRAM IS EXECUTED.  
 PSW2. INDICATES PROGRAM STATUS:  
 001. CONDITION CODE 1 SET.  
 002. CONDITION CODE 2 SET.  
 003. ADDRESS MAP MODE.  
 004. LOGICAL SUM WHEN RESET AND LOGICAL PRODUCT WHEN SET.  
 101. INPUT/OUTPUT SERVICE REQUESTS ARE INHIBITED.  
 102. INPUT/OUTPUT REQUESTS ARE INHIBITED.  
 103. CPU IN MASTER MODE WHEN RESET AND SLAVE MODE WHEN SET.

PROVIDE STATIC DISPLAY OF MEMORY LOCATION DEFINED BY DATA SWITCHES. OR SELECTED REGISTER, AS LONG AS COMPUTE SWITCH IS IN IDLE POSITION. REGISTER ADDRESS IS DYNAMICALLY ALLOCATED AT A 100 CYCLES/SECOND RATE. DATA DISPLAYED ARE:  
 NI. NEXT INSTRUCTION LOCATION DEFINED BY PSW1.  
 NI+. NEXT MEMORY LOCATION DEFINED BY DATA SWITCHES.  
 D. DATA REGISTER.  
 A. ARITHMETIC REGISTER.  
 E. EXTENDED ARITHMETIC REGISTER.  
 Z. INDEX REGISTER.  
 R1. UTILITY REGISTER 1.  
 R2. UTILITY REGISTER 2.  
 R3. UTILITY REGISTER 3.

INDICATORS DISPLAY CONTENTS OF BUSSES AND DATA REGISTER. REGISTER ADDRESS IS TO BE DISPLAYED OR REGISTER ADDRESS THAT LOAD, ON ADDRESS HALL, REGISTER AND BUSES THAT MAY BE DISPLAYED ARE:  
 P00. PROGRAMMED INPUT/OUTPUT ADDRESS.  
 P01. PROGRAMMED INPUT/OUTPUT DATA.  
 P02. PROGRAMMED INPUT/OUTPUT ADDRESS LINES TO I/O DEVICE.  
 P03. PROGRAMMED INPUT/OUTPUT ADDRESS LINES TO ADDRESS DECODE. OPERATIVE ONLY WHILE MANUALLY STEPPING DURING PRO. GRAMMED INPUT OR OUTPUT INSTRUCTION.  
 P04. PROGRAMMED INPUT/OUTPUT STATUS.  
 P05. PROGRAMMED INPUT/OUTPUT ADDRESS MAP BIT FROM DEVICE. OPERATIVE ONLY WHILE MANUALLY STEPPING DURING PRO. GRAMMED INPUT OR OUTPUT INSTRUCTION.  
 I. INSTRUCTION REGISTER.  
 M. MASTER REGISTER.  
 N. CPU WORKING REGISTER.  
 S. CPU WORKING REGISTER.  
 C. CPU WORKING REGISTER.

DEFINE 16-BIT DATA WORDS AND MEMORY ADDRESSES THAT ARE USED WITH OTHER SWITCHES ON CONTROL PANEL. UP POSITION REPRESENTS A LOGIC 1 AND DOWN POSITION REPRESENTS A LOGIC 0.

PSW1. PROGRAM STATUS WORD 1.  
 PSW2. PROGRAM STATUS WORD 2.  
 I. INSTRUCTION REGISTER.  
 S. EFFECTIVE ADDRESS REGISTER.  
 M. CPU WORKING REGISTER.  
 N. CPU WORKING REGISTER.  
 IN I/L POSITION, OPTION BOOTSTRAP PROGRAM MAY BE LOADED IN CONJUNCTION WITH LOAD/RESET IN A HALL POSITION. CPU HALTS WHEN ADDRESS SET INTO DATA SWITCHES IS DETECTED. NEXT INSTRUCTION MAY BE EXAMINED BY SETTING DISPLAY SWITCH TO NI. PROGRAM MAY BE RESTARTED BY SETTING COMPUTE TO HALL, THEN BACK TO RUN.

CONTROLS PRIMARY AC POWER TO EQUIPMENT

CONTROLS COMPUTER MODE OF OPERATION. WHEN SET TO RUN, PROGRAM BEGINS EXECUTION. WHEN SET TO STOP, CPU EXECUTES ONE INSTRUCTION AND HALTS. THE IDLE POSITION OF THE SWITCH IS USED DURING PROGRAM ALTERATION AND MAINTENANCE.

CONTROLS MODE OF 4-MHZ SYSTEM CLOCK. SET TO RUN TO DEVELOP CONTINUOUS CLOCKS DURING PROGRAM EXECUTION. WITH COMPUTE SWITCH IN STOP POSITION, INSTRUCTION IS ADVANCED ONE CLOCK EACH TIME CLOCK SWITCH IS Toggled. THIS IS PRIMARILY FOR USE BY MAINTENANCE PERSONNEL. IS STOP, THE CLOCK IS INOPERATIVE.

CONTROLS CPU INTERRUPTS. SETTING SWITCH TO INTERRUPT DEVELOPES A MOMENTARY CONSOLE INTERRUPT IN CPU IN TRACE POSITION. INTERRUPT PHASES ARE STORED IN MEMORY LOCATION SPECIFIED BY PSW1, AND PSW2 IS INCREMENTED BY ONE.

NOTES: \* TRACE FEATURE IS OPTIONAL  
 \*\* BOOTSTRAP LOADER IS OPTIONAL

CONTROLS SYSTEM RESET AND LOADING OF THE BOOTSTRAP PROGRAM. \*\* WHEN SET TO RESET, SYSTEM AND ALL INPUT/OUTPUT DEVICES ARE RESET. PSW1 IS SET TO HEXADECIMAL 90, AND PSW2 IS SET TO 0. WHEN SET TO LOAD, OPTIONAL BOOTSTRAP PROGRAM IS LOADED INTO MEMORY LOCATION BEGINNING AT LOCATION HEXADECIMAL 50, PROVIDED THE INTERNAL SWITCH HAS BEEN SET TO L.

Figure 1-3. Front Panel Controls and Indicators

LOADING  
PROCEDURES

Loading procedures for SYSTEMS 72 depend upon whether the configuration is equipped with a Model 7214 Automatic Bootstrap Loader. If the optional loader is not installed, program information is assumed to originate at the teletype. In systems where the optional loader is installed, the DATA SWITCHES on the control panel are used to select the peripheral device from which the program information will be loaded. The first set of procedures listed below are for systems without the optional loader, and the second set are for systems with the loader.

LOADING - NO  
OPTIONAL LOADER

- a. On the computer control panel, set the controls as follows:

<u>SWITCH</u>	<u>SETTING</u>
CLOCK	RUN
ALTER/STORE	OFF = Center position
INTERNAL	PSW 1
DISPLAY	OFF
CONSOLE	OFF = Center position
COMPUTE	IDLE
RESET/LOAD	OFF = Center position
DATA	OFF

- b. Momentarily set the RESET/LOAD switch to RESET.

- c. Key in the bootstrap loader by sequentially setting the DATA SWITCHES to the hexadecimal values listed below and momentarily setting the ALTER/STORE switch to STORE.

<u>MEMORY LOCATION</u>	<u>HEXADECIMAL VALUE</u>
50	0B5F
51	0960
*52	* 0608
53	0803
*54	*0834
55	1255
56	
57	
58	005A
59	0000
5A	* 0148
5B	0900
5C	2700
5D	0F03
5E	3756
5F	0200
60	0056



In this example, the memory locations that have asterisks (52, 54, and 5A) are configured for a device on channel 8. If a device is on another channel, the values of the memory slots would change. Example: Paper reader on channel 'B' 52=060B, 54=8037, and 5A=014B.

- d. Momentarily set the RESET/LOAD switch to RESET.
- e. Install appropriate punched paper tape in the teletype paper tape reader.
- f. Set COMPUTE switch on computer control panel to RUN.
- g. When tape stops, set COMPUTE switch to IDLE.
- h. Momentarily set RESET/LOAD switch to RESET.
- i. Set DATA SWITCH 7 to the UP position.
- j. Momentarily set ALTER/STORE switch to ALTER.
- k. Set COMPUTE switch to RUN to start program from location X'0100'.

LOADING  
OPTIONAL LOADER

- a. On the computer control panel, set COMPUTE switch to IDLE, and the INTERNAL Switch To L.
- b. Momentarily set RESET/LOAD switch to RESET, then to LOAD, and back to RESET.
- c. Set the appropriate device address into the DATA SWITCHES right justified.
- d. Install punched paper tape, magnetic tape, etc. on appropriate peripheral device.
- e. Set COMPUTE switch on computer control panel to RUN.
- f. When device stops, set COMPUTE TO IDLE.
- g. Momentarily set RESET/LOAD switch to RESET, then to LOAD.
- h. Set COMPUTE switch to RUN to start program from location X'0101'.

MODIFYING  
MEMORY FROM  
THE CONTROL  
PANEL

Any programmable register or memory location may be modified by operating the switches on the control panel as follows:

- a. Set the COMPUTE switch to IDLE and the INTERNAL switch to PSW1.
- b. Set DATA SWITCHES to hexadecimal address of desired memory location and momentarily set ALTER/STORE switch to ALTER. This sets program status word 1 (PSW1) to the memory location to be modified.

c. Set DATA SWITCHES to hexadecimal value to be stored in selected memory location, and momentarily set ALTER/STORE switch to STORE. This stores the selected value in the location defined by PSW1, after which PSW1 is automatically incremented by one. (The MAP indicator in the PROGRAM STATUS DOUBLEWORD on the control panel indicates that the address defined by PSW1 is virtual when the indicator is set and actual when the indicator is reset.)

d. To store additional hexadecimal values in sequential memory locations, set each new value into the DATA SWITCHES and momentarily set the ALTER/STORE switch to STORE.

DISPLAYING  
MEMORY AT  
THE CONTROL  
PANEL

Any core memory location may be displayed on the DISPLAY indicators by operating the switches on the control panel as follows:

- a. Set the desired memory address into the DATA SWITCHES.
- b. Set the DISPLAY switch to MEM to display the contents of the location specified by the DATA SWITCHES. (A toggle switch located inside the control panel determines whether the memory location selected by the DATA SWITCHES is an actual address or a virtual address.)

In addition to the memory display switch setting, there are settings for the display of any hardware registers (memory locations 0 - 7)

Note

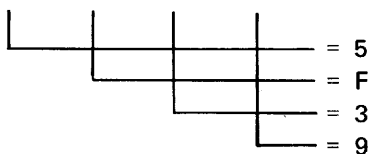
The DISPLAY indicators are updated at a 10-cycle-per-second rate. This allows the user to change the DATA SWITCHES to any memory address as the program is executing and display the contents of the memory location in the DISPLAY indicators.

HEXADECIMAL  
NOTATION

Throughout this manual and related documents, memory address are referenced in hexadecimal notation, which is indicated by the letter X followed by four characters in single quotation marks. Each character corresponds to a four-bit group of basic 16-bit word used by SYSTEMS 72. An example of hexadecimal code and a coded 16-bit word is provided below.

<u>CODE</u>	<u>HEXADECIMAL VALUE</u>	<u>CODE</u>	<u>HEXADECIMAL VALUE</u>
0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F

0101 1111 0011 1001 = X'5F39' in hexadecimal notation



## SECTION II

### CENTRAL PROCESSOR UNIT

#### INTRODUCTION

The Central Processor Unit (CPU) operates under control of an instruction repertoire consisting of 27 basic instructions. (Five optional instructions are reserved for user-selected functions.) Fifteen of the basic instructions are register-expandable, which allows the CPU to operate directly on eight addressable full-word registers. In addition, the CPU allows the user to make use of features such as: condition codes, master/slave operating modes, double indexing, displacement indexing, indirect addressing, and relative addressing forward and backward.

Rapid context switching, in which the CPU changes from the current program environment to a new program environment, assures efficient handling of interrupts and smooth changes in operating mode.

#### CPU OPERATIONAL PHASES

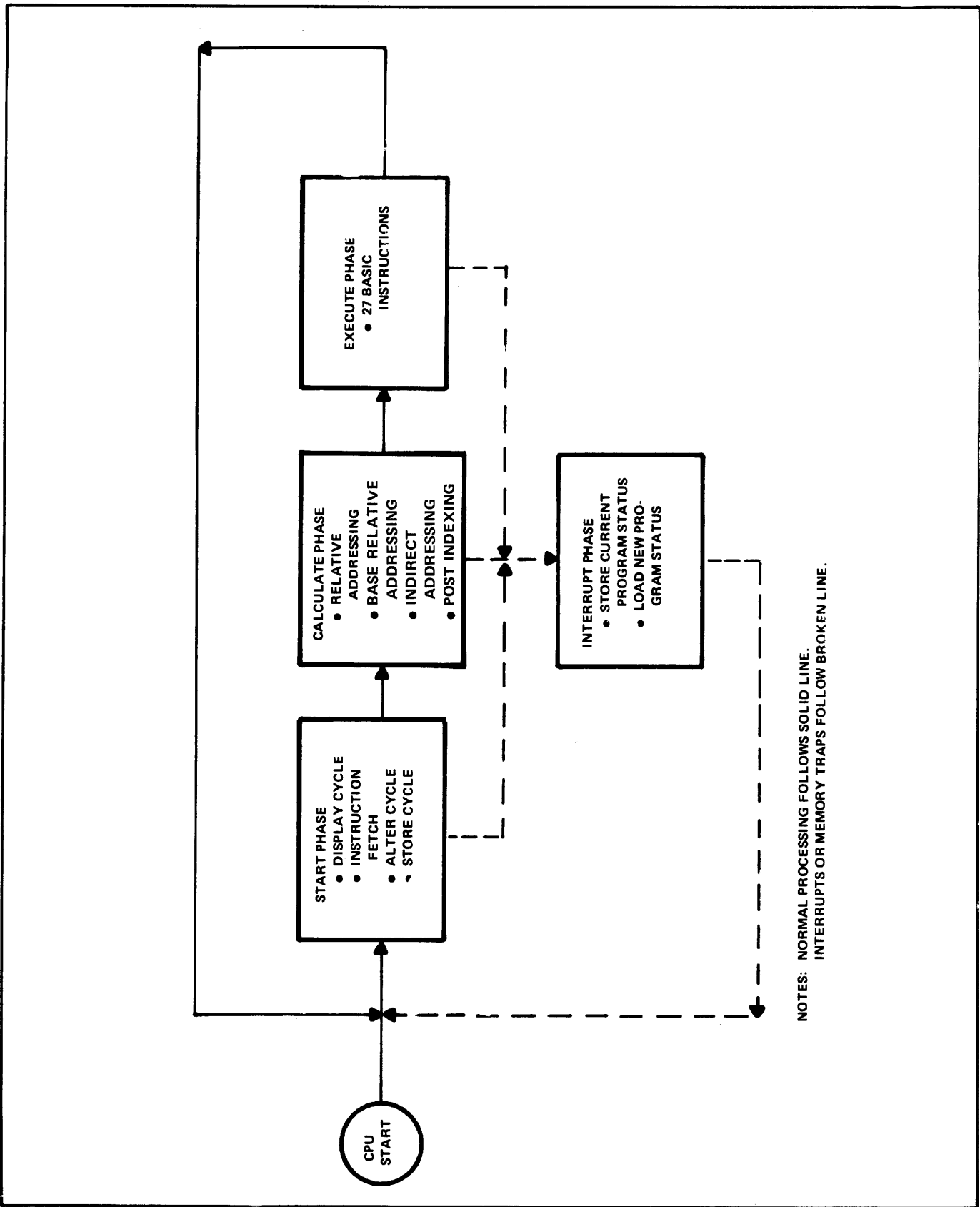
The CPU has four operational phases: start, calculate, execute, and interrupt. These phases correspond to the four INSTRUCTION PHASE indicators located on the control panel. At any given time, the CPU is operating in one of these phases; however, during normal processing more than one indicator may be on due to the speed at which data is being processed.

Figure 2-1 is a general block diagram of the four operational phases. When first initialized, the CPU enters the start phase where it remains until the COMPUTE switch is set to RUN or STEP. This action initiates an instruction fetch cycle, which obtains an instruction from memory, stores the instruction, and causes the CPU to enter the calculate phase.

During the calculate phase, the CPU determines an effective address that is used to define a specific memory location, or provide additional information regarding the operation to be performed. After calculation of the effective address, the CPU enters the execute phase and performs the operation specified in the instruction. When the instruction has been executed, the CPU returns to the start phase; and if the COMPUTE switch is set to RUN, it acquires another instruction. The process is then repeated for the new instruction. (If the COMPUTE switch was initially set to STEP, it must again be set to STEP to execute another instruction.)

The sequence from start, to calculate, to execute, and back to start is the normal processing cycle for the CPU. In the run mode, this cycle will continue until an interrupt is detected, indicating that the CPU is to halt its current operation in order to service the requesting device. A normal interrupt cycle requires the CPU to complete the current instruction, so that the interrupt phase is entered from the execute phase. Two other conditions will also cause the CPU to enter the interrupt phase:

- Receipt of an interrupt request after the CPU has been halted in the start phase.
- Detection of a memory trap by the core/disc memory system. The trap condition causes the CPU to halt its current operation in the start, calculate, or execute phase to process the memory trap.



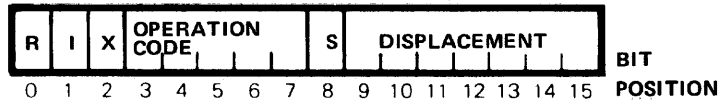
NOTES: NORMAL PROCESSING FOLLOWS SOLID LINE.  
 INTERRUPTS OR MEMORY TRAPS FOLLOW BROKEN LINE.

Figure 2-1. Block Diagram - CPU Operational Phases

Regardless of the conditions under which the CPU enters the interrupt phase, processing of the interrupt is the same. First the CPU stores the current program environment defined by the program status doubleword, which is displayed by the PROGRAM STATUS DOUBLEWORD indicators on the control panel. A program status doubleword associated with the interrupting device is then transferred from memory to the CPU and the CPU returns to the start phase to begin processing the interrupt.

INSTRUCTION  
FORMAT

Processing performed by the CPU is accomplished under control of memory reference instructions, which are in the single-word format indicated below.



The various fields of the memory reference instructions are used to specify the operation to be performed by the CPU and the effective address required during execution of the instruction. Specific functions associated with each field of the instruction are indicated below.

<u>Field</u>	<u>Function</u>
R	<u>Relative Addressing</u> - Specifies that the address is relative to the program count in program status word 1 (PSW1). PSW1 is displayed on the front panel by indicators 0 through 15 of the PROGRAM STATUS DOUBLEWORD.
I	<u>Indirect Addressing</u> - Specifies an indirect reference is to be used in calculating the effective address.
X	<u>Post-Indexing</u> - Specifies calculation of an address relative to the index count stored in the index (X) register. X-Register is displayed by the DISPLAY indicators in conjunction with the DISPLAY switch.
OPERATION	<p><u>Operation Code</u> - Specifies the operation next to be executed by the CPU. The current contents of the instruction (I) register are displayed by the DISPLAY indicators with the DISPLAY switch in the NI position.</p> <p><u>Base-Relative Addressing</u> - Specifies the sign of the displacement when the R-bit is set, indicating relative addressing. Specifies pre-indexing by adding the Base Register when the R-bit is reset.</p>

**DISPLACEMENT**      Address Displacement - Specifies the displacement to be used in calculation of the effective address.

Several instructions executed by the CPU use certain fields for special purposes. For example, conditional branch instructions are incapable of indirect addressing and post-indexing, so for these instructions the I and X fields are used to mask the condition codes. In addition, a branch return and clear instruction does not include post-indexing. In this case, the X field indicates whether the highest active interrupt is to be cleared.

**ADDRESSABLE  
REGISTERS**

SYSTEMS 72 contains eight full-word addressable registers. In configurations without the Model 7212 High-Speed Register option, the registers occupy the first eight core memory locations. In configurations containing the high-speed register option, the addressable registers consist of eight integrated-circuit, flip-flop registers that increase the speed and efficiency of the CPU. In either case, the registers are always addressable as absolute locations X'0000' through X'0007'. (Effective addresses in this range are not mapped.)

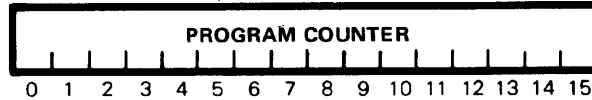
All eight addressable registers have general utility and several perform special functions. A brief summary of each register is provided below:

<u>Register</u>	<u>Name</u>	<u>Address</u>	<u>Special Function</u>
A	Accumulator	X'0001'	Store result of arithmetic, logical, compare, and shift operations.
B	Base Register	X'0004'	Base-Relative Addressing
D	Data Register	X'0000'	Store 16-bit data words output to, or input from, the programmed input/output data bus.
E	Extended Accumulator	X'0002'	Used as the low-order extension of the accumulator (A) register during double register shift operations, and optional Multiply/Divide Instructions
X	Index Register	X'0003'	Post-indexing and looping
R1	Utility Register 1	X'0005'	None
R2	Utility Register 2	X'0006'	None
R3	Utility Register 3	X'0007'	None

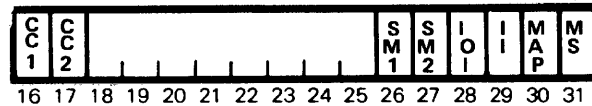
**PROGRAM STATUS  
DOUBLEWORD**

The program status doubleword, which contains the current program environment is displayed on the PROGRAM STATUS DOUBLEWORD indicators on the control panel. The format for the two words is illustrated below:

**Program Status Word 1 (PSW1)**



**Program Status Word 2 (PSW2)**



Since the program status doubleword represents a summary of the current program environment, the CPU is able to transfer control rapidly from one program to another. The transfer process, which is referred to as rapid context switching, increases the efficiency of SYSTEMS 72 in processing interrupts and transferring control between user programs and the operating system.

Rapid context switching occurs during the processing of interrupts, memory traps, or execution of a Call 1 or Call 2 instruction. If one of these events is detected, the current program status doubleword is exchanged for a new doubleword pertaining to the routine being called. After the interrupt has been processed, a branch return and clear instruction is used to return control to the original program status doubleword.

As indicated in the diagram of the program status doubleword, PSW1 contains a program count that indicates the memory address of the next instruction to be executed. PSW1 steps sequentially with every instruction execution, unless redirected by a branch instruction, a call instruction, an interrupt, or a memory trap. Program status word 2 (PSW2) defines the program operating environment through a series of status bits. The purpose of each status bit is discussed below:

<u>Bits</u>	<u>Function</u>
CC1 and CC2	<u>Condition Codes</u> - Indicate the result of an instruction execution or during programmed input/output operations the status of the peripheral device. Instructions that alter the condition codes are defined in table 2-1. Interrupts also alter the condition codes.

Bits

Functions

SM1 and SM2

Compare Sequence Mode - Sequence mode 1 (SM1) places the CPU in the compare sequence mode, in which only the compare instruction is allowed to alter the condition codes. During the compare sequence mode, the result of each comparison combines logically with the current condition code settings as controlled by sequence mode 2 (SM2). If SM2 is set, the bits destined for the condition codes form the logical OR in which logical ONES are developed in those bit positions where either or both inputs are logical ONES. If SM2 is reset, the bits form the logical AND in which logical ONES are developed only in those bit positions where either, but not both, operands contain logical ONES.

IOI

Input/Output Inhibit - IOI blocks all input/output service requests when set. The software-implemented programmed input/output processor (PIOP) normally allows input/output service requests to take priority over interrupt requests, which signal the end of a data transfer. With IOI set, the low priority interrupt requests are allowed, but all input/output service requests are inhibited. (Refer to Section IV for a discussion of the interrupt system.)

II

Interrupt Inhibit - II blocks all interrupt service requests, but not input/output service requests. Blocking the interrupt request prevents an interrupt from becoming active, but does not prevent an armed interrupt from entering the waiting state. (Refer to section IV for a discussion of the interrupt system.)

MAP

Mapped Mode - MAP allows a memory map in the core/disc memory system to translate program addresses in virtual memory to actual addresses in core memory. All programs are usually mapped. (Refer to section III for a discussion of the core/disc memory system.)

MS

Master/Slave - MS prevents execution of privileged instructions while the CPU is in the slave mode (MS reset). The instruction repertoire contains four privileged instructions; PIN, POT, BRC, and CAL1. The operating systems always operate with MS reset; user programs normally do not.



Table 2-1. Condition Code Configurations for Basic Instructions

Instruction	Condition Code Configuration		
POT = 00	Function of Peripheral Device		
PIN = 01	Function of Peripheral Device		
AND = 02	<u>CC1</u>	<u>CC2</u>	
	1	1	A REGISTER < 0
	0	0	A REGISTER = 0
	1	0	A REGISTER > 0
LOR = 03	<u>CC1</u>	<u>CC2</u>	
	1	1	A REGISTER < 0
	0	0	A REGISTER = 0
	1	0	A REGISTER > 0
EOR = 04	<u>CC1</u>	<u>CC2</u>	
	1	1	A REGISTER < 0
	0	0	A REGISTER = 0
	1	0	A REGISTER > 0
LDD = 08	<u>CC1</u>	<u>CC2</u>	
	1	1	D REGISTER < 0
	0	0	D REGISTER = 0
	1	0	D REGISTER > 0
LDA = 09	<u>CC1</u>	<u>CC2</u>	
	1	1	A REGISTER < 0
	0	0	A REGISTER = 0
	1	0	A REGISTER > 0
LBY = 0A	<u>CC1</u>	<u>CC2</u>	
	0	0	A REGISTER = 0
	1	0	A REGISTER > 0
LDX = 0B	<u>CC1</u>	<u>CC2</u>	
	1	1	X REGISTER < 0
	0	0	X REGISTER = 0
	1	0	X REGISTER > 0
LDB = 0C	<u>CC1</u>	<u>CC2</u>	
	1	1	B REGISTER < 0
	0	0	B REGISTER = 0
	1	0	B REGISTER > 0

Table 2-1. Condition Code Configurations for Basic Instructions (Cont'd)

Instruction	Condition Code Configuration		
ADD = 0D	<u>CC1</u>	<u>CC2</u>	
	0	0	NO CARRY - NO OVERFLOW
	0	1	NO CARRY - OVERFLOW
	1	0	CARRY - NO OVERFLOW
SUB = 0E	<u>CC1</u>	<u>CC2</u>	
	0	0	NO CARRY - NO OVERFLOW
	0	1	NO CARRY - OVERFLOW
	1	0	CARRY - NO OVERFLOW
INC = 0F	<u>CC1</u>	<u>CC2</u>	
	0	0	NO CARRY - NO OVERFLOW
	0	1	NO CARRY - OVERFLOW
	1	0	CARRY - NO OVERFLOW
CMP = 10	<u>CC1</u>	<u>CC2</u>	
	1	1	A REGISTER < EFFECTIVE WORD
	0	0	A REGISTER = EFFECTIVE WORD
S = 11	<u>CC1</u>	<u>CC2</u>	
	1	0	A REGISTER > EFFECTIVE WORD
	<u>LEFT SHIFT</u>		
	<u>CC1</u>	<u>CC2</u>	
	0	0	A REGISTER BIT 0 ORIGINALLY = 0 AND HAS RECEIVED ONLY 0's
	0	1	A REGISTER BIT 0 ORIGINALLY = 1 AND HAS RECEIVED AT LEAST ONE 1
	1	0	A REGISTER BIT 0 ORIGINALLY = 1 AND HAS RECEIVED AT LEAST ONE 0
	1	1	A REGISTER BIT 0 WAS 0 AND RECEIVED AT LEAST ONE 1
	<u>RIGHT SHIFT</u>		
	<u>CC1</u>	<u>CC2</u>	
	0	0	A REGISTER BIT 15 AND E REGISTER BIT 15 HAVE RECEIVED ONLY 0's
	0	1	A REGISTER BIT 15 RECEIVED A 0 AT LEAST ONCE AND BIT 15 OF E AT LEAST ONE 1
1	0	A REGISTER BIT 15 RECEIVED A 1 AT LEAST ONCE AND BIT 15 OF E AT LEAST ONE 0	
1	1	A REGISTER BIT 15 AND E REGISTER BIT 15 RECEIVED A 1 AT LEAST ONCE	

ADDRESS  
CALCULATION

The configuration of the R, I, X, and S bits in the instruction is examined in conjunction with the displacement field to compute the effective memory address. The calculation is performed in exactly the same manner, 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 instruction as in a shift instruction. Figure 2-2 illustrates the sequence of the address calculation and figure 2-3 indicates the resulting address.

RELATIVE  
ADDRESSING

The CPU first checks the R-bit to determine if an address relative to the current program count is PSW1 is required. If the R bit is set, indicating relative addressing, the S (sign) bit is checked to determine if the address is forward or backward of the current program count.

Relative Backward - When the S-bit is set, indicating a backward address, the CPU extends the sign bit into bits 0 through 7 of the instruction. Extending the sign is accomplished by providing all ONE's in bits 0 through 7, so that the internal adder develops the TWO's complement of the address. The resultant partial address may be up to 128 locations backward from the current program count in PSW1.

Relative Forward - When the S-bit is reset, indicating a forward address, calculation of the partial address takes place in basically the same manner as a relative backward address, except that the sign is not extended into bits 0 through 7 of the instruction. In this case, all ZERO's are provided in bits 0 through 7, so that the internal adder merely adds the displacement to the program count. The resultant partial address may be up to 127 locations forward of the current program count in PSW1.

BASE-RELATIVE  
ADDRESSING

If the R-bit is reset when checked by the CPU, the S-bit is examined to determine if pre-indexing (base addressing) is required. In this case, the CPU reads the contents of the B (base) register, and adds this value to the current program count. The resultant partial address may be up to 127 locations forward of the count in the B-register.

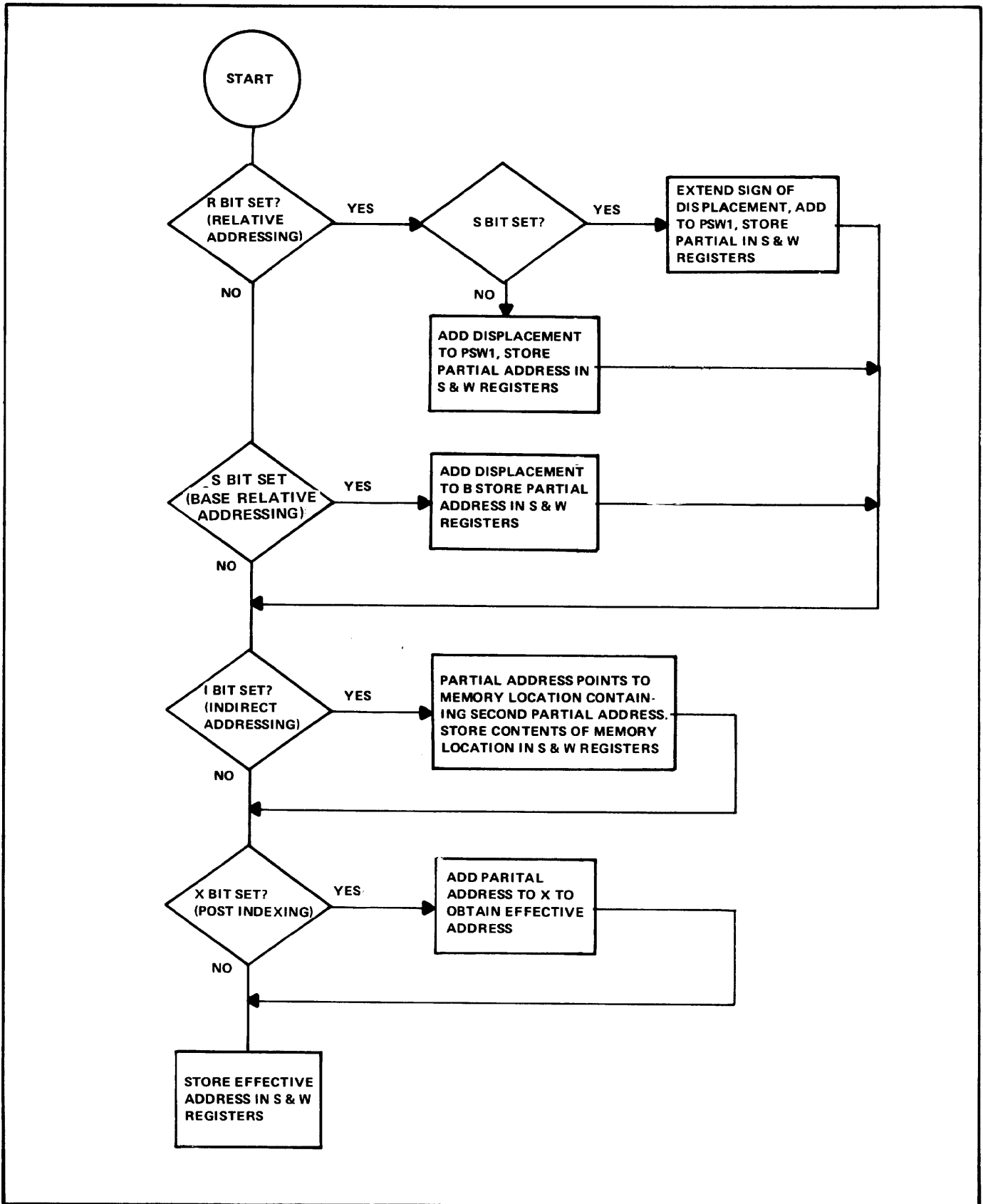


Figure 2-2. Sequence Diagram - Effective Address Calculation

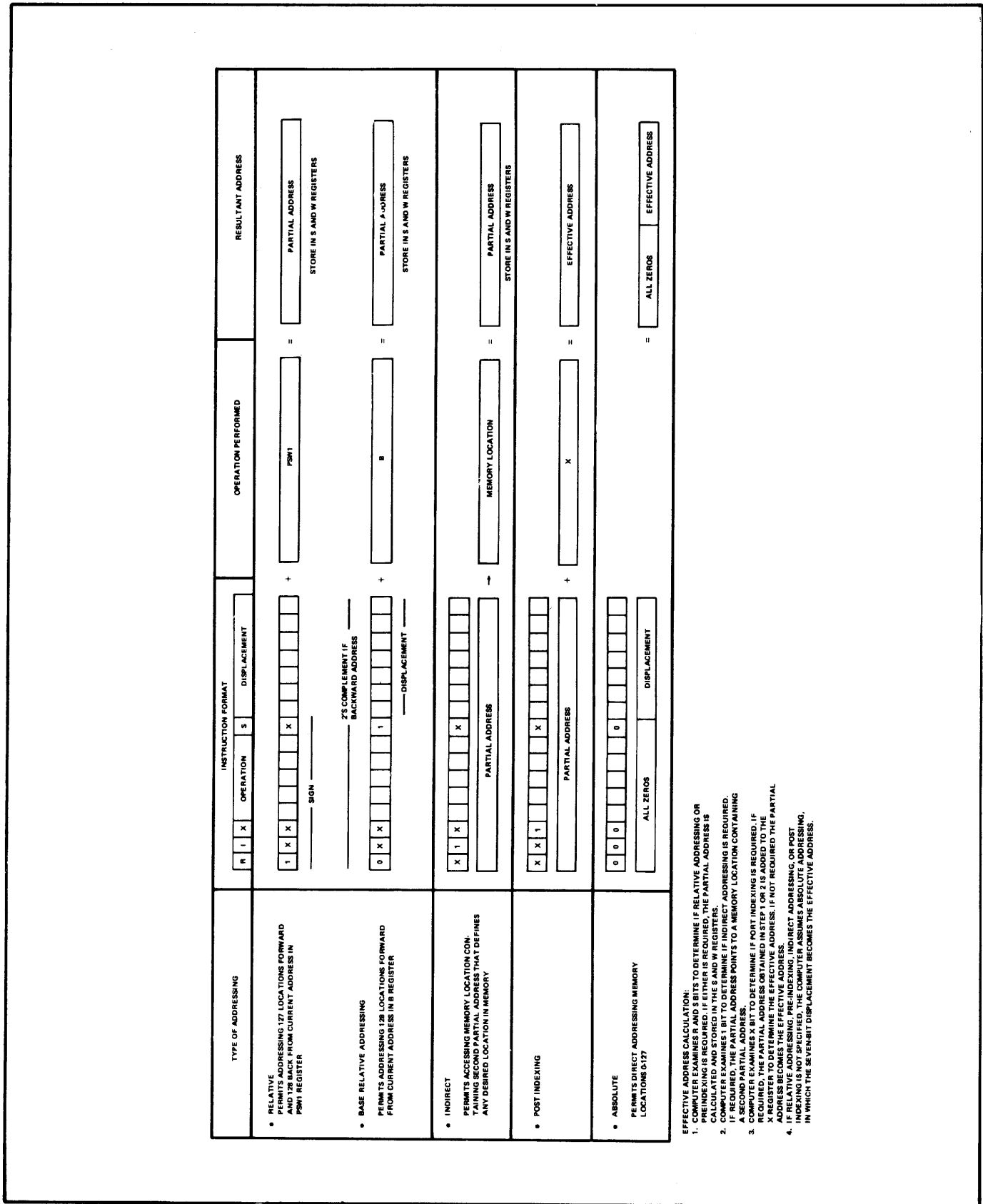


Figure 2-3. Calculation Diagram - Effective Address

### Indirect Addressing

After the R and S bits have been checked, the CPU examines the I bit to determine if indirect addressing is required. If the I bit is set, the partial address calculated for relative addressing or pre-indexing is used to address a memory location containing a new partial address. The new partial address, which is read from memory by the CPU, is capable of accessing any location in memory.

### Post-Indexing

After any indirect address calculations have been completed, the CPU examines the X-bit to determine if post-indexing is required. If the X-bit is set, the partial address calculated for relative addressing, pre-indexing, and/or indirect addressing is used in calculation of the final effective address. In this case, the CPU reads the contents of the X (index) register and adds this value to the previously calculated partial address. The resultant effective address is capable of accessing any location in memory.

### Absolute Addressing

If the R, I, X, and S bits of the instruction are all reset, the CPU assumes absolute addressing. In this case, the displacement field of the instruction is used to address the desired memory location. Absolute addressing permits accessing any memory location between X'0000' and X'007F', which represents the maximum value of the displacement field.

