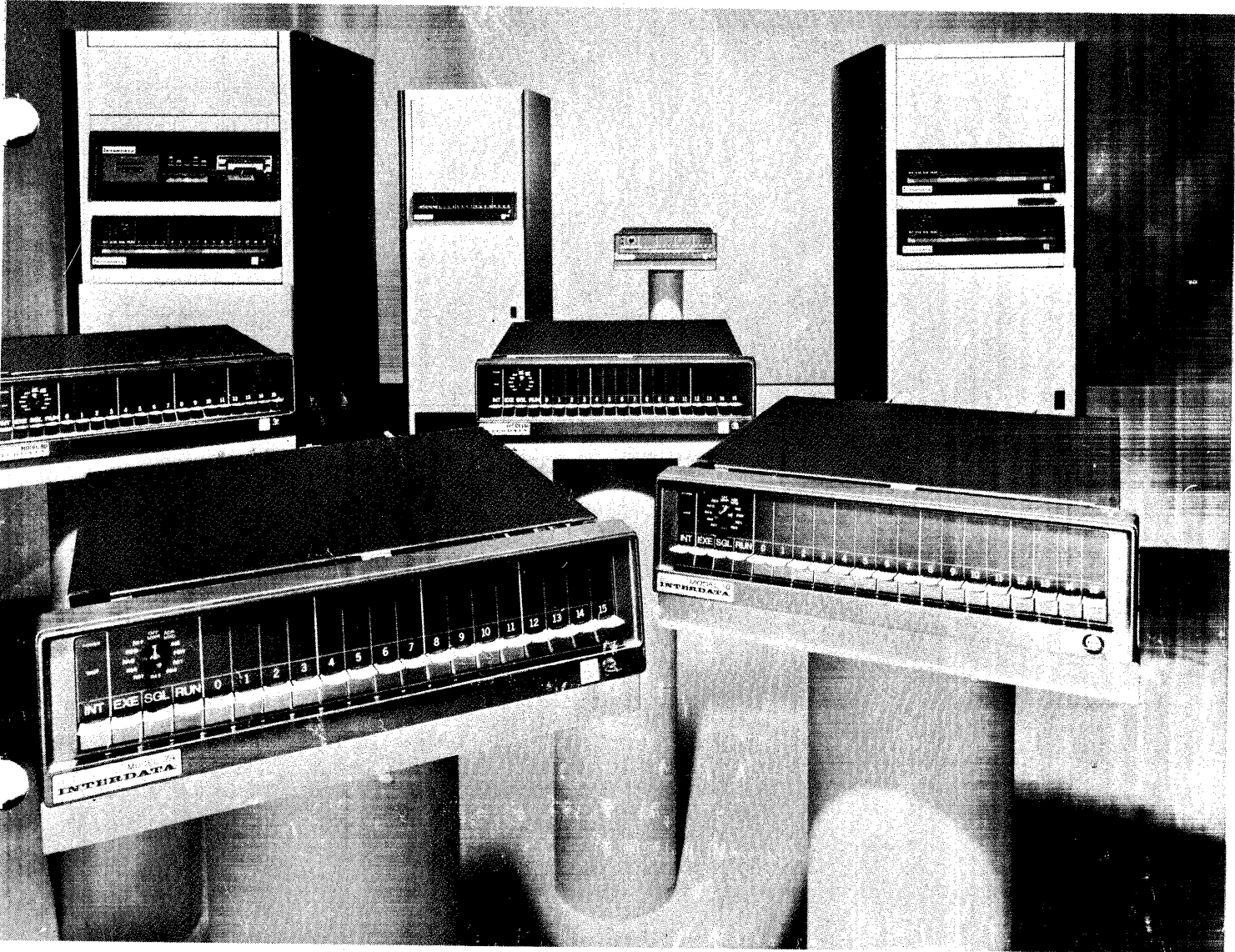


User's Manual

INTERDATA®













User's Manual

Publication Number 29-261R01

**INFORMATION CONTAINED IN THIS
MANUAL IS SUBJECT TO DESIGN
CHANGE OR PRODUCT IMPROVEMENT**

QUICK REFERENCE INDEX

To aid in quickly locating a particular chapter, the index marks on the edge of this page are aligned with similar marks on the first page of each chapter.

Chapter 1	INTRODUCTION	
Chapter 2	PROCESSOR DESCRIPTION	
Chapter 3	DATA AND INSTRUCTION FORMATS AND STORAGE ADDRESSING	
Chapter 4	INSTRUCTION REPERTOIRE	
Chapter 5	INTERFACE DESIGN	
Chapter 6	CONTROL CONSOLE	
Chapter 7	INTERDATA SOFTWARE FAMILY	
Chapter 8	PERIPHERAL DEVICES AND MODULES	
Chapter 9	CONFIGURATION INSTALLATION PLANNING	
	APPENDICES	

USER'S MANUAL

TABLE OF CONTENTS

Chapter		Page
1	INTRODUCTION	1-1
	1.1 The INTERDATA Family of Computers	1-1
	1.2 System Description	1-2
	1.2.1 Memories	1-2
	1.2.2 Direct Memory Access	1-3
	1.2.3 Selector Channel	1-3
	1.2.4 The Processor	1-3
	1.2.5 The Multiplexor Input/Out Bus	1-3
	1.3 Peripherals	1-3
	1.3.1 Digital Multiplexor	1-3
	1.3.2 INTERTAPE Cassette System	1-4
	1.3.3 IBM Compatible Magnetic Tapes	1-4
	1.3.4 Removable Cartridge Disc System	1-4
	1.3.5 Magnetic Drum Storage System	1-4
	1.3.6 Data Communications Equipment	1-4
	1.3.7 High Speed Paper Tape System	1-4
	1.3.8 System Modules	1-4
	1.4 Software	1-4
	1.4.1 Assembler	1-4
	1.4.2 Loader Descriptions	1-5
	1.4.3 Text Editor (TIDE)	1-5
	1.4.4 Debug (CLUB)	1-6
	1.4.5 FORTRAN (Models 70 and 80 only)	1-6
	1.4.6 Basic Operating System (BOSS)	1-6
	1.4.7 Real Time Operating System (RTOS) (Models 70 and 80 only)	1-6
	1.4.8 Test Programs	1-6
	1.5 Customer Support	1-6
	1.5.1 Computation Center Facilities	1-6
	1.5.2 Field Service	1-7
	1.5.3 Training Center	1-7
	1.5.4 Systems Engineering and Programming	1-7
	1.5.5 INTERCHANGE	1-7
2	PROCESSOR DESCRIPTION	2-1
	2.1 Micro-Programmed INTERDATA Processor Architecture	2-1
	2.2 Processor Block Diagram	2-1
	2.3 Processor Operation	2-3
	2.3.1 Program Status Words	2-3
	2.3.2 Instruction Execution	2-3
	2.3.3 Main Memory Allocation	2-4

TABLE OF CONTENTS

(Continued)

Chapter		Page
2.4	Interrupt System	2-6
2.4.1	Interrupt Procedure	2-6
2.4.2	Internal Interrupts	2-7
2.4.3	Input/Output Control Interrupts	2-8
2.4.4	The Automatic Input/Output Channel (Models 70 and 80 only)	2-10
2.4.5	Special Interrupts	2-19
3	DATA AND INSTRUCTION FORMATS AND STORAGE ADDRESSING	3-1
3.1	Introduction	3-1
3.2	Data Formats	3-1
3.2.1	Hexidecimal Notation	3-1
3.2.2	Fixed-Point Data	3-1
3.2.3	Floating-Point Data	3-3
3.2.4	Logical Data	3-4
3.3	Instruction Formats	3-5
3.4	General Register Usage	3-6
3.5	Storage Addressing	3-7
4	INSTRUCTION REPERTOIRE	4-1
4.1	Introduction	4-1
4.2	Fixed-Point Load/Store Instructions	4-3
4.2.1	Load Halfword	4-4
4.2.2	Load Multiple	4-5
4.2.3	Store Halfword	4-5
4.2.4	Store Multiple	4-6
4.3	Fixed-Point Arithmetic Instructions	4-7
4.3.1	Add Halfword	4-8
4.3.2	Add with Carry Halfword	4-9
4.3.3	Subtract Halfword	4-10
4.3.4	Subtract with Carry Halfword	4-11
4.3.5	Compare Logical Halfword	4-12
4.3.6	Compare Halfword	4-13
4.3.7	Multiply Halfword	4-14
4.3.8	Multiply Halfword Unsigned	4-14
4.3.9	Divide Halfword	4-15
4.4	Logical and Bit Manipulating Instructions	4-16
4.4.1	AND Halfword	4-17
4.4.2	OR Halfword	4-18
4.4.3	Exclusive OR Halfword	4-19
4.4.4	Test Halfword Immediate	4-20

TABLE OF CONTENTS

(Continued)

Chapter		Page
4.5	Byte Handling Instructions	4-21
4.5.1	Load Byte	4-22
4.5.2	Store Byte	4-22
4.5.3	Exchange Byte	4-23
4.5.4	Compare Logical Byte	4-23
4.6	Shift/Rotate Instructions	4-24
4.6.1	Shift Left Logical	4-25
4.6.2	Shift Right Logical	4-26
4.6.3	Rotate Left Logical	4-27
4.6.4	Rotate Right Logical	4-28
4.6.5	Shift Left Arithmetic	4-29
4.6.6	Shift Right Arithmetic	4-30
4.7	Branch Instructions	4-31
4.7.1	Branch on True Condition	4-32
4.7.2	Branch on False Condition	4-33
4.7.3	Branch on Index	4-34
4.7.4	Branch and Link	4-35
4.8	Input/Output Instructions	4-36
4.8.1	Acknowledge Interrupt	4-37
4.8.2	Sense Status	4-38
4.8.3	Output Command	4-39
4.8.4	Read Data	4-39
4.8.5	Write Data	4-40
4.8.6	Read Halfword	4-41
4.8.7	Write Halfword	4-42
4.8.8	Autoload	4-43
4.9	Block Input/Output Instructions	4-44
4.9.1	Read Block	4-45
4.9.2	Write Block	4-46
4.10	System Control Instructions	4-47
4.10.1	Load Program Status Word	4-47
4.10.2	Exchange Program Status	4-48
4.10.3	Simulate Interrupt	4-48
4.10.4	Supervisor Call	4-49
4.11	Floating-Point Instructions (Models 70 and 80 only)	4-50
4.11.1	Floating-Point Load	4-51
4.11.2	Floating-Point Store	4-51
4.11.3	Floating-Point Add	4-52
4.11.4	Floating-Point Subtract	4-53
4.11.5	Floating-Point Compare	4-54
4.11.6	Floating-Point Multiply	4-55
4.11.7	Floating-Point Divide	4-56
4.12	List Processing Instructions (Models 70 and 80 only)	4-57
4.12.1	Add to Top/Bottom of List	4-58
4.12.2	Remove From Top/Bottom of List	4-59

TABLE OF CONTENTS

(Continued)

Chapter		Page
5	INTERFACE DESIGN	5-1
5.1	Introduction	5-1
5.2	Systems Interface	5-1
5.2.1	Multiplexor Channel	5-1
5.2.2	Interleaved Data Channel (Models 70 and 80 only)	5-5
5.3	Device Controller Logic Design	5-5
5.3.1	Multiplexor Bus	5-5
5.3.2	Device Controller Addressing	5-9
5.3.3	Data and Status Input	5-9
5.3.3.1	Data	5-9
5.3.3.2	Status	5-9
5.3.4	Data and Command Output	5-12
5.3.4.1	Data	5-12
5.3.4.2	Command	5-12
5.3.5	Interrupt Control	5-12
5.3.6	Multiplexor Bus Wiring	5-15
5.3.7	Multiplexor Channel Timing	5-15
5.3.8	General Multiplexor Bus Interface	5-18
5.3.9	Interleaved Data Channel Interface Design	5-18
5.4	Memory Bus	5-23
5.4.1	Introduction	5-23
5.4.2	Memory Bus Lines	5-24
5.5	Selector Channel	5-29
5.6	General Purpose Interface	5-33
5.6.1	Universal Interface Module	5-33
5.6.2	General Purpose Interface Board	5-35
5.6.2.1	Component Field Numbering	5-35
5.6.2.2	Connector Layout	5-35
5.6.2.3	Available Cables	5-35
6	CONTROL CONSOLE	6-1
6.1	Introduction	6-1
6.2	Control Console Description	6-1
6.2.1	Key Operated Security Lock	6-1
6.2.2	Control Switches	6-1
6.2.3	Function Switches	6-2

TABLE OF CONTENTS

(Continued)

Chapter		Page
6.3	Control Console Operating Procedures	6-3
6.3.1	Power Up	6-3
6.3.2	Power Down	6-5
6.3.3	Program Loading	6-5
6.3.4	Program Execution	6-6
6.3.5	Program Termination	6-7
6.3.6	Manually Initiated Memory Operations	6-7
6.3.6.1	Memory Read	6-7
6.3.6.2	Memory Write	6-7
6.4	Programming Considerations	6-8
6.4.1	Control Console I/O	6-8
6.4.2	Console Interrupt	6-9
6.4.3	Wait State	6-9
6.4.4	Power Fail	6-9
7	INTERDATA SOFTWARE FAMILY	7-1
7.1	Introduction	7-1
7.2	Comparison of OS vs Stand-Alone Programs	7-2
7.3	Program Preparation	7-3
7.3.1	Programming in an Operating System Environment	7-3
7.3.2	Programming in a Stand-Alone Environment	7-7
7.4	Programming Conventions	7-9
7.5	INTERDATA Assembler Program	7-9
7.5.1	General Description	7-9
7.5.2	Assembly Procedures	7-12
7.5.3	Assembler Language	7-13
7.5.4	Machine Instruction Format	7-18
7.5.5	Assembler Instructions (Pseudo Ops)	7-21
7.5.6	Assembly Listing and Object Programs	7-38
7.5.7	Procedures for User-Defined Mnemonic Op-Codes	7-41
7.5.8	Basic Assembler Operating Instructions	7-42
7.5.9	OS Assembler Operating Instructions	7-51
7.6	Assembly Level Programming Techniques	7-61
7.7	FORTRAN IV	7-64
7.7.1	General Description	7-64
7.7.2	FORTRAN Language Specifications	7-64
7.7.3	Loading the FORTRAN Compiler	7-65
7.7.4	Memory Requirements	7-67
7.7.5	Compiler Execution Procedures	7-69
7.7.6	Loading a Compiled Program	7-69

TABLE OF CONTENTS

(Continued)

Chapter		Page
7.8	Interactive FORTRAN	7-70
	7.8.1 General Description	7-70
	7.8.2 Features	7-72
	7.8.3 Use of the System	7-72
	7.8.4 System Capacity	7-73
7.9	Loader Descriptions	7-74
	7.9.1 General	7-74
	7.9.2 50 Sequence Bootstrap Loader	7-77
	7.9.3 Object Tape Format	7-81
	7.9.4 Features of the OS Library Loader	7-82
	7.9.5 Features of Stand-Alone Loaders	7-89
	7.9.6 General Loader Features	7-90
	7.9.7 Operation of Stand-Alone Loaders	7-92
7.10	Editor (TIDE) Program	7-93
	7.10.1 Program Structure	7-94
	7.10.2 Operating Procedures	7-101
7.11	Hexadecimal Debug (CLUB) Program	7-104
	7.11.1 Terminology	7-104
	7.11.2 Description of Operations	7-105
	7.11.3 Bias Definition	7-105
	7.11.4 Cell Examination and Modification	7-106
	7.11.5 Program Control	7-108
	7.11.6 Utilities	7-111
	7.11.7 Operating Procedures	7-115
7.12	Basic Operating System	7-118
	7.12.1 General Description	7-118
	7.12.2 Operational Characteristics	7-119
	7.12.3 Programmable Commands	7-119
	7.12.4 Operator Commands	7-120
	7.12.5 System Configuration	7-120
7.13	Disc Operating System	7-121
	7.13.1 Introduction	7-121
	7.13.2 Features	7-121
	7.13.3 Description	7-122
	7.13.4 DOS Programmable Commands	7-123
	7.13.5 DOS Operator Commands	7-124
	7.13.6 Configuration	7-125
7.14	Real Time Operating System	7-125
	7.14.1 Introduction	7-125
	7.14.2 Features and Characteristics	7-125
	7.14.3 System Concepts	7-126
	7.14.4 System Organization	7-129

TABLE OF CONTENTS

(Continued)

Chapter		Page
8	PERIPHERAL DEVICES AND MODULES	8-1
8.1	Introduction	8-1
8.2	Teletypewriters	8-1
8.2.1	Introduction	8-1
8.2.2	Configuration	8-2
8.2.3	Operating Procedures	8-2
8.2.4	Data Format	8-4
8.2.5	Programming Instructions	8-5
8.2.6	Programming Sequences	8-7
8.2.7	Interrupts	8-8
8.2.8	Initialization	8-8
8.2.9	Device Number	8-11
8.3	High Speed Paper Tape Reader/Punch (HSPTR)	8-11
8.3.1	General Description	8-11
8.3.2	Status and Command	8-12
8.3.3	Interrupts	8-13
8.3.4	Initialization	8-13
8.3.5	Device Number	8-13
8.4	Card Reader	8-14
8.4.1	General Description	8-14
8.4.2	Operator Controls	8-14
8.4.3	Status Indicator Lights	8-14
8.4.4	Status and Command Bytes	8-15
8.4.5	Data Format	8-15
8.4.6	Interrupts	8-15
8.4.7	Initialization	8-15
8.4.8	Operator Procedures	8-15
8.4.9	Programming	8-17
8.5	Removable Cartridge Disc System	8-18
8.5.1	General Description	8-18
8.5.2	Operational Characteristics	8-18
8.5.3	Disc Format	8-18
8.5.4	Data Transfers	8-19
8.5.5	Specifications	8-20
8.6	201 Synchronous Data Set Adapter	8-22
8.6.1	General Description	8-22
8.6.2	Operational Characteristics	8-22
8.6.3	Specifications	8-23
8.7	Programmable Asynchronous Line System (PALS)	8-24
8.7.1	Introduction	8-24
8.7.2	Data Format	8-25
8.7.3	Programming Instructions	8-27
8.7.4	Programming Sequences	8-33
8.7.5	Interrupts	8-34
8.7.6	Initialization	8-35
8.7.7	Device Number	8-35

TABLE OF CONTENTS

(Continued)

Chapter		Page
8.8	INTERTAPE Cassette System	8-36
	8.8.1 Specifications	8-36
8.9	Automatic Memory Protect Controller	8-37
	8.9.1 General Description	8-37
	8.9.2 Operational Characteristics	8-37
	8.9.3 Specifications	8-38
8.10	Universal Clock Module	8-39
	8.10.1 General Description	8-39
	8.10.2 Operational Characteristics	8-39
	8.10.3 Specifications	8-40
8.11	The Eight Line Interrupt Module	8-41
	8.11.1 General Description	8-41
	8.11.2 Operational Characteristics	8-41
	8.11.3 Specifications	8-42
8.12	Serial Line Printer	8-42
	8.12.1 General Description	8-42
	8.12.2 Operational Characteristics	8-42
	8.12.3 Specifications	8-43
8.13	Loader Storage Unit (Model Numbers M70-104, M70-105)	8-43
	8.13.1 Summary	8-43
	8.13.2 Functional Description	8-44
	8.13.3 Applicable Processors	8-44
	8.13.4 Ordering Procedure	8-44
8.14	Automatic Loader for Basic Model 74 (Model No. M74-101)	8-44
	8.14.1 Summary	8-44
	8.14.2 Functional Description	8-44
8.15	Selector Channel	8-45
	8.15.1 General Description	8-45
	8.15.2 Programming Instructions	8-45
	8.15.3 Programming Sequences	8-46
	8.15.4 Interrupts	8-49
	8.15.5 Initialization	8-49
	8.15.6 Device Number	8-49
	8.15.7 Sample Program	8-50
9	CONFIGURATION INSTALLATION PLANNING	9-1
	9.1 Introduction	9-1
	9.2 Integrated Circuit Boards	9-1
	9.3 Basic Processor Chassis	9-2
	9.4 System Expansion Chassis	9-3
	9.4.1 15 Inch System Expansion Chassis	9-3
	9.4.2 10 Inch System Expansion Chassis	9-3

TABLE OF CONTENTS

(Continued)

Chapter		Page
9.5	Circuit Board Distribution	9-3
9.6	System Cabinets	9-4
9.7	System Configuration Data	9-8
	9.7.1 Power Requirement	9-8
	9.7.2 System Cabinets Configuration	9-9

APPENDICES

Appendix

1	INSTRUCTION SUMMARY - ALPHABETICAL	A1-1
2	INSTRUCTION SUMMARY - NUMERICAL	A2-1
3	EXTENDED BRANCH MNEMONICS	A3-1
4	OP CODE MAP	A4-1
5	INSTRUCTION EXECUTION TIMES	A5-1
6	AUTOMATIC I/O OPERATION AND TIMING DATA	A6-1
7	I/O REFERENCES	A7-1
8	ARITHMETIC REFERENCES	A8-1
9	GLOSSARY OF TERMS	A9-1

ILLUSTRATIONS

Figure

1-1	System Block Diagram	1-2
2-1	Processor Block Diagram	2-2
2-2	Program Status Word Format	2-2
2-3	Program Status Bits	2-3
2-4	I/O Channel Operation Block Diagram	2-10
2-5	Automatic Interrupt Service	2-11
2-6	Channel Control Block Diagram	2-12
2-7	Bit Configuration For Channel Command Word	2-13
2-8	Channel Command For Initialize and Output Commands	2-13
2-9	Channel Command Words for I/O Operation	2-14
2-10	Channel Command Words for Termination	2-15

ILLUSTRATIONS

(Continued)

Figure		Page
2-11	Automatic I/O	2-16
2-12	First/Second Channel Command Block	2-18
3-1	Fixed-Point Word Formats	3-2
3-2	Floating-Point Word Format	3-3
3-3	Logical Data Word Formats	3-4
3-4	Instruction Word Formats	3-5
3-5	Data Word Formats	3-7
4-1	Circular List	4-57
5-1	Input/Output Rate Comparison	5-1
5-2	System Interface, Block Diagram	5-2
5-3	Multiplexor Channel, Block Diagram	5-3
5-4	Processor/Device Controller Logic Interface (Sheet 1 of 2)	5-6
5-4	Processor/Device Controller Logic Interface (Sheet 2 of 2)	5-7
5-5	Multiplexor Bus Buffer	5-8
5-6	Device Addressing, Logic Diagram	5-10
5-7	Data and Status Input Logic Diagram	5-11
5-8	Data and Command Output, Logic Diagram	5-13
5-9	Interrupt Control, Logic Diagram	5-14
5-10	Typical Universal Expansion Slot Wiring	5-16
5-11	Multiplexor Channel Timing	5-17
5-12	General Multiplexor Bus Interface	5-19
5-13	Interleaved Data Channel, Block Diagram	5-20
5-14	Data Channel Timing Chart	5-21
5-15	Data Channel/Multiplexor Bus Interface	5-22
5-16	Typical Memory Cycle	5-23
5-17	MOS Memory Cycle	5-23
5-18	Memory Bus Diagram	5-24
5-19	Example of Memory Bus Priorities	5-24
5-20	Model 74 and 70 Memory Bus Timing	5-25
5-21	Model 80 Memory Bus Timing	5-26
5-22	Typical Control Logic for Interfacing to the Memory Bus	5-27
5-23	Selector Channel, Block Diagram	5-30
5-24	Selector Channel, Flow Chart	5-32
5-25	General Purpose Wire Wrap Board	5-36
5-26	Wire Wrap Posts/IC and Component Leads Connection	5-36
5-27	IC Circuit Board Numbering System	5-37
5-28	Example of a Power Gate	5-37
5-29	General Purpose Interface Board Connector Layout	5-38
6-1	Control Console	6-2
6-2	Control Console Data Transfers	6-8
7-1	INTERDATA Software	7-1
7-2	BOSS Features	7-1
7-3	DOS Features	7-1
7-4	RTOS Features	7-1
7-5	Program Preparation Sequence	7-4
7-6	Conditional Assembly Structures	7-37
7-7	Sample Program Source Deck	7-38

ILLUSTRATIONS

(Continued)

Figure		Page
7-8	Sample Program One-Pass Object Tape	7-39
7-9	Sample Program One-Pass Assembly Listing	7-40
7-10	Assembler Print Formats	7-40
7-11	Basic Assembler Memory Map	7-43
7-12	PASS1 Operations	7-48
7-13	PASS2 Operations	7-49
7-14	PASS3 Operations	7-49
7-15	OS Assembler Memory Map	7-52
7-16	OS Assembler PASS2 Assembly with Scratch	7-59
7-17	I/O Error Message Description	7-59
7-18	Memory Map at Compile Time Using BOSS/4B	7-67
7-19	Execution Time Memory Map	7-70
7-20	Loader Memory Maps	7-76
7-21	Loader Tape Format	7-79
7-22	Object Tape Formats	7-83
7-23	Memory Map (as listed by the OS Library Loader)	7-85
7-24	Teletype Keyboard Layout	7-105
7-25	BOSS Core Map	7-118
7-26	RTOS Main Memory	7-127
7-27	Mass Storage Allocation	7-128
7-28	Interaction of RTOS Elements	7-129
7-29	Device Drivers and Memory Map	7-138
7-30	Typical RTOS Configurations	7-139
8-1	35 ASR Operating Modes	8-3
8-2	ASCII Character U (Even Parity), Eleven Bit Code	8-4
8-3	Teletype Keyboard Layout	8-5
8-4	Data Byte Format	8-16
8-5	Removable Cartridge Disc Format	8-19
8-6	201 Synchronous Data Set Adapter Block Diagram	8-22
8-7	PALS Block Diagram	8-24
8-8	Typical ASCII Character Format	8-26
8-9	Answering Calls HDX and FDX	8-30
8-10	Line Turn-Around Read/Write and Write/Read	8-31
8-11	Memory Protect Controller Functional Block Diagram	8-37
8-12	Universal Clock Module Block Diagram	8-39
8-13	Eight Line Interrupt Module Block Diagram	8-41
8-14	Memory Addressing	8-46
8-15	Memory Configuration, End on Byte Boundary	8-47
8-16	Starting and Final Address Data Bytes	8-47
8-17	Read Data Instructions	8-48
8-18	Sample Program	8-50
9-1	Typical 15-Inch Circuit Board, Component Side	9-1
9-2	Basic Processor Chassis	9-2
9-3	Basic Cabinet	9-4
9-4	Cabinet Accessories, Sheet 1 of 2	9-5
9-4	Cabinet Accessories, Sheet 2 of 2	9-6
9-5	Basic Cabinet Physical Dimensions	9-7
9-6	Typical Examples-Configuration Data Sheet A (Sheet 1 of 2)	9-10
9-6	Typical Examples-Configuration Data Sheet A (Sheet 2 of 2)	9-11

TABLES

Table		Page
1-1	INTERDATA Product Evolution	1-1
2-1	Program Status Bit Definitions	2-4
2-2	Core Memory Allocation	2-5
2-3	Interrupts	2-6
	. . .	
3-1	Hexadecimal Binary, and Decimal Cross-Reference	3-2
3-2	Examples of Fixed-Point Representation	3-3
3-3	Examples of Floating-Point Representation	3-4
3-4	Designations for First and Second Operands	3-6
3-5	Memory Addressing Example	3-8
5-1	Model 80 Access and Transfer Times	5-25
6-1	Core Memory Initialization	6-4
6-2	Auto Load Sequence	6-6
7-1	Logical Unit Numbers	7-3
7-2	Typical Source Program	7-10
7-3	Typical Symbol Table	7-10
7-4	Typical Assembly Listing	7-11
7-5	Statement Error Flags	7-12
7-6	Data Flags	7-12
7-7	Symbol Table Error Flags	7-13
7-8	Absolute and Relocatable Expression Rules	7-17
7-9	Instruction Format Summary	7-20
7-10	Summary of Assembler Instructions	7-22
7-11	Symbol Table Capacity	7-44
7-12	I/O Device Definition Table Selection for Basic Assembler	7-45
7-13	Source Program Format	7-46
7-14	Basic Assembler Operations	7-47
7-15	Symbol Table Capacity	7-53
7-16	Logical Unit Number Specifications	7-54
7-17	Logical Source Format	7-55
7-18	OS Assembler Operations	7-55
7-19	OS Assembler Operations with Scratch	7-56
7-20	Examples of I/O Error Messages	7-60
7-21	Intrinsic Functions	7-65
7-22	Basic External Functions	7-66
7-23	Model 70 Interactive FORTRAN Summary	7-71
7-24	FORTRAN Operation Statements	7-72
7-25	Summary of Loader Features	7-74
7-26	Sequence for Models 74, 70, and 80	7-77
7-27	Device Definition Table Entries	7-78
7-28	Loader Summary	7-80
7-29	Control Item Definitions	7-81
7-30	Tape Codes (M08/09 Format)	7-83
7-31	Error Messages	7-91
7-32	Line Addressing Summary	7-95
7-33	Line Arithmetic	7-96
7-34	Command Repertoire	7-97
7-35	Summary of TIDE Features	7-103
7-36	Summary of CLUB Directives	7-117

TABLES

(Continued)

Table		Page
7-37	RTOS Operator Commands	7-131
7-38	RTOS Supervisor Calls	7-134
7-39	RTOS Task Utility Task (TUT) Commands	7-135
7-40	RTOS Task Establisher (TET) Commands	7-136
8-1	TTY Interface Status and Command Byte Data	8-6
8-2	Sample Program Listing	8-8
8-3	Reader and Punch Characteristics	8-11
8-4	Reader/Punch Status and Command Byte Format (Hex Address 13)	8-12
8-5	Card Reader Status and Command Byte Data (Hex Address 04)	8-16
8-6	Sample Program for Card Reader	8-17
8-7	PALM Status and Command Byte Data	8-27
8-8	Interrupt Conditions	8-35
8-9	Selector Channel Status and Command Byte Data	8-45
9-1	Mounting Dimensions	9-12

CHAPTER 1 INTRODUCTION

1.1 THE INTERDATA FAMILY OF COMPUTERS

INTERDATA was incorporated in September of 1966 with the objective of becoming a leader in the fast growing and competitive minicomputer market. Today, INTERDATA is a profitable major factor in this competitive market segment.

Looking at the years since 1966, one can see a tremendous rate of increase in minicomputer performance and a decrease in the price of hardware. This is evident from Table 1-1 which compares the price/performance of past and present INTERDATA Processors. This is the result of an increasing market for inexpensive computers, developments in a very competitive semiconductor market, and the proliferation of minicomputer manufacturers in the late sixties.

TABLE 1-1. INTERDATA PRODUCT EVOLUTION

	1967 MODEL 3	1968 MODEL 4	1970 MODEL 5	MODEL 74	MODEL 70	MODEL 80
MEMORY CYCLE TIME	1.5 (CORE)	1.0 (CORE)	1.0 (CORE)	1.0 (CORE)	1.0 (CORE)	0.32*** (MOS)
MEMORY ACCESS TIME	0.6	0.45	0.45	0.3	0.3	280
MEMORY MODULE SIZE	4KB, 8KB	8KB	8KB	8KB	8KB	16KB
INSTRUCTION TIME: RR	28.0	3.2	3.2	1.5	1.0	0.53
SF	—	—	4.4	2.0	1.5	0.53
RS	36.0	4.0	4.0	2.5	2.25	0.53
RX	38.0	6.0	6.0	3.25	3.25	1.01
FIXED POINT MULTIPLY	140.0	23.0	23.0	40.0	6.0	2.25
FLOATING POINT MULTIPLY	4900.0 *	140.0	140.0	300.0 *	54.0	24.0
PROGRAMMED I/O LOOP	8KBPS	30KBPS	30KBPS	66KBPS	77KBPS	146 KBPS
SELECTOR CHANNEL	500KBPS	500KBPS	500KBPS	2000KBPS	2000KBPS	4200KBPS
FEATURES: SYSTEM 360/370 LIKE INSTRUCTION SET, DIRECT ADDRESSING TO 64 KB, 16 GENERAL REGISTERS, 16 OR 32 BIT INSTRUCTIONS.						
NO. OF STANDARD INSTRUCTIONS	74	77	113	100	113	113
MULTIPLY/DIVIDE	OPTIONAL	OPTIONAL	INCLUDED	INCLUDED	INCLUDED	INCLUDED
FLOATING POINT	—	OPTIONAL	INCLUDED	—	INCLUDED	INCLUDED
LIST PROCESSING	—	OPTIONAL	INCLUDED	—	INCLUDED	INCLUDED
AUTOLOAD BOOTSTRAP	—	INCLUDED	INCLUDED	INCLUDED	INCLUDED	INCLUDED
IMMEDIATE INTERRUPTS	—	—	INCLUDED	INCLUDED	INCLUDED	INCLUDED
AUTOMATIC I/O CHANNELS	—	—	INCLUDED	—	INCLUDED	INCLUDED
8 OR 16 BIT I/O BUS	—	—	OPTIONAL	INCLUDED	INCLUDED	INCLUDED
TELETYPE INTERFACE	OPTIONAL	OPTIONAL	OPTIONAL	—	INCLUDED	INCLUDED
COMPATIBILITY WITH PREVIOUS MODELS ?	—	YES	YES	YES	YES	YES
BASIC LIST PRICE: 8 KB PROCESSOR	\$12,600	\$ 17,400	\$ 11,200	\$ 3,600	\$ 6,800	\$ 14,900 **

* = SOFTWARE *** = WORST CASE AVERAGE WHEN ALTERNATING BETWEEN CPU AND DMA REQUESTED CYCLES.
** = WITH 16 KB

However, as minicomputer hardware prices have decreased, software manpower costs have risen sharply. As a result, system analysis, programming, and interfacing costs frequently overshadow hardware costs. Even in OEM situations involving numbers of identical systems, the simplicity and speed with which these development tasks can be performed is often a major factor in the successful introduction of a computer based product.

These are some of the considerations that resulted in the unique INTERDATA architecture among minicomputers. The INTERDATA architecture and instruction formats are very similar to the IBM system 360-370 series of computers. INTERDATA added to the basic IBM 360-370 like instruction set, several classes of instructions to increase the memory efficiency of the INTERDATA minicomputers. The advantages inherent in this kind of architecture are the large, easy to learn and apply instruction set; direct addressing to avoid the necessity for paging; 16 accumulators for the storage of partial results, 15 index registers for the storage of frequently used pointers and for loop management; plus other features that make an INTERDATA system easier and, therefore, less costly to use. In addition, INTERDATA Processors include such performance features as built-in multiply divide, external cycle stealing, direct memory access ports, and hardware interrupt discrimination, and vectoring for up to 256 devices.

These features are present in all INTERDATA New Series Processors, even the low priced Model 74 OEM unit. The Model 70 adds a higher speed Processor, built-in floating-point, automatic I/O channels, and more external direct memory access channels. The Model 80 includes all Model 70 features implemented in a very high performance Processor and memory configuration. All INTERDATA New Series Processors are fully program and interface upward-compatible so that a quantity user can meet a broad range of performance requirements in a given application area without having to redevelop software.

1.2 SYSTEM DESCRIPTION

Figure 1-1 shows the interrelationship of various elements in an INTERDATA New Series Model 74, 70 or 80 system. The following sections summarize the characteristics of each of the elements shown.

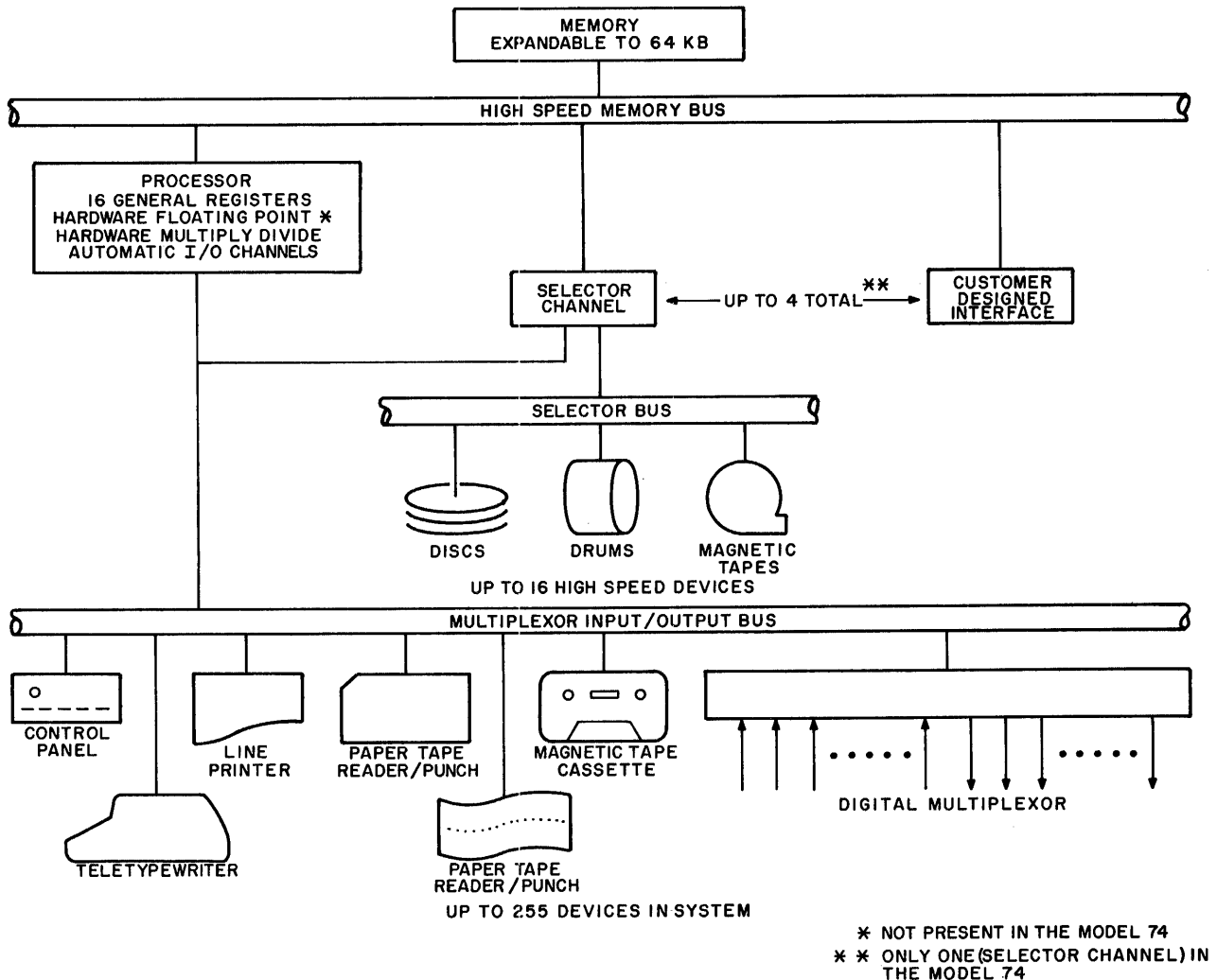


Figure 1-1. System Block Diagram

1.2.1 Memories

The instruction directly addresses 64K 8-bit bytes of memory and Model 74, 70, and 80 memories can be expanded to this capacity. Models 74 and 70 use core memories: 8K byte memory modules, 3 wire 3D design, with a 1.0 microsecond cycle time and a 300 nanosecond access time. The high speed Model 80 uses dual 8K byte Metal Oxide Semiconductor (MOS) LSI memories or 16K byte modules. The dual 8K byte MOS memories can overlap cycle times to a certain extent, resulting in an average cycle time of 320 nanoseconds without overlap and 290 nanoseconds with overlap between 8KB blocks.

Memory Parity and Memory Protect with Privileged instruction hardware are options. Both the core and MOS memory systems present the same high speed Memory Bus for direct memory access.

1.2.2 Direct Memory Access

Up to four Direct Memory Access devices may be added to the Model 70 (core) and Model 80 (MOS) memory systems. One direct Memory Access device can be added in the Model 74. Direct Memory Access devices allow the operation of high speed peripherals (e. g. disks or drums) directly to/from memory. This enables the user to perform simultaneous processing together with the high speed input/output transfer. This technique is memory cycle stealing. It is more efficient than Processor cycle stealing techniques that are presently used in some computers. To differentiate between the two, compare the maximum direct to memory transfer rates to the reciprocal of the memory cycle time. If they are not close, more than likely the less efficient Processor cycle stealing technique is used. In Models 74 and 70, the maximum transfer rate is 2,000,000 bytes per second. The Model 80 transfer rate is 4,750,000 bytes per second maximum in read burst and 6,000,000 bytes per second in write burst.

1.2.3 Selector Channel

The Selector Channel is a standard Direct Memory Access device that allows the connection of up to 9 high speed peripheral devices directly to memory. The same Selector Channel is used on Models 74, 70, or 80. The maximum transfer rates depend on the Processor used, and are the same as the rates in Paragraph 1.2.2. To use the Selector Channel, the program initializes the device itself, sends a starting and final address (each 16 bits) to the Selector Channel, and sends it a command to go. The Processor at this point can proceed to another function. When the Selector Channel terminates the transfer, it generates a hardware interrupt to the Processor. If customers have special high speed requirements, they can design a special interface to the Memory Bus. These special interfaces plug into the Memory Bus just like a Selector Channel.

1.2.4 The Processor

The Processor is the heart of the system. It controls all the activities and performs all the arithmetic and logical functions. It executes instructions in a specific sequence to do a specific job. The Processor is discussed in detail in Chapter 2, however, for the purposes of this section, it should be remembered that the Processor includes 16 hardware General Registers, fifteen of which can be used as index registers, and hardware multiply/divide in all New Series INTERDATA Processors. Models 70 and 80 also include hardware Floating Point instructions and a large number of Automatic Input/Output Channels (details in Chapter 2).

1.2.5 The Multiplexor Input/Output Bus

All medium to low speed devices connect to the Multiplexor Input/Output Bus. This is a request/response bus, consisting of 30 lines; 16 bidirectional data lines, 8 control lines, a sync (response) line, and a system initialize line (System Clear). There is a 15 microsecond automatic time out if a response (sync) is not received. Only one of the 8 control lines is active at any one time defining the contents and use of the 16 data lines. Interrupt detection and hardware vectoring for each of 255 devices is standard in all New Series INTERDATA Processors.

1.3 PERIPHERALS

A complete line of standard, off-the-shelf, peripheral devices are available with the INTERDATA Processors. All system modules and device controllers previously designed for the Models 3, 4, and 5 are plug compatible with the Multiplexor Bus. These field proven designs enable the user to select the devices or modules required for his specific application. The following are examples of what is available.

1.3.1 Digital Multiplexor

The Digital Multiplexor provides an economical set of modular blocks to monitor or control digital lines. A single controller, augmented with input and output modules of 128 lines each, provides the capability for monitoring 2048 inputs and controlling 2048 outputs. The Digital Multiplexor uses a biased core technique for input sampling. This technique insures absolute DC isolation from the sense contact, excellent common mode transient response and DC offset capability, which makes the Digital Multiplexor particularly well suited for reliable use in noise contaminated environments.

1.3.2 INTERTAPE Cassette System

The INTERTAPE Cassette System provides dual drive transports, capable of transferring data at a rate of 1000 characters per second. This reliable and inexpensive unit makes an ideal substitute for paper tape input/output equipment. With a storage capacity up to 500K bytes per cassette, or 1000K bytes total, the INTERTAPE Cassette System is ideal for low speed auxiliary storage.

1.3.3 IBM Compatible Magnetic Tapes

Both the seven and nine track IBM compatible tape transports are available. These units operate with a tape speed of 25 inches per second (ips), and are available with packing densities of 556 or 800 bits per inch (bpi).

1.3.4 Removable Cartridge Disc System

The Removable Cartridge Disc System is a reliable and inexpensive mass storage system, capable of providing 2.5 or 5.0 Megabytes of storage per unit. Up to four discs can operate on each controller, providing a maximum storage capacity of 10.0 Megabytes per system. Average access time is 70 milliseconds and the transfer rate is 180,000 bytes per second.

1.3.5 Magnetic Drum Storage System

The Magnetic Drum Storage System is capable of providing from 131K bytes to 1,048K bytes of storage depending on the drum system selected. Average access time is 8.7 milliseconds and the transfer rate is 230K bytes per second.

1.3.6 Data Communications Equipment

A complete line of character buffered adapters is available to service Bell 103, 201, 202, and 301 Data Sets, as well as the 801 Automatic Dialer. This enables the Processor to easily accommodate applications requiring either synchronous or asynchronous communications.

1.3.7 High Speed Paper Tape System

The High Speed Paper Tape System provides a 300 character per second Reader and 60 character per second Paper Tape Punch. These units can be provided individually, or as a combined package using the same controller.

1.3.8 System Modules

A complete line of system modules provides the user with a simple and convenient means of creating special interfaces. These general purpose interface modules greatly reduce or totally eliminate special design effort. Standard modules are available to handle eight-bit or sixteen-bit parallel input or output, manual data entry, and decimal indicators.

1.4 SOFTWARE

INTERDATA Processors are supported by a comprehensive array of software comparable only with software offered in support of much larger systems. Among these field proven software packages are the Assembler, Editor, Debug, Loaders, Test Programs and a FORTRAN IV Compiler. INTERDATA also provides a Basic Operating System (BOSS) and a Real-Time Operating System (RTOS). These fully supported software packages provide extensive capabilities in both business and scientific applications.

1.4.1 Assembler

The Assembler enables the user to code programs into symbolic language, which is translated by the Assembler into machine language. The Assembler accepts a source deck or tape consisting of user-coded instructions, and outputs a source listing and binary program object tape. The Assembler permits use of extensive pseudo instructions such as EXTRN and ENTRY, enabling the user to code large programs as multiple subroutines which can be linked together at load time. The Assembler also provides extensive diagnostic messages to indicate source program errors.

1.4.1.1 Stand Alone Assembler

The Stand Alone Assembler program tape is provided in relocatable tape format. The Stand Alone Assembler for the INTERDATA Processor requires a minimum of 8KB of core memory.

1.4.1.2 Operating System (OS) Assembler

The OS Assembler program is provided in relocatable paper tape format. The OS Assembler program executes under the supervision of an Operating System. Device independence is one of the significant advantages of the OS Assembler. All I/O is accomplished by Supervisor Call instructions. The OS Assembler also provides the capability to use magnetic tape as a scratch memory, eliminating the need for multiple passes of the source deck or paper tape.

1.4.2 Loader Descriptions

Four variations of compatible loaders are provided for loading binary object tapes generated by the Assemblers, FORTRAN IV Compiler, or Hexadecimal Debug Program. These loaders vary from the basic binary loader requiring minimum memory, to the sophisticated loaders capable of searching, locating, and linking subroutines by name.

1.4.2.1 OS Library Loader

The OS Library Loader is the most comprehensive loader. This loader requires 4,000 bytes of memory, and runs under the supervision of the Basic Operating System (BOSS). It may be operated interactively through the Console Teletype, or it may receive its operator commands from some other device such as the card reader, thus permitting Batch load-and-go processing. All I/O is accomplished through the logical I/O calls to BOSS.

1.4.2.1 General Loader

The General Loader is a stand-alone program, occupying approximately 1,500 bytes. The General Loader accepts paper tape inputs from a Teletype or High Speed Reader, and logs error and information messages on the Teletype. It provides program relocation, ENTRY and EXTRN handling, and handles forward references within programs. This program is operated from the Processor Control Console.

1.4.2.3 Relocating Loader

The Relocating Loader is a stand-alone program occupying approximately 800 bytes. The Relocating Loader accepts paper tape inputs from a Teletype or High Speed Reader, and displays error indications on the Processor Control Console. It allows program relocation, and handles forward references within programs. This program is operated from the Processor Control Console.

1.4.2.4 BOSS Resident Loader

The BOSS Resident Loader provides the same features as the Relocating Loader, but is operated through operator commands entered on the Console Teletype. The BOSS Resident Loader is device independent and allows input from any binary device.

1.4.3 Text Editor (TIDE)

TIDE is an on-line, interactive text editing program. The Text Editor permits the operator to create and modify character-oriented text input from paper tape or the Teletype keyboard. The text may be assembly language source statements, FORTRAN source statements, or any text in the literal sense. The features of TIDE include adding text, modifying text, deleting text, and the copying of paper tape records. The comprehensive set of keyboard commands greatly simplifies error correction, recoding, and character manipulation.

1.4.4 Debug (CLUB)

The Hexadecimal Debug Program (CLUB) provides maximum assistance in debugging user programs while using minimum core memory. This interactive program permits the operator to direct the debugging operation by entering directives and associated data via the Teletype keyboard. Responses to these inputs are shown on the Teletype page printer. The features of CLUB include the ability to monitor, modify, search, print, and punch object tapes. The powerful breakpoint feature enables the operator to trace the logic of his program by passing control between CLUB and the program being debugged.

1.4.5 FORTRAN (Models 70 and 80 Only)

The software includes both an Interactive FORTRAN System and an ANSI Standard FORTRAN IV Compiler. The Interactive FORTRAN provides a Direct Mode for on-line evaluation of arithmetic expressions, and an Editing Mode for the creation and manipulation of stored programs. The system combines the convenience of a desk calculator with the programming power of FORTRAN.

The FORTRAN IV Compiler facilitates simple algebraic solutions to mathematical or scientific problems. Programs are written as a sequence of statements using familiar arithmetic operations and English expressions. From these source statements, the compiler produces machine language programs that can be executed by the Processor. The FORTRAN IV Compiler is supported on Models 70 and 80.

1.4.6 Basic Operating System (BOSS)

The Basic Operating System provides a method of program management that, by establishing conventional responses to external and internal stimuli, creates an environment in which a single-user program can function almost as a pure problem solver. BOSS allows programs to be device independent, which enables the user to program more efficiently by concentrating more on the problem at hand and less on the tasks of input/output processing and interrupt handling.

1.4.7 Real Time Operating System (RTOS) (Models 70 and 80 Only)

The Real Time Operating System maximizes machine utilization by providing a multi-programming capability for interleaving the execution of programs and overlapping I/O operations. Automatic scheduling of its facilities using a priority system enables RTOS to schedule its own resources without constant operator intervention. User flexibility is enhanced by device independent programming through logical device assignments and automatic acknowledgement of interrupts.

1.4.8 Test Programs

The Processor and Memory Tests are comprehensive routines which validate the execution of all instructions and core memory locations through extensive use of interactive loops and worst case patterns. Error messages which indicate the precise test that failed result from any hardware discrepancies. These programs repeat execution until halted or until an error condition is detected. Test programs are also provided for all standard peripheral devices to validate both the controller and the device operational status.

1.5 CUSTOMER SUPPORT

At INTERDATA, customer support is paramount. The comprehensive line of customer support activities maintained by INTERDATA insures total customer back-up, prior to and after delivery of an INTERDATA Digital System.

1.5.1 Computation Center Facilities

INTERDATA maintains computation centers in both New Jersey and California. The facilities at these centers are available to customers at a minimal cost, enabling users to develop and debug application software prior to delivery of their INTERDATA system. For details, contact your local INTERDATA Sales Office.

1.5.2 Field Service

INTERDATA maintains Field Service Engineers throughout the country, with headquarters in each of the regional sales offices. All INTERDATA field service personnel are factory trained and have extensive experience with INTERDATA installations. At the INTERDATA factory, back-up service is available and a repair depot is maintained for all logic boards and memory packages. In the field, maintenance personnel are equipped with complete sets of spares and specialized diagnostic equipment. Working behind all this is the factory Quality Control Team maintaining exacting standards for production and final check-out. A customer's system is subjected to hours of running comprehensive tests and then, finally, "baked" in a heat chamber for a number of additional hours before the system is certified and shipped.

1.5.3 Training Center

INTERDATA maintains a complete, professionally staffed, training center. INTERDATA provides comprehensive courses in both programming and maintenance at a level that customers find interesting, informative, and challenging. The emphasis is placed on "hands on experience". All of the instructors on the training staff have extensive prior experience training computer personnel in military and civilian schools. Hundreds of programmers and customer engineers have already been successfully trained by this staff.

1.5.4 Systems Engineering and Programming

In many instances where the system requirement of a specific application goes beyond the capability of standard hardware and software, customers have looked to INTERDATA for special assistance. To accommodate these requests, an experienced team is available to furnish the exact support required. Members of this application team are selected from communications experts, senior programmers, analog specialists, and process control specialists. They make up an elite group, chosen for their capability and experience in solving specific problems. When a special interface is required, the solution can often be built quickly from off-the-shelf modules. Whenever a solution is required, the application team approaches the problem from the customer's viewpoint.

1.5.5 INTERCHANGE

INTERCHANGE, the INTERDATA user's group, is an active and growing association. All programs written by INTERDATA users and submitted to the INTERCHANGE Library are available to other INTERDATA users. Through this organization, customers gain a valuable "second level" of support by sharing ideas, programs and special interfaces.

CHAPTER 2

PROCESSOR DESCRIPTION

2.1 MICRO-PROGRAMMED INTERDATA PROCESSOR ARCHITECTURE

INTERDATA introduced the first micro-programmed multi-accumulator minicomputer, the Model 3, in 1967. The typical minicomputer of that day had one or two accumulators, one index register, and four to six instruction formats that were physically different from one another. The functions available in these computers were also very limited. For example, in absence of "subtract", "logical OR", or "logical Exclusive OR" instructions, three to seven instructions had to be programmed to accomplish the desired function. These characteristics limited the amount of hardware needed for the minicomputer and made them cheaper for the minicomputer manufacturers. The same characteristics made this kind of minicomputer more difficult to program and, therefore, more costly to the user from a program development point of view. These computers implemented their instructions directly in hardware and had a typical add time of 10 to 30 microseconds.

In 1967, INTERDATA led the industry by introducing a powerful 360-like architecture, at minicomputer prices, in the form of the Model 3. The 360 type Model 3 instruction set was easier to learn and use, simplifying programming, and minimizing development costs. To implement a powerful architecture of this type in hardware would have been prohibitively expensive. Thus, INTERDATA chose to design a very fast Processor, with 16 basic instructions (micro-instructions) and a large number of accumulators, called a Micro-Processor. The micro-instructions were then micro-programmed to fetch, interpret, and execute the user level instruction set. This is how, even today, the micro-instructions of Models 74, 70, and 80 emulate the user level architecture and instruction set.

In 1967, the emulated approach to the INTERDATA architecture sacrificed some speed, but recent advances in ROM technology have eliminated this speed penalty. As one looks at the New Series INTERDATA family of computers, there can be no doubt that it is second to none. It has the easiest to use architecture, and it is more than competitive in both price and performance.

Technology changes from one year to the next. It is possible to arrive at a new optimized minicomputer architecture for each level of technology, but that would very significantly impact software compatibility. Micro-programmed emulation is the answer from INTERDATA. While others had to introduce new, incompatible, architectures as a result of technological changes in the past five years, INTERDATA used micro-programming to adapt the architecture to technology, thereby protecting everyone's investment in INTERDATA software. INTERDATA Micro-Processors are all different, but the user level machine characteristics have remained upward compatible from the Model 3 of 1967 to the Model 80 of 1972.

2.2 PROCESSOR BLOCK DIAGRAM

The Processor block diagram shown in Figure 2-1 illustrates the information flow in the Models 74, 70, and 80.

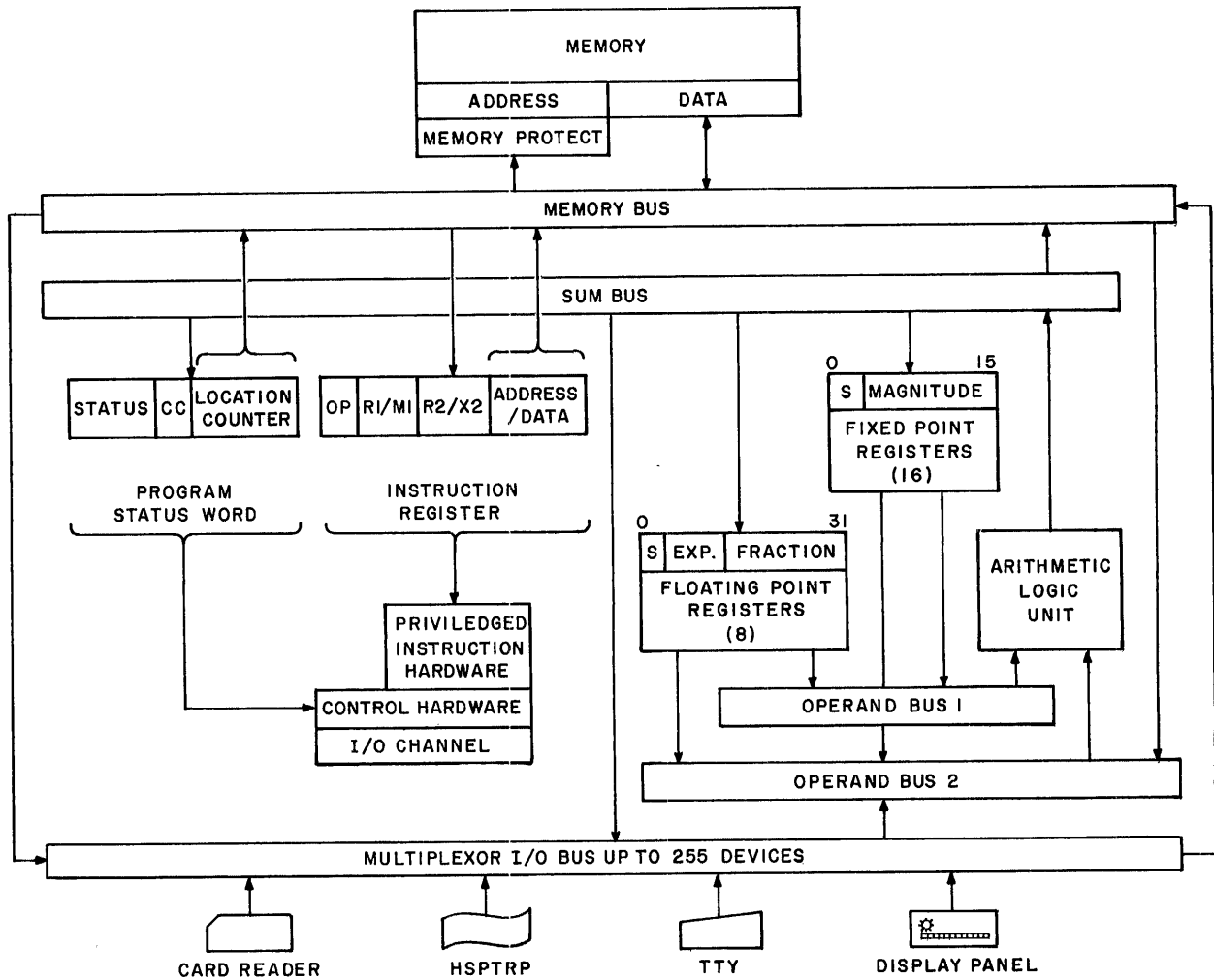


Figure 2-1. Processor Block Diagram

The 32-bit Program Status Word (PSW) defines the state of the Processor at any one time. The individual bits in the 12-bit status portion of the PSW enable or disable different classes of interrupts within the system. The four-bit Condition Code (CC) field reflects the result of an operation (e.g. zero, negative, etc.). The 16-bit Location Counter sequences the instructions in a program. See Figure 2-2.

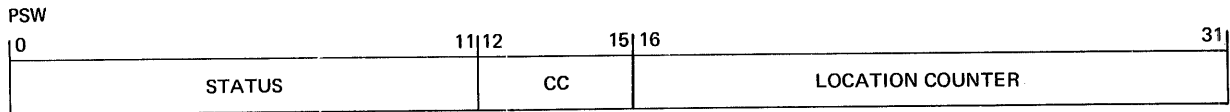


Figure 2-2. Program Status Word Format

The 32-bit Instruction Register holds the instruction for decoding by the hardware and micro-program. The various fields of the instruction word will be explained later. When the Operation Code (OP Code) of an instruction is decoded, the first operand is fetched from the registers and is gated to Operand Bus 1. The second operand is gated to Operand Bus 2 from memory or from a register. The Arithmetic Logic Unit (ALU) performs the operation and gates the result to the register, replacing the first operand, and sets the proper Condition Code (CC) in the Program Status Word (PSW). Input/output operations can also take place directly between the Memory Bus and the Processor's Multiplexor I/O Bus.

2.3 PROCESSOR OPERATION

2.3.1 Program Status Words

The left half of the PSW defines Program Status, the right half is the Location Counter. The Current PSW controls instruction sequencing and maintains the status of the system in relation to the program currently being executed. A program can change the Processor status by loading a New PSW. This is accomplished by executing a Load Program Status Word (LPSW) or Exchange Program Status (EPSR) instruction. These instructions are described in Chapter 4.

The interrupt mechanism of the Processor also involves the PSW. When an interrupt takes place, the Current PSW is stored at a unique four-byte location called the Old PSW. After the Current PSW is stored, the Current PSW Register is loaded from another four-byte location called the New PSW. Each and every interrupt class has a unique set of Old and New PSWs. The PSW swap takes place automatically and, after the PSW swap, the program execution begins at the location specified by the Location Counter of the New PSW. The reserved core locations for Old and New PSWs for all interrupts are defined in Section 2.3.3.

The meaning of each bit in the left half of the PSW, Status, and Condition Code is explained in Table 2-1 and shown in Figure 2-3. The particular meaning or function of each bit applies when the bit is a logical ONE.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
WT	EI	MM	DF	AS	FP	CT	PM	0	0	0	0	C	V	G	L

Figure 2-3. Program Status Bits

2.3.2 Instruction Execution

The 16-bit Location Counter field of the Program Status Word specifies the location of the next instruction to be fetched and processed. The 16-bit address field has the capacity of directly addressing the maximum core memory of 64K bytes, or 32K halfwords.

Note that since instructions are aligned on halfword boundaries, the value of the Location Counter must be even. That is, Bit 15 of the Location Counter must be zero.

During the normal processing of a program, an instruction is fetched from the location specified by the Location Counter, the instruction is executed, the Location Counter is incremented, and another fetch and execute cycle begins. After instruction execution, (except for Branch or Control instructions) the Location Counter is incremented by two if the executed instruction is of the halfword 16-bit format, or by four if the executed instruction is of the fullword 32-bit format.

Following Branch instructions or System Control instructions, the Location Counter is adjusted as a function of the particular instruction.

The sequencing of instructions during program execution is also changed if an interrupt occurs. In this case, the PSW swap procedure saves the Current PSW in main memory so that, after an interrupt is processed, execution can resume at the correct location.

TABLE 2-1. PROGRAM STATUS BIT DEFINITIONS

Bit	Name	Comments
0	WT Wait State	The Wait bit is set to halt program execution. When this bit is set in the Current PSW, no program execution takes place, but the Processor will respond to all I/O and Machine Malfunction Interrupts, if they are enabled.
1	EI External Interrupt Enable	The External Interrupt Enable bit is set to make the Processor responsive to interrupt signals from the Multiplexor Bus.
2	MM Machine Malfunction Interrupt Enable	The Machine Malfunction Enable bit allows an interrupt to occur if a power fail is detected, if the machine is equipped with the Memory Parity Option and a memory parity error occurs, or during the restart process following a power down.
3	DF Fixed Point Divide Fault Interrupt Enable	The Divide Fault Interrupt Enable bit allows the Processor to interrupt when a Fixed-Point Divide instruction is attempted and the result cannot be expressed in 16-bits.
4	AS Automatic Input/Output Service Enable	The Automatic I/O Service Enable bit allows the Processor to acknowledge I/O Interrupts and service them automatically.
5	FP Floating-Point Arithmetic Fault Interrupt Enable	The Floating-Point Arithmetic Fault Interrupt Enable bit allows the Processor to interrupt if exponent overflow or underflow occurs during any floating-point operation.
6	CT Channel Termination Interrupt Enable	Channel Termination Interrupt Enable bit pertains to the Automatic I/O Channel, which can be used in conjunction with the Automatic I/O Service.
7	PM Protect Mode	The Protect Mode bit enables Memory Protect and detection of Privileged instructions. When the Protect Mode is not enabled the Processor is said to be in the Supervisor Mode.
8-11	Unused	Must be zero.
12	C Carry/Borrow	The Condition Code bits are set or adjusted after the execution of instructions by the Processor.
13	V Overflow	
14	G Greater than Zero	
15	L Less than Zero	

2.3.3 Main Memory Allocation

The Processor requires certain locations in main memory for Floating-Point Registers, register save areas, and interrupting processing. These locations are defined in Table 2-2 and described in the following paragraphs.

TABLE 2-2
CORE MEMORY ALLOCATION

Function	Hexadecimal Memory Address	Assignment
Floating-Point Registers (Models 70, 80 only)	00-03 04-07 08-0B 0C-0F 10-13 14-17 18-1B 1C-1F	Floating-Point Register, R0 Floating-Point Register, R2 Floating-Point Register, R4 Floating-Point Register, R6 Floating-Point Register, R8 Floating-Point Register, R10 Floating-Point Register, R12 Floating-Point Register, R14
Power-Fail Locations	20-21 22-23 24-27	Unassigned Register Save Pointer Current PSW Save Area
Interrupt PSWs	28-2B 2C-2F 30-33 34-37 38-3B 3C-3F 40-43 44-47 48-4B 4C-4F	Old PSW FLPT Arithmetic Fault Interrupt New PSW FLPT Arithmetic Fault Interrupt Old PSW Illegal Instruction Interrupt New PSW Illegal Instruction Interrupt Old PSW Machine Malfunction Interrupt New PSW Machine Malfunction Interrupt Old PSW External Interrupt New PSW External Interrupt Old PSW Fixed-Point Divide Fault Interrupt New PSW Fixed-Point Divide Fault Interrupt
Reserved	50-7F	Bootstrap Loader and Device Definition Table
Channel I/O Termination Parameters (Models 70, 80 only)	80-81 82-85 86-89 8A-8B 8C-8F 90-93	Termination Queue Pointer Old PSW Channel I/O Termination Interrupt New PSW Channel I/O Termination Interrupt Overflow Termination Pointer Old PSW Termination Queue Overflow Interrupt New PSW Termination Queue Overflow Interrupt
Supervisor Call Parameters	94-95 96-99 9A-9B 9C-9D 9E-9F A0-A1 A2-A3 A4-A5 A6-A7 A8-A9 AA-AB AC-AD AE-AF B0-B1 B2-B3 B4-B5 B6-B7 B8-B9 BA-BB BC-CF	Supervisor Call Argument Pointer Old PSW Supervisor Call New PSW (Status and Condition Code) Supervisor Call New PSW (Location Counter) Supervisor Call 0 New PSW (Location Counter) Supervisor Call 1 New PSW (Location Counter) Supervisor Call 2 New PSW (Location Counter) Supervisor Call 3 New PSW (Location Counter) Supervisor Call 4 New PSW (Location Counter) Supervisor Call 5 New PSW (Location Counter) Supervisor Call 6 New PSW (Location Counter) Supervisor Call 7 New PSW (Location Counter) Supervisor Call 8 New PSW (Location Counter) Supervisor Call 9 New PSW (Location Counter) Supervisor Call 10 New PSW (Location Counter) Supervisor Call 11 New PSW (Location Counter) Supervisor Call 12 New PSW (Location Counter) Supervisor Call 13 New PSW (Location Counter) Supervisor Call 14 New PSW (Location Counter) Supervisor Call 15 Reserved
Interrupt Service Table	D0-D1 D2-D3 D4-D5 • • • • • 2CC-2CD 2CE-2CF	Service Pointer, Device 0 Service Pointer, Device 1 Service Pointer, Device 2 Service Pointer, Device 254 Service Pointer, Device 255

- Floating-Point Registers - These eight 32-bit registers are used by the Floating-Point instructions in Models 70 and 80.
- Power Fail Locations - The Register Save Pointer at Location X'22', points to the first of 16 consecutive halfword locations in memory where the General Registers are saved in the event of power failure. When power is restored, the General Registers are restored automatically from these locations. The Current PSW is saved and restored in similar fashion from Location X'24' - X'27'.
- Interrupt PSW's - These locations are reserved for the Old and New PSWs for the various internal and external interrupts.
- Bootstrap Loader - The Bootstrap Loader is called the 50 Sequence and is used to load the more sophisticated loaders.
- Channel I/O Termination Parameters - These locations are used in conjunction with Termination interrupts from automatic I/O channel operation in Models 70 and 80.
- Supervisor Call Parameters - These locations are used for the PSW exchange associated with the Supervisor Call (SVC) instruction.
- Interrupt Service Table - The Processor uses this table to uniquely service each interrupting device.

2.4 INTERRUPT SYSTEM

2.4.1 Interrupt Procedure

The interrupt structure of the Processor provides rapid response to internal and external events that require service by special software routines. In the interrupt response procedure, the Processor preserves the current state of the machine, and branches to the required service routine. The service routine may optionally restore the previous machine state upon completion of its service. The types of interrupts, with their associated Enable/Disable PSW bits, are listed in Table 2-3. Interrupts without a controlling PSW bit are always enabled.

TABLE 2-3. INTERRUPTS

<u>Interrupt</u>	<u>PSW Control Bit</u>
External	1
Machine Malfunction	2
Fixed Point Divide Fault	3
Automatic I/O Service	4
Floating-Point Arithmetic Fault	5
Channel Termination	6
Protect Mode	7
Illegal Instruction	Cannot be disabled
Channel Termination Queue Overflow	Cannot be disabled
Supervisor Call	Cannot be disabled

Interrupts can occur at various times during processing. The Arithmetic Fault Interrupts occur during execution of user instructions. The Illegal instruction and Protect Mode Interrupt occur as soon as the offending instruction is recognized. The Supervisor Call Interrupt occurs as part of the execution of the Supervisor Call instruction. The Machine Malfunction and I/O Service Interrupts occur following instruction execution. The Channel Termination Interrupt can also occur during a Load Program Status Word or Exchange Program Status instruction.

The Interrupt Procedure is based on the concepts of Old, Current, and New Program Status Words. The Current PSW, contained in a hardware register, defines the operating status of the machine. When this status must be interrupted, the Current PSW becomes an Old PSW and is stored in a memory location dedicated to the type of interrupt that has occurred. The New PSW becomes the Current PSW by being loaded from a dedicated location into the hardware PSW Register. The status portion of the Current PSW now contains the operating status for the interrupt service routine. New Program Status Words for interrupt controlled by PSW bits should disable interrupts of their own class. Interrupts controlled by Bits 1 and 6 must disable interrupts of their own class to prevent the Processor from going into an endless loop. The dedicated core locations for Old and New Program Status Words are shown in Table 2-2. The Program Status Word exchange procedure does not change the contents of the New PSW location, and subsequent interrupts of the same type are treated in the same way.

2.4.2 Internal Interrupts

The Processor can generate six Internal Interrupts. Of these, the Illegal Instruction and the Supervisor Call cannot be inhibited. Inhibited Internal Interrupts are not queued.

2.4.2.1 Floating-Point Divide Fault Interrupt

The Fixed-Point Divide Fault Interrupt, enabled by Bit 3 of the Program Status Word, is indicative of division by zero or quotient overflow. Quotient overflow is defined as quotient magnitude greater than $2^{15}-1$. The interrupt takes place before modification of the operand registers. After a Fixed-Point Divide Fault Interrupt, the Old PSW Location Counter points to the next instruction following the Divide instruction.

2.4.2.2 Floating-Point Arithmetic Fault Interrupt

The Floating-Point Arithmetic Fault Interrupt enabled by Bit 5 of the Current PSW, occurs on exponent overflow or underflow as well as on division by zero. In the case of division by zero, the interrupt takes place prior to alteration of the operand register. An exponent overflow sets the results to $\pm X'7FFF\ FFFF'$. An exponent underflow sets the results to $X'0000\ 0000'$. The Location Counter of the Old PSW points to the next instruction.

2.4.2.3 Machine Malfunction Interrupt

Bit 2 of the Current Program Status Word controls the Machine Malfunction Interrupt. This error can occur on either a primary power fail, a memory parity error, or during the restart process following a power down. If the memory is equipped with the parity option, the parity bit of each memory word is set to maintain odd parity. This bit is recomputed during each memory read; if the computed bit is not equal to the bit read out of memory and if Bit 2 of the current PSW is set, the Current Program Status Word is stored at the Machine Malfunction Old PSW location, and the Current PSW is loaded from the Machine Malfunction New PSW location, and the Current PSW is loaded from the Machine Malfunction New PSW location. The Condition Code field of the Current PSW is then adjusted by setting the G flag (PSW 14) if the parity error occurred on instruction read, or setting the V flag (PSW 13) if the error occurred on an operand read. It is not possible to guarantee programmed recovery from a parity error.

NOTES

1. The Condition Code field of the Machine Malfunction New PSW location in memory must be zero
2. The Model 74 does not set either V or G on parity error.

If enabled by Bit 2 of the PSW, a Machine Malfunction Interrupt occurs on power fail. Power fail occurs when the optional Primary Power fail detector senses a low voltage, when the Initialize switch is depressed, or when the key-operated power switch is turned off. After the PSW swap, the L flag of the Current PSW is set. The software service subroutine for Machine Malfunction can distinguish power fail from parity errors by Conditional Branch instructions. The user is allowed approximately 1 millisecond before the system is shut down. On shut down, the Processor stores the Current Program Status Word in Locations X'0024' through X'0027', and the General Registers in the consecutive locations starting at the address contained in Location X'0022'. When power is restored, the registers are reloaded and the Current PSW is restored from Locations X'0024' through X'0027'. If Bit 2 (Machine Malfunction) of this PSW is set, the Processor makes a PSW exchange from the Machine Malfunction location. The software service routine for the Machine Malfunction Interrupt can differentiate between the memory parity error, power failure, and power up conditions by testing the Condition Code. Note that the V flag is set for parity error on operand fetch, the G flag is set for parity error on instruction fetch, the L flag is set on power fail, and no flags are set on power restore. Additionally, the Location Counter of the Machine Malfunction Old PSW, may be compared with the contents of Locations X'0026' and X'0027' (Power Fail PSW save area). If they are equal, a power failure and restore sequence has occurred.

2.4.2.4 Illegal Instruction Interrupt

The Illegal Instruction Interrupt is not represented by an enabling bit in the PSW and is, therefore, always operative. An Illegal instruction is defined as an Operation Code that is not in the instruction repertoire of the Processor. No attempt is made to execute the Illegal instruction, nor is the Location Counter of the Current PSW incremented. The Old PSW stored as a result of an Illegal Instruction Interrupt, points to the address of the Illegal instruction.

2.4.2.5 Protect Mode Violation Interrupt

The Protect Mode Violation Interrupt is enabled when Bit 7 of the Current PSW is set, which puts the Processor in the Protect Mode. The interrupt occurs, in this mode, when an attempt is made to execute a Privileged instruction. Privileged instructions are all I/O instructions, and System Control instructions: Load Program Status Word, Exchange Program Status, and Simulate Interrupt. When such an instruction is attempted in this mode, the instruction is not executed, and the Illegal Instruction Interrupt procedure takes place, as describe above. The Location Counter is not incremented, so that the Old PSW points to the Privileged instruction that caused the interrupt. PSW Bit 7, when set, also enables Memory Protect.

2.4.2.6 Supervisor Call (SVC) Interrupt

This interrupt occurs as the result of an SVC instruction, which is used to communicate between running programs and operating systems. The Supervisor Call Interrupt is not inhibitable. When an SVC instruction is executed, the following action takes place:

1. The Current PSW is stored at the Supervisor Call Old PSW location, Location X'0096'.
2. The effective address from the SVC instruction is stored at the Supervisor Call argument pointer, Location X'0094'.
3. The status portion of the Current PSW is loaded from the Supervisor Call New PSW Status location, Location X'009A'.
4. The Current Location Counter is loaded from one of the Supervisor Call New PSW Location Counter locations.

2.4.3 Input/Output Control Interrupts

If individually enabled by the program, a peripheral device is allowed to request Processor service when the device itself is ready to transfer data via an interrupt. The Processor may respond to this signal in several ways depending on the setting of certain bits in the Program Status Word. The Processor has two classes of interrupts directly related to peripheral device handling. These are External Interrupt and Immediate Interrupt. Two other classes, the Channel Termination Interrupt and the Channel Queue Overflow Interrupt can occur upon termination of an Automatic I/O channel sequence. PSW Bits 1 and 4, in combination, control the External and Immediate Interrupts.

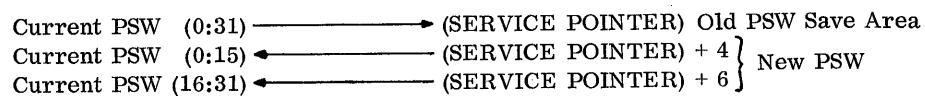
If Bit 1 is reset, I/O Device Interrupt signals are ignored. The signal remains pending, however, until PSW Bit 1 is set and the signal is acknowledged. Bit 6 of PSW controls the Channel Termination Interrupt. The Channel Termination Queue Overflow Interrupt is always enabled.

2.4.3.1 External Interrupt

If Bit 1 of the Current PSW is set, and Bit 4 is reset, an I/O Device Interrupt signal results in the following action: The Current PSW is stored at the Input/Output Interrupt Old PSW location. The Current PSW is loaded from the Input/Output New PSW location. From this point, software must acknowledge the interrupt, identify the device, and take appropriate action. Note that the New PSW for External Interrupts must have Bit 1 reset. This interrupt handling technique has been included in the INTERDATA New Series Processor primarily to maintain compatibility with the older Models 3, 4, and 5.

2.4.3.2 Immediate Interrupt Service

If both Bit 1 and Bit 4 of the Current PSW are set, an interrupt signal from a peripheral device results in the following Automatic I/O Service. The signal is automatically acknowledged and the device number returned is used to index into the Interrupt Service Pointer Table in Locations X'00D0' to X'02CF'. See Table 2-2. The Service Pointer obtained is the address of either an Old PSW save area, or a Channel Command Word for a channel I/O operation. If Bit 15 of the Service Pointer is reset, the Current PSW is stored in the fullword location whose address is contained in the Service Pointer Table. The halfword whose address is the contents of the Service Pointer Table plus four contains the New PSW Status and Condition Code fields. The Location Counter is set to a value equal to the contents of the Service Pointer Table plus six and instruction execution resumes.



Through this Immediate Interrupt mechanism, a unique service routine for any device number can be automatically entered. Exit from the routine is made by executing a Load Program Status Word instruction specifying the Old PSW location (Service Pointer) at the origin of the subroutine. If Bit 15 of the Service Pointer is set in Models 70 and 80, the address contained is that of a Channel Command Word implying that automatic I/O Channel service is required. This Processor activity is described in Chapter 4.

2.4.3.3 Automatic I/O Channel Termination Interrupt

The termination of an Automatic I/O Channel operation may result in the storing of a termination pointer in the circular list located at the address specified by the Queue Pointer location. If, at this time, Bit 6 of the Current PSW is set, the Current PSW is stored at the Channel Termination Old PSW location, and the Current PSW is loaded from the Channel Termination New PSW location. In this way, the control software is notified of the completion of a channel I/O operation. Whenever the Processor executes a Load Program Status Word instruction or an Exchange Program Status instruction, it checks Bit 6 of the newly loaded PSW. If Bit 6 of the loaded PSW is set, and there is an entry in the queue, this interrupt is taken. This is described in detail in Section 2.4.4.

2.4.3.4 Channel Termination Queue Overflow Interrupt

If the Processor attempts to enter a Channel I/O Termination Pointer in the Termination Queue and the queue is already full, it stores the termination pointer at Location X'008A', the Overflow Termination Pointer location; stores the Current PSW in Location X'008C', the Queue Overflow Old PSW location; and loads the Current PSW from Location X'90', the Queue Overflow New PSW location. This action allows the software to clear out the queue before any channel I/O terminations are lost. This interrupt cannot be disabled.

2.4.4 The Automatic Input/Output Channel (Models 70 and 80 Only)

The Automatic Input/Output Channel executes channel programs that control the activities of peripheral devices. The execution of channel programs takes place between the execution of user instructions, and results in a program delay rather than a program interrupt with an exchange of Program Status Words. The I/O Channel may generate an interrupt because of abnormal conditions or because of the occurrence of an event for which the software had requested an interrupt. Bits 1 and 4 of the Current Program Status Word control the operation of the I/O Channel. Both of these bits must be set to permit channel operations. Channel operations also depend on the Automatic I/O Service Table, the Channel Control Block with its associated Channel Command Word, and the Channel Termination Queue, see Figure 2-4. All features of the Automatic I/O Channel may not be compatible with future INTERDATA machines.

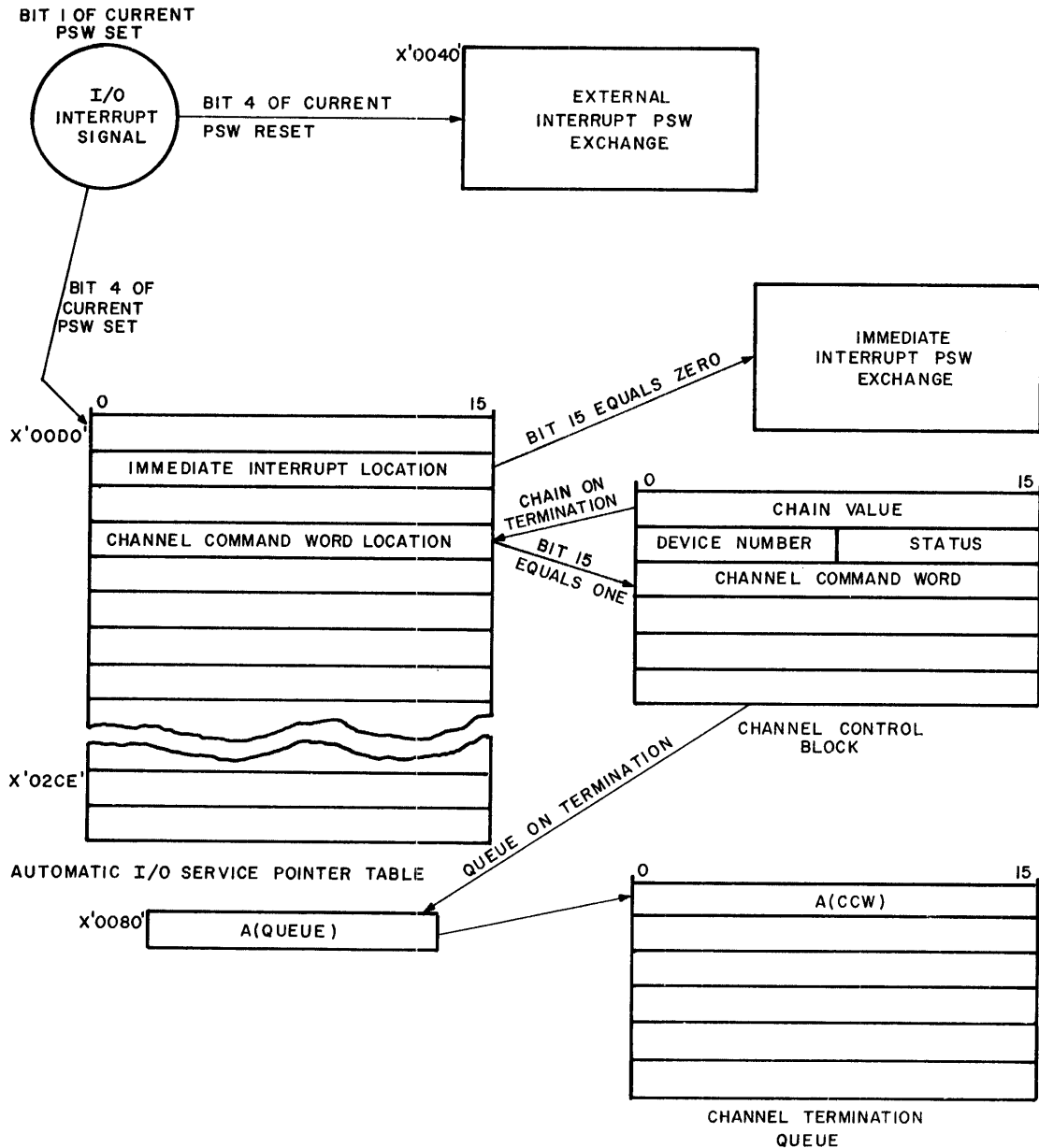


Figure 2-4. I/O Channel Operation Block Diagram

2.4.4.1 Interrupt Pointer Table

The Interrupt Pointer Table starts at location X'00D0'. It contains a halfword entry for each of the 256 possible peripheral device addresses. If Bit 15 of the entry in this table is reset, then the entry is the address of an Immediate Interrupt PSW exchange location. If Bit 15 of the entry is set, then the entry minus one is the address of a Channel Command Word. See Figure 2-5.

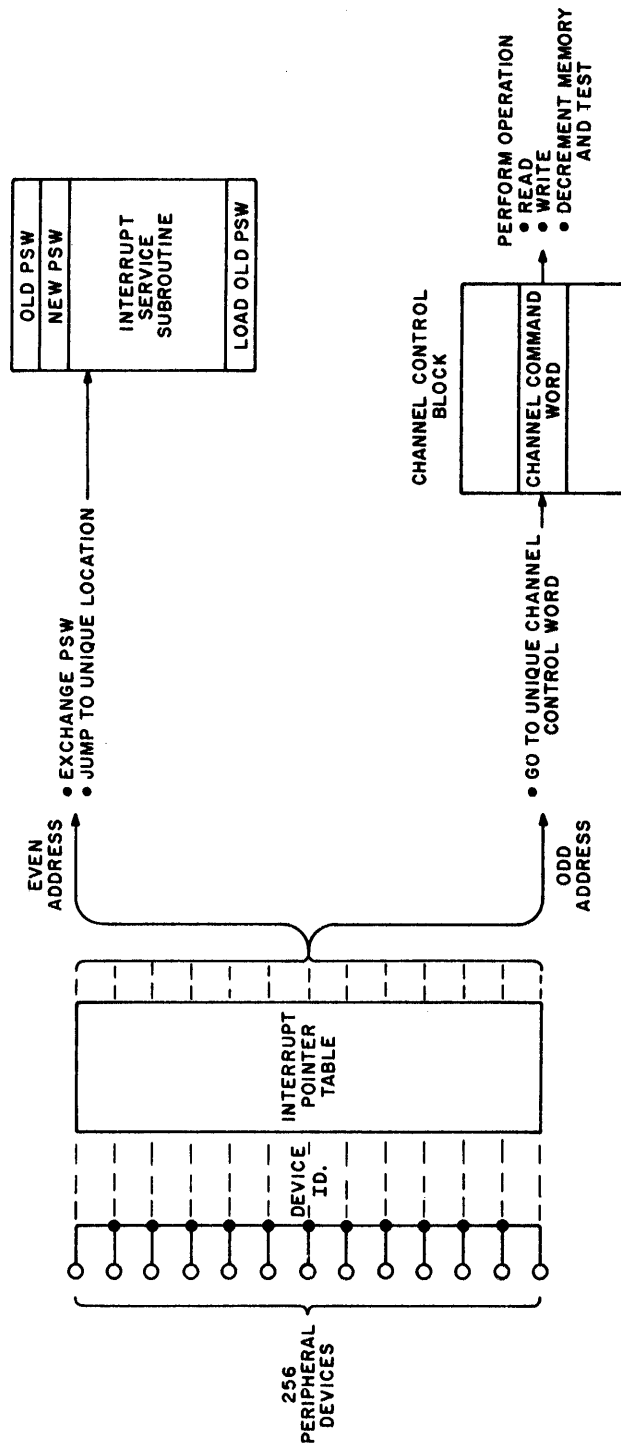


Figure 2-5. Automatic Interrupt Service

2.4.4.2 Channel Control Block

The Channel Control Block contains the Channel Command Word plus the storage locations and data required by the channel operation. The Channel Command Word is a bit encoded command that completely describes the Automatic Channel Operation. Note that it is the address of the Channel Command Word plus one that is entered in the Automatic I/O Service Table. A complete Channel Control Block is shown in Figure 2-6.

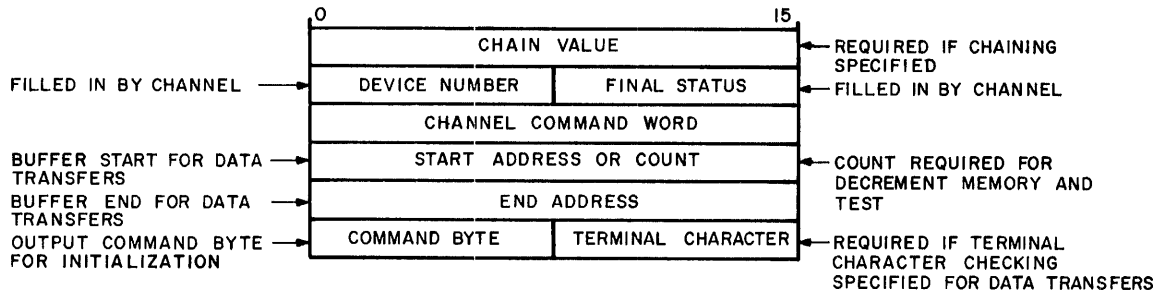


Figure 2-6. Channel Control Block Diagram

2.4.4.3 Channel Termination Queue

The Channel Termination Queue is a circular list identical to those described under List Processing Instructions in Section 4.11. The queue may be set up at any convenient core location. The maximum size of the queue allows for 255 entries, but any convenient length may be used. The address of the queue must be stored at location X'0080' prior to starting any channel program. The Automatic I/O Channel uses the queue to indicate termination of a channel program.

2.4.4.4 General Operation

When the Processor detects the presence of an interrupt signal from a peripheral device, it automatically acknowledges the signal and obtains the address of the device. It uses the device address times two to index into the Automatic I/O Service Table to the entry reserved for the device. If Bit 15 of the entry is reset, the Processor takes an Immediate Interrupt. If Bit 15 is set, the Processor activates the I/O Channel. The I/O Channel uses the entry minus one to locate the Channel Command Word. It decodes the command word and performs the required service, using the data entries in the Channel Command Block as necessary. If the channel operation for this device is not yet complete, the I/O Channel returns control to the Processor. The Processor now checks for pending interrupt signals. If none are present, it continues program execution. If any are present, it services them before returning to program execution.

If the channel determines that the operation for this device is complete, it terminates the channel program by storing the device address and final status in the Control Channel Block (CCB) and, for data transfers, changes the Channel Command Word to a "no operation". This causes subsequent interrupt signals from the device coming to this Channel Command Word to be ignored. At this point, the I/O Channel can take any or all of the following actions:

1. Make an entry in the Termination Queue.
2. Chain to another Channel Command Word.
3. Generate an Immediate Interrupt.

The action taken by the channel depends on the bit configuration of the Channel Command Word. Figure 2-4 shows the Channel Operation in block diagram form.

In the queuing operation, the channel generates a Queue Overflow Interrupt if the queue is full when it attempts to make an entry.

2.4.4.5 Channel Command Words

There are three phases involved in channel operations. These are:

1. Initialization
2. I/O Operation
3. Termination

All three phases are controlled by the bit configuration of the Channel Command Word. A single Channel Command Word can be encoded to perform all three types of operation. The bit assignments for Channel Command Words are shown in Figure 2-7.

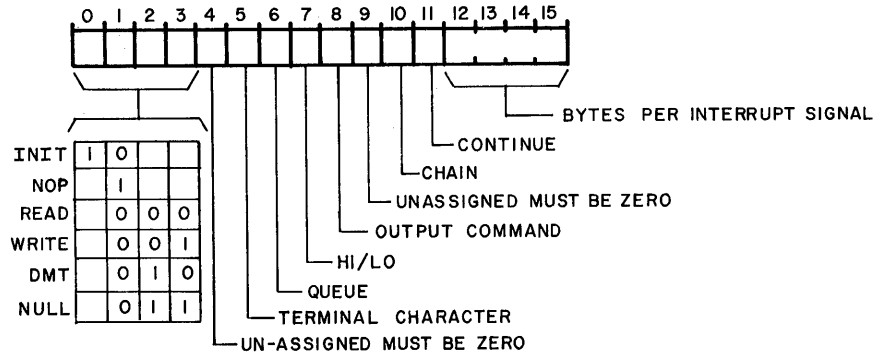


Figure 2-7. Bit Configuration For Channel Command Word

2.4.4.6 Initialization

Bits 0 (INIT) and 8 (Output Command) of the Channel Command Word control the initialize phase of channel operations. If Bit 0 (INIT) is set when the Channel decoded the command word, it resets Bit 0 (INIT) and checks Bit 8 (Output Command). If Bit 8 is set, the channel issues the output command located at the start of the Channel Control Block plus ten and returns control to the Processor. Channel operations with the device resume when an interrupt signal from the device occurs. Since the channel resets bit zero, it can pass through the initialize phase only once. This phase is optional. The software may initialize the device by Output Command instructions prior to starting the channel operation. The bit configurations of the CCW for the Initialization phase are illustrated in Figure 2-8.

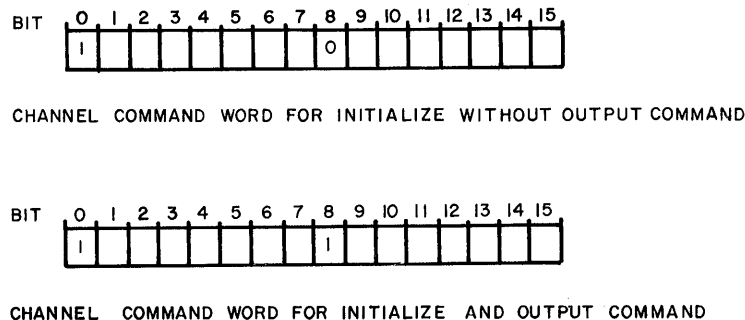


Figure 2-8. Channel Command for Initialize and Output Commands

2.4.4.7 I/O Operation

There are five distinct types of I/O operation the Automatic I/O Channel can perform. These are:

1. Read
2. Write
3. Decrement Memory and Test
4. No Operation
5. Null

The Channel Command Word (CCW) configurations for these operations are illustrated in Figure 2-9.

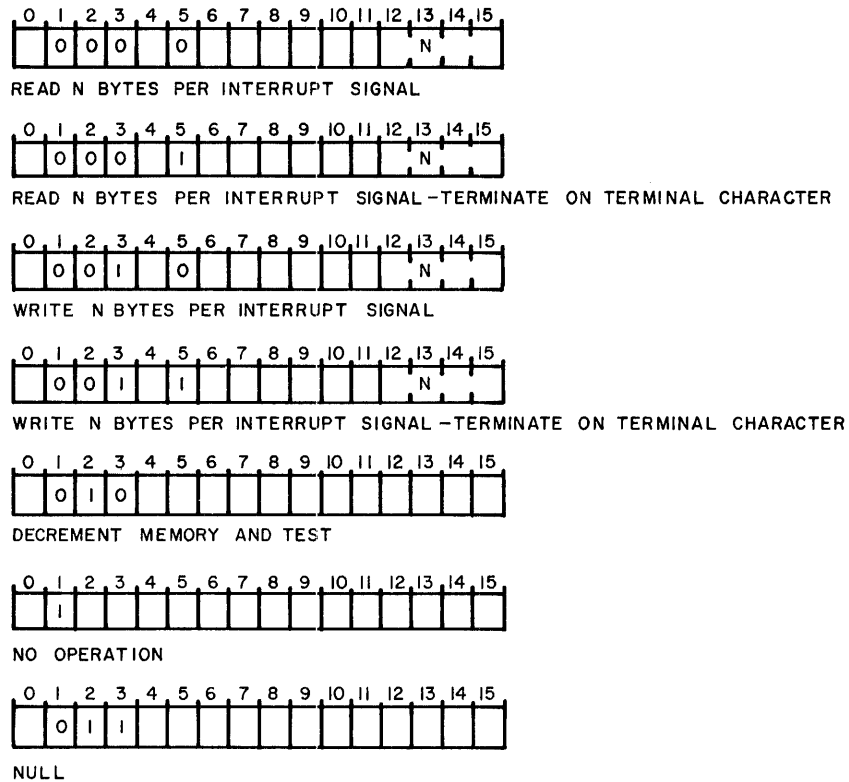


Figure 2-9. Channel Command Words for I/O Operation

For all Read/Write operations, Bits 12 through 15 must contain the number of bytes to be transferred on each interrupt signal. All zeroes in these bit positions indicates that sixteen bytes are to be transferred on each interrupt signal. The two halfwords following the Channel Command Word in the Channel Control Block must contain the starting address of the I/O Buffer and the ending address of the I/O Buffer. After the number of bytes specified for each interrupt signal has been transferred, the starting address is incremented by the appropriate amount and compared to the ending address. If it is greater, the channel enters the termination phase. If it is less, the channel returns control to the Processor for program execution. Bit 5 of the Channel Command Word controls the optional terminal character data transfer. When this bit is set, the transfer proceeds as described above with the exception that the last byte transferred on each interrupt signal is compared with the terminal character byte located at Channel Control Block plus eleven. If these two bytes match, the channel enters the termination phase. In this way, a channel program can terminate because the buffer is exhausted or a terminal character has been found in the data stream.

Before starting a data transfer, the Automatic I/O Channel checks the device status. Any non-zero status condition will stop the transfer and cause the channel to enter the termination phase. Before entering the termination phase, the Initial (INTI) bit and No Operation bit are set in the Channel Command Word, the Queue bit is set to force an entry in the Termination Queue, and the Chain bit and Continue bit are reset to prevent chaining.

The Decrement Memory and Test Operation causes the value contained in the halfword immediately following the Channel Command Word to be decremented by one for each interrupt signal. The new value is compared to zero. If greater than zero, the channel returns control to the Processor for program execution. If equal to zero, the channel enters the termination phase without changing the Channel Command Word to a "no operation". Subsequent interrupt signals from the device will cause the count field to increase negatively. The No Operation code in the Channel Command Word indicates that the channel is to ignore any interrupt signal from the associated device. The channel itself sets this code in the command word on completion of data transfers. The software can use this code to ignore unsolicited interrupt signals. The Automatic I/O Service Table should contain pointers to "no operation" control words for all non-existent devices.

The Null Operation differs from the No Operation in that while no I/O function is performed, the channel enters the termination phase without setting the No Operation code.

2.4.4.8 Termination

The Automatic I/O Channel enters the termination phase upon completion of a data transfer, when the count field of a Decrement Memory and Test Operation has reached zero, or when the Null Operation is decoded. All of the operations in the termination phase are optional. If none are specified, the channel returns control to the Processor. The two termination functions are Queue and Chain. The CCW bit configuration for Queuing and Chaining is shown in Figure 2-10. Bit 6 of the Channel Command Word controls queuing. If this bit is set, the channel, on entering the termination phase, stores the address of the Channel Command Word in the Channel Termination Queue. The condition of Bit 7 of the Channel Word controls positioning with the queue. If Bit 7 is set, the entry is made at the bottom of the queue. If Bit 7 is reset, the entry is made at the top of the queue. See Figure 2-11.

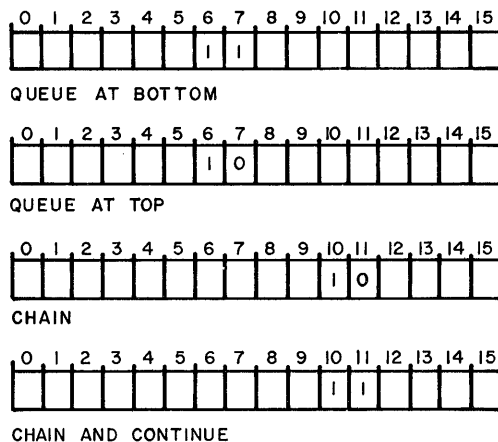


Figure 2-10. Channel Command Words for Termination

Bit 10 of the Channel Command Word controls chaining. In this operation, the channel stores the first halfword of the Channel Control Block in the appropriate location in the Automatic I/O Service Table for this device. This chain value may be either the address of another Channel Command Word or the address of a PSW exchange location for the Immediate Interrupt. Subsequent interrupt signals will be handled as indicated by this value. If the Chain bit and the Continue bit (Bit 11) are both set, the channel checks the new value placed in the Service Pointer Table and takes appropriate action before returning control to the Processor. In this way, depending on the new value stored in the Service Pointer Table, the channel can either generate an Immediate Interrupt or start another channel program.

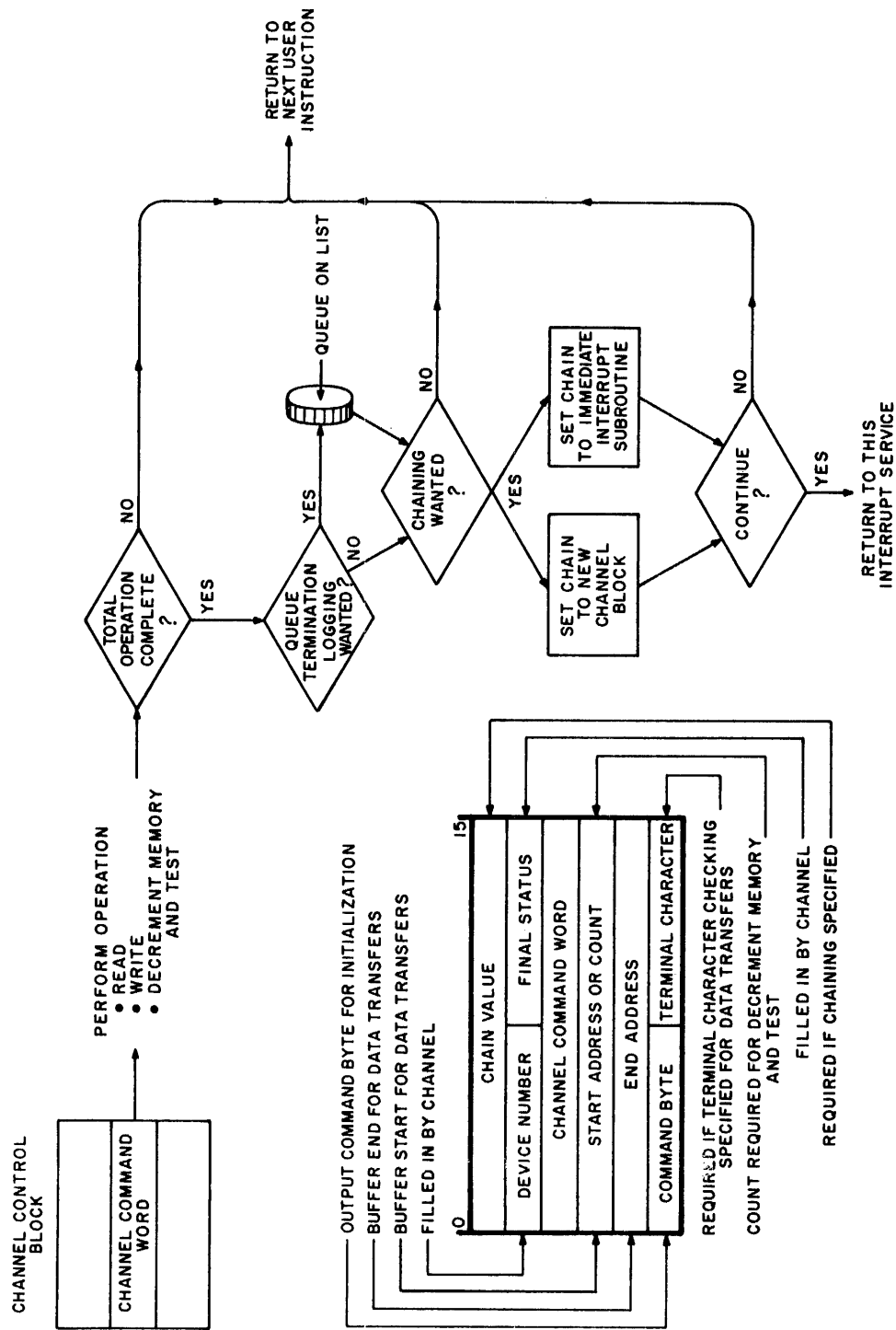


Figure 2-11. Automatic I/O

2.4.4.9 Example of Channel I/O Programming

This example of Channel I/O Programming assumes a Teletype located at physical address X'02'. The program is set up to:

1. Issue an Output Command to start the device.
2. Write 72 bytes from core memory to the device, one byte per interrupt signal.
3. On completion of the transfer, make an entry at the bottom of the Termination Queue and chain to a second Channel Command Block without specifying Continue.
4. The second Channel Command Block writes an additional 72 bytes to the device and terminates by chaining to an Immediate Interrupt and causing the interrupt to occur.

The first Channel Command Block is shown in Figure 2-12A. The Chain value is set to point to the second Channel Command Block. The Status Byte and Device Number are set to zero. The Channel Command word is set for Initialize, Write, Queue, Queue Low, Output Command, and Chain; and transfers one byte per interrupt signal. The next two halfwords point to the beginning and end of the 72 byte buffer. The Output Command byte is set to enable and write. The user program stores the address of this Channel Command Block in location X'00D4', the Service Pointer Table entry for device X'02'. It issues a Simulate Interrupt instruction specifying device X'02' to get the operation started. On execution of the Simulate Interrupt instruction, the channel issues the Output Command and resets the Initialize bit. It gives control to the Processor for the execution of normal instructions. As each subsequent interrupt signal is received from the device, the Channel outputs one byte until it has output the entire buffer of 72 characters. Between each interrupt signal it returns control to the Processor. After the last byte has been transferred, it sets the No Operation bit in the Channel Command Word, and puts the address of the Channel Command Word at the bottom of the Channel Termination Queue located at the address specified by the contents of X'0080' (Termination Queue Pointer). It stores the chain value (the address of the second Channel Command Word) in location X'00D4', Service Pointer Table entry for device X'02'.

On the next and on each subsequent interrupt signal the Channel is directed to the second Channel Command Block. This block is illustrated in Figure 2-12B. The Chain value points to an Immediate Interrupt location. The status and Device Number are set to zero. The Channel Command Word specifies Write, Chain, and Continue. The Channel outputs the data as described above until the last byte is written. It then sets the No Operation bit, stores the Immediate Interrupt address in the Service Pointer Table location for device X'02', and generates an interrupt allowing software to take over servicing this device. If, during the data transfer, the Channel had received an unsatisfactory status from the device, it would have terminated the operation by setting the Initialize and No Operation bits (bad status indicators) in the Channel Command Word, suppressed Chaining, and forced an entry at the top of the Channel Termination Queue. Setting the Continue bit in the second Channel Command Word caused the Channel to generate the Immediate Interrupt on the same interrupt signal that caused output of the last data byte. If this bit were reset, the next interrupt signal from the device would generate the Immediate Interrupt.

2.4.5 Special Interrupts

The Processor provides two classes of special interrupts. These are the Console Interrupt and the Memory Protect Interrupts.

2.4.5.1 Console Interrupt

The Processor provides for operator intervention in the following manner. If Bit 4 of the Current PSW is set, the rotary Function switch is in the OFF position, and the RUN Function switch is depressed, depressing the EXECUTE switch causes an interrupt signal from Device 01. Servicing this signal can be accomplished through the Immediate Interrupt.

2.4.5.2 Memory Protect Interrupt

If Bit 7 (Protect Mode) of the Current PSW is set, the Processor is said to be in the Protect Mode. Should the program attempt to store into a protected core area (as defined by the mask in the Memory Protect Controller) while the Processor is in the Protect Mode, an Interrupt signal is generated by the Memory Protect Controller. Furthermore, if Bit 1 (External Interrupt Enable) of the Current PSW is set, this interrupt is recognized and appropriate action can be taken. Refer to the Appendix for details on the memory Protect Controller. Note that if the External Interrupt is disabled, the Memory Protect Interrupt cannot occur, however, the store operation is ignored and execution resumes at the next instruction.

CHAPTER 3

DATA AND INSTRUCTION FORMATS AND STORAGE ADDRESSING

3.1 INTRODUCTION

A program is a set of instructions which directs the Processor to perform a specific task. Ordinarily, program instructions are stored in sequential memory locations. During the normal processing of a program, an instruction is fetched from the location specified by the Location Counter, the instruction is executed, the Location Counter is incremented, and another fetch and execute cycle begins.

3.2 DATA FORMATS

The INTERDATA Instruction Set manipulates data of three different word lengths: 8-bit bytes, 16-bit halfwords and 32-bit fullwords. This data may represent a fixed-point number, a floating-point number, or logical data. The data is used as operands for the instructions, and is manipulated as indicated by the particular instruction being executed.

3.2.1 Hexadecimal Notation

Binary information is expressed in hexadecimal notation (base 16) for purposes of simplicity. All references to binary instructions, data, or addresses in INTERDATA software are made in hexadecimal notation. Four binary bits of information are conveniently expressed by a single hexadecimal digit. Thus, byte information is expressed by two hexadecimal digits, halfword information by four hexadecimal digits, and fullword information by eight hexadecimal digits. Table 3-1 lists hexadecimal, binary and decimal equivalents.

3.2.2 Fixed-Point Data

The basic Fixed-Point Arithmetic operand is the 16-bit halfword. In multiply and divide operations, 32-bit fullwords are manipulated. See Figure 3-1.

Fixed-point data is treated as signed 15-bit integers in the halfword format, and as signed 31-bit integers in the fullword format. Positive numbers are expressed in true binary form with a sign bit of zero. Negative numbers are represented in two's complement form with a sign bit of one. Refer to Section 3.2.2.1 for further details.

The numerical value of zero is always represented with all bits zero. Table 3-2 shows several examples of the fixed-point number representation used in INTERDATA Systems.

Since halfword arithmetic operands are 16-bit values, Fixed-Point Arithmetic instructions can be used for address arithmetic. Logical, and Shift instructions can also be used for address manipulation or computation.

For details on manipulating fixed-point quantities, refer to Section 4.3.

3.2.2.1 2's Complement Notation

Negative numbers are represented in 2's complement notation. A fixed-point number is negative only if Bit 0 is set. To change the sign of a number, the 2's complement of the number is produced in a two-step procedure:

1. Change all zeros to ones, and change all ones to zeros (complement every bit).
2. Add 1 to the number.

Example: The number five is represented in binary form as
0000 0000 0000 0101

Step 1. 1111 1111 1111 1010 (complement) = 1's complement of 5

Step 2. 1111 1111 1111 1011 (add one) = 2's complement of 5

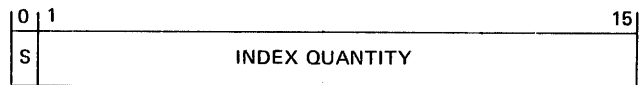
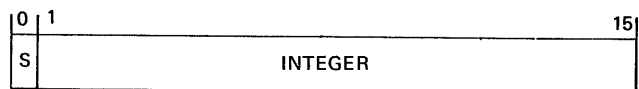
The result is the 2's complement of 5, representing -5.

TABLE 3-1
HEXADECIMAL, BINARY, AND DECIMAL CROSS-REFERENCE

Hexadecimal	Binary	Decimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

Example: The binary 2-byte number (0001111110100101) can be expressed in hex notation as X'1FA5'

HALFWORD



FULLWORD

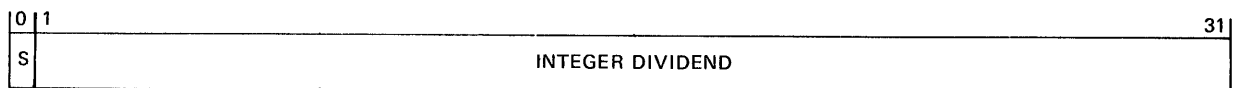
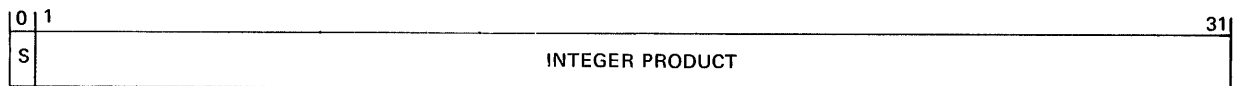


Figure 3-1. Fixed-Point Word Formats

TABLE 3-2
EXAMPLES OF FIXED-POINT REPRESENTATION

Number	Decimal	Binary				Hexadecimal
$2^{15}-1$	32767	0111	1111	1111	1111	7FFF
2^0	1	0000	0000	0000	0001	0001
0	0	0000	0000	0000	0000	0000
-2^0	-1	1111	1111	1111	1111	FFFF
-2^{15}	-32768	1000	0000	0000	0000	8000

3.2.3 Floating-Point Data

A floating-point number consists of a signed exponent and a signed fraction. The quantity expressed by this number is the product of the fraction and the number 16 raised to the power of the exponent. Each floating-point value requires two halfwords. The floating-point format is shown in Figure 3-2.

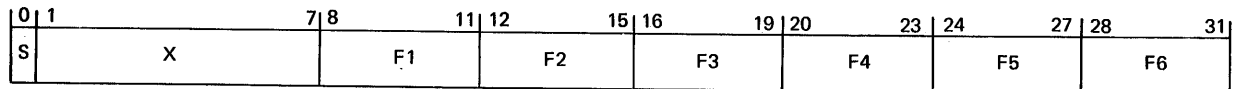


Figure 3-2. Floating-Point Word Format

Sign and magnitude representation is used, in which the sign bit S is zero for positive values, and one for negative values. The exponent X is expressed in excess 64 binary notation; that is, field X contains the true value of the exponent +64.

The fraction contains six hexadecimal digits F1-F6. The value of a floating-point fraction can be expressed as follows: $F_1 \cdot 16^{-1} + F_2 \cdot 16^{-2} + F_3 \cdot 16^{-3} + \dots + F_6 \cdot 16^{-6}$.

A normalized floating-point number has a non-zero high-order hexadecimal fraction digit (F₁). If the high-order hexadecimal fraction digit (F₁) is zero, the number is said to be unnormalized. The range of the magnitude (M) of a normalized floating-point number is:

$$16^{-65} \leq M \leq (1 - 16^{-6}) \cdot 16^{63}$$

or approximately

$$5.4 \cdot 10^{-79} \leq M \leq 7.2 \cdot 10^{75}$$

Table 3-3 shows several examples of the floating-point number representation used in INTERDATA Systems.

All floating-point numbers are assumed to be normalized prior to their use as operands. No pre-normalization is performed, all results are post-normalized. The Floating-Point Load instruction will normalize unnormalized floating-point numbers.

Exponent overflow is defined as a resultant exponent greater than 63. Exponent underflow is defined as a resultant exponent less than -64. The Overflow flag is set whenever exponent overflow or underflow is detected. The Greater Than flag is set on positive overflow, the Less Than flag is set on negative overflow, and both flags are reset on underflow. On overflow, the exponent and fraction of the result are set to all ones. The sign of the result is not affected by the overflow. On underflow, the sign, exponent and fraction of the sum are set to zero.

TABLE 3-3
EXAMPLES OF FLOATING-POINT REPRESENTATION

Decimal Value	Binary				Hexadecimal Value
1.0	0100 0000	0001 0000	0001 0000	0000 0000	4110 0000
-1.0	1100 0000	0001 0000	0001 0000	0000 0000	C110 0000
9.5	0100 0000	0001 0000	1001 0000	1000 0000	4198 0000
-0.5	1100 0000	0000 0000	1000 0000	0000 0000	C080 0000
$-(1-16^{-6}) \cdot 16^{63}$	1111 1111	1111 1111	1111 1111	1111 1111	FFFF FFFF
-16^{-65}	1000 0000	0000 0000	0001 0000	0000 0000	8010 0000
$0.1 + 16^{-6}$	0100 1001	0000 1001	0001 1001	1001 1010	4019 999A

The floating-point value in which all data bits are zero is called true zero. A true zero may arise as the result of an arithmetic operation because of exponent underflow, or when a resultant fraction is zero because of loss of significance. In general, zero values participate as normal numbers in all arithmetic operations.

There are eight 32-bit Floating-Point Registers, which are addressed with the even numbers 0, 2, 4, . . . , 14. The Floating-Point Registers are reserved core memory locations and are addressable only by the Floating-Point instructions, which are described in Section 4.10.

3.2.4 Logical Data

Logical operations manipulate 8-bit bytes, 16-bit halfwords, and 32-bit fullwords. All bits participate in logical operations. The data words have the format shown in Figure 3-3.

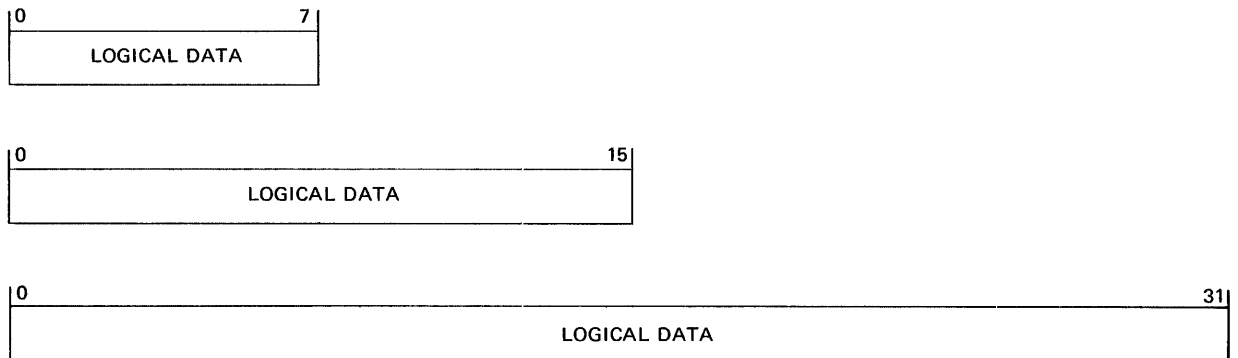


Figure 3-3. Logical Data Word Formats

For upward compatibility with future machines, boundary conventions for halfwords and fullwords should be observed.

3.3 INSTRUCTION FORMATS

Processor instructions represent one of four formats designated Register to Register (RR), Short Format (SF), Register to Indexed Memory (RX) and Register to Storage (RS) instructions.

In general, each format specifies three things: The operation to be performed, the address of the first operand, and the address of the second operand. The first operand is normally the contents of a General Register. The second operand is normally the contents of another General Register, the contents of a core memory location, or a data constant from the instruction word itself.

A 16-bit halfword format is used for Register-to-Register and Short Format instructions. The Short Format instructions may be used to manipulate small quantities or execute short branches relative to the present Location Counter. A 32-bit fullword format is used for the Register to Indexed Memory, and the Register to Storage formats. The specific formats are shown in Figure 3-4.

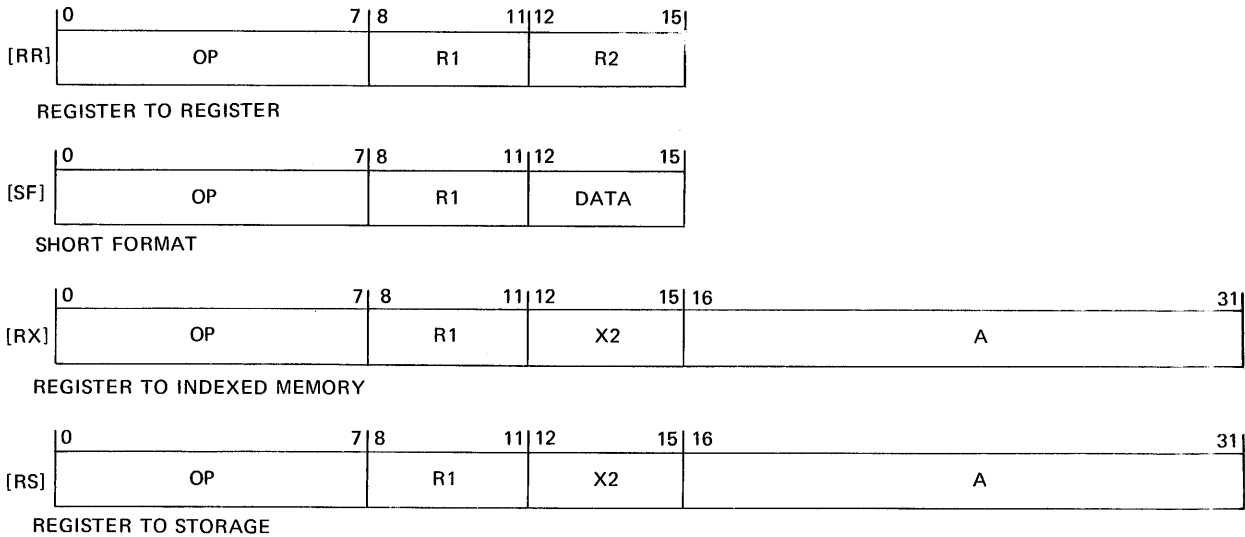


Figure 3-4. Instruction Word Formats

The eight-bit OP field in all formats specifies the machine operation to be performed. Operation codes are represented as two hexadecimal characters.

The four-bit R1 field in the instruction formats specifies the address of the first operand. The R1 field is normally the address of a General Register.

The four-bit R2 field in the RR instruction format specifies the address of the second operand, which is normally a register address.

The four-bit DATA field of the SF instructions supplies data in the case of Fixed-Point Arithmetic instructions, or a displacement from the current Location Counter in the case of Branch instructions.