### INSTRUCTION REPertoire

The five-bit operation code field of the CPU instruction yields 27 basic instructions and five optional instructions. The basic instructions, which are listed by the hexadecimal value of the operation code field in table 2-2, consist of the following types of instructions: Branch, Load, Store, Arithmetic, Logical, Shift, Call, Input/Output, and Other. Table 2-2 also indicates the active steps of each instruction, which are displayed by the INSTRUCTION STEP indicators on the control panel as the instruction is executing. The five optional instructions, for which the hexadecimal value of the operation code field is X'1B' through X'1F', are selected by the user to implement optional circuits, such as high-speed multiply/divide. Any undefined operation code will execute as a No Operation.

Many of the basic instructions used by SYSTEMS 72 are register expandable, which allows the CPU to operate on all eight addressable registers. Other basic instructions also provide derivatives that are recognized by the assembler. The ability of the assembler to recognize a wide variety of derivative instructions reduces the time required to program the SYSTEMS 72. All derivative instructions are listed in appendix B.

Execution timing for the various instructions depends on whether the Model 7212 High-Speed Register option is installed. This option significantly increases the speed and efficiency of the CPU. In configurations with the high-speed registers, timing also depends on whether the operand is in core or the appropriate register, with faster execution occurring when the operand is in the appropriate register.

TABLE 2-2. CPU BASIC INSTRUCTIONS

OPERATION CODE	PHASE/INSTRUCTION MNEMONIC	PHASE/INSTRUCTION DEFINITION	ACTIVE INSTRUCTION STEPS															
			0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
N/A	SP	START PHASE	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	
N/A	CALC	CALCULATE PHASE	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	
N/A	INT	INTERRUPT PHASE	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	
N/A	EXU	EXECUTE PHASE	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	
00	POT	PROGRAMMED OUTPUT	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	
01	PIN	PROGRAMMED INPUT	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	
02	AND	AND MEMORY INTO A	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	
03	LOR	OR MEMORY INTO A	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	
04	EOR	EXCLUSIVE OR MEMORY INTO A	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	
05	STD	STORED D REGISTER	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	
06	STA	STORED A REGISTER	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	
07	SBY	STORED BYTE FROM A	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	
08	LDD	LOAD D REGISTER	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	
09	LDA	LOAD A REGISTER	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	
0A	LBY	LOAD BYTE INTO A	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	
0B	LDX	LOAD X REGISTER	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	
0C	LDB	LOAD B REGISTER	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	
0D	ADD	ADD MEMORY TO A	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	
0E	SUB	SUBTRACT MEMORY FROM A	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	
0F	INC	INCREMENT MEMORY	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	
10	CMP	COMPARE A WITH MEMORY	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	
11	S	SHIFT	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	
12	B	BRANCH	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	
13	BAL	BRANCH AND LINK	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	
14	BIX	BRANCH AND INCREMENT INDEX	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	
15	BCR	BRANCH ON CONDITIONS RESET	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	
16	BCS	BRANCH ON CONDITIONS SET	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	
17	BRC	BRANCH RETURN AND CLEAR	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	
18	CAL1	CALL 1	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	
19	CAL2	CALL 2	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	
1A	CAL3	CALL 3	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	

***BRANCH  
INSTRUCTIONS***

The branch instructions exercise program control by forcing the program count in PSW1 out of its usual sequence, setting up linkages to subroutines, counting the iterations through a loop, testing the condition codes, and restoring the program status doubleword. The ability of the branch instructions to exercise program control makes the use of these instructions appropriate immediately after executing one of the other types of instructions.

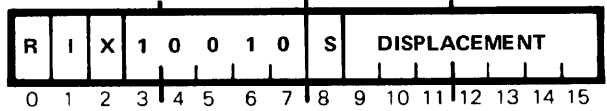
Branch instructions do not affect the condition codes.

Basic branch instructions are: B, BAL, BIX, BCR, BCS, and BRC.



**B** Branch

1200



Affected: PSW1

*DEFINITION*

The B instruction causes an unconditional branch, in which the effective address replaces the program count in PSW1.

*EXAMPLE*

Memory Location:	1000
Hex Instruction:	9250

*BEFORE EXECUTION*

PSW1	PSW2
1000	4002

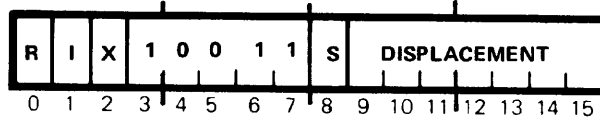
*AFTER EXECUTION*

PSW1	PSW2
1050	4002

# BAL

## BAL Branch and Link

1300



Affected: PSW1  
B-Register

### DEFINITION

The address immediately following the current address replaces the contents of the B-register, and the effective address replaces the program count in PSW1. (Pre-indexing is allowed since the effective address is calculated before the contents of the B register are altered.)

BAL is normally used to branch to a subroutine, which can then use pre-indexing to pass arguments and set up the return address. By storing the linking address in the B-register instead of the effective location, BAL permits branching to reentrant subroutines and subroutines with multiple entry points.

### EXAMPLE

Memory Location: 1010  
Hex Instruction: D350

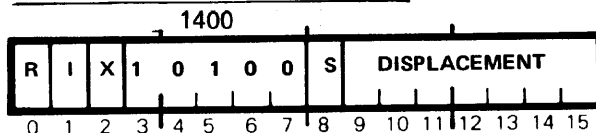
### BEFORE EXECUTION

PSW1	PSW2	B	(1060)
1010	4002	035C	1400

### AFTER EXECUTION

PSW1	PSW2	B	(1060)
1400	4002	1011	1400

BIX Branch and Increment Index



Affected: PSW1  
X-Register

*DEFINITION*

The contents of the X-register are incremented by one. If the contents of the X-register do not equal zero, the effective address replaces the program count in PSW1. If the contents of the X-register are equal to zero, program control passes to the address 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.)

*EXAMPLE*

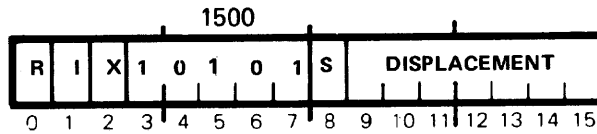
Memory Location: 1420  
Hex Instruction: 94FC

*BEFORE EXECUTION*

PSW1	PSW2	X
1420	2002	FFF9

*AFTER EXECUTION*

PSW1	PSW2	X
141C	2002	FFFA

BCR Branch on Conditions Reset

Affected: PSW 1

*DEFINITION*

The effective address replaces PSW1 in all cases except those in which a condition code indicator and the associated mask bit are both set. The effective address calculation does not include indirect addressing or post-indexing, so that the I field is used to mask CC2. All configurations of the condition codes and mask bits are indicated below, with the letter B marking every combination that will cause branching. If the requirements for branching are not satisfied, the program count in PSW1 is advanced to the next sequential location.

Condition Codes		Mask Bits			
		IX	IX	IX	IX
CC1	CC2	00	01	10	11
0	0	B	B	B	B
0	1	B	--	B	--
1	0	B	B	--	--
1	1	B	--	--	--

The assembler recognizes the following derivatives of BCR:

BEZ	Branch if Equal to Zero
BGEZ	Branch if Greater Than or Equal to Zero
BE	Branch if Equal
BGE	Branch if Greater Than or Equal
BNC	Branch on No Carry
BNO	Branch on No Overflow

*EXAMPLE*

Memory Location: 1020  
Hex Instruction: D507

*BEFORE EXECUTION*

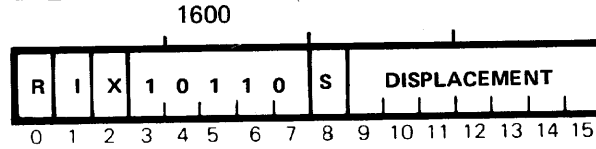
PSW1	PSW2
1020	4002

*AFTER EXECUTION*

PSW1	PSW2
1027	4002

Note: Derivatives of this instruction are given in the Appendix B.

BCS Branch on Conditions Set



Affected: PSW1

**DEFINITION**

The effective address replaces PSW1 if a condition code indicator and the associated mask bit are both set. The effective address calculation does not include indirect addressing or post-indexing, so that the I field of the instruction is used to mask CC1 and the X field is used to mask CC2. All configurations of the condition code and mask bits are indicated below, with the letters B marking every combination that will cause branching. If the requirements for branching are not satisfied, the program count in PSW1 is advanced to the next sequential location.

Condition Codes		Mask Bits			
		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

The assembler recognizes the following derivatives of BCS:

- BNEZ Branch if Not Equal to Zero
- BLZ Branch if Less Than Zero
- BNE Branch if Not Equal
- BL Branch if Less Than
- BC Branch on Carry
- BO Branch on Overflow

**EXAMPLE**

Memory Location: 1090  
Hex Instruction: D606

**BEFORE EXECUTION**

PSW1 PSW2  
1090 C002

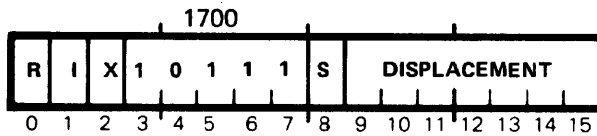
**AFTER EXECUTION**

PSW1 PSW2  
1096 C002

Note: Derivatives of this instruction are given in Appendix B.

# BRC

## BRC Branch Return and Clear



Affected: PSW1  
 PSW2  
 Highest Active Interrupt

### DEFINITION

The contents of the effective address and the following location replace the program status doubleword. Calculation of the effective address does not include post-indexing, so that 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 CPU is in the slave mode.

Call 1 and Call 2 instructions, as well as interrupts and memory traps, store the program status doubleword before branching to a service routine. The new PSW2 normally puts the CPU in the master mode, which allows these routines to use BRC upon returning to the main program. (Returns from the call instructions are normally accomplished with the X field reset.)

### EXAMPLE

Memory Location: 023B  
 Hex Instruction: 5742

### BEFORE EXECUTION

PSW1	PSW2	(1100)	(1101)	(42)
023B	0001	105C	0002	1100

### AFTER EXECUTION

PSW1	PSW2	(1100)	(1101)	(42)
105C	0002	105C	0002	1100

Note: This instruction is used to return from a memory Trap.

## LOAD INSTRUCTIONS

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

Load instructions configure the condition codes as follows:

<u>CC1</u>	<u>CC2</u>	<u>Condition</u>
1	1	Value loaded into receiving register less than zero
0	0	Value loaded into receiving register equal to zero
1	0	Value loaded into receiving register greater than zero

### Note

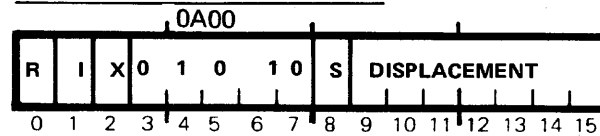
Bits 0 through 7 are always cleared for a load byte instruction, so that the receiving register can never be less than zero.

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

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 equal to zero
BLZ	Branch if contents of receiving register are less than zero

Basic load instructions are: LBY, LDA, LDB, LDD, and LDX.

**LBY Load Byte into A-Register**



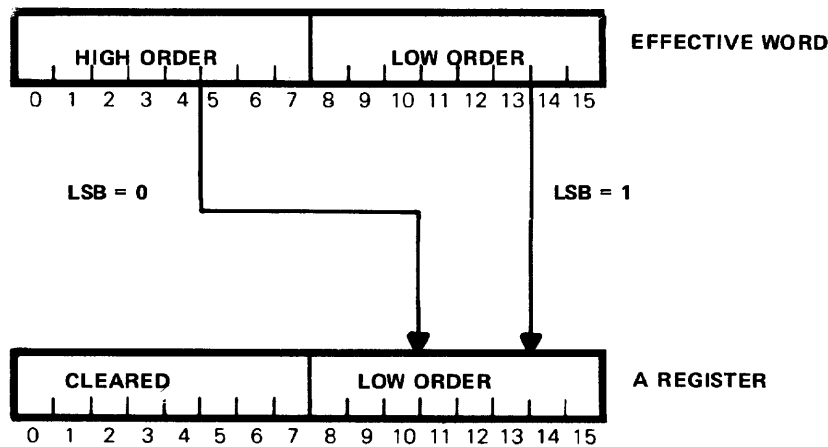
Affected: A-Register  
 CC1  
 CC2 Always reset

**DEFINITION**

The high-order byte (byte 0), bits 0 through 7, or the low-order byte (byte 1), bits 8 through 15, of the effective word is loaded into the low-order byte of the A-register. The high-order byte of the A-register is cleared.

For effective address calculations that do not specify post-indexing, the effective byte is always present in bits 0 through 7 of the effective word. The maximum forward or backward reference is 32K bytes on byte instructions.

For effective address calculations using post-indexing, the CPU develops a 17-bit effective address, in which the 16 most-significant bits represent the address of the effective word, and the least significant bit indicates whether the effective byte is present in bits 0 through 7 or bits 8 through 15 of the effective word. The control exercised by the least significant bit is indicated below:



The assembler recognizes the following derivatives of LBY:

- LBYB Load Byte into A from B
- LBYD Load Byte into A from D
- LBYE Load Byte into A from E
- LBYX Load Byte into A from X
- LBY1 Load Byte into A from R1
- LBY2 Load Byte into A from R2
- LBY3 Load Byte into A from R3



*EXAMPLE 1*

Memory Location: 101A  
Hex Instruction: 8A3A

*BEFORE EXECUTION*

PSW1	PSW2	(1054)	A
101A	8002	3031	38B2

*AFTER EXECUTION*

PSW1	PSW2	(1054)	A
101B	8002	3031	0030

*EXAMPLE 2*

Memory Location: 101B  
Hex Instruction: AA38

*BEFORE EXECUTION*

PSW1	PSW2	(1054)	A	X
101B	8002	3031	0030	0005

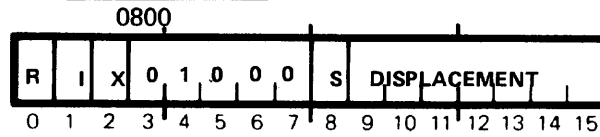
*AFTER EXECUTION*

PSW1	PSW2	(1054)	A	X
101C	8002	3031	0031	0005

Note: Derivatives of this instruction are given in Appendix B.

# LDA

## LDA Load A-Register



Affected: A-Register  
CC1  
CC2

### DEFINITION

The effective word is loaded into the A-register. The condition codes are configured as determined by the value of the effective word.

The assembler recognizes the following derivatives of LDA:

LDAB	Load A from B
LDAD	Load A from D
LDAE	Load A from E
LDAX	Load A from X
LDA1	Load A from R1
LDA2	Load A from R2
LDA3	Load A from R3

### EXAMPLE

Memory Location: 1008  
Hex Instruction: 8949

### BEFORE EXECUTION

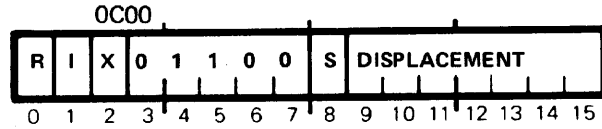
PSW1	PSW2	(1051)	A
1008	8002	7FFF	FFFF

### AFTER EXECUTION

PSW1	PSW2	(1051)	A
1009	8002	7FFF	7FFF

Note: Derivatives of this instruction are given in Appendix B.

LDB Load B-Register



Affected: B-Register  
 CC1  
 CC2

*DEFINITION*

The effective word is loaded into the B-register. The condition codes are configured as determined by the value of the effective word.

The assembler recognizes the following derivatives of LDB:

- LDBA Load B from A
- LDBD Load B from D
- LDBE Load B from E
- LDBX Load B from X
- LDB1 Load B from R1
- LDB2 Load B from R2
- LDB3 Load B from R3

*EXAMPLE*

Memory Location: 1019  
 Hex Instruction: 8C3A

*BEFORE EXECUTION*

PSW1	PSW2	(1053)	B
1009	8002	0001	0300

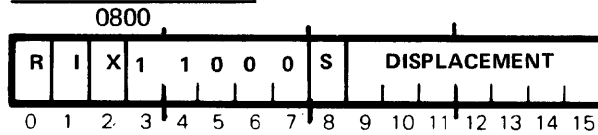
*AFTER EXECUTION*

PSW1	PSW2	(1053)	B
101A	8002	0001	0001

Note: Derivatives of this instruction are given in Appendix B.

# LDD

## LDD Load D-Register



Affected: D Register  
 CC1  
 CC2

### DEFINITION

The effective word is loaded into the D-register. The condition codes are configured as determined by the value of the effective word.

The assembler recognizes the following derivatives of LDD:

LDDA	Load D from A
Lddb	Load D from B
LDDE	Load D from E
LDDX	Load D from X
LDD1	Load D from R1
LDD2	Load D from R2
LDD3	Load D from R3

### EXAMPLE

Memory Location: 1001  
 Hex Instruction: 0801

#### BEFORE EXECUTION

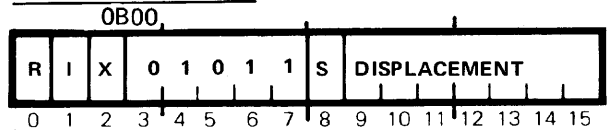
PSW1	PSW2	(1050)	D
1000	8002	8000	4C00

#### AFTER EXECUTION

PSW1	PSW2	(1050)	D
1001	C002	8000	8000

Note: Derivatives of this instruction are given in Appendix B.

LDX Load X-Register



Affected: X Register  
 CC1  
 CC2

*DEFINITION*

The effective word is loaded into the X-register. The condition codes are configured as determined by the value of the effective word.

The assembler recognizes the following derivatives of LDX:

- LDXA Load X from A
- LDXB Load X from B
- LDXD Load X from D
- LDXE Load X from E
- LDX1 Load X from R1
- LDX2 Load X from R2
- LDX3 Load X from R3

*EXAMPLE*

Memory Location: 1011  
 Hex Instruction: 8B41

*BEFORE EXECUTION*

PSW1	PSW2	(1052)	X
1011	8002	FFFF	FFFC

*AFTER EXECUTION*

PSW1	PSW2	(1052)	X
1012	C002		

Note: Derivatives of this instruction are given in Appendix B.

***STORE  
INSTRUCTIONS***

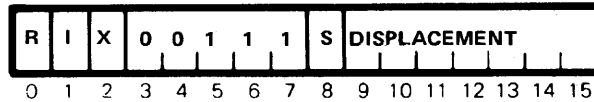
The A and D-registers are both capable of storing their contents in any addressable register or memory location. In addition, the A-register can also store the low-order byte, bits 8 through 15, into the high-order or low-order byte of any addressable register or memory location.

Store instructions do not affect the condition codes.

Basic store instructions are: SBY, STA, and STD.

SBY Store Byte from A-Register

0700



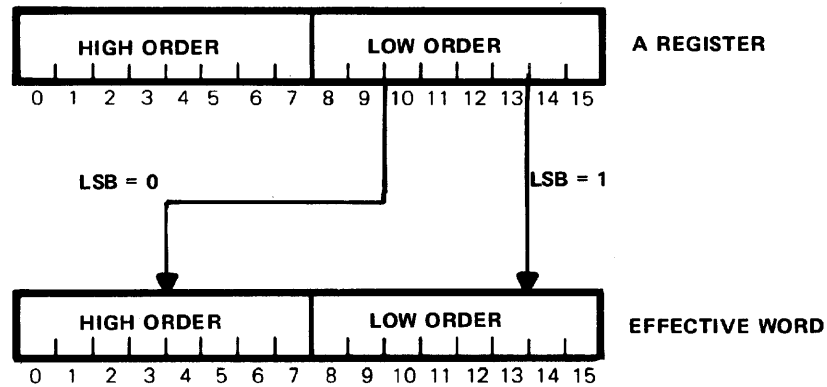
Affected: Effective Byte

DEFINITION

The low-order byte of the A-register, bits 8 through 15, replaces the low-order or high-order byte of the effective word. The high-order byte, bits 0 through 7, of the A-register is not affected.

For effective address calculations that do not specify post-indexing, the effective byte is always bits 0 through 7 of the effective word. As with the load instruction, the SBY can only address 32K bytes forward or backward.

For effective address calculations using post-indexing, the CPU develops a 17-bit effective address, in which the 16 most-significant bits represent the address of the effective word, and the least-significant bit indicates whether the effective byte is bits 0 through 7 or bits 8 through 15 of the effective word. The control exercised by the least-significant bit is indicated below:



The assembler recognizes the following derivatives of SBY;

- SBYB Store Byte from A into B
- SBYD Store Byte from A into D
- SBYE Store Byte from A into E
- SBYX Store Byte from A into X
- SBY1 Store Byte from A into R1
- SBY2 Store Byte from A into R2
- SBY3 Store Byte from A into R3

EXAMPLE

Memory Location: 101E  
Hex Instruction: A737

BEFORE EXECUTION

PSW1	PSW2	(1053)	A	X
101E	8002	0001	0031	0004

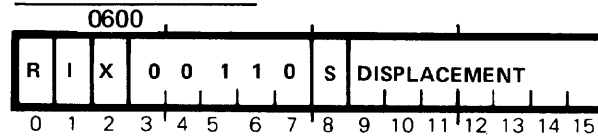
AFTER EXECUTION

PSW1	PSW2	(1053)	A	X
101F	8002	3101	0031	0004

Note: Derivatives of this instruction are given in Appendix B.

# STA

STA Store A-Register



Affected: Effective Word

## DEFINITION

The contents of the A-register replace the effective word.

The assembler recognizes the following derivatives of STA:

STAB	Store A into B
STAD	Store A into D
STAE	Store A into E
STAX	Store A into X
STA1	Store A into R1
STA2	Store A into R2
STA3	Store A into R3

## EXAMPLE

Memory Location: 101D  
Hex Instruction: 8639

## BEFORE EXECUTION

PSW1	PSW2	(1056)	A
101D	8002	FFFF	0031

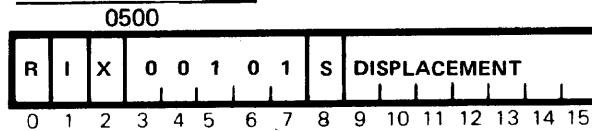
## AFTER EXECUTION

PSW1	PSW2	(1056)	A
101E	8002	0031	0031

Note: Derivatives of this instruction are given in Appendix B.



STD Store D-Register



Affected: Effective Word

*DEFINITION*

The contents of the D-register replace the effective word.

The assembler recognizes the following derivatives of STD:

- STDA Store D into A
- STDB Store D into B
- STDE Store D into E
- STDX Store D into X
- STD1 Store D into R1
- STD2 Store D into R2
- STD3 Store D into R3

*EXAMPLE*

Memory Location: 101C  
Hex Instruction: 8539

*BEFORE EXECUTION*

PSW1	PSW2	(1055)	D
101C	8002	7FFF	4C00

*AFTER EXECUTION*

PSW1	PSW2	(1055)	D
101D	8002	4C00	4C00

Note: Derivatives of this instruction are given in Appendix B.

**ARITHMETIC  
INSTRUCTIONS**

Arithmetic instructions are capable of operating on the contents of any addressable register or memory location. If a carry or an overflow occurs during the execution of an arithmetic instruction, the condition codes are configured as follows:

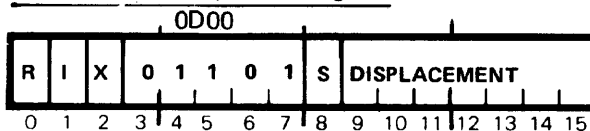
<u>CC1</u>	<u>CC2</u>	<u>Condition</u>
0	0	No Carry - No Overflow
0	1	No Carry - Overflow
1	0	Carry - No Overflow
1	1	Carry - Overflow

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

BO	Branch on Overflow
BNO	Branch on No Overflow
BC	Branch on Carry
BNC	Branch on No Carry

Basic arithmetic instructions are: ADD, INC, and SUB.

**ADD** Add Memory to A-Register



Affected: A-Register  
 CC1  
 CC2

**DEFINITION**

The effective word is added to the contents of the A-register after which the result is stored in the A-register.

The assembler recognizes the following derivatives of ADD:

- ADDB Add B to A
- ADDD Add D to A
- ADDE Add E to A
- ADDX Add X to A
- ADD1 Add R1 to A
- ADD2 Add R2 to A
- ADD3 Add R3 to A

**EXAMPLE**

Memory Location: 101F  
 Hex Instruction: 8D31

**BEFORE EXECUTION**

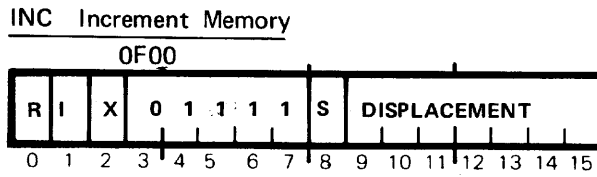
PSW1	PSW2	(1050)	A
101F	8002	8000	0031

**AFTER EXECUTION**

PSW1	PSW2	(1050)	A
1020	0002	8000	8031

Note: Derivatives of this instruction given in Appendix B.

# INC



Affected: Effective Word  
CC1  
CC2

The effective word is incremented by one.

## DEFINITION

The assembler recognizes the following derivatives of INC:

INCA	Increment A
INCB	Increment B
INCD	Increment D
INCE	Increment E
INCX	Increment X
INC1	Increment R1
INC2	Increment R2
INC3	Increment R3

## EXAMPLE

Memory Location: 1022  
Hex Instruction: 8F30

## BEFORE EXECUTION

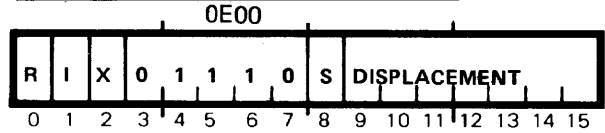
PSW1	PSW2	(1052)
1022	C002	0031

## AFTER EXECUTION

PSW1	PSW2	(1052)
1023	0002	0032

Note: Derivatives of this instruction are given in Appendix B.

SUB Subtract Memory from A-Register



Affected: A Register  
 CC1  
 CC2

*DEFINITION*

The effective word is subtracted from the contents of the A-register, after which the result is stored in the A-register. Sub = 0 always resets CC1.

The assembler recognizes the following derivatives of SUB:

- SUBB Subtract B from A
- SUBD Subtract D from A
- SUBE Subtract E from A
- SUBX Subtract X from A
- SUB1 Subtract R1 from A
- SUB2 Subtract R2 from A
- SUB3 Subtract R3 from A

*EXAMPLE*

Memory Location: 1021  
 Hex Instruction: 8E30

*BEFORE EXECUTION*

PSW1	PSW2	(1051)	A
1021	0002	4C00	8031

*AFTER EXECUTION*

PSW1	PSW2	(1051)	A
1022	C002	4C00	3430

Note: Derivatives of this instruction are given in Appendix B.

*LOGICAL  
INSTRUCTION*

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 addressable register or memory location. Logical product, sum, and difference are defined as follows:

Logical Product	Execution develops logical ONES only in those bit positions in which both operands contain ONES.
Logical Sum	Execution develops logical ONES in those bit positions in which either or both operands contain ONES.
Logical Difference	Execution develops logical ONES only in those bit positions in which either, but not both, operands contain ONES.

Logical instructions configure the condition codes as follows:

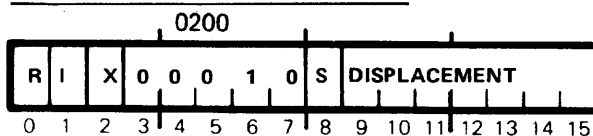
<u>CC1</u>	<u>CC2</u>	<u>Condition</u>
1	1	Result stored in A-register less than zero
0	0	Result stored in A-register equal to zero
1	0	Result stored in A-register greater than zero

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

BEZ	Branch if result in A-register is equal to zero
BNEZ	Branch if result in A-register is not equal to zero
BGEZ	Branch if result in A-register is greater than or equal to zero
BLZ	Branch if result in A-register is less than zero

Basic logical instructions are: AND, EOR, and LOR.

AND AND Memory into A-Register



Affected: A-Register  
 CC1  
 CC2

*DEFINITION*

The effective word is combined with the contents of the A-register to develop a logical product that is stored in the A-register.

The assembler recognizes the following derivatives of AND:

- ANDB AND B into A
- ANDD AND D into A
- ANDE AND E into A
- ANDX AND X into A
- AND1 AND R1 into A
- AND2 AND R2 into A
- AND3 AND R3 into A

*EXAMPLE*

Memory Location: 1023  
 Hex Instruction: 8230

*BEFORE EXECUTION*

PSW1	PSW2	(1053)	A
1023	0002	3101	3430

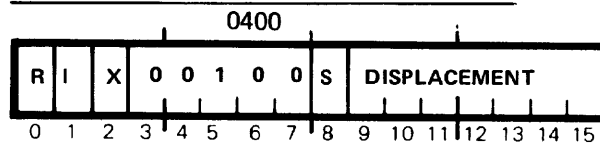
*AFTER EXECUTION*

PSW1	PSW2	(1054)	A
1024	8002	3101	3000

Note: Derivatives of this instruction are given in Appendix B.

# EOR

## EOR Exclusive OR Memory into A-Register



Affected: A Register  
CC1  
CC2

### DEFINITION

The effective word is combined with the contents of the A-register to develop a logical difference that is stored in the A-register.

The assembler recognizes the following derivatives of EOR:

EORB	Exclusive OR B into A
EORD	Exclusive OR D into A
EORE	Exclusive OR E into A
EORX	Exclusive OR X into A
EOR1	Exclusive OR R1 into A
EOR2	Exclusive OR R2 into A
EOR3	Exclusive OR R3 into A

### EXAMPLE

Memory Location: 1025  
Hex Instruction: 0407

### BEFORE EXECUTION

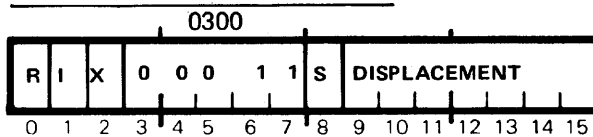
PSW1	PSW2	R3	A
1025	8002	38B2	3031

### AFTER EXECUTION

PSW1	PSW2	R3	A
1026	8002	38B2	0883



## LOR OR Memory into A-Register



Affected: A-Register  
 CC1  
 CC2

*DEFINITION*

The effective word is combined with the contents of the A-register to develop a logical sum that is stored in the A-register.

The assembler recognizes the following derivatives of LOR:

LORB	OR B into A
LORD	OR D into A
LORE	OR E into A
LORX	OR X into A
LOR1	OR R1 into A
LOR2	OR R2 into A
LOR3	OR R3 into A

*EXAMPLE*

Memory Location: 1024  
 Hex Instruction: 8330

*BEFORE EXECUTION*

PSW1	PSW2	(1054)	A
1024	8002	3031	3000

*AFTER EXECUTION*

PSW1	PSW2	(1054)	A
1025	8002	3031	3031

Note: Derivatives of this instruction are given in Appendix B.

**COMPARE  
INSTRUCTION**

The compare instruction determines whether the contents of the A-register are less than, equal to, or greater than an addressable register or memory location. The instruction treats both operands as signed integers in the two's complement format and leaves the operands unchanged.

The compare instruction configures the condition codes as follows:

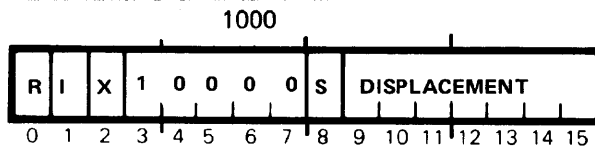
<u>CC1</u>	<u>CC2</u>	<u>Condition</u>
1	1	Contents of A-register less than other operand
0	0	Contents of A-register equal to other operand
1	0	Contents of A-register greater than other operand

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

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

Setting SM1 in the program status doubleword places the CPU in the compare sequence mode, in which only the compare instruction is allowed to alter the condition codes. During the compare sequence mode, the result of each comparison combines logically with the current condition code settings as controlled by SM2 in the program status doubleword. If SM2 is set, the bits destined for the condition codes form the logical sum, in which logical ONES are developed in those bit positions where either or both inputs are logical ONES. If SM2 is reset, the bits destined for the condition codes form the logical product, in which logical ONES are developed only in those bit positions where either, but not both, operands contain ONES.

CMP Compare A-Register With Memory



Affected: CC1  
          CC2

*DEFINITION*

The effective word is compared with the contents of the A-register and the condition codes are set to indicate the results of the comparison.

The assembler recognizes the following derivatives of CMP:

- CMPB    Compare A with B
- CMPD    Compare A with D
- CMPE    Compare A with E
- CMPX    Compare A with X
- CMP1    Compare A with R1
- CMP2    Compare A with R2
- CMP3    Compare A with R3

*EXAMPLE*

Memory Location:    1026  
Hex Instruction:     902C

*BEFORE EXECUTION*

PSW1	PSW2	(1052)	A
1026	8002	0031	0883

*AFTER EXECUTION*

PSW1	PSW2	(1052)	A
1027	8002	0031	0883

Note: Derivatives of this instruction are given in Appendix B.

*SHIFT  
INSTRUCTIONS*

A single shift instruction is used to develop a full complement of shift operations, including the following:

Single Register - The A-register is used for both right and left shift operations.

Double Register - The A and E-registers for a single 32-bit register that is used for both right and left shift operations. The A-register contains the most-significant 16 bits and the E-register contains the least-significant 16 bits.

Logical - Logical left shifts introduce zeros into the least-significant bit position and lose bits from the most-significant bit position. Logical right shifts introduce ZERO bits into the most-significant bit position and lose bits from the least-significant bit position..

Note

In double register logical shifts, the least-significant position is bit 15 of the E-register and the most-significant position is bit 0 of the A-register.

Arithmetic - Arithmetic left shifts introduce zeros into the least-significant bit position and lose bits from bit 1 of the A-register. Arithmetic right shifts propagate the contents of bit position 0 of the A-register and lose bits from the least-significant bit position

Note

In double register arithmetic shifts, the least-significant position is bit 15 of the E-register.

Circular - Circular shifts take bits leaving one end of the register and introduce them into the other end.

Note

In double register circular shifts, the two ends of the combined 32-bit register are bit 0 of the A-register and bit 15 of the E-register.

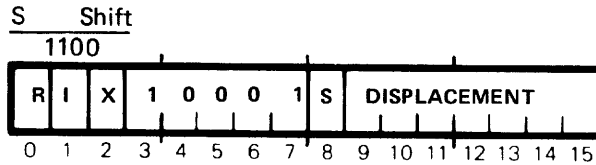
Left and right shift instructions configure the condition codes as indicated below. Note that for left shift instructions, setting CC1 indicates a change in the sign bit and setting CC2 indicates that a ONE was shifted into the sign bit. For right shifts, note that setting CC1 indicates a ONE was shifted into the least significant bit of the A-register and setting CC2 indicates a ONE was shifted into the least significant bit of the E-register.

Left Shift:

<u>CC1</u>	<u>CC2</u>	<u>Condition</u>
0	0	Bit 0 of A-register was ZERO and received all ZERO'S
0	1	Bit 0 of A-register was ONE and received at least one ONE
1	0	Bit 0 of A-register was ONE and received at least one ZERO'S
1	1	Bit 0 of A-register was ZERO and received at least one ONE

Right Shift:

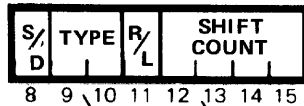
<u>CC1</u>	<u>CC2</u>	<u>Condition</u>
0	0	Bit 15 of A-received all ZEROS and bit 15 of E received all ZERO'S
0	1	Bit 15 of A-received at least one ZERO, bit 15 of E received at least one ONE
1	0	Bit 15 of A-received at least one ONE, bit 15 of E received at least one ZERO
1	1	Bit 15 of A-received at least one ONE, bit 15 of E received at least one ONE



Affected: A-Register (Single and Double Register Shifts)  
 E-Register (Double Register Shifts)  
 CC1  
 CC2

**DEFINITION**

The CPU uses bits 8 through 15 of the effective address to define the following parameters associated with the shift: single/double, arithmetic/logical/circular, right/left, and shift count. Bits 8 through 15 of the effective address and the significance of the various fields are indicated below:



- 12-15: Coded representation of number of places to shift from 1 to 16, where 0000 = 16, 0001 = 1,.....1111 = 15
- 11: Right or left, where 0 = right and 1 = left
- 9-10: Circular, logical, or arithmetic shift, where 00 = Not allowed, 01 = Circular, 10 = Logical, and 11 = Arithmetic
- 8: Single or double, where 0 = single and 1 = double

Note

In a single register shift, the R, I, X, and S-bits are normally reset, so that the entire operation is specified by the displacement field of the instruction. A double register shift must develop the effective address through indirect addressing or indexing.

The assembler recognizes the following derivatives of S:

SLL	Shift Logical Left
SLR	Shift Logical Right
SAL	Shift Arithmetic Left
SAR	Shift Arithmetic Right
SCL	Shift Circular Left
SCR	Shift Circular Right
SLLD	Shift Logical Left Double
SLRD	Shift Logical Right Double
SALD	Shift Arithmetic Left Double
SARD	Shift Arithmetic Right Double
SCLD	Shift Circular Left Double
SCRD	Shift Circular Right Double

*EXAMPLE 1*

Type of Shift: SLR  
 Memory Location: 1027  
 Hex Instruction: 1145

*BEFORE EXECUTION*

PSW1	PSW2	A
1027	8002	0883

*AFTER EXECUTION*

PSW1	PSW2	A
1028	8002	0044

*EXAMPLE 2*

Type of Shift: SAL  
 Memory Location: 1028  
 Hex Instruction: 1177

*BEFORE EXECUTION*

PSW1	PSW2	A
1028	8002	0044

*AFTER EXECUTION*

PSW1	PSW2	A
1029	0002	2200

Note: Derivatives of this instruction are given in Appendix B.

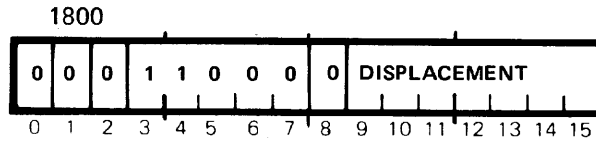
## *CALL INSTRUCTIONS*

Call instructions perform two major functions: (1) rapid context switching, in which the current program environment contained in the program status doubleword is stored and a new doubleword is accessed to initiate a new operation, and (2) subroutine linking, in which the user accesses various subroutines as required.

Call instructions ignore the R, I, X, and S fields in calculating their effective addresses. The call instructions do not configure the condition codes.