A non-zero X2 field in the RX and RS formats specifies a General Register whose contents is used as an index value. The index value (X2) may be positive or negative. If X2 is zero, no address modification takes place. General Registers 1 through 15 can optionally be used for indexing, but General Register 0 can never be used for indexing.

The 16-bit Address field specifies a memory address in the RX format, or contains a value (data) to be used as an immediate operand in the RS format.

The RR instructions are used for operations between registers. The first operand is the contents of the register specified by the R1 field of the instruction word. The second operand is the contents of the register specified by the R2 field.

The SF instructions are used for; short immediates, in which the data field specifies a four-bit data value; short shifts, in which the data field specifies the shift count; and short branches, in which the data field specifies a displacement (in halfwords) from the current instruction address.

The RX instructions are used for operations between register and memory with the option of indexing. The first operand is the register specified by the R1 field of the instruction word. The second operand is the contents of the memory location specified by the A field of the instruction word, or the sum of the A field and the contents of the General Register specified by the X2 field if indexing is specified.

In the RS instructions, the first operand is the contents of the General Register specified by the R1 field of the instruction word. The second operand is the number contained in the A field of the instruction, or the sum of the A field and the contents of the General Register specified by the X2 field if indexing is specified. The second operand of an RS instruction specifies the number of bit positions in Shift instructions, or forms the second operand in Immediate instructions.

There are some exceptions to the first operand-second operand nomenclature used above. For example, with Branch on Condition instructions, the R1 field of the instruction is a four-bit mask (M1) which is ANDed with the Condition Code in the Current PSW. These instructions are discussed in Section 4.7. For all Input/Output instructions, the contents of the register specified by R1 specifies the device number for the I/O operation. For the Supervisor Call instruction, the R1 field specifies 1 out of 16 possible types of supervisor call. With the Load Program Status Word (LPSW), Simulate Interrupt (SINT) and Auto Load (AL) instructions, the R1 field must be zero. These instructions are described in Chapter 4.

Table 3-4 summarizes the first and second operand designations for each instruction format.

TABLE 3-4
DESIGNATIONS FOR FIRST AND SECOND OPERANDS

First Operand:	The contents of the register specified by the R1 Field (R1).	RR, RX, RS and SF
	The M1 Field	RR, RX, and SF Branch on Condition
	The actual value of the R1 Field.	SVC
Second Operand:	The contents of the register specified by the R2 Field (R2).	RR
	The contents of the address derived by adding the A field and the contents of the General Register specified by the X2 Field. $[A + (X2)]$	RX
	The A field plus the contents of the General Register specified by the X2 Field $A + (X2)$	RS
	The actual value of the R2 Field	SF

3.4 GENERAL REGISTER USAGE

The 16 General Registers function as accumulators or Index Registers in all arithmetic and logical operations. Each General Register is a 16-bit halfword consisting of two 8-bit bytes. For arithmetic operations, bit zero (leftmost position) is considered the sign bit using two's complement representation.

The General Registers are numbered from zero to fifteen (decimal), and written in hexadecimal notation as; 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. The General Registers have not been given specific functional assignments. However, the following operational restrictions should be noted:

1. It is not possible to use General Register 0 as an Index Register. In the RX and RS instruction formats, a zero entry in the X2 field indicates that no indexing is to take place.
2. For Fixed-Point Multiply, Divide, and fullword Shift and Rotate instructions, the R1 field must specify an even numbered General Register. See Sections 4.3 and 4.6.
3. For Branch on Index instructions, the R1 field specifies the first of three consecutive General Registers, and the value of the R1 therefore, should be equal to or less than 13. See Section 4.7.3.
4. For Floating-Point instructions the R1 field must be an even value, and specify one of the Floating-Point Registers rather than one of the General Registers.
5. With any RR type instruction, the R1 field and the R2 field can specify the same register, but special attention should be given to note what the instruction will do. For example, with the EPSR instruction, if the R1 field equals the R2 field, the program status is stored in a General Register, but the program status is unchanged.

3.5 STORAGE ADDRESSING

The INTERDATA Instruction Set manipulates data of three different word lengths: 8-bit bytes, 16-bit halfwords, or 32-bit fullwords. In each case, the bits are numbered from left to right, starting with the number zero. The format for each word length is shown in Figure 3-5.

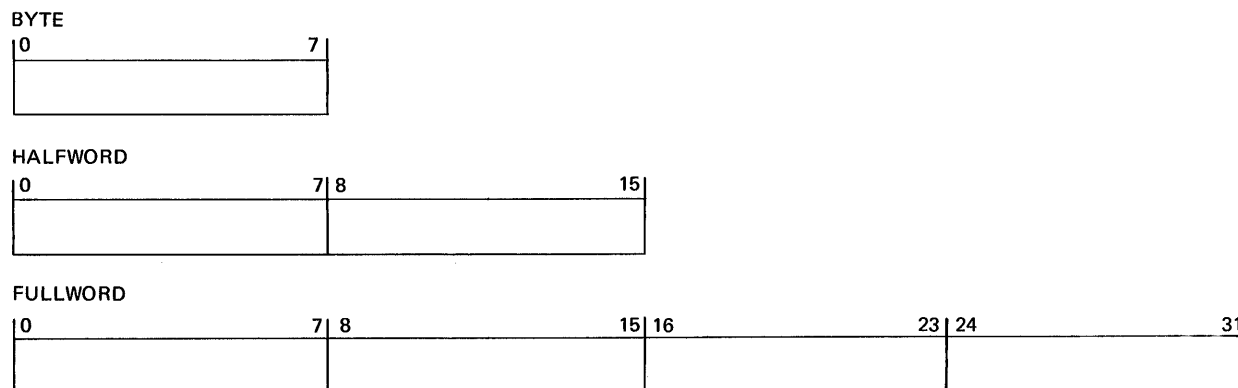


Figure 3-5. Data Word Formats

Core memory locations are numbered consecutively, beginning at 0000, for each 8-bit byte. Operands in memory are addressed by the RX type instructions. Since the address portion (A) of an RX instruction is 16-bits wide, it is possible to directly address 65,536 bytes.

The Processor transfers binary information between memory and the Processor as 16-bit halfwords. The instruction being performed determines if the address specified is that of a byte, a halfword, or a fullword. If a byte of information is desired, either the left or right byte of the halfword read from memory is manipulated as determined by the specific address. If a halfword of information is desired, the entire 16-bits read from memory are used. If a fullword is desired, a second 16-bits are read from memory and combined with the original halfword.

NOTE

Bytes of information are addressed by their specific hexadecimal address. A group of bytes combined to form a halfword or a fullword are addressed by the leftmost byte in the group. Halfword or fullword operands must be positioned at an address which is a multiple of two. Any memory reference for either a halfword or a fullword of information must reference that halfword or fullword with an address which is a multiple of two. The use of an address which is odd may yield an undefined result. Table 3-5 illustrates the addressing scheme.

TABLE 3-5
MEMORY ADDRESSING EXAMPLE

Address	0050	0051	0052	0053	0054	0055	0056	0057
Contents	01	23	45	67	89	AB	CD	EF
Operand Length and Position	Byte	Byte	Byte	Byte	Byte	Byte	Byte	Byte
	← Halfword →		← Halfword →		← Halfword →		← Halfword →	
	← Fullword →				← Fullword →			

For example, if the address referenced in Table 3-5 is 0050_{16} , then:

A Byte-Oriented instruction would extract the value 01_{16} , as an operand.

A Halfword-Oriented instruction would extract the value 0123_{16} as an operand.

A Fullword instruction would extract the value 01234567_{16} as an operand.

CHAPTER 4 INSTRUCTION REPERTOIRE

4.1 INTRODUCTION

The instruction repertoire has been grouped by function in this chapter. The use and operation of each instruction is presented in the following format:

1. An instruction word chart for each instruction including: Mnemonic operation code, and first and second operand designations in the correct assembler format. The format type designated by [SF], [RR], [RS], and [RX]. An instruction diagram with hexadecimal operation code and the locations of all fields is also provided, for example:

SIS	R1,N	[SF]
0	7 8	11 12
27	R1	N

SHR	R1,R2	[RR]
0	7 8	11 12
0B	R1	R2

SH	R1,A(X2)	[RX]
0	7 8	11 12
4B	R1	X2
		15 16
		31
		A

SHI	R1,A(X2)	[RS]
0	7 8	11 12
CB	R1	X2
		15 16
		31
		A

2. A description of instruction operation.
3. An example of a diagrammatic representation of instruction operation is shown below.

SIS: (R1) ← (R1) - N
 SHR: (R1) ← (R1) - (R2)
 SH: (R1) ← (R1) - [A + (X2)]
 SHI: (R1) ← (R1) - A - (X2)

4. A chart illustrating the possible variations of the Condition Code in the Current Program Status Word as a result of performing the instruction: a one indicates set, a zero indicates reset. It is important to note that any instruction which changes the Condition Code can change all four bits. The conditions listed on the chart are only those conditions which are meaningful after a particular instruction. Other bits may be changed, but their condition is not meaningful, for example:

Resulting Condition Code:

12	13	14	15	
C	V	G	L	
		0		DIFFERENCE IS ZERO.
		0	1	DIFFERENCE IS LESS THAN ZERO.
		1	0	DIFFERENCE IS GREATER THAN ZERO.
	1			ARITHMETIC OVERFLOW.
1				BORROW.

5. A programming note to provide additional pertinent or clarifying information. All privileged instructions and those instructions which may cause a memory protect violation are so noted, for example:

Programming Note:

The Subtract Immediate Short (SIS) instruction, causes the four-bit second operand N to be subtracted from the contents of the General Register specified by R1. This instruction is useful for decrementing a register by a small value (e.g. X'2').

The Subtract Halfword Immediate (SHI) instruction, produces a value which is the difference between the first operand General Register (R1), less the sum of the address field itself, and the content of a General Register index (X2).

The symbols and abbreviations used in the instruction diagrams are defined as follows:

()	Parentheses or Brackets. Read as "the content of ...".
[]	Arrow. Read as "is replaced by ..." or "replaces ...".
A	The 16-bit halfword address which is a part of the RX and RS instructions.
R1	The address of a General Register the content of which is the first operand.
M1	Mask of four-bits specifying Branch on Condition testing.
R2	The address of a General Register the content of which is the second operand of an RR instruction.
X2	The address of a General Register the content of which is used as an index value.
N	The four-bit second operand used with Short Format Immediate instructions.
D	The four-bit displacement value used with Short Format Branch instructions.
(0:7)	A bit grouping within a byte, a halfword, or a fullword. Read as "0 thru 7 inclusive"
(8:15)	"Bits 8 through 15 inclusive", etc.
(16:31)	
PSW	Program Status Word of 32 bits containing the Status, Condition Code, and current instruction address.
CC	Condition Code of four-bits contained in the PSW.
C	Carry Bit contained in the Condition Code (Bit 12 of PSW).
V	Overflow Bit contained in the Condition Code (Bit 13 of PSW).
G	Greater Than Bit contained in the Condition Code (Bit 14 of PSW).
L	Less Than Bit contained in the Condition Code (Bit 15 of PSW).
+	Arithmetic operations - Add,
-	Subtract,
*	Multiply,
/	and Divide respectively.
:	Logical comparison, when used (e.g., R1:R2).

4.2 FIXED-POINT LOAD/STORE INSTRUCTIONS

The Fixed-Point Load/Store instructions are used to transfer fixed-point (see 3.2.2) data between the General Registers and core memory. The instructions described in this section are:

- 4.2.1 LIS Load Immediate Short
- LCS Load Complement Short
- LHR Load Halfword RR
- LH Load Halfword
- LHI Load Halfword Immediate

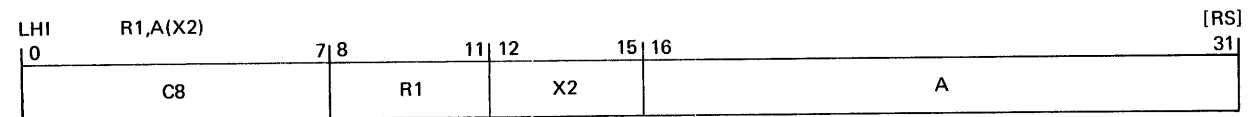
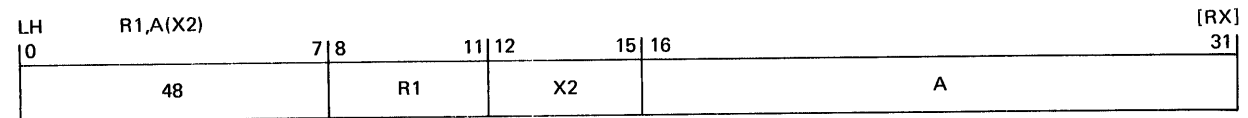
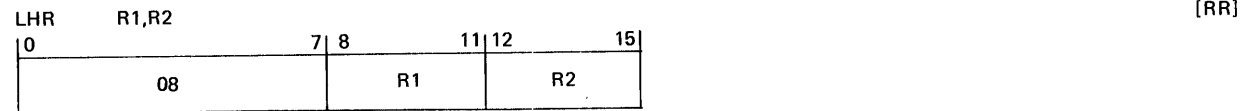
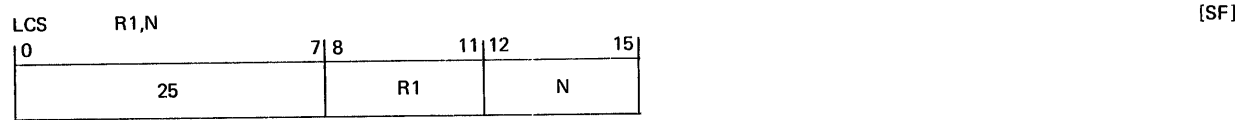
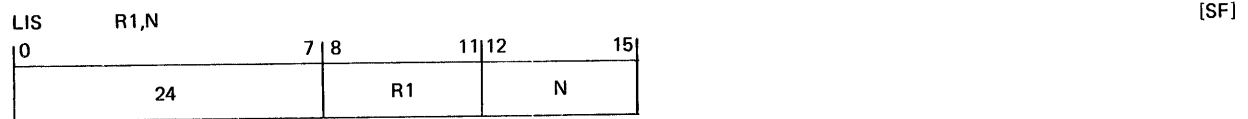
- 4.2.2 LM Load Multiple

- 4.2.3 STH Store Halfword

- 4.2.4 STM Store Multiple

4.2.1 Load Halfword

4.2.1 Load Halfword



The second operand is loaded into the General Register specified by R1.

LIS: (R1) ← N
 LCS: (R1) ← -N
 LHR: (R1) ← (R2)
 LH: (R1) ← [A + (X2)]
 LHI: (R1) ← A + (X2)

Resulting Condition Code:

12	13	14	15	
C	V	G	L	
		0	0	OPERAND IS ZERO.
		0	1	OPERAND IS LESS THAN ZERO.
		1	0	OPERAND IS GREATER THAN ZERO.

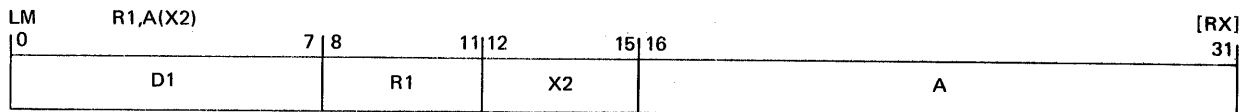
Programming Note:

The Load Immediate Short (LIS) instruction, causes the four-bit second operand to be expanded to a 16-bit halfword with high order bits set to zero. This halfword is loaded into the General Register specified by R1.

The Load Complement Short (LCS) instruction, causes the four-bit second operand to be expanded to a 16-bit halfword with high order bits set to zero. The two's complement of this halfword is loaded into the General Register specified by R1.

These instructions may be used to preset a register with an index value, load a register with the first operand for a subsequent arithmetic operation (e. g. add, multiply), or set the Condition Code for supplemental testing by a Branch on Condition instruction.

4.2.2 Load Multiple



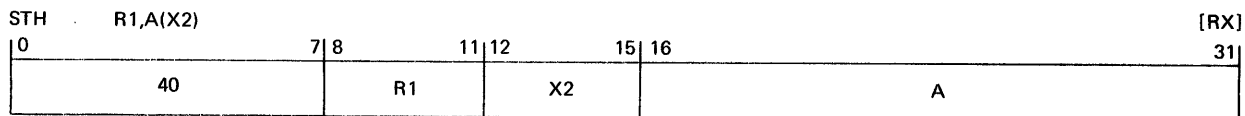
Sequential halfwords from memory are loaded into successive General Registers, beginning with the General Register specified by the R1 field. The first halfword is defined by A + (X2). The operation is terminated when R15 is loaded from memory. Note that any number of sequential General Registers can be loaded in this manner.

1. $(R1) \longleftarrow [A + (X2)]$
2. R1: X'F'
 if R1 = X'F', the instruction is finished
 if R1 \neq X'F', then:
3. $R1 \longleftarrow R1 + 1$
4. $A \longleftarrow A + 2$, return to Step 1

Resulting Condition Code:

Unchanged.

4.2.3 Store Halfword



The 16-bit first operand is stored in the core memory location specified by the second operand. The first operand is unchanged.

STH: $(R1) \longrightarrow [A + (X2)]$

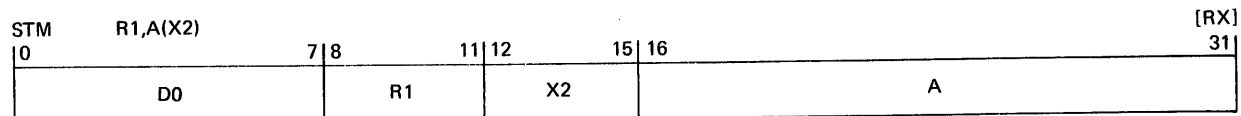
Resulting Condition Code:

Unchanged.

Programming Note:

This instruction is subject to Memory Protect.

4.2.4 Store Multiple



Successive General Registers are stored sequentially into memory, beginning with the General Register specified by the R1 field. The first storage address is determined by $A + (X2)$. The operation is terminated when R15 is stored in memory. Note that any number of sequential General Registers can be transferred in this manner.

1. $(R1) \longrightarrow [A + (X2)]$
2. R1: X'F'
 if $R1 = X'F'$, then instruction is finished
 if $R1 \neq X'F'$, then:
3. $R1 \longleftarrow R1 + 1$
4. $A \longleftarrow A + 2$, return to step 1

Resulting Condition Code:

Unchanged.

Programming Note:

This instruction is subject to Memory Protect.

The Store Multiple (STM) instruction in conjunction with the Load Multiple (LM) instruction is an aid to subroutine execution. They permit the easy saving and restoring of the registers required by the subroutine. The Store Multiple instruction can be used upon entering the subroutine, and the Load Multiple would be the last instruction executed before returning from the subroutine.

4.3 FIXED-POINT ARITHMETIC INSTRUCTIONS

The Fixed-Point Arithmetic instructions provide for addition, subtraction, multiplication and division of fixed-point data (see 3.2.2) contained in the General Registers and/or core memory. Also included are logical and arithmetic compare operations. The instructions described in this section are:

- 4.3.1 AIS Add Immediate Short
 - AHR Add Halfword RR
 - AH Add Halfword
 - AHI Add Halfword Immediate
 - AHM Add Halfword to Memory
- 4.3.2 ACHR Add with Carry Halfword RR
 - ACH Add with Carry Halfword
- 4.3.3 SIS Subtract Immediate Short
 - SHR Subtract Halfword RR
 - SH Subtract Halfword
 - SHI Subtract Halfword Immediate
- 4.3.4 SCHR Subtract with Carry Halfword RR
 - SCH Subtract with Carry Halfword
- 4.3.5 CLHR Compare Logical Halfword RR
 - CLH Compare Logical Halfword
 - CLHI Compare Logical Halfword Immediate
- 4.3.6 CHR Compare Halfword RR
 - CH Compare Halfword
 - CHI Compare Halfword Immediate
- 4.3.7 MHR Multiply Halfword RR
 - MH Multiply Halfword
- 4.3.8 MHUR Multiply Halfword Unsigned RR
 - MHU Multiply Halfword Unsigned
- 4.3.9 DHR Divide Halfword RR
 - DH Divide Halfword

4.3.1 Add Halfword

AIS	R1,N				[SF]
0		7	8	11	12
	26		R1		N
AHR	R1,R2				[RR]
0		7	8	11	12
	0A		R1		R2
AH	R1,A(X2)				[RX]
0		7	8	11	12
	4A		R1		X2
					A
AHI	R1,A(X2)				[RS]
0		7	8	11	12
	CA		R1		X2
					A
AHM	R1,A(X2)				[RX]
0		7	8	11	12
	61		R1		X2
					A

The second operand is added algebraically to the contents of the General Register specified by R1.

AIS: $(R1) \leftarrow (R1) + N$
 AHR: $(R1) \leftarrow (R1) + (R2)$
 AH: $(R1) \leftarrow (R1) + [A + (X2)]$
 AHI: $(R1) \leftarrow (R1) + A + (X2)$
 AHM: $[A + (X2)] \leftarrow (R1) + [A + (X2)]$

Resulting Condition Code:

12	13	14	15	
C	V	G	L	
		0	0	SUM IS ZERO.
		0	1	SUM IS LESS THAN ZERO.
		1	0	SUM IS GREATER THAN ZERO.
	1			ARITHMETIC OVERFLOW.
1				CARRY.

Programming Note:

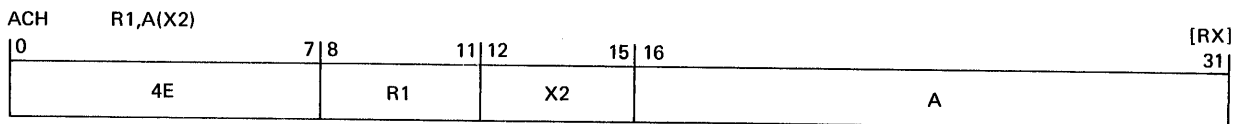
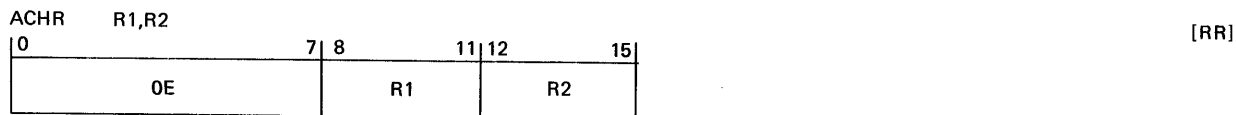
The Add Immediate Short (AIS) instruction, causes the four-bit second operand N to be added to the contents of the General Register specified by R1. The result replaces the contents of R1.

The Add Halfword Immediate (AHI) instruction, produces a value which is the algebraic sum of the address field itself, the content of a General Register index (X2), and the first operand General Register (R1).

The Add Halfword to Memory (AHM) instruction, causes the second operand $[A + (X2)]$ to be added to the contents of the General Register specified by R1. The result of the addition does not replace the contents of R1, but instead is stored in core memory at the address specified by $A + (X2)$. The first operand (R1) remains unchanged. This instruction effectively permits every location in core memory to be used as a counter.

This instruction is subject to Memory Protect.

4.3.2 Add with Carry Halfword



The 16-bit second operand and the Carry Bit of the Condition Code (PSW 12) are added algebraically to the General Register specified by R1. The resulting sum is contained in R1. The second operand is unchanged.

ACHR: $(R1) \leftarrow (R1) + (R2) + C$
 ACH: $(R1) \leftarrow (R1) + [A + (X2)] + C$

Resulting Condition Code:

12	13	14	15	
C	V	G	L	
		0	0	SUM IS ZERO.
		0	1	SUM IS LESS THAN ZERO.
		1	0	SUM IS GREATER THAN ZERO.
	1			ARITHMETIC OVERFLOW.
1				CARRY.

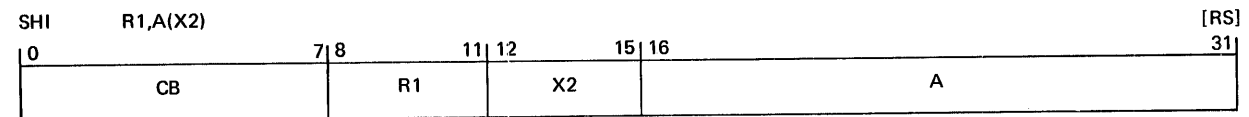
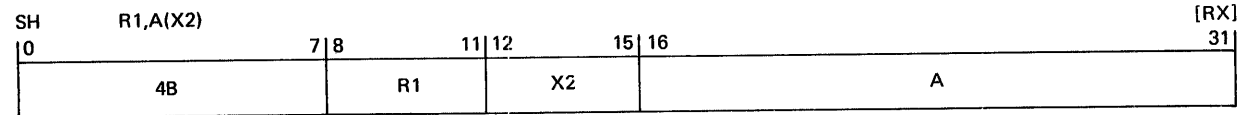
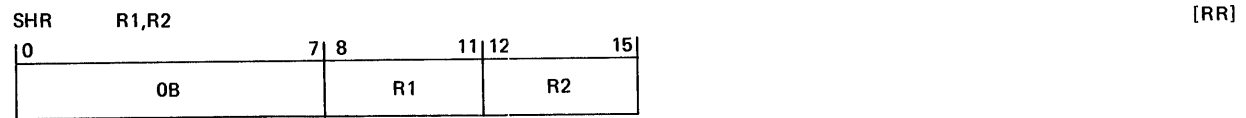
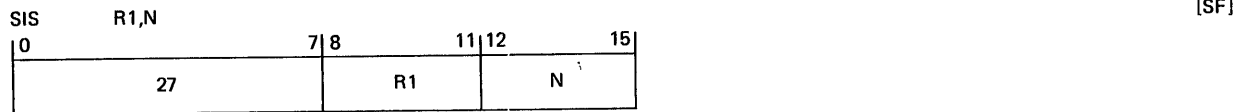
Programming Note:

Multiple precision addition operations require a Carry forward from the least significant operands to the most significant. To accomplish this, the locations containing the least significant portions of the two operands are summed, using the Add Halfword (AH) instruction. A Carry forward, if it occurs, is retained in the Carry Bit position of the Condition Code (PSW 12).

The locations containing the next least significant portions of the two operands are then summed, using the Add with Carry Halfword (ACH) instruction. The Carry Bit contained in the Condition Code (set from the previous addition) participates in this sum; the Carry Bit position is then set to reflect the new result.

The Add with Carry Halfword (ACH) instruction, is used on succeeding pairs of operands until the most significant operands of the multiple precision words have been summed. The resulting Condition Code is valid for testing the multiple precision word.

4.3.3 Subtract Halfword



The second operand is subtracted from the General Register specified by R1. The difference is contained in R1. The second operand is unchanged.

SIS: $(R1) \leftarrow (R1) - N$
 SHR: $(R1) \leftarrow (R1) - (R2)$
 SH: $(R1) \leftarrow (R1) - [A + (X2)]$
 SHI: $(R1) \leftarrow (R1) - A - (X2)$

Resulting Condition Code:

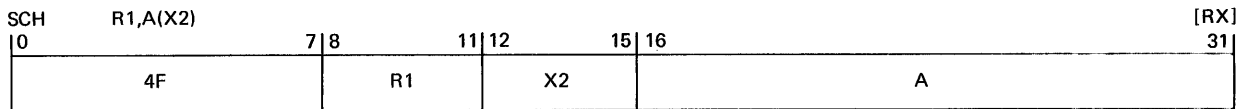
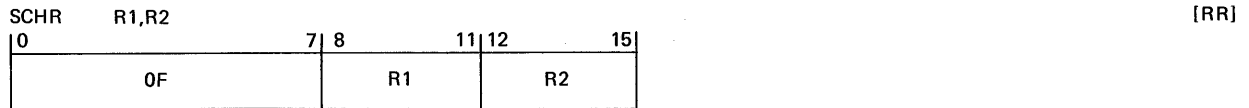
12	13	14	15	
C	V	G	L	
		0	0	DIFFERENCE IS ZERO.
		0	1	DIFFERENCE IS LESS THAN ZERO.
		1	0	DIFFERENCE IS GREATER THAN ZERO.
	1			ARITHMETIC OVERFLOW.
1				BORROW.

Programming Note:

The Subtract Immediate Short (SIS) instruction, causes the four-bit second operand N to be subtracted from the contents of the General Register specified by R1. This instruction is useful for decrementing a register by a small value (e. g. X'2').

The Subtract Halfword Immediate (SHI) instruction, produces a value which is the difference between the first operand General Register (R1), less the sum of the address field itself, and the content of a General Register index (X2).

4.3.4 Subtract with Carry Halfword



The 16-bit second operand with the Carry (borrow) Bit is subtracted from the General Register specified by R1. The difference is contained in R1. The second operand is unchanged.

SCHR: $(R1) \leftarrow (R1) - (R2) - C$
 SCH: $(R1) \leftarrow (R1) - [A + (X2)] - C$

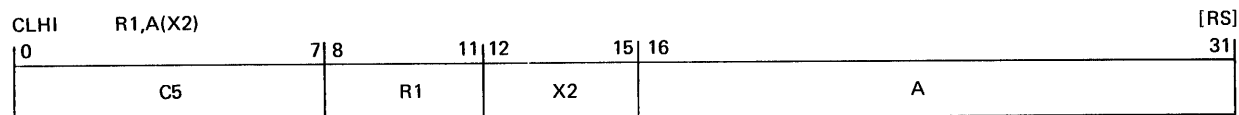
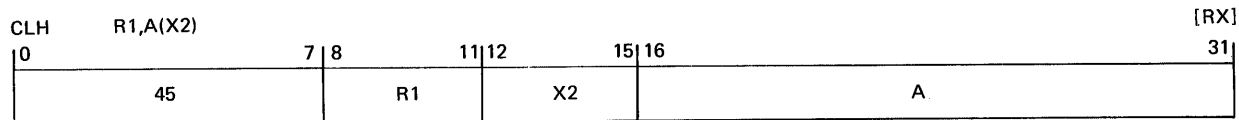
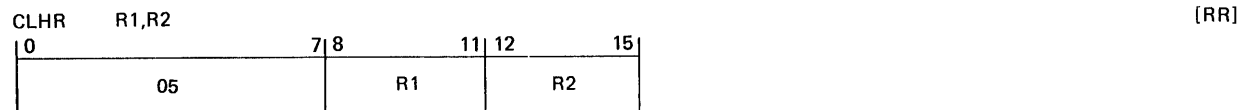
Resulting Condition Code:

12	13	14	15	
C	V	G	L	
		0	0	DIFFERENCE IS ZERO.
		0	1	DIFFERENCE IS LESS THAN ZERO.
		1	0	DIFFERENCE IS GREATER THAN ZERO.
	1			ARITHMETIC OVERFLOW.
1				BORROW.

Programming Note:

See Add with Carry Halfword 4.3.2.

4.3.5 Compare Logical Halfword



The first operand specified by R1 is compared logically to the 16-bit second operand. The result is indicated by the setting of the Condition Code [PSW (12:15)]. Both operands remain unchanged.

CLHR: (R1) : (R2)
 CLH: (R1) : [A + (X2)]
 CLHI: (R1) : A + (X2)

Resulting Condition Code:

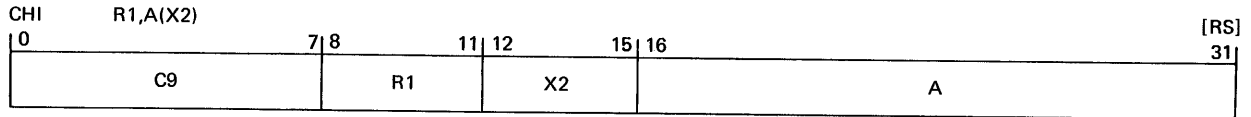
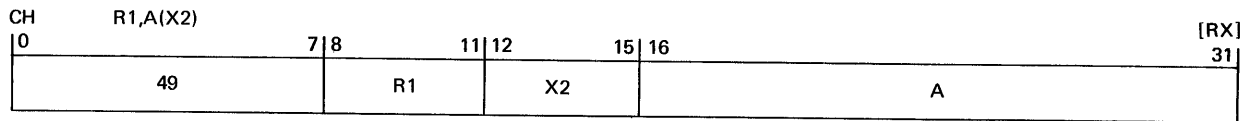
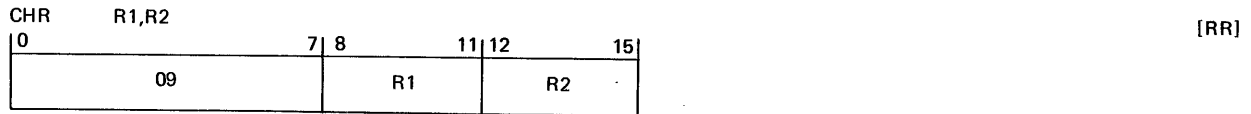
12	13	14	15	
C	V	G	L	
		0	0	FIRST OPERAND EQUAL TO SECOND OPERAND.
		0	1	} FIRST OPERAND NOT EQUAL TO SECOND OPERAND.
		1	0	
1				FIRST OPERAND LESS THAN SECOND OPERAND.
0				FIRST OPERAND EQUAL TO OR GREATER THAN SECOND OPERAND.

Programming Note:

The logical comparison is performed by subtracting the second operand from the first operand. The result is in the Condition Code setting, the operands are not modified.

The Compare Logical Halfword Immediate (CLHI) instruction, produces a value which is the logical comparison of the address field itself plus the content of a General Register index (X2) with the first operand General Register (R1).

4.3.6 Compare Halfword



The first operand specified by R1 is compared to the 16-bit second operand. The comparison is algebraic, taking into account the sign and magnitude of each number. The result is indicated by the setting of the Condition Code [PSW (12:15)]. Both operands remain unchanged.

CHR: (R1) : (R2)
 CH: (R1) : [A + (X2)]
 CHI: (R1) : A + (X2)

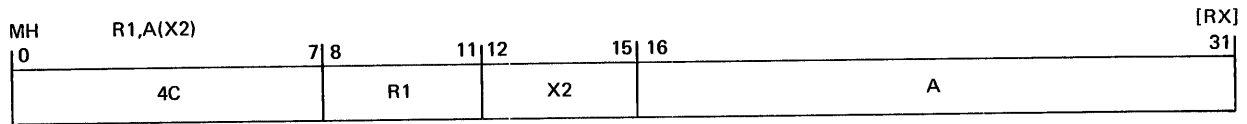
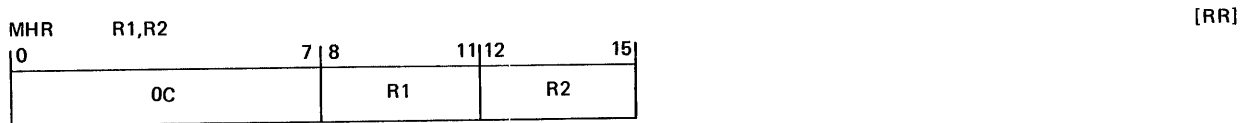
Resulting Condition Code:

	12	13	14	15	
	C	V	G	L	
			0	0	FIRST OPERAND EQUAL TO SECOND OPERAND.
			0	1	FIRST OPERAND LESS THAN SECOND OPERAND.
			1	0	FIRST OPERAND GREATER THAN SECOND OPERAND.
1					FIRST OPERAND LESS THAN SECOND OPERAND.
0					FIRST OPERAND EQUAL TO OR GREATER THAN SECOND OPERAND.

Programming Note:

The Compare Halfword (CH) instructions, permit arithmetic comparison of signed two's complement 16-bit integers. They facilitate fast comparisons for DO loop, and IF statement processing in FORTRAN.

4.3.7 Multiply Halfword



The 16-bit second operand is multiplied by the contents of the General Register specified by R1 + 1. The R1 field of the instruction must specify an even numbered register. The resulting 32-bit product is contained in R1 and R1 + 1, an even-odd pair; the second operand is unchanged. The sign of the product is determined by the rules of algebra.

MHR: (R1, R1 + 1) ← (R1 + 1) * (R2)
 MH: (R1, R1 + 1) ← (R1 + 1) * [A + (X2)]

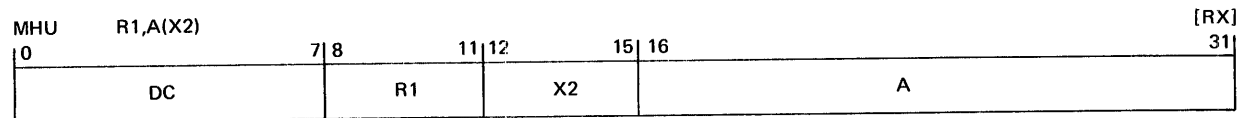
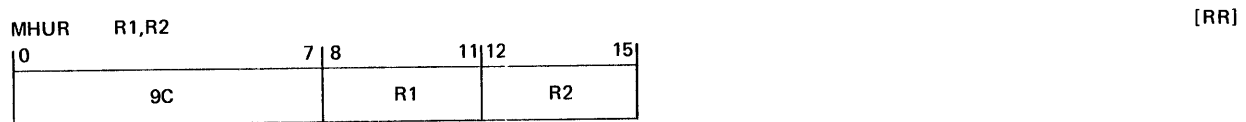
Resulting Condition Code:

Unchanged.

Programming Note:

After multiplication, the most significant 15 bits with sign are contained in R1. The least significant 16 bits are contained in R1 + 1.

4.3.8 Multiply Halfword Unsigned



The 16-bit second operand is multiplied by the contents of the General Register specified by R1 + 1. All 16-bits of both operands are considered to be magnitude. The resulting 32-bit product is contained in R1 and R1 + 1, the second operand is unchanged. The R1 field of the instruction must specify an even numbered register.

MHUR: (R1, R1 + 1) ← (R1 + 1) * (R2)
 MHU: (R1, R1 + 1) ← (R1 + 1) * [A + (X2)]

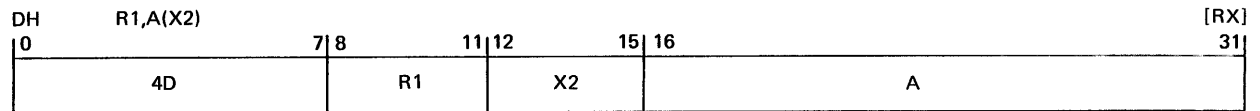
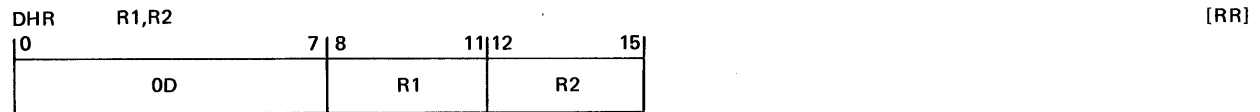
Resulting Condition Code:

Unchanged.

Programming Note:

This instruction is most useful in applications requiring multiple precision multiply capability. Typically, a Multiply Halfword (MH) instruction would be used with the most significant halfwords of the two operands, after the least significant parts of the two operands were multiplied using the Multiply Halfword Unsigned (MHU) instruction. The partial products could then be summed.

4.3.9 Divide Halfword



The 16-bit second operand is divided into the 32-bit dividend contained in the General Register specified by R1 and R1 + 1. The first operand, R1, must specify an even numbered register. The resulting 15-bit quotient with sign is contained in R1 + 1; a 15-bit remainder with sign is contained in R1, the second operand is unchanged. The sign of the result is determined by the rules of algebra; the sign of the remainder is the same as the sign of the dividend.

DHR: (R1 + 1) ← (R1, R1 + 1)/(R2)
 (R1) ← Remainder
 DH: (R1 + 1) ← (R1, R1 + 1)/ [A + (X2)]
 (R1) ← Remainder

Resulting Condition Code:

Unchanged.

Programming Note:

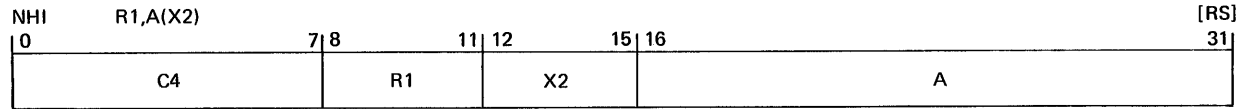
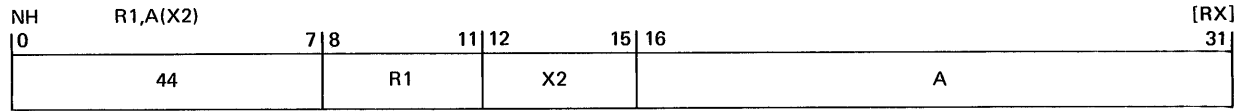
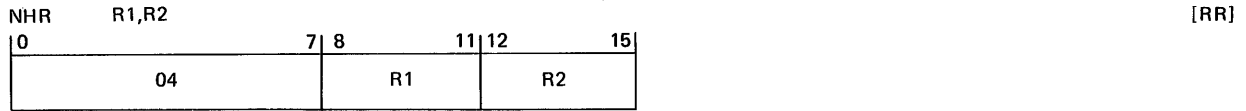
Attempted division by zero or a quotient which would be greater than X'8000' causes a Fixed-Point Divide Fault Interrupt, if enabled by Bit 3 of the Program Status Word. The operands remain unchanged.

4.4 LOGICAL AND BIT MANIPULATING INSTRUCTIONS

The Logical instructions manipulate logical data (see 3.2.4) such that each bit of the first operand is logically combined with the corresponding bit in the second operand. The instructions described in this section are:

- 4.4.1 NHR AND Halfword RR
- NH AND Halfword
- NHI AND Halfword Immediate
- 4.4.2 OHR OR Halfword RR
- OH OR Halfword
- OHI OR Halfword Immediate
- 4.4.3 XHR Exclusive OR Halfword RR
- XH Exclusive OR Halfword
- XHI Exclusive OR Halfword Immediate
- 4.4.4 THI Test Halfword Immediate

4.4.1 AND Halfword



The logical product of the 16-bit second operand and the content of the General Register specified by R1, replaces the content of R1. The 16-bit product is formed on a bit-by-bit basis.

NHR: (R1) ← (R1) AND (R2)
 NH: (R1) ← (R1) AND [A + (X2)]
 NHI: (R1) ← (R1) AND A + (X2)

Resulting Condition Code:

12	13	14	15	
C	V	G	L	
		0	0	} LOGICAL PRODUCT IS NOT ZERO.
		0	1	
		1	0	

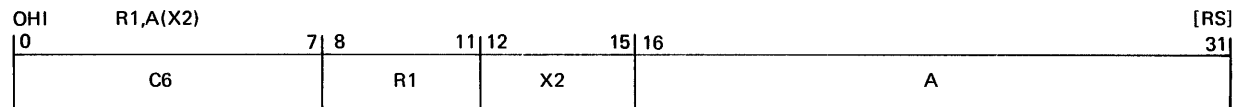
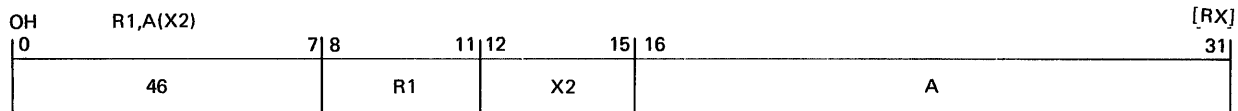
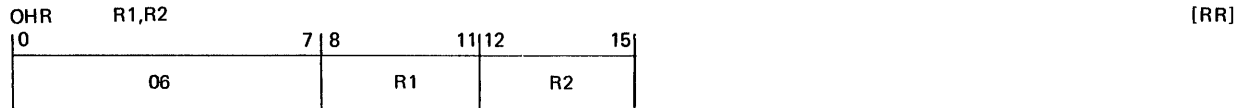
Programming Note:

The AND Halfword Immediate (NHI) instruction, produces a value which is the logical product of the address field itself plus the content of a General Register index (X2) with the first operand General Register (R1).

The truth table for the AND function is:

0 AND 0 = 0
 0 AND 1 = 0
 1 AND 0 = 0
 1 AND 1 = 1

4.4.2 OR Halfword



The logical sum of the 16-bit second operand and the content of the General Register specified by R1, replaces the content of R1. The 16-bit sum is formed on a bit-by-bit basis.

OHR: (R1) ← (R1) OR (R2)
 OH: (R1) ← (R1) OR [A + (X2)]
 OHI: (R1) ← (R1) OR A + (X2)

Resulting Condition Code:

12	13	14	15	
C	V	G	L	
		0	0	} LOGICAL SUM IS NOT ZERO.
		0	1	
		1	0	

LOGICAL SUM IS ZERO.

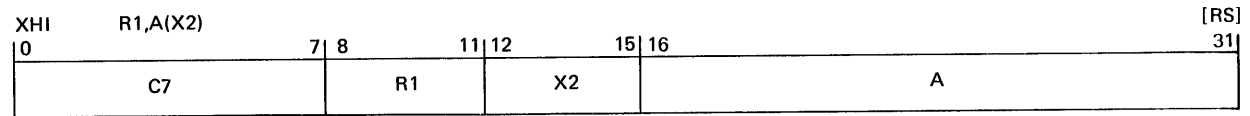
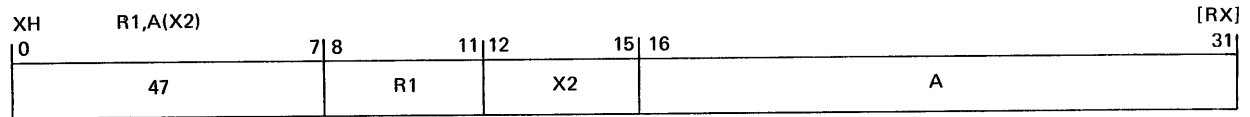
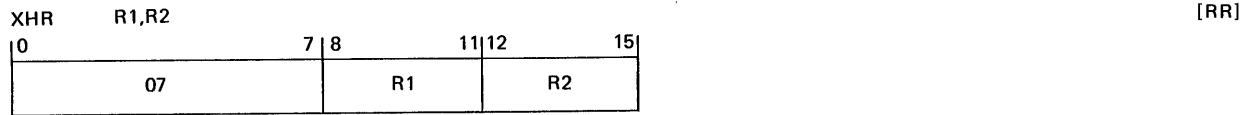
Programming Note:

The OR Halfword Immediate (OHI) instruction, produces a value which is the logical sum of the address field itself plus the content of the General Register index (X2) with the first operand General Register (R1).

The truth table for the OR function is:

0 OR 0 = 0
 0 OR 1 = 1
 1 OR 0 = 1
 1 OR 1 = 1

4.4.3 Exclusive OR Halfword



The logical difference of the 16-bit second operand and the General Register specified by R1, replaces the content of R1. The 16-bit difference is formed on a bit-by-bit basis.

XHR: (R1) ← (R1) XOR (R2)
 XH: (R1) ← (R1) XOR [A + (X2)]
 XHI: (R1) ← (R1) XOR A + (X2)

Resulting Condition Code:

12	13	14	15	
C	V	G	L	
		0	0	} LOGICAL DIFFERENCE IS NOT ZERO.
		0	1	
		1	0	

LOGICAL DIFFERENCE IS ZERO.

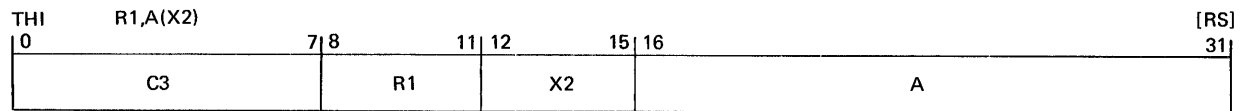
Programming Note:

The Exclusive OR Halfword Immediate (XHI) instruction, produces a value which is the logical difference of the address field itself plus the content of the General Register index (X2) with the first operand General Register (R1).

The truth table for the Exclusive OR function is:

0 XOR 0 = 0
 0 XOR 1 = 1
 1 XOR 0 = 1
 1 XOR 1 = 0

4.4.4 Test Halfword Immediate



Each bit in the 16-bit second operand is logically ANDed with the corresponding bit in the General Register specified by R1. The contents of R1 and the second operand remain unchanged.

THI: (R1) AND A + (X2)

Resulting Condition Code:

	12	13	14	15	
C	V	G	L		
		0	0		NONE OF THE BITS OF THE RESULT SET.
		0	1		BIT 0 OF THE RESULT SET.
		1	0		ONE OR MORE OF BITS 1-15 OF THE RESULT SET.

Programming Note:

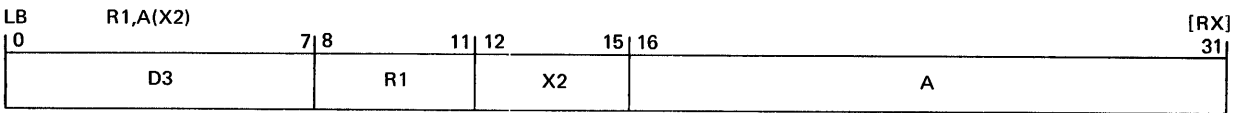
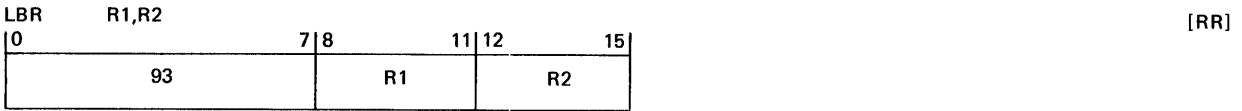
The Test Halfword Immediate (THI) instruction can be used to test the state of individual bits or combinations of bits in a General Register. For example, to test the state of Bit 6 in Register 3, use THI 3, X'0200'.

4.5 BYTE HANDLING INSTRUCTIONS

The Byte Handling instructions provide for transferring bytes between core memory and the General Registers. Compare Logical Byte is useful for testing a particular byte within memory. The instructions described in this section are:

- 4.5.1 LBR Load Byte RR
- LB Load Byte
- 4.5.2 STBR Store Byte RR
- STB Store Byte
- 4.5.3 EXBR Exchange Byte RR
- 4.5.4 CLB Compare Logical Byte

4.5.1 Load Byte



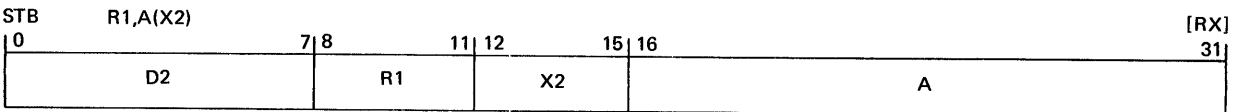
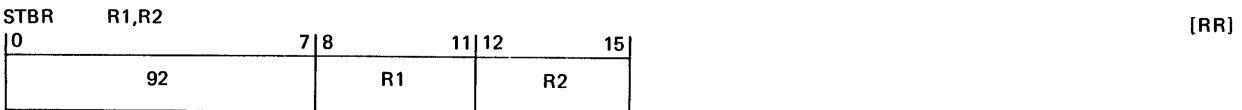
The eight-bit second operand is loaded into the right-most (least significant) eight-bits of the General Register specified by R1. The left-most (most significant) eight-bits of R1 are set to zero. The second operand is unchanged.

LBR: R1 (8:15) ← R2 (8:15)
 R1 (0:7) ← Zero
 LB: R1 (8:15) ← [A + (X2)]
 R1 (0:7) ← Zero

Resulting Condition Code:

Unchanged.

4.5.2 Store Byte



The right-most (least significant) eight-bit byte of the first operand is stored in the General Register or core memory location specified by the second operand. The first operand is unchanged.

STBR: [R1 (8:15)] → R2 (8:15)
 STB: [R1 (8:15)] → [A + (X2)]

Resulting Condition Code:

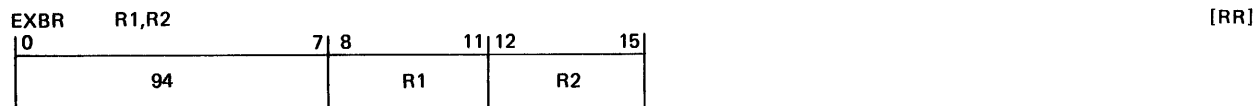
Unchanged.

Programming Note:

In the Register-to-Register (RR) form of this instruction, the left-most byte of R2, (0:7), is unchanged.

The RX Store Byte (STB) instruction is subject to Memory Protect.

4.5.3 Exchange Byte



The two eight-bit bytes of the second operand are exchanged and loaded into the General Register specified by R1.

EXBR: R1 (0:7) ← R2 (8:15)
 R1 (8:15) ← R2 (0:7)

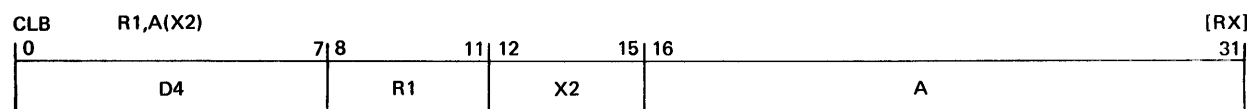
Resulting Condition Code:

Unchanged.

Programming Note:

R1 and R2 may specify the same General Register.

4.5.4 Compare Logical Byte



The least significant eight-bit byte of the first operand is logically compared to the eight-bit second operand. The result is indicated by the setting of the Condition Code [PSW (12:15)]. Neither operand is changed.

CLB: R1(8:15) : [A + (X2)]

Resulting Condition Code:

12	13	14	15	
C	V	G	L	
		0	0	FIRST OPERAND EQUALS SECOND OPERAND.
		0	1	FIRST OPERAND DOES NOT EQUAL SECOND OPERAND.
		1	0	
1				FIRST OPERAND IS LESS THAN SECOND OPERAND.
0				FIRST OPERAND IS EQUAL TO OR GREATER THAN SECOND OPERAND.

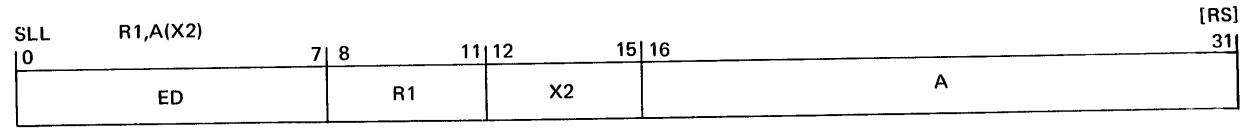
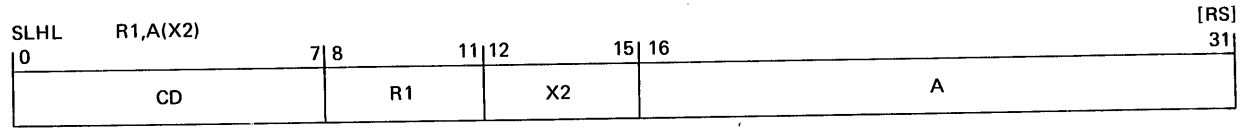
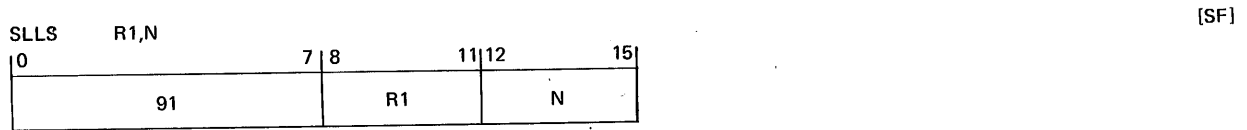
4.6 SHIFT/ROTATE INSTRUCTIONS

The Shift/Rotate instructions provide for arithmetic and logical manipulation of information contained in the General Registers. Bits shifted out of the high or low order end of a General Register are passed through the Carry Bit position of the Condition Code (PSW 12). After execution of a Shift instruction, the last bit which was shifted out is contained in the Carry position. The double-precision Shift and Rotate instructions manipulate a pair of General Registers. The R1 field of these instructions must specify an even numbered register. The register specified contains the most significant 16 bits of the fullword operand. The next sequential General Register contains the least significant 16-bits.

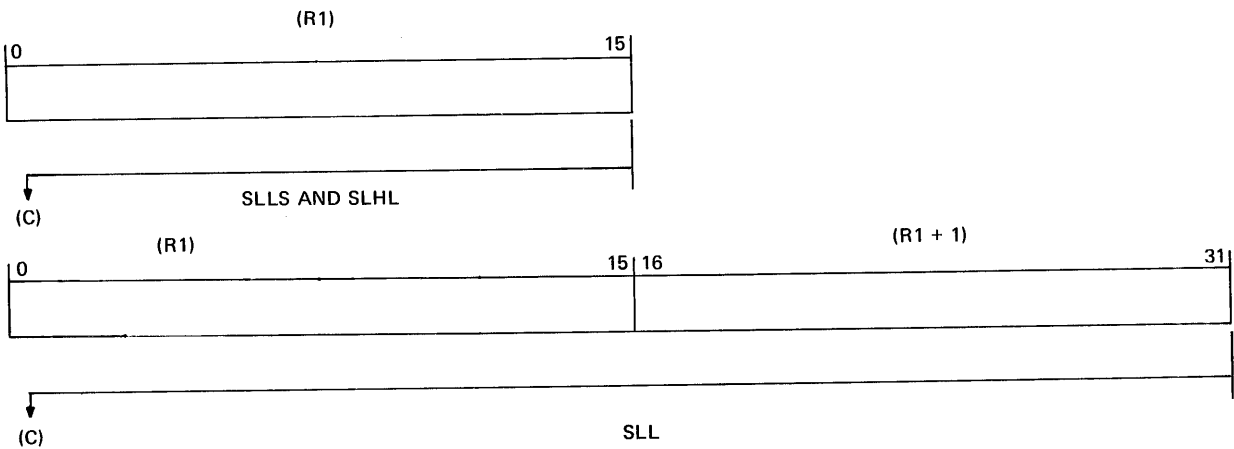
A shift of zero positions causes the Condition Code to be set properly with no alteration to the information contained in the General Register. The instructions described in this section are:

- | | | |
|-------|------|---------------------------------|
| 4.6.1 | SLLS | Shift Left Logical Short |
| | SLHL | Shift Left Halfword Logical |
| | SLL | Shift Left Logical |
| 4.6.2 | SRLS | Shift Right Logical Short |
| | SRHL | Shift Right Halfword Logical |
| | SRL | Shift Right Logical |
| 4.6.3 | RLL | Rotate Left Logical |
| 4.6.4 | RRL | Rotate Right Logical |
| 4.6.5 | SLHA | Shift Left Halfword Arithmetic |
| | SLA | Shift Left Arithmetic |
| 4.6.6 | SRHA | Shift Right Halfword Arithmetic |
| | SRA | Shift Right Arithmetic |

4.6.1 Shift Left Logical



The content of the first operand is shifted left the number of positions specified by the second operand. High order bits shifted out of Position 0 are shifted through the Carry Bit of the PSW and then lost. Zeros are shifted into the low order bit position.



Resulting Condition Code:

	12	13	14	15	
	C	V	G	L	
			0	0	RESULT IS ZERO.
			0	1	RESULT IS LESS THAN ZERO.
			1	0	RESULT IS GREATER THAN ZERO.
0					LAST BIT THAT WAS SHIFTED OUT WAS A ZERO.
1					LAST BIT THAT WAS SHIFTED OUT WAS A ONE.

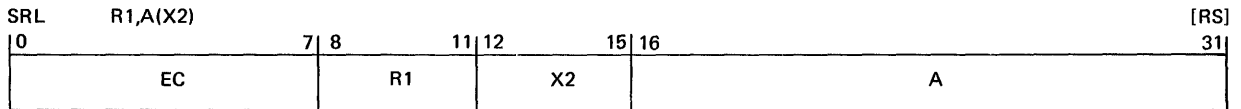
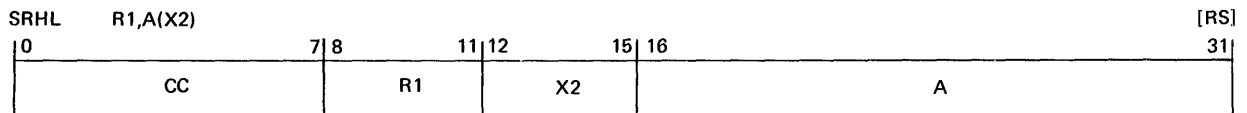
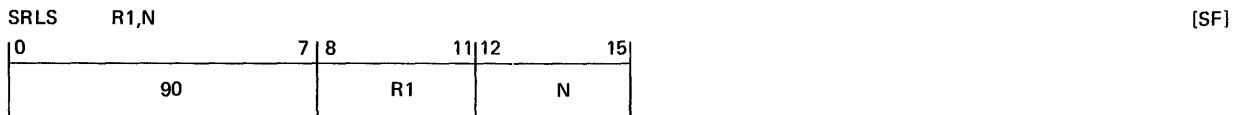
Programming Note:

For the Shift Left Logical Short (SLLS) instruction, the N field (Bits 12 through 15) of the instruction specify the number of positions the content of R1 is to be shifted.

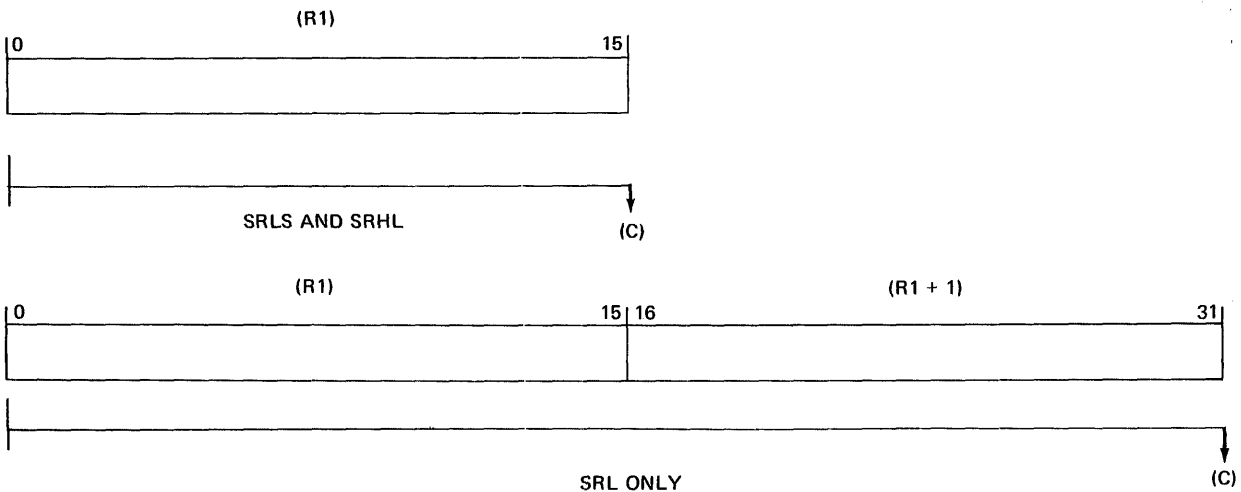
For the Shift Left Halfword Logical (SLHL) instruction, only the low order four-bits (12 through 15) of A + (X2) are used for the shift count.

The Shift Left Logical (SLL) instruction, shifts Registers R1 and R1 + 1, an even-odd pair. The R1 field of the instruction must specify an even register. The shift count is specified by the low order five-bits (11 through 15) of the value A + (X2). The Carry is formed by the output of R1.

4.6.2 Shift Right Logical



The content of the first operand is shifted right the number of bit positions specified by the second operand. Low order bits shifted out of Position 15 are shifted thru the Carry Bit of the PSW and then lost. Zeros are shifted into Position 0.



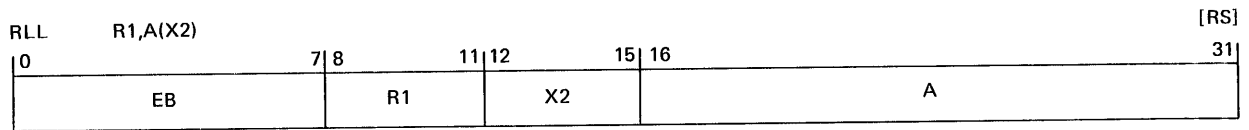
Resulting Condition Code:

	12	13	14	15	
C					
V					
G			0	0	RESULT IS ZERO.
L			0	1	RESULT IS LESS THAN ZERO.
			1	0	RESULT IS GREATER THAN ZERO.
0					LAST BIT THAT WAS SHIFTED OUT WAS A ZERO.
1					LAST BIT THAT WAS SHIFTED OUT WAS A ONE.

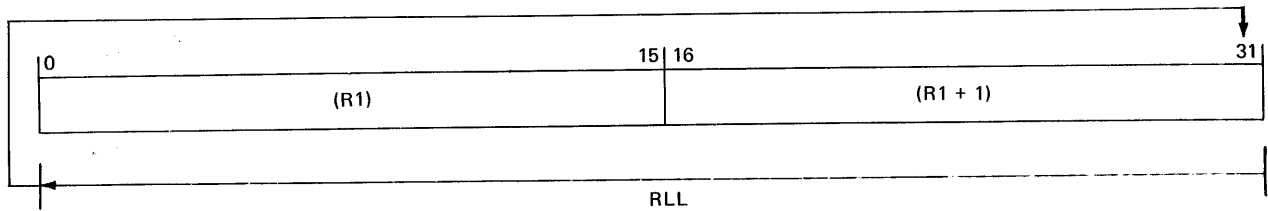
Programming Note:

See Shift Left Logical 4.7.1.

4.6.3 Rotate Left Logical



The 32-bit first operand specified by R1 is shifted left, end around, the number of positions specified by the low order five bits of the value A + (X2). All 32-bits of the fullword are shifted. Bits shifted out of Position 0 are shifted into Position 31. A shift specification of 16-bits interchanges the two halves (R1, R1 + 1) of the first operand.



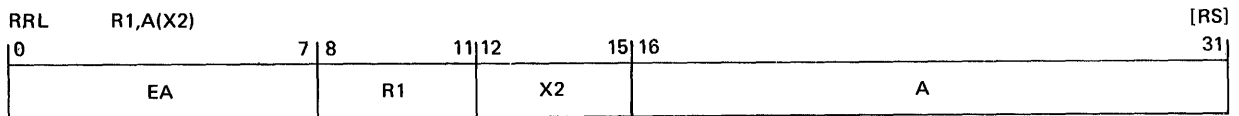
Resulting Condition Code:

12	13	14	15	
C	V	G	L	
		0	0	RESULT IS ZERO.
		1	0	RESULT IS GREATER THAN ZERO.
		0	1	RESULT IS LESS THAN ZERO.

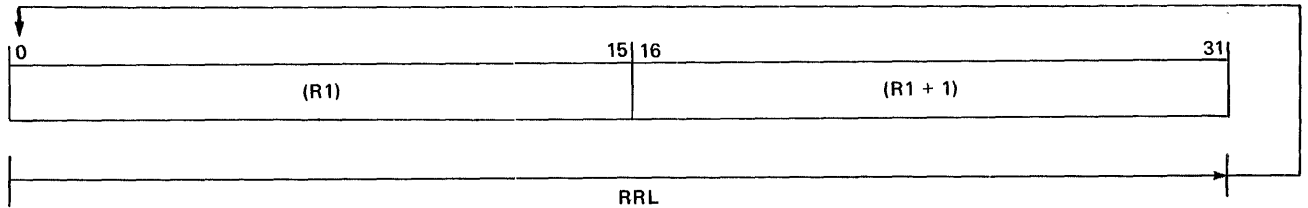
Programming Note:

The Rotate Left Logical (RLL) instruction, rotates Registers R1 and R1 + 1, an even-odd pair. The R1 field of the instruction must specify an even register.

4.6.4 Rotate Right Logical



The 32-bit first operand specified by R1 is shifted right, end around, the number of positions specified by the low order five bits of the value A + (X2). All 32-bits of the fullword are shifted. Bits shifted out of Position 31 are shifted into Position 0. A shift specification of 16-bits interchanges the two halves (R1, R1 + 1) of the first operand.



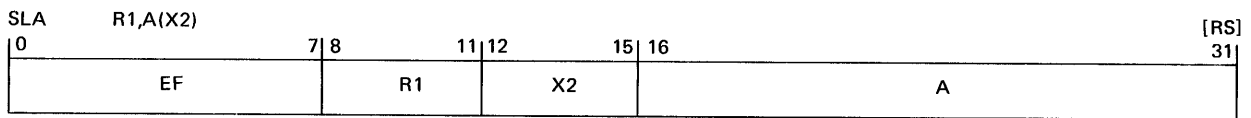
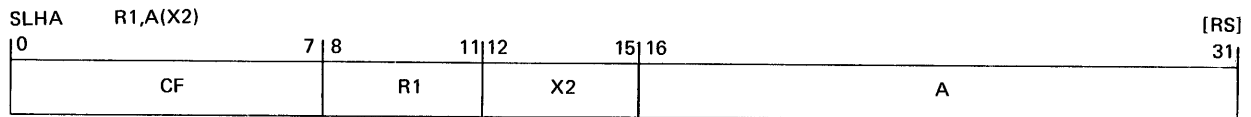
Resulting Condition Code:

12	13	14	15	
C	V	G	L	
		0	0	RESULT IS ZERO.
		1	0	RESULT IS GREATER THAN ZERO.
		0	1	RESULT IS LESS THAN ZERO.

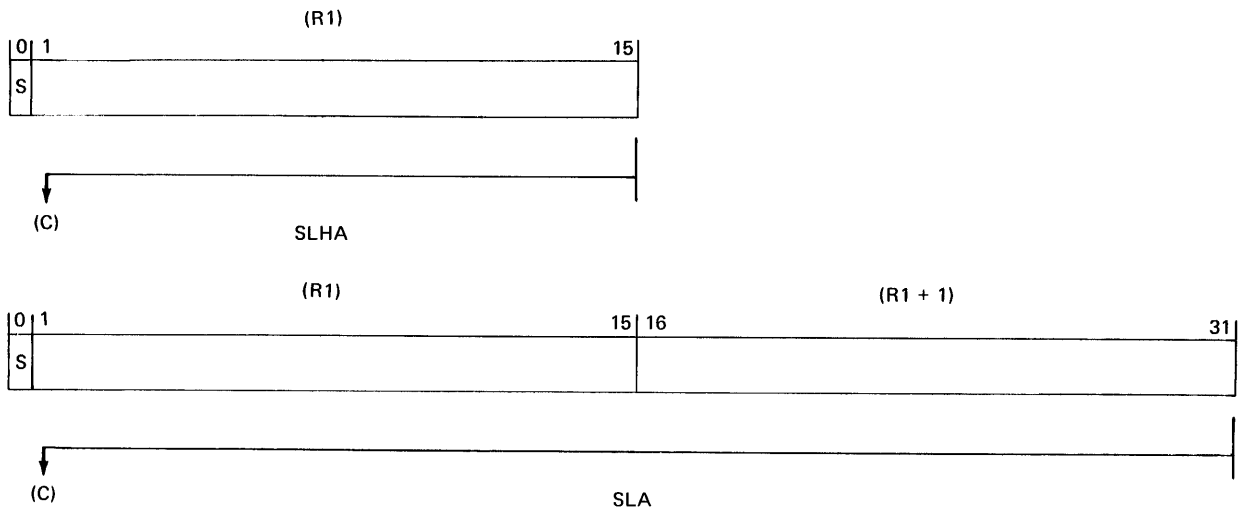
Programming Note:

The Rotate Right Logical (RRL) instruction, rotates Registers R1 and R1 + 1, an even-odd pair. The R1 field of the instruction must specify an even register.

4.6.5 Shift Left Arithmetic



The content of the first operand is shifted left the number of bit positions specified by the second operand. The Sign Bit is unchanged. High order bits shifted out of Position 1 are shifted through the Carry Bit of the PSW and then lost. Zeros are shifted into the low order bit position.



Resulting Condition Code:

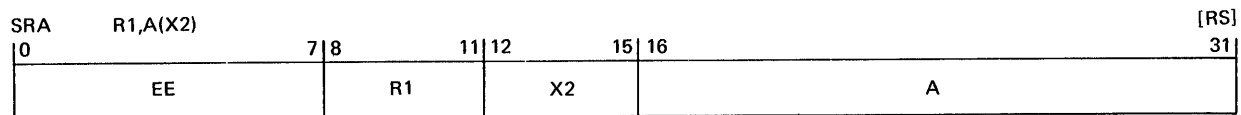
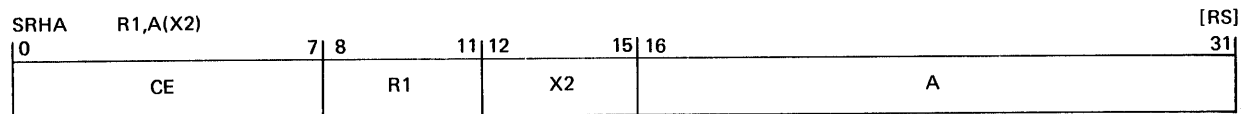
	12	13	14	15	
C		V	G	L	
			0	0	RESULT IS ZERO.
			0	1	RESULT IS LESS THAN ZERO.
			1	0	RESULT IS GREATER THAN ZERO.
0					LAST BIT THAT WAS SHIFTED OUT WAS A ZERO.
1					LAST BIT THAT WAS SHIFTED OUT WAS A ONE.

Programming Note:

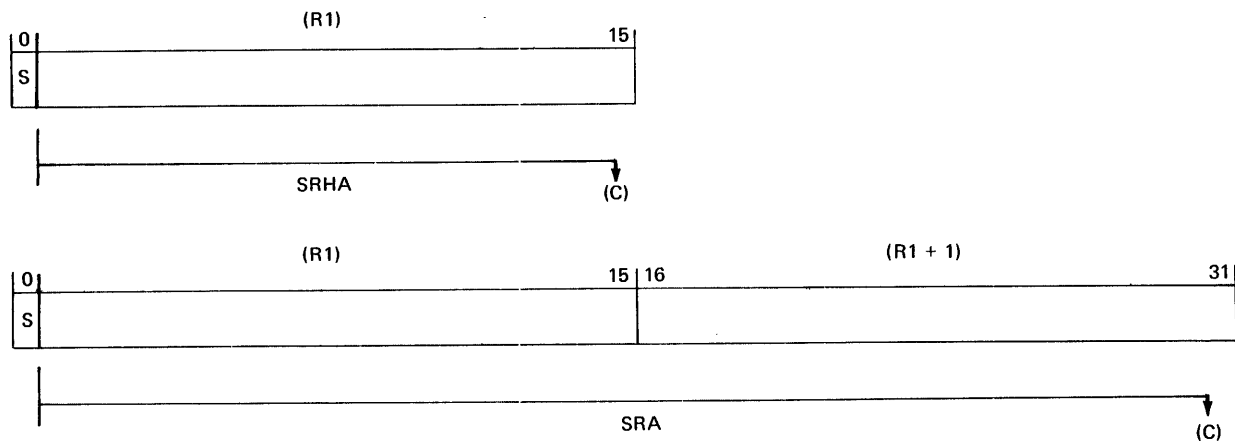
For the Shift Left Halfword Arithmetic (SLHA) instruction, the shift count is specified by the low order four-bits (12 through 15) of the value of A + (X2).

The Shift Left Arithmetic (SLA) instruction, shifts Registers R1 and R1 + 1, an even-odd pair. R1 must specify an even register. The shift count is specified by the low order five-bits (11 through 15) of the value of A + (X2).

4.6.6 Shift Right Arithmetic



The content of the first operand is shifted right the number of bit positions specified by the second operand. The Sign Bit, Bit 0, of R1 is unchanged and is shifted right into Bit 1; therefore, Bit 0, is propagated right as many positions as specified by the second operand. Low order bits of the first operand are shifted through the Carry Bit of the PSW and then lost.



Resulting Condition Code:

12	13	14	15	
C	V	G	L	
		0	0	RESULT IS ZERO.
		0	1	RESULT IS LESS THAN ZERO.
		1	0	RESULT IS GREATER THAN ZERO.
0				LAST BIT THAT WAS SHIFTED OUT WAS A ZERO.
1				LAST BIT THAT WAS SHIFTED OUT WAS A ONE.

Programming Note:

For the Shift Right Halfword Arithmetic (SRHA) instruction, the shift count is specified by the low order four-bits (12 through 15) of the value of A + (X2).

The Shift Right Arithmetic (SRA) instruction, shifts Registers R1 and R1 + 1, an even odd pair. R1 must specify an even register. The shift count is specified by the low order five-bits (11 through 15) of the value of A + (X2). The Carry is formed by the output of R1 + 1 instead of R1.

4.7 BRANCH INSTRUCTIONS

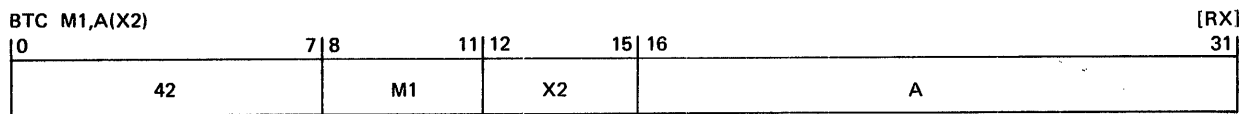
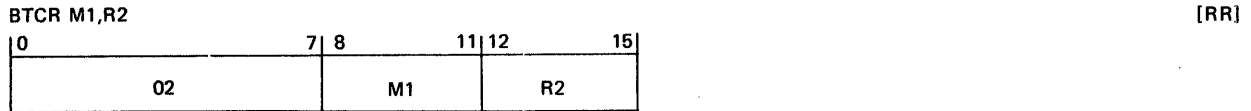
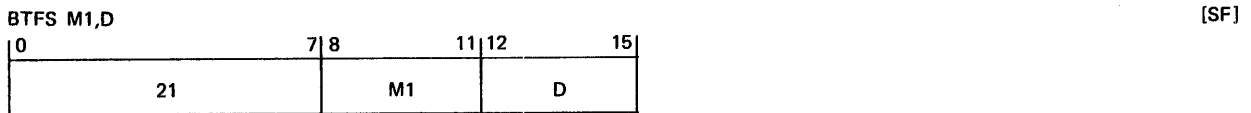
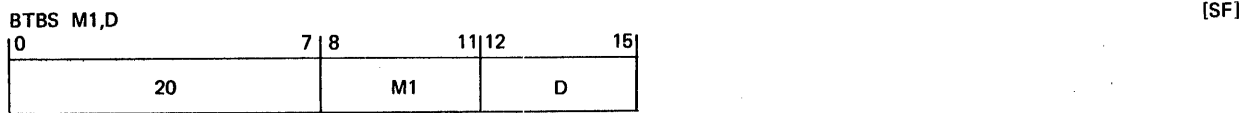
Branch instructions are programmed decisions providing entry to subprograms, as well as testing the result of arithmetic, logical, or indexing operations.

Many Processor operations result in setting of the Condition Code in the Program Status Word [PSW (12:15)]. The Branch on Condition instructions implement the testing of the Condition Code through use of a mask field contained in the instruction itself (M1 field).

The four-bit M1 field is not a register address, but rather an image of the Condition Code to be tested. The instructions described in this section are:

- 4.7.1 BTBS Branch on True Backward Short
 - BTFS Branch on True Forward Short
 - BTCL Branch on True Condition RR
 - BTC Branch on True Condition
- 4.7.2 BFBS Branch on False Backward Short
 - BFFS Branch on False Forward Short
 - BFCL Branch on False Condition RR
 - BFC Branch on False Condition
- 4.7.3 BXH Branch on Index High
 - BXLE Branch on Index Low or Equal
- 4.7.4 BALR Branch and Link RR
 - BAL Branch and Link

4.7.1 Branch on True Condition



The Condition Code field of the Program Status Word PSW (12:15) is tested for the condition specified by the Mask Field (M1). If any of the conditions tested are found to be true, a Branch is executed to the 16-bit address specified by the second operand. If none of the conditions tested are found to be true the next sequential instruction is executed.

Tested Condition True:

BTBS: $[PSW(16:31)] \longleftarrow [PSW(16:31)] - 2D$
 BTFS: $[PSW(16:31)] \longleftarrow [PSW(16:31)] + 2D$
 BTCR: $[PSW(16:31)] \longleftarrow (R2)$
 BTC: $[PSW(16:31)] \longleftarrow A + (X2)$

Tested Condition False:

BTBS: }
 BTFS: } $[PSW(16:31)] \longleftarrow [PSW(16:31)] + 2$
 BTCR: }
 BTC: $[PSW(16:31)] \longleftarrow [PSW(16:31)] + 4$

Programming Note:

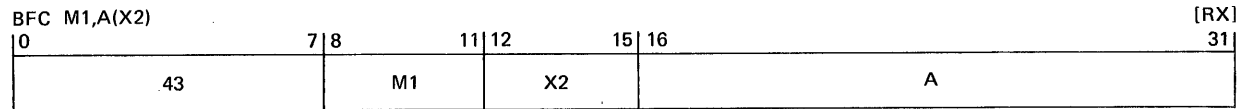
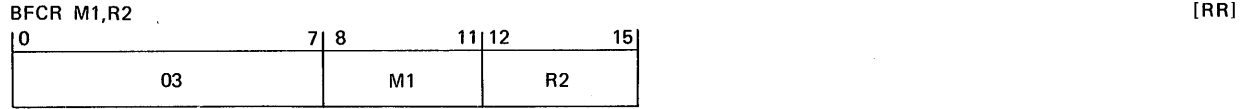
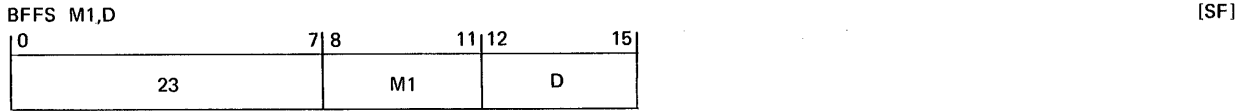
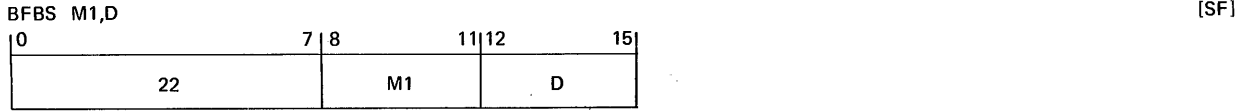
A logical AND is performed between each bit in the Condition Code and its corresponding bit in the M1 field. If any resultant bit is a one, the Branch will occur. The Condition Code $[PSW(12:15)]$ is not changed. For example, if the Condition Code is 1010 and the M1 field is 1000, the Branch occurs with Branch on True instructions.

The Branch on True Backward Short (BTBS) instruction, causes a Branch to an address relative to the present Location Counter when the tested condition is true. The displacement is specified by the D field (Bits 12 through 15) of the instruction. The D field (times two) is subtracted from the present Location Counter to generate the address of the next instruction.

The Branch on True Forward Short (BTFS) instruction, causes a Branch to an address relative to the present Location Counter when the tested condition is true. The displacement is specified by the D field (Bits 12 through 15) of the instruction. The D field (times two) is added to the present Location Counter to generate the address of the next instruction.

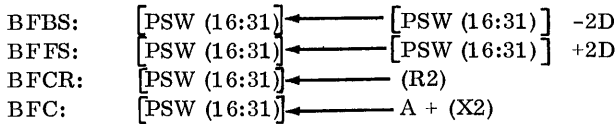
The Short Branch instructions (e. g. BTBS), are appropriate for Branches which specify small displacements from the present Location Counter, for example, in sense status loops used for program controlled I/O.

4.7.2 Branch on False Condition

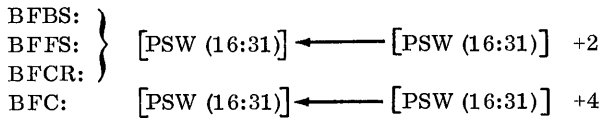


The Condition Code field of the Program Status Word [PSW (12:15)] is tested for the condition specified by the mask field (M1). If all conditions tested are found to be false, a Branch is executed to the 16-bit address specified by the second operand. If any of the conditions tested are found to be true the next sequential instruction is executed.

Tested Condition False



Tested Condition True



Programming Note:

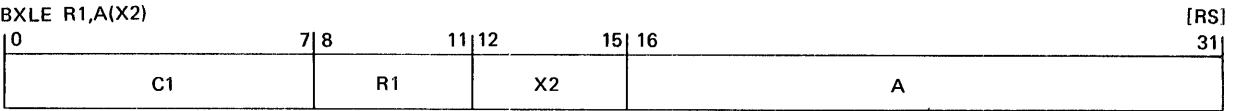
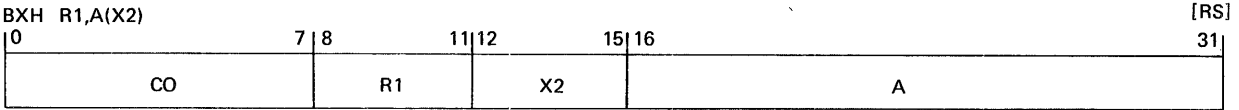
A logical AND is performed between each bit in the Condition Code and its corresponding bit in the M1 field. If any resultant bit is a one, the Branch will not occur. The Condition Code [PSW (12:15)] is not changed. For example, if the Condition Code is 1010 and the M1 field is 1100, the Branch does not occur with the Branch on False instruction.

The Branch on False Backward Short (BFBS) instruction, causes a Branch to an address relative to the present Location Counter when the tested condition is false. The displacement is specified by the D field (Bits 12 through 15) of the instruction. The D field (times two) is subtracted from the present Location Counter to generate the address of the next instruction.

The Branch on False Forward Short (BFFS) instruction, causes a Branch to an address relative to the present Location Counter when the tested condition is false. The displacement is specified by the D field (Bits 12 through 15) of the instruction. The D field (times two) is added to the present Location Counter to generate the address of the next instruction.

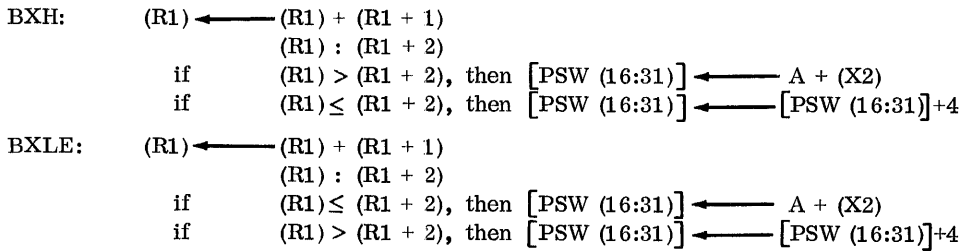
Branch on False Condition with a mask of 0 is an Unconditional Branch.

4.7.3 Branch on Index



Prior to execution of this instruction, the General Register specified by the first operand (R1) must contain a 16-bit starting index value, R1 + 1 must contain a 16-bit increment value, and R1 + 2 must contain a 16-bit comparand (limit or final value). All values may be signed.

Execution of this instruction causes the index (R1) to be incremented by (R1 + 1) and logically compared to the index limit, (R1 + 2).



Resulting Condition Code:

Unchanged.

Programming Note:

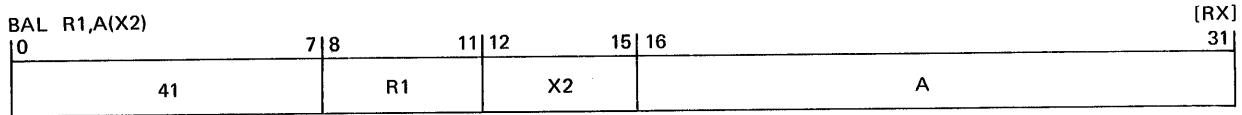
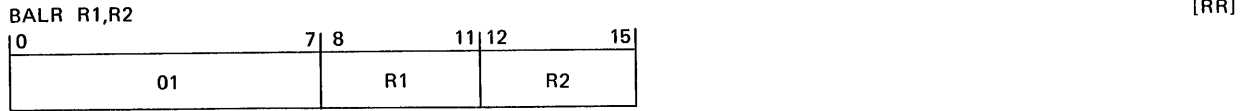
For the Branch on Index High (BXH) instruction, the contents of R1 + 1 should be negative. As long as the index (R1) is greater than the limit (R1 + 2), the 16-bit address specified by the second operand is transferred to the instruction address field of the Program Status Word [PSW (16:31)]. The next instruction executed will be accessed from the location specified by the new instruction address. When the count is not greater than the index limit, the instruction following Branch on Index High will be executed.

For the Branch on Index Low or Equal (BXLE) instruction, the contents of R1 + 1 should be positive. As long as the index (R1) is equal to or less than the limit (R1 + 2), the 16-bit address specified by the second operand is transferred to the instruction address field of the Program Status Word [PSW (16:31)]. The next instruction executed will be accessed from the location specified by the new instruction address. When the count is greater than the limit, the instruction following Branch on Index Low will be executed.

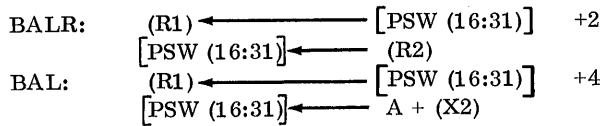
The Branch on Index High and Branch on Index Low instructions are appropriate for rapid loop control, particularly when one or more of the instructions in the loop is indexed.

General Register 13 is the maximum specification for the R1 field.

4.7.4 Branch and Link



The address of the next sequential instruction is saved in the General Register specified by the first operand (R1), and an Unconditional Branch is executed to the 16-bit address specified by the second operand. In all INTERDATA Processors except the Model 3, the effective second operand is derived before the contents of register R1 are changed. See note below.



Condition Code:

Unchanged.

Programming Note:

The Branch and Link instruction may be used for entry to sub-programs. It differs from the Branch Unconditional instruction in that the incremented Location Counter value is preserved in a specified General Register to be used as the sub-program exit address. Exit from the sub-program is effected by a Branch Unconditional instruction through the General Register in which the exit address has been maintained. Note that in the Model 3, if the same register is specified in both the first and second operands of the BALR instruction (R1 = R2), R1 will be loaded with the saved address of the next instruction before it is used to derive the second operand. In all other INTERDATA Processors, the effective second operand is derived before the contents of R1 are changed.

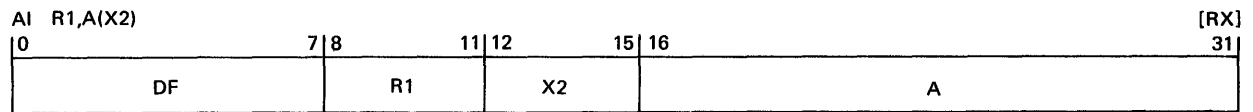
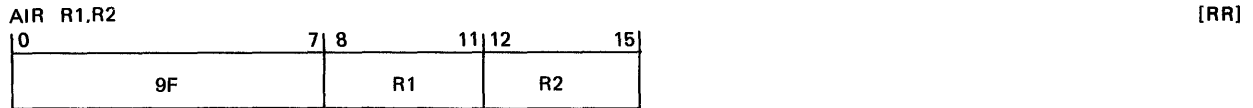
4.8 INPUT/OUTPUT INSTRUCTIONS

The I/O instructions provide for the transfer of data between the Processor and the peripheral devices on the Multiplexor Bus. All of the instructions described in this section are privileged and, if executed with the Processor in Protect Mode (PSW Bit 7 set), result in an Illegal Instruction Interrupt.

Following most I/O instructions, the V flag in the Condition Code indicates an instruction time-out. That is, due to an improper device response - either the addressed device does not exist, or it did not respond correctly - the specified I/O operation was not performed. Following Sense Status or Acknowledge Interrupt instructions, the Condition Code (CVGL) also reflects Bits 4 through 7 of the device status. With standard INTERDATA device controllers, Bit 5 of the status byte, which is reflected in the V flag in the Condition Code, is defined as Examine Status. This means that status byte should be examined. Following Sense Status and Acknowledge Interrupt instructions, therefore, the occurrence of the V flag with status Bits 0 through 3 equal zero indicates instruction time-out. For a complete definition of the bits in either command bytes, or status bytes, refer to documentation on the device in question. The instructions described in this section are:

4.8.1	AIR	Acknowledge Interrupt RR			
	AI	Acknowledge Interrupt			
4.8.2	SSR	Sense Status RR			
	SS	Sense Status			
4.8.3	OCR	Output Command RR	4.8.6	RHR	Read Halfword RR
	OC	Output Command		RH	Read Halfword
4.8.4	RDR	Read Data RR	4.8.7	WHR	Write Halfword RR
	RD	Read Data		WH	Write Halfword
4.8.5	WDR	Write Data RR	4.8.8	AL	Autoload
	WD	Write Data			

4.8.1 Acknowledge Interrupt



The address of the interrupting device replaces the content of the 16-bit General Register specified by the first operand (R1). The eight-bit device status byte replaces the content of the location specified by the second operand. The Condition Code is set equal to the right-most four bits in the device status byte. The device interrupt condition is then cleared.

- AIR: [R1 (8:15)] ← Device address
[R1 (0:7)] ← Zero
[R2 (8:15)] ← Status byte
[R2 (0:7)] ← Zero
[PSW (12:15)] ← Status byte (4:7)
- AI: [R1 (8:15)] ← Device number
[R1 (0:7)] ← Zero
[A + (X2)] ← Status byte
[PSW (12:15)] ← Status byte (4:7)

Resulting Condition Code:

12	13	14	15
C	V	G	L
1	1	1	1

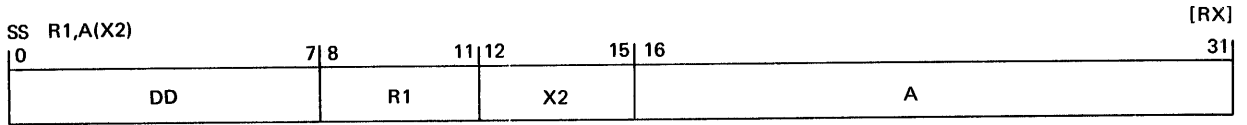
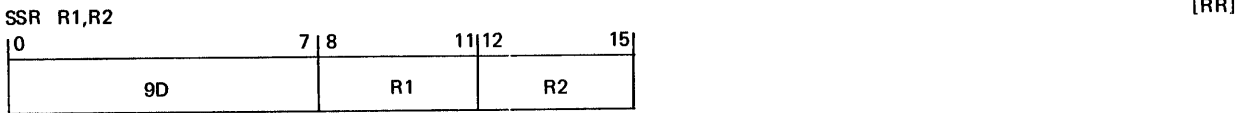
DEVICE BUSY (BSY)
EXAMINE STATUS (EX) OR TIME OUT
END OF MEDIUM (EOM)
DEVICE UNAVAILABLE (DU)

Programming Note:

These instructions are privileged.

The RX form (AI) is subject to Memory Protect.

4.8.2 Sense Status



The 16-bit General Register specified by the first operand (R1) contains the device address. The device is addressed and the eight-bit device status byte replaces the content of the location specified by the second operand. The Condition Code is set equal to the right-most four bits of the device status byte. The first operand is unchanged.

SSR: [R2 (8:15)] ← Status byte
 [R2 (0:7)] ← Zero
 [PSW (12:15)] ← Status byte (4:7)

SS: [A + (X2)] ← Status byte
 [PSW (12:15)] ← Status byte (4:7)

Resulting Condition Code:

12	13	14	15
C	V	G	L
1	1	1	1

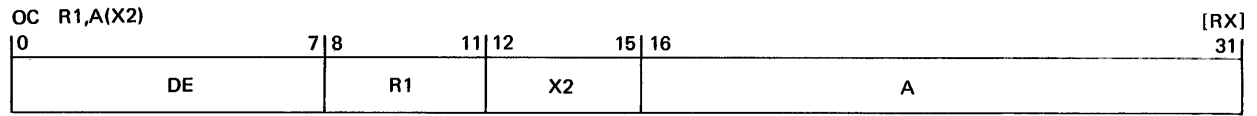
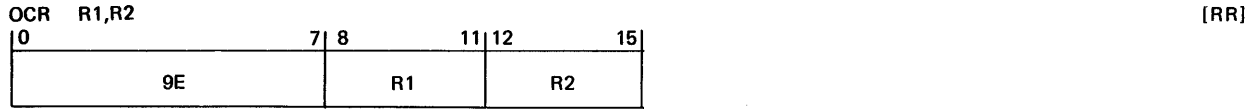
DEVICE BUSY (BSY)
 EXAMINE STATUS (EX) OR TIME OUT
 END OF MEDIUM (EOM)
 DEVICE UNAVAILABLE (DU)

Programming Note:

These instructions are privileged.

The RX form (SS) is subject to Memory Protect.

4.8.3 Output Command



The 16-bit General Register specified by the first operand (R1) contains the device address. The device is addressed and the eight-bit device command byte specified by the second operand is transmitted to the addressed device. Both operands remain unchanged.

OCR: Device ← [R2 (8:15)]
 OC: Device ← [A + (X2)]

Resulting Condition Code:

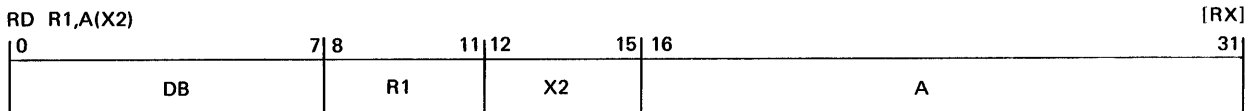
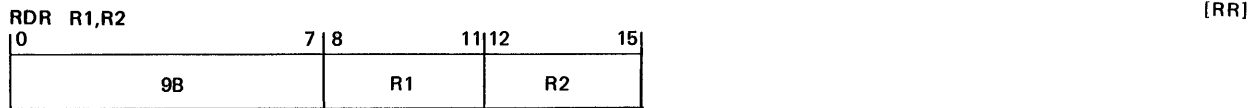
12	13	14	15
C	V	G	L
0	1	0	0

INSTRUCTION TIME OUT

Programming Note:

The Examine Status bit is set if the device cannot complete the command action. These instructions are privileged.

4.8.4 Read Data



The 16-bit General Register specified by the first operand (R1) contains the device address. The device is addressed and a single eight-bit data byte is transmitted from the device replacing the content of the location specified by the second operand.

RDR: [R2 (8:15)] ← Data byte
 [R2 (0:7)] ← Zero
 RD: [A + (X2)] ← Data byte

(Continued on next page)

Resulting Condition Code:

12	13	14	15
C	V	G	L
	1		

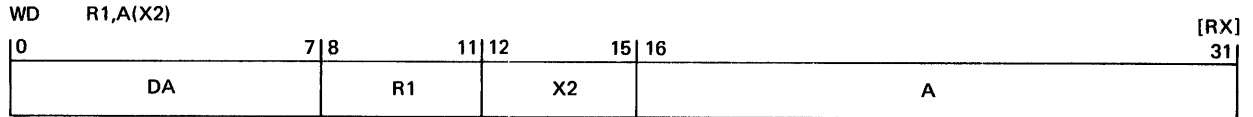
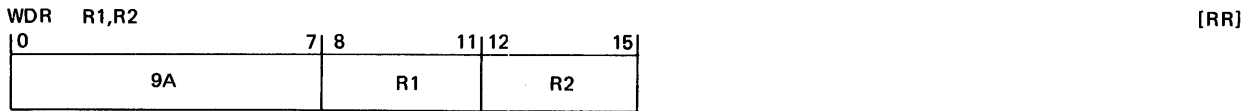
 INSTRUCTION TIME OUT

Programming Note:

These instructions are privileged.
The RX form (RD) is subject to Memory Protect.

These instructions should not be used with 16-bit oriented device controllers. Note that standard INTERDATA peripheral devices use 8-bit oriented device controllers. For 16-bit oriented devices, use Read Halfword/Write Halfword instructions.

4.8.5 Write Data



The 16-bit General Register specified by the first operand (R1) contains the device address. The device is addressed and a single eight-bit data byte is transmitted to the device. Both operands remain unchanged.

WDR: [R2 (8:15)] → (Device)

WD: [A + (X2)] → (Device)

Resulting Condition Code:

12	13	14	15
C	V	G	L
	1		

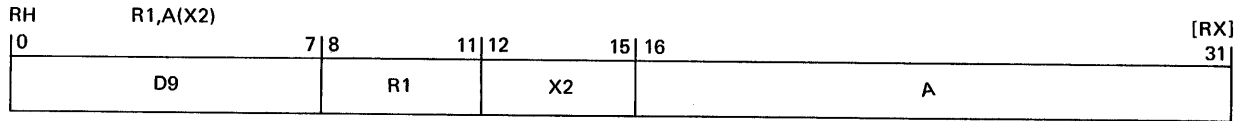
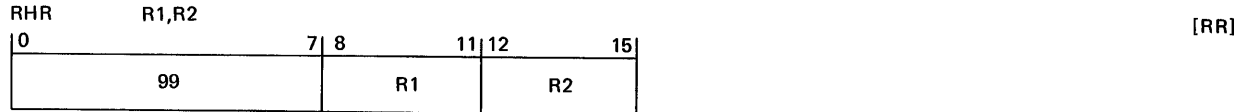
 INSTRUCTION TIME OUT

Programming Note:

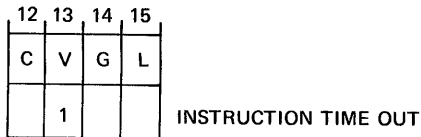
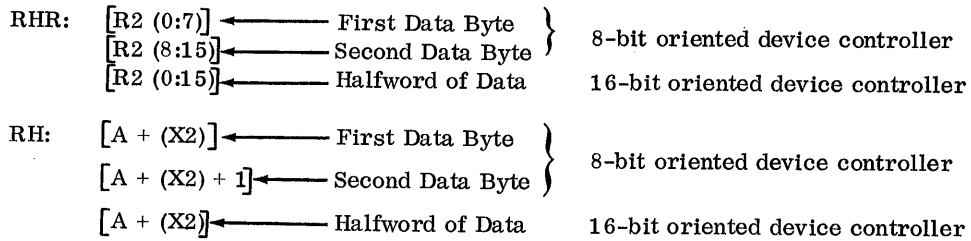
These instructions are privileged.

These instructions should not be used with 16-bit oriented device controllers. Note that standard INTERDATA peripheral devices use 8-bit oriented device controllers. For 16-bit oriented devices, use Read Halfword/Write Halfword instructions.

4.8.6 Read Halfword



The 16-bit General Register specified by R1 contains the device address. The device is addressed and a 16-bit halfword is received from the device replacing the contents of the second operand. The Read Halfword instruction is implemented such that it can work with both 8-bit byte oriented device controllers and with 16-bit halfword oriented device controllers. If the controller is byte oriented the Processor inputs two 8-bit bytes, if the controller is halfword oriented the Processor inputs one 16-bit halfword.

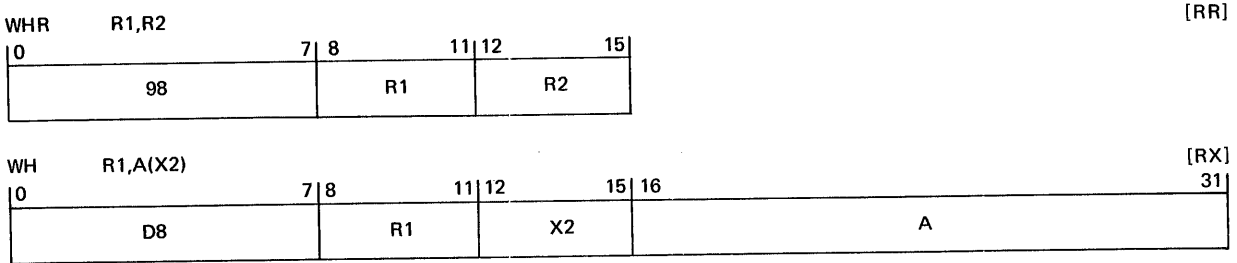


Programming Note:

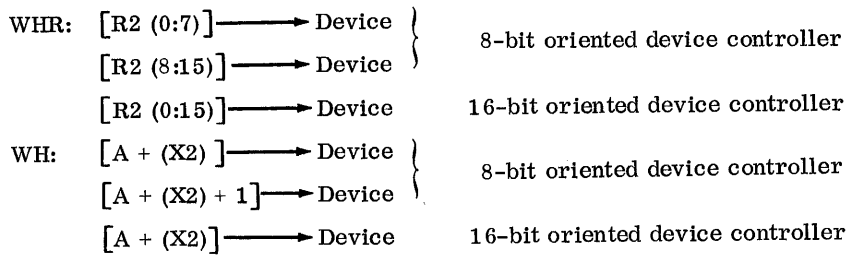
These instructions are privileged.
 The RX form (RH) is subject to Memory Protect.

With the RX form (RH), the effective address $A+(X2)$ must be an even value.

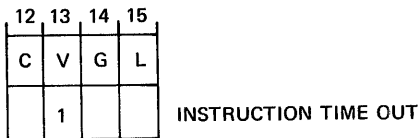
4.8.7 Write Halfword



The 16-bit General Register specified by R1 contains the device address. The device is addressed and a 16-bit halfword is transmitted to the device from the location specified by the second operand. The Write Halfword instruction is implemented such that it can work with both 8-bit byte oriented device controllers and with 16-bit halfword oriented device controllers. If the controller is byte oriented the Processor outputs two 8-bit bytes, if the controller is halfword oriented the Processor outputs one 16-bit halfword.



Resulting Condition Code:



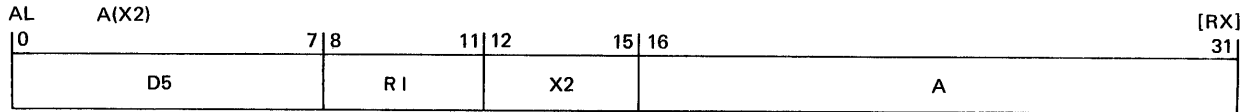
Programming Note:

The Read Halfword and Write Halfword instructions are useful with devices requiring two bytes per transfer. Since the transfer is accomplished with one instruction instead of two, both time and core are saved. Some examples of devices with which these instructions can be used are Halfword I/O Module, 16-line Interrupt Module, conversion equipment (i. e. D/A and A/D Converters), Card Reader, and Control Panel.

With the RX form (WH), the effective address $A+(X2)$ must be an even value.

These instructions are privileged.

4.8.8 Autoload



The Autoload instruction loads memory with a block of data from a byte oriented input device (e. g. Tele-type, photo-electric Paper Tape Reader, Magnetic Tape, etc.). The data is read a byte at a time and stored in successive memory locations starting with location X'80'. The last byte is loaded into the memory location specified by the address of the second operand, A + (X2). Any blank or zero bytes that are input prior to the first non zero byte are considered to be leader and are therefore ignored; all other zero bytes are stored as data. The input device is specified by memory location X'78'. The device command code is specified by memory location X'79'.

1. $n \leftarrow 0$
2. $(X'80' + n) \leftarrow \text{byte}$
3. $n \leftarrow n + 1$
4. If $A + (X2) < X'80' + n$, instruction is finished, otherwise return to equation 2.

Resulting Condition Code:

	12	13	14	15	
	C	V	G	L	
0	0	0	0	0	DATA TRANSFER COMPLETED CORRECTLY.
1					DEVICE BUSY (BSY)
		1			EXAMINE STATUS (EX) OR TIME OUT.
			1		END OF MEDIUM (EOM)
				1	DEVICE UNAVAILABLE (DU).

Programming Note:

This instruction is privileged.

This instruction is subject to Memory Protect.

The R1 field of an Autoload machine instruction must contain 0.

This instruction should not be used with 16-bit oriented device controllers. Note that standard INTERDATA peripheral devices use 8-bit oriented device controllers. For 16-bit oriented devices, use Read Halfword/Write Halfword instructions.

4.9 BLOCK INPUT/OUTPUT INSTRUCTIONS

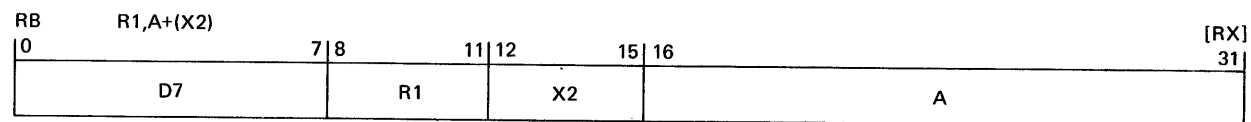
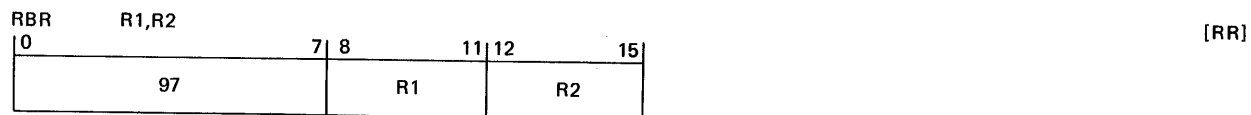
The Block I/O instructions provide for the transfer of blocks of data between the I/O device and memory. These instructions are privileged and, if executed in the Protect Mode (PSW Bit 7 set), result in an Illegal Instruction Interrupt.

Following most I/O instructions, the V flag in the Condition Code indicates an instruction time-out. That is, due to an improper device response - either the addressed device does not exist, or does not respond correctly - the specified I/O operation was not performed. With standard INTERDATA device controllers, Bit 5 of the status byte, which is reflected in the V flag in the Condition Code, is defined as Examine Status. This means that the status byte should be examined. Following Sense Status and Acknowledge Interrupt instructions, therefore, the occurrence of the V flag with status Bits 0 through 3 equal to zero indicates instruction time-out. For a complete definition of the bits in either command bytes or status bytes, refer to the documentation on the device in question. The instructions described in this section are:

4.9.1 RBR Read Block RR
RB Read Block

4.9.2 WBR Write Block RR
WB Write Block

4.9.1 Read Block



The 16-bit General Register specified by the first operand (R1) contains the device address. The 16-bit second operand location, (R2) or $[A + (X2)]$ contains the starting address of the data buffer to be transferred. The next sequential halfword, (R2 + 1) or $[A + (X2) + 2]$ contains the ending address of the data buffer. The starting address must be equal to, or less than, the ending address. Data transfer is inclusive of the buffer limits. If the starting address is greater than the ending address, no transfer takes place and the instruction terminates with the Condition Code equal to zero.

The Read Block instruction causes transfer of eight-bit data bytes from a device to consecutive memory locations. No other instructions are executed during transfer of the data block.

The Condition Code portion of the Program Status Word [PSW (12:15)] will be set to zero after a normal transfer. In the event of an abnormal block data transfer, the Condition Code will not be zero.

Resulting Condition Code:

12	13	14	15	
C	V	G	L	
0	0	0	0	BLOCK DATA TRANSFER COMPLETE CORRECTLY.
1				DEVICE BUSY (BSY)
	1			EXAMINE STATUS (EX) OR TIME OUT
		1		END OF MEDIUM (EOM)
			1	DEVICE UNAVAILABLE (DU)

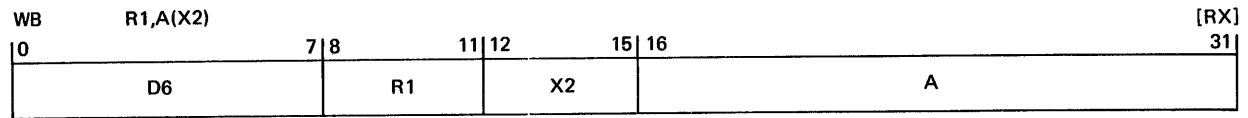
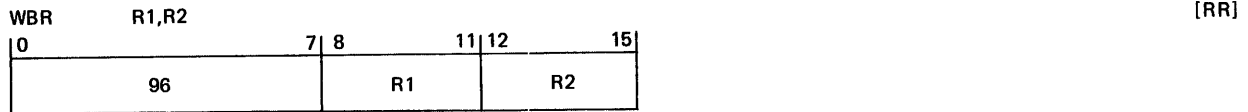
Programming Note:

These instructions are privileged.
 These instructions are subject to Memory Protect.

These instructions should not be used with 16-bit oriented device controllers. Note that standard INTERDATA peripheral devices use 8-bit oriented device controllers. For 16-bit oriented devices, use Read Halfword/Write Halfword instructions.

For RBR, General Register 14 is the maximum specification for the R2 field.

4.9.2 Write Block



The 16-bit General Register specified by the first operand (R1) contains the device address. The 16-bit second operand location, (R2) or $[A + (X2)]$ contains the starting address of the data buffer to be transferred. The next sequential halfword, $(R2 + 1)$ or $[A + (X2) + 2]$ contains the ending address of the data buffer. The starting address must be equal to, or less than, the ending address. Data transfer is inclusive of the buffer limits. If the starting address is greater than the ending address, no transfer takes place and the instruction terminates with the Condition Code equal to zero.

The Write Block instruction causes transfer of eight-bit data bytes from consecutive memory locations to a device. No other instructions are executed during transfer of the data block. The Condition Code portion of the Program Status Word [PSW (12:15)] will be set to zero after a normal transfer. In the event of an abnormal block data transfer, the Condition Code will not be zero.

Resulting Condition Code:

12	13	14	15	
C	V	G	L	
0	0	0	0	BLOCK DATA TRANSFER COMPLETED CORRECTLY.
1				DEVICE BUSY (BSY)
	1			EXAMINE STATUS (EX) OR TIME OUT.
		1		END OF MEDIUM (EOM).
			1	DEVICE UNAVAILABLE (DU).

Programming Note:

These instructions are privileged.

This instruction should not be used with 16-bit oriented device controllers. Note that standard INTERDATA peripheral devices use 8-bit oriented device controllers. For 16-bit oriented devices, use Read Halfword/Write Halfword instructions.

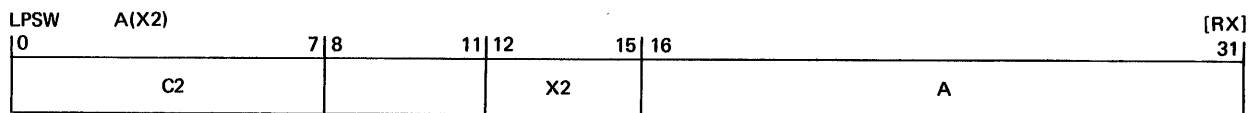
For WBR, General Register 14 is the maximum specification for the R2 field.

4.10 SYSTEM CONTROL INSTRUCTIONS

The System Control instructions provide a means for the program to set the Program Status Word, swap PSW's, trigger special interrupt handling, and communicate with a supervisor program. Some of these instructions are privileged and may be executed only with the Processor in the Supervisor Mode (i. e. Bit 7 of the PSW reset). Any attempt to execute these instructions in the Protect Mode results in an Illegal Instruction Interrupt. The instructions described in this section are:

- 4.10.1 LPSW Load Program Status Word
- 4.10.2 EPSR Exchange Program Status
- 4.10.3 SINT Simulate Interrupt
- 4.10.4 SVC Supervisor Call

4.10.1 Load Program Status Word



A 32-bit operand is loaded into the Current Program Status Word. The second operand is unchanged.

$$[\text{PSW (0:31)}] \leftarrow [A + (X2)]$$

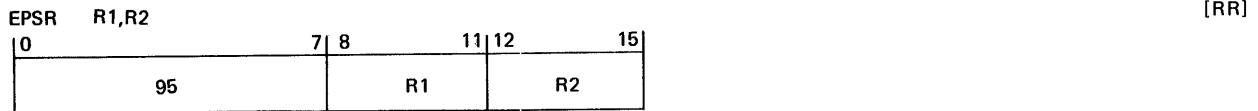
Resulting Condition Code:

Determined by PSW loaded by the instruction.

Programming Note:

This instruction is privileged.
The R1 field of a Load PSW instruction must contain 0.

4.10.2 Exchange Program Status



The Current Program Status, PSW (0:15), is stored into the register specified by R1. The content of R2 then becomes the Current Program Status, [PSW (0:15)]. Note that if R1 = R2, this results in the Program Status being copied into R1, but otherwise remaining unchanged. This instruction is useful for capturing the running Program Status, enabling or disabling interrupts, or loading the Condition Code with a specified value.

EPSR: [PSW (0:15)] → R1
 [PSW (0:15)] ← R2

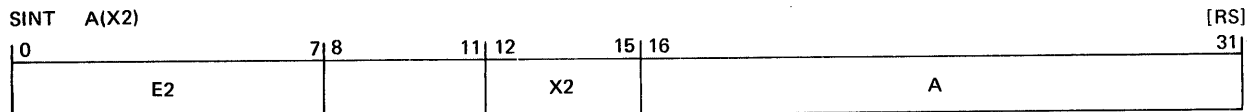
Resulting Condition Code:

Determined by New PSW.

Programming Note:

This instruction is privileged.

4.10.3 Simulate Interrupt



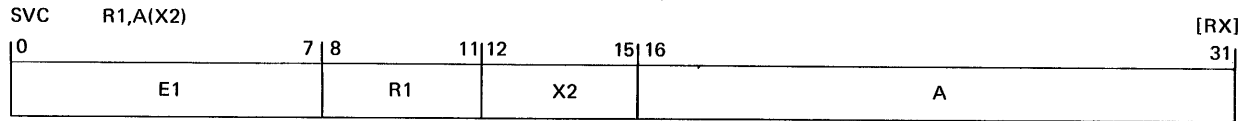
The least significant eight-bits of the second operand, A + (X2), is presented to the Interrupt Handler as a device number. The device number indexes into the Service Pointer Table at X'00D0' and results in either an Immediate Interrupt or an I/O Channel operation.

Programming Note:

This instruction is privileged.

The R1 field of a Simulate Interrupt instruction must contain 0.

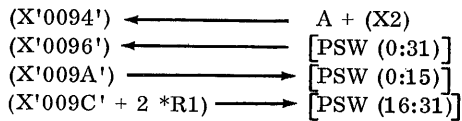
4.10.4 Supervisor Call



The Supervisor Call instruction is used to initiate certain functions in the Supervisor program. The second operand address, $A + (X2)$, may be a pointer to the core location of the parameters the Supervisor program will need to complete the function specified.

The value, $A + (X2)$, is stored in core location X'0094'. The Current Program Status Word is stored in the fullword core location at X'0096'. Core location X'009A' contains the New Program Status value. Core locations X'009C' through X'00BB' contain sixteen new Location Counter values, one for each type of Supervisor call.

The type of Supervisor call is specified in the R1 field of the instruction. Sixteen different calls are provided for. Return from the Supervisor is made by executing a Load Program Status Word instruction specifying the stored "Old" PSW in location X'0096'.



Resulting Condition Code:

Defined by New PSW.

Programming Note:

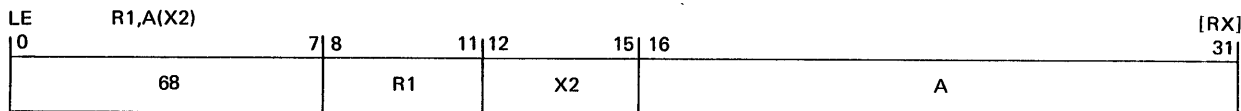
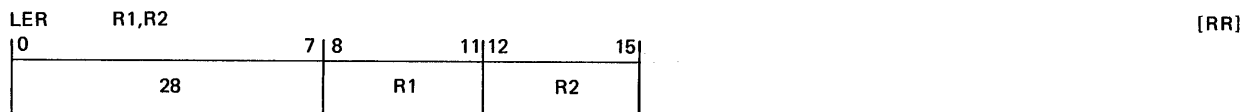
This instruction provides a convenient means of switching from the Protect Mode to the Supervisor Mode. Return to the Protect Mode is accomplished by a Load Program Status Word or Exchange Program Status instruction.

4.11 FLOATING-POINT INSTRUCTIONS (Models 70 and 80 Only)

The Floating-Point instructions provide for loading, storing, adding, subtracting, multiplying, dividing, and comparing of floating-point operands. In order to produce correct normalized results, the Arithmetic instructions require normalized floating-point operands. If the operands are not normalized (with the exception of the floating-point load instructions), the results of the instructions are undefined. The Floating-Point Load instruction normalizes an un-normalized floating-point number. The data format for the Floating-Point instructions is identical to that of the 360 single-precision floating-point number (see 3.2.3). The R1 and R2 fields of the Floating-Point instructions must specify even Floating-Point Registers (0, 2, 4, 6, etc.). Note that the Floating-Point Registers are reserved core memory locations. Quantities in Floating-Point Registers can be manipulated only with Floating-Point instructions. The instructions described in this section are:

4.11.1	LER	Floating-Point Load RR
	LE	Floating-Point Load
4.11.2	STE	Floating-Point Store
4.11.3	AER	Floating-Point Add RR
	AE	Floating-Point Add
4.11.4	SER	Floating-Point Subtract RR
	SE	Floating-Point Subtract
4.11.5	CER	Floating-Point Compare RR
	CE	Floating-Point Compare
4.11.6	MER	Floating-Point Multiply RR
	ME	Floating-Point Multiply
4.11.7	DER	Floating-Point Divide RR
	DE	Floating-Point Divide

4.11.1 Floating-Point Load



The floating-point second operand is normalized and placed in the Floating-Point Register specified as the first operand. During normalization, the fraction is shifted left hexadecimally (four-bits at a time) until the most significant hexadecimal digit is not zero. The exponent is decremented by one for each hexadecimal shift required. Zeros are shifted into the least significant hexadecimal digit of the fraction. The second operand is unchanged.

If the normalization causes exponent underflow, the entire floating-point result is set to zero, and the Overflow (V) flag is set.

LER: (R1) ← (R2)
 LE: (R1) ← [A + (X2)]

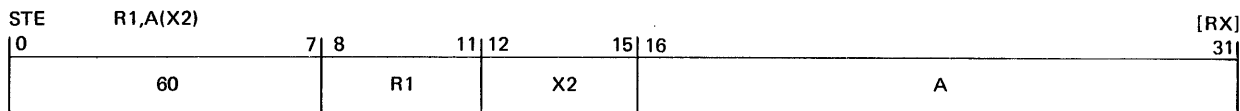
Resulting Condition Code:

12	13	14	15	
C	V	G	L	
		0	0	ZERO.
		0	1	LESS THAN ZERO.
		1	0	GREATER THAN ZERO.
	1	0	0	EXPONENT UNDERFLOW.

Programming Note:

In the event of underflow, the Floating-Point Arithmetic Fault Interrupt is caused, if enabled by Bit 5 of the PSW.

4.11.2 Floating-Point Store



The floating-point first operand is placed in the core memory location specified by A + (X2). The first operand is unchanged.

STE: (R1) → [A + (X2)]

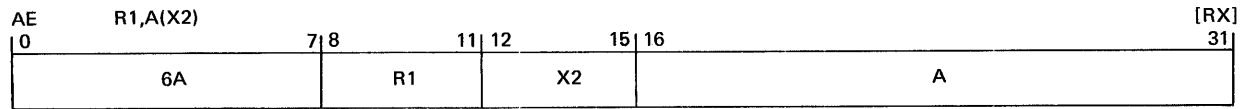
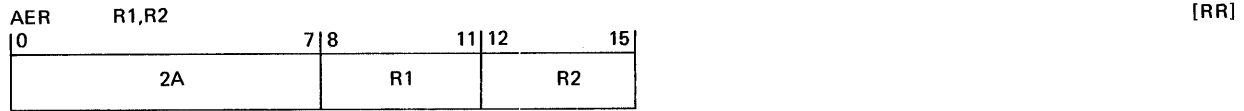
Resulting Condition Code:

Unchanged.

Programming Note:

This instruction is subject to Memory Protect.

4.11.3 Floating-Point Add



The exponents of the two operands are compared. If the exponents differ, the fraction with the smaller exponent is shifted right hexadecimally (four-bits at a time), and its exponent is incremented by one for each hexadecimal shift until the two exponents agree. The fractions are then added algebraically. If a Carry results, the exponent of the sum is incremented by one and the fraction (result) is shifted right one hexadecimal position (four-bits). The Carry is shifted back into the most significant hexadecimal digit of the fraction. If an exponent overflow results, the exponent and fraction of the result are set to all ones, and the Overflow flag is set. The sign of the result is not affected by the overflow.

If no Carry results from the addition of fractions, the sum is normalized. During normalization, the fraction is shifted left hexadecimally (four-bits at a time) until the most significant hexadecimal digit is not zero. The exponent is decremented by one for each hexadecimal shift required. Zeros are shifted into the least significant hexadecimal digit of the fraction.

If the normalization causes exponent underflow; the sign, exponent, and fraction of the sum are set to zero, and the Overflow flag is set. If a zero sum is generated from adding two equal magnitudes with unlike signs, the entire floating-point result is zeroed.

$$\begin{aligned} \text{AER:} & \quad (R1) \leftarrow (R1) + (R2) \\ \text{AE:} & \quad (R1) \leftarrow (R1) + [A + (X2)] \end{aligned}$$

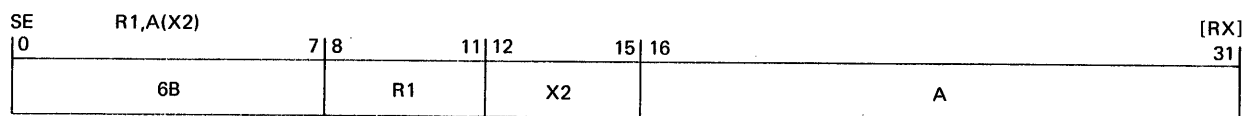
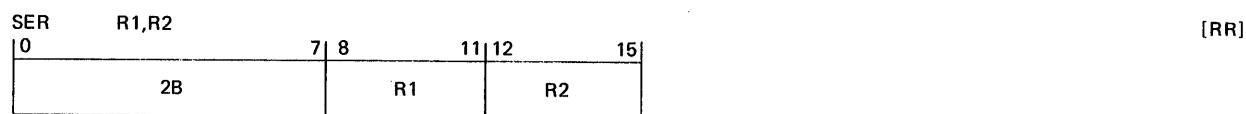
Resulting Condition Code:

	12	13	14	15	
C	V	G	L		
		0	0		SUM IS ZERO.
		0	1		SUM IS LESS THAN ZERO.
		1	0		SUM IS GREATER THAN ZERO.
	1	X	X		EXPONENT OVERFLOW.
	1	0	0		EXPONENT UNDERFLOW.

Programming Note:

In the event of overflow or underflow, the Floating-Point Arithmetic Fault Interrupt is caused, if enabled by Bit 5 of the PSW.

4.11.4 Floating-Point Subtract



The exponents of the two operands are compared. If the exponents differ, the fraction with the smaller exponent is shifted right hexadecimally (four-bits at a time), and its exponent is incremented by one for each hexadecimal shift until the two exponents agree. The fractions are then subtracted algebraically. If a Carry results, the exponent of the difference is incremented by one and the fraction (result) is shifted right one hexadecimal position (four-bits). The Carry is shifted into the most significant hexadecimal digit of the fraction. If an exponent overflow occurs, the exponent and fraction of the result are set to all ones, and the Overflow flag is set. The sign of the result is not affected by the overflow.

If no Carry results from the subtraction of fractions, the difference is normalized by shifting the fraction left hexadecimally (four-bits at a time) until the most significant hexadecimal digit is not zero. The exponent is decremented by one for each hexadecimal shift required. Zeros are shifted into the least significant hexadecimal digit of the fraction.

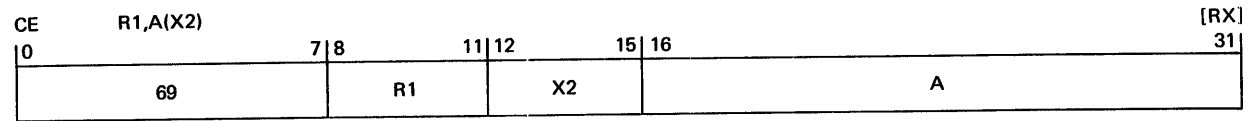
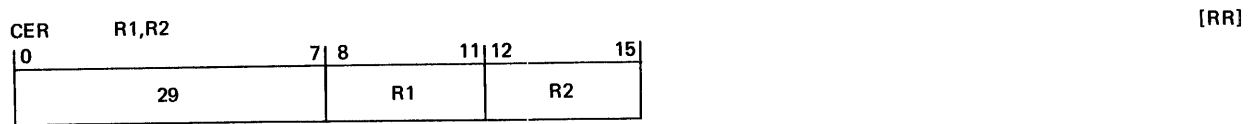
If the normalization causes exponent underflow, the entire floating-point result is set to zero, and the Overflow flag is set.

SER: (R1) ← (R1) - (R2)
 SE: (R1) ← (R1) - [A + (X2)]

Resulting Condition Code:

12	13	14	15	
C	V	G	L	
		0	0	DIFFERENCE IS ZERO.
		0	1	DIFFERENCE IS LESS THAN ZERO.
		1	0	DIFFERENCE IS GREATER THAN ZERO.
	1	X	X	EXPONENT OVERFLOW.
	1	0	0	EXPONENT UNDERFLOW.

4.11.5 Floating-Point Compare



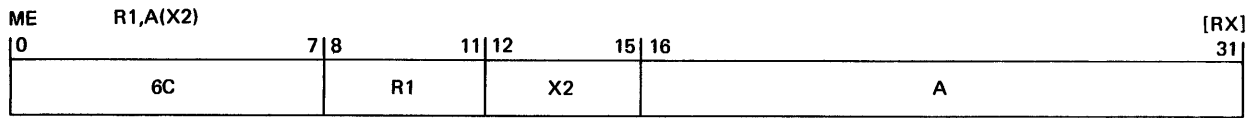
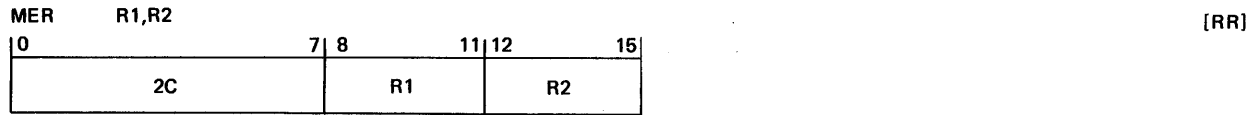
The first operand is compared to the second operand. Comparison is algebraic, taking into account the sign, fraction, and exponent of each number. The result is indicated by the setting of the Condition Code [PSW (12:15)]. Both operands remain unchanged.

CER: (R1) : (R2)
 CE: (R1) : [A + (X2)]

Resulting Condition Code:

	12	13	14	15	
	C	V	G	L	
			0	0	FIRST OPERAND EQUALS SECOND OPERAND.
			0	1	FIRST OPERAND IS LESS THAN THE SECOND OPERAND.
			1	0	FIRST OPERAND IS GREATER THAN THE SECOND OPERAND.
			0		FIRST OPERAND IS LESS THAN OR EQUAL TO THE SECOND OPERAND.
0					FIRST OPERAND IS GREATER THAN OR EQUAL TO THE SECOND OPERAND.
1					FIRST OPERAND IS LESS THAN THE SECOND OPERAND.

4.11.6 Floating-Point Multiply



The exponents of the two operands are added to produce the exponent of the result. The resultant exponent is readjusted to excess 64 notation. If an exponent overflow occurs, the exponent and fraction of the product are set to ones, and the Overflow flag is set. The sign of the product is determined by the rules of algebra. If an exponent underflow occurs, the entire floating-point result is set to zero, and the Overflow flag is set. In either event, the Floating-Point Arithmetic Fault Interrupt is caused, if enabled by Bit 5 in the PSW.

If an exponent overflow or underflow does not occur, the multiplication takes place. If the product is zero, the entire floating-point result is zero. If the result is not zero, normalization may occur. During normalization, the fraction is shifted left hexadecimally (four-bits at a time) until the most significant hexadecimal digit is not zero. The exponent of the result is decremented by one for each hexadecimal shift required. After normalization, the product is rounded to 24-bits.

If normalization causes the exponent to underflow, the entire floating-point result is set to zero, and the Overflow flag is set.

MER: (R1) ← (R1)*(R2)
 ME: (R1) ← (R1)* [A + (X2)]

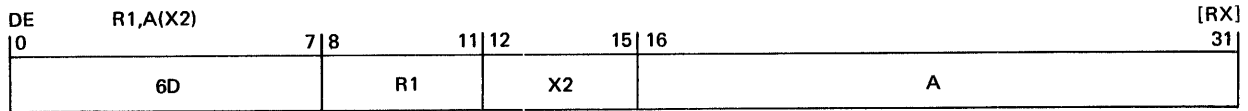
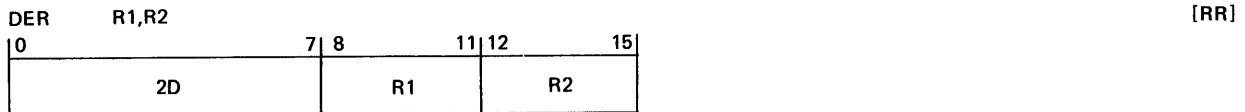
Resulting Condition Code:

12	13	14	15	
C	V	G	L	
		0	0	PRODUCT IS ZERO.
		0	1	PRODUCT IS LESS THAN ZERO.
		1	0	PRODUCT IS GREATER THAN ZERO.
1	X	X		EXPONENT OVERFLOW.
1	0	0		EXPONENT UNDERFLOW.

Programming Note:

The sum of the exponents of the two operands must be less than 64, or overflow occurs, producing the maximum possible value as a product. For example, the multiplication $1/2 \times 16^{63} * 1 = 1/2 \times 16^{63} * 1/16 \times 16^1 = 1/32 \times 16^{64}$ causes an overflow, rather than the result $1/2 \times 16^{63}$. Due to a rounding technique, the result of a floating-point multiply is more accurate in Models 70 and 80 than in Models 4 and 5.

4.11.7 Floating-Point Divide



The exponents of the two operands are subtracted to produce the exponent of the result. The resultant exponent is readjusted to excess 64 notation. If an exponent overflow occurs, the exponent and fraction of the quotient are set to all ones, and the Overflow flag is set. The sign of the quotient is determined by the rules of algebra. If an exponent underflow occurs, the entire floating-point result is set to zero, and the Overflow flag is set. If the divisor (the second operand) is zero, the operands are unchanged. In the event of exponent overflow, underflow, or division by zero; the Floating-Point Arithmetic Fault Interrupt is caused, if enabled by Bit 5 of the PSW.

If the exponent overflow or underflow does not occur, and if the divisor is not zero, the second operand is divided into the first operand. Division continues until the quotient is normalized, adjusting the exponent for each additional division required. If an exponent underflow occurs, the entire floating-point result is set to zero, and the Overflow flag is set.

No remainder is returned to the user. The quotient is rounded to compensate for the loss of the remainder.

DER: $(R1) \leftarrow (R1)/(R2)$
 DE: $(R1) \leftarrow (R1)/[A + (X2)]$

Resulting Condition Code:

12	13	14	15	
C	V	G	L	
		0	0	QUOTIENT IS ZERO.
		0	1	QUOTIENT IS LESS THAN ZERO.
		1	0	QUOTIENT IS GREATER THAN ZERO.
	1	X	X	EXPONENT OVERFLOW.
	1	0	0	EXPONENT UNDERFLOW.
1	1	0	0	DIVISOR EQUAL TO ZERO.

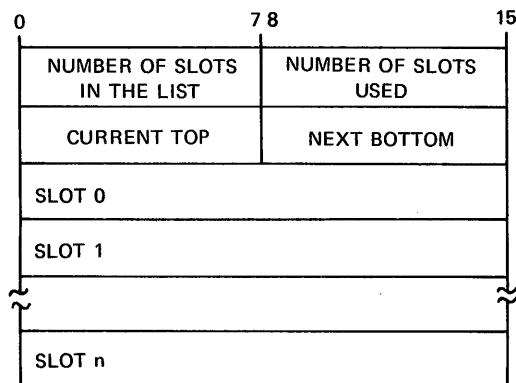
Programming Note:

Division by zero, overflow, or underflow cause a Floating-Point Arithmetic Fault Interrupt, if enabled by Bit 5 of the PSW. Inspection of the Condition Code of the Old PSW indicates the actual cause of the interrupt. If the Carry flag is set, then the divisor was zero. If the Carry flag is not set, then either overflow or underflow caused the interrupt. In this case, if the Greater than (G) or Less than (L) flag is set, the interrupt was caused by an overflow. If the G or L flag is reset, the interrupt was caused by an underflow.

The difference of the exponents of the two operands must be less than 64, or overflow occurs, producing the maximum possible value as a quotient, even when normalization of the computed mantissa would bring the resultant exponent within range.

4.12 LIST PROCESSING INSTRUCTIONS (Models 70 and 80 Only)

The List Processing instructions manipulate a circular list defined as follows:



The first two halfwords contain the list parameters. Immediately following the parameter block is the list itself. The first halfword in the list is designated Slot 0. The remaining slots are designated 1, 2, 3, etc. up to a maximum slot number which is equal to the number in the list minus one. An absolute maximum of 255 halfword slots is specifiable. (Maximum slot designation is equal to X'FE'.)

The first parameter byte indicates the number of slots (halfwords) in the entire list. The second parameter byte indicates the current number of slots being used. When this byte equals zero, the list is empty; when this byte equals the number of slots in the list, the list is full. Once initialized, this byte is maintained automatically. It is incremented when elements are added to the list and decremented when elements are removed.

The third and fourth bytes of the list parameters specify the current top of the list and the next bottom of the list respectively. These pointers are also updated automatically. See Figure 4-1.

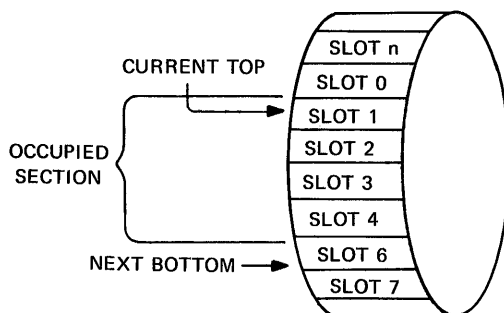
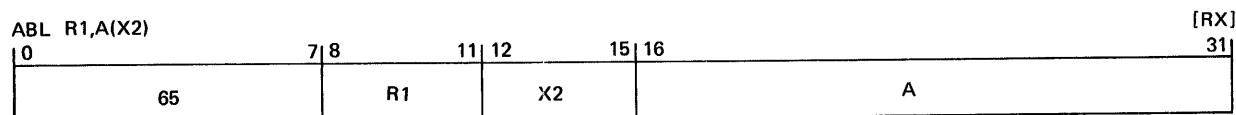
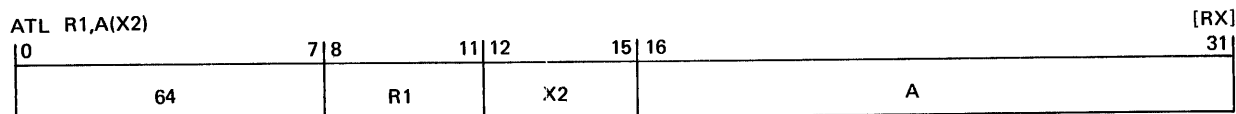


Figure 4-1. Circular List

The instructions described in this section are:

- 4.12.1 ATL Add to Top of List
- ABL Add to Bottom of List
- 4.12.2 RTL Remove from Top of List
- RBL Remove from Bottom of List

4.12.1 Add to Top/Bottom of List



The General Register specified by R1 contains the element to be added to the list. The second operand, A + (X2), specifies the address of the list. The number of slots used tally is compared to the number of slots in the list as specified by the first byte of the list. If the number of slots used tally is equal to the number of slots in the list an overflow condition exists. The element is not added to the list and the instruction terminates with the V flag set in the PSW. If the number of slots used tally is less than the number of slots in the list; it is incremented by one, the appropriate pointer is changed, the element is added to the list, and the instruction terminates with a Condition Code of zero.

Resulting Condition Code:

12	13	14	15	
C	V	G	L	
0	1	0	0	LIST OVERFLOW.
0	0	0	0	ELEMENT ADDED SUCCESSFULLY.

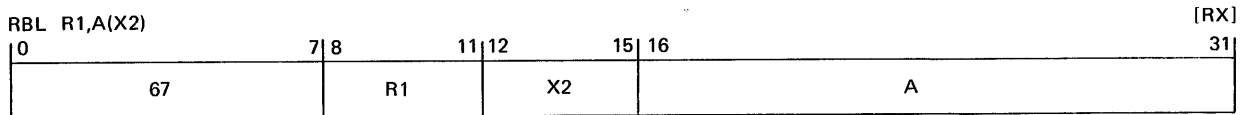
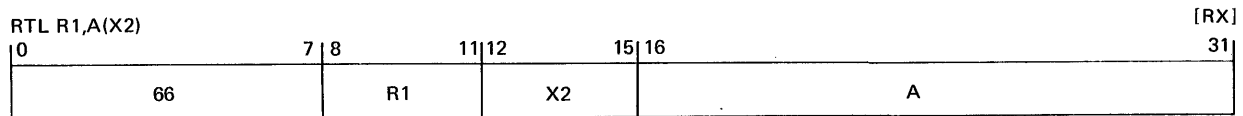
Programming Note:

The Add to Top of List (ATL) instruction, manipulates the Current Top Pointer in the list. If no overflow occurred, the Current Top Pointer, which points to the last element added to the top of the list, is decremented by one and the element is inserted in the slot pointed to by the new Current Top Pointer. If the Current Top Pointer was zero on entering this instruction the Current Top Pointer is set to the maximum slot number in the list. This condition is referred to as list wrap.

The Add to Bottom of List (ABL) instruction, manipulates the Next Bottom Pointer. If no overflow occurred, the element is inserted in the slot pointed to by the Next Bottom Pointer, and the Next Bottom Pointer is incremented by one. If the incremented Next Bottom Pointer is greater than the maximum slot number in the list, the Next Bottom Pointer is set to zero. This condition is referred to as list wrap.

This instruction is subject to Memory Protect.

4.12.2 Remove From Top/Bottom of List



The element removed from the list is placed in the General Register specified by R1. The second operand, A + (X2), specifies the address of the list. If, on entering the instruction the number of slots used tally is zero, the list is already empty and the instruction terminates with V flag set in the PSW. This condition is referred to as list underflow. If underflow does not occur the number of slots used tally is decremented by one, the appropriate pointer is changed, and the element is extracted and placed in R1. The instruction terminates with the Condition Code equal to zero if the list is now empty, or with the G flag set if the list is not yet empty.

Resulting Condition Code:

	12	13	14	15	
	C	V	G	L	
	0	1	0	0	LIST WAS ALREADY EMPTY.
	0	0	0	0	LIST IS NOW EMPTY.
	0	0	1	0	LIST IS NOT YET EMPTY.

Programming Note:

The Remove from Top of List (RTL) instruction, manipulates the Current Top Pointer. If no underflow occurred, the Current Top Pointer points to the element to be extracted. The element is extracted and placed in R1. The Current Top Pointer is incremented and compared to the maximum slot number. If the Current Top Pointer is greater than the maximum slot number, the Current Top Pointer is set to zero. This condition is referred to as list wrap.

The Remove from Bottom of List (RBL) instruction, manipulates the Next Bottom Pointer. If no underflow occurred, and the Next Bottom Pointer is zero it is set to the maximum slot number (list wrap); otherwise it is decremented by one and the element now pointed to is extracted and placed in R1.

This instruction is subject to Memory Protect.

CHAPTER 5 INTERFACE DESIGN

5.1 INTRODUCTION

This chapter discusses the INTERDATA Input/Output (I/O) System. There are several methods of communication between the Processor and peripheral devices and/or other system elements. The methods vary in speed, sophistication, and the amount of attention required by the Processor. Thus, the systems interface may be tailored to the individual user's needs and it may be gracefully expanded as the user's requirements grow.

There are two primary purposes for this chapter; 1. to familiarize the user with the INTERDATA systems interface, and 2. to provide the data required to permit the user to effectively interface peripheral equipment to the INTERDATA Digital System. A functional description of each I/O subsystem is given later in this chapter, followed by a detailed description of each I/O instruction, and rules and specifications for designing interfaces to the Digital System.

5.2 SYSTEMS INTERFACE

Figure 5-2 is a block diagram of an INTERDATA Digital System emphasizing the systems interface capability. Note that there are two separate methods of interfacing to peripheral devices or system elements:

1. To the Multiplexor Bus
2. To the Memory Bus in Models 70, 74 or to the DMA Bus in Model 80

The following paragraphs describe each of the interface methods.

	MODEL 74	MODEL 70	MODEL 80
Multiplexor Bus, Program Loop	66KBPS	76KBPS	150KBPS
Multiplexor Bus, Automatic I/O	————	49KBPS	60KBPS
Multiplexor Bus, Read/Write Block	250KBPS	330KBPS	500KBPS
Multiplexor Bus, Interleaved Data Channel	————	440KBPS	1000KBPS
Memory Bus or DMA via single Selector Channel (SELCH)	2000KBPS	2000KBPS	1850KBPS*

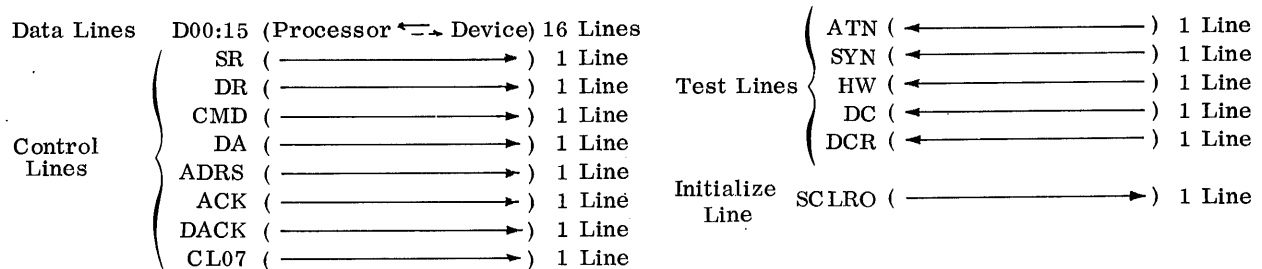
*At the 1850KBPS rate, the Model 80 CPU is guaranteed every other memory cycle, without CPU loading during DMA transfers. Rates to 3.15 MB are possible with a single SELCH and the CPU in the Wait state.

Figure 5-1. Input/Output Rate Comparison

5.2.1 Multiplex

The Multiplexor Channel is a byte or halfword oriented I/O system which communicates with up to 255 peripheral devices. The Multiplexor Bus consists of 30 lines; 16 bi-directional Data Lines, 8 Control Lines, 5 Test Lines, and an Initialize Line.

The lines in the Multiplexor Bus are:



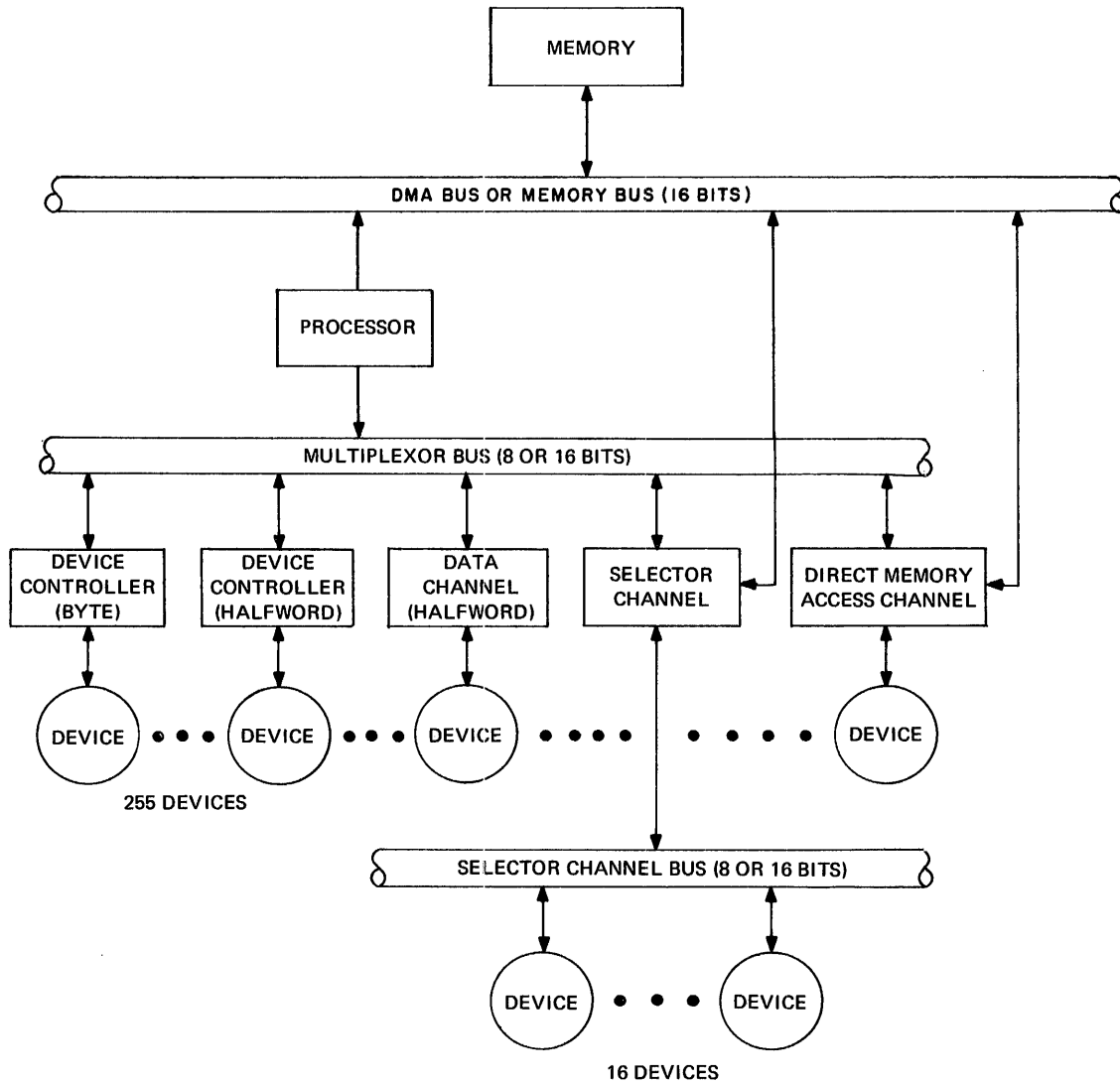


Figure 5-2. System Interface, Block Diagram

Figure 5-3 is a block diagram of the Multiplexor Channel. The following general definitions apply to the lines in the Multiplexor Bus.

Data Lines D00:15

The data lines are used to transfer one 8-bit byte or one 16-bit halfword of data between the Processor and the device. One byte of address or command is transferred from the Processor to the device over Data Lines 8:15 (D08:15) when accompanied by either an Address (ADRS) or a Command (CMD) control line. One byte of data or one halfword of data is transferred from the Processor to the device when accompanied by the Data Available (DA) control line. The device, in response to an Acknowledge (ACK) control line or a Sense Status (SR) control line, sends one byte of address or status information to the Processor over D08:15. In response to a Data Request (DR) control line, the device sends either an 8-bit byte or a 16-bit halfword of data to the Processor. The device always sends a Synchronize (SYN) signal to the Processor to indicate that it has either received the data from the Processor or that it has sent the data to the Processor. The SYN signal is removed immediately after the Processor removes the control line.

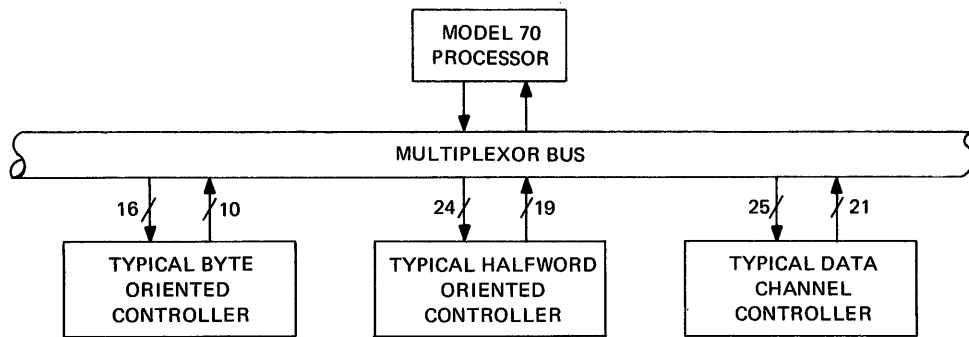


Figure 5-3. Multiplexor Channel, Block Diagram

Control Lines

- SR** Status Request. The device controller must present device status to Data Lines D08:15, followed by a SYN.
- DR** Data Request. The device controller presents data to Data Lines 8:15 or 0:15 (D08:15 or D00:15), followed by a SYN. If a Halfword (HW) of data is presented, the HW test line is also active.
- ACK** Acknowledge. The interrupting device controller presents its address on D08:15, followed by a SYN.
- DA** Data Available. The Processor presents data on D00:15 for transfer to the device. The device controller accepts the low byte or the entire halfword and responds with a SYN.
- CMD** Command. The Processor presents a Command Byte on D08:15. The device controller accepts the Command Byte and responds with a SYN.
- ADRS** Address. The Processor presents an Address Byte on D08:15. The device controller accepts the Address Byte and responds with a SYN.
- DACK** Data Channel Acknowledge (Models 70 and 80 only). The Processor presents an address of zero on D08:15. The ADRS control line and the DACK control line are simultaneously active. The interrupting Data Channel device controller becomes selected and responds with a SYN. As a result of addressing device zero (a null address), only the selected data channel device controller remains addressed.
- CL070** This control line is activated by the Processor when a Power Fail condition is detected by the Processor, if the Power Fail option is equipped. This line is held active until the SCLR0 signal occurs.

Test Lines

- ATN** Attention. Any device desiring to interrupt the Processor will activate the ATN line and hold this line until an ACK is received from the Processor.
- HW** Halfword. The HW line is activated by a halfword oriented device controller whenever it is communicating normally with the Processor. The HW line is not activated when a device controller is operating in the Interleaved Data Channel mode.

- DC Data Channel Request (Models 70 and 80 only). Any Data Channel device desiring to interrupt the Processor will activate the DC line and hold this line until a DACK is received from the Processor.
- DCR Data Channel Read (Models 70 and 80 only). The selected Data Channel device controls the state of the DCR line, high for read, low for write, from the device
- SYN Synchronize. This signal is generated by the device to inform the Processor that it has properly responded to a control line.

Initialize Line

- SCLR System Clear. This is a metallic contact to ground that occurs during Power Fail, Power Up, or Initialize.

NOTE

All control lines, except ACK and DACK, are connected in parallel to all devices. These lines are activated by the Processor in response to an external interrupt. The ACK line is connected in series with all devices. If no interrupt is pending in the first controller when the ACK or DACK signal arrives, the signal is passed on daisy chain fashion to the next controller, and so on until it is captured by the interrupting controller. See definition of ACK and DACK.

Communication over the Multiplexor Bus is performed on a request/response basis where each sequence of events is controlled by the micro-program contained in the Processor's Read-Only-Memory. A typical sequence to perform an I/O instruction with a device controller is:

1. The Processor addresses the device controller by placing an eight-bit address on the data lines and activating the ADRS control line. The device controller whose address corresponds to the Address Byte on the data lines responds by setting its Address flip-flop and returning SYN to the Processor. (All other device controllers reset their Address flip-flops.) Once a device controller is addressed it remains so until another device is addressed or until the system is initialized. The addressed device controller responds to all subsequent activity on the Multiplexor Bus.
2. If the I/O instruction involves transferring data from the Processor to the device controller, the Processor places the data on the data lines and activates the appropriate control line. The addressed device controller responds with a SYN after it has received the data, the Processor then removes the control line.
3. If the I/O instruction involves transferring data to the Processor from the device controller, the Processor activates the appropriate control line, and waits for the device controller to respond by placing the data on the data lines and activating SYN. When the Processor receives SYN, it accepts the data and removes the control line.
4. In all cases the device controller removes the SYN whenever the Processor removes the control line.

The sequence described here is somewhat simplified for the sake of clarity. The exact sequence for each I/O instruction is listed later in this chapter.

Whenever a device controller desires, it may interrupt the Processor by activating the ATN test line. This may be done by any device controller at any time, regardless of whether it is addressed or not. If interrupts are enabled by the Current Program Status Word, the Processor responds to ATN by interrupting the currently running program and directing the Processor to a new program (or a new micro-program) which identifies and services the interrupt as required.

5.2.2 Interleaved Data Channel (Models 70 and 80 Only)

The Interleaved Data Channel provides high speed low cost autonomous memory access on an instruction steal basis. Data transfer between the device controller and memory is accomplished over the Multiplexor Bus. Internal Processor registers provide the necessary buffering between the memory and the device controller. A typical sequence to perform an Interleaved Data Channel cycle is:

1. When the device controller is ready it requests a Data Channel cycle by activating the DC test line. This line is separate from, and of higher priority than, the ATN line. The Processor responds to this line by addressing device zero, and at the same time activating the DACK control line. The DACK line is "daisy chained" through all Data Channel devices until it is captured by the highest priority controller requiring Data Channel service. That device controller becomes the "on line" device.
2. The on line device controller controls the DCR test line which specifies either a Read or a Write operation to memory. If the Data Channel operation is Read from memory, the Processor inputs a 16-bit memory address from the device controller, reads the contents of that address from memory, and outputs the 16-bits read out to the device controller.
3. If the Data Channel operation is Write to memory, the Processor inputs a 16-bit memory address and a 16-bit data halfword, and then it writes that data into the memory at the specified address.

The sequence described here is somewhat simplified for the sake of clarity. The exact sequence for each kind of Data Channel operation is discussed later in this chapter.

5.3 DEVICE CONTROLLER LOGIC DESIGN

This section describes the procedures to follow in designing device controllers which connect to the Multiplexor Bus. While it is impossible to describe all possible controllers, this section explains representative circuits in sufficient detail to permit design of most controllers.

5.3.1 Multiplexor Bus

The Multiplexor Channel is a byte or halfword oriented I/O system which communicates with up to 255 peripheral devices. The Multiplexor Bus consists of 30 lines; 16 bi-directional Data Lines, 8 Control Lines, 5 Test Lines, and an Initialize Line as previously described.

All buses are false type, i.e. low level is active, high level is inactive. The device controller circuits used to communicate with the Multiplexor Bus are shown in Figure 5-4.

In a typical case, a device controller will receive an 8-bit Address Byte, an 8-bit Command Byte, and either an 8-bit data byte or a 16-bit data halfword from the Processor over the 16 bi-directional Data Lines (D00:15). Likewise, a device controller will send an 8-bit Address Byte, an 8-bit Status Byte, and either an 8-bit data byte or a 16-bit data halfword to the Processor over the 16 bi-directional Data Lines (D00:15). When only a byte of data is transferred, that byte is passed over the lower eight Data Lines (D08:15). The load resistors for all lines in the Multiplexor Bus are located in the Processor.

Each device controller is permitted one TTL load on any of the 16 bi-directional Data Lines, the 8 Control Lines, or the single Initialize Line. Each device controller is permitted one high power open collector TTL OR tie onto each of the 16 bi-directional Data Lines and each of the 5 Test Lines. (The open collector bus driver must be capable of sinking 48 ma at 0.4 VDC max collector voltage.) This gives the Processor a basic Multiplexor Bus drive capability of 10 device controllers. A Multiplexor Bus Buffer Figure 5-5, is available for extending the drive capability of the Multiplexor Bus incrementally by 16.

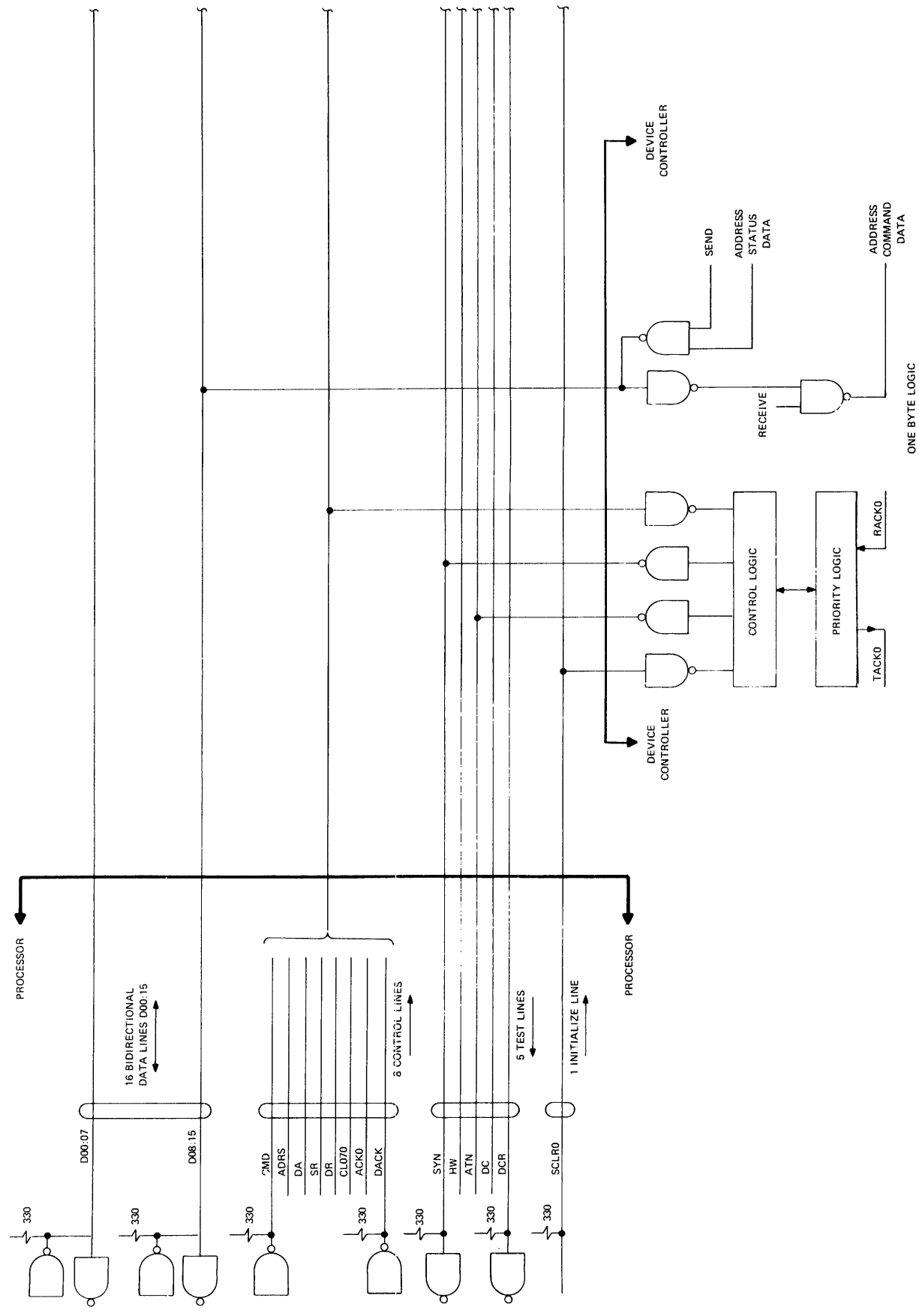


Figure 5-4. Processor/Device Controller Logic Interface (Sheet 1 of 2)

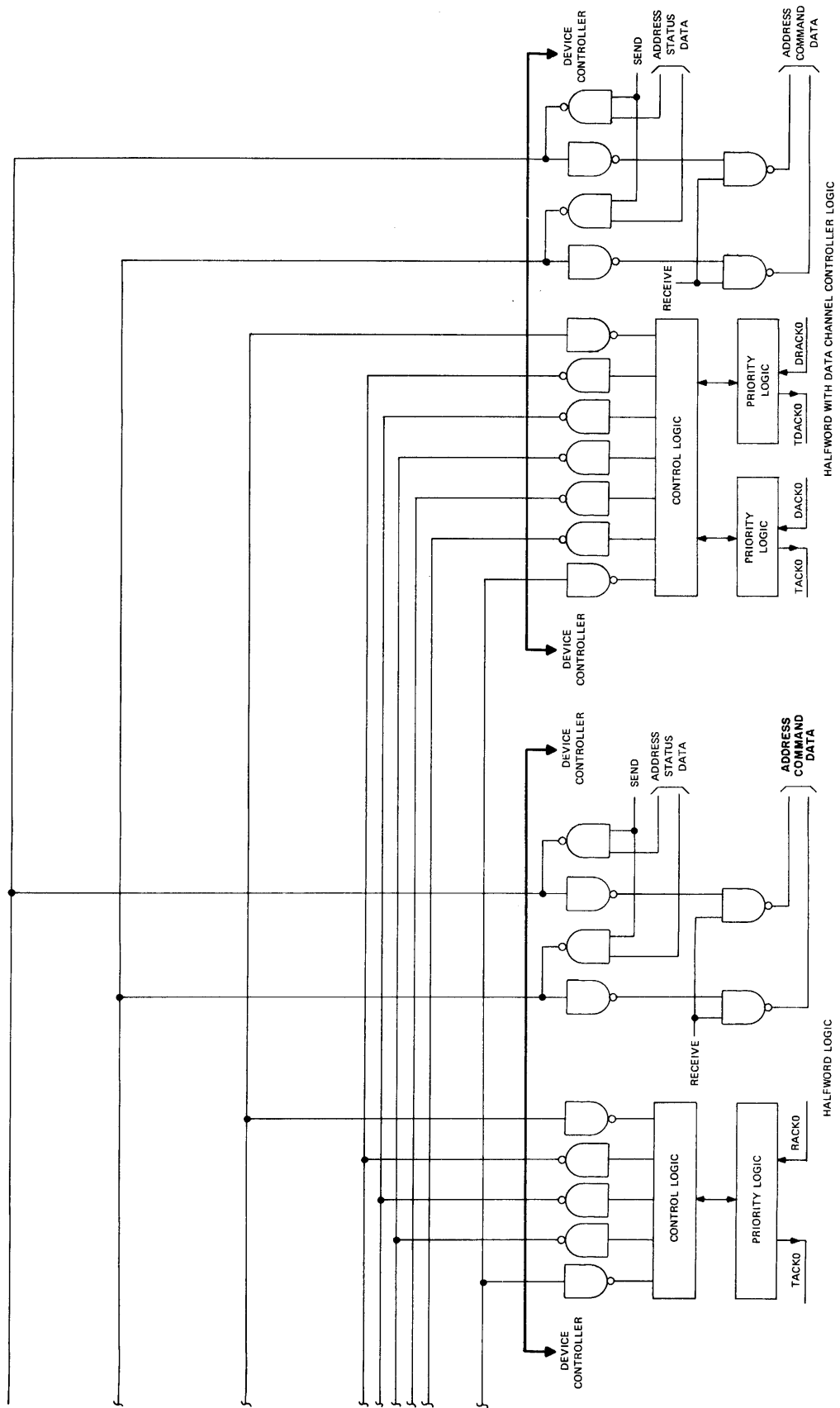


Figure 5-4. Processor/Device Controller Logic Interface (Sheet 2 of 2)

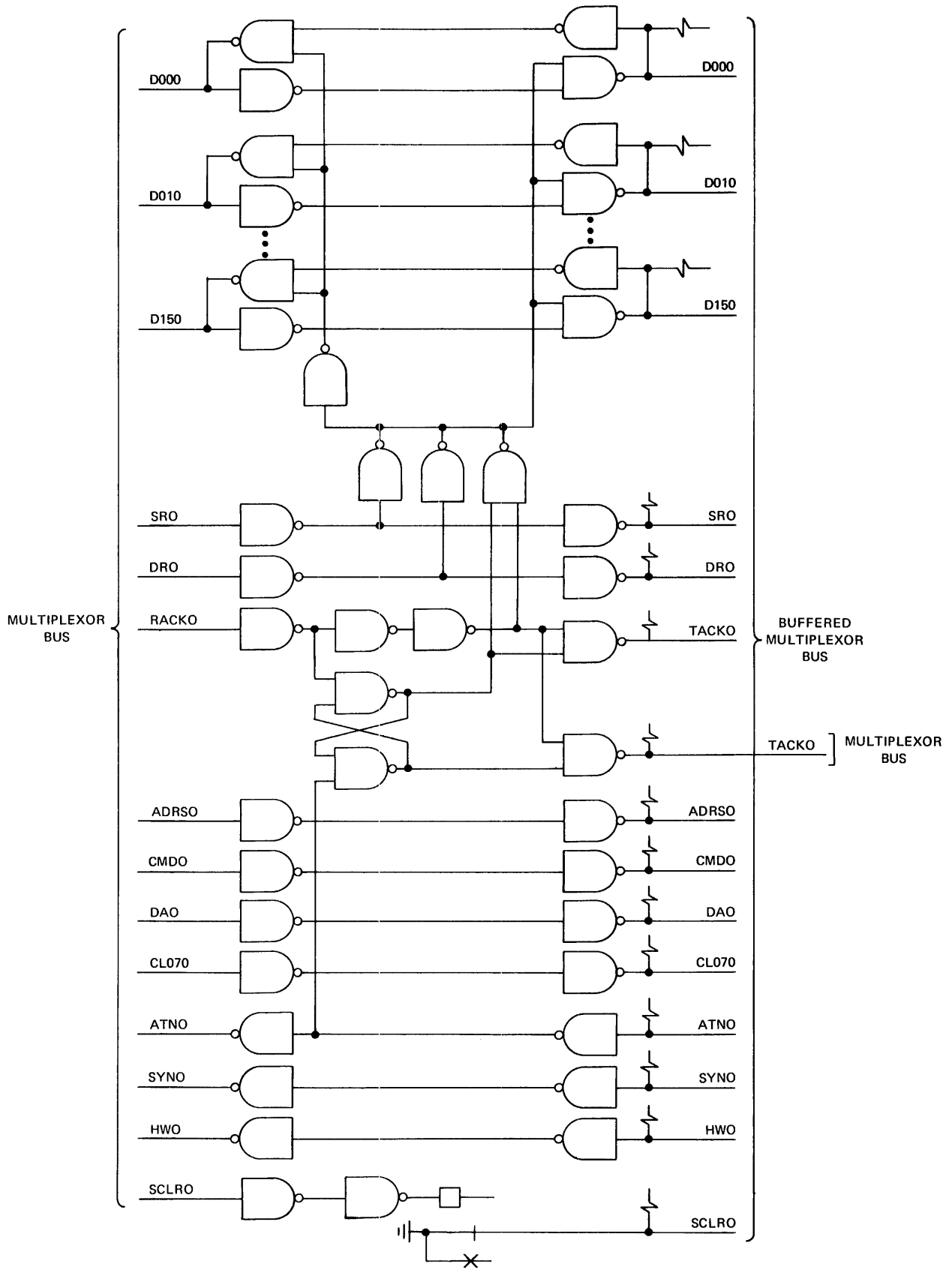


Figure 5-5. Multiplexor Bus Buffer

5.3.2 Device Controller Addressing

Refer to Figure 5-6 during the following description. The dotted lines around the groups of logic functions represent INTERDATA standard logic. Further details on the logic packs may be found later in this chapter. The designer may use his own logic packs provided they are level compatible with INTERDATA logic. When a device controller is addressed, the eight-bit address code is placed on the Data Lines (D080 through D150). The two buffers provide the true and false data lines. The Address Decoder circuit is hard-wired on each controller with its assigned address code, and the eight coded outputs are applied to an eight input gate. Thus, the Decoded Device output (DD1) goes true. The Address control line (ADRS1) then strobes the DD1 line into the ADRS flip-flop.

The Synchronize (SYN) signal is returned to the Processor, during the presence of ADRS1, via the Address Sync line, (ADSY0). Notice that an OR gate is used here for returning the other device Command Sync lines. The set output from the Address flip-flop, called Device Enable (DENB1), is used to gate all other I/O control lines to the device controller. When another device is addressed, the Decoded Device line (DD1) is low, causing the ADRS1 strobe line to reset the Address flip-flop and disabling the controller. Thus, only one device controller may be addressed at any time. During the address cycle, only the device that was addressed returns a SYN.

NOTE

The designer must design the device controller such that when some other device is addressed, the previously addressed controller will clear its Address flip-flop as soon as possible, and, in no case more than 350 nanoseconds. Otherwise the system could have two devices addressed simultaneously.

The device controller logic must delay SYN until it has reacted to the Multiplexor Bus control line, however, unnecessarily long delays serve only to reduce the system input/output operation.

NOTE

If the device controller is a 16-bit halfword oriented controller it activates the Halfword Enable Line (HW0) immediately when its Address flip-flop is set. The HW0 is used by the Processor to determine if the device is capable of sending or receiving 16-bit halfword data in parallel.

5.3.3 Data and Status Input

5.3.3.1 Data

Figure 5-7 shows how a byte or halfword of data may be read into the Processor. When an 8-bit byte oriented device controller is addressed, DENB1 is high, enabling the Data Request (DR) control line from the Processor. The DR enables the data byte onto the eight bottom Data Lines (D080 through D150). If a 16-bit halfword oriented device controller is addressed, one data byte is enabled as described above, and a second data byte is enabled onto the eight top Data Lines (D000 through D070) by an active Halfword (HW) signal. A system requirement is that the addressed controller must respond to all control lines (i. e., Data Request) with a SYN.

5.3.3.2 Status

Figure 5-7 shows how a byte of status may be read into the Processor. When the byte or halfword oriented device controller is addressed, DENB1 is high, enabling the Status Request (SR) control line from the Processor. Open collector gates are used for OR tying multiple data and status sources onto the eight Data Lines (D080 through D150). A system requirement is that the addressed controller must respond to all control lines (i. e., Data Request) with a SYN.

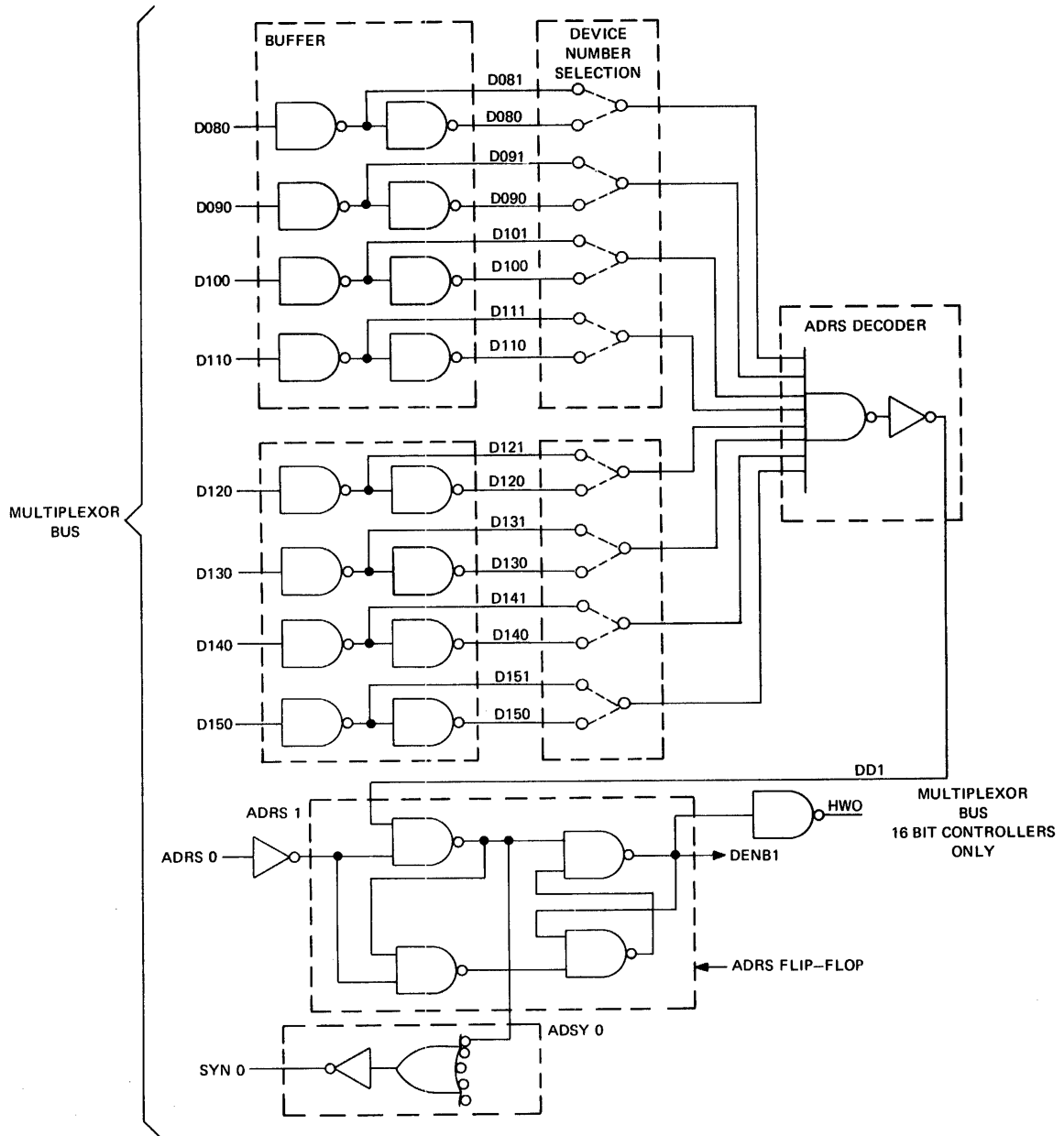


Figure 5-6. Device Addressing, Logic Diagram

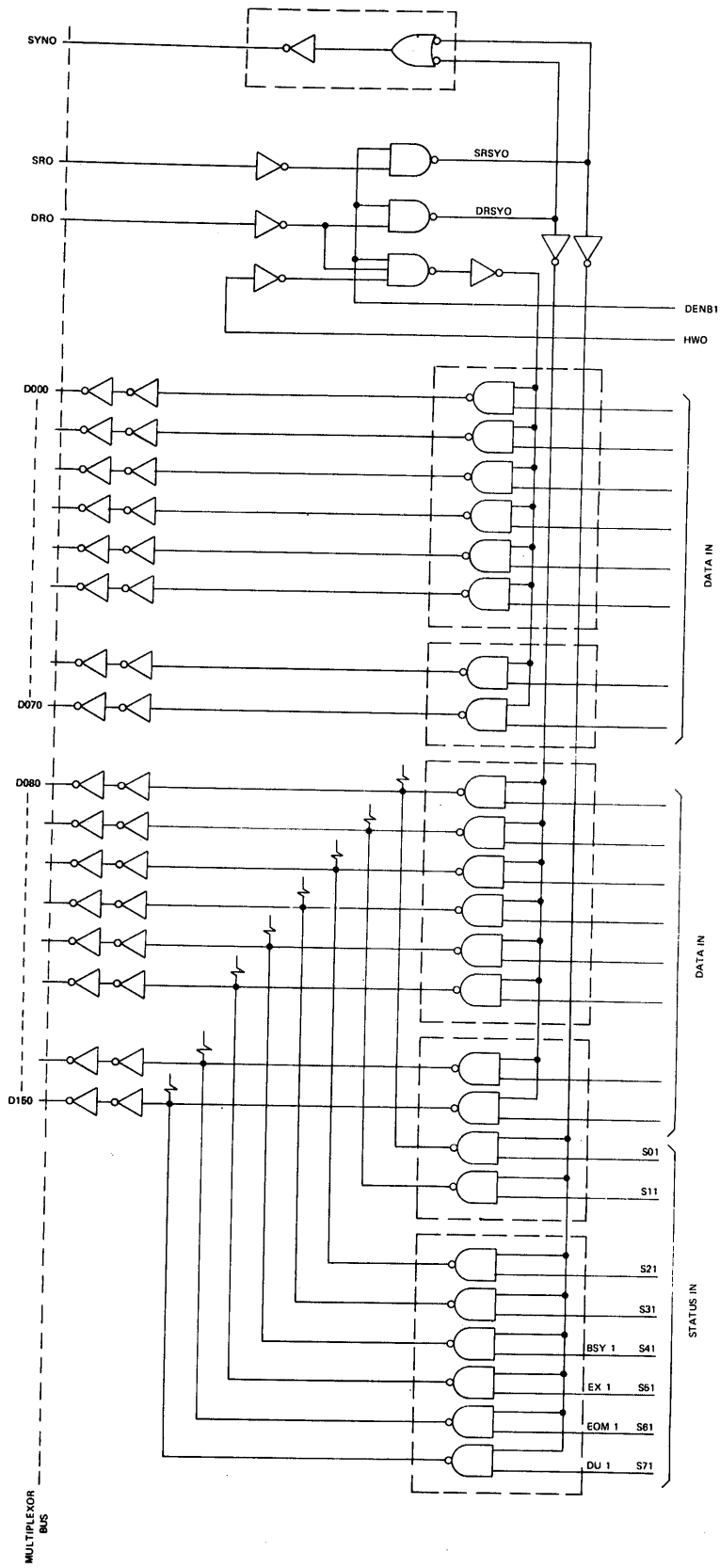


Figure 5-7. Data and Status Input Logic Diagram

The device controller logic should place a high on BSY1 until the data is ready. The Processor may now be synchronized to the device data rate by testing the device status until the Busy bit is low. Then, when the Busy bit is low, the program may transfer data. Device synchronization can also be achieved by generating an interrupt when the data is ready.

The End of Medium (EOM) bit is normally placed high at the termination of the device medium, such as End of Card. The Device Unavailable (DU) bit typically signifies that device power is not turned on.

The Examine Status (EX) bit is used to signify other appropriate device conditions. In this case, the user assigns S01 through S31 to appropriate conditions, such as Parity Error, etc.

It is appropriate to note here that the Busy Status is unconditionally defined such that data cannot be transferred unless Busy is false. The remaining status bits are defined as required by the device controller. Not all device controllers require all eight status bits.

Device controllers must be designed such that the Processor or the Selector Channel maintains the Status Request line once the current status of the device is presented. Specifically, if the status changes while the Status Request line is true, the status byte returned to the Processor or Selector Channel should also change.

5.3.4 Data and Command Output

5.3.4.1 Data

Figure 5-8 shows how a data byte may be output from the Processor. The buffered true and false Data Lines (D081 through D151 and D080 through D150) from Figure 5-6 connect to the set and reset inputs of the Data Register.

NOTE

If the device controller is a 16-bit halfword oriented controller it must typically invert Data Lines D000 through D007 also.

When the device is addressed, DENB1 is high, enabling the control line DAG1 to strobe the data condition into the J-K flip-flop Data Register. The DASY0 line also returns the SYN signal to the Processor.

5.3.4.2 Command

The command lines are shown on Figure 5-8 as being used in the toggle mode. For example, a high on Bit 8 (D081) sets a control relay when CMG1 goes high. A high on Bit 9 (D091) resets the relay. Bits 14 and 15 are shown operating an indicator. Other pairs of bits may be used to enable/disable interrupts, etc.

Again, note that definition of the command bits is a function of the device controller only. Not all device controllers require eight separate commands. However, up to 256 commands are possible.

5.3.5 Interrupt Control

Figure 5-9 shows a complete general purpose interrupt and interrupt acknowledge logic system. When an interrupt is generated, the Queue flip-flop is DC set via a differentiated negative going pulse. The output from the Queue flip-flop generates an Attention signal (ATN0) to the Processor. ATN0 is connected to the interrupt line in the Processor. The program responds with an Acknowledge Interrupt signal, which is received by the controller as Receive Acknowledge (RACK). Since the Queue flip-flop was set prior to receiving RACK, the output of Gate G1 disables Gate G9, holding the Gate G9 output high. The high output from Gate G9 stops TACK0 from sending the Acknowledge to the next device. Thus, RACK1 and the Gate G2 output generate ATSY0 via Gate G3. ATSY0 sends a SYN back to the Processor, and also forces the outputs from G18 through G25 high.

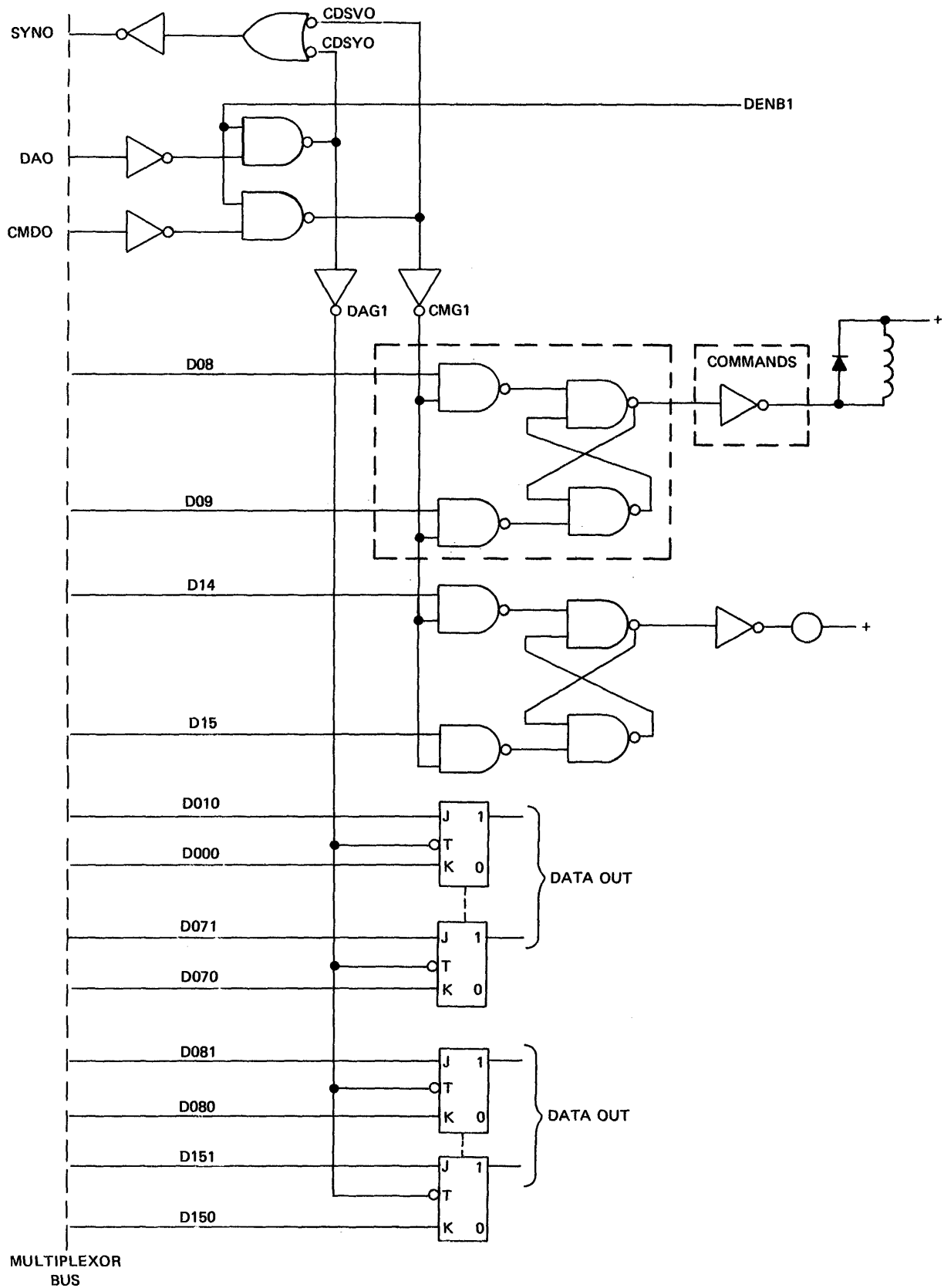


Figure 5-8. Data and Command Output, Logic Diagram

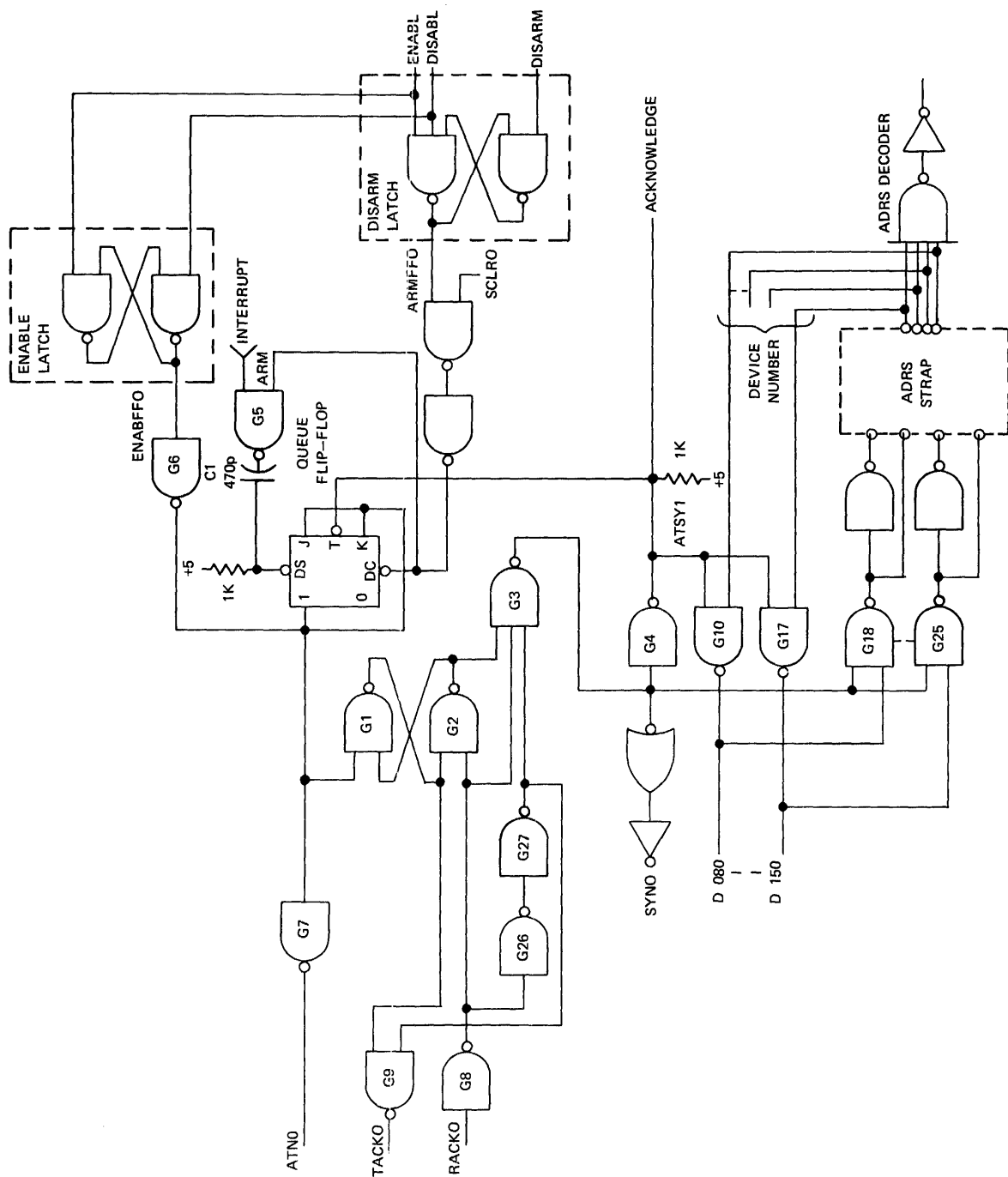


Figure 5-9. Interrupt Control, Logic Diagram

This causes the device number wired in by the address strap board to appear on the inputs of Gates G10 through G17. Thus, the ATSY1 output from Gate G4 enables the device number onto the Data Lines (D080 through D150).

The output from Gate G4 also raises the Acknowledge signal to the device. On receiving the SYN0, the Processor lowers RACK1, causing the output of Gate G4 to drop. This in turn causes the Queue flip-flop to reset.

NOTE

If the interrupt has not set the Queue flip-flop, the RACK1 signal passes through Gate G2 to TACK0, and on to the next device.

If RACK1 is high in response to another device, the output from Gate G2 is low, thus disabling the interrupt from affecting Gate G1. However, the interrupt remains in the Queue flip-flop, and is serviced after completion of the previous interrupt service.

The ENABFF0 and ARMFF0 lines provide control over the Interrupt Queue flip-flop and the ATN0 line to the Processor. Normally, two bits of a command byte (Bits 0 and 1) are decoded such that, with Bit 0 true and Bit 1 false, the Queue flip-flop is disabled. That is, the flip-flop may be set, however, its output is held low. Gate G6, whose input is ENABFF0 from the false side of the ENABLE latch, provides this function. The command byte, with Bit 0 false and Bit 1 true, is decoded (ENABL goes false) and sets the ENABLE latch which allows new interrupts or a queued interrupt to be recognized. Bits 0 and 1, both true, are decoded to drive DISARM false which sets the DISARM latch. The false side of the latch is used to clear the Queue flip-flop and to prevent the interrupt line from setting it. The DISARM latch is cleared whenever the ENABLE or DISABLE commands are recognized. Encoded commands ENABLE/DISABLE/DISARM thus provide interrupt masking or inhibiting within the device controller.

As described previously, the Control Line, CL050, from the Processor carries the Interrupt Acknowledge (ACK) signal. This line breaks up into a series of short lines to form the "daisy-chain" priority system. The ACK signal must pass through every controller that is equipped with Interrupt Control circuits. This includes all device controllers except a few special cases.

Back panel wiring for interrupt control is shown in Figure 5-10. At a given position, the Received ACK (RACK0) appears at Pin 122-1 and the Transmitted ACK (TACK0) at Pin 222-1. The daisy-chain bus is formed by a series of isolated lines which connect Terminal 222-1 of a given position to Terminal 122-1 of the next position (lower priority). On unequipped positions, a jumper shorts 122-1 and 222-1 of the same connector to complete the bus. Back panels are wired with jumpers on all positions. Whenever a card chassis position is equipped with a controller, the jumper from 122-1 to 222-1 must be removed from the back panel at that position.

For controllers that occupy several positions, the jumper is removed only at the position where the controller board has ATN/ACK circuits.

5.3.6 Multiplexor Bus Wiring

Wiring for the Multiplexor Bus and for the Selector Channel Bus is identical (same) in the Processor and expansion chassis. Each card position contains two connectors with the Multiplexor and Memory Buses wired to each at pin positions indicated in Figure 5-10.

5.3.7 Multiplexor Channel Timing

Both the Input and Output operations on the Multiplexor Channel make use of request/response signaling. This allows the system to run at its maximum speed whenever possible, but permits a graceful slowdown if the characteristics of a particular device controller requires signals of longer duration. Device controller designs should keep Multiplexor Channel usage as fast as possible, consistent with practical circuit margins. Doing this assures the fastest computer input/output operation when a system is configured with a number of peripheral devices.

Timing for typical Input/Output operations are shown on Figure 5-11. On the Output operation, the Processor places a signal on the data lines followed by an appropriate control line signal. This stagger (T1) will vary, but it is guaranteed to be at least 100 nanoseconds. When the device controller has received the Output Byte, the SYN signal is returned to the Processor, which terminates the control line signal. Realizing that T5 is 100 nanoseconds minimum, the SYN delay T2 should be only long enough to guarantee proper reception of the Output Byte. The control line/data line removal time (T3) is important where single-rail to double-rail operation is used - e.g. the ADRS flip-flop on Figure 5-6. A minimum of 100 nanoseconds is guaranteed for T3. For SYN generation as per Figure 5-6 and 5-9, the control line signal is DC coupled through the gates to form the SYN signal. The SYN removal time (T4) should be minimized. This delay should not be unnecessarily extended since the Processor will not begin another Input/Output operation until SYN is removed.

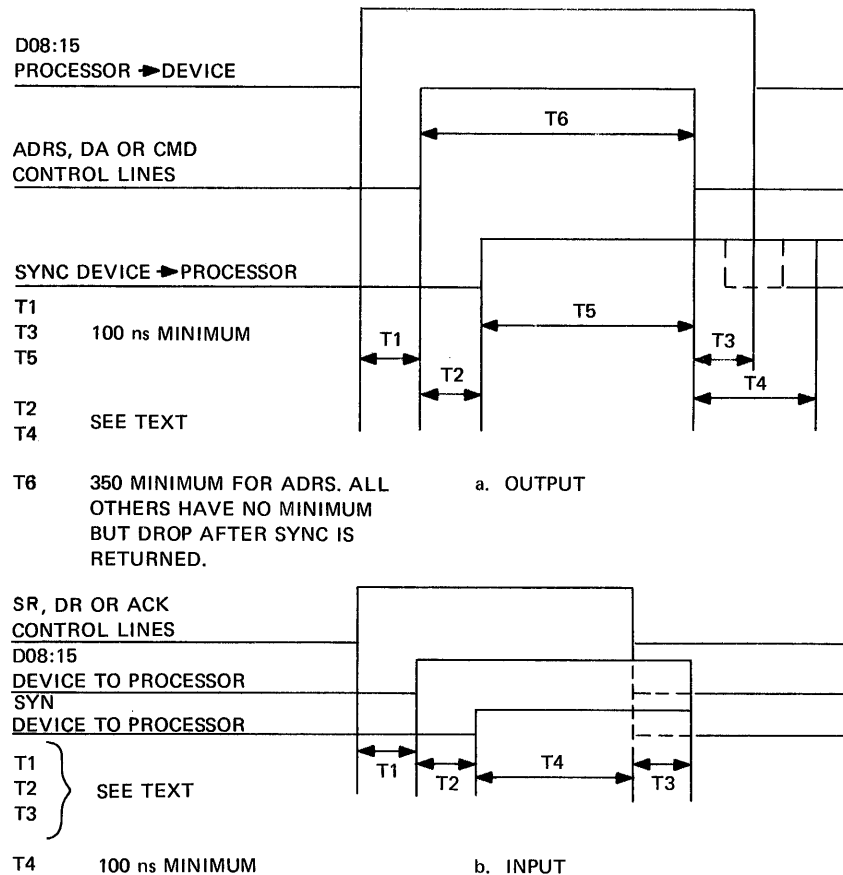


Figure 5-11. Multiplexor Channel Timing

It should be emphasized that the times shown on Figure 5-11 are defined for signals on the Multiplexor Channel. Within a given controller, one signal may flow through more gates than another signal and these delays must be considered.

For the Input operation, the Processor places a signal on a control line. The currently addressed device controller should gate signals to the data lines as soon as possible to keep T1 at a minimum. The SYN delay (T2) must guarantee that the Input Byte is on the data lines considering the slowest data gates and the fastest SYN gates. The Processor will remove the control line signal when SYN is received with a minimum delay (T4) of 100 nanoseconds. With SYN and the byte gate DC coupled to the control line, the removal delay (T3) will be the sum of the corresponding gate delays. The Processor considers the operation complete when SYN falls.

When the control signal is ACK, the delay T1 will include the cumulative Gate G8/G9/G26/G27 delays (See Figure 5-9) for all the controllers between the responding controller and the Processor. This will be less than the Processor time-out even with the maximum limit of 255 controllers.

NOTE

With a SYN delay of 50 nanoseconds, device controllers must be designed to accept a minimum width of 170 nanoseconds on all control line pulses except ADRS and DACK which are guaranteed to be 350 nanoseconds minimum. The SYN delay in the device controller may be increased to effectively lengthen the control line pulses if it is absolutely necessary. It is essential to realize that after the Processor initiates a control line signal, the Processor does nothing until the SYN signal is returned by the device controller; one or more cycles are skipped if necessary and the data transfer rates decreased proportionally. While this may not affect a particular device controller, the overall system performance is degraded. Furthermore, if a device controller fails to respond with a SYN in the time out period of approximately 15-35 microseconds, the Processor will abort the instructions.

5.3.8 General Multiplexor Bus Interface

Figure 5-12 illustrates a general interface to the Multiplexor Bus which may be used when designing custom device controllers, either 8-bit byte or 16-bit halfword oriented. (If an 8-bit byte oriented interface is being designed, Gates connecting to D001:071 and to DAT001:071 can be eliminated.) The address straps can be hardwired by the user for any device number from 03 to 255. Address 01 and 02 are hardwired for the Control Console and Teletypewriter, respectively. The user can use the Gated Status Request (SRG0) or the Gated Data Request (DRG0) control lines to gate status or data from appropriate points in his logic. Data from the Processor is available to the user's circuits, double rail, at the points labeled D001:151 and D000:150. The user can use the Gated Data Available (DAG0) and the Gated Command (CMG0) control lines to gate the data from the Processor to appropriate points in his logic. The delay of the SYN0 signal should be arranged such that it is the minimum delay necessary for the custom controllers to function properly, per Section 5.3.7.

5.3.9 Interleaved Data Channel Interface Design

The Data Channel feature of the Model 70 and 80 Processor provides high speed, autonomous memory access to customer designed device controllers. Halfword data transfer to memory is accomplished over the Multiplexor Bus in the Burst Mode.

Interfaces to the Data Channel must follow the same general design rules which apply to regular interfaces to the Multiplexor Bus.

The program initiates a Data Channel operation, usually by issuing an Output Command instruction. Then the Data Channel device completes the transfer without further direction by the Processor.

When a memory access is desired, the controller activates the Data Channel Request line (DC). This line is separate from and of higher priority than the normal I/O Attention line (ATN).

The Data Channel Request is honored on an instruction stealing basis.

The Processor acknowledges the DC line by addressing device number zero and also activating the DACK control line. (Addressing device number zero resets the Address flip-flop in all devices.)

The Data Channel Acknowledge line (DACK) is "daisy chained" through all device controllers on the Data Channel. The device controller closest to the Processor that originated the Data Channel Interrupt captures the DACK signal. That device becomes the 'on line' device and controls the DCR line. The DCR line tells the micro-program whether a memory Write (DCR=0) or a memory Read (DCR=1) operation is to be performed.

If a memory Read is requested, the micro-program reads the 16-bit memory address from the controller. The selected location is read and the 16-bit data is output to the controller.

If a memory Write is requested, the micro-program reads the 16-bit memory address from the controller, then reads the 16-bit data word. The data word is stored in the selected core location.

If another memory cycle is desired, the device controller re-activates the DC line.

Figure 5-13 shows a block diagram of a typical Data Channel device controller. Figure 5-14 illustrates the timing for a typical Data Channel Read and Write operation.

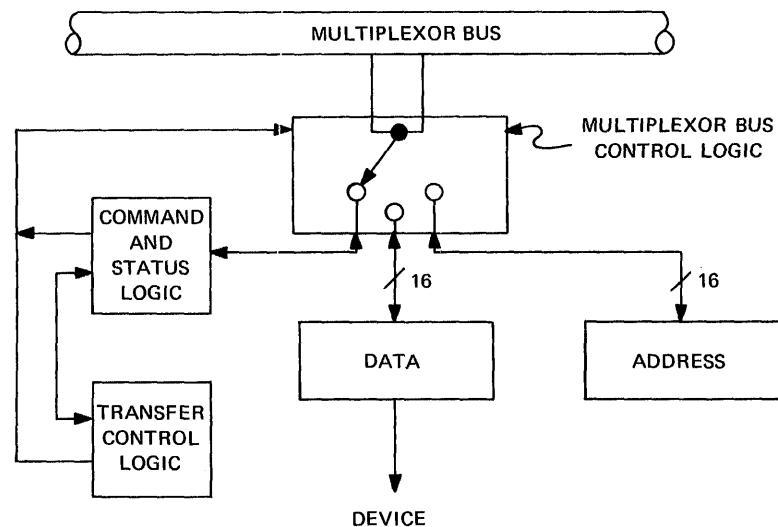


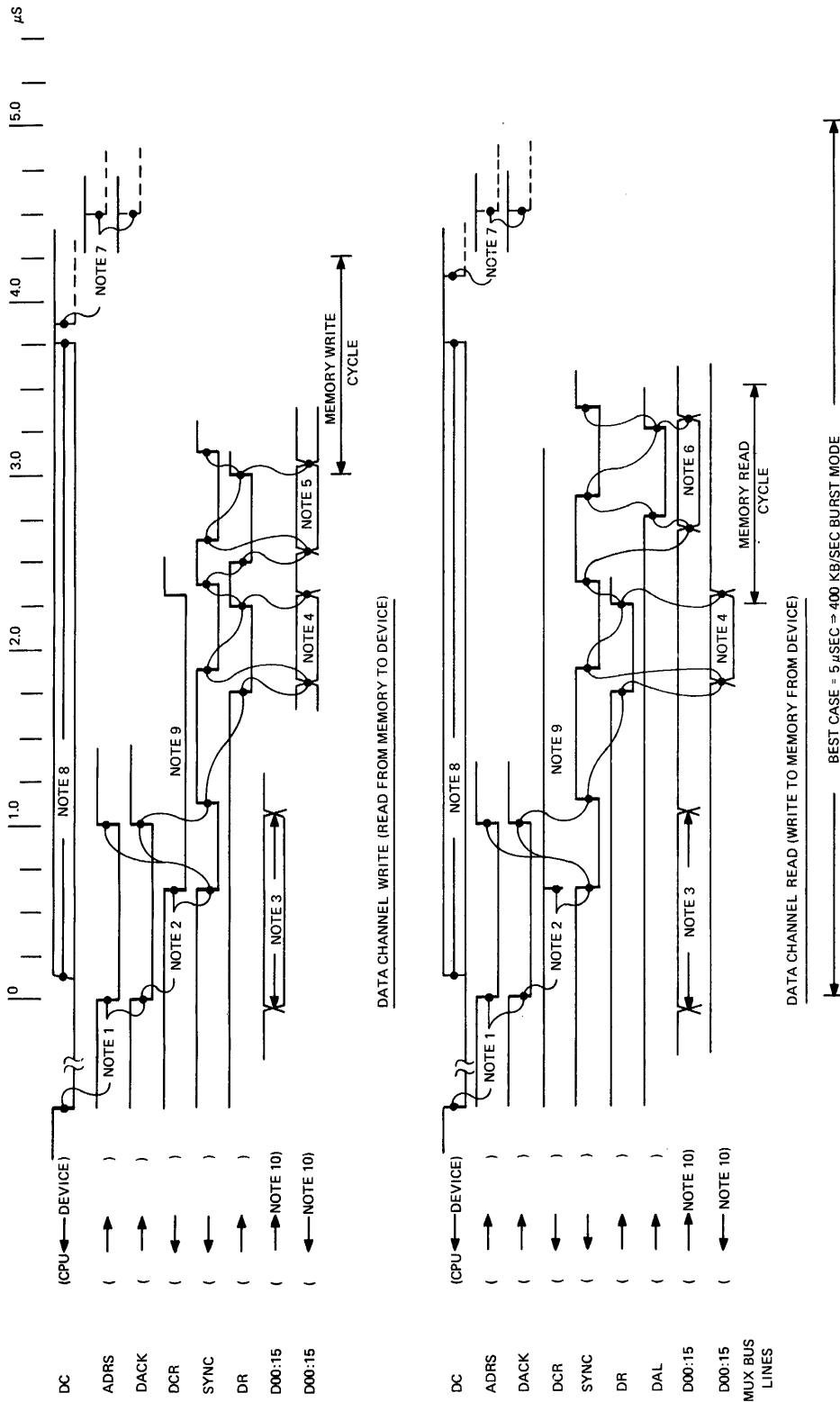
Figure 5-13. Interleaved Data Channel, Block Diagram

Figure 5-15 illustrates a general interface to the Multiplexor Bus which may be used when designing custom interfaces to the Interleaved Data Channel. Note that the logic circuits in Figure 5-15 are nearly identical to those in Figure 5-13, (Standard Multiplexor Bus Interface). A second daisy chain circuit has been added (Gates G94 through G101) to the data channel interface. This daisy chain captures the RDACK0/TDACK0 pulse, and forces the interface Address flip-flop (F1) set.

The user's circuit must provide a Set Data Channel (SDC0) request pulse to initiate a Data Channel (DC) operation. At the same time the user provides a RD1 level (high for read, low for write) to indicate the type of operation. The Select flip-flop (F5) is set when this interface captures the data channel cycle, and remains set until the cycle is complete.

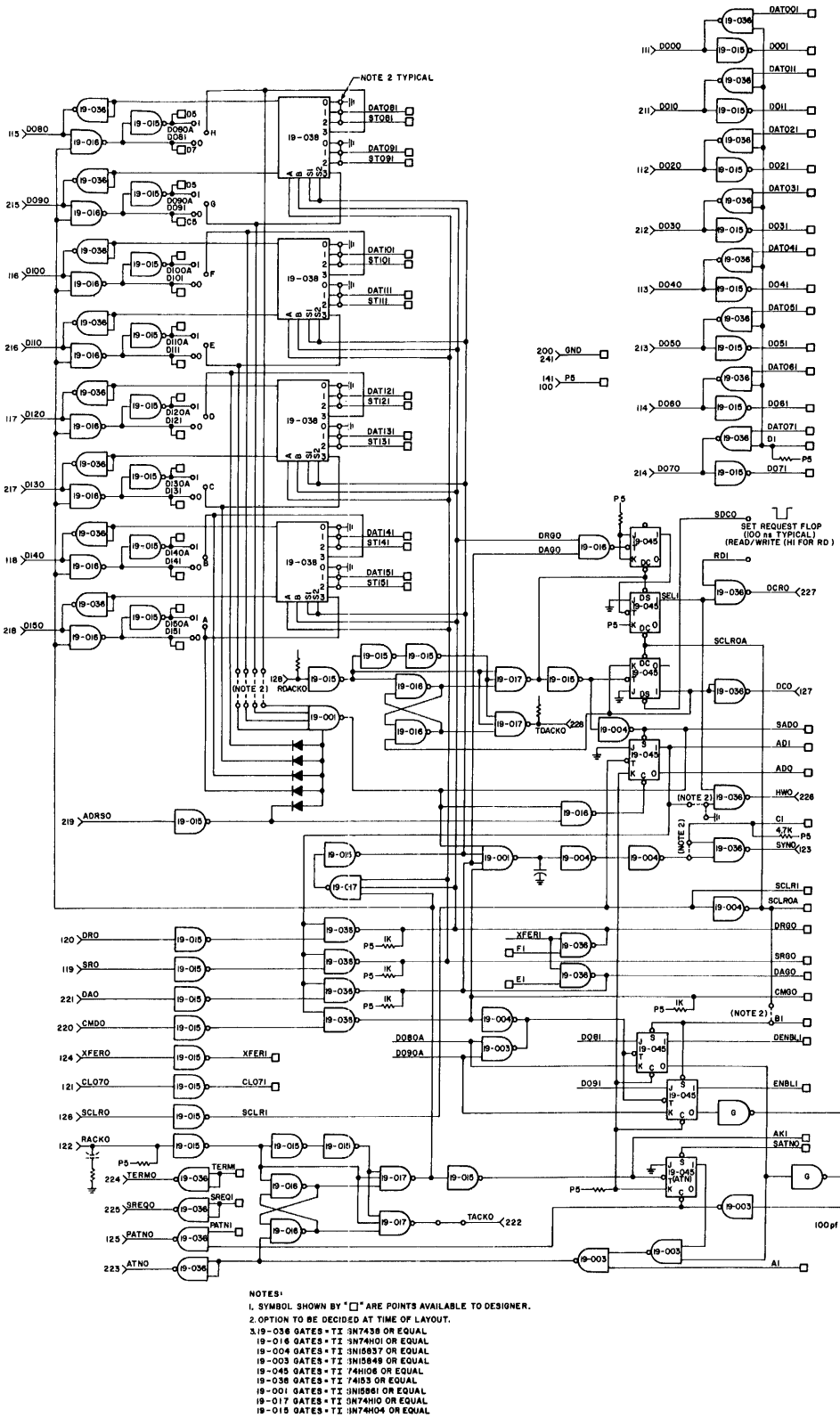
Note from Figure 5-10 that the Interleaved Data Channel control line DCO, DCRO, and the daisy chain priority line RDACK0/TDACK0 appear only on the 0 level connector on the back panel. Therefore Data Channel devices must connect into the 0 level connectors. Note also that these lines are not reconstituted by the Bus Buffer.

Note on Figure 5-15 that when a Data Channel device is selected (i.e., SEL flip-flop set) it must not hold the HW line active even though all data transmitted over the Multiplexor Bus is 16-bit parallel.



- NOTES:
1. DC CAN BE ACTIVATED AT ANY TIME. TYPICAL LATENCY TIME IS 4 μS EXCEPT FOR BLOCK I/O. LOAD/STORE MULTIPLE OR AUTOMATIC I/O CHANNEL.
 2. SYNC SHOULD BE DELAYED LONG ENOUGH TO ASSURE THAT THE ADDRESS FLOPS OF ALL CONTROLLERS ARE RESET. 600 ns TYPICAL. SEE NOTE 3.
 3. D00:15 = ZEROS THIS INTERVAL (A NULL ADDRESS).
 4. THE DEVICE SENDS A 16-BIT MEMORY ADDRESS.
 5. THE DEVICE SENDS A 16-BIT DATA WORD WHICH IS WRITTEN TO MEMORY.
 6. THE PROCESSOR OUTPUTS A 16-BIT READOUT FROM MEMORY. THE NEXT DATA CHANNEL CYCLE BEGINS IN BURST MODE.
 7. DC MAY BE RELEASED ANY TIME IN THIS INTERVAL.
 8. DCR HIGH ⇒ WRITE TO MEMORY; LOW ⇒ READ FROM MEMORY.
 9. D00:15 IS A COMMON BI-DIRECTIONAL BUS SHOWN TWICE FOR CLARITY.
 - 10.

Figure 5-14. Data Channel Timing Chart



NOTES:
 1. SYMBOL SHOWN BY "□" ARE POINTS AVAILABLE TO DESIGNER.
 2. OPTION TO BE DECIDED AT TIME OF LAYOUT.
 3. 19-036 GATES - TI 3N7438 OR EQUAL
 19-016 GATES - TI 3N74H01 OR EQUAL
 19-004 GATES - TI 3N15837 OR EQUAL
 19-003 GATES - TI 3N15849 OR EQUAL
 19-045 GATES - TI 74H108 OR EQUAL
 19-038 GATES - TI 74N33 OR EQUAL
 19-001 GATES - TI 3N15881 OR EQUAL
 19-017 GATES - TI 3N74H01 OR EQUAL
 19-019 GATES - TI 3N74H04 OR EQUAL

Figure 5-15. Interleaved Data Channel/Multiplexor Bus Interface

5.4 MEMORY BUS

5.4.1 Introduction

The Memory Bus provides a high speed communications path between the memory modules on one side, and the Processor, Selector Channels, or specially designed Direct Memory Access Channels on the other side. Before the Memory Bus is covered in detail, the various memory systems used on the Model 74, 70, and 80 are covered in the following paragraphs.

The Models 74 and 70 use 8,192-byte, 3 wire 3D core memory modules with a 1.0 microsecond cycle time and a 300 nanosecond access time. Core memories have a destructive readout; when a 16-bit memory location is read, the read currents (ERO and LRO in Figure 5-16) reset the location to zero in the read process.

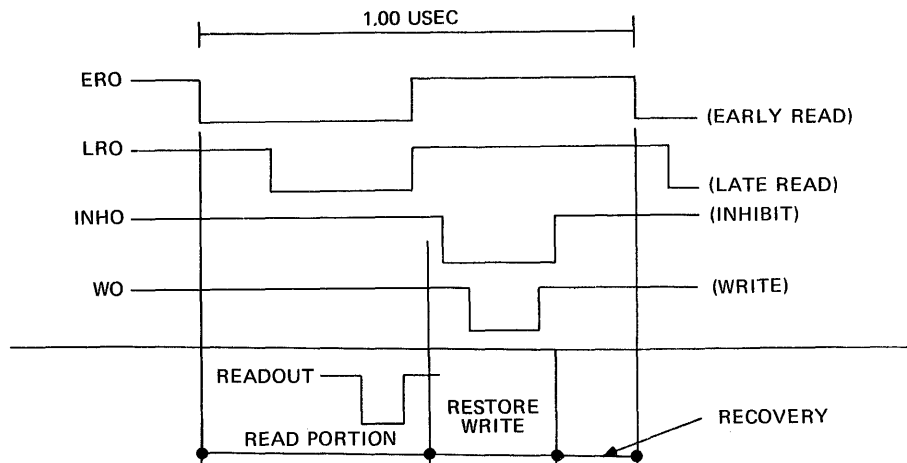


Figure 5-16. Typical Memory Cycle

In the Model 80, 16,384-byte MOS memory modules are used. This module consists of two 8,192 byte blocks, one for even halfword addresses, and one for odd halfword addresses. This arrangement allows for the overlapped fetching of sequential halfwords thereby decreasing the execution time. The cycle time of MOS memories varies depending on how much one can interleave memory cycles. MOS memories have a non-destructive readout; when a 16-bit memory location is read, the contents of that location are not altered. Also, a write cycle request in solid state memories does not necessitate a "read" half cycle to reset the location to zero prior to writing into it. A simplified MOS memory cycle is illustrated in Figure 5-17.

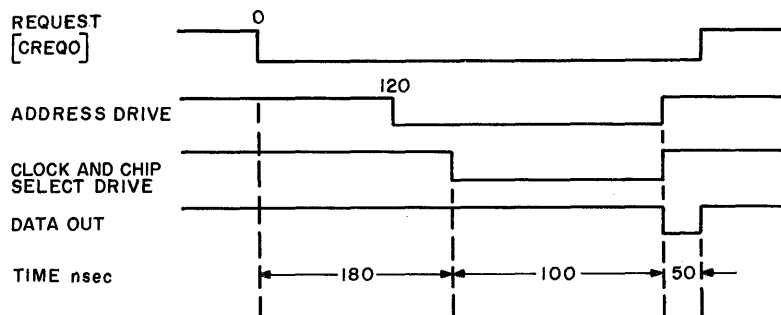


Figure 5-17. MOS Memory Cycle

The appropriate Processor maintenance manual should be consulted for a more detailed description of both the core and MOS memories.

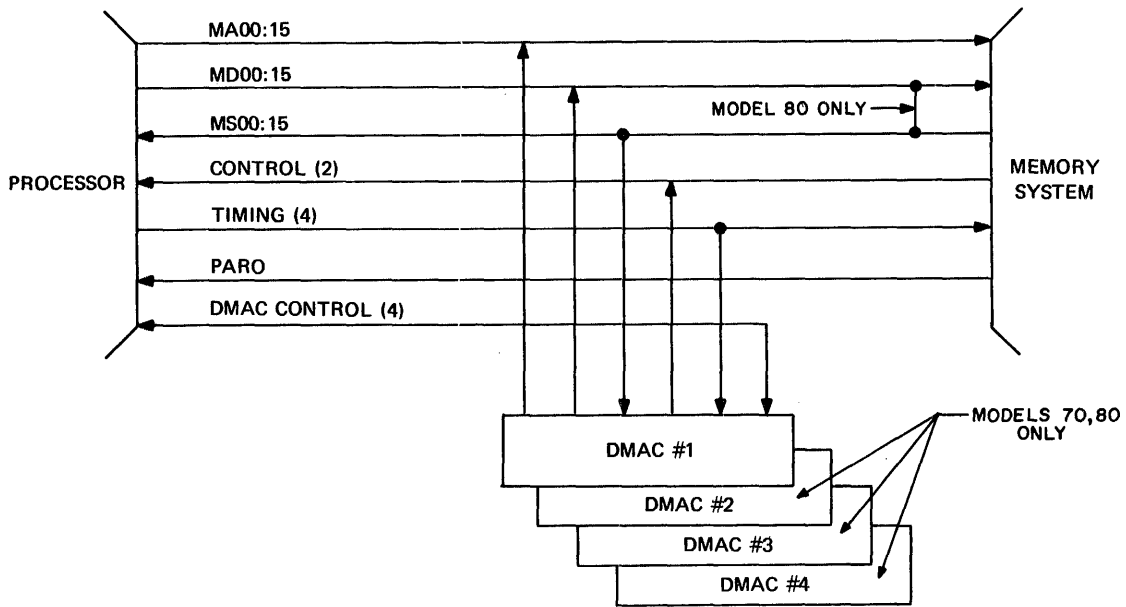


Figure 5-18. Memory Bus Diagram

5.4.2 Memory Bus Lines

Both the core memories and the MOS memories are interfaced through a similar Memory Bus. The Memory Bus consists of 58 lines, and only one device may communicate with it at any one time. The 58 lines are described (see Figure 5-19, 5-20, and 5-21 as needed and Table 5-1):

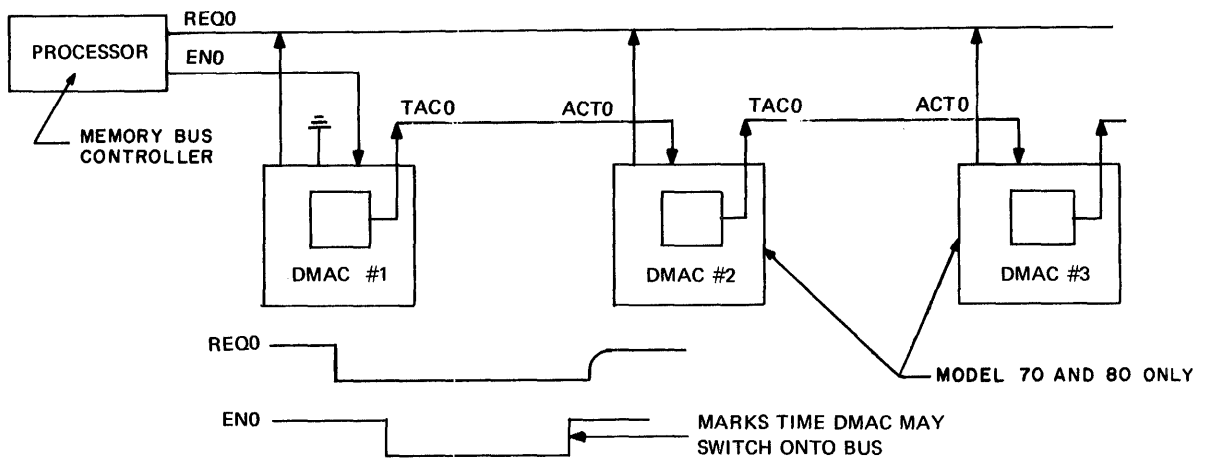


Figure 5-19. Example of Memory Bus Priorities

TABLE 5-1. MODEL 80 ACCESS AND TRANSFER TIMES

FIRST ACCESS TIMES	Options	
	Standard	Burst
Worst Case Access Time (ATWC)	940	940
Best Case Read Access Time (ATBCR)	510	510
Best Case Write Access Time (ATBCW)	400	400

TRANSFER TIMES		
Worst Case with CPU Fetch (32 bit read)	ATWC+n800*	ATWC+n420*
Best Case Read Burst	TABCR+n420*	=
Best Case Write Burst	TABCW+n330*	=

*n=number of transfers

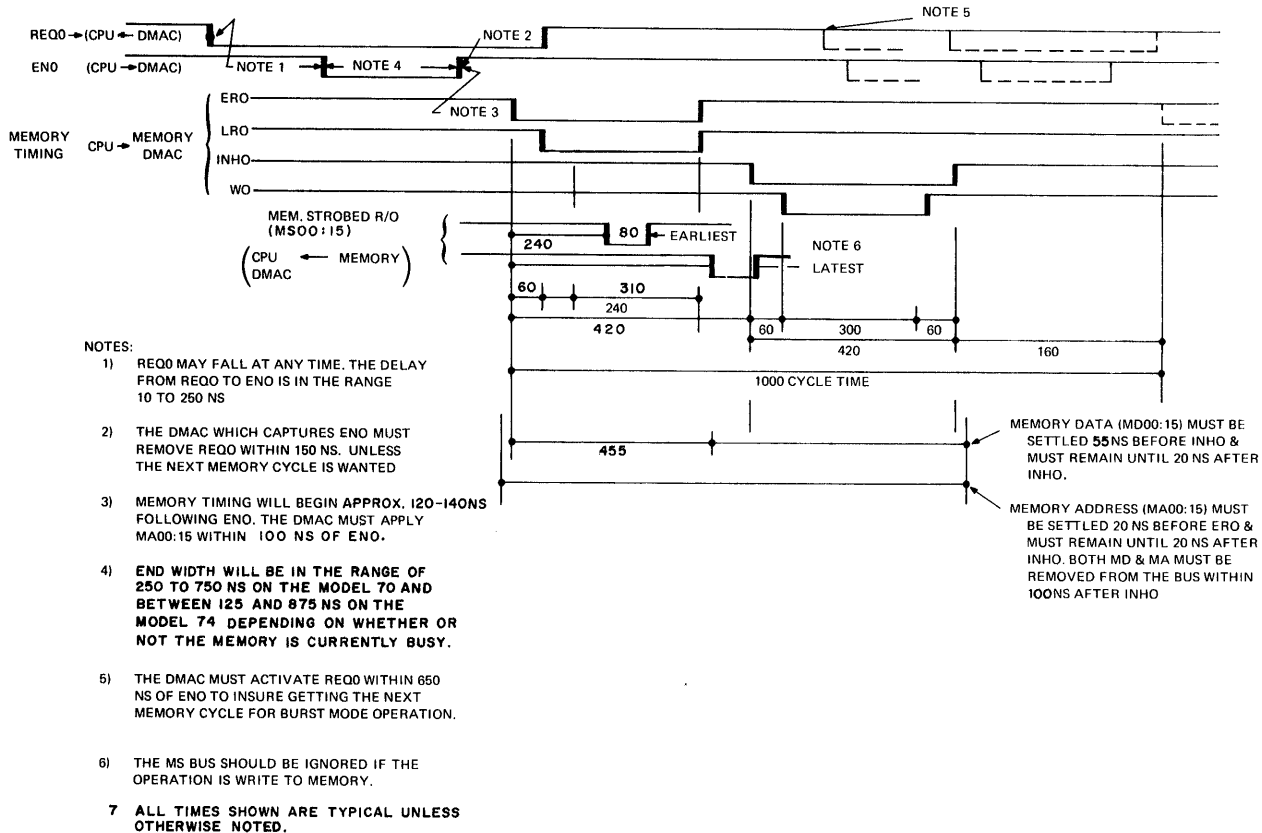
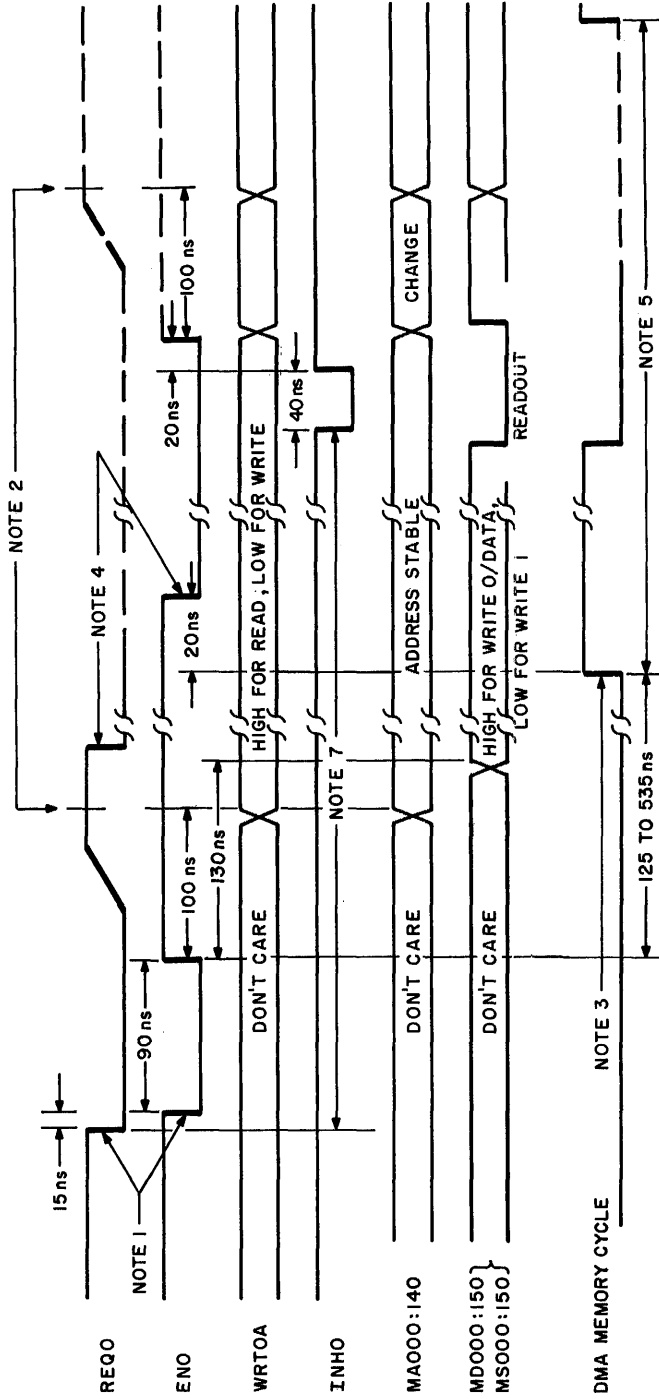


Figure 5-20. Model 74 and 70 Memory Bus Timing



NOTE 1: REQO MAY FALL AT ANY TIME. ENO GOES LOW 15 ns LATER. ENO WIDTH IS 90 ns MINIMUM.

NOTE 2: THE SELECTED DMAC MUST REMOVE REQO AND PROVIDE STABLE ADDRESS DATA AND WRTOA SIGNAL WITHIN 100 ns AND DATA WITHIN 130 ns FROM END OF ENO.

NOTE 3: MAXIMUM LATENCY IS 535 ns. THIS OCCURS WHEN THE CPU STEALS A CYCLE FOR A FULL WORD INSTRUCTION (FETCH) AT THE LATEST POSSIBLE TIME.

NOTE 4: IF REQO GOES LOW DURING A CONCURRENT DMA CYCLE (OR IF REQO STAYS LOW), ENO WILL ALSO BE OVERLAPPED WITH THAT CYCLE.

NOTE 5: MAXIMUM CYCLE TIME IS 800 ns WHEN DMA REQUESTS MEMORY READS AND THE CPU STEALS ALTERNATE CYCLES FOR A FETCH (32 BIT) READ CYCLE. IF BURST STRAP OPTION IS INSTALLED, THE MAXIMUM CYCLE TIME IS 420 ns FOR READ BURSTS, WHILE THE CPU IS LOCKED OUT ONCE THE DMA GAINS ACCESS. WITH THE STANDARD STRAP OPTION, THE BEST CYCLE TIME IS ALSO 420 ns IF THE CPU IS IN WAIT STATE. WRITE BURSTS CYCLE AT 350 ns RESPECTIVELY.

NOTE 6: THE DMAC IS GUARANTEED EVERY OTHER MEMORY CYCLE IF THE REQUEST RATE MEETS CONDITIONS ABOVE. CPU OR REFRESH IS ALLOWED ALTERNATE CYCLES TO UTILIZE DMA BUS TIME; SEE NOTE 6 AT END OF THIS SECTION.

NOTE 7: WORST CASE DMA FIRST CYCLE ACCESS TIME IS 940 ns MAXIMUM WHEN CPU STEALS MEMORY FOR A FETCH CYCLE.

Figure 5-21. Model 80 Memory Bus Timing

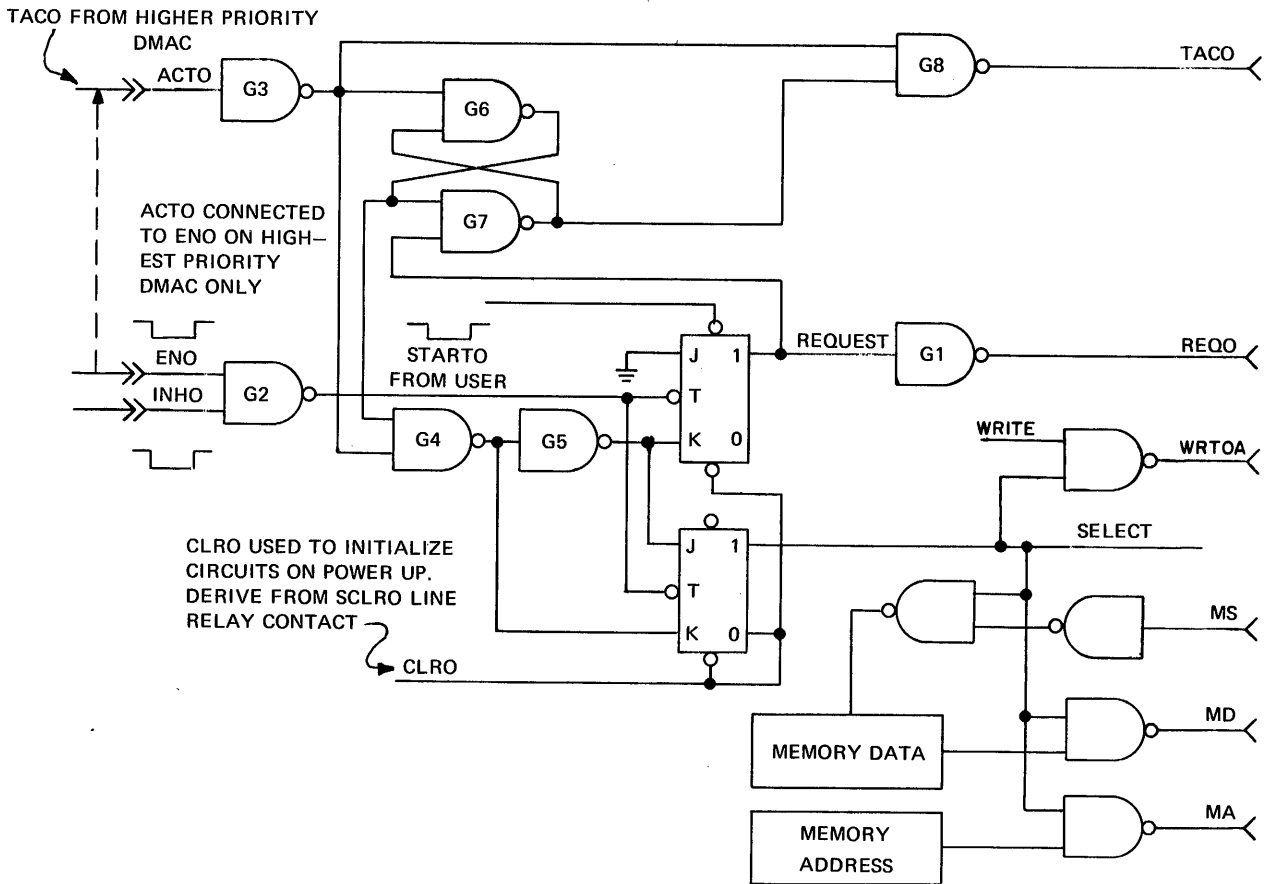
Request (REQ0)

This is a request for memory from a Selector Channel or a custom designed Direct Memory Access Channel (DMAC). More than one device may activate this line at any time. Unless these devices want consecutive memory cycles, REQ0 must be released within 130 nsec after the trailing edge of EN0.

Enable (EN0)

The Processor responds to REQ0 by activating EN0. The delay from REQ0 to EN0, leading edge to leading edge, is 15-250 nsec. depending on state of memory.

EN0 daisy-chains through all Selector Channels and DMAC's (TAC0 and ACT0 on Figure 5-19), and the captured daisy-chain pulse is ANDed with the rising edge EN0 to set the "Select" flip-flop in a DMAC. Figure 5-22 illustrates a suitable circuit for this action.



STARTO (30 TO 100 NS) FROM USER CKT SETS REQUEST FLOP WHEN USER'S MEM. ADRS & DATA REGISTERS ARE READY. CPU RESPONDS TO REQ0 WITH EN0. IF THIS DMAC DID NOT REQUEST SERVICE IT PASSES THE DAISY CHAIN PULSE (ACTO) ON TO THE NEXT DMAC AS TACO. IF THIS DMAC DID REQUEST SERVICE IT CAPTURES THE ACTO PULSE (A 30 NS GLITCH ON TACO IS PERMISSIBLE). ON THE RISING EDGE OF EN0 THE REQUEST FLOP IS RESET & THE SELECT FLOP IS SET. THE SELECT FLOP ENABLES THE DMAC TO THE MEMORY BUS (MS, MD, & MA). THE SELECT FLOP IS RESET AT THE END OF THE MEMORY CYCLE BY THE RISING EDGE OF INHO.

Figure 5-22. Typical Control Logic for Interfacing to the Memory Bus

Early Read (ER0)
Late Read (LR0)

These two lines control the "read" half of a core memory cycle and they are generated by the Processor approximately 120-140 nanoseconds after the trailing edge of EN0. These two signals are not used in the MOS memory system, and are not generated by the Model 80 Processor.

Inhibit (INH0)
Write (W0)

In Models 70 and 74, two lines control the "write" half of a core memory cycle. These are INH0 and W0. They are generated by the Processor. The leading edge of INH0 indicates to the DMAC that data, from a Memory Read operation, is available. The trailing edge of this signal indicates the termination of the memory cycle. W0 is used by the memory.

In the Model 80 System, INH0 (leading edge) indicates that the data readout onto the data lines is valid. The duration of INH0 is 40 nanoseconds. W0 is not generated by the Model 80 Processor.

Write (WRT0A)

WRT0A has no role in the core memory. In the MOS System, a Selector Channel or similar interface specifies to the memory via WRT0A a read or write operation. WRT0A is low for a write operation. WRT0A must be stable from 100 nanoseconds after EN0 until the end of INH0. WRT0A is used in the Models 70 and 74 to enable the Parity logic for the generation of correct Parity on DMAC Write operations.

Memory Address Lines
(MA00:14)

The DMAC may activate these lines no later than 100 nanoseconds following the trailing edge of EN0. These address lines must be settled 50 nanoseconds or more before MD00:15 memory data lines and remain stable until 20 nanoseconds after INH0. The address lines must be released no later than 100 nanoseconds after INH0.

Memory Strobed Readout
Lines (MS00:15)

These lines carry the pulsed signal memory readout on all memory operations. The leading edge of the readout occurs X nanoseconds after the trailing edge of EN0 for M 70, 74 while M80 read out cannot be anticipated but is concurrent with INH0.

- a. M70, 74: 410 to 470 nanoseconds from EN0
- b. M80: 10 nanoseconds before INH0 leading edge to 20 nanoseconds following INH0 trailing edge

The duration of this pulse is:

- a. M70, 74: 60 to 100 nanoseconds
- b. M80: 70 nanoseconds

In M70, 74 systems the selected DMAC must return this readout onto the memory data lines (MD00:15) and it must be settled during the entire INH0 pulse.

In M80 systems MS and MD lines are strapped together, therefore the readout is also present on the memory data lines (MD00:15) and it is settled during the entire INH0 pulse. The selected DMAC therefore shall not return MS contents onto the MD lines.

Memory Data Lines
(MD00:15)

These lines carry data to be written into memory.

MD00:15 must be settled during the interval described in Figure 5-20 for the Models 70 and 74 and Figure 5-21 in the Model 80.

- NOTES:
1. MS16 and MD16 correspond to the Parity Bit. The Processor controls these lines in the Core System. The DMAC must not load these lines. In the MOS System, parity is checked and generated in the memory system itself.
 2. The MS00:15 and MD00:15 lines are OR tied on the MOS memory module side and form a 16-bit bidirectional bus for that system. For the sake of comparability with the Model 74 and 70 Memory Buses however, they appear as separate uni-directional lines.
 3. There are two additional control lines in the Core System to provide externally generated memory busy and memory access timing for future product expansion.
 4. All logic levels are TTL compatible. A true condition corresponds to less than +0.4 vdc, a false condition corresponds to more than +2.5 vdc.
 5. The custom interface is permitted one TTL standard load and two TTL open collector power gate OR ties (48 milliamperes sink at 0.4 VDC). The customer may not OR tie onto ER0, LR0, INH0, W0, PAR0, MS00:16 and the two extra control lines. The Memory Bus may not be extended physically beyond the Processor and first expansion chassis (about 15 inches total).
 6. The larger number given for the MOS System Timing in this section allows the Processor to have alternate memory cycles. It is further recommended that any DMAC that is to support a transfer rate in excess of 2,400,000 bytes per second, be buffered. This prevents data loss as a result of a refresh cycle, which occurs once every 45 microseconds.

5.5 SELECTOR CHANNEL

The optional Selector Channel provides block data transfer between one of up to 16 I/O devices, and memory. Once initiated, the transfer is independent of the Processor. The program specifies the device address, the type of operation (Read or Write), the starting address in memory, and the final memory address of the transfer. The Selector Channel then completes the transfer, cycle stealing from the Processor, without further direction by the Processor. Upon completion of the transfer, or termination of the transfer due to a fault, the Selector Channel Busy condition is dropped and the Processor is notified via an interrupt.

The Selector Channel operates at a maximum rate of 2,000,000 bytes per second in a Model 74 or 70 system. This peak rate does not allow simultaneous processing. On the Model 80, the Selector Channel can transfer at a maximum transfer rate of 3,150,000 bytes per second. This assumes that the Processor is in the Wait state, not vying for memory, and that the data transfer between the SELCH and the device on the SELCH bus is in the Halfword (16-bit) mode. If a user program is being executed simultaneously a worst case transfer rate of 1,850,000 bytes per second can be achieved. This assumes the CPU will get every other memory cycle.

The DMA port on the Model 80 is capable of transfer rates up to 4,760,000 bytes per second. This transfer rate can only be achieved with multi-Selector Channel configurations, or a customer designed DMAC.

Figure 5-23 is a block diagram of the Selector Channel. Address lines to, and data lines to and from the Memory Bus are shown on the right side. The Memory Bus Control Logic (one of several arbitrary functional groupings used only for purposes of this block diagram) gates an address to the Memory Bus and data to or from the bus depending upon the direction of transfer. The Selector Channel Data Register (DR) stores the 16-bit data halfword to/from memory. The Transfer Control Logic gates the data between the Selector Bus (shown on the bottom) and the Data Register in either 8-bit bytes or 16-bit halfwords, depending on the device. The address circuits are shown in the upper right area. The Final Address Register (FR) is loaded in two bytes from the Multiplexor Bus. The Address Register (AR) is loaded with the starting address in two bytes. After each byte of data transfers to/from the device, the Address

Register is incremented and its contents is compared to the contents of the Final Address Register. When $AR=FR$, a terminate signal is sent to the Transfer Control Logic. If $AR \neq FR$, the next data transfer is initiated to/from the device.