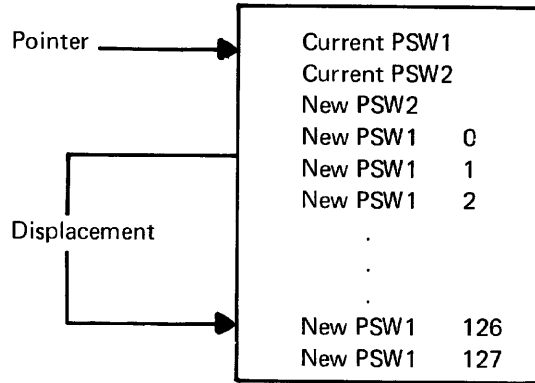


CAL1 Call 1



Affected: PSW1  
PSW2

Call 1 operates only when the CPU is operating in the master mode, in which the MS bit of PSW2 is reset, and always switches program context, so that the instruction is normally used for operating system calls.



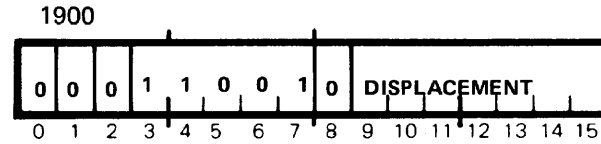
The contents of X'0045' contain a pointer address that is used to address the first location of a 131-word table. As indicated, the current PSW1 and PSW2 are stored in the first two locations of the table. The contents of the third location are used as PSW2 of the new program status doubleword. The effective address is used as a displacement into the remaining 128 locations, so that the contents of the designated location are used as PSW1 of the new program status doubleword.

CAL1 is a privileged instruction, which functions as a No Operation when the CPU is in the slave mode. The instruction also assumes that the MAP bit in PSW2 is reset, so that X'0045' and the contents of X'0045' are treated as actual addresses.

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'.

## CAL2

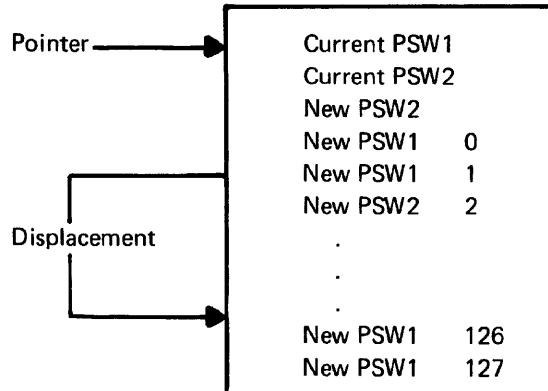
### CAL2 Call2



Affected: PSW1  
PSW2

Call 2 always switches program context but, unlike Call 1, it is not a privileged instruction. The instruction is normally used for user program calls to the operating system.

The contents of X'0046' contain a pointer address that is used to address the first location of a 131-word table. As indicated below, the current PSW1 and PSW2 are stored in the first two locations of the table. The contents of the third location are used as PSW2 of the new program status doubleword. The effective address is used as a displacement into the remaining 128 locations, so that the contents of the designated location are used as PSW1 of the new program status doubleword.



To prevent storage of improper values for PSW1 and PSW2 during the CALL 1 sequence, a BOUND directive must be used to ensure that all entries in the call table are totally contained on one page, 256 words of memory. An example of the sequence required to build the call table is indicated.

	BOUND	256			
CAL2S	DATA	0	OLD	PSW1	
	DATA	0	OLD	PSW2	
	DATA	PSW2	NEW	PSW2	
	DATA	NPSW10	CAL2	DISPLACEMENT	0
	DATA	NPSW11	CAL2	DISPLACEMENT	1
	.	.	.	.	.
	.	.	.	.	.
	.	.	.	.	.
	DATA	NPSW17F	CAL2	DISPLACEMENT	127

Only as much of the table as is used must be on the bound page, so that if only the first 20 Call 2 entries are used, then only 20 + 3 = 23 entries are placed on the bound page.

Note

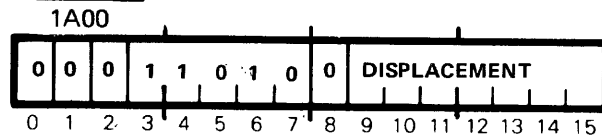
If the user does not develop Call 2 tables, no BOUND directives are required, since the operating system handles the generation of both Call 1 and Call 2 tables.

CAL2 is not a privileged instruction. In addition, the instruction treats X'0046' and the contents of X'0046' as virtual addresses only if the MAP bit in PSW2 is set.

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

# CAL3

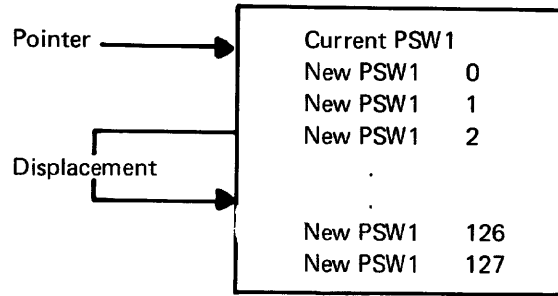
## CAL3 Call 3



Affected: PSW1

Call 3 does not switch program context and is not a privileged instruction. The instruction is normally used to link to subroutines within the user program.

The contents of X'0047' contain a pointer address that is used to address the first location of a 129-word table. As indicated below, the current PSW1 is stored in the first location and the effective address is used as a displacement into the remaining 128 locations. The contents of the designated location are used as PSW1 of the new program status doubleword.



CAL3 does not switch program context and is not a privileged instruction. The instruction treats X'0047' and the contents of X'0047' as virtual addresses only if the MAP bit in PSW2 is set.

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

## *INPUT/OUTPUT INSTRUCTIONS*

The input/output instructions operate on all devices connected to the Programmed Input/Output (PIO) bus. This bus, which consists of 16 address lines, 16 data lines, and six control lines is connected to the following equipment.

- All standard peripheral I/O devices
- Optional Direct Access Channel (DAC)
- Basic system devices, such as the control panel, PSW2 register, disc controller, and memory map

There are two basic input/output instructions: POT and PIN. In both instructions, the effective address specifies the device and the operation to be performed.

### Note

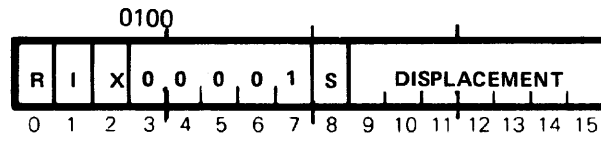
Instructions defined in section IV pertain to standard peripheral equipment and have dedicated effective addresses. All effective addresses with an F in the most significant hexadecimal digit, for example X'F847', are reserved for instructions pertaining to special user-oriented equipment.

The configuration of the condition codes for a given input/output instruction are normally a function of the result of executing the instruction. Condition code indications for the various peripheral devices are provided in the appropriate manuals.

All input/output instructions are privileged, so that while the CPU is operating in the slave mode, the instructions function as No operations.

# PIN

## PIN Programmed Input



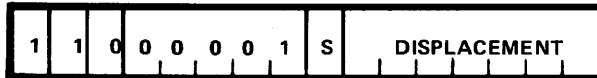
Affected: D-Register  
CC1  
CC2

The device address is output by way of the 16 effective address lines, and the addressed device inputs data to the D-register by way of 16 data lines.

The assembler recognizes the following derivative of PIN:

# TDV

## TDV Test Device



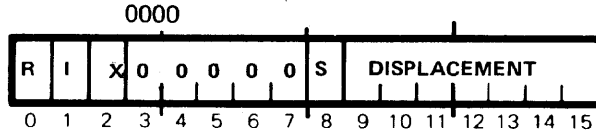
## Effective Address \*

0000 0000 11AA AAAA

\* A = six-bit device address

Note: Special derivatives of this instruction are provided in Appendix I.

POT Programmed Output



Affected: CC1  
CC2

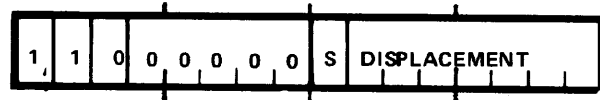
The device address/operation is output by way of the 16 effective address lines, and the contents of the D register are output to the addressed device by way of the 16 data lines.

The assembler recognizes the following derivatives of POT:

SIO Start Input/Output

Effective Address \*

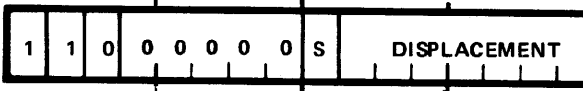
SIO



0000 0001 00AA AAAA

TIO Test Input/Output

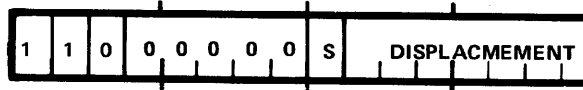
TIO



0000 0001 10AA AAAA

HIO Halt Input/Output

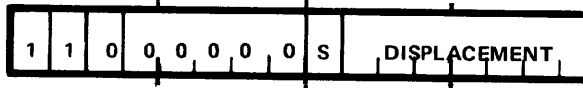
HIO



0000 0001 01AA AAAA

IOR Input/Output Reset

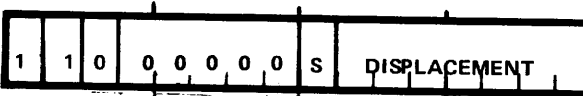
IOR



0000 0100 0000 0011

HLT Program Halt

HLT



0000 0100 0000 0001

\* A = six-bit device address

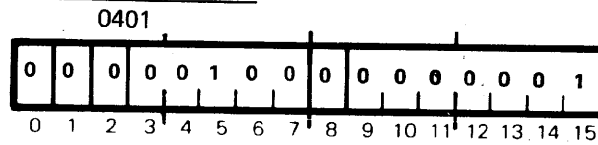
Note: Special derivatives of this instruction are provided in Appendix I.

*OTHER INSTRUCTIONS*

The assembler recognizes several derivative instructions that have general utility:

**CLA**

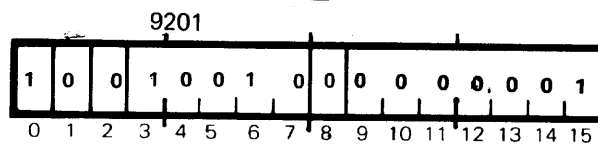
CLA Clear A-Register



Affected: A-Register  
CC1  
CC2

All 16 bit positions of the A-register, and condition codes 1 and 2, are reset.

NOP No Operation



Affected: None

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



## SECTION III

### CORE/DISC MEMORY SYSTEM

#### INTRODUCTION

The core/disc memory system, using a technique known as memory mapping, creates an apparent or virtual memory that equals the capacity of the disc. In the basic configuration of SYSTEMS 72, the virtual programmable memory capacity is 32,768 sixteen-bit words. This capacity is doubled to 65,536 words in configurations using the Model 7232 Memory Map Extension in conjunction with one of the following disc memories:

- Basic 65K Word Disc Memory
- Model 7234 65K-Word Disc Memory
- Model 7235 131K-Word Disc Memory
- Model 7236 262K-Word Disc Memory
- Model 7237 512-K Word Disc Memory

Core memory supplied with the basic configuration of the SYSTEMS 72 consists of a single module with a capacity of 4096 sixteen-bit words; however, since virtual memory is equal to the 32,768-word capacity because of the disc, a larger core is required only if the time-critical portion of a program will not fit into the core memory supplied with the basic configuration. In the event additional core is required, the SYSTEMS 72 permits expansion in 4096-word increments (to 65,536 words) by installing Model 7230 4K Core Memory Modules in a Model 7231 Memory Expansion Chassis.

#### BASIC CORE/ DISC MEMORY SYSTEM

The core/disc memory system supplied with the basic configuration of the SYSTEMS 72 consists of the following items:

- Core Memory Modules
- Disc Memory
- Memory Map (MAP) Circuit Card - Card slot 5
- Memory Access Controller (MAC) Circuit Card - Card Slot 6
- Memory Extension Disc (MED) Circuit Card - Card slot 7

The 4096-word capacity of the Core Memory module is functionally divided into sixteen 256-word pages. Any page may be stored, under control of the core/disc memory system, on a specified sector of the disc.

The 65,536-word capacity of the Model 7016 Disc Memory is functionally divided into 32 tracks, with eight sectors per track and 256 words per sector. By equating 256-word core memory pages with 256-word disc memory sectors, the core/disc memory system is able to transfer 256-word blocks of program information from core to disc or from disc to core.

#### MEMORY SYSTEM FUNCTIONS

The core/disc memory system performs four major functions during program execution:

- Reading data from core memory
- Writing data into core memory
- Transferring data from the core memory to the disc memory
- Transferring data from the disc memory to the core memory

A memory read operation is initiated when the execution of an instruction requires that information must be read from the core memory. The read operation transfers information from the addressed core location to the requesting device.

A memory write operation is initiated when the execution of an instruction requires that information must be written into the core memory. The write operation transfers information from the requesting device to the addressed core location.

Core-to-disc and disc-to-core transfers are initiated when the program requires information not currently resident in core. The transfers, which take place under control of a small core-resident program always transfer one 256-word page at a time.

A core-to-disc transfer is always performed first to clear an area in core memory for the information required by the program. To accomplish the transfer, standard software selects a seldom-used page that is currently core-resident and initiates the transfer to the disc.

A disc-to-Core transfer is initiated after the core-to-disc transfer has cleared an area for the required program information. After the required program information is core-resident, program execution is allowed to continue.

Therefore, during program execution the core/disc memory system is continuously reading from and writing into the core memory until information required by the program is not available in core. At this time, a core-to-disc transfer is initiated to transfer a seldom used page to the disc memory. This transfer is followed by a disc-to-core transfer, which provides the information required to continue the program.

### CORE MEMORY

Although the basic configuration of SYSTEMS 72 contains only one 4096-word core memory module, system design permits easy expansion to sixteen 4096-word modules, which represents a core memory capacity of 65,536 words. Each 4096-word module has independent address and data registers, which permit an early release during a memory read or write operation. The early release feature allows SYSTEMS 72 to overlap successive memory accesses to different core memory modules and thereby increases the efficiency of the CPU. A full memory cycle requires 880 nanoseconds.

### DEDICATED CORE LOCATIONS

Dedicated core locations range from X'0000' to X'01FF'. These locations reserve core area for the eight addressable registers and various interrupts. In the basic configuration of SYSTEMS 72, the addressable registers are assigned the first eight locations in core X'0000' through X'0007'. In configurations with the Model 7212 High Speed Register option, the addressable registers are assigned the same addresses; however, the registers are located in integrated circuit registers external to the core memory. A complete list of dedicated core locations is provided below:

<u>Decimal Address</u>	<u>Hexadecimal Address</u>	<u>Reserved For</u>
0	X'0000'	D-Register
1	X'0001'	A-Register
2	X'0002'	E-Register
3	X'0003'	X-Register
4	X'0004'	B-Register
5	X'0005'	R1-Register
6	X'0006'	R2-Register
7	X'0007'	R3-Register
8-63	X'0008' - X'003F'	Input/Output Service Interrupts
64	X'0040'	Power-Off Interrupt
65	X'0041'	Power-On Interrupt
66	X'0042'	Memory Access Controller Trap

<u>Decimal Address</u>	<u>Hexadecimal Address</u>	<u>Reserved For</u>
67	X'0043'	Memory Extension Disc Interrupt
68	X'0044'	Direct Access Channel Interrupt
69	X'0045'	Call 1
70*	X'0046'	Call 2
71*	X'0047'	Call 3
72*	X'0048'	Real-Time Clock 1
73*	X'0049'	Real-Time Clock 2
74*	X'004A'	Real-Time Clock 3
75*	X'004B'	Real-Time Clock 4
76*	X'004C'	Console Interrupt
128-511	X'0080' - X'01FF'	System Interrupts **

\* May address actual or virtual memory.

\*\* Only as many locations as needed are reserved.

#### *DATA GUARD*

Data Guard is a standard feature that assures the contents of the core memory will remain undisturbed during system shutdown, including power failure, and system restart. Data Guard prevents memory read or write operations during the power on/off sequence by holding the memory data bus at ground. Additional protection is provided by the Power Fail-Safe standard feature.

#### *MEMORY PARITY CHECK*

The Model 7210 Memory Parity option generates even parity on each core memory write operation and checks the parity on each read operation. A parity error results if even parity is not detected during the read operation. The parity error causes the core/disc memory system to generate a memory trap. The trap aborts the current operation, so that an error in the memory system does not affect the instruction register or one of the addressable registers.

Installation of the Model 7210 Memory Parity option lengthens memory read operations by 250 nanoseconds.

#### *MEMORY PORTS*

There are three independent ports to core memory. These are listed below in order of priority:

Port A - Disc Memory

Port B - Direct Access Channel (DAC) or the Multiplexed Input/Output Processor (MIOP) Option

Port C - Central Processor Unit (CPU)

Each port provides 16 address lines and 16 data lines, which are switched into the memory in response to memory read or write requests from one of the requesting devices. The Memory Access Controller (MAC) resolves simultaneous core/disc requests by granting access to memory in order of priority.

Each request indicates whether the access is to be a read or a write and the mapped/unmapped status of the effective address. This allows the disc, DAC, and CPU to exercise independent control over the status of the effective address. (Effective addresses accessing virtual memory are always mapped, while effective addresses accessing actual memory are never mapped. In addition, the MAP bit in PSW2 controls mapped/unmapped status only when the CPU port is selected.)

## DISC MEMORY

Programs written from SYSTEMS 72 virtual memory are stored on the 65,536-word disc. The programs may occupy any combination of 256-word pages, but normally occupy contiguous pages in program sequence.

The 65,536-word disc capacity of the basic SYSTEMS 72 may be expanded to 131,072 words or 262,144 words through the installation of one of the discs listed below. Through installation of the Model 7232 Memory Map Extender, the programmable memory capacity may be expanded to 65,536 words, which represents the limit of the 16-bit effective address; however, any of the discs listed below may store programs to the limit of its capacity.

- Model 7235 131K-Word Disc Memory
- Model 7236 262K-Word Disc Memory

The basic 65,536-word disc has 32 tracks, with eight sectors/track and 256 words/sector. Expansion simply increases the number of tracks on the same side of a single disc.

All discs are fixed-head with a head-per-track, so that the average access time is 16.67 milliseconds, which represents one-half the rotational period of the disc.

The disc controller generates odd parity while writing a sector on the disc and checks for odd parity while reading from the disc.

## ACTUAL/VIRTUAL MEMORY CONFIGURATION

Programs developed for SYSTEMS 72 are normally written for virtual memory, which in the basic configuration is equivalent to the capacity of the disc. As indicated in figure 3-1, the 32,768-word virtual memory capacity of the basic system is derived from a disc containing 32 tracks, with eight sectors (pages) per track, and 256 words per page.

To access a word in virtual memory, the page containing the word must be resident in actual memory. In the basic configuration of SYSTEMS 72 actual memory consists of a single core memory bank, which contains sixteen 256-word pages or 4096 words of total capacity.

In accessing a given memory location, the requesting device first specifies whether the effective memory address is mapped or unmapped. If the address is unmapped, the actual core address is accessed and the memory read or write operation is initiated; however, if the address is mapped, the address must be converted from a location in virtual memory to the corresponding location in actual memory.

Conversion from a virtual address to an address in actual memory is accomplished through the use of a memory map. The sequence of events required to perform a virtual-to-actual conversion depends upon whether the word required by the program is resident on a virtual page that is currently resident in core. If the page is core-resident, the memory map simply converts the virtual page address to the actual page address and the memory system initiates the appropriate read or write operation. If the virtual page is not core-resident, the memory system, under control of standard software initiates core-to-disc and disc-to-core transfers to transfer the desired page to core. Once the virtual page is core-resident, the memory map converts the virtual page address to the actual page address and the memory system initiates the appropriate read or write operation.

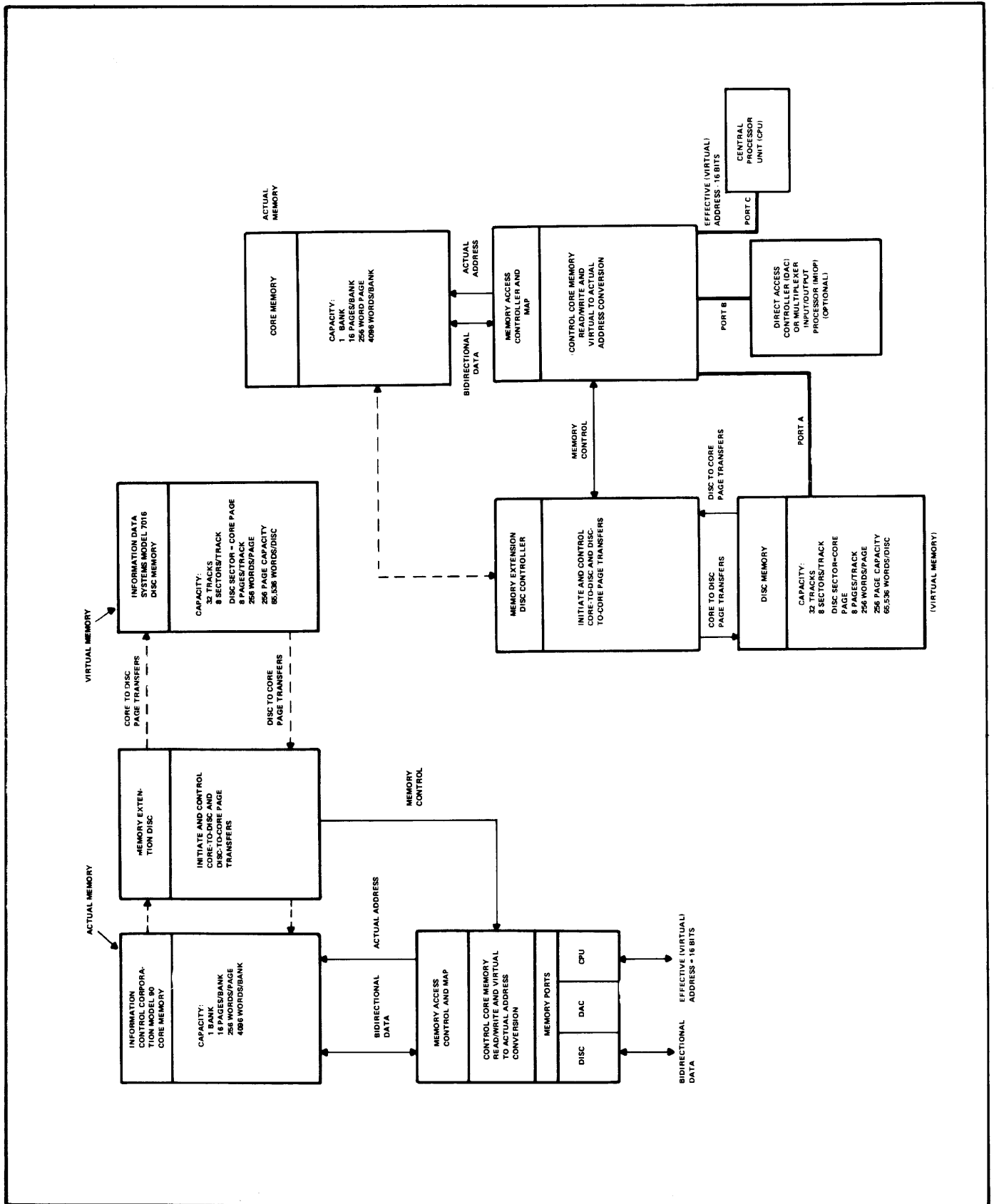
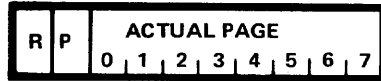


Figure 3-1. Actual/Virtual Memory Configuration (Basic System)

**MEMORY MAP  
FORMAT**

In the basic configuration of SYSTEMS 72, the memory map consists of 128 twelve-bit integrated-circuit registers; one for each 256-word page of virtual memory (32,768 words 256 words/page). Each register uses the format shown below to indicate the status of the associated virtual page:



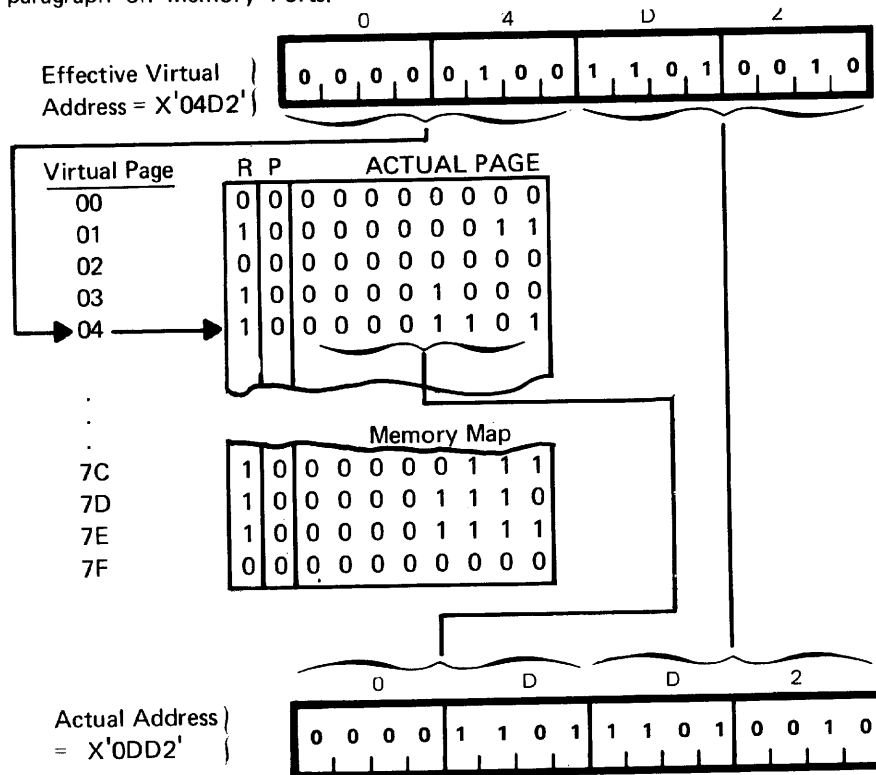
- R                      Page Resident Bit- Indicates whether virtual page is core resident, with 0 = non-resident and 1 = resident.
  
- P                      Memory Protect Bit - Indicates whether virtual page is write protected (from the slave mode user) during CPU memory access, with 0 = unprotected and 1 = protected.
  
- ACTUAL PAGE        Actual Core Location - Defines actual core location of virtual page when R-bit is set, indicating that page is core resident
  
- N                      Not used

**MEMORY MAP  
OPERATION**

Assuming that the virtual page is core resident when a memory read or write operation is requested, the virtual-to-actual conversion is accomplished by examining the eight most-significant bits of the effective address. These bits define one of the 128 or 256 map registers, which contain the actual core page address in the ACTUAL PAGE field. The eight bits removed from the ACTUAL PAGE field define the actual page location of the desired data, while the eight least significant bits of the effective address define a specific word on the page. (The eight least significant bits of the effective address remain intact, since both virtual and actual pages always contain 256 words.)

In the example indicated, a mapped effective address of X'04D2' is translated into an actual address of X'0DD2'. The first two digits of the address, X'04', indicate that the eight most-significant bits of the actual address are stored in map register 004. Note that the R and P bits of map register 004 indicate that the page is core resident and is not write protected. Any attempt to access a non-resident page results in a memory trap, which causes the core-to-disc and dis-to-core page transfers required to transfer the addressed virtual page into core. Attempts to write into protected locations also result in memory traps. Error indications associated with the memory trap are traps. Error indications associated with the memory trap are discussed in the paragraph on Memory Ports.

locations also result in memory traps. Error indications associated with the memory trap are traps. Error indications associated with the memory trap are discussed in the paragraph on Memory Ports.



Virtual memory may be expanded from 32,768 words to 65,536 words through installation of a larger memory map. In this case, the optional Model 7232 Map Extender must be used in conjunction with one of the following discs:

- Model 7235 131K-Word Disc Assembly
- Model 7236 262K-Word Disc Assembly

#### MEMORY MAP UPDATE

The memory map is updated each time a virtual page changes status. To modify the contents of the appropriate map register, the R, P, and ACTUAL PAGE fields are loaded into the 10 least-significant bits of the D-Register and a POT instruction is executed with an effective address, of X'OFRR'. The eight least-significant bits of the effective address, RR, define the map register to be updated by the 10 least-significant bits of the D-Register.

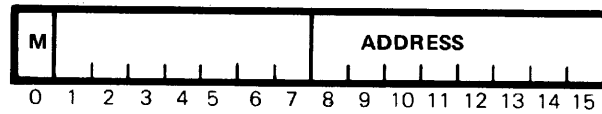
#### MEMORY TRAPS

The operations listed, if attempted by the core/disc memory system, will result in a memory trap:

- An attempt to access a page beyond the range of the memory map
- An attempt to access a non-resident page
- An attempt to write into a protected page
- Incorrect parity during a read operation, providing the Model 7210 Memory Parity option is installed.

Detection of a memory trap while the CPU is connected for service causes the core/disc memory system to interrupt to actual location X'0042' and terminates the memory access. (If the CPU is operating in the master mode with the MS bit of PSW2 reset, the CPU may write into a protected location without causing a memory trap.)

A PIN instruction may be executed with an effective address of X'0F\*\*'. This instruction transfers the contents of a snapshot register in the core/disc memory logic to the D-register in the CPU. The format of the data transferred to the D-register is indicated below. Until unloaded by the PIN instruction, the snapshot register will not accept a new configuration.



**M** Mapped/Unmapped Status - Indicates whether the effective address received by the core/disc memory system is mapped or unmapped, with 0 = unmapped and 1 = mapped

**ADDRESS** Eight Most-Significant Bits of Effective Address  
Defines the eight most-significant bits of the effective address received by the core/disc memory system.

The condition codes resulting from this PIN instruction are as follows:

<u>CC1</u>	<u>CC2</u>	<u>Condition</u>
0	0	Core Memory Parity Error (Optional)
0	1	Write PROTECT Violation
1	0	Page Non-Resident
1	1	Page Out of Range

*Sense Disc  
Rotational  
Position*

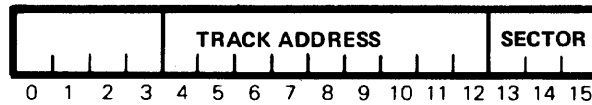
The rotational position may be sensed by executing a PIN instruction with an effective address of X'0E05'. The PIN returns the sector position in the three least-significant bit positions of the D-register and configures the conditions to indicate the appropriate quarter-sector. The configuration of the three least-significant bits of the D-register and the condition are indicated below:

<u>D-Register</u>			<u>Disc Sector</u>	<u>Condition</u>		<u>Quarter Sector</u>
<u>13</u>	<u>14</u>	<u>15</u>		<u>CC1</u>	<u>CC2</u>	
0	0	0	0	0	0	First
0	0	1	1	0	1	Second
0	1	0	2	1	0	Third
0	1	1	3	1	1	Fourth
1	0	0	4			
1	0	1	5			
1	1	0	6			
1	1	1	7			



*Output Disc  
Track And  
Sector Address*

To set up the track and sector address, the D-Register is loaded as indicated below:

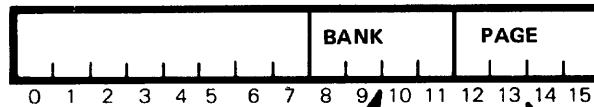


A POT instruction with an effective address of X'0E04', then transfers the disc track and sector to the core/disc memory system for storage.

If a transfer is currently in progress when the POT instruction is executed, the busy condition indicated by setting CC1 and the POT is rejected.

*Output Core  
Bank And  
Page Address*

After storing the disc track and sector address, a POT instruction may load the core bank and page address into bit positions 8 through 15 of the D-register for subsequent transfer to the core/disc memory system. The configuration of the D-register is indicated below:



Defines 1 of up to 16  
4096-word memory banks  
(A' single memory module  
is provided with the basic  
configuration of SYSTEMS 72.)

Defines 1 of 16 pages in  
selected bank

After loading the D-register, a POT instruction may be executed with an effective address from among the following:

<u>Effective Address</u>	<u>Direction and Mapped/Unmapped Status</u>
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 indicated, the effective address defines the direction of the page transfer, and the mapped/unmapped status of the address.

If a transfer is currently in progress when the POT instruction is executed, the busy condition is indicated by setting CC1 and the POT is rejected.

Once started, the page transfer continues to completion unless halted by the program.

*Page  
Transfer  
Terminator*

A page transfer is normally terminated upon completion of the transfer. In this case, the program interrupts to location X'0043'.

The transfer may be halted prematurely by the disc controller if an error is detected during the transfer. In this case, CC2 is set to indicate an incomplete transfer resulting from an error condition.

To determine the origin of the error, the program may execute a PIN instruction with an effective address of X'0E04'. The PIN transfers the error condition to bits 12 through 15 of the D-register, which are configured as follows:

<u>D-Register</u>				<u>Error Condition</u>
<u>12</u>	<u>13</u>	<u>14</u>	<u>15</u>	
1	0	0	0	Non-Resident Page or No Memory
0	1	0	0	Protect Violation (Not CPU)
0	0	1	0	Core Memory Parity Error (Optional)
0	0	0	1	Disc Memory Parity Error

A page transfer may also be halted by executing a POT instruction with an effective address of X'0E05'. Execution of this POT results in an immediate halt of the page transfer and sets CC1 to indicate that a page transfer was in progress.

## SECTION IV

### INPUT/OUTPUT SYSTEM

#### INTRODUCTION

The SYSTEMS 72 input/output system, using an input/output processor (PIOP or MIOP), processes input/output requests on a demand-multiplexed basis. The PIOP is activated by interrupts from a dual-level, priority interrupt system, which assigns the highest priority to input/output service requests and the lowest priority to interrupt requests signalling the end of an input/output (I/O) data transfer. The input/output system also allows the user to interface devices, through the Model 7240 Direct Access Channel (DAC), or through the use of the basic programmed output (POT) and programmed input (PIN) instructions described in Section II.

The PIOP functionally appears to be an integral part of the central processor unit (CPU) and differs from the external MIOP only in that it requires more frequent servicing by the operating system. Operating under control of the PIOP, the CPU outputs data to, and inputs data from, the various peripheral devices connected to the programmed input/output (PIO) bus. Peripheral devices connected to the PIO bus operate independently and asynchronously because the associated device controllers require minimal direction from the PIOP and because each controller contains a separate data buffer. Thumbwheel switches located on device controllers provide maximum flexibility in assigning device addresses.

The SYSTEMS 72 operating systems fully utilize the I/O capability of the PIOP, so that the programmer is relieved of all tasks associated with using the PIOP. The only requirement is to set up a four-word input/output command list (IOCL) table and start the I/O data transfer. This feature provides PIOP/MIOP transparency to the user.

#### DUAL LEVEL INTERRUPT SYSTEM

The dual-level interrupt system processes two types of interrupts as follows:

- Input/Output Service Requests
- Interrupt Requests

Input/output service requests are generated by the peripheral device controllers to inform the PIOP that an I/O data transfer is required by way of the PIO bus. Since many peripheral devices operate at high data transfer rates, input/output service requests are assigned a higher priority than the interrupt requests, which simply inform the PIOP that a data transfer has been completed.

Regardless of the type of interrupt, the method of informing the PIOP of the request and the associated address is the same. The request is transferred from the associated device controller to the CPU which, unless the request is inhibited, provides an interrupt strobe back to the device controller. The strobe causes the device controller to place the 16-bit interrupt address on the PIO bus, indicate the mapped/unmapped status of the address, and configure the condition codes. Service interrupts are always unmapped and have six bit addresses.

When the CPU completes execution of the current instruction, it accesses the location specified by the interrupt address. The contents of this location contain a pointer address that defines the starting location of a four-word table that is used to switch program context to the interrupt subroutine. (If the device controller indicated that the interrupt address was mapped, the pointer address is also mapped.) The CPU stores the current PSW1 in the first word of the four-word table and the current PSW2 in the second. A new PSW1 is then removed from the third word of the table and a new PSW2 is removed from the fourth, after which processing resumes under control of the new program status doubleword, which indicates to the PIOP how the interrupt is to be serviced.

#### *I/O PRIORITY*

The I/O priority of both input/output service requests and interrupt requests is dependent upon the I/O card slot location of the device controllers located in the chassis assembly provided with the basic configuration of the SYSTEMS 72 and the Model 7271 I/O Expansion Chassis provided with an expanded configuration. The device controller installed nearest to the central processor unit (CPU) is assigned the highest priority. In the chassis assembly provided in the basic system, the device controller in card slot 15 has the highest priority and the device controller in card slot 21 has the lowest priority. In expanded configurations, the device controller closest to the I/O extender circuit card has the highest priority of the cards installed in the Model 7271 I/O Expansion Chassis.

The interrupt system is mechanized so that even though the input/output service request and the interrupt request for a given device controller have the same card slot priority, the lower priority interrupt request will be serviced only after all of the active input/output service requests have been serviced. This ensures that all devices waiting to transfer data will be serviced before any lower priority interrupt request is recognized.

#### *INTERRUPT INHIBIT*

Both input/output service requests and interrupt requests may be inhibited by the user. Setting the IOI bit in PSW2 inhibits all input/output service requests, but not interrupt service requests. Setting the bit 11 in PSW2 inhibits all interrupt requests, but not input/output service requests.

#### *INTERRUPT CLEAR*

At the end of the interrupt subroutine, execution of a branch return and clear (BRC) instruction with the X field set will return program control to the previous program status doubleword and clear the highest active interrupt.

#### *SYSTEM INTERRUPTS*

System interrupts are implemented by expanding the basic configuration of SYSTEMS 72 to include Model 7251 Priority Interrupt Pair options. The system interrupts are similar to the interrupts generated by the device controllers; however, the system interrupts function only as low priority interrupt requests, which allows these interrupts to be inhibited by the II bit of PSW2.

Each Model 7251 Priority Interrupt Pair provides two system interrupts, so that each card contains an even-numbered interrupt and an odd-numbered interrupt. Addresses for the systems interrupts are X'0080' through X'01FF', which covers the maximum complement of 384 interrupts. Toggle switches mounted on the cards are used to select the eight most-significant bits of the nine-bit interrupt address. A circuit on the card supplies the least-significant bit in the address, which indicates whether the even-numbered or odd-numbered interrupt is selected.

The system interrupts may be armed, enabled, triggered, sensed, or set. These parameters are defined as follows:

- Armed - Arming turns an interrupt on. A program may disarm an interrupt to reassign a stimulus to a different priority level or to remove the stimulus altogether.
- Enabled - Enabling an armed interrupt allows the interrupt to request service and acknowledge a stimulus. A program may disable an interrupt, which prevents the interrupt from requesting service, but not from acknowledging the stimulus. This allows the program to defer response to the stimulus without losing track of it.
- Triggered - Triggering an interrupt permits a program to initiate an interrupt stimulus of its own. These program-generated interrupts may be used to: simulate external system elements during program checkout, and allocate portions of a program to an external stimuli queue.
- Sensed - Sensing allows the program to determine whether an interrupt is armed/disarmed, enabled/disabled, waiting or active, and whether the associated interrupt address is to be mapped or unmapped.
- Set - Setting permits the program to configure the interrupt status. The interrupts must be set initially and during power restart. In addition, an interrupt may require a change in status during program execution.

*Set System  
Interrupts*

To set the status of a system interrupt, the program must first load the D-register with a bit pattern in the format illustrated below. After loading the desired bit pattern, the program executes a POT instruction with an effective address of X'01TT', where TT represents the toggle-switch address of the interrupt pair.

<u>D-Register Bit Position</u>	<u>Bit Function When Set</u>
0	Enables bits 2, 3, 5, 6, and 7
1	Enables bits 2, 3, 4, and 5 even if bit 0 is reset
2	Arms even interrupt
3	Enables even interrupt
4	Triggers even interrupt
5	Maps even interrupt address
6	Sets even interrupt to waiting state
7	Sets even interrupt to active state
8	Enables bits 10, 11, 13, 14, and 15
9	Enables bits 10, 11, 12, and 13 even if bit 8 is reset
10	Arms odd interrupt
11	Enables odd interrupt
12	Triggers odd interrupt
13	Maps odd interrupt address
14	Sets odd interrupt to waiting state
15	Sets odd interrupt to active state

Note that the high-order byte, bits 0 through 7, of the D-register controls the even interrupt, and the low-order byte, bits 8 through 15, controls the odd interrupt.

*Sense System  
Interrupts*

To sense the status of a system interrupt, the program executes PIN instruction with an effective address of X'01TT', where TT represents the toggle-switch address of the interrupt pair. The status of the interrupt pair is transferred to the D-register in the format indicated below.

<u>D-Register Bit Position</u>	<u>Bit Function When Set</u>
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

Note that the high-order byte, bits 0 through 7, of the D-register indicates the status of the even interrupt, and the low-order byte, bits 8 through 15, indicates the status of the odd interrupt.

**MEMORY TRAPS**

Although not classified as an interrupt, a memory trap represents an interrupt of the highest priority, which causes an immediate interrupt to actual location X'0042'. Refer to the paragraph in Section III on the memory trap for a discussion.

**INPUT/OUTPUT  
COMMAND LIST**

The four-word input/output command list (IOCL) used by the PIOP and MIOP to control I/O data transfers between the CPU and a device controller is illustrated below.

**IOCL TABLE**

Word 0	ORDER BYTE	FLAG BYTE
Word 1	INTERRUPT ADDRESS	
Word 2	DATA ADDRESS	
Word 3	BYTE COUNT	

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

### Word 0

The high-order byte, bits 0 through 7, of word 0 contains the order byte, which specifies the operation to be performed by the peripheral device. The low-order byte, bits 8 through 15, contains the flag byte, which specifies details of the transfer and the method of terminating the transfer. The significance of the bits in the order and flag bytes is indicated below:

#### Order Byte:

<u>Bit Configuration</u>								<u>Device Order</u>
<u>0</u>	<u>1</u>	<u>2</u>	<u>3</u>	<u>4</u>	<u>5</u>	<u>6</u>	<u>7</u>	
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
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	Halt

#### Flag Byte:

<u>Flag Bit</u>	<u>Meaning If Set</u>
8	Map 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

### Word 1

Word 1 of the IOCL contains the terminal interrupt address. In terminating an I/O operation, the device controller interrupts to this address. This address points to the location that is used to save the old PSW1 and PSW2 and get the new PSW1 and PSW2.

### Word 2

Word 2 of the IOCL contains the data address, which points to the first word of the data block. In byte transfers, the left byte is byte 0.

### Word 3

Word 3 of the IOCL contains a byte count that indicates the number of bytes to be transferred.

**INPUT/OUTPUT  
INSTRUCTIONS**

Input/output instructions required to initiate and monitor I/O operations are derivatives of the basic POT and PIN instructions discussed in section III. The configuration of the five I/O instructions, the associated effective address, and the function of each instruction is defined below. In the effective address, AA AAAA in the six least-significant bits represents the device address, which is set using the thumbwheel switches on the associated device controller card.

POT Instructions

SIO Start IO

Effective address



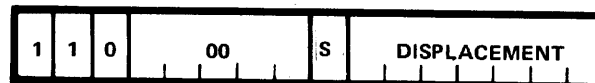
0000 0001 00AA AAAA

TIO Test IO



0000 0001 10AA AAAA

HIO Halt IO



0000 0001 01AA AAAA

IOR IO Reset

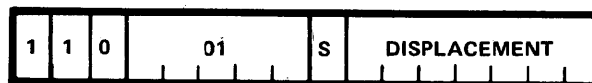


0000 0100 0000 0011

PIN Instruction:

TDV Test Device

0000 0000 11AA AAAA



The six-bit device address ensures that SIO, TIO, HIO, and TDV instructions operate only on the addressed device. The IOR instruction has a fixed effective address and operates on all devices connected on the PIO bus. Execution of an SIO, TIO, HIO, or TDV instruction sets CC1 if the addressed device is busy, while execution of an IOR instruction sets CC1 if any device is busy. On multidevice controllers CC2 is also set.



*BASIC  
INPUT/OUTPUT  
OPERATION*

Once the operating system has constructed the IOCL table specified by the user program, the user initiates the I/O operation by executing an SIO instruction. This sets the device controller busy and causes the controller to: generate an input/output service request, transfer the device address to the CPU or MIOP by way of the PIO bus, and request an order out.

The PIOP or MIOP responds to the request for service by transferring the order out, which consists of word 1 of the IOCL, to the requesting device controller by way of the PIO bus. Assuming that word 1 does not contain an illegal order, the device controller generates another input/output service request, and specifies the size of the data word, which may be a full 16-bit word or an 8-bit byte. The IOP responds to the second input/output service requests by transferring the data in the specified direction, incrementing the data address, and decrementing the transfer count.

Assuming that the transfer count was not decremented to zero by the data transfer, the IOP processes data requests from other device controllers until the original controller generates another input/output service request, indicating another data transfer is required. This process continues until the transfer count has been decremented to zero, which causes the IOP to issue a terminal order by way of the PIO bus. The terminal order inhibits further data requests and causes the device controller to generate an input/output service request, requesting an order in.

When the IOP issues the order in, the device controller transfers status information to the CPU or MIOP by way of the PIO bus. Typically the status information indicates whether the I/O operation is to be halted, or whether data or command chaining is required. Regardless of the status information transferred to the CPU, the PIOP or MIOP issues an order out, which transfers word 2 of the IOCL to the device controller by way of the PIO bus. The interrupt address in word 2 permits the controller to interrupt to the proper location when an interrupt is specified in the flag byte of word 1 of the IOCL.

Assuming that no data or command chaining is specified in the flag byte, but the device is required to interrupt to the PIOP or MIOP, the device controller generates an interrupt request. When the CPU services the interrupt request, the device controller transfers the interrupt address to the CPU by way of the PIO bus and the request is cleared to terminate the I/O operation.

Data Chaining (Multiple IOCL'S associated with one physical record)

Data chaining is specified by the associated bit in the flag byte or word 1 of the IOCL. When data chaining is required at the end of a data transfer, the device controller requests a new order out from the PIOP or MIOP which responds by transferring word 1 of a new IOCL to the device controller by way of the PIO bus. Although the new IOCL specifies a new data address, transfer count, and normally a new termination sequence, the previous operation does not change. To ensure that the operation does not change, the device controller inhibits the new order byte, so that only the new flag byte is stored by the device controller. After the flag byte has been stored, the device controller generates an input/output service request, which causes the PIOP or MIOP to initiate a transfer to or from the new data block. The data transfer process is the same as previously described.

The data chaining feature provides scatter-read/gather-write capability, so that a single instruction with appropriate IOCLs can initiate a series of operations that either: store information in non-sequential blocks of core, or gather information from several areas of core for subsequent printing as a single line on the teletype or line printer.

Command Chaining (Multiple IOCL's associated with multiple physical records)

Command chaining is specified by the associated bit in the flag byte of word 1 of the IOCL. When command chaining is required at the end of a data transfer, the device controller requests a new order out from the PIOP or MIOP, which transfers word 1 of a new IOCL to the device controller by way of the PIO bus. The command chaining process differs from data chaining since in addition to specifying a new data address, transfer count, and termination sequence, the order byte is allowed to specify a different operation. After the new order and flag bytes have been stored the device controller generates an input/output service request to initiate the new data transfer.

An example of command chaining would be to read n records from magnetic tape using the Read configuration of the order byte, space the magnetic tape forward x records using the Control configuration of the order byte, write n records on the magnetic tape using the Write configuration of the order byte, and stop the tape using the Stop configuration of the order byte.

PIN/POT  
I/O CAPABILITY

Devices that do not require the block transfer capability of the Programmed Input/Output Processor (PIOP) or Multiplexed Input/Output Processor (MIOP) may be interfaced by way of the Program Input/Output (PIO) bus. To use this capability, the user program outputs data by executing the basic POT instruction and inputs data by executing the basic PIN instruction.

**APPENDIX A**  
**GLOSSARY OF TERMS**

This glossary defines various technical terms used in discussions of the SYSTEMS 72 Digital Computer. The terms are listed in alphabetical order.

**ABSOLUTE ADDRESSING**

A method of referencing one of the first 127 memory locations, which are represented in hexadecimal by X'0000' through X'007F'. The absolute address is equal to the value of the DISPLACEMENT field of the instruction when the R, I, X, and S bits of the instruction are all reset to zero.

**ACTUAL MEMORY**

A name commonly given to core memory, which may range from 4096 words in the basic configuration of SYSTEMS 72 to 65,536 words in a fully expanded configuration. SYSTEMS 72 is a virtual memory system, which permits the execution of programs up to the 32,768-word capacity of the disc memory in the basic configuration and up to 65,536 words in an expanded configuration. Although the program normally references locations in virtual memory, the required information must be core resident before the program can continue. (See also virtual memory.)

**ADDRESSABLE REGISTER**

A 16-bit register that may be accessed by addressing by way of the DISPLACEMENT field of the instruction. There are eight addressable registers provided with the SYSTEMS 72. In the basic configuration, the registers occupy the first eight locations in actual memory. In configurations with the Model 7212 High-Speed Register option, the registers are located in external integrated circuits. The eight registers and the associated actual memory address are listed below:

<u>Register</u>	<u>Address</u>
D	X'0000'
A	X'0001'
E	X'0002'
X	X'0003'
B	X'0004'
R1	X'0005'
R2	X'0006'
R3	X'0007'

**ASSEMBLER**

A program that assists the user by translating code or notation understandable to the user into machine language. In the SYSTEMS 72, the MAP assembler accepts and converts a mnemonic equivalent of the various instructions, such as ADD, STA, and CAL2.

**AUTOMATIC PROGRAM FRAGMENTATION**

A method used by a small core-resident program to allocate portions of user programs into the virtual memory space provided by SYSTEMS 72. User programs are fragmented into 256-word pages that are transferred from the disc to the core memory on a demand basis as the program executes. Any 256-word disc page may be transferred into any 256-word core page, which allows contiguous program segments to be scattered in core.

**BASE ADDRESSING**

A method of indexing using the B (base) register. (See pre-indexing).

**BOOTSTRAP LOADER**

A method to allow programs to be loaded from any standard binary input device, such as a paper tape reader or magnetic tape. A standard paper tape bootstrap is provided with the basic configuration of SYSTEMS 72. This bootstrap allows programs to be input from paper tapes by way of the paper tape reader on the teletype. The Model 7214 Automatic Bootstrap loader permits loading from any of the binary input devices connected to the SYSTEMS 72.

**BYTE**

An eight-bit segment of data word. Bits 0 through 7 of a 16-bit word are normally referred to as the high-order byte and bits 8 through 15 are referred to as the low-order byte. Most-significant and even byte are also used to describe the high-order byte, while least-significant and odd byte are used to describe the low-order byte.

**CENTRAL PROCESSOR UNIT**

The functional portion of a digital computer that interprets and executes the program instructions. Normally referred to as the CPU, the processor contains various registers that operate on the program data, busses that transfer the data from one area to another, and the arithmetic element that performs the arithmetic functions.

**COMMAND CHAINING**

A method of input/output (I/O) used by SYSTEMS 72 to permit a single instruction to initiate an entire sequence of events. A typical command chaining operation would be to have the user program start an I/O operation by reading n records from magnetic tape, command chain to space the magnetic tape x records forward, command chain to write n records on the magnetic tape, and command chain to stop the tape.

**COMPILER**

A program that assists the user by allowing the program to be written in a completely human-oriented language, which is not associated with any particular computer. The compiler translates a program written in compiler language into a sequence of instructions that cause the computer to perform the program. In addition, the compiler handles memory allocation and generates machine instructions and code. SYSTEMS 72 offers two optional compilers: BASIC and FORTRAN.

**CONDITION CODES**

A method of indicating the result of an instruction execution. In the SYSTEMS 72, the condition codes are displayed on the control panel by indicators CC1 and CC2 in the PROGRAM STATUS DOUBLEWORD.

**DATA CHAINING**

A method of input/output (I/O) used by SYSTEMS 72 to permit a single instruction to initiate high speed block transfers between a peripheral

device and non-contiguous areas of core memory. Data chaining results in a process referred to as scatter-read/gather-write, which permits information stored in non-contiguous areas of core to be gathered for printing on a single line of the teletype or line printer.

#### DEMAND MULTIPLEXING

A method by which requests for input/output (I/O) service are processed on the basis of demand, rather than being scanned in a fixed priority sequence. SYSTEMS 72 input/output is on a demand multiplexed basis under control of a software-implemented programmed input/output processor (PIOP).

#### DIRECT ACCESS CHANNEL

A device that permits external, systems-oriented equipment to interface directly to the core memory. Normally referred to as a DAC, the channel allows a user program to initiate a high-speed I/O data transfer that is not under control of the software-implemented programmed input/output processor (PIOP) or MIOP. SYSTEMS 72 offers a Model 7245 Direct Access Interface.

#### DISPLACEMENT INDEXING

A method of addressing obtained by adding the contents of an index register to the address portion of the instruction. SYSTEMS 72 offers two types of displacement indexing: pre-indexing, in which the B (base) register is added to the DISPLACEMENT field of the instruction; and post-indexing, in which the X (index) register is added to a previously calculated partial address.

#### DOUBLE INDEXING

A method of addressing in which the contents of two separate index registers are used in the calculation of the effective address. In the SYSTEMS 72, pre-indexing adds the DISPLACEMENT field of the instruction to the B (base) register, after which post-indexing may add the previously calculated partial address to the contents of the X (index) register to obtain the final effective address.

#### DYNAMIC PROGRAM RELOCATION

A process used by a small core-resident program to relocate program segments within the virtual memory space provided by SYSTEMS 72.

#### EFFECTIVE ADDRESS

The final memory reference address resulting from calculations performed by the CPU. The effective address indicates a core location containing a 16-bit word that will be used during execution of the current instruction. The R, I, X, and S bits of the instruction define the operations required to calculate the effective address, which may include the following:

- Relative Addressing
- Pre-Indexing
- Indirect Addressing
- Post-Indexing
- Absolute Addressing

#### EFFECTIVE WORD

Effective word is a term used to describe the contents of the effective address.

#### HEXADECIMAL CODE

Hexadecimal refers to a number system of base 16, so that valid digits in the system are: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and

F. In the SYSTEMS 72, 16-bit words are normally divided into four groups containing four bits each, so that each four-bit group represents a hexadecimal character as indicated below:

Code	Character	Code	Character
0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F

Therefore, the 16-bit word illustrated below represents a hexadecimal code of X'143F'. Note that in text, the code is prefixed by the letter X and is enclosed in single quotes.

0001 0100 0011 1111 = X'143F'

#### INDIRECT ADDRESSING

A method of addressing that defines a memory location containing an address capable of specifying any location in memory. In the SYSTEMS 72 the R and S bits of the instruction indicate relative address or pre-indexing, which is performed prior to indirect addressing. The I bit of the instruction defines the requirement for indirect addressing.

#### INPUT/OUTPUT DEVICE

A device used to input data to the central processor unit (CPU) for processing, or output processed data for storage or printout. Typical input/output (I/O) devices used with the SYSTEMS 72 include teletypes, paper tape reader and punch, card reader, line printer, magnetic tapes, and disc memories.

#### INPUT/OUTPUT SERVICE REQUEST

A request from a device controller, which informs the programmed input/output processor (PIOP) that the associated input/output device is ready to transfer data. This type of request represents a high priority interrupt that is serviced by the PIOP according to a priority established by the physical location of the controller cards in the main CPU chassis and the I/O expansion chassis supplied with an expanded configuration. The highest priority is assigned to the device controller located closest to the CPU cards in the main CPU chassis.

#### INSTRUCTION

An instruction represents one operation in a computer program or routine, such as ADD, STA, or CAL2. The SYSTEMS 72 has 27 basic instructions and 5 optional instructions that are reserved for special user-specified operations. Fifteen of the basic instructions have register-expandable derivatives, so that these instructions operate on all eight addressable registers. Many of the remaining 12 basic instructions also have derivatives, so that the total instruction set for SYSTEMS 72 contains 164 instructions, exclusive of the 5 optional instructions. All basic and derivative instructions for SYSTEMS 72 are provided in appendix B.

#### INTERRUPT REQUEST

A request from a device controller, which informs the programmed input/output processor (PIOP) that the associated input/output device has completed an I/O data transfer. This type of request represents a low priority interrupt that is serviced by the PIOP according to the priority assigned to the I/O device. Priority is determined by the physical location

of the device controller cards located in the main CPU chassis and the I/O extender chassis supplied with an expanded configuration. The highest priority is assigned to the device controller located closest to the CPU cards in the main CPU chassis. Interrupt requests are serviced by the PIOP only after all high priority input/output service requests have been serviced.

#### MEMORY MAP

A device that allows the monitoring of the resident/non-resident core memory status of each 256-word page in the virtual memory space provided by SYSTEMS 72. Non-resident pages addressed by the user program causes a small core-resident program to transfer a seldom used-page from core to disc and the addressed page from disc to core so that the program can continue. The map also contains the protected/unprotected status of each virtual page, which prevents writing into protected areas unless overridden by the central processor unit (CPU). In the basic configuration of SYSTEMS 72, the map contains the status of all 128 pages on the 32,768-word disc. In an expanded configuration, the map maintains the status of 256 pages on a disc having a minimum capacity of 65,536 words.

#### MEMORY PORT

A device for accessing core memory. The SYSTEMS 72 provides three dedicated ports to memory, which are assigned fixed priorities according to data transfer requirements. The highest priority is assigned to the disc memory, so that there may be efficient initiation of page transfers as required by the user program. The next highest priority is assigned to the optional direct access channel (DAC), so that the channel can execute high speed block data transfers. The lowest priority is assigned to the central processor unit (CPU), so that the CPU is permitted access to memory only after the higher priority ports have been serviced.

#### MEMORY REFERENCE INSTRUCTION

An instruction that uses the effective address calculated by the central processor unit (CPU) to reference a memory location that contains data used during execution of the instruction. (See also instruction.)

#### MEMORY TRAP

A method by which the core/disc memory system informs a small core-resident program of an attempt to access a non-resident page, an attempt to write into a protected page, or a parity error during a read operation, providing the system is equipped with the Model 7210 Memory Parity option. Detection of any of the previous conditions while the central processor unit (CPU) is connected for service results in termination of the memory access. In the case of a non-resident page, a seldom-used page will be transferred from core to disc and the required page from disc to core so that the user program can continue.

#### OPERATION CODE

Normally referred to as an op code, the operation code represents the portion of an instruction that indicates what operation is to be performed during execution of the instruction. The OPERATION CODE field for SYSTEMS 72, which occupies bits 3 through 7 of the 16-bit instruction, specifies which of the 27 basic instructions or five optional instructions is to be executed. Appendix B lists all basic and derivative instructions in the SYSTEMS 72 instruction repertoire.

#### PARTIAL ADDRESS

An address formed by the central processor unit (CPU) during calculation of the effective address. Until the CPU has performed all address calculations specified by the R, I, X, and S bits of the instruction, the address is referred to as a partial address.

#### POST-INDEXING

A method of addressing that defines a memory location referenced to the contents of the index (X) register. In the SYSTEMS 72, the R, I, and S bits specify relative addressing, indirect addressing, and pre-indexing, respectively, which must be performed prior to post-indexing. The X bit of the instruction defines the requirement for post-indexing.

#### PRE-INDEXING

A method of addressing that defines a memory location referenced to the contents of the base (B) register. In the SYSTEMS 72, if the R bit of the instruction is not set to request relative addressing, the S bit may be used to request pre-indexing, which occurs prior to indirect addressing or post-indexing as specified by the I and X bits, respectively.

#### PRIVILEGED INSTRUCTIONS

A method of permitting certain instructions to execute only when the central processor unit (CPU) is operating in a certain mode. SYSTEMS 72 offers four privileged instructions: POT, PIN, BRC, and CAL1. These instructions execute only when the CPU is operating in the master mode, which occurs when the MS bit in PSW2 is reset.

#### PROGRAMMABLE REGISTER

A register that may be operated upon directly by the user program. (See also addressable register).

#### PROGRAMMED INPUT/OUTPUT PROCESSOR

A software-implemented method of simulating an external hardware input/output processor. Normally referred to as the PIOP, the processor controls block-mode input/output transfers and eliminates time-consuming input/output programming by the user. All the PIOP requires is that the user set up a four-word table defining the control information required for the transfer, the terminal interrupt address, the starting data address, and the word count, after which the user program starts the transfer, which is completed under control of the PIOP.

#### PROGRAM STATUS DOUBLEWORD

Two 16-bit words that define the current program environment. Program status word 1 (PSW1) indicates the current location of the program counter. Program status word 2 (PSW2) indicates various parameters of the current program as indicated below. The doubleword is displayed by the PROGRAM STATUS DOUBLEWORD indicators on the SYSTEMS 72 control panel.

- Condition Codes - CC1 and CC2
- Compare Sequence Mode - SM1 and SM2
- Input/Output Service Request Inhibit - IOI
- Interrupt Request Inhibit - II
- Interrupt Register Inhibit - IIA
- Address Mapped - MAP
- Master/Slave Mode - MS

#### RAPID CONTEXT SWITCHING

A method of rapidly changing the program environment to process memory traps, interrupts, and call instructions. In performing the switching function, the system first stores the current program environment contained in the program status doubleword. A new doubleword is then accessed to define the program environment required to continue processing. After the memory trap, interrupt, or call has been processed, the original program environment is restored, so that processing can continue from where it was previously halted.

## RELATIVE ADDRESSING

A method of addressing that defines a memory location relative to the program count contained in program status word 1 (PSW1). In the SYSTEMS 72, the R bit of the instruction is set to request relative addressing and the S bit indicates whether the desired address is forward or backward of the current program count. Relative addressing occurs prior to indirect addressing or post-indexing as specified by the I and X bits, respectively. No pre-indexing can occur when relative addressing is requested, since the S bit is used to indicate whether the value in the DISPLACEMENT field is forward or backward of the current program count.

## VIRTUAL MEMORY

A concept by which a small programmable memory space, referred to as actual memory, is effectively increased in size by adding mass storage elements that take on the programmable characteristics of the smaller memory. In the basic configuration of SYSTEMS 72, the 4096-word core represents the actual memory space and the 65,536-word disc represents the virtual memory space. Program segments of 256-words,

which are referred to as pages, are transferred between the core and disc memories under control of a small core-resident program that is transparent to the user.

As long as the user program executes portions of a program that are core-resident as indicated by the memory map, processing will continue. When the program attempts to access a page that is not core-resident, a memory trap is generated that causes the transfer of a seldom-used page from core to disc and then transfer the required from disc to core. After the required page is core-resident, the instruction is re-executed and processing resumes.

## WRITE PROTECT

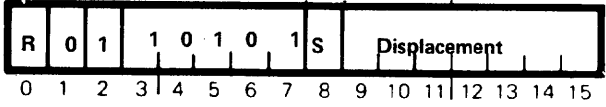
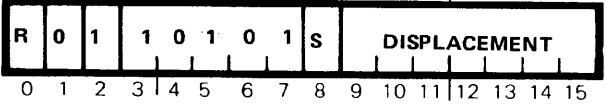
A method used to prevent destroying information stored in specified pages of the virtual memory space. An attempt to write into a protected page results in a memory trap that terminates the memory access. The only exception occurs when the central processor unit (CPU) is connected for service, since the CPU may override the page write protect bit stored in the memory map.

## APPENDIX B

### SYSTEMS 72 DERIVATIVE INSTRUCTIONS

The SYSTEMS 72 provides an instruction repertoire of 27 basic instructions, which are discussed in section II of this manual, and five optional instructions that may be implemented for expanded configurations. Fifteen of the basic instructions are register-expandable, so that derivatives of these instructions operate on the eight programmable registers. Many of the remaining 12 basic instructions also have derivatives. This appendix lists all derivative instructions.

Basic Instruction	Derivative, Format, and/or Function																																																																																																																
<b>BCR-15</b>	<p style="text-align: center;"><b>BE Branch if Equal</b></p> <div style="text-align: center; border: 1px solid black; padding: 5px; margin: 10px auto; width: fit-content;"> <table style="border-collapse: collapse; margin: 0 auto;"> <tr> <td style="border: 1px solid black; padding: 2px 5px;">R</td> <td style="border: 1px solid black; padding: 2px 5px;">1</td> <td style="border: 1px solid black; padding: 2px 5px;">1</td> <td style="border: 1px solid black; padding: 2px 5px;">1</td> <td style="border: 1px solid black; padding: 2px 5px;">0</td> <td style="border: 1px solid black; padding: 2px 5px;">1</td> <td style="border: 1px solid black; padding: 2px 5px;">0</td> <td style="border: 1px solid black; padding: 2px 5px;">1</td> <td style="border: 1px solid black; padding: 2px 5px;">S</td> <td colspan="7" style="border: 1px solid black; padding: 2px 5px; text-align: center;">DISPLACEMENT</td> </tr> <tr> <td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">2</td><td style="text-align: center;">3</td><td style="text-align: center;">4</td><td style="text-align: center;">5</td><td style="text-align: center;">6</td><td style="text-align: center;">7</td><td style="text-align: center;">8</td><td style="text-align: center;">9</td><td style="text-align: center;">10</td><td style="text-align: center;">11</td><td style="text-align: center;">12</td><td style="text-align: center;">13</td><td style="text-align: center;">14</td><td style="text-align: center;">15</td> </tr> </table> </div> <p>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.</p> <table style="margin: 10px auto; border-collapse: collapse;"> <tr> <td style="padding-right: 20px;">EXAMPLE</td> <td style="padding-right: 20px;">Memory Location:</td> <td style="text-align: right;">1050</td> </tr> <tr> <td></td> <td>Hex Instruction:</td> <td style="text-align: right;">E504</td> </tr> <tr><td colspan="3"> </td></tr> <tr> <td>BEFORE EXECUTION</td> <td>PSW1</td> <td style="text-align: right;">PSW2</td> </tr> <tr> <td></td> <td style="text-align: right;">1050</td> <td style="text-align: right;">8003</td> </tr> <tr><td colspan="3"> </td></tr> <tr> <td>AFTER EXECUTION</td> <td>PSW1</td> <td style="text-align: right;">PSW2</td> </tr> <tr> <td></td> <td style="text-align: right;">1051</td> <td style="text-align: right;">8003</td> </tr> </table> <p style="text-align: center; margin-top: 20px;"><b>BEZ Branch if Equal to Zero</b></p> <div style="text-align: center; border: 1px solid black; padding: 5px; margin: 10px auto; width: fit-content;"> <table style="border-collapse: collapse; margin: 0 auto;"> <tr> <td style="border: 1px solid black; padding: 2px 5px;">R</td> <td style="border: 1px solid black; padding: 2px 5px;">1</td> <td style="border: 1px solid black; padding: 2px 5px;">0</td> <td style="border: 1px solid black; padding: 2px 5px;">1</td> <td style="border: 1px solid black; padding: 2px 5px;">0</td> <td style="border: 1px solid black; padding: 2px 5px;">1</td> <td style="border: 1px solid black; padding: 2px 5px;">0</td> <td style="border: 1px solid black; padding: 2px 5px;">1</td> <td style="border: 1px solid black; padding: 2px 5px;">S</td> <td colspan="7" style="border: 1px solid black; padding: 2px 5px; text-align: center;">DISPLACEMENT</td> </tr> <tr> <td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">2</td><td style="text-align: center;">3</td><td style="text-align: center;">4</td><td style="text-align: center;">5</td><td style="text-align: center;">6</td><td style="text-align: center;">7</td><td style="text-align: center;">8</td><td style="text-align: center;">9</td><td style="text-align: center;">10</td><td style="text-align: center;">11</td><td style="text-align: center;">12</td><td style="text-align: center;">13</td><td style="text-align: center;">14</td><td style="text-align: center;">15</td> </tr> </table> </div> <p>The effective address replaces PSW1 if the contents of the relevant register are equal to zero. BEZ is appropriate after a load instruction, in which the receiving register is relevant, or a logical instruction, in which the A-register is relevant.</p> <table style="margin: 10px auto; border-collapse: collapse;"> <tr> <td style="padding-right: 20px;">EXAMPLE</td> <td style="padding-right: 20px;">Memory Location:</td> <td style="text-align: right;">1020</td> </tr> <tr> <td></td> <td>Hex Instruction:</td> <td style="text-align: right;">D507</td> </tr> <tr><td colspan="3"> </td></tr> <tr> <td>BEFORE EXECUTION</td> <td>PSW1</td> <td style="text-align: right;">PSW2</td> </tr> <tr> <td></td> <td style="text-align: right;">1030</td> <td style="text-align: right;">4002</td> </tr> <tr><td colspan="3"> </td></tr> <tr> <td>AFTER EXECUTION</td> <td>PSW1</td> <td style="text-align: right;">PSW2</td> </tr> <tr> <td></td> <td style="text-align: right;">1031</td> <td style="text-align: right;">4002</td> </tr> </table>	R	1	1	1	0	1	0	1	S	DISPLACEMENT							0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	EXAMPLE	Memory Location:	1050		Hex Instruction:	E504				BEFORE EXECUTION	PSW1	PSW2		1050	8003				AFTER EXECUTION	PSW1	PSW2		1051	8003	R	1	0	1	0	1	0	1	S	DISPLACEMENT							0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	EXAMPLE	Memory Location:	1020		Hex Instruction:	D507				BEFORE EXECUTION	PSW1	PSW2		1030	4002				AFTER EXECUTION	PSW1	PSW2		1031	4002
R	1	1	1	0	1	0	1	S	DISPLACEMENT																																																																																																								
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15																																																																																																		
EXAMPLE	Memory Location:	1050																																																																																																															
	Hex Instruction:	E504																																																																																																															
BEFORE EXECUTION	PSW1	PSW2																																																																																																															
	1050	8003																																																																																																															
AFTER EXECUTION	PSW1	PSW2																																																																																																															
	1051	8003																																																																																																															
R	1	0	1	0	1	0	1	S	DISPLACEMENT																																																																																																								
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15																																																																																																		
EXAMPLE	Memory Location:	1020																																																																																																															
	Hex Instruction:	D507																																																																																																															
BEFORE EXECUTION	PSW1	PSW2																																																																																																															
	1030	4002																																																																																																															
AFTER EXECUTION	PSW1	PSW2																																																																																																															
	1031	4002																																																																																																															

Basic Instruction	Derivative, Format, and/or Function
BCR-15 (Cont'd)	<p style="text-align: center;"><b>BGE Branch if Greater than or Equal</b></p> <div style="text-align: center;">  </div> <p>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.</p> <p style="margin-left: 40px;"> <b>EXAMPLE</b> Memory Location: 1060  Hex Instruction: B505 </p> <p style="margin-left: 40px;"> <b>BEFORE EXECUTION</b> PSW1 PSW2  1060 8003 </p> <p style="margin-left: 40px;"> <b>AFTER EXECUTION</b> PSW1 PSW2  1065 8003 </p> <p style="text-align: center; margin-top: 20px;"><b>BGEZ Branch if Greater than or Equal to Zero</b></p> <div style="text-align: center;">  </div> <p>The effective address replaces PSW1 if the contents of the relevant register are greater than or equal to zero. BGEZ is appropriate immediately after a load instruction, in which the receiving register is relevant, or a logical instruction, in which the A-register is relevant.</p> <p style="margin-left: 40px;"> <b>EXAMPLE</b> Memory Location: 1040  Hex Instruction: B505 </p> <p style="margin-left: 40px;"> <b>BEFORE EXECUTION</b> PSW1 PSW2  1040 4002 </p> <p style="margin-left: 40px;"> <b>AFTER EXECUTION</b> PSW1 PSW2  1041 4002 </p>

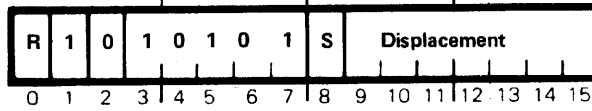


Basic  
Instruction

Derivative, Format, and/or Function

BCR - 15  
(Cont'd)

BNC Branch on No Carry



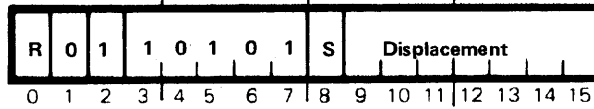
The effective address replaces PSW1 if the last arithmetic operation did not cause a carry. BNC is appropriate immediately after an arithmetic instruction.

EXAMPLE Memory Location: 1070  
Hex Instruction: D506

BEFORE EXECUTION PSW1 PSW2  
1070 8003

AFTER EXECUTION PSW1 PSW2  
1071 9003

BNO Branch on No Overflow



The effective address replaces PSW 1 if the last arithmetic operation did not cause an overflow. BNO is appropriate immediately after an arithmetic instruction.

EXAMPLE Memory Location: 1080  
Hex Instruction: B503

BEFORE EXECUTION PSW1 PSW2  
1080 8003

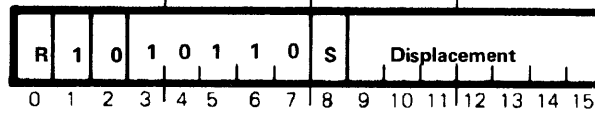
AFTER EXECUTION PSW1 PSW2  
1083 9003

Basic  
Instruction

Derivative, Format, and/or Function

BCS - 16

BC Branch on Carry



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

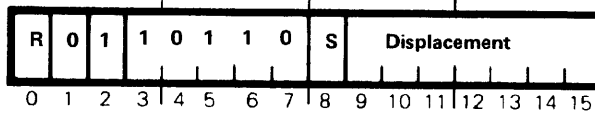
EXAMPLE: Memory Location: 10E0

Hex Instruction: D604

BEFORE EXECUTION: PSW1 PSW2  
10E0 8002

AFTER EXECUTION: PSW1 PSW2  
10E4 8002

BL Branch if Less Than



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.

EXAMPLE: Memory Location: 10D0

Hex Instruction: B602

BEFORE EXECUTION PSW1 PSW2  
10D0 4002

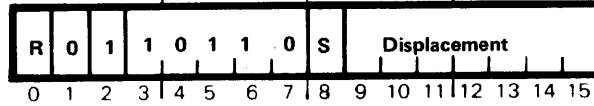
AFTER EXECUTION PSW1 PSW2  
10D2 4002

Basic Instruction

Derivative, Format, and/or Function

BSC-16  
(Cont'd)

**BLZ Branch if Less than Zero**



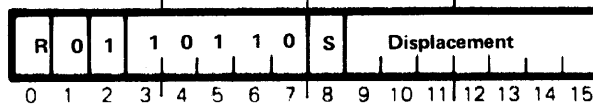
The effective address replaces PSW1 if the contents of the relevant register are less than zero. BLZ is appropriate immediately after a load instruction in which, the receiving register is relevant, or a logical instruction in which the A register is relevant.

EXAMPLE Memory Location: 10B0  
Hex Instruction: B604

BEFORE EXECUTION PSW1 PSW2  
10B0 4002

AFTER EXECUTION PSW1 PSW2  
10B4 4002

**BO Branch on Overflow**

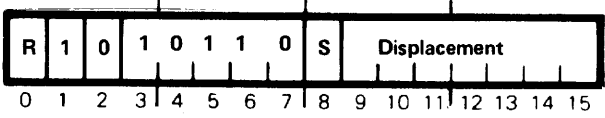
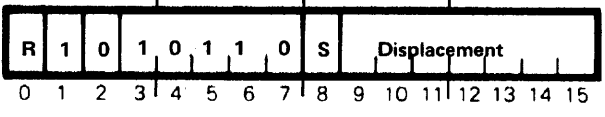


The effective address replaces PSW 1 if the last arithmetic operation caused an overflow. BO is appropriate immediately after an arithmetic instruction.

EXAMPLE: Memory Location: 10F0  
Hex Instruction: B602

BEFORE EXECUTION PSW1 PSW2  
10F0 8002

AFTER EXECUTION PSW1 PSW2  
10F1 8002

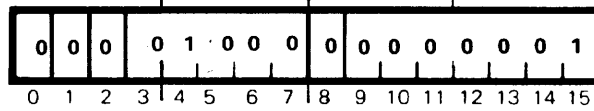
Basic Instruction	Derivative, Format, and/or Function
BCS-1F (Cont'o.	<p style="text-align: center;"><b>BNE Branch if Not Equal</b></p> <div style="text-align: center;">  <p style="margin-left: 100px;">0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15</p> </div> <p>The effective address replaces PSW1 if the contents of the A register and the other operand are not equal. BNE is appropriate immediately after a compare instruction.</p> <p style="margin-left: 100px;"> <b>EXAMPLE</b> Memory Location: 10C0  Hex Instruction: D603 </p> <p style="margin-left: 100px;"> <b>BEFORE EXECUTION</b> PSW1 PSW2  10C0 4002 </p> <p style="margin-left: 100px;"> <b>AFTER EXECUTION</b> PSW1 PSW2  10C1 4002 </p> <p style="text-align: center;"><b>BNEZ Branch if Not Equal to Zero</b></p> <div style="text-align: center;">  <p style="margin-left: 100px;">0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15</p> </div> <p>The effective address replaces PSW1 if the contents of the relevant register do not equal zero. BNEZ is appropriate immediately after a load instruction in which, the receiving register is relevant, or a logical instruction, the A register is relevant.</p> <p style="margin-left: 100px;"> <b>EXAMPLE</b> Memory Location: 10A0  Hex Instruction: D505 </p> <p style="margin-left: 100px;"> <b>BEFORE EXECUTION</b> PSW1 PSW2  10A0 C002 </p> <p style="margin-left: 100px;"> <b>AFTER EXECUTION</b> PSW1 PSW2  10A5 C002 </p>

Basic Instruction

Derivative, Format, and/or Function

LDD - 08

LDDA Load D from A

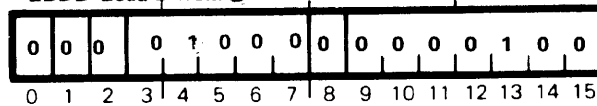


The contents of the A register replace the contents of the D register.