The Multiplexor Bus is shown on the left side of Figure 5-23. Note that the Multiplexor Bus may be gated to any one of six places. The gates are functionally represented by a six position rotary switch. With the gating as shown, and assuming the Transfer Control Logic is also as shown, the Multiplexor Bus including all 16 Data Lines is gated directly to the Selector Bus. This is the condition which exists when the Selector Channel has not been addressed. Thus, all devices on the Selector Channel may be used via the Multiplexor Channel if the Selector Channel is not in use. Four of the remaining five points onto which the Multiplexor Bus may be gated, are the Upper and Lower halves of FR and AR. The sixth point is designated Command and Sense Logic. Commands from the Processor are decoded in this block to produce control signals for the Transfer Control Logic and the Multiplexor Input Control Logic.

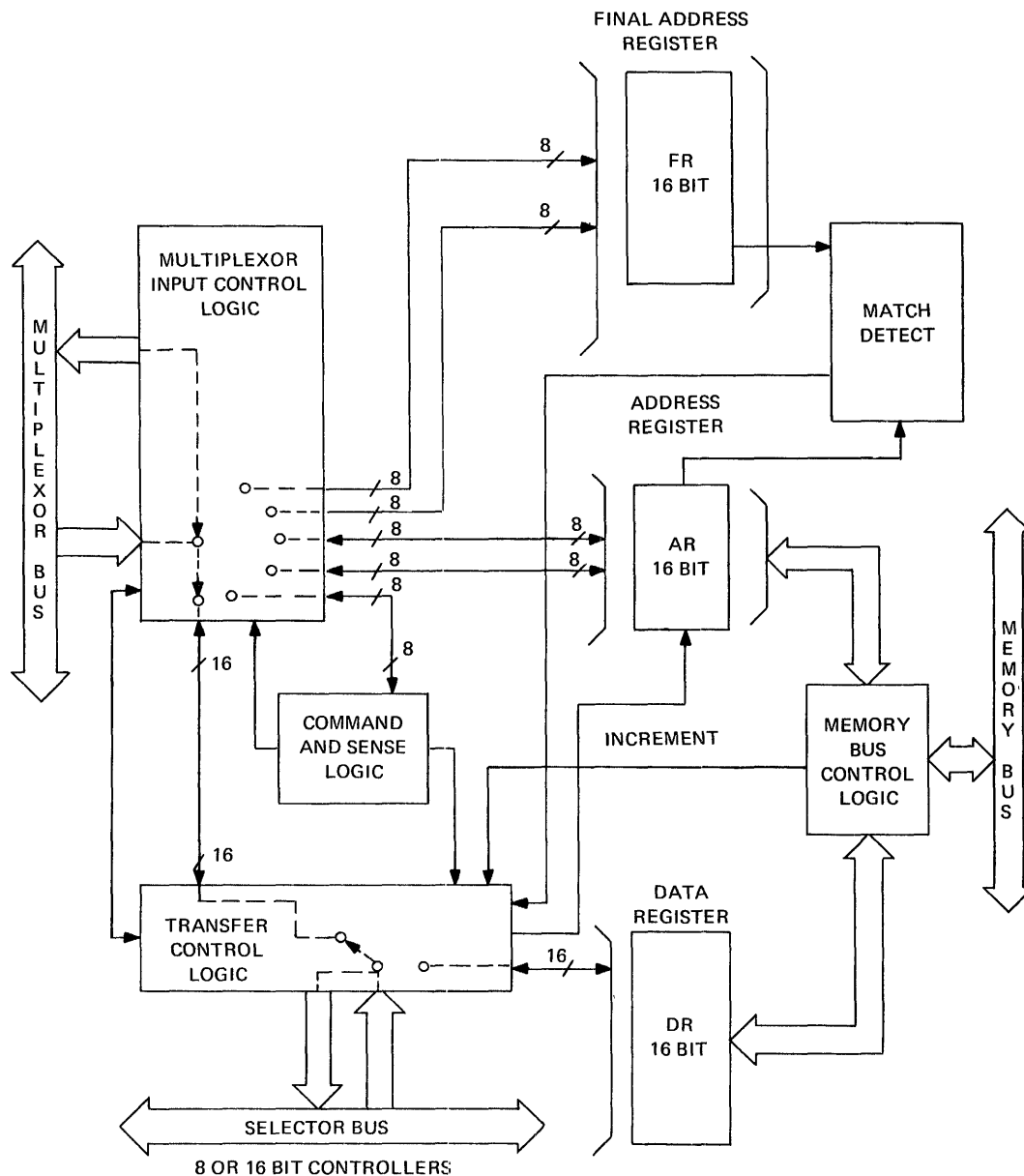


Figure 5-23. Selector Channel, Block Diagram

The following is a typical sequence of operation for a Selector Channel I/O operation. Figure 5-24 is a flow chart of Selector Channel operation. Circled numbers on Figure 5-24 refer to steps in the following sequence:

1. The device controller is addressed and the appropriate command sent to it (for example, Read Tape Forward).
2. The Selector Channel AR and FR are loaded via four byte transfers from the Multiplexor Channel. The FR may be odd or even.

NOTE

Steps 1 and 2 may be reversed.

3. A command which specifies if this is an input operation (information received from the device) is sent to the Selector Channel from the Multiplexor Channel. (The Selector Channel is initialized and returns to the output state - information to the device - whenever a Read Data or Write Data instruction is issued to the Selector Channel.)
4. A GO Command which starts the transfer operation is sent from the Multiplexor Channel to the Selector Channel.

NOTE

The Processor is now free to continue its program while the block I/O transfer is performed by the Selector Channel on a cycle-stealing basis. Steps 5 through 8 apply solely to Read operations (memory to device). Steps 9 through 12 apply to Write operations (device to memory).

5. If this is a Read operation, the Selector Channel requests memory service via the built-in memory port in the Processor. When service is granted, the Selector Channel fetches a halfword from memory.
6. When memory data becomes available, it is gated to the DR.
7. The Status Byte from the device is examined. If the device Busy Bit (Bit 4) is true, the Selector Channel waits for it to become false. If any of the Status Bits 5, 6, or 7 are true, the transfer is terminated.
8. Data (either an 8-bit byte or a 16-bit halfword depending on the controller) is transferred from the DR to the device when the device is not busy. If AR=FR, the transfer is terminated. If not, the AR is incremented, and if there is a carry (AR=0) the transfer is terminated. If the contents of the AR is even another byte is transferred to the device.
9. If this is a Write operation, the Status Byte is input from the device. If the Busy is true, the Selector Channel waits for it to become false. If any of the other three bits in the status code are true, the transfer sequence is terminated. If all bits are false, the sequence continues.
10. Data is transferred from the device to the DR when the device is not busy. If AR=FR, the transfer is terminated. If not, the AR is incremented, and if there is a carry (AR=0) the transfer is terminated.
11. If the contents of the AR is even, another byte is transferred to the device. (The sequence returns to Step 9.)
12. The Selector Channel requests memory service.
13. The halfword in DR is written into the addressed memory location when granted memory.

Steps 14 through 16 describe the termination sequence. Any of the following conditions will cause termination:

- a. AR=FR
- b. AR increments to Zero (carry out of AR).
- c. A status failure from the device (EX, EOM, or DU).
- d. A Stop Command from the Processor.

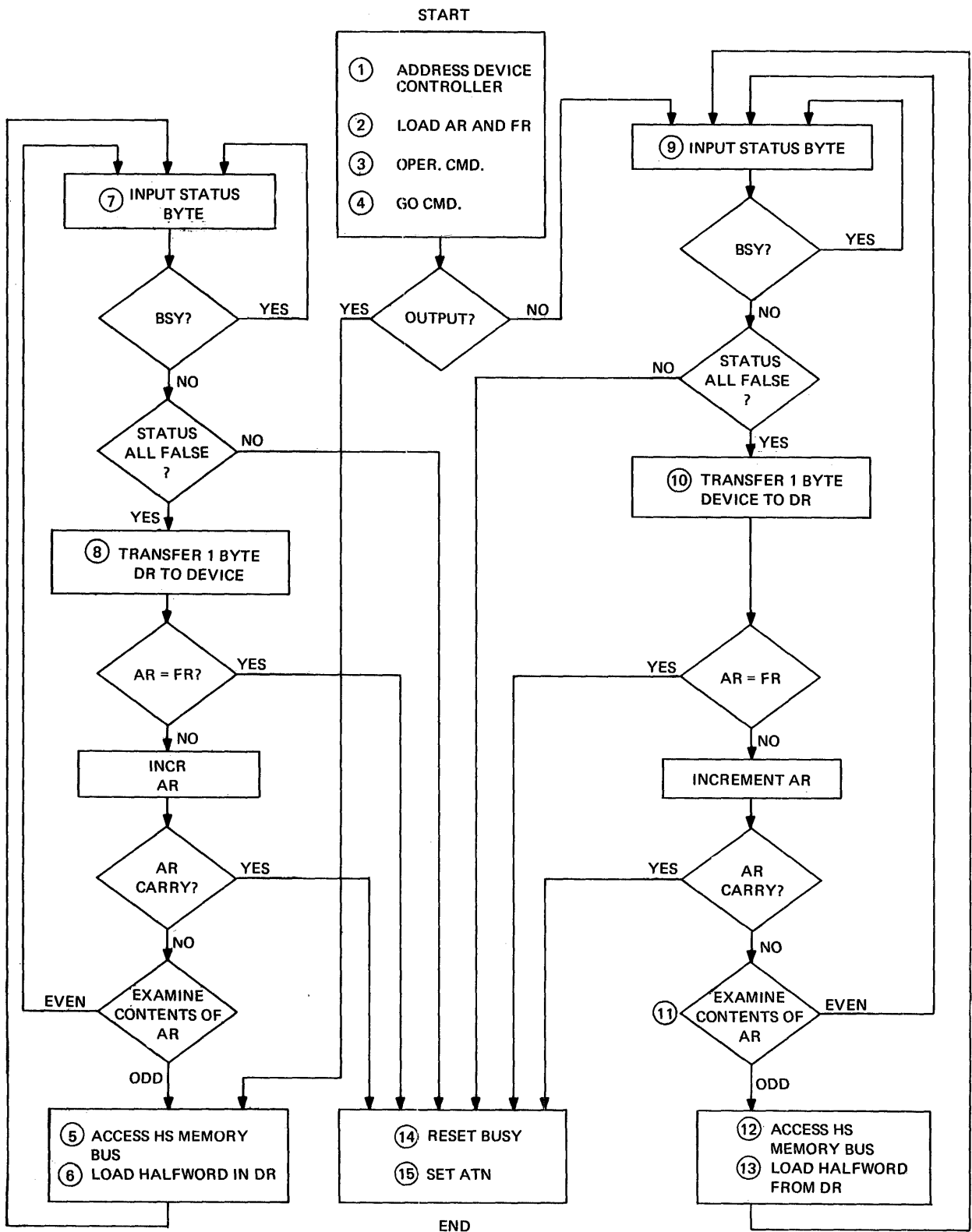


Figure 5-24. Selector Channel, Flow Chart .

NOTE

If the Selector Channel is in a memory cycle when the Stop Command is received from the Processor, execution of the Stop Command will be delayed until the completion of the memory cycle.

14. Reset the Selector Channel Busy indication.
15. Set the Selector Channel Attention flip-flop to generate an interrupt to the Processor.
16. After the Processor acknowledges the interrupt and addresses the Selector Channel, it will send a Status Request to the Selector Channel. The status received will be the status of the active device on the SELCH Bus with the Busy Bit forced to zero. To determine whether the transfer terminated normally, the contents of AR may be Read by issuing two Data Requests to compare its contents with the final memory address.

The Selector Channel is complete on one mother-board which is mounted in any even numbered Universal Expansion Slot.

5.6 GENERAL PURPOSE INTERFACE

INTERDATA provides the user with two general purpose modules to aid in the customer design effort. The Universal Interface Module is a general purpose 16-bit Input/Output interface to the Multiplexor Bus. It is on a 7" x 15" printed circuit board. The General Purpose Interface Board allows the user to easily mount his integrated circuits and other components, to develop a prototype design.

5.6.1 Universal Interface Module

The Universal Interface Module (UIM) provides a fully buffered, byte or halfword (16-bit) oriented interface module for data transfers between the Processor Multiplexor Bus and byte or halfword oriented equipment provided by the user. The UIM is a transparent interface in that the function of any I/O instruction issued to the module depends upon the user's equipment. Data transfer rates can approach 100,000 bytes per second (100KBS) depending on the user's equipment.

Operational Characteristics

This module contains the following major functions:

Sixteen Data Output Lines

Sixteen Data Input Lines

Four Command Lines

Eight Status Input Lines

One Interrupt Line to the Processor

One System Clear Line to the User

Data can be transferred between the Universal Interface Module and the Multiplexor Bus in an eight bit byte mode (two eight bit bytes per halfword) or in a sixteen bit parallel (halfword) mode. The data transfer mode is set up under program control. The interrupt line can be disabled, enabled, or disarmed within the Universal Interface Module under program control.

The sixteen Data Output Lines are used in two modes of operation:

In the eight-bit byte mode, the first of two Write Data instructions places a data byte onto the most significant Data Output lines. The second places the second byte onto the least significant Data Output Lines.

In the Halfword Mode, 16 bits of data are placed simultaneously onto the Data Output lines. The sixteen Data Output lines are latched with flip-flops.

The Sixteen Data Input lines are also available in two modes of operation:

In the eight-bit byte mode, the first of two Read Data instructions read a data byte from the most significant Data Input lines and loads it onto the Multiplexor Bus. The second instruction reads a second data byte from the least significant Data Input lines and loads it onto the Multiplexor Bus.

In the halfword mode, all 16-bits on the Data Input lines are loaded onto the Multiplexor Bus.

The Output Command instruction results in the following action:

The two most significant bits of the command byte disable, enable, or disarm the interrupt logic.

The third bit of the command byte determines the data transfer mode. That is, byte mode or halfword mode. The fourth bit is not used.

The four least significant bits of the command byte are loaded onto the Command Lines to the user. These four lines are latched by flip-flops. These lines are ordinarily used to provide Data Ready or Data Request signals to the user's logic.

Eight Status Input lines are also available to the user. The user connects the desired lines to eight status input lines. To read in the eight status lines, the program executes a Sense Status instruction which gates the eight status lines into the Processor like a status byte.

Specifications

Universal Interface Module, INTERDATA Product No. M48-009

Dimensions - 7" x 15"

Weight - 2 pounds

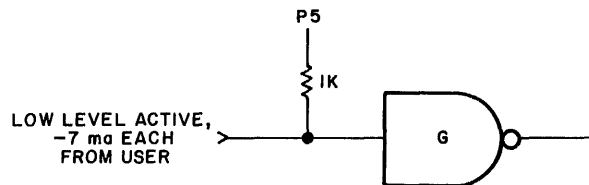
Power Requirements - +5.0 Vdc 0.85 amps

Installation

The Universal Interface Module may be installed in either the right or left half position of a standard 15" chassis, depending on the system configuration. The module also includes a 50 wire, 28 gauge, stranded, 10 foot, open ended cable. This device is not intended for use with a Selector Channel.

Inputs:

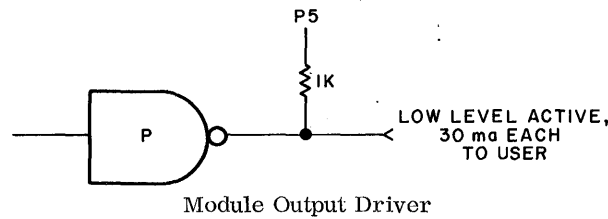
The 16 Data Input lines, 8 Status Input lines, and 1 Interrupt line are terminated on the 7" module as shown below.



Module Input Termination

Outputs:

The 16 Output lines, 4 Control lines, and 1 Initialize line from the module are driven as shown below. Each output can sink 30 ma.



5.6.2 General Purpose Interface Board

The General Purpose Interface Board (M48-002) requires one circuit board slot position in an INTERDATA 15 inch chassis. The board measures 15 inches square and can accommodate up to 117 Integrated Circuit Modules (IC's). Either 14 or 16 pin Dual in Line Packs (DIP's), can be accommodated. The IC's are usually soldered directly into their designated board locations. If removable or interchangeable IC's are desired, commercially available IC sockets, with 300 mil mounting centers, can be soldered into the IC board locations by the user.

The General Purpose Wire Wrap board provides locations for a total of 416 axial lead components (resistors, capacitors, and diodes) mounted to the left (Positions 1 and 2) and to the right (Positions 3 and 4) of the IC packs. See Figure 5-25. All discrete components are soldered into the board. When the components are machine mounted, Position 2 in a field containing a 14 pin IC pack and Positions 2 and 3 in a field containing a 16 pin IC pack, can not be used due to the proximity of the components, board locations and the installed IC packs. Care should be exercised when manually installing components in Positions 2 and 3 of a field containing an IC pack.

The circuit board contains 25 mil square wire wrap posts which correspond directly to the IC pins, component leads, and cable connections. All interconnections between components and the connectors are made by wire wrapping to the 25 mil square wire wrap posts. See Figure 5-26.

5.6.2.1 Component Field Numbering

The wire wrap board is divided into 117 fields. Each field can accommodate one IC (fixed or socket) and up to four other components. Refer to Figure 5-25 for IC field numbers and component location designation.

Use Field A8 as an example (refer to Figure 5-27). The component designation takes the form A8-1TR, and A8-1BR; where A8 is the board field, 1 indicates component location one (as viewed from the Apparatus Side), T designates the top pin, B designates the bottom pin, and R indicates a resistor. The IC pack pin orientation is bottom left to right, Pins 1-8; top right to left, Pins 9-16.

An example of a power gate mounted directly on a circuit board is shown in Figure 5-28. Note that the pin numbers correspond directly with the actual IC pin number. Note further that the IC is located in Field A8 and the diode is in component Position 1 of that field.

5.6.2.2 Connector Layout

Figure 5-29 shows the pin numbering system for the two standard 84 pin back panel connectors, and the two standard 50 pin cable connectors. The fourth and fifth digits indicate the board number, 00-07. The sixth and seventh digits specify the connector field 00 (zero), or 01 (one). INTERDATA uses the symbol (see Figure 5-29) indicated for a back panel connection for schematic drawings.

5.6.2.3 Available Cables

INTERDATA provides two cables for use with this board. Cable 17-179 is a 10 foot, open ended cable with 25 twisted pair, 28 ga. stranded, and is useful for connecting to user circuits. Cable 17-178 is a 3 1/2 foot, 25 twisted pair, 28 ga. stranded cable, suitable for interconnecting more than one GPWW board. Cables are purchased separately.

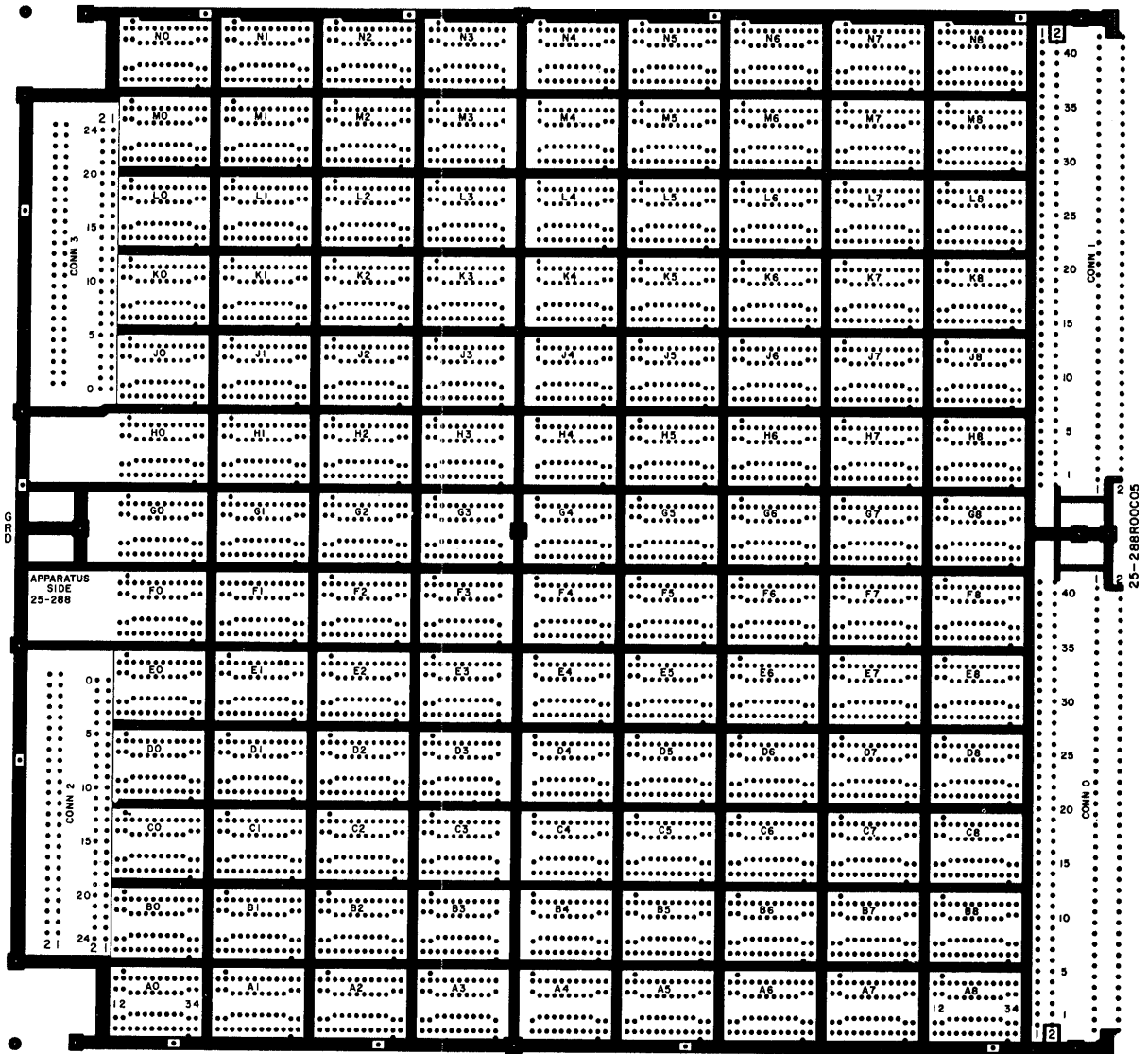


Figure 5-25. General Purpose Wire Wrap Board

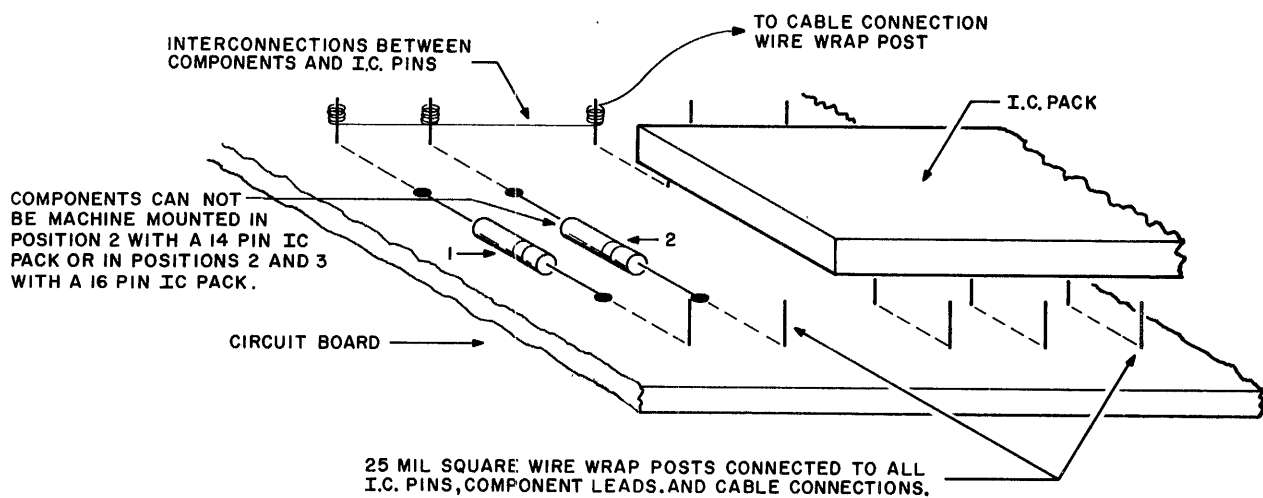


Figure 5-26. Wire Wrap Posts/IC and Component Leads Connection

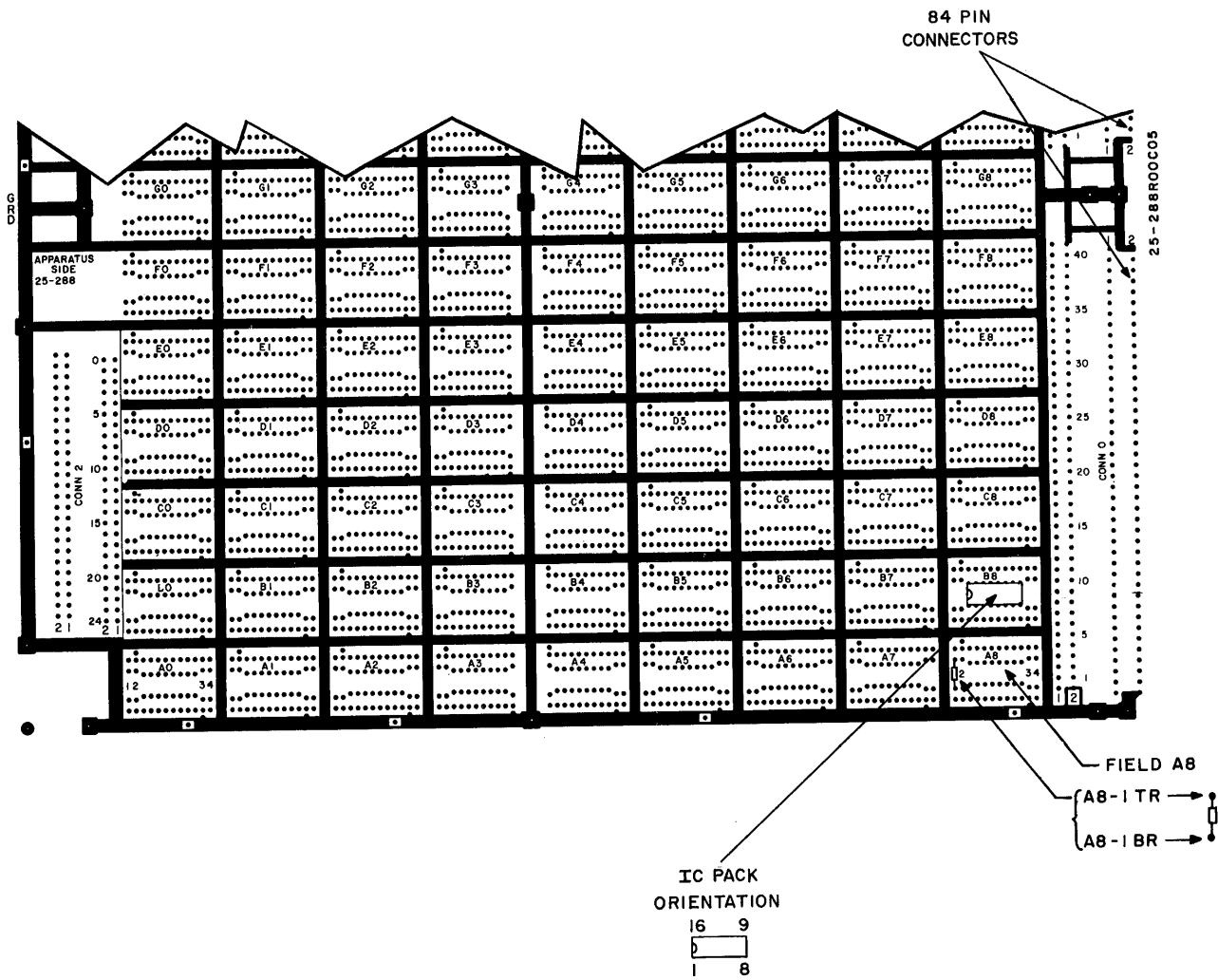


Figure 5-27. IC Circuit Board Numbering System

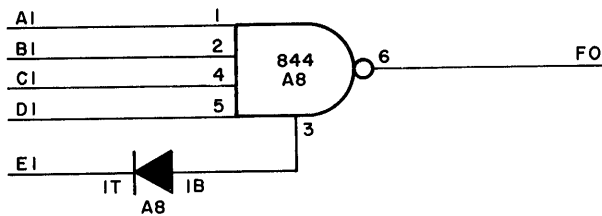
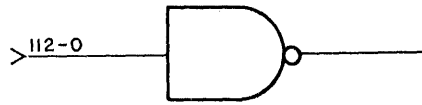


Figure 5-28. Example of a Power Gate



SYMBOL FOR
BACK PANEL CONNECTION

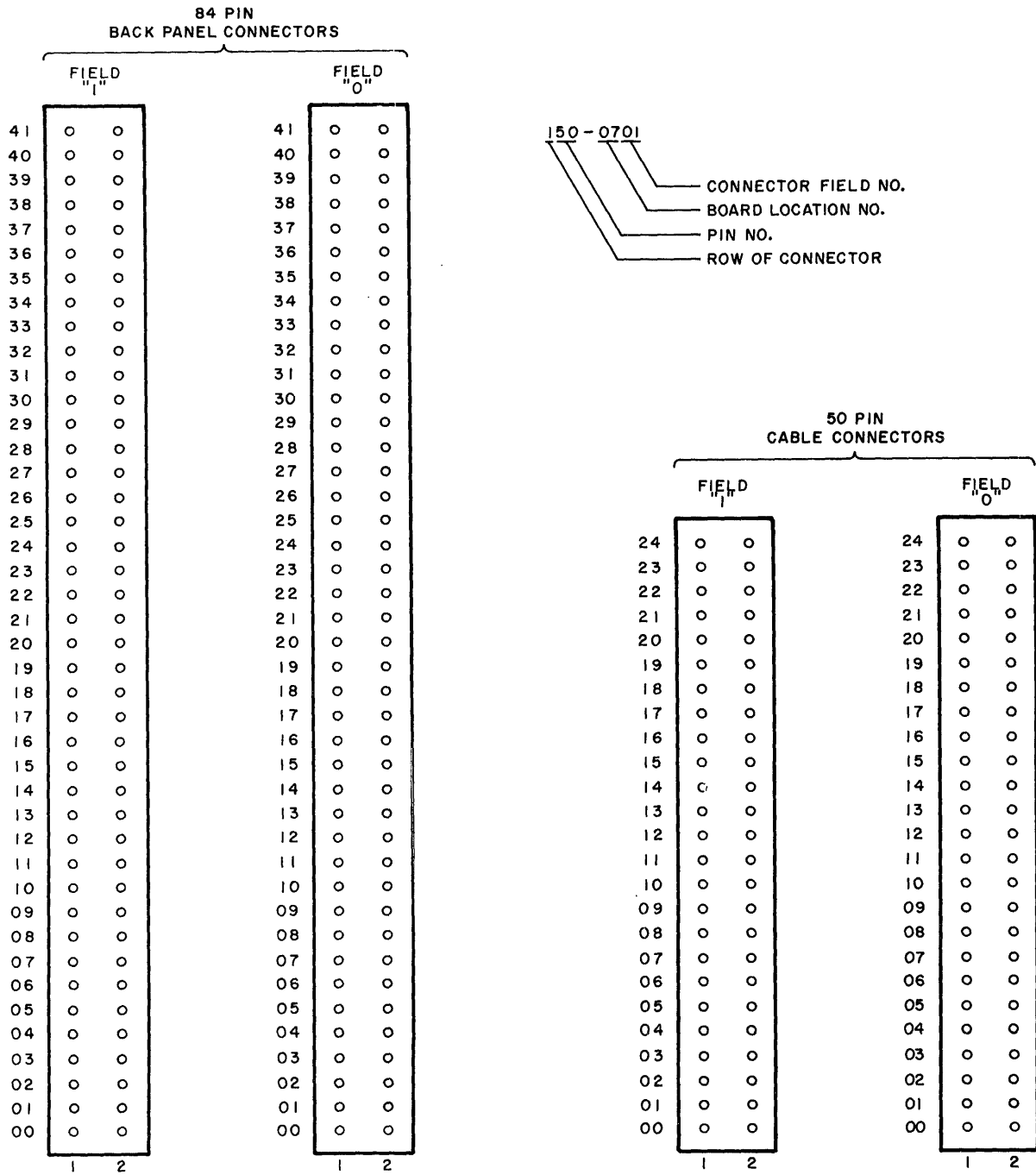


Figure 5-29. General Purpose Interface Board Connector Layout

CHAPTER 6

CONTROL CONSOLE

6.1 INTRODUCTION

This chapter describes the Control Console which operates with the INTERDATA Processors. The Control Console provides the system operator with visual indications of the state of the Processor as well as manual control over the system

6.2 CONTROL CONSOLE DESCRIPTION

The Control Console, shown in Figure 6-1, is a RETMA standard 5 1/4" x 19" panel which is plug removable from the Processor. It displays the current state of the Processor and provides all necessary manual controls for the system. The Control Console includes the following control and display elements:

Indicators	- Power ON Lamp
	- Wait Lamp
	- Display Register D1 - 16 Bits (top)
	- Display Register D2 - 16 Bits (bottom)
Data Switches	- Sixteen Data/Address Switches
Control Switches	- Key operated OFF-ON-LOCK, Security Lock Switch
	- Initialize Switch (INT)
	- Execute Switch (EXE)
Function Switches	- Single Switch (SGL)
	- Run Switch (RUN)
	- Twelve Position Rotary Function Switch

A functional description of each of these control and display elements is given in this section.

6.2.1 Key Operated Security Lock

This is a three-position, OFF-ON-LOCK, key-operated locking switch, which controls the primary power to the system. This switch can also disable the Control Console, thereby preventing any accidental manual input to the system. The POWER indicator lamp associated with the key lock switch is located in the upper left corner of the Control Console. The POWER lamp is lit when the key lock switch is in the ON or LOCK position. The relationship between the key lock switch positions, primary power, and the Control switches is:

OFF	The primary power is Off.
ON	The primary power is On and the Control switches are enabled.
LOCK	The primary power is On and the Control switches are disabled.

6.2.2 Control Switches

The Control switches on the Control Console are active only when the key-operated locking switch is in the ON position. The function of each of these switches is as follows:

INITIALIZE (INT)	The momentary Initialize (INT) switch causes the system to be initialized. After this initialize operation, all device controllers on the system Multiplexor Bus are cleared and certain other functions in the Processor are reset. Initialize is not disabled when the key-switch is in the lock position.
------------------	--

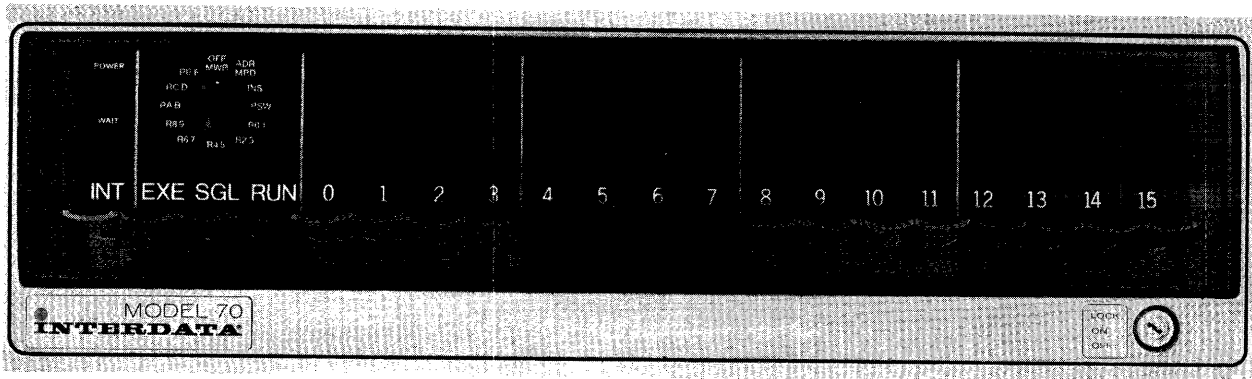


Figure 6-1. Control Console

EXECUTE (EXE) The momentary Execute (EXE) switch causes the Processor to perform the Control Console operation selected by the Function switches, as discussed in Section 6.2.3.

6.2.3 Function Switches

The three Function switches, SINGLE (SGL), RUN (RUN), and the 12 position rotary Function switch are used to place the Processor in various operating modes. The Processor is controlled by setting the function switches in the proper positions and then depressing the Execute (EXE) switch to activate the function. The various Processor modes, and the method of entering each mode, are as follows:

ADDRESS

- To enter the ADDRESS Mode
 - a) the 12 position Function switch must be in the ADR/MRD position
 - b) the SGL switch must be UP
 - c) the RUN switch must be UP

The desired address is specified on the 16 Data/Address switches. When the Execute (EXE) switch is depressed, the specified address is placed into the address portion of the Current Program Status Word PSW (16:31). (When the address is transferred from the Data/Address switches, the least significant bit (PSW31) is cleared so that the resulting address is always even.) The complete PSW is displayed on Display Register D1 (top) and Display Register D2 (bottom). This address can be used to read data from memory, to write data to memory, or to start the execution of a program.

MEMORY READ

- To enter the MEMORY READ Mode
 - a) the 12 position Function switch must be in the ADR/MRD position
 - b) the SGL switch must be DOWN
 - c) the RUN switch must be UP

When the EXE switch is depressed, the data read from memory is displayed on the Display Register D2 (bottom) indicator lamps, and the address incremented by 2 of that data is displayed on the Register Display D1 (top) lamps. The address portion of the PSW is also incremented by 2. Depressing EXE repeatedly displays consecutive locations from memory.

MEMORY WRITE

- To enter the MEMORY WRITE Mode
 - a) the 12 position Function switch must be in the OFF/MWR position
 - b) the SGL switch must be DOWN
 - c) the RUN switch must be UP

The desired word to be written is specified on the 16 Data/Address switches. When the EXE switch is depressed, the data written to memory is displayed on the Display Register D2 (bottom) lamps, and the address incremented by 2 of that data is displayed on the Register Display D1 (top) lamps. The address portion of the PSW is also incremented by 2. Depressing EXE repeatedly writes data from the Data/Address switches into consecutive memory locations.

- RUN**
- To enter the RUN Mode
 - a) the 12 position Function switch may be in any position except OFF/MWR or ADR/MRD
 - b) the SGL switch must be UP
 - c) the RUN switch must be DOWN
- When the EXE switch is depressed, the Processor begins program execution. In the RUN Mode, the Display Registers are idle. The program can use the Control Console as an I/O device. If the program does not output to the Display Registers, these registers retain the last value displayed prior to entering the RUN Mode.

- SINGLE**
- To enter the SINGLE Mode
 - a) the 12 position Function switch may be in any position except OFF/MWR or ADR/MRD
 - b) the SGL switch must be DOWN
 - c) the RUN switch must be DOWN
- This mode allows programs to be executed, one instruction at a time, each time the EXE switch is depressed. The Display Registers 1 and 2 contain various information specified by the 12 position Function switch after each EXE as follows:
- INS - the next instruction to be executed is displayed. The first halfword from memory is shown in Display Register D1 (top) and the second halfword from memory is shown in Display Register D2 (bottom).
 - PSW- the Current Program Status Word is displayed. The Program Status and the Condition Code is shown in Display Register D1 and the Location Counter is shown in Display Register D2.
- | | | |
|------|---|--|
| R0:1 | } | the pair of general registers specified by the switch is displayed. For example: if the switch is in the R8:9 position, General Register 8 is displayed on Display Register D1 and General Register 9 is displayed on Display Register D2. |
| R2:3 | | |
| R4:5 | | |
| R6:7 | | |
| R8:9 | | |
| RA:B | | |
| RC:D | | |
| RE:F | | |

- HALT**
- To enter the HALT Mode
 - a) the 12 position Function switch may be in any position except OFF/MWR or ADR/MRD.
 - b) the SGL switch may be UP or DOWN (don't care)
 - c) the RUN switch must be UP
- When the EXE switch is depressed, the Processor enters a HALT Mode which is non-interruptable. Any of the information listed above under the SINGLE Mode can be displayed on Display Register D1 and D2 by placing the 12 position Function switch in the proper position and depressing the EXE switch.

When the Processor is in the HALT Mode, the WAIT indicator on the Control Console lights. The WAIT indicator also lights between instructions when the Processor is in the SINGLE Mode. Finally, the WAIT indicator lights when an executing program enters the WAIT state by setting Bit 0 of the PSW. Note that the program initiated Wait state is interruptable, while the console initiated HALT Mode is not interruptable.

6.3 CONTROL CONSOLE OPERATING PROCEDURES

6.3.1 Power Up

To power up and initialize (clear) the system:

1. Turn the key-operated security lock clockwise from the OFF position to the ON position.
2. Depress the momentary Initialize (INT) Control switch, to initialize the system.

This action provides electrical power to the system, and leaves the Processor in the HALT Mode. It is recommended that before the system is used, a few important pointers and New PSWs in core memory be initialized. The locations in memory to be adjusted are shown in Table 6-1.

TABLE 6-1
CORE MEMORY INITIALIZATION

Location (hex)	Function	Suggested Setting	Comment
0022	Pointer to register save area	0058	This pointer should contain the address of a block of 32 bytes which are available for register save and restore operations.
0034 0036	New PSW for Illegal Instruction Interrupts	8000 0050	If an Illegal instruction occurs, this New PSW clears all interrupts and puts the Processor into Wait state with Location Counter = 0050.
003C 003E	New PSW for Machine Malfunction Interrupts	8000 0050	This New PSW treats Machine Malfunction Interrupts the same as Illegal instructions for purposes of initialization.
0050 0052 0054 0056 0078	Auto-Load sequence for loading programs	D500 00CF 4300 0080 XXYY	This sequence uses the Auto-Load instruction at 50 followed by an Unconditional Branch to 80 to perform initial program loads. With this sequence location 78 should be loaded with Device Number XX and Command Byte YY. Refer to Appendix 8 for information on I/O devices.

The memory locations mentioned previously can be set using Memory Write operations as follows:

1. Enter '0022' (0000 0000 0010 0010) into the Data/Address switches, release the SGL switch, set the rotary Function switch to ADR, and depress the EXE switch.
2. Enter '0058' (0000 0000 0101 1000) into the Data/Address switches, depress the SGL switch, set the rotary Function switch to MWR, and depress the EXE switch. This enters value '0058' into location '0022'.
3. Enter '0034' in the Data/Address switches, release the SGL switch, set the rotary Function switch to ADR, and depress the EXE switch.
4. Enter '8000' into the Data/Address switches, depress the SGL switch, set the rotary Function switch to MWR, and depress the EXE switch.
5. Enter '0050' into the Data/Address switches, and depress the EXE switch. These steps enter values '8000' and '0050' into memory starting at '0034'.
6. Follow similar steps until all specified locations in memory have been set properly.

Once these locations are set, their contents can be verified using Memory Read operations as follows:

1. Enter '0022' into the Data/Address switches, release the SGL switch, set the rotary Function switch to ADR, and depress the EXE switch.
2. Depress the SGL switch, set the rotary Function switch to MRD, and depress the EXE switch. At this point, the address ('0024') should be displayed in Register Display D1, and the contents of '0022' ('0058') should be displayed in Register Display D2.

3. Enter '0034' into the Data/Address switches, release the SGL switch, set the rotary Function switch to ADR, and depress the EXE switch.
4. Depress the SGL switch. set the rotary Function switch to MRD, and depress the EXE switch. At this point, Register Display D1 should show the address ('0036') and Register Display D2 should show the contents of '0034' ('8000').
5. To examine the next location, depress the EXE switch. At that time, Register Display D1 should show the address ('0038'), and Register Display D2 should show the data ('0050').
6. Follow similar steps until all appropriate locations have been verified.

Once memory locations are set and verified, it is still necessary to initialize the Current PSW. Note that while the Location Counter [PSW (16:31)] can be set using the ADR/MRD position of the rotary Function switch, there is no way to directly adjust the program status [PSW (0:15)] from the Control Panel. When power is turned on, the program status is loaded from memory location 0024, the PSW save area. Following a cold start, this initial setting is arbitrary. The program status can be set in two ways: either by executing an LPSW or EPSR instruction, or by servicing an interrupt with a PSW swap. For system initialization, the recommended procedure is to execute an Illegal instruction, which forces the Illegal instruction PSW swap. Using core memory settings suggested above, this PSW initialization can be performed by starting program execution at location '0034', which is an Illegal instruction. The specific steps are:

1. Enter '0034' into the Data/Address switches, release the SGL switch, set the rotary Function switch to ADR, and depress the EXE switch.
2. Depress the RUN switch, set the rotary Function switch to a position other than OFF/MWR or ADR/MRD, and depress the EXE switch.

The result of performing these two steps is that the Processor attempts to execute the contents of location '0034' ('8000') which is an Illegal instruction. An Illegal instruction PSW swap occurs, which loads the program status with '8000', loads the Location Counter with 50, and leaves the Processor in the Wait State. At this point, if the EXE is depressed, the Processor executes the Auto-Load sequence at '0050'.

6.3.2 Power Down

To shut down power to the system,

1. Place the Processor in the HALT Mode as described in Section 6.2.3.
2. Turn the key-operated security lock to the OFF position.

This removes AC power from the system and forces a Primary Power Fail Interrupt to the Processor. Refer to 6.4.4 for further details.

6.3.3 Program Loading

There are many ways to load the memory with programs and/or data. Most programs are loaded using one of the program loaders associated with the system software. Refer to Chapter 11 for details on loaders and other software programs.

The Auto-Load sequence, referred to in the previous section, is useful for loading programs when the system is being initially loaded, or when no other program loaders are in memory. The sequence recommended in the previous section is described below:

TABLE 6-2
AUTO LOAD SEQUENCE

Location	Contents	Instruction		
50	D500 00CF	AL	X'CF'	AUTO LOAD
54	4300 0080	B	X'80'	BRANCH TO 80
78	XXYY	DC	X'XXYY'	DEV NO AND CMND

This sequence is based on the Auto-Load instruction, which is described in Section 5.3.10. This instruction reads eight-bit data bytes from Device XX into memory, starting at location 0080. The load operation proceeds until the device indicates a termination status, or until a specified upper limit is reached. In the sequence above, the limit is defined as location '00CF', which allows 80 bytes to be read. With the Auto-Load instruction, the device address to be used is specified in byte location '78', and the command byte which starts the device is specified in byte location '79'. Leading zero data bytes are skipped and not loaded. This sequence is appropriate with a Teletype or a paper tape reader which transfers eight-bit data bytes. When the Auto-Load instruction terminates, the Branch instruction transfers control to location '0080'. This Auto-Load sequence can be easily changed to meet other requirements. The upper limit (00CF), at 0052, can be changed to load programs of different length. The transfer address ('0080'), at '0056', can be changed to branch to a different location. Following the Auto-Load instruction, it is possible to test the Condition Code to determine exactly how the Auto-Load operation terminated. An all zero Condition Code implies that the specified program length was loaded. A non-zero Condition Code implies that the device terminated the load sequence before the program length was satisfied.

6.3.4 Program Execution

To begin the execution of a program, the system must be in the Halt Mode.

1. Set the rotary Function switch to ADR/MRD.
2. Release the SGL switch.
3. Enter the program starting address in the 16 Data/Address switches.
4. Depress the momentary EXE switch.
5. Set the rotary Function switch to a position other than OFF/MWR or ADR/MRD.
6. Depress the RUN switch.
7. Depress the momentary EXE switch. The Processor is now in the RUN Mode.

To execute a program in the Single Step Mode—one instruction at a time—the system must be in the HALT Mode.

1. Set the rotary Function switch to ADR/MRD.
2. Release the SGL switch.
3. Enter the program starting address in the 16 Data/Address switches.
4. Depress the EXE switch.
5. Depress the SGL switch and the RUN switch.
6. Set the rotary Function switch to select the register(s) desired for display. (Must be a position other than OFF/MWR or ADR/MRD.)

7. Depress the EXE switch to execute one instruction.
8. Repeat Step 7. to execute successive instructions.

At any time during the single-step sequence, the rotary Function switch can be adjusted to change the selection of registers to be displayed. To examine more than one register pair without executing more instructions, release (raise) the RUN switch. This leaves the Processor in the HALT Mode. Then select the registers to be displayed with the rotary Function switch, and depress the EXE switch to observe those registers. Depress the RUN switch to resume single-step execution. Note that the system is not responsive to I/O Interrupts while in the Single Step Mode.

6.3.5 Program Termination

To manually halt the execution of a program,

1. Set the rotary Function switch to a position other than OFF/MWR or ADR/MRD.
2. Release the RUN Switch.
3. Depress the momentary EXE switch.

When the Processor enters the HALT Mode, the Register Displays are updated as specified by the rotary Function switch. For example; if the EXE switch is depressed with the rotary Function switch set at the PSW position, the execution is halted and the Current PSW is displayed on the Register Displays. Each time the EXE switch is depressed, while in the HALT Mode, the Register Displays are updated as specified by the rotary Function switch. Therefore, to display different register pairs once the Processor is in the HALT Mode, change the rotary Function switch setting and depress the EXE switch.

6.3.6 Manually Initiated Memory Operations

6.3.6.1 Memory Read

To display the contents of memory locations on the Register Displays the Processor must be in the HALT Mode.

1. Set the rotary Function switch to the ADR/MRD position.
2. Release the SGL switch.
3. Enter the memory read starting address in the 16 Data/Address switches.
4. Depress the EXE switch.
5. Depress the SGL switch.
6. Depress the EXE switch.
7. The address read from, plus two, appears in Register Display D1. The data read from memory appears in Register Display D2.
8. Repeat from Step 6. to read successive memory locations. The Location Counter is automatically incremented each time the EXE switch is depressed.

6.3.6.2 Memory Write

To write data from the Data/Address switches into memory the Processor must be in the HALT Mode.

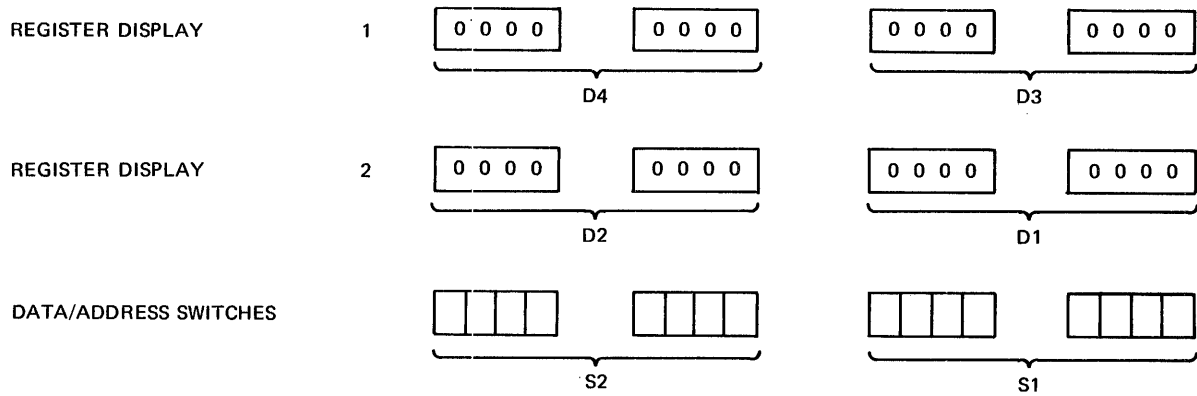
1. Set the rotary Function switch to the ADR/MRD position.
2. Release the SGL switch.
3. Enter the memory write starting address in the 16 Data/Address switches.
4. Depress the EXE switch.
5. Set the rotary Function Switch to the OFF/MWR position.
6. Depress the SGL switch.

7. Enter the data to be written in the 16 Data/Address switches.
8. Depress the EXE switch.
9. The address written into plus two appears in Register Display D1. The data written into memory appears in Register Display D2.
10. Repeat from Step 7. to write into successive memory locations. The Location Counter is automatically incremented each time the EXE switch is depressed.

6.4 PROGRAMMING CONSIDERATIONS

6.4.1 Control Console I/O

The Control Console is available to any running program as an I/O device with Device Address 01. The status and command bytes for the Control Console are summarized in Appendix 8. The status byte indicates the setting of the Function switches. The command byte specifies either Normal or Incremental Mode, which pertains to data transfers. In the Normal Mode, the selection logic which determines which half of the Data/Address switches and which byte of the Register Displays is transferred is reset every time the Control Console is addressed on the Multiplexor Bus. The Control Console is addressed by every I/O instruction using Device Address 01. Subsequent Read or Write instructions transfer subsequent bytes as shown in Figure 6-2. Normal I/O instructions, therefore, can be used to input data from the Data/Address switches, and output data to the Register Displays.



Instructions Executed (RR or RX)	Data Transferred	
	Normal Mode	Incremental Mode
RD	S1	S1
RD	S1	S2
RD	S1	S1
RD	S1	S2
RH	S1, S2	S1, S2
RH	S1, S2	S1, S2
RB*	S1, S2, S1, S2	S1, S2, S1, S2
WD	D1	D1
WD	D1	D2
WD	D1	D3
WD	D1	D4
WH	D1, D2	D1, D2
WH	D1, D2	D3, D4
WB*	D1, D2, D3, D4	D1, D2, D3, D4

* RB and WB instructions can only be used if the rotary Function switch is in the OFF/MWR or ADR/MRD position assuring zeros in Status Bits 4 through 7. Appendix 8.

Figure 6-2. Control Console Data Transfers

6.4.2 Console Interrupt

In the Processor, an interrupt can be generated from the Control Console as follows:

1. The program must have Bit 4 of the Current PSW set, which specifies Automatic I/O Service Mode.
2. The operator must have the rotary Function switch in the OFF/MWR position, the SGL switch released, the RUN switch depressed, and then depress the EXE switch.

This feature enables an operator to inform the running program that some operator service or function is needed. No acknowledgement of the interrupt is needed by the running program. If the Automatic I/O Service Mode is not enabled, console interrupts are not generated and are not queued.

6.4.3 Wait State

The running program can put the Processor into the Wait State by setting Bit 0 of the Current PSW. The operator is informed of this action by the WAIT indicator lamp lighting. The Processor can leave the Wait State and resume execution in two ways:

1. An interrupt can occur, causing a PSW swap and the execution of a routine to service the interrupt. When the routing restores the original PSW, the Wait State will be re-established.
2. The operator places the Processor into the RUN Mode as described in Section 7.2.3, which causes the execution to resume at the address specified by the Location Counter [PSW (16:31)] .

Note that the use of the programmed Wait State must be considered carefully when using Single-Step Mode. That is, with single-step execution it is possible to inadvertently "step through" the Wait State, through rapid or continuous use of the EXE switch.

6.4.4 Power Fail

Depressing the INT switch of the Control Console deactivates the system clear relay (SCLR0) in the system for a brief interval (approximately one-second). When this happens, the Processor saves the Current PSW in memory locations 0024 and 0026, and the 16 General Registers in the block indicated by the contents of 0022, and then follows an orderly shut-down sequence, which preserves the data within memory. When the relay is reactivated, the PSW and General Registers are reloaded from memory, and the device controllers are all initialized. At this point, the Processor interrogates the Control Console. If the Function switches are not set up to enter the RUN Mode, the Processor enters the HALT Mode and the WAIT lamp is illuminated. If the Function switches are set up to enter the RUN Mode, the Processor examines Bit 2 of the restored PSW. If PSW Bit 2 (the Machine Malfunction Interrupt Enable) is set, the Processor then performs the appropriate PSW swap for Machine Malfunction Interrupts. The purpose of this interrupt is to inform the running program that a power failure/restore has occurred. If Bit 2 of the PSW is not set, program execution resumes where it left off, with no Machine Malfunction Interrupt.

The use of the INT switch should be considered carefully when Bit 2 of the Current PSW is enabled.

CHAPTER 7

INTERDATA SOFTWARE FAMILY

7.1 INTRODUCTION

A comprehensive family of software products is available for use on INTERDATA Processors. The INTERDATA software family is outlined in Figure 7-1.

- OPERATING SYSTEMS
 - BASIC OPERATING SYSTEM
 - DISC OPERATING SYSTEM
 - REAL TIME OPERATING SYSTEM
- COMPILERS
 - INTERACTIVE FORTRAN
 - ANSI STD FORTRAN (FORTRAN IV)
- ASSEMBLERS
 - CONDITIONAL ASSEMBLIES
- UTILITIES
 - LOADERS, DUMPS, MEMORY CHANGE, DEBUG
 - TEXT EDITOR

Figure 7-1. INTERDATA Software

This family includes three powerful operating systems to simplify program development and implementation. The Basic Operating System (BOSS) and Disc Operating System (DOS) are batch oriented systems, while the Real Time Operating System (RTOS) is a comprehensive multi-programming, foreground/background system. All systems support device independent programming.

The major features of these operating systems are outlined in Figures 7-2, 7-3, and 7-4.

- DEVICE INDEPENDENT PROGRAMMING
- CONSOLE OPERATOR INTERFACE
- PROGRAM LIBRARY MAINTENANCE
- DISC FILE SUPPORT
- VERY SMALL SIZE

Figure 7-2. BOSS Features

- DEVICE INDEPENDENT PROGRAMMING
- OVERLAPPED I/O
- OPERATOR COMMANDS FROM ANY INPUT DEVICE
- CATALOGED OPERATOR COMMAND FILES
 - JOB CONTROL LANGUAGE
- SEQUENTIAL, RANDOM OR DIRECT FILE ACCESS
- BLOCKING AND DE-BLOCKING OF LOGICAL RECORDS
- BUFFERED FILE OPERATIONS

Figure 7-3. DOS Features

- DEVICE INDEPENDENT PROGRAMMING
- OVERLAPPED I/O
- CONSOLE OPERATOR INTERFACE
- WIDE RANGE OF FUNCTIONS
 - PROGRAM/OPERATOR CALLABLE
- WIDE RANGE OF SUPPORTED DEVICES
- BULK/CORE AND ALL-CORE VERSION
- MULTIPROGRAMMING
- DYNAMIC MEMORY ALLOCATION
- BACKGROUND PROCESSING
 - LANGUAGE PROCESSING
 - PROGRAM TESTING
 - LIBRARY MAINTENANCE
- MEMORY PROTECT-ALL PROGRAMS
- REAL TIME FORTRAN INTERFACE

Figure 7-4. RTOS Features

Two FORTRAN compilers are available for the INTERDATA family of Processors. Interactive FORTRAN is a simple, easy to learn, subset of FORTRAN that allows the user to use the computer system in a time-sharing-like mode. For more complex problems, a full ANSI Standard FORTRAN (FORTRAN IV) compiler is available. A comprehensive real time FORTRAN IV interface is available under the Real Time Operating System (RTOS).

Most programs in the INTERDATA software family are available in stand-alone versions with integral I/O routines, and in OS versions for use with operating system I/O routines. The operator commands used with the various OS programs vary slightly depending on the operating system being used. In this chapter, BOSS commands are used in all operating system version descriptions. For details on operator commands used under the other operating systems, refer to the appropriate operating system manual.

Every program in the INTERDATA standard product family is fully documented and supported. In addition, INTERDATA specifically includes these software products in its standard warranty.

The sections of this chapter dealing with BOSS, DOS, RTOS, and FORTRAN IV present summary information only. For details on these programs, refer to the appropriate programming manual.

This chapter describes only the more widely used software products. For a summary of other programs currently available, refer to the Available Software List, Publication Number 29-239. In addition, many useful software packages are available from the INTERCHANGE User's Group Library. An INTERCHANGE Program Library Catalog of these programs is available upon request.

7.2 COMPARISON OF OS vs STAND-ALONE PROGRAMS

This section contains descriptions of both the Basic Software Package, which includes stand-alone programs, and the OS Software Package, which includes programs that run under an Operating System. The programs in these packages are:

<u>Basic Software Package</u>		<u>OS Software Package</u>	
REL Loader	06-024	REL Loader	06-024
General Loader	06-025	OS Library Loader	03-030
Basic Assembler	03-024	OS Assembler	03-025
Hex Debug	03-013	OS Hex Debug	03-032
Text Editor	03-026	OS Text Editor	03-027

A detailed explanation of these programs is found later in this chapter. This section compares the general usage of Operating System (OS) programs with the stand-alone programs.

The stand-alone programs listed above contain driver routines which allow the programs to work with an ASR-33 or ASR-35 Teletype, a High Speed Paper Tape Reader/Punch, and/or a Line Printer. The Basic Assembler also supports a Card Reader. These programs are device dependent, in the sense that they function with those devices and only those devices. The Device Definition Table in the 50 Sequence is used to select which of the above devices to use. Refer to Section 7.9.2.

The OS programs, on the other hand, use the Basic Operating System (BOSS), the Disc Operating System (DOS), or Real Time Operating System (RTOS) for I/O which means they are device independent. The OS programs use logical unit numbers, and therefore, function with any device by making the proper device assignment. A typical use of logical unit numbers in OS programs is as depicted in Table 7-1.

TABLE 7-1
LOGICAL UNIT NUMBERS

Logical Unit Number	Usage
0	Console or operator device
1	Source input device
2	Binary output device
3	List output device
4	Scratch input/output device
5	Keyboard input/output device
6	Binary input device
7	X

7.3 PROGRAM PREPARATION

The steps required to prepare a program are summarized in this section. The first step in implementing a program is to write the program using assembly language statements as described in Section 7.5 and in the Assembler Language Manual, Publication Number 29-230. Given a symbolic description of a program, the preparation process involves 11 steps as listed below. Programs and their object tapes, mentioned below, are numbered only by their program numbers without revision levels. Refer to a current Available Software List, Publication Number 29-239, for current revision levels.

1. Enter a 50 Sequence into memory, if necessary, and set up the Device Definition Table.
2. Load the REL Loader or an Operating System.
3. Load the Text Editor (TIDE)
4. Prepare program source tapes/card decks.
5. Load the Basic or OS Assembler.
6. Assemble the source tape/card decks.
7. Load an appropriate loader, the REL, General, or OS Library Loader, if necessary.
8. Load the user's object tape(s).
9. Load Hex Debug (CLUB) if necessary.
10. Execute/test/debug the object program.
11. Punch new tape if desired or correct source as necessary and reassemble.

These 11 steps are summarized in Figure 7-5. These steps are discussed in the following two sections, differentiating the Stand-Alone and Operating System environment. BOSS is used as an example in the Operating System section. A similar procedure is used under RTOS or DOS.

7.3.1 Programming in an Operating System Environment

The 11 steps summarized in Figure 7-5 are discussed in the following paragraphs as they relate to the user in a Basic Operating System environment using the BOSS Software Package.

STEP 1. If the appropriate 50 Sequence is not already in memory, it must be manually entered using the Control Panel. This includes the setting up of the Device Definition Table. Location X'0078' must contain the Binary Input Device physical address and command byte information.

STEP 2. Load the desired Basic Operating System.

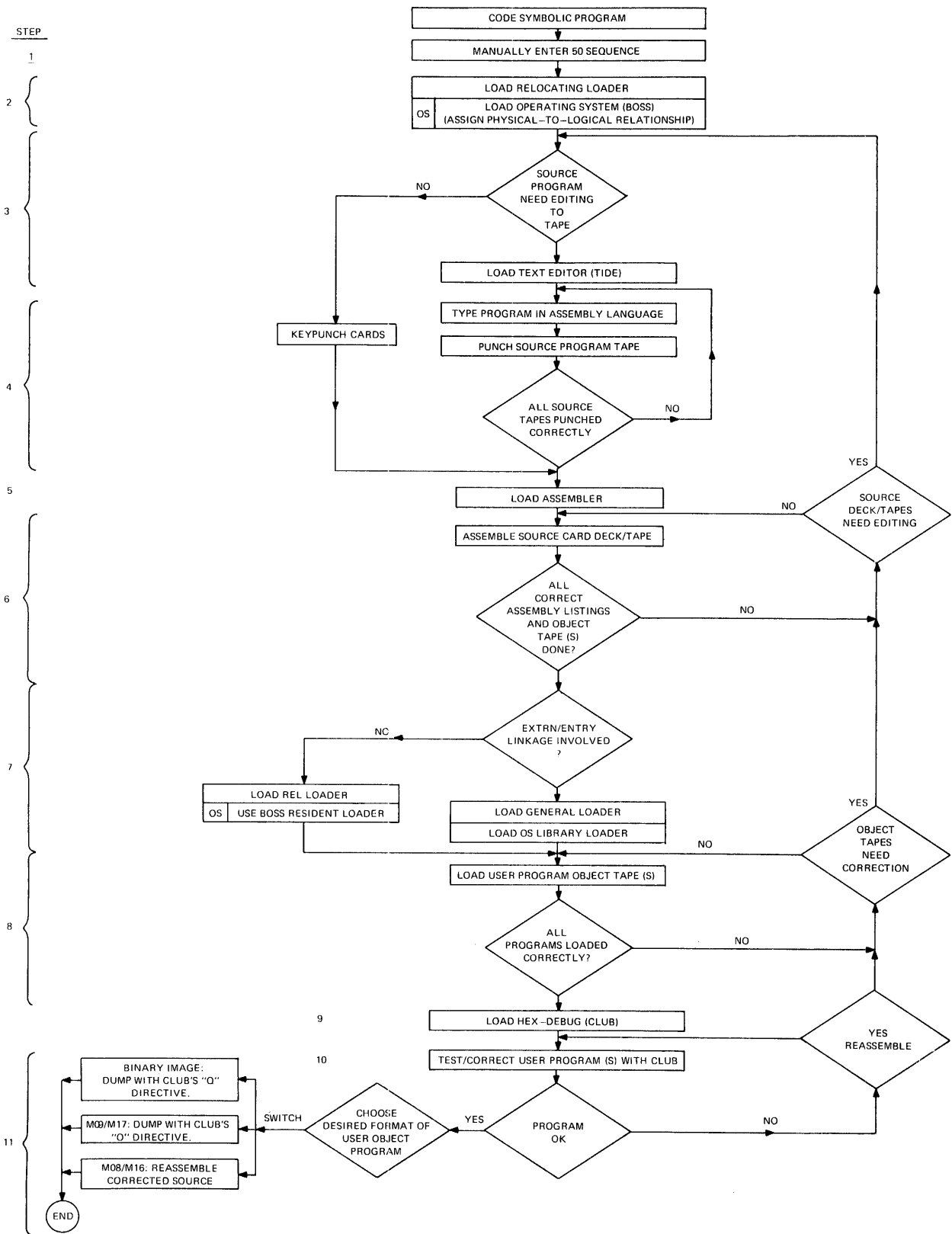


Figure 7-5. Program Preparation Sequence

- A. Put the proper bootstrap BOSS tape in the tape reader.
- B. At the Control Panel, set the Data/Address switches to X'0050', select ADRS MODE, and depress EXECUTE.
- C. Depress INITIALIZE.
- D. Select Run Mode, and depress EXECUTE.
- E. The tape starts to move, unless the reader is a Teletype. In this case, start the tape moving by toggling the reader switch to START (ASR-33) or to RUN (ASR-35).
- F. When errors are detected during the load, the tape stops. In this case, reposition the tape to the previous record gap and depress EXECUTE. Refer to Section 7.9 for details on error recovery procedures.
- G. The load is complete when the BOSS tape has been read to the end and the message "BOSS" is printed on the System Console. With the Basic Operating System in memory, it is necessary to assign physical device addresses to the logical unit numbers to be used. Refer to Table 7-1 for logical unit number usage.

STEP 3. Now that the Basic Operating System has the physical-to-logical device relationships defined, either the Text Editor or OS Assembler can be loaded by the BOSS Resident Loader.

When the configuration contains a card reader and source programs are in card deck form, the user skips Steps 3 and 4 and goes directly to Step 5 to assemble his card deck. Source tapes can be created using the OS Text Editor, Tape Number 03-027M16. To load the OS Text Editor:

- A. Put the OS Text Editor (TIDE) tape in the tape reader.
- B. At the console device, define the desired Bias value with the BIAS command; if the next available location is satisfactory, no BIAS command is needed.
- C. Command BOSS to load from the logical unit (LU) that corresponds to the physical tape reader by typing LOAD LU.
- D. If errors are detected during the load, the tape stops and BOSS notifies the operator on the System Console with a message.

If the Bias selected requests the Resident Loader to load over the BOSS area, the messages printed are:

```
LD ERR
EOJ
```

The user should recheck the defined Bias setting. If the loader detects a sequence record number error or check sum error it prints the respective messages:

```
SEQ ERR           or           CKSM ERR
PAUSE              PAUSE
```

The user can reposition the tape to the previous record gap and type CONTINUE. If the calculated load address exceeds memory the message printed is:

```
MEM FULL
EOJ
```

If Input/Output errors occur, BOSS prints the message:

```
I/O ERR XXNN
PAUSE
```

where XX = bit status of error
and NN = physical device address

E. The load is complete when the tape data has been read to the end, and BOSS reports on the System Console device the next available location (XXXX) in memory that can be loaded by printing the message END XXXX. BOSS awaits further direction.

STEP 4. Now that OS Text Editor (TIDE) is in memory, BOSS can be directed to start the TIDE program by typing START XXXX where XXXX is the starting location of the TIDE program. TIDE uses logical unit number as follows:

<u>LU</u>	<u>Logical Function</u>
01	Source Input Device
02	Binary Output Device
03	List Device
05	Keyboard Device

Prior to TIDE's execution, the user must have these assigned as desired.

The user's source tapes can now be prepared by the methods available with TIDE. For details of the use of TIDE, refer to Section 7.10. To bring control back to BOSS, after the source tapes are created, type the TIDE Control command letter "E" whenever TIDE is in the Command Mode. BOSS responds with the message EOJ (End Of Job).

- STEP 5. Given the source card deck(s), or the source tape(s) as prepared by TIDE, the next step is to assemble the source program into an assembly listing and object program. For this step, the OS Assembler, Program Number 03-025, must be loaded into memory. Again, a Bias is selected, as desired, using the BIAS command in BOSS. Users with more than 16KB of core could load the Assembler above OS TIDE, by typing LOAD LU, where LU is the logical load device and BOSS automatically loads the OS Assembler at the next available location discussed above. However, it is generally advisable, to always allow as much core memory as possible for the OS Assembler's Symbol Table. Procedures to load the OS Assembler are the same as those described in Step 3, for the OS TIDE program.
- STEP 6. Now that the OS Assembler program is in core memory, the operator assembles his source card deck(s) or source tape(s) by following the operating instructions specific to the OS Assembler running under BOSS control. These are described in Section 7.5. At the end of assemblies, BOSS responds with the message EOJ (End Of Job).
- STEP 7. For each source program assembled, the OS Assembler generates an assembly listing which may call attention to a multitude of error conditions, some of which may indicate the object tapes produced are in error. The user should inspect his assembly listing(s) to determine whether to continue at this point or to correct his source and reassemble. The OS Assembler generates object tapes in loader format which are "zoned" when output to a Teletype punch device and "non-zoned" when output to a non-Teletype punch device. The object tapes may be absolute or relocatable depending on how the source program was written. When they are relocatable, a Bias must be selected and defined to the loader that loaded them. If the user programs involve ENTRY's or EXTRN's, the OS Library Loader, Tape Number 03-030M16, must be loaded since the BOSS Resident Loader does not provide linkage capability. If no ENTRY/EXTRN linkage is involved, see Step 8, and use the BOSS Resident Loader to load the user program(s). To load the OS Library Loader, place Tape Number 03-030M16 in the tape reader, reset the Bias to the next available location above BOSS and follow the general loading procedures outlined in Step 2 above.
- STEP 8. With the BOSS Resident Loader or OS Library Loader in memory, the user's object tapes can be loaded. The loading process is as described above for the BOSS Resident Loader or reference may be made to Section 7.9 for details on any loader. If, for some reason, the user's object tapes do not load correctly, the user checks the loaders' error indications, or rechecks his assembly listings, to determine if it is necessary to either reassemble his source program, or to return to the editing process to correct his source program.

- STEP 9. BOSS allows absolute memory examination and modification, so that if these debugging aids suffice, the user can proceed to Step 11. If it is necessary to test or debug the user program just loaded, the OS Hexadecimal Debut (CLUB), Tape Number 03-032M16, should be loaded following similar procedures as described in Step 3 above.
- STEP 10. With BOSS and its Resident Loader or OS Library Loader, the user programs, and OS Hex Debug, in memory, the user is now ready to debug his program. Refer to Section 7.11 for the operating procedures and debugging techniques described in detail. Control is returned to BOSS when executing the CLUB program by typing the OS CLUB "*" asterisk directive.
- STEP 11. If during the debugging, it is determined that a reassembly is in order, the user corrects his source and returns to the editing process. Once the user is assured his object program runs correctly, he may elect one of several ways to format his hard copy of the object program.

The user may punch a binary memory image M14 tape using CLUB's "Q" directive, or an M09/M17 absolute binary object tape in loader format using CLUB's "O" directive, or he may adjust his source and reassemble to obtain a new assembly listing and object tape in M08, M09, M16, or M17 loader format.

7.3.2 Programming in a Stand-Alone Environment

The 11 steps summarized in Figure 7-4 to prepare a user program are discussed in the following paragraphs as they relate to stand-alone programs in the Basic Software Package.

- STEP 1. If the appropriate 50 Sequence is not already in memory, it must be manually entered using the Control Panel.
- STEP 2. In order to get the TIDE Program into memory, the REL Loader, Program Number 06-024 is required. The loader is entered into memory as follows:
- A. Put the loader tape in the tape reader, observing the first character alignment when the 50 Sequence used does not contain the Auto Load feature.
 - B. Set the Data/Address switches to X'50', select ADRS Mode, and depress EXECUTE.
 - C. Depress INITIALIZE.
 - D. Select Run Mode, and depress EXECUTE.
 - E. If a Teletype is in use, start the tape moving by toggling the reader switch to START (ASR-33) or RUN (ASR-35). After a momentary delay, the lights flash on the right half of Display Register 2 indicating that the tape is actually loading. If this does not occur, check to see if all the proper loader procedures above were followed.
 - F. If errors are detected during the load, the tape stops. In this case, reposition the tape to the previous record gap and depress EXECUTE. Refer to Section 7.9 for details on error recovery procedures.
 - G. The load is complete when the tape has been read to the end, and the Processor halts with the WAIT light illuminated.
- STEP 3. Now that the loader is in memory, the TIDE program can be loaded. This is done as follows:
- A. Put the TIDE tape in the tape reader.
 - B. Depress EXECUTE. This executes the REL Loader and starts the TIDE tape moving.
 - C. If errors are detected during the load, the tape stops. In this case, reposition the tape to the previous record gap and depress EXECUTE.
 - D. The load is complete when the tape has been read to the end, and the Processor with the WAIT light illuminated.

- STEP 4. Now that TIDE is in memory, the source tapes can be prepared. For details on the use of TIDE, refer to Section 7.10.
- STEP 5. Given the source tapes as prepared by TIDE, or the source card deck, the next step is to assemble the source. For this step, the Basic Assembler, Tape Number 03-024M10, must be loaded into memory as follows:
- A. Put the assembler tape into the tape reader, observing the first character alignment when the 50 Sequence used does not contain the Autoload feature.
 - B. Set the Data/Address Switches to X'50', select ADRS Mode, and depress EXECUTE.
 - C. Depress INITIALIZE.
 - D. Select Run Mode, and depress EXECUTE.
 - E. If a Teletype is in use, manually start the tape moving.
 - F. If errors are detected, the tape stops. In this case, reposition the tape to the previous record gap and depress EXECUTE.
 - G. The load is complete when the tape has been read to the end, and the Processor halts with the WAIT light illuminated.
- STEP 6. The use of the Basic Assembler is described in Section 7.5.
- STEP 7. For each source program assembled, the Basic Assembler generates an object tape in loader format; zoned when output to a TTY punch, non-zoned to non-TTY punch devices. These object tapes must now be loaded into memory. The object tapes may be absolute or relocatable, depending on how the source program was written. If object tapes are relocatable, a bias must be selected and defined to the loader. If the programs involve ENTRY or EXTRN linkage, the General Loader, Program Number 06-025, is required. The proper loader should be entered into memory, using the same procedure as described under Step 2.
- STEP 8. With the loader in memory, the object tapes can be loaded. The loading process is described in detail in Section 7.9
- STEP 9. If it is necessary to test or debug the user programs just loaded, the Hex Debug Program (CLUB) may be useful.
- STEP 10. The use of CLUB is described in Section 7.11.
- STEP 11. Once the user is assured his object program runs correctly, he may punch a binary core image M14 tape of his program with CLUB's "Q" directive, or may punch an M09/M17 absolute binary object tape with CLUB's "O" directive. He may adjust his source and reassemble to obtain a new assembly listing and object tape in M08, M09, M16, or M17 tape format.

7.4 PROGRAMMING CONVENTIONS

This section summarizes some of the conventions used in programs and documentation supplied by INTERDATA.

<u>Convention</u>	<u>Comments</u>
Hexadecimal Notation	<p>INTERDATA documentation uses hexadecimal notation extensively. The letter X denotes that the following alphanumeric characters, enclosed in single quote marks, form a hexadecimal number. Thus, X'50' indicates 50_{16}.</p> <p>In some contexts, hexadecimal notation is used exclusively. For example, CLUB, the interactive debug program, uses hexadecimal numbers only. Also, a program listing as generated by the assembler, describes the binary form of the program in hexadecimal. In these cases, the X'---' notation is not used and all numbers are assumed to be hexadecimal. In general, memory locations and program starting addresses are also defined in hexadecimal.</p>
ASCII Codes	<p>Standard programs supplied by INTERDATA represent characters in ASCII code. This character code is defined in Appendix 8. The ASCII code represents each character with 8 bits in which the high-order bit is available for parity. The Parity Bit may change according to the input/output devices in use. Internal to the Processor, most programs mask the high order bit to zero, and handle the character in terms of the 7-bit codes. The Assembler generates the 8-bit form of ASCII codes for characters with the high order bit zero.</p>
Slashed Zeros	<p>Certain documents use a slash to identify the digit zero (0) as opposed to the letter "O".</p>
Device Number 2	<p>Some INTERDATA programs, such as CLUB, TIDE and the test programs, assume that Device Number 2 is a Teletype. This device is used for keyboard inputs and message printouts. The REL Loader, General Loader and the Basic Assembler take the special steps when Device Number 2 is specified in the Binary Device Definition at X'78'. These special steps involve XON and XOFF characters which control tape motions.</p>
Device Number 4	<p>The Basic Assembler assumes the card reader is Device Number 4. Refer to Appendix 8 for other Standard-Preferred Device Addresses.</p>
Memory Size Notation	<p>Memory sizes are described in terms of KB, which indicates "binary" thousands of bytes. A "binary" thousand is 1024.</p>

7.5 INTERDATA ASSEMBLER PROGRAM

7.5.1 General Description

This section describes the Assembly Language for the INTERDATA Processor family. It also describes the format of the assembly listing and binary object program.

INTERDATA Digital Systems are centered around a Processor which can be programmed to solve a wide range of problems. The program to be executed by a Processor consists of binary coded instructions and data which are stored in a memory.

To assist the process of defining and generating a program, the user can write his program in a symbolic way, using what is called assembly language. In the assembly language, programs are represented using symbols and mnemonic abbreviations for the instructions and data in the program. The statements in the assembly language which represent the program constitute the source form of the program. Table 7-2 is an example of an assembly language source program that searches an area of memory for the first occurrence of the Number 15.

The translation from the symbolic source program to the binary object program is done by the Assembler. The Assembler reads the source program, statement by statement, from punched paper tape or cards.

TABLE 7-2.
TYPICAL SOURCE PROGRAM

Name	Operation	Operand	Comment
BEGIN	ORG	X'100'	SET THE LOCATION COUNTER
	LHI	2, TOP	TOP OF DATA TABLE
	LHI	3, 2	HALFWORD INCREMENT
	LHI	4, BOTTOM	BOTTOM OF DATA TABLE
	LHI	10, 15	SEARCH VALUE OF 15
LOOK	CLH	10, 0(2)	COMPARE
	BE	FINI	BRANCH ON EQUAL TO FINI
	BXLE	2, LOOK	NOT FOUND GO LOOK FURTHER
FINI	LPSW	WAIT	STOP THE PROGRAM
WAIT	DC	X'8000', A(BEGIN)	
TOP	DS	1000	
BOTTOM	DS	2	
	END	BEGIN	

As the statements are read, a Symbol Table is accumulated. This table contains every symbol and the value of the Location Counter where the symbol was encountered. For the previous example, the complete symbol Table is shown in Table 7-3.

TABLE 7-3.
TYPICAL SYMBOL TABLE

Symbol	Value (in hexadecimal)
BEGIN	0100
BOTTOM	050C
FINI	011C
LOOK	0110
TOP	0124
WAIT	0120

The Assembler generates both an object tape and a listing. The object tape contains the binary information to be loaded into memory. The listing is a printed record which shows each source statement and the binary information generated for that statement. The binary information on a listing is always represented in hexadecimal form as shown in Table 7-4.

There are two versions of the Assembler Program available. They are the Basic Assembler, Program Number 03-024, and the OS Assembler, Program Number 03-025.

TABLE 7-4.
TYPICAL ASSEMBLY LISTING

Location	Data	Name	Operation	Operand	Comments
0100			ORG	X'100'	SET THE LOCATION COUNTER
0100	C820	BEGIN	LHI	2, TOP	TOP OF DATA TABLE
	0124				
0104	C830		LHI	3, 2	HALFWORD INCREMENT
	0002				
0108	C840		LHI	4, BOTTOM	BOTTOM OF DATA TABLE
	050C				
010C	C8A0		LHI	10, 15	SEARCH VALUE OF 15
	000F				
0110	45A2	LOOK	CLH	10, 0(2)	COMPARE
	0000				
0114	4330		BE	FINI	BRANCH ON EQUAL TO FINI
	011C				
0118	C120		BXLE	2, LOOK	NOT FOUND GO LOOK FURTHER
	0110				
011C	C200	FINI	LPSW	WAIT	STOP THE PROGRAM
	0120				
0120	8000	WAIT	DC	X'8000', A(BEGIN)	
	0100				
0124		TOP	DS	1000	
050C		BOTTOM	DS	2	
050E			END	BEGIN	
BEGIN	0100				
BOTTOM	050C				
FINI	011C				
LOOK	0110				
TOP	0124				
WAIT	0120				

The basic Assembler is a stand-alone program that operates in a minimum of 8KB memory. This version of the Assembler has the feature that the floating-point conversion logic, which is used to assemble floating-point data, can be overlaid to provide additional Symbol Table area. With this feature, more symbols, and therefore larger programs, can be assembled if floating-point data conversions are not required. Programs which do involve floating-point data must specify FLOAT in the first option (OPT) statement, which prevents the floating-point logic from being overlaid. Error checks are provided to detect if either FLOAT is specified, or floating-point data is encountered after the floating-point logic is over-written. Refer to Basic Assembler Operating instructions in the Section 7.12 for further details.

The OS Assembler requires an Operating System and a minimum of 16KB memory. It has the additional PAUSE, IF, TITLE, DB functions; sequence check and scratch options; evenness error check; listing error count; and assembles symbolic short format, extended RR branch instructions, and several optional instruction sets.

7.5.2 Assembly Procedures

The Assembly may take one, two, or three passes of the source program to complete the assembly. The number of passes is controlled by an option control statement in the source program. Refer to the OPT pseudo operation for details. When so directed, the Assembler makes an assembly, complete with listing and object tape, in one pass. In this case the assembly time is minimized, but the resulting object tape is longer, and the program listing is not complete because of forward reference representation.

With two-pass assemblies, the first pass is devoted to the development of the Symbol Table. On the second pass, the listing is printed and the object tape is punched. Assemblies are normally performed using two passes. The two-pass procedure is appropriate except where the input-output device configuration prohibits punching and printing on the same pass. In this case, the three-pass assembly can be used. With a three-pass assembly, the Symbol Table is built on pass one, the listing is printed on pass two and the object tape is punched on pass three.

The assembly listing is produced as part of the assembly process. The listing contains the source statements and the data generated from each statement. The first four hexadecimal digits in the left-hand column represent the value of the Location Counter or values of symbols resulting from EQU Assembler statements. The next four hexadecimal digits represent the data generated by the Assembler from the source statement.

Error flags may precede the Location Counter values. These flags indicate that an error was encountered in interpreting the statement. The meaning of each flag is shown in Table 7-5.

TABLE 7-5.
STATEMENT ERROR FLAGS

Error Flag	Meaning
Blank	Correct Assembly
F	Format error
M	Multiple defined symbol
O	Operation mnemonic invalid
T	Truncation error, a constant or expression has overflowed the specified limits
R	Relocation error, a meaningless combination of relocatable symbols in an expression
S	Symbol table overflow
U	Undefined symbol
#	Sequence Number Error
E	Even-number violation in R1 or R2 field

Whenever an invalid op-error (O) occurs, the Assembler always advances the Location Counter by four bytes so that the program can be easily patched for debugging.

A flag immediately following the data generated by the Assembler indicates whether the data is relocatable, absolute, or a forward reference. These flags are defined in Table 7-6.

TABLE 7-6.
DATA FLAGS

Data Flag	Meaning
Blank	Absolute Data
R	Relocatable Data
F	Forward Reference Data

The Symbol Table that was accumulated during PASS1 is printed following the END assembly pseudo-op. Any statements containing symbols preceded by a U (Undefined symbol) error flag can be corrected at this time and PASS1 of the assembly process repeated.

The Symbol Table is again printed following the END assembly pseudo-op of PASS 2. The symbols are listed alphabetically with their values. If the symbol is defined, the value is followed by an R if that value is relocatable. If the symbol is undefined, the last value of the Location Counter for a statement referencing the undefined symbol is printed. This value is called a reference address. The programmer uses the reference address to directly locate where the undefined symbol appears in the program listing.

Preceding each symbol is a field for error flags.

The Symbol Table printout contains one of the two character flags shown in Table 7-7, preceding each symbol:

TABLE 7-7.
SYMBOL TABLE ERROR FLAGS

Flag	Description
2 Spaces	Correct locally defined symbol
U Space	Undefined local symbol
M Space	Multiply defined local symbol
*Space	Properly used EXTRN or ENTRY symbol
**	Symbol used in the operand of EXTRN or ENTRY not used properly or at all within the program
* <	An ENTRY symbol used as an EXTRN in the program (never defined), became referenced
* >	An EXTRN symbol used as an ENTRY in the program (became defined)
D*	Symbol used in the operand of both an EXTRN and an ENTRY line (gets written on object program as an ENTRY, if defined; as an EXTRN, if referenced.)
*M	Any EXTRN or ENTRY that became multiply defined

7.5.3 Assembler Language

Source Statements

There are two basic kinds of source statements, instruction statements and comment statements. Instruction statements are used for Machine instructions and Assembler instructions. The instruction statements may have the following information fields:

- Name
- Operation
- Operand
- Comments

Common statements, which begin with an asterisk (*) in Column 1, should not be confused with the comment field of the instruction statement. Comment statements can occupy the entire statement line.

Instruction Statements - The comment and instruction statements are written by the programmer on a coding form that has the various fields clearly marked. This form, when filled out, is used to generate the source paper tape or source cards that are read by the Assembler during the assembly process. The Name (label) begins in Column 1, the Operation begins in Column 10, the Operand begins in Column 16, and the Comments are usually in Columns 35-60. The fixed field positions are a convenience for the programmer only, and are not required by the Assembler.

The Assembler simply requires that fields be separated by at least 1 space but not more than 14 spaces. The fields are described in the following paragraphs.

Name

A name consists of from one to six characters. The name must be written with the first character in Column 1, and it must not contain any blanks. Names are used by the programmer to identify data and instructions in the program. The first character must be a letter; the remaining five characters can be letters or numbers. Typical names are:

NAME	OPERATION	OPERAND
START		
ARG1		
LOOP2		
GO		

Operation

The operation field specifies a Machine instruction mnemonic that is translated by the Assembler to machine code, or it specifies an Assembler instruction mnemonic to control the assembly process. An operation is always required in an instruction statement, and should be written on the coding form beginning in Column 10. No blanks may be used within the operation. Typical operations are:

NAME	OPERATION	OPERAND
	ORG	
	LHI	
	AHR	
	DC	

Operand

Operands identify the data to be used by the instruction. The type of operand and the number of operands required depend on the particular instruction appearing as the operation. No blanks may appear within or between operands. Typical operands are:

NAME	OPERATION	OPERAND
	AH	R6, TEMP
	BL	OUT
	STH	R6, TABLE (R5)
	B	IN

Comment

Comments are descriptive text. Comments are printed on the assembly listing, along with the name, operation, and operand of the source statement. Comments are written beginning after the first blank in the operand, and can contain 26 characters. In general, Columns 35-60 are used when only 72 columns are available for listing.

Comment Statements - Comment statements are descriptive text that can occupy the entire source statement line. Comment statements are written with an asterisk (*) in Column 1, followed by any descriptive text the programmer desires. They should contain no more than 55 characters in addition to the asterisk when only 72 columns are available for the assembly listing as is the case for users listing on a Teletype printer. Comment statements do not produce binary object information and are used only as documenting aids. Several comment statements are:

```
* THIS IS A COMMENT STATEMENT
* IT CAN BE USED ANYWHERE IN A
* PROGRAM AS A PROGRAMMER
* AND DOCUMENTATION AID
*
*X< OR = Y/Z? IF SO, GO ON
```

Character Set - All source statements are written using the following characters:

Alphabetics	A through Z
Numerics	0 through 9
Special characters	+ - , = * ' () blank
	and all characters printable on a Teletype
Special characters considered alphabetic	\$ or . or @

The dollar sign (\$), period (.), and at-sign (@) are considered by the INTERDATA Basic and OS Assemblers as special alphabetic characters. That is, they may be used as the first character of any symbol, but only as a symbol's first character. This feature is provided so that users who wish to reference the ENTRY name\$ of the INTERDATA FORTRAN Run-Time Library Subroutines may assemble such EXTRN symbols as \$A, .O, or @V.

Assembler Language Structure

Each entry in a source statement may be composed of one or more items depending on the kind of source statement being written.

- A name, when present must be a symbol
- An operation, always present, must be a Machine Instruction mnemonic or an Assembler instruction mnemonic
- An operand may be composed of one or more expressions, which in turn are composed of symbols, constants, and arithmetic combinations of symbols and constants
- A comment is optional

Symbols - A symbol is the name of a location or value. It is used in the name of a statement, as an operand, or within an expression. In either case, symbols consist of from one to six characters. The first character must be alphabetic. The characters that can be used for a symbol are:

Alphabetics	A through Z
Special Alphabetics	\$, . , @ allowed only as the first character of a symbol
Numerics	0 through 9

The following symbols are valid and could be used as a name, as an operand, or within an expression.

```
T2
LOOP25
N
STOP
@V
$N
.O
```

The following symbols are invalid for the reasons given:

SYMBOL	ERROR CONDITION
2TOP	First character is not alphabetic
COMMAND	More than six characters
A TO D	Contains a blank
X4/2	Contains a special character, a slash
AB@12	Contains a special alphabetic character in other than the symbol's first character position

Instruction Constants - Instruction constants appear as an operand for both Machine instructions and Assembler instructions. An instruction constant can be one of three types:

- Decimal
- Hexadecimal
- Character

In general, instruction constants define 16-bits or a halfword of information. The type of constant is identified by a prefix code.

CODE	CONSTANT TYPE
None	Halfword Decimal
H	Halfword Decimal
X	Hexadecimal
C	Character

Decimal constants can be from one to five decimal digits, not to exceed 32,767 maximum or -32,768 minimum, and are written as:

125
32765
-15

Hexadecimal constants can be from one to four digits. The hexadecimal digits are:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

The hexadecimal constant must be enclosed in single quotation marks and be preceded by the letter X. Leading zeros are not necessary. The hexadecimal constants generated are right justified to form 16-bit halfwords. Examples are:

X'F'	generates	X'000F'
X'D4E'	generates	X'0D4E'
X'030'	generates	X'0030'

Character constants used in the operand field of an instruction statement can be from one to two characters. The permissible characters are:

Alphabetics	A through Z
Numerics	0 through 9
Special characteristics	+ - = * () blank and all ASCII coded characters printable on the Teletype except the single quote (').

The character constant must be enclosed in single quotation marks and be preceded by the letter C. A single character within the quotes is right justified to form a 16-bit halfword. Each character is translated into a byte of seven bit ASCII code.

C'*'	generates	X'002A'
C'12'	generates	X'3132'
C'XY'	generates	X'5859'

Expressions - An expression is a symbol, a constant, or a series of such items separated by the arithmetic operations + (addition), and - (subtraction). Examples of valid expressions are:

```
SAM
5
LOOP+4
TABLE+X'12A'
STOP-GO+2
C'A'+1
-FROG
```

Relocatable and Absolute Expressions - An expression is absolute if its value is absolute. Similarly, an absolute expression does not change as a function of the physical location of the program in the machine. The value of a relocatable expression does change when the location of the program changes. The relocatable value changes by the difference in byte locations between the originally assigned area of storage and the newly assigned area of storage.

An expression, when evaluated, produces a value which is considered absolute or relocatable according to the rules outlined in Table 7-8.

TABLE 7-8.
ABSOLUTE AND RELOCATABLE EXPRESSION RULES

	<u>A+B</u>	<u>A-B</u>
A is absolute, B is absolute	Absolute	Absolute
A is absolute, B is relocatable	Relocatable	Invalid
A is relocatable, B is absolute	Relocatable	Relocatable
A is relocatable, B is relocatable	Invalid	Absolute

Location Counter - The value of the Location Counter can be referenced by using an *, which means, "current value of Location Counter." Addressing relative to the Location Counter is on a byte basis. To specify an address that is one-RX instruction forward, the correct expression would be *+4.

In the first two examples below, the Branch (B) instruction transfers to the instruction labeled LOOP 25. In the last example below, the Load Program Status Word instruction would halt the Processor ready to execute the instruction labeled LOOP 25.

NAME	OPERATION	OPERAND
LOOP25	B	*+8
	LHI	R6, 0
	LB	R5, TABLE(R6)
LOOP25	B	*+6
	SHR	R6, R6
	LB	R5, TABLE(R6)
	LPSW	*+4
	DC	X'8000', LOOP25

The proper alignment of the Location Counter to halfword memory boundaries is provided by the Assembler. If a character data constant specification is followed by an instruction or halfword data, halfword alignment is forced. The value of the Location Counter is absolute or relocatable depending on the operand and entry of the Assembler ORG instruction. If the expression appearing as the operand is relocatable, the Location Counter value (*) is relocatable; if the expression is absolute, the Location Counter value is absolute. If no ORG is specified in a program, the Location Counter starts at relocatable zero.

7.5.4 Machine Instruction Format

The Assembler provides the facility for representing all the Machine instruction operation codes with mnemonics. The binary instruction is generated by the Assembler from the operation mnemonic and the operand. Table 7-9 summarizes the formats used.

The mnemonic in the operation field specifies the desired function, i. e., add. Each basic instruction has a unique mnemonic that is used as the operation. These mnemonics, machine operations, and their names are listed in Appendices 1 and 2. Extended Branch mnemonics are listed in Appendix 3. Some instruction examples are:

RR - Format Instructions

NAME	OPERATION	OPERAND
GO	LHR	1, 2
	BALR	R15, R12
LOOP12	AHR	3, 3
	DHR	DEND, ISOR
RETURN	BTCR	8, EXIT
	BFCR	3, ERROR
	BTCR	BUSY, SENSE
	BR	EXIT
	NOPR	
	BCR	CARRY
	BLR	R13

SF - Short Format Instructions

NAME	OPERATION	OPERAND
	LIS	R12, 15
	AIS	R12, 1
	LCS	R3, 1
	SIS	COUNT, ONE
	SLLS	R2, 8
	BTBS	8, 2
	BTFS	4, 6
	BFBS	BUSY, 1
	BFFS	5, 10
LOOP	BNES	NOTEQL
NOTEQL	BNLS	LOOP
	BMS	END
	BNZS	*+8
END	BS	LOOP

RX - Format Instructions

NAME	OPERATION	OPERAND
TEST1	STH	R7, TEMP
	MH	13, TABLE(3)
	LH	TWELV, 0(X7)
	AH	X'B', TOP+4(5)
	BTC	8, HANGUP
	BFC	3, DONE
	BL	HANGUP
	BZ	DONE
	NOP	
	B	ENTRY(INDEX)

RS - Format Instructions

NAME	OPERATION	OPERAND
	LHI	0, X'9DAE'
	AHI	R7, 1
	BXLE	R4, LAST1
	SLHL	R12, 8
	AL	X'CF'
	SINT	2
	LPSW	*+4

TABLE 7-9.
INSTRUCTION FORMAT SUMMARY

Type	Machine Format				Assembly Format*		Applicable Instructions	
	Bits	8	4	4	16	OPERATION		OPERAND
RR		OP	R1	R2		OP	R1, R2	All RR except branches BTBR or BTBR BR or NOPR or Extended RR Branch Mnemonics
		OP	M1	R2		OP	M1, R2	
		OP	M1	R2		OPX	R2	
SF		OP	R1	N		OP	R1, N	Short Format Immediate Short Format Branches Short Format Extended Branch Mnemonics
		OP	M1	D		OP	M1, D	
		OP	M1	D		OPX	A*	
RX		OP	R1	X2	A	OP	R1, A(X2)	All RX except branches BTC or BFC B or NOP or Extended Branch Mnemonics
		OP	M1	X2	A	OP	M1, A(X2)	
		OP	M1	X2	A	OPX	A(X2)	
RS		OP	R1	X2	A	OP	R1, A(X2)	All RS except LPSW, SINT, AL LPSW, SINT, AL
		OP	0	X2	A	OPX	A(X2)	

*Assembly Format Key

Designation

Interpretation

- OP -Basic Machine instruction mnemonic.
- OPX -Extended Machine instruction mnemonic.
- R1 -Any expression whose generated absolute value is a single hexadecimal digit, representing the general/floating register acting as the first operand in Machine instructions.
- R2 -Any expression whose generated absolute value is a single hexadecimal digit, representing the general/floating register acting as the second operand in Machine instructions.
- X2 -Any expression whose generated absolute value is a single hexadecimal digit, representing the General Register acting as an Index Register in Machine instructions.
- M1 -Any expression whose generated absolute value is a single hexadecimal digit, representing the Condition Code field of Machine Branch instructions.
- N -Any expression whose generated absolute value is a single hexadecimal digit, representing an actual numeric constant (0 through 15₁₀) imbedded in the Machine Short Immediate instructions.
- D -Any expression whose generated absolute value is a single hexadecimal digit, representing a displacement by halfwords, in the BTFS, BTBS, BFFS, BFBS instructions.
- A -Any expression whose generated absolute or relocatable value is up to 16 bits or four hexadecimal digits, representing either an address or constant acting as a portion of the second operand of Machine instructions.
- A* -Same source expression as A above, except that the A* value, when subtracted from the current Location Counter and divided by two, generates the object displacement D as defined for the BTFS, BTBS, BFFS, BFBS instructions. That is, the user programs the Extended Branch Mnemonics, with symbolic address field operands when the address appears to be within 15 halfwords of the branch. When the Assembler generates the object op-code, Condition Code, and displacement value D, the programmer is alleviated from the tedious selection of direction, Condition Code, and halfword displacement values at programming time. The recoding of short branches necessitated by maintenance programming changes within the short branch area is lessened by the fact that these extended short branches have symbolic operands in the source as opposed to absolute halfword displacement values in the source.

7.5.5 Assembler Instructions (Pseudo-Ops)

Assembler instructions are used to control the assembly process, define symbols, and generate data. Assembler instruction statements do not always generate data as the Machine instruction statements do. The following paragraphs describe the Assembler instructions. Refer to Table 7-10, for a summary of the Assembler control instruction statements.

Symbol Definition Instructions

EQU - Equate Symbol

NAME	OPERATION	OPERAND
A symbol required	EQU	A defined expression

The EQU Assembler instruction is used to equate a symbol to the value of an expression. The value of the symbol is relocatable or absolute as determined by the expression. Symbols used in the expression must be previously defined. Disregarding this rule, causes assembly listing errors and the object program tape may not load correctly. The EQU Assembler instruction is commonly used to equate symbolic General Register names to their appropriate value.

NAME	OPERATION	OPERAND
LOOP	EQU	LOOP1
TOP	EQU	END-64
DELTA	EQU	BOTTOM-TOP
HERE	EQU	*
START	EQU	X'01FE'
BLANKS	EQU	C' '
R6	EQU	6
R7	EQU	7
	.	
	.	
	LHR	R6, R7

One purpose of symbolic register designations is in the ease with which registers can be reassigned without extensive recoding. To change from General Register 6 to General Register 1 requires changing only the R6 EQU 6 Assembler statement to an R6 EQU 1 Assembler statement.

ENTRY - Identify Entry-Point Symbol

NAME	OPERATION	OPERAND
Not used	ENTRY	One or more symbols separated by a comma

TABLE 7-10.
SUMMARY OF ASSEMBLER INSTRUCTIONS

<u>Symbol Definition Instructions</u>	
EQU	Equate Symbol
ENTRY	Identify Entry-Point Symbol
EXTRN	Identify External Symbol
<u>Data Definition Instructions</u>	
DC	Define Constant, used to specify the following data types
	-C Character Constant
	-X Hexadecimal Constant
	-A Address Constant
	-H Halfword Decimal Constant
	-E Floating-Point Constant (Single-Precision)
	-D Floating-Point Constant (Double-Precision)
DB	Define Byte
DS	Define Storage
<u>Assembler Control Instructions</u>	
OPT	Specify Options
	-PASS1 One Pass Assembly
	-PASS2 Two Pass Assembly
	-PASS3 Three Pass Assembly
	-PUNCH Punch Object Tape
	-NOPNCH No Punching of Object Tape
	-PRINT Print Assembly Listing
	-NOPRNT No Printing of Assembly Listing
	-STOP Stop After Each Pass
	-GO Go, After Each Pass, to the next Pass
	-FLOAT Floating-Point Required (Basic Assembler only)
	-SCRT Write Source on to Scratch Device (OS Assembler only)
	-SQCHK Check sequence numbers of source statement (OS Assembler only)
ORG	Set Location Counter
DO	Conditional/Multiple Assembly of an Instruction
END	End Assembly, and transfer address if operand is present
PAUSE	Assembly Pause
IF	Conditional Assembly
TITLE	Listing Title and Format Control

The ENTRY Assembler instruction identifies symbols that are defined in this program and may be used by some other program. This permits programs that are assembled separately to communicate with each other. Only those symbols identified as entry symbols are available to other separately assembled programs. All ENTRY statements must precede any symbol definitions in the program. An example is:

NAME	OPERATION	OPERAND
	ENTRY	SIN, COSIN
	.	
SIN	LHI	R7, TEMP2
	.	
COSIN	LHI	R8, TEMP3
	.	
	.	
	END	

EXTRN - Identify External Symbol

NAME	OPERATION	OPERAND
Not Used	EXTRN	One or more symbols separated by commas

The EXTRN Assembler instruction identifies symbols that are defined in another program that are referenced by this program. This permits programs that are assembled separately to communicate with each other. Only those symbols identified as ENTRY symbols in another program should be identified as externally defined in this program. All EXTRN statements must precede any references to the symbols called out as EXTRN's in the program.

An example is:

NAME	OPERATION	OPERAND
	EXTRN	SIN, COSIN
	.	
	BAL	R15, SIN
	.	
	BAL	R15, COSIN
	.	
	.	
	END	

Any symbols declared as EXTRN's must be used with the following restrictions:

- EXTRN symbols must not be combined in arithmetic expressions, i. e.

LH	3, SIN+2	
----	----------	--
- EXTRN symbols must not be used in the R1, R2, or X2 fields of an instruction: i. e.

LHR	COS, 2	used in R1 field illegally
LHR	3, SIN	used in R2 field illegally
LH	3, 2(SIN)	used in X2 field illegally
- EXTRN symbols must not be used with Assembler pseudo-ops such as DO, EQU, END, etc.

The utility of the ENTRY, EXTRN Assembler instructions is realized when subroutines are written. Rather than having to assemble the main program and its subroutines at the same time in order to establish correct communication, the ENTRY, EXTRN capability permits the main program to be assembled and then loaded with the previously assembled subroutines. The symbols identified by ENTRY or EXTRN statements are then linked at load time by any linking loader, such as the General Loader.

Consider the following two hypothetical programs:

NAME	OPERATION	OPERAND
* MAIN PROGRAM		
*		
	EXTRN	RIPLE, DABB
START	LHI	R7, 7
	LHI	R15, X'F0F0'
	.	
	.	
	STH	R1, DABB
	BAL	R7, RIPLE
	B	START
	END	
* SUBROUTINE RIPLE		
*		
	ENTRY	RIPLE, DABB
START	LH	R2, DABB
	AHI	R2, C'*-'
	BR	R7
DABB	DS	2
RIPLE	EQU	START
	END	

The symbols RIPLE and DABB are used by the main program, but their values are not known at assembly time. Since they are defined as EXTRN's, the symbols RIPLE and DABB and the location at which they are referenced in the main program, are punched on the object tape or cards along with the rest of the assembled main program. In a similar fashion, when the subroutine is assembled, the symbols RIPLE and DABB and their values are punched along with the subroutine.

As the main program and subroutines are loaded, the loader accumulates a table of references to symbols and their values. This information is used by the loader to link the main program and subroutine by replacing every reference to RIPLE and DABB by the values passed on from the subroutine by the ENTRY Assembler instruction. Note that the General Loader or another linking loader must be used to load the object tape for any program involving ENTRY's or EXTRN's.

Data Definition Instructions

There are three Data Definition instructions, Define Constant (DC), Define Byte (DB), and Define Storage (DS). These Assembler instructions provide a convenient means to define and reserve data storage.

DC - Define Constant

NAME	OPERATION	OPERAND
A symbol optional	DC	One or more operands separated by commas

The DC Assembler instruction is used to define constants and generate actual data. These constants may be character, hexadecimal, decimal, address, or floating-point constants. The type of constant is indicated by a prefix code.

CODE	CONSTANT TYPE	MACHINE FORMAT
C	Character	a byte of seven-bit ASCII Character Code Per Character
X	Hexadecimal	16-bit binary
H	Decimal	16-bit binary (two's complement notation)
A	Address	16-bit binary
E	Floating-Point Single-precision	32-bit binary (floating-point)
D	Floating-Point Double-precision	64-bit binary (floating-point)

C - Character Constant. The character constant can be any length. It must be enclosed in single quotation marks and preceded by a C.

NAME	OPERATION	OPERAND
MESG1	DC	C'LOAD THE TAPE'
	DC	C'EXECUTE AT 19FE'

Each character is translated into one eight-bit byte of storage. If an odd number of characters is specified, a blank character is automatically appended. This maintains halfword boundary alignment for any following machine instructions. If only one character appears between the quote marks, the eight-bit byte is left justified in the halfword, with the code for blank X'20' in the right half. In general, all characters are translated into seven-bit ASCII code, with the most significant bit zero. As an example of this alignment process, the following two data definition instructions are equivalent in length. Each instruction generates 14 bytes of data.

NAME	OPERATION	OPERAND
	DC	C'AN ODD NUMBER'
	DC	C'AN EVEN NUMBER'

X - Hexidecimal Constant. A hexadecimal constant can be from one to four digits. The hexadecimal digits are:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

The hexadecimal constants must be enclosed in single quotation marks and preceded by an X. Examples are:

NAME	OPERATION	OPERAND
DATA1	DC	X'1FE'
	DC	X'C800'

The hexadecimal constant is converted to a properly aligned 16-bit halfword. If fewer than four digits are specified, the digits are right justified and leading zeros generated. For example, the following data constants are equivalent and result in a 16-bit data constant, X'001C'.

NAME	OPERATION	OPERAND
	DC	X'1C'
	DC	X'01C'
	DC	X'001C'

H - Halfword Decimal Constants. A decimal constant can be from one to five digits plus sign. They cannot exceed +32,767 maximum or -32,768 minimum. The decimal digits are enclosed in single quotation marks and preceded by the letter H. No commas marking the thousandths position should be included.

DC H'-792', H'-30000'

The decimal constant is converted to a properly aligned, right justified 16-bit integer in two's complement notation.

H'32767'	generates	X'7FFF'
H'-32768'	generates	X'8000'
H'-1'	generates	X'FFFF'
H'-0' or H'0'	generates	X'0000'
H'+1''	generates	X'0001'

A - Address Constant. An address constant is a storage address that is translated into a constant. It is a relocatable or absolute constant as determined by the combination of symbols and constants in the expression. Unlike other constants, the address constant is enclosed in parentheses and preceded by the letter A.

NAME	OPERATION	OPERAND
	DC	A(LOOP+2)
	DC	A(TABLE)
	DC	A(TOP-BOTTOM)

The address constant stored is relocatable or absolute as determined by the rules given in Table 11-10. The following examples show how a single DC instruction can be used to define different types of data. Each operand is separated from the next with a comma.

NAME	OPERATION	OPERAND
DATUM1	DC	X'0F00', C'ABCD'
MSG2	DC	C'A MESSAGE', H'132'
	DC	A(ARGA1), A(HEX-16), X'39'

Decimal constants and address constants may be created without the H' ' and A () notation if desired. For example:

NAME	OPERATION	OPERAND
	DC	123, H'123'
	DC	SAM, A(SAM)
	DC	TOP+39, X-Y

Note that although the address constant A () notation is optional in a DC constant, the A () notation around a symbol in a Machine instruction operand is illegal.

E - Floating-Point Constant (Single-Precision). The source form of the floating-point constant consists of a decimal number, as formatted below, enclosed in single quotes and preceded by the letter E. The format of the decimal number is as follows:

1. An optional leading plus sign or a minus sign.
2. One or more decimal digits that may include a decimal point.
3. An optional E character followed by an optional leading plus sign or a minus sign and one or two decimal digits, denoting a power of ten.

NAME	OPERATION	OPERAND
	DC	E'±(decimal number)E±NN'

If, however, the decimal number specified cannot be fully expressed in six hexadecimal digits with the resultant exponent of base 16 in excess 64 notation, (for example E'1234567E73'), the proper order of magnitude results, but only six hexadecimal digits of precision can be maintained for single precision. That is, decimal numbers in the range from approximately 5.4×10^{-79} to $7.2 \times 10^{+75}$ can be represented in the above format but only with six hexadecimal digits of precision for single precision floating-point numbers. The conversion is accurate to the most significant six hexadecimal digits.

Each single-precision floating-point data constant (E) entry is translated by the Assembler into a floating-point number in a specified binary representation requiring two halfwords.

There cannot be any blanks within or between E constants, and they must be separated from each other with a comma as in the last two examples below:

NAME	OPERATION	OPERAND
	DC	E'7.2E+75' approximate maximum
	DC	E'5.4E-79' approximate minimum
	DC	E'7.1E+75'
	DC	E'5.5E-79'
	DC	E'+127.47E-45'
	DC	E'-4.007E0'
	DC	E'123456'
	DC	E'1E-74', E'1E-75'

The Assembler produces an error flag when any of the following occurs:

ERROR FLAG	ERROR
F	Multiple decimal points occur before the number is terminated, or an E encountered.
F	Any decimal point occurs after the E is encountered.
T	The specified power of ten is not in the range -99 to 99.

ERROR FLAG	ILLEGAL EXAMPLES
F (Format error)	DC E'10.03.49'
F	DC E'10.03E4.0'
F	DC E'2DA8E-30'
F	DC E'10,000'
T (Truncation error)	DC E'1E-100'
T	DC E'478E+100'

Single precision numbers whose magnitude exceeds the largest possible number are converted to the largest floating-point number which is X'7FFF', X'FFFF' for positive values and X'FFFF', X'FFFF' for negative values. Numbers whose magnitudes are less than the smallest possible number are converted to true floating-point zero, which is X'0000', X'0000'.

NAME	OPERATION	OPERAND
	DC	E'7.3E+75'
	DC	E'5.3E-79'
	DC	E'1E-99'
	DC	E'1E+99'
	DC	E'73E+76'
	DC	E'123456E83'

D - Floating-Point Constant (Double Precision). The double precision floating-point constant source format is identical to single-precision except that the decimal number enclosed in single quotes is preceded by the letter "D".

NAME	OPERATION	OPERAND
	DC	D'±(decimal number)E±NN'

Each double precision floating-point data constant is translated by the Assembler into a specified binary notation requiring four halfwords having the following format:

S	X		F1	F2
	F3	F4	F5	F6
	F7	F8	F9	F10
	F11	F12	F13	F14

Where:

S = Sign of the fraction

X = Exponent of the hexadecimal base, in excess 64 notation

and $F_1 \cdot 16^{-1} + \dots + F_{14} \cdot 16^{-14}$ = fractional number where each F_i = a hexadecimal digit.

For S = 0, the value of the number N generated is:

$$N = + \left\{ F_1 \cdot 16^{-1} + F_2 \cdot 16^{-2} + \dots + F_{14} \cdot 16^{-14} \right\} \cdot 16^X$$

For S = 1

$$N = - \left\{ F_1 \cdot 16^{-1} + F_2 \cdot 16^{-2} + \dots + F_{14} \cdot 16^{-14} \right\} \cdot 16^X$$

The range of values, rules for source formatting, and error notification are the same as those mentioned for single-precision floating-point constants. However, precision is expanded to fourteen hexadecimal digits as opposed to six. The Assembler uses double and half precision arithmetic with rounding to translate both 'E' and 'D' constants and thereby provides such accuracy as demonstrated in the examples below (i. e. : the conversion is accurate to 14 most significant hexadecimal digits).

SOURCE DATA CONSTANT	HEX DATA GENERATED	VALUE
DC D'1'	4110 0000 0000 0000	1
DC D'10'	41A0 0000 0000 0000	10
DC D'7.2E+75'	7FFE BOE3 AD97 8760	7.2X10 ⁷⁵
DC D'+7.3E75'	7FFF FFFF FFFF FFFF	7.3X10 ⁷⁵ max.
DC D'5.4E-79'	0010 01D1 33A9 49F6	5.4X10 ⁻⁷⁹ min.
DC D'5.3E-79'	0000 0000 0000 0000	True zero
DC D'1267650600228229401496703205376'	5A10 0000 0000 0000	16 ²⁵
DC D'.7888609052E-3'	2810 0000 0000 0000	2-100
DC D'9.094947017729282379150390625E-12'	3710 0000 0000 0000	16 ⁻¹⁰

DB - Define Byte

The Define Byte Assembler instruction is used to define consecutive eight-bit bytes of data. It is only available in the OS Assembler.

NAME	OPERATION	OPERAND
a symbol, optional	DB	One or more operands separated by commas
	DB	*

The label of a DB statement receives the value of the current Location Counter, whether even or odd, absolute or relocatable. The use of the label is restricted in that, if it is used in the operand field of any non-DB type instruction, then the label must be on an even boundary.

The operation mnemonics are the letters DB. The operand field may contain one or more operands, each separated by a comma, and the entire string of operands end with a space or carriage return. There may be an even or odd number of operands.

Each of the fields: Label, Operation, Operand and Comment field, must be separated from each other by at least one space.

Each of the DB operands may be any legal assembler expression which represents an absolute value. For each operand, the least significant 8-bits of the 16-bit value are used to generate one 8-bit data byte, and the Location Counter is incremented by one. A DB statement with n operands, therefore, increments the Location Counter by n bytes. Following a DB statement, the Location Counter may have an odd value. Since the Location Counter must be aligned on a halfword boundary (even value) before using normal instruction statements, a special case of the DB operation (DB *) is provided which enables the programmer to achieve halfword instruction alignment. This special case is discussed below.

When the Assembler extracts the right-most 8-bit data byte from the 16-bit expression value, certain error checking is performed.

An Undefined error flag (U) is generated if undefined symbols are used in the expression. A Format error flag (F) is generated if the expression value is relocatable. A Relocatability error flag (R) is generated if the absolute and relocatable combination rules are not followed. A Truncation error flag (T) is generated if the current Location Counter (*) is not referenced in the expression, and the most significant eight bits of the value are not X'00' or X'FF'. A Truncation error flag (T) is also generated if the current Location Counter (*) is referenced in the expression, and the most significant nine bits of the value are not all zeros or all ones. Note: Since there is only one field for assembly errors per line, the order of error precedence from the high to low is S, F, M, O, T, U, R, E. That is, if the error involved both a U and F error, the F error is shown. In general, for DB operands which do not involve the current Location Counter (*), no error flags result if the value of the expression is in the range -256 to +255. For DB operands which do involve the current Location Counter (*), no error flags result if the value of the expression is in the range -128 to +127.

This type of testing for truncation errors allows the use of DB to generate eight-bit 2's complement numbers. For example, legal DB expressions are:

```

A   DB   X'F7'   generates X'F7'
    DB   128     generates X'80'
    DB   B-*     generates X'02' (The displacement from HERE to B = +2 bytes)
    DB   -1      generates X'FF'

B   DB   A-*     generates X'FC' (The displacement A-B = -5 bytes)

```

The exception to the above rules is the special case DB *, in which an asterisk followed by a space or carriage return is the only argument of the DB statement. This special case is provided to allow the programmer to align the Location Counter properly before using normal machine instructions. This statement should be used following any sequence of DB statements to make the Location Counter an even value. If the Location Counter is already even, then DB * has no effect. If the Location Counter is odd, an all-zero's data byte is generated, and the Location Counter is incremented to the next even value. Failure to observe proper halfword alignment may cause the Assembler to produce object tapes that cannot be loaded correctly. Other restrictions related to DB are that the operands can contain neither EXTRN's nor forward references on PASS1 assemblies. On PASS2 or PASS3 assemblies forward references defined later in the program are permissible. Legal and illegal examples of DB statements are shown below.

LEGAL EXAMPLES				ILLEGAL EXAMPLES			
SOURCE		DEFINED BYTE		SOURCE		ERROR FLAG	
RELZRO	EQU	*		RELZRO	EQU	*	
REL80	EQU	**X'80'		REL80	EQU	**X'80'	
	ORG	X'80'			ORG	REL80	
ABS80	EQU	*		ABS80	EQU	X'80'	
LOOP55	DB	ABS80+32	A0		DB	*-RELZRO	
	DB	X'007E'	7E		DB	ABS80-REL80	R
	DB	64	40		DB	*+2	F
	DB	0	00		DB	C'XY'	T
	DB	-2	FE		DB	C'34'	T
	DB	H'80'	50		DB	256	T
	DB	C'A'	41		DB	-257	T
	DB	C'*'	2A		DB	X'107E'	T
	DB	1+5-2	04		DB	RELZRO	F
	DB	LOOP55	80	A	DB	UNDEF	U
	DB	* NONE GENERATED		B	DB	*+A	R
	DB	R15	0F	C	DB	*-A+B+C	R
	DB	-256	00		DB	E'1E10'	F
	DB	255	FF		DB	D'2.5E, 10'	F
	DB	X'FF'	FF		DB	H'257'	T
	DB	-1	FF		LHI	3, 2	CAUSES TAPE TO LOAD INCORRECTLY
	DB	REL80-RELZRO80					
	DB	*-LOOP55	10		DB	*	
	DB	LOOP55-*	EF			END	
	DB	*-LOOP55-2	10				
	DB	*	00				
	DB	X'FF'80', -3, *-*	80				
			FD				
			00				
	DB	ABS80-LOOP55	00				
	DB	* NONE GENERATED					
			END				

To summarize, the processing sequence for the DB statement is as follows:

1. If the statement is DB *, with no other operands, the Location Counter is tested. If it is even, no action is taken. If it is odd, a zero data-byte is generated, and the Location Counter is incremented by one.
2. If other operands are present, excluding the Location Counter *, the absolute 16-bit value of the expression must be in the range X'FF00' to X'00FF' (-256 to +255) or else a truncation error occurs. If other operands are present, including the Location Counter *, the absolute defined 16-bit value of the expression must be in the range X'FF80' to X'007F' (-128 to +127) or else a truncation error occurs. In any case, the right byte of the 16-bit value is generated as the defined byte and the Location Counter is incremented by one.

DS - Define Storage

NAME	OPERATION	OPERAND
A symbol optional	DS	A defined (even) expression

The DS Assembler instruction is used to reserve storage instruction areas. The value of the expression in the operand entry determines the number of bytes reserved. If a symbol appears as a name, the value of the symbol is the location of the first byte reserved. No data is generated and the storage area reserved is not set to zero. When the operand expression mistakenly requests an odd number of bytes to be reserved, the assemblers force the odd number to one more byte in order to maintain halfword alignment for the assembly of the instruction statement.

Example:

NAME	OPERATION	OPERAND
INAREA	DS	80
OUTPUT	DS	TABLE-TABLE2

Assembler Control Instructions

Assembler control instructions are used to control the Location Counter, the number of passes, between pass stops, printing and punching, conditional or multiple assembly of instructions, and assembly termination. None of these Assembler instructions generate Machine code instruction or constants in the object program.

OPT - Specify Options

NAME	OPERATION	OPERAND
Not used	OPT	One or more operands separated by commas

The OPT statement must be the first statement in the program. The OPT Assembler instruction is used to specify the following assembly options.

- Number of Passes: PASS1, PASS2, PASS3
- Printing: PRINT, NOPRNT
- Punching: PUNCH, NOPNCH
- Between Pass Stop: STOP, GO
- Program Label: LAB=ABCDEF

- Scratch: SCRT (OS Assembler only)
- Source Sequence Check: SQCHK (OS Assembler only)
- Program Contains Floating-Point Data Constants: FLOAT (Basic Assembler only)

Typical OPT statements for both Basic and OS Assemblers might be:

NAME	OPERATION	OPERAND
	OPT	PASS2, PUNCH, GO
	OPT	PUNCH, NOPRNT, PASS1
	OPT	STOP, PRINT, PASS3, PUNCH
	OPT	PASS1, LAB=PROG3

PASS 1, One-Pass Assembly Option. Specifying the PASS 1 option, causes the source tape or cards to be read once by the Assembler. The printed assembly listings and punched object tape are produced in accordance with the punching and printing options that have been specified.

An assembly under the "OPT PUNCH, PRINT, PASS 1" option statement produces an assembly listing, writes the object code, and halts after one pass over the source statements. Object tapes from one-pass assemblies must be loaded by some linking loader, such as the General Loader. Therefore, PASS 1 assemblies should only be specified when it is feasible to use a large linking loader, such as the General Loader at load time and when no forward references appeared within expressions in the source deck or tape.

PASS 2, Two-Pass Assembly Option. Specifying the PASS 2 option causes the source tape or cards to be read twice by the Assembler. The printed assembly listing and punched object are produced during the second pass. As with the PASS 1 option, they are produced in accordance with the punching and printing options that have been specified.

PASS 3, Three-Pass Assembly Option. Specifying the PASS 3 option causes the source tape or cards to be read three times by the Assembler. The principal use of the PASS 3 option is to produce two pass assemblies of a program using a Teletypewriter, which cannot accept printing and punching simultaneously. The three pass assembly is identical to the two pass except that the assembly listing is produced during the second pass and the object program is produced during the third pass.

PRINT, Print Assembly Listing Option. Specifying the PRINT option causes the assembly listing to be printed during:

- the first pass of a one-pass assembly,
- the second pass of a two-pass assembly, or
- the second pass of a three-pass assembly.

NOPRNT, No printing option. Specifying the NOPRNT option suppresses any printing of the assembly listing.

PUNCH, Punch object option. Specifying the PUNCH option causes the object program to be output during:

- the first pass of a one-pass assembly,
- the second pass of a two-pass assembly, or
- the third pass of a three-pass assembly.

NOPNCH, No object program option. Specifying the NOPNCH option suppresses production of the object program.

STOP, Stop after each pass option. Specifying the STOP option causes the Assembler to stop after each pass of the assembly.

GO, Go to the next pass option. Specifying the GO option causes the Assembler to go immediately to the next pass of the assembly without operator intervention.

LAB = nnnnnn, object program label. A program label can be one to six characters; the first character must be an alphabetic character, subsequent characters can be letters or digits. The program label is written on the object tape in symbolic form. When using the General Loader, program labels are typed at load time. When using the OS Loader, program labels are held for use in Memory Map printouts.

The options can appear in any order in the OPT statement, except that for the program label to get written on object tape for PASS1 assemblies, the PASS1 option symbol must precede the LAB=SYMBOL option.

The OS Assembler provides two additional options. Their symbols are "SCRT" for electing a scratch operation and "SQCHK" for electing a source sequence check to assure that source statements are positioned in ascending sequential order.

SCRT, Scratch Option. Specifying the SCRT option causes the OS Assembler to scratch (or to copy) the source read in from the logical SOURCE DEVICE onto the logical SCRATCH DEVICE during pass one. On subsequent passes, the source statements are then read from those copied onto the SCRATCH DEVICE.

SQCHK, Sequence Check Option. Specifying the SQCHK option causes the OS Assembler to compare each source statement's sequence number against the preceding statement's sequence number. The identification sequence number of a source statement must reside in a fixed positional field in Columns 73 to 80 (inclusive) of the 80-character source record. Each successive sequence number must be greater (in ASCII) than the one preceding it. The leading option statement does not have its sequence checked. The OS Assembler's first sequence number is initialized to eight spaces X'2020202020202020', so that the user may elect to fill the entire field, Columns 73 to 80 with a sequence identifier or to allow most significant leading spaces and right-justified numbers. In either case, the least significant digit resides in character position 80 of the source record. Under the SQCHK option, when a source statement contains a sequence number which is illegally equal to or lower than its predecessor, the user is notified with an error message on the List Device, unless printing is suppressed (such as when "NOPRINT" has been opted or when the logical LIST DEVICE has been assigned to a null device).

When a sequence check error occurs, the error message consists of a partial or total listing of the source line in error with the Location Counter preceded by a number sign (#). Note that when the sequence error occurs on PASS 1 of multiple-pass assemblies, a partially assembled listing of the line in error follows the (#) error flag. Other errors present, such as "U" for undefined, should be ignored. The user checks and/or corrects sequence errors by repositioning source statements on pass one, aborts the current assembly process and restarts pass one over again. When it can be determined that the statement out of sequence did not affect assembled data or logic, the user may elect to continue his assembly without correcting the error.

When a sequence error occurs on the listing pass, the sequence error message (#) is followed by a fully assembled line, such that any listing of other errors should be noted.

FLOAT, Floating-Point option. The Basic Assembler provides an additional control option. FLOAT, to define the presence of Floating-Point Define Constants (DC) within the user's source program. The Assembler's floating-point logic requires over 1KB of memory. Not specifying FLOAT in the option statement causes the Assembler to overlay the floating-point logic area with the Symbol Table. This overlay allows an expansion of memory available for entering symbols into the Symbol Table by approximately 1,000 bytes. This means that a larger number of symbols and, therefore, a greater number of source statements can be assembled.

Adding 1,000 bytes of memory to the Symbol Table size increases its capacity to handle an additional 100 six-character symbols, or 125 four-to-five character symbols, or 166 two-to-three-character symbols. That is, in a user program that contains, on an average, as much as one four-to-five character symbol-definition per five source statements, approximately 500 source statements can be assembled in addition to the amount provided floating-point users.

This option is provided mainly for users assembling large non-floating-point programs with the Basic Assembler and limited to the required minimum of 8KB memory.

If no OPT statement or specification of a particular option appears, the assumed options are as follows. Both the Basic and Operating System Assemblers assume:

- PASS2
- PRINT
- PUNCH
- STOP

The Operating System Assembler also assumes no SQCHK option unless specified, and no SCRT option unless specified.

The Basic Assembler also assumes no FLOAT option unless specified.

ORG - Set Location Counter

NAME	OPERATION	OPERAND
Not used	ORG	A defined (even) relocatable or absolute expression.

The ORG Assembler instruction is used to control the Location Counter. ORG causes the Location Counter to be set to the value of the expression in the operand entry. The value is relocatable or absolute as determined by the expression.

The Location Counter is initialized to relocatable zero before each assembly. If no ORG Assembler instruction appears at the beginning of the program, the Location Counter begins at relocatable zero.

Symbols appearing in the operand of the ORG must be previously defined.

The ORG Assembler instruction assures proper halfword alignment for any following machine instructions by always forcing the value of the Location Counter to be even. An odd operand value is forced one byte address less. For example, the following two ORG statements produce a Location Counter value of X'019C'.

NAME	OPERATION	OPERAND
	ORG	X'019D'
	ORG	X'019C'

To obtain a relocatable program, no ORG statement is necessary. A program can be made absolute at any time by using an ORG with an absolute operand, like ORG X'100'. Once a program is absolute, it can be made relocatable again by referring to a previously defined relocatable symbol. For example:

NAME	OPERATION	OPERAND
START	OPT	
	EQU	*
	ORG	X'1000'
	ORG	START+100
	END	

DO - Conditional/Multiple Assembly of an Instruction

NAME	OPERATION	OPERAND
A symbol optional	DO	A single defined absolute expression

The DO Assembler instruction causes the statement immediately following the DO statement to be processed as many times as specified by the value of the expression in the operand entry. If the value is zero, the next statement is skipped. The conditional assembly of instructions and generation of data is often used to configure standard programs at assembly time. For example:

NAME	OPERATION	OPERAND
	.	
	.	
	DO	CNFGR1
	BAL	R15, SUBR1
	DO	1-CNFGR1
	BAL	R1, SUBR3
	.	
	.	

If CNFGR1 has a value of 1, the Branch and Link to SUBR1 is generated. If the value of CNFGR1 is 0, the Branch and Link to SUBR3 is generated.

END - End Assembly

NAME	OPERATION	OPERAND
A symbol optional	END	A defined absolute or relocatable expression (optional)

The END Assembler instruction terminates the assembly of the program. The value of the expression, if present, designates the place in the program where control is transferred after the program has been loaded.

If an expression is not present, no automatic transfer of control takes place after loading. An example follows:

NAME	OPERATION	OPERAND
	ORG	100
PLACE1	LHI	R3, DATA2
	.	
	.	
	.	
	.	
LAST	END	PLACE1

The optional symbol, LAST, points to the next sequential halfword address beyond the object program. After loading this example program, Processor control is automatically transferred to location X'0100' (PLACE1).

OS Assembler Control Instructions

In addition to those pseudo-ops described earlier, the OS Assembler, Program Number 03-025, provides three more Assembler control instructions. They are the assembly pause "PAUSE" statement, the conditional assembly "IF" statement, and the listing title and format control "TITLE" statement.

When the PAUSE, IF, or TITLE statement mnemonics are illegally encountered by the Basic Assembler, Program Number 03-024, the "O" operation error is generated on the line's assembly listing and reservation of four bytes occurs in the object program.

PAUSE - Assembly Pause

NAME	OPERATION	OPERAND
Not used	PAUSE	Not used

The PAUSE OS Assembler control instruction appears anywhere in the user's source program to pause or suspend the assembly process. Upon encountering the PAUSE statement, the OS Assembler requests the Operating System under which it is running to suspend execution of the OS Assembler. The OS then notifies the operator of the encounter by listing a "PAUSE" message on the System Console. The operator intervenes, if and as desired, and/or resumes execution of the OS Assembler by using the OS commands available. The OS Assembler continues its assembly process as modified by the operator intervention. When the operator resumes execution directly after the "PAUSE" message, the OS Assembler continues its assembly process of reading source, etc.

The label and operand fields of the PAUSE Source statement are neither required nor processed by the OS Assembler. The operation mnemonic PAUSE must be preceded by at least one space.

IF - Conditional Assembly

NAME	OPERATION	OPERAND
Not used	IF	A defined expression

The IF OS Assembler control instruction provides conditional assembly capability of segments within modularly constructed programs. The IF Statement provides conditional assembly of a sequential set of instructions as distinct from the aforementioned DO statement which provides conditional assembly (or multiple assembly) of a single instruction.

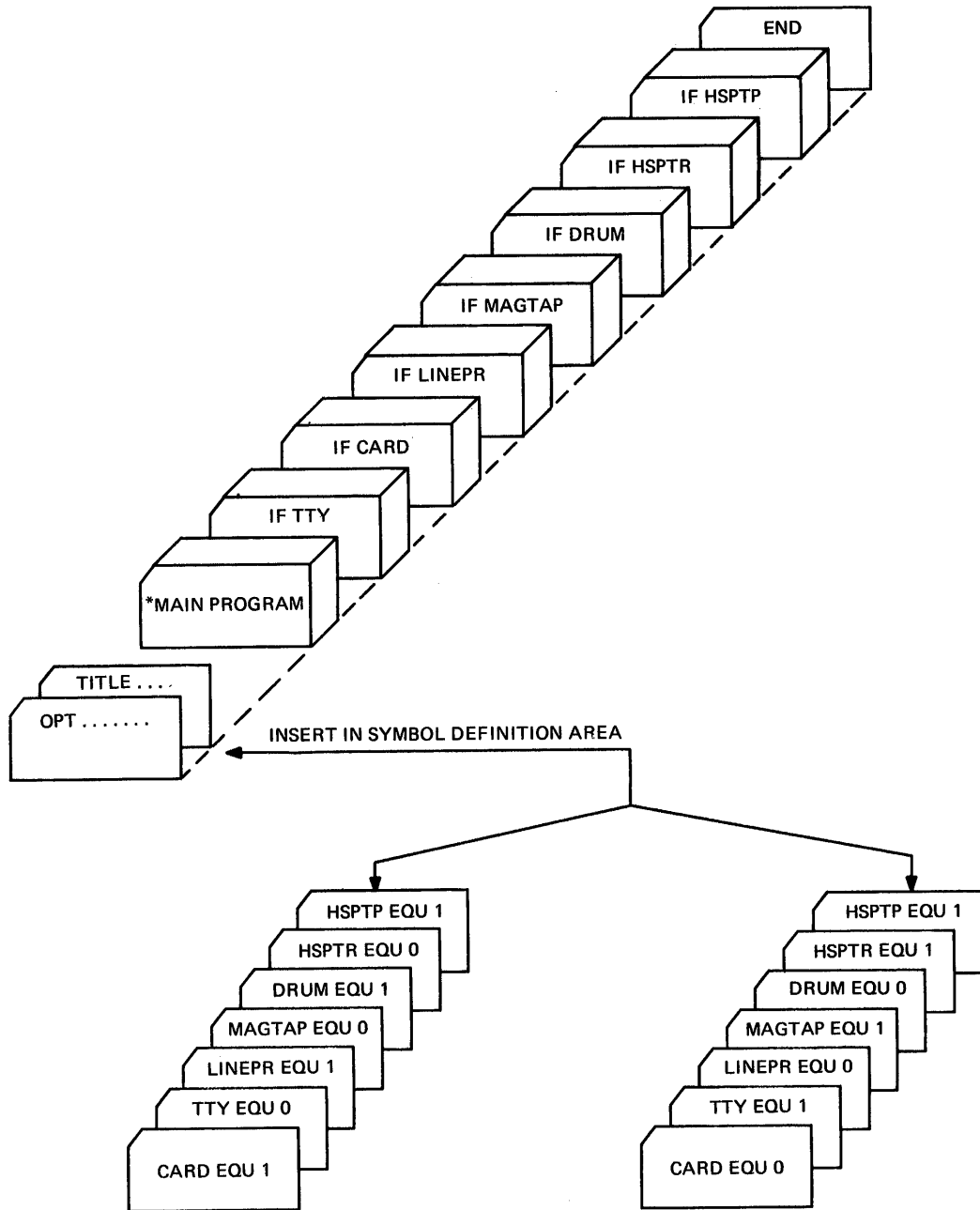
The name field of an IF statement is not required nor processed by the OS Assembler. The operand field contains any defined expression, such as, a single symbol whose definition occurs at the beginning of the source program.

When the IF operand expression has a non-zero value, unconditional assembly of the following source statements take place. When an IF statement is encountered where its operand expression has an absolute zero value, the conditional assembly mode is set into effect. Source statements continue to be read but they do not become assembled. Under conditional assembly mode, source statements are read until either an END statement is read, which terminates an assembly pass, or read to the next IF statement that contains a non-zero valued operand. When another IF statement is read whose operand is non-zero, the Assembler reverts to the unconditional assembly mode and the following statements become assembled at the location where the assembly process left off at any preceding IF (zero) statement. Refer to Figure 7-6, Conditional Assembly Structures.

TITLE - Listing Title and Format Control

NAME	OPERATION	OPERAND
Not used	TITLE	56 characters

The TITLE OS Assembler control instruction provides assembly listing title(s) and format control capability. An assembly listing produced by the OS Assembler contains a header line at the top of each page. The operand of the TITLE statement is placed in this header line. The TITLE statement operand consists of 56 characters not counting leading spaces. However, to maintain consistency throughout the Assembler Language, at least one space must follow the operation mnemonic TITLE. The first non-blank character of the operand is printed in Column 6 on the assembly listing header line.



OS Assembler sequentially assembles the sections preceded by IF statements for: CARD, LINEPR, DRUM, HSPTP; and does not assemble the sections preceded by IF statements for: TTY, MAGTAP, HSPTR.

OS Assembler sequentially assembles the sections preceded by IF statements for: TTY, MAGTAP, HSPTR, HSPTP; and does not assemble the sections preceded by IF statements for: CARD, LINEPR, DRUM.

Figure 7-6. Conditional Assembly Structures

The 72-character page header line on the assembly listing pages are headed by the page number on the upper right corner of 72-column listing pages. Every header line, whether blank or filled, is followed by a blank line to separate it from the printing of assembled instructions. When a single TITLE statement is imbedded in the source program as the first statement or after the option statement, every listing page contains that title in the top header line. During the listing pass, each occurrence of a TITLE statement within the source produces a form feed and change of title in the header line.

Therefore, the format control capability provided can be used two ways:

1. When the user wishes to maintain the same general title throughout his program listing but also wishes to section off certain sets of instructions (such as subroutines), he inserts duplicate TITLE statements with identical operands at those pertinent places within the source.
2. When the user wishes to both change titles and section off sets of instructions on the listing, he inserts multiple TITLE statements with different operands at the applicable places within the source.

7.5.6 Assembly Listing and Object Programs

Refer to Figures 7-7, 7-8, and 7-9 consecutively for a presentation of one program in source form, assembled under the PASS1 option into (zoned standard-loader format) object form, and its assembly listing. Note that the program presented exercises 15 of the 16 loader control items available, from X'0' through X'F' (X'E' is unused by the assemblers). Refer to Figure 7-10 for the general specifications of the Basic and OS Assembler assembly listings.

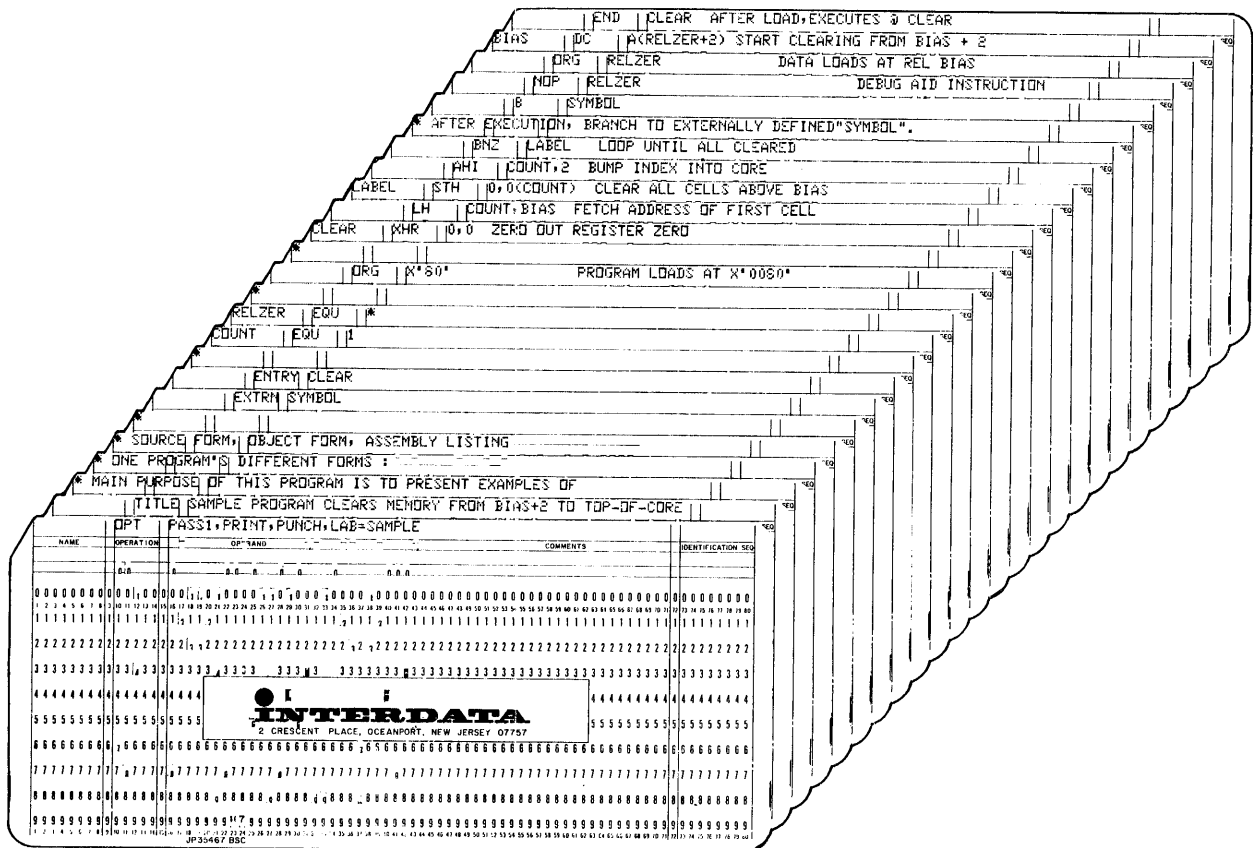
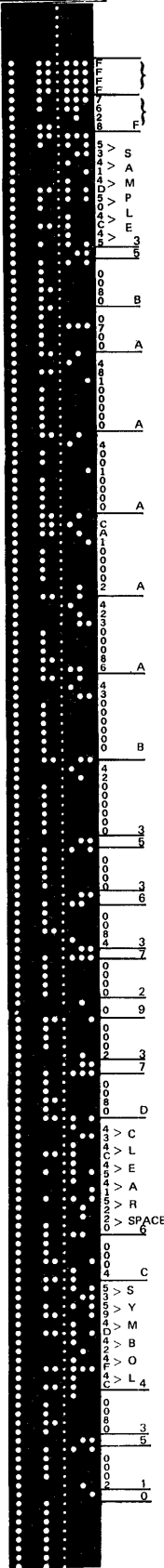


Figure 7-7. Sample Program Source Deck

ZONE CODE	DATA ITEM
--------------	--------------



Record Sequence Number = X'FFFF'

Record Check Sum = X'7628'

Loader Control Item (F) Program Label contained in next 12 data items.

Loader Control Item (3), Toggle from relocatable mode to absolute mode.
 Loader Control Item (5), Load program address X'0080' absolute.
 (ORG X'0080')

Loader Control Item (8), Load 2 bytes absolute into X'0080'
 (XHR 0, 0)

Loader Control Item (A), Load 4 bytes absolute into X'0082'
 (LH COUNT, BIAS)

Loader Control Item (A), Load 4 bytes absolute into X'0086'
 (STH 0,0 (COUNT))

Loader Control Item (A), Load 4 bytes absolute into X'008A'
 (AHI COUNT, 2)

Loader Control Item (A), Load 4 bytes absolute into X'008E'
 (BNZ LABEL)

Loader Control Item (A), Load 4 bytes absolute into X'0092'
 (B SYMBOL)

Loader Control Item (B), Load 4 bytes relocatable into X'0096'
 (NOP RELZER)

Loader Control Item (3), Toggle from absolute mode to relocatable.
 Loader Control Item (5), Load Program address X'0000' relocatable.
 (ORG RELZER)

Loader Control Item (3), Toggle from relocatable mode to absolute.
 Loader Control Item (6), Last Reference address of BIAS = X'0084' absolute.
 (BIAS becomes defined)

Loader Control Item (3), Toggle from absolute mode to relocatable.
 Loader Control Item (7), Definition address of BIAS = X'0000' relocatable.

Loader Control Item (2), Unchain: Place definition value in references, until X'0000' end.
 Loader Control Item (9), Load 2 bytes relocatable in X'0000' relocatable.
 (BIAS DC A(RELZER+2))

Loader Control Item (3), Toggle from relocatable mode to absolute. (END CLEAR statement)
 Loader Control Item (7), Definition address for Global Symbol "CLEAR".

Loader Control Item (D), Definition symbol follows in next 12 data items.

Loader Control Item (6), Reference address for Global Symbol "SYMBOL".

Loader Control Item (C), Reference symbol follows in next 12 data items.

Loader Control Item (4), Load Transfer Address X'0080' (Transfer to X'0080' after load)

Loader Control Item (3), Toggle from absolute mode to relocatable
 Loader Control Item (5), Load program address (next available relocatable location)

Loader Control Item (1), End of Tape signal.
 Loader Control Item (0), Read next record signal, when record contains no (1) control item.

Figure 7-8. Sample Program One-Pass Object Tape

```

* MAIN PURPOSE OF THIS PROGRAM IS TO PRESENT EXAMPLES OF
* ONE PROGRAM'S DIFFERENT FORMS :
* SOURCE FORM, OBJECT FORM, ASSEMBLY LISTING
*
0000R          ENTRY CLEAR
0000R          EXTRN SYMBOL
*
0001          COUNT EQU 1
0000R          RELZER EQU *
*
0080          ORG X'80'          PROGRAM LOADS AT X'0080'
*
0080 0700     CLEAR XHR 0,0      ZERO OUT REGISTER ZERO
0082 4810     LH COUNT,BIAS     FETCH ADDRESS OF FIRST CELL
0000F
0086 4001     LABEL STH 0,0(COUNT) CLEAR ALL CELLS ABOVE BIAS
0000
008A CA10     AH1 COUNT*2      BUMP INDEX INTO CORE
0002
008E 4230     BNZ LABEL        LOOP UNTIL ALL CLEARED
0086
* AFTER EXECUTION, BRANCH TO EXTERNALLY DEFINED "SYMBOL".
0092 4300     B SYMBOL
0000F
0096 4200     NOP RELZER       DEBUG AID INSTRUCTION
0000R
0000R          ORG RELZER      DATA LOADS AT REL BIAS
0000R 0002R   BIAS DC A(RELZER+2) START CLEARING FROM BIAS + 2
0002R          END CLEAR      AFTER LOAD,EXECUTES AT CLEAR
BIAS 0000R
* CLEAR 0080
COUNT 0001
LABEL 0086
RELZER 0000R
* SYMBOL 0094
    
```

Figure 7-9. Sample Program One-Pass Assembly Listing

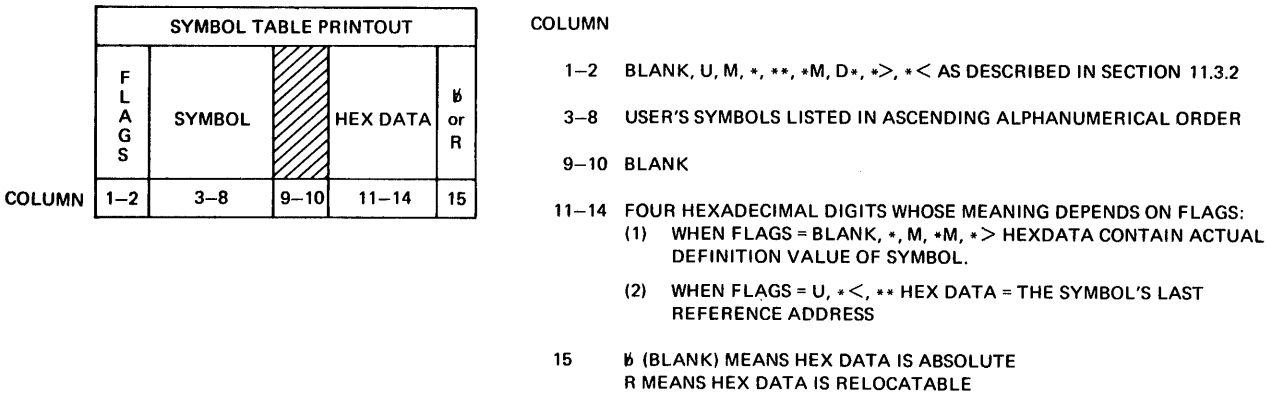
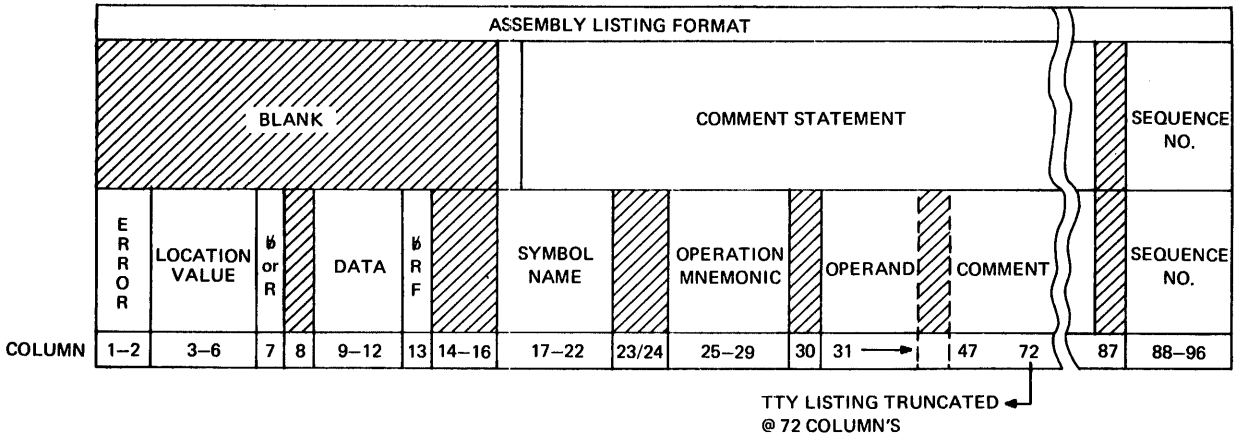


Figure 7-10. Assembler Print Formats

7.5.7 Procedures for User-Defined Mnemonic Op-Codes

A feature has been added to the assembler that permits the user to define his own mnemonics for machine op-codes. This feature is especially useful for those users who have generated the micro-programming necessary for developing new machine instructions for the INTERDATA Processors. This feature also permits the user to assign different mnemonics to already existing machine op-codes.

The method used to define new mnemonics to the Assembler is the EQU statement. The format of the statement is as follows:

NAME	OPERATION	OPERAND
New Mnemonic	EQU	A Constant

The name field of the EQU statement contains the user's desired new mnemonic. The new mnemonic may then be used in the operation field of any succeeding instruction statements. The user's new mnemonic may consist of from one to five characters, the first of which must be a letter and the others must be either letters or numbers. It cannot contain any special characters or blanks between characters.

The operand field of the EQU statement contains a constant, which when interpreted by the Assembler, must have a 16-bit halfword value of the form (in hexadecimal):

nnxy

where nn = hexadecimal digits of an op-code

and x = 0, y = 8 for one word (RR or SF) instruction
 or x = 0, y = 2 for two word (RX or RS) instruction,

or y = C for one word extended (RR or SF) instruction in which x is the Condition Code,
 or y = 3 for two word extended (RX or RS) instruction in which x is the Condition Code.

NOTE

It is suggested that the choice of nn = F0 be restricted to allow compatibility with the HEX-DEBUG programs' use of F000 for breakpoints.

Legal Examples

NAME	OPERATION	OPERAND	GENERATES IN OBJECT PROGRAM
LOOP1	EQU LOOP1	X'2208' 5, 6	X'2256'
MOVE	EQU MOVE	X'3302' 4, 3(7)	X'3347', X'0003'
OP	EQU OP	X'89AC' 4	X'89A4'
LINK	EQU LINK	X'41F3' 2	X'41F0', X'0002'

Illegal Examples

NAME	OPERATION	OPERAND	ILLEGAL BECAUSE
SAM	EQU SAM	X'1238' 5, 6	Third hex digit not 0.
FROG	EQU FROG	X'5555' 4, 3, (7)	Fourth hex digit not legal.
OP	EQU OP	X'89AC' 4, 5`	Too many arguments.
CALL	EQU CALL	X'41F3' SAM(100)	Index value greater than 15.

7.5.8 Basic Assembler Operating Instructions

General Description and Configuration

The Basic Assembler program runs on any of the INTERDATA Processors containing 8KB (8,192 bytes) or more of memory. The Basic Assembler can use any of the following peripheral devices: ASR33 or ASR35 Teletype (TTY), High Speed Paper Tape Reader (HSPTR), High Speed Paper Tape Punch (HSPTP), Card Reader, or Line Printer. The Assembler can be loaded from either a Teletype (TTY) reader or a High Speed Tape Reader (HSPTR). The Assembler reads ASCII-coded source from either a TTY reader or HSPTR, or reads Hollerith coded cards from a Card Reader. It punches binary object output to either a TTY punch device or High Speed Paper Tape Punch (HSPTP). It prints ASCII-coded assembly listing output to either a TTY printer or line printer. When the configuration contains either a TTY or a Card Reader, the Basic Assembler assumes that the TTY is interfaced to the Processor as Device Number 02, and the Card Reader as Device Number 04.

Basic Assembler Program Availability

The Basic Assembler program is identified by Part Number 03-024. The object program is provided as Part Number 03-024M10, a bootstrap binary object paper tape which loads via the 50 Sequence. The assembly listing of the Basic Assembler is provided as Part Number 03-024A13.

Memory Allocation

The Basic Assembler program requires a minimum of 8KB of memory. Refer to Figure 7-11, Basic Assembler Memory Map, for an overall view of the assembler layout in 8KB memory. Approximately 6.4KB is required for basic assembler logic, and 1KB for floating-point conversion logic. Additional memory is required by the Basic Assembler for a Symbol Table. In a minimal system of 8KB, about 600 bytes are available for the Symbol Table when the user requires the Floating-Point option (FLOAT); otherwise, 1600 bytes are available.

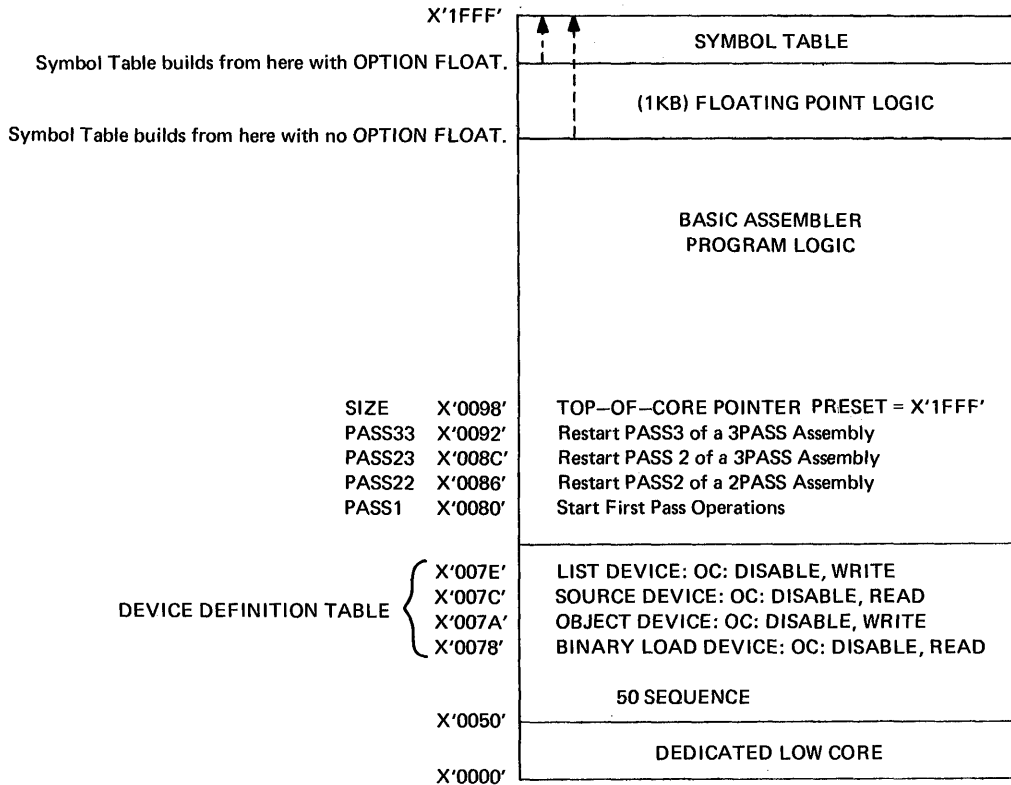


Figure 7-11. Basic Assembler Memory Map

Each unique legal symbol encountered in the user's program is built into a symbol entry which is stored in the Assembler's Symbol Table, building upward from the top of the Basic Assembler program. Therefore, the total memory required by the Basic Assembler to do a particular assembly depends on the number of symbols in the user's program.

The number of bytes required by a symbol entry depends upon the number of characters in the symbol. A one-character symbol requires four bytes, a two or three-character symbol requires six bytes, a four or five-character symbol requires eight bytes, and a six-character symbol requires ten bytes.

The Symbol Table capacity may be expanded further by users who have more than 8KB memory available. The Basic Assembler contains a Top-of-Core pointer residing at X'0098' and pointing to X'1FFF' for 8KB. The Assembler builds its Symbol Table up to this address. Users with 12KB memory should set this pointer to X'2FFF' to use an additional 4KB for their Symbol Tables. Users with 16KB should set this pointer to X'3FFF' to use an additional 8KB for their Symbol Tables, etc.

Refer to Table 7-11, Symbol Table Capacity, for the number of average 4 or 5 character symbols and approximate size of user programs that can be assembled in an 8, 12, or 16KB core memory.

Loading Procedures

The Basic Assembler program object tape, Part Number 03-024M10, is a bootstrap self-loading tape. It does not require any loader program tape to be loaded first. It is loaded by the memory resident 50 Sequence Loader as shown in Table 7-3. Prior to loading the Basic Assembler, a 50 Sequence Loader must be entered into memory, along with the address of the selected Binary Input Device at X'0078'. That is, location X'0078' should contain X'1399' to load 03-024M10 from the High Speed Paper Tape Reader, or X'0294' to load from the TTY reader. Refer to the Section on I/O Device Selection for entering the user's desired Assembler I/O device selections at this same time.

TABLE 7-11.
SYMBOL TABLE CAPACITY

Top-of-Core Pointer	Approximate Core Available for Symbol Table	No. Average* Sized Symbols in User Program	Size** Of User Program
X'1FFF' for 8KB	<u>OPTION FLOAT</u> 640 bytes	80	400 Source Statements
X'1FFF' for 8KB	1640 bytes	205	1,020 Source Statements
X'2FFF' for 12KB	5640 bytes	705	3,525 Source Statements
X'3FFF' for 16KB	9640 bytes	1,205	6,025 Source Statements
<p>*An average sized symbol (each of which requires eight bytes) consists of four or five characters</p> <p>**Size of User Program assumes that a user program contains one symbol definition per five source statements.</p>			

Given the 50 Sequence Loader in memory, the steps required to load the Basic Assembler Tape, 03-024M10, are:

1. Place the bootstrap tape in the tape reader anywhere on the blank leader portion.
2. Set Data/Address Switches to X'50', Select ADRS Mode, and depress EXECUTE.
3. Depress INITIALIZE.
4. Select Run Mode, and depress EXECUTE.
5. If a Teletype is being used as the load device, manually start the tape motion by moving the TTY reader switch to START on the ASR-33 TTY, or to RUN on the ASR-33 TTY. When approximately a foot of tape data has been read, the right half of Register Display 2 flashes to indicate that the tape is actually loading. If this does not occur, check to see that the loading procedures were followed correctly; e.g. that the 50 Sequence Loader is intact and that location X'0078' is set correctly.
6. If the reader stops and the Processor halts before the end of the tape is reached, an error has been detected. In this case, reposition the tape for the previous record gap and push EXECUTE to reread the previous record.
7. When all of the program has been loaded, the tape stops, and Processor control is transferred directly to the Assembler, which halts.

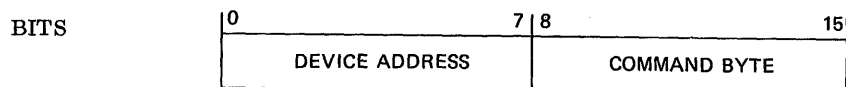
Note that during the bootstrap loading process, memory locations from X'1DF0' to X'1FFE' are used.

I/O Device Selection

The Basic Assembler is programmed to use the Device Definition Table, located in low memory at X'0078' through X'007E'. A physical device address and the device Output Command is found at each of these locations. This information is entered into memory by the user and is necessary for the Basic Assembler to accomplish its I/O operations, i. e., data transfers between the peripherals and the Processor. The halfword at location X'0078', Binary Input Device, is only used to load the Assembler. The Assembler does not use this halfword during its assembly operations.

The three halfwords that must be set prior to executing the Basic Assembler are those it uses for Binary Object Device at X'007A', Source Input Device at X'007C', and List Device at X'007E'.

Each halfword of I/O information in the Device Definition Table has the following format:



Refer to Table 7-12 for a summary of the selection of entries appropriate to the Basic Assembler.

The Basic Assembler reads ASCII coded source paper tape from either a TTY reader or High Speed Paper Tape Reader or it reads Hollerith coded source cards from a Card Reader.

The assembled binary object data is written to either a TTY punch or High Speed Paper Tape Punch.

The assembly listing is printed on either a TTY printer or a Line Printer.

Therefore, when using the Basic Assembler, the user must select either a TTY or HSPTP for his object device, either a TTY reader, HSPTR, or Card Reader for his source device, and either a TTY or Line Printer for his list device.

TABLE 7-12.
I/O DEVICE DEFINITION TABLE SELECTION FOR BASIC ASSEMBLER

Location	Name	I/O Device Selection	
X'007A'	Binary Object Device	0298	TTY Punch
		1392	HSPT Punch
X'007C'	Source Input Device	0294	TTY Reader
		1399	HSPT Reader
		04A0	Card Reader
X'007E'	List Device	0298	TTY Printer
		62C0	Line Printer

Source Format

The Basic Assembler program, Program Number 03-024, accepts source statements written in INTER-DATA Assembler Language.

This program does not recognize the Sequence Check (SQCHK), the Scratch (SCRT) options (specific to the OS Assembler), the Assembly Pause (PAUSE), Listing Format and Title control (TITLE), or the conditional assembly (IF) OS Assembler Control instruction statements.

It recognizes only those instruction statements specified in Section 7.5.4.

Refer to Table 7-13 for the general sequence or order of source statements with a program.

When using the Basic Assembler to assemble source paper tape, the tape may contain any number of statements. Each statement can contain any number of characters (up to 80 legal characters are processed) followed by a carriage return. The characters should be punched in ASCII code. The most significant bit of each character is ignored by the Assembler; therefore, either the seven-bit or eight-bit form of ASCII is acceptable. Blank tape, rub outs, line feeds, and all characters whose seven-bit ASCII code is less than X'20', a space, are ignored. That is, the Assembler's character set consists of the alphanumeric characters A-Z, 0-9, and any ASCII character printable on the TTY printer.

TABLE 7-13.
SOURCE PROGRAM FORMAT

Type Of Statement	Function	Ranking Order
OPT	Option Control	First or not at all
ENTRY	Global Symbol Definition	Prior to symbol's use
EXTRN	Global Symbol Reference	Prior to symbol's use
EQU	Local Symbol Definition	Prior to label's use and after operands' symbol definition.
ORG	Location Specification	Inserted to define load location where necessary. Optional if whole program sequentially relocatable from relocatable zero.
DO	Conditional/Multiple Instruction Assembly	Just prior to the conditional/multiple assembled instruction.
(MACHINE INSTRUCTIONS) (*COMMENT STATEMENTS)	Program Logic Documentation Aid	Intermixed
DC	Define Constants	
DS	Reserve Storage	
END	End of Program Signal	Required as last statement.

If more than 80 legal characters appear in one source statement, the Assembler processes only the first 80 characters but continues reading tape until a terminal carriage return ends the statement. Statements should be separated on the source tape by at least five or six non-printing characters. This statement separation is required due to the start/stop characteristics of a Teletype tape reader. Source tapes typically use eight blank rows between records. Actually, any non-printing character other than carriage return is sufficient. The Basic Assembler, therefore, assembles source paper tapes created by TIDE, by OS TIDE, or those created manually on the TTY punch.

When using the Basic Assembler for assembling from the Card Reader, Hollerith coded characters are converted to bytes of seven-bit ASCII code. The card input driver halts for overflow or motion errors so that the Assembler never processes cards known to be in error. However, when illegal card codes are read, they become converted to an ASCII asterisk, "*". Assembly listings which contain intermittently scattered "*" characters are an indication of card reader errors.

Option Control

The INTERDATA Basic Assembler provides a one-pass, two-pass, or three-pass assembly. It is preset to provide a PASS2 assembly with the PRINT, PUNCH, STOP, and no FLOAT options in effect when no option statement is contained in the source tape or card deck. At assembly time, the programmer may choose the number of passes and other options desired for the assembly. The options are specified with the "OPT" control statement which must be the first source statement of the source program. Refer to Table 7-14 for a summary of Assembler operations that occur during a one, two, or three-pass assembly

The user specifies the symbols PASS1, PASS2, or PASS3 in the option statement operand for a one-pass, two-pass, or three-pass assembly respectively.

TABLE 7-14.
BASIC ASSEMBLER OPERATIONS

Pass Number	Assembly Type		
	PASS1	PASS2	PASS3
1	Read Source Write Listing Write Object	Read Source	Read Source
2		Read Source Write Listing Write Object	Read Source Write Listing
3			Read Source Write Object

Of the three types, a one-pass assembly is fastest, but restricts the programmer's use of forward references within expressions at programming time. For example, such expressions as SYMBOL + 10 are illegal for one-pass assemblies when SYMBOL is not defined because it is a forward reference. Additionally, a one-pass assembly produces larger object program tapes. One-pass assemblies produce a listing in which the forward references are not defined. Two-pass and three-pass assemblies produce a complete listing of the object code which is more useful at debugging time.

Both the assembly listing and object tape are produced simultaneously on a single-pass assembly, or on the second pass of a two-pass assembly. A three-pass assembly permits the printing of the assembly listing on the second pass and the punching of the object on the third pass.

- PRINT Specifies that an assembly listing is desired.
- NOPRNT Suppresses printouts of both the assembly listing and symbol table.
- PUNCH Specifies that an object tape is desired.
- NOPNCH Suppresses object output.
- STOP Causes the Assembler to pause between passes after the operator is notified of the next pass.
- GO Causes the Assembler to immediately read source after the operator is notified of the next pass.
- LAB=SYMBOL Produces an object program label. For one-pass assemblies, the option PASS1 must precede LAB = SYMBOL.
- FLOAT Causes the Assembler to maintain its floating-point conversion logic by adjusting the Symbol Table pointer above the floating-point routines.

The Basic Assembler automatically expands its Symbol Table capacity by the 1,000 bytes, originally contained for floating-point conversion, during any assembly that does not specify FLOAT in its first source option statement.

Users who are limited to exactly 8KB of memory and who desire the assembly of either single or double-precision floating point data constants are required to specify the option symbol FLOAT in their option statement. This option statement must be the first source statement in the user's program. When the Basic Assembler does not see the option, OPT FLOAT, etc., as the first source statement of any program assembly, its floating point conversion logic is overlaid by the Symbol Table.

If the user does specify FLOAT and the floating point has already been destroyed, he is informed by the forced printing of the OPT statement with an "F" format error. The Basic Assembler would have to be reloaded once this error occurs in order to re-establish the floating point logic in memory.

Users requiring the FLOAT option are limited as to the number of symbols the Basic Assembler can process when using only 8KB memory and may need to assemble large programs in smaller segments.

Note that all programs containing floating point constants must individually specify the FLOAT option, and all programs using FLOAT must be assembled consecutively after loading the Basic Assembler.

Operating Procedures

After loading the bootstrap Basic Assembler program tape, 03-024M10, the Processor halts. If it is known that the Device Definition Table is set up properly in memory, the user readies the I/O devices as needed and executes the Basic Assembler by depressing EXECUTE at the Control Panel. To ready the I/O devices as needed means to power-up, bring on-line, supply with paper, etc., those devices that are to be used during the particular pass to be run. For example, for a single-pass assembly to produce both an assembly listing and object tape, the user must ready the list device and object device, in addition to the source device. For a two-pass assembly, the object device need not be readied until the second pass.

Once the Device Definition Table is set up, the following procedures may be used to start execution of the Basic Assembler's first pass operations:

1. Set the Data/Address switches to X'0080', the Basic Assembler's ORIGIN.
2. Select ADRS Mode and depress EXECUTE.
3. Select Run Mode and depress EXECUTE.

Upon execution at ORIGIN, the Assembler issues a form feed to the List Device and prints the message "PASS1" and begins reading source to build its Symbol Table.

Once the assembly starts, the procedures vary according to the number of passes specified in the OPTION statement. Refer to Figures 7-12, 7-13, and 7-14 for a summary of the particular procedures to follow.

ASSEMBLY UNDER PASS1 OPTION (NO FLOAT)	
OPERATOR	BASIC ASSEMBLER
Ready System Load Basic Assembler Set I/O Device Definitions Ready I/O Devices Execute at X'0080'	Prints PASS1 Clears SYMBOL TABLE Reads Source till END Statement Expands SYMBOL TABLE over FLOAT logic Prints Assembly Listing Writes Binary Object Program Lists SYMBOL TABLE Halts at X'0080' for next assembly

Figure 7-12. PASS1 Operations

ASSEMBLY UNDER ASSUMED OPTIONS*	
OPERATOR	BASIC ASSEMBLER
Ready System Load Basic Assembler Set I/O Device Definitions Ready I/O Devices Execute at X'0080' Ready Object Device Re-ready Source Depress EXECUTE	Prints PASS1 Clears SYMBOL TABLE Reads Source till END Statement Expands SYMBOL TABLE over FLOAT logic Lists SYMBOL TABLE Prints PASS2 Halts Reads Source till END Statement Prints Assembly Listing Writes Binary Object Program Halts at X'0080' for next assembly
*Assumed Options: PASS2, PRINT, PUNCH, STOP, (no FLOAT)	

Figure 7-13. PASS2 Operations

ASSEMBLY UNDER PASS3, STOP, FLOAT OPTION	
OPERATOR	BASIC ASSEMBLER
Ready System Load Basic Assembler Set I/O Definitions Ready I/O Devices Execute at X'0080' Re-ready Source Depress EXECUTE Ready Object Device Re-ready Source Depress EXECUTE	Prints PASS1 Clears SYMBOL TABLE Reads Source till END Statement (SYMBOL TABLE does not overlay FLOAT logic) Lists SYMBOL TABLE Prints PASS2 Halts for next pass Reads Source till END Statement Prints Assembly Listing Lists SYMBOL TABLE Prints PASS3 Halts for next pass Reads Source till END Statement Writes Binary Object Program Lists SYMBOL TABLE Halts at X'0080' for next assembly

Figure 7-14. PASS3 Operations

As the Basic Assembler reads the source program, it performs the operations appropriate to the current pass, as shown in Table 7-14.

Each pass proceeds until an END statement is encountered. When this occurs, the current pass is completed. When the END statement is read during PASS1, the Assembler prints out the Symbol Table. All undefined symbols are preceded by a U error flag. All multiple defined symbols are preceded by an M error flag. All improperly used EXTRN/ENTRY symbols are preceded by the asterisk combined with specific error flags such as: *<, *>, *M, *D, ** which are described in Section 7.5.2 and shown in Table 7-7. The source program can be corrected at this time and the first pass restarted.

If an excess number of symbols is read by the Assembler, Symbol Table overflow is shown at the time it occurs by listing the source statement that caused the overflow. These source statements are flagged with an S error flag. When this error occurs, the user must restart and expand his Symbol Table size by discarding the FLOAT option, if he has not already done so, by adjusting the Assembler's Top-of-Core pointer or reduce the number/size of symbols in the user program to obtain a correct assembly.

When the current pass is not the final pass of an assembly, the next pass is identified with a PASS message, and the Assembler halts when the STOP option is in effect. To begin reading source for the next pass, depress EXECUTE after re-readying the Source Input Device with the source tape or card deck. When the GO option is specified in the OPTION statement, the next pass is identified with a PASS message, and the Assembler begins reading source without operator intervention.

The above procedures are repeated for each pass. When the completed pass is the final pass, the Assembler halts at its origin. In this case, if EXECUTE is depressed, the Symbol Table is cleared, the Assembler prepares itself for another assembly, prints PASS1 and starts to read source. Rather than proceed with another program, however, the Assembler can be restarted on PASS2 or PASS3 if desired. The restart addresses are:

PASS 1	X'0080'
PASS 2 of 2	X'0086'
PASS 2 of 3	X'0092'

When all I/O data transfers take place correctly, the Basic Assembler clears the lights in Register Display 2 (lower right byte) on the Control Panel. Whenever a peripheral I/O device cannot transfer data correctly, the Assembler writes into Register Display 2, that peripheral's eight-bit physical device address. The status byte received from the device is contained in General Register 5.

Except for some errors that occur with a Card Reader, the Assembler remains in a sense status loop with the Processor WAIT light off until the condition is corrected, such as bringing on-line a device that was off-line. The Assembler then continues its input or output operation where it left off, transferring the last character available prior to the error condition. When this is not feasible, as in the case of reading tape from a TTY tape reader, and the tape has moved past that character, the source tape may be repositioned manually to the beginning of the last source statement and the Basic Assembler may be restarted at location X'0114' to reassemble that line from its beginning.

Whenever the Assembler detects an I/O error in a card reader data transfer, the Assembler does not process the card in error. Card reader I/O errors must be handled in one of two ways. For a card causing a Pick Fail error, the user must reproduce the bottom card in the hopper if the front edge is mutilated, replace it in the hopper, depress MOTOR-on and depress the card reader START button. For any other card error where a card either passes entirely or partially through the read station, the user must re-read the card that passed into the stacker or got jammed in the read station. When the WAIT light is on after an I/O error, the Assembler has halted the CPU due to Motion Error or Error Overflow. In this case, after the user has transferred the last (TOP) card from the stacker to the bottom of hopper; he depresses both the MOTOR-on and the START button, and must also depress EXECUTE on the Processor.

Assembly Listing and Object Programs

The physical formats of both the assembly listing and object programs differ and depend upon the respective I/O Device Selection in the Device Definition Table.

The assembly listing, when written to a line printer, contains 96 columns of print per listing line. However, on the TTY printer, only 72 columns are printed. The INTERDATA Assemblers allow free formatted source which, upon tabulation with the binary object code generated, produces an assembly listing having an aligned format as described in Section 7.5.3.

The object program contains approximately 30 inches of blank paper tape as leader.

On a High Speed Paper Tape Punch, the object program is punched in INTERDATA standard non-zoned loader format. That is, the program tape is segmented into records of 109 characters with 108 data bytes preceded by a byte of X'F0' and each record is separated from the next by a gap of eight frames of blank tape. When absolute, the object tape is designated M17, when relocatable M16.

On a TTY punch, the object program is punched in INTERDATA standard zoned loader format. That is, the program tape is segmented into records of 216 characters with 108 data bytes using two tape characters per data byte. Each record is separated from the next by a gap of eight frames of blank tape. When absolute, the object tape is designated M08, when relocatable M09.

7.5.9 OS Assembler Operating Instructions

General Description

The OS Assembler accepts the source form of a program, consisting of a sequential set of source statements as described in Section 7.5.3. This section describes the operations of, and the operating procedures for, the OS Assembler, Program Number 03-025.

The INTERDATA OS Assembler program provides users with extended assembler capability in an INTERDATA OS environment. The OS Assembler features:

1. Conditional assembly of modularly segmented source programs by means of the OS Assembler "IF" statement.
2. Automatic page numbering on the listing.
3. Listing format control and header line page titling by means of the "TITLE" statement.
4. Provision of an optional SCRATCH capability by means of specifying "SCRT" in the option statement.
5. Optional selection of a source statement sequence number check for ascending alpha-numerical identification numbers of eight characters positioned in the last eight bytes of an 80 byte source record.
6. Use of double-buffered object data output operations that occur simultaneously with source line reads and listings, executing under an OS that permits I/O to overlap processing.
7. Use of controlled assembly pauses by means of the "PAUSE" statement.

Configuration

The OS Assembler program runs on an INTERDATA Processor with 16KB or more of memory using an INTERDATA Operating System. The OS Assembler does not access peripheral devices directly with Read/Write instructions. It performs all I/O through the OS to logical devices having pre-assigned logical unit numbers. Input/Output operations are initiated by Supervisor Call (SVC) instructions

which, upon execution, are interpreted by the OS to accomplish the direct data transfer. Thus, the OS Assembler may be used with those I/O devices for which the OS provides drivers.

This section describes the application and requirements of the logical, as differentiated from physical, devices required for running an assembly in an OS environment.

The OS Assembler can utilize, depending upon the options selected, up to five logical devices as follows: System Console Device, Source Device, Object Device, List Device, and Scratch Device.

OS Assembler Program Availability

The OS Assembler program tape is provided as Part Number 03-025M16, a relocatable binary object paper tape in INTERDATA standard-loader, non-zoned format. Standard loader formats are described in Loader Description Manual, Publication Number 29-231.

Memory Allocation

The OS Assembler program requires approximately 9KB of memory for its Assembler logic. Additional memory is required by the OS Assembler for its Symbol Table. Refer to Figure 7-15 for a memory map of the OS Assembler layout.

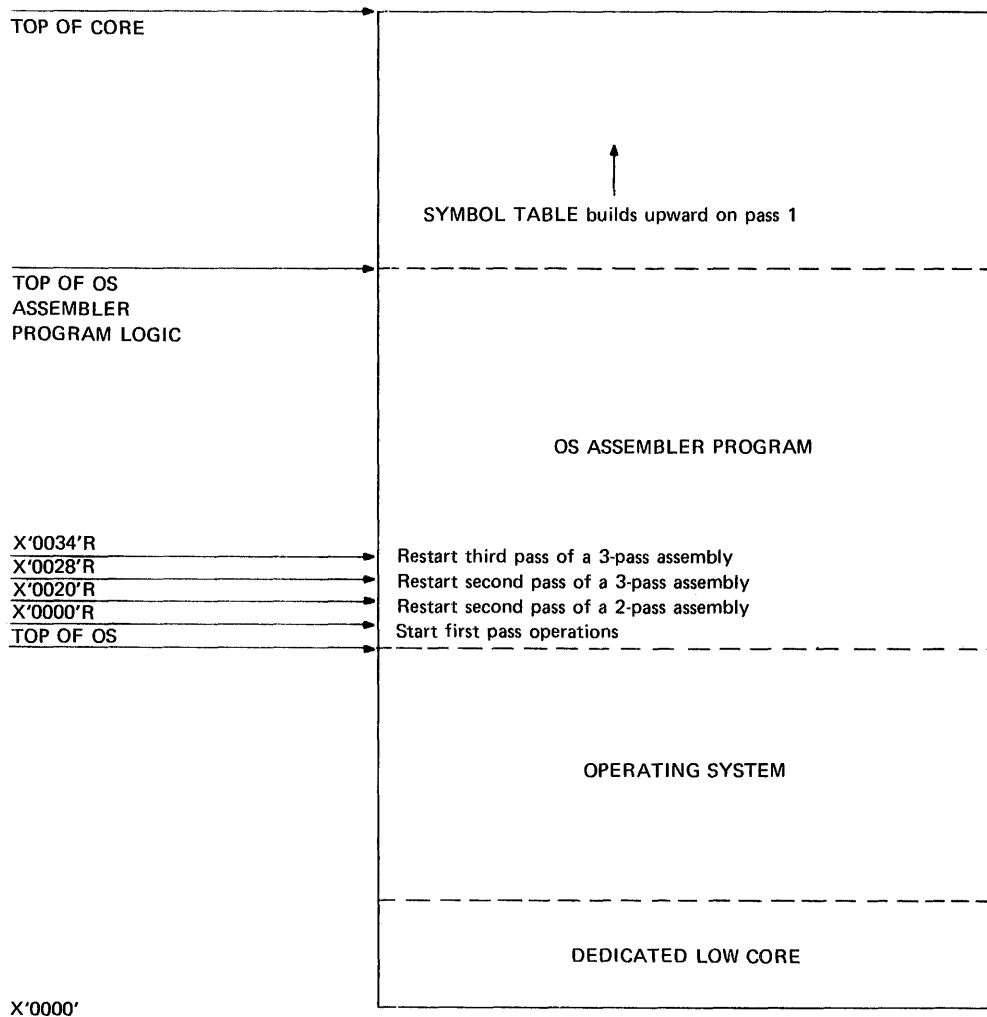


Figure 7-15. OS Assembler Memory Map

Each unique legal symbol encountered in the user's program is built into a symbol entry which is stored in the Assembler's Symbol Table, building upward from the top of the OS Assembler program. Therefore, the total memory required by the OS Assembler depends on the number of symbols in a user's program.

The number of bytes required by a symbol entry depends upon the number of characters in each specific source symbol. A one-character symbol requires four bytes, a two or three character symbol requires six bytes, a four or five character symbol requires eight bytes, and a six character symbol requires ten bytes.

In an INTERDATA OS environment, with a 16KB system, where the OS Assembler takes 8KB, there is 3KB to 5KB available for the Symbol Table. Refer to Table 7-15 for the number of average four to five character symbols acceptable in a source program and approximate size of user programs that can be assembled when 3KB to 5KB is available for the Symbol Table.

TABLE 7-15.
SYMBOL TABLE CAPACITY

Memory Available For The Symbol Table	Number of Average* Sized Symbols in a User Program Acceptable	Size of User Program **
3KB	359	1,795 statements
4KB	512	2,560 statements
5KB	640	3,200 statements
<p>*An average sized symbol consists of four or five characters.</p> <p>**Size of User Program is based on a user program that contains one symbol definition per five source statements.</p>		

Loading Procedures

It is generally assumed that, at the load time of the OS Assembler, an Operating System is resident in memory and in execution. Refer to the particular OS publications on the OS used, for details on loading and executing the OS.

When using the OS Resident Loader, the load command should be used to load the OS Assembler program. The BIAS location defined at load time is also the ORIGIN (ORG) or starting location for execution of the OS Assembler's Pass 1 operations. The Bias selected is also the hexadecimal value that must be added to the restart addresses necessary to restart the OS Assembler on various passes as described in Section 7.5.11. Selection of the Bias depends upon the location and size of the OS. In general, when the OS resides in low memory, the Bias selected should be the next memory location above the OS available for user programs, to make maximum use of memory for the Symbol Table.

Logical Device Selection

Prior to execution of the OS Assembler, the physical device addresses must be assigned with the OS in such a way that each logical device to be utilized during the assembly is related to the user's physical device configuration. Refer to Basic Operating System Program Manual, Publication Number B29-216, for the specific procedures involved to establish this physical to logical relationship. The logical unit numbers associated with the OS Assembler logical device assignment are represented in Table 7-16. The Input/Output logical requirements of each device listed are specified by the function it performs for OS Assembler data transfers.

TABLE 7-16.
LOGICAL UNIT NUMBER SPECIFICATIONS

Logical Unit Number	Logical Device Name	Logical Function Requirements
00	System Console	Logging operator messages; PASS 1, PASS 2, PASS 3, and the "XXNN" I/O Error message
01	Source Device	Read ASCII Source
02	Object Device	Write Binary Object Data
03	List Device	Write ASCII Listing
04	Scratch Device	Write/Read ASCII Source

Logical Source Format

The logical source format of a program acceptable to the OS Assembler is a sequential set of source statements, written in INTERDATA Assembler Language.

Upon execution, the OS Assembler reads the source by requesting the OS to read 80 byte ASCII records from the Source Device, Logical Unit Number 01. Each source statement can be any length up to 80 ASCII characters. When the physical record contains more than 80 characters, the OS transfers only the first 80 characters to the Assembler.

Refer to Table 7-17 for the general sequence or order of source statements within a program.

OS Assembler Operations

The INTERDATA OS Assembler provides a one-pass, two-pass, or three-pass assembly. It is preset to provide an assembly with the PASS2, PRINT, PUNCH, STOP, no SCRT, and no SQCHK options in effect. At assembly time, the programmer may choose the number of passes and other options desired for the assembly. The options are specified with the "OPT" control statement at the beginning of the source program. Refer to Table 7-18 and Table 7-19 for a summary of OS Assembler pass operations.

Of the three types, one-pass assemblies are fastest, but restrict the programmer's use of forward references within source statement expressions at programming time. Additionally, one-pass assemblies produce larger object program tapes requiring longer load time. Two-pass and three-pass assemblies give a more complete listing of the object code which is most useful at debugging time.

Both the assembly listing and object programs can be produced simultaneously on a single pass assembly, or on the second pass of a two-pass assembly. A three-pass assembly permits the singular production of the assembly listing on the second pass and of the object program on the third pass, such that, for users with devices that cannot accept printing and punching simultaneously, one assembly still produces both required listing and object program.

Pass 1 of a Single-Pass Assembly

Pass 1 is executed at the ORIGIN, the OS Assembler clears the Symbol Table, fetches from OS the maximum available core for building the Symbol Table, lists the message "PASS1" on the System Console, and begins to read source statements. When the option statement specifies the symbol "PASS1", the OS Assembler reads source from the Source Device (until the "END" source statement), builds the Symbol Table, and simultaneously outputs the assembly listing to the List Device and the object program to the Object Device, if both are requested. Listing may be suppressed by specifying the symbol "NOPRNT" in

the option statement. Object program production may be suppressed by specifying the symbol "NOPNCH" in the option statement. At the end of this single-pass assembly, the OS Assembler requests the OS to output an END OF JOB message on the System Console. When the symbol "SCRT" (SCRATCH) is specified in the option statement, a copy of the source statements is output to the Scratch Device and it is rewound while the Symbol Table is listed. (When using BOSS/4B, the Symbol Table is not listed until after the Scratch Device is completely rewound.)

TABLE 7-17.
LOGICAL SOURCE FORMAT

Type of Statement	Function	Ranking Order
OPT PAUSE TITLE IF ENTRY	Option Control Assembly Pause Listing Title/Control Conditional Assembly	First or not at all Interjected as desired Interjected as desired Interjected between modular segments
EXTRN	Global Symbol Definition (Internally Defined)	Prior to symbol's use
EQU	Global Symbol Definition (Externally Defined)	Prior to symbol's use
ORG	Local Symbol Definition	Prior to label's use and after operand's symbol definition
DO	Location Specification	Inserted to define load location where necessary. Optional if whole program sequentially relocatable from relocatable zero.
	Conditional/Multiple Instruction Assembly	Prior to conditional/multiple assembled instruction
(MACHINE INSTRUCTIONS) (*COMMENT STATEMENTS)	Program Logic Documentation Aid	} Intermixed
DC	Define Constants	
DS	Reserve Storage	
END	End of program signal	Required as last statement

TABLE 7-18.
OS ASSEMBLER OPERATIONS

Pass Number	Assembly Type		
	PASS 1	PASS 2	PASS 3
1	Read Source Write Listing Write Object	Read Source	Read Source
2		Read Source Write Listing Write Object	Read Source Write Listing
3			Read Source Write Object

TABLE 7-19.
OS ASSEMBLER OPERATIONS WITH SCRATCH

Pass Number	Assembly Type		
	PASS 1	PASS 2	PASS 3
1	Read Source Write Scratch Write Listing Write Object	Read Source Write Scratch	Read Source Write Scratch
	Rewind Scratch	Rewind Scratch	Rewind Scratch
2		Read Scratch Write Listing Write Object	Read Scratch Write Listing
		Rewind Scratch	Rewind Scratch
3			Read Scratch Write Object
			Rewind Scratch

Pass 1 of a Multiple-Pass Assembly

Pass 1 of a Multiple-Pass Assembly is executed at the ORIGIN, the OS Assembler clears the Symbol Table, fetches from OS the maximum available core for building the Symbol Table, assumes this is "PASS1" of a "PASS2" assembly with "PRINT", "PUNCH", "STOP", no "SCRT", no "SQCHK" options and lists the message "PASS1" on the System Console. For a multiple-pass assembly, either the "PASS2" or "PASS3" symbol must be specified in the option statement. The OS Assembler's first pass operations consist of reading the source from the Source Device (until the "END" statement) and building the assembled program's Symbol Table. The Symbol Table is listed on the List Device at the end of pass one, unless the symbol "NOPRNT" is specified in the option statement. When the symbol "SCRT", the SCRATCH option, is specified, the Assembler writes the source statements to the Scratch Device. At the end of the first-pass operations of a multiple-pass assembly, the OS Assembler rewinds the Scratch Device while listing the Symbol Table (under BOSS/4B the Symbol Table is not listed until after the Scratch Device is completely rewound), and prepares to read the source statements from the Scratch Device on subsequent passes. Before proceeding to the next pass, the OS Assembler lists the message "PASS2" on the System Console. The OS Assembler also pauses by listing the message "PAUSE" on the System Console when the option statement does not specify "GO". The operator must then resume execution of the OS Assembler to begin the second pass, e.g., under BOSS by typing CONTINUE.

Pass 2 of a Two-Pass Assembly

Pass 2 of a Two-Pass Assembly is executed automatically after pass one, under the "GO" option, or restarted at ORIGIN (ORG) plus X'0020', the OS Assembler lists the message "PASS2" on the System Console and begins second-pass operations. Reading the source program from either the Source Device or the Scratch Device (under the "SCRT" option), the Assembler simultaneously produces the assembly listing on the List Device and the object program on the Object Device when both are requested. Listing may be suppressed by specifying the symbol "NOPRNT" in the option statement. When the "SCRT" option is in effect, the Scratch Device is again rewound at the end of the pass, while the Symbol Table is listing on the List Device. (Under BOSS/4B, the Symbol Table is not listed until after the Scratch Device is completely rewound.) As this is the final pass of a two-pass assembly, the OS Assembler requests the OS to output an END OF JOB message on the System Console. When the OS has not altered memory, the Symbol Table is still intact so that PASS2 can be restarted at ORIGIN (ORG) plus X'0020' for duplicate assemblies without starting over again from ORIGIN for the first pass.

Pass 2 of a Three-Pass Assembly

Pass 2 of a Three-Pass Assembly is executed automatically after pass one, or restarted at ORIGIN plus X'0028', the OS Assembler lists the message "PASS2" on the System Console and begins two operations. Reading the source from either the Source Device or the Scratch Device, the OS Assembler produces the assembled program's assembly listing on the List Device, followed by a printout of the Symbol Table. No object program is produced on this pass. When the "SCRT" option is specified, the Scratch Device is again rewound during the listing of the Symbol Table. (Under BOSS/4B, the Symbol Table is not listed until after the Scratch Device is completely rewound.) Before proceeding to the final pass, the OS Assembler lists the message "PASS3" on the System Console. The OS Assembler also pauses when the option statement does not specify "GO".

Pass 3 of a Three-Pass Assembly

Pass 3 of a Three-Pass Assembly is executed automatically after a second pass when the option statement specifies "PASS3" assembly, or restarted at ORIGIN (ORG) plus X'0034', the OS Assembler lists the message "PASS3" on the System Console and begins pass three operations. Reading the source from either the Source Device or the Scratch Device, the OS Assembler produces the assembled object program on the Object Device. No assembly listing is produced on this pass. The Symbol Table is again listed on the List Device, unless the symbol "NOPRNT" is specified in the option statement. When the "SCRT" option is specified, the Scratch Device is rewound at the end of this pass. As this is the final pass of a three-pass assembly, the OS Assembler requests that an END OF JOB message be output to the System Console. Again, if the Symbol Table of the assembled program is still intact, then either pass two or pass three may be restarted for duplicating either assembly listings or object programs, respectively. The second pass may be restarted at ORIGIN plus X'0028' for listings or the third pass may be restarted at ORIGIN plus X'0034' for object programs.

Operating Procedures

In general, the OS Assembler is operated as any other user program running under the specific Operating System selected. The specific operations performed by the OS Assembler are mainly controlled by the OPTION statement in the programs source form, once the OS Assembler is executed at ORIGIN.

Option control may be also modified at OS Assembler execution time by assigning a logical unit to a null device. For example, under BOSS/4B, Program Number 03-021, the user input "ASSIGN 0200" assigns the logical Object Device to a null physical device, such that, object program production is suppressed even when the OS Assembler's PUNCH option is in effect. The user input "ASSIGN 0300" assigns the List Device, Logical Unit Number 03, to a null device, physical Device Address 00, such that, assembly listing is suppressed.

When the List Device is assigned to a null device, one-pass assemblies and the second pass of two-pass assemblies produce only the object program.

When the Object Device is assigned to a null device, one-pass assemblies and the second pass of two-pass assemblies produce only the assembly listing.

The basic steps necessary to accomplish an assembly in an OS environment are:

1. Load the Operating System.
2. Load the OS Assembler program as per the load procedures of the Operating System. The first load address or BIAS becomes the ORIGIN or starting execution address of the first-pass operations of the OS Assembler.
3. Using OS commands, assign the logical-to-physical device relationships necessary for the assembly. For each logical unit number used in the assembly, assign a physical device address.
4. Ready the devices for the OS Assembler's first-pass operations; that is, all devices are to be powered-up, on-line, supplied as needed, and ready to transfer their respective data (either send or receive).

5. Option control must be specified in the source program's first statement. Assumed options are PASS2, PRINT, PUNCH, STOP, no SCRT, no SQCHK.
6. Execute the OS Assembler at ORIGIN to start first-pass operations as per the OS procedures for user program execution. The OS Assembler prints the message "PASS1" and reads source until the END statement.
7. For single-pass assemblies, with the symbol "PASS1" in the option statement, the OS Assembler ends operations by issuing to the OS, an END OF JOB supervisor call after all I/O is completed. The user removes both the assembly listing and object program, if both were generated, after the END OF JOB message.

Continue to Step 8 for assemblies that have the symbols "PASS2" or "PASS3" in the option statement.

8. For multiple-pass assemblies, the OS Assembler prints the message "PASS2" on the System Console after the first pass operations are completed. This notifies the operator that the second pass is starting. When the message "PAUSE" is also printed, the operator then uses the OS procedures to continue execution of pass two. When source is to be read from the Source Device, the operator must re-ready source input in the Source Device once the message "PASS2" is printed.
9. Upon completion of second-pass operations, for a "PASS2" assembly, the OS Assembler ends operations by issuing the OS an END OF JOB supervisor call. The user removes both the assembly listing and object program, if both were generated, after the END OF JOB message. For a "PASS3" assembly, the user obtains the assembly listing.

Continue to Step 10 for assemblies that have the symbol "PASS3" in the option statement.

10. For three-pass assemblies, the OS Assembler prints the message "PASS3" on the System Console after second-pass operations are completed. This notifies the operator that the third pass is starting. When the message "PAUSE" is also printed, the operator then uses the OS procedures to continue execution of pass three. When source is to be read from the Source Device, the operator must re-ready the source input in the Source Device once the message "PASS3" is printed.
11. At the end of the third-pass operations, the OS Assembler again issues the OS and END OF JOB Supervisor call. The user removes the object program after the END OF JOB message is printed.

The following steps apply under Operating Systems that do not alter core memory after END OF JOB messages.

12. To restart the entire assembly process, for example, after correcting errors shown on the listing of a user program with listing errors, use the OS command to execute the OS Assembler at ORIGIN.
13. To restart the operations of the "PASS2" second pass, use the OS command to execute the OS Assembler at ORIGIN plus X'0020'. This provides a duplicate copy of both the OS Assembler listing and object program, if both "PRINT" and "PUNCH" options are in effect.
14. To restart the operations of the "PASS3" second pass, use the OS command to execute the OS Assembler at ORIGIN plus X'0023'. This provides a duplicate copy of the assembly listing.
15. To restart the operations of the "PASS3" third pass, use the OS command to execute the OS Assembler at ORIGIN plus X'0034'. This provides a duplicate copy of the object program.

Refer to Figure 7-16 for an example of Operator/System Console printouts during an assembly under the BOSS, Program Number 03-021.

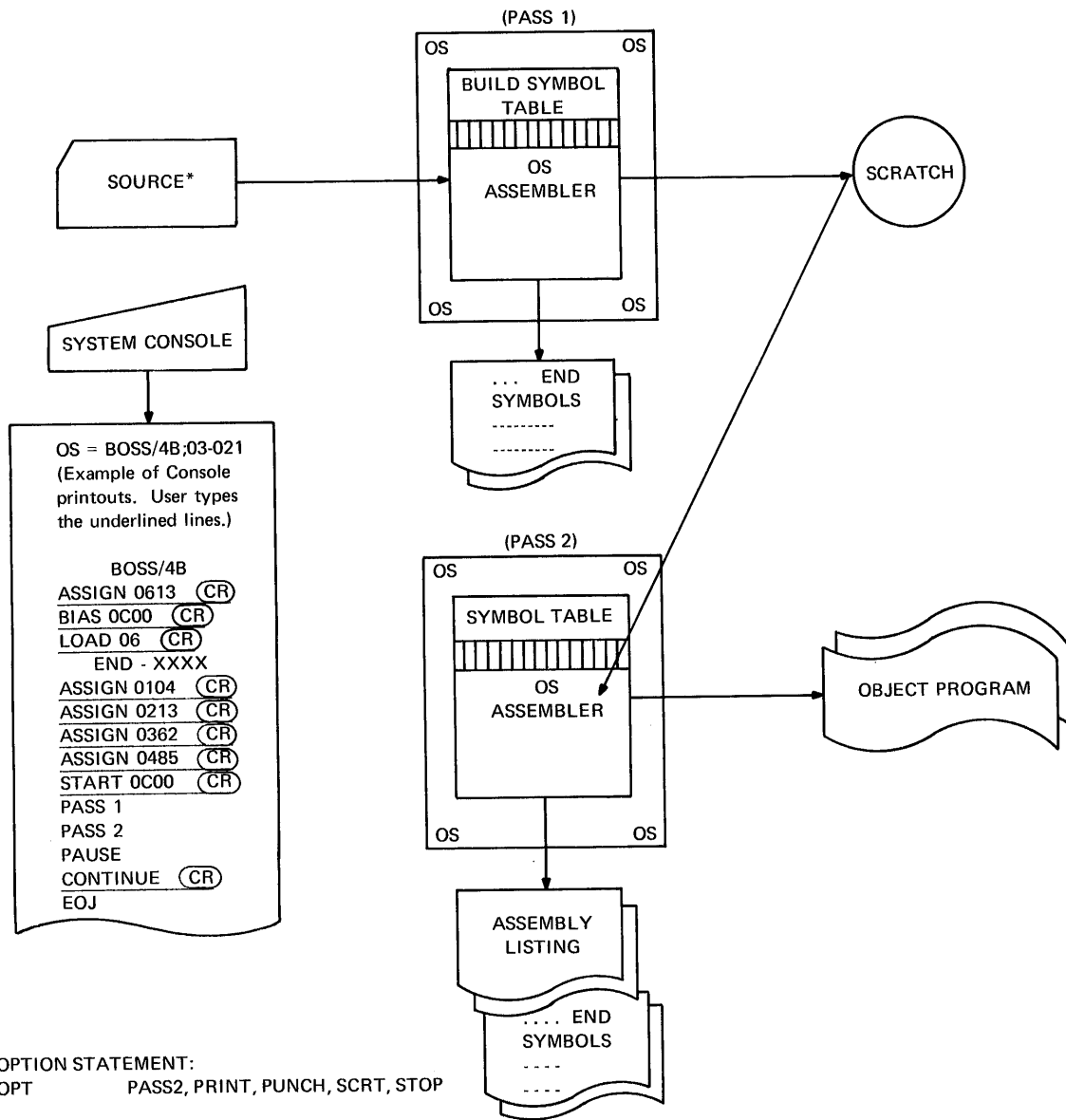


Figure 7-16. OS Assembler PASS2 Assembly With Scratch

The OS Assembler writes the "XXNN I/O ERROR" message to the System Console whenever an error condition exists during its data transfer requests to the Operation System. Refer to Figure 7-17 for a binary interpretation of the "XX" portion of the I/O Error message under the BOSS Operation System.

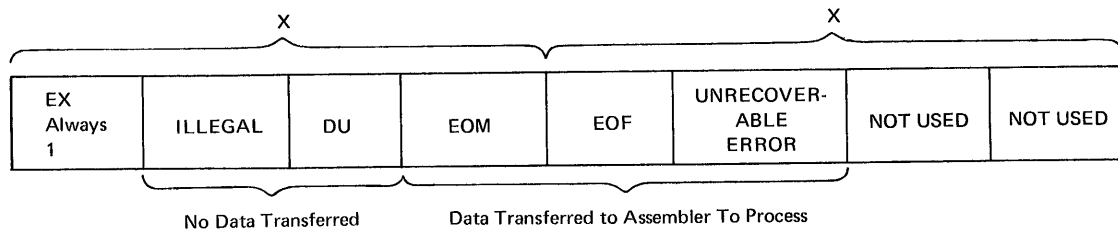


Figure 7-17. I/O Error Message Description

In the "XXNN I/O ERROR" message:

- XX = Two hexadecimal digits representing the eight-bit status byte depicting the type of error conditions that occur before or during a data transfer.
- NN = Two hexadecimal digits representing the eight-bit physical device address of the device from which or to which a data transfer could not take place correctly.

Refer to Table 7-20 for examples of OS Assembler error messages running under the INTERDATA BOSS Operating System.

TABLE 7-20.
EXAMPLES OF I/O ERROR MESSAGES

Error Messages	Meaning	Operator Intervention Required
C004 I/O ERROR	Illegal I/O requested of the physical device assigned to a logical unit.	Check device 04 (card reader) assignment to a logical write device. Reassign and to continue type CONTINUE.
C062 I/O ERROR	Illegal I/O requested	Check device 62 (line printer) assignment to a logical read device. Reassign and to continue type CONTINUE.
A013 I/O ERROR	Device Unavailable	Check device 13 (High Speed Paper Tape Punch) for depression of REMOTE button. Correct and to continue type CONTINUE.
A004 I/O ERROR	Device Unavailable	Check device 04 (card reader) for power-up, motor-on, START button depressed, when this error occurs at beginning of assembly. Check for MOTION, PICK FAIL, LIGHT/DARK errors. On PICK FAIL the card has not passed through the read station; correct the card when the front edge is mutilated and continue. When card causing other errors passes partially or totally through the read station into the card stacker, place it back into the hopper and continue to re-read it by typing CONTINUE.
9085 I/O ERROR	EOM = End of Medium	NONE. Assembly continues.
8885 I/O ERROR	EOF = End of File	... Assembly continues, with Format error. X'1313' (EOF) placed in buffer.
8495 I/O ERROR	UNREC = Unrecoverable Error	NONE. Mag tape #95 may have parity errors. Assembly continues with best guess data transfer. Operator must inspect listing to see if this type of error affects user assembly, such as, causing "M" Multiple Defined errors on source statements with labels when the parity error affected the main fields of the source statements. Aborting the Assembly at this point is at the operator's discretion.

Assembly Listing and Object Programs

The OS Assembler requests the OS to write the assembled program's assembly listing in seven-bit ASCII to the List Device, Logical Unit Number 03. Refer to Section 7.5.6 for a detailed description of the standard INTERDATA assembly listing format.

The OS Assembler requests the OS to write the assembled program's binary object data to the Object Device, Logical Unit Number 02. Refer to Section 7.5.6 for a detailed description of the standard INTERDATA binary object program format.

7.6 ASSEMBLY LEVEL PROGRAMMING TECHNIQUES

The INTERDATA Processors are characterized as being 3rd generation. The principal features of the machine which merit this description are the 16 General Registers, the extensive instruction set, the direct addressability of 64KB memory, and the byte addressing and manipulation features. All of these features contribute to convenient and flexible programming, particularly when contrasted to other machines with limited instruction sets and/or paged memory addressing schemes. The INTERDATA Processors, therefore, in contrast to other machines:

1. Require less programmer effort to generate a given program.
2. Facilitate generating larger and more sophisticated programs.
3. Allow more convenient updating or changing of a given program to meet new requirements.
4. Result in better system efficiency during program execution.

To reap these benefits of the 3rd generation architecture, the machine features must be used properly when programming the machine. The principal key to efficient use of the system lies in the proper allocations of the 16 General Registers. In general, these registers can be used in such a way as to optimize system performance, be it execution speed, core memory usage, or throughput. Some of the uses for General Registers are as follows:

1. Subroutine returns as with BAL instruction.
2. Constants - frequently used values such as 1, 2, or 0.
3. Variables - frequently used values such as pointers or indexes.
4. Device numbers - for use with I/O instructions.
5. Loop Counters - for controlling program loops.
6. General accumulators - for working values and temporary storage.

Register usage directly affects the system performance in the following ways:

1. RR-type instructions occupy two-bytes in memory and RX or RS-type instructions occupy four-bytes. The more quantities that reside in General Registers, the more RR-type-instructions are used, and the faster the program.
2. RR-type instructions in general execute faster than RX or RS type instructions. The more quantities that reside in General Registers, the more RR-type instructions are used, and the faster the program.
3. The more quantities are maintained in General Registers, the fewer load-and-store-type instructions are required for computations, and the more efficient the program in both core usage and execution time.

The actual allocation of registers varies from one program to another, depending on the characteristics of the problem and the performance objectives. By saving and restoring registers from memory at strategic points, the use of the General Registers can vary within a program. A typical register allocation for general programming is as follows:

subroutine returns	2
constants	3
variables	3
device numbers	2
loop counters	3
general accumulators	<u>3</u>
Total =	16 registers

An alternate approach to the use of General Registers that is appropriate to interrupt programming is to devote certain registers for use only by interrupt routines. In this case, no register saving and restoring is required to service an interrupt, which minimizes response time and overhead time for interrupt routines.

Consistent with General Register usage, most assembly language programs assign symbolic names to the General Registers it uses; that is, if Register 3 is used as a counter, rather than writing

```
AHI 3,5
```

a mnemonic symbol such as COUNT is assigned the register number, and the instruction is written

```
AHI COUNT,5
```

In this way, assembly language listings become quite easy to read and are a valuable element of the program documentation.

In addition to General Register usage, there are many specific techniques which can be used to improve program efficiency. Some of these techniques are discussed below.

1. To clear General Register R, use the instruction

```
XHR R,R,  
or  
SHR R,R.
```

2. To test the sign of a value in General Register R, use the instruction

```
LHR R,R
```

which adjusts the Condition Code, but does not affect the contents of the register.

3. To shift General Register R left one bit, use the instruction

```
AHR R,R
```

4. To exchange the contents of General Registers A and B without using a third register, use the sequence

```
XHR A,B  
XHR B,A  
XHR A,B
```

5. To control a loop with the minimum register and memory usage, use one Register C as follows:

```
UP      XHR  C,C      DOWN  LHI  C,START  
LOOP   .           LOOP   .  
      .           .  
      .           .  
      AIS  C,1      SHI  C,1  
      BNZ  LOOP    BP   LOOP
```

6. To control a loop with the fastest possible execution time, use a BXLE loop as follows:

```
      LHI  A,LOWER  
      LHI  B,INCR  
      LHI  C,FINAL  
LOOP  .  
      .  
      .  
      BXLE A,LOOP
```

7. To add both a constant Y, and the contents of Register Z to the contents of Register X, use the indexed RS instruction

```
AHI X,Y (Z).
```

8. In a Model 3, which has no Load Multiple (LM) or Store Multiple (STM) instructions, use the following sequence for subroutines that require full use of the 16 General Registers, where all subroutine calls have the form BAL 15, SUBR.

```

SUBR   STH   0, SAVREG+2
      BAL   0, SAVER
      .
      .
      B     SAVREG

```

where

```

SAVER  STH   1, SAVR1+2
      STH   2, SAVR2+2
      .
      .
      STH  15, SAVR15+2
      BR    0

```

and

```

SAVREG LHI   0, 0000
SAVR1  LHI   1, 0000
      .
      .
SAVR15 LHI  15, 0000
      BR    15

```

9. Where a direct correspondence exists between two sets of discrete values, the fastest method of conversion from one to the other involves a programming technique called table look-up. One application of table look-up procedures is that of code conversion.

```

*CHAR CONTAINS SOME 8-BIT VALUE
      LB    CHAR, ASCII (CHAR)
*CHAR CONTAINS ASCII CODE (CORRESPONDING 8-BIT)
ASCII  DC   X'FIRST TWO ASCII CODES'
      .
      .
      DC   X'LAST TWO ASCII CODES'

```

10. To save memory when passing arguments to subroutines, instead of loading General Registers with the argument values, the called subroutine can load its required arguments into its General Registers by using the linkage register to point to arguments in the calling sequence as follows:

```

      BAL   LINK, SUBR
      DC    A(ARG1)
      DC    A(ARG2)
      DC    A(ARG3)
      DC    A(ERROR)
      CONTINUE
      .
      .
SUBR   LH    R1, 0(LINK)    FETCH 1ST ARGUMENT
      LH    R2, 2(LINK)    FETCH 2ND ARGUMENT
      LH    R3, 4(LINK)    FETCH 3RD ARGUMENT
      .
      .
      B     6(LINK)        ERROR RETURN
      .
      .
      B     8(LINK)        RETURN TO CONTINUE

```

11. To count the number of 1's (bits set) in Register P, and place the result in Register Q, use the sequence below, which is fast although somewhat tricky.

```

        XHR   Q,Q   CLEAR Q
        LHR   P,P   TEST P
        BZ    EXIT
LOOP    AHI   Q,1   INCR Q
        NHI   P,-1(P) CLEAR LEAST SIGNIFICANT BIT SET
        BNZ   LOOP  AND TEST
EXIT    .
        .
        .

```

Following the above sequence for some sample value, such as P = X'0124', will help the reader understand the operation of this sequence.

7.7 FORTRAN IV

7.7.1 General Description

The INTERDATA FORTRAN IV system is, with three minor exceptions, USASI Standard Fortran (X3.9-1966). The FORTRAN IV system consists of the following:

FORTRAN IV Compiler	03-023
OS Library Loader	03-030
Run Time Library	07-040

This section provides a brief summary of the INTERDATA FORTRAN IV system. The FORTRAN language is described in the FORTRAN IV Reference Manual, Publication Number 29-220. The use of the FORTRAN IV Compiler is described in the BOSS FORTRAN User's Guide, Publication Number 29-246. The OS Library Loader is discussed in Section 7.9. The Run Time Library is described in the Run Time Library Description, Publication Number 29-242.

The INTERDATA FORTRAN IV system operates in an INTERDATA Processor equipped with the 16KB or more of memory. In a 16KB system, BOSS is required.

The FORTRAN Compiler is supplied as a relocatable paper tape, 03-023M16. The Run Time Library is supplied as five relocatable paper tapes as follows:

Real/Integer Routines	07-040F01M16
Double Routines	07-040F02M16
Complex Routines	07-040F03M16
Input/Output Routines	07-040F04M16
Miscellaneous Routines	07-040F05M16

The OS Library Loader, 03-030M16, is used to load compiled programs and edit Run Time Library Tapes as needed.

7.7.2 FORTRAN Language Specifications

The INTERDATA FORTRAN IV is a USASI (X3.9-1966) Standard FORTRAN with the following exceptions:

- 1 A FORTRAN statement may consist of 160 characters excluding the statement number, if any, and all non-essential blanks. This number allows, at a minimum, an initial statement and one continuation statement. Since blanks are not included in the 160 character maximum, additional continuation statements may be possible if Columns 7-72 of the initial statement and the first continuation statement contain any blanks.

- All Intrinsic and External Complex, and Double Precision functions must be explicitly declared in the FORTRAN program. Refer to Tables 7-21 and 7-22 for a summary of Intrinsic and External functions.

EXAMPLE: COMPLEX C, CEXP
 $C = \text{CEXP}(C)$

- The auxiliary Input/Output statements REWIND, BACKSPACE, and ENDFILE have been eliminated. These operations may be performed via CALL statements to Run Time Library subroutines. See Run Time Library Description, Publication Number 29-242.

7.7.3 Loading the FORTRAN Compiler

The FORTRAN Compiler requires an operating system for purposes of input/output. The FORTRAN tape can be loaded using the LOAD command in conjunction with the ASSIGN and BIAS commands. For example, an operating system command sequence to load the compiler might be as follows:

```
ASSIGN    0213
BIAS      0C00
LOAD      2
```

This sequence assigns Logical Unit 2 to the High Speed Paper Tape Reader, sets the bias to X'0C00' and loads a tape from Logical Unit 2.

TABLE 7-21.
 INTRINSIC FUNCTIONS

Symbolic Name	Number of Arguments	Intrinsic Functions	Type of:	
			Argument	Function
ABS IABS DABS	1	Absolute Value (a)	Real Integer Double	Real Integer Double
AINT INT IDINT	1	Truncation (Sign of a times largest integer < a)	Real Real Double	Real Integer Integer
AMOD MOD	2	Remaindering* (a ₁ (mod a ₂))	Real Integer	Real Integer
AMAX0 AMAX1 MAX0 MAX1 DMAX1	≥ 2	Choosing Largest Value (Max (a ₁ , a ₂ ,))	Integer Real Integer Real Double	Real Real Integer Integer Double
AMIN0 AMIN1 MIN0 MIN1 DMIN1	≥ 2	Choosing Smallest Value (Min (a ₁ , a ₂ ,))	Integer Real Integer Real Double	Real Real Integer Integer Double
FLOAT	1	Float (Conversion from integer to real)	Integer	Real
IFIX	1	Fix (Conversion from real to integer)	Real	Integer

TABLE 7-21.
INTRINSIC FUNCTIONS (Continued)

Symbolic Name	Number of Arguments	Intrinsic Function	Type of:	
			Argument	Function
SIGN ISIGN DSIGN	2	Transfer of Sign (Sign of a_2 times $ a_1 $)	Real Integer Double	Real Integer Double
DIM IDIM	2	Positive Difference ($a_1 - \text{Min}(a_1, a_2)$)	Real Integer	Real Integer
SNGL	1	Obtain most Significant Part of Double Precision Argument	Double	Real
REAL	1	Obtain Real Part of Complex Argument	Complex	Real
AIMAG	1	Obtain Imaginary Part of Complex Argument	Complex	Real
DBLE	1	Express Single Precision Argument in Double Precision Form	Real	Double
COMPLX	2	Express Two Real Arguments in Complex Form ($a_1 + a_2\sqrt{-1}$)	Real	Complex
CONJG	1	Obtain Conjugate of a Complex Argument	Complex	Complex

*The function MOD or AMOD (a_1, a_2) is defined as $a_1 - |a_1/a_2| \cdot a_2$, where $|x|$ is the integer whose magnitude does not exceed the magnitude of x and whose sign is the same as x .

TABLE 7-22.
BASIC EXTERNAL FUNCTIONS

Symbolic Name	Number of Arguments	Basic External Function	Type of:	
			Argument	Function
EXP DEXP CEXP	1 1 1	Exponential (e^a)	Real Double Complex	Real Double Complex
ALOG DLOG CLOG	1 1 1	Natural Logarithm ($\log_e(a)$)	Real Double Complex	Real Double Complex
ALOG10 DLOG10	1 1	Common Logarithm	Real Double	Real Double
SIN DSIN CSIN	1 1 1	Trigonometric Sine ($\sin(a)$)	Real Double Complex	Real Double Complex
COS DCOS CCOS	1 1 1	Trigonometric Cosine ($\cos(a)$)	Real Double Complex	Real Double Complex

TABLE 7-22.
BASIC EXTERNAL FUNCTIONS (Continued)

Symbolic Name	Number of Arguments	Basic External Functions	Type of:	
			Argument	Function
TANH	1	Hyperbolic Tangent ($\tanh(a)$)	Real	Real
SQRT	1	Square Root ($(a)^{1/2}$)	Real	Real
DSQRT	1		Double	Double
CSQRT	1		Complex	Complex
ATAN	1	Arctangent ($\arctan(a)$) ($\arctan(a_1/a_2)$)	Real	Real
DATAN	1		Double	Double
ATAN2	2		Real	Real
DATAN2	2		Double	Double
DMOD	2	Remaindering * ($a_1 \pmod{a_2}$)	Double	Double
CABS	1	Modulus	Complex	Real

*The function DMOD (a_1, a_2) is defined as $a_1 - |a_1/a_2| a_2$, where $|x|$ is the integer whose magnitude does not exceed the magnitude of x and whose sign is the same as the sign of x .

7.7.4 Memory Requirements

Refer to Figure 7-18 for a memory map at compile time in a 16KB memory.

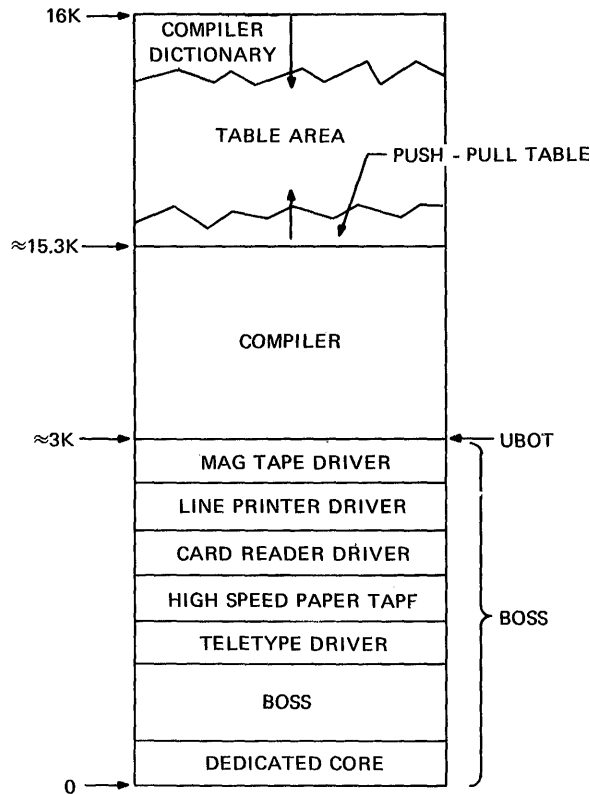


Figure 7-18. Memory Map At Compile Time Using BOSS

In the 16K byte system, as shown in Figure 7-18, the Basic Operating System occupies about 3K bytes. The FORTRAN Compiler occupies about 12.3K bytes. This leaves approximately 0.7K bytes for the FORTRAN Dictionary and Push-Pull Table. The dictionary is the Compiler's Symbol Table. The dictionary includes entries for:

- Statement Labels
- Numeric Constants
- Alphabetic Constants
- Variable Names
- Common Block Names
- Subprogram Names
- Temporary Storage Variables

Entries in the dictionary are variable length, so that Labels and variable names of less than six characters require less table space than a six character Label or variable name.

The Push-Pull Table is a variable length area which is used as a pushdown stack during the decoding of arithmetic expressions. At a maximum, this table is about 150 bytes.

For approximately 0.7K bytes of memory available for Table Area, a FORTRAN program might contain:

15 labels*	= 150 bytes
5 variables (via specification statements)**	= 100 bytes
15 variables (not in specification statements)*	= 150 bytes
6 temporaries	= 48 bytes
10 constants	= 100 bytes
Dictionary area	= 548 bytes
Push Pull Table	= 150 bytes
total table area	= 698 bytes

The FORTRAN compiler is written so that, for systems with more than 16K bytes of memory, the Table Area automatically increases to the top of available memory.

16K system: Table Area	= 0.7K
24K system: Table Area	= 8.7K
etc.	

If during a compilation, the available Table Area (Dictionary + Push Pull) is insufficient for further entries, the FORTRAN Compiler prints the following error message and terminates the job.

*LINE # XXXX ERROR #50
where XXXX is the last line processed.

*assumed to be three or four character names.

**assumed to be two dimensional, with possible use in Common-Equivalence statements.

7.7.5 Compiler Execution Procedures

The FORTRAN Compiler uses four logical units as follows:

<u>LOGICAL UNIT</u>	<u>PURPOSE</u>
1	Source Input
2	Binary Output
3	List Output
7	Error Message Output

These logical unit numbers must be assigned to physical devices via the ASSIGN command in the Operating System prior to starting the Compiler. Normally, the Error Message Device is the same as the List Device, which causes assigned error messages to be listed following the statements which caused the error. However, if a listing is not desired, Logical Unit 3 can be assigned to 0, the null device, which suppresses listing output. Similarly, Logical Unit 2 can be assigned to 0 to suppress binary output. In this fashion, a compilation can be performed generating only error messages.

A typical Operating System command sequence for device assignments might be:

ASSIGN 0104	Source Input Device
ASSIGN 0213	Binary Output Device
ASSIGN 0302	List Device
ASSIGN 0702	Error Message Device

This sequence assigns a Card Reader (X'04') as the Source Input Device, a High Speed Paper Tape Punch (X'13') as the Binary Output Device, and a Teletype (X'02') as the list and Error Message Device.

The starting location for the FORTRAN Compiler is the BIAS or first location in the program. The BIAS value is assigned prior to loading. If no programs are loaded after the Compiler, execution can be started with the Operating System command START, since BOSS retains a transfer address to the correct location. If programs are loaded after the Compiler, the command START bbbb can be used, where bbbb specifies the first location of the Compiler.

Once started, the Compiler reads source statements and produces a symbolic listing and binary object tape in a one-pass operation. The compilation proceeds until an END statement is detected, at which time a Memory Map is printed. When the map is complete, the Compiler returns control to BOSS, and BOSS types EOJ on the Console Device.

7.7.6 Loading a Compiled Program

Compiled programs use an operating system for purposes of input/output and interrupt control.

Binary object tapes generated by the Compiler are relocatable tapes in standard loader format (i. e. M08 or M16 tape formats depending on whether they are zoned or not zoned). Due to the program linkage information on the object tape, it is necessary to use the OS Library Loader, 03-030, to load these tapes.

Refer to Figure 7-19 for an Execution Time Memory Map using BOSS.

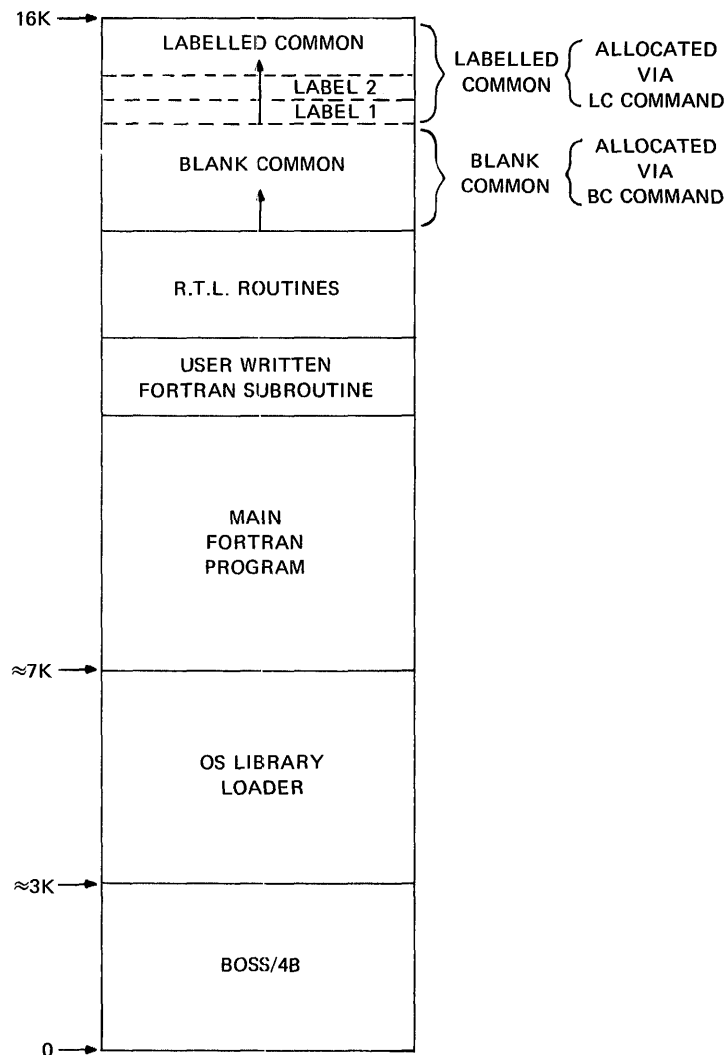


Figure 7-19. Execution Time Memory Map

7.8 INTERACTIVE FORTRAN

7.8.1 General Description

This section contains a brief synopsis of the Interactive FORTRAN System which operates on the INTERDATA family of Processors. Table 7-23 provides a summary of the Interactive FORTRAN System. Several versions of the Interactive FORTRAN, including an OS Interactive FORTRAN, Program Number 03-033, are available. Refer to User's Manual for Interactive FORTRAN, Publication Number 29-014, for further information.

The Interactive FORTRAN System provides a Direct Mode for on-line evaluation of arithmetic expressions, and an Editing Mode for the creation and manipulation of stored programs. The system combines the convenience of a desk calculator with the programming power of FORTRAN.

In many FORTRAN systems designed for small computers, the programming is simple, but the mechanics of program preparation are complex. This is especially true of paper tape oriented machines for which it is necessary to go through the phases of compiling, object tape loading, and system subroutine loading. The handling of the paper tapes becomes laborious, when the whole procedure needs repeating for every program correction. In contrast, the INTERDATA system requires no program preparation other than the entry of the source information; corrections can be made while the programs remain in memory.

7.8.2 Features

The system provides a set of FORTRAN operations and a set of system commands for editing, debugging, and control. The FORTRAN set is chosen to provide the greatest computational power for the least memory space required for its implementation. The system is designed expressly for on-line use. The user communicates with the system through a Teletype. Features which assist the interaction between man and machine are:

1. Single character indications which request input and reflect the mode of the system are as follows:
 - ← direct-mode input request
 - * edit-mode input request
 - = data input request
2. All inputs are terminated by a carriage return. Until the terminating carriage return is received, no processing takes place, and the input line can be corrected or changed at will.
3. Command directives and FORTRAN operations can be abbreviated during input to minimize typing.
4. Commands are provided for listing all defined variables and for listing the names of all defined programs.
5. Commands for program creation, editing, and execution can be freely intermixed.
6. Error messages indicate the point in a program at which an error occurred during execution.
7. A performance improvement feature called FREEZE is provided. This operation alters the stored programs so that they become "more compiled", and results in a substantial decrease in execution time.

The system is designed to operate in systems with 8K bytes or more of memory, sufficient working space is available for the user to create and execute a FORTRAN program with 30-70 statements and 10-50 variables. Any available memory above 8K is used to expand the user's working space for more programs and data.

7.8.3 Use of the System

This interactive FORTRAN is operated and controlled from the Teletype keyboard. The system presents two distinct modes to the user. The Direct Mode, which is characterized by the arrow character (←) in the left margin, permits on-line assignment of variables, evaluation of expressions, etc. The Edit Mode, which is characterized by an asterisk (*) in the left margin, allows the creation and modification of stored programs. When the system is started, the user is given control at the keyboard in Direct Mode.

The user converses with the system in statements, either FORTRAN operation statements or system commands. Until the RETURN key is depressed, which terminates a statement, no processing takes place, and the input line can be corrected or changed at will. The left arrow (←) key can be used anytime to erase the last character in the line. The hashmark (#) key can be used anytime to erase the current line. When # is depressed, the system advances one line so that the current statement can be retyped. This chapter discusses the details of program editing, system command, error messages, etc.

There are four types of operations: control, declarations, assignments, and input/output transfers. Table 7-24 summarizes the operation names in each class.

TABLE 7-24.
FORTRAN OPERATION STATEMENTS

Type	FORTRAN Name	Purpose
Control	CALL RETURN GO TO IF	Execute a Program Exit from a Program Transfer to a Statement Compare an Expression to Zero

TABLE 7-24.
FORTRAN OPERATION STATEMENTS (Continued)

Type	FORTRAN Name	Purpose
Declaration Assignment	DO	Define a Set of Statements to Execute Repeatedly in a Loop
	CONTINUE DIMENSION Name=Value	Define End of a "DO Loop" Define Name and Size of Arrays Assign a Value to Named Variable or Array Element
Input-Output	TYPE	Print Values or Character Strings on the Teletype Printer
User Programmable	ACCEPT	Input Numbers from Teletype Keyboard and Assign to Variables or Array Elements
	WRITE	Transfer to assembly language output routine
	READ FUNCTION	Transfer to assembly language input routine Transfer to assembly language function routine

The commands used for program editing are:

SUBROUTINE	Define a new subroutine or refer to existing subroutine.
OPEN	Refer to a specific statement of referenced subroutine.
LIST	List either entire subroutine or one statement of a subroutine.
DELETE	Delete either entire subroutine or one statement of a subroutine.
END	Terminate editing sequence.

These commands, except for SUBROUTINE, can be abbreviated to the first two characters to minimize typing. The SUBROUTINE command can be abbreviated to the first four characters (SUBR).

7.8.4 System Capacity

Stand-alone systems operate on an INTERDATA Processor with 8K bytes or more of memory. The FORTRAN program itself occupies approximately 6.5K bytes of memory. In an 8K byte memory, this leaves a 1.5K byte working space for user's stored programs and data. Any available memory above 8K bytes can be used to expand the working space. The OS system requires 16KB memory.

Working space in memory is used as follows:

1. Stored statements require 20 bytes per average statement.
2. Defined variables require six bytes each.
3. Defined arrays require $6+4N$ bytes where N is the number of elements in the array.

Each 1000 bytes of working space can hold 50 average statements, over 150 variables, or a 15 X 15 two dimensional array.

7.9 LOADER DESCRIPTIONS

7.9.1 General

Table 7-25 provides a summary of Loader Features. INTERDATA programs are normally supplied as binary object paper tapes in one of six formats. These formats, and the corresponding part number suffix, are as follows:

M08 - Relocatable zoned loader format

M09 - Absolute zoned loader format

M10 - Bootstrap (self-loading)

M14 - Eight-bit core image

M16 - Relocatable non-zoned loader format

M17 - Absolute non-zoned loader format

TABLE 7-25.
SUMMARY OF LOADER FEATURES

<p>REL Loader 06-024</p>	<p>Absolute or relocatable programs Post-load transfer External references (EXTRN's) illegal Common usage illegal Forward-reference-chain definitions legal External definitions (ENTRY) ignored Program labels ignored Zoned or non-zoned tapes</p>
<p>General Loader 06-025</p>	<p>Absolute or relocatable programs Post-load transfer Forward-reference-chain definitions legal External references (EXTRN's) legal External definitions (ENTRY's) legal Programs labels typed-out Common usage illegal Zoned or non-zoned tapes No global symbols</p>

Resident Loader compatible with REL Loader, 06-024. OS Library Loader (03-030) compatible with General Loader, 06-025, plus facility for COMMON usage, plus "OUT" mode.

The bootstrap (M10) tapes are loaded by using the 50 Sequence, as described in Section 7.9.2. Bootstrap tapes are constructed from a combination of eight-bit, fast format, and loader format sections that are appropriate for the program being loaded.

The eight-bit (M14) tapes are loaded by using the eight-bit loader in the 50 Sequence that is adjusted to read the correct amount of data. Normally, the 50 Sequence is modified for this purpose by changing the upper limit from X'CF' to the appropriate value. This tape format is used only for certain test tapes, such as the processor test and memory test, which cannot assume the presence of another loader program in memory.

There are five loaders available for loading binary object tapes as generated by the Assembler, the Compiler, or Hex Debug Program. These loaders are:

<u>Loader</u>	<u>Program Part No.</u>
1. OS Library Loader	03-030
2. The General Loader	06-025
3. The Relocating (REL) Loader	06-024
4. The OS Resident Loader	Part of the Operating System

These loaders are compatible but vary in size and in the number of features provided. The most comprehensive is the OS Library Loader which occupies about 4000 bytes of memory, and runs under the Basic Operating System. It may be operated interactively, through the Console Teletype, or it may read its operator commands from some other device, such as a card reader, allowing latch load-and-go processing. All I/O is accomplished through logical I/O calls to the Operating System. Programs being loaded may be absolute or relocatable, they may contain entry points and external symbol references, forward references with programs, and common block definitions. A program library may be created on a magnetic tape device, and the programs may be called by name, linked to other library or non-library programs, and executed by operator commands. Additionally, programs may be output as an absolute load module suitable for loading by any other loader. Load modules may consist of any number of programs, but all external symbolic references will be converted to internal forward reference chains within the module. In this way, the Loader can be used to link a group of programs which may be loaded by the Resident Loader, overlaying the OS Library Loader. The Loader can print a Memory Map showing programs loaded or programs in a load module.

The General Loader is a stand-alone program, occupying about 1500 bytes. It inputs paper tape from a Teletype or from a highspeed reader, and logs messages on the Teletype. It provides program relocating, ENTRY and EXTRN handling, and allows forward references within programs. This program is operated from the Processor Control Panel.

The Relocating Loader is a stand-alone program occupying about 800 bytes. It reads paper tapes from a Teletype or HSPTR, and displays error indication on the Processor Control Panel. It allows program relocation, and forward references within programs. This program is operated from the Processor Control Panel.

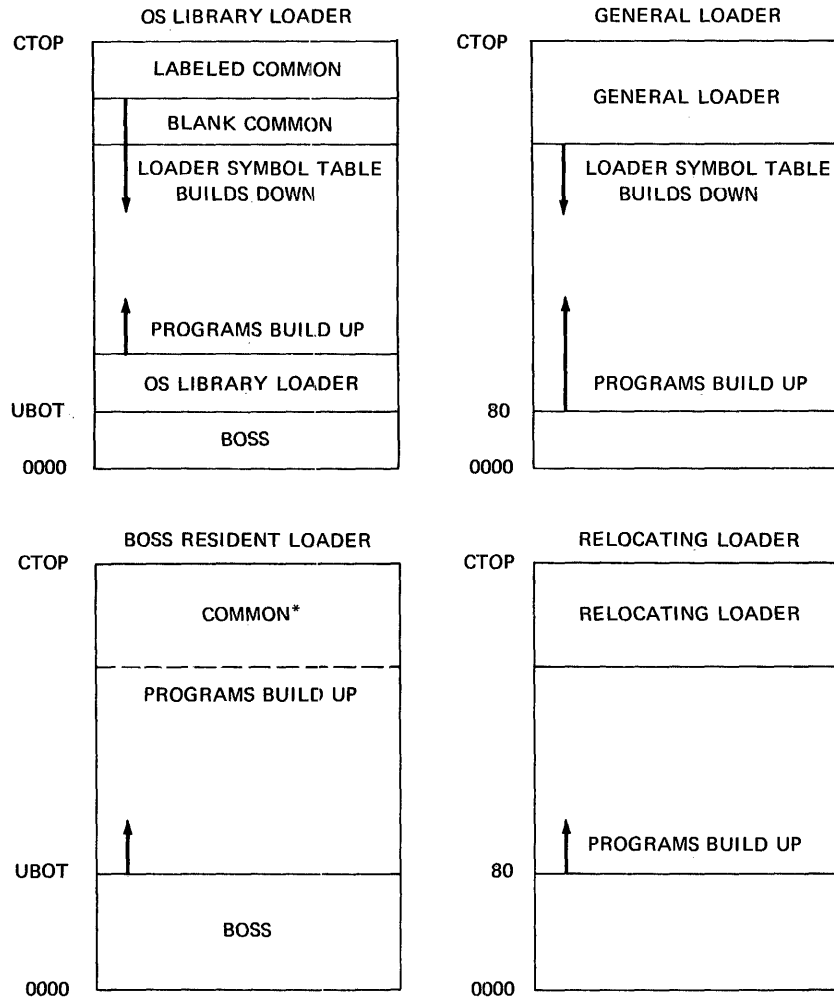
The Resident Loader provides the same features as the Relocating Loader, except that it is operated by operator commands through the system Teletype, and makes use of the Operating System logical I/O, allowing input from any binary device.

Any program to be executed under the Operating System must be loaded by the Resident or the OS Library Loader. If the program defines Common or contains any external references, it must be loaded by the Library Loader, or the Library Loader may be used to produce a load module which may then be loaded by the Resident Loader. Programs which do not require an Operating System may be loaded by one of the stand-alone Loaders. If the program contains references to external symbols, the General Loader must be used. Otherwise, the Relocating Loader may be used.

The binary tapes generated by the Assembler can be absolute or relocatable. A pointer called BIAS identifies the origin used in loading a Relocatable program. Absolute data is stored at the absolute location specified for the data. Relocatable data is stored at its relative address, plus the value of the BIAS pointer.

The BIAS may be set to any address before a program is loaded, causing the program to be loaded upward in memory from that address. Upon completion of any load, the Loader adjusts the BIAS point to the next available location, above the highest location loaded. This occurs whether the loaded program was relocatable or absolute. Thus additional Relocatable programs are loaded in consecutive areas of memory.

During loading, the Loaders block attempts to load any data that would overwrite the Loader, either absolute data at an address within the Loader, or relocatable data at such an address because of the current value of BIAS. In addition, the General Loader will not load above itself, the OS Library Loader will not load below itself, and the Resident Loader will not overwrite any part of the Operating System. The Relocating Loader loads above or below itself in memory. See Memory Maps, Figure 7-20.



CTOP = TOP OF CORE (LAST ADDRESS WITHIN MEMORY).
 *COMMON USED ONLY IF LOADING A LOAD MODULE PRODUCED BY OS LIBRARY LOADER.

Figure 7-20. Loader Memory Maps

7.9.2 50 Sequence Bootstrap Loader

The 50 Sequence for the INTERDATA Processor family is shown on Table 7-26. Note that the eight-bit loader is the same for all memory sizes. The Device Table, from X'78' to X'7F' is changed according to the device configuration at hand. The sequence shown in Table 7-27 is appropriate for Teletype input/output. With this sequence, location X'22' should contain the value X'0058'. When loading bootstrap (M10) tapes, refer to the program document to see which sequence is appropriate.

TABLE 7-26.
SEQUENCE FOR MODELS 74, 70, and 80

50	D500	LOAD	AL	X'CF'
52	00CF			
54	4300		B	X'80'
56	0080			
58				
5A				
5C				
5E				
60	Reserved for Register Save Area			
62				
64				
66				
68				
6A				
6C				
6E				
70				
72				
74				
76				
78	0294	BINDV	DC	X'0294'
7A	0298	BOUTDV	DC	X'0298'
7C	0294	SINDV	DC	X'0294'
7E	0298	LISTDV	DC	X'0298'

The eight-bit loader stores eight-bit data bytes into memory from X'80' to X'CF' and transfers to X'80'. This loader uses the Binary Input Device as defined in X'78'. This loader can read from Teletype, a High Speed Paper Tape Reader, Cassette Tape, Nine-Track Magnetic Tape, or from any device which can transfer 80 eight-bit data bytes, and can be started with a single eight-bit output command. When using the eight-bit loader at X'50', three steps are required:

1. Place the tape in the reader device to be used. Note that with the Auto Load instruction on Models 74, 70, and 80, leading zero data characters are ignored.
2. Prior to starting the loader at X'50', the INITIALIZE switch on the Control Panel should be depressed.
3. When loading from a Teletype, the tape motion must be started manually. After the loader is started at X'50', with an ASR-33, toggle the reader switch to START. With an ASR-35, put the reader switch in RUN with the Teletype MODE Switch in the KT position.

The Device Definition Table contains four halfwords. Each halfword specifies one device.

0	7 8	15
DEVICE NO.	OUTPUT COMMAND	

The left byte contains the device number. The right byte contains the output command required to start that device. The four halfwords are used as follows:

X'78'	BINDV	Binary Input	Used by stand-alone loaders to select the load device.
X'7A'	BOUTDV	Binary Output	Used by Basic Assembler, Hex Debug, and Text Editor to select the punch device.
X'7C'	SINDV	Source Input	Used by Basic Assembler, and Text Editor to select the source input device.
X'7E'	LISTDV	Symbolic Output	Used by Basic Assembler, Hex Debug, and Text Editor to select the list device.

During loading operations, only BINDV at X'78' is used. During Basic Assembler operations, the other three halfwords are used.

Table 7-27 shows the Device Definition Table entry for various devices.

TABLE 7-27.
DEVICE DEFINITION TABLE ENTRIES

Program	Function	Table Location	Device	Table Entry
Loaders	Loading	BINDV at X'78'	Teletype	0294
			High Speed Tape Reader	1399
Basic Assembler Hex Debug and Text Editor	Punching Object	BOUTDV at X'7A'	Teletype	0298
			High Speed Tape Punch	139A
Basic Assembler and Text Editor	Reading Source	SINDV at X'7C'	Teletype	0294
			High Speed Tape Reader	1399
Basic Assembler Hex Debug and Text Editor	Listing	LISTDV at X'7E'	Teletype	0298
			High Speed Tape Punch	139A
			Line Printer	6280

The General and Relocating Loaders are provided in a relocating bootstrap form. The format of the tapes is illustrated in Figure 7-21. The tapes consist of three segments: the boot portion in eight-bit format, a fast formal copy of the REL Boot Loader, and the actual loader in standard M16 binary object tape format. When the tape is loaded using the eight-bit loader at X'50', the following sequence of events takes place.

1. The eight-bit Loader at X'50' reads another Loader into X'80' to X'CF' and transfers to X'80'.
2. The program at X'80' reads the balance of the eight-bit data into X'DF0' to X'FF6', which includes a Fast Format Loader.

3. An arithmetic checksum on the information from X'DF0' to X'FF6' is then tested. If the checksum is correct, the process continues. If the checksum is not correct, the tape is stopped and the program halts.
4. The Fast Format Loader then loads a special REL Boot Loader from X'80' to X'2AA'.
5. The top of memory is then determined with a search technique, and the REL Boot Loader's Bias is set a fixed distance from the top of memory. The REL Loader is placed X'400' from the top of memory. The General Loader is placed X'600' from the top.
6. The REL Boot Loader then reads the Loader program, which is in relocatable form, and relocates it into the top portion of memory.
7. The REL Boot Loader computes checksums on each record, and halts whenever a checksum error is detected. In this case, reposition the tape to the previous record gap and depress EXECUTE to reread the record.
8. When the entire tape has been loaded, the Processor halts with the WAIT light illuminated. Depress EXECUTE to transfer control to the loader just loaded.