EXAMPLE: Memory Location: 1001  
Hex Instruction: 0801

BEFORE EXECUTION	PSW1	PSW2	A	D
	1001	C002	FFFF	8000
AFTER EXECUTION	PSW1	PSW2	A	D
	1002	C002	FFFF	FFFF

Lddb Load D from B



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

EXAMPLE: Memory Location: 1004  
Hex Instruction: 0804

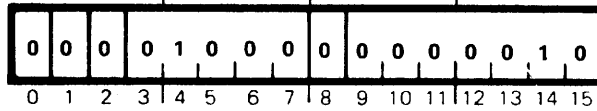
BEFORE EXECUTION:	PSW1	PSW2	B	D
	1004	0002	0300	FFFC
AFTER EXECUTION	PSW1	PSW2	B	D
	1005	8002	0300	0300

**Basic Instruction**

**Derivative, Format, and/or Function**

**LDD - 08**  
Cont'd

**LDDE Load D from E**



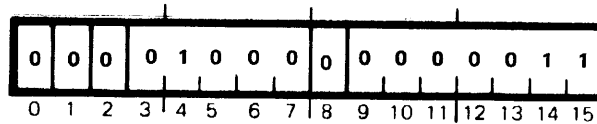
The contents of the E register replace the contents of the D register.

EXAMPLE: Memory Location: 1002  
Hex Instruction: 0802

BEFORE EXECUTION	PSW1	PSW2	E	D
	1002	C002	000A	FFFF

AFTER EXECUTION	PSW1	PSW2	E	D
	1003	8002	000A	000A

**LDDX Load D from X**



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

EXAMPLE: Memory Location: 1003  
Hex Instruction: 0803

BEFORE EXECUTION:	PSW1	PSW2	X	D
	1003	8002	FFFC	000A

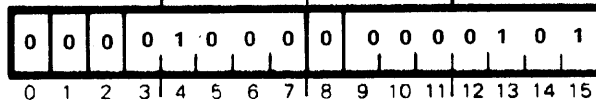
AFTER EXECUTION:	PSW1	PSW2	X	D
	1004	0002	FFFC	FFFC

Basic Instruction

Derivative, Format, and/or Function

LDD-08  
(Cont'd)

LDD1 Load D from R1



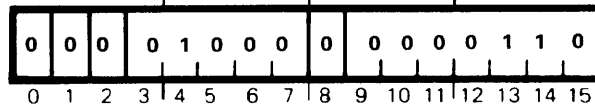
The contents of the R1 register replace the contents of the D register.

EXAMPLE: Memory Location: 1005  
Hex Instruction: 0805

BEFORE EXECUTION:	PSW1	PSW2	R1	D
	1005	8002	0000	0300

AFTER EXECUTION:	PSW1	PSW2	R1	D
	1006	0002	0000	0000

LDD2 Load D from R2



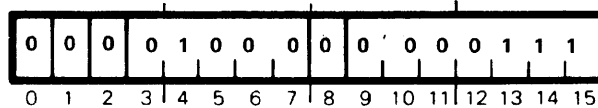
The contents of the R2 register replace the contents of the D register.

EXAMPLE: Memory Location: 1006  
Hex Instruction: 0806

BEFORE EXECUTION:	PSW1	PSW2	R2	D
	1006	0002	0020	0000

AFTER EXECUTION:	PSW1	PSW2	R2	D
	1007	8002	0020	0020

LDD3 Load D from R: 3



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

EXAMPLE: Memory Location: 1007  
Hex Instruction: 0807

BEFORE EXECUTION:	PSW1	PSW2	R3	D
	1007	8002	38B2	0020

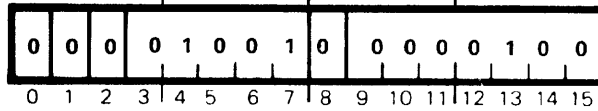
AFTER EXECUTION:	PSW1	PSW2	R3	D
	1008	8002	38B2	38B2

Basic Instruction

Derivative, Format, and/or Function

LDA - 09

LDAB Load A from B



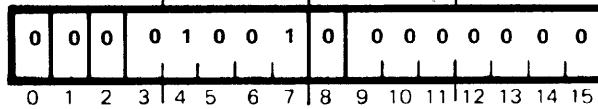
The contents of the B register replace the contents of the A register.

EXAMPLE: Memory Location: 100C  
Hex Instruction: 0904

BEFORE EXECUTION:	PSW1	PSW2	B	A
	100C	C002	0300	FFFF

AFTER EXECUTION:	PSW1	PSW2	B	A
	100D	8002	0300	0300

LDAD Load A from D



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

EXAMPLE: Memory Location: 1009  
Hex Instruction: 0900

BEFORE EXECUTION:	PSW1	PSW2	D	A
	1009	8002	38B2	7FFF

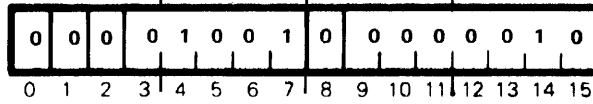
AFTER EXECUTION:	PSW1	PSW2	D	A
	100A	8002	38B2	38B2



Basic Instruction	Derivative, Format, and/or Function
-------------------	-------------------------------------

LDA-09  
(Cont'd)

**LDAE Load A from E**



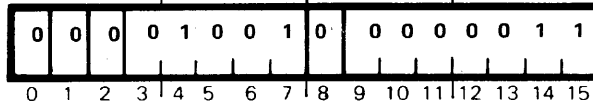
The contents of the E register replace the contents of the A register.

EXAMPLE:       Memory Location:     100A  
                  Hex Instruction:     0902

BEFORE EXECUTION:	PSW1	PSW2	E	A
	100A	8002	000A	38B2

AFTER EXECUTION:	PSW1	PSW2	E	A
	100B	8002	000A	000A

**LDAX Load A from X**



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

EXAMPLE:       Memory Location:     100B  
                  Hex Instruction:     0903

BEFORE EXECUTION:	PSW1	PSW2	X	A
	100B	8002	FFFC	000A

AFTER EXECUTION:	PSW1	PSW2	X	A
	100C	C002	FFFC	FFFC

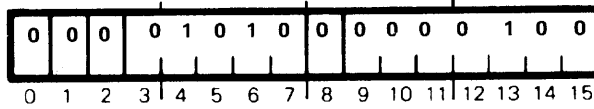
Basic Instruction	Derivative, Format, and/or Function																																																																																																																																																									
<b>LDA - 09</b> (Cont'd)	<p style="text-align: center;"><b>LDA1 Load A from R1</b></p> <div style="text-align: center;"> <table border="1" style="margin: auto;"> <tr> <td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td> </tr> <tr> <td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td> </tr> </table> </div> <p>The contents of the R1 register replace the contents of the A register.</p> <p><b>EXAMPLE</b>    Memory Location: 100D                              Hex Instruction: 0905</p> <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 30%;"><b>BEFORE EXECUTION</b></td> <td style="width: 15%;">PSW1</td> <td style="width: 15%;">PSW2</td> <td style="width: 15%;">R1</td> <td style="width: 15%;">A</td> </tr> <tr> <td></td> <td>100D</td> <td>8002</td> <td>0000</td> <td>0300</td> </tr> <tr> <td><b>AFTER EXECUTION</b></td> <td>PSW1</td> <td>PSW2</td> <td>R1</td> <td>A</td> </tr> <tr> <td></td> <td>100E</td> <td>0002</td> <td>0000</td> <td>0000</td> </tr> </table> <p style="text-align: center;"><b>LDA2 Load A from R2</b></p> <div style="text-align: center;"> <table border="1" style="margin: auto;"> <tr> <td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td> </tr> <tr> <td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td> </tr> </table> </div> <p>The contents of the R2 register replace the contents of the A register.</p> <p><b>EXAMPLE:</b>    Memory Location: 100E                              Hex Instruction: 0906</p> <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 30%;"><b>BEFORE EXECUTION:</b></td> <td style="width: 15%;">PSW1</td> <td style="width: 15%;">PSW2</td> <td style="width: 15%;">R2</td> <td style="width: 15%;">A</td> </tr> <tr> <td></td> <td>100E</td> <td>0002</td> <td>0020</td> <td>0000</td> </tr> <tr> <td><b>AFTER EXECUTION:</b></td> <td>PSW1</td> <td>PSW2</td> <td>R2</td> <td>A</td> </tr> <tr> <td></td> <td>100F</td> <td>8002</td> <td>0020</td> <td>0020</td> </tr> </table> <p style="text-align: center;"><b>LDA3 Load A from R3</b></p> <div style="text-align: center;"> <table border="1" style="margin: auto;"> <tr> <td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td> </tr> <tr> <td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td> </tr> </table> </div> <p>The contents of the R3 register replace the contents of the A register.</p> <p><b>EXAMPLE:</b>    Memory Location: 100F                              Hex Instruction: 0907</p> <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 30%;"><b>BEFORE EXECUTION:</b></td> <td style="width: 15%;">PSW1</td> <td style="width: 15%;">PSW2</td> <td style="width: 15%;">R3</td> <td style="width: 15%;">A</td> </tr> <tr> <td></td> <td>100F</td> <td>8002</td> <td>38B2</td> <td>0020</td> </tr> <tr> <td><b>AFTER EXECUTION:</b></td> <td>PSW1</td> <td>PSW2</td> <td>R3</td> <td>A</td> </tr> <tr> <td></td> <td>1010</td> <td>8002</td> <td>38B2</td> <td>38B2</td> </tr> </table>	0	0	0	0	1	0	0	1	0	0	0	0	1	0	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	<b>BEFORE EXECUTION</b>	PSW1	PSW2	R1	A		100D	8002	0000	0300	<b>AFTER EXECUTION</b>	PSW1	PSW2	R1	A		100E	0002	0000	0000	0	0	0	0	1	0	0	1	0	0	0	0	1	1	0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	<b>BEFORE EXECUTION:</b>	PSW1	PSW2	R2	A		100E	0002	0020	0000	<b>AFTER EXECUTION:</b>	PSW1	PSW2	R2	A		100F	8002	0020	0020	0	0	0	0	1	0	0	1	0	0	0	0	1	1	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	<b>BEFORE EXECUTION:</b>	PSW1	PSW2	R3	A		100F	8002	38B2	0020	<b>AFTER EXECUTION:</b>	PSW1	PSW2	R3	A		1010	8002	38B2	38B2
0	0	0	0	1	0	0	1	0	0	0	0	1	0	1																																																																																																																																												
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15																																																																																																																																											
<b>BEFORE EXECUTION</b>	PSW1	PSW2	R1	A																																																																																																																																																						
	100D	8002	0000	0300																																																																																																																																																						
<b>AFTER EXECUTION</b>	PSW1	PSW2	R1	A																																																																																																																																																						
	100E	0002	0000	0000																																																																																																																																																						
0	0	0	0	1	0	0	1	0	0	0	0	1	1	0																																																																																																																																												
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15																																																																																																																																											
<b>BEFORE EXECUTION:</b>	PSW1	PSW2	R2	A																																																																																																																																																						
	100E	0002	0020	0000																																																																																																																																																						
<b>AFTER EXECUTION:</b>	PSW1	PSW2	R2	A																																																																																																																																																						
	100F	8002	0020	0020																																																																																																																																																						
0	0	0	0	1	0	0	1	0	0	0	0	1	1	1																																																																																																																																												
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15																																																																																																																																											
<b>BEFORE EXECUTION:</b>	PSW1	PSW2	R3	A																																																																																																																																																						
	100F	8002	38B2	0020																																																																																																																																																						
<b>AFTER EXECUTION:</b>	PSW1	PSW2	R3	A																																																																																																																																																						
	1010	8002	38B2	38B2																																																																																																																																																						

Basic  
Instruction

Derivative, Format, and/or Function

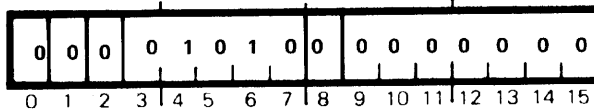
**LBY - 0A**

**LBYB Load Byte into A from B**



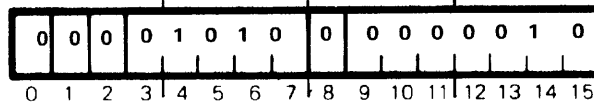
Bits 0 through 7 of the B register replace bits 8 through 15 of the A register and bits 0 through 7 of the A register are cleared.

**LBXD Load Byte into A from D**



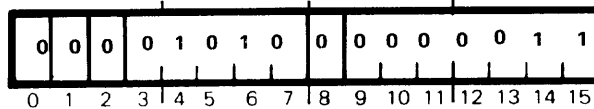
Bits 0 through 7 of the D register replace bits 8 through 15 of the A register and bits 0 through 7 of the A register are cleared.

**LBYE Load Byte into A from E**



Bits 0 through 7 of the E register replace bits 8 through 15 of the A register and bits 0 through 7 of the A register are cleared.

**LBXX Load Byte into A from X**



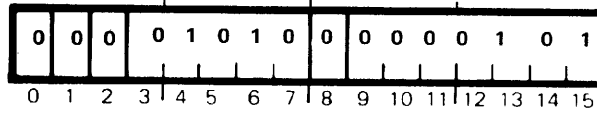
Bits 0 through 7 of the X register replace bits 8 through 15 of the A register and bits 0 through 7 of the A register are cleared.

Basic  
Instruction

Derivative, Format, and/or Function

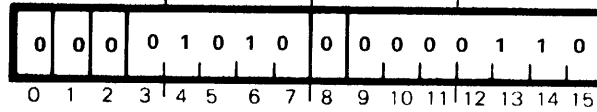
LBY - 0A  
(Cont'd)

LBY1 Load Byte into A from R1



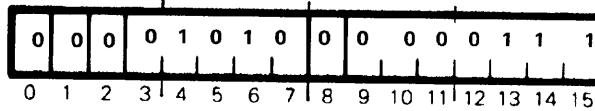
Bits 0 through 7 of the R1 register replace bits 8 through 15 of the A register and bits 0 through 7 of the A register are cleared.

LBY2 Load Byte into A from R2



Bits 0 through 7 of the R2 register replace bits 8 through 15 of the A register and bits 0 through 7 of the A register are cleared.

LBY3 Load Byte into A from R3



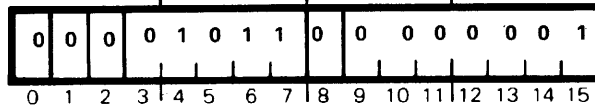
Bits 0 through 7 of the R3 register replace bits 8 through 15 of the A register and bits 0 through 7 of the A register are cleared.

Basic  
Instruction

Derivative, Format, and/or Function

LDX - 0B

LDXA Load X from A



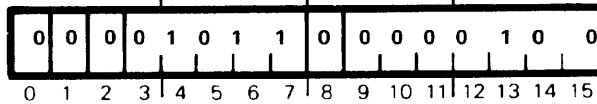
The contents of the A register replace the contents of the X register.

EXAMPLE: Memory Location: 1013  
Hex Instruction: 0B01

BEFORE EXECUTION:	PSW1	PSW2	A	X
	1013	8002	38B2	4000

AFTER EXECUTION:	PSW1	PSW2	A	X
	1014	8002	38B2	38B2

LDXB Load X from B



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

EXAMPLE: Memory Location: 1015  
Hex Instruction: 0B04

BEFORE EXECUTION:	PSW1	PSW2	B	X
	1015	8002	0300	000A

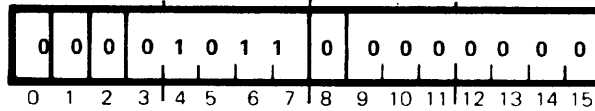
AFTER EXECUTION:	PSW1	PSW2	B	X
	1016	8002	0300	0300

**Basic Instruction**

**Derivative, Format, and/or Function**

**LDX - 0B  
(Cont'd)**

**LDXD Load X from D**



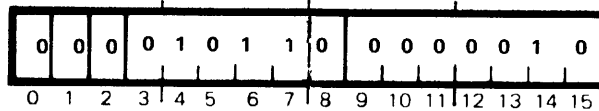
The contents of the D register replace the contents of the X register.

EXAMPLE: Memory Location: 1012  
Hex Instruction: 0B00

BEFORE EXECUTION:	PSW1	PSW2	D	X
	1012	C002	4C00	FFFF

AFTER EXECUTION:	PSW1	PSW2	D	X
	1013	8002	4C00	4C00

**LDXE Load X from E**

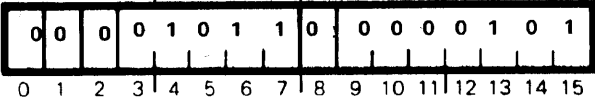
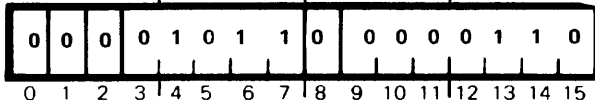
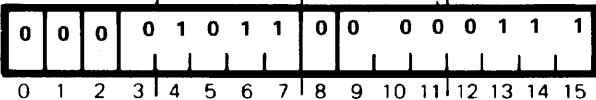


The contents of the E register replace the contents of the X register.

EXAMPLE: Memory Location: 1014  
Hex Instruction: 0B02

BEFORE EXECUTION:	PSW1	PSW2	E	X
	1014	8002	000A	38B2

AFTER EXECUTION:	PSW1	PSW2	E	X
	1015	8002	000A	000A

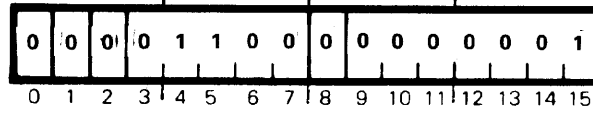
Basic Instruction	Derivative, Format, and/or Function																																																												
<p>LDX - 0B (Cont'd)</p>	<p style="text-align: center;"><b>LDX1 Load X from R1</b></p> <div style="text-align: center;">  <p style="margin-left: 40px;">0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15</p> </div> <p>The contents of the R1 register replace the contents of the X register.</p> <p>EXAMPLE: Memory Location: 1016 Hex Instruction: 0B05</p> <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 30%;">BEFORE EXECUTION:</td> <td style="width: 15%;">PSW1</td> <td style="width: 15%;">PSW2</td> <td style="width: 15%;">R1</td> <td style="width: 25%;">X</td> </tr> <tr> <td></td> <td style="text-align: center;">1016</td> <td style="text-align: center;">8002</td> <td style="text-align: center;">0000</td> <td style="text-align: center;">0300</td> </tr> <tr> <td>AFTER EXECUTION:</td> <td>PSW1</td> <td>PSW2</td> <td>R1</td> <td>X</td> </tr> <tr> <td></td> <td style="text-align: center;">1017</td> <td style="text-align: center;">0002</td> <td style="text-align: center;">0000</td> <td style="text-align: center;">0000</td> </tr> </table> <p style="text-align: center;"><b>LDX2 Load X from R2</b></p> <div style="text-align: center;">  <p style="margin-left: 40px;">0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15</p> </div> <p>The contents of the R2 register replace the contents of the X register.</p> <p>EXAMPLE: Memory Location: 1017 Hex Instruction: 0B06</p> <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 30%;">BEFORE EXECUTION:</td> <td style="width: 15%;">PSW1</td> <td style="width: 15%;">PSW2</td> <td style="width: 15%;">R2</td> <td style="width: 25%;">X</td> </tr> <tr> <td></td> <td style="text-align: center;">1017</td> <td style="text-align: center;">0002</td> <td style="text-align: center;">0020</td> <td style="text-align: center;">0000</td> </tr> <tr> <td>AFTER EXECUTION:</td> <td>PSW1</td> <td>PSW2</td> <td>R2</td> <td>X</td> </tr> <tr> <td></td> <td style="text-align: center;">1018</td> <td style="text-align: center;">8002</td> <td style="text-align: center;">0020</td> <td style="text-align: center;">0020</td> </tr> </table> <p style="text-align: center;"><b>LDX3 Load X from R3</b></p> <div style="text-align: center;">  <p style="margin-left: 40px;">0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15</p> </div> <p>The contents of the R3 register replace the contents of the X register.</p> <p>EXAMPLE: Memory Location: 1018 Hex Instruction: 0B07</p> <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 30%;">BEFORE EXECUTION:</td> <td style="width: 15%;">PSW1</td> <td style="width: 15%;">PSW2</td> <td style="width: 15%;">R3</td> <td style="width: 25%;">X</td> </tr> <tr> <td></td> <td style="text-align: center;">1018</td> <td style="text-align: center;">B002</td> <td style="text-align: center;">38B2</td> <td style="text-align: center;">0020</td> </tr> <tr> <td>AFTER EXECUTION:</td> <td>PSW1</td> <td>PSW2</td> <td>R3</td> <td>X</td> </tr> <tr> <td></td> <td style="text-align: center;">1019</td> <td style="text-align: center;">8002</td> <td style="text-align: center;">38B2</td> <td style="text-align: center;">38B2</td> </tr> </table>	BEFORE EXECUTION:	PSW1	PSW2	R1	X		1016	8002	0000	0300	AFTER EXECUTION:	PSW1	PSW2	R1	X		1017	0002	0000	0000	BEFORE EXECUTION:	PSW1	PSW2	R2	X		1017	0002	0020	0000	AFTER EXECUTION:	PSW1	PSW2	R2	X		1018	8002	0020	0020	BEFORE EXECUTION:	PSW1	PSW2	R3	X		1018	B002	38B2	0020	AFTER EXECUTION:	PSW1	PSW2	R3	X		1019	8002	38B2	38B2
BEFORE EXECUTION:	PSW1	PSW2	R1	X																																																									
	1016	8002	0000	0300																																																									
AFTER EXECUTION:	PSW1	PSW2	R1	X																																																									
	1017	0002	0000	0000																																																									
BEFORE EXECUTION:	PSW1	PSW2	R2	X																																																									
	1017	0002	0020	0000																																																									
AFTER EXECUTION:	PSW1	PSW2	R2	X																																																									
	1018	8002	0020	0020																																																									
BEFORE EXECUTION:	PSW1	PSW2	R3	X																																																									
	1018	B002	38B2	0020																																																									
AFTER EXECUTION:	PSW1	PSW2	R3	X																																																									
	1019	8002	38B2	38B2																																																									

Basic  
Instruction

Derivative, Format, and/or Function

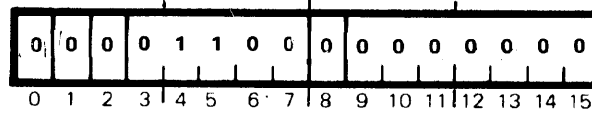
LDB - 0C

LDBA Load B from A



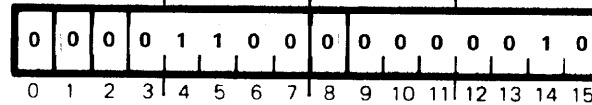
The contents of the A register replace the contents of the B register.

LDBD Load B from D



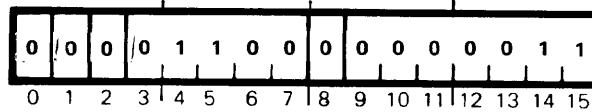
The contents of the D register replace the contents of the B register.

LDBE Load B from E



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

LDBX Load B from X



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

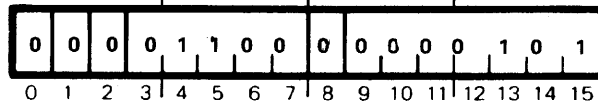


Basic  
Instruction

Derivative, Format, and/or Function

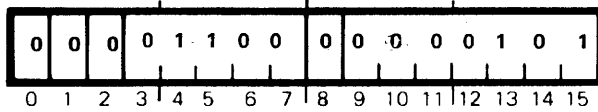
LDB - 0C  
(Cont'd)

LDB1 Load B from R1



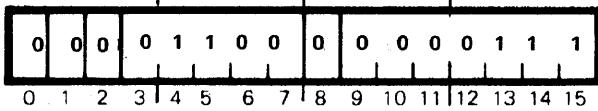
The contents of the R1 register replace the contents of the B register.

LDB2 Load B from R2



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

LDB3 Load B from R3

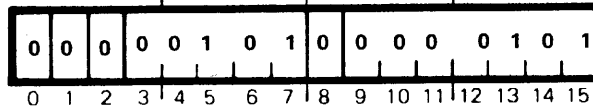


The contents of the R3 register replace the contents of the B register.

Basic Instruction	Derivative, Format, and/or Function																																																																																																																																
<p><b>STD - 05</b></p>	<p style="text-align: center;"><b>STDA Store D into A</b></p> <div style="text-align: center;"> <table border="1" style="margin: auto;"> <tr> <td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td> </tr> <tr> <td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td> </tr> </table> </div> <p>The contents of the D register replace the contents of the A register.</p> <p>Affected: A register</p> <p style="text-align: center;"><b>STDB Store D into B</b></p> <div style="text-align: center;"> <table border="1" style="margin: auto;"> <tr> <td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td> </tr> <tr> <td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td> </tr> </table> </div> <p>The contents of the D register replace the contents of the B register.</p> <p>Affected: B Register</p> <p style="text-align: center;"><b>STDE Store D into E</b></p> <div style="text-align: center;"> <table border="1" style="margin: auto;"> <tr> <td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td> </tr> <tr> <td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td> </tr> </table> </div> <p>The contents of the D register replace the contents of the E register.</p> <p>Affected: E Register</p> <p style="text-align: center;"><b>STDX Store D into X</b></p> <div style="text-align: center;"> <table border="1" style="margin: auto;"> <tr> <td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td> </tr> <tr> <td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td> </tr> </table> </div> <p>The contents of the D Register replace the contents of the X register.</p> <p>Affected: X Register</p>	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15																																																																																																																		
0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	1																																																																																																																		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15																																																																																																																		
0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	0																																																																																																																		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15																																																																																																																		
0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0																																																																																																																		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15																																																																																																																		
0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	1																																																																																																																		

STD - 05  
(cont'd)

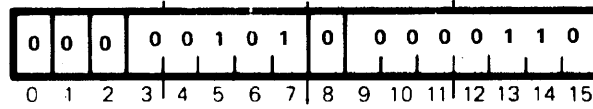
**STD1 Store D into R1**



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

Affected: R1 Register

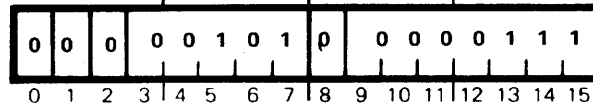
**STD2 Store D into R2**



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

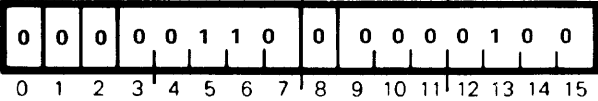
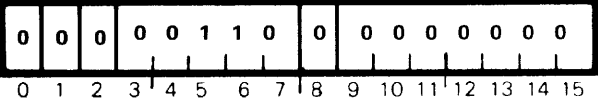
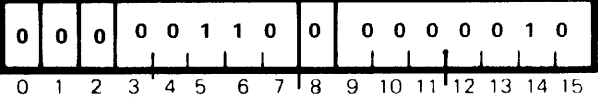
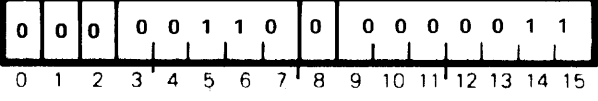
Affected: R2 Register

**STD3 Store D into R3**



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

Affected: R3 Register

Basic Instruction	Derivative, Format, and/or Function
STA - 06	<p data-bbox="625 289 831 310"><b>STAB Store A into B</b></p>  <p data-bbox="378 443 1019 464">The contents of the A register replace the contents of the B register.</p> <p data-bbox="378 495 578 516">Affected: B Register</p> <p data-bbox="625 611 831 632"><b>STAD Store A into D</b></p>  <p data-bbox="378 785 1019 806">The contents of the A register replace the contents of the D register.</p> <p data-bbox="378 837 578 858">Affected: D Register</p> <p data-bbox="625 942 831 963"><b>STAE Store A into E</b></p>  <p data-bbox="378 1104 1019 1125">The contents of the A register replace the contents of the E register.</p> <p data-bbox="378 1157 578 1178">Affected: E Register</p> <p data-bbox="625 1278 831 1299"><b>STAX Store A into X</b></p>  <p data-bbox="378 1442 1019 1463">The contents of the A register replace the contents of the X register.</p> <p data-bbox="378 1495 578 1516">Affected: X Register</p>

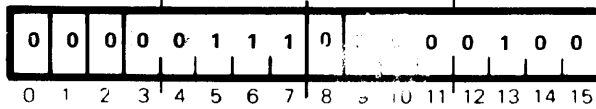
Basic Instruction	Derivative, Format, and/or Function																																																																																																
<p>STA - 06 (Cont'd)</p>	<p style="text-align: center;"><b>STA1 Store A into R1</b></p> <div style="text-align: center;"> <table border="1" style="margin: auto;"> <tr> <td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">1</td><td style="padding: 2px 5px;">1</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">1</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">1</td> </tr> <tr> <td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">2</td><td style="text-align: center;">3</td><td style="text-align: center;">4</td><td style="text-align: center;">5</td><td style="text-align: center;">6</td><td style="text-align: center;">7</td><td style="text-align: center;">8</td><td style="text-align: center;">9</td><td style="text-align: center;">10</td><td style="text-align: center;">11</td><td style="text-align: center;">12</td><td style="text-align: center;">13</td><td style="text-align: center;">14</td><td style="text-align: center;">15</td> </tr> </table> </div> <p>The contents of the A register replace the contents of the R1 register.</p> <p>Affected: R1 Register</p> <p style="text-align: center;"><b>STA2 Store A into R2</b></p> <div style="text-align: center;"> <table border="1" style="margin: auto;"> <tr> <td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">1</td><td style="padding: 2px 5px;">1</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">1</td><td style="padding: 2px 5px;">1</td><td style="padding: 2px 5px;">0</td> </tr> <tr> <td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">2</td><td style="text-align: center;">3</td><td style="text-align: center;">4</td><td style="text-align: center;">5</td><td style="text-align: center;">6</td><td style="text-align: center;">7</td><td style="text-align: center;">8</td><td style="text-align: center;">9</td><td style="text-align: center;">10</td><td style="text-align: center;">11</td><td style="text-align: center;">12</td><td style="text-align: center;">13</td><td style="text-align: center;">14</td><td style="text-align: center;">15</td> </tr> </table> </div> <p>The contents of the A register replace the contents of the R2 register.</p> <p>Affected : R2 Register</p> <p style="text-align: center;"><b>STA3 Store A into R3</b></p> <div style="text-align: center;"> <table border="1" style="margin: auto;"> <tr> <td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">1</td><td style="padding: 2px 5px;">1</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">1</td><td style="padding: 2px 5px;">1</td><td style="padding: 2px 5px;">1</td> </tr> <tr> <td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">2</td><td style="text-align: center;">3</td><td style="text-align: center;">4</td><td style="text-align: center;">5</td><td style="text-align: center;">6</td><td style="text-align: center;">7</td><td style="text-align: center;">8</td><td style="text-align: center;">9</td><td style="text-align: center;">10</td><td style="text-align: center;">11</td><td style="text-align: center;">12</td><td style="text-align: center;">13</td><td style="text-align: center;">14</td><td style="text-align: center;">15</td> </tr> </table> </div> <p>The contents of the A register replace the contents of the R3 register.</p> <p>Affected: R3 Register</p>	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	0	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	0	0	0	0	1	1	0	0	0	0	0	0	1	1	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	1																																																																																		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15																																																																																		
0	0	0	0	0	1	1	0	0	0	0	0	0	1	1	0																																																																																		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15																																																																																		
0	0	0	0	0	1	1	0	0	0	0	0	0	1	1	1																																																																																		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15																																																																																		

Basic Instruction

Derivative, Format, and/or Function

SBY-07

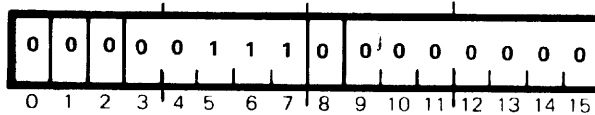
SBYB Store Byte from A into B



Bits 8 through 15 of the A register replace bits 0 through 7 of the B register, leaving bits 8 through 15 of the B register unaffected.

Affected: Bits 0 through 7 of the B register.

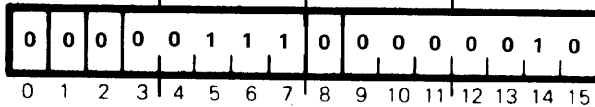
SBYD Store Byte from A into D



Bits 8 through 15 of the A register replace bits 0 through 7 of the D register, leaving bits 8 through 15 of the D register unaffected.

Affected: Bits 0 through 7 of the D register.

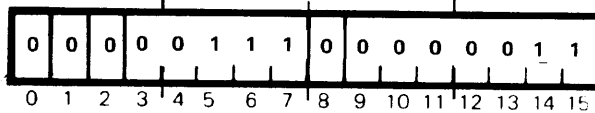
SBYE Store Byte from A into E



Bits 8 through 15 of the A register replace bits 0 through 7 of the E register, leaving bits 8 through 15 of the E register unaffected.

Affected: Bits 0 through 7 of the E register.

SBYX Store Byte from A into X



Bits 8 through 15 of the A register replace bits 0 through 7 of the X register, leaving bits 8 through 15 of the X register unaffected.

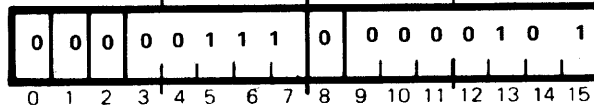
Affected: Bits 0 through 7 of the X register.

Basic  
Instruction

Derivative, Format, and/or Function

SBY-07  
(Cont'd)

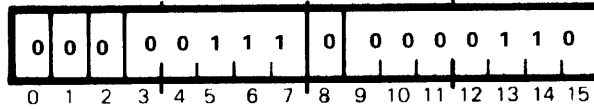
SBY1 Store Byte from A into R1



Bits 8 through 15 of the A register replace bits 0 through 7 of the R1 register, leaving bits 8 through 15 of the R1 register unaffected.

Affected: Bits 0 through 7 of the R1 register.

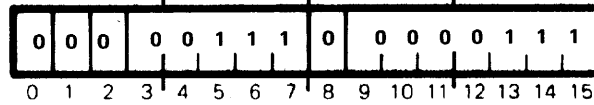
SBY2 Store Byte from A into R2



Bits 8 through 15 of the A register replace bits 0 through 7 of the R2 register, leaving bits 8 through 15 of the R2 register unaffected.

Affected: Bits 0 through 7 of the R2 register.

SBY3 Store Byte from A into R3



Bits 8 through 15 of the A register replace bits 0 through 7 of the R3 register, leaving bits 8 through 15 of the R3 register unaffected.

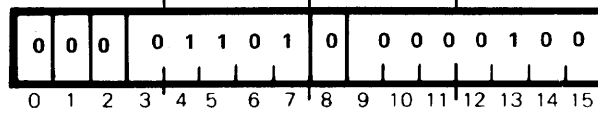
Affected: Bits 0 through 7 of the R3 register.

Basic  
Instruction

Derivative, Format, and/or Function

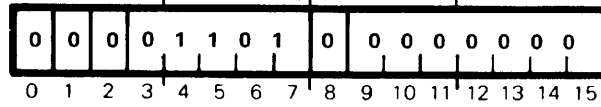
ADD - 0D

ADDB Add B to A



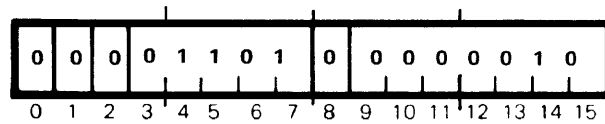
The contents of the B register plus the contents of the A register replace the contents of the A register.

ADDD Add D to A



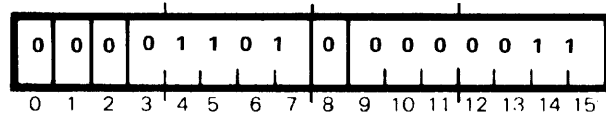
The contents of the D register plus the contents of the A register replace the contents of the A register.