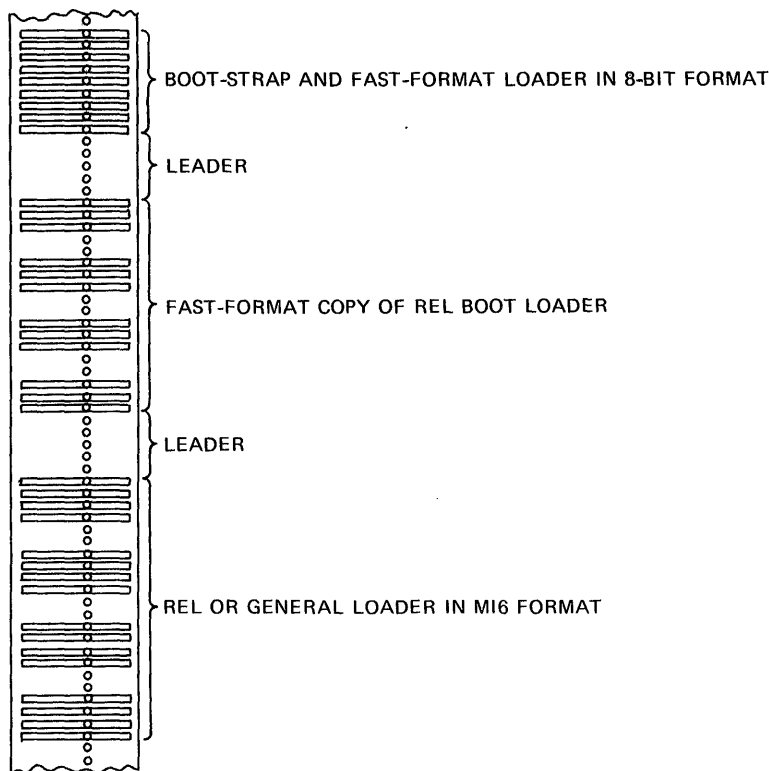


Figure 7-21. Loader Tape Format

This sequence requires that the proper 50 Sequence is used, including the Binary Input Device Definition in X'78'. The 50 Sequences are shown in Table 7-26.

Since the Loader portion of each tape is a relocatable object tape, it is possible to put the Loaders anywhere in memory. This can be done by using a bootstrap load to get the Relocating or General Loader into the top of memory. The BIAS can then be adjusted and any Loader can then be relocated to any arbitrary point in memory. Once relocated, CLUB can be used to dump an absolute tape of the loader in that location.

A Loader Summary is provided in Table 7-28.

CAUTION

Note that when loading the bootstrap Loader tapes, memory from X'0080' to X'02AA' and X'0DF0' to X'0FF6' is used. Any programs in this area of memory are overwritten.

TABLE 7-28. LOADER SUMMARY

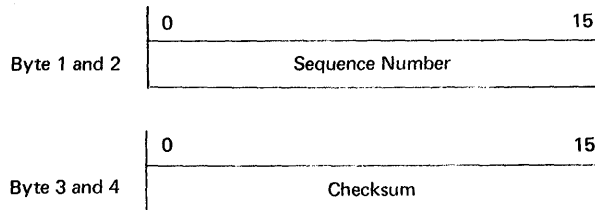
The following table is a summary of OS Library Loader Commands:			
CMD:	ARG1:	ARG2:	MEANING:
BIAS	BBBB		Set Bias to BBBB
BIAS			Set Bias to End of Loader
BC	LLLL		Set Length of Blank Common
LC	LLLL		Set Length of Labeled Common
OUT	LU	LABEL	Output Labeled Load-Module to LU
OUT	LU		Output Load-Module to LU
XOUT			End Load-Module, Write Last Record
GO			Transfer to Loaded Program
MAP	LU		Output Memory-Map to LU
LABEL	LU	NAME	Write Label-Record on File LU
FIND	LU	NAME	Position LU to Start of Named Prog.
COPY	LULU		Copy One Program from LU A to LU B
COPY	LULU	NAME	Find and Copy Named Program
DUPE	LULU		Duplicate Until EOF or Dev END
DUPE	LULU	NAME	Duplicate Until Name is Read
LOAD	LU		Purge Table, Load One Prog. from LU
LOAD	LU	NAME	Find and Load a Program
LINK	LU		Load One Program from LU, Linking
LINK	LU	NAME	Find and Link named Program
EDIT	LU		Load only Referenced Programs from LU
EDIT	LU	NAME	Find Named Label and Edit
REWIND	LU		Rewind LU
EOF	LU		Write a File-Mark on LU
TABLE	LU		Write Table-Of-Contents to LU
END			Exit to the OS
The following values pertain to the Stand Alone Loaders. All values below are expressed in hexadecimal:			
		Relocating Loader	General Loader
Starting address after boot load*		nC00	nA00
Restart Address in general#		ORG	ORG
Bias define address		ORG + 8	ORG + 8
Bias Definition value		ORG + A	ORG + A
Continue address		ORG + 26	ORG +26
Approximate Loader size		X'380'	X'590'
Illegal control items		C, E0, E1, E2, E3	E0, E1, E2, E3
Ignored control items		D, F	-----
*n = 0, 1, 2, 3, ... for memory sizes 4K, 8K, 12K, 16K, etc.			
#General Loader Restart - sets BIAS to X'80'			
- clears symbol table			
- clears any transfer address			
The following display indications are common to both Stand-Alone Loaders.			
Display Lights	Condition	Comment	
XX00	Normal End	Load Complete	
XX01	Checksum Error	Following checksum or sequence number error reposition tape and depress EXECUTE to re-read.	
XX02	Sequence Error		
XX03	Attempt to load over loader		
XX04	Ref-Chain Loader		
XXFn	Load Error	Improper control item detected where n is the bad item. Depress EXECUTE to ignore the data and continue. Refer to Table 7-29 for a definition of loader control items.	

TABLE 7-29.
CONTROL ITEM DEFINITIONS

Control Item	Meaning	Number of Data Items Following
0	Read next record	0
1	End of Program	0
2	Define chain	0
3	Toggle abs/rel mode	0
4	Load transfer address	4
5	Load program address	4
6	Load reference address	4
7	Load definition address	4
8	Data, 2 bytes absolute	4
9	Data, 2 bytes relative	4
A	Data, 4 bytes absolute	8
B	Data, 4 bytes relative	8
C	Symbol, reference	12
D	Symbol, definition	12
E	Decode Next Item	1
E0	Define Common-block	16
E1	Reference Common-block	16
E2	Common data, 2 bytes absolute	20
E3	Common data, 4 bytes absolute	24
E4	Reset sequence number to -1	0
F	Program Label	12

7.9.3 Object Tape Format

Standard format binary object tapes are divided into records; records are separated by bank leader. Each record contains 108 bytes of information. The first four bytes are organized as follows:



The sequence numbers are negative integers -1, -2, -3, etc. represented in two's complement form. The first record in a program must have sequence number -1. Subsequent records must be in proper order to be loaded.

The checksum is an odd parity Exclusive OR sum of all words in the record, except itself, plus a word of all ONE's.

The remainder of the record is a sequence of items; an item is four-bits or a half-byte. There are two types of items, control items and data items. There are 20 different control items, each of which is followed by a certain number (which might be zero) of data items. The control items, and their meaning are listed on Table 7-29.

Physically, two paper tape formats are used, both of which contain 108 bytes of data per record with blank leader between records. The Loaders use the first non-blank character of each record to identify which format is being used.

M16/17 Format: This format is used when punching paper tape on a high speed punch. The first non-blank character of each record contains a X'F0'. This character is followed by 108 frames each containing one data byte. Thus a complete record is 109 frames, of which the first is ignored. Each byte contains two four-bit "items".

M08/09 Format: This format is used when punching paper tape on a Teletype punch. A special set of non-printing characters is used, such that only the four low order bits of each frame contain data. See Table 7-30. If one of these special characters is encountered before a X'F0' character is read, the record is assumed to be of this format. Because only four-bits of each frame are used, the 108-byte record is punched in 216 frames of paper tape, of each the upper four-bits are ignored by the loader input routine.

Either format may be read through any paper tape reader. Figure 7-22 shows a loader record in two formats.

7.9.4 Features of the OS Library Loader

This loader functions under the BOSS or RTOS Operating System, interactively accepting commands and responding with messages to the console operator. Through available operator commands, a Program Library File may be created. This file may be searched for a particular program, selectively copied onto another I/O unit, or be added to under the operator's control. Automatic link editing is available, allowing the operator with one command to load all library programs required for any one particular job. A group of programs may be pseudo loaded, producing a single absolute load module which may be originated at any address, and then loaded by the Resident Loader, the Relocating Loader, or the General Loader. This tape contains forward references but no ENTRY or EXTRN symbols. It may be loaded over the Library Loader by the Resident Loader. A program loaded by the OS Library Loader may reference external symbols, define Common Blocks, and contain forward references. At load time, the operator specifies the logical unit to be used.

Library Loader Operation

The Library Loader should be loaded by the Resident Loader at the bottom of user memory. Any I/O devices to be used by the loader should be assigned. The loader should be started at its origin. It outputs the message "LOADER" indicating that it is ready to accept operator commands (which it request from Logical Unit 5). Every operator command consists of a word (of which the first two characters are decoded), followed optionally by a space and a hexadecimal operand, followed optionally by another space and an alphanumeric argument. A carriage return terminates the command if it is entered via a Teletype. Optional fields which are not used may be ignored. An example and description of each command follows. Following the execution of any command, the message "LOADER" is logged, and another command may be entered. See Table 7-28 for a summary of Loader commands.

BIAS bbbb

This command sets the Loader Bias, and thus the origin of the next relocatable program, to bbbb. After loading any program, unless a Bias command is issued, the Bias is set to the next available location above the highest address used. When the Loader is initially loaded, the Bias is set to the next location above the Loader. This value is not initialized when the Loader is restarted. If no Bias value is specified in this command, the Bias is set to the next location above the Loader.

TABLE 7-30.
TAPE CODES
(M08/09 FORMAT)

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

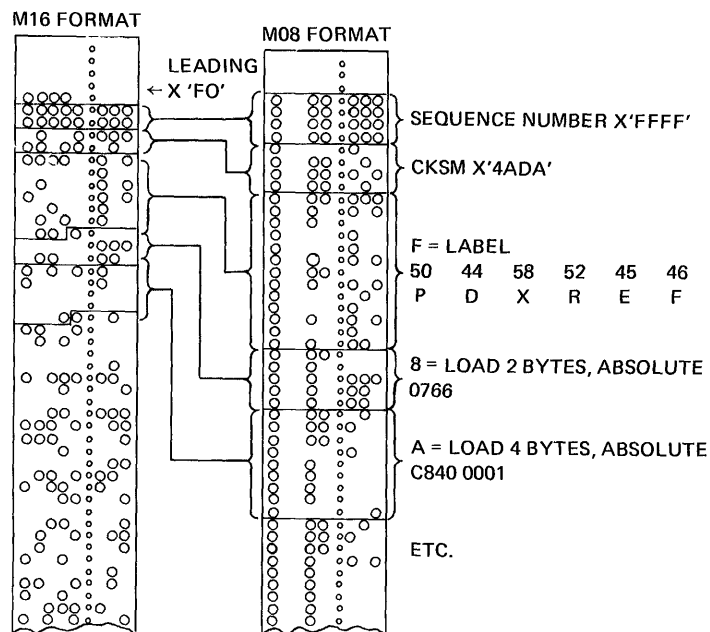


Figure 7-22. Object Tape Formats

FIND LU xxxxxx

This command causes Logical Unit LU to be rewound (if it is a magnetic tape) and searched from the beginning until a program labeled xxxxxx is found. The unit is then backspaced to the beginning of the found program. The program may now be LOAded or LINKed. This command should be used to position a file containing several programs. If this file is not a magnetic tape device, it should be positioned before the first record on the tape before issuing this command, and back-spaced one record after this command is completed, before attempting to load the found program. In the event that the requested program is not found, the Find Command will terminate whenever an end-of-file or device end is encountered on the device. In that case, the messages

EOF
xxxxxx NOT FOUND

are logged on the console device.

LOAD LU xxxxxx

This command is used to initialize the Loader and load a program at the current value of Bias. Before the program is loaded, the Loader's Symbol Table is cleared, and previously defined Common blocks are released. This command should be used to load the first of a group of user programs not related to any previous job. If a program Label is specified in the command, Logical Unit LU is first searched for LU program xxxxxx as described above under Find. If no Label is specified, the first program encountered on Logical Unit LU is loaded. Following a LOAD operation, the logical unit specified is left positioned past the end of the program loaded.

LINK LU xxxxxx

This command is used to load additional programs needed after using the Load command to load the first program. The Symbol Table is not cleared, so the loaded program may be linked to any previously loaded program, and may reference previously defined Common blocks. The program is loaded using the current value of Bias. If a program Label is specified in the command, Logical Unit LU is first searched for program xxxxxx as described previously under Find. If no Label is specified the first program encountered on Logical Unit LU is loaded and linked. Following a LINK operation, the logical unit specified is left positioned past the end of the program loaded.

EDIT LU xxxxxx

This command is used to search a library tape or file, and selectively load those programs needed to satisfy external references (EXTRN's) within programs previously loaded. The criterion for loading during an EDIT operation is if the Label of a program in the library file matches the symbolic name of any EXTRN in the Loader's Symbol Table. The EDIT command causes the following to occur.

1. If a program Label is specified in the command, logical unit LU is searched for program xxxxxx as described above under FIND. If no Label is specified, the logical unit is rewound.
2. The Loader's Symbol Table is searched for undefined EXTRN's. If there are none, the operation terminates.
3. If there are undefined EXTRN's in the Symbol Table, Logical Unit LU is searched forward from its current position for a label matching an undefined EXTRN. If such a Label is found, that program is loaded and linked, and Step 2 is repeated. If an end-of-file or device end is encountered while searching, an appropriate message is logged and the operation terminates. If the Label ENDEVOL is read, the EDIT operation is unconditionally terminated. This Label may be placed at the end of each tape or a paper tape program library to prevent reading off the end of the tape.

Following an EDIT operation, the MAP command can be used to determine if any more undefined EXTRN's remain in the Symbol Table.

MAP LU

This command outputs a Memory Map to Logical Unit LU. This may be a map showing the location of programs loaded into memory, or it may show the locations of programs output in a load module. In either case, the map includes only those programs processed during and after the last LOAD operation. (LINK and EDIT operations add to the map, LOAD begins a new one.) The map shows the starting address of each labeled program, the next available address, the location of every ENTRY defined, the starting address of each Common Block defined, and a list of any undefined EXTRN's. An example of a Memory Map appears in Figure 7-23. Under PROGRAMS in the map, the last value shown which has no name is the next available address in memory, which is the current value of Bias. Under COMMON-BLOCKS in the map, Blank Common is identified by the Label //.

PROGRAMS:			
2362 .U	238A .V	2398	
ENTRY-POINTS:			
2118 SUB1	236E .U	238E .V	
COMMON-BLOCKS:			
7FAE R	7FC2 X	7FCA Y	7FCE W
7F5E //			
UNDEFINED:			
ZERO	@I	.Q	.P

Figure 7-23. Memory Map (As Listed by the OS Library Loader)

BC nnnn
LC nnnn

These commands must be issued before the LOAD command if the program being loaded allocates Common. The Loader handles two types of Common. They are Labeled Common, and Blank Common. The chief difference between the two is that Labeled Common may be preloaded with data (FORTRAN BLOCK DATA) while Blank Common may not. The Loader allocates memory for common variables and links common variable references with the associated blocks of memory at load time. To use memory efficiently, the Loader must know in advance, the total length, in bytes, of each kind of Common used. BC sets the maximum length of Blank Common and LC sets the maximum length of Labeled Common to nnnn, where nnnn is the number of bytes to be reserved for common, expressed hexadecimally.

GO

This command transfers control from the OS Library Loader to the transfer address of the loaded programs. If no transfer address is specified, the message CMD-ERR is logged.

LABEL LU xxxxxx

This command is used in conjunction with adding programs to a library file. If the program to be added to the library does not have a Label, then this command can be used to give a program a Label in the file. The command causes the loader to output one record to Logical Unit LU containing the program Label xxxxxx. The Label must not exceed six alphanumeric digits. If it does, only the first six are used. This command could be used prior to an assembly or compilation which writes the binary object program to Logical Unit LU.

In conjunction with magnetic tape library formats, this operation searches for an end of file and then backspaces over the file mark prior to writing the Label, and generates an end of file and backspaces over the file mark after writing the Label. This command should never be used between an OUT and XOUT command. If it is, the command is rejected, and the message CMD-ERR is logged.

OUT LU xxxxxx

This command selects the output mode which is used to generate a load module rather than load programs into memory. In this mode, during LOAD, LINK, and EDIT operations, all data that would normally be loaded into memory is output in loader format to Logical Unit LU. All programs are output as absolute code, originated at the value of Bias in effect when the load module is created. Any external references or common references are resolved and converted to forward references without symbolic names. Thus, a load module program can be loaded by any INTERDATA loader, including the REL Loader or BOSS Resident Loader. By resetting the value of Bias for the load module, below the top of the OS Library Loader, and loading the load module with the BOSS Resident Loader, a user can effectively overlay the OS Library Loader at run time, although its features may be used at load time.

If a program Label is specified in the command, a Label record with the name xxxxxx is generated as described with the LABEL command prior to selecting the output mode.

XOUT

The command XOUT is used in conjunction with generating a load module. This command, which has no argument, generates the final record of the load module, and cancels the output mode. A typical command sequence to create a load module would be OUT, BIAS, LOAD, LINK, EDIT, and XOUT.

In conjunction with magnetic tape library formats, the OUT command searches for an end of file and then backspaces over the file mark so that the output module always becomes the last program on the tape. Also, the XOUT command generates an end of file following the last record, and then backspaces over the file mark.

REWIND LU

This command rewinds Logical Unit LU.

COPY LULU xxxxxx

This command causes one program to be copied from the first logical unit to the second. If a program Label is specified in the command, the first logical unit is first searched for program xxxxxx as described above under Find. If no Label is specified, the first program encountered on the first logical unit is copied. Following a COPY operation, the first logical unit specified is left positioned past the end of the program copied. The COPY command with an argument xxxxxx cannot be used with a non-backspaceable (paper tape) device. The COPY without an argument can be used to copy paper tape from the input device.

In conjunction with magnetic tape library formats, the COPY command searches the second logical unit for an end of file and then backspaces over the file mark prior to copying the program, and generates an end of file and backspaces over the file mark after copying the program.

EOF LU

This command writes an end of file to Logical Unit LU and then backspaces over the file mark.

DUPE LULU xxxxxx

This command causes programs to be copied continuously from the first logical unit to the second until program xxxxxx is found. The specified program is not copied, and the operation terminates. If no Label is specified, programs are copied until an end of file or device end on the first logical unit is detected.

In conjunction with magnetic tape library formats, the DUPE command searches the second logical unit for an end of file and then backspaces over the file mark prior to copying programs, and generates an end of file and backspaces over the file mark when the operation terminates.

To duplicate an entire file of programs from Logical Unit 1 to Logical Unit 2 the following command sequence is sufficient.

```
REWIND    2
EOF       2
REWIND    1
DUPE      0102
```

TABLE LULU

This command scans the first logical unit and lists all program Labels on the second logical unit. Thus a table of contents of a file containing many Labeled programs may be obtained. Any program in the file without a Label is not shown in the list.

END

This command returns control to the operating system. It may be issued at any time and the Loader may subsequently be restarted from its origin without losing any previously loaded data. The Loader tables are initialized only when executing a LOAD command. It is therefore possible to exit the loader temporarily (such as to reassign an I/O device) and return to it during a sequence of related operations.

Library Loader Error Messages

MEM-FULL

This message is logged if a program being loaded exceeds the available memory. This may happen if too little Common space is allocated (if the program defined Common) or if too large a program is loaded. A table of Labels, Common, and external symbol names is kept, building downward from the bottom of Common (see Memory Maps). If the program overlaps the table, the above message is logged, and the load is aborted.

CMD-ERR

This message is logged when the loader does not recognize an operator command, if a Label command is found between an OUT and XOUT commands or if GO is issued when no transfer address has been specified in any loaded program.

NO PROGRAMS

This message is logged if a MAP is requested and no labeled programs have been loaded.

EOF

A file mark has been read during binary input.

CKSM-ERR

A record has been read in which the checksum does not match the checksum computer by the loader. Following this message, the loader pauses. If the input device is paper tape, it may be manually backspaced one record before continuing the load. Magnetic tape is automatically backspaced on continue.

SEQ ERR

A record has been read out of sequence. Reposition the input unit and continue.

REF LOOP

A Loop has been found in a forward or external reference thread. The input tape has been generated incorrectly.

M xxxxxx

An entry point (xxxxxx) has been defined more than once. The load continues, and the previous definition remains in effect. The new definition is ignored.

LOAD ABORTED

An unrecoverable error condition has caused a load in progress to be aborted. It is necessary to repeat the original LOAD command, as the table may contain incorrect symbol locations.

ADRS-ERR

A program has attempted to load below the top of the Loader. The program may contain absolute data at such an address, or the BIAS may be set incorrectly. The load process is aborted.

DEV END

A device went off line or returned an END-OF-MEDIUM status during input.

I/O DEV ERR

A parity error has been detected on input or on off line condition has been detected on output. If it is an input error, the BOSS status byte and device address in hex-notation appear on the next line.

LOAD ERR

An illegal control item was detected during a load.

xxxxxx NOT FOUND

An end of device condition has occurred before locating the Label xxxxxx during a search operation.

Magnetic Tape File Procedures

The OS Library Loader may be used to create, maintain, and manipulate a file of programs, using the set of operator commands described. A library file must be in standard loader format, and each program must be Labeled. Following the last library program, there must be a file mark. Each of the commands which may be used to write out a binary object record is designed with magnetic tape in mind. Before any record is written by the LABEL, COPY, DUPE, and OUT directives, the output file is positioned to the gap immediately preceding the file mark. This is done by searching forward for the mark, and then backspacing over it. After writing the final record of a program or load module, a file mark is written, and backspaced over. It should be pointed out that these functions have no effect on non-magnetic tape devices, and so these commands are not limited to magnetic tape operations. Because these directives never allow a write to a magnetic tape without first positioning it past its end of information, they may only be used to add to an existing library.

To create a new library, the REWIND and EOF commands are used to place an initial file mark on the beginning of the tape. Once this is done, programs may be copied onto the tape by using the COPY or DUPE commands. If the program to be copied onto the tape does not already have a Label, a preceding Label may be written before it using the LABEL command. As programs are added to the file, the file mark is propagated along, following the last one.

Any time the library is read, and a file mark is encountered, the operation is terminated immediately, the message EOF is logged, and the tape is backspaced over the mark. If a LOAD, LINK, EDIT, FIND, COPY, or DUPE terminates with EOF, it indicates that the system was searching for a Label not found on the tape being searched. It is the normal termination for the TABLE command.

Paper Tape to Mag Tape Procedures

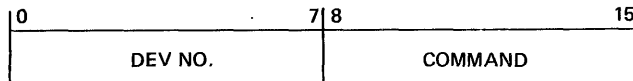
Since paper tape devices are not backspaceable, the FIND LU xxxxxx requires the operator to manually backspace one record when xxxxxx is found. The user may either use a COPY LULU (no name) or a combination of FIND LULU xxxxxx, manually backspace, and then COPY LULU (no name) to copy paper tape programs.

If it is desired to write a program on a library file by some other program (such as the object output from a compiler or an assembler) the tape should first be positioned to the end with the command FIND LU (no name). Then the new material should be added and then the EOF command should be issued. If the LABEL command is issued prior to the assembly or compilation, it leaves the file positioned past the Label written, at the end of information.

7.9.5 Features of Stand-Alone Loaders

Unlike the device independent OS Library Loader, the stand-alone loaders contain drivers for only Teletype and paper tape devices. The following general comments apply to the REL and General Loaders:

1. The input device for loading is defined by the Binary Input Device in the Device Definition Table in low memory. Specifically, the halfword at X'78' is interpreted as follows:



This halfword for various devices is shown below:

Teletype	0294
High Speed Paper Tape Reader	0399
High Speed Reader/Punch	1399

2. When reading binary data from tape, leader and illegal characters are skipped.
3. Checksums and sequence numbers are checked after each binary record is read. Appropriate error halts are used when errors are detected.
4. All loaders are provided in Relocatable bootstrap form. Specifically, each tape contains a re-locating bootstrap sequence followed by the actual loader in normal relocatable object format. When using the bootstrap sequence, the loader is placed into the top of available core memory. Once the REL or General Loader is in the top of core, any of the Loaders can then be relocated to any other arbitrary point in memory.
5. The first location (ORG) of each Loader is the starting location.
6. While a tape is being read, the Loaders output the data bytes to Register Display 2 for confirmation of Loader operations.
7. In the REL Loader, Control Panel Display Register 2 is used to identify the meaning of loader halts. The light patterns used are:

XX00	NORMAL END
XX01	CHECKSUM ERR
XX02	SEQUENCE ERR
XX03	ATTEMPT TO LOAD OVER LOADER
XX04	REF LOOP
XXFX	X-CONTROL ITEM ENCOUNTERED

Refer to Table 7-28 for a summary of improper control items.

8. The Loaders transfer to the program loaded, if specified on the object tape.

When the REL or General Loader is executed at its starting address (ORG), the Bias value is set to X'80'. This Bias value is used during program loading to adjust any relocatable data values. Note that absolute programs are stored at the absolute location specified for the data. Relocatable programs are stored from the location indicated by Bias upward into memory. After a program has been loaded, the Bias value is adjusted to point to the next available location. To indicate that the load is complete, the Loader halts with the WAIT light illuminated, and with XX00 contained in the Register Display. At this time, the adjusted Bias value is held in Register 0. This register may be displayed on the panel.

If more programs are to be loaded, place the next tape in the reader, with the rotary Function switch in RUN, depress EXECUTE. This procedure starts the loader executing at ORG + 26, which is the continue location. The continue operation uses the current value of Bias, and does not reset it to X'80'. Multiple relocatable tapes are thus loaded one after another into adjacent areas of core memory.

If it is desired to load a relocatable program at any arbitrary point in core memory, it is necessary to redefine the Bias value. To adjust the Bias pointer, use the following procedure.

1. Change the halfword at ORG + A in the Loader to the desired Bias value.
2. Start the Loader executing at ORG + 8, rather than the normal start or continue location.

Note that the value at ORG + A remains until changed to a new value. The Loader can always be restarted at ORG + 8 which resets the Bias to the value contained in ORG + A.

7.9.6 General Loader Features

In addition to the capabilities already discussed, the General Loader provides various features not available with the Relocating Loader.

Bias Printout

At the start of every load operation, the General Loader types the current value of the Bias pointer on the Teletype. This printout occurs prior to reading the first record of a new program, and the message is of the form

BIAS = XXXX

where the XXXX represents the current Bias value in hexadecimal form.

Messages

Other messages which are typed on the Teletype are as indicated in Table 7-31.

ENTRY/EXTRN Handling

Programs generated by the Assembler can use ENTRY's or EXTRN to achieve cross referencing and linkage with external programs. In this case, the object tape for these programs contains the symbolic names declared as ENTRY's or EXTRN's. The General Loader uses a Symbol Table to remember these names when a program is loaded. This Symbol Table builds downward in core memory from the origin (ORG) of the General Loader. Each entry in the Loader Symbol Table requires eight bytes of memory.

Since the Loader Symbol Table is building downward into memory, and the programs being loaded are building upward into memory, the Loader checks to see that the loading program does not overwrite the Symbol Table. If the loading program requires data stored above the current bottom of the Symbol Table, a MEMORY FULL message is generated, and the Loader halts.

When the General Loader is executed at its start location (ORG) or its BIAS redefinition location (ORG + 8), the Symbol Table is cleared of all names. Executing the General Loader at its continue location (ORG + 26) does not change the state of the Symbol Table.

At the end of each program load, the Symbol Table is scanned for undefined symbols. Any undefined symbols are typed in the form:

U XXXXXXX

TABLE 7-31.
ERROR MESSAGES

Message	Meaning
CKSM ERR	A checksum error was detected after reading the previous record. Reposition the tape to the beginning of the record and depress EXECUTE to reread the record.
SEQ NUM ERR	A sequence number error was detected after reading the previous error. Reposition the tape to the proper record and depress EXECUTE to try again. This error usually occurs when the tape is improperly positioned following a checksum error.
MEMORY FULL	This message is caused by a conflict between the General Loader and the loading program. The program being loaded has not been loaded to conclusion. The alternatives are: <ul style="list-style-type: none"> A. Load Fewer programs B. Make absolute tapes of the programs to be loaded and then use the REL Loader which requires less memory. C. Eliminate some EXTRN's and ENTRY's to reduce the size of the Symbol Table. <p>Note that the General Loader cannot load programs above itself in memory.</p>
NORMAL END	This case occurs when a program has successfully loaded and no END transfer address has been specified or if undefined external references remain. All undefined external references are listed on the Teletype preceded by a U prior to printing the NORMAL END message. If a transfer address is specified and no undefined symbols remain, the Loader transfers directly to the address specified, and no NORMAL END message occurs.
LOAD ERR	This message results if an illegal control item is detected during load. Depress EXECUTE to ignore the control item and proceed with the load. The E control item is explained in Section 7.9.3.
REF LOOP	This message results if an endless forward reference or external reference chain is encountered. It indicates that the input tape was generated incorrectly.

Where XXXXXX is the symbol name. All such undefined names are printed preceding the NORMAL END message. An undefined symbol results from the fact that the symbol was declared as an EXTRN in some program, and no program yet loaded has declared that same symbol as an ENTRY. As soon as some loading program declares that symbol as an ENTRY, the symbol becomes defined. If more than one program declares a symbol as ENTRY, the message:

M XXXXXX

is typed at the time the multiple definition occurs, where XXXXXX is the symbol name. In this case, the first value defined remains in the Symbol Table, and the second definition value is ignored.

At the end of each program load, the Loader transfers immediately to the program that has been loaded, only if a transfer address is specified on the tape, and if the Symbol Table contains no undefined symbols. If any symbols in the table are undefined at the end of a load, those symbols are listed, NORMAL END is printed, and the Loader halts, waiting to load the next program.

Forward Reference Definitions

Program object tapes generated by one-pass assemblies or load modules involve forward references to symbols which are defined later in the program. The Loader uses a chaining procedure for satisfying any forward references at the time the symbol definition is encountered.

An example of a forward reference in a program is:

```
                OPT      PASS1, PUNCH
                LH       3, SAM
                BR       5
    SAM          DC      3
                END
```

In this case, the reference to SAM occurs before SAM is defined. There are several restrictions on the use of forward reference during one-pass assemblies, and on the use of symbols which are ENTRY's or EXTRN's for the program to be loaded properly. The restrictions are:

1. Such symbols must not be combined in arithmetic expressions such as:

```
LH 3, SAM + 2
```

2. Such symbols must not be used in the R1 or R2 field for an instruction such as:

```
LH 3, 2(SAM)
```

3. Such symbols must not be used as operands assembler pseudo-ops such as DO, EQU, END etc.; for example:

```
DO SAM
```

Note that with the EQU statement, the operand must be defined when it is used, with one, two, or three pass assemblies.

Label Handling

Programs generated by the assembler can be labeled through the use of the OPT pseudo-op such as:

```
OPT PASS2, PUNCH, LAB = ABCDEF
```

The program Label can be up to six characters. The first character must be a letter; subsequent characters can be letters or digits. The object tape, in this case, contains the program LABEL in symbolic form. When the General Loader detects a program Label, the Label is typed in the form

```
LABEL = ABCDEF
```

If object tapes which contain label's are loaded by the REL Loader, the Label is ignored by the Loader.

7.9.7 Operation of Stand-Alone Loaders

The steps required to load and operate the Stand Alone Loaders are summarized below.

1. Manually enter the 50 sequence into memory if it is not already there. Specify the device to be used for loading at location X'78', the Binary Input Device definition. See Table 7-26 for a listing of the 50 Sequences.
2. Place the Loader tape in the tape reader with the first data character before the read fingers or photo diodes.
3. Enter X'0050' into the Control Panel switches, select ADRS Mode and depress EXECUTE.
4. Depress INITIALIZE. Select Run Mode, and depress EXECUTE.

5. If an ASR 33 Teletype is being used as the input device, it is necessary to toggle the reader switch to START, which starts the tape moving. If an ASR 35 Teletype is in use, the MODE switch should be put in RUN to start the tape. If a High Speed Paper Tape Reader is in use, the tape will start by itself.
6. If no input errors occur, the entire tape is read to the end, at which time the Processor halts with the WAIT light illuminated, and XX00 contained in the Display Register.
7. If checksum errors are detected during tape input, the tape will stop and the Processor will halt with the WAIT light illuminated and XX01 contained in the Display Register. When this occurs, reposition the tape to the previous record gap, and depress EXECUTE to reread the record. If the error halt occurs due to the first record on the tape, restart the entire load procedure.
8. Put the tape to be loaded into the tape reader. If the tape to be loaded is relocatable, and a specific Bias value is required, enter the Bias value into $ORG + A$, and set the starting address to $ORG + 8$. If the tape to be loaded is absolute, or if the Bias value of X'0080' is satisfactory, set the starting address to ORG . Depress INITIALIZE. Select Run Mode, and depress EXECUTE.
9. If improper control items are detected during the load, the tape will stop, and the Processor will halt with the WAIT light illuminated and XXFn contained in the Display Register, where n is the bad control item. When this occurs, it must be determined if the right Loader is being used. That is, if the object tape involves ENTRY's or EXTRN's or forward references, the General Loader should be used. If Common blocks are called for, the OS Library Loader should be used. If the Loader is appropriate, and the tape is proper, depress EXECUTE to skip the improper data and proceed with the load.
11. When the load is complete, the tape stops. If no transfer address is specified on the tape, the Processor halts with the WAIT light illuminated and XX00 contained in the Display Register. If a transfer address is specified, the Relocating Loader will transfer directly to the location specified. The General Loader transfers only if the Symbol Table contains no undefined symbols.
10. If checksum errors are detected during the load, the tape stops and the Processor halts with the WAIT light illuminated and XX01 contained in the Display Register. Reposition the tape to the start of the last record read, and depress EXECUTE to reread the record.
12. If more tapes are to be loaded, return to Step 8 and repeat the process.

7.10 EDITOR (TIDE) PROGRAM

TIDE is an on-line, interactive text editor program. It is designed to create and modify character-oriented text material which is stored on paper tape, or input through the Teletype keyboard. The text may be an assembly language program, a FORTRAN program, or any text in the literal sense.

TIDE is directed by an operator through the keyboard of a Teletype terminal. Upon receiving a keyboard input directive, the editor will read text from a specified input device into a designated area of core memory. The editor allows the user to examine, change, delete, and/or modify the text while it remains in core memory. When the editor receives a keyboard output directive, the revised text can then be output to the specified output device.

7.10.1 Program Structure

Operating Modes

TIDE has two modes of operation: Command and Edit. The program indicates the current mode by printing, in Column 1 of the Teletype, a left arrow (←) for the Command Mode or an asterisk (*) for the Edit Mode. In the Command Mode, the program accepts keyboard commands which specify an editing procedure, or which specify a text input or output operation. From an Edit command, TIDE enters the Edit Mode. Edit Mode allows the user to insert, append, or modify the text, after which control returns to the Command Mode.

Basic Unit

The basic unit of the stored text is a variable length line from 1 to 81 ASCII code characters long including a line terminating Carriage Return (CR). Each line of input is stored in a line buffer until the CR terminates the input, or the buffer becomes full. The line buffer contents are then moved to the text buffer. If the text buffer contains a symbolic source program, each source statement is one line of text. The statements, or lines, have unique decimal addresses which are sequenced in ascending order; the first statement in the buffer has address number one (1). This allows editing of any statement by line address rather than core location address.

Line Addressing

A specific line can be referenced by its decimal number address *n*. To examine line *n*, type the decimal number followed by a carriage return. The Teletypewriter will list:

n ZZZ...Z

where *n* is the line number, and *Z* the text contained in line *n*. This becomes the line currently available for examination or modification and is called the open line. All forms of line examinations are exclusive to the Command Mode (←) and will never cause transfer to the Edit Mode (*). Any attempt to examine a line not contained in the buffer, or to reference a nonexistent line is ignored and a question mark (?) is typed on the Teletype.

The execution of some TIDE commands will change the position of a line in the text buffer, and consequently change the line number. This line number change does not affect the contents of the line.

To facilitate line addressing and determine the number of bytes used in the text buffer, the symbols in Table 7-32 have been implemented. All of the symbols with the exception of the up arrow (↑) cause the listed line to become the open line.

TABLE 7-32.
LINE ADDRESSING SUMMARY

Symbol	Terminating Character	Function
n	CR	Opens and lists line number n (a decimal number)
Carriage Return	None	Opens and lists the line preceding the current open line
Line Feed	CR	Opens and lists the line following the current open line
*(asterisk key)	CR	Lists the current open line
.(period key)	CR	Lists the last line in the text buffer
↑ (upper arrow key)	CR	Lists the byte count of the contents of the text buffer. The count is shown in decimal. A count greater than 9999 causes an error message.

Command Formats

Commands are entered through the keyboard in one of the following general formats:

Format	Terminating Character	Description
X	CR	Editor performs Command X
X n	CR	Editor performs Command X on n lines
X a,b	CR	Editor performs Command X on lines a through b inclusive. Both a and b must be positive with b not less than a

In each command, n, a, and b are decimal numbers, hereafter called arguments, and command X directs the editor to perform the specific operations described later in the manual. All command formats are terminated by a CR. If there is an error in the command format, no action is taken and program control returns to the Command Mode (←). One or more spaces should separate the arguments from the command.

Line Addressing and Command Arguments can involve arithmetic using addition and subtraction. Some sample formats are shown in Table 7-33. When using line arithmetic with two argument commands (general format X a, b), the rules for a and b still apply, i. e., both a and b are positive and $b \geq a$.

Commands

The three main functions of the TIDE editor program are input, modification, and output. The input commands are used to enter text into TIDE's buffer. The input commands are Append (A), and Insert (I). The modification commands direct various manipulations of the text already stored in TIDE's buffer. The modify commands are Delete (D), and Change (C). The output commands produce a hard copy of the text on the command-specified output device. That is, TIDE dumps its text buffer in a printout or punches a source tape, as requested. The output commands are Print (P), Output (O), and List (L). Command-specified I/O devices are explained in Section 7.10.2.

TABLE 7-33.
LINE ARITHMETIC

Sample Format	Terminating Character	Description
2+7	CR	List line number 9
.-8	CR	Lists the eighth line before the last line in the text buffer
.*	CR	Lists the line indicated by subtracting the decimal number address of the current open line from the decimal number address of the last line in the text buffer.
X *+3	CR	Editor performs command X on the third line following the current open line.
X *+3,.-6	CR	Editor performs command X on lines (*+3) through (.-6), inclusive.

Additional editing capability is provided with the setup commands. The Tabulate (T) and No Tabulation (N) commands allow the user to select or suppress assembly listing tabulation of his text during Print or List operations. The Kill text buffer (K) command allows the user to start TIDE's execution at its origin, clear the enter text buffer, and enter the Command Mode to begin again.

Reproduce (R) and Skip Tape (S) commands are provided to facilitate the merging of partially edited text tapes with the user's updated text in the buffer. The Reproduce command copies text from one device to another and the Skip command can be used to pass over lines of text on a source tape.

Refer to Table 7-34 for the definitions and descriptions of TIDE's command repertoire.

Command Examples

This section presents examples of the user-TIDE interaction using the Append, Print, Change, and List commands. Example (a) shows a user appending text. Examples (b), (c), and (d) are interdependent in that they show a Print, Change, and List of the same buffer of modified text. In the following, the symbol XX refers to a nonprinting keyboard input.

(a) Append

Append two lines of text to the current text buffer.

←

A CR

* APPEND LINES TO CR

* THE TEXT BUFFER CR

* BK

←

(b) Print

Print lines one through five without tabs.

←

P 1,5 CR

LINE ONE STILL LINE NUMBER ONE

LINE TWO DELETED IN EXAMPLE C

LINE 3 NUMBER WILL CHANGE

LINE 4 NUMBER WILL CHANGE

LINE 5 NUMBER WILL CHANGE

←

(c) Change

Given example b, assume the open line is number two. Insert two lines. Open line number four is printed.

←
C (CR)
* INSERT A LINE (CR)
* INSERT ANOTHER LINE (CR)
* (BK)
4 LINE 3 NUMBER WILL CHANGE

(d) List

List lines one through five for example(c)after C command execution. Note the change in line numbers.

←
L 1,5 (CR)
1 LINE ONE STILL LINE NUMBER ONE
2 INSERT A LINE
3 INSERT ANOTHER LINE
4 LINE 3 NUMBER WILL CHANGE
5 LINE 4 NUMBER WILL CHANGE

←
The letters n, a, and b as used in Table 7-34 have the following definitions:

n = a decimal number

a = first argument, a decimal number

b = second argument, not less than a, a decimal number

TABLE 7-34.
COMMAND REPERTOIRE

Function	Command	Response	Definition	Description
Input	A	*	Append	The editor enters the Edit Mode (*) and accepts input from the keyboard. The typed text line is appended following the last line of text (if any) in the buffer. Each line of input is terminated with a CR. After a * response, the Append operation is terminated by depressing BK. After termination, the last line input, now the open line, and its decimal number address are printed. TIDE returns to the Command Mode (←).
Input	A n	none	Append n lines	TIDE enters the Edit Mode after which n lines are read from the Source Input Device. See Section 7.10.2. After termination, this command continues as the A command.

TABLE 7-34.
COMMAND REPERTOIRE (Continued)

Function	Command	Response	Definition	Description
Input	I	*	Insert	The editor enters the Edit Mode (*) and accepts text lines from the keyboard to be inserted preceding the open line. Insertions are made in the order in which lines are input. Each line is terminated by depressing CR and the operation is terminated by depressing BK. Upon termination, the open line with its corrected decimal address is printed and control goes to the Command Mode (←).
Input	I n	none	Insert n lines	Control transfers to the Edit Mode after which n lines of text are read from the Source Input Device (See Section 7.10.2) and are inserted into the text buffer as described for the I command. Upon termination, the open line with its corrected line number is printed and control goes to the Command Mode (←).
Modify	D	none	Delete	The editor deletes the current open line. The line following the deleted line is now the open line and is printed along with its corrected line number. Control returns to the Command Mode (←). If the last line in the buffer is the open line and it is deleted, the new last line is now the open line and it is listed.
Modify	D a, b	none	Delete lines a through b	Lines a through b inclusive are deleted. Line b+1 becomes the open line and is printed along with its corrected decimal address. TIDE remains in the Command Mode (←). If line b+1 is non-existent, the last line is opened and listed.
Modify	C	*	Change	The open line is deleted. Control then goes to the Edit Mode (*) through which insertions can be made as in the I command. After insertions are made, the command is terminated by depressing BK. The line following the deleted line becomes the open line and is printed along with its corrected decimal number address. Control transfers to the Command Mode (←). If the last line in the buffer is the open line and it is changed without insertion, the new last line is now the open line and is listed.

TABLE 7-34.
COMMAND REPERTOIRE (Continued)

Function	Command	Response	Definition	Description
Modify	C a, b	*	Change lines a through b	This command deletes lines a through b inclusive then continues as the C command.
Output	P	none	Print	This command must be preceded by the T or N command. The P command prints the contents of the text buffer in T or N unnumbered line format on the List Device (See Section 3.2). The editor remains in the Command Mode (←).
Output	P a, b	none	Prints lines a through b	Lines a through b inclusive are printed as in the P Command.
Output	O	none	Output punched tape	The entire text buffer is punched in the standard source tape format on the Binary Output Device. TIDE remains in the Command Mode (←).
Output	O a, b	none	Output lines a through b on punched tape	This command punches lines a through b inclusive and continues as the O Command.
Output	L	none	List	The contents of the text buffer are printed in T or N numbered line format on the teleprinter. Printing may be terminated by depressing BK. The open line will be listed. The editor remains in the Command Mode (←).
Output	L a, b	none	List lines a through b	Lines a through b inclusive are printed and terminated as in the L Command.
Setup	T	←	Tabulate output	This command sets the tabulate flag for format control. Output is similar to the Assembler format when the List and Print commands are subsequently used.
Setup	N	←	Untabulated output	This command resets the tabulate flag for no format control. Output spacing is as in the text buffer when the list and Print commands are subsequently used. This flag state is N when TIDE is started at the origin.
Setup	K	TIDE ←	Kill text buffer	This command starts TIDE at the ORG as described in Section 7.10.2, Starting Location.

TABLE 7-34.
COMMAND REPERTOIRE (Continued)

Function	Command	Response	Definition	Description
Other	R	none	Reproduce text	This command reads lines of text from the Source Input Device and reproduces the text on the Binary Output Device. See Section 7.10.2. No information from the tape enters the text buffer. Duplication may be terminated by taking the Binary Input Device off-line. TIDE remains in the Command Mode (←).
Other	R n	none	Duplicate n lines of text	The number of lines specified in the argument are duplicated on the Binary Output Device. I/O devices are as stated for the R command. No information from the tape enters the text buffer. After n lines are reproduced, the last line reproduced is printed. TIDE remains in the Command Mode (←).
Other	S	none	Skip	The Binary Input Device (See Section 7.10.2) advances until halted. After manual halt, the open line and its number are printed. The editor remains in the Command Mode (←).
Other	S n	none	Skip n lines	The number of lines specified in the argument are skipped as in the S Command. After n lines are skipped, the last line skipped is printed.

Errors

The error message for improper TIDE command entries or for a line of text (from any input device) which exceeds the character limit, is the question mark (?). If a command entry error is made, no action is taken upon the information in the text buffer, TIDE responds with an error message (?), and remains in the Command Mode (←). If the text line exceeds 80 characters, the error message (?) is printed and program control is transferred to the Command Mode (←). None of the characters in the line are entered into the text buffer. If the error occurs in either the command or text entry modes, and is discovered before depressing the CR, the mistake may be corrected. Corrections are made by typing a left arrow (←) which deletes the last character in the line, or by typing a symbol (#) which deletes the entire line. Control remains in the current mode of operation.

Another TIDE error flag is the exclamation point (!) which means the text buffer has overflowed. When this happens, the line which caused the overflow is not entered into the text buffer and the text buffer is unchanged. TIDE returns to the Command Mode (←). To enter more information, it is necessary to delete one or more lines from the buffer. To reread the line which caused overflow, it is necessary to backspace the input tape one line, and adjust the buffer contents to make more room.

In the event of an I/O device error, TIDE prints the message "IOERR XXNN" where XX is a status word specifying device status and NN is the device number.

0	7 8	15
STATUS CODE	DEVICE NUMBER	

Control returns to the Command Mode (←). After the device problem is remedied, re-issue the appropriate TIDE command.

7.10.2 Operating Procedures

Loading

TIDE is available in two versions: Program Number 03-026 is a stand-alone version with self-contained input/output drivers; Program Number 03-027 is designed to execute under the Operating System and consequently has no internal input/output capability. Both versions are functionally the same.

Program Number 03-026 is a relocatable program which requires about 5K bytes of memory including a 2000 byte text buffer. To load 03-026, use the Relocating Loader, Program Number 06-024, or the General Loader, Program Number 06-025. Refer to Loader Descriptions Manual, Publication Number 29-231, for use of these loaders.

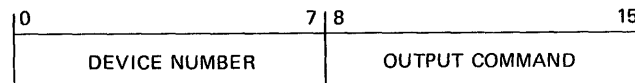
Program Number 03-027, is a relocatable program which requires about 4K bytes of memory including a 2000 byte text buffer. To load 03-027, use the LOAD Command in the Operating System.

I/O Device Selection

Prior to executing the stand-alone version of TIDE, Program Number 03-026, the following halfwords in the Device Definition Table of the 50 Sequence should be set up.

Location	Symbolic Name	Used With TIDE Commands
X'007A'	BOUTDV	Reproduce, Output
X'007C'	SINDV	Append, Insert, Reproduce, Skip
X'007E'	LISTDV	Print

Each halfword contains the following information.



The appropriate halfwords for various devices are shown below.

BOUTDV	X'0298'	Teletype Punch
BOUTDV	X'1392'	High Speed Paper Tape Punch
SINDV	X'0294'	Teletype Input with Printing
SINDV	X'02A4'	Teletype Input, no Printing
SINDV	X'1399'	High Speed Paper Tape Input
LISTDV	X'0298'	Teletype Output
LISTDV	X'6280'	Lineprinter Output

Prior to executing the Operating System version of TIDE, Program Number 03-027, the following device assignments should be made to the logical units listed below.

Logical Unit	Function	Used With TIDE Commands
01	Source Input	Append, Insert, Reproduce, Skip
02	Binary Output	Reproduce, Output
03	List	Print
05	Keyboard	Command Entries, TIDE Responses, Append, Insert, Change, List

Greater device flexibility is available with the Operating System version of TIDE than with the stand-alone version. Refer to Section 7.12 for details on device assignments. Possible specific device assignments follow:

Source Input	01FF	Teletype Input (paper tape)
Source Input	0102	Teletype Input Keyboard
Source Input	0113	High Speed Paper Tape Input
Source Input	0104	Card Reader Input
Source Input	0185	Magnetic Tape Input
Binary Output	02FF	Teletype Punch
Binary Output	0213	High Speed Paper Tape Punch
Binary Output	0285	Magnetic Tape Output
List	0302	Teletype Output
List	0362	Line Printer Output
Keyboard	0502	Teletype Input/Output

Starting Location

If the execution of TIDE is started at the origin (ORG), the text and line buffer pointers are set to the first locations of the buffers. Registers and appropriate locations are initialized, and the message TIDE, along with the Command Mode Flag (↔) is printed.

If TIDE is started at location ORG + 4, no initialization occurs, the current state of the buffers and pointers is unchanged, and the Command Mode Flag (↔) is printed.

Termination Commands: E and !

Two additional commands are available in TIDE to terminate execution and in the operating system version, to return control to the operating system. To momentarily halt program execution, type an exclamation mark (!). Stand-alone TIDE types "PAUSE" then halts. Depressing EXECUTE causes execution to restart at ORG + 4. The OS version issues a SVC pause to the operating system. Typing CONTINUE causes TIDE to restart at ORG + 4. To end program execution, type E. Stand alone TIDE types EOJ (end of job) and halts. Depressing EXECUTE causes execution to restart at ORG. OS TIDE issues an SVC End of Job.

Tape Format

Tapes produced by the Output (O) command of TIDE are always in the standard source tape format. Each line of binary text is followed by a CR, LF, and eight blanks. The format for each character is seven bit ASCII code.

Input tapes for TIDE should be in the standard source tape format; however, the minimum format requirements are that each line should be terminated by a Carriage Return (CR) but must contain no more than 80 characters. In addition, successive statements must be separated by at least five or six blank characters. If a line length error (?) occurs, the tape must be manually adjusted to the next line. If the text buffer capacity is exceeded (!), manually backspace the tape one line. In the latter case, editing can be done on the text already in the text buffer.

Text Buffer Size

When loaded, TIDE provides a text buffer for 2000 characters. The user may adjust the size of the text buffer by inserting the beginning and ending absolute addresses into locations X'0008'R and X'000A'R respectively. The text buffer may be located anywhere in core, providing it does not overwrite TIDE or other resident programs. When started at ORG, TIDE tests the text buffer limits and if they overwrite, TIDE forces them to the locations originally specified in the editor. Table 7-35 is a summary of tide features.

TABLE 7-35.
SUMMARY OF TIDE FEATURES

SYMBOL	DEFINITION
<p>TIDE ← * ? ! IOERR XXNN</p>	<p>Initial Execution Message Command Mode Edit Mode Error Message Text Buffer Overflow I/O Device Error</p>
TIDE CONTROL CHARACTERS	
KEY	DEFINITION/ACTION
<p>* (CR) (CR) ↑ (CR) (LF) (CR) (CR) # ← (BK) E !</p>	<p>Lists the open line Lists the last line in the text buffer Lists the byte count of the current contents of the text buffer Opens and lists the line following the current open line Opens and lists the line preceding the current open line Erase the line just typed from the keyboard, cancels an incorrect command Deletes the last character typed Returns to Command Mode End of job. Pause</p>
SPECIAL TIDE ADDRESSES	
HEXADECIMAL LOCATION	DEFINITION
<p>0000 + Bias 0004 + Bias 0008 + Bias 000A + Bias</p>	<p>Starting location. Program initializes and resets buffer pointers. Restart location. Program does not initialize and buffer pointers are not reset. Location which defines the first address of the text buffer. This address must not overwrite the TIDE Program. Location which defines the last address of the text buffer. This address must be within core limits and greater than the starting address of the text buffer.</p>

7.11 HEXADECIMAL DEBUG (CLUB) PROGRAM

The Hexadecimal Debug Program, known as CLUB, is an on-line hexadecimal debug program. Its purpose is to provide maximum assistance in debugging a user program while using a minimum of memory storage. The user of CLUB directs the debugging operation by entering directives and associated data via the Teletype keyboard. Responses to these inputs are usually shown on the Teletype page printer. Features of CLUB are as follows:

- Memory cell examination and modification
- Register examination and modification
- Relative addressing of cells
- Address arithmetic
- Execution of user program
- Multiple breakpoints
- Search on limits for masked value
- Print the content of memory bounded by limits
- Punch the content of memory bounded by limits in standard loader format
- Punch the content of memory bounded by limits in eight-bit format
- Disassemble the content of memory using instruction mnemonics

The two versions of CLUB that are available are:

1. 03-013 Hex Debug
2. 03-032 OS Hex Debug

Hex Debug, 03-013, is a self-contained version which includes all features of CLUB, along with I/O routines for supporting CLUB operations on a Teletype, High Speed Paper Tape Reader or Punch, and a Line Printer. This version runs on any INTERDATA Processor with 4K bytes or more of memory.

OS Hex Debug, 03-032, is similar in features, but this version requires one of the INTERDATA operating systems for I/O. This version runs on any INTERDATA Processor with 8K bytes or more of memory. All I/O in OS Hex Debug is performed via logical units which are assigned at run time by the OS Assign Command.

7.11.1 Terminology

The term cell, as used in the following discussion, refers to a halfword (16-bit) memory location.

An open cell is the cell that is currently available for modification. To operate on a cell in memory, it must be made the current open cell. Only one cell at a time is considered open.

Directives are instructions to CLUB and consist of a single character (Teletype key), other than 0-9 or A-F. Directives are given to CLUB through the Teletype keyboard, see Figure 7-24. Each directive initiates an action. Most directives are to be preceded by an argument, and followed by a carriage return.

The argument is the address or data value which is used by the directive. Leading zeros are not necessary. Data inputs are in a hexadecimal format using the characters 0-9 and A-F. All other characters are assumed to be directives to CLUB. An undefined directive given to CLUB is completely ignored and no action is taken.

The relative addressing feature permits the user of CLUB to reference the addresses indicated on his program listing when debugging relocatable programs. This feature completely eliminates the problems of address arithmetic normally associated with debugging relocatable programs. The relative addressing feature is implemented by adding a value to the addresses referenced by the user of CLUB. The value is called a Bias.

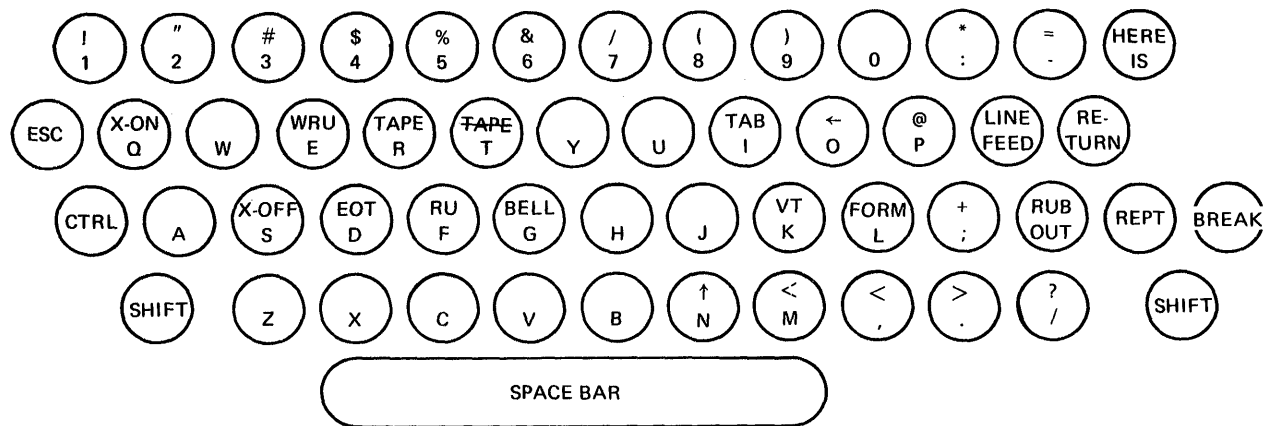


Figure 7-24. Teletype Keyboard Layout

7.11.2 Description of Operations

There are four classes of directives:

1. Bias Definition
2. Cell Examination and Modification
3. Program Control
4. Utilities

The directives for each class are defined by showing the directive (Teletype key) in a circle and the definition to the right of the directive, followed by a description of its function and an example. The underline is used in the examples to differentiate between the user's input and CLUB's output response. The user's input is underlined. A Carriage Return, represented by CR, must terminate every user command. If a # is typed by the user at any time during the input in a command, that line is ignored and a new command should be input.

7.11.3 Bias Definition

I BIAS

The relative addressing feature permits the user to reference the address indicated by his program listing when debugging relocatable programs. In order to make use of the relative addressing feature, the first directive to be used when debugging a relocatable program should be the "I" directive. The "I" directive sets the relative address Bias. If the "I" directive is preceded by a hexadecimal input, the relative address Bias is set to the value of that input. If nothing precedes the "I", the Bias is cleared (set to zero).

Example: 300I CR

The user normally sets the Bias value to the first location within a program. The Bias value is cleared whenever CLUB is executed at its starting location. By never specifying a Bias with the "I" directive, the relative addressing feature can be ignored.

Once a Bias is entered, all addresses equal to or larger than the Bias value, are shown relative to that Bias and are indicated clearly as such by an appended "R". While a Bias is entered, no cell having an address less than the Bias value can be directly opened by using the "b" (space bar) directive. If, however, a cell below the current Bias is to be shown, its address is shown absolute with no appended "R". This could happen, for example, if the user's program branched mistakenly to any point below its origin. Upon returning to CLUB with an Illegal instruction, the address is clearly distinguished as absolute and outside the beginning of his program.

7.11.4 Cell Examination and Modification

The directives described in this section provide memory cell examination and modification. These directives are generally preceded by some hexadecimal input. Hexadecimal inputs are accepted until a directive is received; the last four hexadecimal characters are then used as the address or data. The input of leading zeros is not necessary.

Ⓟ OPEN CELL

The space bar is the cell examination directive. Typed after an address, it causes that cell's address, relative to the Bias, and the content of that cell to be printed. If no address was specified prior to typing the space bar, the address of cell zero, as displaced by a previous Bias entry and its contents are printed. For example, if the user types 1FEⓅ, CLUB outputs the address 01FE and the content of that cell XXXX.

Example: 1FEⓅ CR
01FE XXXX

However, if a previous Bias entry of 100 had been made, in order to open that same cell the user would type:

FEⓅ CR
00FER XXXX

The trailing "R" on addresses indicates that the address is relative to the current value of the Bias.

RUB OUT CLOSE THE OPEN CELL

This directive causes CLUB to close the open cell and output a carriage return and line feed. This directive is not available in OS CLUB.

/ RESHOW CONTENTS MINUS THE BIAS

The slash "/" directive causes CLUB to show the contents of the open cell relative to the Bias. This is particularly useful when debugging a Relocatable program. If no Bias is entered, the contents are reshown absolute. Note that this directive does not change the actual contents of the open cell in memory.

Example: If the bias is set to 2500:

FEⓅ CR
00FER 3A64
/CR
1564R

LINE FEED OPEN NEXT SEQUENTIAL CELL

To open the next cell in sequence, the user types a "LINE FEED", followed by a carriage return. The current open cell is closed, and the address and content of the next sequential cell are printed.

Example: 1FEⓅ CR
01FE XXXX
LINE FEED CR
0200 NNNN



OPEN PRECEDING CELL

To open the previous cell instead of the next cell in sequence as in "LINE FEED" above, the user types a singular carriage return. The current open cell is closed and the address and content of the previous cell are printed.

Example: $\frac{1FE\cancel{\text{ }} \text{ (CR)}}{01FE \quad \text{XXXX}}$
 $\frac{\text{ (CR)}}{01FC \quad \text{YYYY}}$

(T) TRANSFER TO CONTENTS

This directive causes the content of the current open cell to become the address of the new open cell. The address and content of the new open cell are printed.

Example: $\frac{1FE\cancel{\text{ }} \text{ (CR)}}{01FE \quad 03E0}$
 $\frac{T \text{ (CR)}}{03E0 \quad \text{XXXX}}$

Since all addresses are assumed relative to the Bias and the address contained in the current open cell is an absolute address, the Bias is subtracted to maintain correct relative addressing.

Example: If the Bias is set to 100, the previous example would produce the following results:

$\frac{FE\cancel{\text{ }} \text{ (CR)}}{00FER \quad 03E0}$
 $\frac{T \text{ (CR)}}{02E0R \quad \text{XXXX}}$

(+) ADD

The plus directive causes the hexadecimal characters entered just prior to the plus to be added to the address of the current open cell, this sum becomes the address of the new open cell. The address and content of the new open cell are printed.

Example: $\frac{1FE\cancel{\text{ }} \text{ (CR)}}{01FE \quad 03E0}$
 $\frac{202+ \text{ (CR)}}{0400 \quad \text{XXXX}}$

(-) SUBTRACT

The minus directive causes the hexadecimal characters entered just prior to the minus to be subtracted from the address of the current open cell, this difference becomes the address of the new open cell. The address and content of the new open cell are printed.

Example: $\frac{1FE\cancel{\text{ }} \text{ (CR)}}{01FE \quad 03E0}$
 $\frac{1C- \text{ (CR)}}{01E2 \quad \text{XXXX}}$

(R) REGISTER SELECTION

The status of the 16 General Registers at the time of entry to CLUB, or upon encountering a breakpoint, is saved in a reserved area within the CLUB program. The registers are restored from this area when the "G" or "W" directive is executed. This area may be accessed for examination or modification by the "R" directive. The hexadecimal character typed just prior to the "R" directive selects 1 of the 16, saved register. This cell becomes the current open cell and may be operated on as such.

```
Example: 9R (CR)
         REG9 XXXX
         LINE FEED (CR)
         REGA XXXX
         4+ (CR)
         REGE XXXX
         (CR)
         REGD XXXX
         108. (CR)
         REGD 0108
```

The status of the 32-bit PSW register is saved in the fullword location at SAVOLD, as shown on the Hex Debug program listing. This location should be referenced to examine or change the PSW register following a breakpoint, or prior to giving a GO command.

(.) MODIFY CONTENTS ABSOLUTE

The period is the directive to modify the open cell. The content of the current open cell is replaced by the four hexadecimal characters entered just prior to the period. The input of leading zeros is not necessary, although acceptable as in the following example. If no hexadecimal entry is made, the cell is zeroed.

```
Example: 1FE (CR)
         01FE 03F0
         045C. (CR)
         01FE 045C
         . (CR)
         01FE 0000
```

(:) MODIFY CONTENTS RELATIVE

The colon causes the content of the current open cell to be replaced by the sum of the Bias and the four hexadecimal characters entered just prior to the colon. This permits the user to easily insert relative addresses without the necessity of address arithmetic when making modifications to a program.

```
Example: 100I (CR)
         1AB (CR)
         01ABR 9A02
         544: (CR)
         01ABR 0644
```

7.11.5 Program Control

The program control directives "X, Y, Z, K, G, and W" permit the user to direct the operation of his program through the use of CLUB. At a point within the user's program, it is frequently desirable to return control to CLUB. This is accomplished by inserting a breakpoint at the address at which the transfer of control is to take place.

(X) INSERT BREAKPOINT

The "X" directive causes a breakpoint to be inserted at the address specified by the preceding hexadecimal input. If no address is specified, the breakpoint is inserted in the current open cell. All breakpoint addressing is relative to the current value of the Bias.

Up to eight breakpoints at a time may be resident in the user's program. If a breakpoint insertion is requested and eight breakpoints have already been inserted, the message EXCESS BREAKPOINT is printed and the address of the current open cell or the specified cell and its content is printed.

When the breakpoint is inserted, the address at which the breakpoint is inserted and its new content are printed, and that address becomes the current open cell.

Example: If the breakpoint is inserted.

```

1 FEØ (CR)
01 FE 03E0
X (CR)
01 FE E1 F0

```

If the breakpoint is rejected:

```

1 FEØ (CR)
01 FE 03E0
X (CR)
EXCESS BREAKPOINT
01 FE 03E0

```

If an address is specified:

```

1 FEØ (CR)
01 FE 03E0
25AX (CR)
025A E1 F0

```

The E1 F0 is a SVC instruction that is inserted into the user's program by the "X" directive using OS Hex Debug. When this instruction is detected by the Processor, it causes control to be transferred to the OS. The OS returns control to OS Hex Debug so that OS Hex Debug can interpret the breakpoint. In the 03-013 version the inserted breakpoint value is the illegal instruction X'F000'.

When the "X" directive is specified, the open cell address is placed as the first half-word entry in a Breakpoint Table. The next halfword(s) to be entered is the instruction itself: one halfword for an RR; two halfwords for an RX, or RS instruction. The last entry is a Branch to the next executable instruction in the user's instructions. When a breakpoint occurs, the address where the breakpoint occurred is compared to all breakpoint addresses in the Breakpoint Table. If it is not found, the message ILLEGAL INSTRUCTION and the address and contents of the cell are printed. If the address is found, the message BREAKPOINT and the breakpoint address and its content are then printed. The Bias is not cleared and the address printed is relative to the Bias. The copying of the instruction and the inserting of a Branch instruction into the Breakpoint Table for an "X" directive are required for executing the breakpoint with the GO or GO WAIT directives. Caution is required in declaring a breakpoint so that the address specified in the "X" directive is the starting address of an instruction and not the address portion of an RX or RS instruction.

(Z) ZAP BREAKPOINT

The "Z" directive causes the removal of a single breakpoint. If the "Z" directive is preceded by a hexadecimal input, the breakpoint at the address specified by that input is removed. If no address was specified, the breakpoint is removed from the current open cell. All addressing is relative to the current value of the Bias. The breakpoint is removed, the address is restored, and the contents of the cell are printed. If no breakpoint is found at the address indicated, the message NO BREAKPOINT and the address and content of the cell are printed.

```

Example: 01 FEØ (CR)
01 FE E1 F0
Z (CR)
01 FE 03E0
3E0Z (CR)
NO BREAKPOINT
03E0 XXXX

```

(K) KILL ALL BREAKPOINTS

The "K" directive removes all resident breakpoints from the user's program. Upon completion of this directive, the message CLUB is printed. If there is an open cell, that cell is closed.

Example: K CR
CLUB

(Y) SHOW BREAKPOINT LOCATIONS

The "Y" directive causes CLUB to print out a list of the locations of all breakpoints. The addresses are always shown absolute, independent of the Bias values used when the breakpoints are defined. If CLUB has no record of any breakpoint entries, none are shown. The message CLUB is printed upon completion of the "Y" directive and the current open cell, if any, is closed.

Example: Y CR
010E
012B
01FC
0464
0266
20BC
2300
23B8
CLUB

(G) GO

The "G" directive causes CLUB to restore the 16 General Registers and transfer Processor control to the user's program. If the "G" directive is preceded by a hexadecimal input, execution begins at the address specified by that input. If no address is specified, execution begins at the current open cell. All addressing is relative to the current Bias. A carriage return and line feed is output in response to this directive to indicate that transfer of control is about to take place.

Before transferring control to the open address of the "G" directive, this address is compared to the breakpoint addresses found in the Breakpoint Table. If not found, a normal transfer occurs. If this address is indeed a breakpoint, then CLUB executes the Breakpointed instruction, and returns control at the next sequential user instruction. The breakpoint remains in effect.

Example: 01FE C820
G CR
Control transfers to 01FE

Example: 01FE C820
200G CR
Control transfers to 0200

See the NOTE under the "W" directive description for special handling of GO to cell 0000R.

In the OS CLUB, the execution of any Illegal instructions causes the operating system to type an Illegal instruction message and causes control to return to the OS. In the 03-013 version, the message ILLEGAL INSTRUCTION is typed, and CLUB retains control.

Ⓜ GO WAIT

The "W" directive causes CLUB to restore the 16 General Registers and place the Processor in a Wait state at the address specified by the hexadecimal characters input, just prior to the "W". If no address is specified, the Processor is halted at the address of the current open cell. That is, the Processor halts with the PSW Location Counter containing either the address specified or the open cell address. All addressing is relative to the current Bias. A carriage return and line feed is output in response to this directive to indicate that transfer of control is about to take place. Execution of breakpoints is as described in the "G" directive.

Example: 01 FE 03E0
W Ⓜ
Processor halts with 01 FE in PSW Location Counter.

Example: 01 FE 03E0
100W Ⓜ
Processor halts with 0100 in PSW Location Counter.

Example: With a bias of 100 entered:

00FER 03E0
W Ⓜ
Processor halts with 01 FE in PSW Location Counter.

NOTE

There is one exception for both the "G" and "W" directives. Processor control cannot be transferred to cell 0000R by typing "W", or "G". Cell 0000R must first be made the open cell. Once a Bias is entered, cell 0000R can be opened by typing the space bar. Then type the "G" or "W".

7.11.6 Utilities

The Utility directives provide a means of searching or capturing a user's program. They consist of the following directives. "S, P, Q, O, N". At the completion of each of these directives, the message CLUB is typed and any open cell is closed.

Certain directives are required for setting up and controlling the operation of CLUB utilities. The "L, H, V, M" directives must be entered for the Search "S" directive. The "L" and "H" directives must be entered for Print "P", Disassembly "N", and the Dump Core on Tape "O" and "Q" directives. A carriage return and line feed is output in response to each of these directives. The "Break Key" stops all Utility directive operations in the 03-013 versions. In the 03-032 version, the "*" and "!" directives allow communication between CLUB and the OS.

Ⓛ LOW LIMIT

The "L" directive defines the low limit address for Utility operations such as the Search, Print, Disassembly, and the "O" and "Q" Tape Dump directives. The hexadecimal characters entered just prior to the "L" become the low limit. The low limit address is made relative to the Bias at the time of execution of the Utility directives. A carriage return and line feed is output in response to the "L" directive.

Example: 1FEL Ⓜ

Ⓜ HIGH LIMIT

The "H" directive defines the high limit address for Utility operations. The hexadecimal characters entered just prior to the "H" become the high limit. The high limit address is made relative to the Bias at the time of execution of the Utility operations. A carriage return and line feed is output in response to the "H" directive.

Example: 260H Ⓜ

Ⓟ VALUE

The "V" directive defines the value to be used by the Search directive. The hexadecimal characters entered just prior to the "V" character become the search value. A carriage return and line feed is output in response to the "V" directive.

Example: 300V (CR)

Ⓜ MASK

The "M" directive defines the mask to be used by the Search directive. The hexadecimal characters entered just prior to the "M" character become the search mask. A carriage return and line feed is output in response to the "M" directive.

Example: F00M (CR)

Ⓢ SEARCH

The "S" directive causes CLUB to make a search and print the address and content of every cell, in the section of memory bounded by the low and high limits, whose masked content is equal to the masked value. The address printed when relative to a Bias are flagged with a trailing "R". The message CLUB is printed upon completion of the search.

Example: The search for a 3 in the second hexadecimal digit position is accomplished with the following procedure:

<u>1FEL</u>	(CR)	low limit
<u>260H</u>	(CR)	high limit
<u>300V</u>	(CR)	value
<u>F00M</u>	(CR)	mask
<u>S</u>	(CR)	do search
01FE	03E0	address and content
025A	43A0	address and content
CLUB		

The "S" directive is utilized most commonly as a debugging technique for locating instructions branching to a particular address or for locating those instructions changing the content of a particular cell. To accomplish this, set the address of the cell either being branched to or being changed into value, set FFFF into Mask, set the low and high limits desired and type "S".

Ⓟ PRINT

The "P" directive causes CLUB to print the content of every cell in the section of memory bounded by the low and high limits. The addresses printed when relative to a Bias are flagged as such with a trailing R. CLUB masks off the least significant digit of the low limit for format purposes. The message CLUB is printed upon completion of the directive. The printing format is shown below.

Example: 1234L (CR)
124AH (CR)
P (CR)
1230 XXXX XXXX XXXX XXXX XXXX XXXX XXXX
1240 XXXX XXXX XXXX XXXX XXXX XXXX XXXX
CLUB

Where 1230 is the address of the first halfword in the line, and XXXX is the content of the successive cells. With version 03-013, the printing is output to the device previously selected in the Device Definition Table as LISTDV at location X'7E' which should normally contain one of the following:

X'7E'	0298	Teletype Page Printer
	139A	High Speed Paper Tape Punch
	6280	Line Printer

With 03-032, printing is to Logical Unit 3, which must be assigned in the OS.

Ⓞ OUTPUT TAPE IN STANDARD LOADER FORMAT

The "O" directive causes CLUB to output on tape the contents of every cell in the section of memory bounded by the low and high limits relative to the current Bias. The tape is absolute and in standard loader format. If the punch device is the Teletype, core is dumped in the M08 Zoned Format. If the punch device is not the Teletype, the tape is punched in the M16 format. With 03-013, it is output to the punch device selected in the Device Definition Table as the Binary Output Device (BOUTDV) at location X'7A' which should contain one of the following:

X'7A'	0298	Teletype Punch Device
	139A	High Speed Paper Tape Punch

Upon receipt of the "O" directive, the 03-013 CLUB puts the Processor in the Wait state. The punch should then be turned ON and EXECUTE depressed. The tape is then punched with leader and trailer and the Processor returned to the Wait state. The punch should then be turned OFF and EXECUTE depressed. The High Speed Paper Tape Punch can be turned OFF by depressing INITIALIZE before depressing EXECUTE. The message CLUB is then printed out on the Teletype. With 03-032, punching is to Logical Unit 2, which must be assigned in the OS. The resulting tape can be loaded with the REL, General, BOSS Resident, or OS Library Loader.

Example: If the user wished to dump core memory from cell address 1000 through cell address 15AC inclusive:

<u>1000L</u> <u>CR</u>	low limit
<u>15ACH</u> <u>CR</u>	high limit
<u>O</u> <u>CR</u>	Processor then halts.
	User should turn punch ON and depress EXECUTE.
	Tape is punched, Processor then halts.
	User should turn punch OFF (depress INITIALIZE) and depress EXECUTE.

CLUB

Ⓞ DUMP CORE IN EIGHT-BIT FORMAT

The "Q" directive causes CLUB to output to tape the content of every cell in the section of memory defined by the low and high limits. With 03-013 this tape is punched in eight-bit format and output to the device selected in the Device Definition Table as the Binary Output Device (BOUTDV) at location X'7A' which should contain one of the following:

X'7A'	0298	Teletype Punch Device
	139A	High Speed Paper Tape Punch

NOTE

Because of the eight-bit format and the fact that some eight-bit configurations can indicate certain control characters such as WRU, XON, XOFF, etc., to the Teletype, this directive cannot be used with some ASR-33 Teletypes.

Upon receipt of the "Q" character, the 03-013 CLUB puts the Processor in a Wait state. The punch should then be turned ON and EXECUTE depressed. The tape is then punched with leader and trailer and the Processor returned to the Wait state. The punch should then be turned OFF and EXECUTE depressed. The message CLUB is then printed out on the Teletype. The resulting binary tape can be loaded by adjusting the Model 3 50 Sequence loader low and high addresses found at locations '52' and '5A' respectively.

With 03-032, punching is to Logical Unit 2 using a write binary operation to produce a binary record.

NOTE

With OS CLUB the binary tape produced contains a leading "F0" character which should be ignored when loading by positioning the tape past this character on the input device being used.

Example: 100L (CR) low limit
15ACH (CR) high limit
Q (CR) Processor then halts.
 User should depress EXECUTE.
 Tape is punched, Processor then halts.
 User should turn punch OFF and depress EXECUTE.

CLUB

NOTE

CLUB forces halfword alignment on the low and high limits relative to the Bias. CLUB dumps core from an even halfword address obtained from the low limit through an even halfword address obtained from the high limit, both relative to the Bias.

(N) DISASSEMBLY

The "N" directive causes CLUB to output a disassembly of the section of memory bounded by the low and high limits relative to the current Bias. With 03-013, this printout will be on the device selected as LISTDV in the Device Definition Table at location X'7E' just as does the Print "P" directive. Refer back to "P" directive on device selection procedures. With 03-032, printing is to Logical Unit 3, which must be assigned in the OS. Upon completion of the "N" directive, the message CLUB is printed on the Teletype. A 50 Sequence is disassembled as follows:

Example: 50L (CR)
7EH (CR)
N (CR)

0050	C820	LHI	2,0080
	0080		
0054	C830	LHI	3,0001
	0001		
0058	C840	LHI	4,00CF
	00CF		
005C	D3A0	LB	A,0078
	0078		
0060	DEA0	OC	A,0079
	0079		
0064	9DAE	SSR	A, E
0066	08EE	LHR	E, E
0068	4230	BTC	3,0064
	0064		
006C	DBA2	RD	A,0000(2)
	0000		
0070	C120	BXLE	2,0064

	0064		
0074	4300	BFC	0,0080
	0080		
0078	1399		
007A	0392	BFCR	9,2
007C	04A0	NHR	A,0
007E	6280		
CLUB			

NOTE

The Disassembler cannot distinguish data which appears as Legal instructions as contained in X'7A' and X'7C'. However, recognizable data, such as in X'78' produces blanks. Note also, that extended branches are disassembled with mnemonics BTC, BTCR, BFC, or BFCR.

(BREAK) STOP

In the 03-013 version only, the "BREAK KEY" directive causes CLUB to discontinue whatever output operation it is currently engaged in and print the message CLUB. Its intended purpose is to allow the user to stop the operation of the Print "P", Search "S", Disassembly "N", and the Dump Core on Tape "O" and "Q" directives at any time deemed necessary and retain control in CLUB. If the BREAK KEY is hit during a dump tape directive, the Processor halts as with a normal completion of the "O" and "Q" directive in order to allow the user to turn the punch OFF and then depress EXECUTE. The message CLUB is then printed.

(*) END

In the 03-032 version only, the END directive causes CLUB to restore a special pointer in the OS, and return control to the OS via an End of Job supervisor call. This directive is appropriate whenever it is desired to return control to the OS.

(!) PAUSE

In the 03-032 version only, the PAUSE directive enables the CLUB user to return to the OS without disturbing breakpoints, registers, limits, etc. To return to CLUB, the user uses the OS CONTINUE directive.

7.11.7 Operating Procedures

The program tapes available for the two versions of CLUB are:

1. HEX DEBUG 03-013M16
2. OS HEX DEBUG 03-032M16

All tapes are relocatable, and have the starting address at the ORIGIN, or the first location of the program. Note that absolute tapes of CLUB can be obtained by using the output operations in CLUB to punch the CLUB program itself.

Loading Procedures

Since all CLUB programs are relocatable in non-zoned Loader format, they can be loaded with either the General Loader, Program Number 06-025, or Relocating Loader, Program Number 06-024. OS Debug, Program Number 03-032, can be loaded using the LOAD directive in the OS or via the OS Library Loader, Program Number 03-030.

Detailed loading procedures for Hex Debug, Program Number 03-013, are:

1. Load the General or Relocating Loader into memory. Refer to Loader Descriptions Section 7.9 for a detailed explanation of these standard loaders.

2. Place the CLUB program tape in the tape reader.
3. If the user wishes CLUB to be loaded into location X'80', address the Processor at the Loader's starting location and depress EXECUTE. CLUB is loaded at X'80'.
4. To load CLUB elsewhere, enter the Bias to be used by the Loader, the address at which CLUB is to be loaded, into the Bias Definition location within the Loader. Then start the Loader at its Bias Define Restart Address, which loads CLUB at the specified location.

Starting Location

The starting location for CLUB is the first instruction in the program. That is, if relocatable CLUB is loaded without changing the Bias in the Reloading or General Loader, it is loaded at X'80' and execution of CLUB begins at X'80'. However, if the Loader's Bias is set to X'1000', CLUB is loaded at X'1000' and execution would begin at that location.

CLUB can be restarted at any time at that same address. However, in order to restart CLUB without destroying its record of breakpoint entries, CLUB must be re-entered through its Illegal Instruction Trap. A common method for accomplishing this is to have the Processor execute a location above the top of memory, execute any cell containing 0000, or execute any other Illegal instruction, for stand-alone CLUB. For OS CLUB, execute any cell containing X'E1F0'.

Program Size

Both versions of CLUB require about 3KB of memory. If the disassembly or output features are not required by the CLUB user, the coding for these functions, which is at the top of CLUB, may be overlaid to provide more user space. The user should examine the CLUB listing for the starting address of the output routine (symbolic name OUTPUT) and the disassembly routine (symbolic name DISASM). If one or both of these routines are overlaid, the directives that pertain to these routines should not be used. A summary of CLUB Directives is provided in Table 7-36.

Device Selection

In version 03-013, the devices used are:

1. LISTDV - The Print "P" or Disassembly "N" operations cause a printout on the List Device, which is selected in halfword location X'7E' (LISTDV) in the Processor's Device Definition Table. Byte X'7E' (LISTDV) should contain the device number and byte X'7F' (LISTDV+1) should contain the command byte normally used with print device. That is, halfword location X'7E' should contain one of the following:

0298	Teletype Punch Device
139A	High Speed Paper Tape Punch Device
6280	Line Printer

2. BOUTDV - The Dump Core on Tape "O" or "Q" directives cause a tape to be punched on the punch device, which is selected in halfword location X'7A' (BOUTDV) in the Processor's Device Definition Table. Byte X'7A' (BOUTDV) contains the device number and X'7B' (BOUTDV+1) contains the command byte, normally used with a punch device. That is, halfword location X'7A', should contain one of the following:

0298	Teletype Punch Device
139A	High Speed Paper Tape Punch Device

Refer to individual device description manuals for a detailed explanation of command bytes.

In version 03-032, the logical devices used are:

Logical Unit 2	Same as BOUTDV, Binary Output Device
Logical Unit 3	Same as LISTDV, for ASCII Output
Logical Unit 5	Used as keyboard printer device for all common inputs and message outputs.

These logical units must be assigned using the ASSIGN command in the OS prior to starting the execution of OS Hex Debug.

TABLE 7-36.
SUMMARY OF CLUB DIRECTIVES

1. <u>BIAS DEFINITION</u>		3. <u>PROGRAM CONTROL</u>	
I	BIAS	X	INSERT BREAK-POINT
2. <u>CELL EXAMINATION AND MODIFICATION</u>		Z	ZAP A BREAK-POINT
Ø	OPEN CELL	K	KILL ALL BREAK-POINTS
RUB OUT	CLOSE OPEN CELL	Y	SHOW BREAKPOINT LOCATIONS
/	RESHOW CON- TENTS MINUS THE BIAS	G	GO EXECUTE
LINE FEED	OPEN NEXT SEQUENTIAL CELL	W	GO WAIT
RETURN	OPEN PRE- CEDING CELL	4. <u>UTILITIES</u>	
T	TRANSFER TO CONTENTS	L	LOW LIMIT
+	ADD	H	HIGH LIMIT
-	SUBTRACT	V	VALUE
R	REGISTER SELECTION	M	MASK
.	MODIFY CON- TENTS ABSOLUTE	S	SEARCH
:	MODIFY CON- TENTS RELATIVE	P	PRINT
		O	OUTPUT TAPE (STANDARD LOADER FORMAT)
		Q	OUTPUT TAPE (EIGHT BIT FORMAT)
		N	DISASSEMBLE
		BREAK	STOP
		#	DELETE LINE
		*	END
		!	PAUSE

7.12 BASIC OPERATING SYSTEM

7.12.1 General Description

The Basic Operating System (BOSS), the smallest member of the INTERDATA family of operating systems, enables the user to efficiently develop and maintain programs on small hardware configurations. The software package provides physical file handling on many bulk storage devices such as disc, tape or cassette. BOSS allows programs to be device-independent and provides console operator support and general programming service. A System Generation capability is included to provide the flexibility necessary to meet the needs of a variety of individual systems requirements. Figure 7-25 provides a core map.

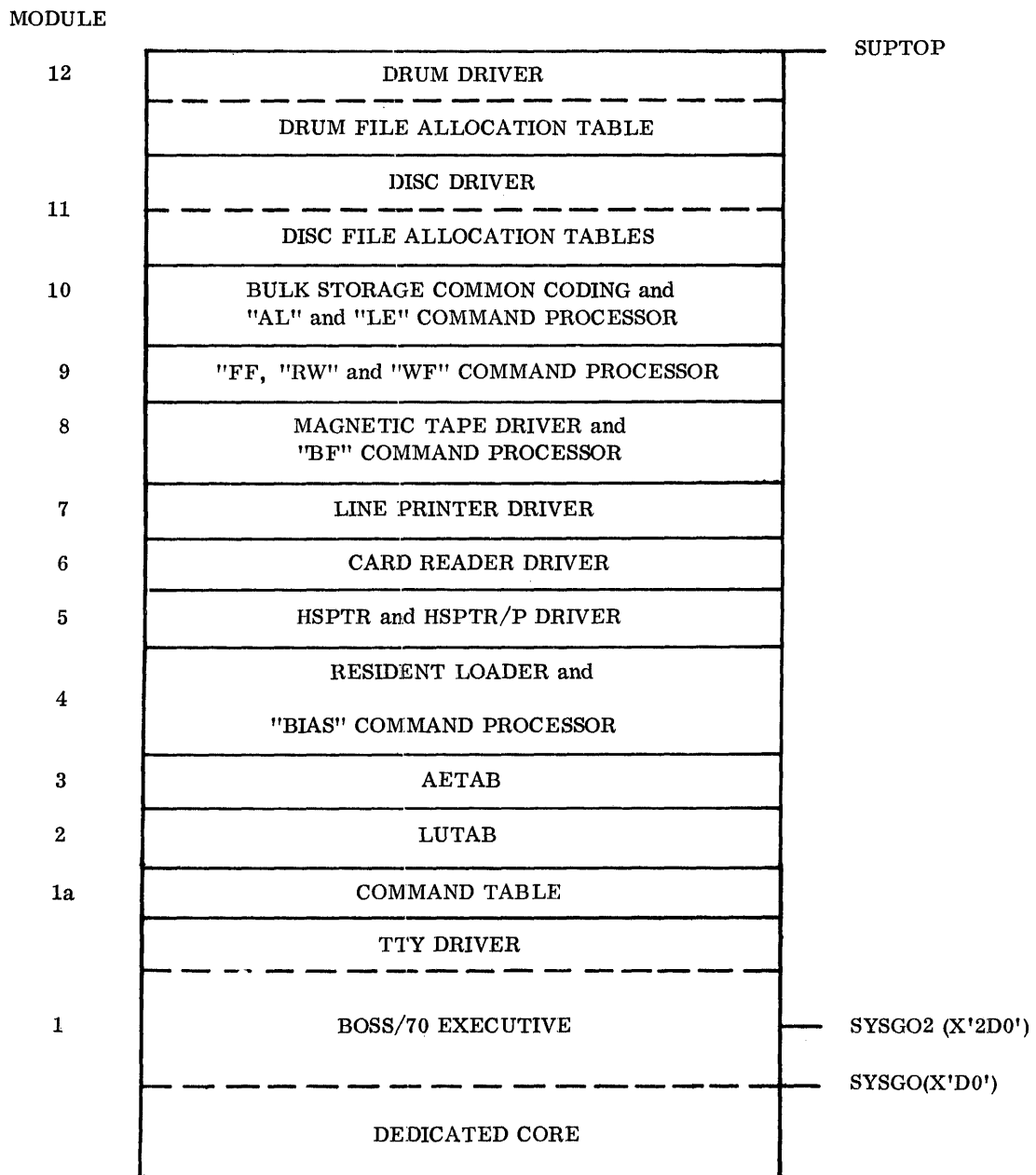


Figure 7-25. BOSS Core Map

7.12.2 Operational Characteristics

- . BOSS is extremely compact, typically requiring less than 4K bytes of main memory.

- . BOSS supports both random and sequential access devices.

- . Disc files may be allocated by the operator, providing both efficiency of operation and ease of use. Files are referenced by logical unit numbers.

- . Input/Output operations are device-independent, allowing hardware swapping without having to alter the existing software.

- . Operator commands can be automatically read from any input device, relieving the operator of lengthy and error-prone console operations, at the same time increasing system utilization.

- . Sets of operator commands may be catalogued on disc for subsequent retrieval by the BOSS supervisor. Thus a FORTRAN source program update, compile, link, load and run sequence may be initiated with a single console command.

- . Up to 256 logical units (files) may be accommodated. A physical disc unit may be partitioned into up to 128 logical units.

- . Commands to BOSS may be accepted from any input device including catalogued Disc Files. This allows the user to batch jobs with job control statements. With the batch capability, FORTRAN compilations and executions are possible using the job control statements.

7.12.3 Programmable Commands

All BOSS programmable commands are performed by issuing SVC instructions. The SVC instructions which are in BOSS are:

1. Read-Write-Control SVC (SVC1) for both I/O wait-proceed, and sequential and Random access.

2. Supervisor Service (SVC) (SVC2).

3. End of Job (SVC3).

7.12.4 Operator Commands

Allocate	Allocate initial number of cylinders and logical record length for a file.
List Extents	List all files and logical units.
Write File Mark	Write File Mark Record.
Skip Forward To File Mark	Skip Forward to File Mark (Magnetic Tape).
Backspace To File Mark	Backspace to File Mark (Magnetic Tape).
Rewind	Rewind File.
Save	Place a specified core area onto a file (LU).
Transfer Control	Transfer operator command input to LU.
Start	Start a program.
Load	Load a program from LU.
Assign	Assign a logical unit to physical device.
Bias	Specify a bias address.
Halt	Halt a program.
Continue	Continue a program.
Open	Open a location.
Replace	Replace value at location.
+	Open at location +2.
=	Open at location — 2. —2.

7.12.5 System Configuration

The minimum BOSS configuration should contain the following items:

Model 74, 70, 80, 85, 50 or 55 Processor with 16KB Model 74, 70, 80, 85, 50 or 55 Processor with 16KB.

Additional devices supported under BOSS include:

- Teletypewriter
- Line Printer — Low or High Speed
- Card Reader
- Moving Head Disc Drives — Up to 4
- High Speed Paper Tape Reader/Punch
- Magnetic Tape
- Drum
- INTERTAPE Cassette

7.13 DISC OPERATING SYSTEM

7.13.1 Introduction

This section is intended only as a summary of the major features of the INTERDATA Disc Operating System. This operating system is fully described in DOS Reference Manual, Publication Number 29-293. The Disc Operating System (DOS) enables the user to efficiently develop and maintain disc resident libraries of programs and data files. It provides for program segmentation, direct or sequential file access, file protection, blocking and de-blocking of logical records, overlapped input/output operations, and catalogued files of operator commands.

7.13.2 Features

Some of the outstanding features of DOS are:

1. All peripheral operations are fully overlapped with both processing and other peripheral operations.
2. Tasks need not be totally memory resident - tasks can be segmented and overlaid from disc.
3. Direct and sequential disc file handlers provide blocking and de-blocking of data for efficient processing of disc and magnetic tape files.
4. Disc throughput is optimized by buffering file accesses. DOS checks a main memory buffer for the presence of a logical record before making an access to the disc.
5. Disc storage is allocated and protected by DOS, thus limiting the effect of program errors on system performance.
6. Disc files may be allocated dynamically by DOS or absolutely by the operator providing both efficiency of operation and ease of use. Files are referenced by a mnemonic file name.
7. Input/Output operations are device independent allowing hardware swapping without having to alter the existing software.
8. Operator commands can be automatically read from any input device, relieving the operator of lengthy and error-prone console operations, at the same time increasing system utilization.
9. Sets of operator commands may be cataloged on disc for subsequent retrieval by the DOS supervisor. Thus a FORTRAN source program update, compile, link, load and run sequence may be initiated with a single console command.
10. The entire DOS program can be rapidly bootstrap loaded from disc.

7.13.3 Description

This section provides a description of the DOS features.

1. Concurrent I/O - I/O can be performed as I/O wait or I/O proceed.
2. Logical to Physical Device Independence - The operator may with minimum constraints transfer control from one device to another without modifying his program.
3. Logical Units - Up to 16 logical units (files) may be open at any one time.
4. Batch Operation - Commands to DOS may be accepted from any input device including cataloged Disc Files. This allows the user to batch jobs with jobs with job control statements.
5. Load and Go Operations - With the batch capability, FORTRAN compilations and executions are possible using the Job Control statements.
6. Named Files - The DOS allows the user to have up to 200 named files on the Disc.
7. Allocation - The DOS allows the user to initially allocate disc space but will automatically allocate further disc space up to maximum of 3 cylinders to satisfy user requirements. This automatic overflow allocation is invisible to the user.
8. File Manipulation - The DOS allows files to be named, examined and manipulated. The user may save a program in core by transferring it to a disc file, may copy a disc file to another disc file, or may append a disc file to another disc file. Additionally, a file may be deleted and all files may be listed with appropriate parameters.
9. File Attributes and Protection - Associated with each disc file is set of attributes which the user may set. The attributes include read and write protection plus data and file descriptors. The file protection will allow a user to save a file and protect against unauthorized access to that file.
10. Access Method - The DOS provides two access methods - sequential and direct access. Sequential mode is normally used to access records consecutively such as in compilations and assemblies whereas direct access mode allows accessing particular record by specifying a logical record number. Direct access is useful in updating master files on disc and in accessing large tables that are kept on disc.
11. Record Format - All disc files consist of a set of logical records. A physical record contains one or more logical records. The physical record size which is the same for all files is a system generation parameter which can be a multiple of the sector size up to 1K bytes. The number of logical records per physical record is calculated based upon the user specifying the logical record length, which may be different for each file. All accesses to the disc by the user are via logical records whereas internally all disc operations are via physical records. The logical record size may not exceed the physical record size. The DOS will not block logical records such that they cross physical record boundaries.
12. File Format - A file requires a 6-character name which is contained in a name directory. The file contains at least one cylinder (about 12KB) and may be as large as 200 cylinders. The file may be designated as direct or sequential and containing ASCII, binary or core image data.
13. Disc Format - The disc consists of up to 200 named files and a file directory. The file directory consists of an unused cylinder table and a table of named files with pointers and attributes.
14. Disc I/O - The user program performs all disc I/O via an SVC 1 instruction. Operations which can be performed are described elsewhere but basically the user specifies the operations Read/Write, Wait/Proceed, Random/Sequential, data type or a specific control operation (i. e. Rewind).

15. Disc Buffering - In order to insure a minimum of accesses, the user may specify the length of the logical records within a physical record as well as the access method. DOS will determine if a logical record is already in core rather than do an automatic access and if a write operation is requested, will place the logical record in the DOS buffer and output the buffer only when necessary. In a sequential file for write operations, a read is not performed.

16. Overlays - The user will be allowed to issue an SVC to fetch a named absolute overlay into core at location UTOP and transfer control to the overlay. The overlay may communicate to the main program via EXTRN/ENTRY but it may not itself issue an SVC fetch overlay, since it would overlay itself. When the overlay is done, control is returned to the main program. This ability is useful in running FORTRAN programs which are too large to fit in core but when divided into a root program and subroutines can fit if the root program is resident and the subroutines can overlay each other. The generation of the named overlays with the linkages to the root program is handled by a separate program external to DOS itself.

17. FORTRAN Programming with Direct Accessing - A FORTRAN program may access a specific logical record by calling a disc position subroutine prior to executing a Read or Write statement. FORTRAN READ or WRITE statements will then access the disc sequentially from the logical record specified.

18. Disc Pack Interchange - The DOS allows the interchange of Disc packs since all of the control information required by the DOS is resident on the Disc pack.

7.13.4 DOS Programmable Commands

All DOS programmable commands are performed by issuing SVC instructions. The DOS is upward compatible with all SVC instructions presently in the Basic Operating System. The SVC instructions in DOS are:

1. Read-Write-Control SVC (SVC 1) for both I/O wait-proceed, and sequential and Random access.

2. Supervisor Service SVC (SVC 2)

3. End of Job (SVC 3)

4. Execute Operator Command (SVC 4)

5. Fetch Overlay (SVC 5)

7.13.5 DOS Operator Commands

ALLOCATE	Allocate initial number of cylinders and logical record length for a file.
ACTIVATE	Activate (ASSIGN) a disc file at beginning or end of file.
PACK	Disc pack has been changed.
INITIALIZE	Initialize the Disc, specifying new pack or old pack.
CLOSE	Close a disc file.
ATTRIBUTE	Specify attribute for a disc file.
DELETE	Delete a disc file.
LIST UNITS PACK	List DOS logical/physical device assignments.
LIST	List all files, attributes on disc, and logical units.
WRITE FILE MARK	Write File Mark Record.
SKIP FORWARD TO FILE MARK	Skip Forward to File Mark (Magnetic Tape).
BACKSPACE TO FILE MARK	Backspace to File Mark (Magnetic Tape).
REWIND	Rewind file.
COPY	Copy from LU to LU (may be disc files).
TRANSFER CONTROL	Transfer operator command input to LU.
START	Start a program.
RUN	Load the program from Disc and begin execution.
LOAD	Load a program from LU.
LOAD DISC	Load a program from Disc.
ASSIGN	Assign a logical unit to physical device.
BIAS	Specify a bias address.
POSITION	Position a file to a subfile.
HALT	Halt a program.
CONTINUE	Continue a program.
OPEN	Open a location.
REPLACE	Replace value at location.
+	Open at location +2
-	Open at location -2

7.13.6 Configuration

The minimum DOS configuration should contain the following items:

The Model 70 or 80 Processor with 24KB
Teletypewriter
High Speed Paper Tape Reader Punch
Moving Head Disc (Diablo) and Controller

Additional devices supported under DOS include:

Cassette Tape
Line Printer - Low or High Speed
Card Reader
Additional Disc Drives - Up to 4
Magnetic Tape

7.14 REAL TIME OPERATING SYSTEM

7.14.1 Introduction

The use of digital computers in areas requiring rapid response to asynchronous events has provided the solution to many problems in industrial control, production monitoring, and data collection. These applications require a sophisticated, versatile, operating system incorporating the following features:

1. Input and output devices must be operated at their rated speeds.
2. The system must be multiprogrammed to assure prompt response to control system events.
3. Actions which occur at definite times, or within definite time periods must be scheduled according to a system clock.
4. The system must be able to quickly reallocate its hardware resources to meet an external demand for a high priority function.
5. Program development and testing should be simplified and speeded through the use of the background processing facility of the operating system.
6. Changes to, and additions of, application programs must be processed with minimum interference with on-line system functions.

The INTERDATA Real Time Operating System (RTOS) incorporates these design characteristics. This section is intended only as a summary of the major features of RTOS. A complete description of this system is contained in the RTOS Reference Manual, Publication Number 29-240.

7.14.2 Features and Characteristics

The Real Time Operating System makes maximum use of the advanced architecture of the INTERDATA Model 70 and 80 Processors. It is specifically designed to operate in on-line industrial control applications. Some of the outstanding features of the Real Time Operating System are:

1. Field proven performance. RTOS is currently operational in a wide variety of industrial applications. Its design philosophy and implementation have been thoroughly proven. RTOS is covered by the comprehensive INTERDATA warranty.
2. Up to 16 levels of priority. In addition, tasks can share time on the same priority level.
3. Multi-programming. Many application programs can operate concurrently with interleaving.
4. Dynamic main memory allocation. Main memory is allocated to non-resident tasks, at task execution time. All tasks are fully relocatable. The number of tasks that can be active at any time is limited only by the amount of memory available.

5. Segmentation facilities. Facilities are provided to permit large tasks to be easily segmented.
6. Device independence. The form of the user task call to an I/O device is insensitive to device type. Thus, changes in I/O device hardware can be made without affecting application software.
7. Background processing. Complete facilities are provided to permit assemblies, compilations, and program testing, in a background mode, while the system is performing its primary control function.
8. Memory protection. All tasks running under RTOS are executed under the surveillance of the INTERDATA comprehensive memory protection hardware. In addition, the RTOS executive provides protection of mass storage information.
9. Upward compatibility. Program written to run under the INTERDATA Basic Operating System (BOSS) will execute under RTOS with no modification required.
10. Distributed processing. A synchronous communication package is available for use with RTOS to support communication between INTERDATA Processors. This package supports multi-dropped, master-slave Processor configurations.
11. FORTTRAN Interface. RTOS provides a re-entrant FORTTRAN interface to facilitate efficient real-time FORTTRAN programming. In addition, a series of re-entrant subroutines are provided under RTOS to bring the FORTTRAN user some of the system capabilities that are usually only available to the assembly language programmer.

7.14.3 System Concepts

RTOS Tasks

The fundamental programming unit within RTOS is the task. A task may consist of a single program, or it may include a main program and a number of subprograms. Tasks may be permanently core resident, or they may be loaded from the system library as required. The total number of tasks allowed in core at any given time is a system generation parameter that is limited only by the amount of core available. Once in core, a task may exist in any one of four states. It may be:

- Active
- Ready
- Suspended
- Dormant

The active task is the one currently executing instructions, and only one task may be in this state at any given instant in time. Other tasks in core are in one of the other states. They may become active depending on various combinations of circumstances. Paragraph 7.14.4 gives detailed descriptions of task states and task management.

Scheduling

Each task within RTOS has a Task Control Block (TCB) that contains all the information required by RTOS for scheduling the task. Task states change dynamically, that is, they go from active to any of the other states, or they go from dormant or suspended to ready. Whenever the state of any task changes, or on periodic clock interrupts, the scheduler looks at the list of ready tasks and determines on the basis of priority level and time of last activation which task to make active. The priority of each task is a parameter contained within the TCB. It is assigned when the task is first established, and it may be changed by means of a special operator command. There are several ways in which a task may move into the ready state. These include:

- The passage of a preset time interval.
- The completion of an I/O transfer.
- A command from the operator.
- The availability of requested core.
- The completion of another task.
- The occurrence of an external interrupt.

Of these, the last three merit further explanation. In RTOS, any task may request that another task be put into the ready state. If the requested task is not in core, RTOS loads it from the system library, if there is sufficient core available. If there is not enough core available, the requesting task moves into the suspended state, waiting for core. As other tasks terminate, core becomes available, and eventually there is enough to load the requested task. At this time, both tasks may move into the ready state, or the requesting task may choose to continue in the suspended state waiting now for completion of the requested task.

The use of external interrupts to schedule tasks requires special coding techniques described in the Real Time Operating System (RTOS) Reference Manual, Publication Number B29-240. Briefly, however, a task may be controlled by up to eight hardware interrupts. The controlling interrupts may originate in the eight Line Interrupt Module or in any device capable of producing unsolicited interrupts. When a controlling interrupt occurs, RTOS notifies the task by making an entry in a queue within the task. If the task was dormant at the time of the interrupt, it is made ready. If the interrupt came from a device capable of transmitting data, one byte of data is passed to the task along with the notification of the occurrence of the interrupt. This mechanism has particular value in the area of data communications.

Core Utilization

Figure 7-26 shows a typical core map for an RTOS installation. RTOS itself occupies low main memory. Within this area are the permanently resident system modules and tables and a reserved area in which non-resident system modules are loaded as required from the system library. Part of the permanently resident RTOS area may optionally be used for user written supervisor programs. These programs may control special I/O devices, or they may be directly linked to the Eight Line Interrupt Module.

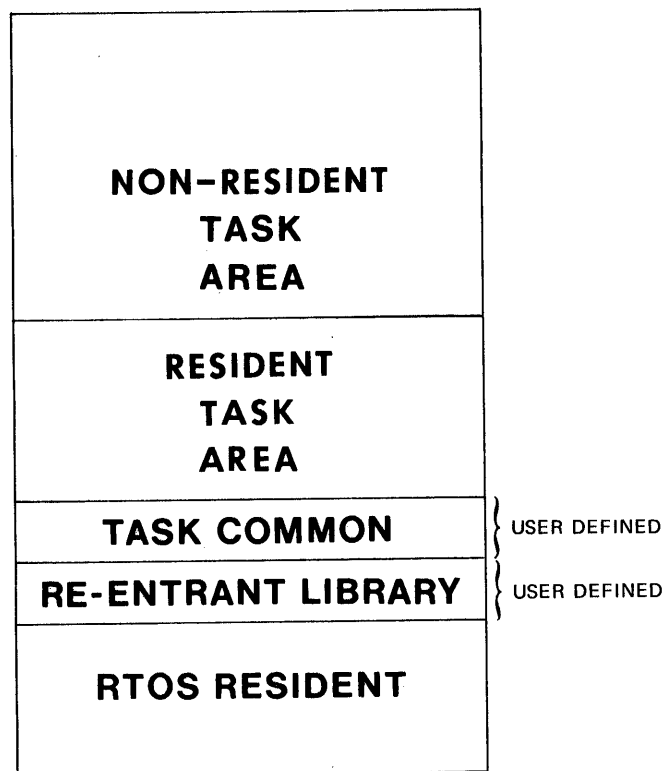


Figure 7-26. RTOS Main Memory

The re-entrant library is an optional feature of RTOS. It may be loaded at system generation time with frequently used re-entrant subroutines. Any task within the system may reference subroutines in the re-entrant library. The RTOS task loader establishes the necessary linkages.

The task common area is another optional feature of RTOS. It is labeled common and established at system generation time. This area provides a convenient means for tasks to set up a common data base.

The remaining resident and non-resident task areas of main memory are allocated in 1024 byte blocks. Each block must start at an address that is a multiple of 1024. The 1024 byte block size corresponds to the hardware memory protect block size. All tasks must be loaded beginning at the start of a block. Non-resident tasks are stored on bulk storage in relocatable form. Memory is allocated dynamically to non-resident tasks on an as needed basis. Thus, the number of non-resident tasks that can be concurrently active is limited only by the amount of memory available.

Security and Protection

RTOS uses the hardware memory protect feature of the Model 70 and Model 80 Processors to protect the integrity of all tasks within the system. When RTOS activates a task, it sets up a unique memory protect pattern that allows that task to modify core only within its own task block (and the task common block if required). The limits of all data transfers into core for a task are checked to make sure they do not violate the protect pattern. Input-output devices may also be protected. They may be assigned to specific tasks, and other tasks are not allowed to use them. In the area of operator commands, RTOS has two modes of operation. It may be unprotected, that is, responsive to all commands entered from the system Teletype. RTOS can also run in a protected mode, that is, not responsive to operator commands. A single operator command puts the system in the protect mode. Thus, if the operator puts the system in the protect mode and locks the console, unauthorized tampering with the system Teletype or console will have no effect on the system.

Mass Storage Allocation

RTOS supports one or more mass storage random access devices. It is organized as shown in Figure 7-27. The first section is reserved for a core image copy of RTOS itself. When the system is initialized, this copy of RTOS is loaded into core by a bootstrap loader. The next portion of storage contains the system library. This library contains relocatable copies of all tasks and task overlays within the system. The size of this library may be changed at any time, and the number of tasks stored in the library has no relation to the number of tasks allowed to be in core. Tasks may be added or deleted at any time. The remainder of the mass storage is divided into user files. Each file is treated somewhat as a separate I/O device, and as such, can be protected and reserved for a particular task or group of tasks. The size of each file is determined by an operator command. Files may be accessed randomly or sequentially.

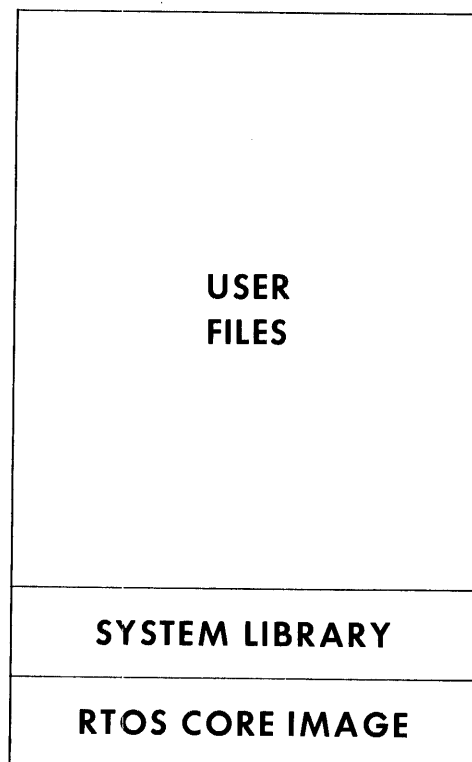


Figure 7-27. Mass Storage Allocation

7.14.4 System Organization

The basic entity within RTOS is the task. A task is a block of code treated as a unit by the scheduler. It may consist of a single program, a program and subprograms, or a program and its overlays. The task is the item scheduled by the scheduler, multiprogrammed with other tasks, protected or unprotected as a unit, and uniquely identified by a task identifier. Each task in the system has a Task Control Block (TCB) associated with it. The TCB becomes an integral part of the task when the task is created. It resides with the task in the system library. When a task is brought into core, a portion of the TCB containing information pertinent to overall system operation is loaded into the RTOS area where it can be protected at all times. The amount of core reserved for protected TCB is a system generation parameter that governs the total number of tasks that may be in core at any time. The remainder of the TCB is loaded into core at the starting location of the task.

Types of Tasks

Tasks are of two types: user tasks and supervisor tasks. User tasks execute in user mode (privileged instructions trapped) with core protected except for the task's own area and, if used by the task, task common. These tasks are always loaded at 1K boundaries so that they may be memory protected as a unit when other user tasks are active. Supervisor tasks execute in the supervisor mode. They are considered a part of RTOS but are scheduled and may share Processor time with other tasks of both types. See Figure 7-28.

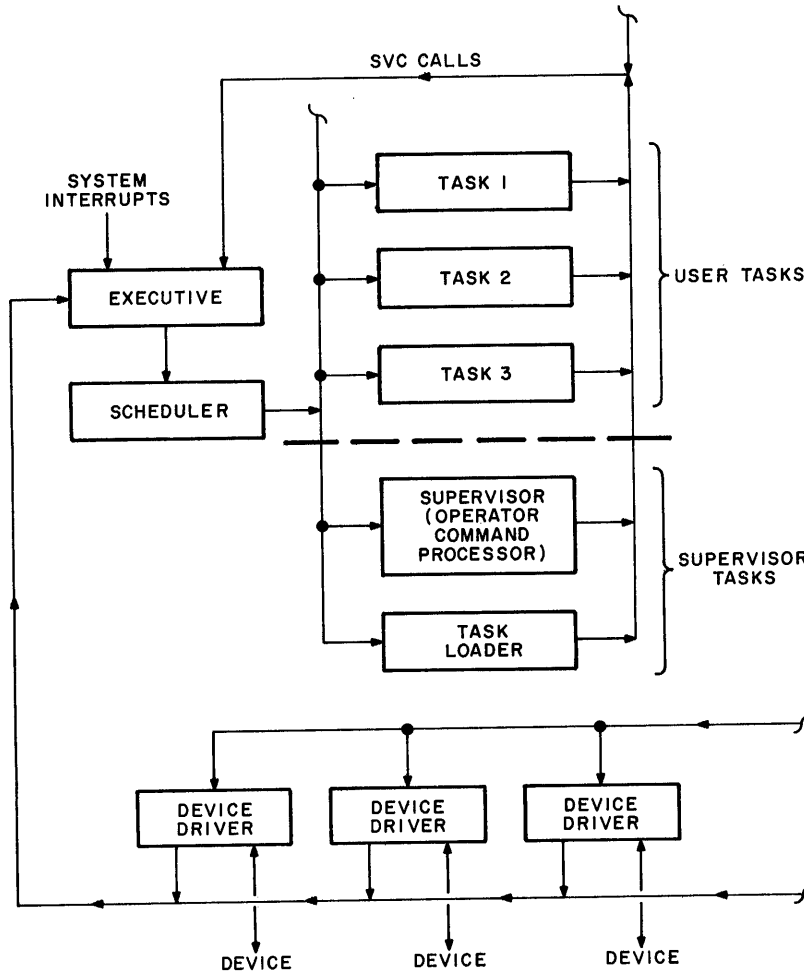


Figure 7-28. Interaction of RTOS Elements

Task Management

Tasks in core exist in one of four states:

- Active** Only one task is active at any one time. The active task is executing instructions and continues to be active until an interrupt occurs. A task may be made active only by the task scheduler.
- Ready** A task is ready whenever it cannot be active because a higher priority task is active. Whenever the scheduler gets control, it makes the highest priority ready task the active task.
- Suspended** A task may be suspended for any of four reasons. It may be in a task wait state, waiting for another task to terminate. It may be in an I/O wait state, waiting to access a busy device, or waiting for the completion of a data transfer. It may be in a console wait state caused by an operator command or a pause request. Or, it may be in a time wait state, waiting for a specified period of time to elapse, or waiting until a specified time of day.
- Dormant** A task is dormant when it has terminated itself or when it has been cancelled by the operator. Unless the task is designated as permanently core resident (by setting the appropriate flag in the TCB), all core allocated to the task, including its protected TCB, is released when the task becomes dormant.

A task is said to be busy whenever it is in the active, ready, or suspended state, and not busy when it is dormant. Whenever a task calls another task, the calling task is placed in the task wait state if the called task is busy. When the called task terminates, the calling task is put in the ready state. When it becomes the active task it reissues the call and may elect to suspend itself until the called task terminates, or it may elect to continue concurrently with it.

Tasks move from state to state depending on conditions that exist within the system at any given instant in time. The active task moves to the ready state whenever the scheduler finds a higher priority ready task. The active task is suspended when it requests any services of the operating system. It becomes dormant when it terminates itself or when a situation (illegal instruction, protect mode violation) arises in which the system terminates the task abnormally. A ready task is made active when it becomes the highest priority ready task. Suspended tasks become ready when the condition that suspended them is removed. Permanently resident dormant tasks become ready when they are called into execution by another task or by an operator command. See Table 7-37.

TABLE 7-37.
RTOS OPERATOR COMMANDS

ONLY FIRST FOUR LETTERS OF EACH COMMAND NEED TO BE ENTERED

ASSIGN	TASK NAME, LU, PA, X	ASSIGN LOGICAL TO PHYSICAL RELATIONSHIP FOR I/O DEVICES WITH PROTECT BIT
ALLOCATE	PA, LLLL, HHHH	SET FILE LIMITS FOR DATA FILES ON DRUM
CALL	NAME	LOAD TASK FROM DRUM INTO CORE
CANCEL	NAME	ABORT NAMED TASK
CONNECT	NAME, LU, PA, X	LINK USER DEVICE DRIVERS TO OPERATING SYSTEM
CONTINUE	NAME	REMOVE NAMED TASK FROM CONSOLE WAIT
DATE	MDDYY	INITIALIZE SYSTEM DATE
DELETE	NAME	UNCONDITIONALLY DELETE FROM CORE
EXECUTE	NAME	LOAD AND GO FROM DRUM
EXTASK	NAME	TYPE OUT TCB INFORMATION
HALT	NAME	PUT NAMED TASK IN CONSOLE WAIT
LEXT	PA	LIST FILE EXTENTS OF PA
MAP		TYPE OUT A MAP OF CORE
OPEN	XXXX	OPEN LOCATION XXXX AND TYPE OUT ITS CONTENTS
OPTION	NAME, bbbb	SET OPTIONS FOR TASK IN BINARY FORMAT
PRIORITY	NAME, XX	CHANGE PRIORITY OF CORE RESIDENT TASK
PROTECT		PUT SYSTEM IN PROTECTED STATE
RDDATE		TYPE OUT SYSTEM DATA IN FORM DD/MM/YY
RDDTIME		TYPE OUT SYSTEM TIME OF DAY HH:MM:SS
REPLACE	XXXX, XXXX	REPLACE CONTENTS OF OPEN LOCATION, HALFWORD OR FULLWORD
START	NAME	PUT NAME IN READY STATE
TELL	NAME, MESSAGE	PASS MESSAGE TO NAME
TIME	HHMMSS	INITIALIZE SYSTEM TIME OF DAY

COMMANDS FOR MAGNETIC TAPE TYPE DEVICES

BSFM	PA	BACKSPACE TO FILE MARK
FRFM	PA	SKIP FORWARD TO FILE MARK
REWIND	PA	REWIND
WTFM	PA	WRITE FILE MARK

Priorities

At any time when more than one task is in the ready state, only one of them can be made active. The task scheduler decides which of the ready tasks to make active. In a real time situation, some tasks are more important, or more urgently needed than others. Whenever a task is established, it is given a priority, a number signifying its importance relative to the other tasks with which it must compete for Processor time, I/O devices, and other system resources. The task scheduler activates tasks according to priority level. Within each priority level it activates task in such a way that tasks share the Processor equitably.

When a new task is created, it is given a numerical priority that is kept in its TCB in the system library. If the operator loads the task, he can specify a new priority that remains with the task until it terminates. If he does not specify a new priority, the task's priority defaults to that contained in the system library TCB. If a task is called by another task, the calling task can specify a new priority that remains with the task until it terminates. If the calling task does not specify a new priority, the called task assumes the higher of two priorities -- its own previously established priority, or that of the calling program. One way or another an incoming task receives a priority number that determines its position in the priority thread for as long as it is core resident or until it is changed by an operator command. As a new task is moved into the ready state it is placed in the priority thread for its level. It time-shares the Processor with other tasks on the same level. In this way, no compute bound task can prevent other tasks on the same priority level from being activated.

RTOS Executive

The RTOS executive is a collection of routines that are entered as a result of internal interrupts. Such interrupts include all Supervisor Calls, Illegal instructions, Arithmetic Faults, I/O Termination, I/O Queue Overflow, and Console Interrupts. The type of interrupt determines the action taken, but in general, interrupts handled by the executive are of two types. Some are handled very quickly within the executive, and others merely cause the executive to start a supervisor task that does the necessary servicing. The executive operates with memory unprotected, in supervisor mode, but with I/O interrupts enabled. Thus I/O transfers may occur without being delayed, but I/O termination and initiation wait. The executive always exits through the task scheduler. Normally the status of at least one task is changed by the executive in the servicing of an interrupt. This means that the task that was active at the time of the interrupt may no longer be the highest priority ready task when the executive exits. For this reason, whenever the executive is entered on an interrupt, it saves the interrupted task's PSW and registers in the save area of the task's TCB. When it exits, the scheduler decides which task is next to be made active and restores its registers and reloads its PSW.

The executive consists of five major sections: the interrupt handler, the task control block administrator, the task scheduler, the task initiator, and the task terminator. The interrupt handler has an entry point for each of the internal interrupt new Program Status Words in the Models 5 and 70. It also contains the task save routine, which saves the current status of the interrupted task, and code for implementation of trivial supervisor calls. The scheduler contains a scanner that can locate the TCB of the next task to be activated. It also contains a task restore routine that restores the registers and loads the PSW of the task being made active. The TCB administrator is called by the supervisor call implementation routines and other supervisor routines to follow threads and modify task status within the task control blocks. This routine, unlike the rest of the RTOS executive, is re-entrant, as it may be called from many different executive routines as well as many supervisor tasks. Its working values are kept in registers, and the only core it modifies is contained in task control blocks. It modifies TCB pointers in an order that leaves all threads continuous at all times so that the scheduler can always activate a task, and the TCB administrator, if re-entered, can always follow all of the TCB pointers.

The task initiator is started by the executive in response to a supervisor call requesting that a particular task be put in the ready state. This routine loads and starts library resident tasks. This includes allocation of core for the task and threading its TCB into the ready thread in a position determined by its priority.

The task terminator routine is started by the executive in response to supervisor call for end of job or a cancel request from the Operator Command Processor task. It adjusts the task status to dormant, removes any tasks waiting for this task from the task wait state, and unless the task is designated as a permanently core resident task, it releases all core associated with the task including its protected TCB.

RTOS Supervisor

The RTOS Supervisor task consists of a collection of subroutines. This task is similar to user-written tasks except that it remains in core after termination and does not appear in the system library. It executes in the supervisor mode, and has access to the TCB administrator. It performs logical I/O and issues supervisor calls as user tasks do, and is scheduled by the RTOS executive. See Table 7-38.

The Operator Command Processor task is a data driven task that accepts, interprets, and acts upon commands issued from the operator device.

The I/O setup task is activated by the executive in response to a request for I/O. It links to the particular driver for the initial setup required. It then triggers the driver by issuing a Simulate Interrupt instruction and terminates itself.

The I/O terminate task is started when an I/O Termination Queue Interrupt occurs. It removes the top entry from the queue and links to the associated driver termination routine. It then removes from I/O wait state any tasks waiting for the device and terminates itself.

The supervisor task is started by the routines in the executive whenever such a routine needs to log an operator message. Because the executive is not a task, it can neither issue I/O calls directly nor be suspended while an I/O device is busy. To handle operator messages, therefore, the executive adds a message-identifier to a circular list and initiates the supervisor task. The task now removes messages from the queue, first-in-first-out, and logs them while the executive continues processing. Processing is halted only in the task responsible for the message, in order to keep the queue from becoming saturated.

The loader-task is initiated by the executive to load tasks and task-overlays from the drum into core. It resembles the INTERDATA OS Library Loader except that its input must come from the system library file. It scans the library-index to find the requested task and then loads the protected (low core resident) portion of the tasks TCB. It then allocates core for the remaining portion of the task control block and the task itself, if sufficient core cannot be found, the caller is placed in core-wait until some non-resident task terminates. If core is available, the remainder of the task is loaded.

When a running tasks requests an overlay, the loader brings the overlay into the space reserved for it in the calling task.

Task Creation

The Task-Establisher Task (TET) is an interactive program which may be used as a task under RTOS to establish other tasks. It loads, links, and edits programs (object files from assemblies or compilations) outputting the resulting code as a relocatable copy of the task, preceded by the TCB, also in relocatable format. This output can only be loaded by the task loader. A Task Utility Task (TUT) may be used to insert the new task in the system task-library. See Tables 7-39 and 7-40.

The Task Establisher requires one scratch-device and an output device, in addition to input device(s) and the console device. It accepts commands interactively and requires three passes (two if no overlays are used) to complete its job. Inputs are scanned and copied onto scratch while a Symbol Table is built up during pass one. Pass two outputs the actual task, and pass three outputs the overlays. Passes two and three use the scratch file output in pass one as input.

RTOS System Library File

The RTOS library is a random access file consisting of an index of named tasks and a relocatable copy of each task and its overlays. This library is accessed by the RTOS Loader when a task is to be called into core, or by the library maintenance programs when tasks are to be inserted or deleted. Deleting a task merely removes its name from the index, and releases its space in the file. Inserting a task adds it to the end.

TABLE 7-38.
RTOS SUPERVISOR CALLS

SVC 1,XXXX	I/O	
SVC 2,XXXX	SERVICE REQUEST	
<u>Code</u>	<u>Function</u>	<u>Meaning</u>
1	PAUSE	PLACE CALLER IN CONSOLE WAIT
2	GET STORAGE	STORAGE ALLOCATION WITHIN TASK'S CORE
3	RELEASE STORAGE	STORAGE DE-ALLOCATION WITHIN TASK'S CORE
4	SET STATUS	MODIFY CALLERS PSW
5	FETCH POINTER	GET ADDRESS OF UNPROTECTED TCB
6	UNPACK	CONTENTS OF RØ CONVERTED TO ASCII HEX
7	LOG MESSAGE	TYPE MESSAGE ON OPERATOR CONSOLE DEVICE
8	INTERROGATE CLOCK	GET TIME OF DAY FROM SYSTEM
9	REQUEST DATE	GET CURRENT DATE FROM SYSTEM
10	TIME WAIT	SUSPEND CALLER UNTIL A SPECIFIED TIME OF DAY
11	INTERVAL WAIT	SUSPEND CALLER A SPECIFIED NUMBER OF MILLISECONDS
12	LOG MSG. & AWAIT RESPONSE	TYPE MESSAGE, ACCEPT ANSWER INTO TELL BUFFER
13	ALLOCATE N 1K BLOCKS	STORAGE ALLOCATION IN 1K BLOCKS
14	RELEASE N 1K BLOCKS	STORAGE DE-ALLOCATION IN 1K BLOCKS
SVC 3,Ø	END OF JOB	
SVC 5,XXXX	FETCH OVERLAY	
SVC 6,XXXX	EXECUTE TASK	
SVC 8,LU	SIMULATE INTERRUPT ON LU	
SVC 10,XXXX	CANCEL TASK	

TABLE 7-39.
RTOS TASK UTILITY TASK (TUT) COMMANDS

COMMANDS ARE ENTERED USING THE TELL COMMAND EXCEPT 'COMMAND' and 'RETURN'	
PURGE	CLEARs TASK LIBRARY OF ALL TASKS
INSERT	READ TASK LOAD MODULE FROM LU2 AND INSERT IN TASK LIBRARY
DELETE NAME	DELETE NAMED TASK FROM SYSTEM LIBRARY
COPY NAME	COPIES NAMED TASK FROM LIBRARY TO LU2
INDEX	GENERATE A LIST OF ALL TASKS IN LIBRARY TO LU 3
OPEN NAME	LOAD TCB INTO CORE - ALLOW MODIFICATIONS
PRIORITY NN	SET PRIORITY
OPTION NNNN NNNN NNNN NNNN	SET OPTIONS IN BINARY
ASSIGN LU, PA	SET LOGICAL UNITS
CORE NNNN	SET CORE SIZE PARAMETER
DISPLAY	LIST TCB ON LU 3
CLOSE	REWRITE TCB, WITH MODIFICATIONS, ONTO LIBRARY
COMMAND	READ FURTHER TUT COMMANDS FROM LU 1 NOT FROM RTOS TELL COMMAND
RETURN	CAUSES A RETURN FROM COMMAND MODE TO TELL MODE
END	CAUSES TUT TO GO TO END-OF-JOB

RTOS Timekeeping

RTOS maintains two clocks, a time of day clock and an interval timer. The time of day counter is a fullword count kept in seconds since midnight. It is driven by the 120Hz interrupt from the Universal Clock. This counter is initialized to zero on system start-up and may be set by the SETTIME operator command. A task may request the current time of day either as a binary fullword denoting seconds since midnight, or as an ASCII character string representing two digits each of hours, minutes, and seconds. A task may request that it be placed in time wait until a specified time of day. The particular time must be specified as a fullword denoting seconds since midnight. The time of day counter is incremented every second and compared with the first task in the time of day wait thread. The tasks in this thread are arranged in an order such that the first in the list is the next to be made ready.

A task may also request that it be placed in time wait for a specified interval. The interval must be specified in a halfword denoting milliseconds from now. These requests are queued with each task in the queue containing the difference between its wait interval and that of the next shortest interval. They are threaded such that the shortest interval from now is first. The Universal Clock is used to drive this thread. It is set to queue an interrupt when the amount of time specified by the task in the top of the queue has elapsed.

TABLE 7-40.
RTOS TASK ESTABLISHER (TET) COMMANDS

1.	ESTABLISH NAME	ASSIGNS NAME TO TASK BEING ESTABLISHED
2.	ASSIGN LU, PA	ESTABLISH LOGICAL TO PHYSICAL RELATIONSHIP
3.	PRIORITY X	SETS PRIORITY OF TASK BEING ESTABLISHED
4.	OPTION bbbb	STORES OPTIONS IN <u>BINARY</u> INTO TCB
5.	GET XXXX	REQUIRED 'GET STORAGE' IN HEX
6.	TELBFR XXXX	RELATIVE ADDRESS OF THE TELL BUFFER
7.	EXCLUDE NAME1,....	SUBROUTINES IN RE-ENTRANT LIBRARY
* 8.	LOAD LU	LOAD MAIN PROGRAM FROM LU
9.	LINK LU	LINK A SINGLE PROGRAM TO THE MAIN PROGRAM
10.	EDIT LU	SEARCH FILE AND LINK NECESSARY PROGRAMS
11.	XOUT LU	OUTPUT BINARY MODULE
12.	END	
	INTQ XXXX	RELATIVE ADDRESS OF INTERRUPT Q
	MAP LU	MAP OF TASK BEING ESTABLISHED
	TCB LU	PRINT OUT GENERATED TCB
* Once the LOAD Command has been issued, commands 2 through 7 are illegal until another task is established.		

RTOS I/O Control

Any task running under RTOS requests I/O through the use of a Supervisor Call instruction. This results in the activation of the I/O setup task which locates the device control block (DCB) for the I/O device and links to the device driver. Within the device driver, there are two distinct programs. One of these operates as an extension of the I/O setup task and runs with interrupts enabled. It never issues I/O instructions directly, but passes parameters to a second driver program that runs with interrupts disabled. This program services device interrupts and directly drives the device.

FORTRAN Programming

The advantages gained from writing application programs in FORTRAN are many. It is an easy language to use; it is familiar to many scientists and engineers; and it is well suited to handling complex numerical calculations that at best are difficult in assembly language. RTOS supports the use of FORTRAN in several ways. The Compiler itself can easily be converted to an RTOS task for doing on-line compilations. The output from the Compiler is accepted by the task establisher which can link and edit from the FORTRAN run time library. Frequently used subroutines from the run time library may be kept in the RTOS re-entrant library so that there need be only one copy of each in memory. There are also six reentrant subroutines supplied with RTOS that bring to the FORTRAN user some of the capabilities of RTOS that would otherwise be available only to assembly language users.

START	— Causes a named task to be started after a specified delay.
TRNON	— Causes a named task to be started at a specified time of day.
IFETCH	— Causes a named overlay to be loaded.
ICLOCK	— Requests the current time of day.
WAIT	— Puts the program in a delayed state.
SYSIO	— Allows the program to perform all I/O functions without going through the standard FORTRAN I/O subroutine.

These subroutines and their use are described in detail in the manual on real time extensions to FORTRAN, 07-048A15.

System Configuration

The first consideration in planning an RTOS system involves the proper selection and allocation of hardware resources. The minimum hardware requirements for an RTOS system, in addition to the Processor, are:

1. 24 KB of memory
2. Memory Protect
3. Power fail/Auto restart
4. Teletype
5. High Speed Paper Tape Reader/Punch
6. Universal Clock

Starting from this point, many configurations are possible. Standard RTOS drivers are available for:

Nine Track Magnetic Tape
 Drum
 Disc
 Line Printer
 Card Reader
 Centronics Printer
 Digital Multiplexor

It is also practical to include in the configuration special devices such as:

Analog to Digital Conversion Equipment
 Digital to Analog Conversion Equipment
 CRT Display Units

Drivers for these devices may be written by the user or provided by INTERDATA as specials.

MEMORY REQUIREMENTS

The principal elements of a complete RTOS system are RTOS proper, the reentrant library, user written system tasks, user tasks, and a data area called task common. Figure 7-29 shows the relative position of these elements in memory, and gives some information regarding their sizes. The exact size of an RTOS system, or the total amount of memory required for any application is a function of many parameters including:

Types and peripheral device supported.

Number and size of user tasks required to be resident in memory at any given time.

The routines resident in the reentrant library.

The size of task common.

THE SIZES GIVEN BELOW IN BYTES ARE SUBJECT TO CHANGE, AND ARE FOR ESTIMATING PURPOSES ONLY.

DEVICE	SIZE	SIZE
	HEXADECIMAL	DECIMAL
TELETYPE	436	1078
PAPER TAPE READER/PUNCH	1DA	474
CARD READER	19C	412
LINE PRINTER	98	152
MAGNETIC TAPE	2C2	706
DRUM	1CA	458
CARTRIDGE DISC *	200	512
UNIVERSAL CLOCK *	142	322
DIGITAL MULTIPLEXOR	92	146
INTERRUPT MODULE	38	56

* THESE DRIVERS MAY REQUIRE 2 DCB'S, 1 PER DEVICE ADDRESS.

Figure 7-29a. RTOS Device Drivers

TOP OF MEMORY	SIZE (BYTES) *		
	TASK COMMON	SOME MULTIPLE OF 1KB	THIS OPTIONAL TASK COMMON AREA IS USED FOR INTER-TASK COMMUNICATION. IF USED, MUST BE A MULTIPLE OF 1KB.
USER TASKS	T _n	AS REQUIRED FOR USER TASKS	USER TASKS RESIDE IN THIS AREA. EACH TASK MUST START ON SOME 1KB BLOCK BOUNDARY AND OCCUPY SOME MULTIPLE OF 1KB CONSECUTIVE BLOCKS. MEMORY REQUIRED FOR USER TASKS IS ALLOCATED DYNAMICALLY AT LOAD TIME.
	•		
	•		
	T ₃		
	T ₂		
	T ₁		
REENTRANT LIBRARY	SUBROUTINES	•	THE REENTRANT LIBRARY HOLDS RESIDENT ROUTINES WHICH ARE AVAILABLE TO EVERY TASK. THE SIZE AND CONTENT OF THIS LIBRARY IS FIXED DURING SYSTEM GENERATION.
	•		
	TABLE	8 PER ENTRY	
RTOS	DRIVERS	SEE LIST, FIG. 4-1a	ONE DRIVER PER TYPE OF DEVICE.
	DCB'S	64 EACH	ONE DCB PER PHYSICAL DEVICE
	TCB'S	52 EACH	ONE TCB PER USER TASK.
	SUPERVISOR	4764	THE BASE PORTION OF RTOS WHICH IS REQUIRED FOR EVERY SYSTEM.
	LOADER	2988	
	EXECUTIVE	4962	
RESERVED LOCATIONS	736		

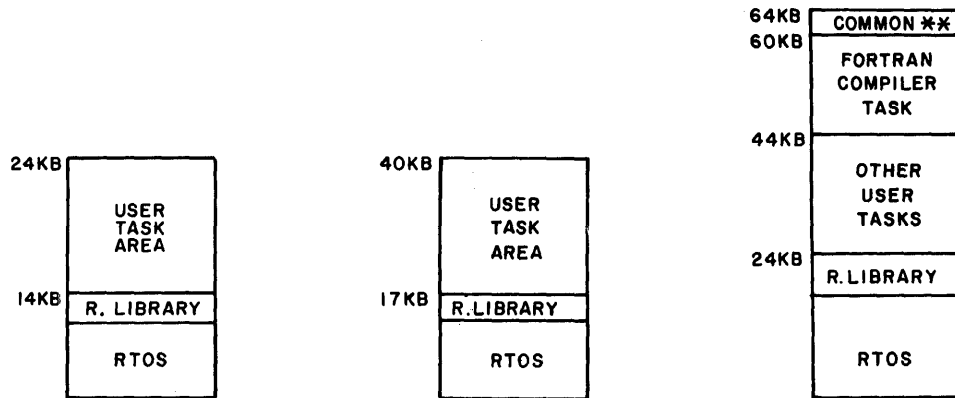
* SIZES SHOWN ARE SUBJECT TO CHANGE, AND ARE FOR ESTIMATING PURPOSES ONLY.

Figure 7-29b. Memory Map

Figure 7-29. Device Drivers and Memory Map

In round numbers the size of RTOS itself varies from 13KB to 20KB, depending on the device drivers, the number of device control blocks, and the number of task control blocks included in the system.

The size of the total memory required varies from 24KB to 64KB depending on the selected configuration of RTOS, and the amount of memory required for the reentrant library, user tasks, and task common. Some examples of complete configurations are shown in Figure 7-30.



RTOS		BASE		BASE	
BASE	12624	BASE	12624	BASE	12624
4 TCB'S	208	8 TCB'S	416	16 TCB'S	832
8 DCB'S	512	15 DCB'S	960	21 DCB'S	1344
TTY DRIVER	1078	TTY DRIVER	1078	(2) TTY DRIVER	1078
PTR/P DRIVER	474	PTR/P DRIVER	474	PTR/P DRIVER	474
CR DRIVER	412	CR DRIVER	412	CR DRIVER	412
UNIV CLOCK DRIVER	322	UNIV CLOCK DRIVER	322	UNIV CLOCK DRIVER	322
ADC/DAC DRIVER *	212	DISC DRIVER	458	DISC DRIVER	458
DIGITAL MUX DRIVER	146	8-LINE INT. MOD DRIVER	56	8-LINE INT. MOD DRIVER	56
	<u>15988</u>	DIGITAL MUX DRIVER	146	DIGITAL MUX DRIVER	146
			<u>16946</u>	ADC/DAC DRIVER *	212
				(2) M. TAPE DRIVER	706
				L. PRINTER DRIVER	152
					<u>18816</u>
REENTRANT LIBRARY					
TABLE (10)	80	TABLE (20)	160	TABLE (30)	240
ROUTINES	214	OTHER ROUTINES	764	FORTRAN I/O **	4102
	<u>16282</u>		<u>17870</u>	OTHER ROUTINES	1864
					<u>25022</u>
	= 16KB		= 18KB		= 25KB
USER TASK AREA	8KB		22KB		39KB
TOTAL	24KB		40KB		64KB
CONFIGURATION A		CONFIGURATION B		CONFIGURATION C	

* NOT STANDARD
** OPTIONAL

Figure 7-30. Typical RTOS Configurations

CHAPTER 8

PERIPHERAL DEVICES AND MODULES

8.1 INTRODUCTION

This chapter describes some of the peripheral devices and modules that are available from INTERDATA.

The ASR 33 and 35 Teletypewriters, the Paper Tape Reader/Punch, the Card Reader and the Selector Channel are covered in detail including sample programs which provide the user with an introduction on programming INTERDATA supplied peripherals. In addition, detailed specifications are provided on the following peripheral devices and modules.

- Removable Cartridge Disk System
- 201 Synchronous Data Set Adapter
- Programmable Asynchronous Line System (PALS)
- INTERTAPE Cassette System
- Automatic Memory Protect Controller
- Universal Clock
- The Eight Line Interrupt Module

8.2 TELETYPEWRITERS

8.2.1 Introduction

This specification contains a description of the M48-010 (02-262) Teletype (TTY) Interface and the information necessary to program the TTY. This programming information also pertains to the Teletype interface which is built into the Model 50, 70, and 80 Processors. This interface is program compatible with previous Teletype interfaces with minor exceptions.

This interface contains a hard-wired character format/ baud rate with a 20 milliamperes TTY loop and may be used with the 33 ASR/KSR or 35 ASR/KSR TTY or any equivalent terminal.

This interface is contained on a single 7" x 15" printed circuit board and will adapt a single TTY to the Multiplexor Channel.

The following is pertinent information associated with this product:

Character Code	8 level, 11 unit code (one start and two stop bits).
Transmission Method	Serial by bit.
Transmission Speed	10 characters/second, 110 baud.

The following is pertinent information for applicable Teletypewriters.

Device INTERDATA Product Number M46-000 (ASR33) and M46-001 (ASR35)

Model Numbers - ASR33 and ASR35

Data Rate - 10 Characters per second

Printer Width - 72 Characters maximum

Paper Feed - Pin feed

Dimensions - W 22", D 18-1/2", H 32-7/8 (without stand) overall ASR33
W 38-1/2", D 24", H 38-1/2" (includes stand) overall ASR35

Weight - 44 pounds desk top ASR33
225 pounds (includes stand) ASR35

Power Requirement - 115VAC 60Hz

15A start up - ASR33
3A running

12A start up - ASR35
4A running

115VAC 50 Hz models are available

8.2.2 Configuration

The 7" x 15" TTY Interface may be used with Model 50, 70, 74, 80 or equivalent INTERDATA Processors.

8.2.3 Operating Procedures

8.2.3.1 ASR 33 Features

A three position power switch is located to the right and below the keyboard. When rotated left to the position marked LINE, power is applied to the TTY and the device is logically connected to the Processor. With the switch in the OFF or LOCAL position, the TTY is logically disconnected from the system (DU=1).

8.2.3.2 ASR 35 Features

The ASR 35 is a heavy-duty version of the ASR 33. Operation of the ASR 35 is similar to the ASR 33 with the following exceptions.

1. The tape reader and tape punch controls are different as explained later in this description.
2. The ASR 35 has a Mode Control switch to the left of the keyboard. The meanings of the five positions of this switch are illustrated in Figure 8-1.
3. Several additional keys, such as Local Line Feed, are provided. The meanings of these keys are self-explanatory.

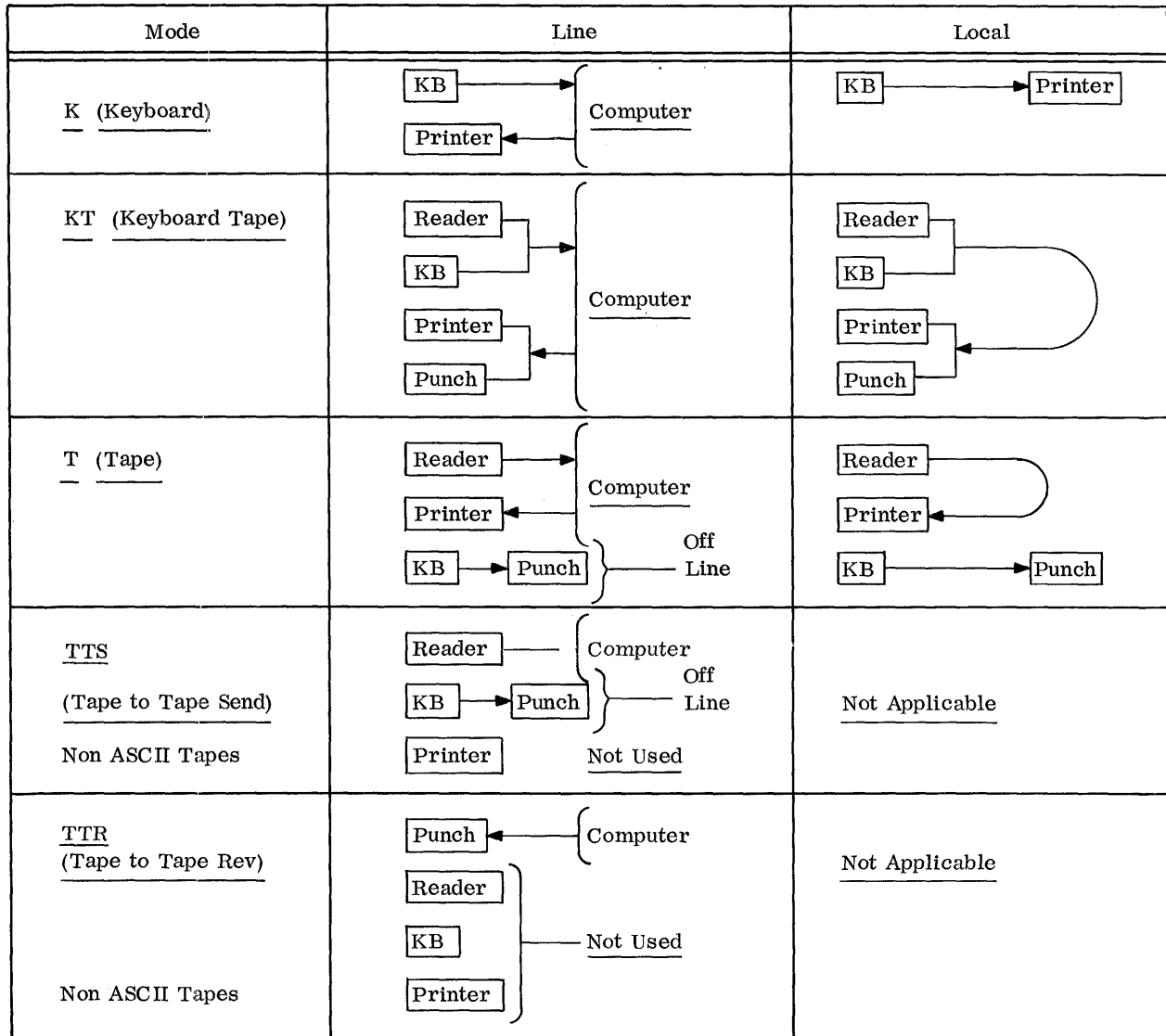


Figure 8-1. 35 ASR Operating Modes

8.2.3.3 Paper Tape Reader

The paper tape reader is controlled by a four-position switch* on the reader. The four positions are MANUAL START, MANUAL STOP, AUTO, and FREE. When the switch is moved to the MANUAL START (START) position, the tape in the reader is advanced at ten characters per second. Tape motion continues until the reader switch is moved to the MANUAL STOP (STOP) position. In the AUTO (STOP) position, tape motion can be under program control, assuming that the Power switch is in the LINE position. The control characters which affect the reader are X-ON (X'91') which starts the tape motion, and X'OFF (X'93') which stops the tape motion. The FREE (FREE) position permits the tape to be moved manually over the read mechanism.

*Some TTYs have a three-position switch labelled START, STOP and FREE. The relationship of the three-position switch to the four-position switch is shown in parenthesis.

8.2.3.4 Paper Tape Punch

The ASR 35 Paper Tape Punch is enabled only when the ASR 35 MODE switch is in the KT or TTR position. In these modes, the punch is controlled via TAPE and TAPE keys. TAPE and TAPE characters are described above. Refer to Figure 8-1 for details. Following the program transfer of a TAPE character to start the ASR 35 Paper Tape Punch, the program should output two or three rubout characters (X'FF') to achieve data synchronization prior to punching the data.

To manually turn off the punch on an ASR 33 TTY, the following steps are required:

1. Turn the Power switch to LOCAL mode.
2. Depress the UNLOCK key on the tape punch.
3. Strike the TAPE key while the CTRL key is depressed.

If the ASR 33 is not in a LOCK "ON" mode (depress UNLOCK to release the LOCK "ON" mode), and the Power switch is in the LINE position, the tape punch can be under program control.

The specific control characters which affect the punch are TAPE (X'92'), which starts the punch, and TAPE (X'94'), which stops the punch. The punch controls are achieved by outputting the appropriate character to the Teletype. Note that the TAPE and TAPE characters, themselves, get punched on the tape.

The tape punch can be manually started in an alternate way. If the punch is not already on, strike the TAPE key with the CTRL key depressed, and the Power switch in LOCAL mode. This technique is equivalent to transferring a TAPE ON character to the Teletype from the Processor.

The ASR 35 Paper Tape Punch is enabled only when the ASR 35 MODE switch is in the KT or TTR position. In these modes, the punch is controlled via TAPE and TAPE keys. TAPE and TAPE characters are described above. Refer to Figure 8-1 for details. Following the program transfer of a TAPE character to start the ASR 35 Paper Tape Punch, the program should output two or three rubout characters (X'FF') to achieve data synchronization prior to punching the data.

8.2.4 Data Format

Figure 8-2 shows the character format and Figure 8-3 shows the ASR 33 keyboard layout.

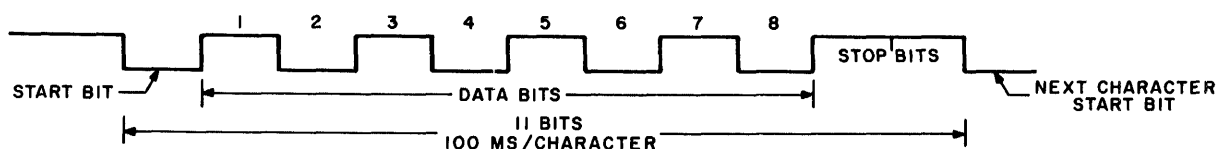


Figure 8-2. ASCII Character U (Even Parity), Eleven Bit Code

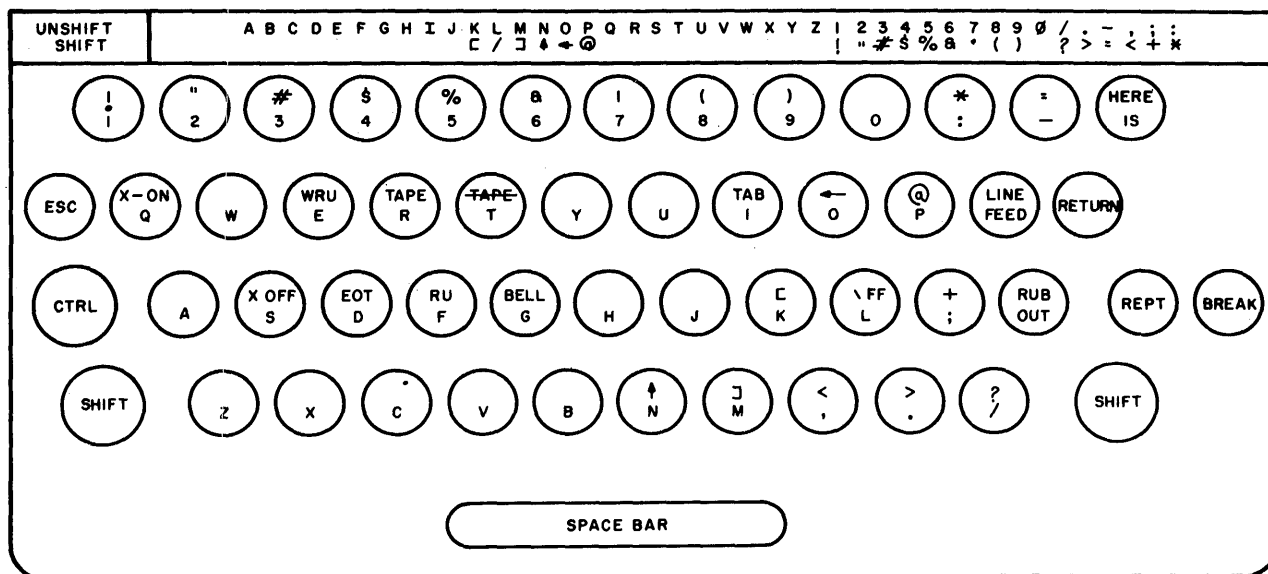


Figure 8-3. Teletype Keyboard Layout

NOTE

Teletypewriters purchased from INTERDATA have the even parity keyboard option implemented.

8.2.5 Programming Instructions

8.2.5.1 Programming Instructions

1. Sense Status (SS or SSR). The Sense Status instruction is used to interrogate the TTY and character status.
2. Output Command (OC or OCR). The Output Command instruction selects Read/Write, Keyboard Block/Unblock, and interrupt Enable/Disable and Disarm.
3. Write Data (WD or WDR). The Write Data instruction is used to load the output character into the Data Register.
4. Read Data (RD or RDR). The Read Data instruction is used to read an assembled character into the Processor.
5. Acknowledge Interrupt (AI or AIR). The Acknowledge Interrupt instruction is used to service interrupts and returns the address and status of the interrupting device.

8.2.5.2 Status and Command Bytes

Table 8-1 contains the TTY Interface status and command byte data.

TABLE 8-1. TTY INTERFACE STATUS AND COMMAND BYTE DATA

BIT NUMBER	0	1	2	3	4	5	6	7
STATUS BYTE	OV	X	BRK	X	BSY	EX	X	DU
COMMAND BYTE	DISABLE	ENABLE	UNBLOCK	BLOCK	WRITE	READ	—	—

DISABLE
ENABLE

 DISARM

X = Unassigned (associated status bit = 0)

— = Unused

STATUS

- OV** **Overflow:** In the read mode, this status bit is set if the previously received character is not read before the present character is assembled. Double character buffering permits a full character grab-time; 100 milliseconds. OV is reset with an Output Command (OC)*, Read Data Instruction, or initialize. In the Write mode, OV is forced reset.
- BRK** **Line Break:** This bit is set when the serial data line from the TTY is a ZERO (Space) for longer than one character period or if the first Stop bit is missing (see Figure 8-2). In the receive mode, the character is assembled and BRK is set when the BSY interrupt occurs. This status bit remains active until the line goes to a ONE (line break on TTY released).
- BSY** **Busy:** This bit is set when it is not possible to transfer a character through the interface. Thus, in the Read mode, it is normally high and goes low whenever a character is received and ready to be read by the Processor. In the Write mode, it is normally low and goes high when the Processor outputs a character, and low when the interface is ready for the next character. An interrupt is generated, if enabled, when Busy goes low.
- EX** **Examine:** This bit is set when either or both of status Bits 0 and 2 are set.
- DU** **Device Unavailable:** This bit is set whenever the Power switch on the Teletype is in the LOCAL or OFF positions.

COMMAND

- DISABLE** **Disable Interrupt:** This bit set prevents the device from interrupting the Processor, but allows interrupts to be queued.
- ENABLE** **Enable Interrupt:** This bit set allows the device to interrupt the Processor.
- DISARM** **Setting both the DISABLE and the ENABLE bits prevents the device from interrupting or queueing the interrupts.**
- UNBLOCK** **Setting this bit causes characters read from the keyboard or tape reader to be printed (and punched if the tape punch is on).**
- BLOCK** **Setting this bit prevents characters read from the keyboard or tape reader from being printed or punched. It also prevents the normal actions of control codes (such as X-ON, WRU, etc.) when entered from the reader or keyboard.**
- WRITE** **Setting this bit places the interface in the Write mode allowing data to be output from the Processor to the printer or punch.**
- READ** **Setting this bit places the interface in the Read mode allowing data to be input from the keyboard or tape reader.**

*Output Command READ for Model 70 built-in TTY.

8.2.6 Programming Sequences

8.2.6.1 Programming Notes

1. Paper Tape Reader. Note that when stopping the tape reader under program control, the tape may advance one or two characters between the time the X OFF character is issued and the time the tape comes to a complete halt. Similarly on starting a tape, one or two characters may be missed before synchronization is attained. Therefore, tapes to be read under program control should be formatted to account for the start/stop characteristics of the Paper Tape Reader.

These procedures apply to both the ASR 33 and the ASR 35 Teletypes. The ASR 35 Paper Tape Reader is enabled, however, only when the ASR 35 Mode switch is in the KT, T, or TTS positions. See Figure 8-1 for details.

2. Paper Tape Punch. When using the punch, if a character is output which is equivalent to the WRU (X'05'), this causes the TTY to send a string of 20 characters which mutilates the characters being punched. For this reason, and the fact that punching a TAPE OFF (X'94') turns off the punch, binary tapes with arbitrary eight bit characters cannot be punched. By convention, binary tapes are usually punched on the Teletype using a "zoned" data format, with four bits of data and four bits of zone per tape frame. The 16 tape characters used are X'90', X'81', X'82', X'83', X'84', X'95':X'9F'.
3. Keyboard. The Carriage Return (CR) character requires approximately two character periods to return the Teletype carriage. Thus, if it is followed immediately by a printing character, this character may be printed while the type-mechanism is still moving. To avoid this possibility, CR may be followed by a non-printing character. By convention, CR is always followed by a Line Feed (LF) at the end of a line.

If the keyboard is operated or the Paper Tape Reader is moving while the Processor outputs data to the Teletype, the output data can be either garbled or lost. In this situation, the conflict in the data transfers can be detected in the program by the response of the Overflow flag (V) following a Write Data (WD or WDR) instruction, which indicates that the Write operation was not successful. If the Overflow flag occurs after a Write Data instruction, the instruction should be re-executed to correctly transfer the character from the Processor to the TTY.

8.2.6.2 Programming Examples

The programmed steps required to start the reader are:

	OC	DEV, WRITE	Set Write Mode
WAIT1	SS	DEV, STATUS	Wait for BSY=0
	BTC	BSY, WAIT1	
	WD	DEV, XON	Start Tape
	OC	DEV, READ	Set Read Mode
WAIT2	SS	DEV, STATUS	Wait for BSY=0
	BTC	BSY, WAIT2	
	RD	DEV, DATA	Read a Character

The programmed steps required to stop the reader are:

	OC	DEV, WRITE	Set Write Mode
WAIT3	SS	DEV, STATUS	Wait for BSY=0
	BTC	BSY, WAIT3	
	WD	DEV, XOFF	Stop the Tape

Other routines for transferring data to and from the TTY are shown in Table 8-2.

TABLE 8-2. SAMPLE PROGRAM LISTING (CONTINUED)

			OPT	PASS2, PRINT, NOPNCH, STOP	
			*OUTPUT		
			*		
			*		
			*A BYTE WILL BE OUTPUT FROM R4 TO THE TELETYPE		
			*REGISTERS R3, R4, R5, R15 WILL BE USED		
			*THE CALL IS BAL R15, OUTPUT		
			*		
			*		
0000R			ENTRY	OUTPUT	
0000R	C830	OUTPUT	LHI	R3, DEVNO	LOAD DEVICE NUMBER
	0002				
0004R	DE30		OC	R3, BLOCK	SET DEVICE MODE
	0012R				
0008R	9D35	SENS	SSR	R3, R5	INPUT STATUS
000AR	42F0		BTC	X'F', SENS	LOOP IF NOT RDY
	0008R				
000ER	9A34		WDR	R3, R4	OUTPUT BYTE
0010R	030F		BR	R15	RETURN TO CALL
0003		R3	EQU	3	
0004		R4	EQU	4	
0005		R5	EQU	5	
000F		R15	EQU	15	
0002		DEVNO	EQU	2	
0012R	9800	BLOCK	DC	X'9800'	DISABLE, BLOCK, WRITE
0014R			END		
BLOCK	0012R				
DEVNO	0002				
* OUTPUT	0000R				
R15	000F				
R3	0003				
R4	0004				
R5	0005				
SENS	0008R				

TABLE 8-2. SAMPLE PROGRAM LISTING (CONTINUED)

		OPT	PASS2, PRINT, NOPNCH, STOP		
		*TTY OUTPUT EXAMPLE DISABLE, BLOCK, WRITE			
		*TYPOUT			
		*			
		*			
		*A SERIES OF BYTES WILL BE OUTPUT			
		*AS DETERMINED BY THE CALLING SEQUENCE			
		*			
		*REGISTERS R3, R4, R5, R6, R15, WILL BE USED			
		*THE CALLING SEQUENCE IS			
		* BAL R15, TYPOUT			
		* DC A(MESS) STARTING ADDRESS			
		* DC A(END) ENDING ADDRESS+1			
		*			
		*			
0000R			ENTRY	TYPOUT	
0000R	486F	TYPOUT	LH	R6, 0(R15)	GET STARTING ADRS
	0000				
0004R	485F		LH	R5, 2(R15)	GET ENDING ADRS
	0002				
0008R	C830		LHI	R3, DEVNO	LOAD DEVICE NUMBER
	0002				
000CR	DE30		OC	R3, BLOCK	SET MODE
	0028R				
0010R	9D34	SENS	SSR	R3, R4	INPUT STATUS
0012R	42F0		BTC	X'F', SENS	TEST STATUS
	0010R				
0016R	DA36		WD	R3, 0(R6)	OUTPUT DATA
	0000				
001AR	CA60		AHI	R6, 1	INCREMENT ADRS
	0001				
001ER	0565		CLHR	R6, R5	TEST FOR END
0020R	4280		BTC	X'8', SENS	LOOP IF NOT
	0010R				
0024R	430F		BFC	0, 4(R15)	RETURN TO CALL
	0004				
0003		R3	EQU	3	
0004		R4	EQU	4	
0005		R5	EQU	5	
0006		R6	EQU	6	
000F		R15	EQU	15	
0002		DEVNO	EQU	2	
0028R	9800	BLOCK	DC	X'9800'	
002AR			END		
BLOCK	0028R				
DEVNO	0002				
R15	000F				
R3	0003				
R4	0004				
R5	0005				
R6	0006				
SENS	0010R				
* TYPOUT	0000R				

8.2.9 Device Number

The TTY Interface is assigned Device Number X'02'. This may be changed by a minor wiring alteration in the interface. When the M48-010 (02-262) Teletype Interface is used on a Model 50, 70, or 80 Processor, the device numbers must be adjusted so that it differs from that used by the Teletype controller built into the Processor.

8.3 HIGH SPEED PAPER TAPE READER/PUNCH (HSPTR)

This section provides information on the operation and programming of the High Speed Paper Tape Reader, and the Combination High Speed Reader/Punch. INTERDATA Product Numbers M46-240 and M46-242 respectively. Included in this section is a general description, a table of status and command bytes, and sample programs for each device.

The above products all include a single-board device controller. Note, that with the Combination Reader/Punch, since there is only one device controller, the devices cannot be used simultaneously. To read and punch tapes at the same time, it is necessary to use products which would provide separate device controllers for each device.

8.3.1 General Description

Table 8-3 lists general characteristics of the reader, the punch, and the controller.

TABLE 8-3
READER AND PUNCH CHARACTERISTICS

Characteristics	Reader	Punch
Type	Photo-electric	Electro-mechanical
Tape Width	adjustable tape guides for 11/16" and 1" tape (6 and 8 level tapes)	same as the Reader
Speed	maximum of 300 characters-per-second	maximum of 75 characters-per-second
Tape handling	paper, paper-mylar, and mylar	same as the Reader
Stop time	capable of stopping on a character	will punch the character and stop
Run/Load Switch	allows loading or changing of tapes of varying widths	same as the Reader
Power Switch	applies AC power to Reader	applies AC power to Punch
Dimension	19"W, 7"H, 6 $\frac{1}{4}$ "D	19"W, 10 $\frac{1}{2}$ "H, 22"D (R/P combination)
Rack Mountable	Yes	No (mount on slide)
Weight	26 $\frac{1}{2}$ lbs.	60 lbs. (R/P combination)
Power Requirement	220VAC/115VAC, 50/60 Hz	same as the Reader

8.3.2 Status and Command

Table 8-4 provides Status and Command Byte Data for the HSPTR/P.

TABLE 8-4
READER/PUNCH STATUS AND COMMAND BYTE FORMAT
(HEX ADDRESS 13)

BIT NUMBER	0	1	2	3	4	5	6	7
STATUS BYTE	OV			NMTN	BSY	EX		DU
COMMAND BYTE	DISABLE	ENABLE	STOP	RUN	INCR	SLEW	WRITE	READ
STATUS BIT DESCRIPTIONS								
Bit	Reader			Punch				
OV	The Overflow bit is set when the Buffer Register is loaded from the Reader before the previous character has been transferred. This condition can only happen in the Slew Mode.			The Overflow bit is always in a reset condition in the Write Mode.				
NMTN	The No-Motion bit is set when the Reader has been issued a Stop command and the tape has stopped on the next character.			The No-Motion bit is always in a reset condition in the Write Mode.				
BSY	The Busy bit is set when the Buffer Register is empty, waiting for a character from the Reader, or the Reader is in Load condition or the Reader power is not stabilized.			The Busy bit is set when the tape is advancing and in punch cycle.				
EX	The Examine bit is set whenever OV=1 or NMTN=1.			The Examine bit is always reset in the Write mode.				
DU	The Device Unavailable bit is set when the power to the Reader is off, or the Run/Load lever is in the Load position or the power is not stabilized, or if the drive signal is received and new feed hole is not sensed within 10 ms, indicates either no tape or torn tape and serves as the out of tape signal.			The Device Unavailable bit is set when the power to the Punch is off, or internal voltages have not stabilized, or Run/Load switch is in Load position.				
COMMAND BIT DESCRIPTIONS								
DISABLE	This command inhibits interrupts from the device controller from interrupting the Processor. interrupts are queued.			Same as the Reader.				
ENABLE	This command permits interrupts from the device controller to interrupt the Processor.			Same as the Reader.				

TABLE 8-4
 READER/PUNCH STATUS AND COMMAND BYTE FORMAT
 (HEX ADDRESS 13) (Continued)

COMMAND BIT DESCRIPTIONS (Continued)		
Bit	Reader	Punch
STOP	This command halts the motion of the tape. The next character to be read is positioned over the sense lights when the tape stops.	Not used.
DISARM	When DISABLE and ENABLE are both set to one. This command prevents the device from interrupting or queuing the interrupts.	Same as the Reader.
RUN	This command starts the tape moving and leaves the controller in the Run Mode.	Not used.
INCR	In this mode of operation, the tape is advanced one character when the controller is in the Run Mode and a Read Data instruction is executed. The tape stops after encountering the next character. The tape remains stopped until a Read Data instruction is issued by the Processor, which will start the tape moving.	Not used.
SLEW	In this mode of operation, the tape is advanced, reading continuous characters, until stopped.	Not used.
WRITE		Designates the High Speed Paper Tape Punch.
READ	Designates the High Speed Paper Tape Reader.	

8.3.3 Interrupts

When enabled in the Read Mode, the device controller generates an external device interrupt when a data character is present in the controller, waiting to be transferred to the Processor or Device Unavailable. When enabled, in the Write Mode, the device controller generates an external device interrupt when the controller is ready for another character to be punched, or Device Unavailable.

8.3.4 Initialization

When the INT switch on the Processor is depressed, the following occurs:

1. Interrupts of all kinds are disarmed.
2. The NMTN and EX Status bits are set.
3. The DISARM, STOP, INCR, and READ command functions are set.

8.3.5 Device Number

The High Speed Reader/Punch is normally assigned address X'03' if using a reader only. If using both a reader and a punch, address X'13' is normally assigned. These device numbers are easily changed by a minor modification to the device controllers.

8.4 CARD READER

8.4.1 General Description

The Card Reader, INTERDATA Product Number 7-510, employs a photoelectric read station and a vacuum throat feed assembly. A special "wide strobe" read technique is used to preclude loss of data, even on cards which have been mispunched by as much as plus or minus one-half column.

The card read rate is in excess of 200 cards per minute with a 500 card capacity for both the input hopper and the output stacker.

Throughout the read operation, light current checks, dark current checks, and card motion checks are continuously performed to verify the performance of the Card Reader.

Card Reader

Dimensions: 13"H, 12"D, 23"W

Weight: 75 lbs.

Power Requirement: 115VAC, 300VA max.

8.4.2 Operator Controls

POWER

The lighted POWER pushbutton applies AC power to all circuits. The pushbutton indicator lights when the power is on.

MOTOR Start

The lighted MOTOR pushbutton starts the drive motor if no error indicator lights are lit. The pushbutton lights when the drive motor is running.

Read START

The lighted START pushbutton clears all error indicators and advances the Card Reader to the "ready" state to begin a read cycle upon receipt of the proper signal. The pushbutton lights when the switch is depressed and no errors have been detected.

Read STOP

The lighted STOP pushbutton inhibits further read cycles until Read START is again depressed. Read STOP action is delayed until the current read cycle is completed. The pushbutton is lights when the switch is depressed, or if the Card Reader is stopped due to an error detection.

8.4.3 Status Indicator Lights

POWER On

The indicator on the POWER switch lights when power is applied to the Card Reader.

MOTOR On

The MOTOR switch indicator lights when the motor is running.

Read START

The START switch lights when the switch is depressed and no malfunctions have been detected.

Read STOP

The STOP switch lights when the switch is depressed or the Card Reader has stopped due to a trouble detection, as described in the following paragraphs.

PICK FAIL

If a card fails to be picked upon command, the PICK FAIL indicator lights.

CARD MOTION Error

If the interval between the time the selected card enters the read station and the time the card leaves, does not correspond to $85 + 1/3$ columns (the total card width), the CARD MOTION indicator lights.

LIGHT CURRENT Error

When all photo-read-cells do not conduct whenever a card is not in the read station, the LIGHT CURRENT indicator lights.

DARK CURRENT Error

The DARK CURRENT indicator lights if all photo-read-cells do not go dark for some instant between the beginning of the card and Column 1, or between Column 80 and the end of the card.

8.4.4 Status and Command Bytes

Table 8-5 illustrates the Status and Command Byte coding for the Card Reader.

8.4.5 Data Format

A card Feed command causes the card to move over the photo-read-cells column by column, starting with Column 1. Every column read (blank columns are read as all bits zero) generates a data strobe for that column and initiates a data transfer cycle. The first Read Data instruction reads the top six rows of the column; the second Read Data instruction reads the bottom six rows of that column. Figure 8-4 is an example of the data byte format.

8.4.6 Interrupts

When enabled (Bit 1 of the Command byte set), the Card Reader device controller generates an external device interrupt for each column read. The interrupt indicates to the Processor that data is available for transfer.

8.4.7 Initialization

When the INT pushbutton on the Processor is depressed, the following occurs:

1. The NMTN and EOM bits are set.
2. The EOVB bit is reset.
3. The BSY and EX bits are set.

8.4.8 Operator Procedures

After applying power to the Card Reader, allow it a few minutes to warm up. Cards should be placed face down in the hopper with the 12-edge toward the operator. Additional cards may be added to the hopper without interfering with the operation.

TABLE 8-5
CARD READER STATUS AND COMMAND BYTE DATA
(HEX ADDRESS 04)

BIT NUMBER	0	1	2	3	4	5	6	7
STATUS BYTE	EOV	TBL	HE	NMTN	BSY	EX	EOM	DU
COMMAND BYTE	DISABLE	ENABLE	FEED					
EOV	The EOV bit is set when the data is not taken from the device controller buffer before the next column of data arrives from the read station. This bit is reset by a FEED Command.							
TBL/DU	<p>These bits are set when the Card Reader fails to pick a card upon command, or when an error condition occurs in the Card Reader. The error conditions are:</p> <ol style="list-style-type: none"> 1. Card Motion Error 2. Light Current Error 3. Dark Current Error <p>These error conditions prevent the reading of any more cards until manually reset by the operator.</p>							
HE	The Hopper Empty (HE) bit is set when there are no cards in the input hopper. The HE bit must be manually reset by the operator.							
NMTN	The NMTN bit is set except for the time between a FEED Command and the time it takes for a card to pass through the read station.							
BSY	The BSY bit is set while the device controller is awaiting data from the Card Reader. It resets when the data is available to be transferred.							
EX	The EX bit sets when any one of the upper four (4) bits of the Status byte is set.							
EOM	The EOM bit is set whenever NMTN bit is set, or when the input hopper becomes empty.							
DISABLE	This command disables the Card Reader Device Interrupt. Interrupts are queued.							
ENABLE	This command enables the Card Reader Device Interrupt.							
FEED	This command initiates a new card feed cycle; however, no action occurs if TBL, DU, or HE is set.							

BIT NUMBER	0	1	2	3	4	5	6	7	
ROW NUMBER			12	11	0	1	2	3	FIRST DATA BYTE
ROW NUMBER			4	5	6	7	8	9	SECOND DATA BYTE

NOTE: Bit numbers 0 and 1 should always be zero.

Figure 8-4. Data Byte Format

8.4.9 Programming

A sample card input routine is shown in Table 8-6. In the sample program, note that the Hopper Empty (HE) is checked before other bits. This bit does not become set until the last card is read. If 80 columns are not read from each card, there is a Card Reader malfunction, as all blank columns should be read as zeros.

See the Hollerith punched-card codes for the ASCII character set shown in the Appendices.

TABLE 8-6
SAMPLE PROGRAM FOR CARD READER

		*	
		*	
		* NON-INTERRUPT CARD READER ROUTINE	
		*	
DEVNUM	EQU	1	REGISTER DEFINITIONS
STATUS	EQU	2	
INDEX	EQU	3	
INCR	EQU	4	
LIMIT	EQU	5	
TEMP	EQU	6	
RETURN	EQU	15	
SINDV	EQU	X'7C'	50 SEQUENCE SOURCE INPUT DEVICE
*			
READ	XHR	INDEX, INDEX	ZERO INDEX VALUE
	LIS	INCR, 2	SET UP INCREMENT
	LHI	LIMIT, 158	SET UP LIMIT
	LB	DEVNUM, SINDV	SET DEVICE NUMBER
WAIT	SSR	DEVNUM, STATUS	
	THI	STATUS, X'20'	HOPPER EMPTY CHECK
	BNZ	EMPTY	GO HANDLE HE CONDITION
	THI	STATUS, X'41'	DON'T ISSUE FEED TILL TBL/DU & HE CLEAR
	BTBS	2, 7	GO WAIT
*			
FEED	OC	DEVNUM, SINDV+1	START CARD READER
SENSE	SSR	DEVNUM, STATUS	
	BTC	7, ERROR	BITS SHOULD NOT BE SET
	BTBS	8, 3	BUSY BIT SET HANG, GO TO SENSE
	RH	DEVNUM, BUFFER(INDEX) FIRST CHAR.	READ ALL ROWS
	BXLE	INDEX, SENSE	80 COLUMNS READ
	SSR	DEVNUM, STATUS	AT END OF CARD'S 80 COLUMNS
	THI	STATUS, X'C1'	CHECK EOVTBL/DU BITS
	BNZ	MOTION	TO CATCH MOTION/CURRENT ERRORS
	THI	STATUS, X'2'	
	BFBS	2, 7	WAIT TILL EOM SETS
*			
		* NOW PROCESS CARD INFORMATION IN BUFFER BY	
		* DOING HOLLERITH TO ASC11 CONVERSION ROUTINE	
		*	
	BR	RETURN	
BUFFER	DS	160	
*			
EMPTY	EQU	*	HANDLE INPUT HOPPER EMPTY
*			
ERROR	EQU	*	HANDLE ERROR CONDITIONS
*			
MOTION	EQU	*	HANDLE MOTION, EOVT, CURRENT ERRORS
END			

8.5 REMOVEABLE CARTRIDGE DISC SYSTEM

8.5.1 General Description

The INTERDATA Removable Cartridge Disc System provides a low cost, random access, bulk storage facility for the INTERDATA family of Processors. The system consists of a removable cartridge disc drive capable of storing 2,457,600 8-bit bytes of formatted data, a disc drive controller which can handle four disc drives, and a system power supply which handles two disc drives.

The disc drives use an IBM 2315 cartridge or equivalent for its storage media. Spindle speed and head positioning are both electronically controlled, thus eliminating many mechanical assemblies and electrical components. This electrical and mechanical simplicity results in high reliability, ease of maintenance, and low operating costs. The physical package results in low power dissipation with no external cooling required.

The disc controller is designed to operate with a Selector Channel for autonomous block transfers. The nominal data transfer rate is 180 K bytes per second. Data transfers in block sizes from 256 bytes to 12,288 bytes are made possible by the controller's ability to cross sector and head boundaries. Simultaneous seek and over-lapping seek/data transfers are permitted in multiple disc drive configurations to minimize access time. Extensive hardware error checking by the controller allows complete data transfer monitoring for use in error detection and recovery programs.

8.5.2 Operational Characteristics

The disc drive assemblies are structured around a rigid base plate resting on three resilient shock mounts which provides for their mounting. The spindle/blower is driven by a DC motor integrally mounted on the spindle shaft. Rotation speed of the motor is servo controlled, from a crystal oscillator to within $\pm 1\%$ of 1500 rpm. This control is independent of variations in line frequency or voltage. Brushes on the DC motor are sealed within the motor housing and have a life expectancy in excess of 10,000 hours. The blower is integrally mounted on the spindle. Incoming air passes through an Absolute Filter (retaining 99.97% of particles over 0.3 microns) and is forced directly into the cartridge, cleaning it and maintaining a positive pressure within the drive. The two read/write heads are positioned over the desired track under control of an electronic servo. While positioning, this servo counts its way to the track and adjusts the speed of the motion dependent on the number of tracks still to be moved. Once in position, the same servo holds the heads over the track. Positioning under this method is extremely precise, guaranteeing complete interchangeability of cartridges between drives, if desired.

The disc drive controller can be functionally divided into three parts: the computer interface, including address decoding; the drive controls which are concerned with drive status and head position; and the data transfer portion which is concerned with writing and reading.

The controller responds to five different addresses, one assigned to each of the four disc drives plus its own, and transfers data to the last disc accessed. Interrupts from any of the discs or the controllers are queued and respond with the proper interrupt address for the interrupt source.

The disc controller portion may be regarded as four separate controllers, except that only one may be addressed at one time. The disc controller accepts commands and responds with the status of the disc drive addressed when issued a Sense Status instruction. The data transfer portion of the controller receives commands initiating requests for Read, Read Check, and Write. It also responds to Sense Status instructions by issuing status conditions of the addressed disc drive.

8.5.3 Disc Format

The 2315 type cartridge contains a single disc with data recorded on both surfaces. See Figure 8-5 for disc format diagram. Since all tracks having the same number are vertically aligned, the cartridge may be considered to be organized into 200 cylinders. Each cylinder contains two tracks having the same track number. The heads are physically positioned to the same track number on both surfaces of the disc allowing data to be accessed by cylinders to minimize access time.

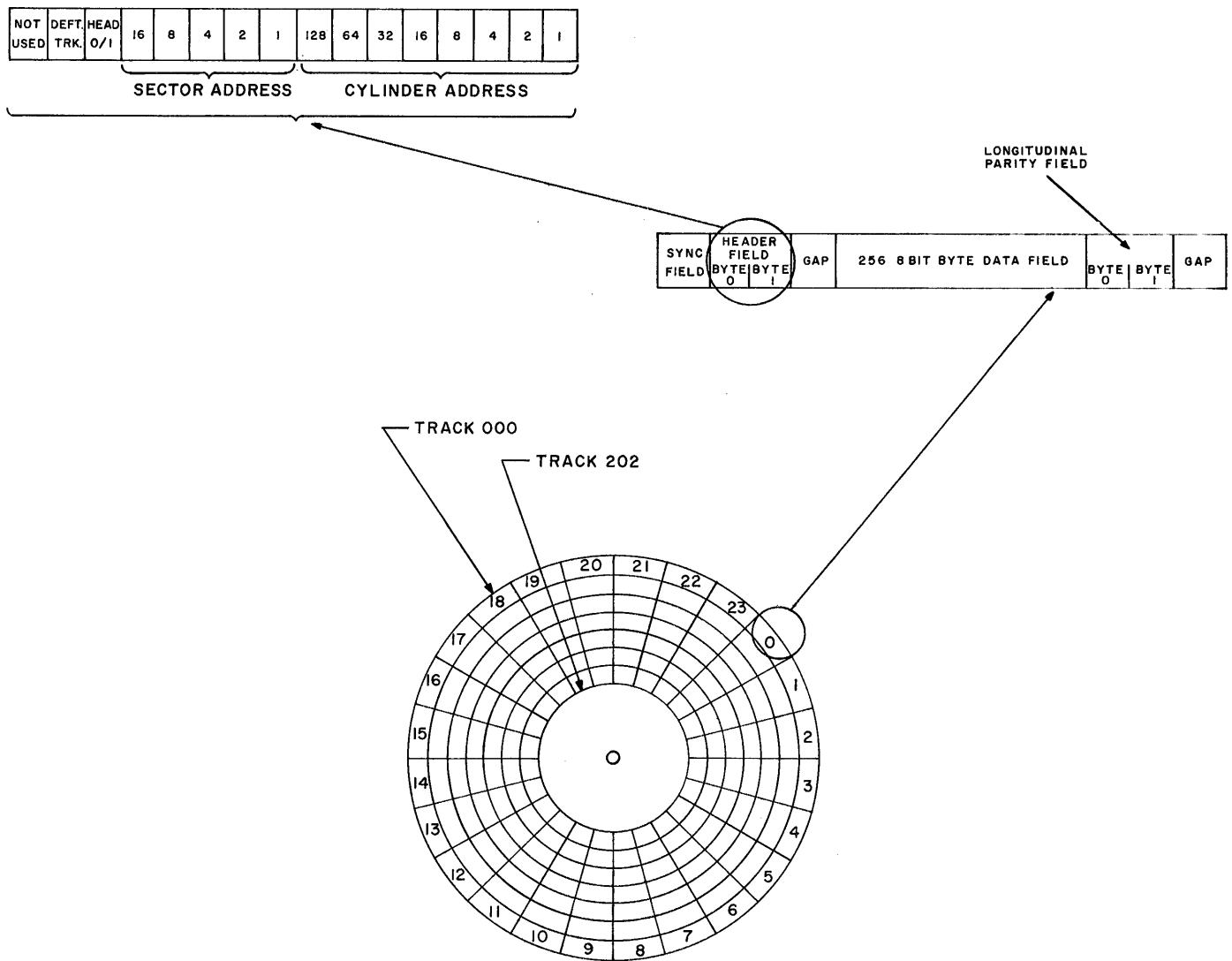


Figure 8-5. Removable Cartridge Disc Format

The data on each track is subdivided into 24 equal sectors. Each sector is divided into four main fields: a Sync Field, a two byte Header Field, 256 byte Data Field and a 16 bit Longitudinal Parity Field. The Header Field contains the cylinder, sector and head address as well as a defective track flag which was recorded on the disk when a surface analysis program was run. The hardware tests the Header Field prior to commencing a data transfer. If a defective track or an address mismatch is detected, the controller aborts the data transfer. The 16 bit Longitudinal Parity Field is added at the end of the data field by the controller during a Write operation and is read at the end of a Read operation to ensure data reliability.

8.5.4 Data Transfers

The normal data transfer commands are Read, Read Check, and Write. Two special commands, Read Format and Write Format, are used only for the operation of disc formatting. Any sector on any track may be accessed by execution of the following general sequence of commands. It is assumed that the heads of the disc file of interest are at the proper cylinder.

To cause data transfers to occur:

1. Sense status of the Selector Channel. If not busy, then
2. Sense status of the data controller. If flag "Controller Idle" is a one, the controller is not busy, then
3. Sense status of the file. If all status bits are zero except for possibly the write protect flag, data may be transferred to or from the disc.
4. Load the Selector Channel with initial and final addresses.
5. Write data giving the cylinder address. This operation selects the file and gives the controller the cylinder address it should find recorded in the Header Field of the disc.
6. Write data to the controller, giving the sector address and the head number.
7. Output Command to the controller, giving the operation required. The operation will begin when the correct sector is under the head.
8. Output Command to start the Selector Channel.

Upon termination of a data transfer the Selector Channel generates an interrupt to the Processor and other disk activities may now begin.

8.5.5 Specifications

Operational Characteristics for Removal Cartridge Disc Drive System.

Data Storage	Uses IBM 2315 Cartridge or approved equivalent: double frequency recording.
	Bit Density 2200 BPI
	Track Density 100 TPI
	Nominal Transfer Rate 180K Bytes per second
Capacity	Per Cartridge 2,457,600 Data Bytes
	Track 6,144 Data Bytes
	Cylinder 12,288 Data Bytes
	Sector 256 Data Bytes
	Sectors per Track 24
	Tracks per Cylinder 2
	Sectors per Cartridge 9,744
	Tracks per Cartridge 406 (including 6 spares)
	Cylinders per Cartridge 203 (including 3 spares)

Access Time	Rotation	1,500 RPM \pm 1%
	Average Latency Time	20 msec
	Head Positioning including settling time	
	Maximum Track to Track	15 msec
	Average	70 msec

Dimensions Disc Drive	Height	6 1/2 inches
	Width	17 1/2 inches
	Depth	22 7/8 inches

This unit is available for either rack mounting in a standard 19 inch RETMA Rack or placement on a table top.

Weight Approximately 40 pounds

Power	Voltage Required	115 VAC \pm 10% 60 cycle \pm 1Hz or 220 VAC, 50Hz (with modification on power supply transformer wiring).
	Input Current (Max.)	7 Amp.

Dimensions Disc Drive Controller Boards

10.5" x 9.75"

Weight 1.5 pounds each

Power 3.0 amps of +5 volts D.C.

Options

- M46-410 Removable Cartridge Disc System (Rack Mounting version). 2.5 million bytes. Includes 1 x 4 controller with first disc drive. For expansion,
- M46-411 Removable Cartridge Disc Drive Expansion 2.5 million bytes (Rack Mounting version).
- M46-412 Removable and Fixed Cartridge Disc System (Rack Mounting version) 5.0 million bytes. Includes 1 x 4 controller with first drive.
- M46-413 Removable and Fixed Cartridge Disc Drive Expansion 5.0 million bytes (Rack Mounting version).
- M46-420 Removable Cartridge Disc interface, with cable.
- M49-023 Disc Expansion power supply.

For additional information, see the Cartridge Disc Controller Instruction Manual, Publication Number B29-254.

8.6 201 SYNCHRONOUS DATA SET ADAPTER

8.6.1 General Description

The 201 Synchronous Data Set Adapter is a double buffered character interface and controller to any Bell 201 Data Set or equivalent. It provides a telecommunication link between the Processor and the data set for common carrier switched or leased lines. Either half duplex (2 wire) or full duplex (4 wire) operation can be accommodated. The use of synchronous modems provide a higher baud rate than may be realized with asynchronous modems over a voice grade facility, thus allowing high speed terminals to transmit data in a more efficient manner. Provisions via hardware options provide a variety of variable functions for the user. These options include such functions as "sync" character detect, character length, parity and half or full duplex modes. Special character recognition (other than Sync), block character checking and generation and code translation is under program control. See Figure 8-6.

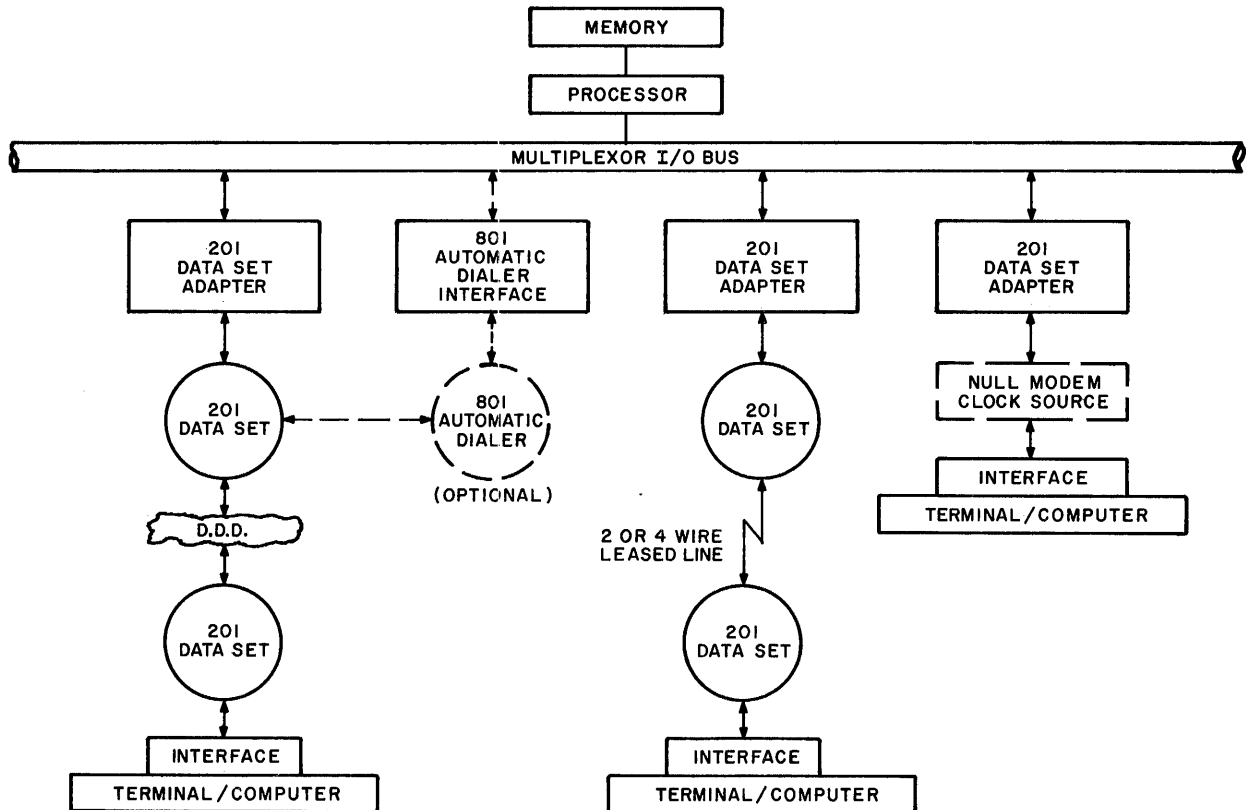


Figure 8-6. 201 Synchronous Data Set Adapter Block Diagram

8.6.2 Operational Characteristics

The 201 Synchronous Data Set Adapter consists of a single board controller which performs three (3) general functions.

1. Data Set Interface - Provides control lines to a synchronous modem which is Bell 201 compatible. Full data set control to and from the modem is available.
2. I/O Interface - Contains the necessary logic to communicate with the Multiplexor Bus of an INTERDATA Processor.
3. Character Control - Provides serial to parallel and parallel to serial conversion, appends and tests parity, provides a match between the serial data stream and the strapped Synch character and automatically switches to the synchronized mode when a match is detected, provides an interrupt for error conditions in addition to the normal End of Character interrupt.

The transmission is serial synchronous by character and bit utilizing a hard-wired "Sync" character for character synchronization. Any six, seven, or eight bit character code may be used. Operation is half or full duplex. All turn-around timing, control-line coordination and status reporting is automatically handled by the adapter.

Data Terminal Ready and Sync Search are controlled by the program for automatic call reception, disconnect, and lockout. The 201 Adapter may also be used in conjunction with an 801 Automatic Dialer Interface for call origination. Ring, Data Carrier and Data Set Ready are brought into the adapter to provide full Data Set status information. The adapter may operate in the interrupt mode to provide maximum program efficiency.

8.6.3 Specifications

The 201 Synchronous Data Set Adapter is contained on a single printed circuit board.

Dimensions	14.88" x 15.38"
Weight	1.5 pounds
Power	+5V at 2.0A -15V at 0.05A +15V at 0.05A
Data Set Interface	RS-232 compatible using DIP, with wide hysteresis and line filters to improve noise immunity.
Mode	Half or Full Duplex (strap option)
Transmission Speed	Up to 9600 baud. Clocking source within the modem
Character Format (strap option)	
-Character Size	Six, Seven, or Eight bits
-Parity*	Odd, Even, None
Synch Character Recognition	Strap option which may include parity, if equipped.
Data Set Status	(generates interrupt if enabled)
-Carrier	
-Data Set Ready	
-Ring	
Other Status	(set at end of character)
-Parity Fail	
-Sync Character Detect	
-Send and Receive Overflow	
Data Set Control	
-Request to Send (Half-Duplex Mode)	
-Data Terminal Ready	

INTERDATA Product Number M47-000

Note that the above data set adapters do not include cables to the data set.

Cable 10-054 Data Set cable, 50 feet

*Parity may be enabled/disabled with a strap option. When parity is enabled, odd or even parity is selectable under program control.

NOTE

There are various options and models available on Western Electric or equivalent 201 Type Data Sets. The user should insure that the options or model he selects are compatible with the selected INTERDATA Adapter and that he understands the programming ramifications if any. The user may obtain information regarding the data set and network from his local telephone company representative or from the appropriate manuals supplied by the Bell System or the Data Set Manufacturer.

8.7 PROGRAMMABLE ASYNCHRONOUS LINE SYSTEM (PALS)

8.7.1 Introduction

The PALS provides an interface between a Multiplexor Bus or Selector Channel and a variety of asynchronous data sets in either the Half-Duplex (HDX) or Full-Duplex (FDX) Mode. See Figure 8-7. Because of the wide variety of data sets and terminals available, the potential user must determine the suitability of the PALS for his given application and network.

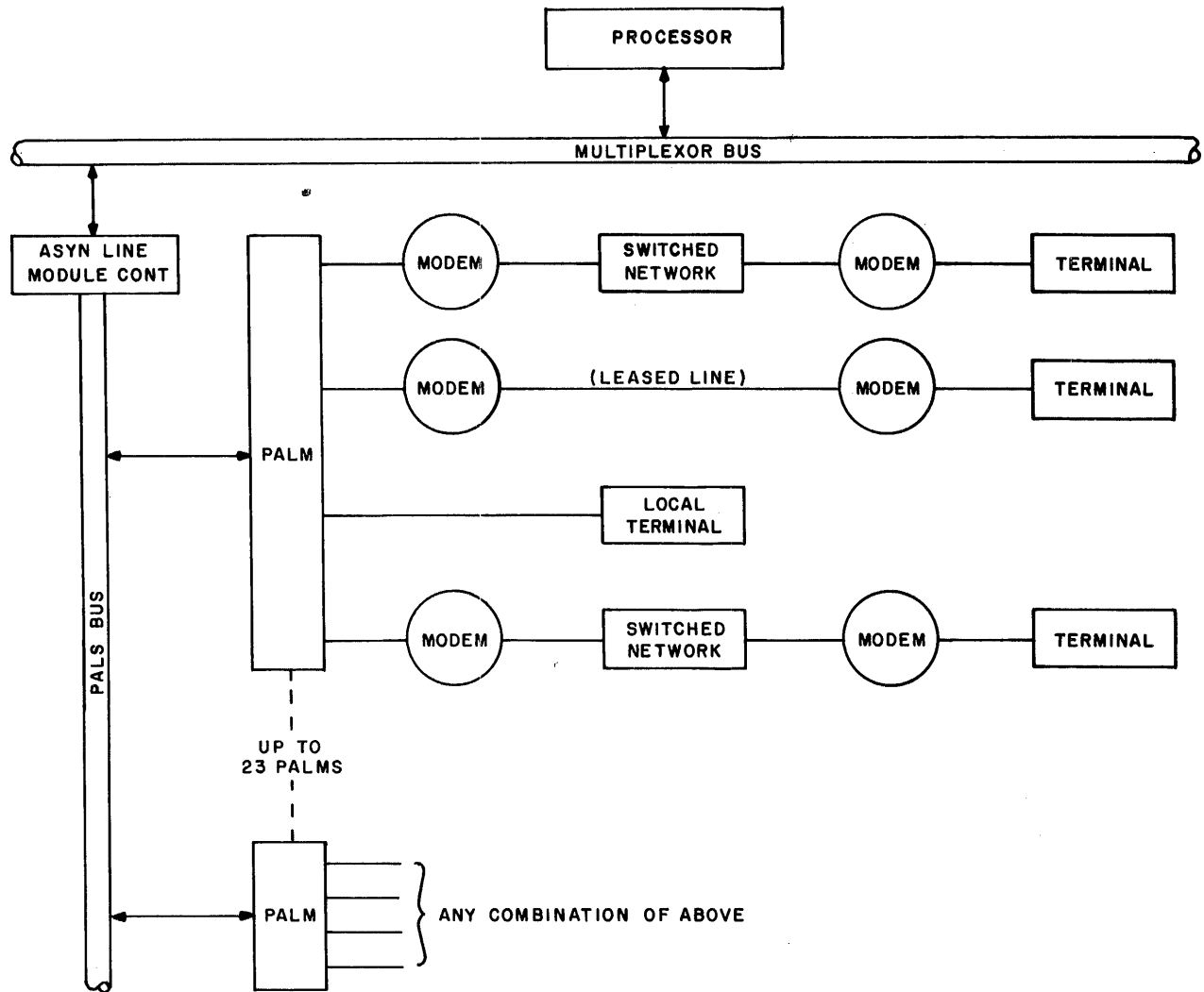


Figure 8-7. PALS Block Diagram

A basic system consists of an Asynchronous Line Module Controller board (ALMC) and one Programmable Asynchronous Line Module (PALM). One PALM contains four FDX or HDX lines. The system may be expanded by adding one PALM for each additional four lines. Each line has two consecutive addresses, an even address for the Receive side and an odd address for the Transmit side. There is an interrupt flip-flop associated with each side.

The ALMC provides eight standard baud rates and optionally four customer specified baud rates. Any of these baud rates may be used by all PALMs in a system.

Each HDX or FDX line can be programmed to adapt the character format and baud rate to a wide variety of data sets and their associated terminals.

8.7.2 Data Format

8.7.2.1 Specifications

The following is a list of the salient PALS specifications:

1. Baud Rates Available - The following standard baud rates are provided by the ALMC: 75, 110, 134.49, 150, 300, 600, 1200, and 1800 baud. In addition, four other baud rates which have the relationship N , $N/2$, $N/4$, and $N/8$ can be provided with a special wiring option.
2. Baud Rates Equipped - Any four of the above may be provided as a customer option.
3. Maximum Number of Lines - Up to 92 lines per ALMC; any combination of HDX/FDX.
4. Character Format (under program control).
 - a. Character Size - 5, 6, 7, or 8 data bits
 - b. Parity - Odd, even, or none
 - c. Stop Bits - One or two
5. RS-232C Interface
6. Data Set Control (Programmable)
 - a. Data Terminal Ready (CD) - Program control is provided over CD to provide for automatic call reception, disconnect, and lockout.

NOTE

Parenthesis indicate RS-232C designation for indicated functions.

- b. Reverse Channel Transmit (SA)* - Permits a supervisory signal to be transmitted over a secondary data path while simultaneously receiving data.
 - c. Request to Send (CA) - Active to maintain the Adapter in the Transmit Mode. In HDX operation, the inactive state maintains the adapter in the Receive Mode.
 - d. Data Terminal Busy* - Enables the "Make Busy" feature when active.
7. Data Set Status - The following lines from the data set effect the status bits: CLEAR TO SEND (CB), CARRIER (CF), RING (CE), REVERSE CHANNEL RECEIVE (SB), and DATA SET READY (CC).

*Optional features in some data sets.

8. Echoplex - A programmable feature to transmit received data back to the data set in addition to assembling the character.
9. Other Features - The PALS provides a double-buffered character to permit a full character "grab time". The Start bit is automatically generated and transmitted by the hardware and, in the Receive Mode, the Start bit must be present for at least one half bit time before the character assembly commences, thereby reducing the noise susceptibility of the system.
10. Method of Transmission - Serial, asynchronous by character, synchronous by bit.
11. Distortion:

Transmit - The transmit data distortion shall be $\leq \pm 3\%$ per character.

Receive - The PALM adjusts the data sampling strobe with each character received and tolerates a data bit distortion of $\pm 43\%$. In addition, the long term transmission rate may vary by $\pm 5\%$.

8.7.2.2 Transfer Format

Asynchronous operation requires that all characters be preceded by one Start bit and have one or two Stop bits appended after the last bit or the parity bit, if selected. The Start and Stop bits serve to delineate characters. A typical format for the Teletypewriter is shown in Figure 8-8.

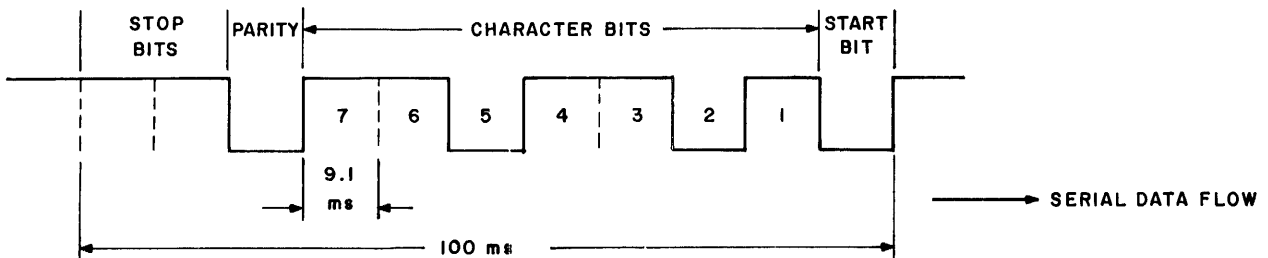


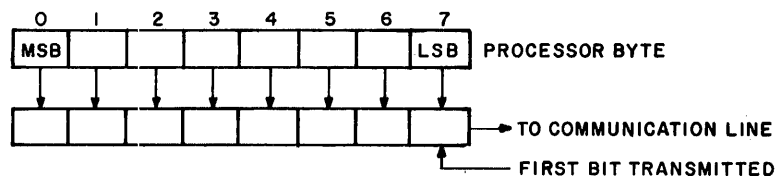
Figure 8-8. Typical ASCII Character Format

Note that to send seven useful bits of information, eleven code elements (bits) are required. Therefore, to send useful information at 70 bits/second, the system must operate at 110 bits/second.

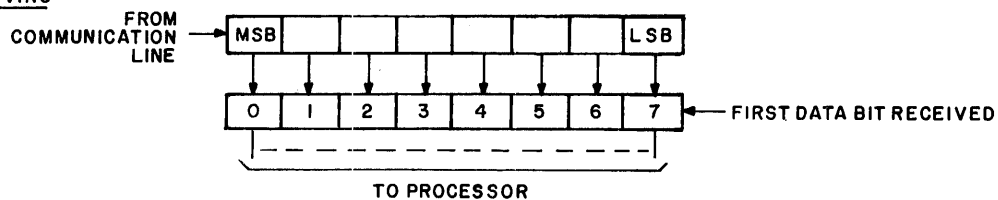
The single Start bit is generated by the hardware, and the character size/parity and number of Stop bits are under program control.

Order of Transmission

TRANSMITTING



RECEIVING



8.7.3 Programming Instructions

The Processor I/O instructions are used to communicate with the PALM. The following paragraphs describe how Processor I/O instructions may be used with the system.

8.7.3.1 PALM Program Instructions

1. Sense Status (SS or SSR). The Sense Status instruction is used to determine if character transfers are complete and correct, and to interrogate the associated data set status.
2. Output Command (OC or OCR). The Output Command instruction is used to answer or disconnect calls, to set the PALM in the Receive or Transmit Mode, and to select the character format. Two Command bytes are required to perform these functions.
3. Write Data (WD or WDR). The Write Data instruction is used to load the output character into the Data Register.
4. Read Data (RD or RDR). The Read Data instruction is used to read an assembled character into the Processor.
5. Acknowledge Interrupt (AI or AIR). The Acknowledge Interrupt instruction is used to service interrupts. Execution of this instruction returns the address and status of an interrupting line.
6. Communications Instructions. The PALM accommodates the Communications instructions in the Communications Processors.

8.7.3.2 Status and Command Bytes

Table 8-7 contains the PALM Status and Command Byte Data.

TABLE 8-7. PALM STATUS AND COMMAND BYTE DATA

INSTRUCTION		BIT NUMBER							
		0	1	2	3	4	5	6	7
PALM STATUS		OV	PF or CL2S	FR ERR	RCR	BSY	EX	CARR OFF	RING
PALM COMMAND 1	RCV	DIS	EN	DTR	ECHO- PLEX	RCT or DTB	TRANS LB	WRT or RD	1
	SND	DIS	EN						
PALM COMMAND 2		CLK SEL		BIT SEL		STOP BIT	PARITY		0

PALM STATUS

OV* The Overflow status bit is set if the previously received character is not read before the present character is assembled. Double-character buffering in the PALM permits a full-character "grab-time". The Overflow status can only become active in the Receive side and is reset at the next end of character only if the failure condition disappears (i. e., cleared by a Read Data instruction). The Overflow character is assembled.

PF* In the Read Mode, this bit is set when the received parity disagrees with the programmed parity. If parity is not selected via an Output Command, this bit remains inactive. The PF status bit is reset at the end of the next character if the failure condition disappears.

CL2S

The lack of Clear to Send signifies that the modem can no longer transmit data. In the Write Mode, this status bit set indicates that Clear to Send (CB) is not being received from the data set. This condition also forces BSY=1. In the Read Mode, this bit is forced inactive. A Transition from CL2S=0 to CL2S=1 will cause an interrupt.

FR ERR*

Framing Error is active to indicate that the received character has no Stop bit(s). That is, the line is in the SPACE state instead of the MARK state at Stop bit time. If the character has two Stop bits, only the first is tested, and the character assembly terminates after the first Stop bit. In the case of a Framing Error, the character is assembled. A non-zero character can signify an illegal character. An all zero bit character with Framing Error can signify the beginning of a line break sequence. In the case of a line break (prolonged space), if the line remains spacing, only the first character is assembled. Subsequent space characters are not assembled until a MARK to SPACE transition is received. Not that because of this characteristic where the line break facility is being employed, a line break decision must be based on the receipt of a single zero character with Framing Error.

RCR

REVERSE CHANNEL RECEIVE (SB). This is an option in some half-duplex data sets (e. g., 202C). This status bit is active if a data set equipped with this option has this line in the SPACE condition. If equipped, this status bit represents the present state of the equivalent data set lead. If the data set does not have the Reverse Channel option, this status bit is always inactive. Any transition on this signal causes an interrupt, if enabled.

BSY

The dropping of BSY (and related interrupt) normally signifies that the adapter is able to transfer data to/from the Processor. This bit is forced active (BSY=1) if DATA SET READY (CC) from the data set is OFF. This indicates that the data set is not able to handle digital data because of an abnormal condition such as being in the Talk or Test Mode. In addition to the above, in the Read Mode, BSY is active when a character is not assembled; and in the Write Mode, BSY is active if CLEAR TO SEND (CB)=0, or if the interface cannot accept another character for transfer. An interrupt is generated, if enabled, when BSY goes inactive.

In the Read Mode, when an OV occurs, the BSY status bit is zero and a Read Data instruction must be issued to set the Busy bit to its correct (ONE) state.

EX

EXAMINE=OV+PF+DATA SET READY+FRERR. This bit is disabled in FDX on the Write side.

CARR OFF

The CARRIER lead off is an indication that no valid incoming data is being received. In the Receive side, this bit is active to indicate that CARRIER (CF) is not being received from the data set. In the Write Mode, this status bit is forced inactive when REQUEST TO SEND (CA) is active. A transition of this status bit in either direction causes an interrupt, if enabled.

RING

This status bit is active when the RING (CE) lead from the data set is active thus indicating the reception of a call. An interrupt is generated, if enabled, when RING (CE) goes active. In FDX operation, RING (CE) is always forced inactive on the Transmit (Send) side. The Ring status represents the present state of the equivalent data set lead.

*These status bits are set at End of Character time when the BUSY bit is reset. Since the resetting of BUSY causes an interrupt (if enabled), these bits do not generate individual interrupts.

PALM COMMANDS

In the PALM Command 1, the DTR, ECHOPLEX, RCT/DTB, TRANS LB, and WRT/RD bits are shared by the Transmitter and Receiver, however, the EN/DIS bits are separate for Transmit and Receive.

PALM Command 2 is shared by both the Transmit and Receive sides, consequently the command may be issued to either address.

In HDX operation, the unused side has the interrupts disarmed. In FDX operation, these bits must be independently programmed as follows. To change EN/DIS on the Receive side, issue a Command with the WRT/RD bit=0. To change the EN/DIS on the Transmit side, issue a Command with the WRT/RD bit=1.

<u>DISABLE</u>	<u>ENABLE</u>	
0	0	No change
0	1	Enable
1	0	Disable (Interrupt queued)
1	1	Complement (Change state)

DTR DATA TERMINAL READY (CD) to the data set. When this command bit is active, it causes CD to be turned ON, allowing automatic answering of an incoming call. This line must be ON to permit the data set to enter and remain in the data mode. When this bit is OFF, it does not permit automatic answering of an incoming call and causes an existing connection to disconnect if held OFF for a period specified by the manufacturer of the data set.

**ECHO-
PLEX** When this bit is active, it causes data received from the data set to be transmitted back to the data set on the TRANSMITTED DATA (BA) line. The PALM also assembles the character as in the normal data mode. This feature, if used, is normally used for HDX operation in the Read Mode to provide visual verification of the data received by the remote computer. This command must not be issued while transmitting a character.

RCT/DTB RCT (Reverse Channel Transmit) (SA) is optional on 202C-type data sets, DTB (Data Terminal Busy) is optional on 103-type data sets. If Reverse Channel equipped, this command bit is gated directly to the Secondary Transmitted Data (SA) line. The inactive state of this bit must be equal to ONE to satisfy the RS-232C requirements, that is, the bit set will cause a MARK state to be gated on the reverse channel. The bit reset will cause a SPACE to be transmitted on the reverse channel. For data sets equipped with the Data Terminal Busy option, the active condition of this command bit will "busy-out" the terminal thereby not allowing a call to be answered and returning the busy signal to the calling terminal.

**TRANS
LB** Transmits a continuous space to the data set. This condition overrides the Echoplex feature. If this command is issued while data is being transmitted, the transmitted data will be mutilated.

WRT/RD This command bit controls REQUEST TO SEND (CA) to the data set. When this bit is programmed active, REQUEST TO SEND is gated to the data set if DATA SET READY* (CC) is active. When this bit is programmed inactive, the hardware deactivates REQUEST TO SEND (CA) after the following delays: If character transfers are in progress, the hardware insures that the last character has been transmitted, then delays one millisecond to permit the last data bit to clear the data set before dropping REQUEST TO SEND (CA). BSY is forced active during this line turn-around and does not drop until a character is received. However, CL2S, CARR OFF, RING, RCR, and DSRDY may still generate interrupts, if enabled. See Figures 8-9 and 8-10.

*DATA SET READY (CC) is always forced active on the Transmit side in FDX operation.

ANSWER CALL HDX MODE (202-TYPE DATA SET)

TO D.S.		FROM DATA SET					INTERRUPT CONDITION	STATUS	COMMENTS
DTR	RQ2S	RING	CARR	DSRDY	CL2S				
							X'0E'	IDLE STATE	
		RING → I					X'0F'	} RING CONTINUES UNTIL DTR IS ACTIVATED	
		RING → I					X'0F'		
							X'0E'	DTR ANSWERS CALL	
							X'0A'	DSRDY INDICATES THAT THE DATA SET IS ON LINE	
			CARROFF → O				X'08'	AT THIS TIME THE COMMUNICATIONS LINK IS ESTABLISHED FOR RECEIVE MODE, BSY DROPS WHEN FIRST CHARACTER IS RECEIVED	
								IF ADAPTER IS TO TRANSMIT FIRST, CONTINUE	
						CL2S=0	X'48'	COMMAND RQ2S=1. CL2S STATUS IS ENABLED IN TRANSMIT MODE	
						CL2S → I	0	THE ADAPTER MAY TRANSMIT WHEN CL2S=1.	

- NOTE 1: CL2S IS DE-ACTIVATED WHEN RQ2S=0
 NOTE 2: CARR IS DE-ACTIVATED WHEN RQ2S=1 (CARR OFF=0)
 NOTE 3: THE STATUS SHOWN ONLY REFLECTS THE LINE CONDITIONS, NOT PF, OV, OR FR ERR. IN ADDITION REVERSE CHANNEL IS IGNORED SINCE THIS IS SUBJECT TO USER'S PROTOCOL.
 NOTE 4: IF DTR IS SET WHEN RING OCCURS, THE CALL WILL BE ANSWERED AUTOMATICALLY (RING TERMINATES AND DSRDY BECOMES ACTIVE).

(A)

ANSWER CALL FDX MODE (103-TYPE DATA SET)

TO D.S.		FROM DATA SET					INTERRUPT CONDITION	STATUS	COMMENTS
DTR	RQ2S	RING	CARR	DSRDY	CL2S				
							X'48' (SND) X'0F' (RCV)	} IN FDX, RING INTERRUPT AND STATUS IS GENERATED ON RECEIVE SIDE ONLY	
		RING → I					X'48' (SND) X'0F' (RCV)		
							X'48' (SND) X'0E' (RCV)	COMMAND DTR ANSWERS CALL	
							X'48' (SND) X'0A' (RCV)	DSRDY ACTIVE	
			CARROFF → O				X'48' (SND) X'08' (RCV)	INTERRUPT ON RECEIVE SIDE	
							X'01' (SND)	INTERRUPT ON SEND SIDE. THE ADAPTER MAY TRANSMIT NOW. CHARACTER ASSEMBLY MAY COMMENCE SHORTLY AFTER CL2S=1.	
						CL2S → I	X'08' (RCV)		

- NOTE 1: CARR OFF AND DSRDY FORCED LOW ON TRANSMIT SIDE.
 NOTE 2: REVERSE CHANNEL NOT APPLICABLE IN FDX (RCR=0)
 NOTE 3: THE STATUS SHOWN ONLY REFLECTS THE LINE CONDITIONS, NOT PF, OV, OR FR ERR.

(B)

Figure 8-9. Answering Calls HDX and FDX

LINE TURN-AROUND READ-WRITE (202-TYPE DATA SET)

	TO D.S.		FROM DATA SET		INTERRUPT CONDITION	STATUS	COMMENTS
	DTR	RQ2S	RING	CARR			
RECEIVE MODE	█	█	█	█	BSY → 0 (RCV)	0 (RCV)	RECEIVE CHARACTER INTERRUPT
						X'08' (RCV)	LAST READ DATA SETS RCV BSY
						X'08' (RCV)	COMMAND ADAPTER TO TRANSMIT MODE (RQ2S IS ACTIVATED 250 μs AFTER THE COMMAND)
						X'48' (SND)	WITH RQ2S SET, CL2S STATUS IS ENABLED.
TRANSMIT MODE	█	█	█	█	CL2S → 1	∅ (SND)	CL2S=1 INTERRUPT. USER MAY NOW OUTPUT DATA.

- NOTE 1: INITIAL CONDITIONS: CONNECTION ESTABLISHED AND ADAPTER IS RECEIVING CHARACTERS.
 NOTE 2: CARR OFF STATUS FORCED INACTIVE AND CARRIER INTERRUPTS INHIBITED.
 NOTE 3: CL2S STATUS FORCED INACTIVE AND CL2S INTERRUPTS INHIBITED.
 NOTE 4: THE STATUS SHOWN ONLY REFLECTS THE LINE CONDITIONS, NOT PF, OV, OR FR ERR. IN ADDITION REVERSE CHANNEL IS IGNORED SINCE THIS IS SUBJECT TO USER'S PROTOCOL.

(A)

LINE TURN-AROUND WRITE-READ (202-TYPE DATA SET)

	TO D.S.		FROM DATA SET		INTERRUPT CONDITION	STATUS	COMMENTS
	DTR	RQ2S	RING	CARR			
TRANSMIT MODE	█	█	█	█	BSY → 0 (SND)	∅	TRANSMIT CHARACTER INTERRUPT
						X'08'	LAST WRITE DATA SETS BSY. AT THIS TIME TWO CHARACTERS ARE QUEUED IN THE ADAPTER.
						X'08'	COMMAND ADAPTER TO RECEIVE MODE. BSY REMAINS ACTIVE UNTIL A CHARACTER IS RECEIVED. DEACTIVATION OF RQ2S IS DELAYED UNTIL 1 ms AFTER THE LAST CHARACTER IS TRANSMITTED.
						X'08'	INHIBIT IS REMOVED FROM CARR WHICH CAUSES AN INTERRUPT WHEN RQ2S → 0. NOTE 3
RECEIVE MODE	█	█	█	█	CARROFF → 0	X'08'	LINE TURN-AROUND COMPLETE
					BSY → 0 (RCV)	∅	FIRST RECEIVE CHARACTER INTERRUPT

- NOTE 1: INITIAL CONDITIONS: CONNECTION IS ESTABLISHED AND ADAPTER IS TRANSMITTING CHARACTERS.
 NOTE 2: CARR OFF STATUS FORCED INACTIVE AND CARRIER INTERRUPTS INHIBITED.
 NOTE 3: WITH A 103 TYPE DATA SET, CARRIER MAY BE PRESENT CONTINUOUSLY IN WHICH CASE THE CARROFF → 1 INTERRUPT WILL NOT OCCUR AFTER DEACTIVATING RQ2S.
 NOTE 4: CL2S STATUS FORCED INACTIVE AND CL2S INTERRUPTS INHIBITED
 NOTE 5: THE STATUS SHOWN ONLY REFLECTS THE LINE CONDITIONS, NOT PF, OV, OR FR ERR. IN ADDITION REVERSE CHANNEL IS IGNORED SINCE THIS IS SUBJECT TO USER'S PROTOCOL.

(B)

Figure 8-10. Line Turn-Around Read/Write and Write/Read

CLK SEL

CLOCK SELECT enables one of four baud rates.

NOTE

The associated ALMC is wired to customer specifications to provide four of the available eight baud rates to all lines in a system.

BIT POS	0	1	CLOCK
	0	0	CLKA (Lowest Baud Rate)
	0	1	CLKB
	1	0	CLKC
	1	1	CLKD

BIT SEL-BIT SELECT

Select the number of data bits/character

BIT POS	2	3	NO. OF DATA BITS
	0	0	5
	0	1	6
	1	0	7
	1	1	8

If fewer than eight data bits are selected when a Write Data is issued in the Write Mode, the data must be right-justified and unused bits are "Don't Care". In the Read Mode, when a Read Data is issued, the character is presented to the Processor right-justified with unused bits (this includes selected Parity and Stop bits) forced to the zero state.

STOP BIT

0=1 Stop Bits

1=2 Stop Bits

When the line is programmed for two Stop bits, the PALM Transmits both. However, the Receiver only samples the first Stop bit.

PARITY

BIT POS	5	6	PARITY
	1	0	ODD
	1	1	EVEN
	0	X	NONE

In the Write Mode, if parity is enabled (Bit 5=1), the PALM generates and transmits the selected parity.

In the Read Mode, if parity is enabled, the PALM compares the received parity with the selected parity and generates the PF status if a disagreement is detected.

If parity is disabled (Bit 5=0), the hardware ignores parity. When transmitting, the hardware appends a Stop bit after the last data bit and, when receiving, disables the Parity Detection Circuit.

NOTE

The least significant bit of the Command Byte must be a 1 or 0 as indicated to permit the hardware to distinguish between the two commands.

8.7.4 Programming Sequences

8.7.4.1 Switched Line Operation

To originate a call, the operator depresses the TALK key on the data set and dials the desired number. When the call is answered, a carrier is heard (being sent by the data set receiving the call). The operator then depresses the DATA key.

The operator may now hang up and depress the AUTO key to return the equipment to automatic receive following this call. When the DATA key is depressed (the Data Light remains lit for the duration of the call, the EX status bit drops (DATA SET READY)(CC) as does the Carrier-Off status bit. The PALM should be initialized to the Read Mode and thus be interrupted by the receiving Carrier. When Carrier-Off drops, the PALM should be switched to the Write Mode to transmit data. Following the call, both sets (originating and receiving) should be issued a Command Read and DTR to disconnect. This procedure is typical, but not necessary. The user is free to design his own hand-shake sequence. Figures 8-9 and 8-10 show timing sequences for answering calls and line turn-around. The following is offered for general information only. With wide variations between data set characteristics and common carrier procedures, the operating procedures may have to be modified. The user should ensure that the characteristics of the devices connected to the PALS are compatible with the descriptions in this specification.

In Figure 8-9, DTR and RQ2S are initially OFF. The status is X'0E' before RINGING commences. The RING causes an interrupt and a status of X'0F'. The RING status bit is active for the period of the RING from the data set.

When RING drops, the status is X'0E' and another interrupt is generated each time RING \rightarrow 1. RING continues until the program activates DTR to answer the call. Shortly after DTR is programmed ON, the data set responds with DSRDY=1. This causes EX \rightarrow 0 and the status at this time is X'0A' (BSY=1 and CARR OFF=1).

When the data link is established, the data set turns CARR ON which generates an interrupt and a status of X'08' (BSY=1). If the adapter remains in the Receive Mode, Busy stays active until a character is received. If the adapter is to transmit first, the program turns RQ2S ON (Command with the WRT bit active). With RQ2S ON, the CL2S status from the data set is enabled. Since this bit is initially OFF, an interrupt is generated when RQ2S is turned ON and another interrupt is generated when the data set responds with CL2S=1. The adapter may now transmit.

The user must be cautioned that Figures 8-9 and 8-10 assume ideal conditions. For example, in a typical switched network environment, more than one interrupt may be generated as carrier is initially established, or the Received Data from the local data set may be active during a connect or disconnect sequence. These problems can be attributed to the type (manufacturer) of data set employed, the options implemented in the data set and also the switched network. In particular, if the Received Data from the data set is active before carrier is established, the PALM will commence to assemble a "garbage" character. This can result in a Receive Busy interrupt with any or all of the character status bits active (PF, FRERR, OV). These status bits will then remain active until a Read Data is executed (to set Busy and reset OV) and a valid character is received (to reset PF and FRERR).

8.7.4.2 Leased Line Operation

In leased line operation, because a connection is permanently established, no dial-up or disconnect is needed. Both stations are normally initialized to the Read Mode. Either end can originate a transfer by going into the Write Mode, which causes the receiving station to interrupt when the Carrier appears. Upon receiving characters, the receiving end is in the Read Mode, and a data transfer takes place. The exact hand-shake protocol is up to the user.

8.7.4.3 Half-Duplex Operation (202-Type Data Set)

In the Half-Duplex operation, only one terminal can transmit at any one time. To change the direction of transmission, the channel must be turned around. The question arises as to who indicates channel turn-around. The convention is normally held that the Processor turns the line around when it has a message to transmit. Data sets (e.g., 202C type) normally used in Half-Duplex operation, should be equipped with the Reverse Channel option which is used to signal the requirement to reverse the direction of transmission or to break the data flow. An important operating convention affecting Reverse Channel operation results from the presence of echo suppressors in long-distance lines. These suppressors normally disallow transmission of an echo.

In data communications, the echo suppressor must be disabled, as it must be possible to transmit simultaneously in both directions (Main Channel and Reverse Channel). The echo suppressor becomes re-enabled if the tone on the line is absent for a period exceeding 100 milliseconds. To prevent the re-enabling of the echo suppressor, the convention should be adopted that the Reverse Channel is held ON (high) when the main channel is OFF and vice versa. This convention ensures that a tone is on the line at all times.

The Reverse Channel is normally held ON when the Processor is accepting data. The Processor signals its desire to transmit by lowering (OFF) the Reverse Channel and switching to the Write Mode. A program delay should normally be introduced to allow the terminal on the other end to turn its Reverse Channel ON and enable its Read Mode. This delay is a function of the terminal on the other end. This delay can be 200 to 1200 milliseconds. If the device at the other end is set up to indicate through the Reverse Channel signaling that it is ready to receive data, this can be used instead of a program delay. When the Processor is transmitting, a break condition sent from the terminal to signify that the terminal wants to transmit is signified by the Receive Reverse Channel going from ON (high) to OFF (low). The Processor should then raise its Reverse Channel lead high (ON) and transfer to the Read Mode. The interface automatically introduces the necessary time delay before presenting data to the Processor to ensure valid data transfer and not transition noise.

8.7.5 Interrupts

The PALS has interrupt control logic which sequentially scans all interrupt sources in the system. This logic has the following characteristics:

1. It scans all equipped lines in groups of 16 addresses, commencing with the lowest addressable line and incrementing to the others.
2. If an interrupt is detected in a group of 16 addresses, the hardware assigns priority to the line with the lowest address. Note that, in FDX operation, this is always the Receive side.
3. After all interrupts in the group have been serviced, scanning continues with the next group.
4. An interrupt which is queued on a line that has interrupts disabled is not recognized by the scanner. See Table 8-8 for interrupt conditions.
5. If an interrupt is detected by the scanner, the hardware requires that an Interrupt Acknowledge followed by a status request be received; i.e., an AI AIR instruction. Interrupt scanning does not continue until the above instruction is issued. When programming with Automatic I/O, the user must issue a Sense Status. Note that any instruction that issues an interrupt acknowledge and status request as part of the microsequence (such as the AI, Communication Instructions, Selector Channel etc.) will satisfy the above requirements.
6. In HDX operation, when an interrupt is detected on the Transmit or Receive side of a given line, the address of the Receive side (even) is always returned.
7. In HDX operation, the side not being used has interrupts disarmed (not queued).
8. The sequencer takes 1.08 microseconds to scan a group of 16 addresses. In a fully implemented system (92 lines), it takes 13.24 microseconds to scan all lines. This is the maximum delay to interrupt the Processor after an interrupt condition occurs on a PALM.
9. If an interrupt is present on an enabled line, it can become queued in the interrupt scanner even if the interrupt is disabled before it is serviced. This condition can result in an interrupt from a disabled line. Servicing this interrupt clears the Attention flip-flop on this line while disabled.

TABLE 8-8. INTERRUPT CONDITIONS

INT. COND.	HDX	FDX	
		REC	TRANS
RING → 1	X	X	
CARR OFF → 1	X (RD)	X	
CARR OFF → 0	X (RD)	X	
$\overline{\text{RCR}}$ → 1	X	X	
$\overline{\text{RCR}}$ → 0	X	X	
DSRDY → 0	X	X	
*BSY → 0	X	X	X
$\overline{\text{CL2S}}$ → 1	X (WRT)		X

*An interrupt is also generated in HDX operation when going from READ to WRITE mode if CL2S initially=0; i. e., when CL2S goes from 0 to 1 which causes a BSY interrupt.

8.7.6 Initialization

When the Initialize pushbutton on the Display Panel is depressed (or power failure restart sequence), the OV, PF, and FR ERR status bits cannot be guaranteed. Because of this, the programmer should take precautions to ignore these bits on the first interrupt. The PALM is placed in the Disable Mode. The interrupt line may be active upon initialization. For this reason, it is necessary to execute an Interrupt Acknowledge on power up or initialization.

The command bits DTR, ECHOPLEX, RCT, TRANS LB and WRT/RD are reset to their inactive state.

8.7.7 Device Number

The PALS has contiguous addressing with the lowest address $\geq X'10'$. This is a strap option.

The addresses on a PALM are determined by the physical position of the mother-board in the system with the first mother-board having the lowest addresses.

Two consecutive addresses are assigned to each FDX line, with the even address for the Receive side and the odd address for the Transmit side. In HDX operation, each side responds to either address.

In FDX operation, only one Command 2 is required. Command 2 should never be issued while a character transfer is in progress since this may mutilate the character (Transmit or Receive).

NOTE

There are various options and models available on Western Electric or equivalent 103/202 Type Data Sets. The user should insure that the options or model he selects are compatible with the selected INTERDATA Adapter and that he understands the programming ramifications if any. The user may obtain information regarding the data set and network from his local telephone company representative or from the appropriate manuals supplied by the Bell System or the Data Set Manufacturer.

8.8 INTERTAPE CASSETTE SYSTEM

The INTERTAPE Cassette System provides the user with a low cost sequential mass storage device in an environment where fast (millisecond) access times are not a necessity. Two significant advantages result from the use of this system. First the magnetic cassettes provide a convenient input/output medium. Second, the cassettes on line may serve as an extension to main memory for the storage of data and program files. Although they cost about the same as high speed paper tape reader/punch combinations, the INTERTAPE Cassette System is faster, especially on output, and much easier to handle in most applications than paper tape.

The system consists of a 15 inch controller that plugs into a standard INTERDATA chassis and the dual transport which mounts into a standard 19 inch RETMA rack. The dual transport (vs single transport) system enables the user to edit, compress or decompress data, and to increase the reliability of the system. The mechanism is manufactured for INTERDATA by a well known manufacturer of MIL-SPEC tape transports.

The system uses cassettes that conform to ECMA-34 and proposed ANSI X3B/523 standards. Data is recorded on the tape in the form of records separated by inter-record gaps. Groups of records may be separated by a file gap. An ECMA/ANSI standard compatible cassette can be generated under program control. The unit also includes hardware read-after-write check capability increasing greatly the reliability of the cassette system.

8.8.1 Specifications

Number of Cassette Transports	two
Storage Capacity	250,000 bytes/cassette/side 500,000 bytes per cassette 1,000,000 bytes per system
Data Packing Density	800 bits per inch (bpi)
Transfer Rate	1,000 bytes/second peak, to 480 bytes/second in read continuous mode with a record size of 80 bytes
Tape Speed	10 inches per second (ips)
Type of Recording Technique	Phase encoded, 1600 fcpi
Rewind Time	40-70 seconds (300 ft.)
Inter-Record Gap	0.85 inch
End of File	Special hardware generated character in 0.85 inch gap
Error Checks	LRC generation and check, signal drop out detection, hardware read-after-write check
Start Time	105 msec
Stop Time	50 msec
Operating Temperature Range	20 ^o -30 ^o C (Cassette Tape) 5 ^o -48 ^o C (Tape Transport)
Operating Relative Humidity Range	20% to 80%
Record Size	Variable, under program control

Power Requirements	115/220 VAC, 60/50 Hz, 80 Watts
Weight	19 lbs + Power Supply
Controls	Common POWER ON switch, ON LINE and REWIND switches for each transport Indicator lights are: ON LINE, REWIND, LOAD POINT, POWER ON, WRITE ENABLE, and Processor SELECT.
Dimensions	7" H x 19" W x 15" D

8.9 AUTOMATIC MEMORY PROTECT CONTROLLER

8.9.1 General Description

The automatic Memory Protect Controller (MPC) permits available memory to be divided into blocks. As any combination from 1 to 64 blocks can be selected, a range of from 512 bytes to 64K bytes of memory can be protected under program control. The Processor is permitted to read data from any core location, but may write data only into unprotected locations. Any attempt to write into a protected block of memory is aborted and the memory protect hardware generates an I/O interrupt as an indication to the Processor that an illegal write was attempted. See Figure 8-11.

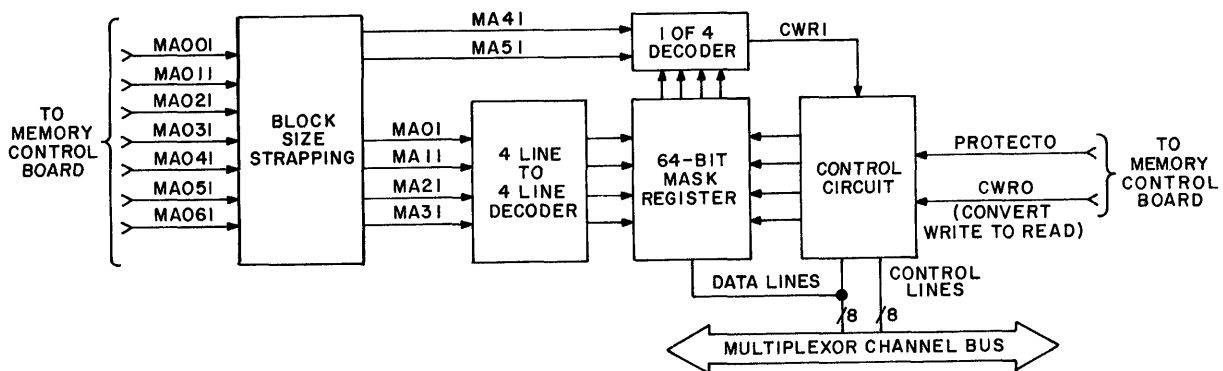


Figure 8-11. Memory Protect Controller Functional Block Diagram

8.9.2 Operational Characteristics

The optional block size of 512, 1,024, or 2,048 bytes per block* is selected by a hardwired strap on the Memory Protect Controller. The block protect pattern is selectable and alterable under program control. The status and command bytes for the Memory Protect Controller are shown below.

Memory Protect Controller status is returned to the Processor upon execution of a Sense Status instruction. The status provides the following indications to the Processor:

PON	Indicates memory protect is active
PWF	Indicates an attempt has been made to write into protected memory area. This bit is reset on an output, or an Acknowledge Interrupt instruction.
EX	Examine is set when PWF is set.

*The automatic MPC used on the Model 80 has a 1,024 byte per block size only.

Output Commands selectively control the protect feature of the Memory Protect Controller as well as its interrupt capability. The output command bits perform the following functions:

- DISARM Disarms interrupt circuit and prohibits queuing of interrupts.
- ARM Arms interrupt circuit and allows for automatic queuing of interrupts.
- PON Enables the protect feature of the controller
- POFF Disables the protect feature of the controller

The following are valid commands for the Memory Protect Controller.

Once an Output Command is given, the protect pattern may be set up with consecutive Write Data instructions. Eight Write Data instructions are required to alter the entire 64 blocks. A ONE in the data byte will cause the corresponding block to become protected memory and a ZERO will cause the block to become unprotected. The relation between the data bytes of the Write Data instructions and the 64 blocks is shown below.

1st	Write data sets up blocks 0-7
2nd	Write data sets up blocks 8-15
3rd	Write data sets up blocks 16-23
4th	Write data sets up blocks 24-31
5th	Write data sets up blocks 32-39
6th	Write data sets up blocks 40-48
7th	Write data sets up blocks 49-56
8th	Write data sets up blocks 57-64

If more than eight Write Data instructions are executed after the Output Command, the protect pattern will wrap around, i. e. , the ninth Write Data instruction will change blocks 0-7, etc.

Initialization preconditions the Memory Protect Controller such that interrupts are disarmed, the PON and PWF flip-flops are cleared and the protect pattern remains unchanged.

8.9.3 Specifications

M70-101 Automatic Memory Protect Controller.

Dimensions	7.5" x 15"
Weight	2 pounds
Number of Blocks	64
Block Size	Standard 1,024 Optional 512, or 2,048 bytes (Model 70 only) (is available for customer wiring)
Protect Pattern	Program Alterable
Preferred Device Address	X'AE'
Power Requirements	+5V at 1.0 amperes

8.10 UNIVERSAL CLOCK MODULE

8.10.1 General Description

The Universal Clock Module consists of two clock devices. The first is a Programmable Precision Interval Clock (PIC) and the second is an AC Line Frequency Derived Clock (LFC). Each of the clocks is completely independent of the other for maximum convenience. Figure 8-12.

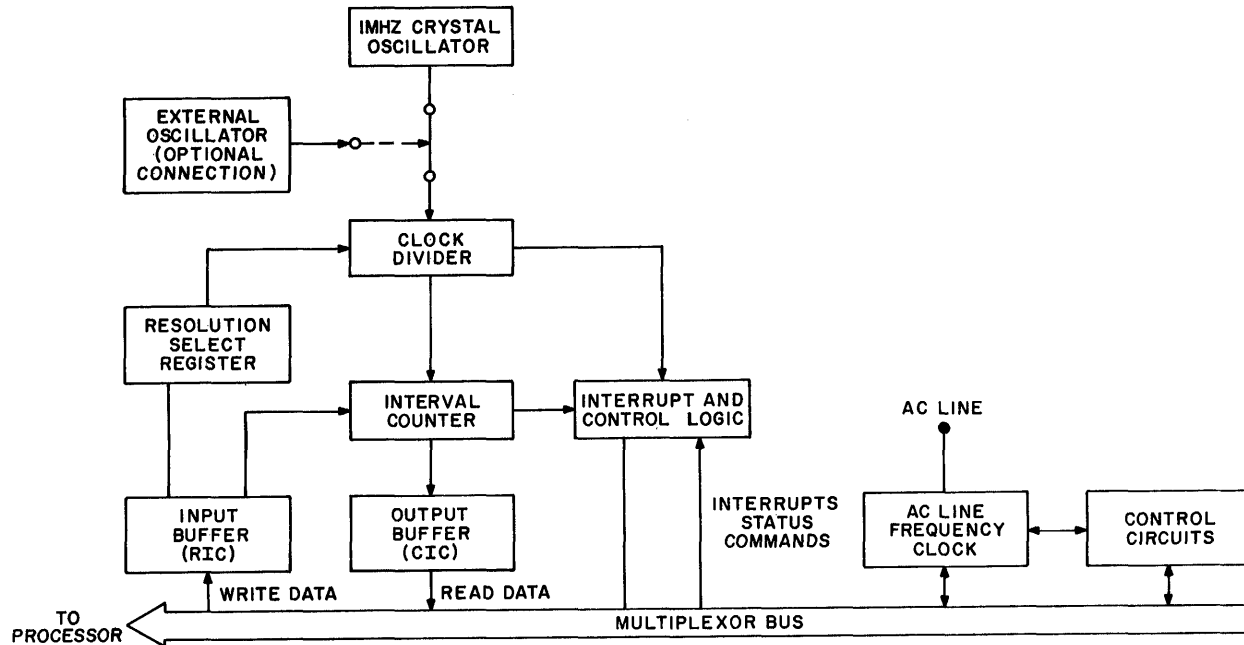


Figure 8-12. Universal Clock Module Block Diagram

8.10.2 Operational Characteristics

PROGRAMMABLE PRECISION INTERVAL CLOCK

The Programmable Precision Interval Clock is dynamically variable through program control. It provides the user with a Processor interrupt and program accessible counter giving resolutions of 1, 10, and 100 microseconds, as well as 1 millisecond through an interval range of 2^{12} . The master time base for the PIC is provided by a one megahertz crystal oscillator which can be disabled so that an external master time base oscillator may be used.

Basic PIC operation is as follows:

1. Resolution and Interval data is sent to the PIC input buffer where it resides until new data is supplied.
2. At the start of the clock through the Command Word, the resolution and interval data in the input buffers are transferred to the Interval Counter and the Resolution Select Register. The Interval Counter begins to decrement at the selected resolution rate.
3. At the conclusion of the interval (Interval Counter equals zero), a program interrupt is generated if enabled, and the Interval Counter and Resolution Select Registers are reloaded from the input buffers. The clock continues to run and once again begins to decrement the Interval Counter. Since the input buffer is not used except after the Interval has completed, the user may reload a new interval anytime before the completion of the present interval.
4. The PIC is provided with an output buffer so that the Interval Counter may be interrogated without disturbing its operation.

The PIC will interrupt at the conclusion of each clock interval, if enabled. It should be noted that the clock does not stop but merely creates an interrupt and restarts using the last selected resolution and interval count.

AC LINE FREQUENCY DERIVED CLOCK

The LFC is derived from the AC power line with a clock rate of twice the line frequency. This clock has not setup procedure other than to enable, disable, or disarm the interrupt circuit.

8.10.3 Specifications

Universal Clock Module.

Power Requirements	+5 VDC \pm 5%, 1 ampere 12 VDC 10 milliamperes
Dimensions	7" x 15 3/8"
Accuracy	PIC - \pm .01% Crystal controlled Oscillator LFC - Line Frequency
Resolution	(PIC) - 1 us, 10 us, 100 us, 1 ms.
Interval	<u>PIC</u> 1 us to 4,095 us, 10 us to 40,950 us, 100 us to 409,500 ms. (corresponding to resolution) <u>LFC</u> 8.33 ms on 60 Hz line 10 ms on 50 Hz line
Program Control	PIC - <u>Control to Clock</u> Command - Disable, Disarm, Enable, Start Status - Overflow Write Data Byte 1, 2, - Resolution and Interval Count <u>Output to Processor</u> Read Data 1, 2, - Current Interval Count LFC - Command - Disable, Enable, Disarm
Product Number	M48-000 Universal Clock Module

8.11 THE EIGHT LINE INTERRUPT MODULE

8.11.1 General Description

The Eight Line Interrupt Module gives the user the capability of generating interrupts in the INTERDATA Processor from his external source. This interrupt module provides a simple, low-cost means of interfacing a variety of alarms, sensors, push button indicators, etc. with the Central Processing Unit (CPU). The utilization of the Eight Line Interrupt Module allows the user to selectively recognize or ignore, on a priority basis, the occurrence of random external events under program control. See Figure 8-13.

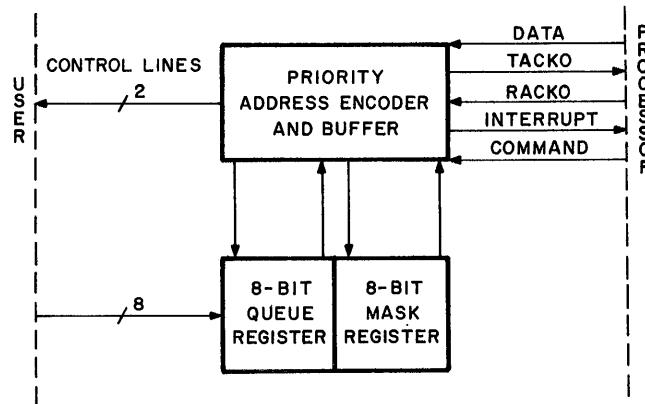


Figure 8-13. Eight Line Interrupt Module Block Diagram

8.11.2 Operational Characteristics

Outstanding features of the Eight Line Interrupt Module are:

- The module is contained on one 7" board.

- Up to eight external interrupt lines may be hardwired in a desired priority.

- The user has the strapping option of edge or level sensing.

- Eight lines to set interrupts and eight lines which indicate interrupt acknowledgement are provided the user.

- Provisions are made so that the lowest priority line can be used to mask off interrupts from lower priority devices on the Multiplexor Bus.

- Two gated command lines are available to the user.

- Each interrupt is reset upon acknowledgement unless strapped for level sensitive operation, in which case the Interrupt Queue Register tracks its input.

- Interrupts may be generated under program control which allow test and diagnostics to take place without the presence of the users external equipment.

The user is provided with eight interrupt lines, each corresponding to a bit (with its own device address) in the Queue Register and 8 more lines, each of which is pulsed, as its corresponding queued interrupt is acknowledged by the Processor. All lines are accessible to the user via a 10 foot open ended twisted pair.

Each bit in the eight bit Queue Register has its own device address. There is also an eight bit Mask Register whose bits enable the recognition of corresponding queued interrupts. The mask bit for the lowest priority interrupt line can be used at the user's option to disable interrupts from lower priority input-output modules on the Multiplexor Bus. The Eight Line Interrupt Module will interrupt the Processor when any bit in the Queue Register is set and its corresponding mask bit is set.

When an Acknowledge Interrupt command is sent by the Processor to this module, the address corresponding to the highest priority interrupt queue flip-flop (whose mask bit was set) is returned to the Processor and the queue bit is reset (with the exception of level sensing). In different command modes, the following actions can take place under program control: Queue Register bits can be reset or set selectively, or in groups; the Mask Register can be loaded with a new pattern to enable or disable selected lines.

8.11.3 Specifications

Eight Line Interrupt Module.

INTERDATA Product Number (M48-001)

References	<u>Eight Line Interrupt Module Instruction Manual,</u> Publication Number 29-288
Dimensions	7" x 15"
Weight	1.0 pounds with cables.
Power Requirements	Voltage - +5 VDC @ .75 amperes Power Consumption - .75 amperes

8.12 SERIAL LINE PRINTER

8.12.1 General Description

The Serial Line Printer offers high performance at low cost making it ideally suited for a diversity of applications. It prints at a rate of 165 characters per second, 10 characters per inch, and up to 132 characters per line. Line printing speed is approximately 60 full lines per minute to 150 short lines per minute. The printer is supplied with an INTERDATA parallel interface.

The Serial Line Printer features pin feed form handling for forms up to 14 3/8 inches in width, a full line buffer, and a two channel vertical format control for top of form and vertical tabulation.

8.12.2 Operational Characteristics

The Serial Printer is a completely self-contained unit which includes the mechanical and electro-mechanical components, control logic, character pattern generator, line buffer and power supply. A matrix method of printing is utilized with a print head consisting of 7 print solenoids mounted on a carrier. Printing is accomplished through selective pulsing of the print wires as the print head moves from left to right across the character space and the print line. The solenoids are activated independently up to five times for any one character. This technique provides enough force to produce a minimum of five legible copies (an original plus four carbons).

Four control switches and three indicator lights are housed on the operator's panel. The four control switches are:

Stop-Start Switch - supplies power to the printer unit in the start position with switch illuminated.

Top of Form Switch - used for manual slewing to top of form.

Select Switch - used to select the printer after turning on power.

Forms Override - provides operator override on the paper out switch allowing the operator to complete the form being printed before changing paper.