ADDE Add E to A



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



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



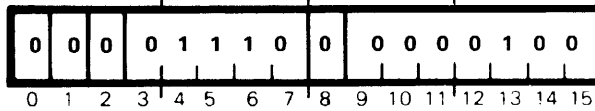
Basic Instruction	Derivative, Format, and/or Function																																																																																																
<b>ADD-0D</b> (Cont'd)	<div data-bbox="711 296 911 323" style="text-align: center;"> <b>ADD1 Add R1 to A</b> </div> <div data-bbox="691 323 1284 428" style="text-align: center;"> <table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <tr> <td style="width: 33.33%;">0</td><td style="width: 33.33%;">0</td><td style="width: 33.33%;">0</td> <td style="width: 33.33%;">0</td><td style="width: 33.33%;">1</td><td style="width: 33.33%;">1</td><td style="width: 33.33%;">0</td><td style="width: 33.33%;">1</td> <td style="width: 33.33%;">0</td><td style="width: 33.33%;">0</td><td style="width: 33.33%;">0</td><td style="width: 33.33%;">0</td><td style="width: 33.33%;">0</td><td style="width: 33.33%;">1</td><td style="width: 33.33%;">0</td><td style="width: 33.33%;">1</td> </tr> <tr> <td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td> </tr> </table> </div> <p data-bbox="459 453 1435 483" style="text-align: center;">The contents of the R1 register plus the contents of the A register replace the contents of the A register.</p> <div data-bbox="711 623 911 651" style="text-align: center;"> <b>ADD2 Add R2 to A</b> </div> <div data-bbox="691 651 1284 756" style="text-align: center;"> <table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <tr> <td style="width: 33.33%;">0</td><td style="width: 33.33%;">0</td><td style="width: 33.33%;">0</td> <td style="width: 33.33%;">0</td><td style="width: 33.33%;">1</td><td style="width: 33.33%;">1</td><td style="width: 33.33%;">0</td><td style="width: 33.33%;">1</td> <td style="width: 33.33%;">0</td><td style="width: 33.33%;">0</td><td style="width: 33.33%;">0</td><td style="width: 33.33%;">0</td><td style="width: 33.33%;">0</td><td style="width: 33.33%;">1</td><td style="width: 33.33%;">1</td><td style="width: 33.33%;">0</td> </tr> <tr> <td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td> </tr> </table> </div> <p data-bbox="459 800 1435 829" style="text-align: center;">The contents of the R2 register plus the contents of the A register replace the contents of the A register.</p> <div data-bbox="711 951 911 978" style="text-align: center;"> <b>ADD3 Add R3 to A</b> </div> <div data-bbox="691 978 1284 1083" style="text-align: center;"> <table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <tr> <td style="width: 33.33%;">0</td><td style="width: 33.33%;">0</td><td style="width: 33.33%;">0</td> <td style="width: 33.33%;">0</td><td style="width: 33.33%;">1</td><td style="width: 33.33%;">1</td><td style="width: 33.33%;">0</td><td style="width: 33.33%;">1</td> <td style="width: 33.33%;">0</td><td style="width: 33.33%;">0</td><td style="width: 33.33%;">0</td><td style="width: 33.33%;">0</td><td style="width: 33.33%;">0</td><td style="width: 33.33%;">1</td><td style="width: 33.33%;">1</td><td style="width: 33.33%;">1</td> </tr> <tr> <td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td> </tr> </table> </div> <p data-bbox="459 1106 1435 1136" style="text-align: center;">The contents of the R3 register plus the contents of the A register replace the contents of the A register.</p>	0	0	0	0	1	1	0	1	0	0	0	0	0	1	0	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	0	0	0	1	1	0	1	0	0	0	0	0	1	1	0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	0	0	0	1	1	0	1	0	0	0	0	0	1	1	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	1	0	1	0	0	0	0	0	1	0	1																																																																																		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15																																																																																		
0	0	0	0	1	1	0	1	0	0	0	0	0	1	1	0																																																																																		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15																																																																																		
0	0	0	0	1	1	0	1	0	0	0	0	0	1	1	1																																																																																		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15																																																																																		

Basic  
Instruction

Derivative, Format, and/or Function

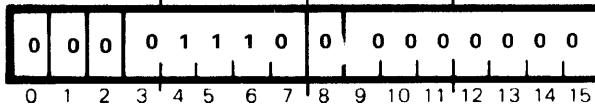
SUB - 0E

**SUBB Subtract B from A**



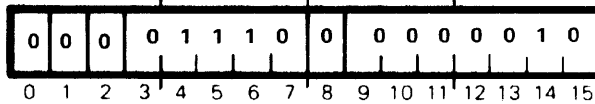
The contents of the A register minus the contents of the B register replace the contents of the A register.

**SUBD Subtract D from A**



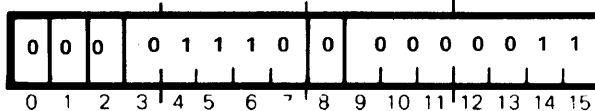
The contents of the A register minus the contents of the D register replace the contents of the A register.

**SUBE Subtract E from A**



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**



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

Basic Instruction	Derivative, Format, and/or Function																																																																																																
<p><b>SUB - 0E</b> (Cont'd)</p>	<p style="text-align: center;"><b>SUB1 Subtract R1 from A</b></p> <div style="text-align: center;"> <table border="1" style="margin: auto;"> <tr> <td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">1</td><td style="padding: 2px 5px;">1</td><td style="padding: 2px 5px;">1</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">1</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">1</td> </tr> <tr> <td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">1</td><td style="padding: 2px 5px;">2</td><td style="padding: 2px 5px;">3</td><td style="padding: 2px 5px;">4</td><td style="padding: 2px 5px;">5</td><td style="padding: 2px 5px;">6</td><td style="padding: 2px 5px;">7</td><td style="padding: 2px 5px;">8</td><td style="padding: 2px 5px;">9</td><td style="padding: 2px 5px;">10</td><td style="padding: 2px 5px;">11</td><td style="padding: 2px 5px;">12</td><td style="padding: 2px 5px;">13</td><td style="padding: 2px 5px;">14</td><td style="padding: 2px 5px;">15</td> </tr> </table> </div> <p style="text-align: center;">The contents of the A register minus the contents of the R1 register replace the contents of the A register.</p> <p style="text-align: center;"><b>SUB2 Subtract R2 from A</b></p> <div style="text-align: center;"> <table border="1" style="margin: auto;"> <tr> <td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">1</td><td style="padding: 2px 5px;">1</td><td style="padding: 2px 5px;">1</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">1</td><td style="padding: 2px 5px;">1</td><td style="padding: 2px 5px;">0</td> </tr> <tr> <td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">1</td><td style="padding: 2px 5px;">2</td><td style="padding: 2px 5px;">3</td><td style="padding: 2px 5px;">4</td><td style="padding: 2px 5px;">5</td><td style="padding: 2px 5px;">6</td><td style="padding: 2px 5px;">7</td><td style="padding: 2px 5px;">8</td><td style="padding: 2px 5px;">9</td><td style="padding: 2px 5px;">10</td><td style="padding: 2px 5px;">11</td><td style="padding: 2px 5px;">12</td><td style="padding: 2px 5px;">13</td><td style="padding: 2px 5px;">14</td><td style="padding: 2px 5px;">15</td> </tr> </table> </div> <p style="text-align: center;">The contents of the A register minus the contents of the R2 register replace the contents of the A register.</p> <p style="text-align: center;"><b>SUB3 Subtract R3 from A</b></p> <div style="text-align: center;"> <table border="1" style="margin: auto;"> <tr> <td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">1</td><td style="padding: 2px 5px;">1</td><td style="padding: 2px 5px;">1</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">1</td><td style="padding: 2px 5px;">1</td><td style="padding: 2px 5px;">1</td> </tr> <tr> <td style="padding: 2px 5px;">0</td><td style="padding: 2px 5px;">1</td><td style="padding: 2px 5px;">2</td><td style="padding: 2px 5px;">3</td><td style="padding: 2px 5px;">4</td><td style="padding: 2px 5px;">5</td><td style="padding: 2px 5px;">6</td><td style="padding: 2px 5px;">7</td><td style="padding: 2px 5px;">8</td><td style="padding: 2px 5px;">9</td><td style="padding: 2px 5px;">10</td><td style="padding: 2px 5px;">11</td><td style="padding: 2px 5px;">12</td><td style="padding: 2px 5px;">13</td><td style="padding: 2px 5px;">14</td><td style="padding: 2px 5px;">15</td> </tr> </table> </div> <p style="text-align: center;">The contents of the A register minus the contents of the R3 register replace the contents of the A register.</p>	0	0	0	0	1	1	1	0	0	0	0	0	0	1	0	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	0	0	0	1	1	1	0	0	0	0	0	0	1	1	0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	0	0	0	1	1	1	0	0	0	0	0	0	1	1	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	1	1	0	0	0	0	0	0	1	0	1																																																																																		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15																																																																																		
0	0	0	0	1	1	1	0	0	0	0	0	0	1	1	0																																																																																		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15																																																																																		
0	0	0	0	1	1	1	0	0	0	0	0	0	1	1	1																																																																																		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15																																																																																		

Basic Instruction	Derivative, Format, and/or Function																																																																																																																																
INC - OF	<p data-bbox="630 306 818 331"><b>INCA Increment A</b></p> <div data-bbox="610 348 1206 453"> <table border="1"> <tr> <td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td> </tr> <tr> <td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td> </tr> </table> </div> <p data-bbox="370 478 808 504">The contents of the A register increase by one.</p> <p data-bbox="370 529 740 609">Affected:           A Register                       CC1                       CC2</p> <p data-bbox="630 642 818 667"><b>INCB Increment B</b></p> <div data-bbox="610 684 1206 789"> <table border="1"> <tr> <td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td> </tr> <tr> <td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td> </tr> </table> </div> <p data-bbox="370 814 808 840">The contents of the B register increase by one.</p> <p data-bbox="370 865 740 945">Affected:           B Register                       CC1                       CC2</p> <p data-bbox="630 999 818 1024"><b>INCD Increment D</b></p> <div data-bbox="610 1041 1206 1146"> <table border="1"> <tr> <td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td> </tr> <tr> <td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td> </tr> </table> </div> <p data-bbox="370 1171 808 1197">The contents of the D register increase by one.</p> <p data-bbox="370 1222 740 1302">Affected:           D Register                       CC1                       CC2</p> <p data-bbox="630 1335 818 1360"><b>INCE Increment E</b></p> <div data-bbox="610 1377 1206 1482"> <table border="1"> <tr> <td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td> </tr> <tr> <td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td> </tr> </table> </div> <p data-bbox="370 1507 808 1533">The contents of the E register increase by one.</p> <p data-bbox="370 1558 740 1638">Affected:           E Register                       CC1                       CC2</p>	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1																																																																																																																		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15																																																																																																																		
0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	0																																																																																																																		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15																																																																																																																		
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0																																																																																																																		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15																																																																																																																		
0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	0																																																																																																																		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15																																																																																																																		

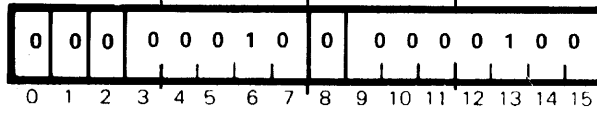
Basic Instruction	Derivative, Format, and/or Function																																																																																																																																
<p><b>INC - OF</b> (Cont'd)</p>	<div data-bbox="732 285 1328 422"> <p><b>INCX Increment X</b></p> <table border="1"> <tr> <td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td> </tr> <tr> <td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td> </tr> </table> </div> <p>The contents of the X register increase by one.</p> <p>Affected: X Register CC1 CC2</p> <div data-bbox="732 615 1328 751"> <p><b>INC1 Increment R1</b></p> <table border="1"> <tr> <td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td> </tr> <tr> <td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td> </tr> </table> </div> <p>The contents of the R1 register increase by one.</p> <p>Affected: R1 Register CC1 CC2</p> <div data-bbox="732 951 1328 1087"> <p><b>INC2 Increment R2</b></p> <table border="1"> <tr> <td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td> </tr> <tr> <td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td> </tr> </table> </div> <p>The contents of the R2 register increase by one.</p> <p>Affected: R2 Register CC1 CC2</p> <div data-bbox="732 1297 1328 1434"> <p><b>INC3 Increment R3</b></p> <table border="1"> <tr> <td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td> </tr> <tr> <td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td> </tr> </table> </div> <p>The contents of the R3 register increase by one.</p> <p>Affected: R3 Register CC1 CC2</p>	0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	0	0	0	1	1	1	1	0	0	0	0	0	1	1	0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	0	0	0	1	1	1	1	0	0	0	0	0	1	1	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	1																																																																																																																		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15																																																																																																																		
0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	1																																																																																																																		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15																																																																																																																		
0	0	0	0	1	1	1	1	0	0	0	0	0	1	1	0																																																																																																																		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15																																																																																																																		
0	0	0	0	1	1	1	1	0	0	0	0	0	1	1	1																																																																																																																		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15																																																																																																																		

**Basic Instruction**

**Derivative, Format, and/or Function**

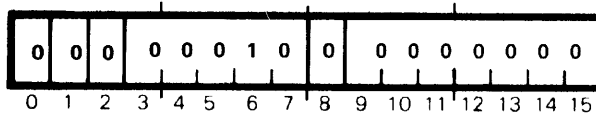
**AND - 02**

**ANDB AND B into A**



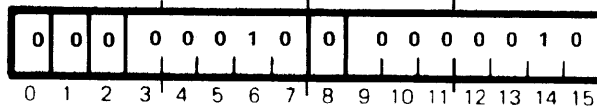
The logical product of the contents of the A register and the contents of the B register replaces the contents of the A register.

**ANDD AND D into A**



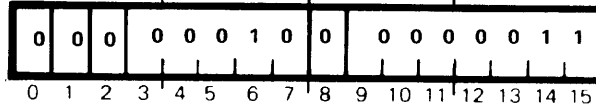
The logical product of the contents of the A register and the contents of the D register replaces the contents of the A register.

**ANDE ANDE into A**



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**



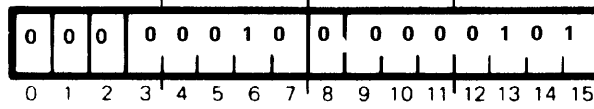
The logical product of the contents of the A register and the contents of the X register replaces the contents of the A register.

Basic  
Instruction

Derivative, Format, and/or Function

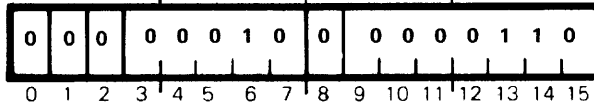
AND - 02  
(Cont'd)

AND1 AND R1 into A



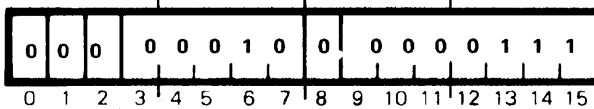
The logical product of the contents of the A register and the contents of the R1 register replaces the contents of the A register.

AND2 AND R2 into A



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



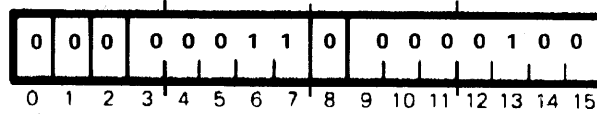
The logical product of the contents of the A register and the contents of the R3 register replaces the contents of the A register.

**Basic  
Instruction**

**Derivative, Format, and/or Function**

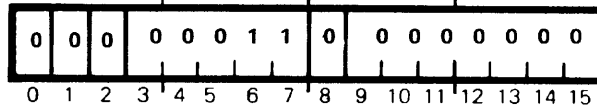
**LOR - 03**

**LORB OR B into A**



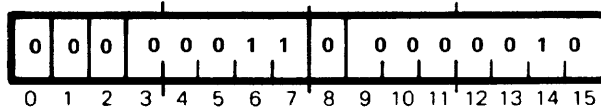
The logical sum of the contents of the B register and the contents of the A register replaces the contents of the A register.

**LORD OR D into A**



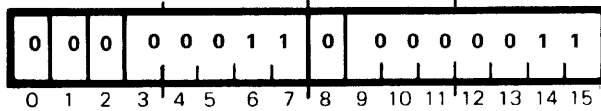
The logical sum of the contents of the D register and the contents of the A register replaces the contents of the A register.

**LORE OR E into A**



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**



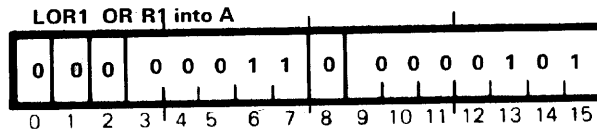
The logical sum of the contents of the X register and the contents of the A register replaces the contents of the A register.



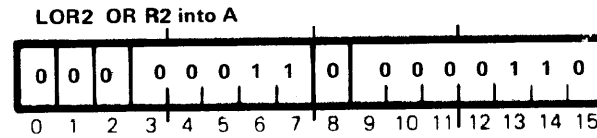
Basic Instruction

Derivative, Format, and/or Function

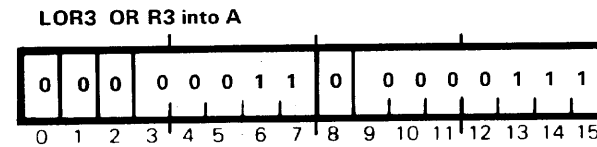
LOR-03  
(Cont'd)



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



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



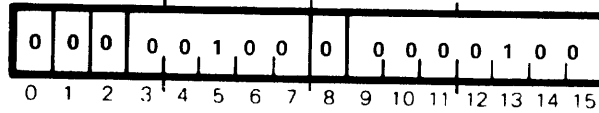
The logical sum of the contents of the R3 register and the contents of the A register replaces the contents of the A register.

Basic Instruction

Derivative, Format, and/or Function

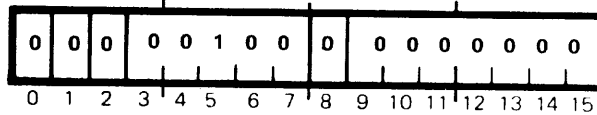
EOR - 04

EORB Exclusive OR B into A



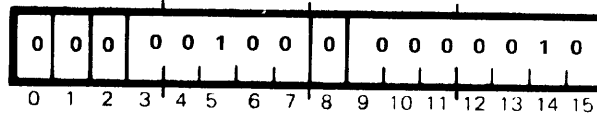
The logical difference of the contents of the B register and the contents of the A register replaces the contents of the A register.

EORD Exclusive OR D into A



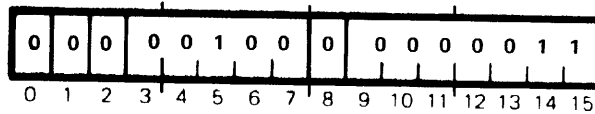
The logical difference of the contents of the D register and the contents of the A register replaces the contents of the A register.

EORE Exclusive OR E into A



The logical difference of the contents of the E register and the contents of the A register replaces the content of the A register.

EORX Exclusive OR X into A



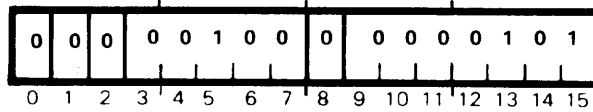
The logical difference of the contents of the X register and the contents of the A register replaces the contents of the A register.

Basic  
Instruction

Derivative, Format, and/or Function

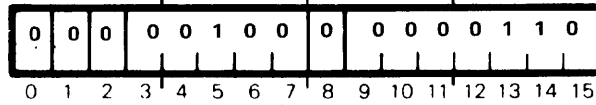
EOR - 04  
(Cont'd)

EOR1 Exclusive OR R1 into A



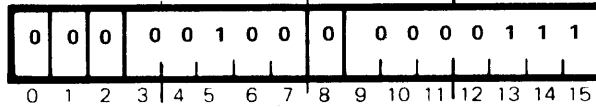
The logical difference of the contents of the R1 register and the contents of the A register replaces the contents of the A register.

EOR2 Exclusive OR R2 into A



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



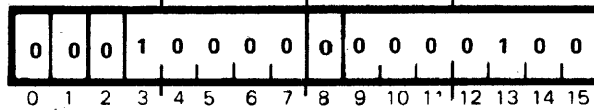
The logical difference of the contents of the R3 register and the contents of the A register replaces the contents of the A register.

Basic Instruction

Derivative, Format, and/or Function

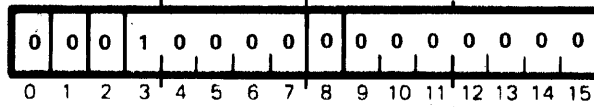
CMP - 10

CMPB Compare A with B



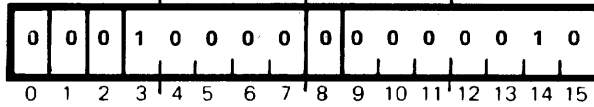
The condition code indicators depict the comparison between the contents of the A register and the contents of the B register.

CMPD Compare A with D



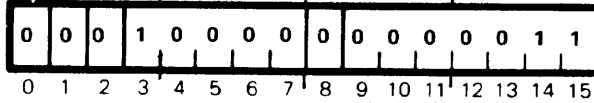
The condition code indicators depict the comparison between the contents of the A register and the contents of the D register.

CMPE Compare A with E



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



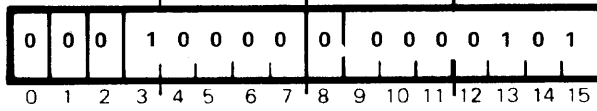
The condition code indicators depict the comparison between the contents of the A register and the contents of the X register.

Basic Instruction

Derivative, Format, and/or Function

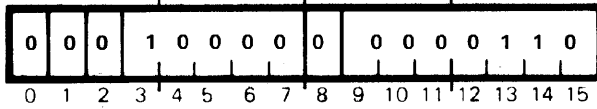
CMP-10  
(Cont'd)

CMP1 Compare A with R1



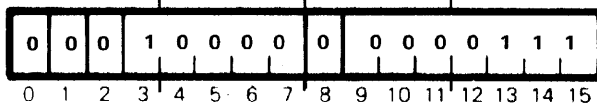
The condition code indicators depict the comparison between the contents of the A register and the contents of the R1 register.

CMP2 Compare A with R2

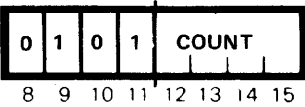
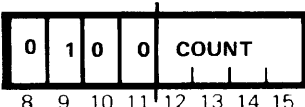
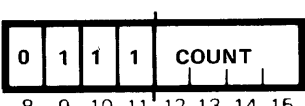
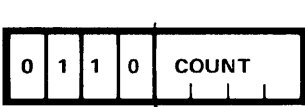
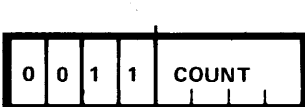
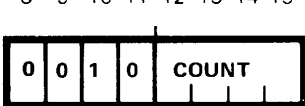
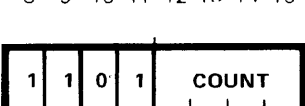
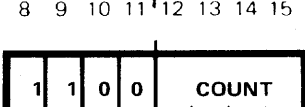
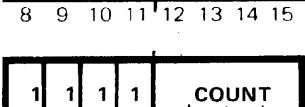
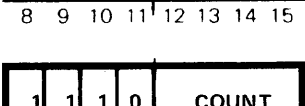
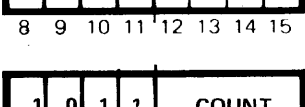
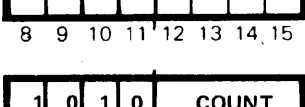


The condition code indicators depict the comparison between the contents of the A register and the contents of the R2 register.

CMP3 Compare A with R3



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

Basic Instruction	Derivative, Format, and/or Function		
S - 11	<u>Mnemonic</u>	<u>Meaning</u>	<u>Bits 8-15 of Effective Address</u>
	SLL	Shift Logical Left	
	SLR	Shift Logical Right	
	SAL	Shift Arithmetic Left	
	SAR	Shift Arithmetic Right	
	SCL	Shift Circular Left	
	SCR	Shift Circular Right	
	SLLD	Shift Logical Left Double	
	SLRD	Shift Logical Right Double	
	SALD	Shift Arithmetic Left Double	
	SARD	Shift Arithmetic Right Double	
	SCLD	Shift Circular Left Double	
	SCRD	Shift Circular Right Double	
* COUNT = Number of places shifted.			

Basic Instruction	Derivative, Format, and/or Function																																																																																																																																																																
<p>POT - 00</p>	<p style="text-align: center;"><b>SIO Start Input/Output</b></p> <table border="1" style="margin: auto;"> <tr> <td style="width: 15px; text-align: center;">1</td> <td style="width: 15px; text-align: center;">1</td> <td style="width: 15px; text-align: center;">0</td> <td style="width: 15px; text-align: center;">0</td> <td style="width: 15px; text-align: center;">0</td> <td style="width: 15px; text-align: center;">0</td> <td style="width: 15px; text-align: center;">0</td> <td style="width: 15px; text-align: center;">0</td> <td style="width: 15px; text-align: center;">0</td> <td style="width: 15px; text-align: center;">S</td> <td colspan="6" style="text-align: center;">Displacement</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td style="text-align: center;">2</td> <td style="text-align: center;">3</td> <td style="text-align: center;">4</td> <td style="text-align: center;">5</td> <td style="text-align: center;">6</td> <td style="text-align: center;">7</td> <td style="text-align: center;">8</td> <td style="text-align: center;">9</td> <td style="text-align: center;">10</td> <td style="text-align: center;">11</td> <td style="text-align: center;">12</td> <td style="text-align: center;">13</td> <td style="text-align: center;">14</td> <td style="text-align: center;">15</td> </tr> </table> <p>Effective Address *: 0000 0001 00AA AAAA</p> <p style="text-align: center;"><b>TIO Test Input/Output</b></p> <table border="1" style="margin: auto;"> <tr> <td style="width: 15px; text-align: center;">1</td> <td style="width: 15px; text-align: center;">1</td> <td style="width: 15px; text-align: center;">0</td> <td style="width: 15px; text-align: center;">0</td> <td style="width: 15px; text-align: center;">0</td> <td style="width: 15px; text-align: center;">0</td> <td style="width: 15px; text-align: center;">0</td> <td style="width: 15px; text-align: center;">0</td> <td style="width: 15px; text-align: center;">0</td> <td style="width: 15px; text-align: center;">S</td> <td colspan="6" style="text-align: center;">Displacement</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td style="text-align: center;">2</td> <td style="text-align: center;">3</td> <td style="text-align: center;">4</td> <td style="text-align: center;">5</td> <td style="text-align: center;">6</td> <td style="text-align: center;">7</td> <td style="text-align: center;">8</td> <td style="text-align: center;">9</td> <td style="text-align: center;">10</td> <td style="text-align: center;">11</td> <td style="text-align: center;">12</td> <td style="text-align: center;">13</td> <td style="text-align: center;">14</td> <td style="text-align: center;">15</td> </tr> </table> <p>Effective Address *: 0000 0001 10AA AAAA</p> <p style="text-align: center;"><b>HIO Halt Input/Output</b></p> <table border="1" style="margin: auto;"> <tr> <td style="width: 15px; text-align: center;">1</td> <td style="width: 15px; text-align: center;">1</td> <td style="width: 15px; text-align: center;">0</td> <td style="width: 15px; text-align: center;">0</td> <td style="width: 15px; text-align: center;">0</td> <td style="width: 15px; text-align: center;">0</td> <td style="width: 15px; text-align: center;">0</td> <td style="width: 15px; text-align: center;">0</td> <td style="width: 15px; text-align: center;">0</td> <td style="width: 15px; text-align: center;">S</td> <td colspan="6" style="text-align: center;">Displacement</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td style="text-align: center;">2</td> <td style="text-align: center;">3</td> <td style="text-align: center;">4</td> <td style="text-align: center;">5</td> <td style="text-align: center;">6</td> <td style="text-align: center;">7</td> <td style="text-align: center;">8</td> <td style="text-align: center;">9</td> <td style="text-align: center;">10</td> <td style="text-align: center;">11</td> <td style="text-align: center;">12</td> <td style="text-align: center;">13</td> <td style="text-align: center;">14</td> <td style="text-align: center;">15</td> </tr> </table> <p>Effective Address *: 0000 0001 01AA AAAA</p> <p style="text-align: center;"><b>IOR Input/Output Reset</b></p> <table border="1" style="margin: auto;"> <tr> <td style="width: 15px; text-align: center;">1</td> <td style="width: 15px; text-align: center;">1</td> <td style="width: 15px; text-align: center;">0</td> <td style="width: 15px; text-align: center;">0</td> <td style="width: 15px; text-align: center;">0</td> <td style="width: 15px; text-align: center;">0</td> <td style="width: 15px; text-align: center;">0</td> <td style="width: 15px; text-align: center;">0</td> <td style="width: 15px; text-align: center;">0</td> <td style="width: 15px; text-align: center;">S</td> <td colspan="6" style="text-align: center;">Displacement</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td style="text-align: center;">2</td> <td style="text-align: center;">3</td> <td style="text-align: center;">4</td> <td style="text-align: center;">5</td> <td style="text-align: center;">6</td> <td style="text-align: center;">7</td> <td style="text-align: center;">8</td> <td style="text-align: center;">9</td> <td style="text-align: center;">10</td> <td style="text-align: center;">11</td> <td style="text-align: center;">12</td> <td style="text-align: center;">13</td> <td style="text-align: center;">14</td> <td style="text-align: center;">15</td> </tr> </table> <p>Effective Address *: 0000 0100 0000 0011</p> <p style="text-align: center;"><b>HLT Program Halt</b></p> <table border="1" style="margin: auto;"> <tr> <td style="width: 15px; text-align: center;">1</td> <td style="width: 15px; text-align: center;">1</td> <td style="width: 15px; text-align: center;">0</td> <td style="width: 15px; text-align: center;">0</td> <td style="width: 15px; text-align: center;">0</td> <td style="width: 15px; text-align: center;">0</td> <td style="width: 15px; text-align: center;">0</td> <td style="width: 15px; text-align: center;">0</td> <td style="width: 15px; text-align: center;">0</td> <td style="width: 15px; text-align: center;">S</td> <td colspan="6" style="text-align: center;">Displacement</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td style="text-align: center;">2</td> <td style="text-align: center;">3</td> <td style="text-align: center;">4</td> <td style="text-align: center;">5</td> <td style="text-align: center;">6</td> <td style="text-align: center;">7</td> <td style="text-align: center;">8</td> <td style="text-align: center;">9</td> <td style="text-align: center;">10</td> <td style="text-align: center;">11</td> <td style="text-align: center;">12</td> <td style="text-align: center;">13</td> <td style="text-align: center;">14</td> <td style="text-align: center;">15</td> </tr> </table> <p>Effective Address *: 0000 0100 0000 0001</p> <p>* A = six bit device address</p>	1	1	0	0	0	0	0	0	0	S	Displacement						0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	1	1	0	0	0	0	0	0	0	S	Displacement						0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	1	1	0	0	0	0	0	0	0	S	Displacement						0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	1	1	0	0	0	0	0	0	0	S	Displacement						0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	1	1	0	0	0	0	0	0	0	S	Displacement						0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	0	0	0	0	0	0	0	S	Displacement																																																																																																																																																							
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15																																																																																																																																																		
1	1	0	0	0	0	0	0	0	S	Displacement																																																																																																																																																							
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15																																																																																																																																																		
1	1	0	0	0	0	0	0	0	S	Displacement																																																																																																																																																							
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15																																																																																																																																																		
1	1	0	0	0	0	0	0	0	S	Displacement																																																																																																																																																							
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15																																																																																																																																																		
1	1	0	0	0	0	0	0	0	S	Displacement																																																																																																																																																							
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15																																																																																																																																																		

Basic Instruction	Derivative, Format, and/or Function																																
PIN - 01	<p style="text-align: center;"><b>TDV Test Device</b></p> <table border="1" style="margin: auto;"> <tr> <td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">S</td><td colspan="6" style="text-align: center;">Displacement</td> </tr> <tr> <td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">2</td><td style="text-align: center;">3</td><td style="text-align: center;">4</td><td style="text-align: center;">5</td><td style="text-align: center;">6</td><td style="text-align: center;">7</td><td style="text-align: center;">8</td><td style="text-align: center;">9</td><td style="text-align: center;">10</td><td style="text-align: center;">11</td><td style="text-align: center;">12</td><td style="text-align: center;">13</td><td style="text-align: center;">14</td><td style="text-align: center;">15</td> </tr> </table> <p>Effective Address * 0000 0000 11AA AAAA</p> <p>* A = six bit device address</p>	1	1	0	0	0	0	0	0	1	S	Displacement						0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	0	0	0	0	0	0	1	S	Displacement																							
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15																		
B - 12	<p style="text-align: center;"><b>NOP No Operation</b></p> <table border="1" style="margin: auto;"> <tr> <td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td> </tr> <tr> <td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">2</td><td style="text-align: center;">3</td><td style="text-align: center;">4</td><td style="text-align: center;">5</td><td style="text-align: center;">6</td><td style="text-align: center;">7</td><td style="text-align: center;">8</td><td style="text-align: center;">9</td><td style="text-align: center;">10</td><td style="text-align: center;">11</td><td style="text-align: center;">12</td><td style="text-align: center;">13</td><td style="text-align: center;">14</td><td style="text-align: center;">15</td> </tr> </table> <p><b>Affected: None</b></p> <p>PSW1 advances normally, but no addressable register, memory location, or condition code indicator is affected.</p>	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1																		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15																		
EOR - 04	<p style="text-align: center;"><b>CLA Clear A Register</b></p> <table border="1" style="margin: auto;"> <tr> <td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td> </tr> <tr> <td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">2</td><td style="text-align: center;">3</td><td style="text-align: center;">4</td><td style="text-align: center;">5</td><td style="text-align: center;">6</td><td style="text-align: center;">7</td><td style="text-align: center;">8</td><td style="text-align: center;">9</td><td style="text-align: center;">10</td><td style="text-align: center;">11</td><td style="text-align: center;">12</td><td style="text-align: center;">13</td><td style="text-align: center;">14</td><td style="text-align: center;">15</td> </tr> </table> <p><b>Affected: A Register</b> CC1 CC2</p> <p>All 16 bit positions of the A register, and condition codes 1 and 2, are reset.</p>	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1																		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15																		



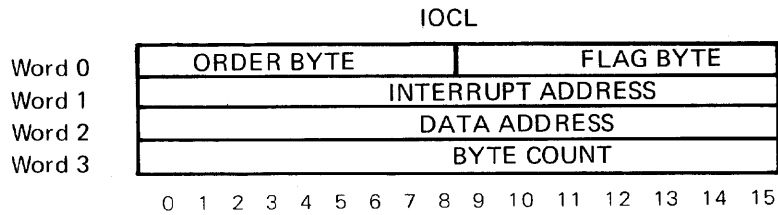
## APPENDIX C

### SAMPLE INPUT/OUTPUT PROGRAM

This appendix contains information necessary to program the standard SYSTEMS peripheral devices provided with the SYSTEMS 72. Information defining the four-word input/output command list (IOCL) that must be set up by the user is provided first, followed by a sample program that illustrates the construction of the IOCL tables and the data/command chaining operations.

#### INPUT/OUTPUT COMMAND LIST

The four-word input/output command list (IOCL) used by the PIOP to control I/O data transfers between the CPU and a device controller is illustrated below.



#### Word 0

The high-order byte, bits 0 through 7, of word 0 contains the order byte, which specifies the operation to be performed by the peripheral device. The low-order byte, bits 8 through 15, contains the flag byte, which specifies details of the transfer and the method of terminating the transfer.

The significance of the bits in the order and flag bytes is indicated below:

#### Order Byte:

##### Bit Configuration

<u>0</u>	<u>1</u>	<u>2</u>	<u>3</u>	<u>4</u>	<u>5</u>	<u>6</u>	<u>7</u>
M	M	M	M	M	M	0	1
M	M	M	M	M	M	1	0
M	M	M	M	M	M	1	1
0	0	0	0	0	0	0	0

##### Device Order

Write  
Read  
Control  
Halt

M = Bits used to  
modify the basic  
device order

### Flag Byte :

<u>Flag Bit</u>	<u>Meaning If Set</u>
8	Map data addresses and terminal interrupt address
9	Interrupt on Zero Byte Count
10	Halt on Transmission Error
11	Suppress Incorrect Length
12	Interrupt on Unusual End
13	Command Chain
14	Data Chain
15	Interrupt on Channel End

### Word 1

Word 1 of the IOCL contains the terminal interrupt address. In terminating an I/O operation, the device controller interrupts to this address, which contains the interrupt pointer.

### Word 2

Word 2 of the IOCL contains the data address, which points to the first word of data block. The first byte is assumed to be byte 0, which is the left-hand byte.

### Word 3

Word 3 of the IOCL contains a byte count that indicates the number of bytes to be transferred.

## SAMPLE PROGRAM

The Sample Program uses the teletype as an input/output device. Input/output command lists (IOCL's) in the sample program are designed to output a message to the teletype for printout, or input a message from the teletype keyboard for subsequent echo printout on the teletype. The first part of the program outputs a message and data chains on a carriage return/line feed (CR/LF). The second part of the program is for command chaining and allows a message of up to 20 characters to be input from the keyboard, echos the message back to the teletype for printout, and data chains on a CR/LF. The program then loops back through the command chain example.

The first four-word IOCL table for data chaining starts at line 14, and a second table begins at line 18. The second table contains an interrupt address labeled INTA, which transfers control to the interrupt processor starting in line 26. The interrupt processor processes the channel end interrupt, which occurs after all mechanical motion resulting from a carriage return and line feed has stopped.

The main program starts on line 46, which also specifies the message to be transferred to the teletype for printout. At the completion of the message, the program branches to the command chain example in line 96. This example is processed by the IOCL's starting at line 79 and allows the user to input a message from the teletype keyboard. The message is then echoed back for printout on the teletype.