The indicator lights are:

Hardware Alarm - indicates head carrier has exceeded right margin.

Paper Out - indicates out of paper or a paper handling malfunction.

Multiple Purpose - used in special applications.

In addition to the alarm indicators, an audio alarm is activated for the out of paper condition, paper handling malfunction, and if the head carrier exceeds the right hand margin. The audio alarm is also sounded when a Bell Code (X'07') is received.

The parallel interface to the Serial Printer is completely contained on a standard INTERDATA 7 inch printed circuit board.

The printer recognizes six non-printing commands as follows:

CR (X'0D')	Causes the printer buffer to print, causes the carriage to return, and causes the form to be advanced one line.
FF (X'0C')	Causes the form to be advanced to the first punch in Channel 1 of the form feed tape.
VT (X'0B')	Causes the form to be advanced to the first punch in Channel 2 of the form feed tape.
LF (X'0A')	Causes the form to be advanced one line.
BELL (X'07')	Activates the audio alarm for 2 seconds.
SO (X'0E')	Causes the buffer to print elongated characters (66 characters per line instead of 132).

When the 132nd character is loaded into the print buffer, an automatic CR is generated (i. e. , the buffer is printed, the carriage is returned, and the form is advanced one line).

8.12.3 Specifications

Dimensions: Width - 27 $\frac{1}{2}$ inches

Height - 11 $\frac{1}{4}$ inches

Depth - 19 $\frac{1}{4}$ inches

Weight: 155 pounds

Print Speed 60 to 150 LPM, 132 columns

Power Requirements: 117 VAC \pm 10% 60 Hz

230 VAC \pm 10% 50 Hz

8.13 LOADER STORAGE UNIT (Model Nos. M70-104, M70-105)

8.13.1 Summary

The Loader Storage Unit is designed to support remote or unattended systems where automatic local or remote reinitialization of the system is required.

8.13.2 Functional Description

The Loader Storage Unit consists of a controller, a hardware watchdog timer, and from one to 16 non-volatile storage modules of 128 bytes each. The whole unit is contained on one 7" x 15" card, has a fixed device assignment, and plugs into the standard INTERDATA Multiplexor Bus.

When entering a restart phase, the Processor checks for the presence of the Loader Storage Unit. If present, the Processor then loads, from the storage modules, information defining the start/end locations in main memory, and the program status word for starting the program once loading is complete. The Loader Storage Unit has a capacity of up to 2048 bytes and comes complete with 16 I/C sockets. Additional Loader Storage Units may be installed if more than 2048 bytes are required, however the first unit must contain the logic to load from successive units.

The hardware watchdog timer feature must be reset under program control periodically or the module will force an automatic restart operation. Depending on a switch setting, this could cause a normal auto-restart or load operation. This watchdog feature may be disabled manually or under program control.

8.13.3 Applicable Processors

Model 74

Model 70

Model 80

} Note: Not all units are micro-programmed to support LCU.

8.13.4 Ordering Procedure

When ordering storage modules for the Load Storage Unit, customers must supply a paper tape and listing with the desired binary load data.

8.14 AUTOMATIC LOADER FOR BASIC MODEL 74 (Model No. M74-101)

8.14.1 Summary

This option is designed to support automatic loading of no-display Model 74s from most input devices.

8.14.2 Functional Description

This module, approximately $2\frac{1}{2}$ " x $4\frac{1}{2}$ ", plugs into the display connector on the Processor board containing the display controller. Sixteen copper straps are contained on the module. A customer may cut these straps to provide a device address and command byte to activate an automatic load sequence in the Model 74. This may load from any suitable standard INTERDATA peripheral (teletypewriter, high speed paper tape reader, a rewound Intertape Cassette or magnetic tape). In general, the automatic loader can support any device which transfers 8 bit data and can be activated by a single output command. Transfer rates of up to 100K bytes per second can be supported.

A simple switch assembly is provided to activate the module and provide basic control functions for a no-display configuration.

8.15 SELECTOR CHANNEL

8.15.1 General Description

The 02-232 Selector Channel (SELCH) controls the transfer of data between I/O devices and memory at rates of up to 2,000,000 bytes per second. Up to 9 I/O devices can be connected to the Selector Channel, but only one device can transfer data at a time. The advantage in using the Selector Channel is that other program processing can occur simultaneously with the transfer of data between the I/O device and memory. This is accomplished by allowing the Selector Channel and the Processor to access memory on a cycle-stealing basis. In some instances, the execution times of the program in progress are affected, while in others, the effect is negligible. This depends upon the rate at which the Selector Channel and Processor both compete for access to memory. Data transfer to the device may be made in either the Byte or Halfword Mode.

8.15.2 Programming Instructions

Table 8-9 illustrates the Selector Channel Status and Command Byte coding. A Sense Status instruction (SS or SSR) is used to transfer the status byte from the Selector Channel Device Controller to the Processor. The least significant four bits (4:7) of the status byte are copied into the Condition Code during the Sense Status operation. Branch instructions can test these four bits directly.

The Output Command instruction (OC or OCR) causes a command byte to be sent to the Selector Channel Controller.

TABLE 8-9. SELECTOR CHANNEL STATUS AND COMMAND BYTE DATA

BIT NUMBER	0	1	2	3	4	6	7
STATUS BYTE					BSY		
COMMAND BYTE			READ	GO	STOP		

STATUS

BSY This bit is set when the Selector Channel is in the process of transferring data.

COMMAND

READ This command changes the mode of the Selector Channel from Write to Read. In the Read Mode, data is transmitted from the active device on the Selector Channel and written into memory. Whenever a data transmission has been completed, the Selector Channel is placed in the Write Mode. Each time a Read operation is required, a Read Command must be issued.

GO This command initiates a data transmission. This command can be issued at the same time the Read/Write Mode is established.

STOP This command halts any data transmission in progress, and initializes the Selector Channel for starting a new operation. It should be given when the Selector Channel terminates.

The Write Data (WD or WDR) or Write Halfword (WH or WHR) instructions may be used to send the starting and final addresses to the Selector Channel Controller.

The Read Data (RD or RDR) or Read Halfword (RH or RHR) instructions may be used to obtain the last Processor memory location either written into or read from memory.

The Write Block (WB or WBR) instruction should not be used since the status byte returned to an idle SELCH, by the WB or WBR instruction, is the status of any active device on the SELCH Bus with Bit-4 (BSY) forced to a zero.

If an interrupt is pending, an Immediate Interrupt or Acknowledge Interrupt instruction (AI or AIR) clears the interrupt and causes the device number of the Selector Channel (X'F0' preferred) and status of the peripheral device to be sent to the Processor.

8.15.3 Programming Sequences

Programming a device on the Selector Channel consists of setting up the device, setting up the Selector Channel, and sending a GO command to the Selector Channel. When all devices on the Selector Channel are idle, the Selector Bus becomes a part of the Multiplexor Bus. This provides the path to set up the device and the Selector Channel.

The last device addressed prior to sending the GO command is the device the Selector Channel controls, assuming that the device is connected to the Selector Channel. The program must, therefore, send the GO command before addressing any other device. Note that when the SELCH device is being addressed, prior to the GO command, the Single Mode may not be used since the Display Panel is addressed in this mode.

During data transfer, the Selector Channel provides a direct data path between the device and memory. Until the transfer is complete, no I/O instruction can be issued to any device on the Selector Channel, including the device transferring data. If devices on the Selector Channel Bus are referenced while the Selector Channel is busy, the False Sync bit is set (V Condition Code).

The initialization of a device on the Selector Channel Bus is accomplished by executing an Output Command (OC or OCR) instruction. Refer to the device Programming Manual for the bit configuration of the Output Command. Note that the Selector Channel has a unique device number just like all other I/O devices. Output Commands, as with all Input/Output instructions, affect only the device addressed.

The Selector Channel has a 16-bit incrementing Address Register and a 16-bit Final Address Register. The user program loads the starting address into the incrementing Address Register and the final address into the Final Address Register. Transfer is completed when the incrementing Address Register matches the Final Address Register. The address limits are expressed inclusively; transfers begin and end on the addresses placed in the starting and final address registers.

The memory is addressed on halfword boundaries; that is, each time memory is accessed, two bytes or a halfword are obtained. 16-bit addressing is used, with the least significant bit, Bit-15, determining the byte desired. See Figure 8-14.

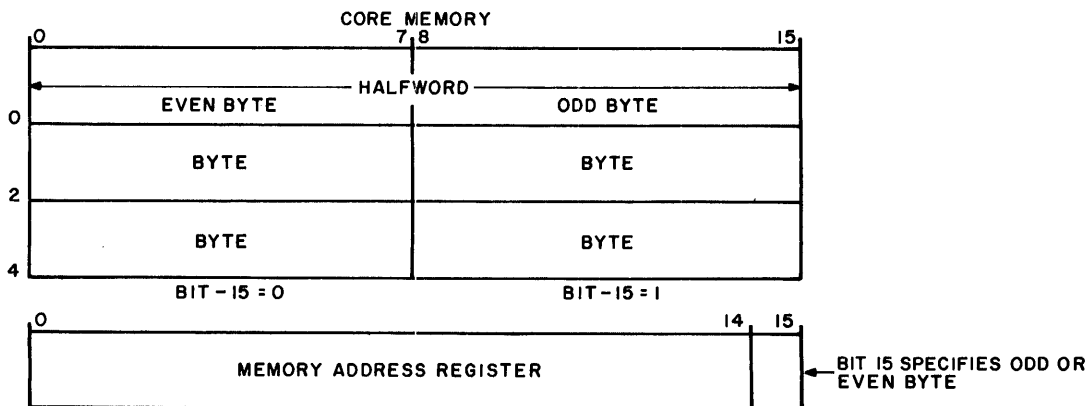


Figure 8-14. Memory Addressing

Each time the Selector Channel accesses memory, two bytes (halfword) are transferred. It is mandatory that data transfers begin on a halfword boundary. The following results if data transfers are ended on byte boundaries:

1. Write Mode (memory to device) - End on byte boundary (Bit-15 = 0) - No effect.
2. Read Mode (device to memory) - End of byte boundary (Bit-15 = 0) - The previous contents of the last odd byte in memory is written into the current odd byte in memory. See Figure 8-15.

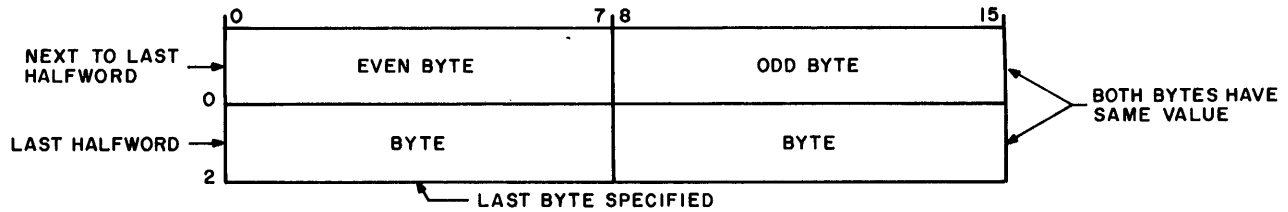
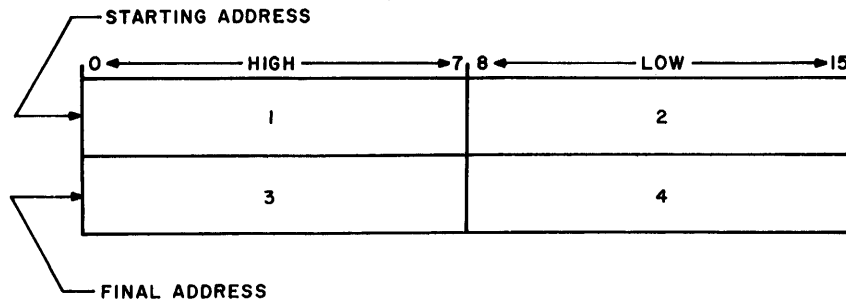


Figure 8-15. Memory Configuration, End on Byte Boundary

The user program specifies the mode, either Read or Write, and gives the GO command. The following sections provide details for programming the Selector Channel.

8.15.3.1 Starting and Final Addresses

An Output Command with the Stop bit set should be issued prior to starting any operation on the Selector Channel to clear any preceding conditions. Four successive bytes are required to specify the starting and final addresses of the user's buffer area. Either the Write Data (WD or WDR) or Write Halfword (WH or WHR) instructions may be used to send the starting and final addresses to the Selector Channel Controller. Figure 8-16 defines the four bytes used for addressing.



1. Starting Address High (Bits 0:7)
2. Starting Address Low (Bits 8:15)
3. Final Address High (Bits 0:7)
4. Final Address Low (Bits 8:15)

Figure 8-16. Starting and Final Address Data Bytes

8.15.3.2 Status and Commands

A Sense Status instruction (SS or SSR) is used to transfer the status byte from the Selector Channel Device Controller to the Processor. The least significant four bits (4:7) of the status byte are copied into the Condition Code during the Sense Status operation. Branch instructions can test these four bits directly. The status byte returned by the SELCH when idle, is the status of any device on the SELCH Bus with Bit-4 (BSY) forced to a zero. The Output Command instruction (OC or OCR) is used for transmitting the command byte to the Selector Channel Controller.

8.15.3.3 Termination

Data transmission between the Selector Channel and the device presently connected to it is halted if any of the following conditions occur:

1. The starting address matches the final address. This denotes a normal termination.
2. The starting (incrementing) address goes from all Ones to all Zeros (maximum count). In this case, a match has not occurred and is considered an abnormal termination.
3. Any of the DU, EOM, or EX status bits of the device presently connected to the Selector Channel changes to a One. This is also an abnormal termination.
4. A Stop command is sent to the Selector Channel via a user program.

The termination condition is determined in one of two ways: by a status scan, or by the interrupt method. These methods are described in the following paragraphs. An Output Command Stop should be issued to the Selector Channel following its termination.

NOTE

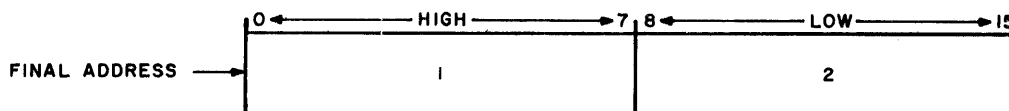
In the status scan method of programming, it is possible for the Busy Bit to change from One to Zero during a Sense Status instruction without returning the SELCH to Idle. To guarantee the Idle Mode after Busy = 0 on a Sense Status instruction, a Stop command should be sent to the SELCH.

Status Scan. The status of the Selector Channel Controller may be examined by issuing a Sense Status (SS or SSR) instruction. The Busy Bit (Bit-4) is a One while transmission is in progress, and Zero when transmission is terminated. One method of testing for termination would be to continually or periodically test the Busy Bit of the Selector Channel Controller. The change from One to Zero would then indicate the termination of a data transfer. When the Selector Channel is busy, only the Busy Bit (Bit-4) is present in the status byte and all other bits are Zero. At termination, the status of the device is presented in the status byte, except for the Busy Bit which is Zero.

Interrupt Method. When data transmission is initiated on the Selector Channel, the interrupt is always enabled. If external device interrupts are enabled (PSW Bit-1 set), the Processor is interrupted when the Selector Channel terminates. The interrupt can be serviced via Immediate Interrupt or Acknowledge Interrupt instruction (AI, AIR), which clears the interrupt and causes the device number of the Selector Channel (X'F0' preferred) and status of the peripheral device to be sent to the Processor. The Busy Bit is treated in the manner described previously for Status Scan.

8.15.3.4 Reading the Final Address

The last Processor memory location either written into or read from may be obtained by executing a pair of Read Data (RD or RDR) instructions or a Read Halfword (RH or RHR) instruction. The Read Block (RB or RBR) instruction should not be used. This information permits a user program to verify a successful data transmission or determine at what address termination occurred. Figure 8-17 illustrates the order in which the data is read into the Processor.



1. Final Address High (Bits 0:7)
2. Final Address Low (Bits 8:15)

Figure 8-17. Read Data Instructions

8.15.4 Interrupts

Refer to Section 8.15.3.3 Termination, Interrupt Method.

8.15.5 Initialization

Whenever the Initialize switch (INT) on the Display Panel is depressed, or a Stop command is issued, the following actions occur:

1. Any data transmission in process is halted and the Stop Mode is effected.
2. The Selector Channel is placed in the Write Mode.
3. The Selector Channel is made idle.
4. The Selector Channel interrupt is reset.

8.15.6 Device Number

The Selector Channel is normally assigned device number X'F0', but may easily be changed by a minor wiring modification on the Selector Channel Device Controller board. Refer to the Installation Specification, 02-232A20, for specific details.

Figure 8-18 presents a sample driver program for a magnetic tape unit connected to the Selector Channel. The purpose of this sample program is to show the program instructions used to control the Selector Channel and the order in which they may be executed.

The function of Subroutine 1 is to prepare the Selector Channel and device for a data transfer. Upon entry to Subroutine 1, Steps 1, 2, and 3 load the device number of the Selector Channel into a register and tests the Selector Channel's Busy Bit. If the Busy Bit is set, return is made to the calling program via the Busy Exit. If the Selector Channel is idle, Steps 5 and 6 test the status of the tape unit. If the tape unit status reveals it is available, Step 7 sends a command to the tape unit. If the tape unit is not available, return is made to the calling program via the error return exit. Steps 8 through 11 load the Selector Channel's Address Register. Step 12 then gives the GO command to the Selector Channel initiating the data transfer.

The function of Subroutine 2 is to service the interrupt caused by the Selector Channel. Step 14 acknowledges the interrupt, and at the same time loads the status of the device into register Status. Steps 15 and 16 read the incrementing (start) register of the Selector Channel and load the results into memory locations, LFINLH and LFIXLL. Steps 17 through 19 compare the actual ending address to that loaded into the Final Address Register. If not equal, return is made to the call program via the error return exit; if equal, return is made to the normal return.

8.15.7 Sample Program

Figure 8-18 provides a sample program for the Selector Channel.

```

*
* SAMPLE DRIVER PROGRAM FOR A MAGNETIC
* TAPE UNIT UNDER CONTROL OF SELECTOR
* CHANNEL INTERFACE
*
* NORMAL RETURN -REGISTER 15
* BUSY RETURN - REGISTER 15+4
* ERROR RETURN - REGISTER 15+8
*
* SUBROUTINE 1 INITIALIZES DEVICE AND SELECTOR CHANNEL
*
0000R C8A0 SUBR1 LHI SCDVNU,X'F0' (1) LOADS DEVICE NUMBER OF
      00F0
*
*                               SEL CHAN IN REGISTER
0004R 9DAC          SSR SCDVNU,STATUS (2) TEST SEL CHAN AVAIL-
0006R 428F          BTC 8,4(RETURN) (3) BUSY RETURN
      0004
000AR C8B0          LHI TPDVNU,X'20' (4) LOADS DEVICE NUMBER OF
      0020
*
*                               TAPE UNIT IN REGISTER
000ER 9DBC          SSR TPDVNU,STATUS (5) TEST TAPE UNIT AVAIL-
*                               ABLE FOR COMDS
0010R 45C0          CLH STATUS,COMSAT (6) TST STATUS OF TPE UNIT
      005AR
0014R 423F          BNE 8(RETURN)      ERROR RETURN
      0008
0018R DEB0          OC TPDVNU,CMDMOD (7 ) COMMANDS TAPE UNIT TO
      0052R
*
*                               READ MODE
001CR DEA0          OC SCDVNU,RESET   RESET SC REGISTERS
      0054R
*
*                               (8 ) SENDS BITS 0-7 OF
0020R DAA0          WD SCDVNU,STARTh
      0056R
*
*                               START ADDR TO SEL CHAN
0024R DAA0          WD SCDVNU,STARTL (9 ) SENDS BITS 8-15 OF
      0057R
*
*                               START ADDR TO SEL CHAN
0028R DAA0          WD SCDVNU,FINALH (10) SENDS BITS 0-7 OF
      0058R
*
*                               END ADDR TO SEL CHAN
002CR DAA0          WD SCDVNU,FINALL (11)SENDS BITS 8-15 OF
      0059R
*
*                               END ADDR TO SEL CHAN
0030R DEA0          OC SCDVNU,GORD (12) STARTS DATA TRANSFER
      0053R
*
*                               BETWEEN TAPE AND CORE
0034R 030F          BR RETURN (13) RETURN TO CALLING ROUTINE
*
*****
* SUBROUTINE 2 SERVICES INTERRUPT RETURN
*
0036R 9F9C          SBUR2 AIR DEVICE,STATUS (14) ACKNOWLEDGE INTERRUPT
*                               AND OBTAIN STATUS
0038R DEA0          OC SCDVNU,RESET SEND STOP TO SELECTOR
      0054R

```

Figure 8-18. Sample Program

```

003CR DBA0      *          RD      SCDVNU,LFINLH      CHANNEL FOLLOWING TERM.
005CR                                     (15) READ SEL CHAN BITS

0040R DBA0      *          RD      SCDVNU,LFINLL      0-7 FOR ENDING ADDR
005DR                                     (16) READ SEL CHAN BITS

0044R 48E0      *          LH      TEST,LFINLH      8-15 FOR ENDING ADDR
005CR                                     (17) LOADS ACTUAL ENDING ADDR
0048R 45E0      CLH      TEST,FINALH      (18) TO COMPARE WITH SPECIFIED
0058R 0058R

004CR 423F      BNE      8(RETURN)      (19) ERROR RETURN
0008

0050R 030F      BR      RETURN      (20) RETURN TO CALLING ROUTINE
0052R 9930      CMDMOD   DC      X'9930'
0053R          GORD     EQU     CMDMOD+1
0054R 0808      RESET    DC      X'0808'
0056R          STARTH   DS      2
0057R          STARTL   EQU     STARTH+1
0058R          FINALH   DS      2
0059R          FINALL   EQU     FINALH+1
0009          DEVICE   EQU     9
000A          SCDVNU   EQU     10
000B          TPDVNU   EQU     11
000C          STATUS   EQU     12
000D          ADDR     EQU     13
000E          TEST     EQU     14
000F          RETURN   EQU     15
005AR          COMSAT   DS      2          CONTAINS EXPECTED STATUS
005CR          LFINLH   DS      2
005DR          LFINLL   EQU     LFINLH+1
005ER          END

ADDR      000D
CMDMOD    0052R
COMSAT    005AR
DEVICE    0009
FINALH    0058R
FINALL    0059R
GORD      0053R
LFINLH    005CR
LFINLL    005DR
RESET     0054R
RETURN    000F
SBUR?     0036R
SCDVNU    000A
STARTH    0056R
STARTL    0057R
STATUS    000C
SUBR1     0000R
TEST      000E
TPDVNU    000B

```

Figure 8-18. Sample Program
(Continued)

CHAPTER 9

CONFIGURATION INSTALLATION PLANNING

9.1 INTRODUCTION

Modularity is the key word which describes the INTERDATA building block approach to configuring digital systems. The highly modular structure of the digital equipment permits custom configuration to suit the user's exact needs. It also provides the means for gracefully expanding a digital system as the user's requirements grow. This chapter describes the organization of the INTERDATA Digital System. Included are descriptions of the integrated circuit boards, the Processor and system expansion chassis, and the system cabinets.

9.2 INTEGRATED CIRCUIT BOARDS

Three sizes of circuit boards (7", 10", and 15") are used in INTERDATA Digital Systems. The user's requirements and system configuration determine which type of circuit board is used in a particular system.

The standard 15 inch circuit board, Figure 9-1, can be used in either the basic Processor chassis or in the 15 inch system expansion chassis, Product Number M49-020.

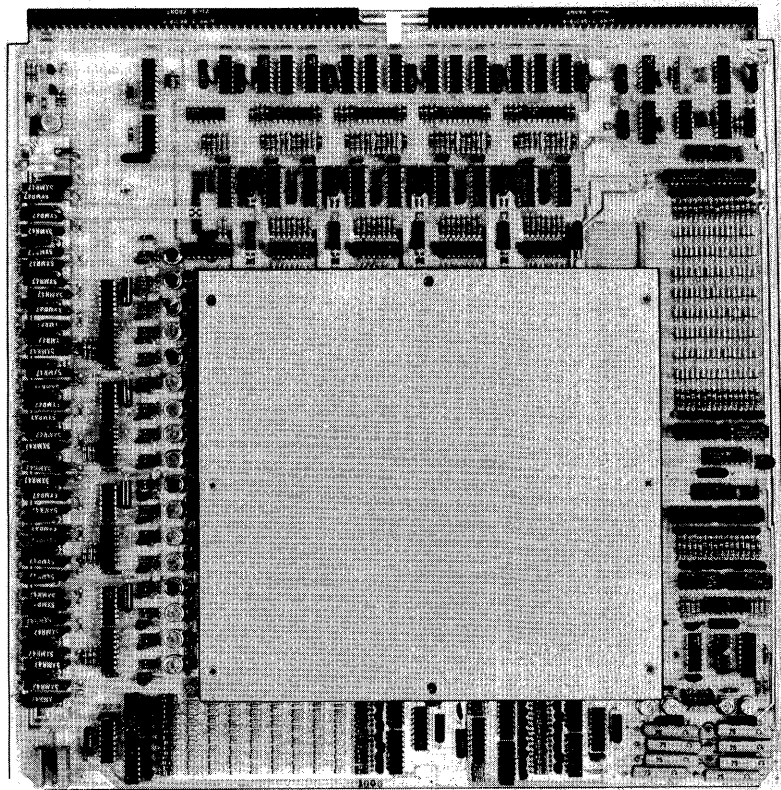


Figure 9-1. Typical 15-Inch Circuit Board, Component Side

The standard 7 inch circuit board is used for less sophisticated device controllers and system modules. The unique back panel wiring of the 15 inch system expansion chassis permits two 7 inch circuit boards (fastened together) to occupy the same card file slot.

This packaging scheme also permits use of the 10 inch INTERDATA circuit board to provide current INTERDATA users (device controller and system module) compatibility with the Model 74, 70, and 80 systems.

The methods of integrating the 10 inch circuit board into a Model 70 Digital System are:

1. With an adapter card, Product Number M49-003, which permits the 10 inch circuit board to be used in either the basic Processor chassis, or in the system 15 inch expansion chassis, Product Number M49-020. This method can only be used for single circuit board interfaces.
2. With a 10 inch system expansion chassis, Product Number M49-000, which is prewired for up to six 10 inch circuit cards.

9.3 BASIC PROCESSOR CHASSIS

The basic Processor is packaged in a single RETMA standard 7 inch rack mountable chassis. (Figure 9-2). This chassis includes a power supply, cooling fans, and room for eight standard 15 inch printed circuit boards. The Model 74 Processor takes two PC boards, the Model 70 Processor takes four PC boards, the Model 80 Processor takes four PC boards. The 8, 192 byte core memory in the Models 74 and 70, and the 16, 384 byte MOS memory in the Model 80 take one PC board position in the chassis. The remaining circuit board slots are universal expansion slots which may accept any combination of the following:

1. Additional 8, 192 byte core memory or 16, 384 MOS memory modules.
2. Standard INTERDATA 15 inch peripheral device controllers.
3. Standard INTERDATA 7 inch peripheral device controllers (two per slot can be accommodated).
4. Standard INTERDATA 10 inch peripheral device controllers with adapter.
5. Standard INTERDATA Selector Channel.
6. User designed interfaces to either the Multiplexor Bus (I/O Bus) or the Memory Bus.

These circuit boards may be placed in the basic Processor chassis as required by the particular system configuration subject to certain configuration constraints.

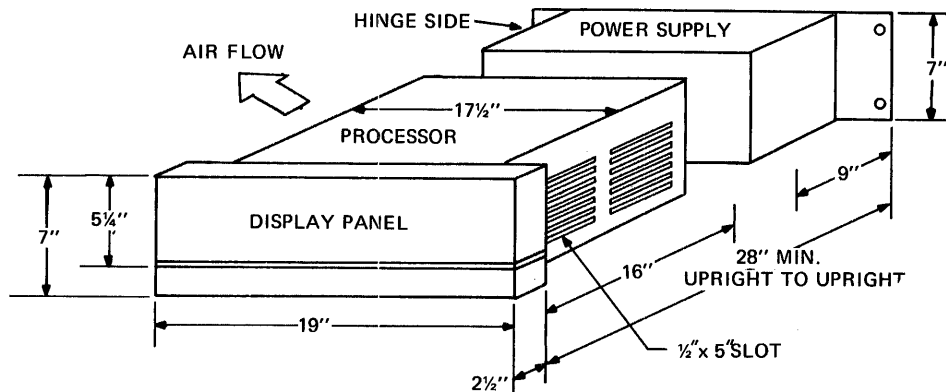


Figure 9-2. Basic Processor Chassis

9.4 SYSTEM EXPANSION CHASSIS

Two system expansion chassis (10 inch and 15 inch) are available for expanding INTERDATA Digital Systems. The system expansion chassis have the same dimensions as the basic Processor chassis. See Figure 9-2. When configuring a multi-chassis system there are four rules that must be followed:

1. The system expansion chassis must be mounted below the basic Processor chassis.
2. All chassis must be contiguous.
3. All 15 inch system expansion chassis (M49-020) must precede the 10 inch system expansion chassis (M49-000).
4. Multiboard peripheral device controllers (on 10 inch circuit boards) can only be used in the 10 inch system expansion chassis.

9.4.1 15 Inch System Expansion Chassis

The 15 inch system expansion chassis, INTERDATA Product Number M49-020, contains eight universal expansion slots which can accept combinations of memory modules, single board peripheral controllers, systems modules, Selector Channels, or user designed interfaces as described for the basic Processor chassis (Section 9.3). Included with this chassis are the cooling fans and system interconnecting cables.

9.4.2 10 Inch System Expansion Chassis

The 10 inch system expansion chassis, INTERDATA Product Number M49-000, contains six 10 inch I/O expansion slots which can accept any combination of up to six wire-wrap or copper peripheral controllers, systems modules, or user designed interfaces as described for the basic Processor chassis (Section 9.3). Included with the chassis are the cooling fans and system interconnecting cables. The power supply is purchased separately. Power for this chassis is provided by the System Power Supply, INTERDATA Product Number M49-002.

9.5 CIRCUIT BOARD DISTRIBUTION

An INTERDATA Digital System may be configured in a variety of ways. However, the following factors must be considered when determining circuit board distribution within the basic Processor and the system expansion chassis.

1. The Selector Channel can only be placed in Slot 0 of the Processor chassis, or Slots 6, 4, 2, or 0 of the system expansion chassis.
2. All slots below the position where the SELCH is inserted become SELCH Bus slots. (This only applies within the chassis containing the SELCH.) The SELCH Bus extends down the left side connectors (front view). Note that all 7", and 10" with adapter, device controllers that are connected to the Multiplexor Bus from the right side connectors (front view) may be inserted in vacant SELCH Bus slots.
3. The SELCH Bus can be extended by cable to any even numbered slot in an I/O chassis adjacent to the chassis containing the SELCH controller.
4. The Universal Clock module (7" x 15") is always mounted on the right side (front view).
5. The Memory Protect module (7" x 15") is always mounted on the left side, and is normally mounted with the Universal Clock module.

NOTE

The Memory Protect Module must be placed ahead of all other I/O device controllers, including the SELCH.

6. All device addresses are hard-wired on the device controller cards, so that the distribution of I/O device controllers in the chassis normally need only be considered as a matter of convenience.
7. The 15 inch system expansion chassis, M49-020, and the basic Processor chassis may only be used for single board I/O device controllers. For multi-board 10 inch device controllers, use the 10 inch system expansion chassis, 49-000.

9.6 SYSTEM CABINET

Two variations of the system cabinets are available for housing INTERDATA Digital Systems.

The system cabinet, INTERDATA Product Number M49-004, is a basic cabinet with accessories. See Figures 9-3, 9-4, and 9-5. This cabinet includes the basic cabinet with two side skins, chassis support rails, exhaust fan and filter, necessary filler panels, and AC Distribution Panel.

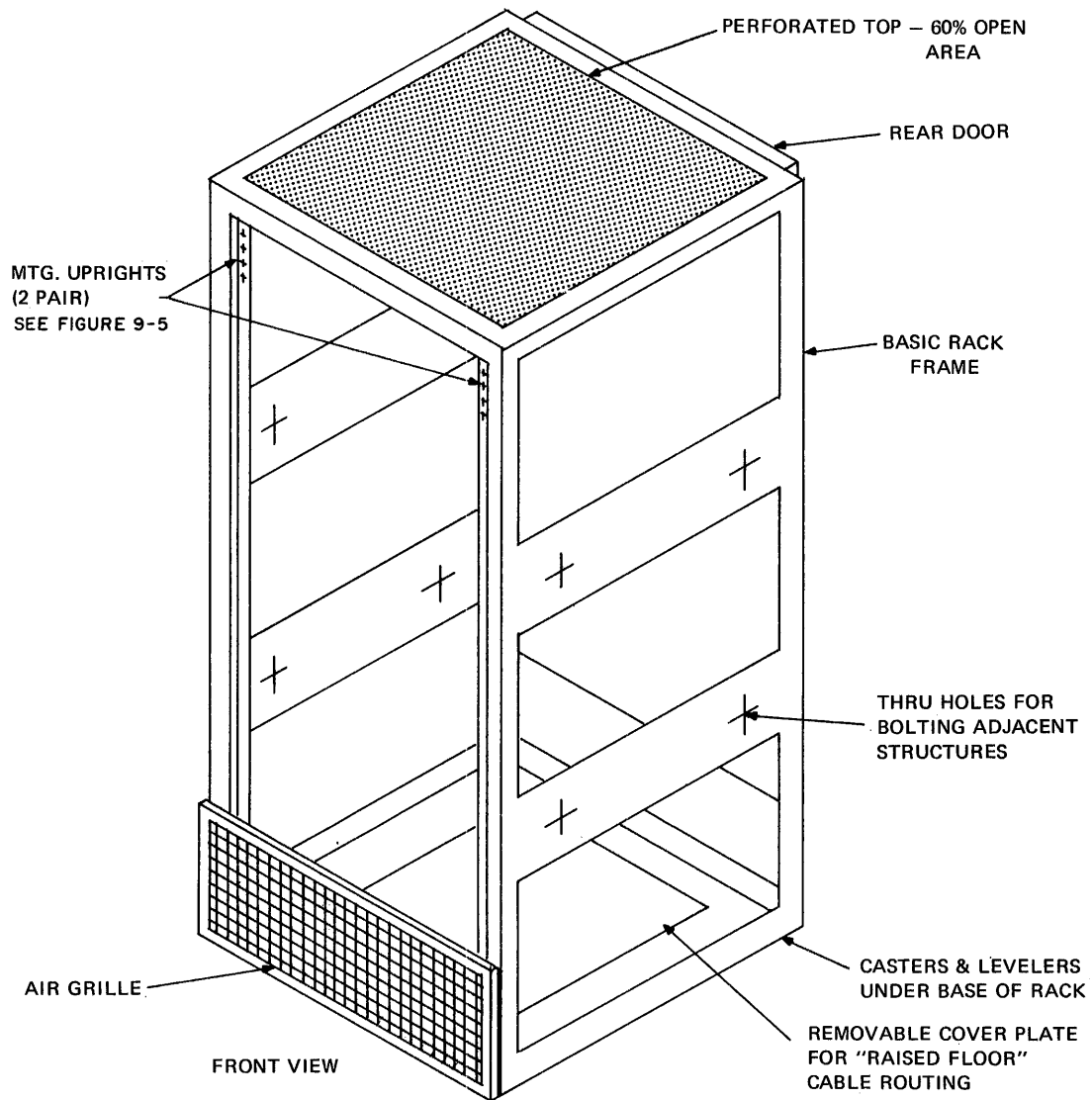


Figure 9-3. Basic Cabinet

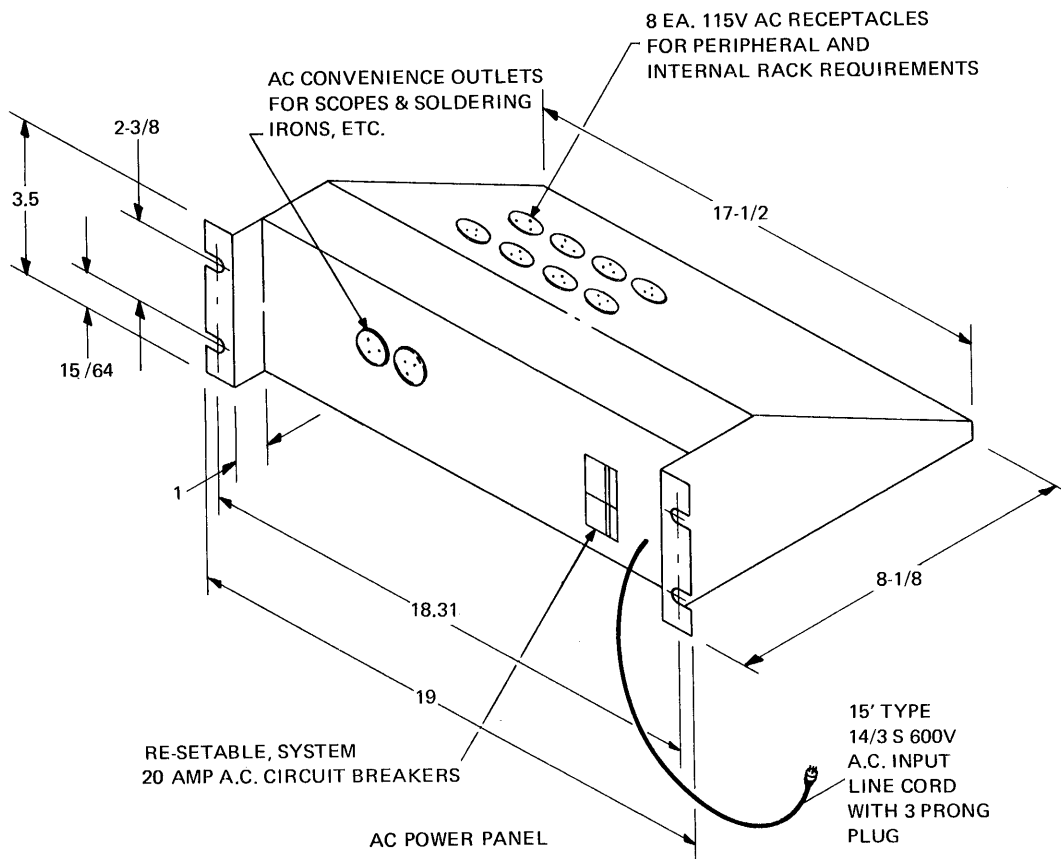
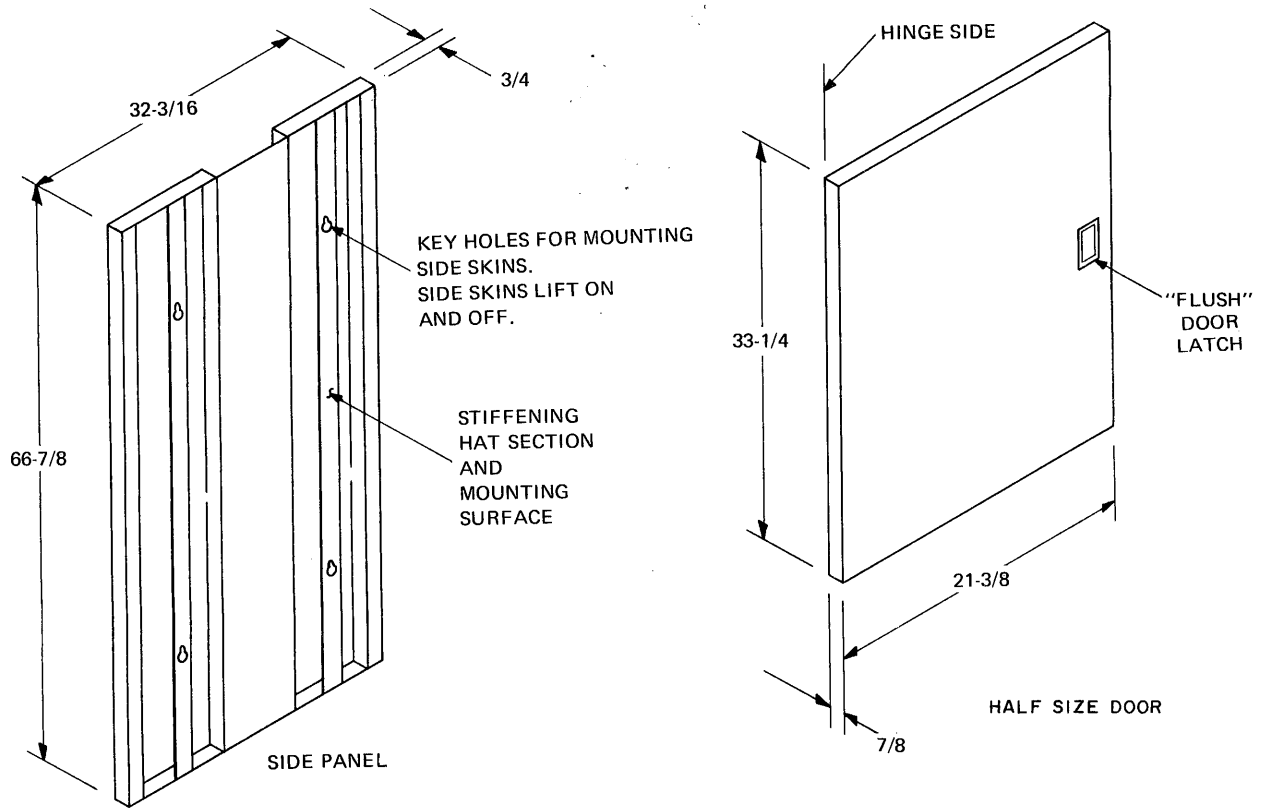


Figure 9-4. Cabinet Accessories, Sheet 1 of 2

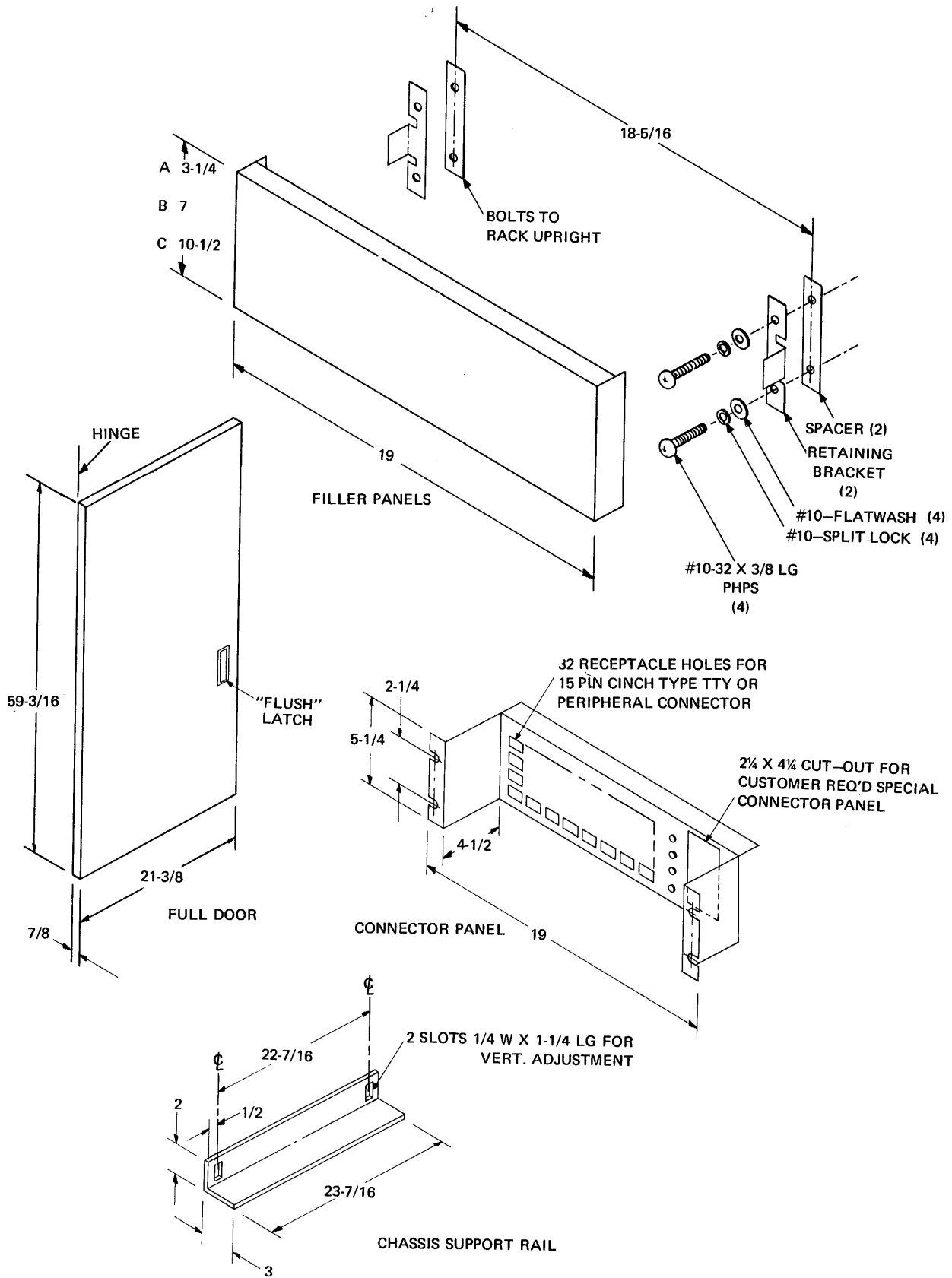


Figure 9-4. Cabinet Accessories, Sheet 2 of 2

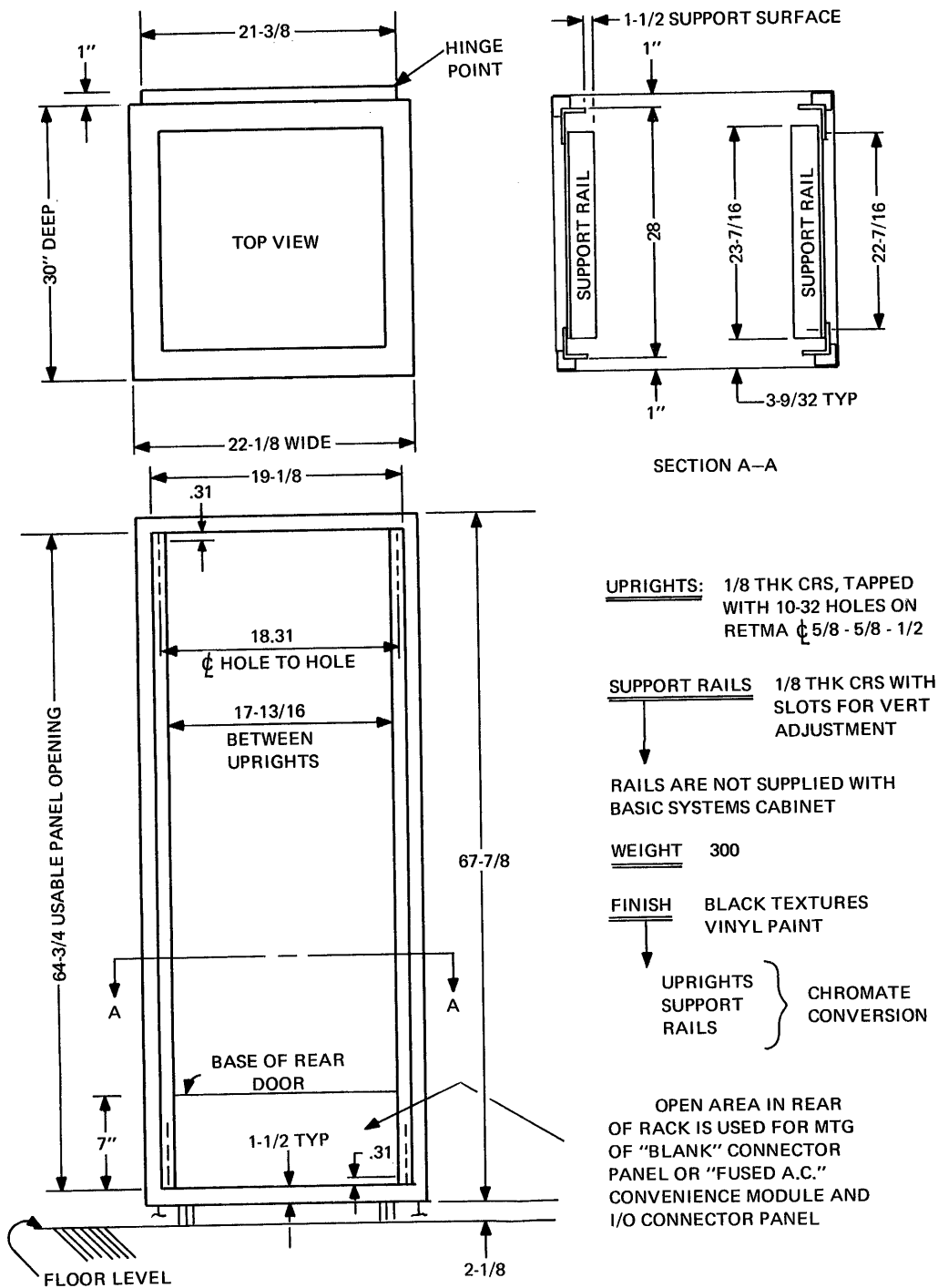


Figure 9-5. Basic Cabinet Physical Dimensions

9.7 SYSTEM CONFIGURATION DATA

The flexibility derived from the modular design concept of the INTERDATA equipment permits a wide variety of system configurations. Therefore, care must be taken during the planning and installation of a Digital System to insure that the resulting system configuration meets the user's requirements, and that the equipment is installed correctly in the system cabinets.

To accomplish this, the digital system configuration is divided into four phases. The first phase is determining the requirements of the system. In this phase, as in all four phases, INTERDATA Sales Engineers will provide detailed information on INTERDATA hardware. To solve specific application oriented problems, the INTERDATA Technical Support Group, and Systems Group are available.

The second phase is the physical distribution of the various circuit boards within the chassis as discussed in Section 9-5.

The last two phases, determining the power requirements of the system, and determining the physical configurations of the equipment in the system cabinets, are discussed in the following sections.

9.7.1 Power Requirements

The Processor power supply provides ample power to the basic Processor chassis. However, as expansion items are integrated into a digital system, care must be taken to insure that adequate regulated power is available for all of the equipment. The following example illustrates the power requirements for a typical Model 70 system. Note that the basic Processor chassis includes a power supply for that chassis only; the 15 inch Expansion chassis includes a power supply for that chassis only; and that a power supply must normally be purchased separately for the 10 inch I/O chassis, unless a complete power balance has been calculated for all chassis other than the Processor chassis.

Configuration			Power		
Quantity	Product Number	Description	+5V	+16V	-16V
1	M70-000	Model 70 Processor with 8,192 Byte Core Memory	17	2.4	2.4
3	M70-300	8,192 Byte Memory Module	4.5	0.3	0.6
1	M70-100	Power Fail Protection	-	-	-
1	M70-103	Selector Channel	2.3	-	-
1	M70-101	Memory Protect	1.0	-	-
1	M48-000	Clock Module	1.0	-	-
1	M46-000	Teletypewriter	-	-	-
1	M07-735	Disc	3.0	-	-
1	M49-000	Expansion Chassis	-	-	-
1	M49-020	Expansion Chassis	-	-	-
1	M49-002	Power Supply	-	-	-
Total Power Required in Amperes			28.8	2.7	3.0

Power Balance Example

<u>Processor</u>	<u>+5V</u>	<u>+16V</u>	<u>-16V</u>
Basic Processor Power Supply	24.0	3	3
Basic Processor Chassis Power requirement	-21.5	2.7	3.0
	<hr/> 2.5	<hr/> .3	<hr/> 0

Processor power is adequate (not normally necessary to do this, since the power supply is designed to handle maximum possible Processor Chassis Configuration).

<u>Expansion Chassis</u>			
M49-002 Expansion Chassis Power Supply	24.0	3	3
Power required for items not in Processor chassis	- 7.3	-	-
	<hr/> 16.7	<hr/> 3	<hr/> 3

Ample power available to support the M49-000 Expansion chassis, in which the disc controller would be mounted. Refer to the current INTERDATA price list for complete power requirement information.

9.7.2 System Cabinets Configuration

Configuration Data Sheet A, Figure 9-6, serves as aid in the physical layout of components within a system cabinet.

When configuring system cabinets, the following information should be noted:

EXTERNAL FRONT

1. The Control and Filler Panels must be mounted immediately in front of the Processor chassis.
2. The half door must occupy location MU5-23.
3. The Air Grill occupies location MU1-4.
4. Filler Panel C should be used in front of any chassis unless a door is present.
5. Doors cannot be mounted over the Control Panel.

INTERNAL FRONT

1. All chassis must be contiguous.
2. The system expansion chassis must be mounted below the Processor chassis.
3. Peripherals are normally mounted above the Processor chassis.
4. The system expansion chassis must be located immediately below the Processor chassis.
5. The optional blower including baffle occupies location MU1-5.

INTERNAL REAR

1. The AC Convenience Panel normally occupies location MU1-2.
2. The I/O Connector Panel normally occupies location MU3-5.
3. The relay module mounting dimension includes 3-1/2" of space to allow for the replacement of the relays.
4. Power supplies for the Processor and system expansion chassis are mounted immediately behind the respective chassis.
5. The paper tape punch or magnetic tapes 'rear' space is not usable.

Table 9-1, Mounting Dimensions, provides vertical measurements and mounting unit requirements for other cabinet accessories and some of the available peripheral devices. Figure 9-6 illustrates the use of Configuration Data Sheet A for a Model 70 Processor with an INTERTAPE Cassette, Digital Multiplexor, Drum, and Magnetic Tape mounted in two M49-004 System Cabinets.

CONFIGURATION DATA SHEET "A"		CUSTOMER _____	
PHYSICAL LAYOUT		P.O. or QUOTE NO. _____ DATE _____	
	EXTERNAL FRONT	INTERNAL FRONT	INTERNAL REAR
37			
36			SPARE
35			DIGITAL MPX CHASSIS
34	FILLER PANEL D	SPARE	
33			
32			
31			
30	CASSETTE DISPLAY	CASSETTE	
29			
28			
27			
26	PROCESSOR DISPLAY	PROCESSOR CHASSIS	PROCESSOR POWER SUPPLY
25			
24	FILLER A		
23		10" EXPANSION CHASSIS	EXPANSION POWER SUPPLY
22			
21			
20			
19			
18			DIGITAL MPX INPUT POWER SUPPLY
17			
16			
15			
14	HALF DOOR		
13			
12			
11			SCREW TERMINATION PANEL
10			
9			
8			
7			SPARE
6			
5		BAFFLE	
4			I/O CONNECTOR PANEL
3	AIR GRILL	BLOWER (OPTIONAL)	A.C. DISTRIBUTION PANEL
2			
1			

MOUNTING UNITS (MU'S)

IMU = 1-3/4"
 TOTAL SPACE AVAILABLE = 64-3/4" = 37 MU

A741

Figure 9-6. Typical Examples - Configuration Data Sheet A (Sheet 1 of 2)

CONFIGURATION DATA SHEET "A"		CUSTOMER _____	
PHYSICAL LAYOUT		P.O. or QUOTE NO. _____ DATE _____	
	EXTERNAL FRONT	INTERNAL FRONT	INTERNAL REAR
37			
36			
35			
34			
33			
32			
31			
30	MAGNETIC TAPE	MAGNETIC TAPE	UNAVAILABLE
29	DISPLAY	DISPLAY	
28			
27			
26			
25			
24			
23			
22			SPARE
21		SPARE	
20			DRUM POWER SUPPLY
19			
18			
17			
16			
15		DRUM MEMORY	UNAVAILABLE
14	HALF DOOR		
13			
12			
11			
10			
9			SPARE
8			
7			
6		SPARE	
5			I/O CONNECTOR PANEL
4			A.C. CONVENIENCE PANEL
3	AIR GRILL		
2			
1			

MOUNTING UNITS (MU'S)

IMU = 1-3/4"
TOTAL SPACE AVAILABLE = 64-3/4" = 37 MU

A74

Figure 9-6. Typical Examples - Configuration Data Sheet A (Sheet 2 of 2)

TABLE 9-1. MOUNTING DIMENSIONS

Product Description	Mounting Units Req'd	Vertical Measurement
External Front		
Display Panel with Filler 'A'	4	7"
Halfsize Door	19	33-1/4"
Air Grill	4	7"
Filler A	1	1-3/4"
Filler B	3	5-1/4"
Filler C	4	7"
Filler D	6	10-1/2"
Internal Front		
Processor Chassis	4	7"
15" Expansion Chassis	4	7"
10" Expansion Chassis	4	7"
*Paper Tape Reader	4	7"
*Paper Tape Punch	9	15-3/4"
*INTERTAPE with Filler A (for air circulation) - Filler goes below INTERTAPE	4	7"
*Mag Tape Drive	14	24-1/2"
Drum Unit (131KB-524KB)	8	14"
Removable Cartridge Disk System	7	12-1/4"
Blower, including 1-3/4" baffle mounted on top of blower	5	8-3/4"
Paper Tape Handler	4	7"
Internal Rear		
Processor Power Supply	4	7"
Expansion Chassis Power Supply	4	7"
Paper Tape Punch	9	15-3/4"
Magnetic Tape	14	24-1/2"
Drum Power Supply	3	5-1/4"
AC Convenience Panel	2	3-1/2"
I/O Connector Panel	3	5-1/4"
Rear Blank Panel (fits below rear door)	4	7"
Digital Mux Chassis	3	5-1/4"
Relay Module - with screw terminals and clearance for relay access	6	10-1/2"
64 Input Screw Terminal Panel	7	12-1/4"
Input Power Supply for Digital Mux	3	5-1/4"

NOTE: Rear door covers all but the four lowest Mounting Units

*Allow for these modules in external front layout.

APPENDIX 1

INSTRUCTION SUMMARY - ALPHABETICAL

INSTRUCTION	OP CODE	MNEMONIC	PAGE NO.
Acknowledge Interrupt	DF	AI	4-37
Acknowledge Interrupt RR	9F	AIR	4-37
Add Halfword	4A	AH	4-8
Add Halfword Immediate	CA	AHI	4-8
Add Halfword RR	0A	AHR	4-8
Add Halfword Memory	61	AHM	4-8
Add Immediate Short	26	AIS	4-8
**Add to Bottom of List	65	ABL	4-58
**Add to Top of List	64	ATL	4-58
Add with Carry Halfword	4E	ACH	4-9
Add with Carry Halfword RR	0E	ACHR	4-9
AND Halfword	44	NH	4-17
AND Halfword Immediate	C4	NHI	4-17
AND Halfword RR	04	NHR	4-17
Autoload	D5	AL	4-43
Branch and Link	41	BAL	4-35
Branch and Link RR	01	BALR	4-35
*Branch on False Condition	43	BFC	4-33
*Branch on False Condition RR	03	BFCR	4-33
*Branch on True Condition	42	BTC	4-32
*Branch on True Condition RR	02	BTCR	4-32
*Branch on True Backward Short	20	BTBS	4-32
*Branch on True Forward Short	21	BTFS	4-32
*Branch on False Backward Short	22	BFBS	4-33
*Branch on False Forward Short	23	BFFS	4-33
Branch on Index High	C0	BXH	4-34
Branch on Index Low or Equal	C1	BXLE	4-34
Compare Halfword	49	CH	4-13
Compare Halfword Immediate	C9	CHI	4-13
Compare Halfword RR	09	CHR	4-13
Compare Logical Byte	D4	CLB	4-23
Compare Logical Halfword	45	CLH	4-12
Compare Logical Halfword Immediate	C5	CLHI	4-12
Compare Logical Halfword RR	05	CLHR	4-12
Divide Halfword	4D	DH	4-15
Divide Halfword RR	0D	DHR	4-15

*See Extended Branch Mnemonics in Appendix 3 for forty-four (44) additional symbolic instructions.

**Models 70, 80 only

INSTRUCTION	OP CODE	MNEMONIC	PAGE NO.
Exchange Byte RR	94	EXBR	4-23
Exchange Program Status RR	95	EPSR	4-48
Exclusive OR Halfword	47	XH	4-19
Exclusive OR Halfword Immediate	C7	XHI	4-19
Exclusive OR Halfword RR	07	XHR	4-19
**Floating - Point Add	6A	AE	4-52
**Floating - Point Add RR	2A	AER	4-52
**Floating - Point Compare	69	CE	4-54
**Floating - Point Compare RR	29	CER	4-54
**Floating - Point Divide	6D	DE	4-56
**Floating - Point Divide RR	2D	DER	4-56
**Floating - Point Load	68	LE	4-51
**Floating - Point Load RR	28	LER	4-51
**Floating - Point Multiply	6C	ME	4-55
**Floating - Point Multiply RR	2C	MER	4-55
**Floating - Point Store	60	STE	4-51
**Floating - Point Subtract	6B	SE	4-53
**Floating - Point Subtract RR	2B	SER	4-53
Load Byte	D3	LB	4-22
Load Byte RR	93	LBR	4-22
Load Complement Short	25	LCS	4-4
Load Halfword	48	LH	4-4
Load Halfword Immediate	C8	LHI	4-4
Load Halfword RR	08	LHR	4-4
Load Immediate Short	24	LIS	4-4
Load Multiple	D1	LM	4-6
Load Program Status Word	C2	LPSW	4-47
Multiply Halfword	4C	MH	4-14
Multiply Halfword RR	0C	MHR	4-14
Multiply Halfword Unsigned	DC	MHU	4-14
Multiply Halfword Unsigned RR	9C	MHUR	4-14
OR Halfword	46	OH	4-18
OR Halfword Immediate	C6	OHI	4-18
OR Halfword RR	06	OHR	4-18
Output Command	DE	OC	4-39
Output Command RR	9E	OCR	4-39

**Models 70, 80 only

INSTRUCTION	OP CODE	MNEMONIC	PAGE NO.
**Read Block	D7	RB	4-45
**Read Block RR	97	RBR	4-45
Read Data	DB	RD	4-39
Read Data RR	9B	RDR	4-39
Read Halfword	D9	RH	4-41
Read Halfword RR	99	RHR	4-41
Rotate Left Logical	EB	RLL	4-27
Rotate Right Logical	EA	RRL	4-28
**Remove from Bottom of List	67	RBL	4-59
**Remove from Top of List	66	RTL	4-59
Sense Status	DD	SS	4-38
Sense Status RR	9D	SSR	4-38
Shift Left (Fullword) Arithmetic	EF	SLA	4-29
Shift Left (Fullword) Logical	ED	SLL	4-25
Shift Left (Halfword) Arithmetic	CF	SLHA	4-29
Shift Left (Halfword) Logical	CD	SLHL	4-25
Shift Left Logical Short	91	SLLS	4-25
Shift Right (Fullword) Arithmetic	EE	SRA	4-30
Shift Right (Fullword) Logical	EC	SRL	4-26
Shift Right (Halfword) Arithmetic	CE	SRHA	4-30
Shift Right (Halfword) Logical	CC	SRHL	4-26
Shift Right Logical Short	90	SRLS	4-26
Simulate Interrupt	E2	SINT	4-48
Store Byte	D2	STB	4-22
Store Byte RR	92	STBR	4-22
Store Halfword	40	STH	4-6
Store Multiple	D0	STM	4-5
Subtract Halfword	4B	SH	4-10
Subtract Halfword Immediate	CB	SHI	4-10
Subtract Halfword RR	0B	SHR	4-10
Subtract Immediate Short	27	SIS	4-10
Subtract with Carry Halfword	4F	SCH	4-11
Subtract with Carry Halfword RR	0F	SCHR	4-11
Supervisor Call	E1	SVC	4-49
Test Halfword Immediate	C3	THI	4-20
**Write Block	D6	WB	4-46
**Write Block RR	96	WBR	4-46
Write Data	DA	WD	4-40
Write Data RR	9A	WDR	4-40
Write Halfword	D8	WH	4-42
Write Halfword RR	98	WHR	4-42

**Models 70, 80 only

APPENDIX 2

INSTRUCTION SUMMARY - NUMERICAL

OP CODE	MNEMONIC	INSTRUCTION	PAGE NO.
01	BALR	Branch and Link RR	4-35
02	BTCR	Branch on True Condition RR	4-32
03	BFCR	Branch on False Condition RR	4-33
04	NHR	AND Halfword RR	4-17
05	CLHR	Compare Logical Halfword RR	4-12
06	OHR	OR Halfword RR	4-18
07	XHR	Exclusive OR Halfword RR	4-19
08	LHR	Load Halfword RR	4-4
09	CHR	Compare Halfword RR	4-13
0A	AHR	Add Halfword RR	4-8
0B	SHR	Subtract Halfword RR	4-10
0C	MHR	Multiply Halfword RR	4-14
0D	DHR	Divide Halfword RR	4-15
0E	ACHR	Add with Carry Halfword RR	4-9
0F	SCHR	Subtract with Carry Halfword RR	4-11
20	BTBS	Branch on True Backward Short	4-32
21	BTFS	Branch on True Forward Short	4-32
22	BFBS	Branch on False Backward Short	4-33
23	BFFS	Branch on False Forward Short	4-33
24	LIS	Load Immediate Short	4-4
25	LCS	Load Complement Short	4-4
26	AIS	Add Immediate Short	4-8
27	SIS	Subtract Immediate Short	4-10
**28	LER	Floating-Point Load RR	4-51
**29	CER	Floating-Point Compare RR	4-54
**2A	AER	Floating-Point Add RR	4-52
**2B	SER	Floating-Point Subtract RR	4-53
**2C	MER	Floating-Point Multiply RR	4-55
**2D	DER	Floating-Point Divide RR	4-56
40	STH	Store Halfword	4-6
41	BAL	Branch and Link	4-35
42	BTC	Branch on True Condition	4-32
43	BFC	Branch on False Condition	4-33
44	NH	AND Halfword	4-17
45	CLH	Compare Logical Halfword	4-12
46	OH	OR Halfword	4-18
47	XH	Exclusive OR Halfword	4-19
48	LH	Load Halfword	4-4
49	CH	Compare Halfword	4-13
4A	AH	Add Halfword	4-8
4B	SH	Subtract Halfword	4-10
4C	MH	Multiply Halfword	4-14
4D	DH	Divide Halfword	4-15
4E	ACH	Add with Carry Halfword	4-9
4F	SCH	Subtract with Carry Halfword	4-11
**60	STE	Floating-Point Store	4-51
61	AHM	Add Halfword Memory	4-8
**64	ATL	Add to Top of List	4-58
**65	ABL	Add to Bottom of List	4-58

**Models 70, 80 only

OP CODE	MNEMONIC	INSTRUCTION	PAGE NO.
**66	RTL	Remove from Top of List	4-59
**67	RBL	Remove from Bottom of List	4-59
**68	LE	Floating-Point Load	4-51
**69	CE	Floating-Point Compare	4-54
**6A	AE	Floating-Point Add	4-52
**6B	SE	Floating-Point Subtract	4-53
**6C	ME	Floating-Point Multiply	4-55
**6D	DE	Floating-Point Divide	4-56
90	SRLS	Shift Right Logical Short	4-26
91	SLLS	Shift Left Logical Short	4-25
92	STBR	Store Byte RR	4-22
93	LBR	Load Byte RR	4-22
94	EXBR	Exchange Byte RR	4-23
95	EPSR	Exchange Program Status RR	4-48
**96	WBR	Write Block RR	4-46
**97	RBR	Read Block RR	4-45
98	WHR	Write Halfword RR	4-42
99	RHR	Read Halfword RR	4-41
9A	WDR	Write Data RR	4-40
9B	RDR	Read Data RR	4-39
9C	MHUR	Multiply Halfword Unsigned RR	4-14
9D	SSR	Sense Status RR	4-38
9E	OCR	Output Command RR	4-39
9F	AIR	Acknowledge Interrupt RR	4-37
C0	BXH	Branch on Index High	4-34
C1	BXLE	Branch on Index Low or Equal	4-34
C2	LPSW	Load Program Status Word	4-47
C3	THI	Test Halfword Immediate	4-20
C4	NHI	AND Halfword Immediate	4-17
C5	CLHI	Compare Logical Halfword Immediate	4-12
C6	OHI	OR Halfword Immediate	4-18
C7	XHI	Exclusive OR Halfword Immediate	4-19
C8	LHI	Load Halfword Immediate	4-4
C9	CHI	Compare Halfword Immediate	4-13
CA	AHI	Add Halfword Immediate	4-8
CB	SHI	Subtract Halfword Immediate	4-10
CC	SRHL	Shift Right (Halfword) Logical	4-26
CD	SLHL	Shift Left (Halfword) Logical	4-25
CE	SRHA	Shift Right (Halfword) Arithmetic	4-30
CF	SLHA	Shift Left (Halfword) Arithmetic	4-29
D0	STM	Store Multiple	4-5
D1	LM	Load Multiple	4-6
D2	STB	Store Byte	4-22
D3	LB	Load Byte	4-22
D4	CLB	Compare Logical Byte	4-23
D5	AL	Auto Load	4-43
**D6	WB	Write Block	4-46
**D7	RB	Read Block	4-45
D8	WH	Write Halfword	4-42
D9	RH	Read Halfword	4-41
DA	WD	Write Data	4-40
DB	RD	Read Data	4-39
DC	MHU	Multiply Halfword Unsigned	4-14
DD	SS	Sense Status	4-38

**Models 70, 80 only

OP CODE	MNEMONIC	INSTRUCTION	PAGE NO.
DE	OC	Output Command	4-39
DF	AI	Acknowledge Interrupt	4-37
E1	SVC	Supervisor Call	4-49
E2	SINT	Simulate Interrupt	4-48
EA	RRL	Rotate Right Logical	4-28
EB	RLL	Rotate Left Logical	4-27
EC	SRL	Shift Right (Fullword) Logical	4-26
ED	SLL	Shift Left (Fullword) Logical	4-25
EE	SRA	Shift Right (Fullword) Arithmetic	4-30
EF	SLA	Shift Left (Fullword) Arithmetic	4-29

APPENDIX 3

EXTENDED BRANCH MNEMONICS

INSTRUCTION	OP CODE (M1)	MNEMONIC	OPERANDS
Branch on Carry	428	BC	A(X2)
Branch on Carry RR	028	BCR	R2
Branch on No Carry	438	BNC	A(X2)
Branch on No Carry RR	038	BNCR	R2
Branch on Equal	433	BE	A(X2)
Branch on Equal RR	033	BER	R2
Branch on Not Equal	423	BNE	A(X2)
Branch on Not Equal RR	023	BNER	R2
Branch on Low	428	BL	A(X2)
Branch on Low RR	028	BLR	R2
Branch on Not Low	438	BNL	A(X2)
Branch on Not Low RR	038	BNLR	R2
Branch on Minus	421	BM	A(X2)
Branch on Minus RR	021	BMR	R2
Branch on Not Minus	431	BNM	A(X2)
Branch on Not Minus	031	BNMR	R2
Branch on Plus	422	BP	A(X2)
Branch on Plus RR	022	BPR	R2
Branch on Not Plus	432	BNP	A(X2)
Branch on Not Plus RR	032	BNPR	R2
Branch on Overflow	424	BO	A(X2)
Branch on Overflow RR	024	BOR	R2
Branch Unconditional	430	B	A(X2)
Branch Unconditional RR	030	BR	R2
Branch on Zero	433	BZ	A(X2)
Branch on Zero RR	033	BZR	R2
Branch on Not Zero	423	BNZ	A(X2)
Branch on Not Zero RR	023	BNZR	R2
No Operation	420	NOP	
No Operation RR	020	NOPR	
Branch on Carry Short	208	BCS	A (Backward Reference)
	218	BCS	A (Forward Reference)
Branch on No Carry Short	228	BNCS	A (Backward Reference)
	238	BNCS	A (Forward Reference)
Branch on Equal Short	223	BES	A (Backward Reference)
	233	BES	A (Forward Reference)
Branch on Not Equal Short	203	BNES	A (Backward Reference)
	213	BNES	A (Forward Reference)

INSTRUCTION	OP CODE (M1)	MNEMONIC	OPERANDS
Branch on Low Short	208	BLS	A (Backward Reference)
	218	BLS	A (Forward Reference)
Branch on Not Low Short	228	BNLS	A (Backward Reference)
	238	BNLS	A (Forward Reference)
Branch on Minus Short	201	BMS	A (Backward Reference)
	211	BMS	A (Forward Reference)
Branch on Not Minus Short	221	BNMS	A (Backward Reference)
	231	BNMS	A (Forward Reference)
Branch on Plus Short	202	BPS	A (Backward Reference)
	212	BPS	A (Forward Reference)
Branch on Not Plus Short	222	BNPS	A (Backward Reference)
	232	BNPS	A (Forward Reference)
Branch on Overflow Short	204	BOS	A (Backward Reference)
	214	BOS	A (Forward Reference)
Branch Unconditional Short	220	BS	A (Backward Reference)
	230	BS	A (Forward Reference)
Branch on Zero Short	223	BZS	A (Backward Reference)
	233	BZS	A (Forward Reference)
Branch on Not Zero Short	203	BNZS	A (Backward Reference)
	213	BNZS	A (Forward Reference)

APPENDIX 4 OP CODE MAP

	0	2	4	6	9	C	D	E
0		BTBS	STH	STE	SRLS	BXH	STM	
1	BALR	BTFS	BAL	AHM	SLLS	BXLE	LM	SVC
2	BTCR	BFBS	BTC		STBR	LPSW ^P	STB	SINT ^P
3	BFCR	BFFS	BFC		LBR	THI	LB	
4	NHR	LIS	NH	ATL	EXBR	NHI	CLB	
5	CLHR	LCS	CLH	ABL	EPSR ^P	CLHI	AL ^P	
6	OHR	AIS	OH	RTL	WBR ^P	OHI	WB ^P	
7	XHR	SIS	XH	RBL	RBR ^P	XHI	RB ^P	
8	LHR	LER	LH	LE	WHR ^P	LHI	WH ^P	
9	CHR	CER	CH	CE	RHR ^P	CHI	RH ^P	
A	AHR	AER	AH	AE	WDR ^P	AHI	WD ^P	RRL
B	SHR	SER	SH	SE	RDR ^P	SHI	RD ^P	RLL
C	MHR	MER	MH	ME	MHUR	SRHL	MHU	SRL
D	DHR	DER	DH	DE	SSR ^P	SLHL	SS ^P	SLL
E	ACHR		ACH		OCR ^P	SRHA	OC ^P	SRA
F	SCHR		SCH		AIR ^P	SLHA	AI ^P	SLA
	RR	RR	RX	RX	RR	RS	RX	RS

P = Privileged Instructions

APPENDIX 5

INSTRUCTION EXECUTION TIMES

MODEL 74

<u>Instr.</u>	<u>RR or SF</u>	<u>RS No Index</u>	<u>RS Indexed</u>	<u>RX No Index</u>	<u>RX Indexed</u>	<u>Comments</u>
ACH	1.75			3.50	3.50	
AH	1.50	2.50	3.00	3.25	3.25	
AHM				4.75	4.75	
AI	4.25			6.75	7.00	
AIS	2.25					
AL				6.25+3.25n	6.75+3.25n	n=no. of bytes n=no. of bytes
BAL	2.00			3.00	3.50	
BFBS	2.25/3.75					No Branch/Branch
BFC	2.25/2.00			3.25/3.00	3.75/3.50	No Branch/Branch
BFBS	2.25/3.50					No Branch/Branch
BTBS	2.00/4.00					No Branch/Branch
BTC	2.00/2.25			3.00/3.25	3.50/3.75	No Branch/Branch
BTFS	2.00/3.75					No Branch/Branch
BXH		4.50/5.00	4.50/5.00			No Branch/Branch
BXLE		4.75/4.75	4.75/4.75			No Branch/Branch
CH	2.25/2.50	3.25/3.50	3.75/4.00	4.25/4.50	4.25/4.50	Signs alike/differ
CLB				4.25	4.25	
CLH	1.50	2.50	3.00	3.25	3.25	
DH	54.25/54.25/56.5			56.00/56.00/58.25	56.0/56.0/58.25	Best/Ave/Worst
EPSR	4.00					
EXBR	2.00					
LB	2.25			4.00	4.00	
LCS	2.25					
LH	1.50	2.50	3.00	3.25	3.25	
LIS	2.25					
LM				4.0+1.5n	4.0+1.5n	n=Registers
LPSW		6.25	6.75			
MH	36.50/43.50/50.50			38.25/45.25/52.25	38.25/45.25/52.25	Best/Ave/Worst
MHU	33.75/40.75/47.75			35.50/42.50/49.50	35.50/42.50/49.50	Best/Ave/Worst
NH	1.50	2.50	3.00	3.25	3.25	
OC	3.50			4.50	4.50	
OH	1.50	2.50	3.00	3.25	3.25	
RB	5.75+3.5n+1.25XR2			5.25+3.5n	5.25+3.5n	n=Bytes
RD	3.25			6.25	6.25	
RH	4.25/3.25			5.75/5.50	5.50/5.50	Byte/HW
RLL		5.00+1.50n	5.00+1.50n			n=Shifts
RRL		5.00+1.50n	5.00+1.50n			n=Shifts
SCH	1.75			3.50	3.50	
SH	1.50	2.50	3.00	3.25	3.25	
SINT		8.25	8.75			
SIS	2.25					
SLA		6.00+1.50n	6.00+1.50n			n=Shifts
SLHA		6.00+1.00n	6.00+1.00n			n=Shifts

<u>Instr.</u>	<u>RR or SF</u>	<u>RS No Index</u>	<u>RS Indexed</u>	<u>RX No Index</u>	<u>RX Indexed</u>	<u>Comments</u>
SLHL		4.25+1.25n	4.25+1.25n			n=Shifts
SLL		6.00+1.50n	6.00+1.50n			n=Shifts
SLLS	3.25+1.25n					n=Shifts
SRA		6.75+1.75n	6.75+1.75n			n=Shifts
SRHA		4.25+1.25n	4.25+1.25n			n=Shifts
SRHL		4.25+1.25n	4.25+1.25n			n=Shifts
SRL		6.75+1.50n	6.75+1.50n			n=Shifts
SRLS	3.25+1.25n					n=Shifts
SS	3.50			6.25	6.25	
STB	2.75			4.50	4.50	
STH				3.75	3.75	
STM				5.0+1.25n	5.0+1.25n	n=Registers
SVC		10.50	10.50			
THI		2.50	2.50			
WB	6.25+2.75n+1.25XR2			5.75+2.75n	5.75+2.75n	n=Bytes
WD	3.50			4.50	4.50	
WH	5.00/3.75			6.00/4.50	6.00/4.50	Byte/HW
XH	1.50	2.50	3.00	3.25	3.25	

Normal I/O Interrupt - 7.75 microseconds
 Immediate Interrupt - 7.75 microseconds
 Machine Malfunction - 8.00 microseconds
 Illegal Instruction - 8.25 microseconds
 Auto-Boot Loader - 3 microseconds/byte

PSW Bit 6, No Queue Service Interrupt: add 2.75 microseconds to LPSW or EPSR or PSW SWAP.

Queue Service Interrupt: add 8.50 microseconds to LPSW or EPSR or PSW SWAP.

MODEL 70

<u>Instr.</u>	<u>RR or SF</u>	<u>RS</u>	<u>RS Indexed</u>	<u>RX</u>	<u>Comments</u>
ABL				5/13/13	OVF/NORM/WRAP
ACH	1.25			3.25	
AE	21.25/25.25/34			23/26.5/33.5	MIN/AVE/MAX
AH	1.0	2.25	3.25	3.25	
AHM				4.0	
AI	2.75			5.0	
AIS	1.5				
AL				6.5 + 4.5n	n=no. of bytes
ATL				5/11.5/11.75	OVF/NORM/WRAP
BAL	1.5			2.5	
BFBS	1.5/3.0				No BR/BR
BFC	1.5			2.75/3.0	No BR/BR
BFBS	1.5/3.25				No BR/BR
BTBS	1.25/3.25				No BR/BR
BTC	1.5			2.75/3.0	No BR/BR
BTFS	1.25/3.5				No BR/BR
BXH		5.0/4.5	6.0/5.5		No BR/BR
BXLE		5.0/5.0	6.0/6.0		No BR/BR
CE	10.75/12.75/9.75			12.0/13.0/9.75	+,+/-,-/+, -
CH	2.0/2.25	3.0/3.25	4.0/4.25	4.0/4.25	Signs alike/Signs differ
CLB				3.75	
CLH	1.0	2.25	3.25	3.25	
DE	105/108.25/116.25			106.75/109.5/117.25	MIN/AVE/MAX
DH	10.25/12/10.25/10.5			12.25/14/12.25/12.5	++/+-/-+/-
EPSR	3.25				
EXBR	1.0				
LB	1.0			3.25	
LCS	1.5				
LE	12.5/13.25/19.25/27.5			13.25/14/20/29	0/BEST/AVE/WORST
LH	1.0	2.0	3.0	3.0	
LIS	1.25				
LM				4.5 + 1.5n	n=no. of regs.
LPSW		5.25	6.25		
ME	60.5/74.25/91.25			54.25/73.5/91.75	MIN/AVE/MAX
MH	8/9/8.75/8.25			10/11/10.75/10.25	++/+-/-+/-
MHU	6.0			8.0	
NH	1.0	2.25	3.25	3.25	
OC	2.25			4.0	
OH	1.0	2.25	3.25	3.25	
RB	5.5+3n			6.5 + 3n	n=no. of bytes
RBL				4.25/11.75/12	EMPTY/NORM/WRAP
RD	2.0			4.5	
RH	2.75/2			5.5/4.75	BYTE/HALFWORD
RLL		3.5+1.0(n-1)	4.5+1.0(n-1)		n=no. of shifts
RRL		3.5+1.0(n-1)	4.5+1.0(n-1)		n=no. of shifts
RTL				4.25/13/13	EMPTY/NORM/WRAP

<u>Instr.</u>	<u>RR or SF</u>	<u>RS</u>	<u>RS Indexed</u>	<u>RX</u>	<u>Comments</u>
SCH	1.25			3.25	
SE	22/30.5/29.25			25/31.5/48.75	MIN/AVE/MAX
SH	1.0	2.25	3.25	3.25	
SINT	See I/O Channel Timing				
SIS	1.5				
SLA		3.5+.25(n-1)	4.5+.25(n-1)		n=no. of shifts
SLHA		3.5+.25(n-1)	4.5+.25(n-1)		n=no. of shifts
SLHL		2.75+.25(n-1)	3.75+.25(n-1)		n=no. of shifts
SLL		3.0+.25(n-1)	4.0+.25(n-1)		n=no. of shifts
SLLS	2.0+.25(n-1)				n=no. of shifts
SRA		3.25+.25(n-1)	4.25+.25(n-1)		n=no. of shifts
SRHA		3.25+.25(n-1)	4.25+.25(n-1)		n=no. of shifts
SRHL		2.75+.25(n-1)	3.75+.25(n-1)		n=no. of shifts
SRL		3.0+.25(n-1)	4.0+.25(n-1)		n=no. of shifts
SRLS	2.0+.25(n-1)				n=no. of shifts
SS	2.25			4.5	
STB	2.0			4.0	
STE				7.0	
STH				3.25	
STM				4.5+1.25n	n=no. of regs.
SVC		7.0	8.0		
THI		2.25	3.25		
WB	5.0+3n			5.5+3n	n=no. of bytes
WD	2.25			4.25	
WH	3.5/2.75			4.75/4.0	BYTE/HALFWORD
XH	1.0	2.25	3.25	3.25	

MODEL 80

<u>Instr.</u>	<u>RR or SF</u>	<u>RS*</u>	<u>RX</u>	<u>Comments</u>
ABL			2.95/5.51/5.81	OVFL/NORM/WRAP
ACH	.53		1.01	
AE	12.01/13.61/23.01		11.41/13.01/22.41	MIN/AVE/MAX
AH	.53	.53	1.01	
AHM			1.49	
AI	1.61		2.29	
AIS	.53			
AL			3.99+1.8L+2.4n	L=Leador n=Bytes
ATL			2.95/5.97/6.27	OVF/NORM/WRAP
BAL	1.13		1.13	
BFBS	.53/1.33			No Branch/Branch
BFC	.53/1.01		.63/1.13	No Branch/Branch
BFFS	.53/1.33			No Branch/Branch
BTBS	.53/1.33			No Branch/Branch
BTC	.53/1.01		.63/1.13	No Branch/Branch
BTFS	.53/1.33			No Branch/Branch
BXH		2.45/3.13		No Branch/Branch
BXLE		2.45/3.13		No Branch/Branch
CCS				
CE	4.97/5.67		4.37/5.07	Signs alike/differ
CH	1.33/1.73	1.13/1.53	1.41/1.81	Signs alike/differ
CLB			1.38	
CLM	.53	.53	1.01	
DE	32.89/32.89/34.09		32.29/32.29/33.49	MIN/AVE/MAX
DH	3.03		3.23	
ECS				
EPSB	1.73			
EXBR	.53			
LB	.53		1.06	
LCS	.53			
LE	4.76/8.76/14.76		4.16/8.16/14.16	MIN/AVE/MAX
LH	.53	.53	1.01	
LIS	.53			
LM			1.03+.5n	n=no. of Registers
LPSW		2.29		
ME	23.27/24.47/24.77		19.27/20.47/20.77	MIN/AVE/MAX
MH	2.25		2.73	
MHU	2.25		2.73	
NH	.53	.53	1.01	
OC	1.06		2.12	
OH	.53	.53	1.01	

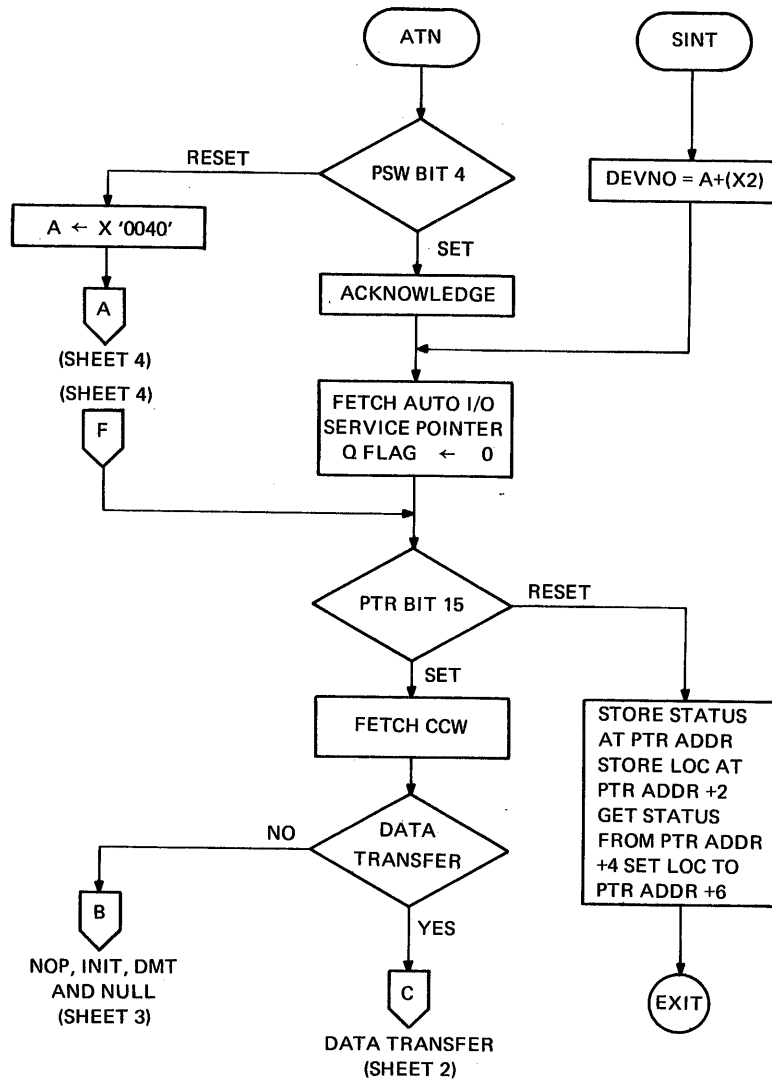
* No additional time is required for indexing.

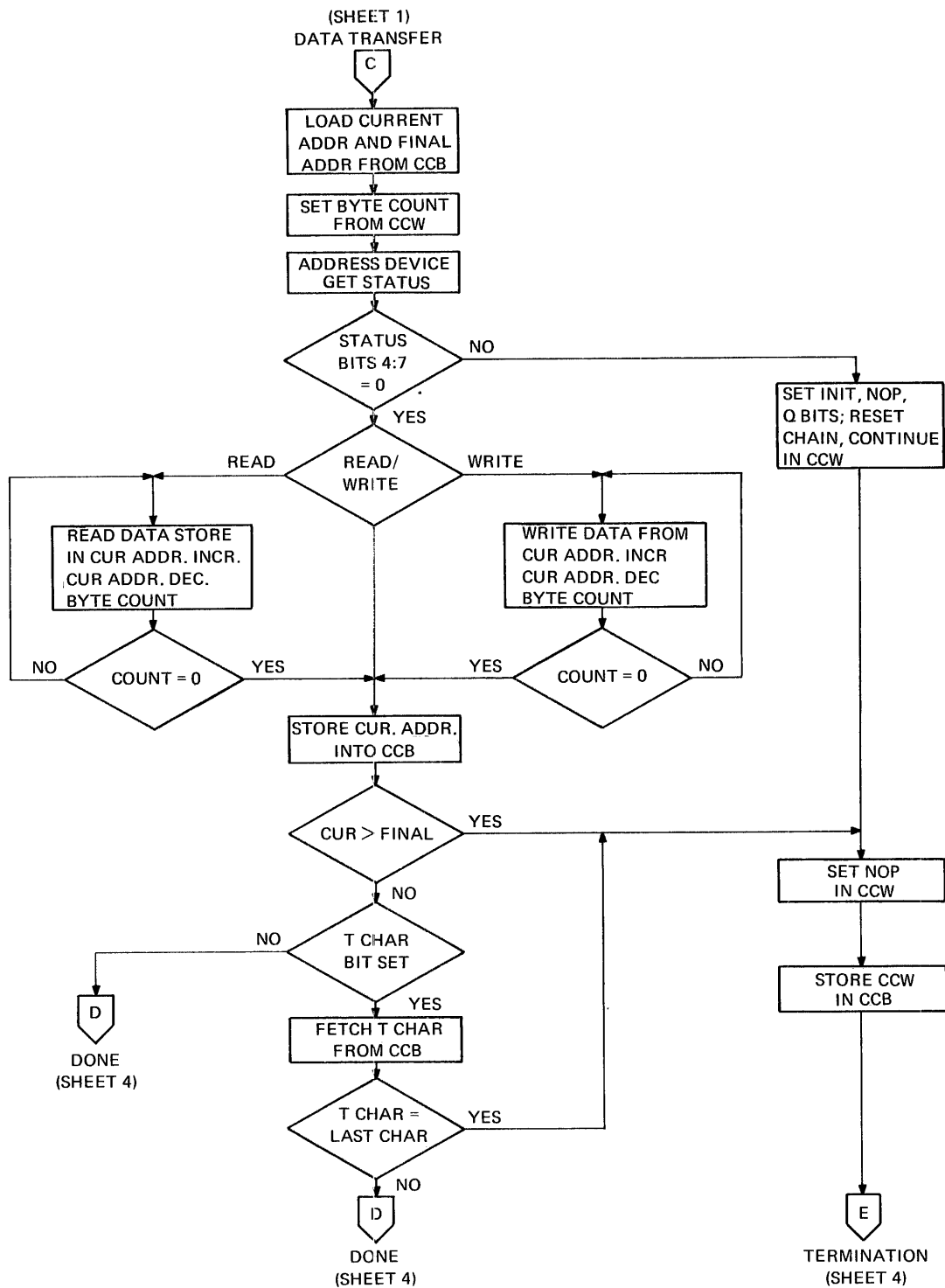
<u>Instr.</u>	<u>RR or SF</u>	<u>RS*</u>	<u>RX</u>	<u>Comments</u>
RB	3.64+2.4n		3.72+2.4n	n=Bytes
RBL			3.18/4.16/4.46	EMPTY/NORM/WRAP
RD	1.06		2.12	
RH	1.06/1.46		1.94/2.34	HALFWORD/BYTE
RLL		.85+.1n		n=no. of shifts - 1
RRL		.85+.1n		n=no. of shifts - 1
RTL			3.18/4.16/4.46	EMPTY/NORM/WRAP
SCH	.53		1.01	
SE	12.26/13.86/23.26		11.61/13.21/22.61	MIN/AVE/MAX
SH	.53	.53	1.01	
SINT		See Automatic I/O Times		
SIS	.53			
SLA		.85+.1n		n=no. of shifts - 1
SLHA		.65+.1n		n=no. of shifts - 1
SLHL	.55+.1n	.65+.1n		n=no. of shifts - 1
SLL		.85+.1n		n=no. of shifts - 1
SRA		.85+.1n		n=no. of shifts - 1
SRHA		.65+.1n		n=no. of shifts - 1
SRHL	.55+.1n	.65+.1n		n=no. of shifts - 1
SRL		.85+.1n		n=no. of shifts - 1
SS	1.06		2.12	
STB	.53		1.26	
STE			3.17	
STH			1.41	
STM			1.23+.5n	n=no. of Registers
SVC		3.65		
THI		.53		
WB	3.44+2.4n		3.72+2.4n	n=Bytes
WD	1.06		2.32	
WH	1.06/1.46		1.64/2.04	HALFWORD/BYTE
XH	.53	.53	1.01	

* No additional time is required for indexing.

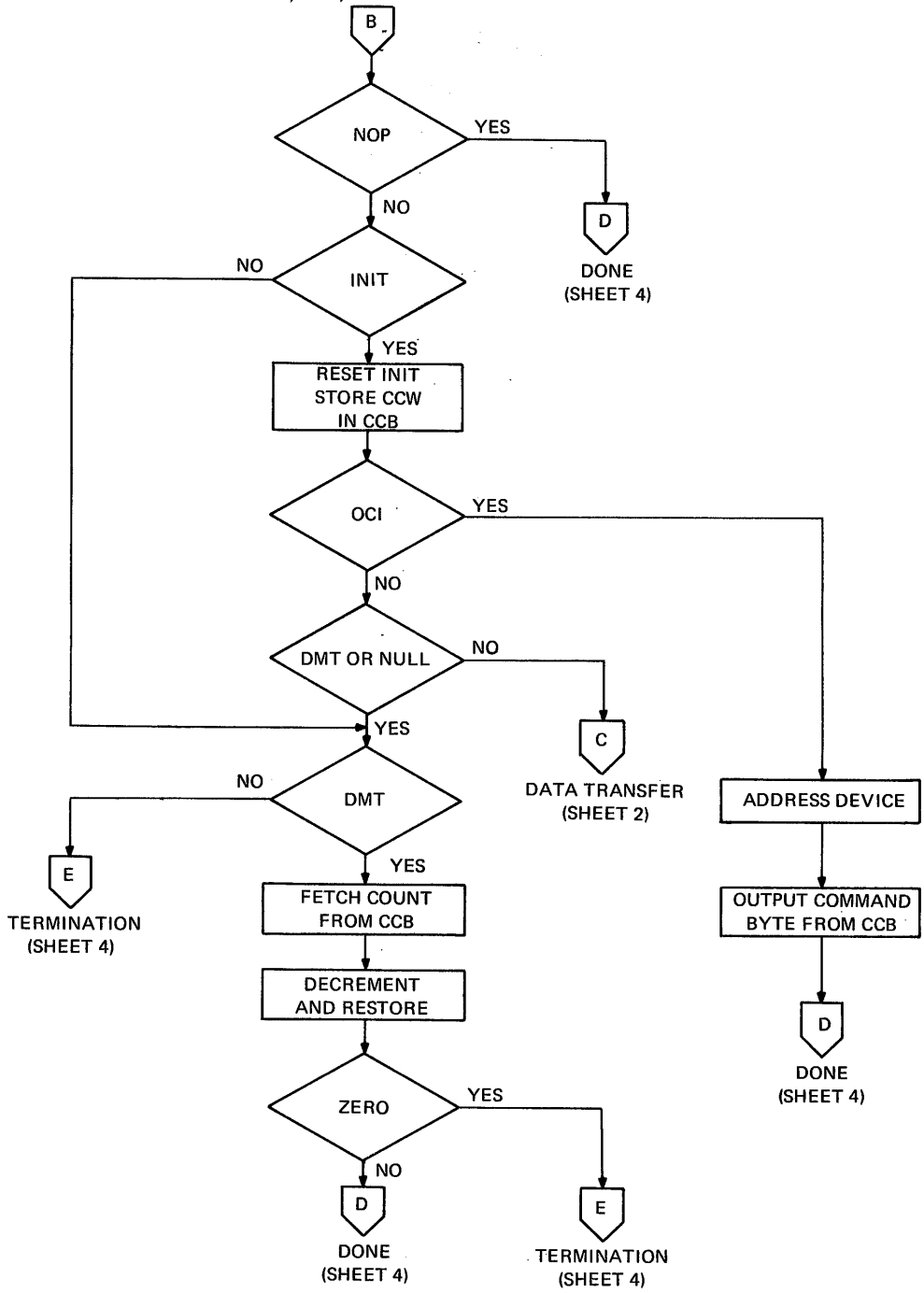
APPENDIX 6

AUTOMATIC I/O OPERATION AND TIMING DATA





(SHEET 1)
NOP, INIT, DMT AND NULL



MODEL 70

AUTOMATIC INPUT/OUTPUT INTERRUPT SERVICE TIMES

	NOP	NULL	DMT	OCI	READ	WRITE	BAD STATUS
BASE	12.00	15.25	17.00	17.25	18.00+2.5n	18.75+2.25n	20.00
INIT	—	3.50	3.50	—	4.75	4.75	4.75
TCHAR no match	—	—	—	—	2.75	2.75	—
1 {	TCHAR match	—	—	—	6.5	6.5	—
	COUNT=0 or CUR=final	—	—	1.75	—	3.75	—
2 {	QUEUE high	—	11.00	11.00	—	11.00	—
	QUEUE low	—	12.75	12.75	—	12.75	—
	CHAIN	—	3.0	3.0	—	3.0	—
	CONTINUE	—	1.25+	1.25+	—	1.25+	—
	QSVC INT.	3.5	3.5	3.5	3.5	3.5	3.5

1. Reason for termination
2. Termination Procedure

All times are given in microseconds. To determine the execution time of a particular interrupt, add to the base time, the time for each pertinent option. For example: a Write of one character using a Termination Character (TCHAR) with no match takes 21.00 (BASE)
 plus 2.75 (TCHAR no match)
 23.75 microseconds

SINT Execution time is the same. Add 1.6 microseconds if indexed.
 On Read and Write times, n is the number of bytes transferred per interrupt.

Normal Interrupt Latency Time: 4 microseconds

Non-Interruptable Instructions: Load/Store Multiple, Read/Write Block, Autoload, Automatic I/O Service, Supervisor Call, Remove from Top/Bottom of List, Add to Top/Bottom of List

Machine Malfunction Interrupt: 7.75us

Normal I/O Interrupt: 7.25us

Immediate I/O Interrupt: 7.75us

	One HW	Burst
Interleaved Data Channel Read	7.5 us	4.5n
Interleaved Data Channel Write	7.75 us	4.75n

n = halfwords

These times assume the standard 1.00 microsecond core memory with no interference from a Selector Channel or any other device on the memory bus.

MODEL 80

AUTOMATIC INPUT/OUTPUT INTERRUPT SERVICE TIMES

Assume Ideal Device Response

	NOP	NULL	DMT	OC	READ	WRITE	BAD STATUS
BASE	6.40	8.35	7.70	9.38	11.09+1.6n	11.29+1.6n	17.41
INIT	--	1.60	1.60	--	2.40	2.40	2.40
TCHAR no match	--	--	--	--	1.68	1.68	--
1 { TCHAR match	--	--	--	--	3.86	3.86	--
	Count=0 Strt=End	--	1.65 --	--	1.78	1.78	--
QUEUE high	--	6.87	6.87	--	6.87	6.87	--
QUEUE low	--	7.30	7.30	--	7.30	7.30	--
2 { QUEUE* overflow	--	-2.68	-2.68	--	-2.68	-2.68	-2.68
CHAIN	--	1.85	1.85	--	1.85	1.85	--
CONTINUE	--	-3.74 +	-3.74 +	--	-3.74 +	-3.74 +	--
QSVC INT.	3.12	3.12	3.12	3.12	3.12	3.12	3.12

1. Reason for termination

2. Termination procedure

*If queue overflow occurs, no chain or continue or queue service interrupt can occur.

n = Number of bytes per interrupt

Times are given in microseconds.

SINT Execution times 0.65 microseconds less than IO service times.

Immediate interrupt = 4.42 microseconds

Normal I/O interrupt = 4.10 microseconds

MALF interrupt = 4.05 microseconds

Interleaved Data Channel Read 1.85 microseconds/transfer

Interleaved Data Channel Write 2.07 microseconds/transfer

APPENDIX 7

I/O REFERENCES

CONTROL CONSOLE STATUS AND COMMAND BYTE DATA (HEX ADDRESS 01)

BIT NUMBER	0	1	2	3	4	5	6	7
STATUS BYTE	MODE				REGISTER DISPLAY			
COMMAND BYTE	NORM	INC						

STATUS:

MODE	{	SINGLE STEP	0	1	0	0	x	x	x	x
		RUN	1	0	0	0	x	x	x	x
		HALT	1	1	0	0	x	x	x	x
		MEM WRITE	0	0	0	1	0	0	0	0
		MEM READ	0	0	1	0	0	0	0	
		ADRS	0	0	1	1	0	0	0	
		OFF					0	0	0	

REGISTER DISPLAY	{	INST	0	0	1	0
		PSW	0	1	0	0
		R0, R1	1	0	0	0
		R2, R3	1	0	0	1
		R4, R5	1	0	1	0
		R6, R7	1	0	1	1
		R8, R9	1	1	0	0
		RA, RB	1	1	0	1
		RC, RD	1	1	1	0
		RE, RF	1	1	1	1

COMMAND:

- NORM
In the Normal Mode, Byte 0 of the Register Display or Data/Address switches is accessed each time an I/O operation is directed to the Control Console.
- INC
In the Incremental Mode, subsequent I/O operations access subsequent bytes of the Register Display or Data/Address switches.

TELETYPE/ASCII HEX CONVERSION TABLE

HEX (MSD) →					0	1	2	3	4	5	6	7
(LSD) ↓	Teletype Tape Channels → ↓			8	DEPENDS UPON PARITY*							
				7	0	0	0	0	1	1	1	1
				6	0	0	1	1	0	0	1	1
				5	0	1	0	1	0	1	0	1
	4	3	2	1								
0	0	0	0	0	NULL	DC ₀	SPACE	0	@	P		
1	0	0	0	1	SOM	X-ON	!	1	A	Q		
2	0	0	1	0	EOA	TAPE ON	"	2	B	R		
3	0	0	1	1	EOM	X-OFF	#	3	C	S		
4	0	1	0	0	EOT	TAPE OFF	\$	4	D	T		
5	0	1	0	1	WRU	ERR	%	5	E	U		
6	0	1	1	0	RU	SYNC	&	6	F	V		
7	0	1	1	1	BELL	LEM	'	7	G	W		
8	1	0	0	0	FE ₀	S ₀	(8	H	X		
9	1	0	0	1	HT/SK	S ₁)	9	I	Y		
A	1	0	1	0	LF	S ₂	*	:	J	Z		
B	1	0	1	1	VT	S ₃	+	;	K	[
C	1	1	0	0	FF	S ₄	,	<	L	\		ACK
D	1	1	0	1	CR	S ₅	-	=	M]		ALT. MODE
E	1	1	1	0	SO	S ₆	.	>	N	↑		ESC
F	1	1	1	1	SI	S ₇	/	?	O	←		DEL

*Parity bit adjusted for even parity (even number of 1's) on input from Teletype keyboard. Parity bit is ignored on output to Teletype printer.

ASCII CARD CODE CONVERSION TABLE

<u>GRAPHIC</u>	<u>7-BIT ASCII CODE</u>	<u>CARD CODE</u>	<u>GRAPHIC</u>	<u>7-BIT ASCII CODE</u>	<u>CARD CODE</u>
SPACE	20	BLANK	@	40	8-4
!	21	12-8-7	A	41	12-1
"	22	8-7	B	42	12-2
#	23	8-3	C	43	12-3
\$	24	11-8-3	D	44	12-4
%	25	0-8-4	E	45	12-5
&	26	12	F	46	12-6
'	27	8-5	G	47	12-7
(28	12-8-5	H	48	12-8
)	29	11-8-5	I	49	12-9
*	2A	11-8-4	J	4A	11-1
+	2B	12-8-6	K	4B	11-2
,	2C	0-8-3	L	4C	11-3
-	2D	11	M	4D	11-4
.	2E	12-8-3	N	4E	11-5
/	2F	0-1	O	4F	11-6
0	30	0	P	50	11-7
1	31	1	Q	51	11-8
2	32	2	R	52	11-9
3	33	3	S	53	0-2
4	34	4	T	54	0-3
5	35	5	U	55	0-4
6	36	6	V	56	0-5
7	37	7	W	57	0-6
8	38	8	X	58	0-7
9	39	9	Y	59	0-8
:	3A	8-2	Z	5A	0-9
;	3B	11-8-6	[5B	12-8-2
<	3C	12-8-4	\	5C	11-8-1
=	3D	8-6]	5D	11-8-2
>	3E	0-8-6	↑	5E	11-8-7
?	3F	0-8-7	←	5F	0-8-5

EIGHT-LINE INTERRUPT MODULE STATUS
AND COMMAND BYTE DATA
(HEX ADDRESS 20-27)

BIT NUMBER	0	1	2	3	4	5	6	7
STATUS BYTE	0	0	0	0	0	0	0	0
COMMAND BYTE	DISABLE	ENABLE	RESET	SET	CLEAR	GCMD0	GCMD1	*

*Bit not used.

The status byte is always zero.

DIS - Disable DEVICE INTERRUPT (but allow queueing)

ENAB - Enable DEVICE INTERRUPT

RESET - Establish Reset Mode, one Write Data selectively reset interrupt lines.

SET - Establish Set Mode, one Write Data selectively set interrupt lines.

CLEAR - Clear all pending interrupts

GCMD0, GCMD1 - These Command bits may be optionally gated-out to a user's own equipment. Their function, if used, is dependent upon this external equipment.

Any command in which Bit 2 and 3 = 0 places the module in the Load Mask Mode. In the Load Mask Mode, one Write Data is required to set the mask.

INITIALIZATION - Disables, interrupts, clears all commands, clears all pending interrupts, and places the module in the Load Mask Mode.

AUTOMATIC MEMORY PROTECT
STATUS AND COMMAND BYTE DATA
(HEX ADDRESS AE)

BIT NUMBER	0	1	2	3	4	5	6	7
STATUS BYTE	*	*	P ON	PWF	*	EX	*	*
COMMAND BYTE	DISARM	ARM	P ON	P OFF	*	*	*	*

* Bit not used.

STATUS

- P ON - This Status bit is set when memory protection is enabled.
- PWF - The Protected Write Flag is set when an attempt has been made to write into a protected area of memory. PWF is reset only by an Output Command (OC or OCR), an Acknowledge Interrupt (AI or AIR), or the INITIALIZE pushbutton.
- EX - The Examine bit is set whenever the PWF bit is set.

COMMAND

- DISARM - This Command bit disables the device interrupt feature and prevents interrupts from being queued.
- ARM - This Command bit enables a device interrupt to occur when an attempt is made to write into a protected area of memory.
- P ON - This Command bit enables memory to be protected as per the protection pattern.
- P OFF - This Command bit overrides all memory protection.

PROTECT PATTERN -

After an output command (any output command), the protect pattern may be set up with consecutive Write Data instructions. If more than eight Write Data instructions are issued, then the protect pattern will wrap around. The ninth WD instruction will change Blocks 0-7 etc.

INITIALI-
ZATION -

Dis-arm interrupts, clears PON and PWF flip-flops, and leaves the protect pattern un-changed.

SELECTOR CHANNEL STATUS AND
COMMAND BYTE DATA
(HEX ADDRESS F0)

BIT NUMBER	0	1	2	3	4	5	6	7
STATUS BYTE					BSY			
COMMAND BYTE			READ	GO	STOP			

BSY This bit is set when the Selector Channel is in the process of transferring data.

READ This command changes the mode of the Selector Channel from Write to Read. In the Read Mode, data is transmitted from the active device on the Selector Channel and written into core memory. Whenever a Read Data or a Write Data Instruction is issued to the Selector Channel, the Selector Channel is placed in the Write Mode. Each time a READ operation is required, a Read Command must be issued.

GO This command initiates a data transmission. This command can be issued at the same time the Read/Write Mode is established.

STOP This command halts any data transmission in process, and initializes the Selector Channel for starting a new operation. It should be given when the Selector Channel Terminates.

DEVICE NUMBER

The Selector Channel is normally assigned device number X'F0', but may easily be changed by a minor wiring modification on the Selector Channel device controller board. Refer to the maintenance manual for specific details.

INITIALIZATION

Whenever the Initialize switch on the Processor is depressed, or a Stop Command is issued, the following actions occur:

1. Any data transmission in process is halted and the Stop Mode is effected.
2. The Selector Channel is placed in the Write Mode.
3. The Selector Channel is made idle.
4. The Selector Channel interrupt is reset.

APPENDIX 8

ARITHMETIC REFERENCES

TABLE OF POWERS OF TWO

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																	
68	719	476	736	36	0.000	000	000	014	551	915	228	366	851	806	640	625																	
137	438	953	472	37	0.000	000	000	007	275	957	614	183	425	903	320	312	5																
274	877	906	944	38	0.000	000	000	003	637	978	807	091	712	951	660	156	25																
549	755	813	888	39	0.000	000	000	001	818	989	403	545	856	475	830	078	125																
1	099	511	627	776	40	0.000	000	000	000	909	494	701	772	928	237	915	039	062	5														

TABLE OF POWERS OF SIXTEEN

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
	17	592	186	044		416	11
	281	474	976	710		656	12
	4	503	599	627	370	496	13
	72	057	594	037	927	936	14
1	152	921	504	606	846	976	15

Decimal Values

HEXADECIMAL TO DECIMAL CONVERSION TABLE

BYTE				BYTE			
HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC
0	0	0	0	0	0	0	0
1	4,096	1	256	1	16	1	1
2	8,192	2	512	2	32	2	2
3	12,288	3	768	3	48	3	3
4	16,384	4	1,024	4	64	4	4
5	20,480	5	1,280	5	80	5	5
6	24,576	6	1,536	6	96	6	6
7	28,672	7	1,792	7	112	7	7
8	32,768	8	2,048	8	128	8	8
9	36,864	9	2,304	9	144	9	9
A	40,960	A	2,560	A	160	A	10
B	45,056	B	2,816	B	176	B	11
C	49,152	C	3,072	C	192	C	12
D	53,248	D	3,328	D	208	D	13
E	57,344	E	3,584	E	224	E	14
F	61,440	F	3,840	F	240	F	15

HEXADECIMAL ADDITION TABLE

	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	1
2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	2
3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	3
4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	4
5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	5
6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	6
7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	7
8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	8
9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	9
A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	A
B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	B
C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	C
D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	D
E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	E
F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	F
	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

HEXADECIMAL MULTIPLICATION TABLE

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

TABLE OF MATHEMATICAL CONSTANTS

Constant	Decimal Value			Hexadecimal Value	
π	3.14159	26535	89793	3.243F	6A89
π^{-1}	0.31830	98861	83790	0.517C	C1B7
$\sqrt{\pi}$	1.77245	38509	05516	1.C5BF	891C
$\text{Ln } \pi$	1.14472	98858	49400	1.250D	048F
e	2.71828	18284	59045	2.B7E1	5163
e^{-1}	0.36787	94411	71442	0.5E2D	58D9
\sqrt{e}	1.64872	12707	00128	1.A612	98E2
$\log_{10} e$	0.43429	44819	03252	0.6F2D	EC55
$\log_2 e$	1.44269	50408	88963	1.7154	7653
γ	0.57721	56649	01533	0.93C4	67E4
$\text{Ln } \gamma$	-0.54953	93129	81645	-0.8CAE	9BC1
$\sqrt{2}$	1.41421	35623	73095	1.6A09	E668
$\text{Ln} 2$	0.69314	71805	59945	0.B172	17F8
$\log_{10} 2$	0.30102	99956	63981	0.4D10	4D42
$\sqrt{10}$	3.16227	76601	68379	3.298B	075C
$\text{Ln} 10$	2.30258	50929	94046	2.4D76	3777

APPENDIX 9

GLOSSARY OF TERMS

This appendix explains some terms and concepts used in INTERDATA programs and program documentation. The terms are arranged in alphabetical order for easy reference.

- 8-Bit Loader** Any loader which reads 8-bit bytes from some device such as a Teletype or paper tape reader, and stores the bytes directly into memory. The "50 Sequence" which resides from X'50' to X'7F' in memory includes an 8-bit loader. This loader reads 8-bit bytes from the Binary Input Device and stores them from X'80' to X'CF', and then transfers to X'80'. Refer to Section 9.3.5 for more details on the 50 Sequence.
- 50 Sequence** The 50 Sequence resides in core memory from X'50' to X'7F' and contains an 8-bit loader and a Device Definition Table. This sequence must be manually entered into memory using control panel memory Write operations. This area of core memory should be reserved for the 50 Sequence; once keyed into memory, this sequence normally remains there, available for use. The Device Definition Table uses four halfwords to define devices for four functions: binary inputs, binary outputs, source inputs, and symbolic outputs. Various programs refer to the 50 Sequence for device definition. Refer to Section 9.3.5 for more details on the 50 Sequence.
- Absolute** Programs designed to occupy a fixed set of locations in the core memory are called absolute programs. For example, an absolute program designed for bytes 80-99 in memory does not execute correctly if moved (relocated) to bytes 180-199 in memory. See Relocatable.
- Assembler** The assembler programs translate the source form of a program into a form which can be conveniently loaded into the system by a loader program. INTERDATA provides assembler programs which convert assembly - language programs into binary object tapes. See Object and Source.
- Bias** The base value used by a loader to load a relocatable program is called the bias. The bias value is added to all relocatable quantities during the loading process. See General Loader.
- Bootstrap Tapes** Certain program tapes are provided with the appropriate loaders on the tape itself. These tapes are loaded into memory using the 8-bit loader at 50. All bootstrap tapes have a part number with an M10 designation. See 50 Sequence and Fast Format.
- Compiler** A compiler is a program for translating a higher-level language program into machine language, for execution by the Processor. INTERDATA provides a FORTRAN IV Compiler which recognizes USASI FORTRAN as defined by standard X3.9-1966.

Editor	An editor is a program which manipulates symbolic or textual information. It facilitates the creation, examination, and modification of character oriented data. Such a program is useful for the creation and editing of source tapes. See <u>Source</u> .
Fast Format	Bootstrap tapes for programs, such as the General Loader or the Basic Assembler, employ a fast format for data organization. This format is essentially an 8-bit format which minimizes loading time on slow devices. Fixed length records are used, however, to facilitate checksum procedures. A transfer address may be specified in the first record of a Fast Format tape. See <u>Bootstrap</u> .
Firmware	Micro-programs which are written for a Read-Only-Memory (ROM) are called firmware, as opposed to conventional machine language programs which are called software. See <u>Micro-Program</u> .
Floating-Point	A method of representing numbers with a mantissa or fraction, and an exponent or characteristic. For example, in the number $.5 \times 10^3$, the .5 is the mantissa and the 3 is the exponent of the base 10. INTERDATA systems represent floating-point numbers in a floating hexadecimal format using a 24 bit fraction and a 7 bit exponent of the base 16, in excess 64 notation.
FORTTRAN	The FORTRAN language permits the statement of arithmetic problems in an algebraic-type format. INTERDATA provides both a FORTRAN IV Compiler and an Interactive FORTRAN interpreter which permits problems to be created and executed in an interactive manner. The FORTRAN IV Compiler recognizes the USASI standard FORTRAN language; the Interactive FORTRAN program uses an abbreviated form of the FORTRAN language.
General Loader	The General Loader is used in a stand-alone (not operating system) environment. This loader handles absolute or relocatable programs with external program linkages and forward reference definitions which occur on object tapes from one-pass assemblies. The loader bias and error messages are printed on the Teletype for operator convenience. See <u>One-Pass</u> and <u>Bias</u> .
Listing	The assembler inputs a source tape and generates an object tape and a listing. The object tape contains the binary information to be loaded into memory. The listing is a printed record which shows each source statement, and the binary information generated for that statement. The binary information is represented in hexadecimal form.
Loader	A loader is a type of program which, when executed by the machine, reads information from a peripheral device and loads the core memory with instructions and data.
Loader Format	The object tapes generated by the assembler programs use a specific tape format known as loader format. These object tapes include control and linkage information in addition to the instructions and data of the assembled program. Tapes in this format must be loaded by the REL Loader, or General Loader; BOSS Resident Loader, or OS Library Loader.
Micro-Programs	INTERDATA machines involve a Read-Only-Memory (ROM) used to control basic Processor operations. The sequence of commands which reside in the ROM is called a micro-program. See <u>Firmware</u> .
Object	Object tapes are tapes produced by the assembler in standard loader format. For each source tape assembled, there is an object tape. A loader reads the object tapes and places the corresponding instructions and data in core memory. See <u>Assembler</u> , <u>Loader</u> , and <u>Source</u> .

One-Pass	The assembler takes one, two, or three passes across the source tape to complete an assembly. The number of passes is controlled by an option control statement in the source program. When so directed, the assembler makes an assembly, complete with listing and object tape, in one pass. One pass assemblies minimize the time required for an assembly, although there are some restrictions on the programs that can be assembled. Two pass assemblies are normally used. Three pass assemblies are appropriate when the physical devices in use prohibit printing and punching on the same pass.
Operating System	An operating system is a program which handles all the house-keeping or overhead tasks within a computer so that programs can be as independent from hardware details as possible. The operating system also sets standards and conventions so that programs are compatible with one another. INTERDATA provides a Basic Operating System (BOSS) for the Models 4, 5 and 70. BOSS handles all I/O transfers in a device independent fashion, services all systems interrupts and services operator commands.
Part Number	Each program is identified by a part number which defines the program and the revision level. For example, the part number for the REL loader is 06-024R01. In this number, the 06-024 identifies the specific program, and R01 indicates the revision level. The various elements of this program, such as the object tape, carry a suffix to the basic number. For example, 06-024R01M10 identifies the bootstrap object tape, and 06-024R01A13 identifies the assembly listing. Part numbers are discussed in Section 9.8.
Program	A program is a set of machine instructions which, when executed by the machine, performs some useful functions. Programs are often referred to as software.
Relocatable	Programs designed to be loaded anywhere in core memory are called relocatable. For example, a program which occupies 26 bytes could be loaded into X'80' - X'99' or X'180' - X'199' and executed from either location. Relocatable programs may be loaded with the REL Loader, General Loader, the BOSS Resident Loader, or the OS Library Loader. Some INTERDATA programs are absolute and some are relocatable. When loading an absolute or relocatable program, it is important to determine that the required memory space is available. See <u>Absolute</u> .
Relocating Loader	The Relocating Loader (REL) is appropriate for loading absolute or relocatable binary object tapes from 1, 2, or 3 pass assemblies. This loader is not appropriate for linking to external programs. See <u>Relocatable</u> and <u>One-Pass</u> .
Source	A source is a mnemonic or easy-to-read representation of a program. Source programs can be written using Assembler language or FORTRAN Compiler language. The source is often prepared as a source paper tape, or source deck of punched cards. See <u>Object</u> .

Notes

PUBLICATION COMMENT FORM

Please use this postage-paid form to make any comments, suggestions, criticisms, etc. concerning this publication.

From _____ Date _____

Title _____ Publication Title _____

Company _____ Publication Number _____

Address _____

FOLD

FOLD

Check the appropriate item.

Error (Page No. _____, Drawing No. _____)

Addition (Page No. _____, Drawing No. _____)

Other (Page No. _____, Drawing No. _____)

Explanation:

FOLD

FOLD

CUT ALONG LINE

Fold and Staple
No postage necessary if Mailed in U.S.A.

STAPLE

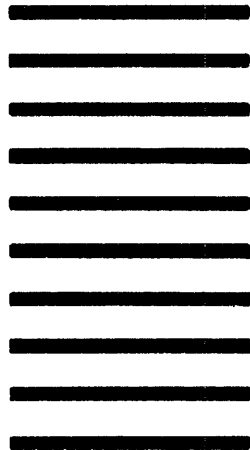
STAPLE

FOLD

FOLD

BUSINESS REPLY MAIL
 No Postage Necessary If Mailed In U.S.A.

FIRST CLASS
 PERMIT No. 22
 OCEANPORT, N.J.



Postage Will Be Paid By:



INTERDATA®

2 Crescent Place
 Oceanport, New Jersey 07757

PUBLICATIONS DEPT.

FOLD

FOLD

STAPLE

STAPLE