EXAMPLE 1. IOCL COMMAND AND DATA CHAIN

		1			
		2	*		
		3	*****EQUATES*****		
		4	*		
	0080	5	INTMAPED EQU X'80'		INTERRUPT MAPPED
	0100	6	PRINT EQU X'100'		WRITE ORDER
	0002	7	DATACH EQU 2		DATA CHAIN
	0001	8	CHEND EQU 1		INT ON CHANNEL END
0000	0000	9	CFLAG DATA 0		
0001	0008	10	TYADR DATA 8		ADDRESS OF TELETYPE
		11	*		
		12	*****IOCL FOR DATA CHAIN*****		
		13	*		
0002	0102	14	IOCLDCH DATA PRINT + DATACH		
0003	0000	15	DATA 0		NO INTERRUPT WAS REQUESTED
0004	0000	16	MSGADR DATA 0		LOCATION OF MESSAGE
0005	0000	17	MSGLEN DATA 0		LENGHT OF MESSAGE
0006	0181	18	IOCLCRLF DATA PRINT+INTMAPED+CHEND		
0007	0000 *	19	DATA INTA		POINTER TO INTERRUPT ADDRESS POINTER
0008	0000 *	20	DATA CRLF		LOCATION OF CR AND LF
0009	0002	21	DATA 2		LENGTH OF 2 CHARACTERS
		22	*		
000A	0D0A	23	CRLF DATA X'D0A'		CARRIAGE RETURN LINE FEED

EXAMPLE 2. INTERRUPT PROCESSOR

		24			
		25	*		
000B	0000 *	26	INTA DATA INT		POINTER TO INTERRUPT LOCATION
000C	0000	27	INT DATA 0.0		OLD PSW1 AND PSW2
000D	0000				
000E	0010	28	DATA \$+2		NEW PSW1
000F	0002	29	DATA 2		MAPPED PSW2
		30	*		
0010	8BF1	31	LDX TYADR		GET TELETYPE ADDRESS
0011	E100 *	32	PIN X'C0:1		TEST DEVICE STATUS
0012	9201	33	NOP		.
0013	9201	34	NOP		.
0014	9201	35	NOP		CODE TO TEST STATUS AND SET APPROPRIATE FLAGS
0015	9201	36	NOP		.
0016	9201	37	NOP		.
0017	9201	38	NOP		.
0018	8FE8	39	INC CFLAG		COMPLETION FLAG
0019	B7F3	40	BRC INT.1		CLEAR INTERRUPT AND RETURN
		41	*		

EXAMPLE 3. MAIN PROGRAM FOR IOCL

```

42
43 *
44 *****PROGRAM TO USE IOCL *****
45 *
001A 5448 46 MESSAGE TEST 'THIS IS A DATA CHAIN EXAMPLE'
001B 4953
001C 2049
001D 5320
001E 4120
001F 4441
0020 5441
0021 2043
0022 4841
0023 494E
0024 2045
0025 5841
0026 4D50
0027 4C45

47 *
0028 48 DATCHAIN EQU $ PROGRAM ORIGIN --DATA CHAIN
49 *
0028 8900 * 50 LDA =28 MESSAGE LENGTH
0029 86DC 51 STA MSGLEN
002A 8900 * 52 LDA = MESSAGE LOCATION OF MESSAGE
002B 86D9 53 STA MSGADR
002C 0401 54 CLA
002D 86DS 55 STA CFLAG ZERO COMPLETION FLAG
56 *
002E 8800 * 57 LDD =IOCLDCH D REGISTER IS ADDRESS OF IOCL
002F 8BD2 58 LDX TYADR LOAD ADDRESS OF DEVICE---
TELETYPE
0030 E000 * 59 POT X'100',1 START I/O ON TELETYPE
60 *
0031 89CF 61 WAIT LDA CFLAG WAIT UNTIL MESSAGE IS OUTPUT
0032 D5FF 62 BEZ $-1
63 *
0033 D200 * 64 B COMCHAIN DO COMMAND CHAIN EXAMPLE
65 *
0034 00C0 66 LPOOL
0035 001C
0036 001A
0037 0002
0038 0100
0039 0000 *
67 *

```

EXAMPLE 4. COMMAND CHAIN

		68			
		69	*		
		70	*****EQUATES*****		
		71	*		
	0200	72	READKB EQU X'200'	READ	
	0004	73	CMDCH EQU 4	COMMAND CHAIN	
		74	*		
003A		75	ECHO RES 11	BUFFER FOR INPUT/OUTPUT	
				MESSAGE	
		76	*		
		77	*****IOCL FOR COMMAND CHAIN EXAMPLE*****		
		78	*		
0045	0204	79	IOCLCMD DATA READKB+CMDCH		
0046	0000	80	DATA 0	NO INTERRUPT	
0047	003A	81	DATA ECHO	BUFFER FOR MESSAGE	
0048	0014	82	DATA 20	20 CHARACTERS	
		83	*		
0049	0102	84	DATA PRINT +DATACH	OUTPUT AND DATA CHAIN	
				ON CR LF	
004A	0000	85	DATA 0	NO INTERRUPT	
004B	003A	86	DATA ECHO	MESSAGE LOCATION	
004C	0014	87	DATA 20	SIZE OF MESSAGE	
		88	*		
004D	0100	89	DATA PRINT	OUTPUT CR LF	
004E	0000	90	DATA 0		
004F	000A	91	DATA CRLF	LOCATION OF CR LF	
0050	0002	92	DATA 2	2 CHARTERS	
		93	*		

EXAMPLE 5. PROGRAM FOR INPUT AND OUTPUT

		94			
		95	*		
	0051	96	COMCHAIN EQU \$	COMMAND CHAIN EXAMPLE	
0051	8800 *	97	LDD =IOCLCMD	SETUP IOCL D REG	
0052	8BAF	98	LDX TYADR	DEVICE TO XR	
0053	E0E5	99	POT X'100'.1	SIO	
0054	F5FF	100	BCR3 \$-1	SEE IF I/O ACCEPTED	
		101	*		
0055	92FC	102	B COMCHAIN	DO IT ONE MORE TIME	
		103	*		
	0028	104	END DATCHAIN		
0056	0045				

NO ERRORS



TABLE D-1. POWERS OF TWO TABLE

$2^n$	$n$	$2^{-n}$
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25

TABLE D-2. HEXADECIMAL TABLES

The following tables aid in converting hexadecimal values to decimal values, or the reverse.

For numbers outside the range of the table, add the following values to the table figures:

Direct Conversion Table

This table provides direct conversion of decimal and hexadecimal numbers in these ranges :

Hexadecimal-    Decimal  
000 to FFF      0000 to 4095

HEXADECIMAL	DECIMAL
1000	4096
2000	8192
3000	12288
4000	16384
5000	20480
6000	24576
7000	28672
8000	32768
9000	36864
A000	40960
B000	45056
C000	49152
D0000	53248
E000	57344
F000	61440

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	0010	0011	0012	0013	0014	0015
01	0016	0017	0018	0019	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	0030	0031
02	0032	0033	0034	0035	0036	0037	0038	0039	0040	0041	0042	0043	0044	0045	0046	0047
03	0048	0049	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	0060	0061	0062	0063
04	0064	0065	0066	0067	0068	0069	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079
05	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	0090	0091	0092	0093	0094	0095
06	0096	0097	0098	0099	0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	0110	0111
07	0112	0113	0114	0115	0116	0117	0118	0119	0120	0121	0122	0123	0124	0125	0126	0127
08	0128	0129	0130	0131	0132	0133	0134	0135	0136	0137	0138	0139	0140	0141	0142	0143
09	0144	0145	0146	0147	0148	0149	0150	0151	0152	0153	0154	0155	0156	0157	0158	0159
0A	0160	0161	0162	0163	0164	0165	0166	0167	0168	0169	0170	0171	0172	0173	0174	0175
0B	0176	0177	0178	0179	0180	0181	0182	0183	0184	0185	0186	0187	0188	0189	0190	0191
0C	0192	0193	0194	0195	0196	0197	0198	0199	0200	0201	0202	0203	0204	0205	0206	0207
0D	0208	0209	0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	0220	0221	0222	0223
0E	0224	0225	0226	0227	0228	0229	0230	0231	0232	0233	0234	0235	0236	0237	0238	0239
0F	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	0250	0251	0252	0253	0254	0255
10	0256	0257	0258	0259	0260	0261	0262	0263	0264	0265	0266	0267	0268	0269	0270	0271
11	0272	0273	0274	0275	0276	0277	0278	0279	0280	0281	0282	0283	0284	0285	0286	0287
12	0288	0289	0290	0291	0292	0293	0294	0295	0296	0297	0298	0299	0300	0301	0302	0303
13	0304	0305	0306	0307	0308	0309	0310	0311	0312	0313	0314	0315	0316	0317	0318	0319
14	0320	0321	0322	0323	0324	0325	0326	0327	0328	0329	0330	0331	0332	0333	0334	0335
15	0336	0337	0338	0339	0340	0341	0342	0343	0344	0345	0346	0347	0348	0349	0350	0351
16	0352	0353	0354	0355	0356	0357	0358	0359	0360	0361	0362	0363	0364	0365	0366	0367
17	0368	0369	0370	0371	0372	0373	0374	0375	0376	0377	0378	0379	0380	0381	0382	0383



TABLE D-2. HEXADECIMAL TABLES (CONT'D)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
18	0384	0385	0386	0387	0388	0389	0390	0391	0392	0293	0394	0395	0396	0397	0398	0399
19	0400	0401	0402	0403	0404	0405	0406	0407	0408	0409	0410	0411	0412	0413	0414	0415
1A	0416	0417	0418	0419	0420	0421	0422	0423	0424	0425	0426	0427	0428	0429	0430	0431
1B	0432	0433	0434	0435	0436	0437	0438	0439	0440	0441	0442	0443	0444	0445	0446	0447
1C	0448	0449	0450	0451	0452	0453	0454	0455	0456	0457	0458	0459	0460	0461	0462	0463
1D	0464	0465	0466	0467	0468	0469	0470	0471	0472	0473	0474	0475	0476	0477	0478	0479
1E	0480	0481	0482	0483	0484	0485	0486	0487	0488	0489	0490	0491	0492	0493	0494	0495
1F	0496	0497	0498	0499	0500	0501	0502	0503	0504	0505	0506	0507	0508	0509	0510	0511
20	0512	0513	0514	0515	0516	0517	0518	0519	0520	0521	0522	0523	0524	0525	0526	0527
21	0528	0529	0530	0531	0532	0533	0534	0535	0536	0537	0538	0539	0540	0541	0542	0543
22	0544	0545	0546	0547	0548	0549	0550	0551	0552	0553	0554	0555	0556	0557	0558	0559
23	0560	0561	0562	0563	0564	0565	0566	0567	0568	0569	0570	0571	0572	0573	0574	0575
24	0576	0577	0578	0579	0580	0581	0582	0583	0584	0585	0586	0587	0588	0589	0590	0591
25	0592	0593	0594	0595	0596	0597	0598	0599	0600	0601	0602	0603	0604	0605	0606	0607
26	0608	0609	0610	0611	0612	0613	0614	0615	0616	0617	0618	0619	0620	0621	0622	0623
27	0624	0625	0626	0627	0628	0629	0630	0631	0632	0633	0634	0635	0636	0637	0638	0639
28	0640	0641	0642	0643	0644	0645	0646	0647	0648	0649	0650	0651	0652	0653	0654	0655
29	0656	0657	0658	0659	0660	0661	0662	0663	0664	0665	0666	0667	0668	0669	0670	0671
2A	0672	0673	0674	0675	0676	0677	0678	0679	0680	0681	0682	0683	0684	0685	0686	0687
2B	0688	0689	0690	0691	0692	0693	0694	0695	0696	0697	0698	0699	0700	0701	0702	0703
2C	0704	0705	0706	0707	0708	0709	0710	0711	0712	0713	0714	0715	0716	0717	0718	0719
2D	0720	0721	0722	0723	0724	0725	0726	0727	0728	0729	0730	0731	0732	0733	0734	0735
2E	0736	0737	0738	0739	0740	0741	0742	0743	0744	0745	0746	0747	0748	0749	0750	0751
2F	0752	0753	0754	0755	0756	0757	0758	0759	0760	0761	0762	0763	0764	0765	0766	0767
30	0768	0769	0770	0771	0772	0773	0774	0775	0776	0777	0778	0779	0780	0781	0782	0783
31	0784	0785	0786	0787	0788	0789	0790	0791	0792	0793	0794	0795	0796	0797	0798	0799
32	0800	0801	0802	0803	0804	0805	0806	0807	0808	0809	0810	0811	0812	0813	0814	0815
33	0816	0817	0818	0819	0820	0821	0822	0823	0824	0825	0826	0827	0828	0829	0830	0831
34	0832	0833	0834	0835	0836	0837	0838	0839	0840	0841	0842	0843	0844	0845	0846	0847
35	0848	0849	0850	0851	0852	0853	0854	0855	0856	0857	0858	0859	0860	0861	0862	0863
36	0864	0865	0866	0867	0868	0869	0870	0871	0872	0873	0874	0875	0876	0877	0878	0879
37	0880	0881	0882	0883	0884	0885	0886	0887	0888	0889	0890	0891	0892	0893	0894	0895
38	0896	0897	0898	0899	0900	0901	0902	0903	0904	0905	0906	0907	0908	0909	0910	0911
39	0912	0913	0914	0915	0916	0917	0918	0919	0920	0921	0922	0923	0924	0925	0926	0927
3A	0928	0929	0930	0931	0932	0933	0934	0935	0936	0937	0938	0939	0940	0941	0942	0943
3B	0944	0945	0946	0947	0948	0949	0950	0951	0952	0953	0954	0955	0956	0957	0958	0959
3C	0960	0961	0962	0963	0964	0965	0966	0967	0968	0969	0970	0971	0972	0973	0974	0975
3D	0976	0977	0978	0979	0980	0981	0982	0983	0984	0985	0986	0987	0988	0989	0990	0991
3E	0992	0993	0994	0995	0996	0997	0998	0999	1000	1001	1002	1003	1004	1005	1006	1007
3F	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023

TABLE D-2. HEXADECIMAL TABLES (CONT'D)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
40	1024	1025	1026	1027	1028	1029	1030	1031	1032	1033	1034	1035	1036	1037	1038	1039
41	1040	1041	1042	1043	1044	1045	1046	1047	1048	1049	1050	1051	1052	1053	1054	1055
42	1056	1057	1058	1059	1060	1061	1062	1063	1064	1065	1066	1067	1068	1069	1070	1071
43	1072	1073	1074	1075	1076	1077	1078	1079	1080	1081	1082	1083	1084	1085	1086	1087
44	1088	1089	1090	1091	1092	1093	1094	1095	1096	1097	1098	1099	1100	1101	1102	1103
45	1104	1105	1106	1107	1108	1109	1110	1111	1112	1113	1114	1115	1116	1117	1118	1119
46	1120	1121	1122	1123	1124	1125	1126	1127	1128	1129	1130	1131	1132	1133	1134	1135
47	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151
48	1152	1153	1154	1155	1156	1157	1158	1159	1160	1161	1162	1163	1164	1165	1166	1167
49	1168	1169	1170	1171	1172	1173	1174	1175	1176	1177	1178	1179	1180	1181	1182	1183
4A	1184	1185	1186	1187	1188	1189	1190	1191	1192	1193	1194	1195	1196	1197	1198	1199
4B	1200	1201	1202	1203	1204	1205	1206	1207	1208	1209	1210	1211	1212	1213	1214	1215
4C	1216	1217	1218	1219	1220	1221	1222	1223	1224	1225	1226	1227	1228	1229	1230	1231
4D	1232	1233	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243	1244	1245	1246	1247
4E	1248	1249	1250	1251	1252	1253	1254	1255	1256	1257	1258	1259	1260	1261	1262	1263
4F	1264	1265	1266	1267	1268	1269	1270	1271	1272	1273	1274	1275	1276	1277	1278	1279
50	1280	1281	1282	1283	1284	1285	1286	1287	1288	1289	1290	1291	1292	1293	1294	1295
51	1296	1297	1298	1299	1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311
52	1312	1313	1314	1315	1316	1317	1318	1319	1320	1321	1322	1323	1324	1325	1326	1327
53	1328	1329	1330	1331	1332	1333	1334	1335	1336	1337	1338	1339	1340	1341	1342	1343
54	1344	1345	1346	1347	1348	1349	1350	1351	1352	1353	1354	1355	1356	1357	1358	1359
55	1360	1361	1362	1363	1364	1365	1366	1367	1368	1369	1370	1371	1372	1373	1374	1375
56	1376	1377	1378	1379	1380	1381	1382	1383	1384	1385	1386	1387	1388	1389	1390	1391
57	1392	1393	1394	1395	1396	1397	1398	1399	1400	1401	1401	1403	1404	1405	1406	1407
58	1408	1409	1410	1411	1412	1413	1414	1415	1416	1417	1418	1419	1420	1421	1422	1423
59	1424	1425	1426	1427	1428	1429	1430	1431	1432	1433	1434	1435	1436	1437	1438	1439
5A	1440	1441	1442	1443	1444	1445	1446	1447	1448	1449	1450	1451	1452	1453	1454	1455
5B	1456	1457	1458	1459	1460	1461	1462	1463	1464	1465	1466	1467	1468	1469	1470	1471
5C	1472	1473	1474	1475	1476	1477	1478	1479	1480	1481	1482	1483	1484	1485	1486	1487
5D	1488	1489	1490	1491	1492	1493	1494	1495	1496	1497	1498	1499	1500	1501	1502	1503
5E	1504	1505	1506	1507	1508	1509	1510	1511	1512	1513	1514	1515	1516	1517	1518	1519
5F	1520	1521	1522	1523	1524	1525	1526	1527	1528	1529	1530	1531	1532	1533	1534	1535
60	1536	1537	1538	1539	1540	1541	1542	1543	1544	1545	1546	1547	1548	1549	1550	1551
61	1552	1553	1554	1555	1556	1557	1558	1559	1560	1561	1562	1563	1564	1565	1566	1567
62	1568	1569	1570	1571	1572	1573	1574	1575	1576	1577	1578	1578	1580	1581	1582	1583
63	1584	1585	1586	1587	1588	1589	1590	1591	1592	1593	1594	1595	1596	1597	1598	1599
64	1600	1601	1602	1603	1604	1605	1606	1607	1608	1609	1610	1611	1612	1613	1614	1615
65	1616	1617	1618	1619	1620	1621	1622	1623	1624	1625	1626	1627	1628	1629	1630	1631
66	1632	1633	1634	1635	1636	1637	1638	1639	1640	1641	1642	1643	1644	1645	1646	1647
67	1648	1649	1650	1651	1652	1653	1654	1655	1656	1657	1658	1659	1660	1661	1662	1663

TABLE D-2. HEXADECIMAL TABLES (CONT'D)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
68	1664	1665	1666	1667	1668	1669	1670	1671	1672	1673	1674	1675	1576	1677	1678	1679
69	1680	1681	1682	1683	1684	1685	1686	1687	1688	1689	1690	1691	1692	1693	1694	1695
6A	1696	1697	1698	1699	1700	1701	1702	1703	1704	1705	1706	1707	1708	1709	1710	1711
6B	1712	1713	1714	1715	1716	1717	1718	1719	1720	1721	1722	1723	1724	1725	1726	1727
6C	1728	1729	1730	1731	1732	1733	1734	1735	1736	1737	1738	1739	1740	1741	1742	1743
6D	1744	1745	1746	1747	1748	1749	1750	1751	1752	1753	1754	1755	1756	1757	1758	1759
6E	1760	1761	1762	1763	1764	1765	1766	1767	1768	1769	1770	1771	1772	1773	1774	1775
6F	1776	1777	1778	1779	1780	1781	1782	1783	1784	1785	1786	1787	1788	1789	1790	1791
70	1792	1793	1794	1795	1796	1797	1798	1799	1800	1801	1802	1803	1804	1805	1806	1807
71	1808	1809	1810	1811	1812	1813	1814	1815	1816	1817	1818	1819	1820	1821	1822	1823
72	1824	1825	1826	1827	1828	1829	1830	1831	1832	1833	1834	1835	1836	1837	1838	1839
73	1840	1841	1842	1843	1844	1845	1846	1847	1848	1849	1850	1851	1852	1853	1854	1855
74	1856	1857	1858	1859	1860	1861	1862	1863	1864	1865	1866	1867	1868	1869	1870	1871
75	1872	1873	1874	1875	1876	1877	1878	1879	1880	1881	1882	1883	1884	1885	1886	1887
76	1888	1889	1890	1891	1892	1893	1894	1895	1896	1897	1898	1899	1900	1901	1902	1903
77	1904	1905	1906	1907	1908	1909	1910	1911	1912	1913	1914	1915	1916	1917	1918	1919
78	1920	1921	1922	1923	1924	1925	1926	1927	1928	1929	1930	1931	1932	1933	1934	1935
79	1936	1937	1938	1939	1940	1941	1942	1943	1944	1945	1946	1947	1948	1949	1950	1951
7A	1952	1953	1954	1955	1956	1957	1958	1959	1960	1961	1962	1963	1964	1965	1966	1967
7B	1968	1969	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983
7C	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999
7D	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015
7E	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031
7F	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047
80	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063
81	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079
82	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095
83	2096	2097	2098	2099	2100	2101	2102	2103	2104	2105	2106	2107	2108	2109	2110	2111
84	2112	2113	2114	2115	2116	2117	2118	2119	2120	2121	2122	2123	2124	2125	2126	2127
85	2128	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143
86	2144	2145	2146	2147	2148	2149	2150	2151	2152	2153	2154	2155	2156	2157	2158	2159
87	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	2170	2171	2172	2173	2174	2175
88	2176	2177	2178	2179	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191
89	2192	2193	2194	2195	2196	2197	2198	2199	2200	2201	2202	2203	2204	2205	2206	2207
8A	2208	2209	2210	2211	2212	2213	2214	2215	2216	2217	2218	2219	2220	2221	2222	2223
8B	2224	2225	2226	2227	2228	2229	2230	2231	2232	2233	2234	2235	2236	2237	2238	2239
8C	2240	2241	2242	2243	2244	2245	2246	2247	2248	2249	2250	2251	2252	2253	2254	2255
8D	2256	2257	2258	2259	2260	2261	2262	2263	2264	2265	2266	2267	2268	2269	2270	2271
8E	2272	2273	2274	2275	2276	2277	2278	2279	2280	2281	2282	2283	2284	2285	2286	2287
8F	2288	2289	2290	2291	2292	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	2303

TABLE D-2. HEXADECIMAL TABLES (CONT'D)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
90	2304	2305	2306	2307	2308	2309	2310	2311	2312	2313	2314	2315	2316	2317	2318	2319
91	2320	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330	2331	2332	2333	2334	2335
92	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351
93	2352	2353	2354	2355	2356	2357	2358	2359	2360	2361	2362	2363	2364	2365	2366	2367
94	2368	2369	2370	2371	2372	2373	2374	2375	2376	2377	2378	2379	2380	2381	2382	2383
95	2384	2385	2386	2387	2388	2389	2390	2391	2392	2393	2394	2395	2396	2397	2398	2399
96	2400	2401	2402	2403	2404	2405	2406	2407	2408	2409	2410	2411	2412	2413	2414	2415
97	2416	2417	2418	2419	2420	2421	2422	2423	2424	2425	2426	2427	2428	2429	2430	2431
98	2432	2433	2434	2435	2436	2437	2438	2439	2440	2441	2442	2443	2444	2445	2446	2447
99	2448	2449	2450	2451	2452	2453	2454	2455	2456	2457	2458	2459	2460	2461	2462	2463
9A	2464	2465	2466	2467	2468	2469	2470	2471	2472	2473	2474	2475	2476	2477	2478	2479
9B	2480	2481	2482	2483	2484	2485	2486	2487	2488	2489	2490	2491	2492	2493	2494	2495
9C	2496	2497	2498	2499	2500	2501	2502	2503	2504	2505	2506	2507	2508	2509	2510	2511
9D	2512	2513	2514	2515	2516	2517	2518	2519	2520	2521	2522	2523	2524	2525	2526	2527
9E	2528	2529	2530	2531	2532	2533	2534	2535	2536	2537	2538	2539	2540	2541	2542	2543
9F	2544	2545	2546	2547	2548	2549	2550	2551	2552	2553	2554	2555	2556	2557	2558	2559
A0	2560	2561	2562	2563	2564	2565	2566	2567	2568	2569	2570	2571	2572	2573	2574	2575
A1	2576	2577	2578	2579	2580	2581	2582	2583	2584	2585	2586	2587	2588	2589	2590	2591
A2	2592	2593	2594	2595	2596	2597	2598	2599	2600	2601	2602	2603	2604	2605	2606	2607
A3	2608	2609	2610	2611	2612	2613	2614	2615	2616	2617	2618	2619	2620	2621	2622	2623
A4	2624	2625	2626	2627	2628	2629	2630	2631	2632	2633	2634	2635	2636	2637	2638	2639
A5	2640	2641	2642	2643	2644	2645	2646	2647	2648	2649	2650	2651	2652	2653	2654	2655
A6	2656	2657	2658	2659	2660	2661	2662	2663	2664	2665	2666	2667	2668	2669	2670	2671
A7	2672	2673	2674	2675	2676	2677	2678	2679	2680	2681	2682	2683	2684	2685	2686	2687
A8	2688	2689	2690	2691	2692	2693	2694	2695	2696	2697	2698	2699	2700	2701	2702	2703
A9	2704	2705	2706	2707	2708	2709	2710	2711	2712	2713	2714	2715	2716	2717	2718	2719
AA	2720	2721	2722	2723	2724	2725	2726	2727	2728	2729	2730	2731	2732	2733	2734	2735
AB	2736	2737	2738	2739	2740	2741	2742	2743	2744	2745	2746	2747	2748	2749	2750	2751
AC	2752	2753	2754	2755	2756	2757	2758	2759	2760	2761	2762	2763	2764	2765	2766	2767
AD	2768	2769	2770	2771	2772	2773	2774	2775	2776	2777	2778	2779	2780	2781	2782	2783
AE	2784	2785	2786	2787	2788	2789	2790	2791	2792	2793	2794	2795	2796	2797	2798	2799
AF	2800	2801	2802	2803	2804	2805	2806	2807	2808	2809	2810	2811	2812	2813	2814	2815
B0	2816	2817	2818	2819	2820	2821	2822	2823	2824	2825	2826	2827	2828	2829	2830	2831
B1	2832	2833	2834	2835	2836	2837	2838	2839	2840	2841	2842	2843	2844	2845	2846	2847
B2	2848	2849	2850	2851	2852	2853	2854	2855	2856	2857	2858	2859	2860	2861	2862	2863
B3	2864	2865	2866	2867	2868	2869	2870	2871	2872	2873	2874	2875	2876	2877	2878	2879
B4	2880	2881	2882	2883	2884	2885	2886	2887	2888	2889	2890	2891	2892	2893	2894	2895
B5	2896	2897	2898	2899	2900	2901	2902	2903	2904	2905	2906	2907	2908	2909	2910	2911
B6	2912	2913	2914	2915	2916	2917	2918	2919	2920	2921	2922	2923	2924	2925	2926	2927
B7	2928	2929	2930	2931	2932	2933	2934	2935	2936	2937	2938	2939	2940	2941	2942	2943

TABLE D-2. HEXADECIMAL TABLES (CONT'D)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
B8	2944	2945	2946	2947	2948	2949	2950	2951	2952	2953	2954	2955	2956	2957	2958	2959
B9	2960	2961	2962	2963	2964	2965	2966	2967	2968	2969	2970	2971	2972	2973	2974	2975
BA	2976	2977	2978	2979	2980	2981	2982	2983	2984	2985	2986	2987	2988	2989	2990	2991
BB	2992	2993	2994	2995	2996	2997	2998	2999	3000	3001	3002	3003	3004	3005	3006	3007
BC	3008	3009	3010	3011	3012	3013	3014	3015	3016	3017	3018	3019	3020	3021	3022	3023
BD	3024	3025	3026	3027	3028	3029	3030	3031	3032	3033	3034	3035	3036	3037	3038	3039
BE	3040	3041	3042	3043	3044	3045	3046	3047	3048	3049	3050	3051	3052	3053	3054	3055
BF	3056	3057	3058	3059	3060	3061	3062	3063	3064	3065	3066	3067	3068	3069	3070	3071
C0	3072	3073	3074	3075	3076	3077	3078	3079	3080	3081	3082	3083	3084	3085	3086	3087
C1	3088	3089	3090	3091	3092	3093	3094	3095	3096	3097	3098	3099	3100	3101	3102	3103
C2	3104	3105	3106	3107	3108	3109	3110	3111	3112	3113	3114	3115	3116	3117	3118	3119
C3	3120	3121	3122	3123	3124	3125	3126	3127	3128	3129	3130	3131	3132	3133	3134	3135
C4	3136	3137	3138	3139	3140	3141	3142	3143	3144	3145	3146	3147	3148	3149	3150	3151
C5	3152	3153	3154	3155	3156	3157	3158	3159	3160	3161	3162	3163	3164	3165	3166	3167
C6	3168	3169	3170	3171	3172	3173	3174	3175	3176	3177	3178	3179	3180	3181	3182	3183
C7	3184	3185	3186	3187	3188	3189	3190	3191	3192	3193	3194	3195	3196	3197	3198	3199
C8	3200	3201	3202	3203	3204	3205	3206	3207	3208	3209	3210	3211	3212	3213	3214	3215
C9	3216	3217	3218	3219	3220	3221	3222	3223	3224	3225	3226	3227	3228	3229	3230	3231
CA	3232	3233	3234	3235	3236	3237	3238	3239	3240	3241	3242	3243	3244	3245	3246	3247
CB	3248	3249	3250	3251	3252	3253	3254	3255	3256	3257	3258	3259	3260	3261	3262	3263
CC	3264	3265	3266	3267	3268	3269	3270	3271	3272	3273	3274	3275	3276	3277	3278	3279
CD	3280	3281	3282	3283	3284	3285	3286	3287	3288	3289	3290	3291	3292	3293	3294	3295
CE	3296	3297	3298	3299	3300	3301	3302	3303	3304	3305	3306	3307	3308	3309	3310	3311
CF	3312	3313	3314	3315	3316	3317	3318	3319	3320	3321	3322	3323	3324	3325	3326	3327
D0	3328	3329	3330	3331	3332	3333	3334	3335	3336	3337	3338	3339	3340	3341	3342	3343
D1	3344	3345	3346	3347	3348	3349	3350	3351	3352	3353	3354	3355	3356	3357	3358	3359
D2	3360	3361	3362	3363	3364	3365	3366	3367	3368	3369	3370	3371	3372	3373	3374	3375
D3	3376	3377	3378	3379	3380	3381	3382	3383	3384	3385	3386	3387	3388	3389	3390	3391
D4	3392	3393	3394	3395	3396	3397	3398	3399	3400	3401	3402	3403	3404	3405	3406	3407
D5	3408	3409	3410	3411	3412	3413	3414	3415	3416	3417	3418	3419	3420	3421	3422	3423
D6	3424	3425	3426	3427	3428	3429	3430	3431	3432	3433	3434	3435	3436	3437	3438	3439
D7	3440	3441	3442	3443	3444	3445	3446	3447	3448	3449	3450	3451	3452	3453	3454	3455
D8	3456	3457	3458	3459	3460	3461	3462	3463	3464	3465	3466	3467	3468	3469	3470	3471
D9	3472	3473	3474	3475	3476	3477	3478	3479	3480	3481	3482	3483	3484	3485	3486	3487
DA	3488	3489	3490	3491	3492	3493	3494	3495	3496	3497	3498	3499	3500	3501	3502	3503
DB	3504	3505	3506	3507	3508	3509	3510	3511	3512	3513	3514	3515	3516	3517	3518	3519
DC	3520	3521	3522	3523	3524	3525	3526	3527	3528	3529	3530	3531	3532	3533	3534	3535
DD	3536	3537	3538	3539	3540	3541	3542	3543	3544	3545	3546	3547	3548	3549	3550	3551
DE	3552	3553	3554	3555	3556	3557	3558	3559	3560	3561	3562	3563	3564	3565	3566	3567
DF	3568	3569	3570	3571	3572	3573	3574	3575	3576	3577	3578	3579	3580	3581	3582	3583

TABLE D-2. HEXADECIMAL TABLES (CONT'D)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
E0	3584	3585	3586	3587	3588	3589	3590	3591	3592	3593	3594	3595	3596	3597	3598	3599
E1	3600	3601	3602	3603	3604	3605	3606	3607	3608	3609	3610	6311	3612	3613	3614	3615
E2	3616	3617	3618	3619	3620	3621	3622	3623	3624	3625	3626	3627	3628	3629	3630	3631
E3	3632	3633	3634	3635	3636	3637	3638	3639	3640	3641	3642	3643	3644	3645	3646	3647
E4	3648	3649	3650	3651	3652	3653	3654	3655	3656	3657	3658	3659	3660	3661	3662	3663
E5	3664	3665	3666	3667	3668	3669	3670	3671	3672	3673	3674	3675	3676	3677	3678	3679
E6	3680	3681	3682	3683	3684	3685	3686	3687	3688	3689	3690	3691	3692	3693	3694	3695
E7	3696	3697	3698	3699	3700	3701	3702	3703	3704	3705	3706	3707	3708	3709	3710	3711
E8	3712	3713	3714	3715	3716	3717	3718	3719	3720	3721	3722	3723	3724	3725	3726	3727
E9	3728	3729	3730	3731	3732	3733	3734	3735	3736	3737	3738	3739	3740	3741	3742	3743
EA	3744	3745	3746	3747	3748	3749	3750	3751	3752	3753	3754	3755	3756	3757	3758	3759
EB	3760	3761	3762	3763	3764	3765	3766	3767	3768	3769	3770	3771	3772	3773	3774	3775
EC	3776	3777	3778	3779	3780	3781	3782	3783	3784	3785	3786	3787	3788	3789	3790	3791
ED	3792	3793	3794	3795	3796	3797	3798	3799	3800	3801	3802	3803	3804	3805	3806	3807
EE	3808	3809	3810	3811	3812	3813	3814	3815	3816	3817	3818	3819	3820	3821	3822	3823
EF	3824	3825	3826	3827	3828	3829	3830	3831	3832	3833	3834	3835	3836	3837	3838	3839
F0	3840	3841	3842	3843	3844	3845	3846	3847	3848	3849	3850	3851	3852	3853	3854	3855
F1	3856	3857	3858	3859	3860	3861	3862	3863	3864	3865	3866	3867	3868	3869	3870	3871
F2	3872	3873	3874	3875	3876	3877	3878	3879	3880	3881	3882	3883	3884	3885	3886	3887
F3	3888	3889	3890	3891	3892	3893	3894	3895	3896	3897	3898	3899	3900	3901	3902	3903
F4	3904	3905	3906	3907	3908	3909	3910	3911	3912	3913	3914	3915	3916	3917	3918	3919
F5	3920	3921	3922	3923	3924	3925	3926	3927	3928	3929	3930	3931	3932	3933	3934	3935
F6	3936	3937	3938	3939	3940	3941	3942	3943	3944	3945	3946	3947	3948	3949	3950	3951
F7	3952	3953	3954	3955	3956	3957	3958	3959	3960	3961	3962	3963	3964	3965	3966	3967
F8	3968	3969	3970	3971	3972	3973	3974	3975	3976	3977	3978	3979	3980	3981	3982	3983
F9	3984	3985	3986	3987	3988	3989	3990	3991	3992	3993	3994	3995	3996	3997	3998	3999
FA	4000	4001	4002	4003	4004	4005	4006	4007	4008	4009	4010	4011	4012	4013	4014	4015
FB	4016	4017	4018	4019	4020	4021	4022	4023	4024	4025	4026	4027	4028	4029	4030	4031
FC	4032	4033	4034	4035	4036	4037	4038	4039	4040	4041	4042	4043	4044	4045	4046	4047
FD	4048	4049	4050	4051	4052	4053	4054	4055	4056	4057	4058	4059	4060	4061	4062	4063
FE	4064	4065	4066	4067	4068	4069	4070	4071	4072	4073	4074	4075	4076	4077	4078	4079
FF	4080	4081	4082	4083	4084	4085	4086	4087	4088	4089	4090	4091	4092	4093	4094	4095

TABLE D-3. HEXADECIMAL AND DECIMAL INTEGER CONVERSION TABLE

BITS: 0123	HALFWORD		BYTE		BYTE		BYTE		BYTE		HALFWORD		BYTE	
	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	268,435,456	1	16,777,216	1	1,048,576	1	65,536	1	4,096	1	256	1	16	1
2	536,870,912	2	33,554,432	2	2,097,152	2	131,072	2	8,192	2	512	2	32	2
3	805,306,368	3	50,331,648	3	3,145,728	3	196,608	3	12,288	3	768	3	48	3
4	1,073,741,824	4	67,108,864	4	4,194,304	4	262,144	4	16,384	4	1,024	4	64	4
5	1,342,177,280	5	83,886,080	5	5,242,880	5	327,680	5	20,480	5	1,280	5	80	5
6	1,610,612,736	6	100,663,296	6	6,291,456	6	393,216	6	24,576	6	1,536	6	96	6
7	1,879,048,192	7	117,440,512	7	7,340,032	7	458,752	7	28,672	7	1,792	7	112	7
8	2,147,483,648	8	134,217,728	8	8,388,608	8	524,288	8	32,768	8	2,048	8	128	8
9	2,415,919,104	9	150,994,944	9	9,437,184	9	589,824	9	36,864	9	2,304	9	144	9
A	2,684,354,560	A	167,772,160	A	10,485,760	A	655,360	A	40,960	A	2,560	A	160	A
B	2,952,790,016	B	184,549,376	B	11,534,336	B	720,896	B	45,056	B	2,816	B	176	B
C	3,221,225,472	C	201,326,592	C	12,582,912	C	786,432	C	49,152	C	3,072	C	192	C
D	3,489,660,928	D	218,103,808	D	13,631,488	D	851,968	D	53,248	D	3,328	D	208	D
E	3,758,096,384	E	234,881,024	E	14,680,064	E	917,504	E	57,344	E	3,584	E	224	E
F	4,026,531,840	F	251,658,240	F	15,728,640	F	983,040	F	61,440	F	3,840	F	240	F
8		7		6		5		4		3		2		1

TABLE D-3. HEXADECIMAL AND DECIMAL INTEGER CONVERSION TABLE

TO CONVERT HEXADECIMAL TO DECIMAL

1. Locate the column of decimal numbers corresponding to the left-most digit or letter of the hexadecimal; select from this column and record the number that corresponds to the position of the hexadecimal digit or letter.
2. Repeat step 1 for the next (second from the left) position.
3. Repeat step 1 for the units (third from the left) position.
4. Add the number selected from the table to form the decimal number.

<u>EXAMPLE</u>	
Conversion of Hexadecimal Value	D34
1. D	3328
2. 3	48
3. 4	<u>4</u>
4. Decimal	3380

TO CONVERT DECIMAL TO HEXADECIMAL

1. (a) Select from the table the highest decimal number that is equal to or less than the number to be converted.  
 (b) Record the hexadecimal of the column containing the selected number.  
 (c) Subtract the selected decimal from the number to be converted.
2. Using the remainder from step 1 (c) repeat all of step 1 to develop the second position of the hexadecimal (and a remainder).
3. Using the remainder from step 2 repeat all of step 1 to develop the units position of the hexadecimal.
4. Combine terms to form the hexadecimal number.

<u>EXAMPLE</u>	
Conversion of Decimal Value	3380
1. D	<u>-3328</u>
	52
2. 3	<u>-48</u>
	4
3. 4	<u>-4</u>
4. Hexadecimal	D34

To convert Integer numbers greater than the capacity of table, use the techniques below:

HEXADECIMAL TO DECIMAL

Successive cumulative multiplication from left to right, adding units position

Example:  $D34_{16} = 3380_{10}$

$$\begin{array}{r}
 D = 13 \\
 \times 16 \\
 \hline
 208 \\
 3 = +3 \\
 \hline
 211 \\
 \times 16 \\
 \hline
 3376 \\
 4 = +4 \\
 \hline
 3380
 \end{array}$$

DECIMAL TO HEXADECIMAL

Divide and collect the remainder in reverse order.

Example:  $3380_{10} = X_{16}$

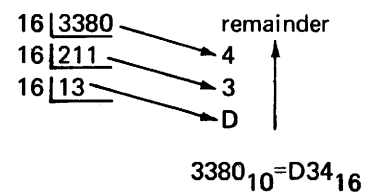




TABLE D-4. HEXADECIMAL AND DECIMAL FRACTION CONVERSION TABLE

Hexadecimal and Decimal Fraction Conversion Table

HALFWORD																
BYTE				BYTE												
BITS 0123		4567				0123				4567						
Hex	Decimal	Hex	Decimal			Hex	Decimal			Hex	Decimal Equivalent					
.0	.0000	.00	.0000	0000	.000	.0000	0000	0000	.0000	.0000	0000	0000	0000			
.1	.0625	.01	.0039	0625	.001	.0002	4414	0625	.0001	.0000	1525	8789	0625			
.2	.1250	.02	.0078	1250	.002	.0004	8828	1250	.0002	.0000	3051	7578	1250			
.3	.1875	.03	.0117	1875	.003	.0007	3242	1875	.0003	.0000	4577	6367	1875			
.4	.2500	.04	.0156	2500	.004	.0009	7656	2500	.0004	.0000	6103	5156	2500			
.5	.3125	.05	.0195	3125	.005	.0012	2070	3125	.0005	.0000	7629	3945	3125			
.6	.3750	.06	.0234	3750	.006	.0014	6484	3750	.0006	.0000	9155	2734	3750			
.7	.4375	.07	.0273	4375	.007	.0017	0898	4375	.0007	.0001	0681	1523	4375			
.8	.5000	.08	.0312	5000	.008	.0019	5312	5000	.0008	.0001	2207	0312	5000			
.9	.5625	.09	.0351	5625	.009	.0021	9726	5625	.0009	.0001	3732	9101	5625			
.A	.6250	.0A	.0390	6250	.00A	.0024	4140	6250	.000A	.0001	5258	7890	6250			
.B	.6875	.0B	.0429	6875	.00B	.0026	8554	6875	.000B	.0001	6784	6679	6875			
.C	.7500	.0C	.0468	7500	.00C	.0029	2968	7500	.000C	.0001	8310	5468	7500			
.D	.8125	.0D	.0507	8125	.00D	.0031	7382	8125	.000D	.0001	9836	4257	8125			
.E	.8750	.0E	.0546	8750	.00E	.0034	1796	8750	.000E	.0002	1362	3046	8750			
.F	.9375	.0F	.0585	9375	.00F	.0036	6210	9375	.000F	.0002	2888	1835	9375			
1			2			3				4						

POWERS OF 16 TABLE

Example:  $268,435,456_{10} = (2.68435456 \times 10^8)_{10} = 1000\ 0000_{16} = (10^7)_{16}$

$16^n$	n
1	0
16	1
256	2
4 096	3
65 536	4
1 048 576	5
16 777 216	6
268 435 456	7
4 294 967 296	8
68 719 476 736	9
1 099 511 627 776	10 = A
17 592 186 044 416	11 = B
281 474 976 710 656	12 = C
4 503 599 627 370 496	13 = D
72 057 594 037 927 936	14 = E
1 152 921 504 606 846 976	15 = F

Decimal Values

TABLE D-4 HEXADECIMAL AND DECIMAL FRACTION CONVERSION TABLE (Cont'd)

TO CONVERT .ABC HEXADECIMAL TO DECIMAL

Find .A in position 1     .6250  
 Find .0B in position 2   .0429 6875  
 Find .00C in position 3   .0029 2968 7500  
  
 ABC Hex is equal to     .6708 9843 7500

TO CONVERT .13 DECIMAL TO HEXADECIMAL

1. Find .1250 next lowest to     .1300                             =.2 Hex  
    Subtract                         -.1250  
 2. Find .0039 0625 next lowest   .0050 0000                             =.01  
    to                                     -.0039 9625  
 3. Find .0009 7656 2500           .0010 9375 0000                         =.004  
    -.0009 7656 2500  
 4. Find .0001 0681 1523 4375     .0001 1718 7500 0000                     =.0007  
    -.0001 0681 1523 4375  
 5. 13 Decimal is approximately    .0000 1037 5976 5625                     =.2147 Hex  
    equal to   ↑

To convert fractions beyond the capacity of table, use techniques below:

HEXADECIMAL FRACTION TO DECIMAL

Convert the hexadecimal fraction to its decimal equivalent using the same technique as for integer numbers. Divide the results by  $16^n$  (n is the number of fraction positions).

Example:                     .8A7    =.540771<sub>10</sub>  
                                    .8A7<sub>16</sub> = 2215<sub>10</sub>                                     .540771  
                                    16<sup>3</sup>    = 4096                                     4096 2215.000000

DECIMAL FRACTION TO HEXADICIMAL

Collect Integer parts of product in the order of calculation

Example:                     .5408<sub>10</sub> = .8A7<sub>16</sub>  
                                    .5408  
                                      X16  
 8 ←                       8 .6528  
                                      X16  
 A ←                      10 .4448  
                                      X16  
 7 ←                       7 .1168

TABLE D-5. HEXADECIMAL ADDITION AND SUBTRACTION TABLE

Example:  $6 + 2 = 8$ ,  $8 - 2 = 6$ , and  $8 - 6 = 2$

	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10
2	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11
3	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12
4	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13
5	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14
6	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15
7	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16
8	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17
9	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18
A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19
B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A
C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

TABLE D-6. HEXADECIMAL MULTIPLICATION TABLE

Example:  $2 \times 4 = 08$ ,  $F \times 2 = 1E$

	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
2	02	04	06	08	0A	0C	0E	10	12	14	16	18	1A	1C	1E
3	03	06	09	0C	0F	12	15	18	1B	1E	21	24	27	2A	2D
4	04	08	0C	10	14	18	1C	20	24	28	2C	30	34	38	3C
5	05	0A	0F	14	19	1E	23	28	2D	32	37	3C	41	46	4B
6	06	0C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A
7	07	0E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69
8	08	10	18	20	28	30	38	40	48	50	58	60	68	70	78
9	09	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87
A	0A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96
B	0B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5
C	0C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4
D	0D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3
E	0E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2
F	0F	1E	20	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1

APPENDIX E

ASCII CHARACTER SET AND HEXADECIMAL CODES

		0000	0001	0010	0011	0100	0101	0110	0111
COL		00	10	20	30	40	50	60	70
4321	ROW								
0000	0	NULL	DCO	BLANK	Ø	@	P		
0001	1	SOM	X-ON	!	1	A	Q		
0010	2	EOA	TAPE	"	2	B	R		
0011	3	EOM	X-OFF	#	3	C	S		
0100	4	EOT	TAPE	\$	4	D	T		
0101	5	WRU	NAK	%	5	E	U		
0110	6	RU	SYN	&	6	F	V		
0111	7	BEL	ETB	'	7	G	W		
1000	8	BS	SO	(	8	H	X		
1001	9	TAB	SI	)	9	I	Y		
1010	A	LF	S2	*	:	J	Z		
1011	B	VT	S3	+	;	K	[		
1100	C	FORM	S4	,	<	L	\		ACK
1101	D	RETURN	S5	-	=	M	]		ALT MODE
1110	E	SO	S6	.	>	N	↑		ESC
1111	F	SI	S7	/	?	O	←		RUB OUT

**APPENDIX F**  
**SYSTEMS 72 INSTRUCTIONS**  
**ALPHABETICAL LISTING**

INSTRUCTION FORMAT														
<table border="1" style="margin: auto; border-collapse: collapse;"> <tr> <td style="padding: 2px 5px;">R</td> <td style="padding: 2px 5px;">I</td> <td style="padding: 2px 5px;">X</td> <td style="padding: 2px 5px;">OP CODE</td> <td style="padding: 2px 5px;">S</td> <td style="padding: 2px 5px;">DISPLACEMENT</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td style="text-align: center;">2</td> <td style="text-align: center;">3 4 5 6 7</td> <td style="text-align: center;">8 9</td> <td style="text-align: center;">10 11 12 13 14 15</td> </tr> </table>			R	I	X	OP CODE	S	DISPLACEMENT	0	1	2	3 4 5 6 7	8 9	10 11 12 13 14 15
R	I	X	OP CODE	S	DISPLACEMENT									
0	1	2	3 4 5 6 7	8 9	10 11 12 13 14 15									
<u>MNEMONIC</u>	<u>CODE</u>	<u>INSTRUCTION</u>												
ADD	0D	Add Memory to A Register												
AND	02	AND Memory to A Register												
B	12	Branch												
BAL	13	Branch and Link												
BC	56	Branch on Carry												
BCR	15	Branch on Conditions Reset												
BCS	16	Branch on Conditions Set												
BE	75	Branch if Equal												
BEZ	75	Branch if Equal to Zero												
BGE	35	Branch if Equal To or Greater Than												
BGEZ	35	Branch if Equal To or Greater Than Zero												
BIX	14	Increment X and Branch if Not Equal to Zero												
BL	36	Branch if Less Than												
BLZ	36	Branch if Less Than Zero												
BNC	55	Branch on No Carry												
BNE	56	Branch if Not Equal To												
BNEZ	56	Branch if Not Equal to Zero												
BNO	35	Branch on No Overflow												
BO	36	Branch on Overflow												
BRC	17	Branch Return and Clear												
CAL1	18	Call 1												
CAL2	19	Call 2 to Monitor Services												
CAL3	1A	Call												
CLA	0401	Clear A Register												
CMP	10	Compare A Register to Memory												
DIV	1C	Divide												
EOR	04	Exclusive OR Memory into A Register												
INC	0F	Increment Memory by One												
LBY	0A	Load Byte into A Register												
LDA	09	Load A Register												
LDB	0C	Load A Register												
LDD	08	Load D Register												
LDX	0B	Load X Register												
LOR	03	Logical OR Memory into A Register												
MPY	1B	Multiply												
NOP	9201	No Operation												
PIN	01	Programmed Input												
POT	00	Programmed Output												
S	11	Shift												
SBY	07	Store Byte												
STA	06	Store A Register												
STD	05	Store D Register												
SUB	0E	Subtract Memory from A Register												

Note: See Appendix B for register-expandable derivative instructions.

**APPENDIX G**  
**SYSTEMS 72 INSTRUCTIONS**  
**NUMERICAL LISTING**

INSTRUCTION FORMAT																								
<table border="1" style="margin: auto; border-collapse: collapse;"> <tr> <td style="padding: 2px 5px;">R</td> <td style="padding: 2px 5px;">I</td> <td style="padding: 2px 5px;">X</td> <td style="padding: 2px 5px;">OP CODE</td> <td style="padding: 2px 5px;">S</td> <td style="padding: 2px 5px;">DISPLACEMENT</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td style="text-align: center;">2</td> <td style="text-align: center;">3</td> <td style="text-align: center;">4</td> <td style="text-align: center;">5</td> <td style="text-align: center;">6</td> <td style="text-align: center;">7</td> <td style="text-align: center;">8</td> <td style="text-align: center;">9</td> <td style="text-align: center;">10</td> <td style="text-align: center;">11</td> <td style="text-align: center;">12</td> <td style="text-align: center;">13</td> <td style="text-align: center;">14</td> <td style="text-align: center;">15</td> </tr> </table>			R	I	X	OP CODE	S	DISPLACEMENT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	I	X	OP CODE	S	DISPLACEMENT																			
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15									
<u>MNEMONIC</u>	<u>CODE</u>	<u>INSTRUCTION</u>																						
POT	00	Programmed Output																						
PIN	01	Programmed Input																						
AND	02	AND Memory to A Register																						
LOR	03	Logical OR Memory into A Register																						
EOR	04	Exclusive OR Memory into A Register																						
STD	05	Store D Register																						
STA	06	Store A Register																						
SBY	07	Store Byte																						
LDD	08	Load D Register																						
LDA	09	Load A Register																						
LBY	0A	Load Byte into A Register																						
LDX	0B	Load X Register																						
LDB	0C	Load B Register																						
ADD	0D	Add Memory to A Register																						
SUB	0E	Subtract Memory from A Register																						
INC	0F	Increment Memory by One																						
CMP	10	Compare A Register to Memory																						
S	11	Shift																						
B	12	Branch																						
BAL	13	Branch and Link																						
BIX	14	Increment X and Branch if Not Equal to Zero																						
BCR	15	Branch on Conditions Reset																						
BCS	16	Branch on Conditions Set																						
BRC	17	Branch Return and Clear																						
CAL1	18	Call 1																						
CAL2	19	Call 2 to Monitor Services																						
CAL3	1A	Call 3																						
MPY	1B	Multiply																						
DIV	1C	Divide																						
BGE	35	Branch if Equal To or Greater Than																						
BGEZ	35	Branch if Equal To or Greater Than Zero																						
BNO	35	Branch on No Overflow																						
BL	36	Branch if Less Than																						
BLZ	36	Branch if Less Than Zero																						
BO	36	Branch on Overflow																						
BNC	55	Branch on No Carry																						
BC	56	Branch on Carry																						
BNE	56	Branch if Not Equal To																						
BNEZ	56	Branch if Not Equal to Zero																						
BE	75	Branch if Equal																						
BEZ	75	Branch if Equal to Zero																						
CLA	0401	Clear A Register																						
NOP	9201	No Operation																						

Note: See Appendix B for register-expandable derivative instructions.

**APPENDIX H**  
**EFFECTIVE ADDRESS CALCULATION TIMES**  
(time in microseconds)

R	I	X	S	With High Speed Registers	Without High Speed Registers	Remarks
0	0	0	0	0	0	Absolute Addressing
1	X	X	X	0	0	Relative Addressing
X	1	X	X	1.00	1.00	Indirect Addressing- operand in core memory
X	1	X	X	0.50	1.00	Indirect Addressing- operand in A Register
X	X	1	X	0.75	1.25	Post-indexing
0	X	X	1	0.75	1.25	Pre-indexing (base addressing)

INST.	DESCRIPTION	NOTE	HIGH SPEED REGISTERS INCLUDED		HIGH SPEED REGISTERS EXCLUDED		
			INST. IN CORE			INST. IN REGISTER	
			Operand in Register	Operand in Core		Operand in Register	Operand in Core
ADD	ADD Memory to A Register	A	3.50	4.00	3.50	3.50	4.75
ADDB	ADD B to A	A	3.50	****	****	3.00	4.75
ADDD	ADD D to A	A	3.50	****	****	3.00	4.75
ADDE	ADD E to A	A	3.50	****	****	3.00	4.75
ADDX	ADD X to A	A	3.50	****	****	3.00	4.75
ADD1	ADD R1 to A	A	3.50	****	****	3.00	4.75
ADD2	ADD R2 to A	A	3.50	****	****	3.00	4.75
ADD3	ADD R3 to A	A	3.50	****	****	3.00	4.75
AND	AND Memory into A		3.50	4.00	3.50	3.00	4.75
ANDB	AND B into A	A	3.50	****	****	3.00	4.75
ANDD	AND D into A	A	3.50	****	****	3.00	4.75
ANDE	AND E into A	A	3.50	****	****	3.00	4.75
ANDX	AND X into A	A	3.50	****	****	3.00	4.75
AND1	AND R1 into A	A	3.50	****	****	3.00	4.75
AND2	AND R2 into A	A	3.50	****	****	3.00	4.75
AND3	AND R3 into A	A	3.50	****	****	3.00	4.75
B	Branch		2.25	****	****	1.75	2.25
BAL	Branch and Link		2.75	****	****	2.25	2.25
BC	Branch on Carry		2.25	****	****	2.25	3.00
BCR	Branch on Conditions Reset		2.25	****	****	1.75	2.25
BCS	Branch on Conditions Set		2.25	****	****	1.75	2.25
BE	Branch if Equal		2.25	****	****	1.75	2.25
BEZ	Branch if Equal to Zero		2.25	****	****	1.75	2.25
BGE	Branch if Greater than Equal		2.25	****	****	1.75	2.25
BGEZ	Branch if Greater than or Equal to Zero		2.25	****	****	1.75	2.25
BIX	Branch and Increment Index		2.25	****	****	1.75	2.25



INST.	DESCRIPTION	NOTE	HIGH SPEED REGISTERS INCLUDED				HIGH SPEED REGISTERS EXCLUDED
			INST. IN CORE		INST. IN REGISTER		
			Operand in Register	Operand in	Operand in Register	Operand in	
BL	Branch if Less Than		2.25	***	1.75	***	2.25
BLZ	Branch if Less Than Zero		2.25	***	1.75	***	2.25
BNC	Branch on No Carry		2.25	***	1.75	***	2.25
BNE	Branch if Not Equal		2.25	***	1.75	***	2.25
BNEZ	Branch if Not Equal to Zero		2.25	***	1.75	***	2.25
BNO	Branch on No Overflow		2.25	***	1.75	***	2.25
BO	Branch on Overflow		2.25	***	1.75	***	2.25
BRC	Branch Return and Clear		3.00	4.00	2.50	3.50	4.00
CAL1	Call 1	A	4.50	6.00	4.00	5.50	6.00
CAL2	Call 2	A	4.50	6.00	4.00	5.50	6.00
CAL3	Call 3	A	4.25	5.00	3.75	4.50	5.00
CLA	Clear A Register	A	***	3.50	***	***	3.50
HIO	Halt Input/Output	A	***	4.50	***	***	5.00
HLT	Halt	A	***	4.50	***	***	5.00
IOR	Input/Output Reset	A	***	4.50	***	***	5.00
MPY	Multiply		8.00	8.50	7.50	8.00	
NOP	No Operation	A	***	2.25	***	***	2.25
PIN	Programmed Input	C	3.00	3.00	2.50	2.50	3.25
POT	Programmed Output	C	3.50	3.50	3.00	3.00	4.00
RDS	Read Data Switches	A	***	4.00	***	***	4.25
RPS2	Read PSW2	A	***	4.00	***	***	4.25

INST.	DESCRIPTION	NOTE	HIGH SPEED REGISTERS INCLUDED		HIGH SPEED REGISTERS EXCLUDED		
			INST. IN CORE	INST. IN REGISTER			
			Operand in Register	Operand in Core	Operand in Register	Operand in Core	
CMP	Compare A Register Memory		3.00	3.50	2.50	3.00	4.00
CMPB	Compare A with B	A	3.00	****	2.50	****	4.00
CMPD	Compare A with D	A	3.00	****	2.50	****	4.00
CMPE	Compare A with E	A	3.00	****	2.50	****	4.00
CMPX	Compare A with X	A	3.00	****	2.50	****	4.00
CMP1	Compare A with R1	A	3.00	****	2.50	****	4.00
CMP2	Compare A with R2	A	3.00	****	2.50	****	4.00
CMP3	Compare A with R3	A	3.00	****	2.50	****	4.00
DIV	Divide		9.25	9.75	8.75	9.25	
INC	Increment Memory		3.00	3.75	2.50	3.25	3.75
INCA	Increment A	A	3.00	****	2.50	****	3.75
INCB	Increment B	A	3.00	****	2.50	****	3.75
INCD	Increment D	A	3.00	****	2.50	****	3.75
INCE	Increment E	A	3.00	****	2.50	****	3.75
INCX	Increment X	A	3.00	****	2.50	****	3.75
INC1	Increment R1	A	3.00	****	2.50	****	3.75
INC2	Increment R2	A	3.00	****	2.50	****	3.75
INC3	Increment R3	A	3.00	****	2.50	****	3.75
LOR	OR Memory into A Register		3.50	4.00	3.00	3.50	4.75
LORB	OR B into A	A	3.50	****	3.00	****	4.75
LORD	OR D into A	A	3.50	****	3.00	****	4.75
LORE	OR E into A	A	3.50	****	3.00	****	4.75

INST.	DESCRIPTION	NOTE	HIGH SPEED REGISTERS INCLUDED				HIGH SPEED REGISTERS EXCLUDED
			INST. IN CORE		INST. IN REGISTER		
			Operand in Register	Operand in Core	Operand in Register	Operand in Core	
LOR1	OR R1 into A	A	3.50	****	3.00	****	4.75
LOR2	OR R2 into A	A	3.50	****	3.00	****	4.75
LOR3	OR R3 into A	A	3.50	****	3.00	****	4.75
EOR	Exclusive OR Memory into A Register		4.25	4.75	3.75	4.25	5.50
EORB	Exclusive OR B into A	A	4.25	****	3.75	****	5.50
EORD	Exclusive OR D into A	A	4.25	****	3.75	****	5.50
EORE	Exclusive OR E into A	A	4.25	****	3.75	****	5.50
EORX	Exclusive OR X into A	A	4.25	****	3.75	****	5.50
EOR1	Exclusive OR R1 into A	A	4.25	****	3.75	****	5.50
EOR2	Exclusive OR R2 into A	A	4.25	****	3.75	****	5.50
EOR3	Exclusive OR R3 into A	A	4.25	****	3.75	****	5.50
LBV	Load Byte into A Register		3.00	3.50	2.50	3.00	4.00
LBVB	Load Byte into A from B	A	3.00	****	2.50	****	4.00
LBVD	Load Byte into A from D	A	3.00	****	2.50	****	4.00
LBVE	Load Byte into A from E	A	3.00	****	2.50	****	4.00
LBVX	Load Byte into A from X	A	3.00	****	2.50	****	4.00
LBV1	Load Byte into A from R1	A	3.00	****	2.50	****	4.00
LBV2	Load Byte into A from R2	A	3.00	****	2.50	****	4.00
LBV3	Load Byte into A from R3	A	3.00	****	2.50	****	4.00

INST.	DESCRIPTION	NOTE	HIGH SPEED REGISTERS INCLUDED				HIGH SPEED REGISTERS EXCLUDED
			INST. IN CORE		INST. IN REGISTER		
			Operand in Register	Operand in Core	Operand in Register	Operand in Core	
LDA	Load A Register		2.75	3.25	2.25	3.00	3.50
LDAB	Load A from B	A	2.75	****	2.25	****	3.50
LDAD	Load A from D	A	2.75	****	2.25	****	3.50
LDAE	Load A from E	A	2.75	****	2.25	****	3.50
LDAX	Load A from X	A	2.75	****	2.25	****	3.50
LDA1	Load A from R1	A	2.75	****	2.25	****	3.50
LDA2	Load A from R2	A	2.75	****	2.25	****	3.50
LDA3	Load A from R3	A	2.75	****	2.25	****	3.50
LDB	Load B Register		2.75	3.25	2.25	3.00	3.50
LDBA	Load B from A	A	2.75	****	2.25	****	3.50
LDBD	Load B from D	A	2.75	****	2.25	****	3.50
LDBE	Load B from E	A	2.75	****	2.25	****	3.50
LDBX	Load B from X	A	2.75	****	2.25	****	3.50
LDB1	Load B from R1	A	2.75	****	2.25	****	3.50
LDB2	Load B from R2	A	2.75	****	2.25	****	3.50
LDB3	Load B from R3	A	2.75	****	2.25	****	3.50
LDD	Load D Register		2.75	3.25	2.25	3.00	3.50
LDDA	Load D from A	A	2.75	****	2.25	****	3.50
Lddb	Load D from B	A	2.75	****	2.25	****	3.50
LDDE	Load D from E	A	2.75	****	2.25	****	3.50
LDDX	Load D from X	A	2.75	****	2.25	****	3.50
LDD1	Load D from R1	A	2.75	****	2.25	****	3.50
LDD2	Load D from R2	A	2.75	****	2.25	****	3.50
LDD3	Load D from R3	A	2.75	****	2.25	****	3.50
LDR	Load Display Register	A	****	4.50	****	****	5.00

INST.	DESCRIPTION	NOTE	HIGH SPEED REGISTERS INCLUDED				HIGH SPEED REGISTERS EXCLUDED
			INST. IN CORE		INST. IN REGISTER		
			Operand in Register	Operand in Core	Operand in Register	Operand in Core	
LDX	Load X Register	A	2.75	3.25	2.25	3.00	3.50
LDXA	Load X from A	A	2.75	****	2.25	****	3.50
LDXB	Load X from B	A	2.75	****	2.25	****	3.50
LDXD	Load X from D	A	2.75	****	2.25	****	3.50
LDXE	Load X from E	A	2.75	****	2.25	****	3.50
LDX1	Load X from R1	A	2.75	****	2.25	****	3.50
LDX2	Load X from R2	A	2.75	****	2.25	****	3.50
LDX3	Load X from R3	A	2.75	****	2.25	****	3.50
LPS2	Load PSW2	A	****	4.50	****	****	5.00
SBY	Store Byte from A Register		4.25	4.75	3.75	4.25	5.50
SBYB	Store Byte from A into B	A	4.25	****	3.75	****	5.50
SBYD	Store Byte from A into D	A	4.25	****	3.75	****	5.50
SBYE	Store Byte from A into E	A	4.25	****	3.75	****	5.50
SBYX	Store Byte from A into X	A	4.25	****	3.75	****	5.50
SBY1	Store Byte from A into R1	A	4.25	****	3.75	****	5.50
SBY2	Store Byte from A into R2	A	4.25	****	3.75	****	5.50
SBY3	Store Byte from A into R3	A	4.25	****	3.75	****	5.50

INST.	DESCRIPTION	NOTE	HIGH SPEED REGISTERS INCLUDED			HIGH SPEED REGISTERS EXCLUDED		
			INST. IN CORE				INST. IN REGISTER	
			Operand in Register	Operand in Core	Operand in Register		Operand in Core	
STA	Store A Register		3.00	3.25	2.50	2.75	3.75	
STAB	Store A into B	A	3.00	****	2.50	****	3.75	
STAD	Store A into D	A	3.00	****	2.50	****	3.75	
STAE	Store A into E	A	3.00	****	2.50	****	3.75	
STAX	Store A into X	A	3.00	****	2.50	****	3.75	
STA1	Store A into R1	A	3.00	****	2.50	****	3.75	
STA2	Store A into R2	A	3.00	****	2.50	****	3.75	
STA3	Store A into R3	A	3.00	****	2.50	****	3.75	
STD	Store D Register		3.00	3.25	2.50	2.75	3.75	
STDA	Store D into A	A	3.00	****	2.50	****	3.75	
STDB	Store D into B	A	3.00	****	2.50	****	3.75	
STDE	Store D into E	A	3.00	****	2.50	****	3.75	
STDX	Store D into X	A	3.00	****	2.50	****	3.75	
STD1	Store D into R1	A	3.00	****	2.50	****	3.75	
STD2	Store D into R2	A	3.00	****	2.50	****	3.75	
STD3	Store D into R3	A	3.00	****	2.50	****	3.75	
SUB	Subtract Memory from A Register		3.50	4.00	3.00	3.50	4.25	
SUBB	Subtract B from A	A	3.50	****	3.00	****	4.25	
SUBD	Subtract D from A	A	3.50	****	3.00	****	4.25	
SUBE	Subtract E from A	A	3.50	****	3.00	****	4.25	
SUBX	Subtract X from A	A	3.50	****	3.00	****	4.25	
SUB1	Subtract R1 from A	A	3.50	****	3.00	****	4.25	
SUB2	Subtract R2 from A	A	3.50	****	3.00	****	4.25	
SUB3	Subtract R3 from A	A	3.50	****	3.00	****	4.25	

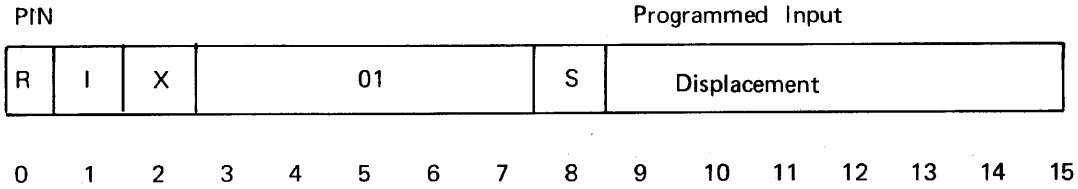
INST.	DESCRIPTION	NOTE	HIGH SPEED REGISTERS INCLUDED				HIGH SPEED REGISTERS EXCLUDED
			INST. IN CORE		INST. IN REGISTER		
			Operand in Register	Operand in Core	Operand in Register	Operand in Core	
S	Shift - single	D		3.25		2.75	4.00
S	Shift - double	D		5.00		4.50	7.75
SAL	Shift Arithmetic Left	D,A	***	3.25	***	2.75	4.00
SALD	Shift Arithmetic Left Double	D,A	***	6.00	***	5.50	8.75
SAR	Shift Arithmetic Right	D,A	***	3.25	***	2.75	4.00
SARD	Shift Arithmetic Right Double	D,A	***	6.00	***	5.50	8.75
SCL	Shift Circular Left	D,A	***	3.25	***	2.75	4.00
SCLD	Shift Circular Left Double	D,A	***	6.00	***	5.50	8.75
SCR	Shift Circular Right	D,A	***	3.25	***	2.75	4.00
SCRD	Shift Circular Right Double	D,A	***	6.00	***	5.50	8.75
SLL	Shift Logical Left	D,A	***	3.35	***	2.75	4.00
SLLD	Shift Logical Left Double	D,A	***	6.00	***	5.50	8.75
SLR	Shift Logical Right	D,A	***	3.25	***	2.75	4.00
SLRD	Shift Logical Right Double	D,A	***	6.00	***	5.50	8.75
SIO	Start Input/Output	A	***	4.50	***	***	5.00
TDV	Test Device	A	***	4.00	***	***	4.25
TIO	Test Input/Output Interrupt Response	A	***	4.50	***	***	5.00
			4.00	***	***	***	6.00

## Notes

- A Includes time for all effective address calculations
- B "\*\*\*\*" indicates that this situation is not possible or not probable
- C Plus delay imposed by external device
- D Plus 0.25 microseconds for each bit position shifted



**APPENDIX I**  
**SYSTEMS 72 SPECIAL DERIVATIVES INSTRUCTIONS**



The effective address serves two purposes. It drives the Programmed Input/Output Bus Address lines, and it provides an augment code for the many derivatives of PIN. The Programmed Input/Output Bus data lines are loaded into the D register. The special derivatives of PIN for input output operation are as follows.

Test I/O instruction TIO

PIN        EFA=X'0180' + DADD  
 where DADD is the device address  
 affected:    CC1 if controller busy  
               CC2 if device busy

Test Device instruction TDV

PIN        EFA = X'0100' + DADD  
 where DADD is the device address  
 affected:    CC1 if controller busy  
               CC2 if device busy  
               D contains device status

Order in instruction

PIN        EFA = X'0080' + DADD  
 where DADD is the device address  
 affected:    CC1 if controller busy  
               CC2 if device busy  
               D contains device status

Data In Word instruction

PIN        EFA = 0000 0000 00DD DDDD  
 where DD DDDD is the device address  
 affected:    CC1 if Burst mode  
               CC2 if data not available  
               D contains one data word

Data In Byte instruction

PIN        EFA = 0000 0000 01DD DDDD  
 where DD DDDD is the device address  
 affected:    CC1 if Burst mode  
               CC2 if data not available  
               D contains input data in byte 1

The special derivatives of PIN for display panel inspection are as follows:

Read Display Switch instruction

PIN EFA = X'0400'

CC1 and CC2 unchanged

affected: D contains the contents of the register specified by the display switch, or the contents of the memory location defined by the data switches.

Read Data Switches instruction

PIN EFA = X'0401'

CC1 and CC2 unchanged

affected: D contains the contents of the data switches

Read PSW2 instruction

PIN EFA = X'0402'

CC1 and CC2 unchanged

affected: D contains the contents of PSW2

The special derivative of PIN for the Console Interrupt option is as follows.

Read Console Interrupt Status instruction

PIN EFA = X'0404'

CC1 and CC2 unchanged

affected: D contains the console interrupt status

The special derivatives of PIN for the Direct Access Channel are as follows.

Sense DAC Address instruction

PIN EFA = X'0405'

affected: CC1 if DAC busy

CC2 always zero

D contains dynamic data address

Sense DAC Word Count instruction

PIN EFA = X'0406'

affected: CC1 if DAC busy

CC2 always zero

D contains dynamic word count

Sense DAC Status instruction

PIN EFA = X'0409'

affected: CC1 if DAC busy

CC2 if core access error

D contains current device status

The special derivatives of PIN for the Real-Time Clock option are as follows.

Read RTC time Instruction

PIN EFA = X'0208' + N

where N is the clock address

affected: CC1 and CC2 always zero

D contains the current time

Sense RTC Interval instruction

PIN EFA = X'020C' + N

where N is the clock address

affected: CC1 and CC2 always zero

D interrupt interval

Sense RTC Status instruction

PIN EFA = X'0210' + N

where N is the clock status

affected: CC1 and CC2 always zero

The special derivative of PIN for the Interrupt Pair option is as follows.

Sense Interrupt Status instruction

PIN EFA = X'200' + IADD even

where IADD is the interrupt address

affected: CC1 and CC2 unchanged

D contains current interrupt status Even in byte 0, Odd in byte 1

The special derivative of PIN for memory mapping is as follows.

Read Snapshot Register instruction

PIN EFA = X'0F00' or X'0E10'

affected: CC1 if page fault

CC2 if protect violation

CC1 and CC2 zero if last trap

was memory parity error

D contains the contents of the snapshot register

The special derivatives of PIN for page transfers are as follows.

Test Page Transfer instruction

PIN EFA = X'0E04'

affected: CC1 if MED busy

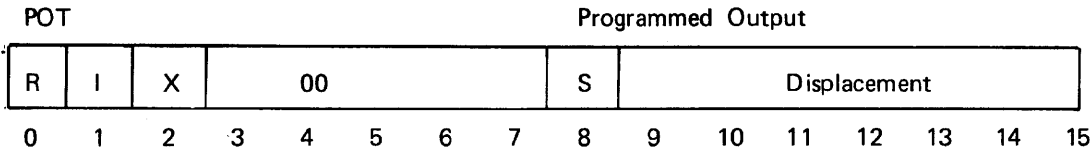
CC2 always zero

D contains MED status

Test Disc Position instruction

PIN EFA = X'0E05'

affected: CC1 and CC2 equal sector quarter D contains current sector number



The effective address serves two purposes. It drives the Programmed Input/Output Bus address lines, and it provides an augment code for the many derivatives of POT. The Programmed Input/Output Bus data lines are driven by the D register . The special derivatives of POT for input/output operation are as follows.

Start I/O instruction      SIO  
 POT      EFA = X'0100' + DADD  
 where DADD is the device address  
 affected:    CC1 if controller busy  
               CC2 if device busy  
               D is meaningless

Order Out 1 instruction  
 POT      EFA is X'0080' + DADD  
 where DADD is the device address  
 affected:    CC1 if controller busy  
               CC2 if device busy  
               D contains IOCL word 1

Order Out 2 instruction  
 POT      EFA is X'00C0' + DADD  
 Where DADD is the device address  
 affected:    CC1 if controller busy  
               CC2 if device busy  
               D contains IOCL word 2

Terminal Order instruction  
 POT      EFA = X'01C0' + DADD  
 where DADD is the device address  
 affected:    CC1 if controller busy  
               CC2 is device busy  
               D contains terminal order

Data Output Word instruction  
 POT      EFA = X'0000' + DADD  
 where DADD is the device address  
 affected:    CC1 if Burst mode  
               CC2 if data not accepted  
               D contains data to be output

Data Output Byte instruction  
 POT      EFA = X'0040' + DADD  
 where DADD is the device address  
 affected:    CC1 if Burst mode  
               CC2 if data not accepted  
               D contains data to be output  
               from byte 1

Reset I/O instruction                    IOR  
POT            EFA = X' 0404'  
affected:    CC1 if any controller busy  
              CC2 if any device busy  
              D is meaningless

The special derivatives of POT for display panel operation are as follows.

Load Display Register instruction  
POT            EFA = X'0400'  
affected:    CC1 and CC2 unchanged  
              D contains data to be displayed

Halt Program instruction                    HLT  
POT            EFA = X'0401'  
affected:    CC1 and CC2 always zero  
              D is meaningless

Load PSW2 instruction  
POT            EFA = X'0402'  
affected:    CC1 and CC2 unchanged  
              D contains data for PSW2

The special derivative of POT for the Console Interrupt option is as follows.

Set Console Interrupt Status instruction  
POT            EFA = X'0404'  
affected:    CC1 and CC2 unchanged  
              D contains the interrupt status

The special derivatives of POT for the Direct Access Channel are as follows.

Load DAC Address instruction  
POT            EFA = X'0405'  
affected:    CC1 if DAC busy  
              CC2 always zero  
              D contains initial data address

Load DAC Word Count instruction

POT EFA = X'0406'  
affected: CC1 if DAC busy  
          CC2 always zero  
          D contains initial word count

Start DAC Transfer instruction

POT EFA = X'0407'  
affected: CC1 if DAC busy  
          CC2 always zero  
          D is meaningless

Halt DAC Transfer instruction

POT EFA is X'0408'  
affected: CC1 if DAC busy  
          CC2 always zero  
          D is meaningless

Set DAC Status instruction

POT EFA = X'0409'  
affected: CC1 if DAC busy  
          CC2 always zero  
          D contains DAC new status

The special derivatives of POT for the Real-Time Clock option are as follows.

Start RTC Clock instruction

POT EFA = X'0200' + N  
where N is the clock address  
affected: CC1 and CC2 always zero  
          D is meaningless

Stop RTC Clock instruction

POT EFA = X'0204' + N  
where N is the clock address  
affected: CC1 and CC2 always zero  
          D is meaningless

Continue RTC Clock instruction

POT EFA = X'0208' + N  
where N is the clock address  
affected: CC1 and CC2 always zero  
          D is meaningless

Set RTC Status instruction

POT EFA = X'0210' + N

where N is the clock address

affected: CC1 and CC2 always zero  
D contains initial count

The special derivative of POT for the Interrupt Pair option is as follows.

Set Interrupt Status instruction

POT EFA = X'200' + IADD even

where IADD is the interrupt address

affected: CC1 and CC2 unchanged  
D contains initial interrupt status Even in byte 0, Odd in byte 1

The special derivative of POT for memory mapping is as follows.

Memory Map Update

POT EFA = X'0F00' + PADD

where PADD is the virtual page address

affected: CC1 and CC2 unchanged  
D contains new map data

The special derivatives of POT for page transfers are as follows.

Core-to-Disc Mapped instruction

POT EFA = X'0E00'

affected: CC1 if MED busy  
CC2 always zero  
D contains the core page address

Disc-to-Core Mapped instruction

POT EFA = X'0E01'

affected: CC1 if MED busy  
CC2 always zero  
D contains the core page address

Core-to-Disc Unmapped instruction

POT EFA = X'0E02'

affected: CC1 if MED busy  
CC2 always zero  
D contains the core page address

Disc to Core Unmapped instruction

POT      EFA = X'0E03'  
affected: CC1 if MED busy  
          CC2 always zero  
          D contains the core page address

Start Page Transfer instruction

POT      EFA = X'0E05'  
affected: CC1 if MED busy  
          CC2 always zero  
          D contains the disc page address

Halt Page Transfer instruction

POT      EFA = X'0E05'  
affected: CC1 if transfer complete  
          CC2 if transfer error  
          D is meaningless